Sistema de Monitorización para el Sistema Solar Fotovoltaico de la UPB

Proyecto Aplicado en TIC II

PARTICIPANTES

Juan Esteban Galeano

Luisa Álvarez Bello

Sofía Arango

Felipe Buitrago

DOCENTE

Jackson Reina

Universidad Pontificia Bolivariana

Escuela de Ingeniería

Facultad de TIC

2023

TABLA DE CONTENIDO

TABLA DE FIGURAS

1. INTRODUCCIÓN

Las ciudades inteligentes se apoyan en las tecnologías de la información y la comunicación (TIC) para avanzar en la mejora de la calidad de vida de sus habitantes, la eficiencia de los servicios públicos y aminorar el impacto en el medio ambiente. Utilizan sensores, dispositivos móviles, sistemas de información geográfica, inteligencia artificial y otras tecnologías avanzadas para recopilar y analizar datos en tiempo real.

Por ello, se deben tener sistemas de monitoreo y análisis para los dispositivos. En ese orden de ideas, aparece la arquitectura Fiware, que junto a unos componentes como el agente IoT (que actúa como una interfaz entre los dispositivos del mundo real y la plataforma), permiten la recopilación de datos desde los sensores y facilitan la toma de decisiones. Debido a eso, surgen incógnitas de cómo se deben organizar esos componentes.

Este informe proporciona una visión detallada del progreso del proyecto "Sistema de Monitorización para el Sistema Solar Fotovoltaico de la UPB". En el proyecto se desarrolló una solución tecnológica basada en la arquitectura Fiware para monitorizar y gestionar el sistema solar fotovoltaico de la Universidad Pontificia Bolivariana (UPB) en tiempo real de forma óptima para su correcta integración.

2. RESUMEN EJECUTIVO

En el marco del proyecto "Sistema de Monitorización para el Sistema Solar Fotovoltaico de la UPB", se lograron avances significativos en la creación de una arquitectura FIWARE robusta mediante el uso de contenedores Docker. Esta arquitectura incorpora componentes clave, como el Orion Context Broker, Quantumleap, Grafana y CrateDB. Además, se desarrolló un agente IoT personalizado para la lectura de datos provenientes de las fuentes Fronius Devices, Fronius Data Manager y Enphase Data Manager. La arquitectura y la integración de todos los componentes se completaron, estableciendo así las bases para una monitorización efectiva.

En adición, se crearon prototipos de dashboards en Grafana que posibilitan la visualización de información y datos en tiempo real. Asimismo, se finalizó el cálculo de los indicadores de interés para el cliente, el Smart Energy Center (SEC). Estos logros representan sentaron las bases hacia la implementación de una solución tecnológica que permite la monitorización eficiente y en tiempo real del sistema solar fotovoltaico de la Universidad Pontificia Bolivariana (UPB).

3. OBJETIVOS

3.1. Objetivo General

- Diseñar y desarrollar una arquitectura Fiware que permita la recopilación y procesamiento eficiente de datos del sistema solar fotovoltaico.

3.2. Objetivos Específicos

- Integrar los componentes clave, incluyendo el Orion Context Broker, Quantumleap, Grafana, y CrateDB, en la arquitectura.
- Desarrollar un agente IoT personalizado para la lectura de datos de fuentes específicas, como Fronius Devices, Fronius Data Manager y Enphase Data Manager.
- Crear prototipos de dashboards en Grafana para la visualización en tiempo real de los datos del sistema solar.
- Iniciar el cálculo de indicadores de interés para el cliente, el SEC.

4. PRESENTACIÓN DEL CONTEXTO

En el ámbito de la UPB, se ha establecido un importante centro académico conocido como Smart Energy Center (SEC), cuyo objetivo principal es investigar de manera exhaustiva las tecnologías emergentes en el entorno metropolitano. Este centro ha implementado un modelo piloto en el campus universitario, basado en tres pilares fundamentales que lo definen y caracterizan: Flexibilidad Energética, Territorios Inteligentes y, Energía Sostenible y Regenerativa.

Este estudio se enfoca exclusivamente en la dimensión de los Territorios Inteligentes, un ámbito que abarca tres variables interrelacionadas. En primer lugar, se aborda la relación entre el gobierno y los ciudadanos, explorando temas de democracia, gobierno abierto, ciudades inteligentes, seguridad ciudadana, situaciones de emergencia y contingencias. En paralelo, surge el aspecto de la infraestructura, donde los desafíos involucran cuestiones energéticas, de gas, iluminación, movilidad, *agri-food* inteligente y procesos cíclicos. Por último, se pone atención en el entorno y la calidad de vida, con enfoques en planificación, salud inteligente, educación avanzada, turismo y entretenimiento.

Cabe destacar que este panorama se enmarca en el iHub UPB Medellín, un centro tecnológico pionero en su clase en el contexto urbano. Este espacio se dedica a abordar todos los problemas mencionados anteriormente en el contexto de Medellín, con el propósito de encontrar soluciones y fomentar la adopción de tecnologías avanzadas, digitalización, formación, incubación de startups y otras iniciativas, todas orientadas a impulsar el desarrollo del entorno.

Dentro del iHub de Territorios Inteligentes, se desarrollan diferentes aspectos como el iHub Lab y el EcoCampus Inteligente UPB, que ejemplifican la variedad de tecnologías implementadas. Este conjunto de elementos incluye desde medidores de energía y almacenamiento energético hasta un centro de control y generación fotovoltaica, entre otros componentes. Un software integral supervisaba rigurosamente estos elementos, encargándose de recopilar, gestionar y visualizar gráficamente los datos provenientes de los sensores correspondientes, aunque actualmente no se encuentra operativo.

5. PRESENTACIÓN DEL PROBLEMA

Es de suma importancia para el SEC haber contado con una visualización de los datos generados por los inversores de sistemas solares fotovoltaicos (SFV), con el objetivo de respaldar los procesos de monitorización de la transformación en el eje de Energía y Desarrollo Sostenible en el EcoCampus UPB.

El foco principal de este problema residía en la falta de recolección de datos provenientes de estos inversores, ya que el software integral encargado del ciclo de los datos no estaba operativo. En una fase inicial, fue necesario abordar la solución para la recolección de datos, seguida de su limpieza, almacenamiento, procesamiento y, finalmente, visualización una vez que completaran todo el proceso.

El abordaje de este problema ha permitido cubrir integralmente el ciclo de vida de los datos provenientes de los inversores fotovoltaicos, desde su recolección hasta su visualización, facilitando así la toma de decisiones informadas. Cabe destacar que el proyecto ha finalizado con éxito, logrando implementar soluciones efectivas para la gestión de datos en el contexto de sistemas solares fotovoltaicos.

6. PROPUESTA DE SOLUCIÓN

Se desarrolló una solución completa, integral, robusta y eficiente para la gestión de datos generados por el sistema solar fotovoltaico (SFV) en el entorno del Smart Energy Center (SEC). Esta solución se planteó e implementó para ser totalmente compatible con la plataforma Fiware y abordar el tratamiento, limpieza, monitorización, visualización y análisis de los datos en tiempo real. La solución se basó en una arquitectura modular desplegada en contenedores Docker, cumpliendo con los requisitos establecidos por el cliente y se describe a continuación:

6.1. Arquitectura Propuesta

- **Cliente (WEB):** Los usuarios acceden a la solución a través de un navegador web, lo que les permite monitorear y acceder a los datos generados por los SFV.

- **Sistemas Solares Fotovoltaicos:** El sistema completo genera datos como la producción de energía, voltajes, corrientes y otros. Estos datos serán capturados y transmitidos al sistema de solución.

- **Agente IoT:** Se implementó un agente IoT que se encargará de recopilar los datos del SFV y enviarlos al Orion Context Broker (OCB) de Fiware. Este agente asegura una comunicación fluida y segura entre los dispositivos y la plataforma.

- **Plataforma Fiware:** Se utilizó una solución basada en Fiware que incluyó componentes como OCB, MongoDB para el almacenamiento de datos de contexto y metadatos, Quantumleap para la persistencia de datos históricos y un Data Lake para el almacenamiento inicial de datos crudos.

- **Componente de ETL:** Se desarrolló un componente de Extracción, Transformación y Carga (ETL) que extrae los datos del agente IoT, realiza procesos de limpieza, normalización y transformación, y luego carga los datos procesados en una bodega de datos o *Data Warehouse* (DW).

- **Data Warehouse (DW):** Este almacén de datos contiene los datos procesados y transformados, organizados para análisis y consulta eficientes. Proporcionó una base sólida para la toma de decisiones informadas y análisis avanzados.

- **Componente de visualización (Grafana):** Se integró Grafana, una herramienta de visualización de datos, para crear paneles personalizados y gráficos interactivos que permitan a los usuarios comprender fácilmente los datos y patrones, facilitando la toma de decisiones basada en datos.

6.2. Tecnologías y Metodologías

La solución se implementó utilizando contenedores Docker para garantizar la portabilidad y escalabilidad. Se manejaron tecnologías y herramientas de código abierto compatibles con la filosofía de *Open Source*, así como con el modelo de información NGSI y la plataforma Fiware. El equipo de desarrollo tuvo total libertad de elección sobre el lenguaje de programación a usar, que en este caso fue Python.

6.3. Documentación y Capacitación

La solución está documentada siguiendo estándares establecidos, lo que incluye descripciones técnicas, diagramas de arquitectura, instrucciones de implementación y configuración.

6.4. Implementación y Despliegue

El desarrollo se realizó en un servidor local montado en la nube de Amazon Web Services (AWS), este enfoque garantizará la validación y pruebas adecuadas antes de la implementación final.

6.4.1. Beneficios

- **Monitorización Avanzada:** El sistema permitió un monitoreo en tiempo real y un análisis detallado de los sistemas solares fotovoltaicos (SFV), facilitando la detección temprana de problemas y optimizando su rendimiento.
- **Toma de Decisiones Informada:** Los datos procesados y visualizados proporcionan información valiosa para la toma de decisiones relacionadas con el mantenimiento, la eficiencia y la planificación futura.
- **Cumplimiento de Requisitos:** La solución cumple con todas las restricciones y requisitos establecidos por el cliente, asegurando una implementación exitosa y una alineación con los objetivos del SEC.
- **Eficiencia energética:** Al permitir una mejor gestión y optimización de los SFV, la solución contribuirá directamente a los objetivos de sostenibilidad y eficiencia energética del cliente.
- **Escalabilidad:** La arquitectura modular basada en contenedores Docker permitió la escalabilidad de la solución a medida que se agreguen más SFV y aumente la cantidad de datos a manejar.
- **Facilitación de adopción:** La documentación detallada ayudarán al personal del SEC a adoptar y utilizar eficazmente la solución.

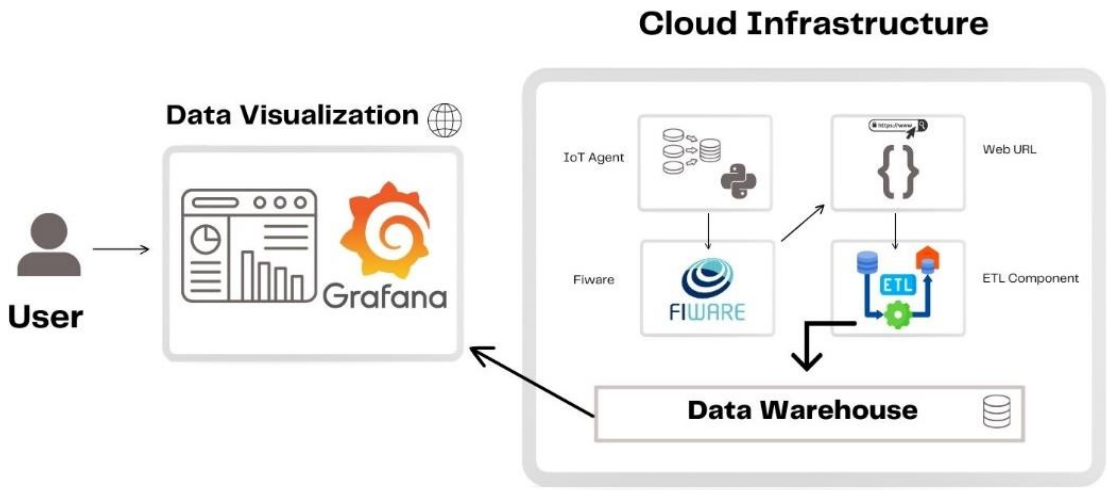


Fig. 1 Diseño Arquitectónico de Elaboración propia.

Se consideró poco eficiente leer los datos directamente desde la fuente debido a que allí pueden existir valores nulos, atípicos, y los datos no tienen una estructura que facilite su visualización. Para ello se desarrolló un componente ETL que extrae, procesa los datos y calcula los indicadores requeridos, haciendo control de valores atípicos y nulos, además de estructurar los datos de una manera que sea propicia para la visualización. Adicionalmente, la bodega de datos mejora el rendimiento en la lectura, lo que evita posibles latencias en la actualización del dashboard.

En resumen, la arquitectura propuesta proporcionará una solución robusta y escalable para la gestión de datos de sistemas solares fotovoltaicos en el ecosistema del SEC. Esto permitirá una gestión más eficiente de la energía solar y contribuirá al logro de los objetivos de sostenibilidad y eficiencia energética del cliente.

6.5. Arquitectura del Agente IoT

El Agente IoT es una aplicación modular desarrollada en Python diseñada para desempeñar diversas funciones en el proyecto. Estas funciones clave incluyen:

- Extracción de datos en tiempo real, con una frecuencia de cada 30 segundos, procedentes de los dispositivos que componen el sistema solar fotovoltaico de la Universidad Pontificia Bolivariana (UPB).

- Realización de un proceso de Extracción, Transformación y Carga (ETL) a los datos extraídos del sistema, seguido de su envío al Orion Context Broker, que actúa como el núcleo en la arquitectura Fiware.
- Ejecución de un proceso adicional de ETL que obtiene los datos en tiempo real del Orion Context Broker, los transforma y posteriormente los envía a una bodega de datos (Data Warehouse - DW).
- Para llevar a cabo estas tareas, el Agente IoT opera mediante la ejecución de varios hilos de trabajo de procesamiento simultáneo, lo que permite una gestión eficiente de las operaciones.
- La aplicación hace uso de protocolos de comunicación HTTP para extraer datos de las fuentes originales y enviarlos al Orion Context Broker. Además, utiliza un módulo especializado en realizar un proceso de web scraping para la extracción de datos desde uno de los dispositivos del sistema, lo que contribuye a la recopilación y análisis efectivo de la información.

7. CRONOGRAMA

El desarrollo del cronograma de todas las fases del proyecto, donde se ve evidenciado una aproximación del tiempo que se dedicó a cada una de las tareas propuestas, siendo estas últimas tareas relativamente generales e interrelacionadas.

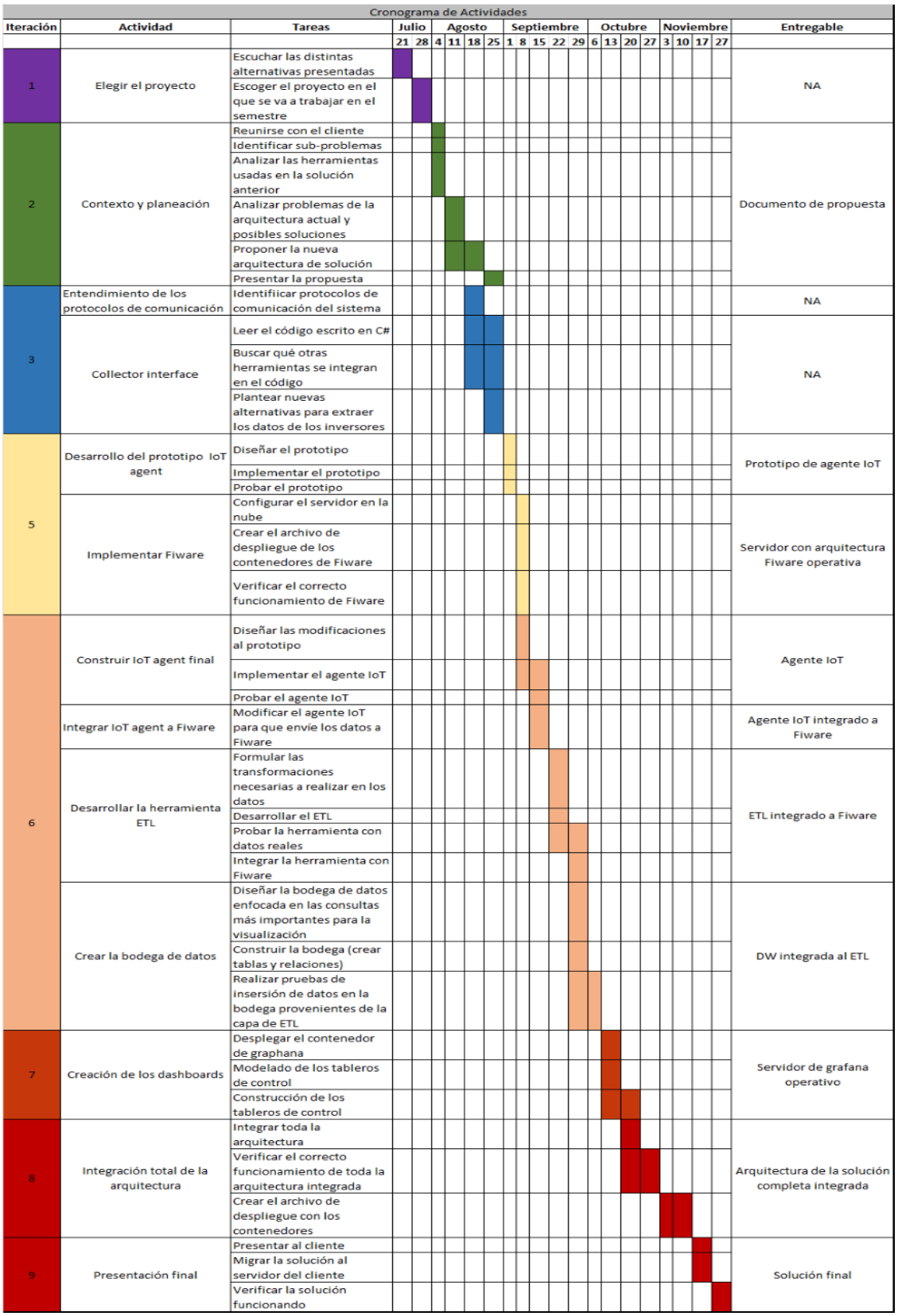


Fig. 2 Cronograma de Elaboración propia

8. DESARROLLO

Se han alcanzado los siguientes hitos y logros:

- Creación exitosa de la arquitectura Fiware, implementada en contenedores Docker.
- Desarrollo del agente IoT personalizado para la lectura de datos de las fuentes mencionadas.
- Integración completa de los componentes clave de la arquitectura, incluyendo Orion Context Broker, Quantumleap, Grafana y CrateDB.
- Prototipos de dashboards en Grafana para la visualización en tiempo real de datos.
- Creación de la bodega de datos en el motor PostgreSQL, con el fin de cargar allí los datos procesados necesarios para el cálculo de los indicadores.
- Extracción, transformación y carga de los datos provenientes de los sensores con destino a la bodega de datos.

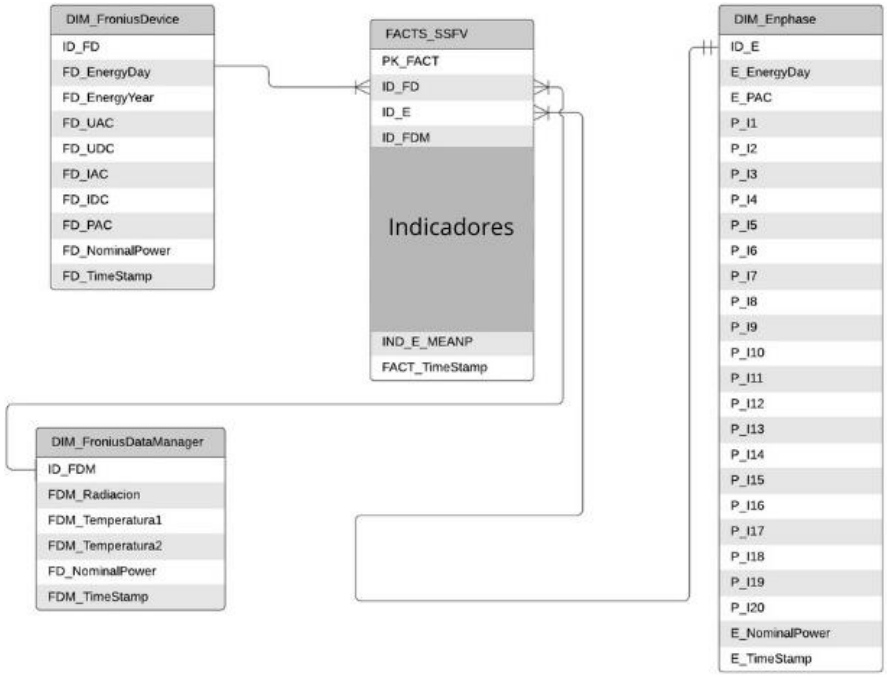


Fig. 3 Diagrama UML del DW

- Refactorización de códigos que permitió mejorar la recopilación de datos y optimizar la lectura de datos en Grafana.
- Visualización de todos los indicadores y variables correspondientes a los agentes IoT.

8.1.Definición de Indicadores

En la concepción y desarrollo de este proyecto, la adopción de indicadores ha sido fundamental para brindar una comprensión rápida y precisa del estado operativo y las tendencias clave. Como mencionado previamente, los datos crudos recolectados de cada uno de los sensores se han sometido a un proceso de transformación y consolidación en el DW. Una vez procesados, los indicadores se han integrado de manera fluida en los Dashboard de Grafana, destinados al monitoreo continuo.

Aquí, los clientes pueden acceder a una representación visual de estos indicadores, lo que les permite identificar anomalías o variaciones inesperadas de manera inmediata. A partir de los datos recopilados se hicieron unos cálculos para hallar los indicadores necesarios para la toma de decisiones:

Indicador	Nombre	Descripción	Unidades	Dispositivos por medir
FC	Factor de Carga	Eficiencia Eléctrica	Kilowatts Hora (kWh)	Fronius
				Enphase
HSP	Horas Solares Pico	Cantidad de energía solar que recibe un m2 de superficie	Horas	Fronius
				Enphase
UACVSUDC	Voltaje AC VS. Voltaje DC	Relación del voltaje de corriente alterna con voltaje de corriente directa	Voltios (V)	Fronius
IACVSIDC	Corriente Alterna VS. Corriente Directa	Relación de la corriente alterna con corriente directa	Amperios (A)	Fronius
MEANP	Promedio Potencias	Promedio de las potencias de todos los microinversores Emphase Device	Watts (W)	Enphase

Fig. 4 Indicadores

8.2.Documentación Códigos

En esta sección, se proporciona una descripción general de los programas empleados en la construcción de la arquitectura propuesta. Se detalla el propósito y funcionamiento general de cada programa, brindando una visión general de su contribución al sistema.

Es importante destacar que para un entendimiento más detallado sobre cómo manipular los métodos y operaciones de cada programa, se recomienda consultar el repositorio asociado en GitHub. A continuación, se presenta el nombre de cada programa, una breve descripción y una explicación general de sus funciones en el contexto de la arquitectura propuesta. Todo esto para proporcionar información más específica y detallada, proporcionando instrucciones detalladas sobre el uso, configuración y personalización de cada programa.

Main.py

Este código organiza y ejecuta múltiples hilos para realizar las tareas de recolección, asegurando una ejecución continua y concurrente de las operaciones.

- Se importan los módulos **time**, **threading**, **fronius**, **froniusDataManager**, **enphaseManager**, **InsertFacts**, y **json**.
- La función **openJson(fileName) a**bre y carga un archivo JSON especificado por **fileName** y devuelve los datos contenidos en el archivo JSON.
- **intervalo_consulta** establece el intervalo de tiempo (en segundos) entre cada consulta de datos. En este caso, se fija en 30 segundos (**intervalo_consulta = 30**).
- Las funciones de configuración de hilos **(setupThread, setupThreadEnphase, setupThreadInsertFacts)** configuran hilos para realizar tareas específicas de manera continua en un intervalo dado.
- La función **setupThread** recibe una función **getDataFunction** y una URL como argumentos, y realiza llamadas a esta función en un bucle, seguido de un tiempo de espera (**time.sleep(intervalo_consulta)**).
- La función **setupThreadEnphase** realiza una tarea similar, pero para la función **getDataFunction** asociada con el Enphase Device.
- La función **setupThreadInsertFacts** está diseñada para llamar a la función **insertIntoDwh** del módulo **InsertFacts** en un bucle con un intervalo de espera.
- Las funciones (**urls_fronius** y **urls_fronius_data_manager**) abren archivos JSON que contienen las URL necesarias para la obtención de datos de dispositivos Fronius y Fronius Data Manager.
- La función **threading.Thread** configura varios hilos para realizar la recolección de datos de Fronius Devices, Fronius Data Manager, Enphase Devices y la inserción de hechos en el DWH.
- La función (**thread.start()**) inicia los hilos.
- Y la función (**thread.join()**) espera a que todos los hilos finalicen. Este bloque **join** bloqueará la ejecución del programa principal hasta que todos los hilos hayan completado su ejecución.

InsertFacts.py

Este código se enfoca en la manipulación de datos provenientes de la base de datos, realiza los cálculos para generar los indicadores previamente mencionados y finalmente inserta esos indicadores y detalles adicionales en una tabla en la misma base de datos.

- La función **connect_to_database** establece una conexión a la base de datos PostgreSQL utilizando los parámetros suministrados.
- La función **get_last_dim_enphase_record** recupera el registro más reciente de la tabla llamada dim_enphase.
- La función **get_last_dim_froniusdatamanager_record** obtiene el registro más reciente de la tabla llamada dim_froniusdatamanager.
- La función **get_last_dim_froniusdevice_records** recupera múltiples registros de latabla llamada dim_froniusdevice.
- La función **calculateFdInd** calcula los indicadores de Fronius Device a partir de los registros de la tabla dim_froniusdevice.
- La función **calculateEInd** realiza cálculos para generar los indicadores de Enphase Device a partir del registro de la tabla dim_enphase.
- La función **InsertIntoDwh** accede a la base de datos utilizando la conexión establecida previamente, calcula indicadores adicionales, organiza datos de manera más específica, asignando variables a diferentes campos y valores de los registros obtenidos y lleva a cabo la inserción de datos en la tabla facts_ssfv.

dwInsetions.py

Esta clase dw_insertions proporciona métodos para insertar datos relacionados con los dispositivos Fronius, Enphase y Fronius Data Manager en la base de datos PostgreSQL.

- **constructor (__init__)** Inicializa la clase con los parámetros de la base de datos (**db_params**) y establece la conexión a la base de datos llamando a la función **connect_to_database**.
- La función **connect_to_database(self) e**stablece la conexión a la base de datos PostgreSQL utilizando los parámetros proporcionados en el método (**self.db_params**).

- La función **insert_fronius(self, data)** inserta datos en la tabla **DIM_FroniusDevice** de la base de datos, recibe un diccionario de datos (**data**) relacionados con los dispositivos Fronius e itera sobre el diccionario y ejecuta una consulta SQL de inserción para cada dispositivo.
- La función **insert_enphase(self, data)**, inserta datos en la tabla **DIM_Enphase** de la base de datos, recibe un diccionario de datos (**data**) relacionados con dispositivos Enphase e itera sobre el diccionario y ejecuta una consulta SQL de inserción para cada dispositivo.
- La función **insert_froniusdm(self, data)** inserta datos en la tabla **DIM_FroniusDataManager** de la base de datos, recibe un diccionario de datos (**data**) relacionados con dispositivos Fronius Data Manager, e itera sobre el diccionario y ejecuta una consulta SQL de inserción para cada dispositivo.
- La función **close_connection(self) c**ierra la conexión a la base de datos si existe.

enphaseManager.py

Este código automatiza la extracción de datos de la plataforma web enphaseenergy, procesa y organiza estos datos, y luego los utiliza para actualizar una entidad en Orion y almacenarlos en la base de datos PostgreSQL.

- Importa las bibliotecas necesarias, como **webdriver** de Selenium, y define las credenciales de inicio de sesión, URL de inicio de sesión y opciones para ejecutar Firefox en modo headless.
- La función **scroll_to_element(driver, element)** define una función para desplazarse hasta que un elemento sea visible en la página.
- Configura el navegador Firefox en modo headless y maximiza la ventana.
- Utiliza Selenium para navegar a la página de inicio de sesión, ingresar las credenciales y hacer clic en el botón de inicio de sesión.
- Obtiene datos como la energía total, la energía del día y la potencia generada por microinversores desde la página web de Enphase.
- Utiliza Selenium para iterar sobre las filas de una tabla que contiene información sobre microinversores.
- La función **microInvertersData** crea un diccionario que mapea el número de serie de los microinversores a la potencia generada por cada uno.

- La función **createPostJSON** construye un JSON que contiene información sobre la energía del día, la energía total, la potencia generada y la potencia de cada microinversor.
- La función **actualizarEntidad** utiliza la biblioteca **requests** para realizar una solicitud PATCH a una entidad en el servicio Orion Context Broker, actualizando los datos con el JSON construido anteriormente. También inserta los datos en una base de datos PostgreSQL utilizando la clase **dw_insertions**.

fronius.py

Este código obtiene datos de dispositivos Fronius desde URLs específicas, procesa y transforma estos datos, y luego actualiza una entidad en Orion (OCB). Además, realiza inserciones de datos en la base de datos PostgreSQL.

- La función **getData(urls)** realiza solicitudes HTTP GET a las URLs especificadas en el diccionario **urls**. Si la solicitud es exitosa (código de estado 200), extrae datos específicos de dispositivos Fronius y llama a la función **crearPostJSON** para procesar y enviar estos datos.
- La función **extrerDatosFronisDevices(json_froniusDevice, url)** extrae información específica de dispositivos Fronius del JSON recibido como respuesta de la solicitud HTTP. Luego construye un nuevo diccionario (**data_final**) con datos seleccionados para su posterior procesamiento, y llama a la función **crearPostJSON** con los datos procesados y la URL original.
- La función **crearPostJSON(data_final, url)** construye un nuevo JSON (**post_json**) a partir de los datos procesados, transforma la URL original mediante la función **transform_url** y llama a la función **actualizarEntidad** para realizar una solicitud PATCH al Orion Context Broker (OCB) y actualizar la entidad correspondiente.
- La función **transform_url(url, deviceId)** transforma la URL original para adaptarla a un formato específico esperado por el OCB, extrae el ID del dispositivo original de la URL y reemplaza este ID con uno nuevo basado en la IP y el ID del dispositivo Fronius y devuelve la nueva URL transformada.

- La función **actualizarEntidad(url, json_data)** realiza una solicitud PATCH al OCB utilizando la URL transformada y el JSON de datos e inserta datos en labase de datos PostgreSQL utilizando la clase **dw_insertions**.

froniusDataManager.py

Este código obtiene datos del Fronius Data Manager desde URLs específicas, procesa y transforma estos datos, y luego actualiza una entidad Orion (OCB). También realiza inserciones de datos en la base de datos PostgreSQL.

- La función **getData(urls)** realiza solicitudes HTTP GET a las URLs especificadas en el diccionario **urls**. Si la solicitud es exitosa (código de estado 200), extrae datos de Fronius Data Manager utilizando la función **extraerDatosFroniusDataManager**. En caso de un error en la solicitud, imprime el código de estado.
- La función **metodo_patch(datos)** realiza una solicitud PATCH al Orion Context Broker (OCB) para actualizar una entidad específica con los datos proporcionados. Utiliza la clase **dw_insertions** para insertar los datos en la base de datos PostgreSQL y devuelve un mensaje indicando el resultado de la solicitud PATCH.
- La función **extraerDatosFroniusDataManager(json_froniusDM)** extrae datos específicos del Fronius Data Manager del JSON recibido como respuesta de la solicitud HTTP. Construye un nuevo diccionario (**data_final**) con los datos seleccionados para su posterior procesamiento y actualización mediante la función **metodo_patch**. Devuelve el diccionario **data_final**.

getJson.py

Este código proporciona funciones para realizar solicitudes HTTP GET a las URLs especificadas, obtener datos JSON de esas URLs y cargar estos datos en un formato estructurado.

- La función **get_json_from_url(url)** realiza una solicitud HTTP GET a la URL proporcionada. Si la solicitud es exitosa (código de estado 200), convierte la respuesta JSON en un diccionario de Python y lo devuelve.

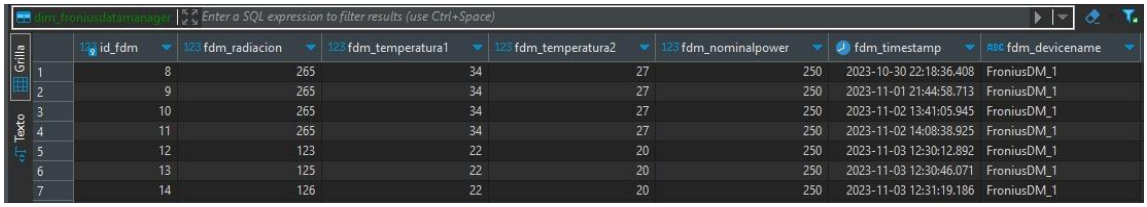
- La función **load_data(urls)** abre y lee un archivo JSON que contiene un diccionario de URLs, donde cada entrada tiene una identificación (**url_id**) y una URL asociada. Itera sobre las entradas del diccionario y utiliza la función **get_json_from_url** para obtener datos JSON de cada URL. Para cada conjunto de datos JSON obtenido, crea un nuevo diccionario (**result_dict**) donde las claves son las claves originales del JSON y los valores son los valores correspondientes. Elimina la clave 'id' del diccionario resultante y usa su valor como clave principal en un nuevo diccionario (**final_dict**). Devuelve el diccionario final, donde las claves son los valores originales de la clave 'id' y los valores son diccionarios con los datos obtenidos de las URLs.

9. PRUEBAS Y VALIDACIÓN

Dentro de los logros alcanzados se han establecidos varias métricas técnicas para evaluar el buen funcionamiento de la solución y la óptima respuesta. Se realizaron las siguientes pruebas:

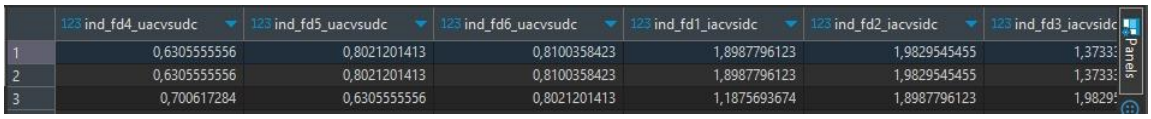
- **Pruebas de Integración:** Se validó que los componentes clave de Fiware se integren correctamente en la solución y que la comunicación entre ellos sea fluida.
- **Pruebas de Rendimiento:** Se evaluaron diferentes códigos y el uso de hilos para la optimización de estos hasta conseguir una respuesta rápida.
- **Pruebas de Conectividad:** Se validó la capacidad de la solución para conectarse a los dispositivos IoT y se comprobó la recolección y envío de datos de manera controlable.
- **Pruebas de Funcionalidad:** Se verificaron todos los procesos dentro de los aplicativos y métodos en Python, se automatizó la captura de errores y las alertas relacionadas necesarias para que el correcto funcionamiento no se viera afectado.

  Para validar el correcto funcionamiento de la extracción, transformación y carga de los datos provenientes de los sensores con destino a la bodega de datos, se visualizaron las inserciones cada 30 segundos en la DW



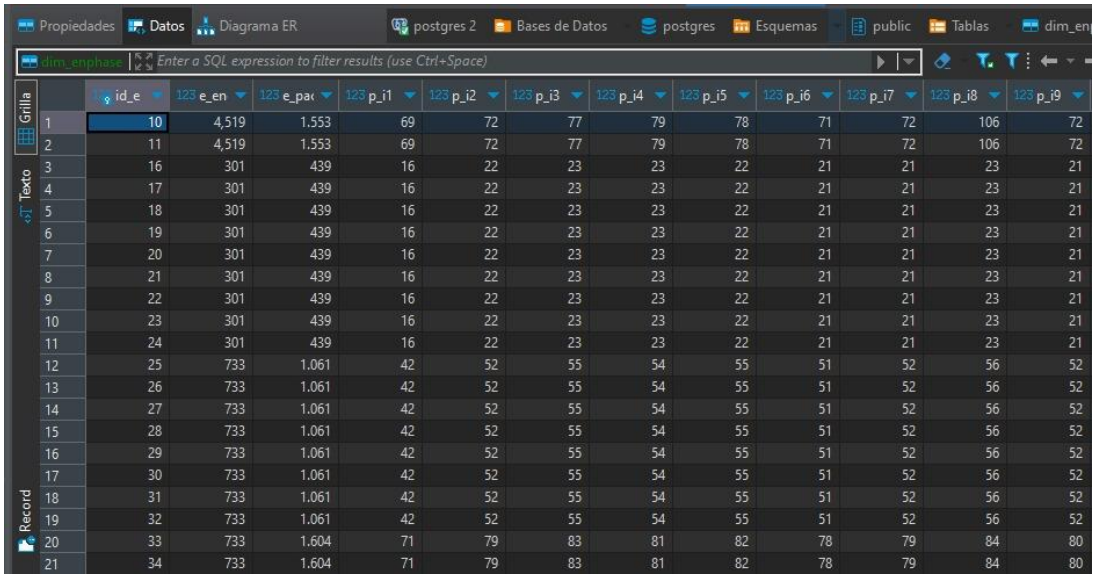
	id_fdm	fdm_radiacion	fdm_temperatura1	fdm_temperatura2	fdm_nominalpower	fdm_timestamp	fdm_devicename
1	8	265	34	27	250	2023-10-30 22:18:36.408	FroniusDM_1
2	9	265	34	27	250	2023-11-01 21:44:58.713	FroniusDM_1
3	10	265	34	27	250	2023-11-02 13:41:05.945	FroniusDM_1
4	11	265	34	27	250	2023-11-02 14:08:38.925	FroniusDM_1
5	12	123	22	20	250	2023-11-03 12:30:12.892	FroniusDM_1
6	13	125	22	20	250	2023-11-03 12:30:46.071	FroniusDM_1
7	14	126	22	20	250	2023-11-03 12:31:19.186	FroniusDM_1

Fig. 5 Validación datos Fronius DM



	ind_fd4_uacvsudc	ind_fd5_uacvsudc	ind_fd6_uacvsudc	ind_fd1_iacvsidc	ind_fd2_iacvsidc	ind_fd3_iacvsidc
1	0,6305555556	0,8021201413	0,8100358423	1,8987796123	1,9829545455	1,3733
2	0,6305555556	0,8021201413	0,8100358423	1,8987796123	1,9829545455	1,3733
3	0,700617284	0,6305555556	0,8021201413	1,1875693674	1,8987796123	1,9829

Fig. 6 Validación datos Fronius Devices



	id_e	e_en	e_pac	p_i1	p_i2	p_i3	p_i4	p_i5	p_i6	p_i7	p_i8	p_i9
1	10	4,519	1.553	69	72	77	79	78	71	72	106	72
2	11	4,519	1.553	69	72	77	79	78	71	72	106	72
3	16	301	439	16	22	23	23	22	21	21	23	21
4	17	301	439	16	22	23	23	22	21	21	23	21
5	18	301	439	16	22	23	23	22	21	21	23	21
6	19	301	439	16	22	23	23	22	21	21	23	21
7	20	301	439	16	22	23	23	22	21	21	23	21
8	21	301	439	16	22	23	23	22	21	21	23	21
9	22	301	439	16	22	23	23	22	21	21	23	21
10	23	301	439	16	22	23	23	22	21	21	23	21
11	24	301	439	16	22	23	23	22	21	21	23	21
12	25	733	1.061	42	52	55	54	55	51	52	56	52
13	26	733	1.061	42	52	55	54	55	51	52	56	52
14	27	733	1.061	42	52	55	54	55	51	52	56	52
15	28	733	1.061	42	52	55	54	55	51	52	56	52
16	29	733	1.061	42	52	55	54	55	51	52	56	52
17	30	733	1.061	42	52	55	54	55	51	52	56	52
18	31	733	1.061	42	52	55	54	55	51	52	56	52
19	32	733	1.061	42	52	55	54	55	51	52	56	52
20	33	733	1.604	71	79	83	81	82	78	79	84	80
21	34	733	1.604	71	79	83	81	82	78	79	84	80

Fig. 7 Validación datos Enphase Device

10. RECURSOS UTILIZADOS

Se emplearon los siguientes elementos y herramientas:

- Equipo de desarrollo con experiencia en Fiware.
- Contenedores Docker para implementar la arquitectura Fiware.
- Fuentes de datos, incluyendo Fronius Devices, Fronius Data Manager y Enphase Data Manager.
- Hardware y recursos de servidores para alojar la arquitectura.
- Herramientas de desarrollo de software y licencias necesarias, las cuales incluyen librerías necesarias para extraer los datos, transformarlos y cargarlos en dos destinos, un histórico de datos, y una bodega de datos.

11. DESAFÍOS Y PROBLEMAS

Uno de los desafíos principales que enfrentamos durante el desarrollo del proyecto fue la dificultad en la recolección de datos en tiempo real provenientes de diversas fuentes, especialmente en relación con la sincronización y la disponibilidad constante de los datos. Esto se debió a que los diferentes dispositivos emitían los datos en momentos temporales distintos. Además, experimentamos inconvenientes con la utilización de Grafana y con la conexión entre la plataforma y la base de datos. Finalmente, se presentaron dificultades en la simplificación del componente ETL escrito en Python, generando un retraso en su integración con el ecosistema FIWARE. Estos obstáculos fueron superados mediante estrategias específicas que permitieron avanzar hacia soluciones efectivas.

12. EVALUACIÓN DE RIESGOS

Los riesgos asociados a la toma de datos en tiempo real, la cantidad de recursos que utiliza la base de datos y la posible falla en la lectura de datos desde Grafana son consideraciones críticas para el proyecto. Hemos implementado medidas de contingencia para abordar estos riesgos, como el monitoreo constante y la configuración del servidor necesaria para aumentar los recursos necesarios para la base de datos empleada.

Así mismo, se desarrolló un script para ejecutar en el servidor, el cual estará configurado con reglas CRONTAB para su ejecución diaria con el objetivo de liberar recursos del servidor y reiniciar los servicios de los contenedores.

Es necesario señalar que los datos asociados a los dispositivos Enphase poseen una naturaleza delicada, ya que su obtención se realiza exclusivamente mediante web scraping. En numerosas ocasiones, la página web presenta tiempos de carga prolongados, aumentando la probabilidad de pérdida de algunas lecturas. Además, la eventual caída de dicha página web puede ocasionar interrupciones en la adquisición de datos.

13. CONCLUSIÓN

A lo largo del proyecto, enfrentamos desafíos en la obtención y sincronización de datos en tiempo real, así como en la interfaz entre la plataforma y la base de datos. Estrategias específicas superaron estos obstáculos, avanzando hacia soluciones operativas.

La evaluación de riesgos identificó factores críticos, como la captura de datos en tiempo real y la mitigación de posibles disrupciones en la lectura desde Grafana. Se establecieron medidas de contingencia, incluido un monitoreo constante y la optimización de recursos.

El proyecto alcanzó hitos fundamentales, desde la implementación exitosa de la arquitectura Fiware en contenedores Docker hasta el desarrollo del agente IoT personalizado. La integración efectiva de componentes clave y la introducción de prototipos en Grafana permitieron la visualización en tiempo real de datos.

La creación de la bodega de datos en PostgreSQL facilitó la carga de datos procesados, y la extracción, transformación y carga de datos desde los sensores se ejecutaron con éxito. La refactorización de códigos mejoró la recopilación y lectura de datos en Grafana, posibilitando la visualización completa de indicadores y variables.

En síntesis, el proyecto cumplió con los resultados esperados y requeridos por el cliente dentro de los plazos establecidos. La superación de desafíos, la implementación de medidas de contingencia y el cumplimiento exitoso de hitos clave evidencian el éxito del proyecto al proporcionar una solución tecnológica integral y eficiente para la monitorización del sistema solar fotovoltaico de la Universidad Pontificia Bolivariana.

13.1. Recomendaciones

Dadas las particularidades de este proyecto, se formulan diversas recomendaciones que revisten importancia en el proceso de implementación de la solución:

- Ambiente de Ejecución en la Red de la UPB: La solución debe ser implementada y ejecutada en la red de la Universidad Pontificia Bolivariana (UPB) por motivos de seguridad y confidencialidad.

- Servidor de Alta Capacidad Multihilo: Se hace hincapié en la necesidad de contar con un servidor de alta capacidad con capacidad multihilo, que permita un óptimo rendimiento y escalabilidad del sistema.
- Respaldo Semanal de Bases de Datos: Se recomienda realizar copias de seguridad de las bases de datos de la solución de forma semanal. Esto garantizará la integridad de los datos y la capacidad de restauración en caso de incidentes.
- Intervalos de Actualización Mínimos: Los tiempos de actualización de la solución no deben ser inferiores a 30 segundos, con el fin de mantener un flujo de información eficiente y evitar posibles inconvenientes de rendimiento.
- Diversificación de Fuentes para Agentes Enphase: Se aconseja sustituir la fuente de datos de Enphase por una alternativa más segura, empleando un protocolo de comunicación que inspire mayor confianza, como HTTP, TCP/IP, entre otros.

14. CONTACTO DE INTEGRANTES

Datos del Estudiante			
Nombre	Juan Esteban Galeano		
Correo electrónico	Juan.galeanoh@upb.edu.co		
ID Usuario (cédula)	1017241499	ID UPB	000138715

Datos del Estudiante			
Nombre	Luisa Álvarez		
Correo electrónico	Luisa.alvarezb@upb.edu.co		
ID Usuario (cédula)	1000654535	ID UPB	000433140

Datos del Estudiante	
Nombre	Sofía Arango

Correo electrónico	Sofia.arangop@upb.edu.co		
ID Usuario (cédula)	1000410366	ID UPB	000427556

Datos del Estudiante			
Nombre	Felipe Buitrago		
Correo electrónico	Felipe.buitragoj@upb.edu.co		
ID Usuario (cédula)	1001138381	ID UPB	000440004