Departamento de Matemática Aplicada a las Tecnologías de la Información y las Comunicaciones.

TRABAJO FIN DE GRADO

Diseño e implementación de un sistema de monitoreo y visualización de producción y consumo de energía dirigido a usuarios de Comunidades Energéticas descentralizadas

Grado en Sistemas de la Información

Autor: **Diego Ruiz Clemares**

Tutor: **Luis Miguel Carrasco Moreno**

Cotutor: **Rita Hogan Teves de Almeida**

Trabajo de Fin de Grado – Madrid - Julio de 2025

Índice

Contenido

Índice de Figuras

Índice de abreviaturas

- **FV**, Fotovoltaica
- **GWp**, Gigavatios pico (potencia máxima bajo condiciones estándar en energía fotovoltaica)
- **CE**, Comunidad Energética
- **CCE**, Comunidad Ciudadana de Energía
- **CER**, Comunidad de Energía Renovable
- **CERCA**, Comunidad de Energía Renovable de la Comarca de Calatayud
- **LIFE**, Programa LIFE de la Comisión Europea (Instrumento financiero para el medio ambiente y la acción climática)
- **UPM**, Universidad Politécnica de Madrid
- **TFG**, Trabajo de Fin de Grado
- **CUPS**, Código Universal de Punto de Suministro
- **IoT**, Internet of Things (Internet de las Cosas)
- API, Application Programming Interface (Interfaz de Programación de Aplicaciones)
- **MQTT**, Message Queuing Telemetry Transport (Protocolo de mensajería ligero para IoT)
- **kWh**, Kilovatio-hora (unidad de energía eléctrica)
- **ECODES**, Fundación Ecología y Desarrollo (Organización española dedicada a proyectos ambientales y sociales)
- **IDAE**, Instituto para la Diversificación y Ahorro de la Energía
- **ENERAGEN**, Asociación de Agencias Españolas de Gestión de la Energía
- **NIPO**, Número de Identificación de Publicaciones Oficiales
- **ARIMA**, AutoRegressive Integrated Moving Average (Modelo de Promedio Móvil Integrado Autoregresivo)
- **RNN**, Recurrent Neural Network (Red Neuronal Recurrente)
- **LSTM**, Long Short-Term Memory (Memoria a Largo y Corto Plazo, tipo de red neuronal)
- **JSON**, JavaScript Object Notation (Formato ligero de intercambio de datos)
- **cmd**, Command (Símbolo del sistema en Windows, consola de comandos)
- **npm**, Node Package Manager (Gestor de paquetes de Node.js)
- **http**, Hypertext Transfer Protocol (Protocolo de Transferencia de Hipertexto)
- **exe**, Executable (Archivo ejecutable en sistemas Windows)
- **cpu**, Central Processing Unit (Unidad Central de Procesamiento; aquí como ejemplo de un campo de medición)
- **KW**, Kilovatios (potencia instantánea)
- **KW/h**, Kilovatios por hora (forma común de referirse al consumo o producción energética por hora; lo correcto sería kWh)
- **NE**, Número de Estación (en tu contexto parece un identificador del sistema fotovoltaico)

Introducción

1. Justificación del tema seleccionado

Los países de la Unión Europea están desarrollando estrategias para afrontar el reto de la transición energética con la que "transitar" de un sistema energético basado en los combustibles fósiles (como el petróleo, el gas y el carbón) a otro basado en fuentes de energía renovables con bajas emisiones de carbono, como la solar, eólica, hidroeléctrica, y geotérmica. El Pacto Verde Europeo (Green Deal) [1] es una estrategia de la Unión Europea para transformarse en una economía sostenible y climáticamente neutra para 2050.

La energía solar fotovoltaica (FV) juega un importante papel en este proceso, siendo hoy en día la tecnología de producción eléctrica que más crece en el mundo y con menores costes de producción. Se estima que en 2025 haya instalados 655 GWp de potencia en toda Europa [2]. En 2024, más del 10% de toda la energía eléctrica en Europa se generó a partir de FV [3], y en España este valor fue del 23% ese mismo año [4].

Dentro de las tipologías de instalaciones FV, éstas se pueden dividir en 3 grandes bloques: las grandes plantas FV de varios megavatios de potencia (utility scale); las grandes instalaciones de autoconsumo para sectores comercial e industrial (commercial & industrial); y los sistemas de autoconsumo en viviendas (residential). Esta segmentación se distribuyó en un 42%, 38% y 20% respectivamente en las nuevas instalaciones desarrolladas en Europa en 2024 [5].

De una manera transversal a esa segmentación, actúan las Comunidades Energéticas (CE), definidas dentro de ordenamiento jurídico español (Ley 24/2013, de 26 de diciembre, del Sector Eléctrico) como "*entidades jurídicas basadas en la participación abierta y voluntaria, autónomas y efectivamente controladas por socios o miembros que están situados en las proximidades de los proyectos de energías renovables que sean propiedad de dichas entidades jurídicas y que estas hayan desarrollado, cuyos socios o miembros sean personas físicas, pymes o autoridades locales, incluidos los municipios y cuya finalidad primordial sea proporcionar beneficios medioambientales, económicos o sociales a sus socios o miembros o a las zonas locales donde operan, en lugar de ganancias financieras*." [6]. Tal y como expresa la Comisión Europea, "*las comunidades energéticas permiten acciones energéticas colectivas e impulsadas por los ciudadanos para apoyar la transición hacia una energía limpia, contribuyen a aumentar la aceptación pública de los proyectos de energías renovables y facilitan la atracción de inversiones privadas en la transición hacia energías limpias*".

La UE distingue entre Comunidades Ciudadanas de Energía, CCE (Directiva UE 2019 / 944, sobre normas comunes para el mercado interior de la electricidad, Art. 16) y Comunidades de Energía Renovable, CER (Directiva UE 2018 / 2001, fomento uso de energía procedente de fuentes renovables, Art. 22).

Según el último informe publicado del Observatorio de comunidades energéticas en España, la cifra total de comunidades energéticas identificadas en 2024 fue de 659

(Observatorio de Comunidades Energéticas. (2024). Informe de Indicadores 2024, [7]). Una de estas CE es CERCA (Comunidad de Energía Renovable de la Comarca de Calatayud), nacida de un proyecto del programa LIFE de la Comisión Europea (LIFE21-CET-ENERCOM-JALON/101076395, [8]) que coordina la UPM. CERCA es una CE descentralizada de ámbito regional que abarca toda la Comarca de Calatayud, en Zaragoza, un territorio de más de 2.500 m^2, que integra 67 pueblos y 20 pedanías. CERCA la integran ciudadanos, asociaciones, pymes y municipios de la Comarca, y desarrolla proyectos de energías renovables en sus pueblos, principalmente autoconsumos FV colectivos.

Este Trabajo de Fin de Grado se enmarca en la experiencia de este proyecto de CE (CERCA), en el que se ha detectado la necesidad de implementar nuevas estrategias de monitorización de la energía de cara a los usuarios finales de los proyectos de energías renovables. En concreto, se busca la integración de los datos de energía eléctrica de las instalaciones FV de generación con la energía eléctrica consumida por los consumidores asociados a una planta FV. El objetivo es que cualquier participante en un autoconsumo colectivo (en adelante prosumidor) pueda monitorizar en una sola aplicación web tanto su consumo eléctrico como la parte correspondiente de la energía generada por la planta FV de autoconsumo en la que participa de acuerdo a su coeficiente de reparto.

La tecnología de monitorización existente en el mercado está limitada a sistemas FV para autoconsumo individual, y no existen aún soluciones estandarizadas para autoconsumos colectivos, en los que existen varios “prosumidores” asociados a una o varias plantas FV.

La meta final de este trabajo es aportar una solución técnica a este problema. Para ello se propone construir una aplicación web para el monitoreo del consumo eléctrico de un punto de suministro (*Código Universal de Punto de Suministro* - CUPS) y la generación eléctrica de una planta FV, facilitando a usuarios, particulares y empresas, un seguimiento en tiempo real de la generación FV y disponer de un histórico de datos registrados. Sumado a esto, la aplicación ofrece información completa sobre el ahorro energético logrado por las instalaciones, ayudando a los usuarios a tomar decisiones informadas sobre su consumo y a optimizar el uso de la energía renovable.

El proyecto se alinea con las iniciativas de las Comunidades Energéticas, un modelo de participación ciudadana que impulsa el autoconsumo colectivo y emplear energías renovables a nivel local. ECODES menciona lo siguiente “Se trata de iniciativas de naturaleza democrática que persiguen obtener beneficios ambientales sociales y económicos por encima de financieros a la comunidad donde se desarrolla.” [9]. Esto permite que tanto los ciudadanos como las pymes y autoridades locales sean capaces de participar activamente en la generación y la gestión de la energía promoviéndose la autonomía energética y se reducen, las desigualdades que provienen de la pobreza energética.

Figura 1: Imagen que representa el mensaje que ECODES busca comunicar.

La monitorización para los prosumidores debe dar acceso a datos en tiempo real sobre el rendimiento de las instalaciones y datos de consumo. No solamente puede ayudar en la eficiencia energética si no que, también, aumenta la capacidad de decisión de los usuarios, dándoles la información que necesitan para poder manejar sus necesidades energéticas de manera más sostenible y adecuada.

Este proyecto, va más allá de una implementación tecnológica, puede integrar un componente social y de participación, que es el punto más importante en las comunidades energéticas.

De esta forma encaja perfectamente con los ideales de democratizar la energía, avanzando así hacia un sistema energético distribuido y renovable, algo que la legislación europea sobre energías renovables y comunidades energéticas defiende.

En resumen, este proyecto ayuda a cambiar el modelo energético actual, volviéndolo más sostenible, accesible, y responsable. Se emplea la tecnología como una herramienta para optimizar la gestión de recursos energéticos, y al mismo tiempo se incentiva a que la gente participe activamente en decisiones sobre el uso de su energía.

2. Problema de estudio

El problema principal en el que se ha basado el desarrollo de este proyecto, es que las comunidades energéticas no disponen de sistemas de monitorización para autoconsumo fotovoltaico colectivo. La monitorización de la generación y consumo existente actualmente en el mercado es para sistemas FV de autoconsumo individual, y no existe una solución estándar para colectivos.

En estos casos, un único sistema solar da energía a varios hogares, y la administración de la energía generada necesita ser distribuida entre todos, aumentando así las complejidades al monitorear y repartir la energía.

Para solucionar este problema se propone el desarrollo e implementación de un sistema de monitoreo IoT y una aplicación web, valiéndose de múltiples herramientas: Mosquitto, Node-Red, InfluxDB, Grafana y la API son ejemplos de las mismas. La meta de este sistema es el monitoreo del consumo y generación de energía FV, enfocándose especialmente en instalaciones FV dentro de las CE.

Este proyecto busca superar esta barrera y ofrecer una solución adaptada para las CE, permitiendo un monitoreo y una gestión eficientes de la producción y el consumo de energía en un sistema FV colectivo.

Este enfoque permite personalizar la gestión energética en las CE, adaptando los sistemas de monitorización para que sean más inclusivos y eficientes. Además, se proporciona a los usuarios una herramienta fácil de usar que les permitirá gestionar su consumo y producción de energía de manera transparente y autónoma. Esto resulta en un ahorro económico para los usuarios, así como una optimización del uso energético.

Las herramientas que propone este Trabajo de Fin de Grado son las siguientes:

- **Mosquitto**: Broker MQTT que gestiona la transmisión eficiente de datos entre los dispositivos IoT y la plataforma central.
- **Node-Red**: Facilita la transmisión y transformación de datos entre dispositivos y sistemas antes de enviarlos a la base de datos.
- **InfluxDB**: Base de datos de series temporales para almacenar datos históricos de consumo y producción, optimizando consultas rápidas y visualización de tendencias.
- **Grafana**: Plataforma para crear paneles interactivos que muestran datos en tiempo real, útil para el monitoreo del sistema

3. Objetivos

El objetivo de este TFG es contribuir al desarrollo de herramientas de monitoreo de energía eléctrica consumida y producida por prosumidores en el seno de las CE. Para ello se propone la implementación de un sistema de monitoreo IoT [10] que integre varias plataformas y herramientas como Mosquitto, Node-Red, InfluxDB, Grafana y API para recolectar, guardar, tratar y mostrar los datos de una manera clara y sencilla al usuario. Este sistema proporcionará detalles sobre consumo eléctrico y la generación de energía diaria, hora a hora, permitiendo, además, realizar previsiones de consumo y producción en base a los datos pasados.

Aparte de poder mostrar estos datos, el sistema también ofrecerá funciones extras, como calcular la fracción de energía solar de una estación FV asignada al usuario de acuerdo a su coeficiente de reparto. Esto permite adaptar su consumo a la generación solar. También una predicción de datos de consumo y producción futuros. Todo esto, se mostrará en gráficos interactivos, facilitando al usuario el entendimiento sobre cómo evoluciona su consumo y su producción energética.

Este proyecto aspira a más que el simple monitoreo del consumo eléctrico. Se busca optimizar el empleo de la energía solar fotovoltaica. Así, se promueve la sostenibilidad energética, ayudando a los usuarios a disminuir sus costos eléctricos sustancialmente.

Por ello, los objetivos específicos del proyecto son los siguientes:

- Monitoreo del consumo diario, hora a hora, en kWh de un usuario (CUPS), mostrado en una gráfica interactiva.
- Realizar un forecast de consumo de 1 día a futuro, prediciendo el consumo de energía de los usuarios y visualizándolo gráficamente.
- Monitoreo de la producción diaria, hora a hora, en kWh de una planta FV para autoconsumo colectivo, representando los datos de producción en tiempo real.
- Hacer un forecast de producción de 1 día a futuro, prediciendo la producción de energía de la instalación FV y representándola en una gráfica.
- Determinar el porcentaje de la producción FV que le pertenece al usuario según su coeficiente de reparto (%), calculando los kWh que le corresponden cada hora.
- Calcular la energía auto consumida por el usuario y mostrar los ahorros generados por el sistema FV.

Marco teórico

1. Tecnología e información ya existente

Comunidades Energéticas

Las comunidades de energías renovables permiten a la ciudadanía, municipios y pymes generar y consumir su propia energía, ofreciendo una solución más segura, sostenible y económica, a la vez que promueven la colaboración de sus miembros, el sentido de pertenencia y construyen un futuro más ecológico para todos. Este modelo promueve la autonomía energética y reduce la dependencia de grandes corporaciones, ofreciendo beneficios económicos, sociales y ambientales.

Beneficios:

- Económicos: Reducción de costes de energía para los miembros y ahorro significativo en las facturas.
- Ambientales: Uso de energías renovables que disminuye las emisiones de CO_2.
- Sociales: Fomento de la participación ciudadana y creación de empleo local.

Las comunidades energéticas operan con una estructura democrática, donde los miembros tienen voz en las decisiones, y su objetivo principal es mejorar la eficiencia energética y combatir la pobreza energética.

Diferencia entre autoconsumo colectivo y comunidades energéticas:

- Autoconsumo colectivo: Varios usuarios comparten la energía generada por un sistema FV, pero no tienen una estructura formal.
- Comunidades energéticas: Son entidades legales que gestionan no solo el autoconsumo, sino también el almacenamiento, distribución y comercialización de la energía.

Este modelo promueve un futuro más sostenible y justo mediante la participación activa en la transición energética [11].

Consumo individual y colectivo

El autoconsumo energético a partir de la instalación de placas solares FV es una solución cada vez más extendida. Con este sistema se pretende reducir la factura de la luz o bien desconectarse completamente de la red eléctrica nacional. La independencia de los productores y consumidores de energía está cada vez más cerca y cada día es más asequible económicamente.

Autoconsumo Individual

En este modelo, la energía generada por el sistema FV se destina exclusivamente al consumo de una única persona o entidad. Es ideal para personas que poseen un tejado o terreno suficiente para instalar el sistema solar, y la inversión recae completamente en

ellos. El principal beneficio es la autonomía energética, pero la inversión inicial es elevada. En este caso, los excedentes de energía se venden a la red eléctrica, pero solo la persona o entidad propietaria del sistema se beneficia directamente de la energía generada.

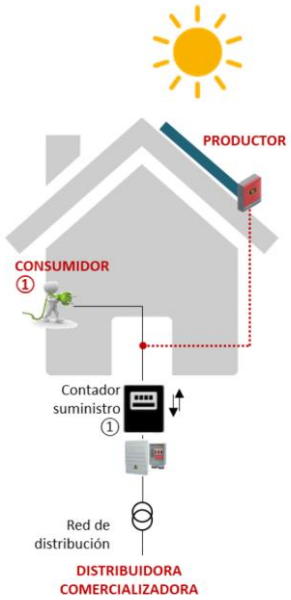


Figura 2: Esquema de un autoconsumo individual. Fuente: Guía IDAE 021: Guía Profesional de Tramitación del Autoconsumo (edición v.6) Madrid, julio de 2024. Autor: Departamento Solar y Autoconsumo del IDAE. Grupo de Trabajo de Autoconsumo de ENERAGEN. NIPO: 629-24-004-5

Autoconsumo Colectivo

En el autoconsumo colectivo, varias personas, familias o entidades comparten una instalación FV ubicada en una zona común (como el tejado de un edificio o un terreno compartido). La inversión y los costos asociados (instalación, mantenimiento, ...) se reparten entre todos los miembros de la comunidad, lo que reduce el coste inicial para el usuario. La energía generada se reparte de entre los miembros según su participación o consumo. Este modelo es ideal para comunidades de vecinos o zonas urbanas donde los usuarios tienen un consumo energético elevado [12].

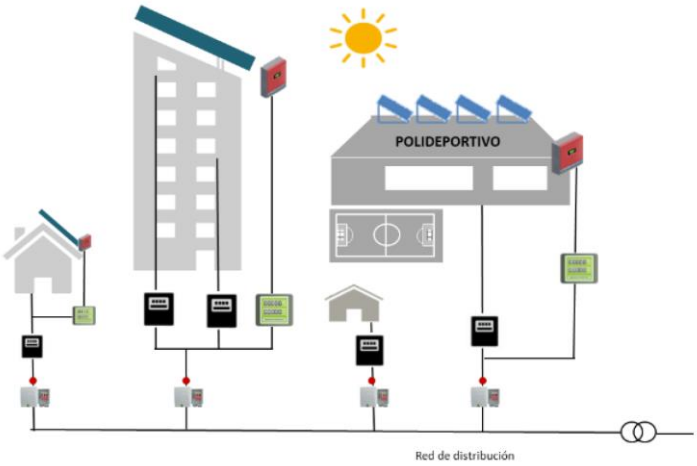


Figura 3: Esquema de un autoconsumo colectivo. Fuente: fuente: Guía IDAE 026: Guía de autoconsumo colectivo (versión v.2.1), Madrid, julio de 2024, Departamento Solar y Autoconsumo – IDAE, Instituto para la Diversificación y Ahorro de la Energía, NIPO: 629-24-003-X

Coeficiente de reparto:

El coeficiente de reparto es el valor con el que se determina cómo se distribuye la energía generada en un sistema de autoconsumo colectivo entre los miembros de una comunidad. Se calculan según factores como la participación económica y el consumo de cada usuario.

Estos coeficientes pueden ser constantes o variar, pero deben sumar siempre un total de 1. Son cruciales para asegurar una distribución justa y eficiente de la energía generada.

Pueden modificarse con el acuerdo de todos los miembros, permitiendo adaptar el reparto a cambios en el consumo [13].

Energía Solar: Conceptos Básicos sobre la Generación de Energía Fotovoltaica

La energía solar FV se basa en transformar directamente la luz solar en electricidad, con ayuda de células FV. Dichas células, son fabricadas con materiales semiconductores (el silicio, por ejemplo), en los que, al incidir la luz solar, generan electricidad. El proceso comienza con la excitación de electrones en el semiconductor, creándose así una corriente eléctrica. Esta corriente se acumula, canalizándose para dar energía a cargas eléctricas con las cuales se proporciona energía a hogares y edificios.

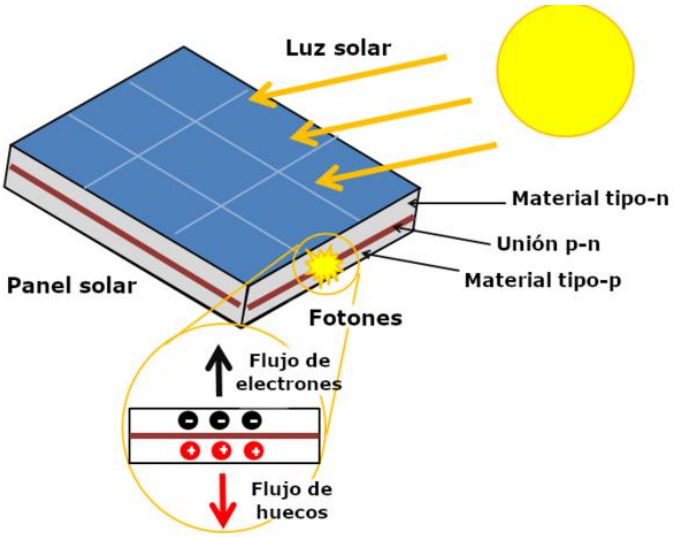


Figura 4: Se muestra como generan electricidad las placas solares.

En la eficiencia de un sistema FV influyen múltiples factores como la intensidad de la radiación solar, la temperatura, la orientación de los paneles, así como las condiciones climáticas. Todos estos son factores importantes en cuanto a la generación de electricidad. Esta varía según la hora, el día y año, lo cual requiere un seguimiento constante para optimizar su uso y distribución.

Los sistemas FV que nos ocupan son los destinados a autoconsumo, ya sea residencial como comercial o industrial. El autoconsumo FV está regulado en España por el Real Decreto 244/2019, en el que se especifican las diversas modalidades permitidas, que básicamente se dividen en autoconsumos individuales o colectivos con y sin excedentes, que pueden integrar acumulación con una batería electroquímica, normalmente de tecnología de litio.

Los sistemas se componen de:

- Un generador FV compuesto de una serie de módulos FV conectados entre sí y soportados por una estructura que les mantiene fijos al suelo o cubierta de edificio de manera estable y resistente.
- Un inversore de conexión a red que se encarga de transformar la corriente continua del generador FV en corriente alterna para inyectarla en la red eléctrica.
- Un sistema eléctrico de distribución de la energía y de protecciones.
- Un sistema de monitorización remota, generalmente desarrollada por el fabricante del inversor, que, mediante medidores de energía y una conexión a internet, ofrecen datos de generación eléctrica y de consumo eléctrico del edificio donde se ubica el sistema. La monitorización remota ofrece datos eléctricos útiles para labores de mantenimiento y permite al usuario hacer un seguimiento a la producción y a su consumo, dotándole de una herramienta para optimizar sus ahorros energéticos.
- Opcionalmente pueden integrar una batería para acumular excedentes energéticos.

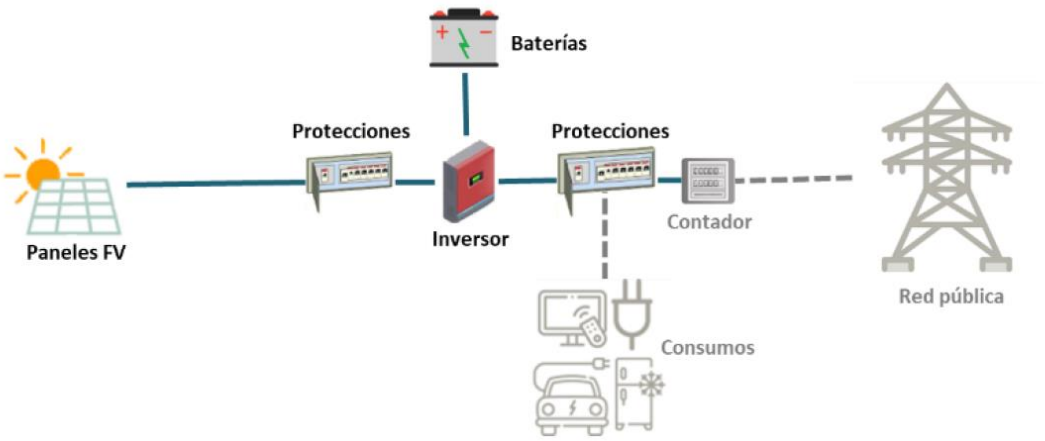


Figura 5: Esquema básico de un sistema de autoconsumo individual. (fuente: Guía IDAE 026: Guía de autoconsumo colectivo (versión v.2.1), Madrid, julio de 2024, Departamento Solar y Autoconsumo – IDAE, Instituto para la Diversificación y Ahorro de la Energía, NIPO: 629-24-003-X)

Monitoreo de Datos Eléctricos

La monitorización de autoconsumo a nivel de usuario ha evolucionado significativamente en los últimos años, gracias a plataformas avanzadas que permiten a los usuarios controlar y optimizar su consumo de energía en tiempo real. Una de las plataformas que se utilizan en España es FusionSolar, la plataforma de monitorización de la marca Huawei, que es la misma que se empleará en este TFG.

¿Qué es FusionSolar?

FusionSolar es una plataforma de Huawei cuyo uso es el de monitorizar instalaciones FV, está disponible en web y móvil. Da a los usuarios una visión detallada de su autoconsumo FV, permitiéndoles visualizar en tiempo real el rendimiento de sus paneles solares, el consumo energético y la venta de energía sobrante [14].

Características principales:

- Gráficos interactivos: FusionSolar tiene gráficas visuales que muestran el flujo de energía desde los paneles solares hacia el consumo del hogar y la red eléctrica.
- Predicción de energía: La plataforma también ofrece predicciones sobre la cantidad de energía que se generará y consumirá en los próximos días.
- Contribución al medioambiente: FusionSolar incluye un panel de sostenibilidad que muestra los beneficios medioambientales, como la reducción de CO_2 y el equivalente en árboles plantados.

Curvas de energía y su análisis

En la plataforma FusionSolar, se visualizan curvas de energía, mostrando el ritmo de producción y consumo energético a lo largo del día. Estas curvas ayudan a los usuarios a entender mejor los hábitos de consumo en sus hogares o negocios, y les permiten determinar si la energía solar, producida por sus paneles, es suficiente para satisfacer las necesidades energéticas o, al contrario, es necesario consumir de la red.

Por ejemplo, durante las horas con más sol, la curva de energía exhibirá un punto alto de producción solar, en cambio, la curva de consumo indicará si el usuario está usando toda esa energía o si está vertiendo el excedente a la red.

Funcionamiento de la monitorización

La monitorización de FusionSolar se basa en el uso de inversores inteligentes y sensores de medición instalados en el sistema FV. Los datos de estos dispositivos son enviados en tiempo real a la plataforma, donde se procesan y se muestran a los usuarios en una interfaz accesible.

1. **Instalación y sensores**: Los inversores de Huawei son capaces de recopilar datos de tensión, corriente y rendimiento de cada panel solar, transmitiendo esta información a la plataforma FusionSolar.
2. **Visualización y análisis**: La plataforma es capaz de mostrar estos datos de manera gráfica, permitiendo al usuario ver la energía generada, el consumo inmediato y el exceso de energía. Además, FusionSolar permite realizar un análisis histórico para que el usuario pueda comparar su consumo y la producción a lo largo del tiempo.
3. **Optimización**: Gracias a la información detallada que ofrece FusionSolar, los usuarios pueden optimizar su consumo de energía, maximizando el uso de la energía solar generada y reduciendo su dependencia de la red eléctrica.

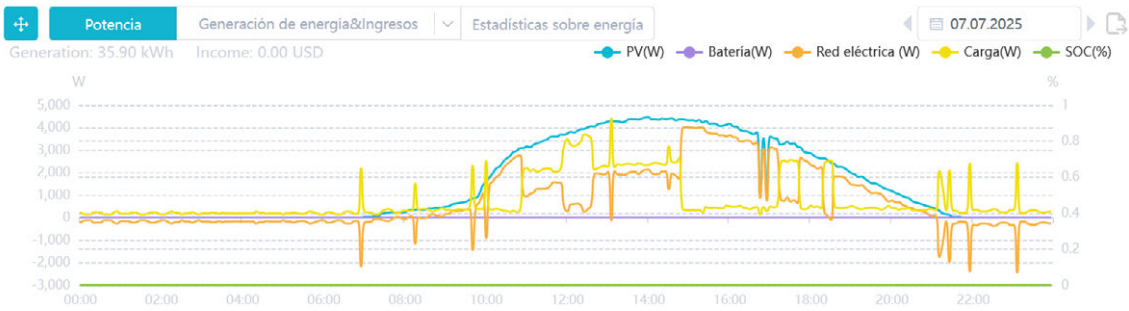


Figura 6: Vista de la interfaz de una aplicación FV para monitorizar las distintas componentes de energía a lo largo de un día de una instalación FV de autoconsumo individual.

El monitoreo de los datos eléctricos obtenidos es vital para una administración eficiente, tanto del consumo como de la energía generada. En el contexto de la energía solar, un seguimiento continuo de la energía consumida en hogares o edificios, y de la producida por los paneles, es clave. Ayuda a descubrir tendencias, optimizar el uso de energía, y a identificar fallos que pudiera haber en los sistemas FV.

En este proyecto, se usan tecnologías y herramientas avanzadas para recopilar datos en tiempo real, sobre el consumo eléctrico de los clientes y la producción de sus paneles. Este monitoreo no solo da una imagen clara del consumo actual, sino que también permite hacer análisis predictivos que mejoran el uso de la energía renovable y disminuyen los costos para los usuarios.

Herramientas de Software

En este proyecto se ha buscado usar herramientas de código abierto, con flexibilidad, facilidad de integración y un amplio soporte comunitario como Mosquitto, Node-RED, InfluxDB y Grafana.

Estas herramientas han permitido la gestión, el procesamiento, almacenamiento y visualización de los datos eficientemente y de manera económica. Se han elegido por su capacidad para cumplir los requisitos del proyecto, su rendimiento y bajo coste en comparación con otras soluciones comerciales, garantizando al mismo tiempo una implementación escalable y accesible.

Mosquitto: Broker de MQTT

Mosquitto es un broker MQTT (Message Queuing Telemetry Transport), es un protocolo de mensajería, liviano y muy eficiente que se usa en aplicaciones IoT. Con MQTT los dispositivos se comunican de forma eficaz, empleando un modelo de publicación suscripción. En este proyecto, Mosquitto ha sido usado para manejar el traspaso de datos entre los sensores y otros sistemas de monitoreo. Es ligero y confiable, esto lo hace perfecto para aplicaciones en tiempo real como la recolección de datos del consumo y producción de energía.

Figura 7: Logo de Mosquitto.

Node-Red: Plataforma de Flujo para Integrar y Procesar Datos

Node-Red, es una plataforma de flujo de trabajo basada en JavaScript que nos permite integrar y procesar datos, es 100% visual ya que los flujos se crean a través de la interfaz de Node-Red. Esta herramienta ayuda a crear flujos de datos entre diferentes sistemas y dispositivos sin la necesidad de programación compleja. Para el proyecto, Node-Red es usado para la captura de los datos de Mosquitto, procesarlos y finalmente enviarlos a InfluxDB para su almacenamiento.

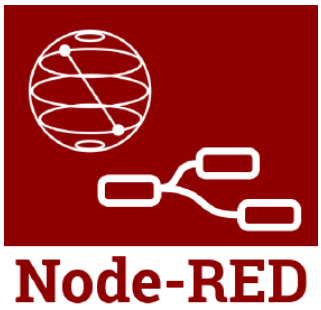


Figura 8: Logo de Node-Red.

InfluxDB: Base de Datos de Series Temporales

InfluxDB es una base de datos usada para el almacenamiento y análisis de series temporales. Está diseñada para ser capaz de manejar grandes volúmenes de datos que se inserten manualmente o a través de sensores y dispositivos IoT, lo que la convierte en la opción perfecta para almacenar los datos de consumo y producción de energía generados en este proyecto. InfluxDB permite realizar consultas rápidas sobre grandes cantidades de datos como si fuese SQL, lo cual es fundamental para el monitoreo en tiempo real y para realizar análisis de tendencias y patrones históricos.

Figura 9: Logo de InfluxDB.

Grafana: Plataforma de Visualización de Datos

Grafana es una herramienta de visualización de datos que nos permite crear paneles interactivos para la monitorización en tiempo real de los datos de nuestro proyecto. Se pueden representar gráficamente los datos de consumo y producción de energía almacenados en InfluxDB, permitiendo a los usuarios poder visualizar sus mediciones de una forma clara y sencilla.

Figura 10: Logo de Grafana.

Predicción de Consumo y Producción

La predicción del consumo y la producción energética resulta fundamental para optimizar los sistemas FV. Usando modelos predictivos construidos sobre datos históricos, se puede estimar lo que ocurrirá con la producción solar y el consumo de electricidad. Se usan algoritmos de forecast para analizar los datos históricos de consumo y producción, de esta forma prediciendo los datos del siguiente día con precisión según los test realizados.

Algunos de los algoritmos más comunes para la predicción de energía son:

- Modelos de series temporales: Como el ARIMA (AutoRegressive Integrated Moving Average), que se utiliza para modelar y predecir datos que dependen del tiempo, como el consumo de energía.
- Redes neuronales artificiales: Modelos como las Redes Neuronales Recurrentes (RNN) o LSTM (Long Short-Term Memory), que son muy eficaces para trabajar con secuencias temporales complejas y pueden mejorar la precisión de las predicciones a largo plazo.
- Árboles de decisión y Random Forest: Estos modelos se utilizan para hacer predicciones basadas en patrones complejos y no lineales en los datos históricos de consumo y producción.
- Modelos de regresión: Como la regresión lineal o regresión múltiple, que pueden ser usados para predecir el consumo de energía según variables como la hora del día, la temperatura, entre otros factores.

Las predicciones creadas pueden ayudar a los usuarios a saber con adelanto como va a ser su consumo energético para el día siguiente, optimizando de esta forma el uso de la energía solar generada y reduciendo la dependencia de la red eléctrica. Esto no solo mejora la eficiencia energética, sino que también puede facilitar la planificación y la toma de decisiones con información relevante sobre el uso y compra de energía.

Herramientas ya existentes: aplicaciones de fabricantes de inversores, software disponible, Datadis, API, etc.

Los sistemas actuales, como los que ofrecen los fabricantes de los inversores, proporcionan algunas funciones de monitoreo, pero están diseñadas para sistemas FV de autoconsumo individual como se ha mencionado anteriormente. Las comunidades energéticas usan sistemas FV de autoconsumo colectivo para que se distribuya la energía a varios prosumidores, eso hace que se necesite un enfoque distinto para distribuir la

energía y gestionar los datos de producción obtenidos. Para ello, el acceso usando API a los inversores FV se vuelve necesario. Esto permite la extracción de los datos de producción, facilitando su posterior distribución a los diferentes miembros de la comunidad y permitiendo un monitoreo colectivo.

Al utilizar API, los sistemas de autoconsumo colectivo pueden integrar estos datos con herramientas como Node-RED, InfluxDB y Grafana, creando un sistema de gestión centralizada que optimiza tanto la distribución de energía como el análisis de los datos generados, permitiendo a los prosumidores tomar decisiones sobre su consumo y ahorro energético.

Además, usamos Datadis, una plataforma gratuita impulsada por las empresas distribuidoras de electricidad en España, que permite a los consumidores acceder a sus datos de consumo eléctrico. Esta plataforma agrupa datos de diferentes distribuidoras, permitiendo acceder a la información de consumo de todos los suministros eléctricos en un único lugar. Proporciona información detallada sobre el consumo eléctrico por horas, días, semanas o meses de cualquier usuario o CUPS, previa obtención de autorización por parte del titular de dicho consumo. Se puede acceder a estos datos de consumo eléctrico, en formato JSON, del día anterior a través de una API. En este proyecto se usarán APIs, permitiendo agregar fuentes de datos variadas, facilitando la conexión entre los sistemas y la plataforma de muestra de datos.

El proyecto está basado en tecnologías ya existentes y comprobadas, se han adaptado a las necesidades especiales de las comunidades energéticas, con el objetivo de lograr optimizar la energía fotovoltaica, mejorar la eficiencia energética y reducir los costos de consumo eléctrico de los usuarios.

Desarrollo del proyecto

1. Análisis de requisitos y especificaciones

Para la construcción del sistema de monitoreo, se necesitan datos reales de la producción y el consumo energético que provengan tanto de los sistemas FV como de los consumidores respectivamente. Estos datos tienen que ser precisos, deben poder ser obtenidos en el momento que se requieran, y ser accesibles para su análisis y visualización en todo momento. Además, es muy importante que el sistema sea adaptable, versátil y sencillo de usar, teniendo una interfaz amigable para que los usuarios sean capaces de interpretar los datos sin problema. Otro punto importante del proyecto es el uso de herramientas gratuitas para ser capaz de realizar las siguientes funciones:

- **Datos de producción fotovoltaica:** Medidos en intervalos de tiempo definidos, idealmente en KWh.
- **Datos de consumo de energía:** También en intervalos de tiempo específicos y representados en KWh.

- **Interfaz de usuario:** La aplicación debe proporcionar gráficos interactivos que visualicen el consumo y la producción de energía de forma clara.
- **Función de forecast (predicción):** El sistema debe predecir el consumo y la producción de energía en el próximo día, basándose en los datos históricos.

El sistema debe ser escalable, siendo capaz de manejar grandes volúmenes de datos, considerando tanto la expansión horizontal como vertical. La capacidad de procesar y visualizar datos en tiempo real es esencial, con un retardo mínimo y actualizaciones rápidas, garantizando que la información de consumo y producción se refleje al momento. Para el almacenamiento, se empleará una base de datos optimizada para series temporales, como InfluxDB, capaz de gestionar grandes volúmenes de registros de manera eficiente, permitiendo consultas rápidas y efectivas.

Requisitos esenciales:

- **Interfaz amigable y accesible**:
    - La interfaz de usuario de la aplicación web debe ser intuitiva y fácil de usar para que los usuarios puedan navegar por ella sin dificultad.
    - El diseño debe ser limpio y claro, con gráficos que faciliten la visualización de los datos.
- **Tiempo de respuesta rápido**:
    - La aplicación web debe cargar y procesar los datos de manera rápida, asegurando que los usuarios no tengan que esperar largos períodos de tiempo para ver los resultados.
    - El sistema debe actualizarse en tiempo real o en intervalos cortos sin demoras notables.
- **Precisión en el monitoreo**:
    - Los datos de consumo y producción deben ser precisos y consistentes, sin errores de medición o de transmisión.
    - El sistema debe ser capaz de manejar grandes volúmenes de datos de manera eficiente, sin comprometer la precisión.
- **Capacidad para manejar datos históricos**:
    - El sistema debe permitir la consulta de datos históricos para facilitar el análisis de patrones de consumo y producción a lo largo del tiempo.
    - Los usuarios deben poder seleccionar rangos de fechas para ver los datos acumulados o promedios de diferentes períodos (diarios, semanales, mensuales, etc.).
- **Escalabilidad y adaptabilidad**:
    - El sistema debe ser escalable para poder gestionar aumento de usuarios o dispositivos IoT sin afectar el rendimiento.

- Debe poder adaptarse a futuras expansiones de la infraestructura energética (más usuarios, más paneles solares, etc.).

- **Integración con dispositivos IoT**:
  - El sistema debe ser capaz de recibir y procesar datos de diferentes dispositivos IoT que miden el consumo y la producción de energía, como los inversores FV.
  - La comunicación entre dispositivos debe ser segura y confiable, utilizando protocolos como MQTT para la transmisión de datos.
- **Facilidad de integración con APIs externas**:
  - El sistema debe permitir la integración con APIs externas, como las de Huawei para obtener datos de los inversores FV o Datadis para

2. Diseño y desarrollo

Descripción del Sistema de Monitoreo Propuesto

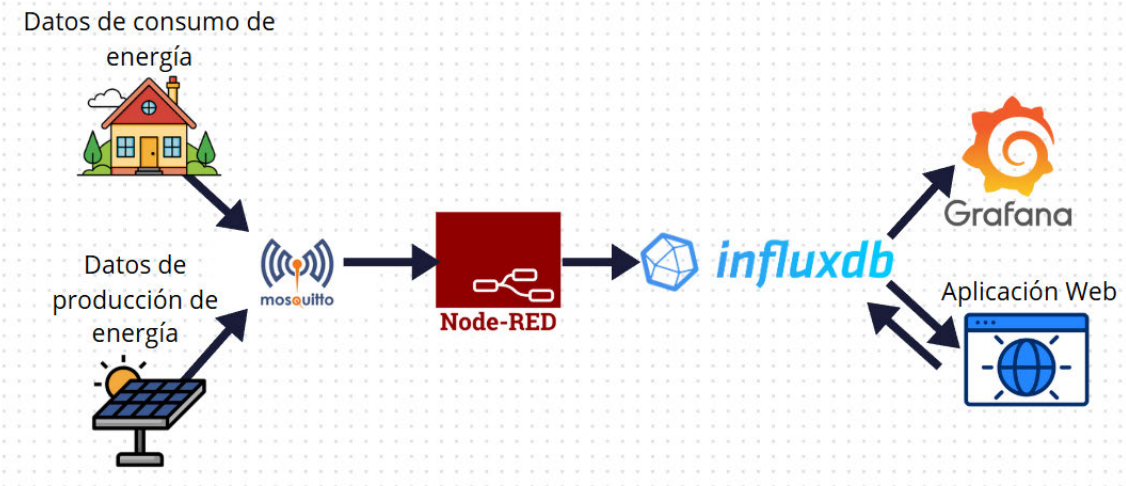


Figura 11: Flujo que recorren los datos desde su envío hasta su visualización

El sistema de monitoreo (Figura 11) propuesto se basa en la integración de diversas herramientas y plataformas para recibir, almacenar y visualizar los datos relacionados con el consumo de energía y la producción solar. A continuación, se describen las funcionalidades y el flujo de trabajo del sistema:

El flujo inicia con Mosquitto, una herramienta que funciona con una metodología de publicación y subscripción, por tanto, se ha creado un tema para el envío de datos de consumo y otro para el envío de datos de producción. Usando códigos Python se envían solicitudes API tanto a Huawei para obtener los datos de producción como a Datadis para los de consumo. Cada uno de estas solicitudes se realiza usando un tema distinto en mosquitto, de esta forma al recibirlos podemos clasificarlos según el tipo de dato.

Se conecta Mosquitto con Node-Red a través de nodos al crear el flujo en la interfaz gráfica que posee, en el nodo correspondiente se ha seleccionado el tema al que se ha subscrito. Se usa otro nodo function, conectado al nodo anterior que recibe los mensajes de

Mosquitto, este nodo organiza y estructura la información recibida usando 3 variables principales, hora, fecha y un valor cuyo nombre varía dependiendo del tipo de dato guardado. Una vez se ha estructurado la información se envía a InfluxDB, esta herramienta es la encargada del almacenamiento de los datos en una base de datos temporal, una vez guardados los datos es posible realizar consultas sobre ellos para obtener los datos deseados.

```
payload: {
    consumptionKWh: consumo,
    cups: d.cups,
    hora,
    fecha
}
```

Figura 12: Estructuración de los datos para su posterior envío a InfluxDB.

Para recibir la información estructurada en InfluxDB se usa otro nodo en Node-Red, el cual se ha conectado al nodo function, este último nodo es el encargado transmitir los datos recibidos por Mosquitto y editados por function a nuestra base de datos en InfluxDB.

Por último, todos los datos en nuestra base de datos son accesibles desde Grafana y la aplicación web. Grafana es una herramienta que está especializada en la visualización de datos dentro de bases de datos. Por otra parte, en la aplicación web también se muestran los datos en gráficos, pero de manera más simplificada.

Metodología a seguir:

Se busca integrar las herramientas para montar un flujo de datos entre ellas. Así, los usuarios pueden monitorear cuánta energía consumen y cuanta producen. Se ha diseñado un sistema para predecir datos futuros, y se han creado gráficos y tablas para que los datos sean fáciles de ver y entender.

El flujo creado se puede ver en la (Figura 11).

Los datos provienen de hogares y sistemas FV, son enviados a través de Mosquitto hasta Node-Red, ahí son procesados y enviados a InfluxDB para su guardado. Desde la base de datos se puede consultar a través de gráficos en Grafana los datos correspondientes. La aplicación web permite mostrar datos de InfluxDB, así como hacer solicitudes de datos.

Descripción de la Función de Forecast

La función de forecast tiene como objetivo predecir el consumo y la producción de energía del próximo día, basándose en los datos históricos obtenidos del sistema. Para ello, se emplean modelos de predicción que analizan las series temporales de consumo y producción para calcular estimaciones de estos datos para el futuro.

- Modelo de Predicción. El modelo de predicción utiliza datos pasados sobre el consumo y la producción para realizar una estimación de los valores futuros. Se emplean técnicas estadísticas y algoritmos de predicción de series temporales para generar estos pronósticos.

- Visualización de Predicciones.
  Las predicciones generadas por el modelo se visualizan en gráficos interactivos, permitiendo a los usuarios anticipar su consumo y la producción de energía solar. Los resultados de la predicción se pueden mostrar junto a los datos reales para facilitar la comparación.

Para desarrollar la herramienta de monitorización se han utilizado datos reales de una planta FV instalada en el seno de la mencionada comunidad energética CERCA, en la Comarca de Calatayud, y el consumo de un CUPS asociado a dicha instalación.

La planta FV integra un inversor de la marca Huawei [15] que dispone de la herramienta de monitorización FusionSolar y de la API que permite a los usuarios acceder y gestionar datos de sus sistemas solares de forma remota. Para activar el acceso a la API se requieren permisos de acceso.

Los datos de consumo eléctrico pertenecen a una vivienda y el acceso a estos a través de la plataforma Datadis nos da información sobre el consumo horario de energía (kWh) del contrato de suministro, con una antigüedad máxima de unos 2 años y mínima de la medianoche anterior. Para poder acceder a la información de un contrato, es necesaria la autorización del titular.

3. Instalación, uso e implementación de las herramientas:

Herramientas usadas

Se han usado una serie de herramientas y plataformas especializadas en el manejo de datos de consumo y producción energética. Cada herramienta ha sido seleccionada en función de sus características y capacidades para cumplir las necesidades del sistema de monitoreo fotovoltaico:

- **Mosquitto (Broker MQTT):** Esta herramienta se ha utilizado como el componente central para gestionar la comunicación entre los dispositivos IoT que miden el consumo y la producción energética. Su protocolo de mensajería ligero (MQTT) es idóneo para aplicaciones de monitoreo en tiempo real, garantizando la eficiencia en la transmisión de datos.

- **Node-Red (Plataforma de flujo de datos):** Con Node-Red, se ha creado un flujo de trabajo visual para procesar los datos recibidos de Mosquitto. Node-Red facilita y mejora la integración de los distintos componentes del sistema, permitiendo el manejo y transformación de datos antes de enviarlos a InfluxDB para su almacenamiento.

- **InfluxDB (Base de datos de series temporales):** InfluxDB ha sido usada para almacenar los datos generados en tiempo real. Su capacidad para gestionar grandes volúmenes de datos de series temporales es clave para el monitoreo constante del consumo y la producción de energía a lo largo del tiempo.

- **Grafana (Visualización de datos):** Con Grafana, se ha creado un panel de control interactivo donde los usuarios pueden visualizar su consumo y producción energética en tiempo real. Esta herramienta permite la creación de gráficos

dinámicos y personalizables, lo que facilita la toma de decisiones informadas sobre el uso de energía.

- **API:** Para la obtención de los datos de producción desde los inversores fotovoltaicos de Huawei, y de consumo desde Datadis, se usan API. Estas API han permitido la extracción en tiempo real directamente desde los sistemas, estos datos se publican en Mosquitto y se procesan posteriormente.

Implementación

La implementación del sistema ha seguido un enfoque escalable y modular. El primer paso ha sido configurar Mosquitto para recibir datos de sensores de consumo y producción a través de las plataformas de Huawei y de Datadis. Se han diseñado los flujos de trabajo en Node-Red para procesar los datos y enviarlos a InfluxDB, donde se almacenan en intervalos regulares para facilitar el análisis temporal. Finalmente, Grafana se ha configurado para leer los datos de InfluxDB y crear visualizaciones que permitan a los usuarios observar su consumo energético en tiempo real.

La función de forecasting también se ha implementado utilizando algoritmos predictivos basados en los datos históricos de consumo y producción. Estos modelos permiten calcular predicciones de consumo y producción futura, que se visualizan junto con los datos reales.

A continuación, se describe paso a paso la instalación, configuración y programación de cada una de las herramientas que intervienen en el sistema de monitoring.

Mosquitto

Instalación:

Para la instalación **Mosquitto**, primero se debe descargar desde su página oficial [16] y ejecutar el archivo correspondiente desde la terminal del sistema operativo (cmd en Windows) [17].

Se ejecuta el archivo desde el cmd (Figura 13):

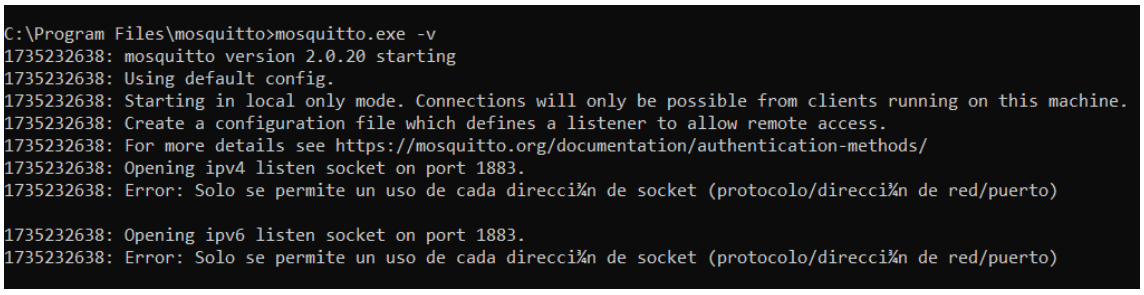


Figura 13: Ejecución de Mosquitto desde la cmd de Windows.

Prueba:

Una vez instalado, se realiza una prueba básica para comprobar su funcionamiento. Este comando permite suscribirse al **tema test**, lo que permitirá escuchar los mensajes publicados en ese tema [18]:

- Suscripción al tema: **mosquitto_sub -h localhost -t test**

Se publica un mensaje simple ("Hola") en el tema test para comprobar que la transmisión de datos entre **Mosquitto** y otros sistemas funciona correctamente (Figura 14):

- Publicar mensaje: **mosquitto_pub -h localhost -t test -m "Hola"**

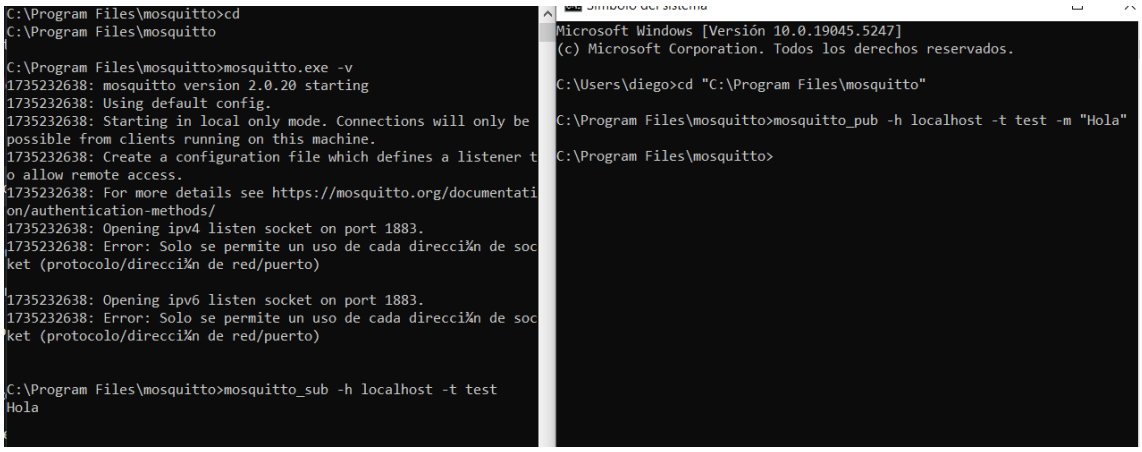


Figura 14: Prueba de subscripción a un tema y envió de mensaje a este.

Node-Red

Instalación:

Node-Red es una plataforma que facilita la creación de flujos de datos visuales. Para su instalación, se ejecuta el siguiente comando en la terminal (Figura 15):

- Comando para instalar en cmd: **npm install -g --unsafe-perm node-red**

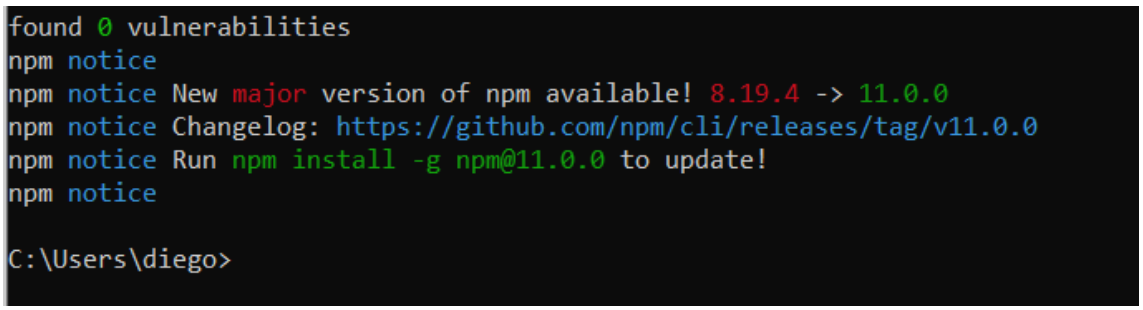


Figura 15: Instalación de Node-Red.

En la página web de Node-Red [19] se descarga la versión que se prefiera de Node-Red.

Una vez descargado se termina de instalar usando la Windows PowerShell.

Para la ejecución de **Node-Red** se usa el siguiente comando [20] (Figura 16):

- Comando para ejecutar Node-Red: **node-red**

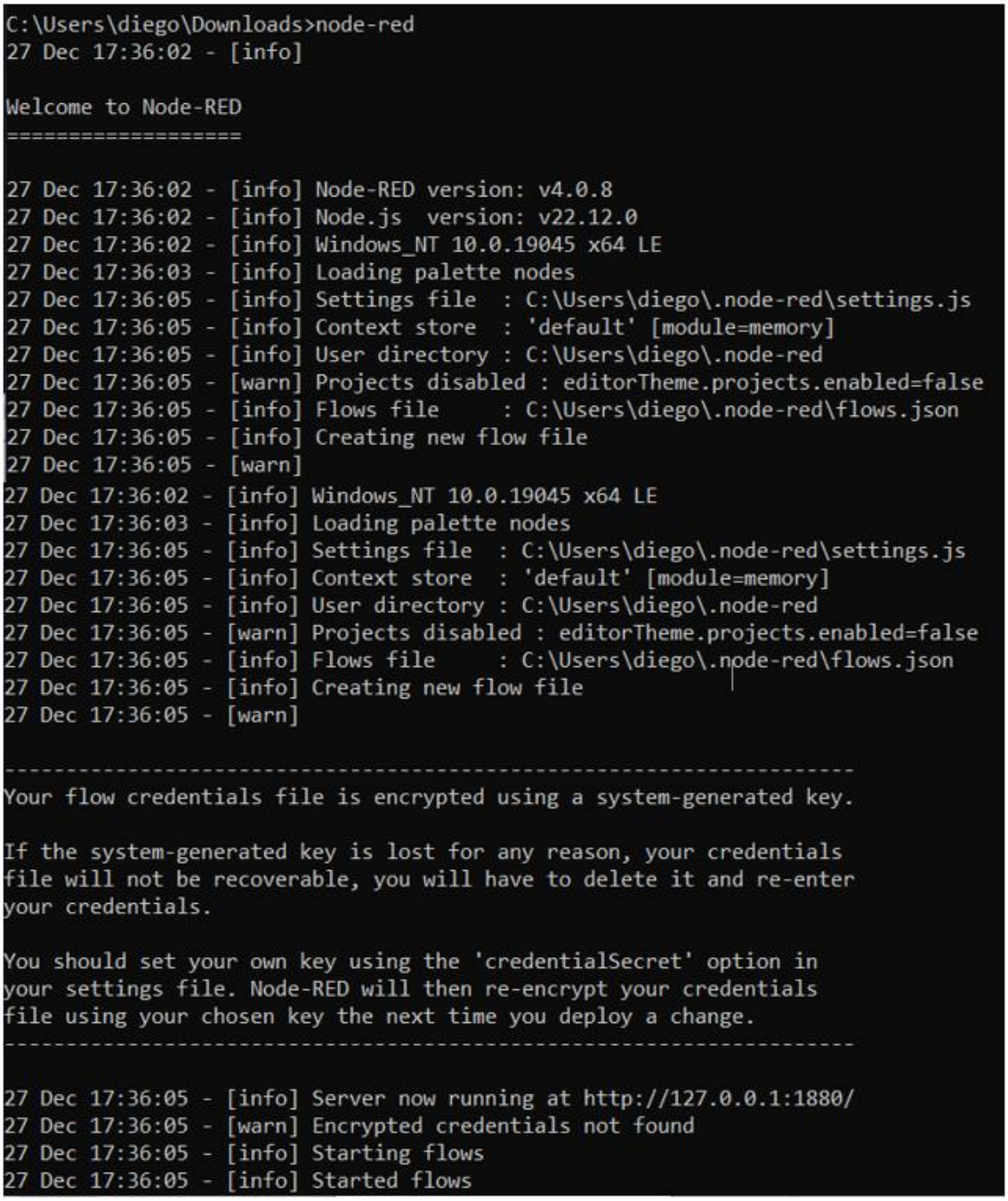


Figura 16: Ejecución de Node-Red a través del cmd de Windows.

Uso de Node-Red:

Node-Red una vez ejecutado en el cmd, devuelve un **http://127.0.0.1:1880/** el cual se coloca en el navegador web.

- http recibido de Node-Red: **http://127.0.0.1:1880/**

Figura 17: http://127.0.0.1:1880/ en el buscador.

Una vez iniciado nuestro servidor local de Node-Red se puede observar por pantalla una pizarra en la que se crearan flujos usando nodos para ello (Figura 18).

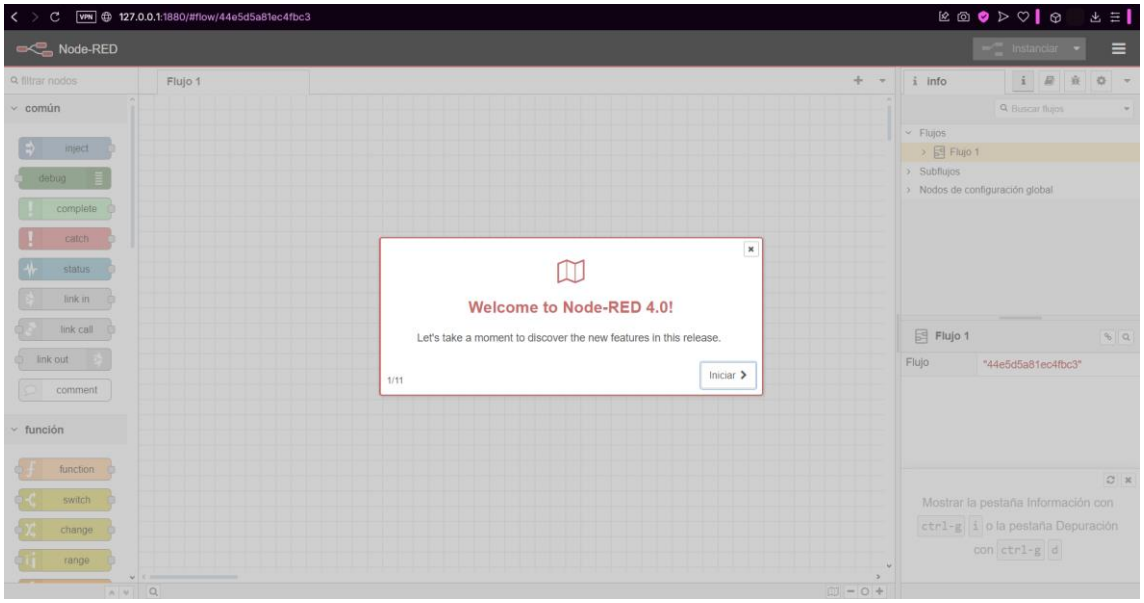


Figura 18: Página de inicio al abrir Node-Red en el explorador.

Para el iniciar el proyecto, se ha añadido un nodo de MQTT para recibir los datos que se envían desde Mosquitto [21].

La configuración es la siguiente [22], [23], [24]:

1. **Configuración del Nodo MQTT** (Figura 19)**:**
    - El tema debe ser el que se haya creado en Mosquitto, en este caso test.
    - Este nodo se conecta a un nodo de debug para comprobar que los mensajes publicados se reciben correctamente.

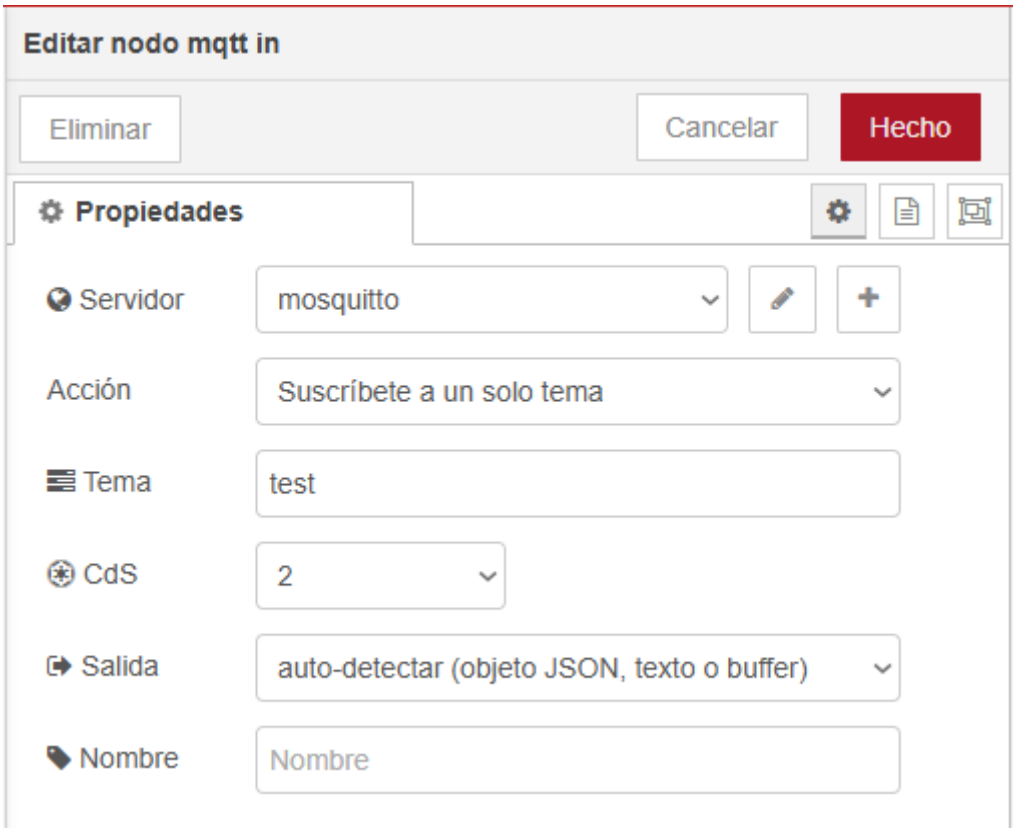


Figura 19: Contenido del nodo mqtt.

Este nodo a continuación se ha unido a un nodo debug. El objetivo de este nodo es obtener el mensaje del nodo anterior e imprimirlo en el apartado de depuración.

En la siguiente imagen se puede ver cómo se ha enviado un mensaje a el tema **test.**

- Comando para enviar el mensaje: **mosquitto_pub -h localhost -t test -m "Hola desde MQTT"**

```
C:\Program Files\mosquitto>mosquitto_pub -h localhost -t test -m "Hola des
de MQTT"
```

Figura 20: Comando para enviar mensaje en Mosquitto.

A continuación (Figura 21), se puede ver el mensaje que se ha recibido en el nodo mqtt y que, una vez es enviado al nodo debug, se muestra por pantalla el siguiente mensaje: **"Hola desde MQTT"**

Figura 21: Prueba de recepción de mensaje en Mosquitto con nodo debug.

InfluxDB

Instalación:

Para instalar InfluxDB [25], debemos seleccionar la versión adecuada desde su sitio web oficial. Ahí mismo saldrá un link el cual al ejecutarlo en la Powershell descargará un zip de InfluxDB.

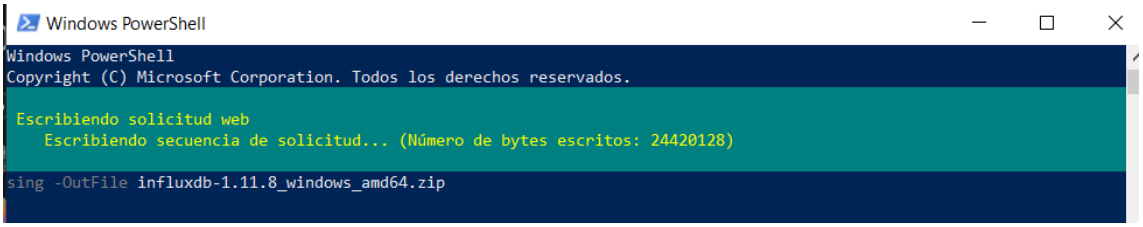


Figura 22: Descarga de los archivos necesarios para InfluxDB.

El zip de InfluxDB contiene los siguientes elementos (Figura 23), nos interesan 2 principalmente:

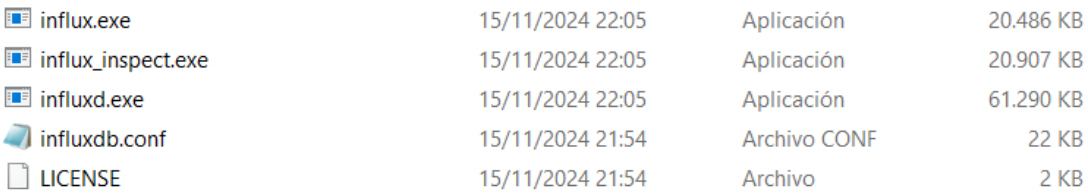


Figura 23: Archivos descargados de InfluxDB.

- Influxd.exe: Al ejecutar este exe en la cmd se iniciará el servidor con la base de datos local (Figura 24).

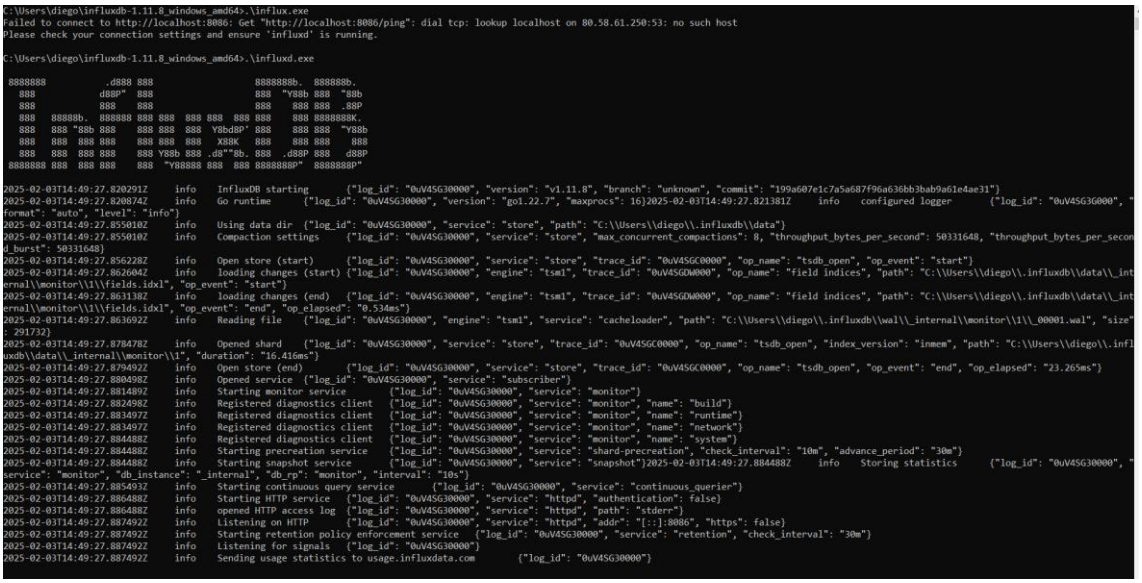


Figura 24: Ejecución de influxd.exe.

- Influx.exe: Al ejecutar este archivo en la cmd ya habiendo usado anteriormente el .exe anterior, se abrirá la base de datos en nuestra cmd (Figura 25).

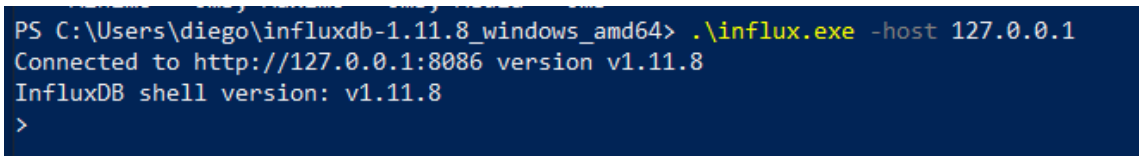


Figura 25: Ejecución de influx.exe.

El siguiente paso es crear una base de datos para guardar todas las mediciones futuras:

- Comando para crear base de datos: **create database testdb**

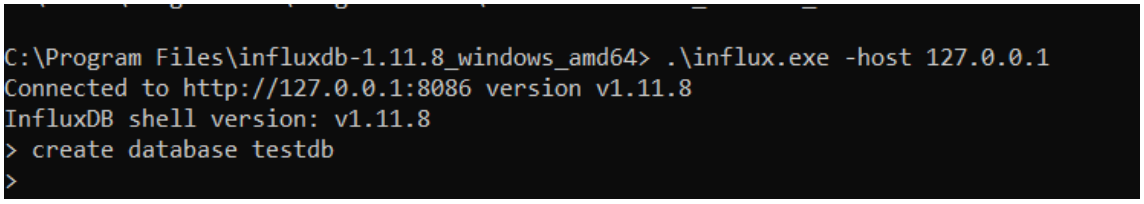


Figura 26: Creación de una base de datos de prueba.

Para entrar dentro de la base de datos creada se usa el siguiente comando:

- Comando para seleccionar una base de datos creada: **use testdb**

Una vez dentro de esta, insertamos datos de la siguiente forma:

- Ejemplo de inserción de datos: **insert cpu_load, host=server01 value=0.64**

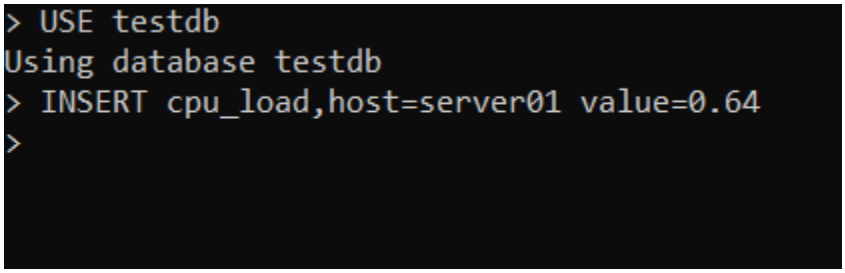


Figura 27: Inserción de datos dentro de la base de datos.

Comandos principales de InfluxDB [26]:

Mostrar todas las bases de datos creadas: Este comando ha sido usado para ver todas las bases de datos que se han creado en nuestro servidor local.

Show databases

Entrar en una base de datos: Con este comando podemos ingresar a una base de datos ya creada con anterioridad.

Use textdb

Ver todas las mediciones: Usando este comando se observan todas las mediciones existentes dentro de una base de datos, en nuestro caso serían las siguientes: datosPruebas, datosPruebas1, datosPruebas2, datosPruebas3, datosPruebas4, datosPruebasF, datosPruebasF1.

SHOW MEASUREMENTS

Ver los campos (fields) de esa medición: Este comando nos permite visualizar los campos que hay dentro de una medición, por ejemplo: fecha, hora, valor.

SHOW FIELD KEYS FROM "datosPruebas"

Ver los valores guardados: Con este comando podemos ver todos los valores que han sido guardados en un measurement.

SELECT * FROM "datosPruebas"

Filtrar por tiempo: Este comando nos permite mostrar todos los datos dentro de un measurement, pero al mismo tiempo filtrarlos según el tiempo.

SELECT * FROM "datosPruebas" WHERE time > now() - 1h

Eliminar todos los datos de una medición: Con este comando se pueden borrar todos los datos de una medición, se ha usado en varias ocasiones para borrar datos de prueba en las mediciones.

DELETE FROM "datosPruebas"

Eliminar los datos de una medición según un field: Gracias a este comando se ha sido capaz de eliminar una serie de datos de la base de datos en base a un campo en la medición, por ejemplo, según el tiempo.

DELETE FROM "datosPruebas" WHERE “time”=xxxxxxxxxx

Grafana:

Grafana ha sido usada principalmente para las pruebas, ya que más a delante se ha preferido usar gráficas y tablas integradas en la app web.

Para instalar Grafana [27] [28], se descargan los archivos con la versión adecuada para nuestro sistema operativo desde la página web.

Una vez descargado se ejecuta el “Grafana.exe server” desde el cmd, y cuando termine de ejecutar, se abre el navegador con “localhost:3000”. Esto redirige a un login donde la primera vez la contraseña y usuario es admin y admin (Figura 28).

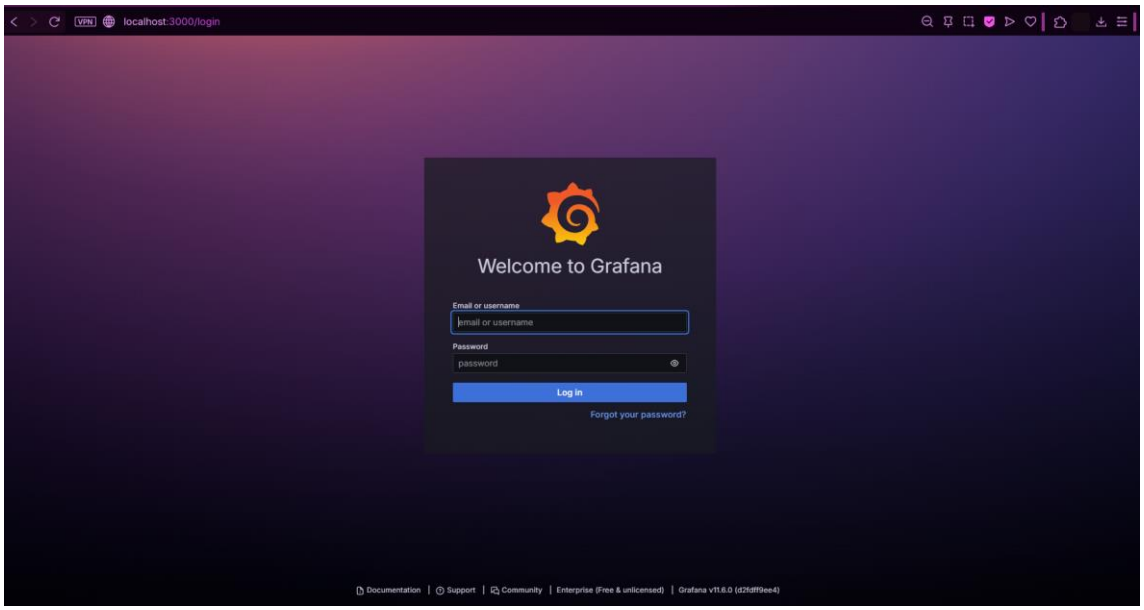


Figura 28: Página de inicio al abrir Grafana en el explorador.

Hay que añadir la base de datos que se ha creado en InfluxDB [29] [30], para la prueba se ha creado una que se llama “prueba”:

En esta primera imagen (Figura 29), se selecciona el nombre con el que queremos guardar la base de datos y la url en la que está alojado el servidor localhost de la base de datos.

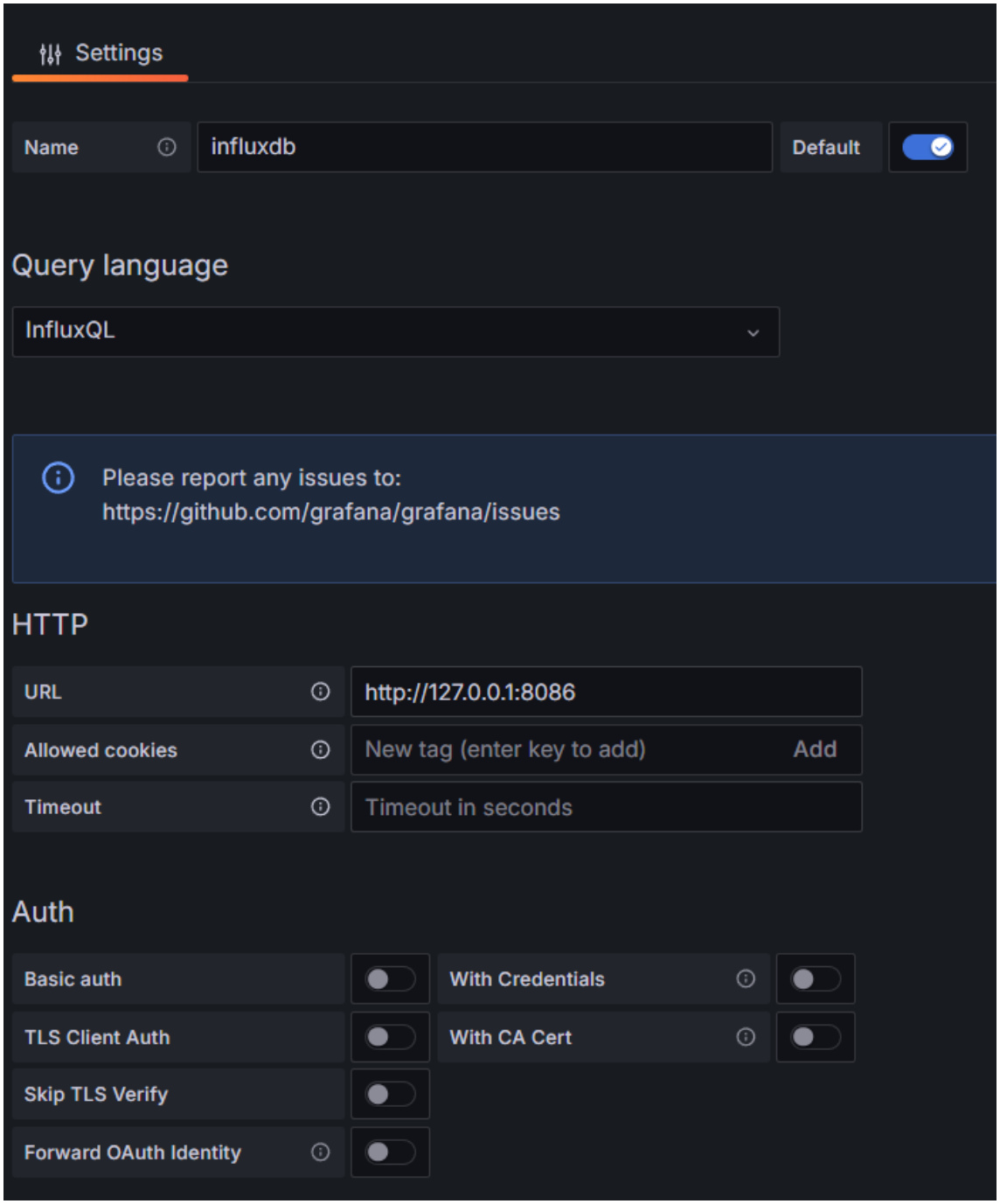


Figura 29: Contenido a insertar para añadir una fuente de datos. (1)

En esta segunda imagen (Figura 30) se puede ver cómo se guarda la información de nuestra base de datos.

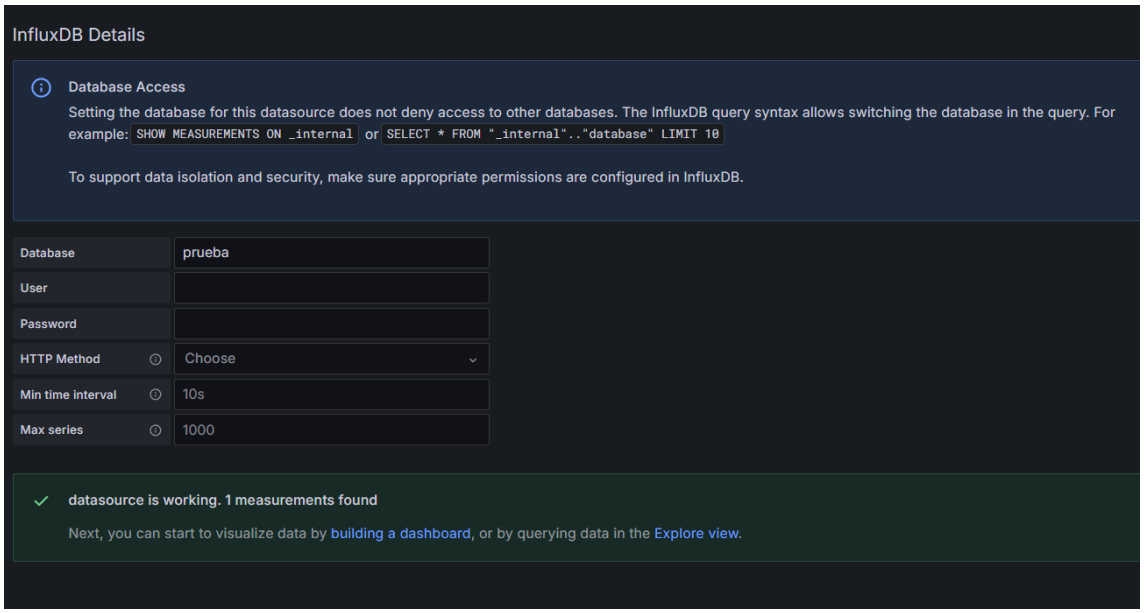


Figura 30: Contenido a insertar para añadir una fuente de datos. (2)

Después de configurar la fuente de datos, se crea un dashboard para representar gráficamente los datos de consumo y producción de energía. Utilizando las capacidades de Grafana, se cran paneles interactivos para visualizar los datos en tiempo real y realizar análisis más profundos.

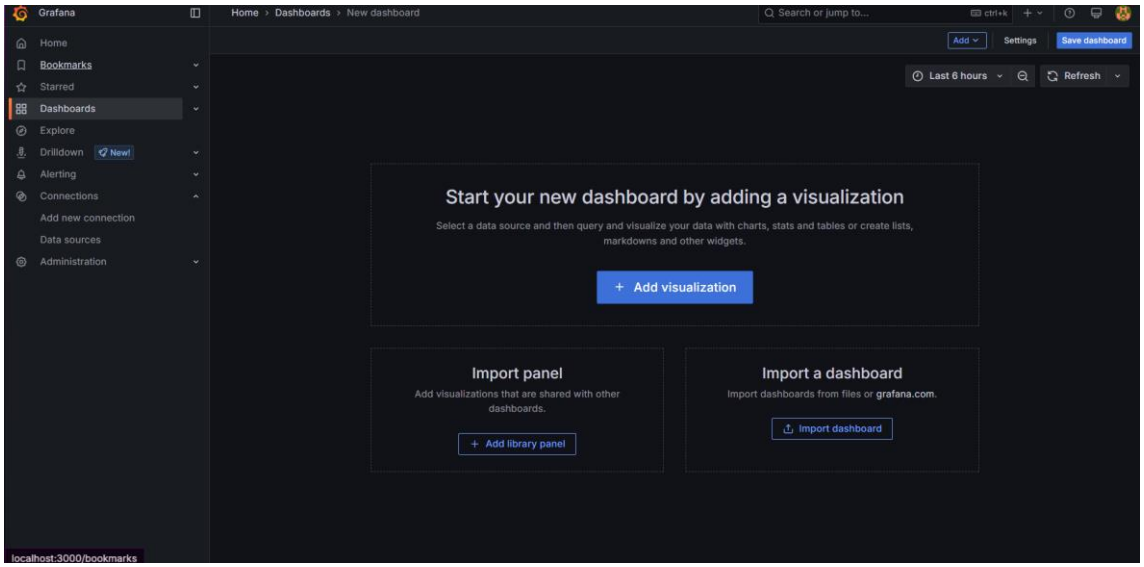


Figura 31: Página de inserción de visualizaciones dentro de Grafana.

Para ello, se añade una visualización. Dentro de la misma, se escoge la base de datos de la que se quiera sacar la información. Se selecciona el measurement que queramos mostrar en el grafo e indicamos el tipo de grafo que queremos usar (Barras, rosco, ...).

Figura 32: Ejemplo de creación de una visualización.

Una vez que se hayan creado varias visualizaciones, se puede ver los distintos datos que queremos mostrar (Figura 33):

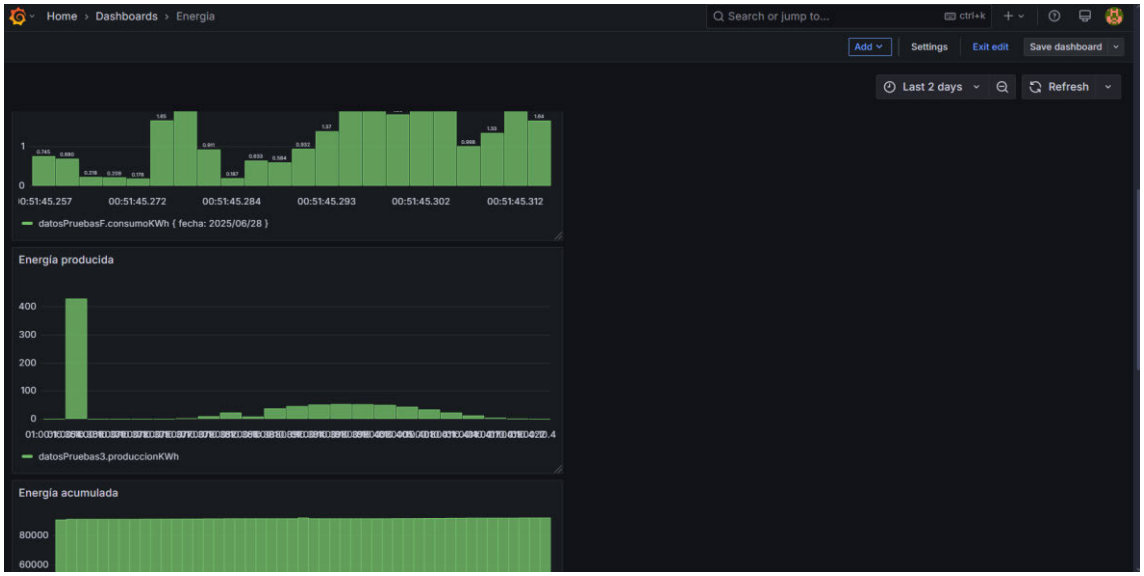


Figura 33: Vista del dashboard con varias visualizaciones.

Datadis:

Figura 34: Logo de Datadis.

Datadis permite consultar los datos de consumo propios o a los que se tenga acceso del día anterior.

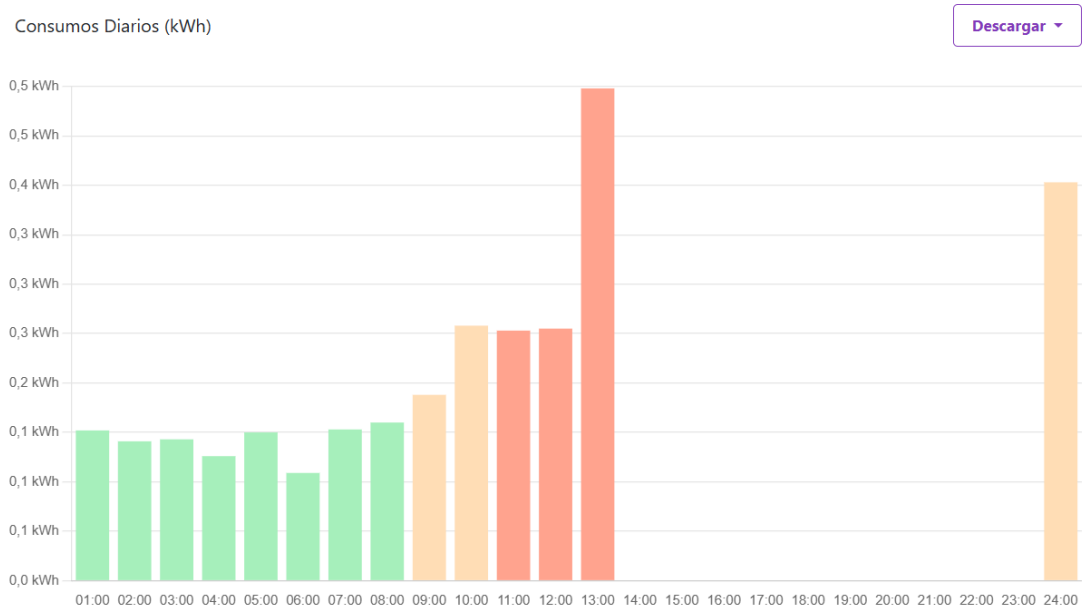


Figura 35: Ejemplo de consumo de un CUPS perteneciente a una vivienda obtenido de Datadis.

Datadis ofrece una API que permite consultar datos de consumo. Utilizando Postman, realizamos una solicitud GET para obtener los datos de consumo.

Obtener un token desechable:

Primero se debe obtener un token desechable que se podrá usar una vez al día, pera ello se insertará el DNI y la contraseña asociada a la cuenta.

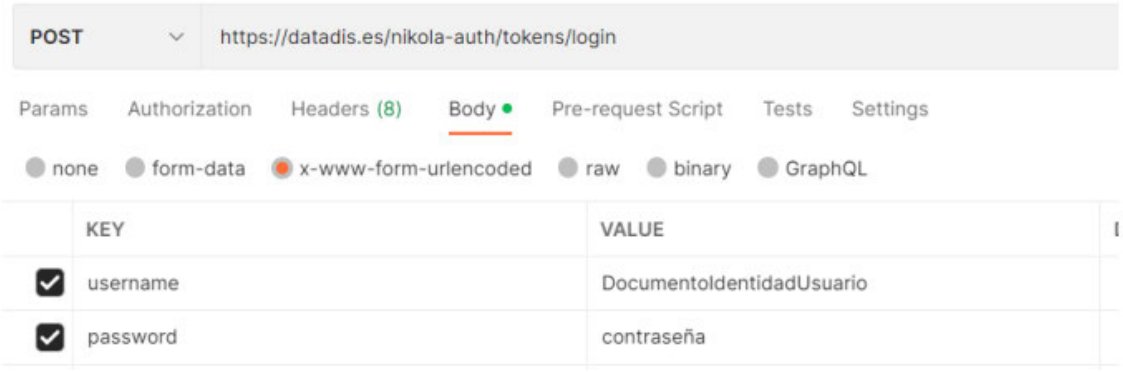


Figura 36: Obtención del token desechable en Postman.

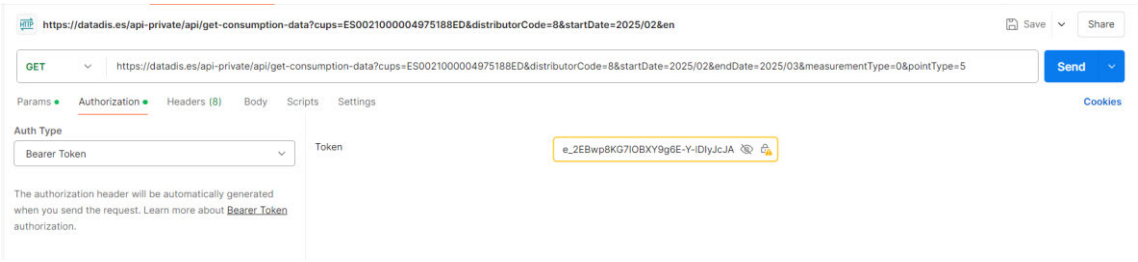


Figura 37: Guardado del token desechable en Postman.

Buscar los datos de consumo:

Una vez obtenido el token se usará como bearer token para validar al usar el API para la obtención de los datos de consumo.

API para la obtención de datos: https://Datadis.es/api-private/api/get-consumption-data

Esta API no está completa aún, hay que añadir una serie de parámetros para su correcto funcionamiento.

Parámetros necesarios para el API [31]:

cups string required
Los CUPS de los que querremos saber los datos de consumo
distributorCode string required
Código del distribuidor, que se obtiene con la solicitud de obtención de suministros

startDate string(date-time) required
Fecha de inicio entre los datos de búsqueda. Formato: AAAA/MM. Ejemplo = 2020/02.
endDate string(date-time) required
Fecha de finalización entre los datos de búsqueda. Formato: AAAA/MM. Ejemplo = 2020/02.
measurementType string required
Establézcalo en 0 (Cero) si desea obtener el consumo por hora y en 1 (Uno) si desea obtener el consumo por cuarto de hora. La consulta cuarta horaria solo está disponible para los PointType 1 y 2, y en el caso de la distribuidora E-distribución adicionalmente el PointType 3
pointType string required
Código de tipo de punto, que se obtiene con la solicitud de obtención de suministros
authorizedNif string
Solo en caso que se quiera obtener los datos de consumo de un NIF autorizado.

En la siguiente imagen se puede observar la ejecución del API con todos los datos completos (Figura 38) [32]:

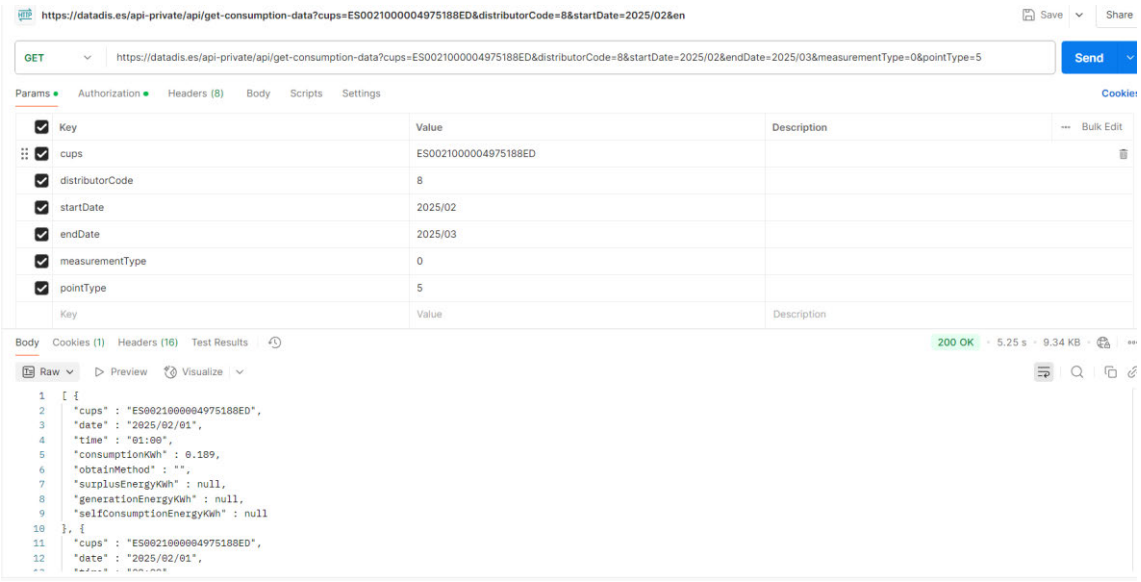


Figura 38: Resultados obtenidos de la consulta sobre datos de consumo de Datadis desde Postaman.

Conectar Node-Red con InfluxDB:

Para poder subir los datos recibidos de Mosquitto, es necesario conectar Node-Red con nuestra base de datos de InfluxDB [33] [34].

Tenemos 2 datos los cuales se quieren subir a través de Node-Red a InfluxDB:

- Datos de consumo
- Datos de producción

Para poder subir ambos datos, se necesitan dos flujos distintos.

Flujo Datos de consumo (Figura 39)

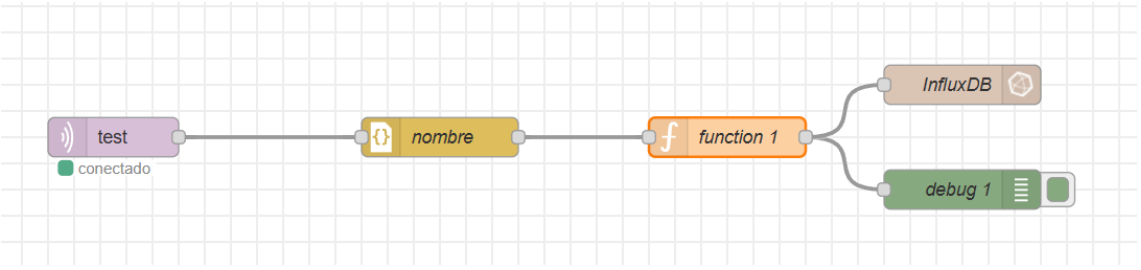


Figura 39: Flujo creado en Node-Red para la obtención de datos de consumo.

Primero se instala la paleta **node-red-contrib-InfluxDB** para usarla a continuación:

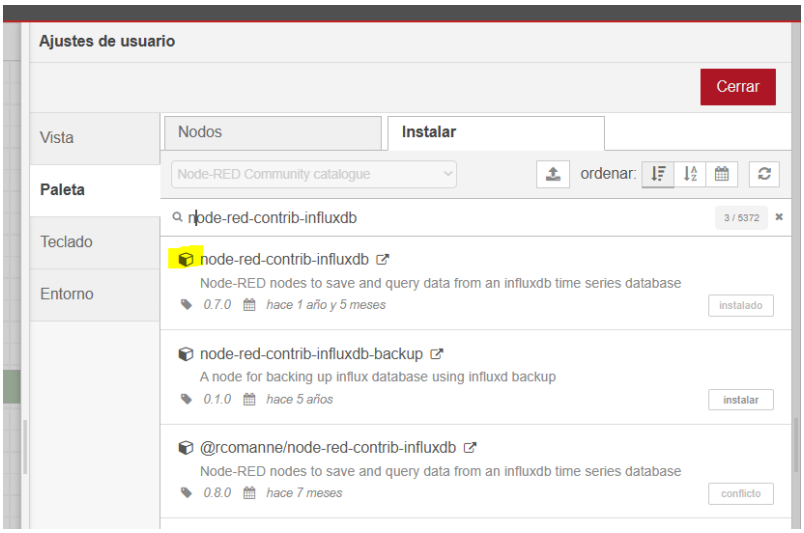


Figura 40: Descarga de la paleta node-red-contrib-InfluxDB.

Se arrastra un nodo InfluxDB out para extraer los datos desde Node-Red a InfluxDB.

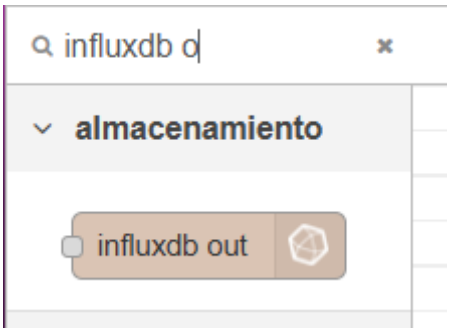


Figura 41: Nodo necesario para la extracción de datos desde Node-Red a InfluxDB.

Se configura el nodo InfluxDB out (Figura 42), primero se añade el servidor completando el nombre que queramos, la versión se mantiene la 1.x, y el host y port se usan los mismo que se han usado en nuestra base de datos de InfluxDB es decir **Host: 127.0.0.1** y **Port: 8086**.

Por último, se añade el nombre de la base de datos en el apartado Database.

Figura 42: Datos a insertar para añadir una base de datos en el nodo InfluxDB out.

Se selecciona el servidor que se ha añadido (Figura 43), y en el apartado de Medición se completa el nombre de la medición en la que se quiere insertar los datos (En este caso “datosPruebas”).

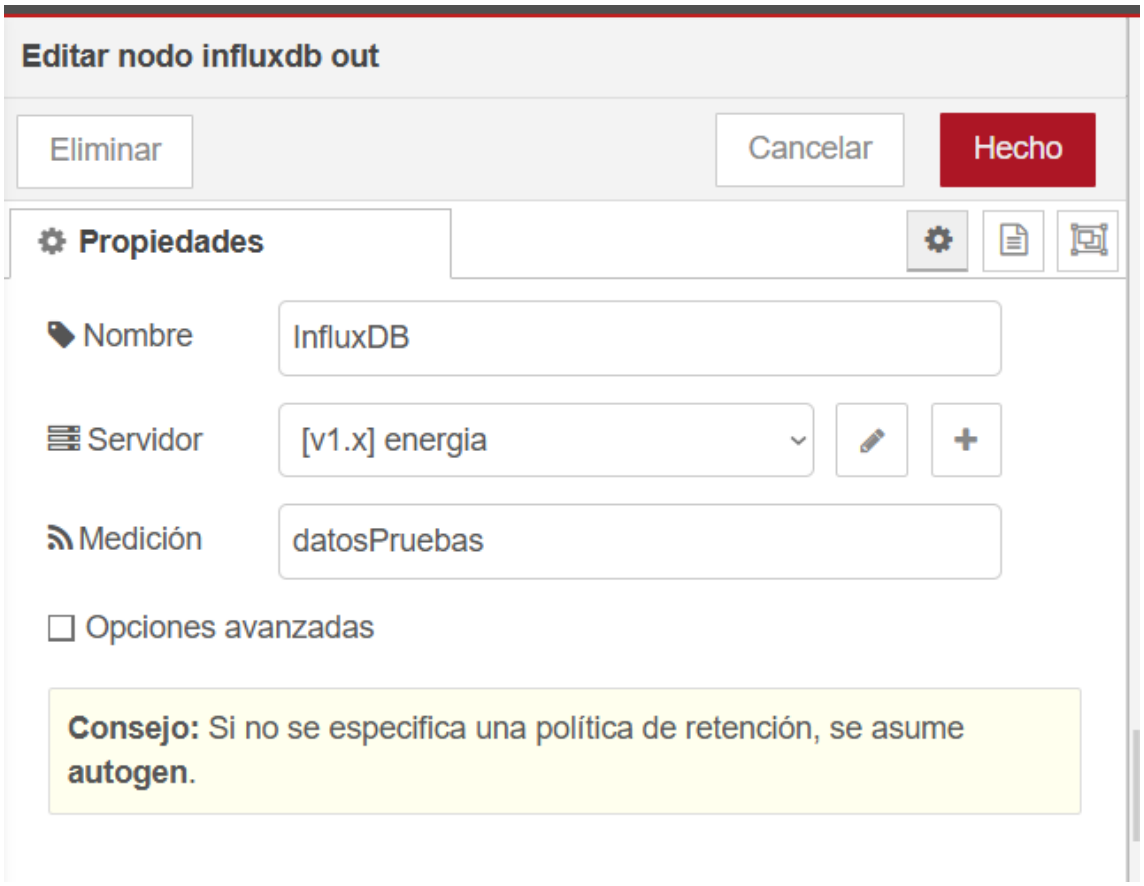


Figura 43: Datos a insertar para el correcto funcionamiento de InfluxDB out.

A continuación, se añade y se une al nodo mqtt un nodo json para convertir los datos recibidos por el nodo mqtt a objetos JavaScript.

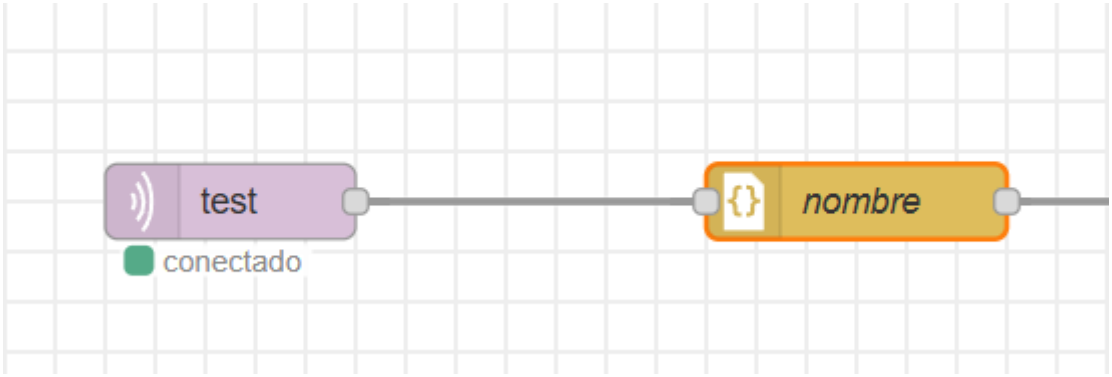


Figura 44: Nodo Json unido a nodo mqtt.

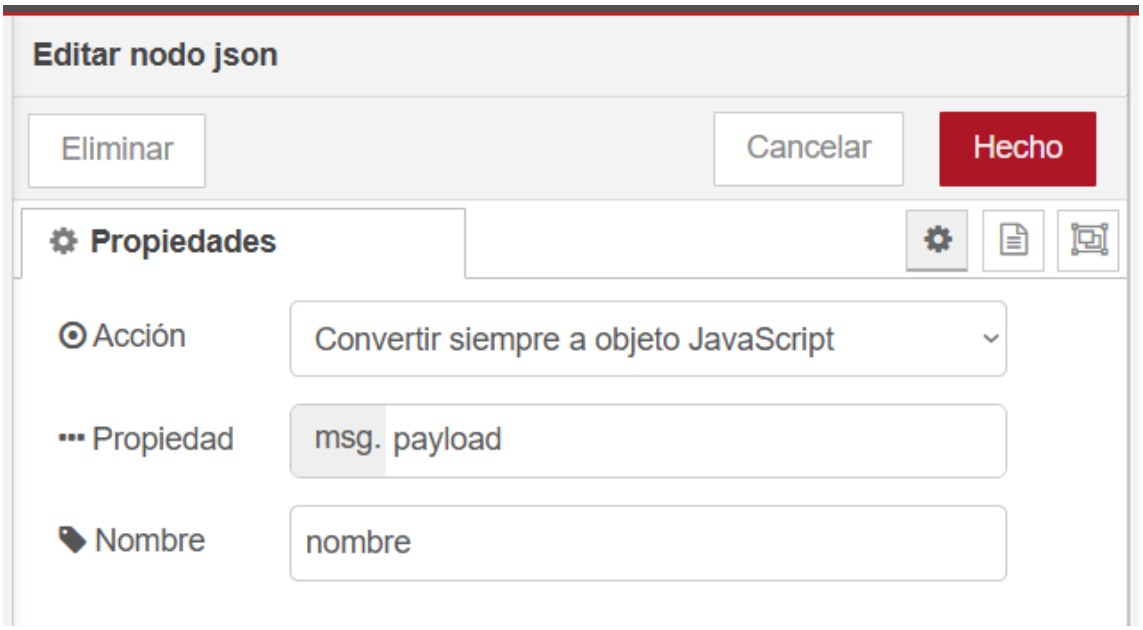


Figura 45: Datos dentro del nodo Json.

A continuación, se usa un nodo function para transmitir los datos recibidos en Mosquitto en formato JSON a un formato legible por InfluxDB:

Figura 46: Nodo function a añadir.

Se edita la función para que devuelva el formato que precisamos (Figura 47):

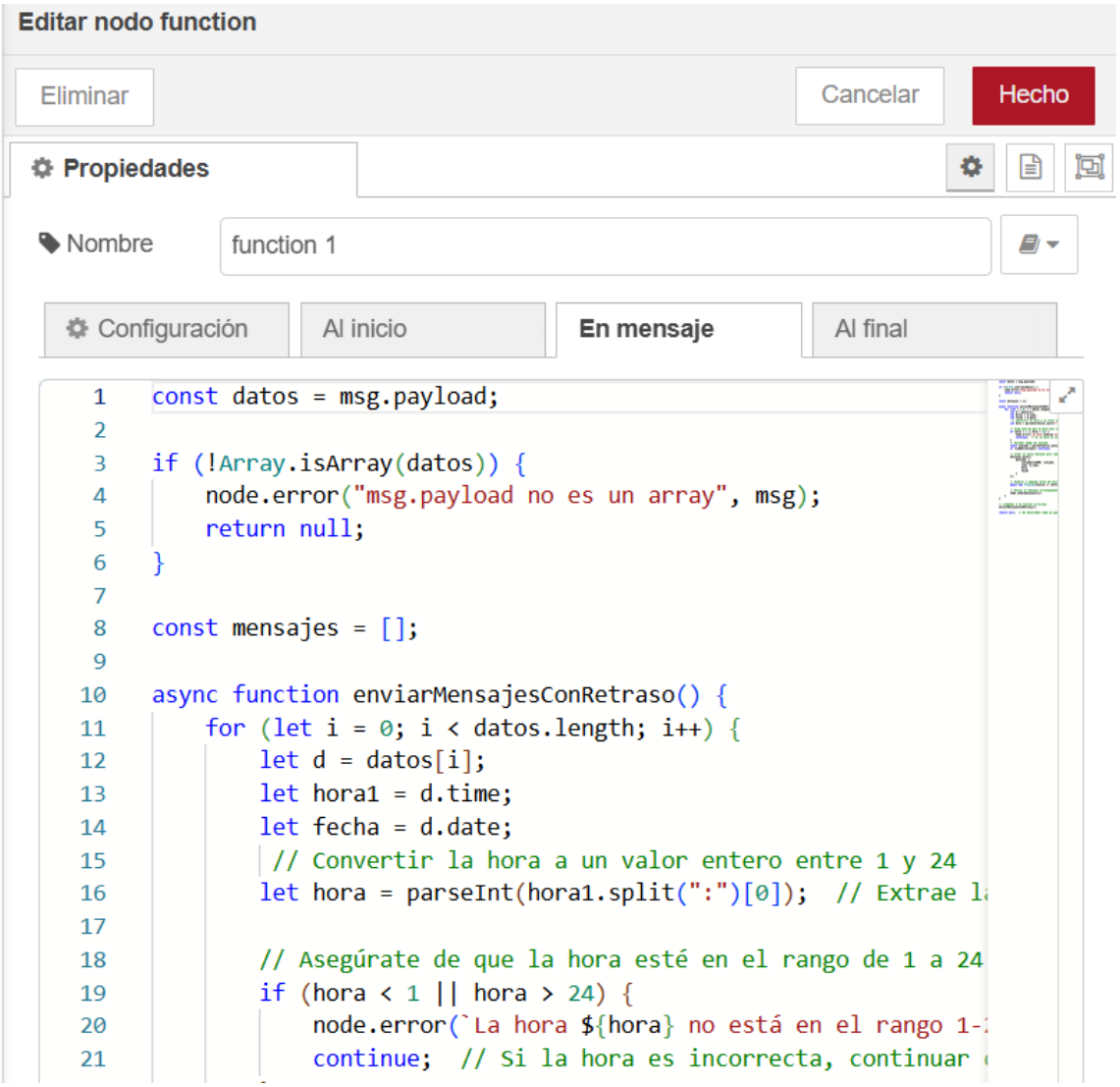


Figura 47: Vistazo al interior del nodo function.

Código para la función:

```
const datos = msg.payload;
if (!Array.isArray(datos)) {
  node.error("msg.payload no es un array", msg);
  return null;
}
const mensajes = [];
datos.forEach(d => {
```

```
  let hora = d.time;
  let fecha = d.date;
  // Validar campo de consumo
  const consumo = parseFloat(d.consumptionKWh);
  if (isNaN(consumo)) return;
  // Crear un nuevo mensaje para cada entrada
  mensajes.push({
    payload: {
      consumptionKWh: consumo,
      cups: d.cups,
      hora
      fecha
    }
  });
});
return [mensajes];
```

Explicación del código utilizado en Node-Red para pasar los datos a InfluxDB

En este fragmento de código, se está procesando la información contenida en el msg.payload, que se espera sea un array de datos. El código se encarga de validar y transformar estos datos para luego enviarlos a InfluxDB:

1. Validación del tipo de datos de msg.payload:

```
const datos = msg.payload;

if (!Array.isArray(datos)) {
  node.error("msg.payload no es un array", msg);
  return null;
}
```

- **Descripción**: La primera parte del código toma los datos de msg.payload (que se espera que sean un array) y valida si realmente es un array utilizando Array.isArray(). En caso contrario, se genera un error con node.error() y la ejecución del flujo se detiene devolviendo null.

2. Creación de un nuevo array mensajes:

```
const mensajes = [];
```

- **Descripción**: Se inicializa un array vacío llamado mensajes, donde se almacenarán los nuevos objetos que serán enviados hacia InfluxDB.

3. Iteración sobre cada elemento de datos:

```
datos.forEach(d => {
  let hora = d.time;
  let fecha = d.date;
```

- **Descripción**: Se utiliza el método forEach() para iterar sobre cada elemento del array datos. Cada elemento (representado por d) contiene información relevante como la hora y la fecha (d.time y d.date), que se extraen y se almacenan en las variables hora y fecha.

4. Validación y conversión del campo consumptionKWh:

```
const consumo = parseFloat(d.consumptionKWh);
if (isNaN(consumo)) return;
```

- **Descripción**: Se toma el valor de consumptionKWh de cada objeto dentro del array y se convierte a un número flotante utilizando parseFloat(). Luego, se valida si el valor resultante es un número mediante isNaN(). Si no es un número válido, se salta la iteración actual utilizando return, lo que evita procesar datos incorrectos.

5. Creación de un nuevo mensaje con los datos procesados:

```
mensajes.push({
  payload: {
    consumptionKWh: consumo,
    cups: d.cups,
    hora,
    fecha
  }
});
```

- **Descripción**: Si el valor de consumptionKWh es válido, se crea un nuevo objeto con los datos relevantes (hora, fecha, consumptionKWh y cups). Este objeto se añade al array mensajes mediante el método push().

6. Retorno del array mensajes:

return [mensajes];

- **Descripción**: Finalmente, se devuelve el array mensajes, que ahora contiene todos los objetos con los datos procesados. Este array será utilizado en el siguiente nodo de Node-Red, probablemente para enviarlo a InfluxDB o realizar alguna otra acción.

Flujo Datos de producción

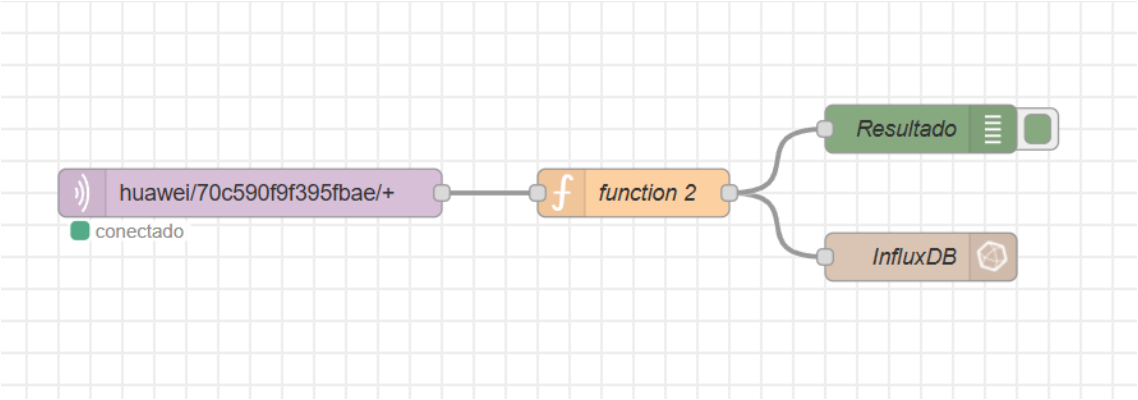


Figura 48: Flujo creado en Node-Red para la obtención de datos de producción.

Se arrastra un nodo InfluxDB out para extraer los datos desde Node-Red a InfluxDB:

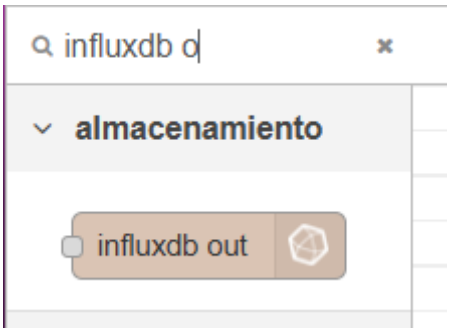


Figura 49: Nodo necesario para la extracción de datos desde Node-Red a InfluxDB.

Se edita el nodo InfluxDB out (Figura 50), primero se añade el servidor completando el nombre que queramos, la versión se mantiene la 1.x, y el host y port se usan los mismo que hemos usado en nuestra base de datos de InfluxDB es decir **Host: 127.0.0.1** y **Port: 8086**.

Por último, se rellena el nombre de nuestra base de datos en el apartado Database.

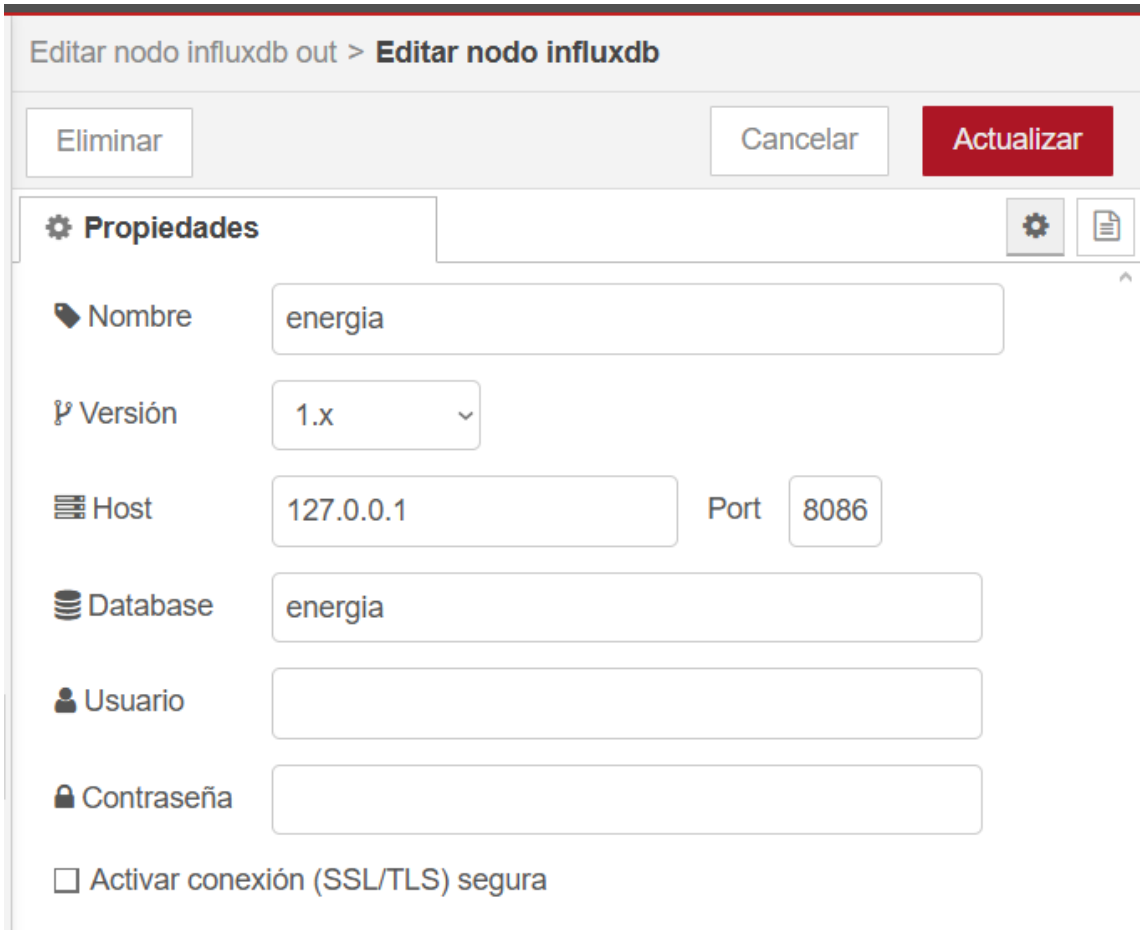


Figura 50: Datos a insertar para añadir una base de datos en el nodo InfluxDB out.

Se selecciona el servidor que se acaba de añadir (Figura 51), y en el apartado de Medición se rellena el nombre de la medición en la que se quiere insertar los datos (En este caso “datosPruebas1”).

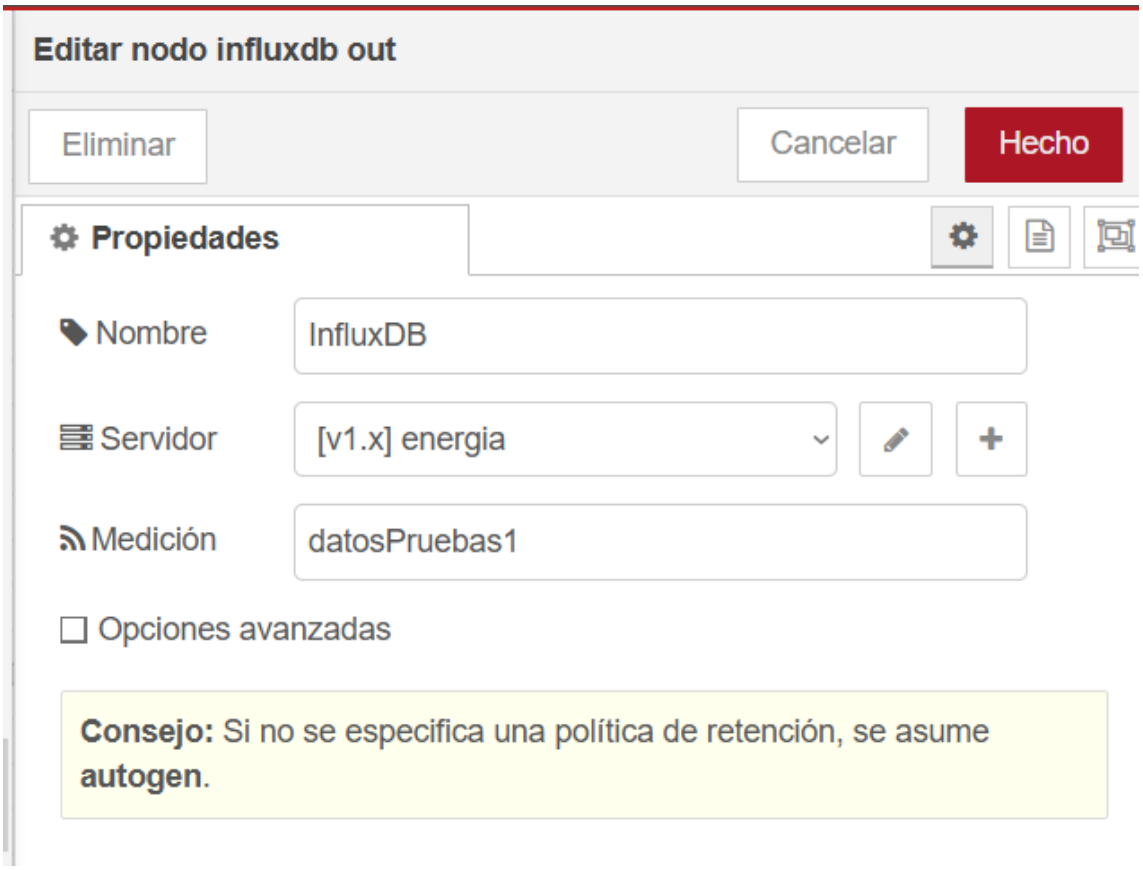


Figura 51: Datos a insertar para el correcto funcionamiento de InfluxDB out.

A continuación, se usa un nodo function para transmitir los datos recibidos en Mosquitto en formato JSON a un formato legible por InfluxDB:

Figura 52: Nodo function a añadir.

Se edita la función para que devuelva el formato que precisamos (Figura 53):

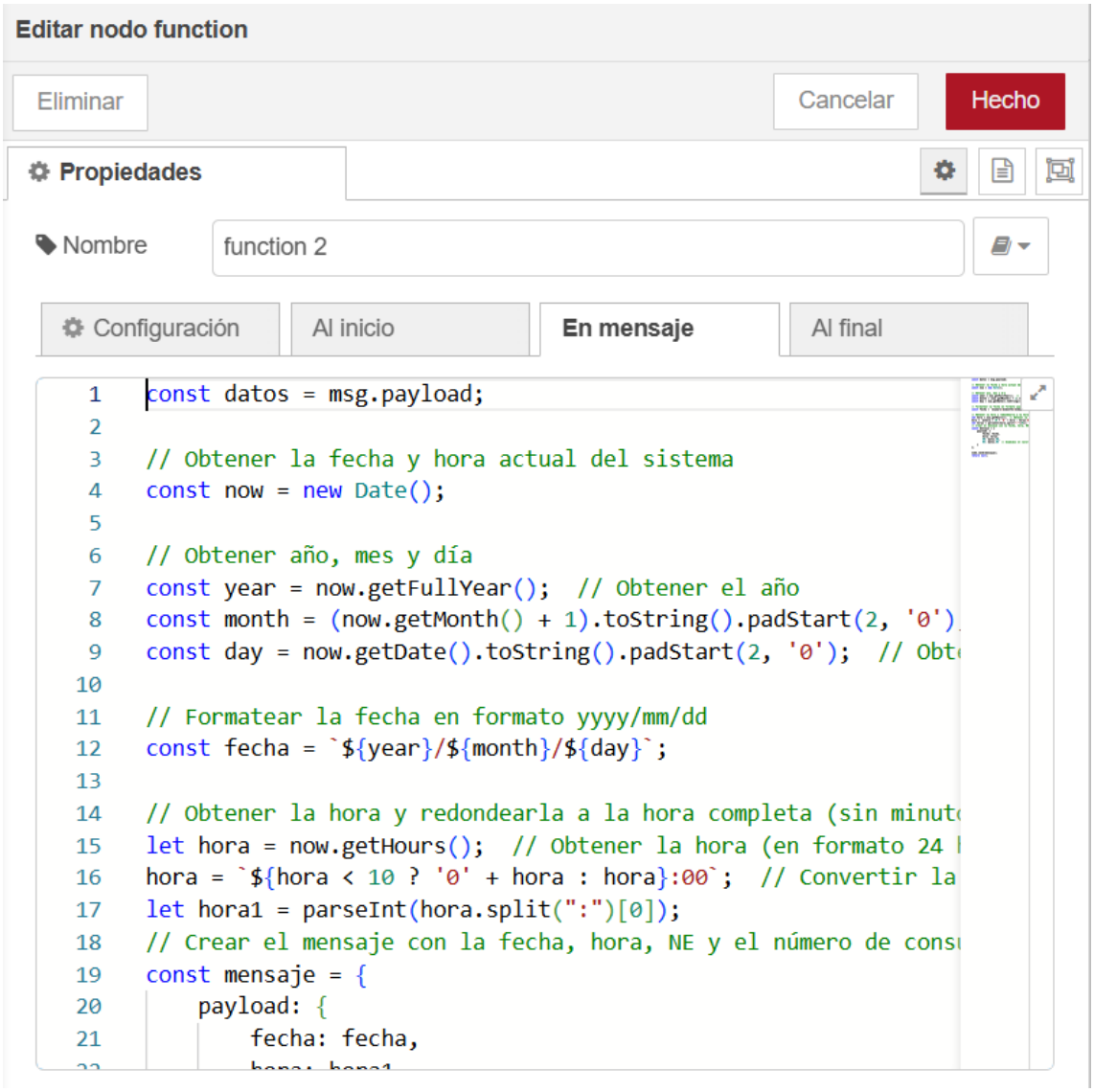


Figura 53: Vistazo al interior del nodo function.

Código para la función:

```
const datos = msg.payload;


// Obtener la fecha y hora actual del sistema
const now = new Date();
```

```
// Formatear la fecha en formato dd/mm/yyyy
const fecha = now.toLocaleDateString("es-ES");  // Usamos el formato español (día/mes/año)

// Formatear la hora en formato hh:mm:ss
const hora = now.toLocaleTimeString("es-ES", { hour12: false });  // Usamos el formato de 24 horas

// Crear el mensaje con la fecha, hora, NE y el número de consumo
const mensaje = {
  payload: {
    fecha: fecha,
    hora: hora,
    number: datos.KW,
    NE: datos.NE  // Añadimos el valor de NE
  }
};

node.send(mensaje);
return null;
```

Explicación del código utilizado para procesar los datos y formatear la fecha y hora

Este código en Node-Red recibe los datos en msg.payload, obtiene la fecha y hora actuales del sistema, y luego crea un mensaje con estos datos formateados y algunos valores adicionales (KW y NE):

1. Obtención de los datos de entrada:

```
const datos = msg.payload;
```

- **Descripción**: Se extraen los datos del msg.payload y se almacenan en la variable datos. Se asume que msg.payload contiene un objeto con valores como KW (consumo) y NE (un identificador o algún tipo de valor asociado).

2. Obtener la fecha y hora actual del sistema:

```
const now = new Date();
```

- **Descripción**: Se crea una nueva instancia de Date, que contiene la fecha y la hora actuales del sistema. La variable now almacenará esta información.

3. Formatear la fecha en formato dd/mm/yyyy:

const fecha = now.toLocaleDateString("es-ES");

- **Descripción**: Se utiliza el método toLocaleDateString() con el parámetro "es-ES" para obtener la fecha en formato español (día/mes/año). Esta fecha se almacena en la variable fecha.

4. Formatear la hora en formato hh:mm:ss:

const hora = now.toLocaleTimeString("es-ES", { hour12: false });

- **Descripción**: Similar al paso anterior, se utiliza toLocaleTimeString() con el parámetro "es-ES" para obtener la hora en formato de 24 horas (hh:mm:ss). El parámetro { hour12: false } asegura que se utilice el formato de 24 horas. Esta hora se almacena en la variable hora.

5. Crear el mensaje con la fecha, hora, NE, y el número de consumo (KW):

```
const mensaje = {
  payload: {
    fecha: fecha,
    hora: hora,
    number: datos.KW,
    NE: datos.NE  // Añadimos el valor de NE
  }
};
```

- **Descripción**: Se crea un objeto mensaje que contiene un nuevo payload. Este payload incluye:
    - **fecha**: La fecha actual formateada como dd/mm/yyyy.
    - **hora**: La hora actual formateada como hh:mm:ss.
    - **number**: El valor de consumo, extraído de datos.KW (se asume que KW está presente en los datos).
    - **NE**: El valor de NE, que es otro campo en los datos entrantes, este valor sirve como identificador del sistema fotovoltaico.

6. Enviar el mensaje a Node-Red:

node.send(mensaje);

return null;

- **Descripción**: Finalmente, se utiliza node.send(mensaje) para enviar el mensaje al siguiente nodo en Node-Red. Esto permite que los datos formateados sean procesados o enviados a otro sistema (como una base de datos o API). El return null; asegura que no se pase nada más al siguiente nodo si no es necesario.

Códigos desarrollados en el proyecto:

A lo largo del proyecto se han desarrollado un gran número de códigos, estos están encargados de realizar todos los cálculos y movimientos de datos necesarios para obtener una experiencia completa tanto del productor como del consumidor.

Para ello hemos creado los siguientes códigos:

abrir.py

Es el encargado de iniciar todo el servidor, lo primero que hace es encender el servidor de Node-Red para así poder recibir mensajes a través de Mosquitto. Una vez encendido el servidor Node-Red se pone en marcha el servidor de Mosquitto el cual es el encargado de enviar tanto los datos de consumo desde Datadis, como los de producción de las estaciones fotoeléctricas a Node-Red

Por último, se inicia el servidor de InfluxDB (Figura 54), este es el encargado de almacenar y gestionar nuestros datos para más adelantes poder mostrarlos.

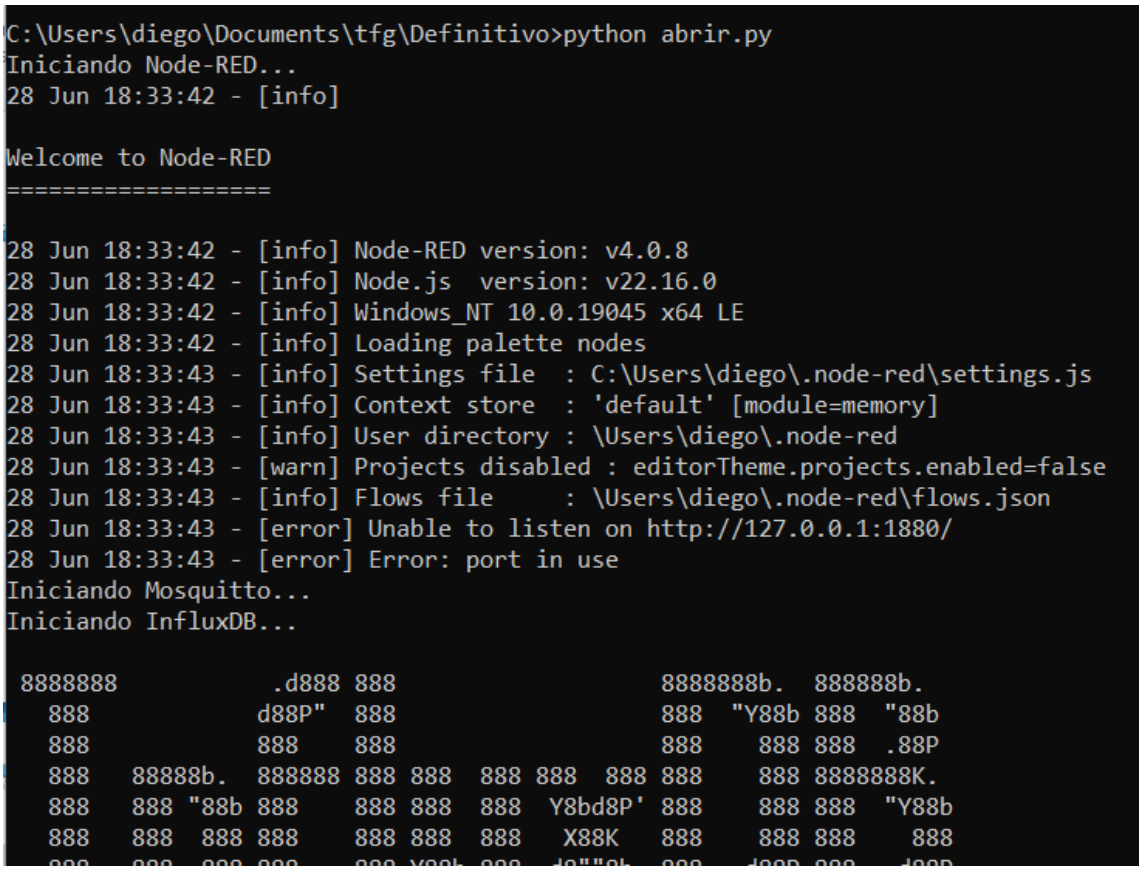


Figura 54: Ejecución del servicio Node-Red en la cmd.

ObtenerDiario.py

Es el encargado de obtener los datos de consumo del día anterior en Datadis a través de una API, para ello necesita realizar una serie de acciones:

- Primero se obtiene el token de seguridad, para ello necesitamos tener una cuenta creada en Datadis y usar el siguiente API el cual devolverá un token que se usará para obtención de datos:

  url = "https://Datadis.es/nikola-auth/tokens/login"
  params = "username=Usuario&password=Contraseña!"

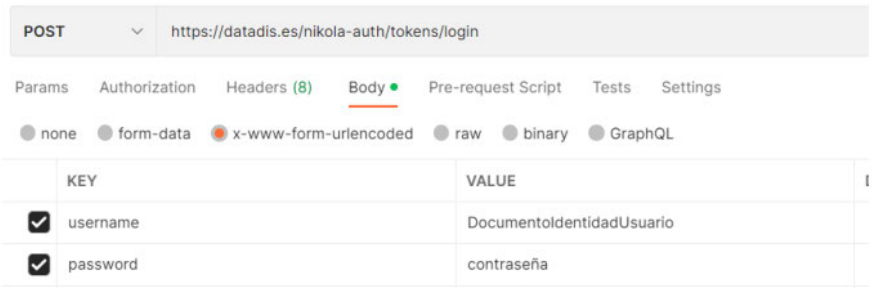


Figura 55: Obtención del token desechable con Postman.

- El segundo paso a realizar es obtener las fechas de inicio y fin de los datos que se quieren obtener, así como el código de distribuidor, el measurementType y el pointType.
  En el siguiente código podemos ver la estructura que se envía a través del API para obtener los datos de consumo del día de ayer:

```
url = "https://Datadis.es/api-private/api/get-consumption-data"
start_date, end_date = obtener_fechas()

  params = {
    "cups": "xxxxxxxxxxxxxxxxxxxx",  # Cambia por el CUPS que necesites
    "distributorCode": "8",  # Código de distribuidor
    "startDate": start_date,
    "endDate": end_date,
    "measurementType": "0",
    "pointType": "5"
  }

  headers = {
    "Authorization": token,
    "accept": "application/json"
  }
```

- A continuación, se envía la petición al API, Datadis tiene el límite de poder realizar solamente 1 petición al día (Figura 56):

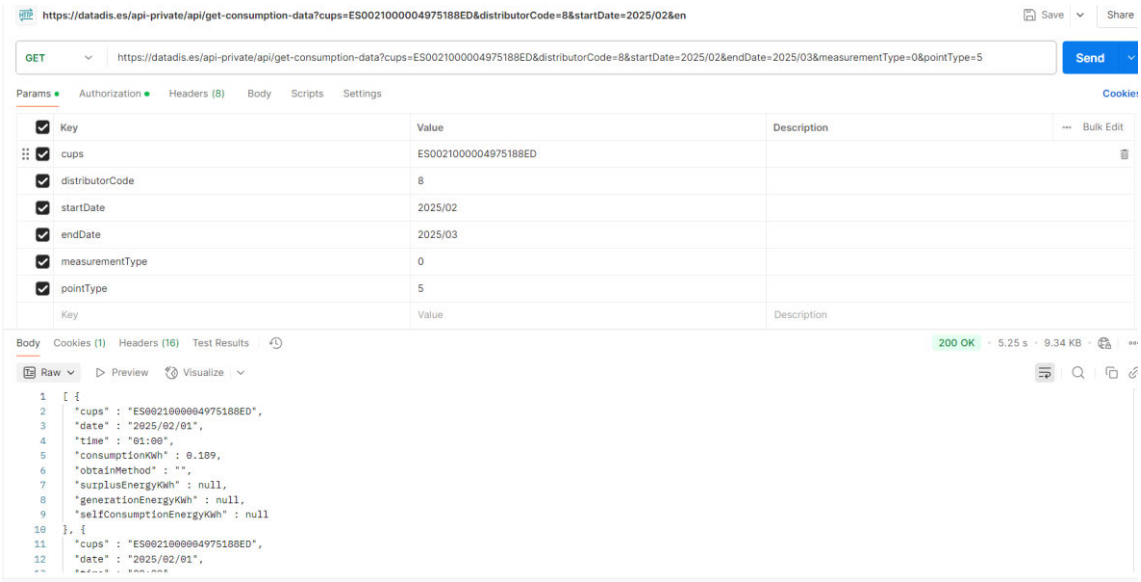


Figura 56: Envió de la solicitud a Datadis para recibir los daros de consumo.

- Una vez se han recibido los datos de consumo del día de ayer, estos se envían al servidor de Mosquitto, estos a su vez se reciben en Node-Red y se envían

nuevamente a InfluxDB, una vez en InfluxDB estos datos se almacenan en la medición “datosPruebas”.

getStationRealKpi_v6.py

Este código ha sido proporcionado por el Instituto de Energía Solar de la UPM para acceder a los inversores de la marca Huawei de 3 instalaciones fotovoltaicas. El resultado de ejecutar este código con las credenciales que nos han otorgado es el envío de los KW acumulados totales de la instalación, este valor se envía a través del servidor de Mosquitto y se recibe en Node-Red.

Estos datos se envían desde Node-Red a InfluxDB y se almacenan en la medición “datosPruebas1” (Figura 57):

Figura 57: Resultados obtenidos por hora del código getStationRealKpi_v6.py.

KWH.py

Este código convierte los datos obtenidos de getStationRealKpi_v6.py en KW/h ya que antes se obtenían KW totales acumulados.

Para ello lo que se hace es lo siguiente:

- Se obtienen KW[x] y KW[x+1] donde x es la hora
- Se resta KW[x+1] - KW[x]

```
generacion_solar = p['KW'] - generacion_anterior  # Diferencia entre la hora actual y la anterior
```

Figura 58: Cálculo para la obtención de KW/h.

- El resultado son los KW que se han obtenido desde la hora X a la hora X+1
- Una vez se han guardado los 24 datos por hora del día anterior se envían directamente a InfluxDB con la siguiente estructura (Figura 59):

```
# Guardar el valor calculado en "datosPruebas3"
datos_finales.append({
    "measurement": "datosPruebas3",
    "tags": {
        "fecha": fecha_solar,
        "hora": hora_solar  # Usamos la hora como un entero entre 1 y 24
    },
    "fields": {
        "produccionKWh": float(generacion_solar)
    }
})
```

Figura 59: Formato de inserción de datos KW/h en datosPruebas3.

Lo que hace esta estructura es guardar la fecha, hora y produccionKWh en el measurement seleccionado, en este caso “datosPruebas3”

ConsumoAhorro.py

Este código obtiene la cantidad de KW que se ha auto consumido teniendo en cuenta lo consumido por hora y el % que se obtiene de la producción total en KW/h

Para ello seguimos los siguientes pasos:

- Primero se obtienen los datos de consumo y los datos de producción KW/h (Figura 60):

```python
# Función para obtener datos de consumo y mostrar los resultados
def obtener_consumo():
    fecha_ayer = obtener_fecha_ayer()  # Obtener la fecha de ayer
    query_consumo = f'SELECT * FROM "datosPruebas" WHERE fecha = \'{fecha_ayer}\''
    result_consumo = client.query(query_consumo)
    # Imprimir los resultados de la consulta
    print("Datos de consumo:")
    for point in result_consumo.get_points(measurement='datosPruebas'):
        print(point)
    return list(result_consumo.get_points(measurement='datosPruebas'))

# Función para obtener datos de producción de panel solar utilizando fecha en lugar de time
def obtener_produccion_solar():
    fecha_ayer = obtener_fecha_ayer()  # Obtener la fecha de ayer
    query_solar = f'SELECT * FROM "datosPruebas3" WHERE fecha = \'{fecha_ayer}\''
    result_solar = client.query(query_solar)
    # Imprimir los resultados de la consulta
    print("Datos de producción solar:")
    for point in result_solar.get_points(measurement='datosPruebas3'):
        print(point)
    return list(result_solar.get_points(measurement='datosPruebas3'))
```

Figura 60: Código para la obtención de consumo y producción del día de ayer desde InfluxDB.

- A continuación, se multiplican los KW/h producidos por el % que te pertenece, por ejemplo, si se han producido 40 KW y tu posees los derechos sobre un 2% de ellos, te corresponde 0.8 KW
- En base a los datos obtenidos se realiza una operación IF, si la generación KW/h que se ha obtenido en base al % que te pertenece es mayor que el consumo de esa hora, entonces el valor del autoconsumo será igual al valor del consumo de esa hora. En el caso contrario, si la generación KW/h que se ha obtenido en base al % que te pertenece es menor que el consumo de esa hora, entonces el valor del autoconsumo será igual al valor de la generación de KW/h ajustada (Figura 61):

```python
# Aplicar el porcentaje de la producción solar
generacion_solar_ajustada = generacion_solar * porcentaje_solar
if(generacion_solar_ajustada>consumo_kwh):
    consumo_neto = consumo_kwh
elif(generacion_solar_ajustada<consumo_kwh):
    consumo_neto =  generacion_solar_ajustada
```

Figura 61: Calculo para la obtención de la energía auto consumida.

- Una vez están hechos los cálculos de las últimas 24h, estos se envían a InfluxDB, en este caso se envían a la medición “datosPruebas2” (Figura 62):

```
# Guardar el valor calculado en "datosPruebas2"
datos_finales.append({
    "measurement": "datosPruebas2",
    "tags": {
        "fecha": fecha_consumo,
        "hora": hora_consumo  # Usamos la hora como número entero entre 1 y 24
    },
    "fields": {
        "consumoNetoKWh": consumo_neto
    }
})
```

Figura 62: Formato de inserción de datos de auto consumo en datosPruebas2.

ForecastConsumo.py

Este código es el encargado de predecir el consumo del día de mañana, está formado por varias partes:

- La primera parte es la encargada de recopilar datos de InfluxDB, en concreto queremos obtener los datos de consumo de los últimos 4 días (Figura 63):

```
def obtener_datos_influxdb():
    # Consultar los datos de "datosPruebas" para los últimos 4 días
    ayer = (datetime.today() - timedelta(days=1)).strftime('%Y/%m/%d')
    hace_dos = (datetime.today() - timedelta(days=2)).strftime('%Y/%m/%d')
    hace_tres = (datetime.today() - timedelta(days=3)).strftime('%Y/%m/%d')
    hace_cuatro = (datetime.today() - timedelta(days=4)).strftime('%Y/%m/%d')

    query = f"""
    SELECT * FROM "datosPruebas"
    WHERE "fecha" = '{ayer}' OR "fecha" = '{hace_dos}' OR "fecha" = '{hace_tres}' OR "fecha" = '{hace_cuatro}'
    """

    # Ejecutar la consulta y obtener los resultados
    result = client.query(query)
    # Convertir los resultados a un DataFrame
    df = pd.DataFrame(list(result.get_points(measurement='datosPruebas')))

    return df
```

Figura 63: Función encargada de obtener los datos de consumo de los últimos 4 días.

- La segunda parte del código es la encargada de registrar la temperatura de mañana usando un API de openweathermap**:**

  https://api.openweathermap.org/data/2.5/forecast?lat=40.4168&lon=-3.7038&appid=xxxxxxxxxxxxxxxxxxxxxxxx
  Esta API gratuita devuelve una serie de valores cada 3 horas de los 5 próximos días, el valor que nos interesa guardar en este caso es la temperatura.

```
# Filtrar los datos de la API para el día siguiente (17/06/2025)
forecast_data = data['list']
forecast_filtered = []
fecha_manana = (datetime.today() + timedelta(days=1)).date()
# Almacenar las predicciones disponibles
for hour in forecast_data:
    timestamp = pd.to_datetime(hour['dt'], unit='s')
    if timestamp.date() == fecha_manana:
        clima = hour['weather'][0]['description']  # Clima
        temperatura_kelvin = hour['main']['temp']  # Temperatura en Kelvin
        temperatura_celsius = temperatura_kelvin - 273.15  # Convertir a Celsius
        hora = timestamp.strftime('%H:%M')  # Hora en formato de 24 horas
        forecast_filtered.append([hora, round(temperatura_celsius, 2), clima])

# Ahora, realizamos la interpolación lineal de temperatura entre las horas
all_hours = np.arange(0, 24)
interpolated_temperatures = []
```

Figura 64: Código que obtiene la temperatura por hora del día siguiente gracias a un API.

- La tercera parte del código es la obtención de una media basada en los KWh consumidos de los últimos 4 días (Figura 65):

```
for i, temperature in enumerate(interpolated_temperatures, start=1):
    print(f"Hora: {i:02}:00 | Temperatura: {temperature:.2f}°C")

    df_hour = df[df['hora'] == i]  # Filtrar datos para esa hora
    media_consumo = df_hour['consumptionKWh'].mean()

    # Almacenar la media de consumo en la lista
    medias_consumo.append(media_consumo)

    # Calcular el consumo ajustado con la regla
    consumo_ajustado = calcular_consumo(temperature, i, media_consumo)
```

Figura 65: Código que obtiene la media del consumo de los últimos 4 días según la hora.

- Por último, en base a los resultados obtenidos en la media de los últimos 4 días se aplica una última operación, con la cual se busca ajustar lo máximo posible la estimación del forecast. Teniendo en cuenta la temperatura y hora actual, se aumenta el consumo en un x% asumiendo el uso de la calefacción o el aire acondicionado (Figura 66):

```
def calcular_consumo(temperatura, hora, consumo_base):
    # Regla para calefacción (Noche)
    if 12 <= temperatura < 16 and (6 <= hora <= 7 or 18 <= hora <= 23):
        calefaccion = 0.02  # 5% de aumento en consumo por calefacción
        consumo_base *= (1 + calefaccion)
    elif temperatura < 12 and (6 <= hora <= 7 or 18 <= hora <= 23):
        calefaccion = 0.05  # 10% de aumento en consumo por calefacción
        consumo_base *= (1 + calefaccion)

    # Regla para aire acondicionado (Día)
    if temperatura > 30 and (8 <= hora <= 18):
        aire_acondicionado = 0.15  # 15% de aumento por aire acondicionado
        consumo_base *= (1 + aire_acondicionado)
    elif 25 <= temperatura <= 30 and (8 <= hora <= 18):
        aire_acondicionado = 0.10  # 10% de aumento por aire acondicionado
        consumo_base *= (1 + aire_acondicionado)

    return consumo_base
```

Figura 66: Cálculo para aproximar el consumo obtenido al real en base a la temperatura.

- Una vez obtenidos todos los datos estimados del forecast de mañana, estos se envían con la estructura de la imagen.
  Se almacenan en InfluxDB en el measurement “datosPruebasF” (Figura 67):

```
# Almacenar los datos para enviar a InfluxDB
datos_finales.append({
    "measurement": "datosPruebasF",  # El nombre del measurement
    "tags": {
        "fecha": fecha_manana,  # Formato de fecha: YYYY/MM/DD
        "hora": i
    },
    "fields": {
        "consumoKWh": consumo_ajustado  # Consumo ajustado
    }
})
```

Figura 67: Formato de inserción de datos de forecast de consumo en datosPruebasF.

ForecastProduccion.py

Este código es el encargado de predecir la producción en KW/h del día de mañana, está formado por varias partes:

- La primera parte es la encargada de recopilar datos de InfluxDB, en concreto buscamos obtener los datos de producción KW/h de los últimos 4 días (Figura 68):

```python
def obtener_datos_influxdb():
    # Consultar los datos de "datosPruebas3" para los últimos 4 días
    ayer = (datetime.today() - timedelta(days=1)).strftime('%Y/%m/%d')
    hace_dos = (datetime.today() - timedelta(days=2)).strftime('%Y/%m/%d')
    hace_tres = (datetime.today() - timedelta(days=3)).strftime('%Y/%m/%d')
    hace_cuatro = (datetime.today() - timedelta(days=4)).strftime('%Y/%m/%d')

    query = f"""
    SELECT * FROM "datosPruebas3"
    WHERE "fecha" = '{ayer}' OR "fecha" = '{hace_dos}' OR "fecha" = '{hace_tres}' OR "fecha" = '{hace_cuatro}'
    """

    # Ejecutar la consulta y obtener los resultados
    result = client.query(query)
    # Convertir los resultados a un DataFrame
    df = pd.DataFrame(list(result.get_points(measurement='datosPruebas3')))
    print("Datos obtenidos de InfluxDB:")

    return df
```

Figura 68: Obtención de los datos de consumo de los 4 últimos días.

- La segunda parte del código es la encargada de registrar la temperatura y el % de nubes en el cielo de mañana usando un API de openweathermap**:**

  https://api.openweathermap.org/data/2.5/forecast?lat=40.4168&lon=-3.7038&appid=xxxxxxxxxxxxxxxxxxxxxxxx

  Esta API gratuita devuelve una serie de valores cada 3 horas de los 5 próximos días, el valor que nos interesa guardar en este caso es la temperatura y el % de nubes en el cielo.

```python
# Almacenar las predicciones disponibles
for hour in forecast_data:
    timestamp = pd.to_datetime(hour['dt'], unit='s')
    if timestamp.date() == fecha_manana:  # Filtrar solo el día de mañana
        clima = hour['weather'][0]['description']  # Clima
        temperatura_kelvin = hour['main']['temp']  # Temperatura en Kelvin
        temperatura_celsius = temperatura_kelvin - 273.15  # Convertir a Celsius
        hora = timestamp.strftime('%H:%M')  # Hora en formato de 24 horas
        nubes = hour['clouds']['all']  # Porcentaje de nubes
        forecast_filtered.append([hora, round(temperatura_celsius, 2), nubes])
```

Figura 69: Obtención de la temperatura y % de nubes del día siguiente a través de una API.

- La tercera parte del código es la obtención media de la produccionKWh de los últimos 4 días (Figura 70):

```
for i, (temperature, nubes) in enumerate(zip(interpolated_temperatures, interpolated_nubes), start=1):
    print(f"Hora: {i:02}:00 | Temp: {temperature:.2f}°C | Nubes: {nubes:.2f}%")

    # Realizar la regresión lineal por hora para predecir la producción fotovoltaica
    df_hour = df[df['hora'] == i]  # Filtrar datos para esa hora
    media_produccion = df_hour['produccionKWh'].mean()

    # Almacenar la media de consumo en la lista
    medias_produccion.append(media_produccion)


    # Calcular la radiación solar estimada
    radiacion_solar = calcular_radiacion(temperature, nubes)

    # Ajustar la producción fotovoltaica en base a la radiación solar
    produccion_fv = calcular_produccion_fv(radiacion_solar, media_produccion)
```

Figura 70: Código que obtiene la media en la producción de los últimos 4 días según la hora.

- Ahora en base a los resultados obtenidos en la media de los últimos 4 días se aplica una serie de operaciones con las cuales se busca ajustar lo máximo posible la estimación del forecast.

  La primera operación (Figura 71) es una estimación de la radicación actual en base a dos variables, la temperatura y el % de nubes en el cielo actualmente.

```
# Regla para calcular la radiación solar ajustada en base a la temperatura y nubes
def calcular_radiacion(temperatura, nubes):
    # Estimación de la radiación solar directa
    G_direct = 1000 * ((temperatura+ 273.15) / 300)  #100 Estimación simple de irradiancia directa (1000 W/m² cuando T ~ 300 K)

    # Ajuste por nubes
    G_solar = G_direct * (1 - nubes / 100)  # Ajuste de irradiancia por el porcentaje de nubes

    return G_solar
```

Figura 71: Código que obtiene una aproximación de la radiación en base a la temperatura y % de nubes.

La segunda operación (Figura 72) que se realiza, es en base a la radiación actual obtenida en el punto anterior y a los resultados de la media, en base a ambos cálculos vamos a aumentar/disminuir un x% el resultado teniendo en cuenta la radiación actual:

```
# Regla para calcular la producción fotovoltaica ajustada
def calcular_produccion_fv(radiacion_solar, produccion_base):
    # Suponiendo que la eficiencia del sistema fotovoltaico es del 15% (puedes ajustarlo)
    eficiencia = 0.15
    # La producción fotovoltaica es la irradiancia solar multiplicada por la eficiencia
    P_AC = radiacion_solar * eficiencia
    # Ajuste de la producción fotovoltaica en función de la radiación solar
    produccion_ajustada = P_AC * produccion_base / 100  # Ajustamos la producción de acuerdo con la radiación
    return produccion_ajustada
```

Figura 72: Código que en base a la radiación y la producción calculada obtiene un cálculo final.

- Una vez obtenidos todos los datos estimados del forecast de mañana, estos se envían con la estructura de la imagen, se almacenan en InfluxDB en el measurement “datosPruebasF1” (Figura 73):

```
# Almacenar los datos para enviar a InfluxDB
datos_finales.append({
    "measurement": "datosPruebasF1",  # El nombre del measurement
    "tags": {
        "fecha": fecha_manana,  # Formato de fecha: YYYY/MM/DD
        "hora": i
    },
    "fields": {
        "produccionKWh": produccion_fv  # Producción fotovoltaica ajustada
    }
})
```

Figura 73: Formato de inserción de datos de forecast de producción en datosPruebasF1.

Ejecución.py

Este código es el encargado de ejecutar y gestionar el resto de códigos siguiendo un orden y unos tiempos de ejecución (Figura 74):

```
def ejecutar_diariamente():
    while True:
        # Esperar hasta medianoche
        esperar_hasta_medianoche()

        # Ejecutar los scripts a las 00:00
        ejecutar_obtener_diario()
        time.sleep(25)  # Puede agregar un pequeño retraso si es necesario entre los scripts
        ejecutar_kwh()
        time.sleep(5)
        ejecutar_consumo_ahorro()
        time.sleep(5)
        ejecutar_convertir_a_dolar()
        time.sleep(5)
        ejecutar_forecast_consumo()
        time.sleep(5)
        ejecutar_forecast_produccion()

# Iniciar la ejecución diaria
ejecutar_diariamente()
```

Figura 74: Función que ejecuta según un orden varios códigos.

En la imagen (Figura 74) se observa ver cómo se ejecutan todos los códigos anteriores después de una función llamada esperar_hasta_medianoche() esta función crea una espera inactiva haciendo de esta forma que se ejecuten todos los códigos una sola vez al día:

```python
# Función para esperar hasta medianoche y luego ejecutar los scripts
def esperar_hasta_medianoche():
    now = datetime.now()
    # Calcular el tiempo que falta hasta las 12:05 PM del día actual o el siguiente
    twelve_five_pm = datetime.combine(now.date(), datetime.min.time()) + timedelta(hours=10,minutes=5)

    # Si ya ha pasado las 12:05 PM hoy, espera hasta las 12:05 PM del día siguiente
    if now > twelve_five_pm:
        twelve_five_pm += timedelta(days=1)

    time_to_wait = (twelve_five_pm - now).total_seconds()
    print(f"Esperando hasta las 12:05 PM... Tiempo restante: {time_to_wait} segundos")
    time.sleep(time_to_wait)
```

Figura 75: Función que ejecuta una espera inactiva hasta las 10:05 am.

Se puede ver que la función (Figura 75) espera hasta las 10:05 am, esto es debido que al realizar varias pruebas de inserción se observó que Datadis no sube los datos del día anterior hasta pasadas las 10:00 am. Por lo tanto, si se quiere obtener los datos de consumo de ayer, se debe ejecutar el código pasadas las nueve de la mañana.

app.py

El objetivo del código de esta aplicación [35], es poder con un solo click iniciar todo el servidor y abrir una interfaz de usuario en el explorador donde se pueden consultar todos los datos (Detallado más adelante), este código está formado por dos partes:

1. La primera son los API de los distintos measurements de nuestro proyecto, lo que hacen estos API es usar una query para obtener ciertos datos de nuestra base de datos en InfluxDB. Estas rutas API más tarde se llaman desde index.html que es el frontend de la aplicación web.

```python
# Ruta para obtener los datos de consumo
@app.route('/api/consumo', methods=['GET'])
def obtener_consumo():
    try:
        # Realizar la consulta a InfluxDB
        query = 'SELECT * FROM "datosPruebas"'
        result = client.query(query)

        # Verificar que la consulta ha devuelto resultados
        if not result:
            return jsonify({"error": "No data found for consumption"}), 404

        # Procesar los datos y devolverlos en formato JSON
        data = list(result.get_points(measurement='datosPruebas'))
        return jsonify(data)
    except Exception as e:
        return jsonify({"error": str(e)}), 500  # En caso de error, retornamos un error 500
```

Figura 76: Código que obtiene los datos de consumo desde InfluxDB y puede ser llamado por una API.

2. La segunda parte es el main, en este apartado se realizan varias cosas:
    a. Primero, se ejecuta usando Threads el código ya mencionado abrir.py para así inicializar todo el servidor.
    b. Luego de la misma forma se ejecuta el código Ejecucion.py, este se encarga de programar el resto de códigos para que se ejecuten una vez al día.

c. Más adelante, se usa una función que abre automáticamente la app web en el buscador de internet (open_browser).
d. Por último, se usa app.run con el host y port que se prefiera para que el servidor de la app se ejecute correctamente en el buscador.

```python
if __name__ == "__main__":

    # Crea los hilos para ejecutar las funciones
    threading.Thread(target=ejecutar_abrir).start()
    threading.Thread(target=ejecutar_ejecucion).start()
    threading.Thread(target=open_browser).start()

    # Ejecuta la aplicación Flask o el servidor que estés usando
    app.run(debug=True, host='0.0.0.0', port=8080, use_reloader=False)

    # Cambia el valor de 'una' para evitar que el código se ejecute nuevamente
```

Figura 77: Código que ejecuta otros códigos usando threads.

Index.html

Este último código es el frontend de la aplicación web [36], que a su vez está conectado con app.py para la obtención de datos en InfluxDB, se divide en tres partes:

1. **El diseño de la aplicación web,** esta primera parte es la encargada de darle forma y un aspecto visual agradable a la aplicación web (Figura 78):

Figura 78: Captura de la aplicación creada.

Para ello se han usado varios elementos como checkboxes, tablas, gráficos, botones, … Y de esta forma permitir al usuario entender y visualizar todos los datos de un solo vistazo. También se le ha dado un style a cada uno de los elementos para hacer que la aplicación tenga un aspecto más atractivo.

2. **Script de la aplicación web,** se ha usado varias funciones para manejar la lógica del rango de días, así como de la visualización de los datos tanto en la gráfica como en la tabla (Figura 79):

```
// Función para establecer el rango de fechas según el botón presionado
function setDateRange(days) {
    const now = new Date();
    if (days === 'all') {
        selectedStartDate = '2000-01-01'; // Fecha inicial muy antigua para mostrar todos los datos
        selectedEndDate = now.toISOString().split('T')[0]; // Fecha de fin es el día actual
    }else if(days==1){
        selectedStartDate = now.toISOString().split('T')[0]; // Fecha de fin es el día actual
        selectedEndDate = now.toISOString().split('T')[0]; // Fecha de fin es el día actual
    }else if(days==-1){
        selectedStartDate = new Date(now - days * 24 * 60 * 60 * 1000).toISOString().split('T')[0]; // Hace "days" días
        selectedEndDate = new Date(now - days * 24 * 60 * 60 * 1000).toISOString().split('T')[0]; // Hace "days" días
    }
    else {
        selectedStartDate = new Date(now - days * 24 * 60 * 60 * 1000).toISOString().split('T')[0]; // Hace "days" días
        selectedEndDate = now.toISOString().split('T')[0]; // Fecha de fin es el día actual
    }

    document.getElementById('startDate').value = selectedStartDate;
    document.getElementById('endDate').value = selectedEndDate;
    fetchData();
}
```

Figura 79: Parte del código encargada de gestionar el rango de días.

3. **Creación de tabla,** se ha creado una tabla con los datos seleccionados por las mediciones. Con esta tabla se busca facilitar la comprensión de los datos por parte del usuario.

```
// Función para mostrar la tabla de los datos
function displayTable(data,tipo) {
    let tableContent = document.getElementById('table-content');
    let table = '<table>';
    table += '<tr><th>Fecha</th><th>Hora</th><th>Valor</th></tr>';

    data.forEach(item => {
        table += `<tr><td>${item.fecha}</td><td>${item.hora}</td><td>${item.consumptionKWh || item.KW || item.consumoNetoKWh || item.produccionKWh || item.costeDia || item.produccionKWh}
    });

    table += '</table>';
    // Añadir la tabla al contenedor
    tableContent.innerHTML += `<div class="table-container"><h3>Tabla: ${tipo}</h3>${table}</div>`;
}
```

Figura 80: Función que genera la tabla.

4. **Creación del gráfico,** [37] se ha creado un gráfico con las mediciones seleccionadas, contiene los mismos datos que la tabla, pero en un formato de grafico de barras.

```
function createChart(dataSets) {
    const ctx = document.getElementById('chart').getContext('2d');

    // Crear datasets para el gráfico
    const datasets = dataSets.map((set, index) => {
        const horas = set.data.map(item => item.hora);
        const valores = set.data.map(item => item.consumptionKWh || item.KW || item.consumoNetoKWh || item.produccionKWh

        return {
            label: `Datos de ${set.tipo.charAt(0).toUpperCase() + set.tipo.slice(1)}`,
            data: valores,
            backgroundColor: `rgba(${(index + 1) * 50}, ${(index + 1) * 30}, ${(index + 1) * 100}, 0.2)`,
            borderColor: `rgba(${(index + 1) * 50}, ${(index + 1) * 30}, ${(index + 1) * 100}, 1)`,
            borderWidth: 1
        };
    });

    // Si ya existe un gráfico, lo destruye antes de crear uno nuevo
    if (window.chartInstance) {
        window.chartInstance.destroy();
    }

    // Crear el gráfico con Chart.js
    window.chartInstance = new Chart(ctx, {
        type: 'bar',
        data: {
            labels: dataSets[0].data.map(item => item.hora),  // Usamos las horas del primer set de datos
            datasets: datasets
        },
        options: {
            scales: {
                y: {
                    beginAtZero: true
                }
            },
            responsive: true,
            maintainAspectRatio: false,
        }
    });
}
```

Figura 81: Función que genera el gráfico.

Aplicación web:

Creación:

Para la creación de la aplicación web se ha instalado pyinstaller a través de la cmd usando el comando: **pip install Flask pyinstaller** [38].

Ahora es necesario la creación de un código en nuestro caso app.py para que gestione las operaciones que se realizaran de fondo.

Por último, se crea una carpeta con el nombre “templates” y dentro se crea index.html, este código es el encargado de dar forma y un aspecto visual a la página.

Una vez se tiene tanto el frontend (index.html) como el backend (app.py), ejecutamos el siguiente código, que es el encargado de comprimir todo en un .exe, para así simplemente ejecutando este exe poder abrir la aplicación: **C:\Users\xxxx\pyinstaller --onefile --add-data "templates;templates" --noconsole app.py**

Funcionamiento:

Esta app web es capaz de conectarse a InfluxDB solamente ejecutando app.exe, y a partir de los datos obtenidos en app.py mostrarlos en diferentes gráficos y tablas.

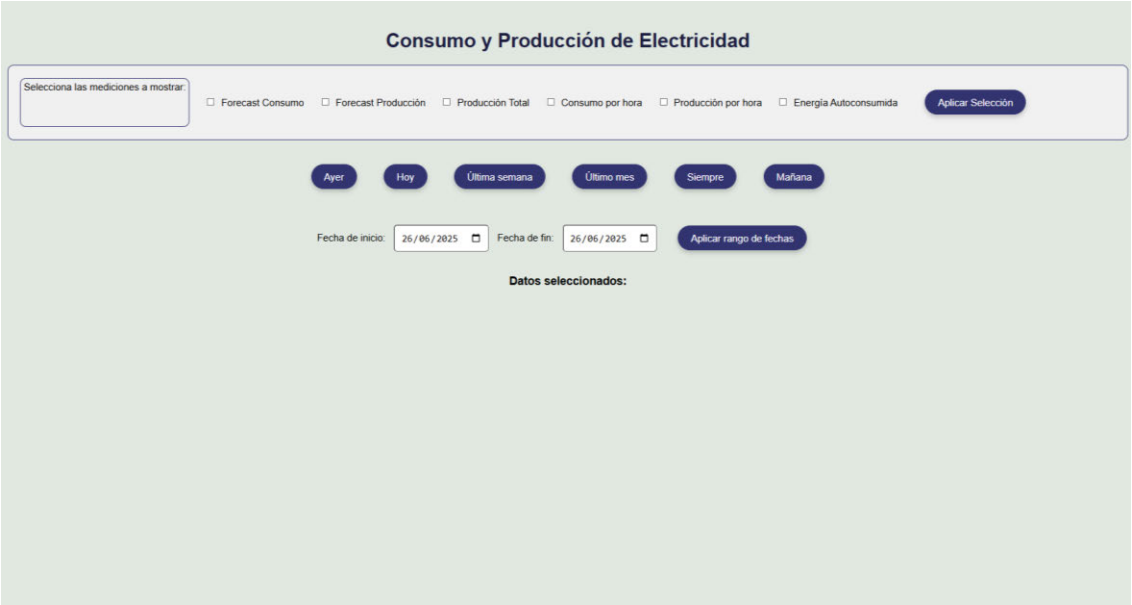


Figura 82: Captura de la aplicación nada más ingresar a ella.

Tiene varias partes, la primera es la selección de mediciones (Figura 83), donde se puede seleccionar que datos se quiere ver, es posible mostrar 1 solo dato o varios al mismo tiempo.

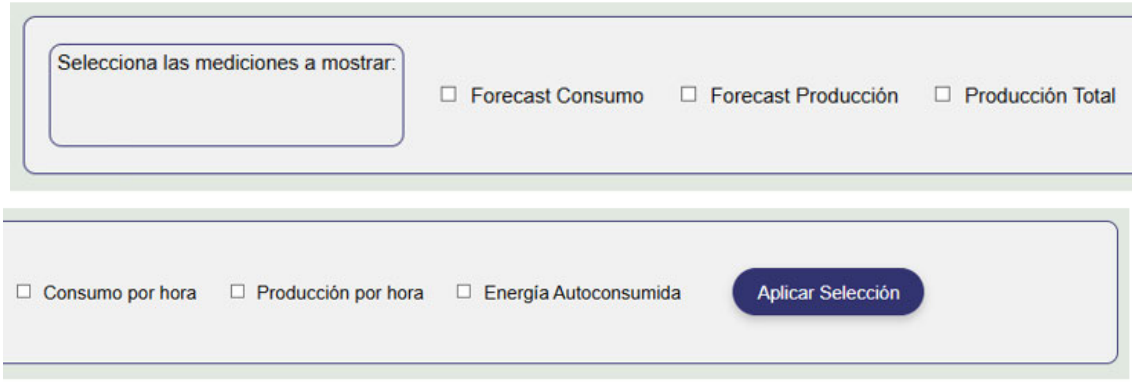


Figura 83: Selector de mediciones de la aplicación.

En la parte inferior se puede observar un grafo con los datos obtenidos de consumo en las últimas 24 h (Figura 84).

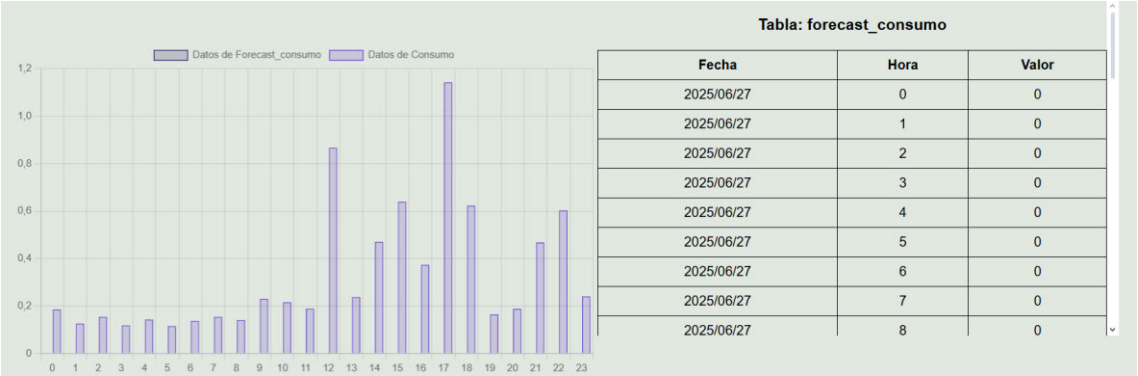


Figura 84: Gráfico con los datos obtenidos de consumo en las últimas 24 h.

A la derecha se ha creado una tabla para facilitar el análisis, esta tabla contiene los mismos datos que el gráfico. En la parte superior del grafo y la tabla, se pueden apreciar una serie de botones en los que se indica: Último día, Última semana, …. Con estos se puede seleccionar el rango de fechas que se quiere ver. Por otra parte, tiene una opción de fecha inicio y fecha fin, para seleccionar el rango de fechas deseado.

En la siguiente imagen (Figura 85) se puede ver el mismo dato (Consumo) pero con un rango de fecha de 1 semana, a la derecha está la tabla también con los datos de 1 semana.

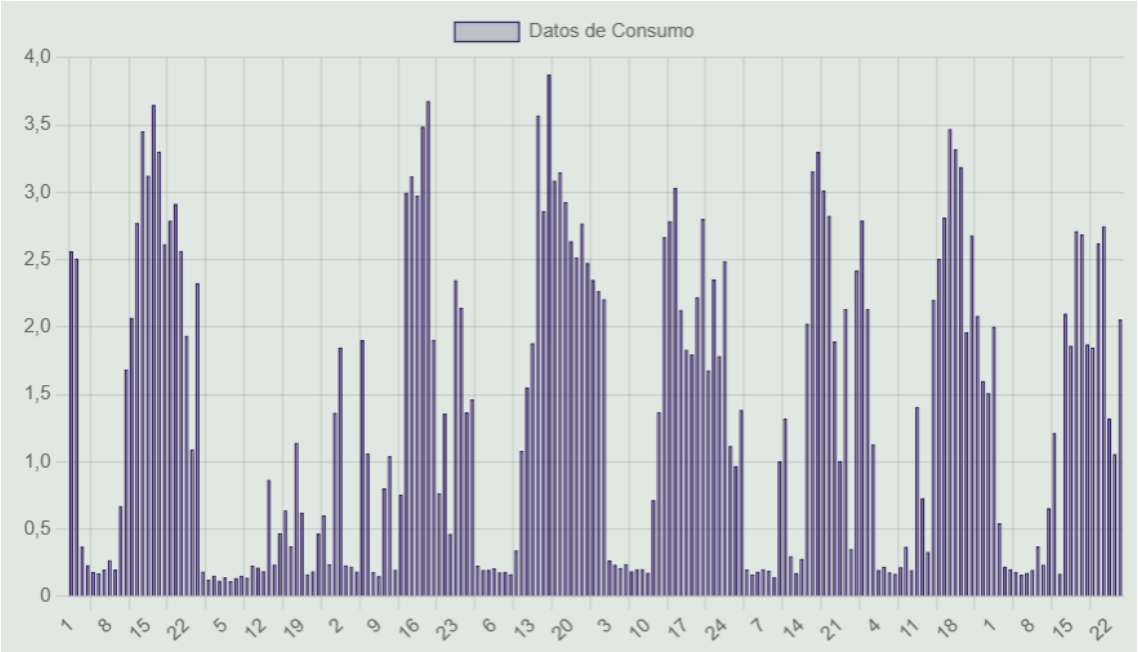


Figura 85: Captura de la aplicación mostrando el consumo por hora en un rango de una semana.

Ahora se puede ver que se ha seleccionado 2 datos, (Datos de consumo y Forecast de Consumo) Ambos para el día 21 (Figura 86).

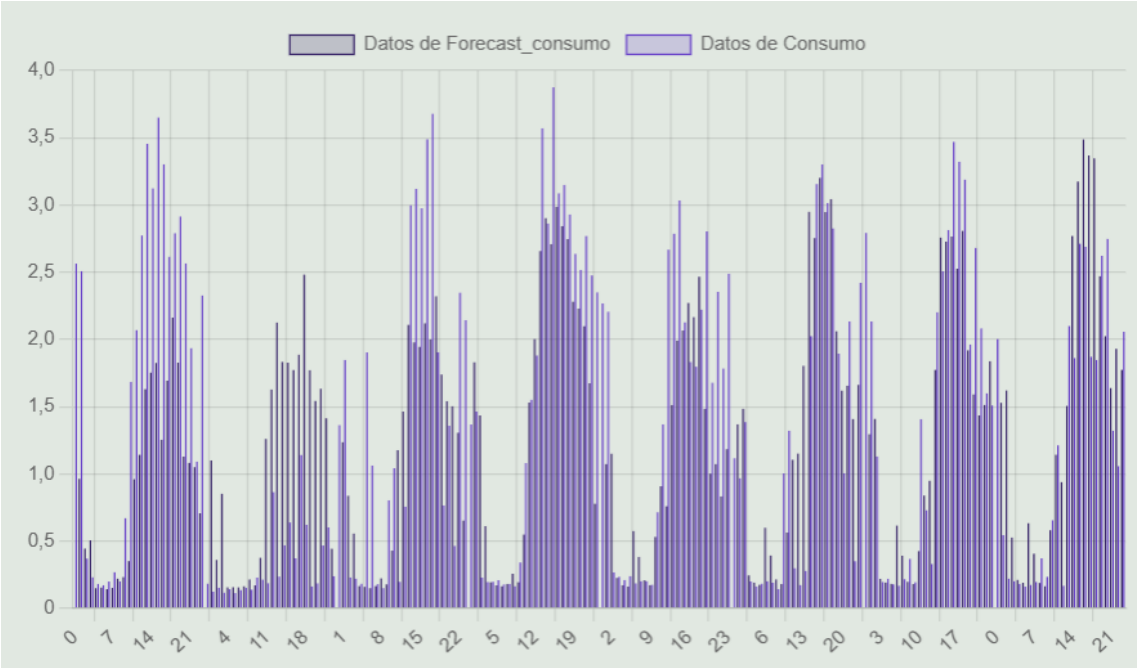


Figura 86: Captura de la aplicación mostrando consumo por hora junto a forecast de consumo.

Figura 87: Se observa que hay dos mediciones seleccionadas para la visualización.

Aquí se puede ver como en un solo grafico están ambas mediciones (Figura 88, 89):

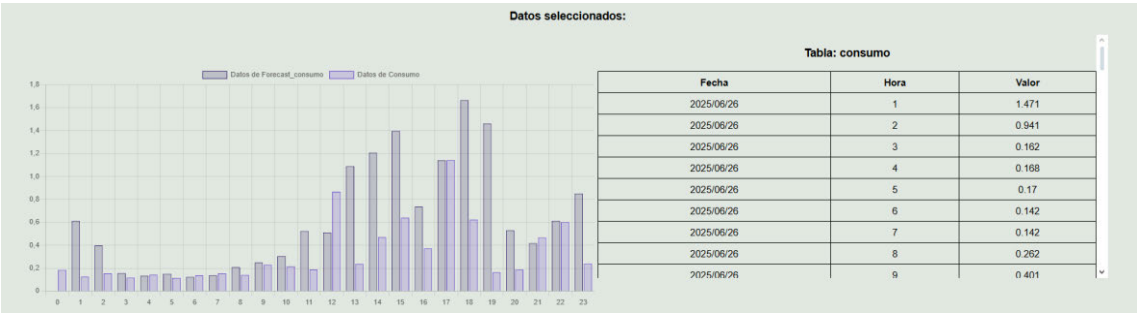


Figura 88: Captura donde se muestra a la derecha la tabla consumo.

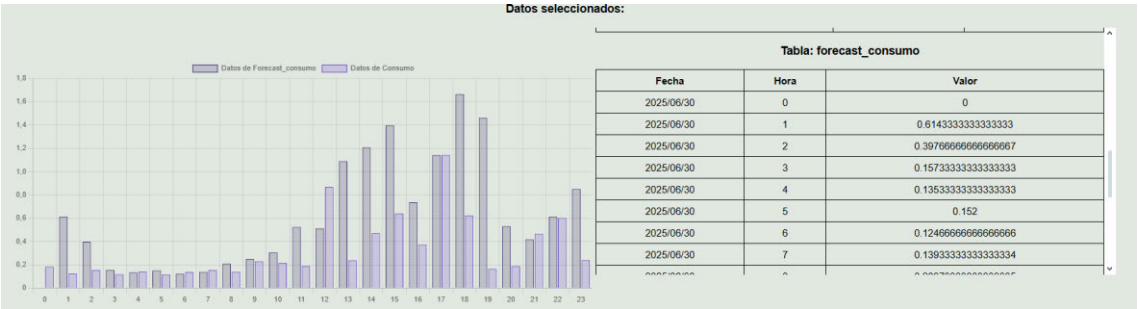


Figura 89: Captura donde se muestra a la derecha después de la tabla de consumo la de forecast consumo.

4. Pruebas y resultados

El sistema ha sido probado con datos reales tanto de consumo como de producción, lo que ha permitido verificar su precisión en el monitoreo y en las predicciones

En cuanto a las visualizaciones, los gráficos interactivos creados en la aplicación web permiten a los usuarios analizar los patrones de consumo y producción en función de su participación en el sistema FV. Los resultados mostraron que la herramienta era efectiva para visualizar datos en tiempo real, con la capacidad de poder ayudar a los usuarios a tomar decisiones informadas sobre su consumo energético y su participación en la comunidad energética.

Prueba de almacenamiento básico en la base de datos

- **Objetivo**: Verificar que los datos se están almacenando correctamente en InfluxDB.
- **Método**: Se inserta algunos datos de prueba en la base de datos. Luego, realiza una consulta básica para asegurarte de que los datos se han almacenado correctamente.

Para las pruebas se han creado variaciones del código real, estas nos permiten insertar datos preseleccionados.

InsertarConsumo(Prueba).py

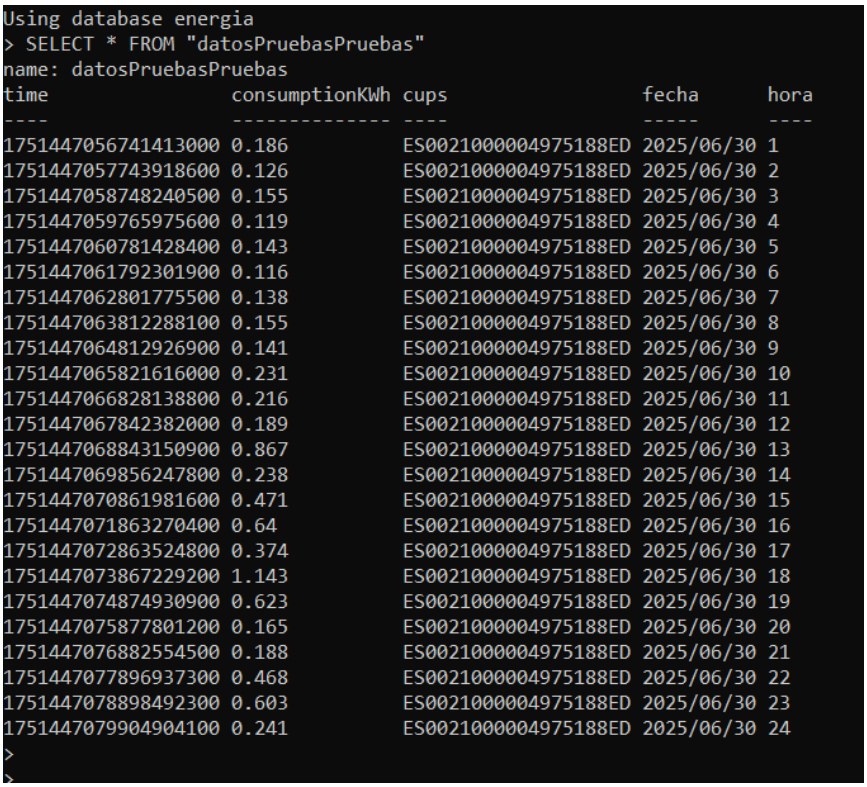
Para el caso de consumo se usa InsertarConsumo(Prueba).py un código que en base a unos datos en un documento data.txt los sube a Mosquitto para más tarde guardarlos en la base de datos.

Figura 90: Datos de consumo enviados a Mosquitto.

Como se observa en la imagen (Figura 90) los datos se envían correctamente a Mosquitto. Y podemos observar en Node-Red como los 24 registros han sido enviados correctamente (Figura 91).

Figura 91: Datos de consumo recibidos por Mosquitto.

Estos datos una vez han sido procesados por Node-Red se envían a InfluxDB, en concreto a un nuevo measurement que ha sido creado para las pruebas, este se llama “datosPruebasPruebas”:



```
Using database energia
> SELECT * FROM "datosPruebasPruebas"
name: datosPruebasPruebas
time                consumptionKWh cups                fecha      hora
----                -------------- ----                -----      ----
1751447056741413000 0.186          ES0021000004975188ED 2025/06/30 1
1751447057743918600 0.126          ES0021000004975188ED 2025/06/30 2
1751447058748240500 0.155          ES0021000004975188ED 2025/06/30 3
1751447059765975600 0.119          ES0021000004975188ED 2025/06/30 4
1751447060781428400 0.143          ES0021000004975188ED 2025/06/30 5
1751447061792301900 0.116          ES0021000004975188ED 2025/06/30 6
1751447062801775500 0.138          ES0021000004975188ED 2025/06/30 7
1751447063812288100 0.155          ES0021000004975188ED 2025/06/30 8
1751447064812926900 0.141          ES0021000004975188ED 2025/06/30 9
1751447065821616000 0.231          ES0021000004975188ED 2025/06/30 10
1751447066828138800 0.216          ES0021000004975188ED 2025/06/30 11
1751447067842382000 0.189          ES0021000004975188ED 2025/06/30 12
1751447068843150900 0.867          ES0021000004975188ED 2025/06/30 13
1751447069856247800 0.238          ES0021000004975188ED 2025/06/30 14
1751447070861981600 0.471          ES0021000004975188ED 2025/06/30 15
1751447071863270400 0.64           ES0021000004975188ED 2025/06/30 16
1751447072863524800 0.374          ES0021000004975188ED 2025/06/30 17
1751447073867229200 1.143          ES0021000004975188ED 2025/06/30 18
1751447074874930900 0.623          ES0021000004975188ED 2025/06/30 19
1751447075877801200 0.165          ES0021000004975188ED 2025/06/30 20
1751447076882554500 0.188          ES0021000004975188ED 2025/06/30 21
1751447077896937300 0.468          ES0021000004975188ED 2025/06/30 22
1751447078898492300 0.603          ES0021000004975188ED 2025/06/30 23
1751447079904904100 0.241          ES0021000004975188ED 2025/06/30 24
>
>
```

Figura 92: Datos de consumo guardados en InfluxDB.

Como se puede observar en la imagen (Figura 92) todos los datos para las 24 horas del día se han subido a nuestra base de datos sin ningún problema.

InsertarProduccion(Prueba).py

Para el caso de producción se usa InsertarProduccion(Prueba).py que en base a unos datos ya insertados en el propio código los sube directamente a InfluxDB sin pasar por Mosquitto:

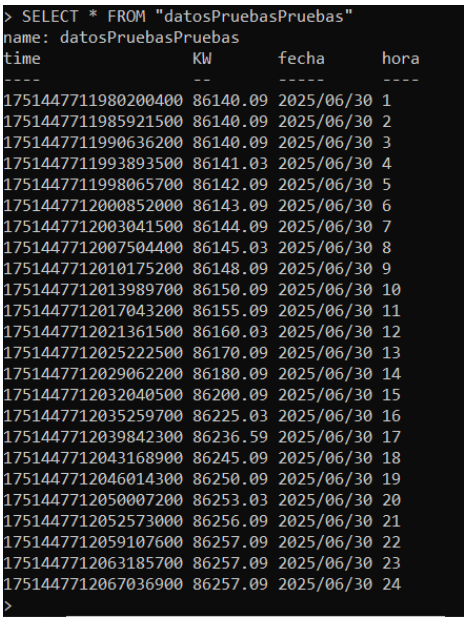
```
registros_solar = [
    {"hora": "1", "number": 86140.09},
    {"hora": "2", "number": 86140.09},
    {"hora": "3", "number": 86140.09},
    {"hora": "4", "number": 86141.03},
    {"hora": "5", "number": 86142.09},
    {"hora": "6", "number": 86143.09},
    {"hora": "7", "number": 86144.09},
    {"hora": "8", "number": 86145.03},
    {"hora": "9", "number": 86148.09},
    {"hora": "10", "number": 86150.09},
    {"hora": "11", "number": 86155.09},
    {"hora": "12", "number": 86160.03},
    {"hora": "13", "number": 86170.09},
    {"hora": "14", "number": 86180.09},
    {"hora": "15", "number": 86200.09},
    {"hora": "16", "number": 86225.03},
    {"hora": "17", "number": 86236.59},
    {"hora": "18", "number": 86245.09},
    {"hora": "19", "number": 86250.09},
    {"hora": "20", "number": 86253.03},
    {"hora": "21", "number": 86256.09},
    {"hora": "22", "number": 86257.09},
    {"hora": "23", "number": 86257.09},
    {"hora": "24", "number": 86257.09},
]
```

```
C:\Users\diego\Documents\tfg\Definitivo>python InsertarProduccion(Prueba).py
Introduce la fecha (formato YYYY/MM/DD): 2025/06/30
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '1', 'KW': 86140.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '2', 'KW': 86140.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '3', 'KW': 86140.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '4', 'KW': 86141.03}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '5', 'KW': 86142.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '6', 'KW': 86143.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '7', 'KW': 86144.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '8', 'KW': 86145.03}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '9', 'KW': 86148.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '10', 'KW': 86150.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '11', 'KW': 86155.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '12', 'KW': 86160.03}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '13', 'KW': 86170.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '14', 'KW': 86180.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '15', 'KW': 86200.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '16', 'KW': 86225.03}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '17', 'KW': 86236.59}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '18', 'KW': 86245.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '19', 'KW': 86250.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '20', 'KW': 86253.03}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '21', 'KW': 86256.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '22', 'KW': 86257.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '23', 'KW': 86257.09}
Inserting into datosPruebas2: {'fecha': '2025/06/30', 'hora': '24', 'KW': 86257.09}
```

Figura 93: Datos de producción enviados a InfluxDB.

Como se observa en la imagen (Figura 93) los datos se envían correctamente a InfluxDB.

Estos datos ya procesados se envían a InfluxDB, en concreto a un nuevo measurement que ha sido creado para las pruebas, este se llama “datosPruebasPruebas”:



```
> SELECT * FROM "datosPruebasPruebas"
name: datosPruebasPruebas
time                KW       fecha      hora
----                --       -----      ----
1751447711980200400 86140.09 2025/06/30 1
1751447711985921500 86140.09 2025/06/30 2
1751447711990636200 86140.09 2025/06/30 3
1751447711993893500 86141.03 2025/06/30 4
1751447711998065700 86142.09 2025/06/30 5
1751447712000852000 86143.09 2025/06/30 6
1751447712003041500 86144.09 2025/06/30 7
1751447712007504400 86145.03 2025/06/30 8
1751447712010175200 86148.09 2025/06/30 9
1751447712013989700 86150.09 2025/06/30 10
1751447712017043200 86155.09 2025/06/30 11
1751447712021361500 86160.03 2025/06/30 12
1751447712025222500 86170.09 2025/06/30 13
1751447712029062200 86180.09 2025/06/30 14
1751447712032040500 86200.09 2025/06/30 15
1751447712035259700 86225.03 2025/06/30 16
1751447712039842300 86236.59 2025/06/30 17
1751447712043168900 86245.09 2025/06/30 18
1751447712046014300 86250.09 2025/06/30 19
1751447712050007200 86253.03 2025/06/30 20
1751447712052573000 86256.09 2025/06/30 21
1751447712059107600 86257.09 2025/06/30 22
1751447712063185700 86257.09 2025/06/30 23
1751447712067036900 86257.09 2025/06/30 24
>
```

Figura 94: Datos de producción guardados en InfluxDB.

Como se puede observar en la imagen (Figura 94) todos los datos para las 24 horas del día se han subido a la base de datos sin ningún problema.

Prueba de tiempo de carga de la aplicación

- **Objetivo**: Medir cuánto tiempo tarda la aplicación en cargar y mostrar los datos.
- **Método**: Se realiza una prueba de inserción y se realiza una consulta a través de la aplicación web a la base de datos.

La prueba ha sido realizada desde un ordenador portátil con una red wifi, es posible que los resultados de tiempos de carga varíen en base a la potencia de la red y del ordenador.

Según la prueba realizada, el tiempo de inserción de datos en la base de datos es casi inmediato. Para la prueba se han subido datos de producción una vez más al measurement “datosPruebasPruebas” y ha tardado menos de 1 segundo en recibir los datos y mostrarlos por pantalla.

En cuanto al tiempo de carga de la aplicación, desde que se ejecuta el app.exe hasta que se abre en el explorador y carga la aplicación tarda 4´43 segundos.

Hay que aclarar que, aunque la aplicación este activa, el servidor de InfluxDB aún no, ya que tarda exactamente otros 5´45 segundos extra después de haberse iniciado la aplicación.

El total en segundos desde la ejecución de app.exe hasta visualización de datos en la aplicación web es de: 9´88 segundos

Prueba de desconexión temporal de servicios

- **Objetivo**: Verificar cómo se comporta el sistema si alguno de los componentes (como InfluxDB, Mosquitto o Node-Red) se desconecta temporalmente.
- **Método**: Se desconectan los servicios y se observa si el sistema se comporta correctamente. Luego, vuelve a conectar el servicio y comprueba si se reanudan las operaciones correctamente.

Se ha realizado 3 fases para comprobar el funcionamiento de la app:

1. Desconexión de Mosquitto:

Al desconectar el servidor de Mosquitto mientras la aplicación web esta activa, no ocurre ningún error, ya que la aplicación depende solamente del servidor de InfluxDB para funcionar.
Por lo tanto, la aplicación es capaz de seguir mostrando los datos sin problema alguno.
Por otra parte, el servidor de Mosquitto es esencial para la obtención de datos por parte del consumidor y de los productores, por ello los datos obtenidos mientras el servidor está apagado, no se guardarán ni reflejarán en la base de datos InfluxDB ni en la aplicación web.

2. Desconexión de Node-Red:

Al igual que en el apartado anterior, si desconectamos el servidor Node-Red, la aplicación no mostrará ningún cambio ni visualmente ni en su comportamiento ya que no depende de Node-Red.
El problema que surgiría es que, durante el tiempo que Node-Red este apagado ningún dato obtenido desde Mosquitto podrá ser guardado en nuestra base de datos de InfluxDB.
Por tanto, los datos que se intenten insertar durante el tiempo que Node-Red no este activo no se reflejaran en la aplicación web.

3. Desconexión de InfluxDB:

En este caso al desconectar el servidor de InfluxDB, si ocurriría un error en la aplicación web (Figura 95). Esto se debe a que la aplicación web obtiene los datos directamente de la base de datos InfluxDB.
Por otra parte, los datos que se intenten guardar en la base de datos en el tiempo en el que este apagado el servidor, no se guardarán.

Figura 95: Captura de la aplicación mostrando error de consulta.

Una vez que se enciende de nuevo el servidor de InfluxDB y se realiza una consulta desde nuestra aplicación web, todo vuelve a funcionar correctamente (Figura 96), tanto la visualización de los datos como el guardado de los datos.

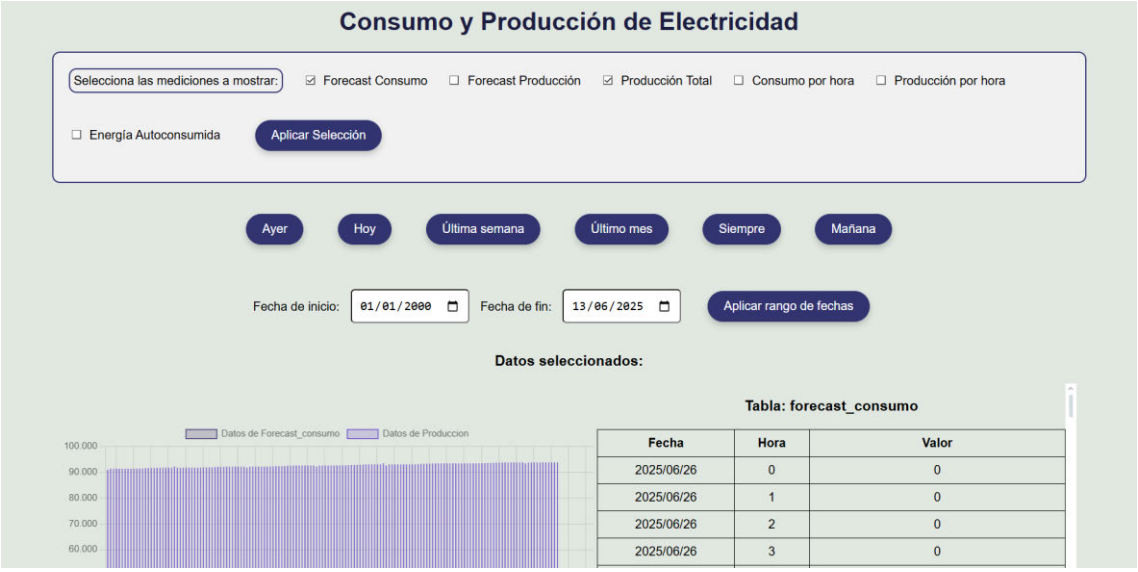


Figura 96: Captura de la aplicación con el error resuelto.

Prueba de visualización en diferentes exploradores web

- **Objetivo**: Comprobar que la visualización de los datos (gráficos, tablas, etc.) se adapta bien a diferentes exploradores web.
- **Método**: Se abre la aplicación web en distintos exploradores, para la prueba se ha probado en OperaGX, Google Chrome, Firefox y Microsoft Edge.

1. **OperaGX:**
   Se puede observar el correcto funcionamiento de la aplicación web en OperaGX (Figura 97).

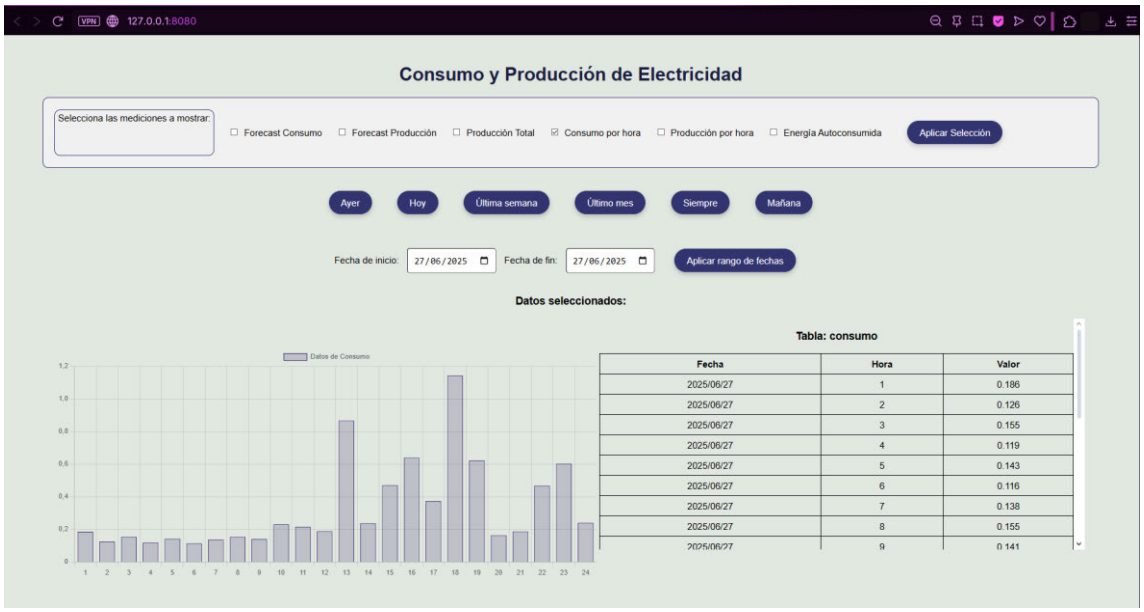


Figura 97: Captura de la aplicación usando OperaGX.

2. **Google Chrome:**
   Se puede observar el correcto funcionamiento de la aplicación web en Google Chrome (Figura 98).

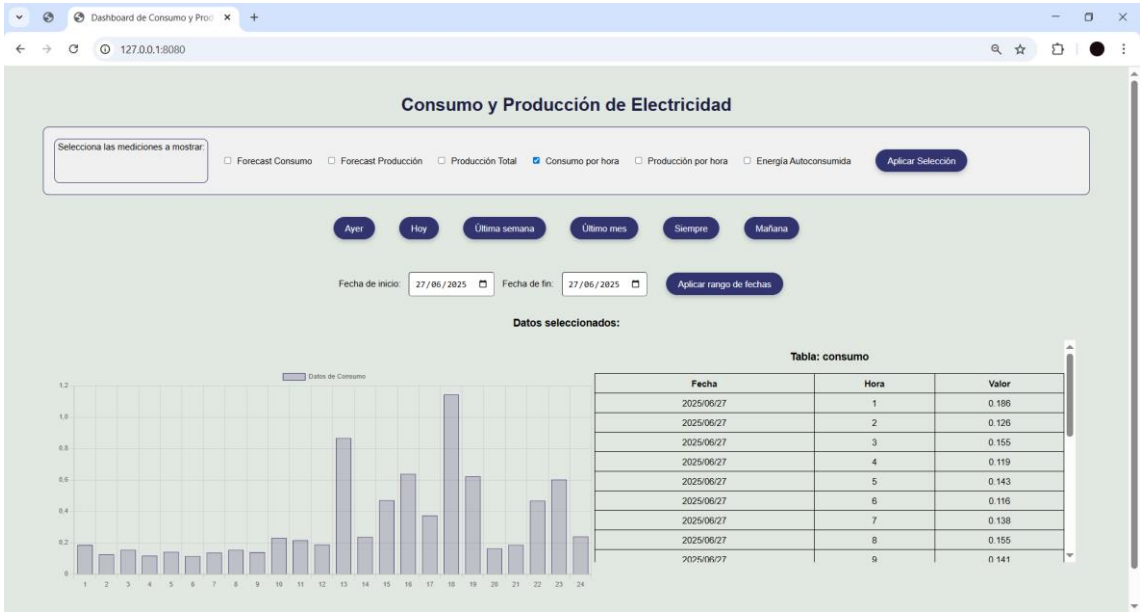


Figura 98: Captura de la aplicación usando Google Chrome.

3. **Firefox:**

Se puede observar el correcto funcionamiento de la aplicación web en Firefox (Figura 99).

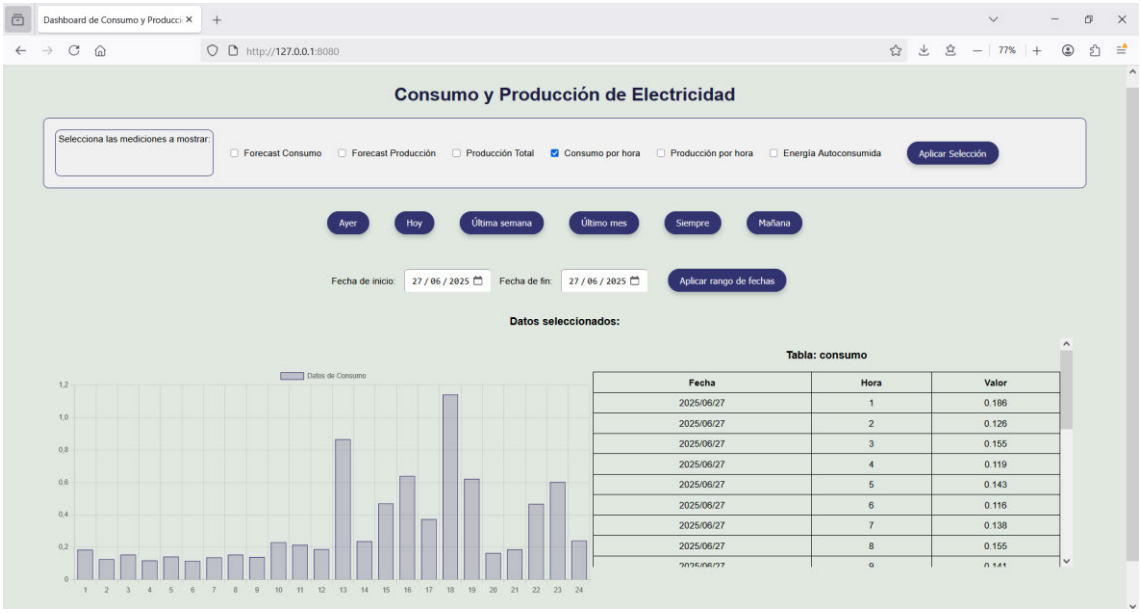


Figura 99: Captura de la aplicación usando Firefox.

4. **Microsoft Edge:**
   Se puede observar el correcto funcionamiento de la aplicación web en Microsoft Edge (Figura 100).

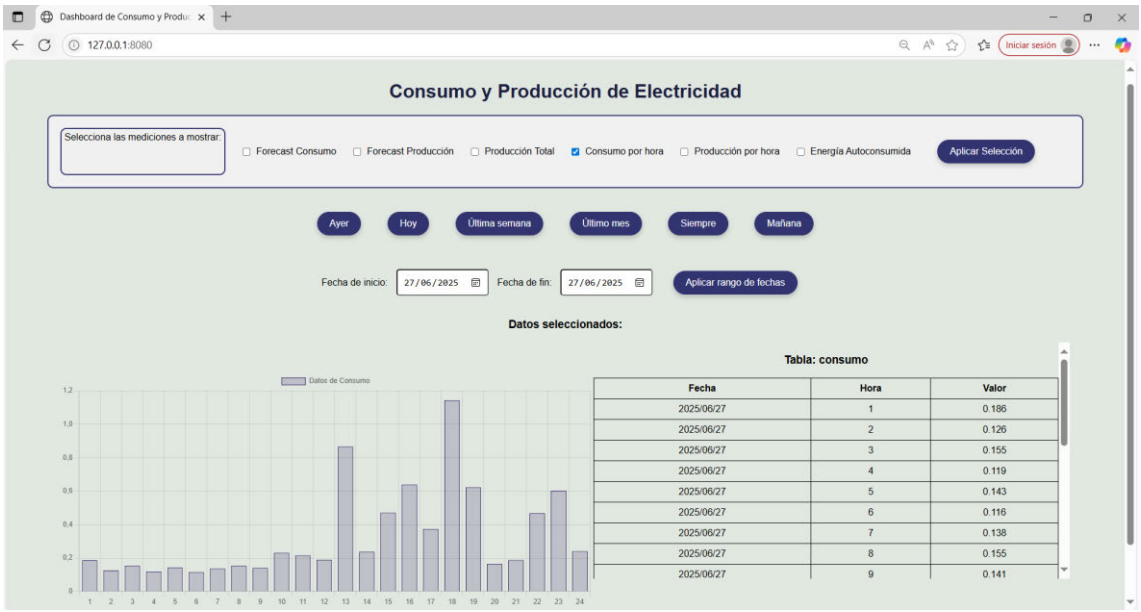


Figura 100: Captura de la aplicación usando Microsoft Edge.

Conclusión: Las pruebas en distintos exploradores, nos indican que tanto la aplicación como los distintos servicios, funcionan correctamente en todos ellos.

Prueba de test de wilcoxon entre datos

La prueba de Wilcoxon (Prueba de los rangos con signo de Wilcoxon) determina si dos grupos dependientes difieren significativamente entre sí. Para ello, el test de Wilcoxon utiliza los rangos de los grupos en lugar de los valores medios [39] [40] [41].

- **Objetivo**: Comprobar que los datos obtenidos para la predicción de consumo y producción tienen una precisión aceptable.
- **Método**: Se ha realizado test de wilcoxon para probar la eficacia tanto del forecast de consumo como el de producción, los resultados obtenidos son los siguientes.

Producción / Predicción de producción:

Resultados:

• **W-value**: 47
• **Mean Difference**: 25.13
• **Sum of positive ranks**: 89
• **Sum of negative ranks**: 47
• **Z-value**: -1.0859
• **p-value**: 0.27572
• **Critical value for W at N = 16 (p < 0.05)**: 29

Explicación breve:

• **W-value (47)** es mayor que el valor crítico (29), lo que indica que **no hay una diferencia significativa** entre las muestras.
• **Z-value (-1.0859)** con un **p-value de 0.27572** muestra que la diferencia **no es estadísticamente significativa**.
• **Mean Difference (25.13)** sugiere una **gran diferencia media** entre las muestras, pero el valor de W y Z indican que **no es relevante estadísticamente**.

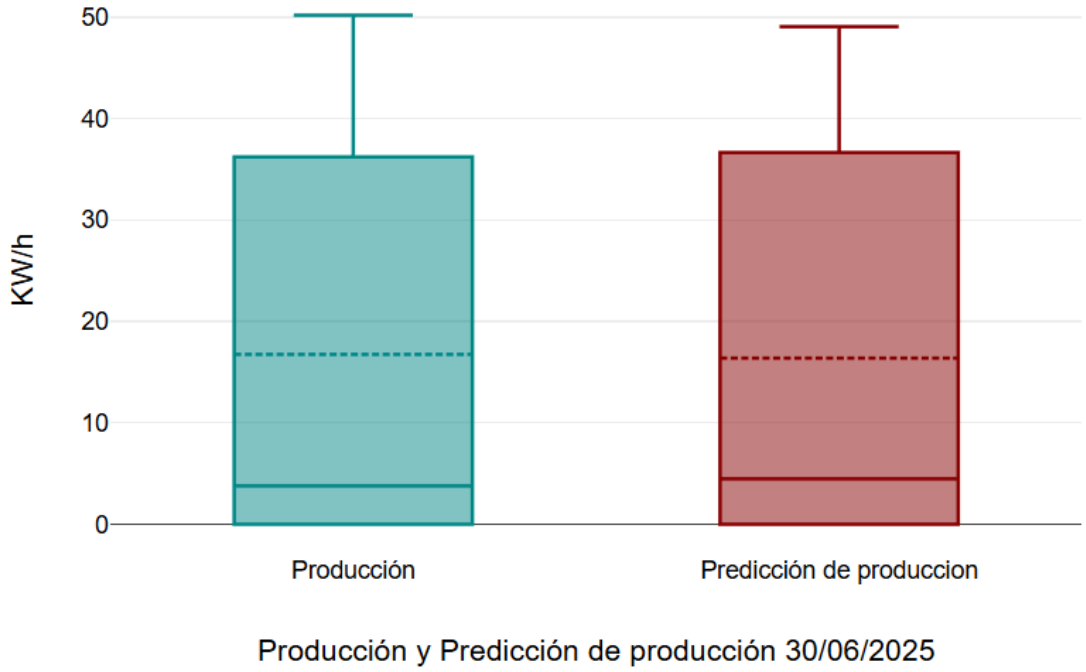


Figura 101: Imagen de prueba de wilcoxon

Conclusión:

Los resultados del test de Wilcoxon indican que no hay una diferencia estadísticamente significativa entre las predicciones y los valores reales. A pesar de que la diferencia media entre las muestras es considerable (25.13), el p-value (0.27572) y el W-value (47, mayor que el valor crítico de 29) sugieren que la diferencia observada podría deberse al azar. Por lo tanto, no se puede rechazar la hipótesis nula, lo que implica que las predicciones no son significativamente diferentes de los valores reales en este caso.

Consumo / Predicción de consumo:

Resultados:

- **W-value**: 49
- **Mean Difference**: -0.28
- **Sum of positive ranks**: 49
- **Sum of negative ranks**: 251
- **Z-value**: -2.8857
- **p-value**: 0.00386
- **Critical value for W at N = 24 (p < 0.05)**: 81

Explicación breve:

- **W-value** (49) es menor que el valor crítico (81), indicando una diferencia significativa.
- **Z-value** (-2.8857) con un **p-value** de 0.00386 muestra que la diferencia es estadísticamente significativa.
- **Mean Difference** (-0.28) sugiere que las predicciones tienden a ser mayores que los valores reales.

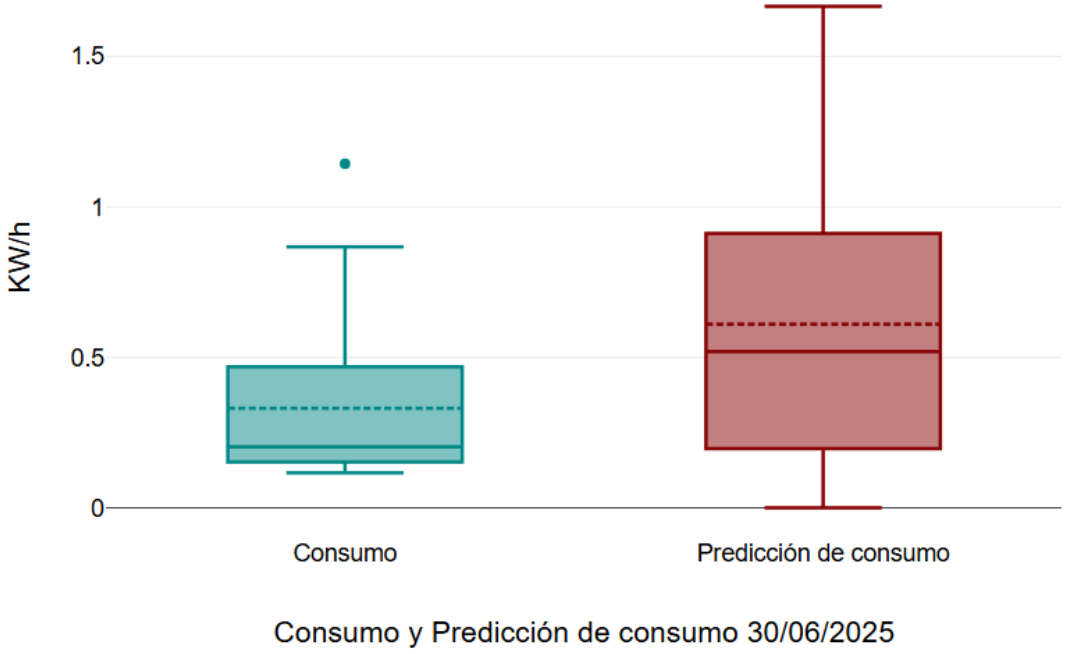


Figura 102: Imagen de prueba de wilcoxon

Conclusión:

Los resultados del test de Wilcoxon muestran que las predicciones son significativamente diferentes de los valores reales. El W-value (49) es menor que el valor crítico (81), y el p-value (0.00386) es menor que 0.05, lo que indica que la diferencia observada es estadísticamente significativa. Además, el Z-value negativo (-2.8857) y la Mean Difference (-0.28) sugieren que las predicciones tienden a ser mayores que los valores reales.

Por lo tanto, podemos concluir que las predicciones no son iguales a los valores reales y hay una diferencia estadística relevante entre ellos. Sin embargo, es importante señalar que las predicciones no son exactas, ya que el consumo de energía puede variar significativamente entre un día y otro debido a muchos factores variables, como las condiciones meteorológicas, el comportamiento del consumidor o el tipo de actividad realizada, lo que introduce una incertidumbre inherente en las estimaciones

de consumo. Además, es posible la implementación de algoritmos adicionales basados en otros supuestos, como las condiciones meteorológicas específicas del día o utilizar datos de forecast de organismos externos, como la Red Eléctrica Española, lo que podría mejorar la precisión de las predicciones al considerar información más detallada y relevante sobre el comportamiento de la red y la demanda energética.

5. Lecciones aprendidas sobre las herramientas empleadas durante el desarrollo del proyecto.

A lo largo del desarrollo del proyecto, se han encontrado varios problemas y dificultades en su avance, que se han ido subsanando a medida que se progresaba en su desarrollo. Varios de estos problemas han estado relacionados al código y flujos de datos desarrollados por mi parte, pero también ha habido algunos problemas con el uso de las herramientas y sistemas empleados.

Mosquitto:

En Mosquitto no se ha tenido ningún problema ni error durante el proyecto, sigue un flujo sencillo, en el que publicas un mensaje y si estas subscrito a el tema en el que se ha enviado el mensaje, te llega.

Node-Red:

En Node-Red se ha tenido varios problemas de formato principalmente. Esto es un problema crítico, ya que, se necesita una estructura limpia y ordenada en la base de datos para su correcto funcionamiento.

1. **Problema**: El primer problema importante con el que nos hemos topado usando Node-Red, ha sido al pasar datos hacia InfluxDB, el problema ha sido que, una vez guardados los datos en la base de datos, al hacer una comprobación de estos, se podía observar que faltaban algunos valores que no habían sido guardados

   Solución: Para solucionar esto, después de varias pruebas se ha decidido subir los datos a InfluxDB de uno en uno, ya que, al intentar subir los 24 valores de golpe, algunos datos se perdían.
   Para ello se ha usado un **for (let i = 0; i < datos.length; i++) {** el cual al final de cada iteración tenía **node.send(mensajes[i]);** por lo que se enviaba un mensaje solamente por cada iteración.

2. **Problema:** Al subir los datos a InfluxDB la hora se subía como un string con el siguiente formato “hh:mm”, por tanto, en la creación de grafos tanto en Grafana como en nuestra aplicación web no era posible ordenar los datos por hora ya que no es posible ordenar strings numeralmente.

Solución: Se ha decidido que la mejor opción es guardar las horas como un int, es decir si son las **16:00** pm se guardara como un **16,** para ello se ha usado **let hora = parseInt(hora1.split(":")[0]);** el cual divide el string obtenido quedándose solamente con la parte antes del **:** y guardándolo como un int.

3. **Problema:** Se ha tenido problemas al estructurar los datos recibidos de Mosquitto en Node-Red, se ha tenido que hacer una estructura común para todos los datos para que así sean comparables entre sí.

   Solución: Se ha decidido crear una serie de parámetros comunes, la fecha y la hora. La fecha con un formato Date (yyyy/mm/dd) y la hora con un formato int de una cifra. Para realizar esta estructura se han enviado los datos a InfluxDB así:

```
mensajes.push({
        payload: {
          consumptionKWh: consumo,
          cups: d.cups,
          hora,
          fecha
        }
    });
```

InfluxDB:

En InfluxDB solamente se ha tenido un problema relacionado a abrir el servidor:

1. **Problema**: Una vez iniciado el servidor, al intentar abrir la base de datos usando el comando **.\influx.exe** nos ha salido error (Figura 103) diciendo que no se ha encontrado el host para localhost:8086. Lo cual no tiene sentido ya que el servidor local ha sido iniciado con anterioridad.

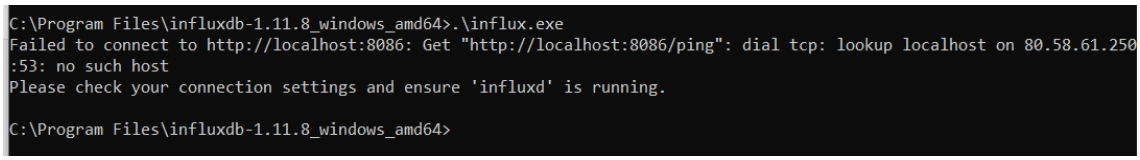


Figura 103: Fallo en la entrada a la base de datos.

   Solución: Para solucionarlo se ha buscado información sobre este error en internet y como resultado se ha usado el comando **.\influx.exe -host 127.0.0.1** Con este comando se nos ha permitido ingresar en la base de datos de forma correcta (Figura 104).

```
C:\Program Files\influxdb-1.11.8_windows_amd64>.\influx.exe
Failed to connect to http://localhost:8086: Get "http://localhost:8086/ping": dial tcp: lookup localhost on 80.58.61.250
:53: no such host
Please check your connection settings and ensure 'influxd' is running.

C:\Program Files\influxdb-1.11.8_windows_amd64>.\influx.exe -host 127.0.0.1
Connected to http://127.0.0.1:8086 version v1.11.8
InfluxDB shell version: v1.11.8
>
```

Figura 104: Se permite la entrada a la base de datos.

Grafana:

El único problema con Grafana ha sido al abrir el servicio.

1. **Problema**: Grafana además de tener un servidor local como el que se ha usado en la práctica, también tiene una página web (Figura 105). Al principio del proyecto se intentó conectar la base de datos local de InfluxDB a la página web de Grafana sin éxito.

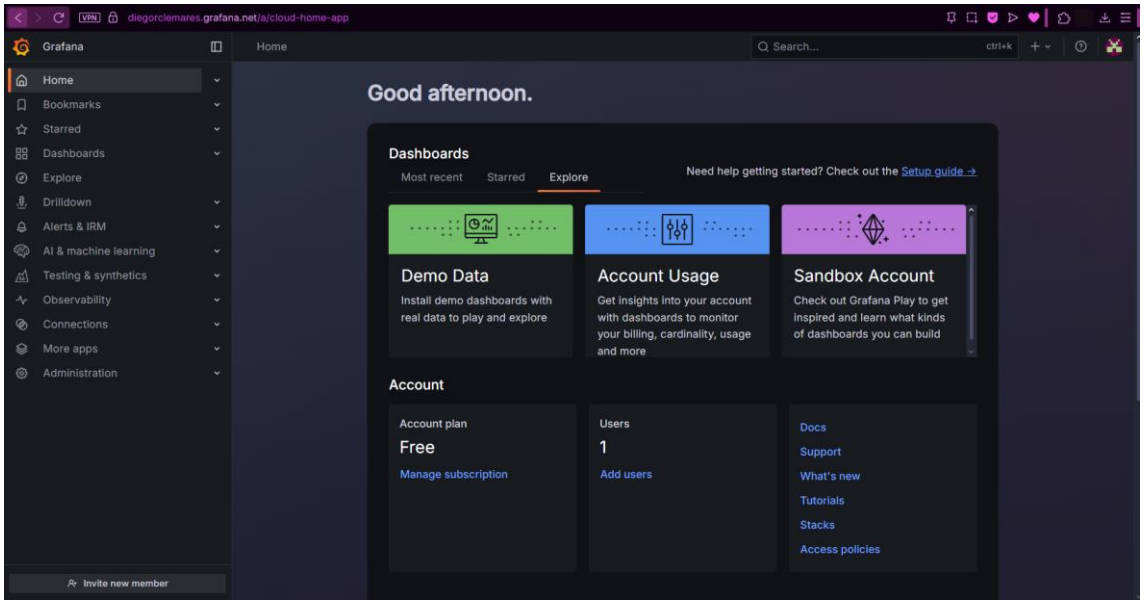


Figura 105: Pagina web de Grafana.

Solución: Después de investigar en internet y ver varios vídeos de como conectar InfluxDB y Grafana, entendí que para conectar la base de datos local con Grafana, no se podía hacer a través de la página web, si no que había que descargarse los archivos necesarios, para abrir un servidor local de Grafana y ejecutar **grafana.exe server** en el cmd.

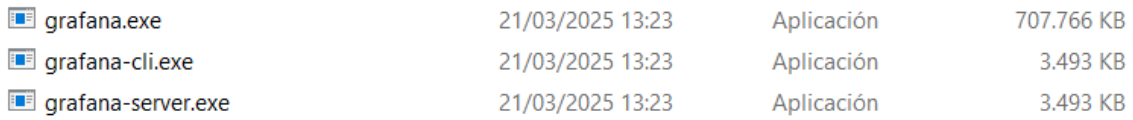


Figura 106: Archivos descargados para iniciar Grafana.

Datadis:

Con Datadis se han tenido varios problemas, aunque no todos ellos han tenido solución.

1. **Problema**: El primer problema que se ha detectado usando la API de Datadis es que solo permite una consulta diaria, es decir, que no es posible en caso de un error en el guardado de datos volver a pedir los datos a Datadis.

   Solución: Aunque este problema no ha tenido una solución efectiva, ya que se sale de nuestro control, se ha creado un sistema robusto que es capaz de recibir todos los datos de Datadis sin perder ninguno. De esta forma, aunque no se soluciona el problema en caso de un error del sistema, si se puede disminuir la probabilidad de error.

2. **Problema**: El segundo problema que se ha tenido con Datadis es que no sube los datos de consumo del día anterior a una hora consistente, es decir, un día puede ser a las 5 am y otro a las 9 am.

   Solución: Después de realizar varias pruebas se ha llegado a la conclusión de que, para no tener problemas en la obtención de datos, el mejor momento para realizar la consulta y obtener los datos de consumo es a las 10:05 am. De esta forma nos aseguramos de que los datos que recibimos sean los del ayer y no los de antes de ayer.

```
# Función para esperar hasta medianoche y luego ejecutar los scripts
def esperar_hasta_medianoche():
    now = datetime.now()
    # Calcular el tiempo que falta hasta las 12:05 PM del día actual o el siguiente
    twelve_five_pm = datetime.combine(now.date(), datetime.min.time()) + timedelta(hours=10,minutes=5)

    # Si ya ha pasado las 12:05 PM hoy, espera hasta las 12:05 PM del día siguiente
    if now > twelve_five_pm:
        twelve_five_pm += timedelta(days=1)

    time_to_wait = (twelve_five_pm - now).total_seconds()
    print(f"Esperando hasta las 12:05 PM... Tiempo restante: {time_to_wait} segundos")
    time.sleep(time_to_wait)
```

Figura 107: Función para esperar hasta las 10:05 am.

Código:

En los distintos códigos que se han creado se han tenido decenas de pequeños errores, entre ellos se pueden destacar los siguientes:

1. **Problema**: Es necesario un orden en la ejecución de los códigos ya que, por ejemplo, no podemos saber cuánta energía se ha auto consumido, sin antes saber cuánta se ha consumido en un día.

Solución: Se han ordenado los códigos según los requisitos de cada uno, el orden es el siguiente: Primero se obtiene la energía consumida por hora, luego en base a la producción total se obtiene KW/h, en base a los dos datos anteriores se obtiene la energía auto consumida, luego se realiza el forecast de consumo y producción para el día siguiente.

```
def ejecutar_diariamente():
    while True:
        # Esperar hasta medianoche
        esperar_hasta_medianoche()

        # Ejecutar los scripts a las 00:00
        ejecutar_obtener_diario()
        time.sleep(25)  # Puede agregar un pequeño retraso si es necesario entre los scripts
        ejecutar_kwh()
        time.sleep(5)
        ejecutar_consumo_ahorro()
        time.sleep(5)
        ejecutar_convertir_a_dolar()
        time.sleep(5)
        ejecutar_forecast_consumo()
        time.sleep(5)
        ejecutar_forecast_produccion()
```

Figura 108: Función que ejecuta los códigos en orden.

2. **Problema**: Después de haber hecho varias pruebas para acercarnos lo máximo posible, a una predicción del consumo del día siguiente, se ha observado gracias a el test de Wilcoxon que los datos de predicción no son todo lo precisos que deberían.

   Solución: Para este apartado no se ha logrado una solución, el problema principal que se ha detectado, es que el consumo puede variar drásticamente dependiendo del día, por lo que es difícil hacer una predicción, por ejemplo, si un día usamos el aire acondicionado a las 4 pm y al día siguiente no, la diferencia de KW será enorme entre las 4 pm de un día y del otro.

3. **Problema**: Al recopilar datos de producción, usando el código que se nos ha proporcionado desde Huawei, se ha observado que se producen errores entre las 0:00 y 2:00 am, este problema es que se recibe un valor de energía acumulado menor que el anterior (Lo cual no es posible), en la siguiente imagen se puede observar el error (Figura 109):

Figura 109: Captura donde se ve el error de producción.

Solución: No existe una solución ya que son datos que obtenemos de Huawei por lo tanto no sabemos a que se debe ese error. Se ha planteado poner a 0 el valor que sobresale, pero se ha decidido dejar los datos tal y como se obtienen, ya que, de esta forma Huawei puede ver el momento en el que su sitema detecta un valor incorrecto.

Aplicación

En la creación de la aplicación no ha habido demasiadas complicaciones.

1. **Problema**: Se necesitaba poder controlar el rango de días en los que se mostraban los datos en los gráficos, ya que, al mostrar todos los datos que hay en la base de datos sin segmentar no es legible como se puede ver en la imagen (Figura 110):

Figura 110: Captura donde se ven datos sin fecha límite.

Solución: Para solucionar este problema se han creado tanto unos botones para seleccionar “Ayer” “Hoy” “Mañana” …. También se ha creado una fecha de inicio y una fecha de fin editables, con esto podemos seleccionar el rango en el que queremos mostrar nuestros datos (Figura 111):

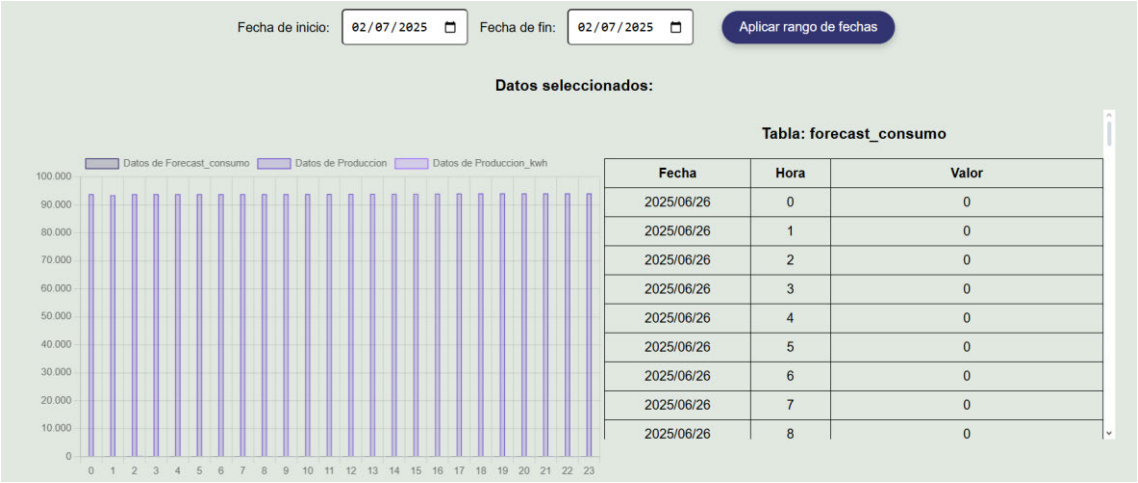


Figura 111: Captura donde se ven datos con fecha límite.

6. Impacto social y medioambiental

Las comunidades energéticas han surgido como un elemento esencial para impulsar la participación ciudadana en la transición ecológica, un paso necesario con urgencia para disminuir la dependencia a fuentes de energía no renovables y favorecer el uso de energías limpias. El contexto de este proyecto se sitúa dentro de esta tendencia, proporcionando una herramienta tecnológica que no solo hace más sencillo el monitoreo de la energía en comunidades energéticas que disponen de sistemas FV de autoconsumo colectivo, sino que también, mejora el nivel de autosuficiencia de los usuarios al permitirles manejar los datos de su propio consumo y producción energética con eficacia.

El impacto social de este proyecto, radica en que las comunidades energéticas impulsan un modelo de autonomía energética, donde la ciudadanía, las pequeñas empresas y las autoridades locales gestionan su energía, todo en conjunto. Este modelo no solo mejora la cohesión social, ya que promueve la colaboración entre los miembros de la comunidad, también combate la pobreza energética. Al usar la energía solar para el autoconsumo colectivo, los usuarios pueden disminuir su dependencia hacia las grandes empresas eléctricas, y así, reducir sus costos de energía.

Este proyecto tiene, además, impacto social directo, puesto que la herramienta de monitorización propuesta puede ser implementada en la comunidad energética CERCA de la comarca de Calatayud, puesto que, como se ha mencionado, surge de un proyecto europeo liderado por la UPM y que ha servido de merco para la realización de este TFG.

Ambientalmente hablando, este proyecto ayuda a disminuir las emisiones de CO_2 al promover el empleo de energías renovables, en vez de combustibles fósiles. A través de mejorar la eficiencia de los sistemas fotovoltaicos por medio del monitoreo y análisis de la

información, se puede reducir el gasto energético, lo cual desemboca en una administración más eficiente de los recursos naturales y un impacto ambiental menor.

Las aplicaciones de monitoreo y plataformas de visualización de datos, como las implementadas en este proyecto, facilitan la visualización del impacto de estas acciones, impulsando la colaboración activa de los ciudadanos en la transición energética. Gracias a las interfaces visuales, los usuarios pueden ver al momento cómo su consumo y producción de energía impactan en sus finanzas y en el planeta.

7. Planificación y coste del trabajo

Planificación

El proyecto ha sido dividido en fases claramente definidas, cada una con unos objetivos. A continuación, se describen las fases del proyecto:

Fase 1: Investigación y análisis de los requisitos del proyecto

- **Objetivo**: Definir los objetivos y requisitos del sistema para el monitoreo de datos.
- **Actividades**:
  - Revisión de las necesidades del consumidor y del productor para el sistema.
  - Establecimiento de los requisitos técnicos para la integración del sistema.
  - Estudio de las mejores herramientas para la integración de sistemas IoT, como Mosquitto, Node-Red, InfluxDB y Grafana.
- **Duración estimada**: 2 semanas.

Fase 2: Investigación de herramientas y plataformas en base a los requisitos

- **Objetivo**: Seleccionar las herramientas más adecuadas para el proyecto, teniendo en cuenta las necesidades de monitoreo y predicción de consumo y producción.
- **Actividades**:
  - Estudio e investigación de herramientas de base de datos y plataformas de visualización de gráficos (InfluxDB y Grafana).
  - Estudio e investigación de plataformas de procesamiento de datos como Node-Red y brokers de mensajería como Mosquitto.
  - Selección de las herramientas más compatibles y eficientes.
- **Duración estimada**: 1 semana.

Fase 3: Desarrollo de pruebas de funcionamiento de las distintas herramientas

- **Objetivo**: Realizar pruebas con las herramientas seleccionadas para asegurar que funcionan correctamente entre sí y cumplen con los requisitos.
- **Actividades**:
  - Pruebas iniciales para comprobar la conexión entre Mosquitto y Node-Red.
  - Pruebas de almacenamiento de datos en InfluxDB.
  - Pruebas de visualización de datos en Grafana.
- **Duración estimada**: 2 semanas.

Fase 4: Desarrollo del sistema de monitoreo uniendo las distintas herramientas

- **Objetivo**: Unión las herramientas seleccionadas para crear un sistema de monitoreo para los datos de consumo y producción, y que los almacene en InfluxDB.
- **Actividades**:
  - Configuración de Mosquitto para recibir datos de consumo y producción.
  - Creación de los flujos de datos en Node-Red para procesar y enviar los datos a InfluxDB.
  - Configuración de Grafana para visualizar los datos almacenados en InfluxDB.
- **Duración estimada**: 3 semanas.

Fase 5: Pruebas para comprobar el correcto funcionamiento e inserción de los datos

- **Objetivo**: Verificar que los datos de consumo y producción se insertan correctamente en la base de datos de InfluxDB, y se pueden visualizan correctamente en la interfaz.
- **Actividades**:
  - Pruebas de inserción y extracción de datos en InfluxDB.
  - Pruebas para la visualización en Grafana.
- **Duración estimada**: 1 semana.

Fase 6: Desarrollo del código asociado a las funciones de forecast tanto de consumo como de producción

- **Objetivo**: Desarrollar funciones que permitan hacer predicciones para el consumo y la producción en base a los datos históricos.
- **Actividades**:
  - Pruebas para la creación de algoritmos para el forecast de consumo y producción.
  - Validación de las predicciones contra los datos reales para asegurar la precisión de los modelos.
- **Duración estimada**: 2 semanas.

Fase 7: Desarrollo de una aplicación web que sirva como interfaz de usuario

- **Objetivo**: Crear una aplicación web fácil de usar, para que los usuarios sean capaces de interactuar con el sistema y ver tanto el consumo, como la producción y las predicciones de energía.
- **Actividades**:
  - Diseño de la aplicación web para la visualización de los datos.
  - Desarrollo del backend de la aplicación para integrar con InfluxDB.
  - Creación de gráficos y tablas para la visualización de datos.
- **Duración estimada**: 3 semanas.

Fase 8: Pruebas de funcionamiento de la aplicación web

- **Objetivo**: Asegurar que la aplicación web funcione correctamente y sea fácil de usar.

- **Actividades**:
    - Pruebas funcionales para validar que todas las características de la aplicación estén operativas.
    - Verificar la compatibilidad y resolución de errores en diferentes navegadores y dispositivos.
    - Verificar la consistencia de los datos en las tablas y gráficos.
- **Duración estimada**: 1 semana.

Duración total estimada: 14 semanas (3-4 meses)

Coste del trabajo

El proyecto se ha desarrollado usando herramientas gratuitas, lo que ha permitido minimizar los costes asociados. A continuación, se detallan puntos del coste:

- **Herramientas y plataformas**: Las principales herramientas utilizadas para el desarrollo del sistema de monitoreo, como Mosquitto, Node-RED, InfluxDB y Grafana, son gratuitas para todos los usuarios. Gracias a esto, se ha podido crear el sistema sin ningún gasto en licencias ni software.
- **Infraestructura**: El sistema ha sido ejecutado en una infraestructura propia (ordenador propio) y en un servidor local para la base de datos, por tanto, no ha sido necesario alquilar servidores o servicios en la nube. Esto ha contribuido a mantener en cero los gastos totales. Se han aprovechado al máximo el uso de recursos locales.
- **Optimización de recursos**: Una vez que todas las herramientas usadas han sido sin coste, el coste del trabajo ha quedado limitado a los recursos humanos necesarios para la investigación, desarrollo y pruebas del sistema, sin que ello implicara gastos significativos en licencias o suscripciones.

En resumen, el coste del trabajo ha sido mínimo, ya que las herramientas, sistemas y plataformas que se han usado son gratuita. El único recurso invertido ha sido el tiempo de desarrollo y pruebas, lo que ha permitido llevar a cabo el proyecto sin gastos adicionales.

Conclusiones y prospectiva

1. De acuerdo con los objetivos planteados

El proyecto ha logrado las metas que se plantearon al inicio, tanto en la funcionalidad como en los resultados finales. Se ha desarrollado un sistema para el monitoreo de la energía eléctrica en el seno de sistemas FV de autoconsumo colectivo en comunidades energéticas, que no solo mejora el seguimiento de los datos de consumo y producción, sino que también permite a los usuarios tomar decisiones basadas en información real.

El sistema se basa en la integración de herramientas y tecnología avanzada como Mosquitto, Node-Red, InfluxDB y Grafana, todos ellas trabajando en conjunto, para ofrecer una solución robusta y eficiente.

Uno de los resultados clave del proyecto ha sido la creación de la aplicación web, la cual permite a los usuarios no solo monitorear en tiempo real el comportamiento energético de sus paneles solares, sino también controlar y mejorar su consumo. Para nuestro proyecto, el poder mostrar a los usuarios rápidamente el consumo y producción es fundamental.

El sistema ha demostrado funcionar correctamente recogiendo datos en tiempo real, gracias a ello se han realizado predicciones precisas sobre el consumo y la generación de energía. Estos avances no solo optimizan el uso de energías limpias, también contribuyen significativamente al ahorro energético de los usuarios.

En resumen, el proyecto ha logrado los objetivos planteados, creando un sistema de monitoreo eficaz, interactivo y fácil de usar para los usuarios, que les permite controlar eficientemente su consumo y producción de energía.

2. Nuevas líneas de actuación

Se han obtenidos resultados muy positivos, pero hay varias líneas de actuación con las que se podría mejorar y ampliar nuestro proyecto en un futuro:

Mejoras en el forecast:

Aunque el modelo de predicción actual es efectivo, se podrían usar algoritmos de machine learning, para aumentar la precisión de las predicciones de consumo y producción, usando variables como el clima, las estaciones del año, y el comportamiento histórico de los usuarios.

Creación de sistema de gestión de usuarios:

Como objetivo a futuro se puede implementar esta aplicación en una comunidad energética, donde se va a necesitar gestionar diferentes usuarios, cada uno de ellos con unos datos de consumo y producción diferentes. Por tanto, se necesita crear una base de datos con nombres de usuario y contraseñas, para poder iniciar sesión.

Ampliación de las funcionalidades de la aplicación:

Se podrían integrar más funcionalidades interactivas, como poder realizar ajustes automáticos en el consumo energético de los usuarios, usando para ello las predicciones

de producción y consumo. También un sistema de recomendaciones personalizadas para optimizar el uso de la energía, basándose en el análisis de datos históricos y preferencias de los usuarios. Otra opción que se podría añadir es la generación de informes en base a los datos obtenidos, estos informes se descargarían en un csv o un Excel.

Sostenibilidad económica:

Un aspecto importante es la sostenibilidad económica del sistema. A medida que se amplíe el proyecto, se deben buscar fuentes de financiación pública o privada que ayuden a cubrir los costes de implementación y mantenimiento, especialmente en comunidades más grandes o en zonas rurales.

Expansión a otras tecnologías de energía renovable:

Aunque el sistema está actualmente centrado en la energía solar fotovoltaica, en el futuro podría ser fácilmente expandido para incluir otras tecnologías de energía renovable como energía eólica o biomasa, lo que permitiría una diversificación en la generación de energía.

Referencias bibliográficas

[1] *El Pacto verde europeo*. (s. f.). Comisión Europea. https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/european-green-deal_es

[2] PV Europe. (2025, 27 mayo). *New report: Global PV capacity could reach 655 GW in 2025*. https://www.pveurope.eu/markets/new-report-global-pv-capacity-could-reach-655-gw-2025

[3] Hedley, N. (2025, 8 enero). *EU power sector emissions fall sharply again in 2024 as renewables advance*. The Progress Playbook. https://theprogressplaybook.com/2025/01/07/eu-power-sector-emissions-fall-sharply-again-in-2024-as-renewables-advance/

[4] Redeia. (s. f.). *Las energías renovables generan el 56% del 'mix' eléctrico español en 2024*. Red Eléctrica. https://www.ree.es/es/sala-de-prensa/actualidad/nota-de-prensa/2024/12/las-energias-renovables-generan-el-56-del-mix-electrico-espanol-2024

[5] *EU Market Outlook for Solar Power 2024-2028 - SolarPower Europe*. (s. f.). https://www.solarpowereurope.org/insights/outlooks/eu-market-outlook-for-solar-power-2024-2028/detail

[6] *Comunidades energéticas | IDAE*. (s. f.). https://www.idae.es/ayudas-y-financiacion/comunidades-energeticas

[7] Veridika. (s. f.). *Energía común*. https://www.energiacomun.org/

[8] LIFE 3.0 - LIFE project public page. Retrieved Jul 9, 2025, from https://webgate.ec.europa.eu/life/publicWebsite/project/LIFE21-CET-ENERCOM-JALON-101076395/joining-actors-for-local-development-of-new-large-scale-regional-energy-communities

[9] *Comunidades energéticas. (s. f.). ECODES - Tiempo de Actuar. https://ecodes.org/hacemos/energia-y-personas/comunidades-energeticas*

[10] *¿Qué es internet de las cosas? | IoT explicada | SAP*. (s. f.). SAP. https://www.sap.com/spain/products/technology-platform/what-is-iot.html

[11] *¿Qué es una comunidad energética y qué solución aporta?* (2025, 16 abril). CERES. https://comunidadceres.es/noticias/que-es-una-comunidad-energetica/?gad_source=1&gad_campaignid=22750846507&gclid=CjwKCAjwg7PDBhBxEiwAf1CVu4rvUrICmTAOmDkzkj3EtYox8tI3_chUJLSOLvlu5Hi7p63UqSeUpxoCT6sQAvD_BwE

[12] Ineradmin. (2023, 18 agosto). Diferencias entre el autoconsumo colectivo y el autoconsumo individual - INERNOVA. *INERNOVA*. https://inernova.es/diferencias-autoconsumo-colectivo-autoconsumo-individual/

[13] SABIA, Energía inteligente. (2024, 25 abril). *» Coeficientes de Reparto par autoconsumo colectivo ¿Qué son ?* SABIA. https://sabiaenergia.es/noticias/coeficientes-de-reparto-para-autoconsumo-colectivo/

[14] Energetico, C. (2024, 30 agosto). Portal de monitorización FusionSolar de Huawei para autoconsumo. *Cambio Energético*. https://www.cambioenergetico.com/blog/monitorizacion-fusionsolar-huawei/

[15] *Huawei-FusionSolar-contribuye-a-la-instalacion-de-la-primera-planta-fotovoltaica-flotante-en-Espana*. (s. f.). Huawei. https://www.huawei.com/es/news/es/2020/huawei-fusionsolar-contribuye-a-la-instalacion-de-la-primera-planta-fotovoltaica-flotante-en-espana

[16] *Eclipse Mosquitto*. (2018, 8 enero). Eclipse Mosquitto. https://mosquitto.org/

[17] industry4.0 systems. (2021, 18 noviembre). *Instalar y configurar InfluxDB v1 (Base de datos de series temporales)* [Vídeo]. YouTube. https://www.youtube.com/watch?v=5v9URGEqOmM

[18] *Documentation*. (2020, 6 julio). Eclipse Mosquitto. https://mosquitto.org/documentation/

[19] *Low-code programming for event-driven applications : Node-RED*. (s. f.). https://nodered.org/

[20] *User Guide : Node-RED*. (s. f.). https://nodered.org/docs/user-guide/

[21] Pedro González Gil. (2022, 30 marzo). *Node RED #4 - Publicación y suscripción con MQTT* [Vídeo]. YouTube. https://www.youtube.com/watch?v=_XAy1m571n0

[22] Guia de NODERED: Nodos más usuales . eMARSITAK. Retrieved Jul 9, 2025, from https://proiektuak.maristak.com/mod/book/view.php?id=949&chapterid=423

[23] EDUCATRONICOS ISC. (2022, 10 octubre). ✔ *Aprende a cómo usar Node Red 2022* [Vídeo]. YouTube. https://www.youtube.com/watch?v=Qcn9tNJLHzs

[24] Hugo Alexander Peña. (2024, 9 mayo). *IoT con Mqtt con NodeRed Influxdb y Grafana* [Vídeo]. YouTube. https://www.youtube.com/watch?v=n9mmAzFCozY

[25] Martínez, J. (2022b, febrero 25). Qué es InfluxDB y primeros pasos. *OpenWebinars.net*. https://openwebinars.net/blog/que-es-influxdb-y-primeros-pasos/

[26] *Using influx - InfluxDB command line interface | InfluxDB OSS v1 Documentation*. (s. f.). InfluxData Inc. https://docs.influxdata.com/influxdb/v1/tools/influx-cli/use-influx-cli/

[27] Martínez, J. (2021, 27 diciembre). Qué es Grafana y primeros pasos. *OpenWebinars.net*. https://openwebinars.net/blog/que-es-grafana-y-primeros-pasos/

[28] cloudnixiass. (2024, 30 marzo). *Easy to Install and use Grafana in Windows | Step-by-Step Tutorial* [Vídeo]. YouTube. https://www.youtube.com/watch?v=v7Bxka2Fb1g

[29] iSebas. (2023, 6 diciembre). *TALLER IoT CONECTANDO INFLUXDB CON GRAFANA* [Vídeo]. YouTube. https://www.youtube.com/watch?v=_ERvVJ1oh6c

[30] Neural Connections. (2024, 16 julio). *Como Conectar Grafana con InfluxDB Usando Docker para Integración con Home Assistant!* [Vídeo]. YouTube. https://www.youtube.com/watch?v=o6xsxIbpuLQ

[31] *Datadis*. (s. f.). https://datadis.es/private-api

[32] Suntropy AI. (2022, 12 enero). *Datadis | Enerlence - Webinar formativo: Cómo obtener los datos de consumo de tu cliente* [Vídeo]. YouTube. https://www.youtube.com/watch?v=HbJzpHAT3uM

[33] *node-red-contrib-influxdb*. (s. f.). https://flows.nodered.org/node/node-red-contrib-influxdb

[34] iSebas. (2023a, diciembre 5). *TALLER IoT + CONECTANDO NODERED CON INFLUXDB* [Vídeo]. YouTube. https://www.youtube.com/watch?v=ELBS0BwpTOo

[35] Develoteca - Oscar Uh. (2021, 30 junio). *Cómo hacer una aplicación web en python* [Vídeo]. YouTube. https://www.youtube.com/watch?v=X5E2SXOsr4E

[36] *Estilos en HTML*. (s. f.). Abrirllave.com. https://www.abrirllave.com/html/estilos.php

[37] Ramos, L. C. (2022, 7 julio). *Cómo añadir gráficos en tu web con Chart.js*. Adictos Al Trabajo. https://adictosaltrabajo.com/2022/07/01/como-anadir-graficos-en-tu-web-con-chart-js/

[38] *PyInstaller Manual — PyInstaller 6.14.2 documentation*. (s. f.). https://pyinstaller.org/en/stable/

[39] *T-Test, Chi-Square, ANOVA, Regression, Correlation*. . . *(s. f.-c). https://datatab.es/tutorial/wilcoxon-test*

[40] *T-Test, Chi-Square, ANOVA, Regression, Correlation*. . . (s. f.). https://datatab.es/statistics-calculator/hypothesis-test/wilcoxon-test-calculator

[41] *Wilcoxon Signed-Rank Test Calculator*. (s. f.-b). Social Science Statistics. https://www.socscistatistics.com/tests/signedranks/default2.aspx