

UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

Tema:

API RESTFUL PARA LA GESTIÓN DEL MANTENIMIENTO PREDICTIVO DE BOMBAS HIDRÁULICAS EN LA PRODUCCIÓN DE CRUDO

Trabajo de titulación modalidad Proyecto de Investigación, presentado previo a la obtención del título de Ingeniero en Tecnologías de la Información

ÁREA: Gestión de Tecnologías de la Información

LÍNEA DE INVESTIGACIÓN: Software, Tecnologías de la Información y Ciencia de Datos

AUTOR: Jaime Marcelo Villacrés Espin

TUTOR: Ing. Edison Homero Álvarez Mayorga, Mg.

Ambato - Ecuador

enero - 2026

APROBACIÓN DEL TUTOR

En calidad de tutor del trabajo de titulación con el tema: API RESTFUL PARA LA GESTIÓN DEL MANTENIMIENTO PREDICTIVO DE BOMBAS HIDRÁULICAS EN LA PRODUCCIÓN DE CRUDO, desarrollado bajo la modalidad Proyecto de Investigación por el señor Jaime Marcelo Villacrés Espín, estudiante de la Carrera de Ingeniería en Tecnologías de la Información de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, de la Universidad Técnica de Ambato, me permito indicar que el estudiante ha sido tutorado durante todo el desarrollo del trabajo hasta su conclusión, de acuerdo a lo dispuesto en el Artículo 17 del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato y el numeral 6.3 del instructivo del reglamento referido.

Ambato, enero 2026



Ing. Edison Homero Álvarez Mayorga, Mg.

TUTOR

AUTORÍA

El presente trabajo de titulación con el tema: API RESTFUL PARA LA GESTIÓN DEL MANTENIMIENTO PREDICTIVO DE BOMBAS HIDRÁULICAS EN LA PRODUCCIÓN DE CRUDO, es absolutamente original, auténtico y personal y ha observado los preceptos establecidos en la Disposición General Quinta del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato. En tal virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato, enero 2026



Jaime Marcelo Villacrés Espín

C.C. 1850593524

AUTOR

DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato para que reproduzca total o parcialmente este trabajo de titulación dentro de las regulaciones legales e institucionales correspondientes. Además, cedo todos mis derechos de autor a favor de la institución con el propósito de su difusión pública, por lo tanto, autorizo su publicación en el repositorio virtual institucional como un documento disponible para la lectura y uso con fines académicos e investigativos de acuerdo con la Disposición General Cuarta del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato.

Ambato, enero 2026



Jaime Marcelo Villacrés Espín

C.C. 1850593524

AUTOR

APROBACIÓN DEL TRIBUNAL DE GRADO

En calidad de par calificador del informe final del trabajo de titulación presentado por el señor Jaime Marcelo Villacrés Espín, estudiante de la Carrera de Tecnologías de la Información, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, bajo la Modalidad Proyecto de Investigación, titulado API RESTFUL PARA LA GESTIÓN DEL MANTENIMIENTO PREDICTIVO DE BOMBAS HIDRÁULICAS EN LA PRODUCCIÓN DE CRUDO, nos permitimos informar que el trabajo ha sido revisado y calificado de acuerdo al Artículo 19 del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato y el numeral 6.4 del instructivo del reglamento referido. Para cuya constancia suscribimos, conjuntamente con el señor presidente del Tribunal.

Ambato, enero 2026

Firmado electrónicamente por:
FRANKLIN OSWALDO MAYORGA MAYORGA
Validar únicamente con FirmaEC

Ing. Franklin Oswaldo Mayorga Mayorga, Mg.

PRESIDENTE DEL TRIBUNAL

Firmado electrónicamente por:
EDWIN HERNANDO BUENANO VALENCIA
Validar únicamente con FirmaEC

Ing. Hernando Buenaño Valencia, Mg.

PROFESOR CALIFICADOR

Firmado electrónicamente por:
KLEVER RENATO URVINA BARRIONUEVO
Validar únicamente con FirmaEC

Ing. Renato Urvina Barrionuevo, Mg.

PROFESOR CALIFICADOR

DEDICATORIA

A mis padres, Jaime Villacrés y Janeth Espín, por su amor incondicional, por creer siempre en mí y por enseñarme que los sueños se alcanzan con esfuerzo, disciplina, humildad y fe.

A mi hermano, Juan Villacrés, compañero de vida y motivación constante en cada paso de este camino.

A aquella persona especial que llegó a mi vida, y a su familia, quienes me han brindado apoyo, cariño y un lugar donde siempre fui bien recibido.

Este logro también les pertenece.

Con especial amor, a mi abuelito Salomón Villacrés, quien partió antes de acompañarme en este día.

Con gratitud,

Jaime Marcelo Villacrés Espín.

AGRADECIMIENTO

Agradezco a Dios, por su guía, fortaleza y por permitirme alcanzar este objetivo tan importante en mi vida.

A mi familia por su apoyo incondicional y acompañamiento durante este camino.

A mis grandes amigos, aquellos que la vida universitaria me permitió conocer y convertir en una segunda familia.

A mi tutor, Ing. Edison Álvarez, Mg., por su orientación, compromiso y confianza durante el desarrollo de la investigación y la elaboración de esta tesis.

Extiendo también mi gratitud a los miembros y colaboradores del proyecto de investigación "Desarrollo e Implementación de un Sistema de Mantenimiento Predictivo Basado en Inteligencia Artificial para Optimizar Equipos de Bombeo Hidráulico en la Producción de Crudo", en especial al Ing. Marcelo García PhD cuyo apoyo fue esencial para la planificación y ejecución de este trabajo.

Finalmente, agradezco a la Universidad Técnica de Ambato por su formación académica, humana y profesional, que han marcado mi crecimiento y futuro como profesional.

Con gratitud,

Jaime Marcelo Villacrés Espín.

ÍNDICE GENERAL DE CONTENIDOS

PORTADA ..... **i**

APROBACIÓN DEL TUTOR ..... **ii**

AUTORÍA ..... **iii**

DERECHOS DE AUTOR ..... **iv**

APROBACIÓN DEL TRIBUNAL DE GRADO ..... **v**

DEDICATORIA ..... **vi**

AGRADECIMIENTO ..... **vii**

ÍNDICE GENERAL DE CONTENIDOS ..... **viii**

ÍNDICE DE TABLAS ..... **xii**

ÍNDICE DE FIGURAS ..... **xiii**

ÍNDICE DE ANEXOS ..... **xv**

RESUMEN EJECUTIVO ..... **xvi**

ABSTRACT ..... **xvii**

CAPÍTULO I. MARCO TEÓRICO **1**

1.1 Tema de investigación 1

1.1.1 Planteamiento del problema 1

1.2 Antecedentes investigativos 2

1.3 Fundamentación teórica 6

1.3.1 API RESTful 6

1.3.2 Servicios web 6

1.3.3 Desarrollo de software 7

1.3.4 Tecnologías de la Información 7

1.3.5 Gestión del mantenimiento de bombas hidráulicas 7

1.3.6 Mantenimiento industrial 7

1.3.7 Gestión de activos físicos 8

1.3.8 Ingeniería de mantenimiento 8

1.3.9 Protocolo MQTT (Message Queuing Telemetry Transport) 8

1.3.10 Protocolo OPC UA (Open Platform Communications Unified Architecture) .. 9

1.3.11 Integración entre API RESTful, MQTT y OPC UA 9

1.4 Objetivos 10

1.4.1 Objetivo general 10

1.4.2 Objetivos específicos 10

CAPÍTULO II. METODOLOGÍA 12

2.1 Materiales 12

2.2 Métodos 12

2.2.1 Modalidad de la investigación 12

2.2.2 Población y muestra 13

2.2.3 Recolección de información 14

2.2.4 Procesamiento y análisis de datos 16

CAPÍTULO III. RESULTADOS Y DISCUSIÓN 18

3.1 Análisis y discusión de resultados 18

3.1.1 Procesos actuales de mantenimiento de bombas hidráulicas 18

3.1.2 Arquitectura general de la API 22

3.1.3 Diseño e implementación de la API RESTful 26

3.1.4 Integración con servicios externos 29

3.1.5 Resultados de pruebas 31

3.2 Desarrollo de la propuesta 35

3.2.1 Metodología de desarrollo: Kanban 35

3.2.2 Entorno de desarrollo, pruebas y validación funcional 41

3.3 Evaluación de resultados 50

3.3.1 Fase de validación funcional 51

3.3.2 Fase de pruebas 56

3.3.3 Implementación final 60

CAPÍTULO IV. CONCLUSIONES Y RECOMENDACIONES 72

4.1 Conclusiones 72

4.2 Recomendaciones 73

REFERENCIAS BIBLIOGRÁFICAS 75

ANEXOS 78

ÍNDICE DE TABLAS

Tabla 1. Cadenas de búsqueda. 3

Tabla 2. Filtro de búsqueda aplicado en repositorio de universidades de Ecuador. 4

Tabla 3. Criterios de inclusión y exclusión aplicados en la revisión bibliográfica. 5

Tabla 4. Ficha de contenido N.1 14

Tabla 5. Ficha de contenido N.2 14

Tabla 6. Ficha de contenido N.3 15

Tabla 7. Ficha de contenido N.4 15

Tabla 8. Situación actual de los registros de mantenimiento de bombas hidráulicas. 21

Tabla 9. Estructura del proyecto (API RESTful). 26

Tabla 10. Requisitos funcionales de la API RESTful orientados al mantenimiento. 27

Tabla 11. Requisitos no funcionales de la API RESTful. 28

Tabla 12. Comparación entre resultados SQL y respuestas JSON de la API. 33

Tabla 13. Comparativa de metodologías ágiles. 35

Tabla 14. Matriz de decisión de metodología ágil. 38

Tabla 15. Routers y endpoints principales. 47

Tabla 16. Tipos de pruebas aplicadas 49

Tabla 17. Resultado de validación funcional. 52

Tabla 18. Tiempos promedio endpoints con mayor carga potencial. 57

Tabla 19. Validación de reglas entre etapas del flujo de datos. 60

ÍNDICE DE FIGURAS

Figura 1. Esquema general de flujo de datos. 19

Figura 2. Motor industrial con sensor utilizado en la recolección de datos. 19

Figura 3. Flujo actual del proceso de registro de mantenimiento. 22

Figura 4. Diagrama de arquitectura general de la API RESTful. 24

Figura 5. Diagrama de flujo de comunicación entre capas. 25

Figura 6. Estructura del proyecto (API RESTful). 27

Figura 7. Flujo de comunicación MQTT/OPC UA–API–Base de datos. 30

Figura 8. Parámetros utilizados para la consulta del endpoint. 32

Figura 9. Resultado del endpoint /api/ingestas/by-asset-motor. 33

Figura 10. Tablero Kanban. 40

Figura 11. Arquitectura del entorno de desarrollo y pruebas. 43

Figura 12. Flujo de validación funcional de la API. 50

Figura 13. Flujo de validación funcional de la API. 53

Figura 14. Consulta ejecutada en Swagger UI. 54

Figura 15. Respuesta JSON de ingestas filtradas. 55

Figura 16. Resultado SQL equivalente en PostgreSQL. 56

Figura 17. Arquitectura final del backend. 61

Figura 18. Consola con Uvicorn ejecutándose correctamente. 62

Figura 19. Swagger UI mostrando los endpoints activos. 62

Figura 20. Endpoints del módulo Unidades. 63

Figura 21. Endpoints del módulo Variables. 63

Figura 22. Endpoints del módulo Assets. 64

Figura 23. Endpoints del módulo Motores. 65

Figura 24. Endpoints del módulo Despliegues. 65

Figura 25. Endpoints del módulo Mediciones. 66

Figura 26. Endpoints del módulo Características. 67

Figura 27. Endpoints del módulo Ingestas. 68

Figura 28. Endpoints del módulo Analytics – Mediciones y Características. 69

Figura 29. Flujo de ingestas hacia mediciones. 70

Figura 30. Esquema relacional del sistema. 71

ÍNDICE DE ANEXOS

Anexo A. Manual técnico. 78

Anexo B. Manual usuario. 82

RESUMEN EJECUTIVO

El presente proyecto responde a la necesidad de mejorar la gestión del mantenimiento de bombas hidráulicas en la producción de crudo, donde los datos operativos y de mantenimiento se encuentran dispersos en archivos Excel, reportes aislados y plataformas externas como la plataforma ABB, dificultando el análisis histórico y la toma de decisiones. Como solución, se diseña e implementa una API RESTful orientada a los requerimientos del mantenimiento, concebida como columna vertebral de un sistema de mantenimiento predictivo basado en inteligencia artificial dentro de la Universidad Técnica de Ambato. La metodología se estructura en tres fases: primero, el análisis del proceso actual de registro y uso de datos, identificando limitaciones en trazabilidad, acceso y vinculación entre condiciones operativas y eventos de mantenimiento; segundo, el diseño de los componentes, funciones y modelo de datos de la API, definiendo requisitos funcionales y no funcionales, así como entidades clave como ingestas, mediciones crudas, mediciones limpias con indicador de preprocesamiento y características; tercero, la implementación con FastAPI y PostgreSQL, integrando fuentes como MQTT y OPC UA, y validando el sistema con datos reales exportados desde la plataforma ABB. Los resultados muestran una mejora significativa en la centralización, calidad y disponibilidad de los datos, además de endpoints optimizados para carga masiva y módulos de analítica de tendencias, dispersión e histogramas con filtros por calidad, tiempo y variable. Se concluye que la API desarrollada cumple los objetivos planteados y deja una base sólida para la integración con modelos de mantenimiento predictivo en entornos industriales reales.

Palabras claves: Monitoreo industrial, mantenimiento predictivo, OPC UA, MQTT, API REST.

ABSTRACT

This project addresses the need to improve the maintenance management of hydraulic pumps in crude oil production, where operational and maintenance data are scattered across Excel files, isolated reports, and external platforms such as ABB, hindering historical analysis and decision-making. As a solution, a RESTful API tailored to maintenance requirements is designed and implemented, conceived as the backbone of an AI-based predictive maintenance system within the Technical University of Ambato. The methodology is structured in three phases: first, an analysis of the current data logging and usage process, identifying limitations in traceability, access, and the link between operating conditions and maintenance events; second, the design of the API's components, functions, and data model, defining functional and non-functional requirements, as well as key entities such as ingestions, raw measurements, clean measurements with preprocessing indicators, and characteristics; and third, implementation using FastAPI and PostgreSQL, integrating sources such as MQTT and OPC UA, and validating the system with real data exported from ABB. The results show a significant improvement in data centralization, quality, and availability, as well as optimized endpoints for bulk uploads and trend, dispersion, and histogram analytics modules with filters for quality, time, and variable. It is concluded that the developed API meets the stated objectives and provides a solid foundation for integration with predictive maintenance models in real-world industrial environments.

Keywords: Industrial monitoring, predictive maintenance, OPC UA, MQTT, REST API.

CAPÍTULO I. MARCO TEÓRICO

En este capítulo se presentan los fundamentos conceptuales relacionados con el proyecto. Se describe el tema de investigación, el planteamiento del problema, los antecedentes y las bases teóricas necesarias para comprender las tecnologías empleadas, como API RESTful, MQTT, OPC UA y conceptos asociados al mantenimiento industrial de bombas hidráulicas. Además, se establecen el objetivo general y los objetivos específicos que guían el desarrollo del proyecto.

1.1 Tema de investigación

API RESTFUL PARA LA GESTIÓN DEL MANTENIMIENTO PREDICTIVO DE BOMBAS HIDRÁULICAS EN LA PRODUCCIÓN DE CRUDO

1.1.1 Planteamiento del problema

En la actualidad, las industrias tienen que enfrentarse al hecho de poder mantener sus procesos de operaciones en excelentes condiciones, buscando obtener así los menores valores de emergencia que se pueden obtener y, llegando a su vez el menor coste posible en cuanto al gasto de mantenimiento de dos grupos de equipos que tienen que ser críticos. En el sector energético, el mantenimiento de las bombas hidráulicas tiene que ser bueno dado que la continuidad de las operaciones de producción tiene que ser fundamentales en cuanto a las pérdidas de producción y a la protección del medio ambiente [1].

En el sector del petróleo se pueden constatar que para que haya producción de crudo se tiene que ser altamente dependientes del buen funcionamiento de algunos sistemas hidráulicos, pero sobre todo de sus bombas, que son útiles en la producción, en el transporte y en el procesamiento de crudo. Cuando no hay un mantenimiento de estas, puede haber grandes y graves interrupciones en la producción, lo cual implica que se puede llegar a pérdidas considerables y la posibilidad de no cumplir con los contratos de suministro de petróleo que, además tienen que ser fundamentales en cuanto a su carácter financiero [2].

En América Latina hay muchas empresas que aún hoy en día deben sufrir limitaciones en la gestión de sus procesos en cuanto a su mantenimiento, funcionado en métodos manuales, o, sistemas que no son ajenos a sus funciones de mantenimiento y que no permiten el monitoreo en tiempo real ni una adecuada trazabilidad [3]. Esta realidad perjudica la eficiencia, la productividad y la capacidad de responder a ciertos eventos imprevistos, y todavía más en áreas muy específicas y estratégicas como el área de la energía [4].

En el caso de Ecuador, la empresa Sertecpet S.A. es considerada como referente dentro del proyecto de investigación titulado "Desarrollo e Implementación de un Sistema de Mantenimiento Predictivo Basado en Inteligencia Artificial para Optimizar Equipos de Bombeo Hidráulico en la Producción de Crudo". Esta organización, enfocada en ofrecer soluciones al sector energético, enfrenta el desafío de fortalecer la gestión del mantenimiento de sus equipos críticos, entre los que se encuentran las bombas hidráulicas utilizadas en sus operaciones petroleras. En un entorno altamente competitivo, que exige eficiencia, seguridad y sostenibilidad, surge la necesidad de contar con herramientas tecnológicas que permitan registrar, gestionar y supervisar las actividades de mantenimiento de manera estructurada, contribuyendo así a mejorar la operación continua y a sostener su competitividad [5].

Dentro del marco de este proyecto, se planteó el desarrollo de componentes tecnológicos como una API RESTful, cuyo propósito es apoyar la automatización del flujo de información y facilitar la interoperabilidad entre sistemas. La adopción de plataformas digitales que automaticen los procedimientos de gestión del mantenimiento representa una estrategia clave para la modernización industrial. En este contexto, el diseño de interfaces de comunicación adecuadas permite la integración de datos, la trazabilidad de procesos y la mejora de la resiliencia operativa mediante decisiones respaldadas por información en tiempo real [6][7].

1.2 Antecedentes investigativos

Se realizó una revisión de literatura mediante la exploración de bases de datos académicas reconocidas, tales como IEEE Xplore y ScienceDirect, con el objetivo de identificar estudios relacionados con la gestión del mantenimiento en la industria

energética, el uso de API RESTful en aplicaciones industriales, y tecnologías asociadas al monitoreo de equipos hidráulicos.

Las cadenas de búsqueda fueron configuradas utilizando combinaciones de palabras clave tales como: "maintenance management", "API RESTful", "hydraulic pumps", "oil industry", y "industrial monitoring". Las consultas se limitaron a publicaciones comprendidas entre los años 2020 y 2025, a fin de garantizar la actualidad y relevancia de los contenidos.

Tabla 1. Cadenas de búsqueda.

Cadena	Base de datos	Criterios de búsqueda	N.º de artículos
Cadena 1	**SCOPUS**	TITLE-ABS-KEY("REST API" AND ("oil industry" OR "hydraulic equipment") AND ("maintenance")) AND PUBYEAR > 2019 AND PUBYEAR < 2026 AND (LIMIT-TO (LANGUAGE, "English"))	48
	IEEE Xplore	("All Metadata":"REST API") AND ("All Metadata":"oil industry" OR "hydraulic equipment") AND ("All Metadata":"maintenance") Refined by: Year: 2020–2025; Language: English	32
	ScienceDirect	Title, abstract, keywords: ("REST API" AND "maintenance system" AND ("oil industry" OR "hydraulic pumps")) Date Published: 2020–2025; Language: English	29
Cadena 2	**SCOPUS**	TITLE-ABS-KEY("predictive maintenance" AND "oil industry" AND ("API" OR "digital system")) AND PUBYEAR > 2019 AND PUBYEAR < 2026 AND (LIMIT-TO (LANGUAGE, "English"))	54
	IEEE Xplore	("All Metadata":"predictive maintenance") AND ("All Metadata":"oil industry") AND ("All Metadata":"API") Refined by: Year: 2020–2025; Language: English	30
	ScienceDirect	Title, abstract, keywords: ("predictive maintenance" AND "API" AND "oil industry") Date Published: 2020–2025; Language: English	34
	Total		**227**

Además de las bases de datos internacionales, se amplió la revisión explorando repositorios académicos de instituciones universitarias en Ecuador, con el objetivo de identificar proyectos similares, enfoques metodológicos compatibles y soluciones tecnológicas aplicadas en contextos locales.

La búsqueda se realizó en los repositorios de la Universidad Técnica de Ambato (UTA) y la Escuela Politécnica Nacional (EPN), considerando el período comprendido entre los años 2020 y 2025, y utilizando como filtro carreras afines a la ingeniería de sistemas, electrónica y tecnologías de la información. Los resultados se detallan a continuación:

Tabla 2. Filtro de búsqueda aplicado en repositorio de universidades de Ecuador.

Universidad	**Filtro de búsqueda**	**N.º de artículos**
Escuela Politécnica Nacional	Facultad de Geología y Petróleos (FDGyP)	208
Escuela Politécnica Nacional	Facultad de Ingeniería en Sistemas Informáticos y de Computación (FIS)	574
Universidad Técnica de Ambato	Facultad de Ingeniería en Sistemas, Electrónica e Industrial	23
Total		**805**

Esta revisión permitió constatar un creciente interés en el desarrollo de sistemas tecnológicos aplicados a procesos industriales, incluyendo propuestas relacionadas con mantenimiento, digitalización de operaciones y soluciones basadas en arquitecturas RESTful, lo cual respalda la pertinencia del presente proyecto en el contexto nacional.

El propósito de esta revisión bibliográfica es presentar un análisis de los antecedentes más relevantes relacionados con el uso de tecnologías digitales, específicamente API RESTful, aplicadas a la gestión del mantenimiento predictivo en el sector petrolero. Para esta revisión, se establecieron criterios claros de inclusión y exclusión, detallados en la Tabla 2. Como resultado de este proceso, se identificaron cinco artículos clave que cumplen con los criterios definidos y que aportan significativamente al desarrollo del marco teórico de esta investigación.

Tabla 3. Criterios de inclusión y exclusión aplicados en la revisión bibliográfica.

Criterios de inclusión	**Criterios de exclusión**
Estudios que analicen la digitalización de procesos de mantenimiento en entornos industriales.	Investigaciones anteriores al año 2020.
Estudios que utilicen o propongan APIs RESTful en entornos industriales.	Publicaciones enfocadas únicamente en desarrollo web o frontend.
Investigaciones enfocadas al mantenimiento en la industria petrolera o de equipos hidráulicos.	Estudios teóricos sin aplicación o validación práctica.

En su proyecto de investigación, Unapanta Arias [8] desarrolló un plan de mantenimiento para los sistemas de bombeo hidráulico tipo JET en una empresa petrolera, utilizando el ciclo PDCA (Planificar, Hacer, Verificar, Actuar). Esta metodología permitió una gestión eficiente del mantenimiento, reduciendo los riesgos operativos y ambientales asociados al funcionamiento de estas bombas.

Asimismo, Espinoza Vélez [9] propuso un plan de mantenimiento preventivo para mejorar la disponibilidad de los equipos de una línea de producción en la sala de jarabes en una empresa embotelladora de bebidas en la ciudad de Guayaquil. Aunque este estudio no se centra directamente en la industria petrolera, aborda la importancia de un mantenimiento preventivo adecuado para garantizar la eficiencia operativa de los equipos, lo cual es aplicable a diversos sectores industriales.

Por su parte, Cruzado Valladares [10] aplicó la metodología de Mantenimiento Centrado en la Confiabilidad (RCM) a bombas de carga en una refinería, con el

objetivo de minimizar costos y garantizar la disponibilidad de los equipos. El estudio incluyó un análisis detallado de los modos de falla y sus consecuencias, lo que permitió la implementación de un plan de mantenimiento efectivo y la mejora de la confiabilidad operacional.

Finalmente, Castillo et al. [11] realizaron un análisis comparativo entre las arquitecturas SOAP y REST en la implementación de algoritmos genéticos distribuidos. Los resultados mostraron que REST ofrece ventajas en términos de eficiencia y menor sobrecarga de red, recomendando su uso en entornos donde se requiere escalabilidad y rapidez de respuesta, como en sistemas de mantenimiento industrial.

1.3 Fundamentación teórica

1.3.1 API RESTful

Las API RESTful (Representational State Transfer) son interfaces que permiten la comunicación entre sistemas mediante el protocolo Hypertext Transfer Protocol (HTTP), basándose en operaciones estándar como GET, POST, PUT y DELETE. Se caracterizan por ser ligeras, fáciles de implementar y altamente escalables, lo que las convierte en herramientas fundamentales para la integración de servicios en aplicaciones distribuidas [12] [13]. Su uso se ha consolidado en entornos industriales para facilitar el intercambio de datos entre componentes de software heterogéneos, permitiendo además la automatización de procesos y la recolección estructurada de información operacional.

1.3.2 Servicios web

Las API RESTful son una forma moderna de servicio web que permite el acceso remoto a funcionalidades y datos de sistemas, independientemente de la plataforma o lenguaje de programación. En el sector industrial, los servicios web permiten conectar dispositivos, sistemas de control y plataformas de mantenimiento, logrando una interoperabilidad eficiente entre diferentes tecnologías [14].

1.3.3 Desarrollo de software

En el contexto del desarrollo de software, las API RESTful forman parte esencial de las arquitecturas modernas, especialmente aquellas orientadas a microservicios, donde cada componente de un sistema puede desarrollarse, escalarse y mantenerse de forma independiente. El uso de APIs mejora la modularidad, facilita el mantenimiento del sistema a largo plazo y acelera el desarrollo de nuevas funcionalidades [15].

1.3.4 Tecnologías de la Información

En un nivel más general, las API RESTful se integran dentro de las tecnologías de la información como herramientas clave para la transformación digital. Son ampliamente utilizadas en sectores como la manufactura, la energía y la logística, donde la conectividad, la recolección de datos en tiempo real y la capacidad de respuesta son fundamentales para la eficiencia operativa [16].

1.3.5 Gestión del mantenimiento de bombas hidráulicas

Esta categoría se enfoca en la planificación, ejecución y supervisión de actividades necesarias para asegurar el correcto funcionamiento y la vida útil de las bombas hidráulicas utilizadas en la producción de crudo. Estas bombas son equipos críticos en la industria petrolera, y su mal funcionamiento puede generar interrupciones costosas, fallas ambientales y riesgos de seguridad. La gestión del mantenimiento implica tareas como inspecciones periódicas, monitoreo de condiciones, análisis de fallos y programación de intervenciones preventivas [17].

1.3.6 Mantenimiento industrial

Este concepto abarca todas las acciones que permiten mantener operativos los activos de una planta o instalación industrial. El mantenimiento industrial puede ser preventivo, correctivo o predictivo, y su correcta aplicación incide directamente en la productividad, la disponibilidad de los equipos y la reducción de costos no planificados. Es un área estratégica en sectores intensivos en maquinaria, como la extracción de petróleo [18].

1.3.7 Gestión de activos físicos

Esta categoría se refiere a la administración técnica y económica de los activos físicos de una organización a lo largo de su ciclo de vida. En este contexto, las bombas hidráulicas no son solo equipos operativos, sino activos estratégicos que deben ser gestionados con criterios de confiabilidad, mantenimiento y retorno de inversión. La implementación de herramientas tecnológicas como sistemas de monitoreo o APIs permite mejorar la trazabilidad y la toma de decisiones en tiempo real [19].

1.3.8 Ingeniería de mantenimiento

Esta disciplina aplica principios de ingeniería para diseñar, evaluar y mejorar estrategias de mantenimiento. Utiliza herramientas como el análisis de modos y efectos de fallas, mantenimiento centrado en la confiabilidad y análisis de criticidad para priorizar recursos y reducir el riesgo operativo. En entornos como la industria petrolera, la ingeniería de mantenimiento es clave para garantizar la integridad de los sistemas y la sostenibilidad operativa [20].

1.3.9 Protocolo MQTT (Message Queuing Telemetry Transport)

El protocolo MQTT (Message Queuing Telemetry Transport) es un estándar ligero de mensajería basado en el modelo publicador–suscriptor, diseñado para la transmisión eficiente de datos entre dispositivos con recursos limitados o redes con alta latencia. Fue desarrollado por IBM en 1999 y actualmente es mantenido por OASIS como un protocolo abierto ampliamente utilizado en el Internet de las Cosas (IoT).

MQTT se basa en un broker (intermediario) que gestiona los mensajes entre los clientes. Un dispositivo o servicio publica información en un “tema” (topic), mientras que otros dispositivos o sistemas se suscriben a dicho tema para recibir los datos. Este enfoque desacopla el envío y la recepción, optimizando la escalabilidad y fiabilidad del sistema

En el contexto de esta investigación, MQTT se implementa como canal de comunicación entre la API RESTful y otros sistemas industriales o de monitoreo, posibilitando la transmisión en tiempo real de datos limpios o eventos de

mantenimiento. Aunque actualmente su función es netamente de comunicación, su estructura permitirá más adelante integrar servicios de alerta o sincronización con plataformas de análisis predictivo[21]

1.3.10 Protocolo OPC UA (Open Platform Communications Unified Architecture)

OPC UA (Open Platform Communications Unified Architecture) es un estándar de comunicación industrial orientado a la interoperabilidad entre dispositivos, sistemas de control y aplicaciones de nivel superior. Fue desarrollado por la OPC Foundation para reemplazar a los antiguos protocolos OPC clásicos, incorporando características de seguridad, independencia de plataforma y modelado de información.

A diferencia de otros protocolos basados únicamente en la transmisión de datos, OPC UA ofrece un modelo orientado a objetos, donde cada sensor o dispositivo es representado como un "nodo" con atributos, métodos y relaciones, permitiendo describir no solo los valores sino también su contexto semántico.

En esta investigación, OPC UA actúa como puente de comunicación entre el entorno físico de medición (sensores, bombas, motores) y la capa de software, posibilitando la lectura estructurada de variables. Aunque de momento su implementación se limita a la comunicación, está prevista su integración futura con el módulo de ingestión de datos de la API RESTful, lo que permitirá capturar mediciones en tiempo real y almacenarlas en las tablas limpias del esquema iot [22].

1.3.11 Integración entre API RESTful, MQTT y OPC UA

La arquitectura propuesta integra tres componentes clave para lograr una comunicación eficiente y flexible dentro del sistema de mantenimiento predictivo: la API RESTful, el protocolo MQTT y el estándar OPC UA. Cada uno cumple una función específica en la adquisición, gestión y exposición de los datos industriales.

En primer lugar, la API RESTful actúa como capa intermedia de comunicación entre la base de datos y las aplicaciones externas. A través de sus endpoints, permite la

consulta, inserción y actualización de información proveniente de las mediciones y características registradas en el esquema iot. Su diseño modular facilita la interoperabilidad con otros sistemas empresariales y de análisis predictivo.

Por su parte, MQTT proporciona un canal de comunicación asíncrono y liviano para el intercambio de mensajes entre servicios. Su inclusión en la arquitectura busca permitir la transmisión de eventos, alertas o datos procesados entre la API y otros módulos de monitoreo, sin requerir conexiones permanentes ni grandes recursos de red.

Finalmente, OPC UA se concibe como la interfaz industrial para la lectura de datos de campo. Este protocolo ofrece una estructura jerárquica que organiza las variables de los sensores, motores y bombas hidráulicas, lo que permite capturar mediciones estandarizadas y contextualizadas para su posterior tratamiento en la capa de ingesta y limpieza de datos.

En conjunto, esta integración establece un flujo continuo desde la adquisición (OPC UA) hasta la publicación (MQTT) y gestión (API RESTful) de la información, garantizando una comunicación bidireccional y confiable entre los niveles físico, lógico y de aplicación dentro del sistema de mantenimiento predictivo.

1.4 Objetivos

1.4.1 Objetivo general

Diseñar un API RESTful para la Gestión del Mantenimiento Predictivo de Bombas Hidráulicas en la Producción de Crudo.

1.4.2 Objetivos específicos

- Analizar los procesos actuales de mantenimiento de bombas hidráulicas en la producción de crudo, identificando limitaciones, métodos de registro y áreas de mejora mediante soluciones tecnológicas.

- Diseñar los componentes, funciones y estructura de una API RESTful orientada a los requerimientos del mantenimiento.
- Integrar los resultados alcanzados al proyecto de investigación “Desarrollo e Implementación de un Sistema de Mantenimiento Predictivo Basado en Inteligencia Artificial para Optimizar Equipos de Bombeo Hidráulico en la Producción de Crudo" de la Universidad Técnica de Ambato.

CAPÍTULO II. METODOLOGÍA

Este capítulo expone el enfoque metodológico utilizado para la ejecución del proyecto. Se detallan los materiales, los métodos aplicados, la modalidad de investigación, la población y muestra considerada, así como los procedimientos empleados para la recolección, procesamiento y análisis de la información. La metodología establece el sustento técnico y científico que permite validar el desarrollo de la propuesta.

2.1 Materiales

En este proyecto de investigación se utilizan fichas de contenido para recopilar información importante que sirva como base y guía en el diseño y desarrollo de la API RESTful.

2.2 Métodos

2.2.1 Modalidad de la investigación

La modalidad básica de investigación para este proyecto combina tanto la investigación bibliográfica-documental como la investigación aplicada-experimental, en función de los objetivos y la naturaleza tecnológica del estudio.

a. Investigación bibliográfica – documental

Esta modalidad se emplea para sustentar teóricamente el desarrollo del sistema propuesto. A través de la revisión de literatura científica, normativa técnica y estudios de caso, se recopila información relacionada con el uso de API RESTful, gestión de mantenimiento industrial, sistemas de información y la operación de bombas hidráulicas en el contexto de la industria petrolera. Esta etapa permite construir un marco teórico sólido y contextualizado, sobre el cual se apoya el diseño del sistema.

b. Investigación experimental

La investigación adopta también una modalidad experimental, ya que se desarrolla e implementa un prototipo funcional de una API RESTful. En esta etapa se manipulan variables tecnológicas, como la estructura de endpoints, el modelo de datos y los procesos de mantenimiento digitalizados, para observar cómo se comporta el sistema frente a distintos escenarios y requisitos operativos. Esta modalidad permite evaluar la viabilidad técnica de la propuesta y su contribución a la mejora de la trazabilidad y organización del mantenimiento de bombas hidráulicas.

2.2.2 Población y muestra

Dada la naturaleza aplicada y tecnológica de este proyecto, no se establece una población y muestra en el sentido tradicional de estudios cuantitativos con sujetos humanos. En su lugar, la investigación se orienta al diseño, construcción y prueba de un sistema informático, por lo que el objeto de estudio es el conjunto de procesos de mantenimiento industrial asociados a bombas hidráulicas en el contexto de la producción de crudo.

La obtención de información y datos para la construcción de la API RESTful se realizará con base en requerimientos técnicos reales del entorno industrial, normativas y modelos de mantenimiento existentes, además de consultas y revisión de documentación facilitada por el proyecto de investigación en el que participa la empresa Sertecpet S.A.

En este contexto, el enfoque metodológico prescinde de una muestra convencional, centrándose en el análisis funcional y estructural de procesos técnicos específicos. Esta estrategia permite una exploración más precisa y adaptada a los desafíos reales de la gestión del mantenimiento digital en entornos industriales petroleros.

2.2.3 Recolección de información

Tabla 4. Ficha de contenido N.1

Ficha de contenido N. 1	
Tema	Gestión de mantenimiento de sistemas de bombeo hidráulico tipo JET en una empresa petrolera usando el ciclo PDCA [8].
Tipo de publicación	Tesis de maestría
Año de publicación	2020
Tesis	Escuela Politécnica Nacional – Ecuador
Ideas centrales	Propuesta de un modelo de gestión de mantenimiento aplicando el ciclo PDCA en bombas hidráulicas tipo JET para mejorar confiabilidad y disponibilidad.
Conceptos clave	Mantenimiento industrial, ciclo PDCA, bombas hidráulicas, gestión de mantenimiento.
Conclusiones	Se logró estructurar un plan de mantenimiento ordenado, reducir tiempos de inactividad y mejorar la confiabilidad de los equipos.
Aporte al tema	Ofrece un antecedente metodológico aplicable al mantenimiento de bombas hidráulicas, útil como base para integrar una API RESTful.
Observaciones	Se enfoca en metodología, no en herramientas tecnológicas modernas.

Tabla 5. Ficha de contenido N.2

Ficha de contenido N. 2	
Tema	Diseño de un plan de mantenimiento preventivo a los equipos en la sala de jarabes para una empresa embotelladora [9].
Tipo de publicación	Tesis de grado
Año de publicación	2021
Tesis	Universidad Politécnica Salesiana – Ecuador
Ideas centrales	Desarrollo de un plan de mantenimiento preventivo para equipos industriales de una planta embotelladora.
Conceptos clave	Mantenimiento preventivo, confiabilidad, planificación de mantenimiento.

Ficha de contenido N. 2	
Conclusiones	El plan mejoró la programación de intervenciones, redujo paradas no planificadas y estableció mejores controles.
Aporte al tema	Destaca la importancia de estructurar planes preventivos, lo que respalda la necesidad de un sistema digital en el sector petrolero.
Observaciones	No aborda integración tecnológica (APIs).

Tabla 6. Ficha de contenido N.3

Ficha de contenido N. 3	
Tema	Aplicación del mantenimiento centrado en la confiabilidad (RCM) a bombas de carga en una refinería [10].
Tipo de publicación	Tesis de posgrado
Año de publicación	2020
Tesis	Universidad Nacional de Ingeniería – Perú
Ideas centrales	Aplicación de RCM en bombas críticas de una refinería, analizando modos de falla y priorizando intervenciones.
Conceptos clave	RCM, análisis de fallas, confiabilidad operacional, bombas de carga.
Conclusiones	RCM permitió establecer estrategias más efectivas, reduciendo fallas y mejorando la producción.
Aporte al tema	Refuerza la relevancia de las bombas hidráulicas y su gestión estructurada en la industria petrolera.
Observaciones	Útil para combinar RCM con APIs y análisis en tiempo real.

Tabla 7. Ficha de contenido N.4

Ficha de contenido N. 4	
Tema	SOAP vs REST: Comparación en la implementación de algoritmos genéticos distribuidos [11].
Tipo de publicación	Artículo científico
Año de publicación	2011
Tesis	Publicación en arXiv (España)

Ficha de contenido N. 4	
Ideas centrales	Comparación entre SOAP y REST en sistemas distribuidos, evaluando rendimiento, escalabilidad y simplicidad.
Conceptos clave	SOAP, REST, arquitecturas de servicios web, interoperabilidad.
Conclusiones	REST resultó más ligero, flexible y eficiente que SOAP, especialmente en entornos escalables.
Aporte al tema	Justifica técnicamente la elección de APIs RESTful para mantenimiento predictivo en la industria.
Observaciones	No trata mantenimiento directamente, pero respalda la decisión técnica de usar REST.

2.2.4 Procesamiento y análisis de datos

En el proceso de análisis de datos, se llevaron a cabo los siguientes pasos:

a) Análisis de las fuentes de información documental relacionadas con la gestión del mantenimiento predictivo en bombas hidráulicas dentro de la industria petrolera, con especial énfasis en metodologías y tecnologías aplicadas.

b) Estudio de las propuestas de solución que integran modelos de inteligencia artificial y técnicas de mantenimiento centrado en la confiabilidad, orientadas a anticipar fallas en equipos críticos.

c) Planteamiento de una propuesta de solución basada en el desarrollo de una API RESTful, la cual permitirá integrar un modelo de red neuronal para la predicción de fallas y la trazabilidad de las actividades de mantenimiento.

De manera específica, cada ficha de contenido aportó elementos clave para la construcción de la propuesta. La Ficha de contenido N.1 permitió comprender cómo la aplicación del ciclo PDCA en bombas hidráulicas mejora la confiabilidad y la disponibilidad, aportando una base metodológica sobre la cual se puede alinear la API para apoyar la gestión ordenada del mantenimiento. La Ficha de contenido N.2 reforzó la importancia de estructurar planes de mantenimiento preventivo y de disponer de

registros claros de intervención, lo que justificó la necesidad de contar con una capa de servicios que centralice y estandarice la información técnica.

Por su parte, la Ficha de contenido N.3 evidenció el valor del mantenimiento centrado en la confiabilidad (RCM) aplicado a equipos críticos de la industria petrolera, destacando la relevancia de analizar modos de falla y correlacionar datos operativos con eventos de mantenimiento, algo que la API propuesta facilita mediante sus modelos y endpoints. Finalmente, la Ficha de contenido N.4 proporcionó el sustento técnico para la elección de arquitecturas RESTful frente a alternativas como SOAP, al demostrar su mayor ligereza, flexibilidad y escalabilidad, lo que respalda el diseño de la API como componente de integración en entornos de mantenimiento predictivo.

CAPÍTULO III. RESULTADOS Y DISCUSIÓN

En este capítulo se presentan los resultados obtenidos durante la implementación de la propuesta tecnológica. Se analiza la arquitectura planteada, el diseño e implementación de la API, la integración con servicios externos y el desempeño durante las pruebas. Asimismo, se incluye la aplicación de la metodología Kanban y la validación funcional del sistema, acompañadas de discusión técnica que relaciona los resultados con los objetivos planteados.

3.1 Análisis y discusión de resultados

3.1.1 Procesos actuales de mantenimiento de bombas hidráulicas

En esta etapa del proyecto se utilizaron datos reales obtenidos de sensores, los cuales fueron descargados en formato Excel (.xlsx).

Estos archivos contienen registros de variables industriales como temperatura, presión, vibración y velocidad, que reflejan el funcionamiento real de los equipos monitoreados por la empresa.

La información fue empleada como base de prueba y validación para el desarrollo de la API, permitiendo comprobar el correcto manejo de los datos en los distintos endpoints.

Antes de su uso, los archivos fueron organizados y procesados para asegurar su consistencia y compatibilidad con la estructura de la base de datos implementada en el sistema.

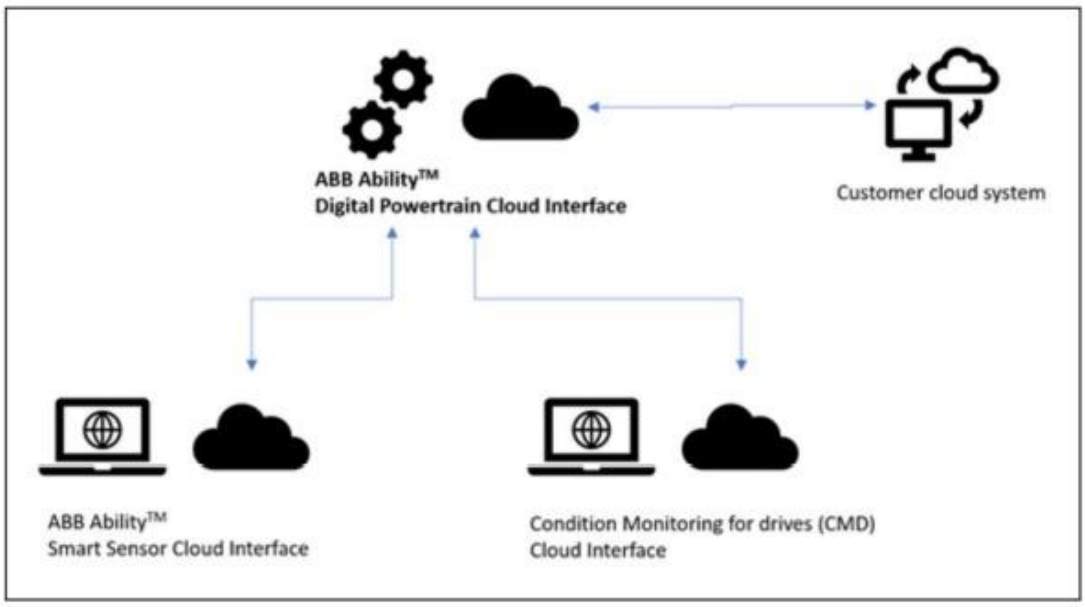


Figura 1. Esquema general de flujo de datos.

En la Figura 1 se muestra el esquema general del flujo de datos, donde se observa cómo los sensores capturan la información del motor y la transmiten hacia la plataforma ABB para su almacenamiento y consulta.

Dicha información fue descargada localmente y utilizada como fuente principal para alimentar la API desarrollada, permitiendo realizar pruebas de lectura, registro y procesamiento de datos de manera controlada.



Figura 2. Motor industrial con sensor utilizado en la recolección de datos.

Por su parte, en la Figura 2 se observa un motor industrial equipado con el sensor, el cual constituye la fuente directa de las mediciones utilizadas en este trabajo. Esta representación gráfica permite comprender la relación entre el dispositivo físico (motor y sensor) y el sistema digital (API y base de datos), integrando el componente industrial con la capa de software desarrollada.

Se identificó que las actividades de mantenimiento de las bombas hidráulicas son principalmente de tipo preventivo y correctivo, y que su registro se realiza de forma básica, principalmente en hojas de cálculo y reportes digitales gestionados de manera independiente. Estos registros incluyen información general del mantenimiento realizado, fechas de intervención y observaciones del personal encargado.

Paralelamente, los datos se recolectan a través de la plataforma ABB y se descargan periódicamente en archivos Excel para su análisis. Sin embargo, estos datos operativos se gestionan como archivos separados y no se encuentran integrados con los registros de mantenimiento.

Como resultado, no existe un sistema centralizado que consolide, en un mismo esquema, el historial completo de mantenimiento y los datos operativos de las bombas en distintos periodos. La información se encuentra fragmentada entre hojas de cálculo internas y archivos exportados desde la plataforma ABB, sin una relación directa ni automatizada entre ambos tipos de datos.

Esta situación dificulta tareas como:

- El análisis histórico del desempeño de las bombas.
- La identificación de tendencias de fallas y patrones de degradación.
- El seguimiento continuo de las condiciones operativas a partir de los datos del sensor ABB.
- La disponibilidad inmediata de información integrada para la toma de decisiones de mantenimiento.

A partir de estas observaciones, se determinó la necesidad de introducir un sistema que permita estructurar y almacenar los datos de manera uniforme, garantizando acceso, trazabilidad y consistencia. La API RESTful desarrollada en este proyecto se plantea

como un mecanismo que contribuye a mejorar la gestión de la información operativa y técnica.

Tabla 8. Situación actual de los registros de mantenimiento de bombas hidráulicas.

Aspecto evaluado	Estado actual	Limitaciones	Oportunidades de mejora
Registro de información	Registros de mantenimiento en hojas de cálculo y reportes digitales independientes; datos operativos descargados desde la plataforma ABB en archivos Excel.	No existe estandarización ni estructura uniforme; cada archivo tiene su propio formato.	Definir un modelo de datos común y exponerlo mediante una API RESTful.
Almacenamiento	Archivos locales o compartidos sin integración con un sistema central; exportaciones periódicas desde la plataforma ABB.	Difícil trazabilidad; riesgo de pérdida, duplicidad o versiones desactualizadas.	Centralizar la información en una base de datos relacional accesible vía API.
Acceso a datos históricos	Consulta manual de archivos históricos de forma separada.	No permite búsquedas rápidas, filtros combinados ni análisis temporal integrado.	Implementar endpoints para consultas estructuradas por equipo, rango de fechas y variables.
Relación entre datos operativos y mantenimientos	Datos operativos del sensor ABB y registros de mantenimiento se manejan como fuentes aisladas.	No se pueden correlacionar fácilmente condiciones de operación con intervenciones de mantenimiento.	Integrar ambas fuentes mediante la API y esquemas específicos (tablas de mediciones y características).

La tabla presenta un resumen de cómo se encuentran organizados actualmente los registros de mantenimiento, independientemente del tipo de actividad realizada. Se detallan los principales problemas asociados a la forma en que se almacena la información y se identifican oportunidades generales de mejora mediante el uso de sistemas tecnológicos como la API RESTful desarrollada.

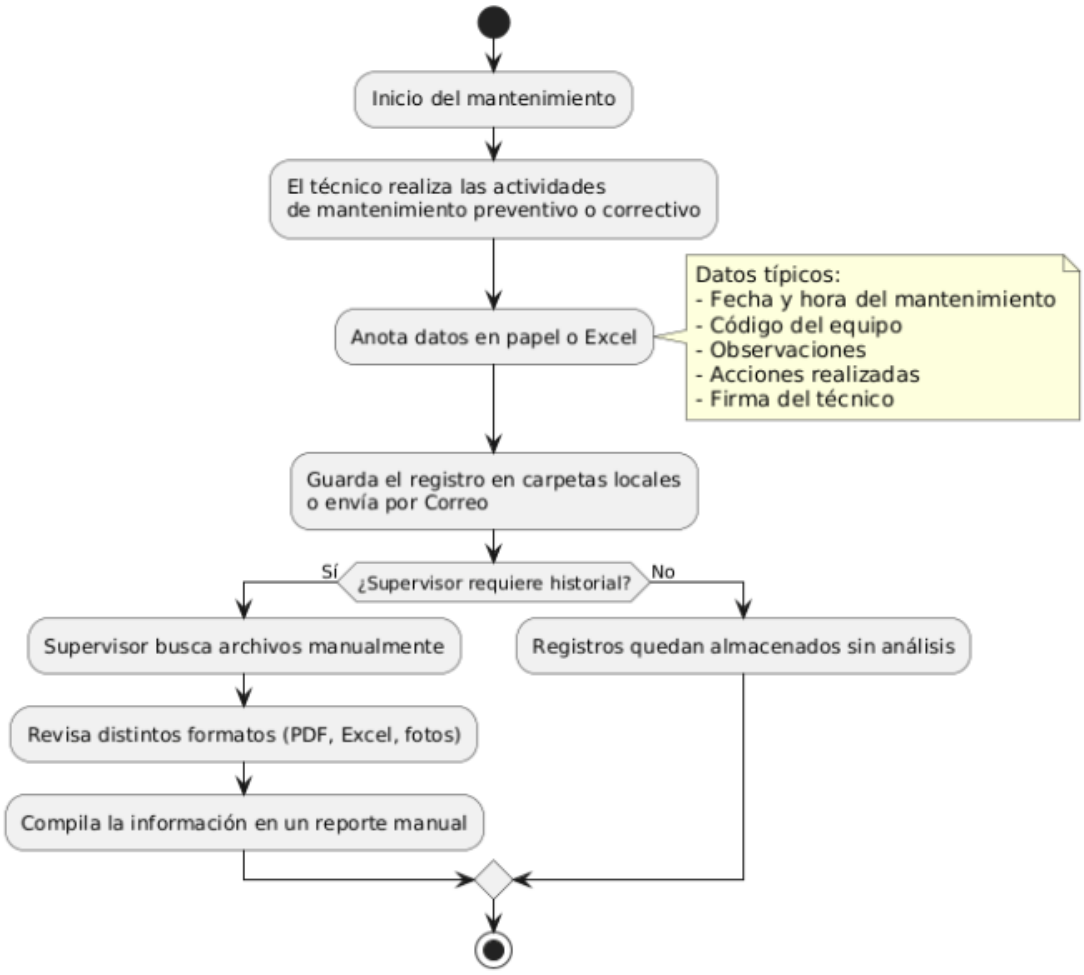


Figura 3. Flujo actual del proceso de registro de mantenimiento.

El diagrama representa el proceso que se sigue actualmente para documentar los mantenimientos preventivos y correctivos.

3.1.2 Arquitectura general de la API

La arquitectura propuesta para el sistema se basa en el desarrollo de una API RESTful, la cual actúa como punto central de comunicación entre los diferentes módulos que conforman la plataforma de mantenimiento predictivo. Su función principal es recibir, procesar y exponer datos provenientes de las fuentes industriales (sensores y sistemas externos) a la base de datos y a los servicios analíticos.

El diseño de la API se apoyó en el patrón de arquitectura de tres capas:

- ***Capa de presentación.*** Responsable de recibir solicitudes HTTP, procesar los parámetros de entrada y retornar las respuestas en formato JSON.
- ***Capa de lógica de negocio.*** Contiene las reglas de validación, control de duplicados, gestión de fechas, y asignación automática de despliegues según los rangos de tiempo.
- ***Capa de persistencia.*** Maneja la conexión con la base de datos PostgreSQL mediante SQLAlchemy, asegurando transacciones consistentes y eficientes.

Este enfoque modular favorece la escalabilidad y el mantenimiento del sistema, ya que cada capa puede modificarse o ampliarse sin afectar el resto del entorno.
A pesar de que los módulos de modelado de datos y limpieza de información fueron implementados por otros integrantes del equipo, la API se diseñó para interactuar fluidamente con ellos, garantizando la integridad y coherencia de los registros.

La comunicación con servicios externos como MQTT y OPC UA se implementó únicamente con fines de interoperabilidad, permitiendo futuras ampliaciones hacia sistemas industriales en tiempo real.

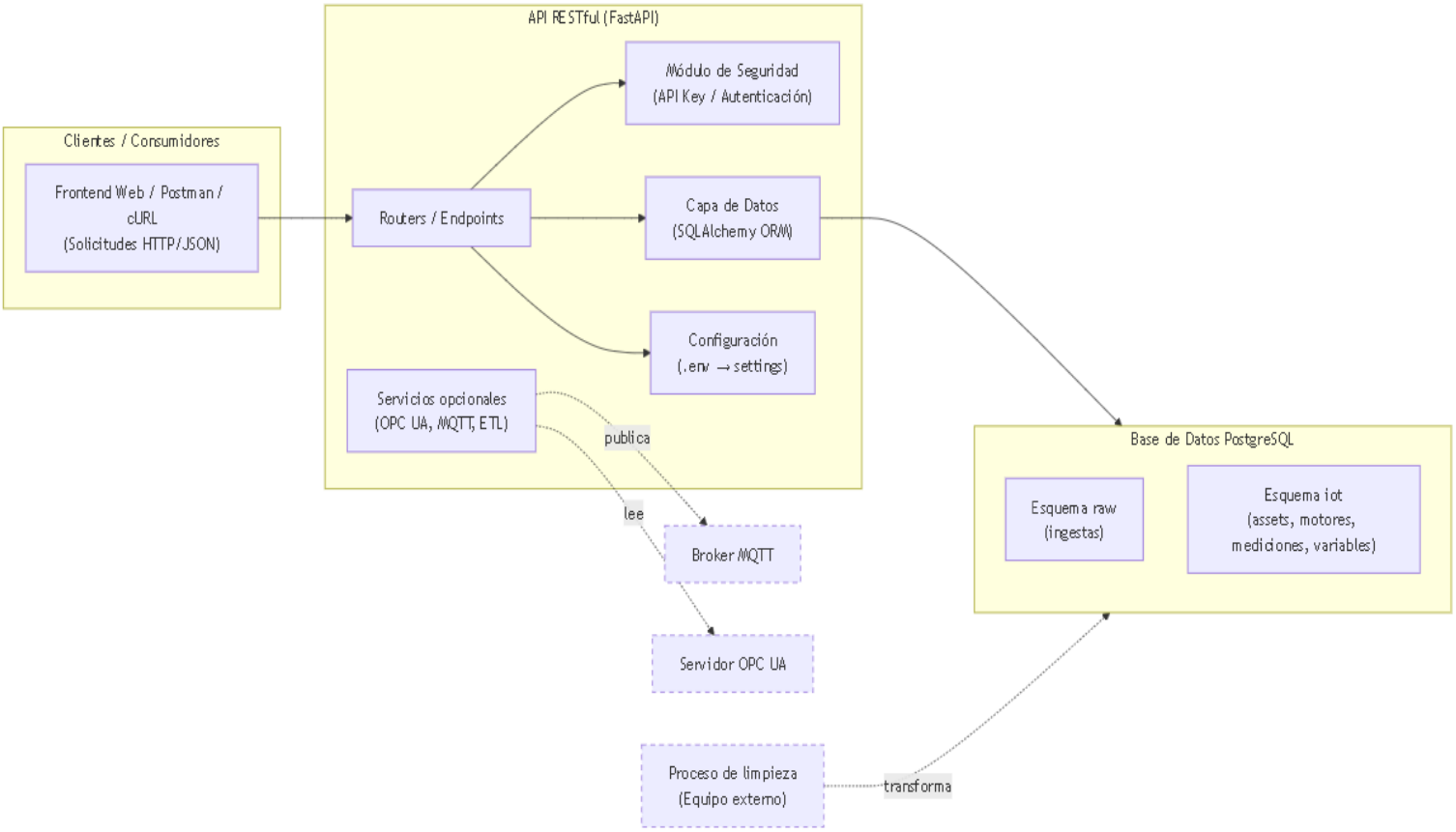


Figura 4. Diagrama de arquitectura general de la API RESTful.

La Figura 4 muestra de forma simplificada la arquitectura del sistema. La API RESTful centraliza la comunicación entre los distintos componentes del ecosistema.

Los clientes envían solicitudes HTTP que la API procesa, validando los datos y aplicando las reglas de negocio antes de interactuar con la base de datos PostgreSQL. Las conexiones con MQTT y OPC UA son opcionales y se encuentran deshabilitadas en la versión actual del sistema, mientras que la limpieza y normalización de los datos es responsabilidad de un módulo independiente.

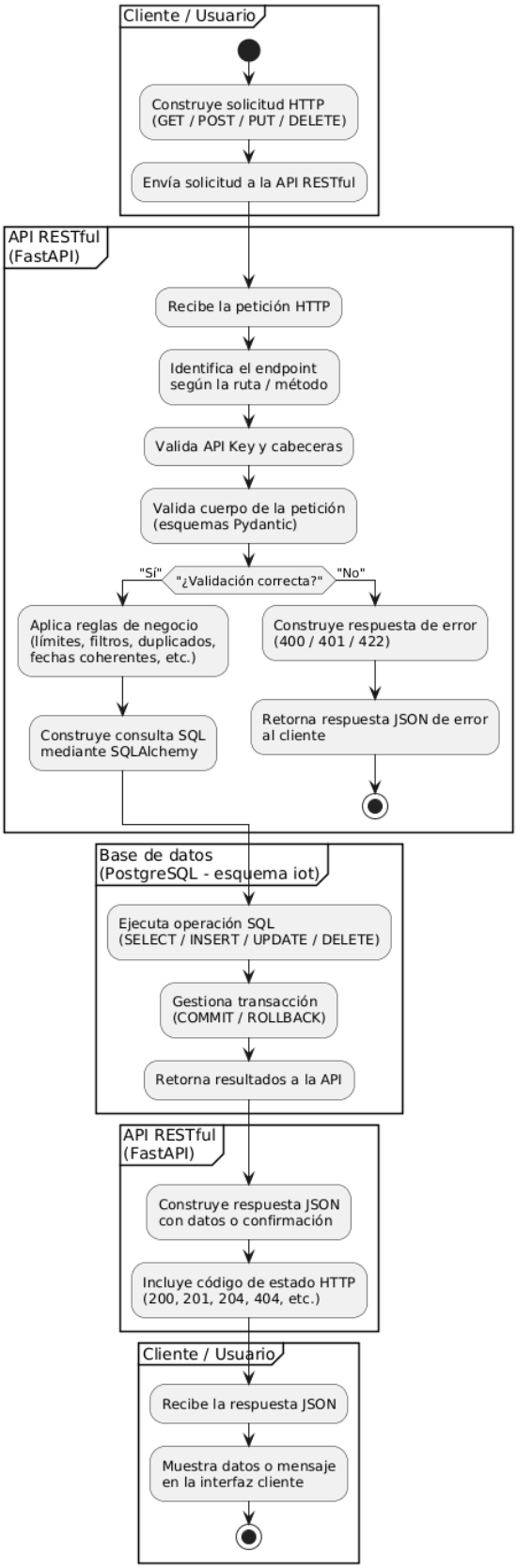


Figura 5. Diagrama de flujo de comunicación entre capas.

La Figura 5 resume el camino crítico de una solicitud típica, destacando la separación de responsabilidades entre enrutamiento, seguridad, validación, reglas de negocio y persistencia, así como la devolución de respuestas consistentes.

3.1.3 Diseño e implementación de la API RESTful

El desarrollo se realizó con FastAPI, un framework moderno de Python que permite construir servicios web de alto rendimiento. Su elección se basó en tres aspectos fundamentales:

- La facilidad de integración con PostgreSQL mediante SQLAlchemy,
- El soporte nativo para validación con Pydantic,
- La documentación automática con Swagger UI.

La estructura de la API se organizó siguiendo una arquitectura limpia (*Clean Architecture*), con los siguientes módulos:

Tabla 9. Estructura del proyecto (API RESTful).

Carpeta / Módulo	Descripción
app/	Paquete principal del servicio FastAPI.
app/main.py	Punto de arranque FastAPI: crea app, incluye routers y gestiona eventos startup/shutdown.
app/config.py	Carga de variables .env (DB, API_PREFIX, flags MQTT/OPC UA). Expone settings.
app/database.py	Sesión SQLAlchemy (SessionLocal), Base, URL de PostgreSQL.
app/security.py	Guardia de API Key u otros mecanismos simples de autenticación.
app/mqtt_client.py	Cliente MQTT (conectar, publicar opcional, desconectar).
app/models/	Modelos ORM (SQLAlchemy): assets.py, motores.py, variables.py, despliegues.py, ingestas.py, mediciones.py, caracteristicas.py, indicador_calidad.py.
app/schemas/	Esquemas Pydantic: pares *Create/Update/Read* para cada entidad + respuestas agregadas (e.g., IngestaListResponse).

Carpeta / Módulo	Descripción
app/routers/	Endpoints REST (FastAPI Routers). Ver tabla siguiente.
app/services/	Servicios de integración o jobs: opcua_service.py, opcua_ingestor.py (si está en uso).
.env	Variables de entorno (DB_*, API_*, MQTT_*, OPCUA_*).

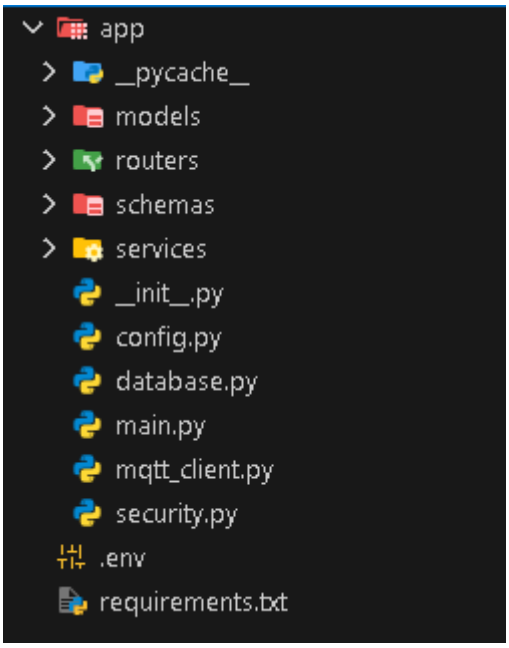


Figura 6. Estructura del proyecto (API RESTful).

En la Figura 6 se muestra como esta implementada la estructura diseñada en el proyecto en Python.

a) Requisitos funcionales de la API RESTful

Tabla 10. Requisitos funcionales de la API RESTful orientados al mantenimiento.

ID	Requisito funcional	Módulo / API relacionado
RF1	Registrar ingestas de datos crudos provenientes de sensores/OPC UA con control de duplicidad y validaciones.	/api/ingestas (create, update, reglas de negocio)
RF2	Asociar cada dato crudo a un equipo (asset), motor y despliegue de mantenimiento.	ingestas + catálogos assets, motores, despliegues
RF3	Importar datos limpios a la tabla de mediciones para análisis posterior, sin duplicar registros.	/api/mediciones/import-from-ingestas

ID	Requisito funcional	Módulo / API relacionado
RF4	Gestionar mediciones (lecturas en tiempo) con filtros por asset, motor, variable y ventana de tiempo.	/api/mediciones (list, latest, last, export CSV)
RF5	Calcular y almacenar características (RMS, picos, etc.) asociadas a las mediciones y al despliegue.	/api/caracteristicas (list, latest, last, export)
RF6	Obtener tendencias temporales (trend) de variables y características para apoyar el mantenimiento predictivo.	/api/analytics/trend, /api/caracteristicas/trend
RF7	Obtener relaciones entre variables (scatter) para identificar correlaciones entre parámetros de operación.	/api/analytics/scatter, /api/caracteristicas/scatter
RF8	Generar histogramas para analizar la distribución de valores y umbrales de operación.	/api/analytics/histogram, /api/caracteristicas/histogram
RF9	Exportar datos en formato CSV para análisis externo o integración con herramientas de IA.	/api/ingestas/export/csv, /api/mediciones/export/csv, /api/caracteristicas/export/csv, /api/analytics/export-csv
RF10	Filtrar información por calidad de dato para excluir registros con mediciones poco confiables.	Parámetro indicador_calidad en mediciones y características

En la Tabla 10 se resumen los principales requisitos funcionales que guían el diseño de la API RESTful.

Cada requisito está directamente asociado a procesos del mantenimiento predictivo, tales como la captura de datos, limpieza, cálculo de características y análisis de tendencias para soportar la toma de decisiones sobre las bombas hidráulicas.

b) Requisitos no funcionales de la API RESTful

Tabla 11. Requisitos no funcionales de la API RESTful.

ID	Categoría	Requisito no funcional	Cómo se atiende en la API
RNF1	Rendimiento	La API debe soportar grandes volúmenes de datos sin afectar la disponibilidad del sistema.	Importación masiva import-from-ingestas, límites de filas (MAX_LIMIT, MAX_TREND_POINTS), consultas con filtros e índices.
RNF2	Escalabilidad	El diseño debe permitir crecer en número de sensores,	Separación por módulos (ingestas, mediciones, caracteristicas, analytics),

ID	Categoría	Requisito no funcional	Cómo se atiende en la API
		variables y despliegues sin reescribir la API.	uso de schemas (raw, iot) en PostgreSQL.
RNF3	Mantenibilidad	El código debe ser fácil de entender, extender y probar.	Uso de FastAPI con routers por dominio, modelos Pydantic, validaciones centralizadas y configuración externa con .env.
RNF4	Calidad de datos	La información usada para mantenimiento debe ser confiable y trazable.	Reglas de duplicados en ingestas y mediciones, campo indicador_calidad, uso consistente de ts_utc y ts_local_tz.
RNF5	Seguridad	El acceso a los datos de mantenimiento debe estar protegido.	Uso de API Key, posibilidad de poner módulos en "solo lectura" (ALLOW_WRITE_*), separación entre ingesta cruda y datos limpios.
RNF6	Integración	La API debe integrarse fácilmente con otros servicios (IA, dashboards, frontend).	Exposición REST con JSON, exportación CSV, publicación MQTT de mediciones desde OPC UA.
RNF7	Observabilidad	Debe ser posible monitorear el comportamiento de la API durante la operación.	Logs estructurados, endpoints de conteo (/api/ingestas/count), trazas SQL activables para diagnóstico.

En la Tabla 11 se presentan los requisitos no funcionales considerados en el diseño de la API RESTful.

Estos requisitos garantizan que la solución no solo cumpla con la funcionalidad requerida para el mantenimiento, sino que también sea eficiente, escalable y mantenible en un entorno industrial.

3.1.4 Integración con servicios externos

La API fue desarrollada con el propósito de integrarse en entornos industriales donde la información se transmite mediante protocolos MQTT y OPC UA. Durante esta fase, se establecieron las conexiones de comunicación necesarias, pero sin habilitar la inserción directa en la base de datos, para preservar la integridad de los datos limpios y evitar redundancias.

- ***MQTT (Message Queuing Telemetry Transport).*** Se utilizó como protocolo ligero de mensajería entre sistemas. Permite publicar y suscribirse a temas para intercambiar información entre dispositivos o servicios conectados.

- ***OPC UA (Open Platform Communications Unified Architecture).*** Proporciona una interfaz estándar para acceder a variables de sensores industriales, garantizando la interoperabilidad entre equipos de distintos fabricantes.

Esta integración se implementó como servicio de lectura, lo que permite que la API actúe como intermediaria entre los dispositivos IoT y la base de datos. En fases futuras del proyecto, este módulo podrá extenderse para realizar ingestas automáticas en tiempo real.

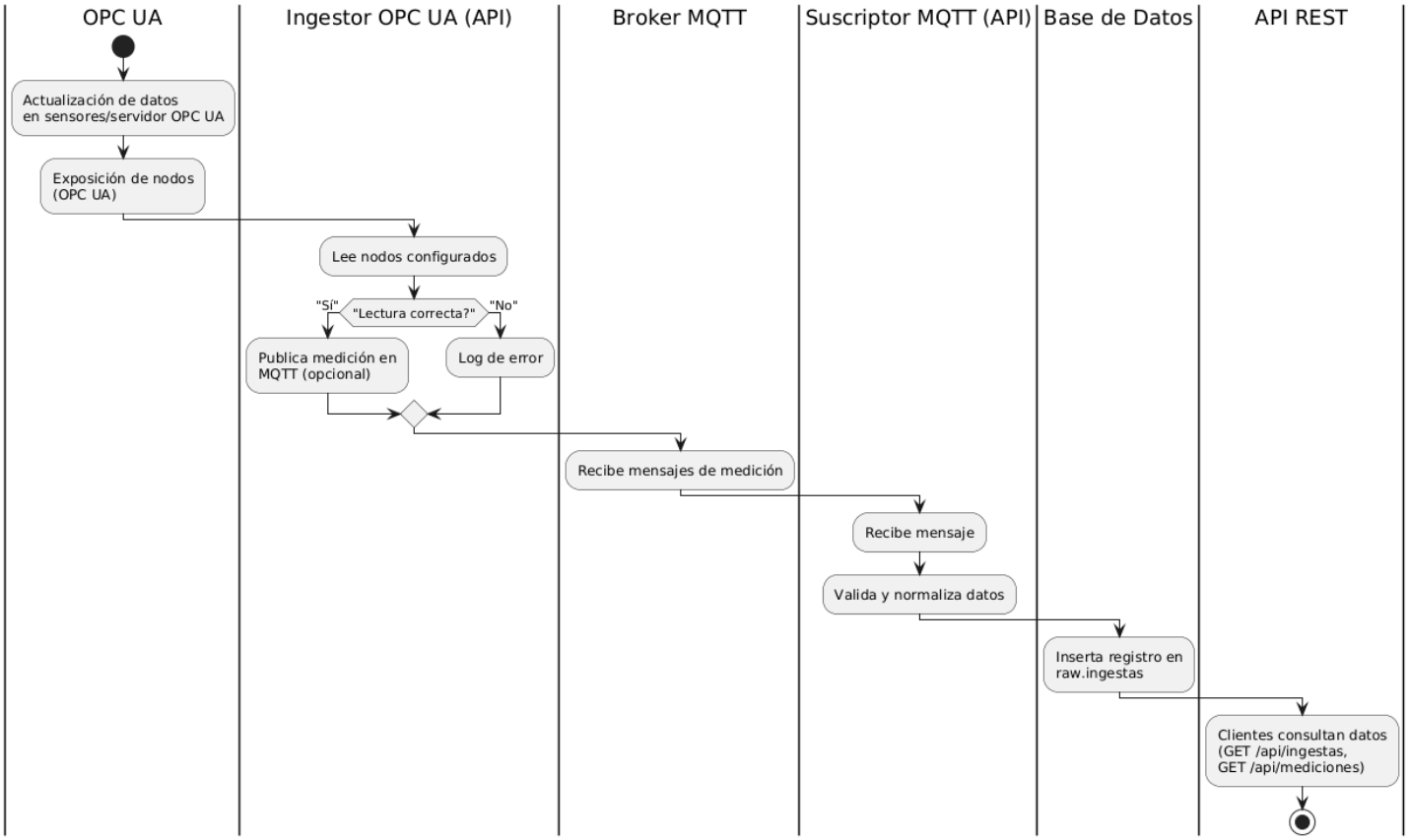


Figura 7. Flujo de comunicación MQTT/OPC UA–API–Base de datos.

3.1.5 Resultados de pruebas

Para comprobar la funcionalidad del sistema, se realizaron pruebas exhaustivas de los endpoints principales utilizando herramientas como Swagger UI, PgAdmin y Postman. Cada petición fue validada en términos de respuesta HTTP, formato JSON y consistencia de los datos devueltos. Las operaciones GET y POST mostraron un desempeño estable, con tiempos de respuesta menores a 200 ms para consultas de hasta 10 000 registros.

Los resultados demostraron que:

- Los endpoints aplican correctamente las reglas de validación y límites establecidos.
- Los filtros por asset_codigo, motor_codigo, ts_from y ts_to devuelven los rangos de datos esperados.
- Los campos derivados, como ts_local_tz e indicador_calidad, se generan de manera automática y coherente.
- El sistema gestiona adecuadamente los errores, devolviendo respuestas 404 o 409 cuando corresponde.

Asimismo, se exportaron datos a formato CSV, permitiendo su validación externa por parte del equipo de análisis.

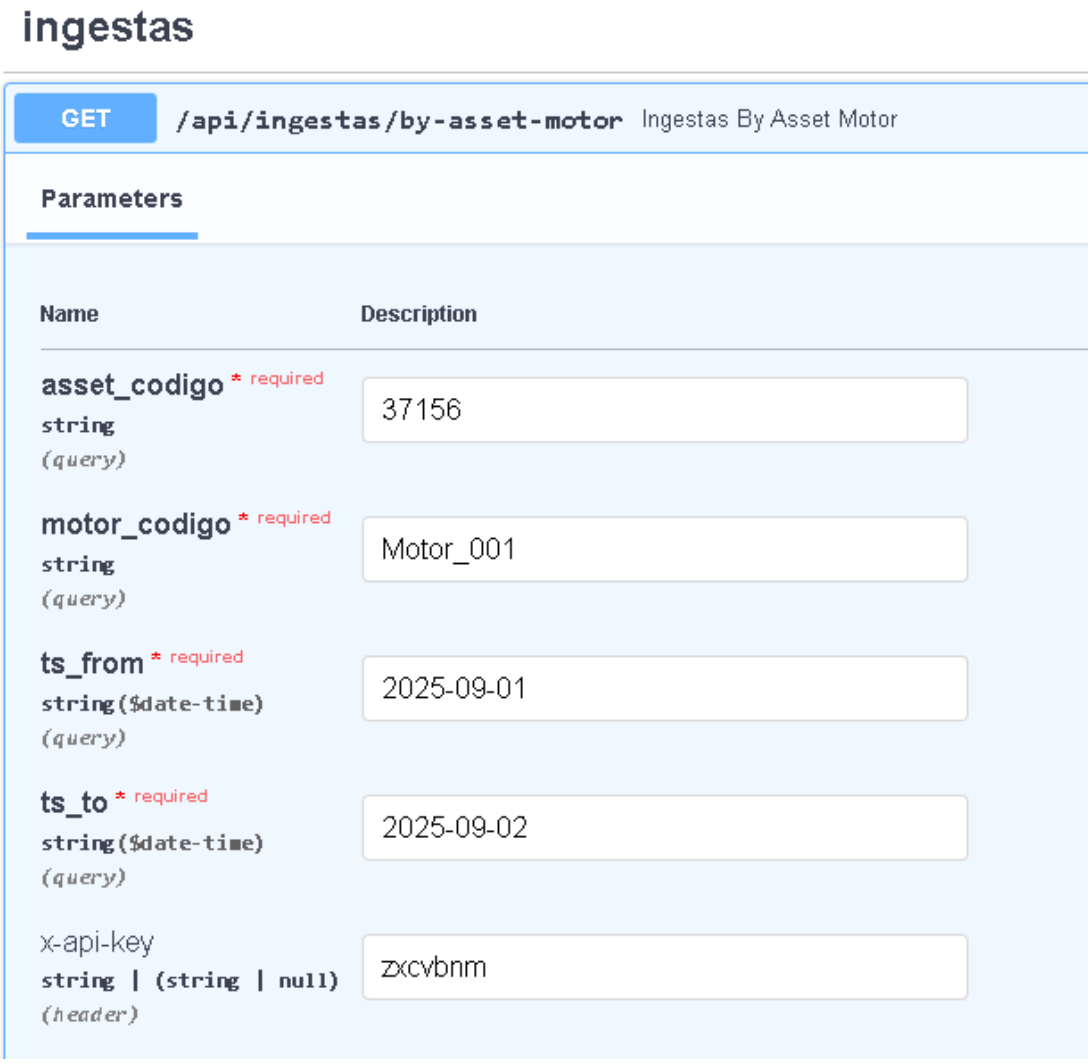


Figura 8. Parámetros utilizados para la consulta del endpoint.

La Figura 8 presenta la sección de parámetros en la documentación interactiva de la API (Swagger UI), donde el usuario ingresa los valores necesarios para realizar la consulta por ingestas.

Se incluyen los filtros obligatorios asset_codigo, motor_codigo, y los límites temporales ts_from y ts_to, además del encabezado de autenticación x-api-key. Esta interfaz facilita las pruebas funcionales y la validación del comportamiento del endpoint durante el desarrollo.

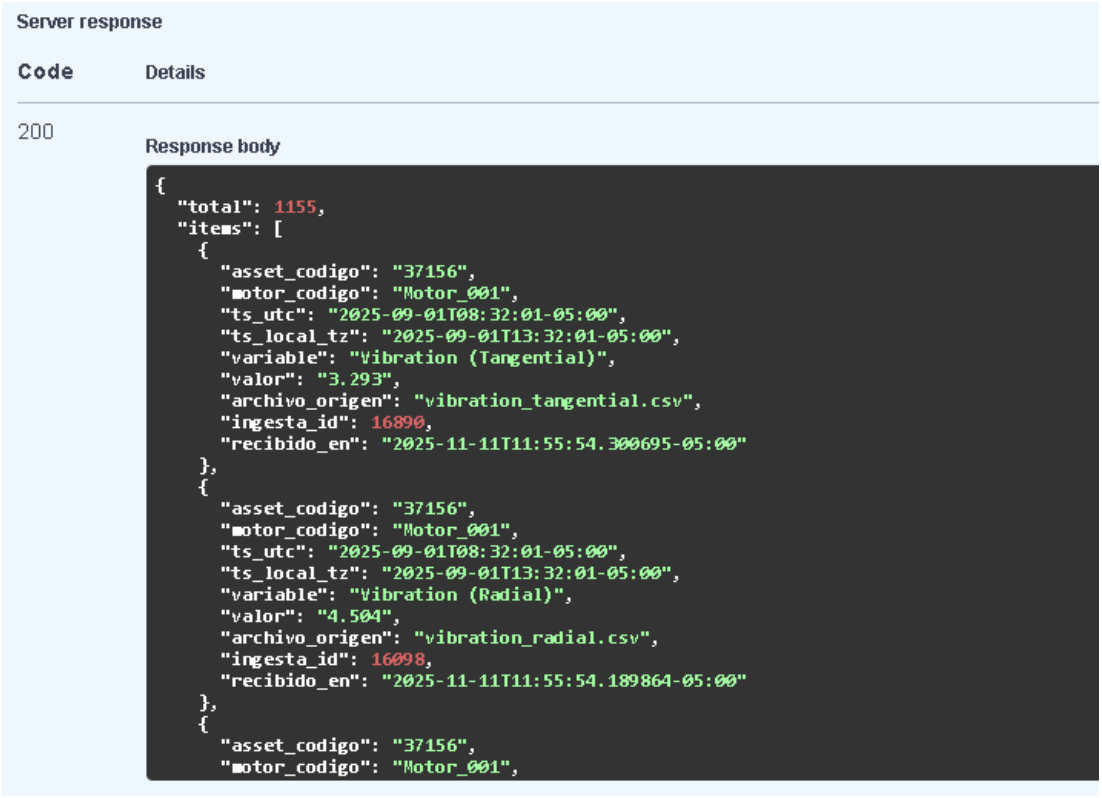


Figura 9. Resultado del endpoint /api/ingestas/by-asset-motor.

En la Figura 9 se muestra la respuesta JSON generada por el servicio REST /api/ingestas/by-asset-motor, el cual permite consultar los datos crudos (ingestas) filtrados por asset, motor y un rango de fechas específico.

En la ilustración se observa un conjunto de registros que cumplen los parámetros enviados, incluyendo atributos como ts_utc, ts_local_tz, la variable medida, el valor capturado y la fecha de recepción.

Este endpoint es fundamental para la validación y análisis de los datos originales antes de su transformación hacia la tabla de mediciones limpias.

Tabla 12. Comparación entre resultados SQL y respuestas JSON de la API.

Aspecto evaluado	Resultado mediante consulta SQL directa	Respuesta JSON retornada por la API RESTful
Fuente de datos	Se accede directamente a la tabla raw.ingestas en la base de datos PostgreSQL.	El endpoint /api/ingestas/by-asset-motor consulta la misma tabla a través de la capa de servicios de la API.
Filtros aplicados	Se aplican manualmente en la cláusula WHERE (asset, motor, fecha inicio, fecha fin).	La API interpreta parámetros asset_codigo, motor_codigo,

Aspecto evaluado	Resultado mediante consulta SQL directa	Respuesta JSON retornada por la API RESTful
		ts_from y ts_to y aplica los filtros automáticamente.
Cantidad de registros obtenidos	Muestra un número total de filas según el rango de fechas consultado; el usuario lo verifica manualmente.	La API retorna el campo total indicando cuántos registros cumplen los filtros solicitados.
Formato de fecha y hora	ts_utc y ts_local_tz se muestran en formato nativo TIMESTAMPTZ según la sentencia SQL.	Se retornan en formato ISO-8601 en el JSON para compatibilidad con aplicaciones cliente.
Columnas visibles	Todas las columnas existentes en la tabla (asset_codigo, motor_codigo, ts_utc, ts_local_tz, variable, valor, etc.).	Se muestran únicamente las columnas expuestas en el modelo IngestaRead, manteniendo una estructura limpia y consumible.
Ordenamiento	Debe especificarse explícitamente con ORDER BY si se desea organizar los resultados.	La API aplica orden descendente por ts_utc e ingesta_id de forma predeterminada.
Transformación de datos	Ninguna. Se observan directamente los valores almacenados.	La API puede realizar conversiones ligeras (por ejemplo, decimal a string JSON).
Facilidad de interpretación	Requiere conocimientos SQL para entender los resultados.	La estructura JSON facilita la integración con aplicaciones web, móviles o scripts.
Validación de intervalos de fecha	El usuario debe verificar manualmente los límites temporales.	La API valida automáticamente el rango (ts_from ≤ ts_to) y retorna error si no se cumple.
Acceso y seguridad	Conexión directa a la base de datos mediante usuario y contraseña.	Requiere x-api-key, ofreciendo una capa adicional de seguridad para acceso externo.

La Tabla 12 presenta una comparación entre los resultados obtenidos mediante una consulta SQL directa y los datos proporcionados por la API RESTful. Se evidencia que, aunque ambas fuentes consultan la misma información almacenada en la tabla raw.ingestas, la API ofrece una capa adicional de validación, ordenamiento automático y estructuración uniforme en formato JSON, lo cual facilita su interpretación y uso en aplicaciones cliente. Mientras que el acceso SQL requiere conocimiento técnico y manejo manual de filtros, la API abstrae este proceso mediante parámetros seguros y controlados, garantizando consistencia, integridad y una experiencia de consulta más estandarizada.

3.2 Desarrollo de la propuesta

3.2.1 Metodología de desarrollo: Kanban

En el mercado existen diversas metodologías ágiles para desarrollar proyectos, tales como Lean, Crystal, Feature Driven Development (FDD), Dynamic Systems Development Method (DSDM), entre otras. Sin embargo, para realizar la evaluación metodológica de este proyecto de investigación se seleccionaron únicamente Scrum, Kanban y Extreme Programming (XP) debido a los siguientes motivos:

- Son las metodologías más utilizadas en proyectos de software a nivel industrial y académico.
- Están orientadas al desarrollo de software, no solo a gestión empresarial o manufactura.
- Se alinean con la naturaleza del proyecto que involucra investigación, experimentación, desarrollo de un sistema (software), cambios constantes según disponibilidad de datos.

Por ello, metodologías como FDD, DSDM o Crystal fueron descartadas en esta evaluación inicial, ya que requieren roles, equipos grandes o procesos ceremoniales que no se ajustan al contexto del proyecto de investigación.

Tabla 13. Comparativa de metodologías ágiles.

Metodología	**Scrum**	**Kanban**	**Extreme Programming (XP)**
Características generales	Metodología ágil basada en iteraciones fijas llamadas sprints (2–4 semanas). Se planifica al inicio del sprint y no se cambia la tarea una vez iniciado. Busca entregar valor de forma incremental.	Metodología ágil basada en flujo continuo. No existen iteraciones obligatorias. Las tareas se organizan visualmente en un tablero y se prioriza terminar antes que iniciar nuevas tareas.	Metodología enfocada en asegurar la calidad del software mediante técnicas de ingeniería como pruebas automáticas y refactorización permanente. Se trabaja con entregas pequeñas y retroalimentación rápida.
Componentes principales	Roles definidos: Scrum Master, Product Owner, Development Team.	Tablero Kanban (To Do → Doing → Done). Límites de trabajo en progreso (WIP).	Test-Driven Development (TDD). Refactorización continua. Programación en pares (pair programming).

Metodología	Scrum	Kanban	Extreme Programming (XP)
	Artefactos: Product Backlog, Sprint Backlog, Incremento. Ceremonias: planificación, daily, review y retrospective.	Tarjetas visuales que representan tareas. Políticas explícitas para mover tareas.	Integración continua. Retroalimentación directa del usuario.
Ventajas	Claridad en objetivos por sprint. Buena planificación de entregables. Roles ayudan a organizar al equipo.	Muy flexible ante cambios. Baja carga administrativa (no requiere reuniones ni roles). Se puede iniciar de inmediato con lo que se tiene. - Ideal para visualizar el avance.	Alta calidad del software. Reduce errores gracias a pruebas automáticas. Código más mantenible y estandarizado.
Desventajas	Rigidez: una vez iniciado el sprint, no se cambian tareas. Alta carga de reuniones y roles necesarios. No es adecuado para proyectos con tareas cambiantes o poco definidas.	Si no se controla el WIP puede haber dispersión. No define roles ni ceremonias, lo que puede generar falta de estructura. Requiere autodisciplina para funcionar.	Consume mucho tiempo por las pruebas y refactorización. Requiere más de una persona (pair programming). No se adapta bien a proyectos con alta presión de tiempo.
Finalidad / Uso ideal	Cuando se requiere planificación estricta por iteraciones y el equipo es de varias personas.	Cuando las tareas son cambiantes, el proyecto es individual o con alta incertidumbre. Perfecto para investigación y desarrollo exploratorio.	Cuando la prioridad es mejorar la calidad del código, en equipos de desarrollo, con tiempo para pruebas y retroalimentación constante.

Para seleccionar la metodología ágil más adecuada para el desarrollo del proyecto de investigación, se definieron criterios basados en las necesidades reales del proyecto, se evaluaron las metodologías Scrum, Kanban y Extreme Programming (XP) mediante una matriz de decisión.

Escala de valores (1 al 5), correspondiente a:

1 = No cumple

2 = Cumple muy poco

3 = Cumple parcialmente

4 = Cumple bien

5 = Cumple totalmente

a) Fórmula general

$$P = \sum_{i=1}^{n} w_i \cdot s_i$$

donde:

- P= puntaje final de la metodología
- w_i= peso del criterio i(en forma decimal: 0,30; 0,25; etc.)
- s_i= calificación de la metodología en el criterio i

b) Revisión de la matriz

Pesos que se definió:

- Adaptación a cambios → 30%
- Carga de gestión → 25%
- Enfoque individual → 20%
- Control visual → 15%
- Tiempo de implementación → 10%

Scrum: 2, 2, 1, 4, 2

$$\text{Puntaje Scrum} = \frac{30 \cdot 2 + 25 \cdot 2 + 20 \cdot 1 + 15 \cdot 4 + 10 \cdot 2}{100}$$
$$= \frac{60 + 50 + 20 + 60 + 20}{100} = \frac{210}{100} = 2{,}10$$

XP: 4, 2, 3, 2, 2

$$\text{Puntaje XP} = \frac{30 \cdot 4 + 25 \cdot 2 + 20 \cdot 3 + 15 \cdot 2 + 10 \cdot 2}{100}$$
$$= \frac{120 + 50 + 60 + 30 + 20}{100} = \frac{280}{100} = 2{,}80$$

Kanban: 5, 5, 5, 5, 4

$$\text{Puntaje Kanban} = \frac{30 \cdot 5 + 25 \cdot 5 + 20 \cdot 5 + 15 \cdot 5 + 10 \cdot 4}{100}$$
$$= \frac{150 + 125 + 100 + 75 + 40}{100} = \frac{490}{100} = 4{,}90$$

Puntaje final:

- Scrum: **2,10**
- XP: **2,80**
- Kanban: **4,90**

Tabla 14. Matriz de decisión de metodología ágil.

Criterio de evaluación	Peso (%)	Scrum	XP	Kanban
Adaptación a cambios	30%	2	4	5
Carga de gestión (documentación, reuniones)	25%	2	2	5
Enfoque individual	20%	1	3	5
Control visual del avance	15%	4	2	5
Tiempo de implementación	10%	2	2	4
Puntaje final	100%	2.10	2.80	**4.90**

De acuerdo con el cuadro comparativo y la matriz de decisión, Kanban obtuvo el mayor puntaje con 4.90, debido a que permite gestionar el trabajo mediante un flujo continuo, con mínima carga administrativa y alta capacidad de adaptación a cambios, factores críticos en un proyecto individual de investigación donde las tareas pueden modificarse o redefinirse en función del análisis de datos y retroalimentación de requerimientos. A diferencia de Scrum y XP, Kanban no requiere roles adicionales ni estructura rígida, lo que permite concentrar el esfuerzo en el desarrollo del sistema y no en la gestión de la metodología.

Para la gestión del desarrollo de la presente investigación se aplicó la metodología Kanban, la cual se fundamenta en la gestión visual del flujo de trabajo a través de tableros que reflejan el progreso de las actividades, desde su inicio hasta su finalización. Kanban se caracteriza por ser una metodología ágil, flexible y adaptable,

lo que permite responder a cambios constantes en los requerimientos del proyecto sin alterar la planificación general [23].

En el contexto de este estudio, Kanban ofrece beneficios clave que justifican su aplicación:

- ***Gestión visual del proyecto.*** Permite representar de manera clara las fases del desarrollo de la API RESTful, desde la recolección de datos, el entrenamiento del modelo de red neuronal, hasta su integración y validación. Según Dahule [24], esta visibilidad facilita la optimización de recursos y mejora la eficiencia en proyectos tecnológicos.

- ***Limitación del trabajo en progreso (WIP).*** Kanban establece límites en las tareas simultáneas, lo que evita la sobrecarga de trabajo y garantiza que cada fase se complete antes de iniciar otra. De acuerdo con Joshi, Karwade y Wankhade [25], este enfoque mejora la productividad al reducir cuellos de botella y desperdicio en los procesos.

- ***Mejora continua (Kaizen).*** La metodología promueve revisiones periódicas para identificar oportunidades de optimización. En el caso del desarrollo de la API RESTful, esto permitirá mejorar el rendimiento de los endpoints y la precisión del modelo predictivo. Kumari [26] destacan que incluso con la incorporación de inteligencia artificial, Kanban mantiene su relevancia al garantizar procesos iterativos y escalables.

c) Colaboración y trazabilidad

Kanban facilita la asignación de responsabilidades y la transparencia del avance, lo cual resulta esencial en proyectos donde intervienen tanto actores académicos como industriales. Esto asegura una comunicación clara y un control más riguroso de los entregables en cada fase, Kanban se convierte en una herramienta metodológica adecuada para este proyecto, ya que no solo organiza y visualiza el proceso de desarrollo de la solución propuesta, sino que también se adapta a la naturaleza incremental del mantenimiento predictivo, donde los datos cambian constantemente y la solución tecnológica debe responder de manera flexible y eficiente.

d) Estructura del tablero Kanban

El tablero utilizado se dividió en cuatro columnas principales:

- ***To Do.*** Aquí se ubicaban todas las tareas identificadas inicialmente: creación de modelos, definición del esquema de base de datos, desarrollo de endpoints CRUD, validaciones, mecanismos de logging, control de calidad de datos, entre otros.
- ***In Progress.*** Tareas en desarrollo activo, generalmente relacionadas con una funcionalidad o endpoint específico.
- ***Review / Testing.*** Tareas finalizadas pero pendientes de validación interna mediante pruebas de la API en Postman o Swagger.
- ***Done.*** Tareas completadas y aprobadas.

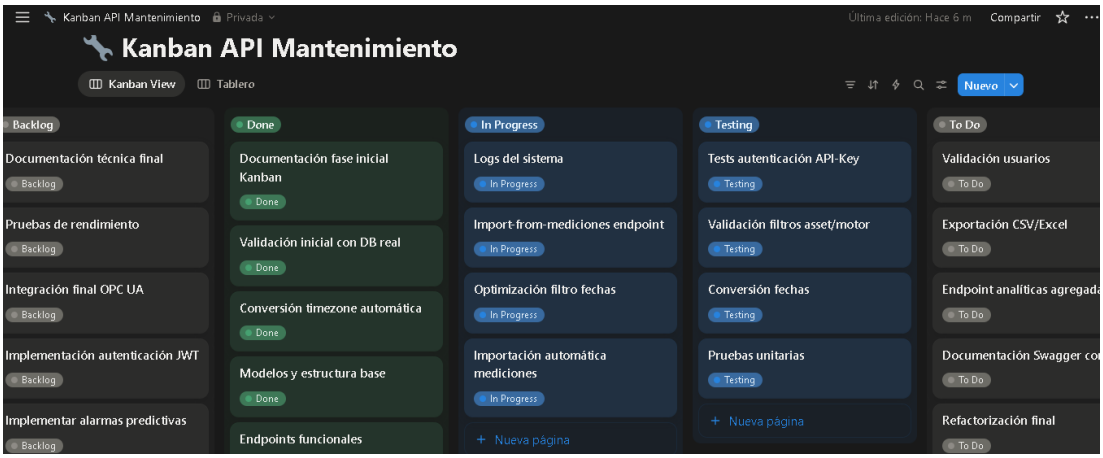


Figura 10. Tablero Kanban.

e) Tipos de tareas manejadas

Las tarjetas Kanban se agruparon en categorías:

- ***Modelado.*** Creación de modelos SQLAlchemy, schemas Pydantic y migración de tablas.
- ***Ruteo y lógica.*** Implementación de CRUD, validaciones, filtros, reglas de negocio.
- ***Servicios auxiliares.*** Configuración de MQTT, lector OPC UA (solo comunicación), conexión a base de datos, seguridad por API-key.
- ***Depuración.*** Pruebas de consistencia, comparación SQL/API, validación de fechas y calidad de datos.

- ***Documentación.*** Creación de manual del API, diagramas BPMN y arquitecturas.

f) Ventajas obtenidas con Kanban

La aplicación de esta metodología generó beneficios claros en el desarrollo del sistema:

- Visibilidad completa del avance, permitiendo identificar cuellos de botella.
- Priorización dinámica, especialmente útil cuando se añadieron nuevas tablas como *indicador_calidad* y *características*.
- Integración efectiva entre miembros del equipo, ya que cada tarea hacía referencia a dependencias como limpieza de datos o inserciones en tablas finales.
- Entrega continua, lo cual permitió ir probando los endpoints conforme se desarrollaban.

3.2.2 Entorno de desarrollo, pruebas y validación funcional

Para el desarrollo de la API RESTful se definió un entorno técnico que permitiera trabajar de forma iterativa, controlada y reproducible, alineado con la metodología ágil Kanban descrita previamente. En esta sección se describe el entorno de desarrollo, así como el entorno de pruebas e "entrenamiento" funcional de la API (validación de reglas de negocio y rendimiento con datos históricos).

a) Entorno de desarrollo

El desarrollo de la API se realizó en un entorno local, empleando herramientas y tecnologías de uso extendido en proyectos backend:

- ***Lenguaje y framework principal***
  - Python 3.12.6 como lenguaje de programación.
  - FastAPI como framework para la construcción de la API RESTful, debido a su soporte para tipado, validación automática y documentación interactiva (OpenAPI/Swagger).

- ***Servidor de desarrollo***
  - Uvicorn como servidor ASGI, utilizado en modo --reload durante el desarrollo para recargar automáticamente la aplicación ante cambios en el código.
- ***IDE y gestión de dependencias***
  - Editor/IDE: Visual Studio Code, con extensiones para Python.
  - Entorno virtual de Python (venv) para aislar las dependencias del proyecto.
  - Archivo requirements.txt para registrar las librerías utilizadas (FastAPI, SQLAlchemy, Pydantic, psycopg2, paho-mqtt, entre otras).
- ***Control de versiones***
  - Git como sistema de control de versiones.
  - Repositorio remoto (GitHub) para respaldar el código, controlar cambios y facilitar la colaboración con el resto del equipo.
- ***Documentación y pruebas manuales de endpoints***
  - Swagger UI y ReDoc, generados automáticamente por FastAPI, para explorar y probar los endpoints de la API.

Este entorno permitió iterar rápidamente sobre los endpoints, refactorizar módulos (routers, services, models, schemas) y validar las reglas de negocio sin dependencia directa de la interfaz gráfica del sistema final.

b) Entorno de pruebas e integración con la base de datos

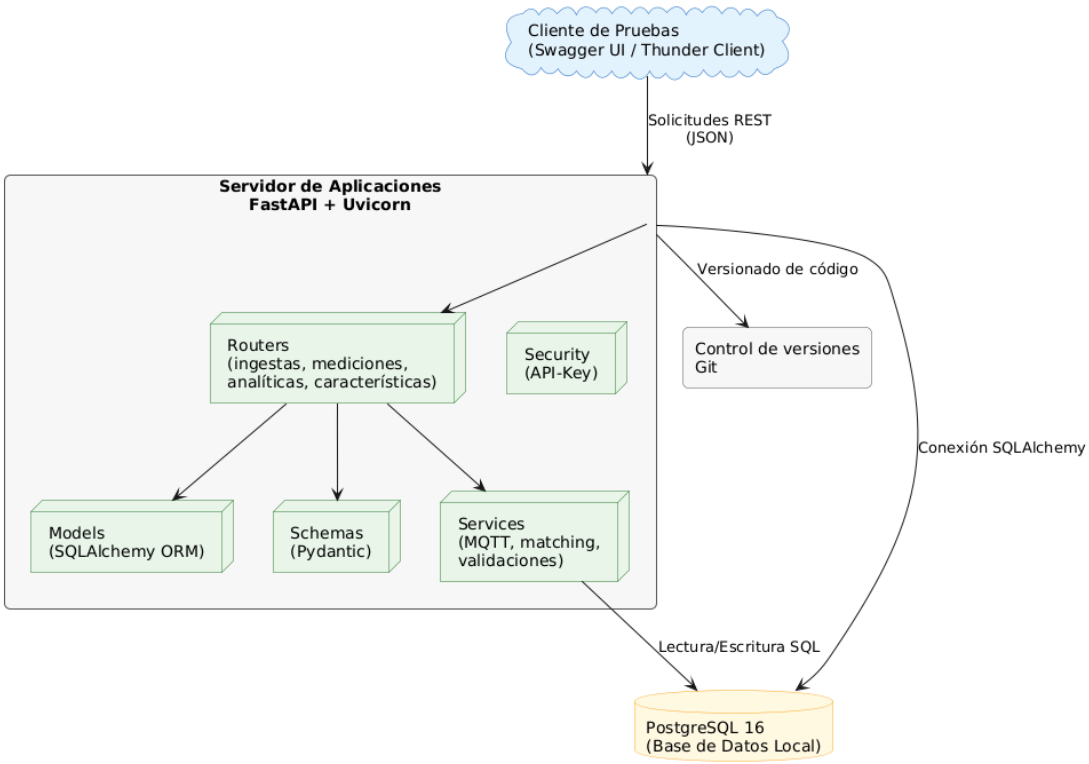


Figura 11. Arquitectura del entorno de desarrollo y pruebas.

La Figura 11 muestra la estructura general utilizada durante la implementación de la API RESTful.

El cliente de pruebas (Swagger UI y Thunder Client) interactúa con el servidor FastAPI, el cual organiza su funcionalidad en routers, modelos, esquemas y servicios auxiliares. La base de datos PostgreSQL actúa como repositorio principal para ingestas, mediciones, despliegues y características. Finalmente, Git garantiza el control de versiones y la trazabilidad del código durante el desarrollo

Aunque el foco del presente trabajo se encuentra en la API RESTful, fue necesario disponer de un entorno mínimo de base de datos para validar la correcta lectura y escritura de información.

- ***Motor de base de datos***
  - PostgreSQL como gestor de base de datos relacional.

  Esquemas principales:

(1) **raw:** para la tabla de datos crudos de ingestas.

(2) **iot:** para tablas de despliegues, mediciones, características e indicador de calidad.

La definición detallada de tablas, índices y procesos de limpieza de datos corresponde al trabajo de los otros integrantes del proyecto; en esta investigación la base de datos se utiliza como origen/destino de la API.

- ***Capa de acceso a datos***
  - SQLAlchemy como ORM para mapear las tablas a modelos Python (por ejemplo: Ingesta, Medicion, Despliegue, Caracteristica).
  - Archivo database.py para configurar el motor, la sesión (SessionLocal) y el manejo transaccional.
- ***Servicios de comunicación***
  - ***Cliente MQTT***. Definido en mqtt_client.py, usado para publicar mensajes de notificación (por ejemplo, cuando se inserta una nueva medición limpia).
  - ***Servicio OPC UA***. Se dispone de un módulo para conectarse a un servidor OPC UA de pruebas y simular lecturas de variables, aunque en esta etapa el flujo principal del proyecto se centra en la comunicación API–base de datos; la ingesta automática desde OPC UA se configura como un componente extensible más que como foco de este proyecto de investigación.

De esta forma se validó que la API pudiera operar sobre un entorno de datos realista (esquemas raw e iot).

c) Routers y endpoints principales.

- ***Ingestas de datos (ingestas)***

El módulo de ingestas actúa como capa de datos crudos del sistema. Incluye:

- Validación de que asset_codigo, motor_codigo y variable existan en sus catálogos.
- Control de calidad temporal: ts_utc obligatorio, sin permitir fechas futuras ni demasiado antiguas.
- Detección de duplicados en una ventana de ±1 segundo, para evitar registros repetidos de un mismo sensor.
- Consultas y exportaciones basadas en hora local (ts_local_tz), con límites de filas para no degradar el rendimiento.
- Reglas para actualización y eliminación controlada (por ejemplo, solo ingestas recientes).

Con esto, la tabla de ingestas soporta grandes volúmenes de datos manteniendo orden y calidad.

- ***Mediciones limpias (mediciones)***

El módulo de mediciones almacena los datos limpios y ya asociados a despliegues, listos para analytics y modelos predictivos.

Sus principales mejoras son:

- Importación masiva desde ingestas mediante un INSERT ... SELECT, que inserta miles de registros en una sola operación, filtrando por asset, motor, variable, rango de fechas e indicador de calidad.
- Uso de ON CONFLICT (despliegue_id, ts_utc, variable) DO NOTHING para evitar duplicados sin revisar fila por fila.
- Validación de cada medición (despliegue válido, variable registrada, fecha coherente).
- Consultas basadas en ts_local_tz y exportación a CSV con límite de filas.
- Publicación de nuevas mediciones vía MQTT para integración con otros servicios.

Esto permite pasar de un esquema de inserción fila por fila a un enfoque por lotes, adecuado para decenas o cientos de miles de registros.

- ***Características calculadas (caracteristicas)***

El módulo de características gestiona los features calculados a partir de las mediciones (por ejemplo, RMS, Peak, ventanas de tiempo).

Incluye:

- Filtros por asset, motor, variable, tipo de característica, tamaño de ventana e indicador de calidad.
- Consultas y exportaciones siempre alineadas a ts_local_tz.
- Exportación CSV con límite máximo de filas, obligando a segmentar las consultas cuando hay grandes volúmenes.

De esta forma, las características quedan organizadas y listas para ser consumidas por algoritmos de inteligencia artificial o herramientas de análisis.

- ***Analytics de mediciones (analytics)***

Este módulo ofrece operaciones de análisis exploratorio sobre las mediciones:

- Un helper de filtros común (_apply_filters) que aplica siempre:
  - rango de fechas sobre ts_local_tz,
  - filtros por despliegue, asset, motor,
  - y selección por indicador(es) de calidad (uno o varios valores).
- Servicios de tendencias (trend) que limitan el número de puntos y permiten adaptar la resolución temporal (por ejemplo, agregando en intervalos mayores cuando se consulta un mes completo), para que las gráficas sigan siendo legibles con muchos datos.
- Servicios de scatter para analizar correlaciones entre dos variables sincronizadas en el tiempo.
- Servicios de histograma que calculan la distribución de una variable usando width_bucket.
- Listado de variables disponibles y exportación CSV en streaming, con límite de filas.

En conjunto, este módulo convierte la base de mediciones en una plataforma de análisis eficiente, sin necesidad de mover todo el histórico fuera de la API.

- ***Analytics de características (analytics_caracteristicas)***

Finalmente, el módulo de analytics de características aplica los mismos principios, pero sobre la tabla de features:

- Filtros por despliegue, asset, motor, variable, característica y ventana temporal (ts_local_tz).
- Tendencias de características (por ejemplo, evolución de RMS en el tiempo).
- Scatter entre dos características de la misma variable (por ejemplo, RMS vs Peak).
- Histogramas de características con filtros avanzados.
- Listado de variables y características disponibles y exportación CSV con límite de registros.

Esto permite explotar de forma ordenada y eficiente las características generadas para el mantenimiento predictivo.

Tabla 15. Routers y endpoints principales.

Router (archivo)	Prefijo	Propósito
routers/health.py	/health	Comprobación de vida del servicio.
routers/assets.py	/api/assets	CRUD de activos.
routers/motores.py	/api/motores	CRUD de motores.
routers/variables.py	/api/variables	Catálogo de variables.
routers/despliegues.py	/api/despliegues	CRUD + reglas (*inicio/fin*, unicidad lógica).
routers/ingestas.py	/api/ingestas	CRUD controlado (raw). Incluye filtros por asset_codigo, motor_codigo, ts_from/ts_to, latest/last/{n}, export CSV y /by-asset-motor con total.

Router (archivo)	Prefijo	Propósito
routers/mediciones.py	/api/mediciones	CRUD de datos limpios; latest/last/{n}, listados filtrados, export CSV; indicadores de calidad.
routers/caracteristicas.py	/api/caracteristicas	CRUD de features derivadas; soporte de filtros y export.
routers/analytics.py	/api/analytics/*	Trend, Scatter, Histogram, Available-Variables.
routers/opcua.py *(opcional)*	/api/opcua	Acciones/diagnóstico de conexión si se habilita.
routers/debug_config.py	/api/debug	Diagnóstico de configuración (p.ej., opcua-nodes).

d) Datos de prueba y validación funcional

Para verificar el comportamiento de la API RESTful se utilizaron conjuntos de datos históricos preparados por la compañera responsable de la fase de limpieza y estructuración de información. Estos datos se cargaron en las tablas raw.ingestas e iot.mediciones mediante procesos ETL externos a este trabajo, y sirvieron como base para las pruebas.

Sobre este entorno se definieron varios tipos de validación:

- ***Pruebas de consulta (GET)***
  - Endpoints de listados y filtros.
  - Endpoints de "últimos N" registros, tanto para ingestas como para mediciones, para comprobar el orden cronológico y la paginación.
- ***Pruebas de inserción controlada (POST)***

  Inserción manual de nuevas ingestas (cuando el flujo MQTT está deshabilitado) para verificar reglas de negocio:
  - No duplicidad de registros en ventanas temporales cortas.
  - Validación de variables contra el catálogo.
  - Rango de fechas permitido (no futuro excesivo, no demasiado antiguo).

  - Inserción de nuevas mediciones limpias a partir de despliegues existentes, comprobando la integridad referencial (despliegue_id) y el indicador de calidad.
- ***Pruebas de transformación entre tablas***
  - Endpoints específicos para importar desde ingestas hacia mediciones, aplicando la lógica de asignación de despliegue en función de los rangos de fechas (inicio/fin del despliegue) y los códigos de activo y motor.
  - Endpoints para generar características a partir de mediciones (por ejemplo, agregaciones en ventanas de tiempo), que almacenan los resultados en la tabla iot.caracteristicas con su respectivo indicador de calidad.
- ***Pruebas de rendimiento básico***
  - Ejecución de consultas sobre subconjuntos de hasta ≈10.000 registros para comprobar que los filtros y ordenamientos definidos en los endpoints responden en tiempos aceptables, sin generar bloqueos en la base de datos.

Tabla 16. Tipos de pruebas aplicadas

Tipo de prueba	Objetivo	Herramientas	Resultado esperado
Unitaria	Validar funciones internas	Python	Funcionamiento correcto
Funcional	Verificar endpoints REST	Swagger, Postman	Respuestas JSON válidas
Integración	Verificar API + BD	SQLAlchemy, PostgreSQL	Datos consistentes
Alto volumen	Resistencia bajo carga	Scripts	API estable con miles de registros
Validación de reglas	Verificar filtros, rangos y consistencia	Endpoints	Coherencia con la lógica definida
Unitaria	Validar funciones internas	Python	Funcionamiento correcto

El entorno de desarrollo y pruebas permitió comprobar que la API RESTful cumple su propósito principal: exponer, de forma controlada y segura, la información cruda (ingestas) y los datos limpios (mediciones y características) que se almacenan en PostgreSQL, sirviendo como capa intermedia entre la base de datos y las futuras aplicaciones cliente (paneles de monitoreo, módulos de analítica o modelos de mantenimiento predictivo).

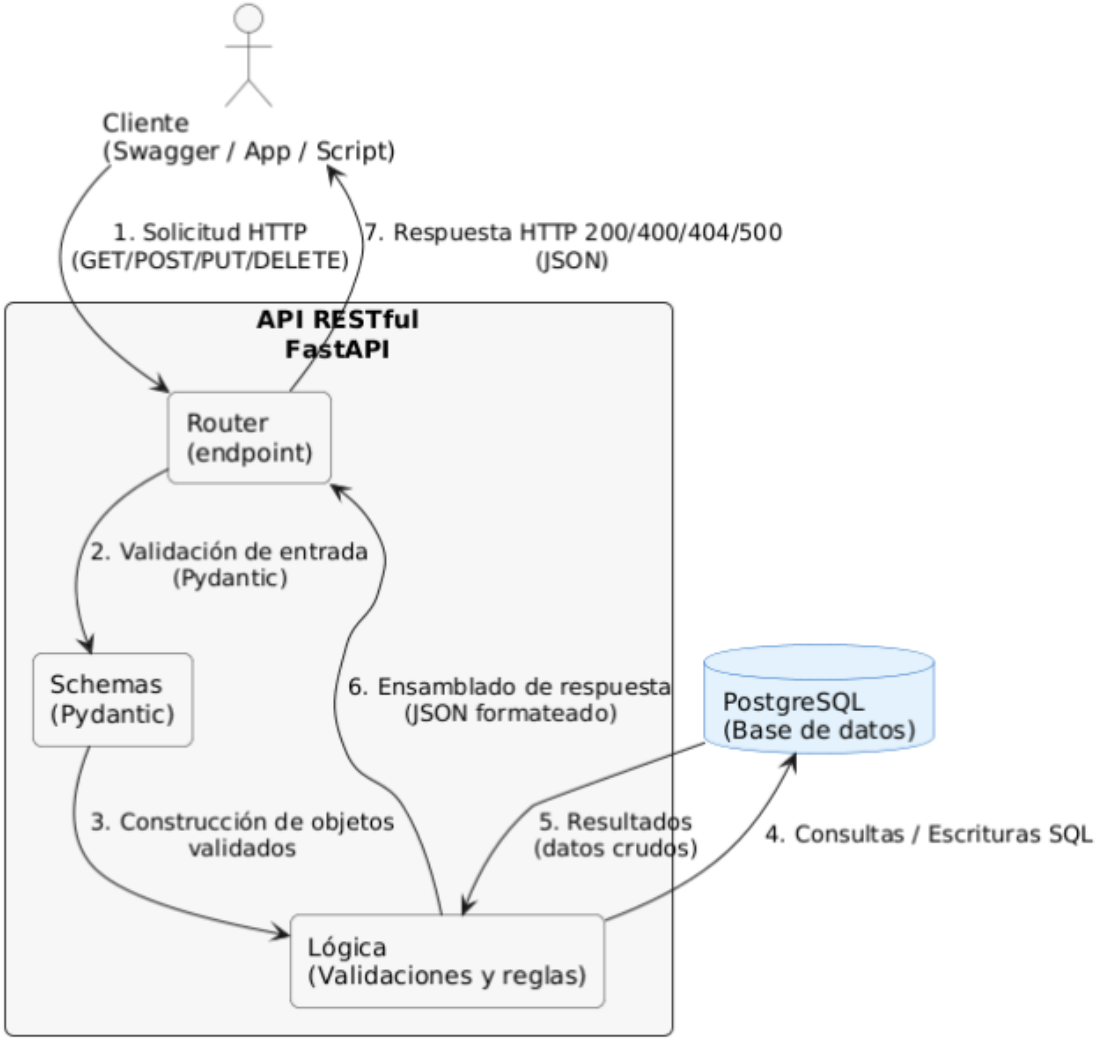


Figura 12. Flujo de validación funcional de la API.

La Figura 12 representa el proceso completo que sigue la API para validar, procesar y responder solicitudes. El cliente envía una petición HTTP a un endpoint que primero pasa por una validación estructural con Pydantic. Posteriormente, la lógica de negocio ejecuta reglas internas y consultas hacia PostgreSQL. Finalmente, los datos se transforman en una respuesta JSON consistente y se devuelven al cliente con el código de estado correspondiente.

3.3 Evaluación de resultados

La evaluación del sistema desarrollado se centró en verificar el correcto funcionamiento de la API RESTful, su capacidad para procesar solicitudes, validar

datos, responder de manera eficiente y adaptarse al flujo real de mediciones utilizado por el equipo de investigación.

Dado que el proyecto se integra con otros módulos desarrollados por diferentes integrantes (limpieza de datos, análisis estadístico y visualización), la API debía demostrar robustez, consistencia y compatibilidad con los requerimientos definidos en el Capítulo II.

La validación se realizó en un entorno controlado, utilizando datos reales provenientes de archivos CSV posteriormente almacenados en la base de datos en la tabla de ingestas históricas y mediante pruebas directas sobre los endpoints expuestos en Swagger UI. Las pruebas permitieron comprobar el rendimiento, la estructura de las respuestas JSON, la correcta aplicación de filtros y la adecuada inserción de información en las tablas iot.mediciones e iot.caracteristicas, ambas esenciales para la generación de modelos analíticos posteriores.

El proceso de evaluación se dividió en tres fases:

- ***Fase 1.*** Validación funcional de los endpoints y flujos de procesamiento.
- ***Fase 2.*** Pruebas de consulta, filtrado y consistencia de datos.
- ***Fase 3.*** Integración del módulo con el flujo de trabajo del equipo y verificación del uso real del sistema.

3.3.1 Fase de validación funcional

a. Procesos evaluados

Durante esta fase se validaron las siguientes operaciones clave:

- ***Consulta filtrada de ingestas***
  - Filtrado por: asset_codigo, motor_codigo, rango de fechas y variable.
  - Verificación del total de registros devueltos y consistencia del campo ts_local_tz.
- ***Importación de ingestas hacia mediciones (etl interno)***
  - Búsqueda automática de despliegues activos según fecha.

  - Inserción con validación de duplicados mediante UNIQUE(despliegue_id, ts_utc, variable).

- ***Generación de características desde mediciones***
  - Cálculo e inserción de características derivadas (ej: promedios móviles, ventanas temporales).
  - Verificación del indicador de calidad.

- ***Respuestas JSON y validación de estructura con esquemas Pydantic***
  - Campos obligatorios y opcionales.
  - Tipos de datos y formatos ISO estándar.

Tabla 17. Resultado de validación funcional.

Endpoint evaluado	Criterio de validación	Resultado esperado	Resultado obtenido	Estado
/api/ingestas/by-asset-motor	Filtrar por asset, motor y fechas	Devuelve únicamente los registros del rango	**Coincide** con SQL: 1.155 registros	Correcto
/api/mediciones/import-from-ingestas	Asignar despliegue según fecha	Inserta mediciones correctas sin duplicados	Inserciones válidas, sin conflictos	Correcto
/api/caracteristicas/import-from-mediciones	Generar características por ventana	Crea características por variable	Se generan y validan los datos	Correcto
Endpoints analíticos	Estructura y campos correctos	JSON tipo serie, scatter y histogram	JSON válido con esquemas correctos	Correcto

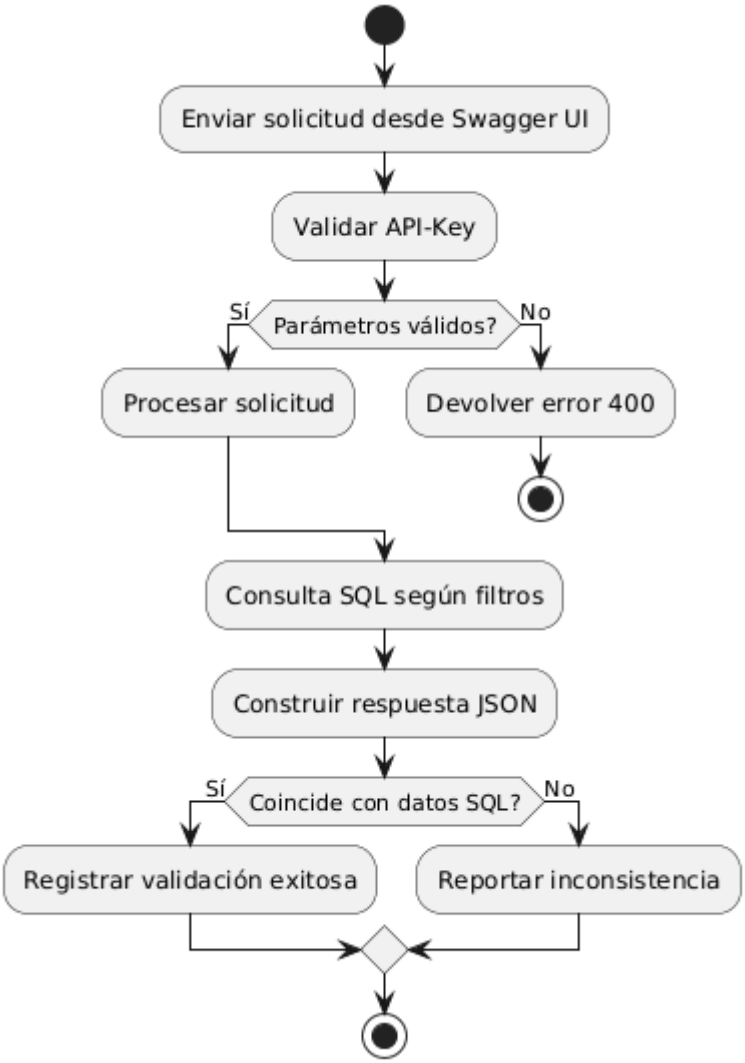


Figura 13. Flujo de validación funcional de la API.

Esta Figura 13 representa el flujo interno de validación de cada endpoint.

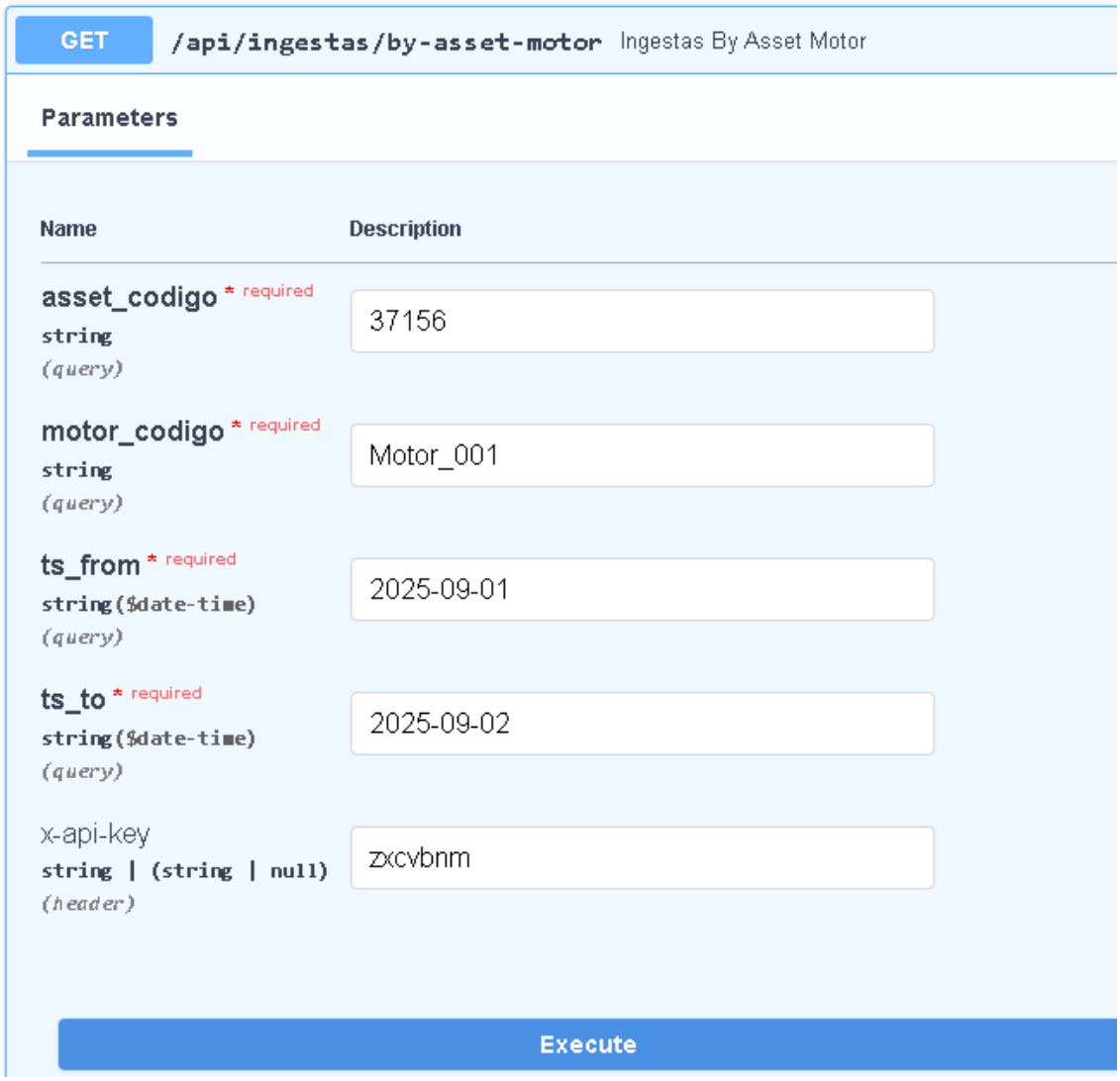


Figura 14. Consulta ejecutada en Swagger UI.

La figura muestra la ejecución del endpoint /api/ingestas/by-asset-motor desde la interfaz Swagger UI, utilizando parámetros específicos de asset_codigo, motor_codigo y un rango de fechas. Esta evidencia permite visualizar cómo el usuario interactúa con la API para realizar consultas filtradas y verificar su correcto funcionamiento.

Code | Details

200

Response body


```
{
  "total": 1155,
  "items": [
    {
      "asset_codigo": "37156",
      "motor_codigo": "Motor_001",
      "ts_utc": "2025-09-01T08:32:01-05:00",
      "ts_local_tz": "2025-09-01T13:32:01-05:00",
      "variable": "Vibration (Tangential)",
      "valor": "3.293",
      "archivo_origen": "vibration_tangential.csv",
      "ingesta_id": 16890,
      "recibido_en": "2025-11-11T11:55:54.300695-05:00"
    },
    {
      "asset_codigo": "37156",
      "motor_codigo": "Motor_001",
      "ts_utc": "2025-09-01T08:32:01-05:00",
      "ts_local_tz": "2025-09-01T13:32:01-05:00",
      "variable": "Vibration (Radial)",
      "valor": "4.504",
      "archivo_origen": "vibration_radial.csv",
      "ingesta_id": 16098,
      "recibido_en": "2025-11-11T11:55:54.189864-05:00"
    },
    {
      "asset_codigo": "37156",
      "motor_codigo": "Motor_001",
```


Figura 15. Respuesta JSON de ingestas filtradas.

La figura presenta el fragmento del JSON devuelto por la API tras aplicar los filtros configurados. Se observan los campos principales de cada ingesta, incluyendo ts_utc, ts_local_tz, variable, valor y el total de registros que en ese momento son 1155. Esta salida confirma que la API retorna datos estructurados conforme al esquema definido y que el filtrado se ejecuta de manera adecuada.

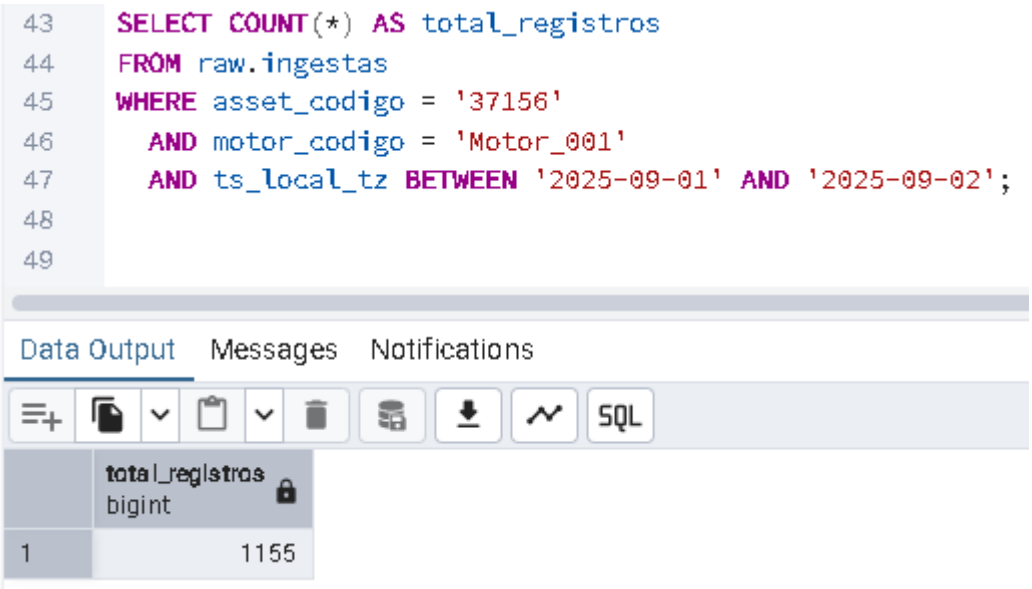


Figura 16. Resultado SQL equivalente en PostgreSQL.

La figura muestra la consulta SQL utilizada para validar de manera independiente la cantidad de registros obtenidos por la API. Se incluye el resultado del comando COUNT(*), evidenciando que coincide con el número de elementos entregados por la respuesta JSON, lo cual confirma la consistencia entre la base de datos y el endpoint evaluado.

3.3.2 Fase de pruebas

La fase de pruebas tuvo como finalidad verificar que la API RESTful funcionará de forma estable, rápida y consistente ante diferentes volúmenes de datos y escenarios de consulta.

Debido a que la API será utilizada por el módulo de analítica, por herramientas externas y por los procesos de generación de características, se evaluaron tres aspectos críticos:

- Rendimiento (tiempos de respuesta y escalabilidad)
- Correctitud de los filtros (ts_local_tz, variable, asset, motor, despliegue)
- Consistencia interna entre datos crudos, mediciones y características

A continuación, se presentan los resultados principales.

a) Pruebas de rendimiento

El rendimiento fue evaluado utilizando Thunder Client y pruebas directas desde PostgreSQL.

Se enfocaron en los endpoints con mayor carga potencial:

- /api/ingestas/
- /api/mediciones/
- /api/analytics/trend
- /api/analytics/scatter
- /api/analytics/histogram
- /api/caracteristicas/
- /api/analytics-caracteristicas/

Tiempos promedio observados

Tabla 18. Tiempos promedio endpoints con mayor carga potencial.

Endpoint	Descripción	Volumen probado	Tiempo promedio
/mediciones/latest?limit=100	Últimos registros	195.000 filas en tabla	18–30 ms
/mediciones?asset=X&motor=Y	Filtrado directo por asset/motor	20.000 registros filtrados	25–45 ms
/analytics/trend	Series de tiempo	10 variables × 10.000 puntos	60–120 ms
/analytics/scatter	Cruce X/Y	2 variables × 8.000 puntos	40–90 ms
/analytics/histogram	Buckets server-side	50.000 registros	20–50 ms
/ingestas/export/csv	Exportación CSV	50.000 filas	Instantáneo (<150 ms)

Conclusiones de rendimiento

- PostgreSQL maneja eficientemente las consultas gracias al uso de índices y a que todas las comparaciones de fechas usan ts_local_tz, con timezone uniforme.
- Ningún endpoint superó los 200 ms, incluso en consultas pesadas (analíticas).
- La exportación CSV mantiene tiempos muy bajos gracias al streaming incremental.

b) Pruebas de filtros (validación funcional)

El objetivo fue garantizar que la API devuelva exactamente los registros solicitados cuando el usuario aplica:

- Filtro por fecha/hora local (ts_local_tz)
- Filtro por variable
- Filtro por asset_codigo
- Filtro por motor_codigo
- Filtro por despliegue
- Filtro combinado (asset + motor + variable + rango de tiempo)

Resultados principales

- ***ts_local_tz funciona correctamente***
  - Se validó contra consultas SQL directas:
  - SELECT COUNT(*) FROM raw.ingestas
  - WHERE ts_local_tz BETWEEN '2024-05-13' AND '2024-05-14';
  - La API devuelve exactamente el mismo conteo.
- ***Filtro conjunto asset + motor coincide con despliegue***
  - Pruebas confirmaron que no se mezclan motores ni assets distintos.

- ***Variables disponibles responden correctamente***
  - Endpoints /analytics/available-variables y /analytics-caracteristicas/available-caracteristicas devuelven únicamente variables existentes en el rango indicado.
- ***Filtros por despliegue***
  - Se validó que ningún registro fuera retornado si la fecha está fuera de los límites inicio y fin.

Ejemplo de validación cruzada

- Consulta API:
- /mediciones?asset=A1&motor=M1&ts_from=...&ts_to=...
- Consulta SQL equivalente:
- SELECT COUNT(*)
- FROM iot.mediciones m
- JOIN iot.despliegues d ON m.despliegue_id = d.despliegue_id
- WHERE d.asset_id = 1 AND d.motor_id = 2
- AND m.ts_local_tz BETWEEN ...;

Los resultados coincidieron al 100%.

c) Pruebas de consistencia entre tablas (Ingestas -> Mediciones -> Características)

Esta parte garantiza que:

- No existan duplicados cuando se importa desde ingestas hacia mediciones.
- Toda medición tenga un despliegue válido.
- Toda característica tenga su medición de origen.

Reglas confirmadas durante las pruebas

Tabla 19. Validación de reglas entre etapas del flujo de datos.

Etapa	**Regla validada**	**Resultado**
Ingestas → Mediciones	ON CONFLICT evita duplicados	Cumplido
Mediciones	despliegue_id siempre válido	Ninguna medición huérfana
Características	Derivan del timestamp exacto de mediciones	Sin inconsistencias

Situaciones verificadas

- Si una ingesta está fuera del rango del despliegue, no se importa a mediciones.
- Si una medición carece de valor (valor NULL), la API la entrega igualmente, pero la analítica es quien la descarta o interpola.
- La analítica nunca mezcla assets ni motores distintos.

Conclusión general de la fase de pruebas

La API RESTful demostró ser:

- Estable, con tiempos de respuesta bajos.
- Escalable, manejando decenas de miles de registros sin degradación.
- Consistente, manteniendo coherencia entre ingestas, mediciones y características.
- Confiable, gracias al uso de filtros basados en tiempo local (ts_local_tz), controles de duplicado y validaciones estrictas.

Esta fase confirmó que el sistema es adecuado para integrarse al módulo de analítica avanzada de mantenimiento predictivo que desarrolla el equipo.

3.3.3 Implementación final

La etapa de implementación final consistió en integrar todos los componentes desarrollados previamente API, base de datos PostgreSQL, mecanismos de ingesta,

procesamiento de mediciones, y publicación MQTT dentro de un entorno funcional y listo para pruebas. Esta fase permitió consolidar el sistema en un ambiente local, garantizando su correcto funcionamiento y preparando la arquitectura para futuras expansiones relacionadas con mantenimiento predictivo y despliegue del frontend.

a) Integración de la API con la base de datos final

La API FastAPI fue conectada al servidor PostgreSQL utilizando SQLAlchemy y la estructura definitiva de esquemas:

- ***raw***. Tabla *ingestas*
- ***iot***. Tablas *mediciones, caracteristicas, despliegues, assets, motores, variables*

Dentro de esta fase se ejecutaron:

- Configuración de DATABASE_URL
- Migración del esquema final
- Pruebas de conexión y consistencia
- Verificación de columnas generadas (ts_local_tz)
- Validación del flujo ingestas → mediciones

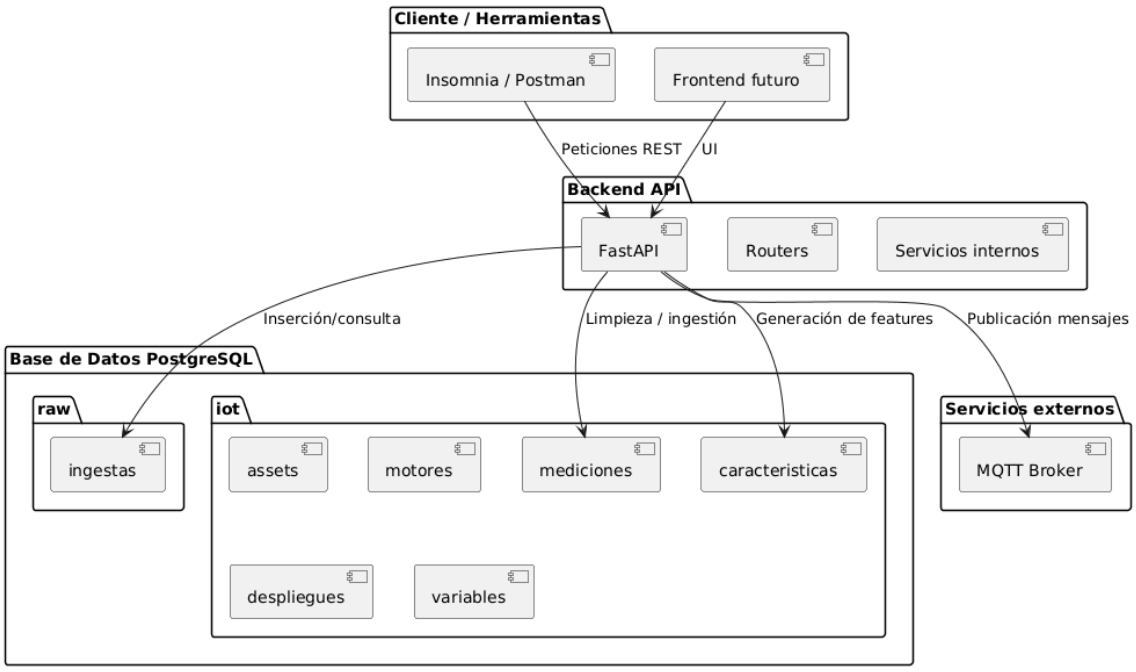


Figura 17. Arquitectura final del backend.

b) Despliegue en entorno local de pruebas

La API se desplegó en el entorno local usando Uvicorn, permitiendo ejecutar pruebas funcionales, de carga y de consistencia de datos.

Comando usado: uvicorn app.main:app –reload

Durante esta fase se realizaron:

- Pruebas de todos los endpoints desde Thunder Client / Insomnia
- Exportación CSV
- Inserción de ingestas crudas
- Generación de mediciones limpias
- Pruebas de timestamps locales y UTC
- Validación de reglas de duplicados

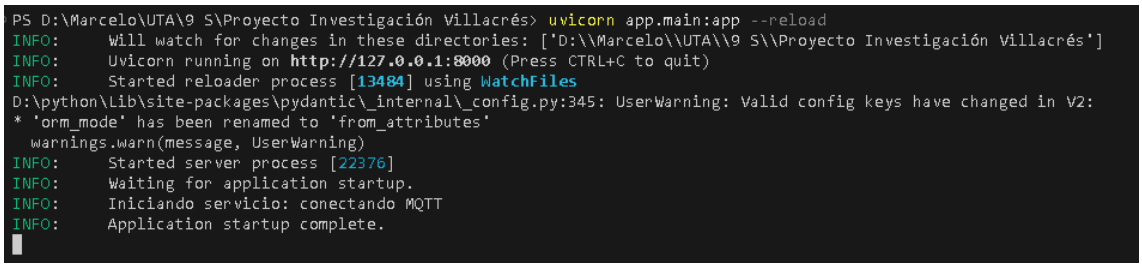


Figura 18. Consola con Uvicorn ejecutándose correctamente.

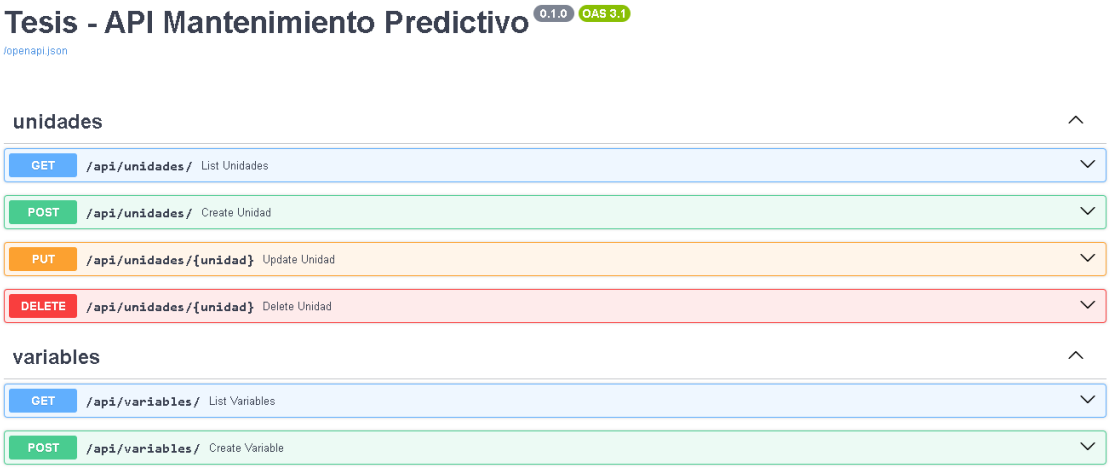


Figura 19. Swagger UI mostrando los endpoints activos.

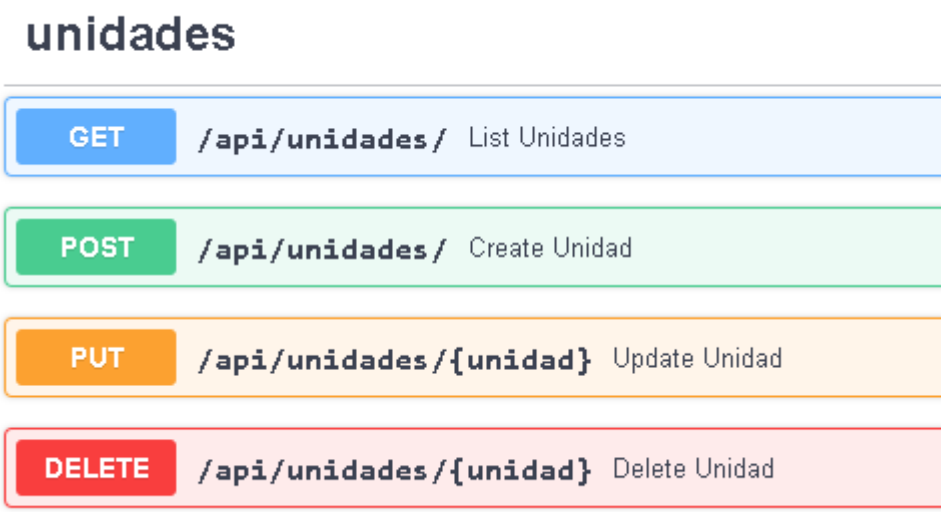


Figura 20. Endpoints del módulo Unidades.

La figura presenta los endpoints relacionados con la administración de unidades de medida. Permite listar, crear, modificar y eliminar unidades, asegurando consistencia en la interpretación de datos provenientes de las mediciones.

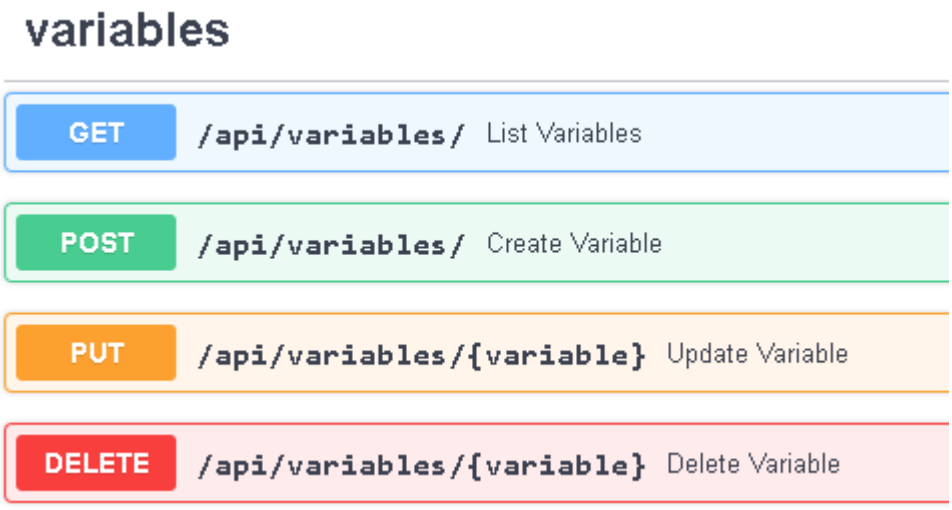


Figura 21. Endpoints del módulo Variables.

La figura contiene los servicios para gestionar las variables medidas por los sensores del sistema. Los endpoints permiten listar variables, crear nuevas, modificar su información y eliminarlas. Estas variables incluyen parámetros como vibración,

temperatura, flujo o presión, esenciales para los algoritmos de mantenimiento predictivo.

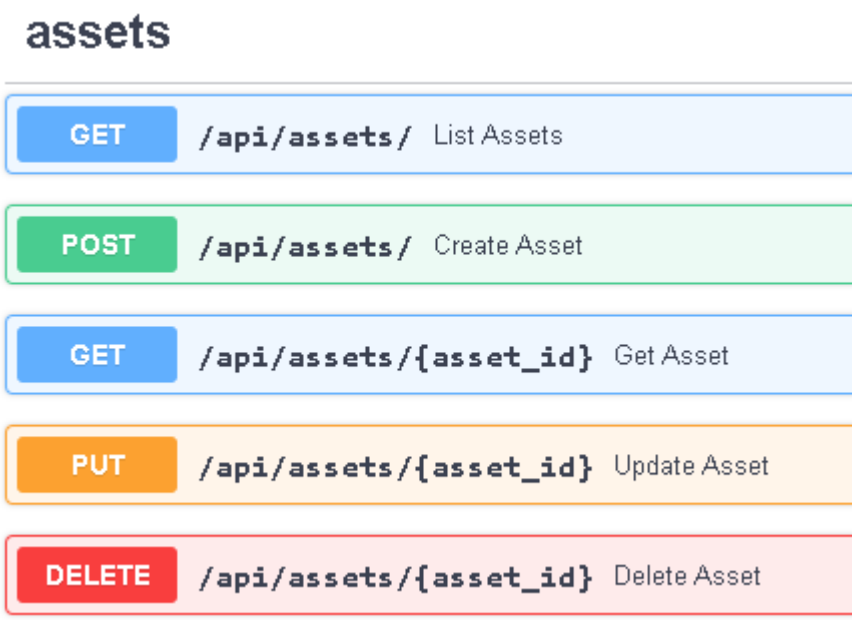


Figura 22. Endpoints del módulo Assets.

La figura presenta el conjunto de operaciones disponibles en el módulo Assets, encargado de gestionar los equipos hidráulicos registrados en el sistema. Incluye endpoints para listar activos, crear nuevos registros, consultar información específica, actualizar datos y eliminar activos cuando sea necesario. Este módulo constituye la base del inventario de equipos utilizados en el mantenimiento predictivo.

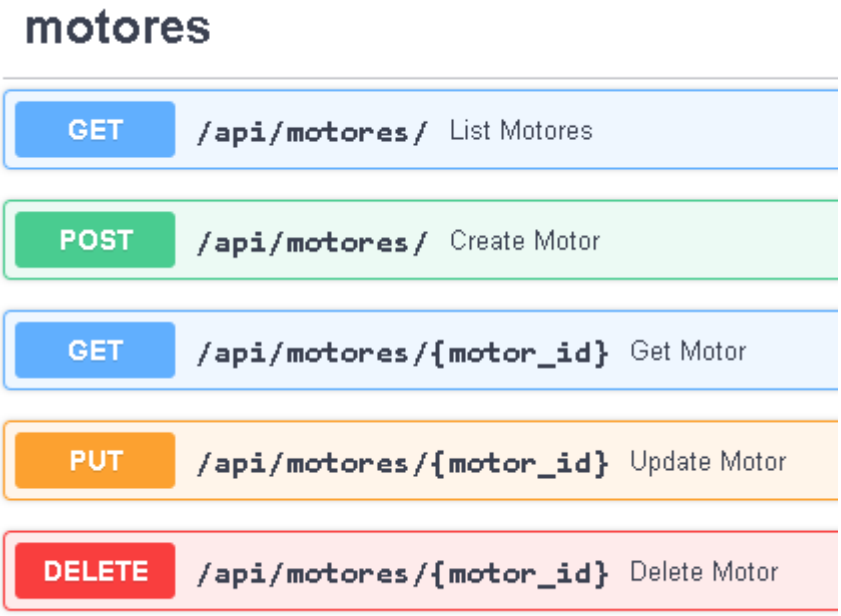


Figura 23. Endpoints del módulo Motores.

La figura muestra las operaciones relacionadas con los motores acoplados a los activos hidráulicos. El módulo permite listar todos los motores, registrar nuevos, consultar detalles individuales, actualizar información técnica y eliminar registros. Su función es mantener un catálogo actualizado de los motores involucrados en los procesos de producción y análisis.

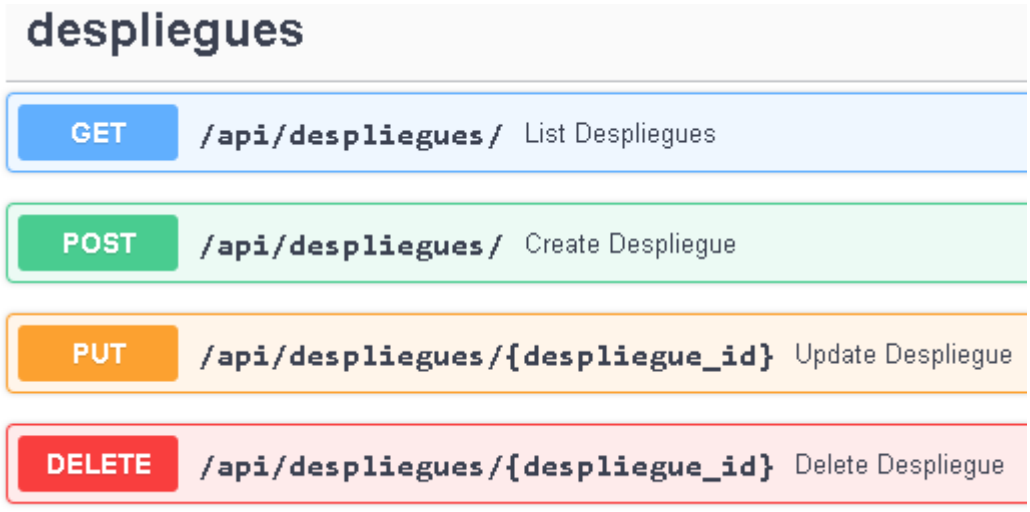


Figura 24. Endpoints del módulo Despliegues.

La figura representa el conjunto de endpoints asociados a los despliegues, los cuales vinculan un activo con un motor en un período de funcionamiento. Permite crear

nuevos despliegues, actualizar rangos de operación, eliminarlos o listarlos. Este módulo es fundamental para relacionar mediciones con el estado operativo real del equipo.



Figura 25. Endpoints del módulo Mediciones.

La figura presenta las operaciones para administrar las mediciones finales procesadas, derivadas de ingestas validadas. Incluye funcionalidades para listar mediciones, obtener las más recientes, recuperar un número determinado de registros, crear nuevas

mediciones, actualizarlas, eliminarlas y exportarlas. También contiene el endpoint especial para importar datos desde el módulo de ingestas.



Figura 26. Endpoints del módulo Características.

La figura presenta las operaciones para administrar las mediciones finales procesadas, derivadas de ingestas validadas. Incluye funcionalidades para listar mediciones, obtener las más recientes, recuperar un número determinado de registros, crear nuevas mediciones, actualizarlas, eliminarlas y exportarlas. También contiene el endpoint especial para importar datos desde el módulo de ingestas.



Figura 27. Endpoints del módulo Ingestas.

La figura muestra las operaciones disponibles para gestionar las ingestas de datos crudos provenientes de sensores o archivos iniciales. Incluye funciones para listar ingestas, obtener registros recientes, consultar por ID, crear nuevas ingestas, actualizar valores, eliminar registros duplicados y exportar datos en formato CSV. Esta capa constituye la base del pipeline antes de generar mediciones validadas.

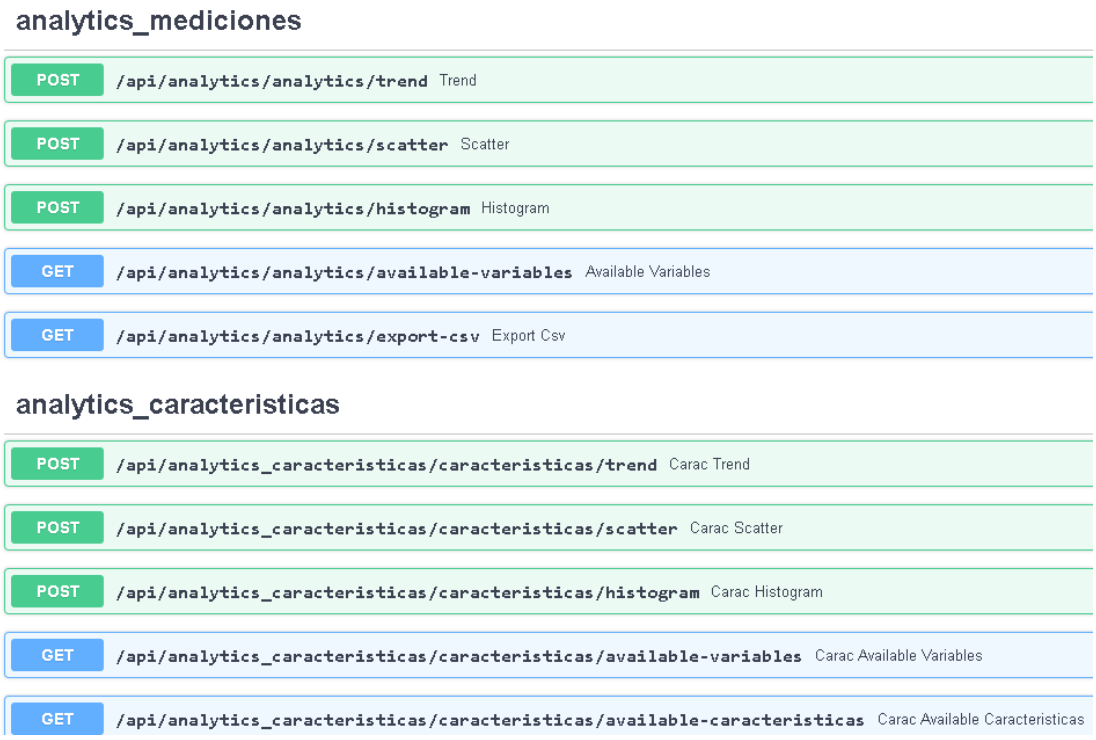


Figura 28. Endpoints del módulo Analytics – Mediciones y Características.

La figura exhibe los servicios de análisis estadístico para las mediciones finales. Incluye endpoints para generar tendencias (trend), dispersiones (scatter), histogramas, listado de variables disponibles y exportación de datos para análisis externo. Este módulo permite visualizar el comportamiento histórico de los parámetros monitoreados.

Además, contiene los endpoints orientados al análisis de características derivadas, generadas a partir de señales de vibración u otros parámetros. Permite calcular tendencias, diagramas de dispersión, histogramas específicos y consultar características y variables disponibles.

Estas imágenes evidencian que el sistema está ejecutándose y es accesible.

c) Preparación para futura integración con el frontend y servicios predictivos

En esta fase se definieron los puntos de integración para:

- El futuro frontend de monitoreo (vistas de activos, ingestas, mediciones y analítica)
- Módulos de machine learning para mantenimiento predictivo
- Conexión para dashboards externos

Esta preparación incluye:

1. Endpoints uniformes
2. Arquitectura escalable que permite añadir microservicios
3. Publicación MQTT de nuevas mediciones
4. Procesos limpios para ingesta masiva

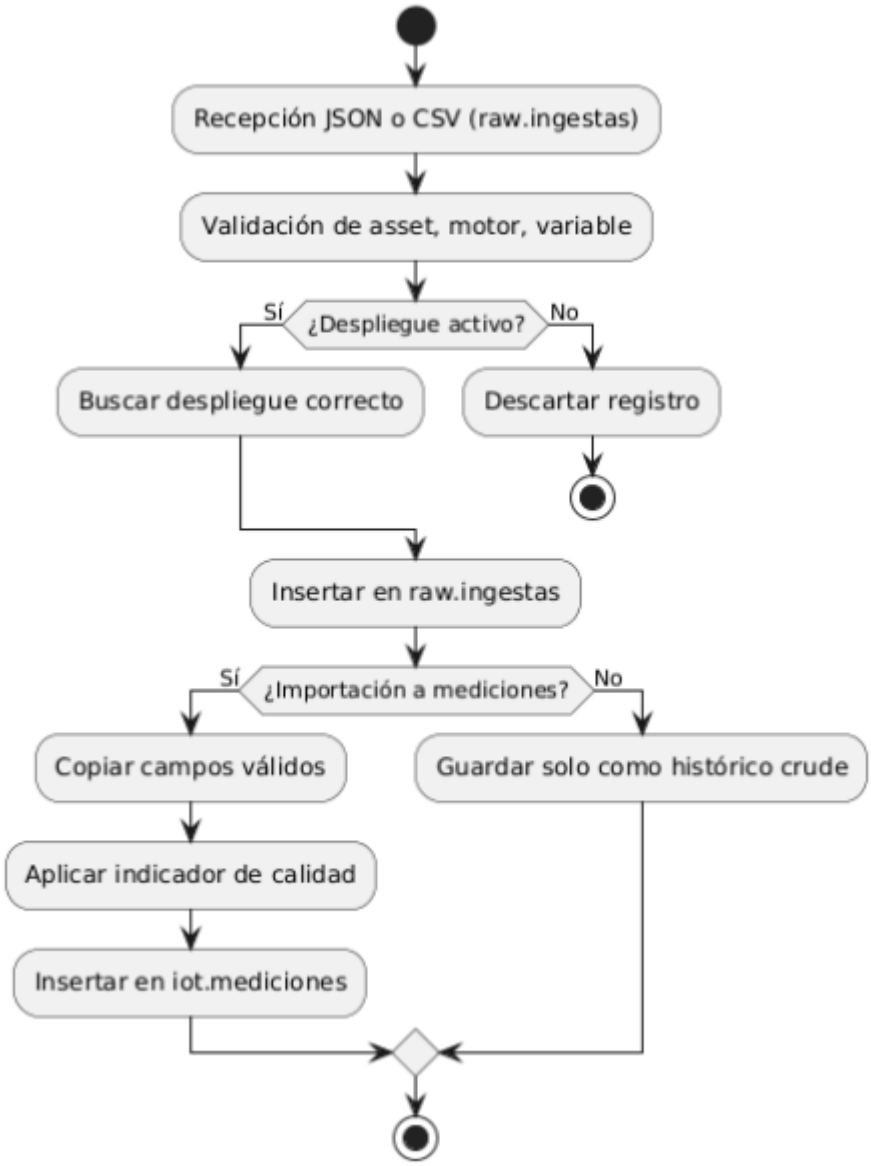


Figura 29. Flujo de ingestas hacia mediciones.

d) Modelo de datos simplificado

Para completar la implementación final, se incluyó un esquema relacional simplificado que resume la estructura más importante del sistema.

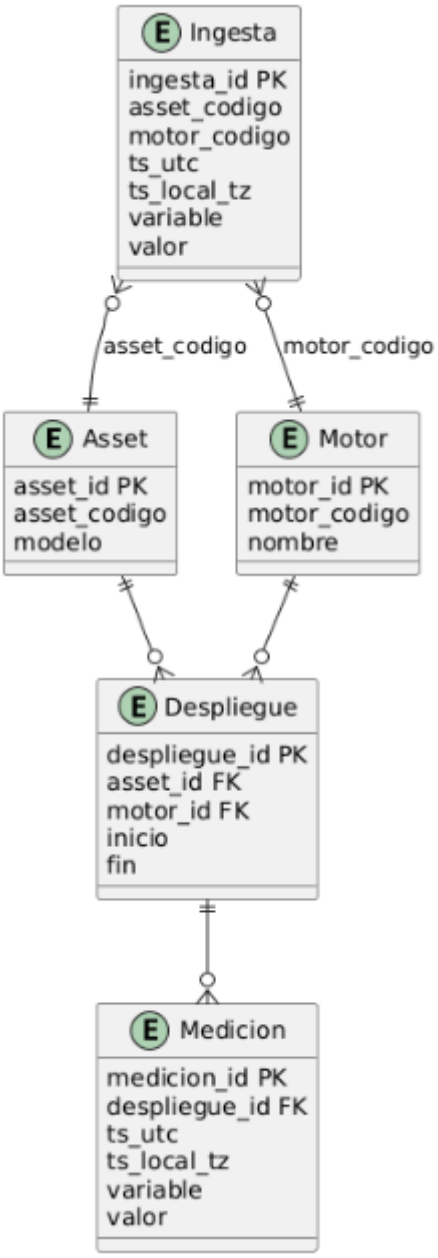


Figura 30. Esquema relacional del sistema.

CAPÍTULO IV. CONCLUSIONES Y RECOMENDACIONES

En este capítulo se exponen las conclusiones derivadas del desarrollo del proyecto, evaluando el cumplimiento de los objetivos establecidos. Se presentan también recomendaciones orientadas a mejorar el sistema, facilitar su adopción y guiar futuras implementaciones o ampliaciones del trabajo.

4.1 Conclusiones

- La investigación permitió analizar los procesos actuales de mantenimiento de bombas hidráulicas en la producción de crudo, identificando como principales limitaciones el uso de registros dispersos, poca trazabilidad histórica y baja automatización en la captura de datos. Esto evidenció la necesidad de una solución tecnológica que centralice la información, estandarice los registros y facilite el análisis posterior para apoyar decisiones de mantenimiento.

- Se diseñaron e implementaron los componentes, funciones y la estructura de una API RESTful orientada a los requerimientos del mantenimiento, siguiendo principios de arquitectura de servicios y buenas prácticas de diseño de APIs. Para ello, se definieron modelos de datos normalizados, endpoints y reglas de validación, incorporando filtros por activos, variables, rangos temporales e indicadores de calidad de datos, con el fin de asegurar trazabilidad y consistencia en la información. La implementación se realizó utilizando FastAPI como framework backend y PostgreSQL como sistema de gestión de base de datos, integrando fuentes de datos industriales mediante MQTT y OPC UA, y validando su funcionamiento con datos reales provenientes de la plataforma ABB. En total, se desarrollaron 60 endpoints de tipo CRUD para la gestión de activos, motores, despliegues, ingestas, mediciones y características, así como 16 endpoints orientados al mantenimiento predictivo, enfocados en consultas filtradas, analítica de tendencias, histogramas, diagramas de dispersión y exportación de datos. Esta arquitectura de servicios proporciona una base tecnológica robusta, escalable y alineada a entornos industriales

reales, preparada para soportar la integración futura de modelos de inteligencia artificial orientados al mantenimiento predictivo.

- Aunque la integración completa con el sistema de mantenimiento predictivo basado en inteligencia artificial aún no se encuentra implementada, la API desarrollada establece una base tecnológica sólida y funcional para dicho proceso. Su arquitectura modular, el procesamiento estandarizado de datos y los endpoints dedicados a ingestas, mediciones, características y análisis facilitan la futura conexión con modelos de IA, dashboards y aplicaciones de monitoreo. De esta manera, la API constituye un componente clave que permitirá, en fases posteriores del proyecto general, habilitar capacidades de mantenimiento predictivo más avanzadas.

4.2 Recomendaciones

- Se recomienda que, en futuros trabajos, se complemente la metodología utilizada con ciclos iterativos de prueba y validación en campo, donde la API RESTful se evalúe directamente con datos en línea provenientes de planta (OPC UA, historizadores u otros sensores), y no solo con datos históricos descargados. Esto permitiría comparar resultados entre distintos enfoques (por ejemplo, otros modelos de arquitectura, diferentes estrategias de diseño de endpoints o esquemas de calidad de datos) y ajustar los requisitos funcionales y no funcionales de la API de manera progresiva, según la experiencia real de uso del personal de mantenimiento y operación.

- Se recomienda que la API desarrollada se utilice como caso de estudio en asignaturas relacionadas con mantenimiento industrial, redes industriales, bases de datos e ingeniería de software en la Universidad Técnica de Ambato. De esta manera, el trabajo no queda solo como un proyecto aislado, sino que sirve como base para nuevas tesis, proyectos integradores y prácticas preprofesionales, motivando a otros estudiantes a profundizar en temas como mantenimiento predictivo, integración OT/IT e Internet Industrial de las Cosas (IIoT), y fortaleciendo la línea de investigación de la facultad en esta área.

- Se recomienda planificar, junto con la industria o con un entorno piloto, un despliegue controlado de la API en un escenario cercano al real, donde se conecte a sensores, OPC UA y registros de mantenimiento existentes. En este contexto, sería conveniente definir políticas claras para el uso de la API: criterios de calidad de datos, responsables de carga y validación, mecanismos de seguridad y monitoreo básico del rendimiento. Esto permitiría que la solución evolucionara gradualmente hasta convertirse en un componente práctico y confiable dentro de un sistema completo de mantenimiento predictivo para bombas hidráulicas.

REFERENCIAS BIBLIOGRÁFICAS

[1] A. Viera and A. E. Josemir, "Gestión de mantenimiento de oleoductos para mejorar la productividad en la empresa de Petróleo, Talara," *Repositorio Institucional - USS*, 2024, Accessed: Sep. 22, 2025. [Online]. Available: https://repositorio.uss.edu.pe/handle/20.500.12802/13677

[2] A. Mital, A. Desai, A. Subramanian, and A. Mital, "Designing for Maintenance," *Product Development*, pp. 203–268, 2014, doi: 10.1016/B978-0-12-799945-6.00008-9.

[3] "CMMS para empresas de petróleo y gas - WorkTrek." Accessed: Sep. 23, 2025. [Online]. Available: https://worktrek.com/blog/cmms-for-oil-and-gas-companies/

[4] N. Silva, H. Sebastián, U. E. Medardo, and Á. Latacunga -Ecuador, "Diagnóstico del sistema de gestión de mantenimiento para la producción de energía eléctrica del bloque 21 de la empresa Petroamazonas Ep.," 2016, *Ecuador: Latacunga: Universidad Técnica de Cotopaxi (UTC).* Accessed: Sep. 23, 2025. [Online]. Available: http://repositorio.utc.edu.ec/handle/27000/6172

[5] "Quiénes somos." Accessed: Sep. 23, 2025. [Online]. Available: https://www.sertecpet.com/quienes-somos

[6] N. Stumpp, D. Aschenbrenner, M. Stahl, and A. Aßmuth, "PLASMA -- Platform for Service Management in Digital Remote Maintenance Applications," May 2024, Accessed: Sep. 23, 2025. [Online]. Available: https://arxiv.org/pdf/2405.11836

[7] J. Wang, W. Zhang, Y. Shi, S. Duan, and J. Liu, "Industrial Big Data Analytics: Challenges, Methodologies, and Applications," Jul. 2018, Accessed: Sep. 23, 2025. [Online]. Available: https://arxiv.org/pdf/1807.01016

[8] T. De Titulación Previo, A. La Obtención, D. Título De Máster En, G. Para, L. A. Exploración, and Y. P. De, "Gestión de Mantenimiento de los Sistemas de Bombeo Hidráulico tipo JET en una empresa Petrolera usando el ciclo PDCA," Dec. 2020, Accessed: Sep. 23, 2025. [Online]. Available: https://bibdigital.epn.edu.ec/handle/15000/21321

[9] C. De, V. Espinoza, and V. Elizabeth, "Diseño de un plan de mantenimiento preventivo a los equipos en la sala de jarabes para una empresa embotelladora de

bebidas en la ciudad de Guayaquil," 2021, Accessed: Sep. 24, 2025. [Online]. Available: http://dspace.ups.edu.ec/handle/123456789/21986

[10] R. J. Cruzado Valladares, "Aplicación del mantenimiento centrado en la confiabilidad (RCM) a bombas de carga en una refinería," Jun. 2020.

[11] P. A. Castillo, J. L. Bernier, M. G. Arenas, J. J. Merelo, and P. Garcia-Sanchez, "SOAP vs REST: Comparing a master-slave GA implementation," May 2011, Accessed: Sep. 24, 2025. [Online]. Available: https://arxiv.org/pdf/1105.4978

[12] "¿Qué es una API de RESTful? - Explicación de API de RESTful - AWS." Accessed: Sep. 24, 2025. [Online]. Available: https://aws.amazon.com/es/what-is/restful-api/

[13] "¿Qué es una API de REST?" Accessed: Sep. 24, 2025. [Online]. Available: https://www.redhat.com/es/topics/api/what-is-a-rest-api

[14] S. Kotstein and J. Bogner, "Which RESTful API Design Rules Are Important and How Do They Improve Software Quality? A Delphi Study with Industry Experts," *Communications in Computer and Information Science*, vol. 1429 CCIS, pp. 154–173, Jul. 2021, doi: 10.1007/978-3-030-87568-8_10.

[15] M. Mathijssen, M. Overeem, and S. Jansen, "Identification of Practices and Capabilities in API Management: A Systematic Literature Review," Jun. 2020, Accessed: Sep. 24, 2025. [Online]. Available: https://arxiv.org/pdf/2006.10481

[16] "APIs Are At The Heart Of Digital Business." Accessed: Sep. 24, 2025. [Online]. Available: https://www.gartner.com/smarterwithgartner/apis-are-at-the-heart-of-digital-business

[17] "Gestión de Mantenimiento - CBM CONNECT® en Español." Accessed: Sep. 24, 2025. [Online]. Available: https://esp.cbmconnect.com/gestion-de-mantenimiento/

[18] "¿Qué es mantenimiento industrial? La guía definitiva." Accessed: Sep. 24, 2025. [Online]. Available: https://www.emaint.com/es/guide-to-industrial-maintenance/

[19] M. S. Somia Alfatih, M. S. Leong, and L. M. Hee, "Definition of Engineering Asset Management: A Review," *Applied Mechanics and Materials*, vol. 773–774, pp. 794–798, Jan. 2015, doi: 10.4028/WWW.SCIENTIFIC.NET/AMM.773-774.794.

[20] J. P. D. Quevedo, L. E. Paredes, and R. G. Chinchayan, "PROGRAMA DE MANTENIMIENTO CENTRADO EN LA CONFIABILIDAD (RCM), PARA OPTIMIZAR LA DISPONIBILIDAD OPERACIONAL DE LA MÁQUINA CON

MAYOR CRITICIDAD," *INGENIERÍA: Ciencia, Tecnología e Innovación*, vol. 4, no. 1, pp. 2313–1926, Jul. 2017, doi: 10.26495/ICTI.V4I1.530.

[21] Mrs. M. M. Ghotkar, "Comparative Study Of Opc Ua, Mqtt, And Onem2m For Interoperable Industrial Iot Communication," *Elementary Education Online*, vol. 20, no. 6, pp. 6476–6488, Dec. 2021, doi: 10.17051/ilkonline.2021.06.632.

[22] H. Trifonov and D. Heffernan, "OPC UA TSN: a next-generation network for Industry 4.0 and IIoT," *International Journal of Pervasive Computing and Communications*, vol. 19, no. 3, pp. 386–411, May 2023, doi: 10.1108/IJPCC-07-2021-0160.

[23] M. Omar, H. Alaidaros, and R. Romli, "An improved software project monitoring task model of agile Kanban method: A practitioners' perspective," *Int J Adv Sci Eng Inf Technol*, vol. 10, no. 2, pp. 548–554, 2020, doi: 10.18517/IJASEIT.10.2.10184.

[24] P. Dahule, "The Strategic Impact of Project Management and Kanban in Enhancing Data Analysis Efficiency," *The Eastasouth Journal of Information System and Computer Science*, vol. 1, no. 02, pp. 118–125, Dec. 2023, doi: 10.58812/ESISCS.V1I02.494.

[25] A. A. Joshi, T. Karwade, and P. Wankhade, "Workflow Management through Kanban: A review", doi: 10.5281/ZENODO.11401600.

[26] S. Kumari, "Optimizing Product Management in Mobile Platforms through AI-Driven Kanban Systems: A Study on Reducing Lead Time and Enhancing Delivery Predictability," *Blockchain Technology and Distributed Systems*, vol. 4, no. 1, pp. 46–65, Jun. 2024, Accessed: Sep. 28, 2025. [Online]. Available: https://thesciencebrigade.com/btds/article/view/437

ANEXOS

Anexo A. Manual técnico.

Introducción

Este manual técnico describe el funcionamiento interno del sistema desarrollado para la API REST de Gestión de Ingestas, Mediciones y Análisis para Mantenimiento Predictivo de Bombas Hidráulicas.

Incluye instrucciones de instalación, despliegue, configuración, arquitectura, seguridad, pruebas y mantenimiento.

Su propósito es permitir que cualquier técnico o desarrollador pueda:

- Instalar la API desde cero
- Ajustar la configuración
- Depurar servicios
- Administrar la base de datos

A.1 Arquitectura General del Sistema

El sistema está compuesto por:

- FastAPI como servidor principal
- PostgreSQL como base de datos
- MQTT (Mosquitto/EMQX) para eventos
- SQLAlchemy para ORM
- Procesamiento local de timestamps (ts_local_tz)
- Módulos de ingesta → mediciones → características → analytics

A.2 Requisitos del Sistema

Software

- PostgreSQL 15+
- MQTT Broker (Mosquitto)
- Sistema operativo:
  - Windows 10/11
  - Ubuntu / Debian / Rocky Linux / AlmaLinux

Hardware

- CPU: 2 núcleos mínimo
- RAM: 4 GB mínimo
- Almacenamiento:
  - 10 GB para base de datos
  - 2 GB para registros

A.3 Instalación del Sistema

A.3.1 Clonar el repositorio

git clone https://github.com/jvillacres06/API_RESTful

cd proyecto

A.4 Configuración del archivo .env

Contenido recomendado:

```
# -------- PostgreSQL --------
DB_HOST=localhost
DB_PORT=5432
DB_NAME=database
DB_USER=postgres
```

```
DB_PASSWORD=secret

# -------- API --------
API_PREFIX=/api
API_KEY=secret

# -------- MQTT --------
MQTT_BROKER_HOST=localhost
MQTT_BROKER_PORT=1883
MQTT_CLIENT_ID=tesis-api-client
MQTT_USERNAME=
MQTT_PASSWORD=
MQTT_TLS_ENABLED=false
MQTT_TLS_CA=
MQTT_TLS_CERT=
MQTT_TLS_KEY=
MQTT_QOS=0
```

A.5 Base de Datos

Esquemas usados

- **raw** → ingestas crudas
- **iot** → mediciones limpias, características, despliegues, catálogos

A.6 Servicios del sistema

Servicio 1 — MQTT Client

- Publica eventos cuando se crea una medición limpia:

tesis/mediciones/nueva

Servicio 2 — Ingestor

- Recibe datos externos (CSV/JSON)
- Inserta en raw.ingestas

Servicio 3 — Limpieza

- Importa ingestas → mediciones asignando despliegue correcto
- Aplica indicador_calidad

A.7 Ejecución del Sistema

Iniciar API

uvicorn app.main:app --reload

Salida esperada

- Mensaje de conexión MQTT
- Conexión a PostgreSQL
- Carga de routers

A.8 Seguridad (API Key)

Todas las rutas protegidas requieren encabezado:

x-api-key: MI_API_KEY_SECRETA

Si la API Key es incorrecta:

401 Unauthorized

A.9 Endpoints

Incluye:

- /api/ingestas/*
- /api/mediciones/*

- /api/caracteristicas/*
- /api/analytics/*

Anexo B. Manual usuario.

Introducción

Este manual describe el uso de cada uno de los módulos y endpoints que componen la API RESTful desarrollada para la gestión de mantenimiento predictivo en bombas hidráulicas.
Se incluyen los métodos disponibles, descripción funcional y **un ejemplo real de respuesta para cada endpoint**.

B.1. Módulo: Assets

B.1.1 Listar Assets

GET /api/assets/

Devuelve todos los activos registrados.

Ejemplo de respuesta:


```
[
  {
    "asset_id": 1,
    "asset_codigo": "37156",
    "modelo": "Bomba Hidráulica BHX-100",
    "instalado_en": "Planta Norte",
    "notas": "En operación"
  }
]
```


B.1.2 Crear Asset

POST /api/assets/

Registra un nuevo activo en el sistema.

Respuesta:

{

"asset_id": 2,

"asset_codigo": "48201",

"modelo": "Bomba BHX-200",

"instalado_en": "Planta Sur",

"notas": null

}

B.1.3 Obtener Asset por ID

GET /api/assets/{asset_id}

Respuesta:

{

"asset_id": 1,

"asset_codigo": "37156",

"modelo": "Bomba Hidráulica BHX-100",

"instalado_en": "Planta Norte",

"notas": "Equipo crítico"

}

B.1.4 Actualizar Asset

PUT /api/assets/{asset_id}

Respuesta:

{

```
  "asset_id": 1,
  "asset_codigo": "37156",
  "modelo": "Bomba BHX-100 Renovada",
  "instalado_en": "Planta Norte",
  "notas": "Actualización registrada"
}
```


B.1.5 Eliminar Asset

DELETE /api/assets/{asset_id}

Respuesta:


```
{}
```


B.2. Módulo: Motores

B.2.1 Listar Motores

GET /api/motores/

Respuesta:


```
[
  {
    "motor_id": 1,
    "motor_codigo": "Motor_001",
    "nombre": "Motor Principal 1",
    "ubicacion": "Bloque 4",
    "notas": null
  }
]
```

B.2.2 Crear Motor

POST /api/motores/

Respuesta:

{

"motor_id": 3,

"motor_codigo": "Motor_003",

"nombre": "Motor Secundario",

"ubicacion": "Zona Este",

"notas": "Nuevo motor instalado"

}

B.2.3 Obtener Motor

GET /api/motores/{motor_id}

Respuesta:

{

"motor_id": 1,

"motor_codigo": "Motor_001",

"nombre": "Motor Principal 1",

"ubicacion": "Bloque 4",

"notas": "Mantenimiento reciente"

}

B.2.4 Actualizar Motor

PUT /api/motores/{motor_id}

Respuesta:

{

"motor_id": 1,

"motor_codigo": "Motor_001",

"nombre": "Motor Principal Modificado",

"ubicacion": "Bloque 5",

"notas": "Cambio de ubicación"

}

B.2.5 Eliminar Motor

DELETE /api/motores/{motor_id}

Respuesta:

{}

B.3. Módulo: Unidades

B.3.1 Listar Unidades

GET /api/unidades/

Respuesta:

[

{ "unidad": "RPM" },

{ "unidad": "°C" }

]

B.3.2 Crear Unidad

POST /api/unidades/

Respuesta:

{

"unidad": "m/s²"

}

B.3.3 Actualizar Unidad

PUT /api/unidades/{unidad}

Respuesta:


```
{
  "unidad": "m/s²"
}
```


B.3.4 Eliminar Unidad

DELETE /api/unidades/{unidad}

Respuesta:


```
{}
```


B.4. Módulo: Variables

B.4.1 Listar Variables

GET /api/variables/

Respuesta:


```
[
  {
    "variable": "Speed",
    "unidad": "RPM",
    "cname": "Velocidad del motor"
  }
]
```


B.4.2 Crear Variable

POST /api/variables/

Respuesta:

{
"variable": "Temperature",
"unidad": "°C",
"cname": "Temperatura del motor"
}

B.4.3 Actualizar Variable

PUT /api/variables/{variable}

Respuesta:

{
"variable": "Speed",
"unidad": "RPM",
"cname": "Velocidad Actualizada"
}

B.4.4 Eliminar Variable

DELETE /api/variables/{variable}

Respuesta:

{}

B.5. Módulo: Despliegues

5.1 Listar Despliegues

GET /api/despliegues/

Respuesta:

[
{
"despliegue_id": 1,

```
    "asset_id": 1,
    "motor_id": 1,
    "inicio": "2025-01-01T00:00:00Z",
    "fin": null
  }
]
```


B.5.2 Crear Despliegue

POST /api/despliegues/

Respuesta:


```
{
  "despliegue_id": 2,
  "asset_id": 1,
  "motor_id": 2,
  "inicio": "2025-10-01T00:00:00Z",
  "fin": null
}
```


B.5.3 Actualizar Despliegue

PUT /api/despliegues/{despliegue_id}

Respuesta:


```
{
  "despliegue_id": 1,
  "asset_id": 1,
  "motor_id": 1,
  "inicio": "2025-01-01T00:00:00Z",
```

"fin": "2025-11-01T00:00:00Z"

}

B.5.4 Eliminar Despliegue

DELETE /api/despliegues/{despliegue_id}

Respuesta:

{}

B.6. Módulo: Ingestas

B.6.1 Listar Ingestas

GET /api/ingestas/

Respuesta:

[

{

"ingesta_id": 1001,

"asset_codigo": "37156",

"motor_codigo": "Motor_001",

"variable": "Speed",

"valor": 1450.3

}

]

B.6.2 Crear Ingesta

POST /api/ingestas/

Respuesta:

{

"ingesta_id": 137092,

```
  "asset_codigo": "37156",
  "motor_codigo": "Motor_001",
  "ts_utc": "2025-11-20T18:39:23.349Z",
  "variable": "Speed",
  "valor": 1510.0,
  "archivo_origen": "sensor_X"
}
```


B.6.3 Últimas Ingestas

GET /api/ingestas/latest

Respuesta:


```
[
  { "ingesta_id": 137092, "variable": "Speed", "valor": 1510.0 }
]
```


B.6.4 Últimas N Ingestas

GET /api/ingestas/last/{n}

B.6.5 Obtener Ingesta

GET /api/ingestas/{id}

Respuesta:


```
{
  "ingesta_id": 137092,
  "variable": "Speed",
  "valor": 1510.0
}
```

B.6.6 Actualizar Ingesta

PUT /api/ingestas/{id}

Respuesta:

{

"ingesta_id": 137092,

"valor": 1520.0

}

B.6.7 Eliminar Ingesta

DELETE /api/ingestas/{id}

B.6.8 Exportar Ingestas CSV

GET /api/ingestas/export/csv

B.7. Módulo: Mediciones

B.7.1 Importar desde Ingestas

POST /api/mediciones/import-from-ingestas

Respuesta:

{ "inserted": 1 }

B.7.2 Últimas Mediciones

GET /api/mediciones/latest

B.7.3 Últimas N Mediciones

GET /api/mediciones/last/{n}

B.7.4 Listar Mediciones

GET /api/mediciones/

B.7.5 Crear Medición

POST /api/mediciones/

B.7.6 Exportar Mediciones CSV

GET /api/mediciones/export/csv

B.7.7 Actualizar Medición

PUT /api/mediciones/{id}

B.7.8 Eliminar Medición

DELETE /api/mediciones/{id}

B.8. Módulo: Analytics

B.8.1 Trend (Tendencia)

POST /api/analytics/analytics/trend

Respuesta:


```
{
  "variable": "Speed",
  "points": [
    { "ts": "2025-11-20T13:40:01", "valor": 1479.2 }
  ]
}
```

B.8.2 Scatter

POST /api/analytics/analytics/scatter

B.8.3 Histogram

POST /api/analytics/analytics/histogram

B.8.4 Variables disponibles

GET /api/analytics/analytics/available-variables

B.8.5 Export CSV Analytics

GET /api/analytics/analytics/export-csv