UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Industrial

Diseño e implementación de un sistema de control, supervisión y adquisición de datos (SCADA) mediante ordenadores empotrables SBC y lenguaje Python de los dispositivos electrónicos de potencia inteligentes conectados a una SmartGrid

Trabajo Fin de Máster

Máster Universitario en Ingeniería Industrial

AUTOR/A: Ziliute, Klaudija

Tutor/a: González Medina, Raúl

Cotutor/a: Figueres Amorós, Emilio

CURSO ACADÉMICO: 2023/2024

AGRADECIMIENTOS

“A mis padres, por permitirme cursar mis estudios

A mi hermana Kornelija, por animarme y ayudarme con el diseño

A mis amigos por apoyarme y motivarme

A mis tutores, por permitirme desarrollar esta idea

Y a mis compañeros de trabajo por estar juntos, especialmente los viernes”

RESUMEN

Las SmartGrids aprovechan las posibilidades de comunicación y procesado de datos que ofrecen las tecnologías de la información para mejorar las prestaciones de las redes eléctricas de transmisión de energía tradicionales. Algunas de esas ventajas son la monitorización y el control remoto de los equipos conectados a la smartgrid, la recopilación de datos, la gestión de sistemas de generación y almacenamiento de energía distribuidos, etc.

En este TFM se pretende implementar un sistema para la visualización y el control básico de los dispositivos mediante instrumentación virtual y la recopilación de datos históricos de los valores más relevantes de cada dispositivo.

En cada punto de conexión de la smartgrid se dispondrá de un ordenador tipo "Single Board Computer" (SBC) con procesador de la familia ARM y un sistema operativo Linux al que se le podrán conectar varios dispositivos de potencia empleando un protocolo MODBUS sobre RS-485 (TIA-485). Cada dispositivo de potencia tendrá una dirección única para poder acceder individualmente. La visualización y el control se realizarán mediante instrumentación virtual en un ordenador de escritorio equipado con monitor, teclado y ratón. Todos los ordenadores estarán conectados entre sí por ethernet.

Para todos los programas que se desarrollen en este TFM se empleará el lenguaje Python. Durante el desarrollo de los programas se utilizarán máquinas virtuales con sistema operativo Linux corriendo en un hipervisor VMware ESXi y conectadas a hardware real mediante USB. Finalmente, el programa se portará a las SBCs para su implementación final.

Palabras Clave: Smargrid, SBC, Beaglebone Black, Raspberry Pi, Python; Linux, Máquina Virtual, SCADA, Docker, Ethernet, Generación Distribuida, EMS, Energía

RESUM

Les *SmartGrids* aprofiten les possibilitats de comunicació i processament de dades que ofereixen les tecnologies de la informació per a millorar les prestacions de les xarxes elèctriques de transmissió d'energia tradicionals. Algunes d'aqueixos avantatges són el monitoratge i el control remot dels equips connectats a la *smartgrid*, la recopilació de dades, la gestió de sistemes de generació i emmagatzematge d'energia distribuïts, etc.

En aquest TFM es pretén implementar un sistema per a la visualització i el control bàsic dels dispositius mitjançant instrumentació virtual i la recopilació de dades històriques dels valors més rellevants de cada dispositiu.

En cada punt de connexió de la *smartgrid* es disposarà d'un ordenador tipus "Single Board Computer" (SBC) amb processador de la família ARM i un sistema operatiu Linux al qual se li podran connectar diversos dispositius de potència emprant un protocol MODBUS sobre RS-485 (TIA-485). Cada dispositiu de potència tindrà una direcció única per a poder accedir individualment. La visualització i el control es realitzaran mitjançant instrumentació virtual en un ordinador d'escriptori equipat amb monitor, teclat i ratolí. Tots els ordinadors estaran connectats entre si per *ethernet*.

Per a tots els programes que es desenvolupen en aquest TFM s'emprarà el llenguatge *Python*. Durant el desenvolupament dels programes s'utilitzaran màquines virtuals amb sistema operatiu Linux corrent en un *hipervisor VMware ESXi* i connectades a maquinari real mitjançant USB. Finalment, el programa es portarà a les *SBCs* per a la seua implementació final.

Paraules clau: *Smargrid, SBC, Beaglebone Black, Raspberry Pi, Python*; Linux, Màquina Virtual, *SCADA, Docker, Ethernet*, Generació Distribuïda, EMS, Energia

ABSTRACT

SmartGrids take advantage of the communication and data processing capabilities offered by information technologies to improve the performance of traditional power transmission networks. Some of these advantages include the monitoring and remote control of the equipment connected to the smart grid, data collection, distributed energy generation and storage systems management, etc.

In this master's Thesis, the goal is to implement a system for visualization and basic control of devices through virtual instrumentation and the collection of historical data of the most relevant values from each device.

At each smart grid connection point, there will be a "Single Board Computer" (SBC) with an ARM-family processor and a Linux operating system, to which several power devices can be connected using the MODBUS protocol over RS-485 (TIA-485). Each power device will have a unique address for individual access. The visualization and control will be carried out via virtual instrumentation on a desktop computer equipped with a monitor, keyboard, and mouse. All computers will be connected via Ethernet.

All the programs developed for this master's Thesis will use the Python programming language. During the development process, virtual machines with a Linux operating system running on a VMware ESXi hypervisor will be used and connected to real hardware via USB. Finally, the program will be ported to the SBCs for its final implementation.

Keywords: SmartGrid, SBC, Beaglebone Black, Raspberry Pi, Python, Linux, Virtual Machine, SCADA, Docker, Ethernet, Distributed Generation, EMS, Energy

ÍNDICE DE DOCUMENTOS

- MEMORIA
- PRESUPUESTO
- ANEXOS

MEMORIA

ÍNDICE DE LA MEMORIA

Contenido

ÍNDICE DE LA FIGURAS

ÍNDICE DE LOS FRAGMENTOS DE CÓDIGO

CAPÍTULO 1. INTRODUCCIÓN

1.1. Objetivos

El objetivo general de este Trabajo de Fin de Máster (TFM) es implementar un sistema básico de visualización y control de dispositivos en una ***Smart Grid*** (Red Inteligente), utilizando tecnologías de la información y simulaciones para mejorar la gestión y eficiencia de las redes eléctricas tradicionales. El sistema simulará y gestionará la generación y el almacenamiento distribuido, aprovechando las capacidades de comunicación y procesamiento de datos mediante el uso de contenedores, *APIs Flask*, bases de datos y protocolos de seguridad.

Los objetivos específicos se enumeran a continuación:

- **Implementación de un sistema de visualización y control básico:** Desarrollar una solución web que permita la visualización en tiempo real de los dispositivos conectados a la *Smart Grid* (simulados por las máquinas virtuales), así como la capacidad de control básico de dichos dispositivos mediante el uso de *APIs Flask* para enviar y recibir datos entre las máquinas y el *backend*.
- **Recopilación y almacenamiento de datos históricos:** Desarrollar una infraestructura que permita la recopilación y almacenamiento de datos históricos relevantes generados por las simulaciones de los dispositivos (generación solar, estado de baterías, y generación eólica). Los datos se almacenarán en bases de datos PostgreSQL, permitiendo su posterior análisis.
- **Simulación de dispositivos de generación y almacenamiento:** Emplear máquinas virtuales que simulen el comportamiento de los dispositivos (SBCs) conectados a la *Smart Grid*, generando datos de dispositivos de energía solar, baterías y energía eólica. Utilizar *APIs Flask* para exponer estos datos de forma que el *backend* pueda interactuar con ellos mediante peticiones *GET* y *POST*.
- **Desarrollo del sistema utilizando Python y Docker:** Implementar todos los scripts de simulación y procesamiento de datos en el lenguaje de programación Python, aprovechando la modularidad y portabilidad de Docker para agrupar los diferentes contenedores del sistema (*backend*, *frontend*, bases de datos y máquinas virtuales), asegurando su escalabilidad y funcionamiento en distintos entornos.
- **Implementación de un entorno seguro de comunicación:** Establecer un entorno seguro para la transmisión de datos entre las máquinas virtuales y el *backend* mediante la utilización del protocolo SSH. Encapsular las peticiones HTTP de *Flask* dentro de túneles SSH para garantizar la integridad y confidencialidad de los datos durante su transmisión.
- **Visualización web en tiempo real:** Desarrollar una interfaz web interactiva que permita a los usuarios visualizar los datos de simulación en tiempo real, accediendo a los datos almacenados en las bases de datos y procesados por el *backend*. El *frontend* permitirá la consulta y análisis de los datos, facilitando la toma de decisiones.

El final del proyecto quedaría de esta manera:

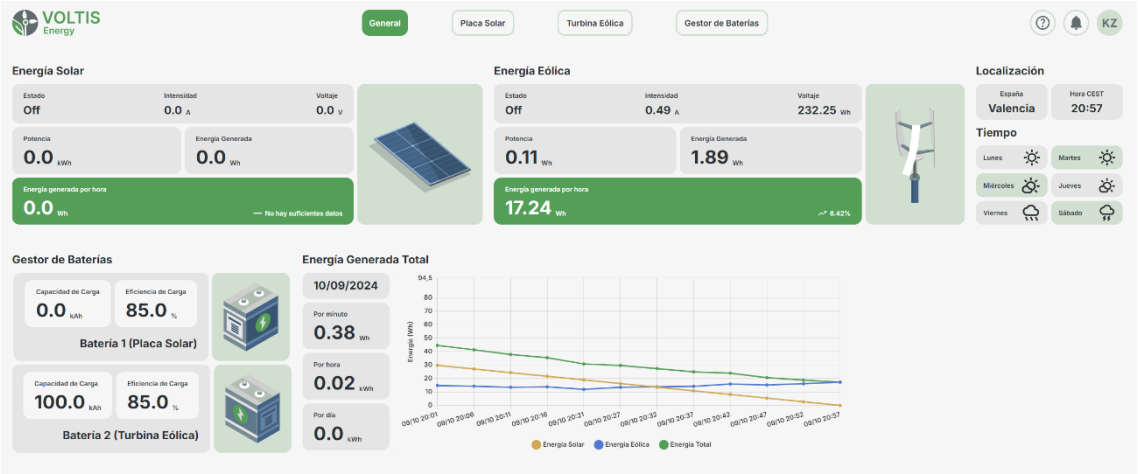

Figura 1. Visualización general de la pantalla principal.

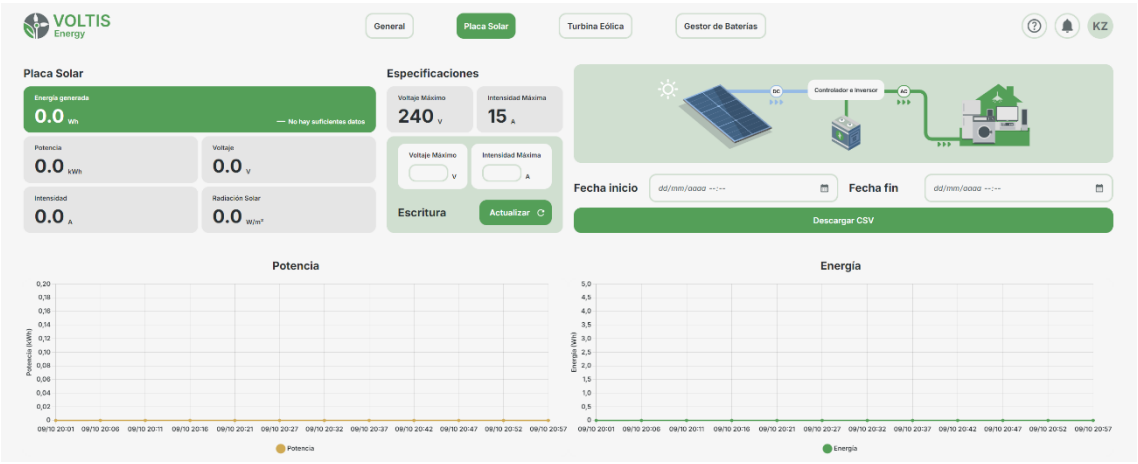

Figura 2. Visualización general de la placa solar simulada.

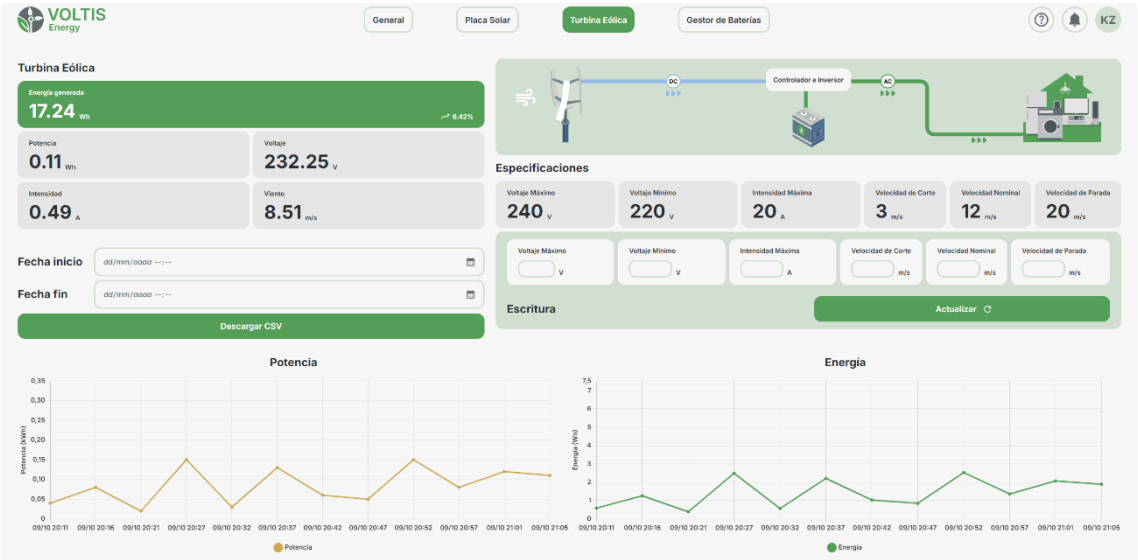

Figura 3. Visualización general de la turbina eólica simulada.

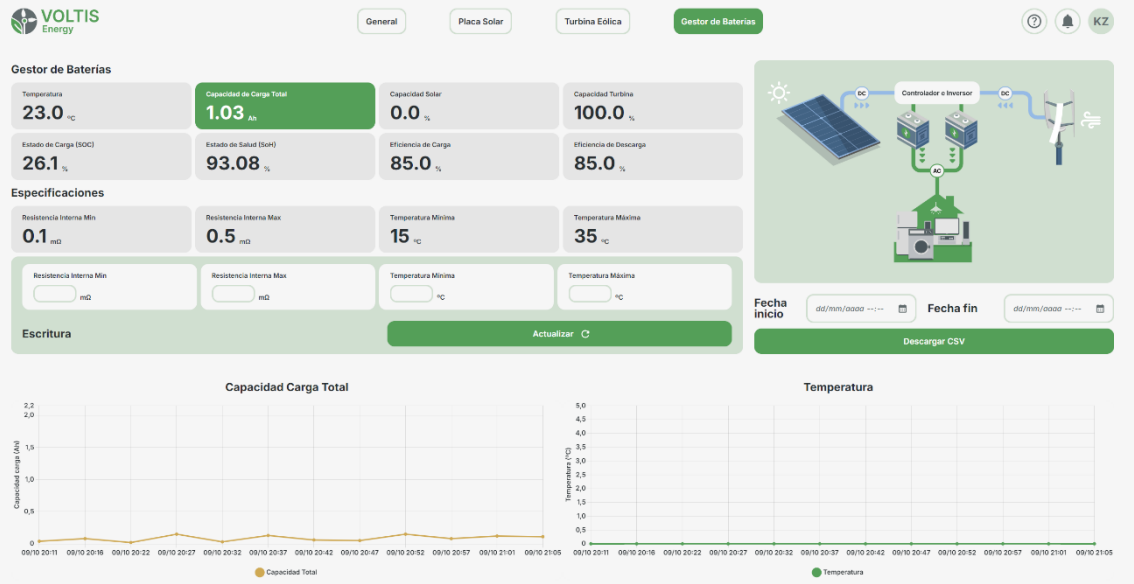

Figura 4. Visualización general de las baterías simuladas.

1.2. Motivación

Varios factores me han impulsado a realizar este trabajo. Por un lado, haber cursado la especialización en electrónica me ha permitido obtener una visión más global sobre la electrónica de potencia y sus diversas aplicaciones, lo que me ha facilitado entender cómo los sistemas energéticos, como las *Smart Grids*, pueden ser gestionados y controlados de manera más eficiente mediante el uso de tecnologías avanzadas.

Por otro lado, durante mis prácticas en una empresa dedicada al sector del agua y la digitalización de la información, adquirí conocimientos prácticos sobre la implementación de soluciones tecnológicas para el control y tratamiento de datos de dispositivos electrónicos. Estos conocimientos me resultaron fundamentales para entender cómo integrar la recopilación y procesamiento de datos en tiempo real, y cómo estos pueden ser utilizados para monitorear y optimizar el rendimiento de diferentes sistemas. Esta experiencia me ha llevado a querer aplicar dichas técnicas al desarrollo de un sistema para la gestión de dispositivos en redes eléctricas inteligentes, como es el caso de las *Smart Grids*.

1.3. Antecedentes

El crecimiento de la demanda energética y la necesidad de integrar tecnologías avanzadas en la gestión de las redes eléctricas han llevado al desarrollo de redes eléctricas inteligentes, conocidas como *Smart Grids*. Estas redes buscan mejorar la eficiencia, fiabilidad y sostenibilidad de las infraestructuras energéticas tradicionales, las cuales fueron diseñadas en un contexto tecnológico muy diferente al actual. En este marco, las *Smart Grids* se presentan como una solución innovadora, integrando tecnologías de la información y comunicación para transformar la infraestructura energética [1].

1.3.1. Evolución de las redes eléctricas

Las redes eléctricas convencionales surgieron bajo un modelo centralizado, donde la electricidad se generaba en grandes plantas y se distribuía a los consumidores finales a través de una red unidireccional. Aunque este modelo fue eficiente durante décadas, con el crecimiento de la demanda y la urbanización, comenzaron a surgir sus limitaciones en términos de flexibilidad y capacidad de respuesta. Además, el cambio hacia una mayor sostenibilidad, con la adopción de fuentes de energía renovable, ha puesto de manifiesto la necesidad de una mayor adaptabilidad.

Estas fuentes de energía, como la solar y la eólica, son intermitentes y difíciles de predecir, lo que hace que las redes convencionales resulten inadecuadas. En respuesta a este desafío, las *Smart Grids* permiten gestionar estos recursos de forma más eficiente, integrando comunicación avanzada y tecnología de control [2].

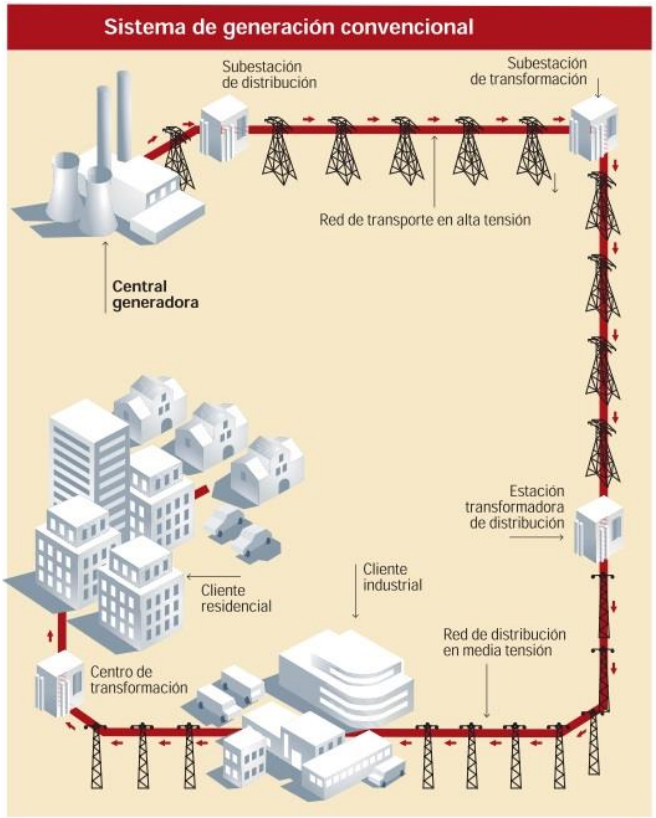

Figura 5. Esquema red eléctrica convencional [10].

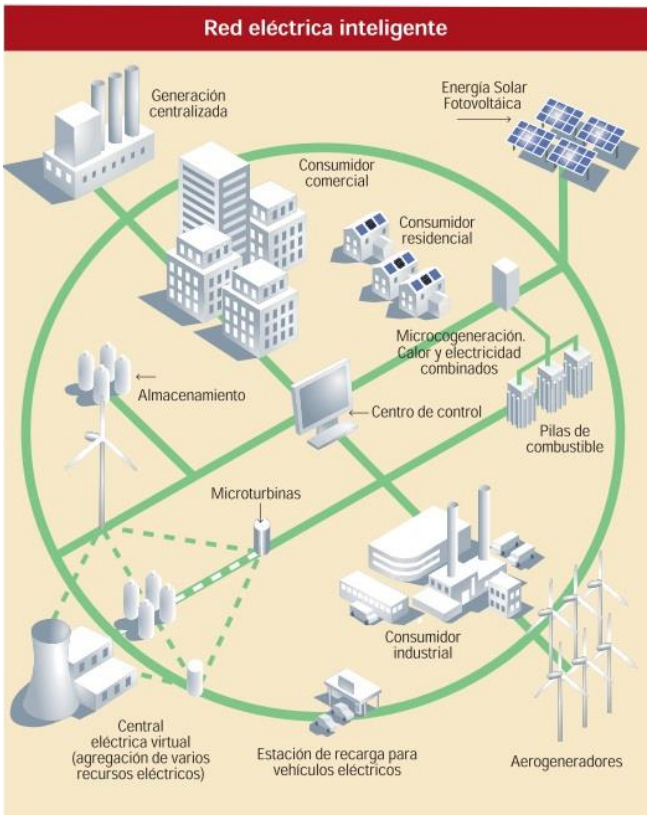

Figura 6. Esquema red eléctrica *Smart Grid* [10].

1.3.2. Introducción de las Tecnologías de la Información en las redes eléctricas

El avance de las tecnologías de la información (TI) ha permitido que las redes eléctricas evolucionen hacia un modelo más inteligente, donde los datos se recopilan y procesan en tiempo real para optimizar la gestión de la red [3]. Esto incluye la implementación de sensores y sistemas de control distribuidos que monitorean continuamente el estado de la red y ajustan su operación para mejorar la eficiencia energética. Además, la integración de estas tecnologías facilita la gestión remota de los dispositivos, desde la regulación de la carga hasta la coordinación de la generación y almacenamiento distribuido de energía [4].

1.3.3. El rol de las *Smart Grids* en el futuro energético

Las *Smart Grids* se perfilan como la próxima etapa en la evolución de las redes eléctricas, con la capacidad de gestionar la energía de manera distribuida y facilitar la integración de fuentes de energía renovable. Estas redes inteligentes permiten una mayor participación de los consumidores, quienes pueden también actuar como productores de energía, generando su propia electricidad mediante sistemas como paneles solares.

Este nuevo paradigma, en el que los consumidores son también productores (conocidos como "prosumidores"), fomenta un uso más eficiente y responsable de la energía [5]. Además, el enfoque distribuido de las *Smart Grids* mejora la resiliencia de la red, permitiendo una mayor

capacidad de respuesta ante eventos imprevistos, como desastres naturales o fluctuaciones en la demanda [6].

1.3.4. Desafíos en la implementación de *Smart Grids*

Aunque las *Smart Grids* ofrecen numerosas ventajas, su implementación no está exenta de desafíos. La infraestructura actual necesita ser actualizada para soportar las tecnologías avanzadas que integran estas redes, lo que supone una inversión significativa. La interoperabilidad entre distintos dispositivos y sistemas es crucial para garantizar un funcionamiento eficiente y cohesivo de la red. Además, la ciberseguridad es un aspecto fundamental, ya que la interconexión de dispositivos hace que las *Smart Grids* sean vulnerables a ciberataques. Por lo tanto, es esencial desarrollar medidas robustas de seguridad para proteger la red de posibles amenazas [7].

Otro aspecto relevante es la gestión y protección de los datos generados por la *Smart Grid*. La recopilación de grandes volúmenes de datos plantea cuestiones sobre la privacidad de la información de los consumidores, lo que requiere un marco legal y regulatorio claro para su manejo [8].

1.3.5. Proyecto de implementación y control de *Smart Grids*

Este proyecto busca aprovechar las ventajas de las *Smart Grids* mediante el desarrollo de una plataforma para la visualización y control de dispositivos en una red simulada.

Las máquinas virtuales simulan la operación de dispositivos de generación y almacenamiento permitiendo el monitoreo en tiempo real a través de una plataforma web. El uso de tecnologías como *Flask* para la creación de *APIs*, y Docker para la gestión de contenedores, proporciona una arquitectura escalable y flexible. Además, se asegura la transmisión segura de datos mediante SSH, encapsulando las peticiones de los dispositivos hacia el servidor. De esta forma, los datos pueden procesarse de manera segura y eficiente en el *backend* y almacenarse en bases de datos PostgreSQL para su análisis. Esto permite no solo el monitoreo en tiempo real, sino también la recopilación de datos históricos que pueden ser utilizados para optimizar la gestión de la red.

El proyecto también utiliza contenedores y simulaciones en un entorno virtual para probar el sistema antes de su implementación final. Este enfoque asegura que el sistema sea escalable y fácil de mantener, y sienta las bases para futuras investigaciones en la integración de *Smart Grids* [9].

1.4. Viabilidad

El proyecto ha sido diseñado y probado en un entorno de simulación, lo que permite verificar la viabilidad del sistema antes de su implementación en entornos reales. Para ello, se ha utilizado una infraestructura basada en máquinas virtuales, que simulan el comportamiento de dispositivos de energía solar, almacenamiento en baterías y energía eólica. Estas máquinas están interconectadas mediante un entorno de contenedores Docker, lo que permite una configuración flexible y escalable del sistema.

La simulación incluye la transmisión automática de datos entre las máquinas virtuales y el *backend*, utilizando protocolos de seguridad como SSH, y la interacción mediante *APIs Flask*.

Este enfoque facilita el desarrollo y prueba de los componentes del sistema en un entorno controlado, asegurando que la implementación final en hardware físico o plataformas como los *Single Board Computers* (SBCs) sea fiable y eficiente.

Aunque en este proyecto se ha trabajado principalmente con simulaciones y contenedores, la infraestructura puede ser fácilmente adaptada para correr en dispositivos de bajo costo como la *Raspberry Pi* o *BeagleBone Black*. Estos equipos, al igual que las máquinas virtuales, pueden operar con sistemas operativos Linux, lo que garantiza la portabilidad del software desarrollado. Además, la capacidad de estos dispositivos para ejecutar entornos Python y contenedores Docker refuerza la viabilidad de migrar el sistema simulado a una implementación física cuando sea necesario.

CAPÍTULO 2. DESCRIPCIÓN DEL SISTEMA

2.1. Arquitectura general

El sistema está diseñado para gestionar la simulación, procesamiento y visualización en tiempo real de datos generados por dispositivos de energía solar, un aerogenerador y almacenamiento de energía en baterías.

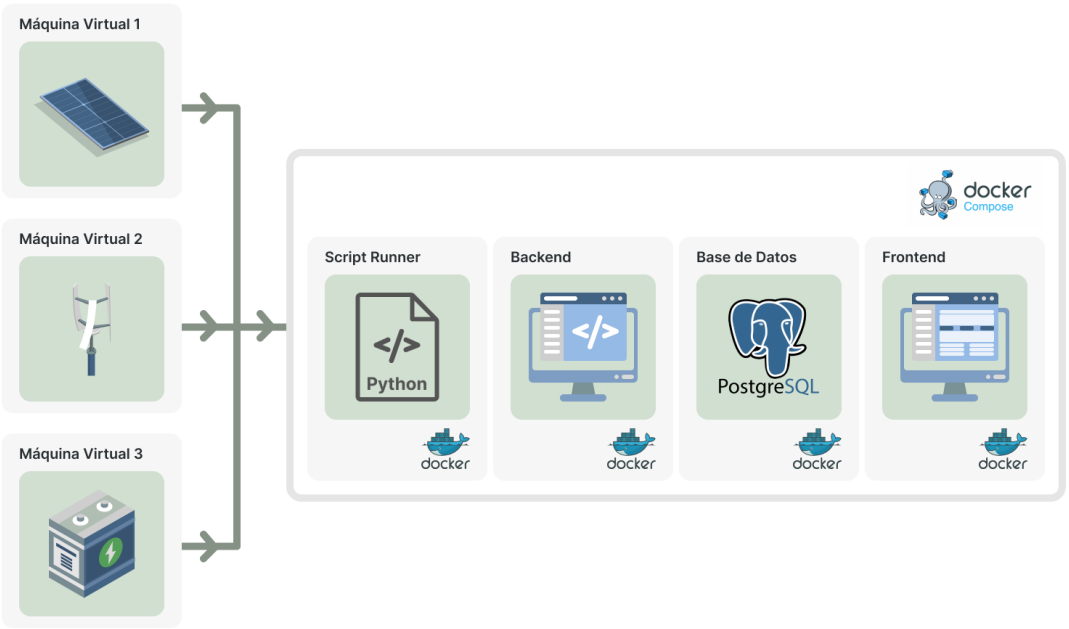

Figura 7. Esquema del sistema.

Los componentes principales de la arquitectura son:

- **Máquinas Virtuales**: Cada una de las tres máquinas virtuales está dedicada a simular un tipo de dispositivo de la *Smart Grid*:
  - Máquina 1 simula la producción de energía solar.
  - Máquina 2 simula el almacenamiento de energía en baterías.
  - Máquina 3 simula un aerogenerador (generador eólico).
- **Script Runner:** Este componente se encarga de enviar las peticiones desde el *backend* a las máquinas virtuales, solicitando la generación y transmisión de datos. Las máquinas virtuales, en respuesta a estas peticiones, generan los datos simulados en tiempo real.
- **Backend (Flask)**: El *backend* es responsable de gestionar las peticiones de datos a las máquinas virtuales, recibir la información simulada y almacenarla en bases de datos. Además, el *backend* expone una API REST que es consultada por el *frontend* para la visualización de los datos.

- **Bases de Datos (PostgreSQL)**: Los datos simulados se almacenan en la base de datos PostgreSQL. Cada máquina virtual tiene su propia tabla de datos, permitiendo un almacenamiento estructurado y eficiente.
- **Frontend**: El *frontend* proporciona una interfaz gráfica que permite visualizar en tiempo real los datos simulados. Se comunica con el *backend* mediante API REST para obtener los datos que se almacenan en las bases de datos.

2.2. Virtualización de las *SBCs*

Uno de los objetivos del TFM es poder gestionar la conexión y control de los dispositivos de potencia mediante *Wifi* a través de las *SBCs*, *"Single Board Computer".* Dichas *SBCs* no serán físicas, sin embargo, van a emular el mismo funcionamiento como máquinas virtuales con su sistema operativo.

A partir de ahora, en todo el proyecto solo se mencionará las siglas del SBC para que no sea demasiado recurrente.

2.2.1. Definición de SBC

Las *SBCs* son ordenadores construidos en una sola placa de circuito impreso (PCB), que incluyen un procesador, memoria, almacenamiento y puertos de entrada/salida, como USB, HDMI, y otros. A diferencia de un ordenador PC tradicional que tiene múltiples componentes separados, como la placa base, la memoria RAM, la unidad de almacenamiento, etc., una SBC tiene todo integrado en una única placa.

2.2.2. Características de la SBC

Algunas características clave de las *SBCs* son:

- **Compactas**: Debido a que todo está integrado en una sola placa, las *SBCs* son generalmente pequeñas y portátiles.
- **Coste:** Suelen ser más económicas que los ordenadores tradicionales, lo que las hace accesibles para proyectos educativos, prototipos y usos especializados.
- **Versatilidad**: Pueden utilizarse para una amplia variedad de aplicaciones, desde servidores pequeños y centros de medios hasta proyectos de Internet de las cosas, "*IoT*", robótica, y sistemas embebidos.
- **Eficiencia energética**: Muchas *SBCs* están diseñadas para ser eficientes en términos de consumo de energía, lo que las hace ideales para aplicaciones donde se requiere bajo consumo.
- **Aprendizaje**: Las *SBCs* son muy populares para las personas que quieren aprender a programar, experimentar con la electrónica y construir proyectos personalizados.

2.2.3. Ejemplos de *SBCs*

Existen varios tipos de *SBCs*, a continuación, se lista algunos ejemplos:

- ***Raspberry Pi***: Es una SBC bastante popular entre la educación, proyectos personales y prototipos por ser muy versátil y tener bajo coste.

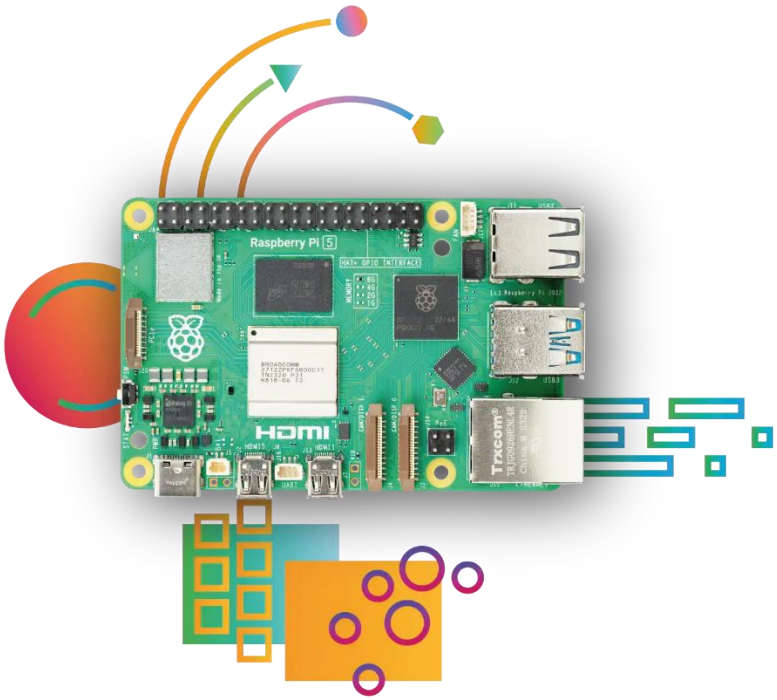

Figura 8. Raspberry Pi 5. [11]

- ***BeagleBone***: También es una SBC popular, sin embargo, su uso es más para aplicaciones más industriales y proyectos avanzados.

Figura 9. BeagleBone Black Industrial. [12]

- **Arduino**: No es una SBC en el sentido estricto, sino más bien es una plataforma de microcontroladores, sin embargo, se puede dar el mismo uso para los proyectos personales y computación embebida.

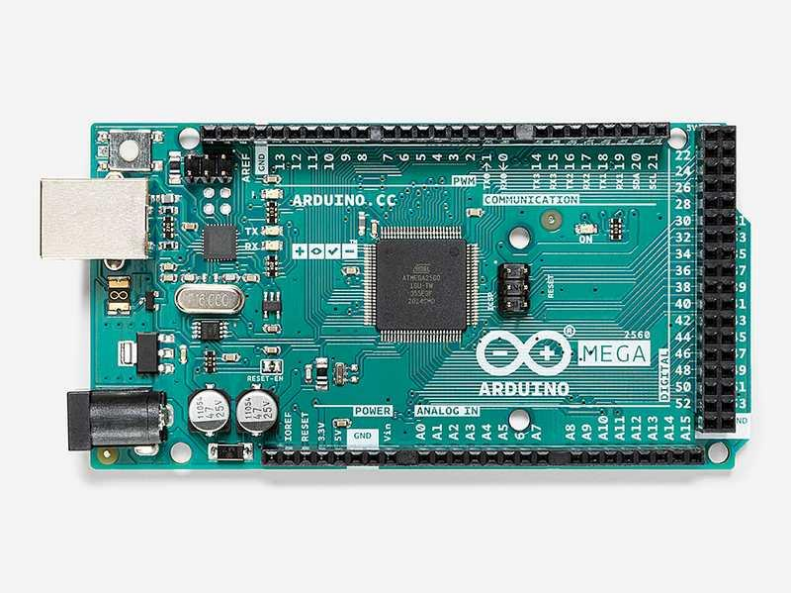

Figura 10. Arduino MEGA 2560 Rev3. [13]

La placa que se va a simular en este trabajo es una SBC de tipo *BeagleBone Black Industrial*.

2.2.4. Definición de máquina virtual

Las máquinas virtuales, "*VMs*", son entornos de software que emulan un ordenador completo, permitiendo ejecutar un sistema operativo y aplicaciones en un entorno independiente dentro de otra máquina física, en este caso un ordenador portátil.

La creación y gestión de estas máquinas virtuales se realiza a través de un software especializado conocido como hipervisor.

2.2.5. Tipos de hipervisores

Existen dos tipos principales de hipervisores:

- ***Bare-metal***: Este tipo de hipervisor se ejecuta directamente sobre el hardware físico de la máquina, sin necesidad de un sistema operativo anfitrión. Es común en entornos de servidor y es muy eficiente en cuanto a rendimiento. Ejemplos incluyen "***VMware ESXi***" y "***Microsoft Hyper-V***". Estos hipervisores son ideales para la consolidación de servidores y entornos de computación en la nube [14].
- ***Hosted***: Este hipervisor se ejecuta sobre un sistema operativo existente, como una aplicación más. Es más común en entornos de usuario final, donde se necesita ejecutar múltiples sistemas operativos en una sola máquina física. Ejemplos incluyen "***VMware Workstation***" y "***Oracle VirtualBox***". Aunque es más fácil de configurar, este tipo introduce una cierta latencia debido a la capa adicional del sistema operativo anfitrión [14].

El hipervisor que se va a utilizar para el TFM es el "*Hosted*" Oracle VirtualBox.

2.3. Script Runner

El script runner es un componente clave en el sistema, encargado de gestionar las peticiones de datos desde el *backend* hacia las máquinas virtuales. En lugar de que las máquinas generen datos de forma continua o en intervalos fijos, este diseño se basa en un modelo de petición-respuesta. El script runner envía solicitudes específicas cuando el *backend* requiere nuevos datos, lo que asegura un uso más eficiente de los recursos y optimiza la gestión de los datos generados.

2.3.1. Optimización de recursos

Al generar datos solo bajo demanda, se evita la sobrecarga de las máquinas virtuales, que no necesitan estar procesando y enviando datos continuamente. Este modelo asegura que los recursos del sistema (CPU, memoria y red) se utilicen únicamente cuando es necesario, lo que es especialmente útil en escenarios de simulación intensiva o cuando se requiere eficiencia energética.

Al reducir el volumen de datos innecesarios, también se mejora el rendimiento general del sistema, lo que puede ser crucial en entornos con recursos limitados o cuando las máquinas virtuales están ejecutándose en infraestructura compartida.

2.3.2. Flexibilidad

Este diseño permite que el *backend* tenga un mayor control sobre el momento y frecuencia con la que se solicitan los datos, haciendo el sistema más adaptable a distintas necesidades. Por ejemplo, se pueden programar las solicitudes de datos en función de eventos externos o análisis en tiempo real, en lugar de depender de una programación fija.

Esta flexibilidad es fundamental en entornos simulados donde las condiciones pueden cambiar rápidamente y se requiere una respuesta dinámica del sistema. Las peticiones pueden ser ajustadas según las condiciones de simulación, las métricas o parámetros que se estén evaluando en tiempo real.

2.3.3. Modularidad y escalabilidad

Al separar la lógica de generación de datos (que reside en las máquinas virtuales) de la lógica de petición de datos (gestionada por el script runner), se logra un diseño modular. Esto facilita la escalabilidad, permitiendo añadir más máquinas virtuales o servicios sin modificar el sistema de peticiones.

Si se introducen nuevas simulaciones o dispositivos en el futuro, solo se necesita extender la funcionalidad del script runner para integrar nuevas peticiones, manteniendo la coherencia del sistema.

2.3.4. Simulación controlada

Este enfoque permite un control total sobre la simulación de los dispositivos. En un entorno de prueba y desarrollo, es importante poder generar datos en momentos específicos para evaluar el comportamiento del sistema bajo diferentes condiciones controladas.

Con el script runner, se puede determinar cuándo y qué tipo de datos se generan, facilitando las pruebas de rendimiento, análisis de errores o simulaciones específicas.

2.3.5. Mejora en la seguridad y gestión de datos

El hecho de que los datos sean generados bajo demanda también mejora la seguridad del sistema, al reducir la exposición innecesaria de datos durante la transmisión. Esto ayuda a minimizar el riesgo de interceptación o pérdida de información, ya que se limitan las ventanas de tiempo en las que se están enviando datos sensibles.

2.4. *Backend*

El *backend*, construido utilizando el *framework (*marco de trabajo*) Flask*, es una pieza central en la arquitectura del sistema. Su diseño está orientado a recibir, procesar y almacenar datos provenientes de las máquinas virtuales, actuando como el enlace entre el *frontend*, las bases de datos y las simulaciones ejecutadas en las máquinas.

La elección de *Flask* y su integración en el *backend* se fundamenta en los siguientes motivos clave.

2.4.1. Simplicidad y ligereza

Flask es un *framework* ligero para el desarrollo de aplicaciones web y *APIs REST*, lo que permite crear una infraestructura funcional sin añadir una sobrecarga innecesaria al sistema. Esto es ideal en un proyecto que busca simular dispositivos en tiempo real y procesar datos de manera eficiente.

A diferencia de *frameworks* más pesados como Django, *Flask* ofrece mayor control sobre los componentes del sistema, permitiendo desarrollar un *backend* sencillo pero potente, que se ajusta perfectamente a las necesidades de este proyecto. [15]

2.4.2. Flexibilidad y extensibilidad

Flask proporciona un alto nivel de flexibilidad, permitiendo al desarrollador estructurar el *backend* según los requerimientos del sistema sin imponer restricciones rígidas. Esto ha permitido que el *backend* pueda manejar fácilmente la recepción de datos desde múltiples máquinas virtuales, procesarlos y luego almacenarlos en varias bases de datos.

Además, *Flask* es fácilmente extensible. Esto significa que se pueden añadir nuevas funcionalidades (como *endpoints* adicionales para manejar nuevos tipos de simulaciones o dispositivos) sin reestructurar el *backend* completo. La modularidad que permite *Flask* es clave para la escalabilidad del sistema. [16]

2.4.3. API REST para comunicación eficiente

El *backend* está diseñado para exponer una API REST, que permite que tanto las máquinas virtuales como el *frontend* interactúen con él de manera eficiente. *Flask* facilita la creación de rutas específicas para cada tipo de interacción, como el procesamiento de datos provenientes de las simulaciones o la consulta de información por parte del *frontend*.

Esto es importante en sistemas donde múltiples componentes necesitan comunicarse, y *Flask* proporciona una estructura simple pero eficaz para manejar estas interacciones. Las solicitudes HTTP (GET, POST) permiten una integración fluida entre los módulos del sistema. [17]

2.4.4. Escalabilidad y mantenimiento

Dado que *Flask* es un *framework* minimalista, el sistema puede escalar fácilmente. Si en el futuro se añaden más dispositivos simulados o se aumentan los volúmenes de datos, el *backend* puede ampliarse sin requerir una reescritura significativa del código base.

La estructura modular que permite *Flask* también hace que el mantenimiento del sistema sea más manejable. Los diferentes componentes del *backend* pueden desarrollarse y actualizarse de forma independiente, reduciendo la complejidad del código y facilitando la implementación de mejoras o nuevas características. [18]

2.4.5. Soporte para bases de datos y ORM

Aunque *Flask* es ligero, proporciona soporte para la integración con bases de datos, algo esencial en este proyecto. El *backend* está conectado a bases de datos PostgreSQL, donde se almacenan los datos generados por las simulaciones. *Flask* facilita esta integración utilizando bibliotecas como *SQLAlchemy*, un ORM (*Object-Relational Mapping*) que simplifica las consultas y la interacción con las bases de datos.

Esta elección permite que las operaciones de almacenamiento y consulta sean más fluidas y menos propensas a errores, al traducir las operaciones SQL en objetos de Python, reduciendo la posibilidad de errores de sintaxis o lógica. [19]

2.4.6. Seguridad

Aunque *Flask* es un *framework* sencillo, no compromete la seguridad. Su diseño permite implementar medidas de seguridad esenciales, como la autenticación y la protección de datos sensibles durante las transacciones entre los diferentes componentes del sistema.

Si bien en este proyecto las transmisiones de datos entre las máquinas virtuales y el *backend* están protegidas por SSH, *Flask* puede manejar la protección de las API, asegurando que solo usuarios o sistemas autorizados puedan acceder a los *endpoints* expuestos. [20]

2.4.7. Popularidad y comunidad

Flask es ampliamente utilizado y cuenta con una comunidad activa de desarrolladores. Esto proporciona acceso a una gran cantidad de recursos, bibliotecas y extensiones que permiten mejorar el *backend* con facilidad. Además, su popularidad garantiza compatibilidad con las últimas tecnologías web y mejores prácticas de desarrollo [21].

2.5. Base de datos

El sistema utiliza PostgreSQL como el sistema de gestión de bases de datos (SGBD) para almacenar los datos simulados generados por las máquinas virtuales. Esta elección no es casual, ya que PostgreSQL es conocido por ser un SGBD de código abierto, robusto y altamente escalable, lo que lo convierte en una excelente opción para proyectos que requieren almacenamiento seguro, eficiente y con gran capacidad de gestión de datos.

Las razones para la elección del PostgreSQL son las siguientes.

2.5.1. Robustez y confiabilidad

PostgreSQL se destaca por su enfoque en la integridad de los datos y el cumplimiento de las transacciones. En sistemas como este, que requieren almacenar grandes cantidades de datos generados por múltiples simulaciones de dispositivos, es crucial que la base de datos maneje bien la consistencia y confiabilidad de la información almacenada.

Además, su capacidad para gestionar bases de datos complejas y de gran volumen es uno de los principales factores por los que fue seleccionado. La gestión eficiente de múltiples tablas relacionadas, como la energía solar, el almacenamiento en baterías y los aerogeneradores, se ve facilitada por la robustez de PostgreSQL [22].

2.5.2. Escalabilidad

Una de las ventajas clave de PostgreSQL es su capacidad para escalar a medida que aumentan las necesidades de almacenamiento de datos. Esto es esencial en este sistema, donde se espera que el volumen de datos aumente con el tiempo, ya que los dispositivos simulados continúan generando información en función de las peticiones realizadas desde el *backend*.

PostgreSQL permite manejar bases de datos de tamaños considerables, garantizando un rendimiento óptimo tanto para sistemas pequeños como para aquellos que requieren una escalabilidad vertical u horizontal en un entorno de producción a gran escala [23].

2.5.3. Compatibilidad con SQL y extensibilidad

PostgreSQL es totalmente compatible con SQL, lo que permite realizar consultas complejas y altamente personalizables en los datos almacenados. En este proyecto, las consultas SQL juegan un papel clave al permitir al *backend* interactuar eficientemente con la base de datos para almacenar y recuperar información de manera rápida.

Además, PostgreSQL es extensible, lo que significa que se puede adaptar y ampliar con nuevas funcionalidades a medida que el sistema lo requiera. Esto es útil en proyectos como este, donde es posible que se necesiten nuevas funcionalidades a largo plazo, como optimizaciones en la búsqueda o análisis avanzado de los datos generados [24].

2.5.4. Soporte para tipos de datos complejos

PostgreSQL ofrece un soporte excelente para tipos de datos complejos, lo que es particularmente útil en este sistema. Los datos generados por los dispositivos simulados no son simplemente valores numéricos simples, también incluyen estructuras más complejas que deben ser almacenadas de manera eficiente. Por ejemplo, los datos simulados de un aerogenerador incluyen variables relacionadas con la velocidad del viento y la producción de energía, que necesitan un tratamiento adecuado.

PostgreSQL permite definir estructuras personalizadas y utilizar tipos de datos avanzados para optimizar el almacenamiento y las consultas de estos datos, garantizando la integridad y el acceso rápido a la información relevante [25].

2.5.5. Compatibilidad con Python y *SQLAlchemy*

Otro motivo para seleccionar PostgreSQL es su compatibilidad con Python y *SQLAlchemy*, que se utiliza en el *backend* para gestionar la interacción con la base de datos. *SQLAlchemy* es una biblioteca de mapeo objeto-relacional (ORM) que facilita la traducción de las operaciones de bases de datos en consultas SQL, haciendo que el desarrollo sea más intuitivo y menos propenso a errores.

Este enfoque reduce la complejidad de escribir consultas SQL manualmente y permite que las operaciones con la base de datos sean más eficientes y seguras, minimizando la posibilidad de errores de lógica o sintaxis durante la manipulación de datos en el sistema [15].

2.5.6. Seguridad y control de acceso

PostgreSQL cuenta con un sistema robusto de seguridad que incluye autenticación, roles y permisos detallados para gestionar el acceso a la base de datos. En este sistema, donde los datos generados por los dispositivos simulados podrían ser sensibles, es importante garantizar que solo los usuarios y sistemas autorizados puedan acceder o modificar los datos almacenados.

El control de acceso granular y la capacidad de encriptar los datos son características esenciales que permiten asegurar que la información esté protegida contra accesos no autorizados o pérdidas de datos [20].

2.5.7. Manejo de transacciones y ACID

PostgreSQL sigue el modelo ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), lo que asegura que cada transacción en la base de datos se ejecute de manera segura y confiable. Esto significa que si ocurre un fallo durante el procesamiento de una transacción (por ejemplo, al insertar datos de simulación), la base de datos garantiza que los datos permanecen en un estado consistente y no corrupto.

En proyectos como este, donde las simulaciones generan grandes volúmenes de datos a intervalos frecuentes, la integridad transaccional de PostgreSQL es crucial para evitar inconsistencias o pérdidas de información crítica [26].

2.6. *Frontend*

El *frontend* es la interfaz de usuario que permite visualizar y analizar en tiempo real los datos generados por las máquinas virtuales. Está diseñado para ofrecer una experiencia sencilla y eficiente, permitiendo a los usuarios interactuar con la información simulada de los dispositivos (energía solar, baterías y aerogenerador) de manera gráfica e intuitiva.

Las razones para la estructura del *frontend* son las siguientes.

2.6.1. Interfaz gráfica intuitiva

La elección de un *frontend* con una interfaz gráfica fácil de usar es crucial para asegurar que los usuarios puedan interpretar y analizar los datos sin necesidad de conocimientos técnicos avanzados. Los gráficos y las tablas permiten visualizar tendencias en la producción de energía solar, el almacenamiento en baterías y la operación del aerogenerador, facilitando la toma de decisiones basada en los datos presentados.

Este enfoque visual reduce la complejidad de interpretar datos numéricos, lo que mejora la experiencia del usuario final, ya sea que estén monitoreando las simulaciones o analizando el rendimiento del sistema en tiempo real.

2.6.2. Interacción con el *backend* mediante API REST

La arquitectura del *frontend* está diseñada para comunicarse con el *backend* a través de una API REST. Esto permite una interacción fluida y eficiente entre la interfaz y los datos almacenados en el servidor. Mediante peticiones HTTP (GET), el *frontend* obtiene los datos procesados por el *backend* y los presenta en formato gráfico para facilitar su análisis.

La separación de responsabilidades entre el *frontend* y el *backend* mediante API REST es una decisión clave de diseño, ya que permite que ambos componentes evolucionen de manera independiente. Esto facilita el mantenimiento del sistema y permite mejoras tanto en el rendimiento de la interfaz como en el procesamiento de datos del *backend*.

2.6.3. Escalabilidad y modularidad

El *frontend* se ha diseñado teniendo en cuenta la escalabilidad del sistema. A medida que se añadan nuevas simulaciones o dispositivos, la interfaz puede adaptarse fácilmente para incluir nuevas métricas y gráficos. Esta modularidad es crucial para proyectos que evolucionan con el tiempo, ya que permite integrar nuevas funcionalidades sin necesidad de reescribir grandes partes del código.

2.6.4. Interactividad y experiencia del usuario

El *frontend* proporciona una experiencia de usuario interactiva, donde los usuarios pueden explorar los datos simulados en tiempo real. La interfaz ofrece la posibilidad de filtrar, ordenar o seleccionar distintos rangos de datos, lo que permite un análisis más detallado de las simulaciones. Esta capacidad de interacción directa con los datos mejora la experiencia del usuario y ofrece mayor control sobre la visualización de la información.

La utilización de gráficos interactivos permite ver las tendencias y patrones de manera más clara, facilitando la interpretación de grandes volúmenes de datos de manera rápida y efectiva.

CAPÍTULO 3. FLUJO DE DATOS

El flujo de datos en este sistema es un proceso clave que asegura la comunicación efectiva entre las máquinas virtuales (donde se generan los datos), el *backend* (donde se procesan y almacenan), y el *frontend* (donde se visualizan). A continuación, se explican en detalle los pasos involucrados en este proceso.

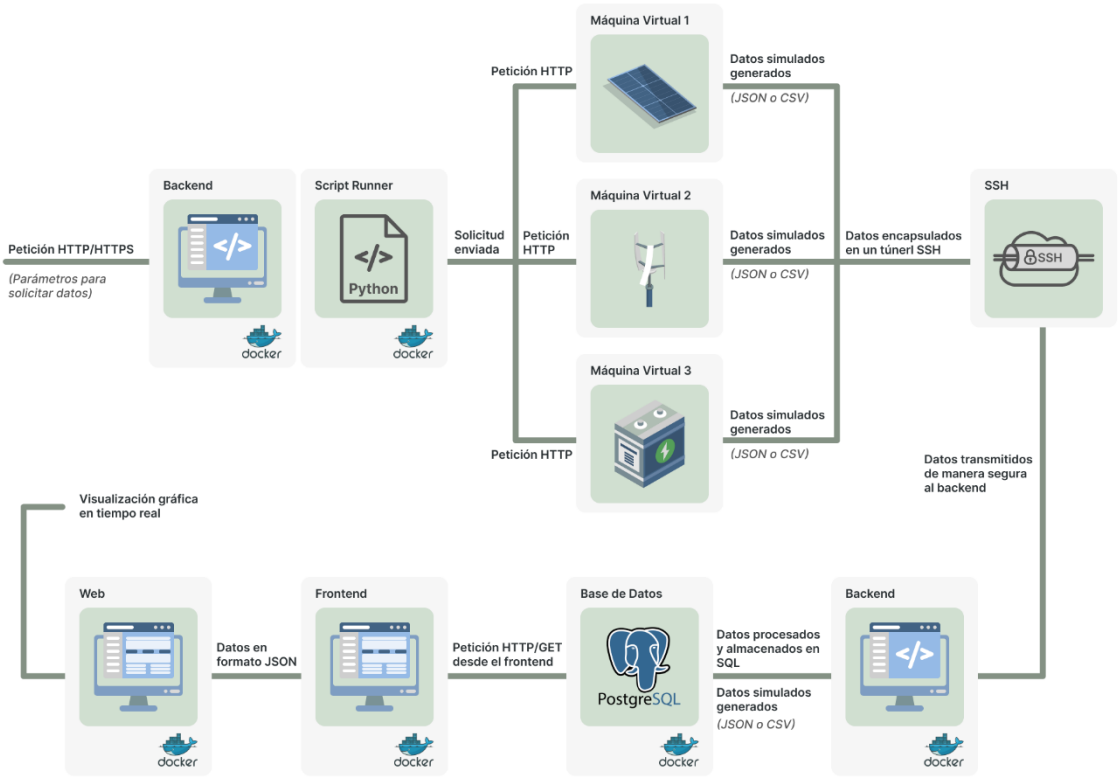

Figura 11. Esquema general del flujo de datos.

3.1. Generación de datos en las máquinas virtuales

Las máquinas virtuales son responsables de simular los datos de dispositivos de energía solar, almacenamiento en baterías y energía eólica. Estos datos no se generan automáticamente a intervalos, sino que se producen bajo demanda en respuesta a las peticiones realizadas por el *backend* a través del script runner. Este enfoque basado en peticiones permite optimizar el uso de recursos y gestionar los datos de manera eficiente.

Cada máquina virtual utiliza un servidor *Flask* para exponer una API que permite la comunicación con el *backend. Flask* maneja las solicitudes HTTP entrantes y genera los datos solicitados en tiempo real. Este proceso es el siguiente:

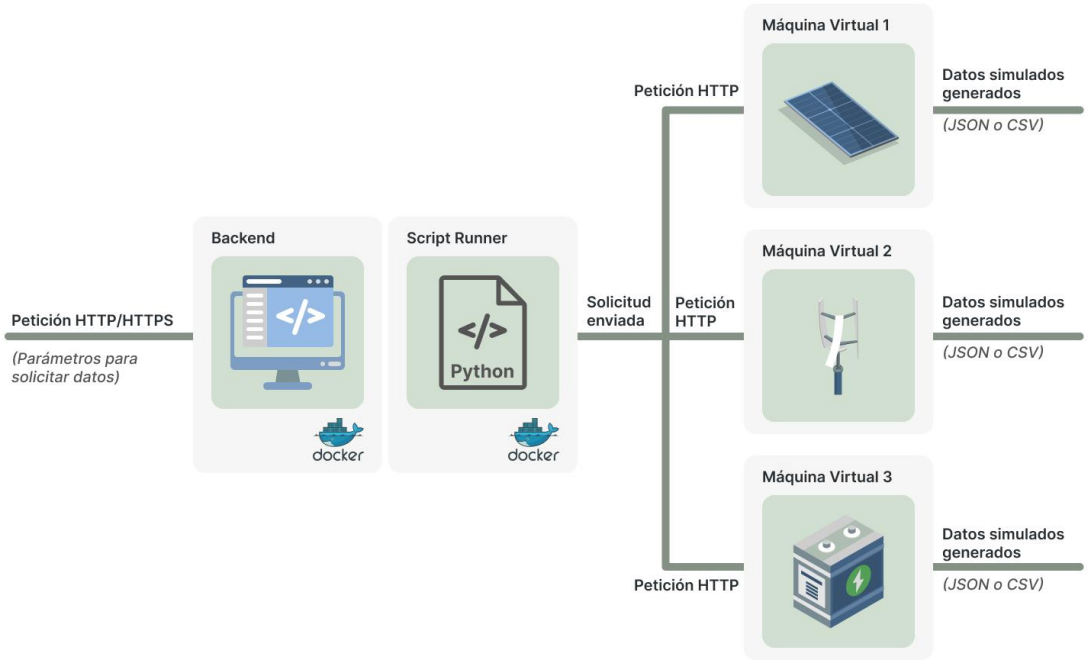

Figura 12. Flujo entre el *backend* y las máquinas virtuales.

1. **Solicitud de datos**: El *backend*, a través del script runner, envía una petición HTTP a la API *Flask* de la máquina virtual solicitando datos. Esta petición puede ser una solicitud GET (para obtener datos) o una POST (si se envían parámetros específicos).
2. **Generación de datos**: Los scripts de simulación generan los datos simulados en función de las características del dispositivo que están replicando (energía solar, estado de las baterías o potencia generada por el aerogenerador, etc).
3. **Formato de los datos**: Los datos generados son estructurados en formato JSON o CSV (dependiendo de la implementación), lo que facilita su posterior procesamiento y almacenamiento. Estos datos incluyen variables clave como la producción de energía, voltaje, corriente, velocidad del viento, entre otros.
4. **Respuesta al *backend***: Los datos simulados son devueltos al *backend* como respuesta a la petición HTTP, encapsulados dentro del túnel SSH para asegurar su transmisión.

Este enfoque garantiza que los datos se generen de manera dinámica y específica, optimizando la eficiencia del sistema.

3.2. Transmisión segura de los datos

Una vez que los datos han sido generados por las máquinas virtuales, es crucial garantizar su transmisión segura hacia el *backend*. Para ello, el sistema emplea un túnel SSH (*Secure Shell*), que garantiza la confidencialidad, integridad y autenticidad de los datos durante la transmisión.

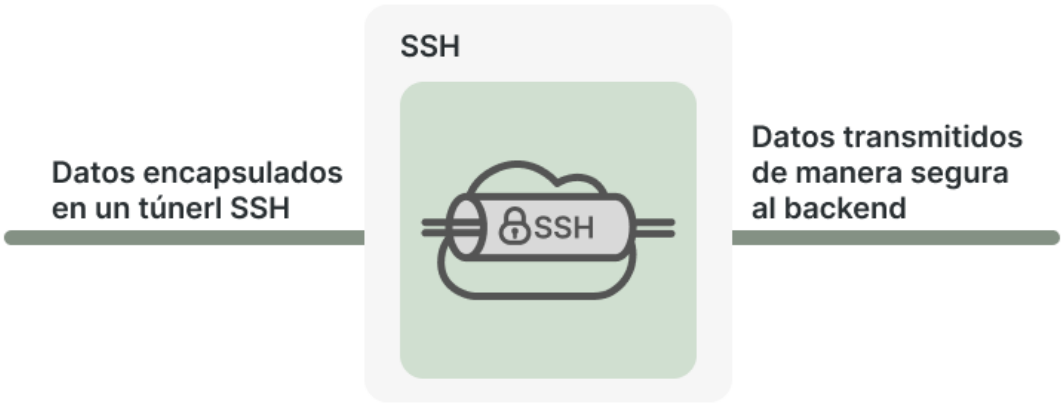

Figura 13. Flujo de datos del túnel SSH.

1. **Túnel SSH**: El túnel SSH crea una conexión segura entre las máquinas virtuales y el *backend*. SSH utiliza cifrado asimétrico para autenticar la conexión inicial, y luego establece una sesión cifrada mediante cifrado simétrico (generalmente utilizando el algoritmo AES). Esto asegura que los datos transmitidos no puedan ser interceptados ni modificados por terceros.
2. **Encapsulación de HTTP**: Las peticiones HTTP que viajan entre el *backend* y las máquinas virtuales son encapsuladas dentro del túnel SSH, lo que proporciona una capa adicional de seguridad. Esto es esencial para proteger tanto los datos como la comunicación en sistemas distribuidos. El encapsulamiento en SSH refuerza esta protección al evitar vulnerabilidades en redes externas.
3. **Transferencia controlada**: A diferencia de una transmisión continua de datos, el sistema solo transmite información cuando es solicitada. Esto optimiza el uso de recursos de red y asegura que los datos solo viajan cuando se necesitan, reduciendo la posibilidad de exposición innecesaria.

En resumen, la utilización de SSH y la encapsulación de las peticiones HTTP dentro del túnel aseguran que el sistema funcione en un entorno seguro y que los datos sensibles estén siempre protegidos durante su transmisión.

3.3. Procesamiento y almacenamiento de Datos en el *backend*

Una vez que los datos han sido recibidos desde las máquinas virtuales, el *backend* se encarga de procesarlos y almacenarlos en la base de datos PostgreSQL para su posterior consulta y análisis.

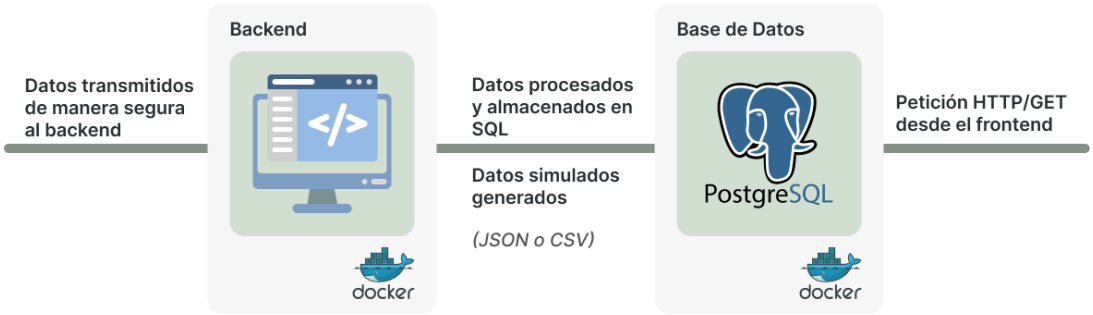

Figura 14. Flujo de datos entre el *backend* y base de datos.

1. **Recepción de datos mediante GET/POST**: El *backend*, utilizando *Flask*, recibe los datos generados por las máquinas virtuales a través de peticiones HTTP. Dependiendo de la configuración del sistema, los datos pueden ser recibidos mediante peticiones GET (cuando el *backend* solicita datos) o peticiones POST (cuando los datos son enviados por las máquinas). Una vez recibidos, los datos pasan por un proceso de validación para asegurar que estén correctamente formateados y que no haya errores o inconsistencias.
2. **Procesamiento de datos**: El *backend* ejecuta cualquier lógica necesaria para procesar los datos antes de almacenarlos. Por ejemplo, puede agregar metadatos como marcas de tiempo, identificar relaciones entre diferentes tipos de datos (como la correlación entre la energía generada por los paneles solares y el estado de carga de las baterías), o realizar cálculos adicionales.
3. **Almacenamiento en PostgreSQL usando SQL**: Una vez procesados, los datos se almacenan en la base de datos PostgreSQL. Se utiliza el lenguaje SQL para insertar los datos en las tablas correspondientes. Cada máquina virtual tiene su propia tabla en la base de datos, lo que facilita la consulta y gestión de los datos según su origen.

3.4. Visualización en el *frontend*

El *frontend* está diseñado para proporcionar a los usuarios una vista gráfica e interactiva de los datos simulados generados por las máquinas virtuales y almacenados en el *backend*. La visualización de estos datos en tiempo real es un componente clave del sistema, ya que permite a los usuarios monitorear el rendimiento de los dispositivos simulados de forma intuitiva.

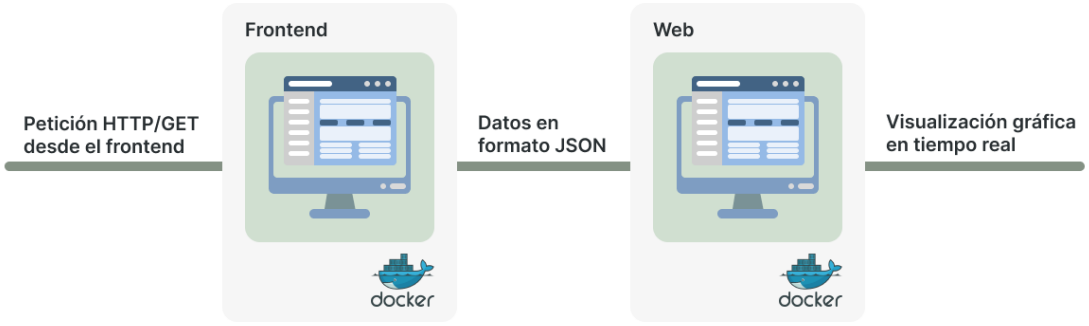

Figura 15. Flujo de datos entre el *frontend* y *backend*.

1. **Acceso a los datos mediante API/REST**: El *frontend* se comunica con el *backend* mediante peticiones API REST. Estas peticiones permiten al *frontend* solicitar datos específicos almacenados en la base de datos. Los datos se recuperan mediante solicitudes HTTP GET, que son manejadas por el *backend* y transmitidas de manera segura al *frontend*. La arquitectura *RESTful* asegura una separación clara entre la lógica del *backend* y la presentación en el *frontend*, lo que facilita el mantenimiento y escalabilidad del sistema.
2. **Visualización en tiempo real**: Los datos obtenidos por el *frontend* se presentan a los usuarios en forma de gráficos interactivos, tablas y otras visualizaciones. Esto permite a los usuarios explorar los datos de simulación, analizar tendencias y tomar decisiones basadas en la información presentada. Las visualizaciones son dinámicas y se actualizan en tiempo real, lo que asegura que los usuarios siempre tengan acceso a los datos más recientes generados por las simulaciones.

Este enfoque garantiza que el sistema sea intuitivo y accesible, proporcionando a los usuarios una manera fácil de interpretar grandes volúmenes de datos sin necesidad de analizar la información en formato bruto.

CAPÍTULO 4. PROTOCOLOS INVOLUCRADOS

El sistema utiliza una variedad de protocolos estándar para garantizar la transmisión segura de datos, la comunicación eficiente entre las diferentes partes del sistema, y el almacenamiento confiable de los datos generados. A continuación, se describen los principales protocolos utilizados en cada etapa del flujo de datos.

4.1. SSH (*Secure Shell*) para asegurar la transmisión

El protocolo SSH se encarga de asegurar la transmisión de los datos entre las máquinas virtuales y el *backend*. SSH es fundamental para garantizar que los datos que se generan y transmiten no sean interceptados o alterados durante su transmisión. Sus principales características incluyen:

- **Cifrado de los Datos:** SSH utiliza un cifrado fuerte para asegurar que toda la información transmitida esté protegida durante su transferencia. Esto impide que cualquier tercero pueda acceder a los datos, lo cual es esencial cuando se manejan simulaciones de dispositivos que pueden contener información sensible.
- **Autenticación Segura:** SSH emplea autenticación basada en claves públicas y privadas, lo que asegura que solo las máquinas virtuales autorizadas puedan conectarse al *backend*. Este mecanismo de autenticación refuerza la seguridad en el sistema.
- **Encapsulación de HTTP:** Las peticiones HTTP que viajan entre el *backend* y las máquinas virtuales son encapsuladas dentro de un túnel SSH. Esto asegura que, incluso si se usa una API HTTP, los datos transmitidos estén siempre protegidos por una capa de cifrado adicional.

4.2. HTTP para la comunicación entre *backend* y máquinas virtuales

El protocolo HTTP se utiliza para gestionar la comunicación entre las máquinas virtuales y el *backend*. En este contexto, HTTP se emplea para manejar las peticiones y respuestas entre estos componentes.

- **HTTP (*Hypertext Transfer Protocol*):** Es el protocolo utilizado para el envío de solicitudes GET y POST entre las máquinas virtuales y el *backend*. Cada vez que el *backend* solicita datos simulados o las máquinas virtuales envían información, se utiliza HTTP para estructurar y gestionar estas interacciones.
    - **Ejemplo:** El *backend* puede enviar una solicitud GET a las máquinas virtuales para solicitar datos específicos, o recibir una solicitud POST con datos generados en formato JSON o CSV.

4.3. SQL para el almacenamiento de datos en las bases de datos

SQL (*Structured Query Language*) es el protocolo estándar utilizado para gestionar el almacenamiento y la consulta de datos en la base de datos PostgreSQL. SQL permite ejecutar

operaciones estructuradas en la base de datos, garantizando que los datos generados por las máquinas virtuales sean almacenados y consultados de manera eficiente.

- **Inserción de Datos:** Una vez que el *backend* recibe los datos generados por las máquinas virtuales, estos se almacenan en la base de datos mediante sentencias SQL. Las sentencias SQL permiten insertar los datos en tablas estructuradas, separando los datos por origen (energía solar, baterías, aerogenerador).
    - **Ejemplo:** Una sentencia SQL de tipo INSERT se ejecuta para almacenar los datos de producción solar, estado de carga de las baterías o potencia generada por el aerogenerador.
- **Consultas de Datos:** Las consultas SQL son utilizadas cuando el *frontend* solicita datos almacenados. SQL permite extraer información de manera rápida y precisa, facilitando la visualización de los datos simulados en tiempo real.
    - **Ejemplo:** El *backend* ejecuta una consulta SELECT para recuperar datos históricos sobre la energía solar generada o el estado de las baterías, y luego los envía al *frontend* para ser visualizados en gráficos.

CAPÍTULO 5. INFRAESTRUCTURA TÉCNICA

Una vez explicados los flujos tanto de componentes como de datos necesarios para realizar el proyecto, así como los protocolos utilizados, se puede pasar a explicar el código de cada componente. Este capítulo será el más extenso de todos, pero también el más enriquecedor y esclarecedor y en el siguiente capítulo, será más visual.

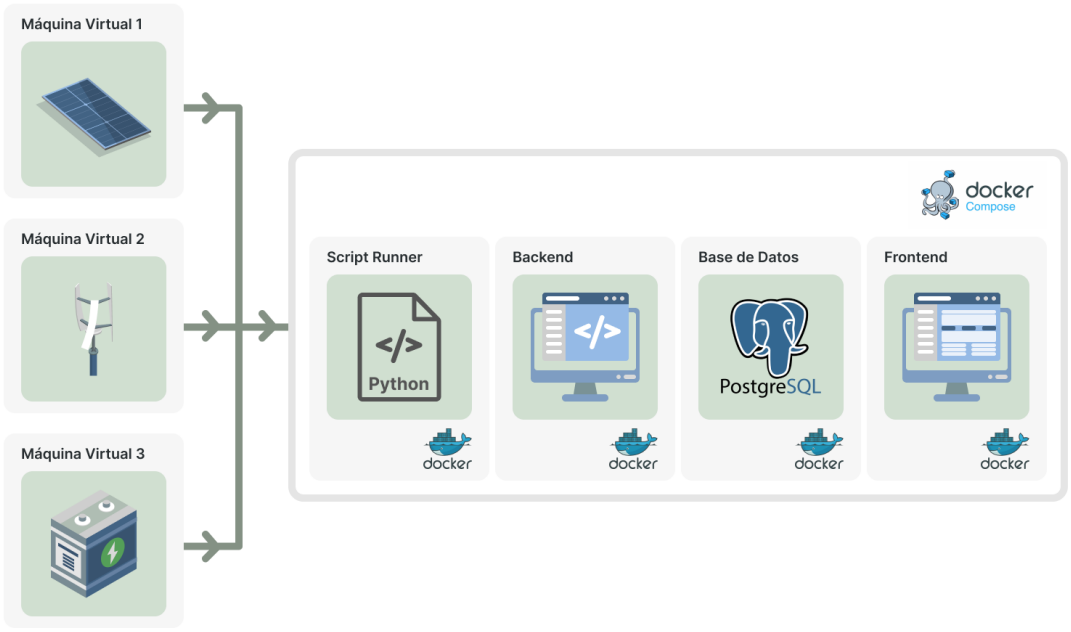

Figura 16. Esquema principal de toda la infraestructura técnica.

Los temas que se tratarán a continuación seguirán el orden del flujo de datos, mostrado a continuación:

1. **Máquinas virtuales:** En este apartado se detallará desde cómo se han instalado las máquinas hasta los scripts de simulación utilizados en las tres máquinas.
2. ***Docker Compose*:** Como se mencionó anteriormente, el proyecto está compuesto por varios contenedores que pueden comunicarse entre ellos. Durante este apartado se explicará cómo se han creado los contenedores y las ventajas de utilizar Docker Compose.
3. **Script runner:** Este componente es uno de los contenedores de *Docker Compose*. En este apartado se explicará el código que permite solicitar los datos a las máquinas virtuales y enviarlos al *backend*.

4. ***Backend* + base de datos:** Son dos contenedores de Docker. Del mismo modo que el script runner, en este apartado se explicará el código del *backend* y cómo se consultan y procesan los datos desde el script runner hasta la base de datos.
5. ***Frontend*:** Este es el último contenedor de Docker Compose, que contiene la aplicación de la web.

En conclusión, el proyecto se divide en dos partes: una parte son las máquinas virtuales, que simulan *las BeagleBone Black Industrial*, y la otra parte es Docker Compose, que contiene todo lo necesario para visualizar y controlar dichos datos a través de una interfaz gráfica.

5.1. Máquinas virtuales

Los programas utilizados para crear las máquinas virtuales han sido el hipervisor *VirtualBox* y el intérprete *Visual Studio Code* para realizar los scripts con el lenguaje Python.

5.1.1. Instalación de Oracle VirtualBox y configuración entorno virtual de Python

En primer lugar, se ha descargado e instalado VirtualBox [25]. Durante la instalación, ha sido necesario seleccionar la opción "*VirtualBox Extension Pack*" para habilitar funciones avanzadas, como el soporte para USB 2.0/3.0 y otros dispositivos virtuales.

Una vez instalado, se procedió a la creación de un entorno virtual de Python, lo cual es una buena práctica a la hora de programar, ya que permite aislar las dependencias de los proyectos y evitar conflictos entre diferentes versiones de paquetes.

El proceso de creación de un entorno virtual puede variar dependiendo del sistema operativo y del terminal utilizado. En este proyecto se ha utilizado Windows como sistema operativo y *Bash* como terminal. Los pasos para la creación del entorno virtual han sido los siguientes:

- Se abre el terminal y se navega hasta el directorio del proyecto con el siguiente comando:

```
cd Desktop/TFM/Creacion_VMs
```

Fragmento código 1. Comando *Bash* navegación entre carpetas

- A continuación, se crea el entorno virtual con el siguiente comando:

```
Python -m venv venv
```

Fragmento código 2. Comando *Bash* instalación entorno virtual

- Una vez creado, se debe activar el entorno con el siguiente comando:

```
Source venv/Scripts/activate
```

Fragmento código 3. Comando *Bash* activación entorno virtual

Con estos tres sencillos pasos, el entorno virtual queda activado, y todas las librerías que se instalen en este proyecto estarán aisladas y se usarán exclusivamente en este TFM.

5.1.2. Creación de máquinas virtuales

Para realizar el proyecto, ha sido necesario un ordenador portátil con suficiente memoria RAM, porque al encender todas las máquinas, consume muchos recursos.

Las tres máquinas son iguales, y tienen las siguientes características:

- Sistema operativo: Linux – Ubuntu
- Memoria RAM: 3 Gb
- Disco de almacenamiento: 10 Gb

Se ha creado un código que automatiza la creación y configuración de tres máquinas virtuales utilizando VirtualBox, instalando el sistema operativo Ubuntu desde una ISO, asignando memoria y configurando la red y almacenamiento. Este código puede ser fácilmente ampliado por otras máquinas o incluso modificar sus parámetros a la vez. A continuación, se va a explicar el código:

- **Librerías importadas y configuración**

```
# Librerías empleadas
import subprocess

import os
```

Fragmento código 4. Librerías empleadas para la creación máquinas virtuales.

Al principio del código es necesario indicar las librerías empleadas, en este caso la librería ***subprocess***, permite ejecutar comandos del sistema operativo desde el script, como si los estuvieras escribiendo en la terminal o el símbolo del sistema. En cuanto a la librería ***os***, dicha librería proporciona funciones para interactuar con el sistema operativo, como la manipulación de rutas de archivos y variables de entorno.

```
# Añadir manualmente la ruta a VBoxManage.exe al PATH
vbox_path = r"C:\Program Files\Oracle\VirtualBox"

os.environ["PATH"] += os.pathsep + vbox_path
```

Fragmento código 5. Configuración ruta VirtualBox.

Se añade la ruta de ***VirtualBox*** al ***PATH*** del sistema para que VBoxManage se pueda ejecutar sin especificar la ruta completa.

- **Función *create_vm:***
  - Crea una máquina virtual usando el comando *VBoxManage createvm*, configurando Ubuntu de 64 bits como sistema operativo.
  - Se ajusta la memoria RAM con *VBoxManage modifyvm* según el valor proporcionado por el usuario.

- La red se configura en modo *bridged* (puente), conectando la máquina virtual directamente a la red del host.
- Se crea un disco duro virtual (10 GB) que se adjunta a la VM a través de un controlador SATA.
- Se conecta una imagen ISO de Ubuntu como unidad óptica para la instalación del sistema operativo.
- Una vez configurada, la VM se inicia con interfaz gráfica mediante el comando *VBoxManage startvm.*

```
def create_vm(vm_name, iso_path, ram_size_mb):
    # Crear la VM
    subprocess.run([
        "VBoxManage", "createvm", "--name", vm_name, "--ostype",
"Ubuntu_64", "--register"
    ])

    # Ajustar la memoria RAM
    subprocess.run([
        "VBoxManage", "modifyvm", vm_name, "--memory",
str(ram_size_mb)
    ])

    # Ajustar el adaptador de red
    subprocess.run([
        "VBoxManage", "modifyvm", vm_name, "--nic1", "bridged", "-
-bridgeadapter1", "Intel(R) Wi-Fi 6 AX201 160MHz"
    ])

    # Crear y añadir un disco duro virtual
    hdd_path = os.path.join(os.path.expanduser("~"),
f"{vm_name}.vdi")
    subprocess.run([
        "VBoxManage", "createhd", "--filename", hdd_path, "--
size", "10240"
    ])
    subprocess.run([
        "VBoxManage", "storagectl", vm_name, "--name", "SATA
Controller", "--add", "sata"
    ])
    subprocess.run([
        "VBoxManage", "storageattach", vm_name, "--storagectl",
"SATA Controller", "--port", "0", "--device", "0", "--type",
"hdd", "--medium", hdd_path
    ])

    # Añadir la unidad óptica con la ISO de Ubuntu
    subprocess.run([
        "VBoxManage", "storageattach", vm_name, "--storagectl",
"SATA Controller", "--port", "1", "--device", "0", "--type",
"dvddrive", "--medium", iso_path
    ])

    # Iniciar la VM
    subprocess.run([
        "VBoxManage", "startvm", vm_name, "--type", "gui"
    ])

    print(f"VM '{vm_name}' created and started.")
```

Fragmento código 6. Función creación de una máquina virtual.

- **Bucle para crear múltiples máquinas**

El script crea tres máquinas virtuales, nombradas secuencialmente como "Ubuntu_VM_1", "Ubuntu_VM_2", y "Ubuntu_VM_3", cada una con 3 GB de RAM.

```
# Directorio de la iso de Ubuntu
iso_path = "ubuntu-24.04-desktop-amd64.iso"

# Creación de las VMs
for i in range(3):
    vm_name = f"Ubuntu_VM_{i+1}"
    ram_size_mb = 3072  # 3 GB

    create_vm(vm_name, iso_path, ram_size_mb)
```

Fragmento código 7. Bucle para crear múltiples máquinas.

5.1.3. Instalación del protocolo SSH en las máquinas

Antes de ejecutar los scripts que simulan los dispositivos, hay que instalar el protocolo SSH en las máquinas y saber la IP de las tres máquinas para poder establecer un túnel seguro de SSH.

Con el siguiente comando se instala el servidor SSH:

```
Sudo apt-get install openssh-server -y
```

Fragmento código 8. Comando *bash* instalación SSH para máquinas virtuales.

Por defecto, el servidor SSH se está ejecutando en el puerto 22. Por otro lado, para saber cuál es la IP de cada máquina, se escribe el comando *"ip a"*.

- IP de la máquina (Placa solar): 192.168.1.54

```
(venv) ubuntu@ubuntu:~/scripts$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:7e:e6:3e brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.54/24 brd 192.168.1.255 scope global dynamic noprefixroute enp0s3
       valid_lft 85843sec preferred_lft 85843sec
    inet6 fe80::a00:27ff:fe7e:e63e/64 scope link
       valid_lft forever preferred_lft forever
```

Figura 17. IP máquina placa solar.

- IP de la máquina (Aerogenerador): 192.168.1.55
- IP de la máquina (Baterías): 192.168.1.56

Una vez instalado el servidor SSH y conociendo las *IPs* de cada máquina, se puede comprobar si se puede conectar desde el portátil a las máquinas. Con el siguiente comando se comprueba las conexiones a las máquinas:

```
$ ssh -p 22 ubuntu@192.168.1.54
```

Fragmento código 9. Comando *bash* conexión a la máquina virtual desde local.

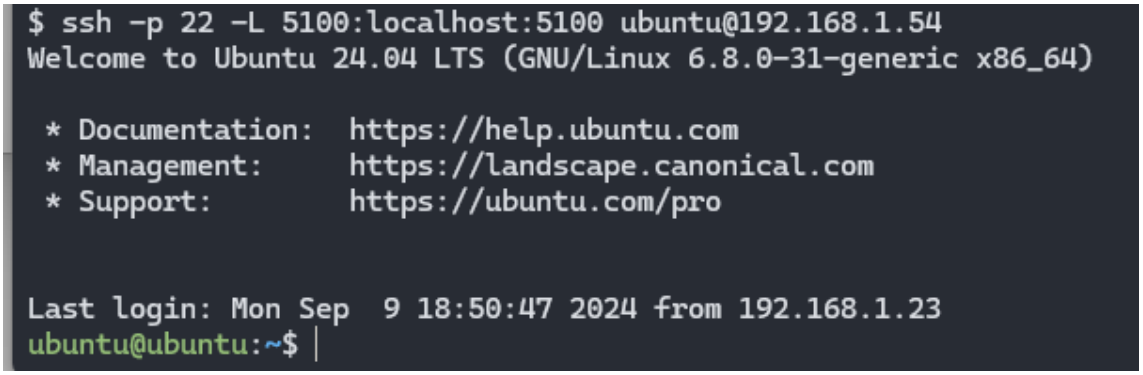

Figura 18. Conexión realizada a la máquina virtual.

5.1.4. Script de simulación de la placa solar

El siguiente código simula la generación de datos de producción de energía solar. Utiliza *FLASK* para exponer una API que permite consultar estos datos y modificar los límites de operación. A lo largo del día, ajusta la radiación solar y, en consecuencia, calcula el voltaje, la intensidad, la potencia y la energía generada en función de las condiciones simuladas.

A continuación, se va a explicar partes de dicho código:

- **Importación de librerías**

```
from flask import Flask, jsonify, request
from flask_cors import CORS  # Importa la extension
CORS
import math
import random
import requests

import datetime
```

Fragmento código 10. Librerías importadas para el script de placa solar.

La librería *Flask*, se utiliza para crear una API web y la librería CORS, permite que la API acepte peticiones de otros dominios, en este caso, el *backend*. La librería *datetime*, se utiliza para obtener la hora actual y generar marcas de tiempo. Las librerías restantes, es para realizar los cálculos.

- **Configuración de Flask y CORS**

```
app = Flask(__name__)
CORS(app)  # Habilita CORS para todas las rutas
```

Fragmento código 11. Configuración de *Flask* y CORS.

- **Definición de límites para voltajes e intensidad**

```
# Límites globales para voltage y current
voltage_min = 220
voltage_max = 240
current_min = 5

current_max = 15
```

Fragmento código 12. Límites globales placa solar.

Se definen los límites globales para el voltaje e intensidad mínimos y máximos paraque se puedan modificar desde la web.

- **Función para calcular voltaje e intensidad basado en la radiación solar**

```
def
get_voltage_current_based_on_radiation(solar_radiatio
n, voltage_max, current_max):
    """
    Calcula la tensión (voltaje) y la corriente
(intensidad) basadas en la radiación solar.
    """
    # Asumimos una relación lineal para simplificar
    # La radiación máxima se toma como 1000 W/m²
    voltage = (solar_radiation / 800) * voltage_max
    # Relación lineal también para la corriente
    current = (solar_radiation / 800) * current_max

    # Limitamos la tensión y la corriente a sus
valores máximos configurados
    voltage = min(voltage, voltage_max)
    current = min(current, current_max)


    return voltage, current
```

Fragmento código 13. Función para cálculo de voltaje e intensidad en función de la radiación solar.

Esta función calcula el voltaje y la intensidad en función de la radiación solar. La radiación solar tiene como valor entre el 0 y 800 W/m2, y el voltaje e intensidad se ajusta proporcionalmente.

- **Función para calcular la radiación solar en función de la hora**

```
def get_solar_radiation_curve(hour):
    """
    Calcula la radiación solar en función de la hora del día
utilizando una función
    que simula la curva de radiación típica en un día soleado.
    """
    # Parámetros de la función para un día completo de 6:00 a 20:00
    sunrise = 6  # 6:00 AM
    sunset = 20  # 8:00 PM
    peak_hour = 14  # 2:00 PM (aproximadamente el pico)

    if hour < sunrise or hour > sunset:
        return 0  # No hay radiación solar fuera de las horas del día

    # Normalización de la hora para que vaya de 0 a 1 durante el día
    normalized_hour = (hour - sunrise) / (sunset - sunrise)

    # Curva parabólica invertida para simular el incremento y
decremento de la radiación
    radiation = -4 * (normalized_hour - 0.5) ** 2 + 1

    # Escalar la radiación para que vaya de 0 a 1000 W/m²
    max_radiation = 800  # Pico máximo de radiación alrededor del
mediodía
    solar_radiation = radiation * max_radiation

    return solar_radiation
```

Fragmento código 14. Función para el cálculo de radiación solar en función de la hora.

Esta función simula una curva de radiación solar durante el día, desde las 6 de la mañana hasta las 8 de la tarde. Genera una curva parabólica simulando un día soleado en Valencia, que tiene como un máximo de 800 W/m2 de radiación solar alrededor de las 2 de la tarde.

- **Función para generar datos de la placa solar**

La función genera los siguientes datos:

  - Radiación solar: calculada en función de la hora actual.
  - Voltaje e intensidad son calculados a partir de la radiación.
  - Potencia y energía generada se calculan en función del voltaje e intensidad.

```
def get_solar_data():
    """
    Genera datos de simulación para la radiación solar, voltaje,
corriente, potencia y energía.
    """
    current_hour = datetime.datetime.now().hour

    solar_radiation = get_solar_radiation_curve(current_hour)  #
W/m²

    # Calcula la tensión y la corriente en función de la radiación
solar
    voltage, current = get_voltage_current_based_on_radiation(
        solar_radiation, voltage_max, current_max)

    # Calcula la potencia
    power = voltage * current  # Potencia en vatios

    # Calcula la energía producida en 1 minuto
    energy = power / 60  # Energía en kWh (para un intervalo de 1
minuto)

    data = {
        "timestamp": datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S"),
        "location": "Valencia, Spain",
        "solar_radiation": solar_radiation,
        "voltage": voltage,
        "current": current,
        "power": power,
        "energy_generated": energy_generated
    }
    return data
    return solar_radiation
```

Fragmento código 15. Función de generación de datos de la placa solar.

- **Ruta para peticiones de los datos simulados de la placa solar**

Esta ruta **GET** devuelve los datos simulados cuando se solicita. Se convierte en formato **JSON** antes de enviarse.

```
@app.route('/solar', methods=['GET'])
def solar_data():
    data = get_solar_data()
    return jsonify(data)
```

Fragmento código 16. Ruta para peticiones de datos placa solar.

```
{
  "current": 5.8775510204081645,
  "energy_generated": 9.212161599333616,
  "energy_wh": 9.212161599333616,
  "power": 552.7296959600169,
  "solar_radiation": 391.83673469387764,
  "timestamp": "2024-09-11 18:42:14",
  "voltage": 94.04081632653063
}
```


Figura 19. Salida datos en formato JSON de la placa solar.

- **Ruta para consultar y modificar los límites**

```
@app.route('/limits', methods=['GET', 'POST'])
def limits():
    global voltage_min, voltage_max, current_min, current_max

    if request.method == 'POST':
        limits_data = request.json
        if 'voltage_max' in limits_data and
limits_data['voltage_max'] is not None:
            voltage_max = limits_data['voltage_max']
        if 'current_max' in limits_data and
limits_data['current_max'] is not None:
            current_max = limits_data['current_max']
        return jsonify({"message": "Limits updated successfully"}),
200

    return jsonify({
        "voltage_max": voltage_max,
        "current_max": current_max

    })
```

Fragmento código 17. Ruta para consultar y modificar límites de la placa solar.

Esta ruta permite dos acciones:

- GET: Consulta los límites actuales de voltaje y corriente.
- POST: Permite modificar los límites máximos para el voltaje y la intensidad con los valores enviados en el cuerpo de la solicitud.

- **Inicio de la aplicación**

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5100)
```

Fragmento código 18. Inicio aplicación de la placa solar.

Se configura la aplicación para que escuche en todas las interfaces de red (`0.0.0.0`) en el puerto **5100**, lo que hace que la API esté accesible externamente en la red.

5.1.5. Script de simulación de la turbina eólica

Del mismo modo que en la placa solar, el siguiente código simula la generación de datos de una turbina eólica, calculando la energía, voltaje, corriente y potencia en función de la velocidad del viento. También se puede obtener los datos a través de la API y modificar los límites de operación.

Las librerías importadas y el inicio son iguales que la placa solar, solo cambia el puerto que en este caso es **5200.**

- **Definición de los límites globales y lista para almacenar valores**

```
# Límites globales para voltage y current
voltage_max = 240
voltage_min = 220
current_max = 20
v_cut_in = 3   # Velocidad de arranque (m/s)
v_nominal = 12  # Velocidad nominal (m/s)
v_cut_out = 20  # Velocidad de corte (m/s)

# Inicialización de la lista para almacenar los últimos 12 valores
energia_intervalos = deque(maxlen=12)  # Solo guarda los últimos 12 valores
```

Fragmento código 19. Definición de límites globales para la turbina eólica.

Se definen varios límites globales, incluyendo el voltaje, la corriente, y los umbrales para la velocidad del viento (velocidad de arranque, nominal y de corte). Estos se pueden ajustar mediante la API.

Se utiliza una estructura ***deque*** para almacenar los últimos 12 valores de energía generada (correspondientes a intervalos de 5 minutos, por lo que esta lista almacenará una hora de datos).

- **Función para obtener datos de la turbina eólica**

```
def get_wind_data():
    # Simulación basada en tramos horarios
    now = datetime.datetime.now()
    hour = now.hour

    # Velocidades del viento típicas (en m/s) en Valencia a lo largo de un
día
    if 0 <= hour < 6:
        wind_speed = random.uniform(3, 5)  # madrugada, vientos bajos
    elif 6 <= hour < 12:
        wind_speed = random.uniform(5, 10)  # mañana, vientos moderados
    elif 12 <= hour < 18:
        wind_speed = random.uniform(10, 20)  # tarde, vientos fuertes
    else:
        wind_speed = random.uniform(5, 10)  # noche, vientos moderados
```

Fragmento código 20. Velocidad del viento en función de la hora.

La función ***get_wind_data*,** genera datos simulados de la turbina eólica en función de la hora. En función de la hora, se determina la velocidad del viento, en Valencia, la velocidad máxima suele ser de 20m/s.

```
    if wind_speed < v_cut_in or wind_speed > v_cut_out:
        voltage = 0
        current = 0
        power = 0
    elif v_cut_in <= wind_speed < v_nominal:
        # Escalado lineal entre el voltaje mínimo y máximo
        voltage = voltage_min + (wind_speed - v_cut_in) / \
            (v_nominal - v_cut_in) * (voltage_max - voltage_min)
        # Potencia proporcional al cubo de la velocidad del viento
        power = 0.5 * 1.225 * 1 * wind_speed**3 * 0.3  # Asumiendo Cp=0.3
y A=1 m²
        current = power / voltage
    else:
        # Potencia constante a partir de la velocidad nominal
        voltage = voltage_max
        power = voltage * current_max
        current = current_max
```

Fragmento código 21. Cálculo de la potencia, voltaje e intensidad de la turbina eólica.

El fragmento anterior, se calcula la potencia, voltaje e intensidad en función del viento:

- Si está por debajo de la velocidad de arranque (*v_cut_in*) o por encima de la velocidad de corte (*v_cut_out*), no se genera energía.
- Si la velocidad está entre los límites, el voltaje y la corriente se calculan en función de la velocidad del viento.
- La potencia se calcula proporcionalmente al cubo de la velocidad del viento para simular el comportamiento de la turbina eólica.

A continuación, se calcula la energía generada y se actualiza la lista de los 12 valores, para calcular la energía generada en una hora.

```
    # Energía generada en el intervalo de 5 min en Wh
    energy_generated = power * (1/60)

    # Agregar el valor actual a la lista (si no hay 12 valores aún, se agregarán)
    energia_intervalos.append(energy_generated)

    # Energía total en la última hora (suma de los últimos 12 valores)
    energia_total_hora = sum(energia_intervalos)

    data = {
        "timestamp": datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        "wind_speed": wind_speed,
        "voltage": voltage,
        "current": current,
        "power": power,
        "energy_generated": energy_generated,
        "energy_wh": energia_total_hora
    }

    return data
```

Fragmento código 22. Cálculo de energía generada y retorno de los datos simulados de la turbina eólica.

- **Ruta para las peticiones de datos simulados de la turbina eólica**

Los datos se devuelven en el formato JSON cuando son pedidos por la API.

```
@app.route('/wind', methods=['GET'])
def wind():
    return jsonify(get_wind_data())
```

Fragmento código 23. Ruta para las peticiones de datos simulados de la turbina eólica.

Un ejemplo de salida de datos sería la siguiente:

```
{
   "current": 0.16434798707029225,
   "energy_generated": 0.6200916357122196,
   "energy_wh": 18.517699549991555,
   "power": 37.20549814273318,
   "timestamp": "2024-09-11 19:18:55",
   "voltage": 226.38243890885167,
   "wind_speed": 5.872097508983247
}
```


Figura 20. Salida datos en formato JSON de la turbina eólica.

- **Ruta para consultar y modificar los límites**

La siguiente ruta permite:

- GET: Consultar los límites actuales de voltaje, corriente y velocidades del viento.
- POST: Actualizar esos límites con los valores enviados en la solicitud.

```
@app.route('/limits', methods=['GET', 'POST'])
def limits():
    global voltage_min, voltage_max, current_max, v_cut_in, v_nominal,
v_cut_out

    if request.method == 'POST':
        limits_data = request.json
        if 'voltage_max' in limits_data and limits_data['voltage_max']
is not None:
            voltage_max = limits_data['voltage_max']
        if 'voltage_min' in limits_data and limits_data['voltage_min']
is not None:
            voltage_min = limits_data['voltage_min']
        if 'current_max' in limits_data and limits_data['current_max']
is not None:
            current_max = limits_data['current_max']
        if 'v_cut_in' in limits_data and limits_data['v_cut_in'] is
not None:
            v_cut_in = limits_data['v_cut_in']
        if 'v_nominal' in limits_data and limits_data['v_nominal'] is
not None:
            v_nominal = limits_data['v_nominal']
        if 'v_cut_out' in limits_data and limits_data['v_cut_out'] is
not None:
            v_cut_out = limits_data['v_cut_out']
        return jsonify({"message": "Limits updated successfully"}),
200

    return jsonify({
        "voltage_max": voltage_max,
        "voltage_min": voltage_min,
        "current_max": current_max,
        "v_cut_in": v_cut_in,
        "v_nominal": v_nominal,
        "v_cut_out": v_cut_out
    })
```

Fragmento código 24. Ruta para modificar y consultar los límites de la turbina eólica.

5.1.6. Script de simulación del gestor de las baterías

Este código simula el comportamiento de una batería, calculando su capacidad total y otros parámetros clave como el estado de carga, resistencia interna, temperatura, y eficiencia. A través de una API creada con *Flask*, permite obtener datos de la batería y ajustar los límites de operación. Sin embargo, hay una diferencia respecto a los dos scripts anteriores, este dispositivo pide antes de calcular todos los datos simulados, el voltaje e intensidad de la placa solar y la turbina eólica.

Las librerías utilizadas y el inicio de la aplicación *Flask* es la misma que en los dos códigos anteriores, solo cambia el puerto que es **5300.**

```
internal_resistance_min = 0.1
internal_resistance_max = 0.5
temperature_min = 15
temperature_max = 35


capacidad_intervalos = deque(maxlen=12)
```

Fragmento código 25. Límites globales de las baterías.

Se definen la resistencia interna y la temperatura como límites para poder modificarlo desde la web. La *capacidad_intervalos*, es una lista que guarda los últimos 12 valores para calcular la capacidad de carga total en una hora.

- **Función para calcular la capacidad de carga total**

```
def calculate_capacity(current_solar, current_generator):
    # Capacidad de la placa solar
    capacity_solar = current_solar * 0.08
    # Capacidad del generador
    capacity_generator = current_generator * 0.08
    # Capacidad total es la suma de ambas capacidades
    total_capacity = capacity_solar + capacity_generator


    return total_capacity
```

Fragmento código 26. Función para el cálculo de la capacidad de carga total.

La capacidad de carga total es la suma de la capacidad de carga de la placa solar y de la turbina eólica. Está calculada en un intervalo de 5 minutos.

- **Función para el cálculo de la eficiencia**

```
def calculate_efficiency(voltage, current):
    # Un ejemplo simple de cómo podrías relacionar la eficiencia con
el voltaje y la corriente
    if 12 <= voltage <= 14 and 5 <= current <= 10:
        return 95  # Alta eficiencia cuando los valores están en un
rango óptimo
    elif voltage < 12 or voltage > 14 or current < 5 or current > 10:
        return 85  # Eficiencia más baja cuando los valores están
fuera del rango óptimo
    else:
        return 90  # Eficiencia media por defecto
```

Fragmento código 27. Función para el cálculo de eficiencia.

La función calcula la eficiencia de la batería en función del voltaje y la intensidad, el valor tendrá alta eficiencia (95%) cuando los valores están dentro de un rango óptimo, de lo contrario la eficiencia será más baja del 85%.

- **Función para generar datos de la batería**

```
def get_battery_data(voltage_solar=None, current_solar=None,
voltage_generator=None, current_generator=None):
    global internal_resistance_min, internal_resistance_max,
temperature_min, temperature_max

    # Verifica si los valores de voltaje e intensidad fueron
proporcionados para la placa solar
    current_solar = current_solar if current_solar is not None else
random.uniform(
        1, 10)

    # Verifica si los valores de voltaje e intensidad fueron
proporcionados para el generador
    current_generator = current_generator if current_generator is
not None else random.uniform(
        1, 10)

    # Calcula la capacidad usando la función anterior
    capacity = calculate_capacity(current_solar, current_generator)
    capacidad_intervalos.append(capacity)
    capacidad_total_hora = sum(capacidad_intervalos)

    # Calcula la eficiencia de carga y descarga
    charge_efficiency = calculate_efficiency(voltage_solar,
current_solar)
    discharge_efficiency = calculate_efficiency(
        voltage_generator, current_generator)

    # Define variables adicionales para la simulación
    soc = random.uniform(20, 100)  # State of Charge as a percentage
    soh = random.uniform(80, 100)  # State of Health as a percentage
    internal_resistance = random.uniform(
        internal_resistance_min, internal_resistance_max)  #
Milliohms
    temperature = random.uniform(
        temperature_min, temperature_max)  # Degrees Celsius
```

Fragmento código 28. Función de generación de datos de las baterías.

La función pide el voltaje e intensidad de la placa solar y de la turbina, si no se tiene valor, la función genera valores aleatorios dentro de un rango típico para las fuentes de energía.

Otros parámetros como el estado de carga (*SoC*), estado de salud (*SoH*), resistencia interna y la temperatura, son generados de forma aleatoria. Los datos que se envían son los siguientes:

```
    battery = {
        "capacity_solar": (current_solar*0.08)/capacity*100,
        "capacity_generator": (current_generator*0.08)/capacity*100,
        # Capacidad total calculada en Watts (o cualquier otra
unidad que elijas)
        "capacity": capacity,  # Capacidad Ah en un intervalo 5 min
        "capacity_ah": capacidad_total_hora,  # Capacidad Ah en un
intervalo 5 min
        "soc": soc,  # State of Charge
        "soh": soh,  # State of Health
        "internal_resistance": internal_resistance,  # Internal
resistance in milliohms
        "temperature": temperature,  # Temperature in degrees
Celsius
        # Eficiencia de carga dependiente del voltaje e intensidad
        "charge_efficiency": charge_efficiency,
        # Eficiencia de descarga dependiente del voltaje e
intensidad
        "discharge_efficiency": discharge_efficiency,

    }

    data = {
        "timestamp": datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S"),
        "location": "Valencia, Spain",
        "battery": battery
    }
    return data
```

Fragmento código 29. Variables que se generan de la simulación de baterías.

- **Ruta para peticiones de la simulación de baterías**

```
@app.route('/batteries', methods=['GET'])
def battery_data():
    # Retrieve custom voltage and current for solar panel and
generator from request parameters
    voltage_solar = request.args.get('voltage_solar', type=float)
    current_solar = request.args.get('current_solar', type=float)
    voltage_generator = request.args.get('voltage_generator',
type=float)
    current_generator = request.args.get('current_generator',
type=float)

    data = get_battery_data(voltage_solar=voltage_solar,
current_solar=current_solar,
                            voltage_generator=voltage_generator,
current_generator=current_generator)
    return jsonify(data)
```

Fragmento código 30. Ruta para peticiones de los datos simulados de las baterías.

La ruta permite obtener datos de la batería. Los valores de voltaje y corriente para el panel solar y el generador se pueden pasar como parámetros opcionales. Se muestra a continuación un ejemplo de datos simulados por las baterías:


```
{
   "capacity": 230.6707860481162,
   "capacity_ah": 1304.5272609048764,
   "capacity_generator": 49.573840059686816,
   "capacity_solar": 50.426159940313184,
   "charge_efficiency": 95,
   "discharge_efficiency": 95,
   "internal_resistance": 0.29296823155130236,
   "soc": 90.60049137929423,
   "soh": 95.95537068368799,
   "temperature": 31.509843168046906,
   "timestamp": "2024-09-11 20:14:26"
}
```


Figura 21. Ejemplo de datos simulados de las baterías en formato JSON.

- **Ruta para consultar y modificar los límites**

```
@app.route('/limits', methods=['GET', 'POST'])
def limits():
    global internal_resistance_min, internal_resistance_max,
temperature_min, temperature_max

    if request.method == 'POST':
        limits_data = request.json
        if 'internal_resistance_min' in limits_data and
limits_data['internal_resistance_min'] is not None:
            internal_resistance_min =
limits_data['internal_resistance_min']
        if 'internal_resistance_max' in limits_data and
limits_data['internal_resistance_max'] is not None:
            internal_resistance_max =
limits_data['internal_resistance_max']
        if 'temperature_min' in limits_data and
limits_data['temperature_min'] is not None:
            temperature_min = limits_data['temperature_min']
        if 'temperature_max' in limits_data and
limits_data['temperature_max'] is not None:
            temperature_max = limits_data['temperature_max']
        return jsonify({"message": "Limits updated successfully"}),
200

    return jsonify({
        "internal_resistance_min": internal_resistance_min,
        "internal_resistance_max": internal_resistance_max,
        "temperature_min": temperature_min,
        "temperature_max": temperature_max,
    })
```

Fragmento código 31. Ruta para consultar y modificar los límites de las baterías.

La ruta permite hacer dos tipos de peticiones:

- GET: Permite consultar los límites actuales de la resistencia interna y la temperatura.
- POST: Permite modificar estos límites mediante una solicitud HTTP.

5.2. *Docker Compose*

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones con múltiples contenedores. A través de un archivo de configuración, *Docker Compose* facilita la orquestación de todos los servicios que componen una aplicación, lo que simplifica su despliegue y gestión.

5.2.1. Ventajas de usar *Docker Compose*

Docker Compose ofrece diversas ventajas para la gestión y coordinación de los contenedores involucrados en este proyecto:

- **Facilidad de Configuración y Orquestación**: *Docker Compose* permite definir todos los servicios de la aplicación en un solo archivo de configuración (***docker-compose.yml***). Esto significa que con un único comando (*docker-compose up*), todos los contenedores

necesarios (*backend*, *frontend,* bases de datos, *script_runner*, etc.) se inician y configuran automáticamente. Esto es especialmente útil para aplicaciones que requieren múltiples servicios que dependen entre sí.

- **Aislamiento y Reproducibilidad**: Cada contenedor de Docker opera en un entorno aislado, lo que garantiza que las dependencias y configuraciones de cada servicio no interfieran entre sí. Este aislamiento asegura que el proyecto funcione siempre de la misma manera, independientemente del entorno en el que se ejecute (local o en producción). Además, *Docker Compose* facilita la replicación del entorno para otros desarrolladores o equipos, lo que reduce los problemas de configuración.
- **Escalabilidad**: *Docker Compose* permite escalar los servicios según sea necesario. En este proyecto, si uno de los componentes (por ejemplo, el *backend* o la base de datos) requiere más capacidad, se puede aumentar fácilmente el número de instancias del servicio sin necesidad de reconfigurar el sistema. Este enfoque asegura que el sistema puede crecer de manera flexible.
- **Gestión de Redes y Volúmenes**: Con *Docker Compose*, todos los servicios definidos en el archivo de configuración se conectan automáticamente a una red compartida, lo que facilita la comunicación entre los contenedores. Esto es crucial en este proyecto, ya que los servicios (*backend*, bases de datos y *frontend*) deben interactuar constantemente. Además, Docker Compose permite la creación de volúmenes persistentes, asegurando que los datos almacenados en las bases de datos no se pierdan al reiniciar los contenedores.

5.2.2. Creación de contenedores

La creación de contenedores en *Docker Compose* requiere varios archivos clave:

- El archivo YAML, *docker_compose.yml*.
- Los archivos *Dockerfile* para contenedor.
- Los archivos de dependencias, *requirements.txt*.

El archivo YAML, es el núcleo de la configuración de *Docker Compose*. Define cómo se crean y conectan los diferentes contenedores que se compone el proyecto.

```yaml
version: '3.8'

services:
  db:
    image: postgres:latest
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: datadb
    volumes:
      - db_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  backend:
    build: ./backend
    volumes:
      - ./backend:/app
    ports:
      - "5000:5000"
    depends_on:
      - db

  frontend:
    build: ./frontend
    volumes:
      - ./frontend:/app
    ports:
      - "8000:8000"
    depends_on:
      - backend

  script_runner:
    build: ./script_runner
    volumes:
      - ./script_runner:/app
    depends_on:
      - backend
    environment:
      - PYTHONUNBUFFERED=1
    network_mode: "host"

volumes:
  db_data:
```

Fragmento código 32. Archivo *YAML* del *Docker Compose*.

En este archivo se define los cuatro contenedores como servicios. Los contenedores *backend*, *frontend* y *script_runner*, necesitan tener un archivo *Dockerfile*, que define la imagen *Flask* y requirements.txt, para instalar las dependencias necesarias.

Como se puede observar, el *backend* expone el puerto 5000, que será accesible en localhost:5000, del mismo modo que el *frontend*, que expone el puerto 8000, desde el cual se accede a la web.

El contenedor PostgreSQL usa una imagen oficial de PostgreSQL y se conecta con el *backend* para almacenar los datos simulados de las máquinas virtuales. Los datos se guardan en la base de datos *db_data*.

Los contenedores están conectados a la misma red (*app-network*).

Es necesario tener instalado el programa ***Docker Compose*** para crear los contenedores. Además, el proyecto debe seguir una estructura de carpetas para organizar correctamente los contenedores. En el proyecto se ha utilizado la siguiente estructura:

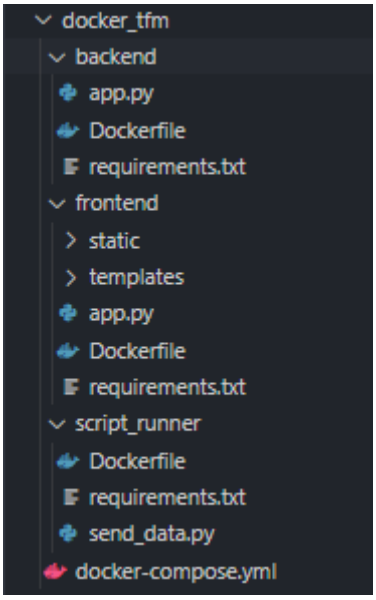

Figura 22. Estructura de carpetas del *Docker Compose*.

Cada contenedor debe tener su archivo *Dockerfile* y *requirements.txt*, para definir cómo se genera la imagen de dicho contenedor.

- ***Backend*:**

La aplicación se inicia con Python y se instala todas las dependencias indicadas en el archivo *requirements.txt*.

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["python", "app.py"]
```

```
flask
flask_sqlalchemy
psycopg2-binary
```

Fragmento código 33. Archivos *Dockerfile* y *requirements* para el *backend*.

- ***Frontend*:**

El archivo *Dockerfile* es similar que el *backend*, y también se ejecuta la aplicación con Python.

```
flask
requests
```

Fragmento código 34. Archivo *requirements* para el *frontend*.

- **Script runner:**

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["sh", "-c", "while true; do python send_data.py; sleep 300; done"]
```

```
requests
```

Fragmento código 35. Archivos *Dockerfile* y *requirements* para el script runner.

5.3. Script runner

Este código realiza la orquestación y envío de datos simulados desde varias máquinas virtuales hacia un *backend* centralizado para su procesamiento. A continuación, se va a explicar por partes el código.

5.3.1. *URLs* para las máquinas virtuales y el *backend*

El código empieza con la definición de las *URLs* de las máquinas virtuales y del *backend*. Cada máquina virtual simula un dispositivo específico, y las *URLs* corresponden a los *endpoints* que exponen las *APIs* de estos dispositivos.

```
# URL de la máquinas virtuales
vm_url_solar = "http://192.168.1.54:5100/solar"
vm_url_wind = "http://192.168.1.55:5200/wind"
vm_url_batteries = "http://192.168.1.56:5300/batteries"
```

Fragmento código 36. *URLs* máquinas virtuales.

Estas son las *URLs* de las máquinas que simulan un sistema solar, uno eólico y un conjunto de baterías, respectivamente. Las máquinas virtuales generan datos simulados sobre la producción de energía, que luego se envían al *backend*.

Las *URLs* del *backend* local son:

```
# URL de los backends
backend_url_solar = "http://localhost:5000/data_solar"
backend_url_wind = "http://localhost:5000/data_wind"

backend_url_batteries = "http://localhost:5000/data_batteries"
```

Fragmento código 37. *URLs* del *backend*.

Estas *URLs* se utilizan para enviar los datos simulados procesados al *backend* para su almacenamiento y análisis.

5.3.2. Función para obtener datos de las máquinas virtuales

La función *fetch_data_from_machine* hace una solicitud GET a la URL de la máquina virtual y devuelve los datos en formato *JSON*.

```
def fetch_data_from_machine(vm_url):
    try:
        response = requests.get(vm_url)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Error fetching data from {vm_url}: {e}")

        return None
```

Fragmento código 38. Función para obtener datos de las máquinas virtuales.

Esta función usa la librería *requests* para conectarse a la máquina virtual y recuperar los datos. Si hay un error en la solicitud, el error se maneja y se imprime un mensaje, devolviendo *None* en caso de fallo.

5.3.3. Función para extraer voltaje e intensidad

La función *get_voltage_and_current* extrae los valores de voltaje y corriente del JSON devuelto por la máquina virtual.

```
def fetch_data_from_machine(vm_url):
    try:
        response = requests.get(vm_url)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Error fetching data from {vm_url}: {e}")

        return None
```

Fragmento código 39. Función para extraer voltaje e intensidad.

Esta función toma los datos *JSON* de la máquina y devuelve el voltaje y la corriente. Si no encuentra los valores, devolverá *None*.

5.3.4. Función para enviar datos a la máquina virtual de baterías

Se necesita crear una función para enviar los datos de la placa solar y la turbina eólica. Esta función construye una URL con los valores de voltaje e intensidad de las otras dos máquinas, y hace una solicitud GET a la máquina de baterías.

```
def send_data_to_machine_3(voltage_solar, current_solar,
voltage_generator, current_generator):
    # Construir la URL con los parámetros necesarios
    url =
f"http://192.168.1.56:5300/batteries?voltage_solar={voltage_solar}&
current_solar={current_solar}&voltage_generator={voltage_generator}
&current_generator={current_generator}"

    try:
        response = requests.get(url)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Error sending data to machine 3: {e}")
        return None
```

Fragmento código 40. Función para enviar datos a la máquina de baterías.

La función utiliza los valores de voltaje e intensidad de las otras dos máquinas virtuales para alimentar el *endpoint* de la máquina de baterías y obtener una respuesta con los datos simulados.

5.3.5. Función para enviar datos al *backend*

La función *send_data_to_backend* envía los datos procesados al *backend* utilizando una solicitud POST.

```
def send_data_to_backend(data, backend_url):
    try:
        response = requests.post(backend_url, json=data)
        response.raise_for_status()
        print("Data sent successfully to backend.")
    except requests.exceptions.RequestException as e:
        print(f"Error sending data to backend: {e}")
```

Fragmento código 41. Función de envío datos al backend.

Aquí, los datos simulados son enviados al *backend* a través de una solicitud POST en formato JSON. Si la solicitud falla, se imprime un mensaje de error.

5.3.6. Funciones para transformas los datos

Existen tres funciones que transforman los datos obtenidos de las máquinas virtuales al formato requerido por el *backend*:

- **Transformar datos de la placa solar:**

```
def transform_data_solar(vm_data):
    # Transformar el JSON de la VM al formato requerido por el
backend
    return {
        "timestamp": datetime.now().isoformat(),
        "voltage": round(vm_data.get("voltage"), 2),
        "current": round(vm_data.get("current"), 2),
        "power": round(vm_data.get("power")/1000, 2),
        "energy_generated": round(vm_data.get("energy_generated"),
2),
        "solar_radiation": round(vm_data.get("solar_radiation"),
2),
        "energy_wh": round(vm_data.get("energy_wh"), 2)

    }
```

Fragmento código 42. Función de transformación datos de la placa solar.

- **Transformar datos de la turbina eólica:**

```
def transform_data_wind(vm_data):
    # Transformar el JSON de la VM al formato requerido por el
backend
    return {
        "timestamp": datetime.now().isoformat(),
        "voltage": round(vm_data.get("voltage"), 2),
        "current": round(vm_data.get("current"), 2),
        "power": round(vm_data.get("power")/1000, 2),
        "energy_generated": round(vm_data.get("energy_generated"),
2),
        "energy_wh": round(vm_data.get("energy_wh"), 2),
        "wind_speed": round(vm_data.get("wind_speed"), 2)

    }
```

Fragmento código 43. Función de transformación datos de la turbina eólica.

- **Transformar datos de las baterías:**

```
def transform_data_batteries(vm_data):
    # Transformar el JSON de la VM al formato requerido por el
backend
    return {
        "timestamp": datetime.now().isoformat(),
        "capacity_solar": round(vm_data.get("capacity_solar"), 2),
        "capacity_generator":
round(vm_data.get("capacity_generator"), 2),
        "capacity": round(vm_data.get("capacity")/1000, 2),
        "capacity_ah": round(vm_data.get("capacity_ah")/1000, 2),
        "soc": round(vm_data.get("soc"), 2),
        "soh": round(vm_data.get("soh"), 2),
        "internal_resistance":
round(vm_data.get("internal_resistance"), 2),
        "temperature": round(vm_data.get("temperature"), 2),
        "charge_efficiency":
round(vm_data.get("charge_efficiency"), 2),
        "discharge_efficiency":
round(vm_data.get("discharge_efficiency"), 2)
    }
```

Fragmento código 44. Función de transformación datos de las baterías.

Cada una de estas funciones toma los datos brutos de la máquina virtual y los ajusta al formato esperado por el *backend*. Además, se asegura de redondear los valores a dos decimales para mantener precisión y legibilidad.

5.3.7. Ejecución del programa principal

Finalmente, el código principal obtiene los datos de las máquinas virtuales, transforma esos datos, y los envía al *backend*.

```
if __name__ == "__main__":
    # Obtener datos de la máquina solar
    data_solar = fetch_data_from_machine(vm_url_solar)
    print(data_solar)
    if data_solar:
        voltage_solar, current_solar =
get_voltage_and_current(data_solar)
    else:
        voltage_solar, current_solar = None, None

    # Obtener datos de la máquina 2
    data_wind = fetch_data_from_machine(vm_url_wind)
    print(data_wind)
    if data_wind:
        voltage_generator, current_generator =
get_voltage_and_current(
            data_wind)
    else:
        voltage_generator, current_generator = None, None

    # Verificar que todos los datos necesarios estén disponibles
    if None not in (voltage_solar, current_solar,
voltage_generator, current_generator):
        # Hacer la solicitud a la máquina 3
        data_batteries = send_data_to_machine_3(
            voltage_solar, current_solar, voltage_generator,
current_generator)
        print(data_batteries)
    else:
        print("No se pudo obtener todas las variables necesarias
para enviar la solicitud a la máquina 3.")

    if data_solar:
        time.sleep(1)  # Pausa de 1 segundo entre solicitudes
        transformed_data_solar = transform_data_solar(data_solar)
        send_data_to_backend(transformed_data_solar,
backend_url_solar)
        time.sleep(1)  # Pausa de 1 segundo entre solicitudes

    if data_wind:
        time.sleep(1)  # Pausa de 1 segundo entre solicitudes
        transformed_data_wind = transform_data_wind(data_wind)
        send_data_to_backend(transformed_data_wind,
backend_url_wind)
        time.sleep(1)  # Pausa de 1 segundo entre solicitudes

    if data_batteries:
        time.sleep(1)  # Pausa de 1 segundo entre solicitudes
        transformed_data_batteries =
transform_data_batteries(data_batteries)
        send_data_to_backend(transformed_data_batteries,
backend_url_batteries)
        time.sleep(1)  # Pausa de 1 segundo entre solicitudes
```

Fragmento código 45. Programa principal del script runner.

Este bloque de código ejecuta las funciones necesarias para recolectar los datos de las máquinas virtuales, transformar esos datos en el formato requerido y enviarlos al *backend*. Se puede observar que el envío de datos al *backend* no es al mismo tiempo, hay un tiempo de espera de 1 segundo, para no saturar el *backend*.

5.4. *Backend*

Este código define una API que interactúa con una base de datos **PostgreSQL** para almacenar y recuperar datos simulados de un sistema solar, eólico y de baterías. Los *endpoints* **POST** permiten enviar datos a la base de datos, mientras que los *endpoints* **GET** permiten consultar los datos almacenados en las últimas 24 horas.

5.4.1. Configuración inicial

```
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'postgresql://user:password@db/datadb'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)
```

Fragmento código 46. Configuración inicial del *backend*.

Este bloque inicializa la aplicación ***Flask*** y configura el acceso a una base de datos ***PostgreSQL*** utilizando la extensión ***SQLAlchemy*** para manejar la base de datos en la aplicación. La configuración incluye la URL de la base de datos, que contiene las credenciales de acceso y el nombre de la base de datos. Además, se desactiva el seguimiento de modificaciones de *SQLAlchemy* para mejorar el rendimiento.

5.4.2. Definición de columnas para las tablas de PostgreSQL

- Placa solar:

```
class Data_solar(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    timestamp = db.Column(db.DateTime, nullable=False)
    voltage = db.Column(db.Float, nullable=False)
    current = db.Column(db.Float, nullable=False)
    power = db.Column(db.Float, nullable=False)
    energy_generated = db.Column(db.Float, nullable=False)
    energy_wh = db.Column(db.Float, nullable=False)
    solar_radiation = db.Column(db.Float, nullable=False)
```

Fragmento código 47. Definición de las columnas de la placa solar.

El modelo *Data_solar* define la estructura de la tabla que almacenará los datos simulados del sistema solar. Cada fila contiene información como el voltaje, intensidad, potencia, energía generada y la radiación solar en un momento determinado (*timestamp*). Esta estructura se repetirá para los otros sistemas.

Otros modelos, como *Data_wind* y *Data_batteries*, tienen una estructura similar:

- Turbina eólica:

```
class Data_wind(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    timestamp = db.Column(db.DateTime, nullable=False)
    voltage = db.Column(db.Float, nullable=False)
    current = db.Column(db.Float, nullable=False)
    power = db.Column(db.Float, nullable=False)
    energy_generated = db.Column(db.Float, nullable=False)
    wind_speed = db.Column(db.Float, nullable=False)

    energy_wh = db.Column(db.Float, nullable=False)
```

Fragmento código 48. Definición de las columnas de la turbina eólica.

La clase ***Data_wind*** añade la variable ***wind_speed*** para almacenar la velocidad del viento. Este modelo se encarga de registrar los datos del sistema eólico.

- Baterías:

```
class Data_batteries(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    timestamp = db.Column(db.DateTime, nullable=False)
    capacity_solar = db.Column(db.Float, nullable=False)
    capacity_generator = db.Column(db.Float, nullable=False)
    capacity = db.Column(db.Float, nullable=False)
    capacity_ah = db.Column(db.Float, nullable=False)
    soc = db.Column(db.Float, nullable=False)
    soh = db.Column(db.Float, nullable=False)
    internal_resistance = db.Column(db.Float, nullable=False)
    temperature = db.Column(db.Float, nullable=False)
    charge_efficiency = db.Column(db.Float, nullable=False)

    discharge_efficiency = db.Column(db.Float, nullable=False)
```

Fragmento código 49. Definición de las columnas de las baterías.

La clase ***Data_batteries*** almacena información sobre las baterías, como la **capacidad**, **estado de carga (*SoC*)** y **eficiencia de carga y descarga**.

5.4.3. Creación de tablas

```
def create_tables():
    with app.app_context():

        db.create_all()
```

Fragmento código 50. Creación de tablas en la base de datos PostgreSQL.

La función *create_tables* se asegura de que todas las tablas definidas en los modelos se creen en la base de datos al iniciar la aplicación. Esto se ejecuta dentro de un contexto de la aplicación *Flask*.

5.4.4. *Endpoints* para recibir datos

Cada uno de los siguientes *endpoints* recibe datos a través de solicitudes POST y los almacena en la base de datos:

- *Endpoint* para datos de la placa solar:

```
@app.route('/data_solar', methods=['POST'])
def receive_data_solar():
    data = request.json
    new_data = Data_solar(
        timestamp=datetime.fromisoformat(data['timestamp']),
        voltage=data['voltage'],
        current=data['current'],
        power=data['power'],
        energy_generated=data['energy_generated'],
        energy_wh=data['energy_wh'],
        solar_radiation=data['solar_radiation']
    )
    db.session.add(new_data)
    db.session.commit()

    return jsonify({"message": "Data added successfully"}), 201
```

Fragmento código 51. *Endpoint* para datos de la placa solar.

Este *endpoint* recibe datos en formato JSON y los convierte en una instancia del modelo *Data_solar*. Luego, los agrega a la sesión de la base de datos y confirma la transacción para guardarlos permanentemente.

- *Endpoint* para datos de la turbina eólica:

El *endpoint* para los datos de la turbina eólica sigue una estructura similar:

```
@app.route('/data_wind', methods=['POST'])
def receive_data_wind():
    data = request.json
    new_data = Data_wind(
        timestamp=datetime.fromisoformat(data['timestamp']),
        voltage=data['voltage'],
        current=data['current'],
        power=data['power'],
        energy_generated=data['energy_generated'],
        wind_speed=data['wind_speed'],
        energy_wh=data['energy_wh']
    )
    db.session.add(new_data)
    db.session.commit()

    return jsonify({"message": "Data added successfully"}), 201
```

Fragmento código 52. *Endpoint* para datos de la turbina eólica.

Este *endpoint* registra los datos del sistema eólico, incluyendo la velocidad del viento.

- *Endpoint* para datos de las baterías:

```
@app.route('/data_batteries', methods=['POST'])
def receive_data_batteries():
    data = request.json
    new_data = Data_batteries(
        timestamp=datetime.fromisoformat(data['timestamp']),
        capacity_solar=data['capacity_solar'],
        capacity_generator=data['capacity_generator'],
        capacity=data['capacity'],
        capacity_ah=data['capacity_ah'],
        soc=data['soc'],
        soh=data['soh'],
        internal_resistance=data['internal_resistance'],
        temperature=data['temperature'],
        charge_efficiency=data['charge_efficiency'],
        discharge_efficiency=data['discharge_efficiency']
    )
    db.session.add(new_data)
    db.session.commit()

    return jsonify({"message": "Data added successfully"}), 201
```

Fragmento código 53. *Endpoint* para datos de las baterías.

Este último *endpoint* almacena la información sobre las baterías, como la capacidad y el estado de salud (*SoH*).

5.4.5. *Endpoints* para obtener datos

Los *endpoints* de GET permiten obtener los datos almacenados en las últimas 24 horas. Para los datos de la placa solar, se utiliza el siguiente código:

```
@app.route('/data_batteries', methods=['POST'])
def receive_data_batteries():
    data = request.json
    new_data = Data_batteries(
        timestamp=datetime.fromisoformat(data['timestamp']),
        capacity_solar=data['capacity_solar'],
        capacity_generator=data['capacity_generator'],
        capacity=data['capacity'],
        capacity_ah=data['capacity_ah'],
        soc=data['soc'],
        soh=data['soh'],
        internal_resistance=data['internal_resistance'],
        temperature=data['temperature'],
        charge_efficiency=data['charge_efficiency'],
        discharge_efficiency=data['discharge_efficiency']
    )
    db.session.add(new_data)
    db.session.commit()

    return jsonify({"message": "Data added successfully"}), 201
```

Fragmento código 54. *Endpoints* para obtener datos.

Este *endpoint* filtra los datos de las últimas 24 horas y devuelve los registros formateados en JSON. Lo mismo se aplica a los *endpoints* de GET para los datos de la turbina eólica y de las baterías, con ligeras variaciones según los campos de cada modelo.

5.4.6. Inicio de la aplicación

```
if __name__ == '__main__':
    create_tables()  # Crear tablas antes de iniciar la aplicación
    app.run(host='0.0.0.0')
```

Fragmento código 55. Inicio de la aplicación del *backend*.

Este bloque asegura que las tablas se creen antes de iniciar la aplicación *Flask*, que está configurada para escuchar en todas las interfaces de red.

5.5. *Frontend*

Este código construye una aplicación *Flask* que se conecta con un *backend* para obtener datos de tres fuentes, los presenta en una interfaz web y permite descargar los datos en formato CSV. Cada parte de la aplicación está diseñada para filtrar datos por rango de fechas, brindando a los usuarios la posibilidad de analizar los datos de manera personalizada.

5.5.1. Configuración inicial y *URLs* de las *APIs*

```
app = Flask(__name__)

SOLAR_API_URL = 'http://backend:5000/data_solar'
WIND_API_URL = 'http://backend:5000/data_wind'
BATTERIES_API_URL = 'http://backend:5000/data_batteries'
```

Fragmento código 56. Configuración inicial *frontend*.

Este bloque inicializa la aplicación *Flask* y define las *URLs* de los *endpoints* de la API del *backend* para los datos de la placa solar, de la turbina eólica y de las baterías. El *backend* está configurado para ejecutarse en la misma red de Docker, por lo que se usa *backend* como *hostname*.

5.5.2. Función principal de la página de inicio

```
@app.route('/')
def index():
    response_solar = requests.get(SOLAR_API_URL)
    data_solar = response_solar.json()
    response_wind = requests.get(WIND_API_URL)
    data_wind = response_wind.json()
    response_batteries = requests.get(BATTERIES_API_URL)
    data_batteries = response_batteries.json()
    # Combinar todos los datos en un solo diccionario
    data = {
        'solar': data_solar,
        'wind': data_wind,
        'batteries': data_batteries
    }
    return render_template('index.html', data=data)
```

Fragmento código 57. Función principal de la página de inicio.

La función *index* es responsable de la página de inicio de la aplicación. Esta función realiza solicitudes GET a las *APIs* del *backend* para obtener los datos de la placa solar, de la turbina eólica y de las baterías. Una vez que se han obtenido los datos en formato JSON, se combinan en un solo diccionario y se pasan a la plantilla *index.html* para su visualización.

5.5.3. Funciones para mostrar datos individuales

Se incluyen tres rutas que muestran los datos de cada sistema por separado: solar, eólico y baterías. Cada una de estas rutas obtiene los datos correspondientes desde su API respectiva y los pasa a las plantillas de visualización.

- Datos de la placa solar:

```
@app.route('/solar_overview')
def solar():
    response = requests.get(SOLAR_API_URL)
    data_solar = response.json()
    return render_template('solar.html', data=data_solar)
```

Fragmento código 58. Función para mostrar datos de la placa solar.

- Datos de la turbina eólica:

```
@app.route('/wind_overview')
def wind():
    response = requests.get(WIND_API_URL)
    data_wind = response.json()
    return render_template('wind.html', data=data_wind)
```

Fragmento código 59. Función para mostrar datos de la turbina eólica.

- Datos de las baterías:

```
@app.route('/batteries_overview')
def batteries():
    response = requests.get(BATTERIES_API_URL)
    data_batteries = response.json()

    return render_template('batteries.html', data=data_batteries)
```

Fragmento código 60. Función para mostrar datos de las baterías.

Cada una de estas funciones es sencilla: hace una solicitud GET al *backend* para obtener los datos y renderiza la plantilla correspondiente (solar.html, wind.html, batteries.html) para mostrar la información.

5.5.4. Función para descargar datos en CSV

La ruta */download_solar* permite descargar los datos de la placa solar en formato CSV. El flujo de trabajo de esta función es el siguiente:

- **Obtener Rango de Fechas**: Se capturan las fechas de inicio y fin desde el *frontend*.
- **Convertir Fechas**: Se convierten las fechas proporcionadas al formato *datetime*.

```
@app.route('/download_solar')
def download_data_solar():
    # Obtener las fechas desde el frontend (formato: 'YYYY-MM-
DDTHH:MM')
    start_date = request.args.get('start_date')
    end_date = request.args.get('end_date')

    # Convertir las fechas del frontend a objetos datetime
    if start_date:
        start_date = datetime.strptime(start_date, '%Y-%m-
%dT%H:%M')
    if end_date:
        end_date = datetime.strptime(end_date, '%Y-%m-%dT%H:%M')

    # Imprimir para depuración
    print(f"Fecha de inicio en el backend (convertida):
{start_date}")
    print(f"Fecha de fin en el backend (convertida): {end_date}")
```

Fragmento código 61. Función para descargar datos en CSV de la placa solar parte 1.

- **Filtrar Datos**: Se obtienen los datos de la API del *backend* y se filtran para que coincidan con el rango de fechas seleccionado.

```
    # Filtrar los datos por rango de fechas
    filtered_data = []
    for d in data:
        try:
            # Convertir el 'timestamp' de los datos a un objeto
datetime usando el formato '%d/%m/%Y %H:%M'
            data_timestamp = datetime.strptime(
                d['timestamp'], '%d/%m/%Y %H:%M')

            # Imprimir para depuración
            print(
                f"Timestamp de los datos convertido a datetime:
{data_timestamp}")

            # Comparar el timestamp con el rango de fechas
(start_date y end_date)
            if start_date <= data_timestamp <= end_date:
                filtered_data.append(d)
        except ValueError as e:
            # Manejar el caso donde el 'timestamp' no tenga el
formato esperado
            print(
                f"Error al convertir el timestamp
'{d['timestamp']}' a datetime: {e}")

    # Si no hay datos en el rango de fechas, devolver un CSV vacío
o mensaje
    if not filtered_data:
        output = io.StringIO()
        writer = csv.writer(output)
        writer.writerow(['timestamp', 'voltage', 'current',
                         'power', 'energy_generated',
'solar_radiation'])
        output.write(
            "No hay datos disponibles para el rango de fechas
seleccionado.\n")
        output.seek(0)
        return send_file(io.BytesIO(output.read().encode('utf-8')),
                         mimetype='text/csv',
                         download_name='data_solar.csv',
                         as_attachment=True)
```

Fragmento código 62. Función para descargar datos en CSV de la placa solar parte 2.

- **Generar CSV**: Se genera un archivo CSV con los datos filtrados.

```
    # Generar el CSV con los datos filtrados
    output = io.StringIO()
    writer = csv.writer(output)
    writer.writerow(['timestamp', 'voltage', 'current',
                     'power', 'energy_generated',
'solar_radiation'])
    for row in filtered_data:
        writer.writerow([row['timestamp'], row['voltage'],
                         row['current'], row['power'],
row['energy_generated'], row['solar_radiation']])

    output.seek(0)
    return send_file(io.BytesIO(output.read().encode('utf-8')),
                     mimetype='text/csv',
                     download_name='data_solar.csv',
                     as_attachment=True)
```

Fragmento código 63. Función para descargar datos en CSV de la placa solar parte 3.

Las funciones para la turbina eólica y las baterías tienen la misma estructura, solo cambia las rutas, las columnas de escritura y el nombre del archivo.

5.5.5. Ejecución de la aplicación

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000)
```

Fragmento código 64. Ejecución de la aplicación del *frontend*.

Finalmente, la aplicación *Flask* se inicia en el puerto 8000, escuchando en todas las interfaces *(0.0.0.0).* Esto asegura que la aplicación sea accesible desde cualquier máquina en la red.

5.6. Conclusión

En este capítulo se ha explicado cómo se han creado las máquinas virtuales con sus respectivos scripts de simulación, así como la activación del protocolo SSH. También se ha detallado la instalación de los contenedores utilizando *Docker Compose*.

Por último, se ha descrito lo que se ejecuta en cada contenedor, quedando pendiente la descripción de los archivos HTML. En este caso, hay cuatro archivos: uno para la página principal y otro archivo para cada una de las máquinas.

En el siguiente capítulo, se describirá cómo se ha diseñado la interfaz gráfica de cada página.

CAPÍTULO 6. MAQUETACIÓN DE LA INTERFAZ GRÁFICA

6.1. Introducción

En este capítulo se explicará cómo se ha maquetado la página web y cómo se han creado los componentes para visualizar los datos de las máquinas virtuales, descargar archivos, escribir en las variables límite y, por último, la visualización de gráficos.

La interfaz se compone de cuatro páginas:

- **Index.html:** Es la página principal, donde se puede visualizar el resumen de las tres máquinas.
- **Solar.html**: En esta página se detallan todos los parámetros de la placa solar.
- **Wind.html:** En esta página se detallan todos los parámetros de la turbina eólica.
- **Batteries.html**: En esta página se detallan todos los parámetros de las baterías.

Los archivos *.html* se encargan de la maquetación, pero el diseño visual se logra gracias al archivo *.css* correspondiente a cada página.

6.2. Página principal como index.html

Este código HTML integra datos en tiempo real de diferentes fuentes de energía y los muestra en una página web con gráficos interactivos y visualizaciones detalladas de energía solar, eólica y de baterías. Utiliza *Flask* para procesar los datos del *backend* y *Chart.js* para crear gráficos dinámicos.

Se detallarán los componentes importantes que se han desarrollado para la página. A continuación, se muestra la visualización completa de la pantalla principal:

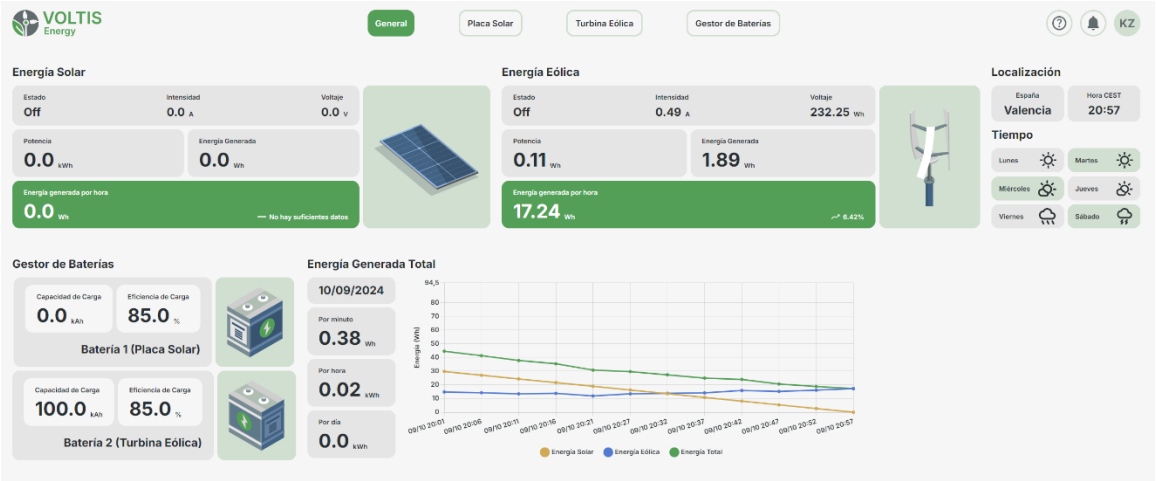

Figura 23. Visualización general de la pantalla completa.

6.2.1. Estructura básica del documento HTML

Este código es una plantilla HTML básica para visualizar los datos de energía solar, eólica y de baterías. La estructura incluye el uso de *Flask, CSS,* y la biblioteca *Chart.js* para gráficos interactivos.

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1, width=device-
width">
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/styles_index.css') }}">
    <link rel="stylesheet"
href="https://fonts.googleapis.com/css2?family=Inter:wght@600;700&di
splay=swap" />
    <!-- Asegúrate de incluir el script para Chart.js -->
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script type="text/javascript">
        // Recargar la página cada 30 segundos
        setInterval(function () {
            location.reload();
        }, 30000); // 30000 ms = 30 segundos
    </script>
</head>
```

Fragmento código 65. Estructura básica de index.html.

- Se incluye el enlace al archivo CSS que define el estilo visual de la página. Esto es crucial para la maquetación de la interfaz.
- La biblioteca Chart.js se importa para crear gráficos de líneas que representarán la energía generada por las diferentes fuentes.
- Un script de JavaScript recarga la página cada 30 segundos, lo que permite actualizar automáticamente los datos visualizados.

6.2.2. Encabezado

El **encabezado** de la página incluye un logo y botones de navegación que permiten a los usuarios cambiar entre las diferentes secciones de la interfaz (general, placa solar, turbina eólica y gestor de baterías). También se incluyen íconos para ayuda y alarmas, junto con un marcador de cuenta de usuario.

Su visualización es la siguiente:

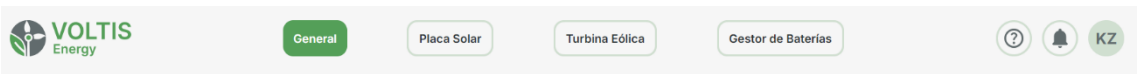

Figura 24. Encabezado de la página principal.

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1, width=device-
width">
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/styles_index.css') }}">
    <link rel="stylesheet"
href="https://fonts.googleapis.com/css2?family=Inter:wght@600;700&di
splay=swap" />
    <!-- Asegúrate de incluir el script para Chart.js -->
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script type="text/javascript">
        // Recargar la página cada 30 segundos
        setInterval(function () {
            location.reload();
        }, 30000); // 30000 ms = 30 segundos
    </script>
</head>
```

Fragmento código 66. Encabezado.

6.2.3. Visualización de energía solar

Aquí se muestra el estado actual de la energía solar. El valor de "Estado" (*On/Off*) se determina con base en la potencia generada por la fuente solar. Si la potencia es mayor que 0, el estado es "*On*". También se muestran la intensidad en amperios y el voltaje en voltios.

La parte visual de este componente se representaría de esta manera:

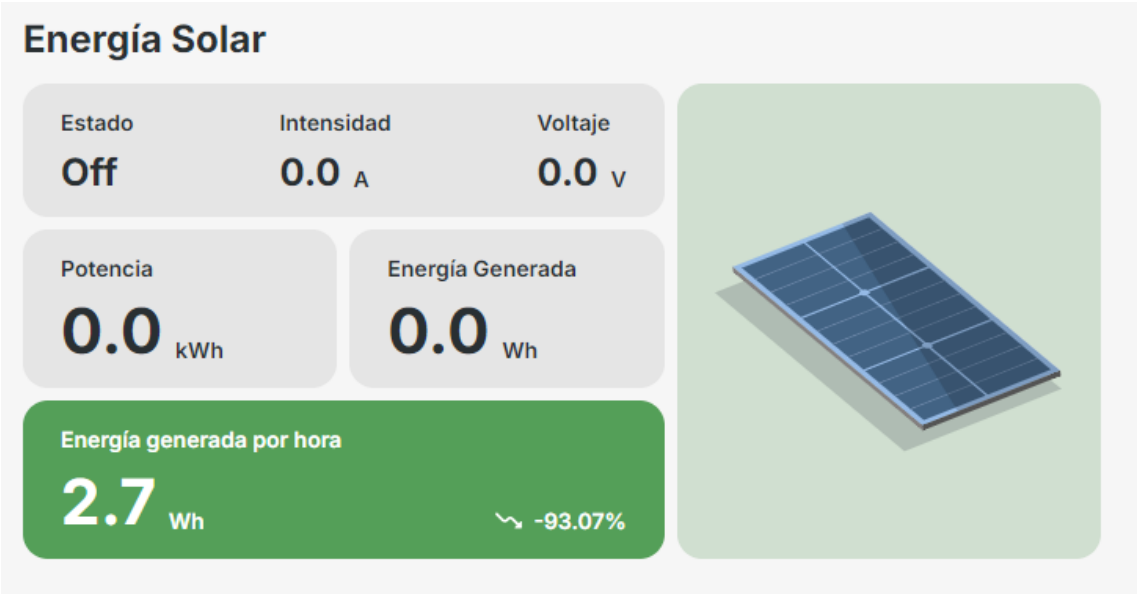

Figura 25. Energía solar de la página principal.

```
<div class="energia-solar">
    <b class="titulo">Energía Solar</b>
    <div class="cuadros-energia">
        <!-- Estado, Intensidad, Voltaje -->
        <div class="estado-grupo">
            <div class="titulo1">Estado</div>
            <b class="variable">
                {% if data['solar'][-1]['power'] and data['solar'][-
1]['power'] > 0 %}
                On
                {% else %}
                Off
                {% endif %}
            </b>
        </div>
        <div class="intensidad-grupo">
            <div class="titulo1">Intensidad</div>
            <div class="variable-grupo">
                <b class="variable1">{{ data['solar'][-1]['current']
}}</b>
                <div class="unidad">A</div>
            </div>
        </div>
        <div class="intensidad-grupo">
            <div class="titulo1">Voltaje</div>
            <div class="variable-grupo">
                <b class="variable1">{{ data['solar'][-1]['voltage']
}}</b>
                <div class="unidad">V</div>
            </div>
        </div>
    </div>
</div>
```

Fragmento código 67. Código de la visualización de energía solar.

6.**2.4. Visualización de energía eólica**

El bloque de código para la visualización de la energía eólica es similar al de la energía solar. Se visualizan los mismos parámetros (estado, voltaje, intensidad y potencia), con datos provenientes del sistema eólico.

La visualización gráfica:

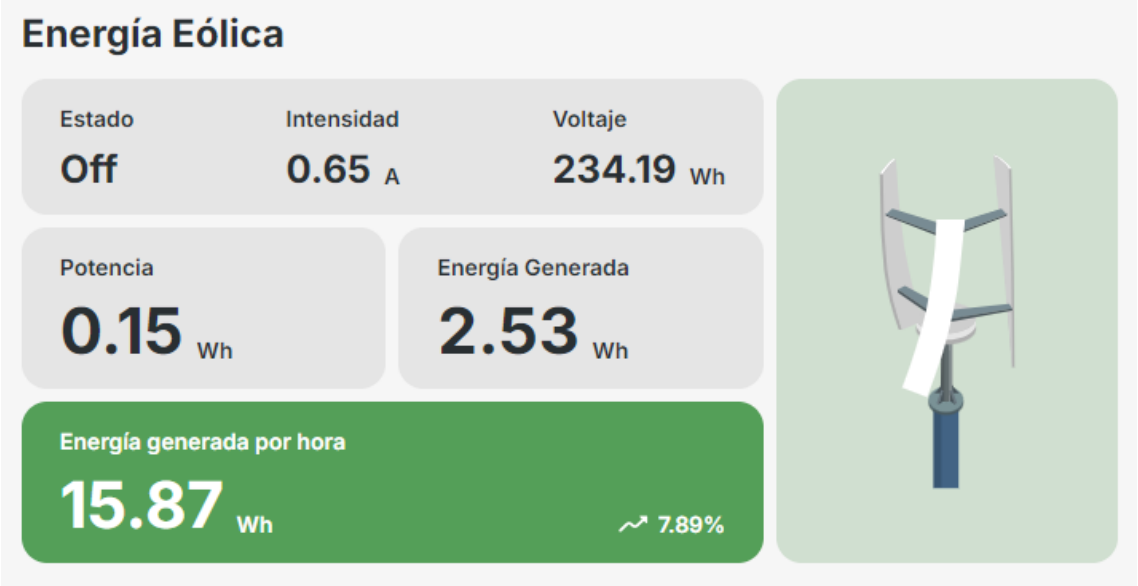

Figura 26. Energía eólica de la página principal.

```
<div class="energia-solar">
    <b class="titulo">Energía Eólica</b>
    <div class="cuadros-energia">
        <!-- Estado, Intensidad, Voltaje -->
        <div class="estado-grupo">
            <div class="titulo1">Estado</div>
            <b class="variable">
                {% if data['wind'][-1]['power'] and data['wind'][-
1]['power'] > 0 %}
                On
                {% else %}
                Off
                {% endif %}
            </b>
        </div>
        <div class="intensidad-grupo">
            <div class="titulo1">Intensidad</div>
            <div class="variable-grupo">
                <b class="variable1">{{ data['wind'][-1]['current']
}}</b>
                <div class="unidad">A</div>
            </div>
        </div>
        <div class="intensidad-grupo">
            <div class="titulo1">Voltaje</div>
            <div class="variable-grupo">
                <b class="variable1">{{ data['wind'][-1]['voltage']
}}</b>
                <div class="unidad">V</div>
            </div>
        </div>
    </div>
</div>
```

Fragmento código 68. Código de la visualización de energía eólica.

6.2.5. Visualización de las baterías

La sección de las **baterías** visualiza la capacidad de carga y eficiencia de las baterías, tanto para la energía solar como la eólica. Cada batería tiene su propia visualización de los datos.

- Batería 1 (Placa Solar):
  - Muestra la capacidad de carga de la batería que se carga con la energía solar, medida en kAh.
  - También muestra la eficiencia de carga, medida en porcentaje.
  - Un ícono gráfico de la batería acompaña a los valores.
- Batería 2 (Turbina Eólica):
  - De manera similar, esta batería muestra la capacidad de carga y la eficiencia de la carga, pero para la energía generada por la turbina eólica.

El diseño presenta ambas baterías de manera paralela, lo que permite comparar de un vistazo los datos de ambas fuentes de energía.

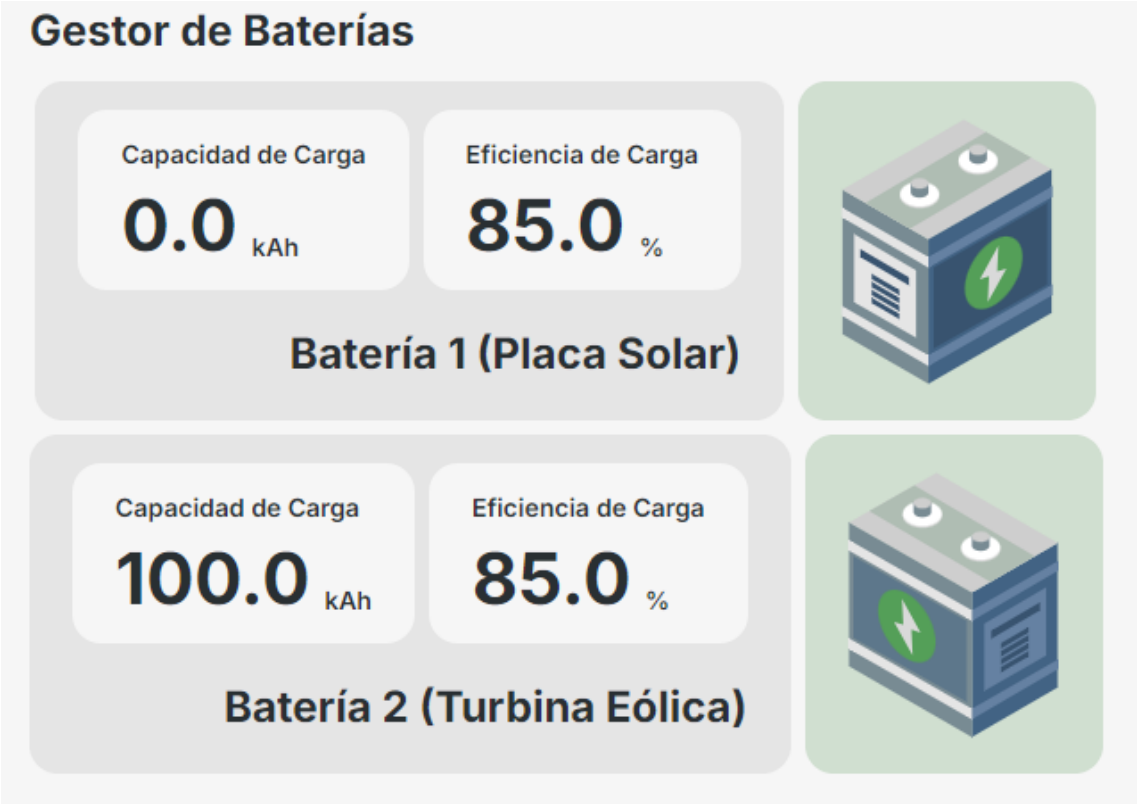

Figura 27. Gestor de baterías en la página principal.

```
<div class="gestor-de-baterias">
    <b class="titulo">Gestor de Baterías</b>
    <div class="grupo4">
        <!-- Batería 1 (Placa Solar) -->
        <div class="div17">
            <div class="div18">
                <div class="grupo5">
                    <div class="capacidad-de-carga">
                        <div class="titulo1">Capacidad de Carga</div>
                        <div class="variable-grupo2">
                            <b class="variable3">{{ data['batteries'][-
1]['capacity_solar'] }}</b>
                            <div class="unidad">kAh</div>
                        </div>
                    </div>
                    <div class="capacidad-de-carga">
                        <div class="titulo1">Eficiencia de Carga</div>
                        <div class="variable-grupo2">
                            <b class="variable3">{{ data['batteries'][-
1]['charge_efficiency'] }}</b>
                            <div class="unidad">%</div>
                        </div>
                    </div>
                </div>
                <b class="titulo21">Batería 1 (Placa Solar)</b>
            </div>
            <div class="div19">
                <img class="icon-bateria-1" alt="" src="{{
url_for('static', filename='images/bateria_1.svg') }}">
            </div>
        </div>
```

Fragmento código 69. Código de la visualización de la batería 1.

```
        <!-- Batería 2 (Turbina Eólica) -->
        <div class="div20">
            <div class="div21">
                <div class="grupo6">
                    <div class="capacidad-de-carga1">
                        <div class="titulo1">Capacidad de
Carga</div>
                        <div class="variable-grupo2">
                            <b class="variable3">{{
data['batteries'][-1]['capacity_generator'] }}</b>
                            <div class="unidad">kAh</div>
                        </div>
                    </div>
                    <div class="capacidad-de-carga1">
                        <div class="titulo1">Eficiencia de
Carga</div>
                        <div class="variable-grupo2">
                            <b class="variable3">{{
data['batteries'][-1]['charge_efficiency'] }}</b>
                            <div class="unidad">%</div>
                        </div>
                    </div>
                </div>
                <b class="titulo21">Batería 2 (Turbina Eólica)</b>
            </div>
            <div class="div19">
                <img class="icon-bateria-2" alt="" src="{{
url_for('static', filename='images/bateria_2.svg') }}">
            </div>
        </div>
    </div>
</div>
```

Fragmento código 70. Código de la visualización de la batería 2.

6.2.6. Gráfica de energía total

La visualización de los gráficos se genera utilizando la biblioteca *Chart.js*. Se procesan los datos obtenidos de las fuentes de energía solar y eólica para mostrar un gráfico combinado.

La visualización gráfica quedaría de esta manera:

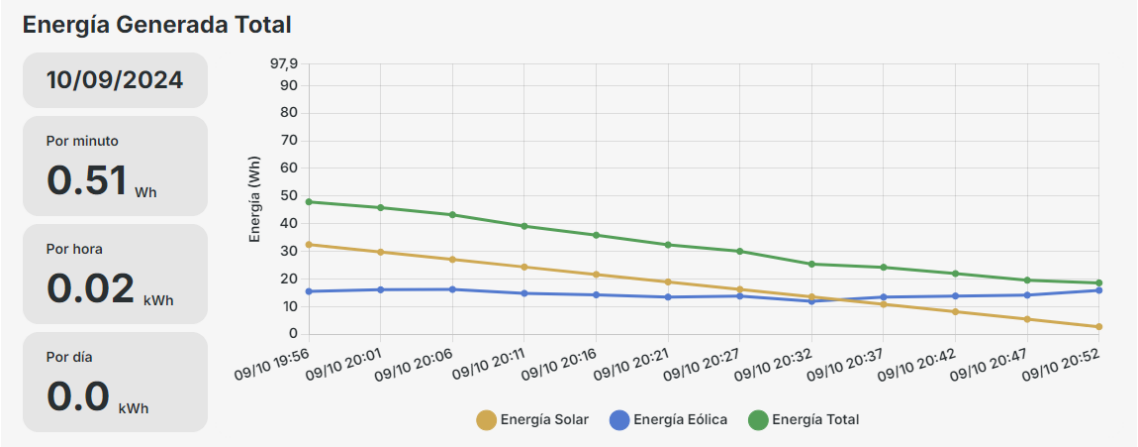

Figura 28. Gráfica de energía generada total.

```
<div class="energia-generada-total">
    <b class="titulo">Energía Generada Total</b>
    <div class="container1">
        <div class="fecha">
            {% if data['solar'][-1]['timestamp'] is not none %}
            {% set date_part = data['solar'][-
1]['timestamp'].split(' ')[0] %}
            <b class="lugar">{{ date_part }}</b>
            {% else %}
            <b class="lugar">Sin fecha disponible</b>
            {% endif %}
        </div>
        <div class="minuto">
            <div class="titulo1">Por minuto</div>
            <div class="variable-grupo14">
                <b class="variable3">{{ ((data['solar'][-
1]['energy_generated'] + data['wind'][-1]['energy_generated']) / 5)
| round(2) }}</b>
                <div class="unidad14">Wh</div>
            </div>
        </div>
        <div class="minuto">
            <div class="titulo1">Por hora</div>
            <div class="variable-grupo14">
                <b class="variable3">{{ ((data['solar'][-
1]['energy_wh'] + data['wind'][-1]['energy_wh'])/1000) | round(2)
}}</b>
                <div class="unidad14">kWh</div>
            </div>
        </div>
        <div class="minuto">
            <div class="titulo1">Por día</div>
            <div class="variable-grupo14">
                <b class="variable3">
                    {{ (total_energy_solar + total_energy_wind)/1000
| round(2) }}
                </b>
                <div class="unidad14">kWh</div>
            </div>
        </div>
    </div>
</div>
```

Fragmento código 71. Código de la energía total.

- Por minuto: Muestra la energía total generada por minuto combinando las energías solar y eólica.
- Por hora: Calcula la energía total por hora.
- Por día: Muestra la energía generada por día, sumando los valores de los sistemas solar y eólico de las últimas 24 horas.

Para la gráfica ha sido necesario crear un script aparte, indicando las variables que se quieren utilizar y dar el estilo necesario.

```
<canvas id="myChart" style="width:100%; height:100%;"></canvas>

<script>
    const data_solar = {{ data['solar'][-12:] | tojson }};
    const data_wind = {{ data['wind'][-12:] | tojson }};
    const timestamps = data_solar.map(d => {
        const date = new Date(d.timestamp);
        return `${date.getDate()}/${date.getMonth()+1}
${date.getHours()}:${date.getMinutes()}`;
    });
    const solarData = data_solar.map(d => d.energy_wh || 0);
    const windData = data_wind.map(d => d.energy_wh || 0);
    const totalData = solarData.map((val, i) => val + windData[i]);

    var ctx = document.getElementById('myChart').getContext('2d');
    new Chart(ctx, {
        type: 'line',
        data: {
            labels: timestamps,
            datasets: [{
                label: 'Energía Solar',
                data: solarData,
                borderColor: '#CFA953',
            }, {
                label: 'Energía Eólica',
                data: windData,
                borderColor: '#537ACF',
            }, {
                label: 'Energía Total',
                data: totalData,
                borderColor: '#549F58',
            }]
        },
        options: {
            scales: {
                y: { beginAtZero: true, title: { display: true,
text: 'Energía (Wh)' } },
                x: { ticks: { color: '#292F32' } }
            },
            plugins: {
                legend: { display: true, position: 'bottom' }
            }
        }
    });
</script>
```

Fragmento código 72. Código para configurar la gráfica de energía total.

Este bloque de código crea un gráfico de líneas que muestra los datos de energía solar, eólica y total. Se utilizan los últimos 12 valores de cada fuente de energía y se representan en el gráfico con diferentes colores para cada fuente.

6.2.7. Tendencias de energía y cálculos

```
<img class="icon" alt="tendencia-icon" src="{% if data['solar'] |
length > 1 %}
{% set current_energy = data['solar'][-1]['energy_wh'] %}
{% set previous_energy = data['solar'][-2]['energy_wh'] %}
{% if current_energy and previous_energy and current_energy >
previous_energy %}
    /static/images/tendencia_alc.svg
{% elif current_energy and previous_energy and current_energy <
previous_energy %}
    /static/images/tendencia_baj.svg
{% else %}
    /static/images/tendencia_neutral.svg
{% endif %}
{% else %}
    /static/images/tendencia_neutral.svg
{% endif %}">
```

Fragmento código 73. Tendencias de la página principal.

Este fragmento compara la energía generada actualmente con la generada en el pasado y selecciona un ícono que muestra la tendencia (ascendente, descendente o neutral). Se ah aplicado en la energía generada tanto en la solar como en la eólica.

6.2.8. Navegación entre páginas

Finalmente, el código incluye enlaces que permiten navegar entre las diferentes páginas que muestran datos específicos de cada tipo de energía.

```
<script>
    var placaSolarContainer =
document.getElementById("placaSolarContainer");
    if (placaSolarContainer) {
        placaSolarContainer.addEventListener("click", function (e) {
            window.location.href = "/solar_overview";
        });
    }

    var turbinaEolicaContainer =
document.getElementById("turbinaEolicaContainer");
    if (turbinaEolicaContainer) {
        turbinaEolicaContainer.addEventListener("click", function
(e) {
            window.location.href = "/wind_overview";
        });
    }

    var gestorDeBaterias =
document.getElementById("gestorDeBaterias");
    if (gestorDeBaterias) {
        gestorDeBaterias.addEventListener("click", function (e) {
            window.location.href = "/batteries_overview";
        });
    }
</script>
```

Fragmento código 74. Navegación entre páginas.

Este código añade funcionalidad de JavaScript para manejar los clics en diferentes secciones de la página y redirigir a las vistas correspondientes (*solar_overview, wind_overview y batteries_overview*).

6.3. Páginas de detalle

Se explicarán en detalle todos los componentes y scripts que han sido necesarios para las páginas de detalle. En este caso, solo se explicará cómo se ha desarrollado la página de detalle de la placa solar. En cuanto a la turbina eólica y el gestor de baterías, se utilizan los mismos componentes, por lo que no es necesario explicarlos para evitar repeticiones.

El encabezado y la navegación es el mismo código que se ha utilizado en la página principal, solo se tiene que cambiar las rutas.

La visualización general de las tres pantallas son las siguientes:

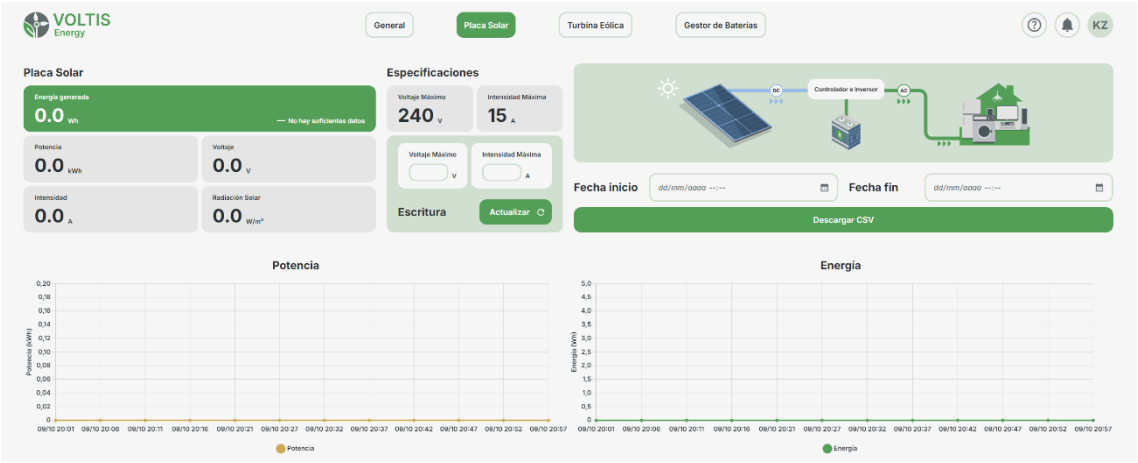

Figura 29. Visualización general placa solar.

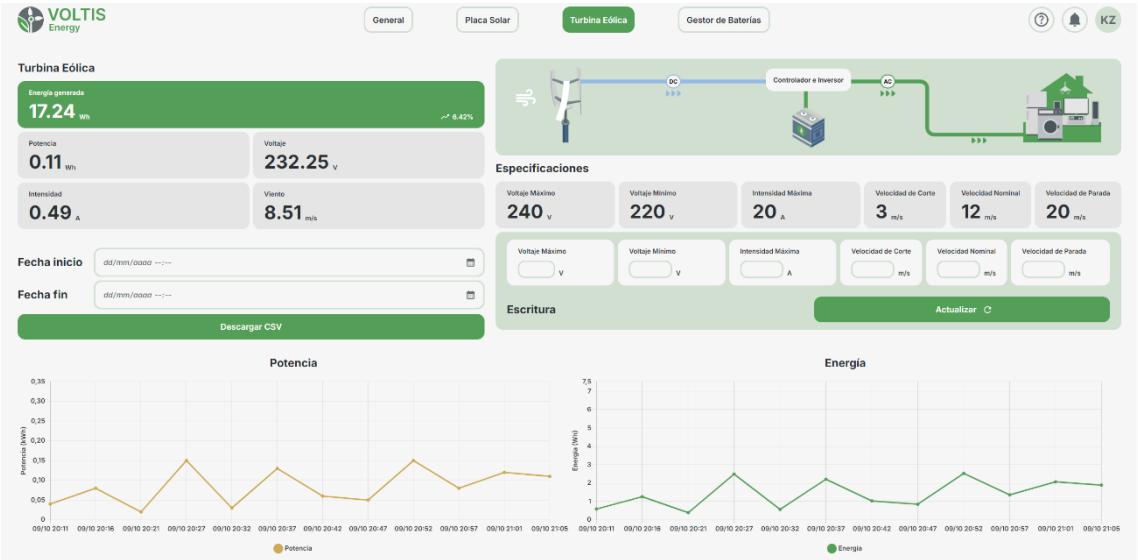

Figura 30. Visualización general turbina eólica.

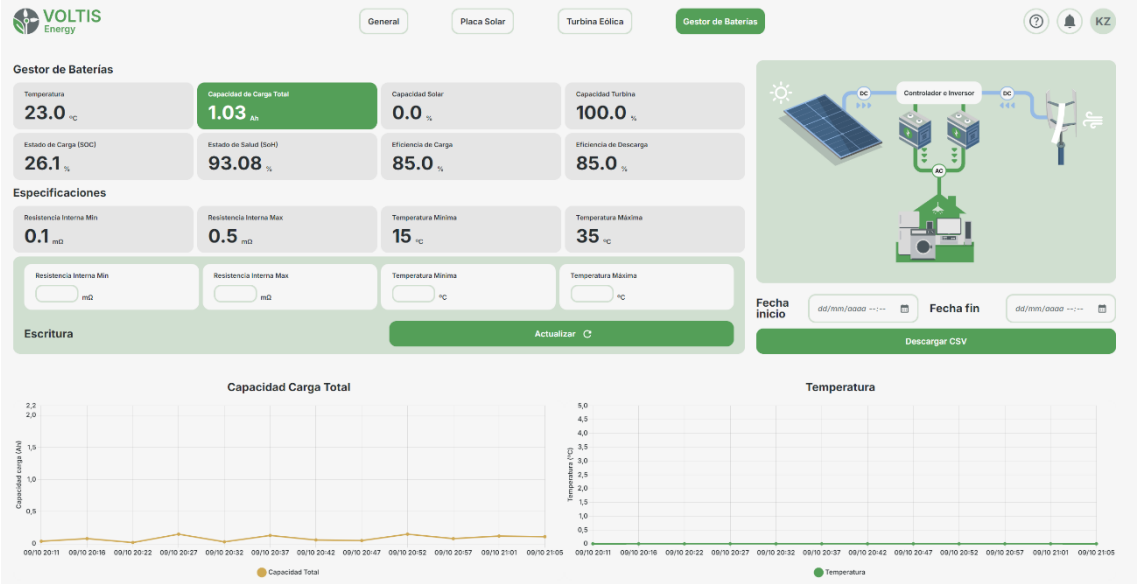

Figura 31. Visualización general del gestor de baterías.

6.3.1. Panel principal de la placa solar

La visualización del panel principal las tres páginas quedarían de la siguiente manera:

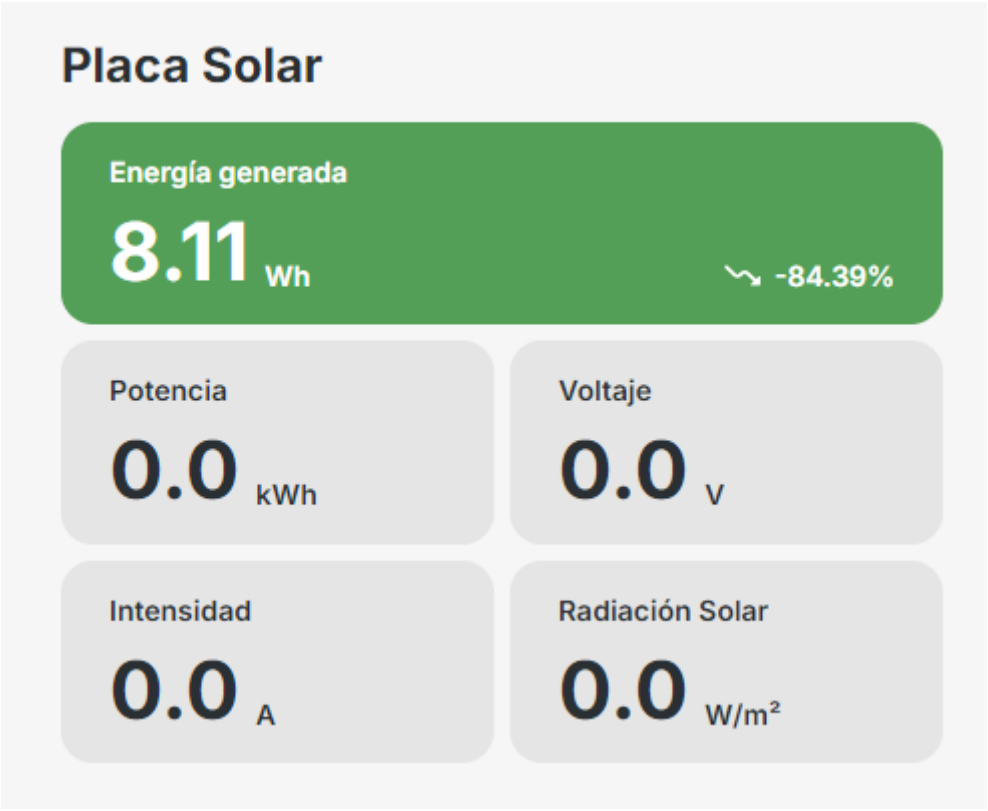

Figura 32. Panel principal placa solar.

Figura 33. Panel principal turbina eólica.

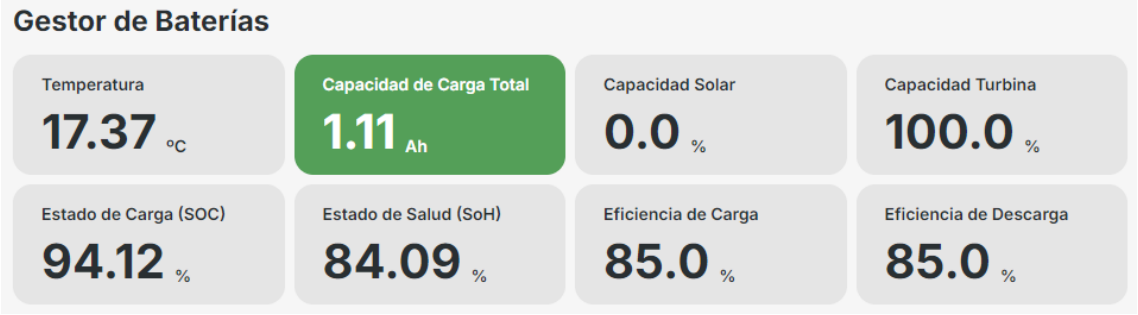

Figura 34. Panel principal del gestor de baterías.

```
<div class="placa-solar">
    <div class="container">
        <div class="cuadros">
            <div class="grupo">
                <div class="placa-solar1">
                    <b class="titulo">Placa Solar</b>
                    <div class="container1">
                        <!-- Muestra los datos de la energía
generada -->
                        <div class="grupo1">
                            <div class="titulo1">Energía
generada</div>
                            <div class="variable-grupo">
                                <b class="numero">{{ data[-
1]['energy_wh'] }}</b>
                                <div class="unidad">Wh</div>
                            </div>
                        </div>
```

Fragmento código 75. Panel principal de la placa solar.

6.3.2. Sección de especificaciones y actualización de parámetros

La visualización de las tres páginas sobre este componente:

Figura 35. Especificaciones de la placa solar.

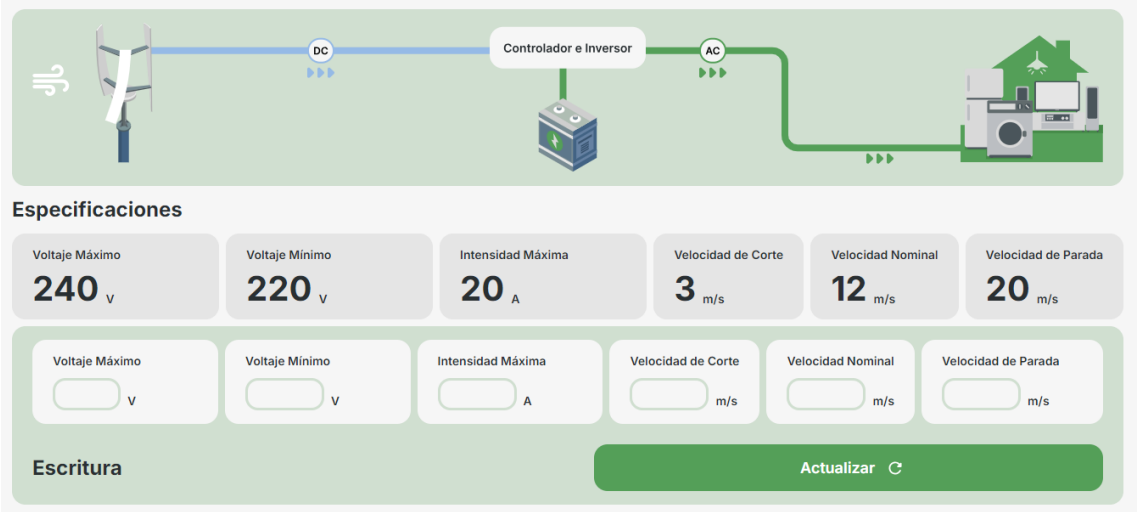

Figura 36. Especificaciones de la turbina eólica.

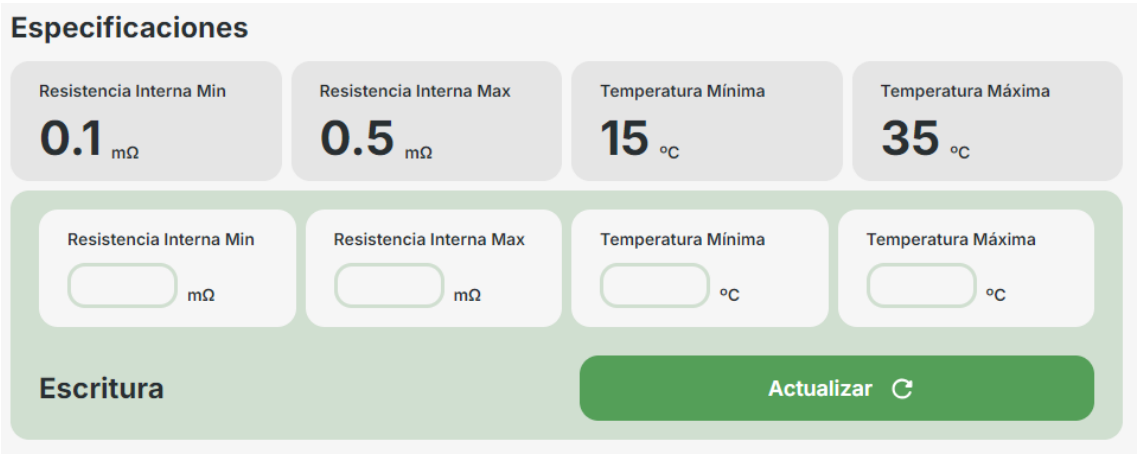

Figura 37. Especificaciones del gestor de baterías.

```
<div class="especificaciones">
    <b class="titulo">Especificaciones</b>
    <div class="container2">
        <div class="intensidad">
            <div class="titulo1">Voltaje Máximo</div>
            <div class="variable-grupo">
                <b class="numero" data-variable="voltage_max"></b>
                <div class="unidad">V</div>
            </div>
        </div>
    </div>
```

Fragmento código 76. Especificaciones y actualización de parámetros.

Los usuarios pueden ver y editar los límites del sistema, como el voltaje e intensidad máximos. Los datos se muestran de forma dinámica y pueden actualizarse mediante el envío de valores a

través del formulario interactivo que incluye campos de entrada. Para que los datos se muestren y puedan modificarse, ha sido necesario desarrollar dos scripts.

- Script para obtener y mostrar los límites actuales.

Este script realiza una solicitud a la URL *http://localhost:5100/limits* para obtener los valores actuales de los límites de voltaje e intensidad, y los muestra en la página.

```
<script>
    // Obtener los límites actuales y mostrarlos
    function getLimits() {
        fetch('http://localhost:5100/limits')
            .then(response => response.json())
            .then(data => {
                // Seleccionamos todos los elementos con la clase
'numero'
                const elementosNumero =
document.querySelectorAll('.numero');

                // Iteramos sobre cada elemento y asignamos el valor
correspondiente basado en 'data-variable'
                elementosNumero.forEach(element => {
                    const variable = element.getAttribute('data-
variable');
                    if (variable === 'voltage_max') {
                        element.textContent = data.voltage_max;
                    } else if (variable === 'current_max') {
                        element.textContent = data.current_max;
                    }
                });

                // Asignar valores a los elementos de escritura
                document.querySelector('.numero7').textContent =
data.voltage_max;
                document.querySelector('.numero8').textContent =
data.current_max;
            })
            .catch(error => console.error('Error fetching limits:',
error));
    }

    // Obtener los límites al cargar la página
    getLimits();
</script>
```

Fragmento código 77. Código para obtener y mostrar los límites actuales.

- Script para actualizar los límites

Este script permite a los usuarios actualizar los límites de voltaje e intensidad a través de un formulario. Envía los datos al servidor utilizando una solicitud **POST**.

```
<script>
    // Manejar la actualización de los límites
    document.querySelector('.boton
button').addEventListener('click', function (event) {
        event.preventDefault();
        const limitsData = {};

        // Obtener los valores actuales de los elementos 'numero7' y
'numero8'
        const voltageMax = document.querySelector('.numero7').value;
        const currentMax = document.querySelector('.numero8').value;

        if (voltageMax) {
            limitsData.voltage_max = parseFloat(voltageMax);
        }

        if (currentMax) {
            limitsData.current_max = parseFloat(currentMax);
        }

        console.log("Enviando los siguientes datos:", limitsData);

        fetch('http://localhost:5100/limits', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(limitsData)
        })
            .then(response => response.json())
            .then(data => {
                alert(data.message);
                getLimits();  // Actualizar los límites mostrados

                // Limpiar los inputs después de actualizar
                document.querySelector('.numero7').value = "";
                document.querySelector('.numero8').value = "";
            })
            .catch(error => console.error('Error actualizando los
límites:', error));
    });
</script>
```

Fragmento código 78. Código para actualizar los límites.

6.3.3. Descarga de datos

Los usuarios pueden seleccionar un rango de fechas y descargar los datos de generación solar en formato CSV.

```
<div class="especificaciones">
    <b class="titulo">Especificaciones</b>
    <div class="container2">
        <div class="intensidad">
            <div class="titulo1">Voltaje Máximo</div>
            <div class="variable-grupo">
                <b class="numero" data-variable="voltage_max"></b>
                <div class="unidad">V</div>
            </div>
        </div>
    </div>
```

Fragmento código 79. Descarga de datos.

Ha sido necesario un script que maneja la interacción del usuario con los campos de selección de fecha para descargar los datos en formato de CSV.

```
<script>
    // Agregar event listener para el campo de fecha de inicio
    document.getElementById('start_date').addEventListener('change',
function () {
        const startDate = this.value;
        console.log("Fecha de inicio seleccionada:", startDate);
    });

    // Agregar event listener para el campo de fecha de fin
    document.getElementById('end_date').addEventListener('change',
function () {
        const endDate = this.value;
        console.log("Fecha de fin seleccionada:", endDate);
    });

    // Ver las fechas en el formulario cuando se envía
    document.getElementById('download-
form').addEventListener('submit', function (event) {
        const startDate =
document.getElementById('start_date').value;
        const endDate = document.getElementById('end_date').value;

        // Asignar los valores a los inputs ocultos
        document.getElementById('hidden_start_date').value =
startDate;
        document.getElementById('hidden_end_date').value = endDate;

        // Validar que las fechas no estén vacías
        if (!startDate || !endDate) {
            alert('Por favor, selecciona ambas fechas antes de
descargar el CSV.');
            event.preventDefault();  // Evitar que se envíe el
formulario si están vacías
        }

        console.log("Fecha de inicio en el formulario:", startDate);
        console.log("Fecha de fin en el formulario:", endDate);
    });
</script>
```

Fragmento código 80. Configuración de descarga de datos.

La visualización de descarga de datos es la misma para las tres pantallas.

Figura 38. Selector de fechas y descarga de datos.

6.3.4. Gráficos interactivos con Chart.js

Y la visualización gráfica queda de la siguiente manera:

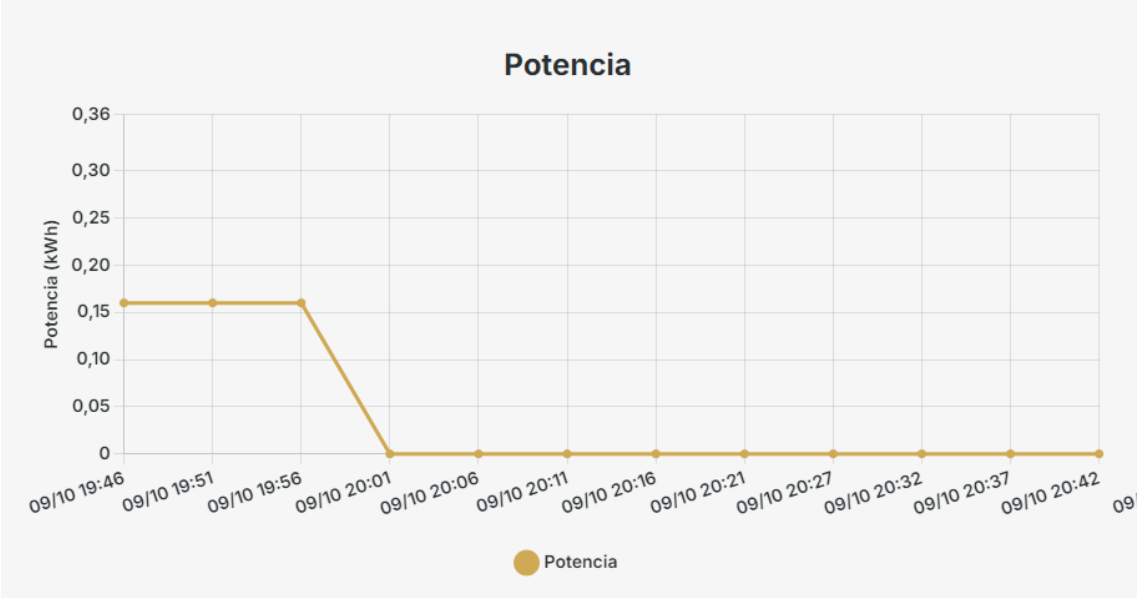

Figura 39. Gráfica de potencia.

En esta sección, se utilizan gráficos interactivos generados con **Chart.js**. Se crean cuatro gráficos que muestran la evolución de la **potencia**, **energía**, **voltaje** e **intensidad** a lo largo del tiempo. Los datos para los gráficos se extraen del *backend* y se procesan dinámicamente.

```
<canvas id="chart_power" style="width:100%; height:100%;"></canvas>
```

Fragmento código 81. Localización del gráfico.

Hace falta un script donde se configura las gráficas y las señales que se quiere mostrar. En primer lugar, se declara todas las variables de las gráficas.

```
<script>
    // Asignar los datos de energía solar, eólica y sus timestamps
    const data_solar = {{ data[-12:] | tojson }};

    // Extraer los timestamps del array de datos de solar
    const timestamps = data_solar.map(d => {
        const date = new Date(d.timestamp);
        const day = String(date.getDate()).padStart(2, '0');
        const month = String(date.getMonth() + 1).padStart(2, '0');
        const hours = String(date.getHours()).padStart(2, '0');
        const minutes = String(date.getMinutes()).padStart(2, '0');
        return `${day}/${month} ${hours}:${minutes}`;
    });

    // Extraer la energía de solar para la gráfica
    const powerData = data_solar.map(d => d.power || 0);
    const energyData = data_solar.map(d => d.energy_generated || 0);
    const voltageData = data_solar.map(d => d.voltage || 0);
    const currentData = data_solar.map(d => d.current || 0);
```

Fragmento código 82. Declaración de las variables de las gráficas.

A continuación, se muestra el código de una gráfica, en la pantalla de la placa solar hay 4, solo se tendría que cambiar las variables en el resto de las gráficas.

```
    // Configuración para la gráfica Potencia
    var power_ctx =
document.getElementById('chart_power').getContext('2d');
    new Chart(power_ctx, {
        type: 'line',
        data: {
            labels: timestamps,
            datasets: [{
                label: 'Potencia',
                data: powerData,
                borderColor: '#CFA953',
                backgroundColor: '#CFA953',
            }]
        },
        options: {
            scales: {
                y: {
                    beginAtZero: true,
                    max: Math.max(...powerData) + 0.2,
                    title: {
                        display: true,
                        text: 'Potencia (kWh)',
                    },
                }
            }
        }
    });
```

Fragmento código 83. Creación de la gráfica de potencia.

6.4. Conclusión

En esta sección, se ha mostrado cómo las páginas HTML están diseñadas para interactuar con los datos del sistema, visualizarlos y permitir su manipulación. A través del uso de tecnologías como *Flask*, Chart.js y el manejo de solicitudes HTTP, se crea una interfaz gráfica que permite al usuario acceder a datos en tiempo real sobre la energía generada, el voltaje, la intensidad y otros parámetros clave de las máquinas simuladas.

El sistema también permite descargar estos datos en formato CSV, lo cual facilita su análisis y almacenamiento. Además, el uso de JavaScript para la actualización automática de la página y la creación de gráficos interactivos asegura que el usuario tenga una visión actualizada y detallada del funcionamiento del sistema.

Finalmente, la integración de estilos mediante CSS y el uso de bibliotecas externas como Google *Fonts* mejora la usabilidad y la experiencia visual del sistema, haciendo la interfaz clara, intuitiva y eficiente para monitorear y gestionar los dispositivos en tiempo real.

CAPÍTULO 7. TRABAJOS FUTUROS Y MEJORAS PROPUESTAS

Este proyecto, aunque funcional y completo en su implementación, abre la puerta a múltiples oportunidades de mejora y expansión. A continuación, se describen algunas líneas de trabajo futuras y mejoras que podrían potenciar el rendimiento y alcance del sistema:

7.1. Ampliación de dispositivos y fuentes de energía

Una mejora inmediata sería integrar más dispositivos al sistema, como paneles solares adicionales, generadores eólicos de diferentes características o incluso otras fuentes de energía renovable, como turbinas hidroeléctricas o sistemas de almacenamiento avanzado. Esta ampliación permitiría evaluar cómo interactúan las distintas fuentes energéticas y cómo se puede optimizar el uso de la energía generada en tiempo real.

7.2. Optimización del almacenamiento y procesamiento de datos

Actualmente, el sistema almacena los datos generados en bases de datos relacionales. A medida que se incremente el número de dispositivos o la frecuencia de adquisición de datos, podría resultar más eficiente el uso de soluciones de *Big Data* o bases de datos distribuidas como *Cassandra* o *InfluxDB*. Estas alternativas facilitarían la gestión y el análisis de grandes volúmenes de datos, permitiendo, además, la creación de predicciones a largo plazo utilizando algoritmos de *Machine Learning.*

7.3. Automatización y gestión inteligente

Otra línea de trabajo futura sería la automatización de la toma de decisiones basadas en los datos recopilados. Se podría diseñar un sistema que, utilizando técnicas de **inteligencia artificial**, gestione automáticamente los límites de voltaje e intensidad en función de las condiciones climáticas o los niveles de energía generada y almacenada. Además, este sistema inteligente podría priorizar el uso de fuentes de energía según su eficiencia en tiempo real.

7.4. Mejora en la seguridad del sistema

Aunque el sistema actual utiliza un túnel SSH para la transmisión segura de datos, la seguridad siempre es un área por mejorar. Se podría implementar autenticación *multifactor* (MFA) para el acceso a la plataforma o utilizar certificados digitales para cada dispositivo conectado a la red. Asimismo, la adopción de protocolos de seguridad como TLS 1.3 para todas las comunicaciones sería una mejora considerable.

7.5. Visualización de datos avanzada

El sistema podría beneficiarse de una interfaz gráfica más avanzada, que incluya visualizaciones 3D o mapas interactivos que muestren la localización de los dispositivos, su estado y el rendimiento energético en tiempo real. Además, se podría permitir al usuario configurar alertas personalizadas para ciertos parámetros críticos, como el sobrecalentamiento de las baterías o la baja producción de energía solar.

7.6. Implementación de módulos de energía predictiva

Una extensión importante sería la implementación de un sistema predictivo basado en el análisis de datos históricos. Este sistema podría predecir la producción futura de energía basada en factores como las condiciones climáticas, la demanda energética, o el ciclo solar y eólico, ayudando a optimizar el uso y almacenamiento de energía.

7.7. Escalabilidad y despliegue en la nube

Finalmente, el despliegue del sistema en una infraestructura de **nube** pública o privada permitiría una mayor escalabilidad, facilitando la gestión de un mayor número de dispositivos y fuentes de energía distribuidas. Plataformas como **AWS** o **Azure** podrían ofrecer servicios adicionales, como análisis de datos en tiempo real o almacenamiento masivo, sin comprometer la velocidad y seguridad del sistema.

7.8. Conclusión

Con estas mejoras y expansiones, el sistema podría evolucionar de una solución funcional para la gestión de energía en tiempo real a una plataforma robusta, segura y escalable, capaz de adaptarse a entornos más complejos y ofrecer un mayor valor en términos de eficiencia y automatización.

CAPÍTULO 8. BIBLIOGRAFÍA

[1] A. Ipakchi y F. Albuyeh, «Grid of the future,» *IEEE Power and Energy Magazine*, vol. 7, no. 2, pp. 52-62, 2009. [En línea]. Available: https://doi.org/10.1109/MPE.2008.931384.

[2] G. Andersson, «Overview of Electric Power Systems,» *IEEE Power and Energy Magazine*, vol. 5, no. 2, pp. 20-27, 2007. [En línea]. Available: https://doi.org/10.1109/MPE.2007.290877.

[3] S. Massoud Amin y B. F. Wollenberg, «Toward a smart grid,» *IEEE Power and Energy Magazine*, vol. 3, no. 5, pp. 34-41, 2005. [En línea]. Available: https://doi.org/10.1109/MPAE.2005.1507024.

[4] H. Farhangi, «The path of the smart grid,» *IEEE Power and Energy Magazine*, vol. 8, no. 1, pp. 18-28, 2010. [En línea]. Available: https://doi.org/10.1109/MPE.2009.934876.

[5] M. D. Galus, «Integration of Renewable Energy in Smart Grids,» *IEEE Transactions on Smart Grid*, vol. 3, no. 3, pp. 1581-1589, 2012. [En línea]. Available: https://doi.org/10.1109/TSG.2012.2208635.

[6] M. A. Zamani, et al., «Smart Grid Cyber-Physical System Resilience,» *IEEE Transactions on Smart Grid*, vol. 10, no. 3, pp. 2572-2584, 2019. [En línea]. Available: https://doi.org/10.1109/TSG.2018.2877087.

[7] A. R. Metke y R. L. Ekl, «Security Technology for Smart Grid Networks,» *IEEE Transactions on Smart Grid*, vol. 1, no. 1, pp. 99-107, 2010. [En línea]. Available: https://doi.org/10.1109/TSG.2010.2046347.

[8] S. McLaughlin et al., «Cyber-Physical Security of Smart Grid Infrastructure,» *Proceedings of the IEEE*, vol. 100, no. 1, pp. 69-85, 2012. [En línea]. Available: https://doi.org/10.1109/JPROC.2011.2161428.

[9] E. W. Gunther et al., «Cyber-Physical Approaches for Electric Power Grid Resilience,» *IEEE Transactions on Smart Grid*, vol. 6, no. 5, pp. 2514-2523, 2015. [En línea]. Available: https://doi.org/10.1109/TSG.2015.2408749.

[10] Global Electricity, «Diferencias entre Smart Grids y redes eléctricas convencionales,» 19 de diciembre de 2013. [En línea]. Available: https://globalelectricity.wordpress.com/2013/12/19/diferencias-entre-smart-grids-y-redes-electricas-convencionales/.

[11] Raspberry Pi, «Raspberry Pi 5,» 2023. [En línea]. Available: https://www.raspberrypi.com/products/raspberry-pi-5/.

[12] BeagleBoard, «BeagleBone Black Industrial,» [En línea]. Available: https://www.beagleboard.org/boards/beaglebone-black-industrial.

[13] Arduino, «Arduino Hardware,» [En línea]. Available: https://www.arduino.cc/en/hardware.

[14] IBM, «Hypervisors,» [En línea]. Available: https://www.ibm.com/topics/hypervisors.

[15] M. Grinberg, *Flask Web Development: Developing Web Applications with Python*, 2nd ed. O'Reilly Media, 2018.

[16] Flask Documentation, "Why Flask?", Flask Official Documentation, [En línea]. Available: https://flask.palletsprojects.com/en/2.0.x/foreword/#why-flask

[17] J. Palacio y M. Smith, *Flask by Example: Unleash the full potential of Flask to create a smart web application*, Packt Publishing, 2018.

[18] A. Palacio, "Best Practices for Flask Application Structure," Flask Documentation, 2021. [En línea]. Available: https://flask.palletsprojects.com/en/2.0.x/patterns/packages/

[19] A. G. Wilson, "Why SQLAlchemy is Perfect for Flask and Web Development," *Practical Flask*, 2020.

[20] A. Metke y R. L. Ekl, «Security Technology for Smart Grid Networks,» *IEEE Transactions on Smart Grid*, vol. 1, no. 1, pp. 99-107, 2010. [En línea]. Available: https://doi.org/10.1109/TSG.2010.2046347.

[21] M. Grinberg, *Flask Web Development*, O'Reilly Media, 2014.

[22] M. Stonebraker y A. Kemnitz, «The Postgres Next-Generation Database Management System,» *Communications of the ACM*, vol. 34, no. 10, pp. 78-92, 1991. [En línea]. Available: https://doi.org/10.1145/125223.125262.

[23] PostgreSQL Global Development Group, "PostgreSQL: Introduction and Concepts," *PostgreSQL Documentation*, 2021. [En línea]. Available: https://www.postgresql.org/docs/current/index.html.

[24] A. Dignös, M. Fischer y S. Kemper, «SQL-based Temporal Query Languages,» *Proceedings of the 2019 ACM SIGMOD International Conference on Management of Data*, 2019. [En línea]. Available: https://doi.org/10.1145/3299869.3319875.

[25] J. S. A. Bridges y K. G. Satos, "Extending PostgreSQL to Support Complex Data Types," *Practical PostgreSQL*, O'Reilly Media, 2002.

[26] PostgreSQL Global Development Group, "Transactions in PostgreSQL," *PostgreSQL Documentation*, 2021. [En línea]. Available: https://www.postgresql.org/docs/current/transaction-iso.html.

PRESUPUESTO

PRESUPUESTO

Este documento muestra el presupuesto estimado del proyecto, de manera que se proporcione información suficiente sobre la dotación económica necesaria para efectuar su realización.

1. Cuadro mano de obra

Código	Ud.	Denominación	Precio (€)
MO.ING	h	Ingeniero de Industrial	30
MO.DIS	h	Diseñador UX/UI	25

2. Cuadro maquinaria y software

Código	Ud.	Denominación	Precio anual (€)	Precio (€)
M.ORD	h	Ordenador de Desarrollo	750	0,43
M.PYTH	h	Python	0	0
M.VSCODE	h	Visual Code	0	0
M.FLASK	h	Flask	0	0
M.MSOFF	h	Microsoft Office	69	0,04
M.POSTG	h	PostgreSQL	0	0
M.FIGMA	mes	Figma	15	15
M.VBOX	h	VirtualBox (Software Libre)	0	0
M.DOCK	h	Docker Compose	0	0
M.CFIBRA	h	Conexión fibra internet	396	0,23

Para determinar el precio por hora del ordenador, de 1500€, se ha considerado un periodo de amortización de 2 años. Estimando que en un año se trabajan 11 meses, con aproximadamente 20 días de trabajo al mes con jornadas de 8 horas, se obtienen 1760 h/año.

3. Cuadro precios descompuestos

Etapa 1: Diseño y programación

UO1.1	**Diseño, Programación y Maquetación de la Web**				
Descripción:	Diseño, redacción de documentos realizados para el proyecto y programación				
Código	Ud.	Denominación	Precio (€)	Rendimiento	Importe (€)
MO.ING	h	Ingeniero Industrial	30,00 €	300	9.000,00 €
MO.DIS	h	Diseñador UX/UI (Figma)	25,00 €	15	375,00 €
M.ORD	h	Ordenador de Desarrollo	0,43 €	315	135,45 €
M.FIGMA	mes	Figma (Herramienta Diseño)	15,00 €	1	15,00 €
M.MSOFF	h	Microsoft Office	0,04 €	315	12,60 €
M.VSCODE	h	Visual Code	- €	1	- €
M.POSTG	h	PostgreSQL	- €	1	- €
M.FLASK	h	Flask (Software Libre)	- €	1	- €
M.CFIBRA	h	Conexión fibra internet	0,23 €	315	72,45 €
CDC	%	Costes directos complementarios	96,11 €	4	384,42 €
		Total unidad de obra			9.994,92 €

Etapa 2: Compilación y verificación

UO1.2	**Instalación y Verificación**				
Descripción:	Compilación del código y verificación del buen funcionamiento				
Código	Ud.	Denominación	Precio (€)	Rendimiento	Importe (€)
M.VBOX	h	VirtualBox (Software Libre)	- €	1	- €
M.DOCK	h	Docker Compose	- €	1	- €
MO.ING	h	Ingeniero Industrial	30,00 €	100	3.000,00 €
M.ORD	h	Ordenador de Desarrollo	0,43 €	100	43,00 €
M.CFIBRA	h	Conexión fibra internet	0,23 €	100	23,00 €
CDC	%	Costes directos complementarios	30,66 €	3	91,98 €
Total					3.157,98 €

4. Presupuesto total

Código	Ud.	Denominación	Medición	Precio (€)	Importe (€)
UO1.1	u	Programación de la página web	1	9.994,92 €	9.994,92 €
UO2.1	u	Implementación del código	1	3.157,98 €	3.157,98 €

Presupuesto de ejecución material	**13.152,90 €**
Gastos generales 12%	1.578,35 €
Beneficio industrial 6%	789,17 €
Presupuesto de ejecución por contrata	**15.520,42 €**
I.V.A 21%	3.259,29 €
Presupuesto base de licitación	**18.779,71 €**

Asciende el presente presupuesto a la expresada cantidad de:

DIECIOCHO MIL SETECIENTOS SETENTA Y NUEVE EUROS Y SETENTA Y UNO CÉNTIMOS

ANEXOS

Python Scripts máquinas virtuales

create_vms.py

```
# Librerías empleadas
import subprocess
import os

# Añadir manualmente la ruta a VBoxManage.exe al PATH
vbox_path = r"C:\Program Files\Oracle\VirtualBox"
os.environ["PATH"] += os.pathsep + vbox_path


def create_vm(vm_name, iso_path, ram_size_mb):
    # Crear la VM
    subprocess.run([
        "VBoxManage", "createvm", "--name", vm_name, "--ostype",
"Ubuntu_64", "--register"
    ])

    # Ajustar la memoria RAM
    subprocess.run([
        "VBoxManage", "modifyvm", vm_name, "--memory",
str(ram_size_mb)
    ])

    # Ajustar el adaptador de red
    subprocess.run([
        "VBoxManage", "modifyvm", vm_name, "--nic1", "bridged", "--
bridgeadapter1", "Intel(R) Wi-Fi 6 AX201 160MHz"
    ])

    # Crear y añadir un disco duro virtual
    hdd_path = os.path.join(os.path.expanduser("~"), f"{vm_name}.vdi")
    subprocess.run([
        "VBoxManage", "createhd", "--filename", hdd_path, "--size",
"10240"
    ])
    subprocess.run([
        "VBoxManage", "storagectl", vm_name, "--name", "SATA
Controller", "--add", "sata"
    ])
    subprocess.run([
        "VBoxManage", "storageattach", vm_name, "--storagectl", "SATA
Controller", "--port", "0", "--device", "0", "--type", "hdd", "--
medium", hdd_path
    ])

    # Añadir la unidad óptica con la ISO de Ubuntu
    subprocess.run([
        "VBoxManage", "storageattach", vm_name, "--storagectl", "SATA
Controller", "--port", "1", "--device", "0", "--type", "dvddrive", "--
medium", iso_path
    ])

    # Iniciar la VM
    subprocess.run([
        "VBoxManage", "startvm", vm_name, "--type", "gui"
    ])

    print(f"VM '{vm_name}' created and started.")
```

```
# Directorio de la iso de Ubuntu
iso_path = "ubuntu-24.04-desktop-amd64.iso"

# Creación de las VMs
for i in range(3):
    vm_name = f"Ubuntu_VM_{i+1}"
    ram_size_mb = 3072  # 3 GB
    create_vm(vm_name, iso_path, ram_size_mb)
```

solar_simulation.py

```
from flask import Flask, jsonify, request
from flask_cors import CORS  # Importa la extension CORS
import math
import random
import requests
import datetime

app = Flask(__name__)
CORS(app)  # Habilita CORS para todas las rutas

# Límites globales para voltage y current
voltage_min = 220
voltage_max = 240
current_min = 5
current_max = 15


def get_voltage_current_based_on_radiation(solar_radiation,
voltage_max, current_max):
    """
    Calcula la tensión (voltaje) y la corriente (intensidad) basadas
en la radiación solar.
    """
    # Asumimos una relación lineal para simplificar
    # La radiación máxima se toma como 1000 W/m²
    voltage = (solar_radiation / 800) * voltage_max
    # Relación lineal también para la corriente
    current = (solar_radiation / 800) * current_max

    # Limitamos la tensión y la corriente a sus valores máximos
configurados
    voltage = min(voltage, voltage_max)
    current = min(current, current_max)

    return voltage, current


def get_solar_radiation_curve(hour):
    """
    Calcula la radiación solar en función de la hora del día
utilizando una función
    que simula la curva de radiación típica en un día soleado.
    """
    # Parámetros de la función para un día completo de 6:00 a 20:00
```

```
    sunrise = 6  # 6:00 AM
    sunset = 20  # 8:00 PM
    peak_hour = 14  # 2:00 PM (aproximadamente el pico)

    if hour < sunrise or hour > sunset:
        return 0  # No hay radiación solar fuera de las horas del día

    # Normalización de la hora para que vaya de 0 a 1 durante el día
    normalized_hour = (hour - sunrise) / (sunset - sunrise)

    # Curva parabólica invertida para simular el incremento y
decremento de la radiación
    radiation = -4 * (normalized_hour - 0.5) ** 2 + 1

    # Escalar la radiación para que vaya de 0 a 1000 W/m²
    max_radiation = 800  # Pico máximo de radiación alrededor del
mediodía
    solar_radiation = radiation * max_radiation

    return solar_radiation


def get_solar_data():
    """
    Genera datos de simulación para la radiación solar, voltaje,
corriente, potencia y energía.
    """
    current_hour = datetime.datetime.now().hour

    solar_radiation = get_solar_radiation_curve(current_hour)  # W/m²

    # Calcula la tensión y la corriente en función de la radiación
solar
    voltage, current = get_voltage_current_based_on_radiation(
        solar_radiation, voltage_max, current_max)

    # Calcula la potencia
    power = voltage * current  # Potencia en vatios

    # Calcula la energía producida en 1 minuto
    energy = power / 60  # Energía en kWh (para un intervalo de 1
minuto)

    data = {
        "timestamp": datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S"),
        "location": "Valencia, Spain",
        "solar_radiation": solar_radiation,
        "voltage": voltage,
        "current": current,
        "power": power,
        "energy_generated": energy_generated
    }
    return data


@app.route('/solar', methods=['GET'])
def solar_data():
```

```
    data = get_solar_data()
    return jsonify(data)


@app.route('/limits', methods=['GET', 'POST'])
def limits():
    global voltage_min, voltage_max, current_min, current_max

    if request.method == 'POST':
        limits_data = request.json
        if 'voltage_max' in limits_data and limits_data['voltage_max']
is not None:
            voltage_max = limits_data['voltage_max']
        if 'current_max' in limits_data and limits_data['current_max']
is not None:
            current_max = limits_data['current_max']
        return jsonify({"message": "Limits updated successfully"}),
200

    return jsonify({
        "voltage_max": voltage_max,
        "current_max": current_max
    })


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5100)
```

wind_generator.py

```
from flask import Flask, jsonify, request
from flask_cors import CORS  # Importa la extension CORS
from collections import deque
import random
import datetime

app = Flask(__name__)
CORS(app)  # Habilita CORS para todas las rutas

# Límites globales para voltage y current
voltage_max = 240
voltage_min = 220
current_max = 20
v_cut_in = 3   # Velocidad de arranque (m/s)
v_nominal = 12  # Velocidad nominal (m/s)
v_cut_out = 20  # Velocidad de corte (m/s)

# Inicialización de la lista para almacenar los últimos 12 valores
energia_intervalos = deque(maxlen=12)  # Solo guarda los últimos 12
valores


def get_wind_data():
    # Simulación basada en tramos horarios
    now = datetime.datetime.now()
    hour = now.hour
```

```
    # Velocidades del viento típicas (en m/s) en Valencia a lo largo
de un día
    if 0 <= hour < 6:
        wind_speed = random.uniform(3, 5)  # madrugada, vientos bajos
    elif 6 <= hour < 12:
        wind_speed = random.uniform(5, 10)  # mañana, vientos
moderados
    elif 12 <= hour < 18:
        wind_speed = random.uniform(10, 20)  # tarde, vientos fuertes
    else:
        wind_speed = random.uniform(5, 10)  # noche, vientos moderados

    if wind_speed < v_cut_in or wind_speed > v_cut_out:
        voltage = 0
        current = 0
        power = 0
    elif v_cut_in <= wind_speed < v_nominal:
        # Escalado lineal entre el voltaje mínimo y máximo
        voltage = voltage_min + (wind_speed - v_cut_in) / \
            (v_nominal - v_cut_in) * (voltage_max - voltage_min)
        # Potencia proporcional al cubo de la velocidad del viento
        power = 0.5 * 1.225 * 1 * wind_speed**3 * 0.3  # Asumiendo
Cp=0.3 y A=1 m²
        current = power / voltage
    else:
        # Potencia constante a partir de la velocidad nominal
        voltage = voltage_max
        power = voltage * current_max
        current = current_max

    # Energía generada en el intervalo de 5 min en Wh
    energy_generated = power * (1/60)

    # Agregar el valor actual a la lista (si no hay 12 valores aún, se
agregarán)
    energia_intervalos.append(energy_generated)

    # Energía total en la última hora (suma de los últimos 12 valores)
    energia_total_hora = sum(energia_intervalos)

    data = {
        "timestamp": datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S"),
        "wind_speed": wind_speed,
        "voltage": voltage,
        "current": current,
        "power": power,
        "energy_generated": energy_generated,
        "energy_wh": energia_total_hora
    }
    return data


@app.route('/wind', methods=['GET'])
def wind():
    return jsonify(get_wind_data())
```

```
@app.route('/limits', methods=['GET', 'POST'])
def limits():
    global voltage_min, voltage_max, current_max, v_cut_in, v_nominal,
v_cut_out

    if request.method == 'POST':
        limits_data = request.json
        if 'voltage_max' in limits_data and limits_data['voltage_max']
is not None:
            voltage_max = limits_data['voltage_max']
        if 'voltage_min' in limits_data and limits_data['voltage_min']
is not None:
            voltage_min = limits_data['voltage_min']
        if 'current_max' in limits_data and limits_data['current_max']
is not None:
            current_max = limits_data['current_max']
        if 'v_cut_in' in limits_data and limits_data['v_cut_in'] is
not None:
            v_cut_in = limits_data['v_cut_in']
        if 'v_nominal' in limits_data and limits_data['v_nominal'] is
not None:
            v_nominal = limits_data['v_nominal']
        if 'v_cut_out' in limits_data and limits_data['v_cut_out'] is
not None:
            v_cut_out = limits_data['v_cut_out']
        return jsonify({"message": "Limits updated successfully"}),
200

    return jsonify({
        "voltage_max": voltage_max,
        "voltage_min": voltage_min,
        "current_max": current_max,
        "v_cut_in": v_cut_in,
        "v_nominal": v_nominal,
        "v_cut_out": v_cut_out
    })


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5200)
```

batteries.py

```
from flask import Flask, jsonify, request
from flask_cors import CORS  # Importa la extension CORS
import random
import datetime
from collections import deque

app = Flask(__name__)
CORS(app)  # Habilita CORS para todas las rutas

internal_resistance_min = 0.1
internal_resistance_max = 0.5
temperature_min = 15
temperature_max = 35
```

```
capacidad_intervalos = deque(maxlen=12)


def calculate_capacity(current_solar, current_generator):
    # Capacidad de la placa solar
    capacity_solar = current_solar * 0.08
    # Capacidad del generador
    capacity_generator = current_generator * 0.08
    # Capacidad total es la suma de ambas capacidades
    total_capacity = capacity_solar + capacity_generator

    return total_capacity


def calculate_efficiency(voltage, current):
    # Un ejemplo simple de cómo podrías relacionar la eficiencia con
el voltaje y la corriente
    if 12 <= voltage <= 14 and 5 <= current <= 10:
        return 95  # Alta eficiencia cuando los valores están en un
rango óptimo
    elif voltage < 12 or voltage > 14 or current < 5 or current > 10:
        return 85  # Eficiencia más baja cuando los valores están
fuera del rango óptimo
    else:
        return 90  # Eficiencia media por defecto


def get_battery_data(voltage_solar=None, current_solar=None,
voltage_generator=None, current_generator=None):
    global internal_resistance_min, internal_resistance_max,
temperature_min, temperature_max

    # Verifica si los valores de voltaje e intensidad fueron
proporcionados para la placa solar
    current_solar = current_solar if current_solar is not None else
random.uniform(
        1, 10)

    # Verifica si los valores de voltaje e intensidad fueron
proporcionados para el generador
    current_generator = current_generator if current_generator is not
None else random.uniform(
        1, 10)

    # Calcula la capacidad usando la función anterior
    capacity = calculate_capacity(current_solar, current_generator)
    capacidad_intervalos.append(capacity)
    capacidad_total_hora = sum(capacidad_intervalos)

    # Calcula la eficiencia de carga y descarga
    charge_efficiency = calculate_efficiency(voltage_solar,
current_solar)
    discharge_efficiency = calculate_efficiency(
        voltage_generator, current_generator)

    # Define variables adicionales para la simulación
    soc = random.uniform(20, 100)  # State of Charge as a percentage
```

```
    soh = random.uniform(80, 100)  # State of Health as a percentage
    internal_resistance = random.uniform(
        internal_resistance_min, internal_resistance_max)  # Milliohms
    temperature = random.uniform(
        temperature_min, temperature_max)  # Degrees Celsius

    battery = {
        "capacity_solar": (current_solar*0.08)/capacity*100,
        "capacity_generator": (current_generator*0.08)/capacity*100,
        # Capacidad total calculada en Watts (o cualquier otra unidad
que elijas)
        "capacity": capacity,  # Capacidad Ah en un intervalo 5 min
        "capacity_ah": capacidad_total_hora,  # Capacidad Ah en un
intervalo 5 min
        "soc": soc,  # State of Charge
        "soh": soh,  # State of Health
        "internal_resistance": internal_resistance,  # Internal
resistance in milliohms
        "temperature": temperature,  # Temperature in degrees Celsius
        # Eficiencia de carga dependiente del voltaje e intensidad
        "charge_efficiency": charge_efficiency,
        # Eficiencia de descarga dependiente del voltaje e intensidad
        "discharge_efficiency": discharge_efficiency,

    }

    data = {
        "timestamp": datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S"),
        "location": "Valencia, Spain",
        "battery": battery
    }
    return data


@app.route('/batteries', methods=['GET'])
def battery_data():
    # Retrieve custom voltage and current for solar panel and
generator from request parameters
    voltage_solar = request.args.get('voltage_solar', type=float)
    current_solar = request.args.get('current_solar', type=float)
    voltage_generator = request.args.get('voltage_generator',
type=float)
    current_generator = request.args.get('current_generator',
type=float)

    data = get_battery_data(voltage_solar=voltage_solar,
current_solar=current_solar,
                            voltage_generator=voltage_generator,
current_generator=current_generator)
    return jsonify(data)


@app.route('/limits', methods=['GET', 'POST'])
def limits():
    global internal_resistance_min, internal_resistance_max,
temperature_min, temperature_max
```

```
    if request.method == 'POST':
        limits_data = request.json
        if 'internal_resistance_min' in limits_data and
limits_data['internal_resistance_min'] is not None:
            internal_resistance_min =
limits_data['internal_resistance_min']
        if 'internal_resistance_max' in limits_data and
limits_data['internal_resistance_max'] is not None:
            internal_resistance_max =
limits_data['internal_resistance_max']
        if 'temperature_min' in limits_data and
limits_data['temperature_min'] is not None:
            temperature_min = limits_data['temperature_min']
        if 'temperature_max' in limits_data and
limits_data['temperature_max'] is not None:
            temperature_max = limits_data['temperature_max']
        return jsonify({"message": "Limits updated successfully"}),
200

    return jsonify({
        "internal_resistance_min": internal_resistance_min,
        "internal_resistance_max": internal_resistance_max,
        "temperature_min": temperature_min,
        "temperature_max": temperature_max,
    })


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5300)
```

Docker Compose archivos

/docker_tfm/docker-compose.yml

```
version: '3.8'

services:
  db:
    image: postgres:latest
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: datadb
    volumes:
      - db_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  backend:
    build: ./backend
    volumes:
      - ./backend:/app
    ports:
      - "5000:5000"
    depends_on:
      - db
```

```
  frontend:
    build: ./frontend
    volumes:
      - ./frontend:/app
    ports:
      - "8000:8000"
    depends_on:
      - backend

  script_runner:
    build: ./script_runner
    volumes:
      - ./script_runner:/app
    depends_on:
      - backend
    environment:
      - PYTHONUNBUFFERED=1
    network_mode: "host"

volumes:
  db_data:
```

/docker_tfm/script_runner/Dockerfile

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["sh", "-c", "while true; do python send_data.py; sleep 300;
done"]
```

/docker_tfm/script_runner/requirements.txt

```
Requests
```

/docker_tfm/script_runner/send_data.py

```
import requests
import time
from datetime import datetime

# URL de la máquinas virtuales
vm_url_solar = "http://192.168.1.54:5100/solar"
vm_url_wind = "http://192.168.1.55:5200/wind"
vm_url_batteries = "http://192.168.1.56:5300/batteries"
```

```
# URL de los backends
backend_url_solar = "http://localhost:5000/data_solar"
backend_url_wind = "http://localhost:5000/data_wind"
backend_url_batteries = "http://localhost:5000/data_batteries"


def fetch_data_from_machine(vm_url):
    try:
        response = requests.get(vm_url)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Error fetching data from {vm_url}: {e}")
        return None


def get_voltage_and_current(vm_data):
    voltage = vm_data.get('voltage')
    current = vm_data.get('current')
    return voltage, current


def send_data_to_machine_3(voltage_solar, current_solar,
voltage_generator, current_generator):
    # Construir la URL con los parámetros necesarios
    url =
f"http://192.168.1.56:5300/batteries?voltage_solar={voltage_solar}&cur
rent_solar={current_solar}&voltage_generator={voltage_generator}&curre
nt_generator={current_generator}"

    try:
        response = requests.get(url)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Error sending data to machine 3: {e}")
        return None


def send_data_to_backend(data, backend_url):
    try:
        response = requests.post(backend_url, json=data)
        response.raise_for_status()
        print("Data sent successfully to backend.")
    except requests.exceptions.RequestException as e:
        print(f"Error sending data to backend: {e}")


def transform_data_solar(vm_data):
    # Transformar el JSON de la VM al formato requerido por el backend
    return {
        "timestamp": datetime.now().isoformat(),
        "voltage": round(vm_data.get("voltage"), 2),
        "current": round(vm_data.get("current"), 2),
        "power": round(vm_data.get("power")/1000, 2),
        "energy_generated": round(vm_data.get("energy_generated"), 2),
        "solar_radiation": round(vm_data.get("solar_radiation"), 2),
```

```
        "energy_wh": round(vm_data.get("energy_wh"), 2)
    }


def transform_data_wind(vm_data):
    # Transformar el JSON de la VM al formato requerido por el backend
    return {
        "timestamp": datetime.now().isoformat(),
        "voltage": round(vm_data.get("voltage"), 2),
        "current": round(vm_data.get("current"), 2),
        "power": round(vm_data.get("power")/1000, 2),
        "energy_generated": round(vm_data.get("energy_generated"), 2),
        "energy_wh": round(vm_data.get("energy_wh"), 2),
        "wind_speed": round(vm_data.get("wind_speed"), 2)
    }


def transform_data_batteries(vm_data):
    # Transformar el JSON de la VM al formato requerido por el backend
    return {
        "timestamp": datetime.now().isoformat(),
        "capacity_solar": round(vm_data.get("capacity_solar"), 2),
        "capacity_generator": round(vm_data.get("capacity_generator"),
2),
        "capacity": round(vm_data.get("capacity")/1000, 2),
        "capacity_ah": round(vm_data.get("capacity_ah")/1000, 2),
        "soc": round(vm_data.get("soc"), 2),
        "soh": round(vm_data.get("soh"), 2),
        "internal_resistance":
round(vm_data.get("internal_resistance"), 2),
        "temperature": round(vm_data.get("temperature"), 2),
        "charge_efficiency": round(vm_data.get("charge_efficiency"),
2),
        "discharge_efficiency":
round(vm_data.get("discharge_efficiency"), 2)
    }


if __name__ == "__main__":
    # Obtener datos de la máquina solar
    data_solar = fetch_data_from_machine(vm_url_solar)
    print(data_solar)
    if data_solar:
        voltage_solar, current_solar =
get_voltage_and_current(data_solar)
    else:
        voltage_solar, current_solar = None, None

    # Obtener datos de la máquina 2
    data_wind = fetch_data_from_machine(vm_url_wind)
    print(data_wind)
    if data_wind:
        voltage_generator, current_generator =
get_voltage_and_current(
            data_wind)
    else:
        voltage_generator, current_generator = None, None
```

```
    # Verificar que todos los datos necesarios estén disponibles
    if None not in (voltage_solar, current_solar, voltage_generator,
current_generator):
        # Hacer la solicitud a la máquina 3
        data_batteries = send_data_to_machine_3(
            voltage_solar, current_solar, voltage_generator,
current_generator)
        print(data_batteries)
    else:
        print("No se pudo obtener todas las variables necesarias para
enviar la solicitud a la máquina 3.")

    if data_solar:
        time.sleep(1)  # Pausa de 1 segundo entre solicitudes
        transformed_data_solar = transform_data_solar(data_solar)
        send_data_to_backend(transformed_data_solar,
backend_url_solar)
        time.sleep(1)  # Pausa de 1 segundo entre solicitudes

    if data_wind:
        time.sleep(1)  # Pausa de 1 segundo entre solicitudes
        transformed_data_wind = transform_data_wind(data_wind)
        send_data_to_backend(transformed_data_wind, backend_url_wind)
        time.sleep(1)  # Pausa de 1 segundo entre solicitudes

    if data_batteries:
        time.sleep(1)  # Pausa de 1 segundo entre solicitudes
        transformed_data_batteries =
transform_data_batteries(data_batteries)
        send_data_to_backend(transformed_data_batteries,
backend_url_batteries)
        time.sleep(1)  # Pausa de 1 segundo entre solicitudes
```

/docker_tfm/backend/Dockerfile

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["python", "app.py"]
```

/docker_tfm/backend/requeriments.txt

```
flask
flask_sqlalchemy
psycopg2-binary
```

/docker_tfm/backend/app.py

```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime, timedelta

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'postgresql://user:password@db/datadb'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)


class Data_solar(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    timestamp = db.Column(db.DateTime, nullable=False)
    voltage = db.Column(db.Float, nullable=False)
    current = db.Column(db.Float, nullable=False)
    power = db.Column(db.Float, nullable=False)
    energy_generated = db.Column(db.Float, nullable=False)
    energy_wh = db.Column(db.Float, nullable=False)
    solar_radiation = db.Column(db.Float, nullable=False)


class Data_wind(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    timestamp = db.Column(db.DateTime, nullable=False)
    voltage = db.Column(db.Float, nullable=False)
    current = db.Column(db.Float, nullable=False)
    power = db.Column(db.Float, nullable=False)
    energy_generated = db.Column(db.Float, nullable=False)
    wind_speed = db.Column(db.Float, nullable=False)
    energy_wh = db.Column(db.Float, nullable=False)


class Data_batteries(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    timestamp = db.Column(db.DateTime, nullable=False)
    capacity_solar = db.Column(db.Float, nullable=False)
    capacity_generator = db.Column(db.Float, nullable=False)
    capacity = db.Column(db.Float, nullable=False)
    capacity_ah = db.Column(db.Float, nullable=False)
    soc = db.Column(db.Float, nullable=False)
    soh = db.Column(db.Float, nullable=False)
    internal_resistance = db.Column(db.Float, nullable=False)
    temperature = db.Column(db.Float, nullable=False)
    charge_efficiency = db.Column(db.Float, nullable=False)
    discharge_efficiency = db.Column(db.Float, nullable=False)


def create_tables():
    with app.app_context():
        db.create_all()


@app.route('/data_solar', methods=['POST'])
def receive_data_solar():
```

```
    data = request.json
    new_data = Data_solar(
        timestamp=datetime.fromisoformat(data['timestamp']),
        voltage=data['voltage'],
        current=data['current'],
        power=data['power'],
        energy_generated=data['energy_generated'],
        energy_wh=data['energy_wh'],
        solar_radiation=data['solar_radiation']
    )
    db.session.add(new_data)
    db.session.commit()
    return jsonify({"message": "Data added successfully"}), 201


@app.route('/data_wind', methods=['POST'])
def receive_data_wind():
    data = request.json
    new_data = Data_wind(
        timestamp=datetime.fromisoformat(data['timestamp']),
        voltage=data['voltage'],
        current=data['current'],
        power=data['power'],
        energy_generated=data['energy_generated'],
        wind_speed=data['wind_speed'],
        energy_wh=data['energy_wh']
    )
    db.session.add(new_data)
    db.session.commit()
    return jsonify({"message": "Data added successfully"}), 201


@app.route('/data_batteries', methods=['POST'])
def receive_data_batteries():
    data = request.json
    new_data = Data_batteries(
        timestamp=datetime.fromisoformat(data['timestamp']),
        capacity_solar=data['capacity_solar'],
        capacity_generator=data['capacity_generator'],
        capacity=data['capacity'],
        capacity_ah=data['capacity_ah'],
        soc=data['soc'],
        soh=data['soh'],
        internal_resistance=data['internal_resistance'],
        temperature=data['temperature'],
        charge_efficiency=data['charge_efficiency'],
        discharge_efficiency=data['discharge_efficiency']
    )
    db.session.add(new_data)
    db.session.commit()
    return jsonify({"message": "Data added successfully"}), 201


@app.route('/data_solar', methods=['GET'])
def get_data_solar():
    # Calcular el límite de tiempo para las últimas 24 horas
    last_24_hours = datetime.now() - timedelta(hours=24)
```

```
    # Filtrar los datos que están dentro del rango de las últimas 24
horas
    data = Data_solar.query.filter(Data_solar.timestamp >=
last_24_hours).all()

    result = []
    for d in data:
        result.append({
            'timestamp': d.timestamp.strftime('%d/%m/%Y %H:%M'),
            'voltage': d.voltage,
            'current': d.current,
            'power': d.power,
            'energy_generated': d.energy_generated,
            'energy_wh': d.energy_wh,
            'solar_radiation': d.solar_radiation
        })
    return jsonify(result), 200


@app.route('/data_wind', methods=['GET'])
def get_data_wind():
    # Calcular el límite de tiempo para las últimas 24 horas
    last_24_hours = datetime.now() - timedelta(hours=24)

    # Filtrar los datos que están dentro del rango de las últimas 24
horas
    data = Data_wind.query.filter(Data_wind.timestamp >=
last_24_hours).all()

    result = []
    for d in data:
        result.append({
            'timestamp': d.timestamp.strftime('%d/%m/%Y %H:%M'),
            'voltage': d.voltage,
            'current': d.current,
            'power': d.power,
            'energy_generated': d.energy_generated,
            'wind_speed': d.wind_speed,
            'energy_wh': d.energy_wh
        })
    return jsonify(result), 200


@app.route('/data_batteries', methods=['GET'])
def get_data_batteries():
    # Calcular el límite de tiempo para las últimas 24 horas
    last_24_hours = datetime.now() - timedelta(hours=24)

    # Filtrar los datos que están dentro del rango de las últimas 24
horas
    data = Data_batteries.query.filter(
        Data_batteries.timestamp >= last_24_hours).all()

    result = []
    for d in data:
        result.append({
            'timestamp': d.timestamp.strftime('%d/%m/%Y %H:%M'),
            'capacity_solar': d.capacity_solar,
```

```
            'capacity_generator': d.capacity_generator,
            'capacity': d.capacity,
            'capacity_ah': d.capacity_ah,
            'soc': d.soc,
            'soh': d.soh,
            'internal_resistance': d.internal_resistance,
            'temperature': d.temperature,
            'charge_efficiency': d.charge_efficiency,
            'discharge_efficiency': d.discharge_efficiency
        })
    return jsonify(result), 200


if __name__ == '__main__':
    create_tables()  # Crear tablas antes de iniciar la aplicación
    app.run(host='0.0.0.0')
```

/docker_tfm/frontend/Dockerfile

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["python", "app.py"]
```

/docker_tfm/frontend/requeriments.txt

```
flask
requests
```

/docker_tfm/frontend/app.py

```
from flask import Flask, render_template, jsonify, request, send_file
import requests
import io
import csv
from datetime import datetime

app = Flask(__name__)

SOLAR_API_URL = 'http://backend:5000/data_solar'
WIND_API_URL = 'http://backend:5000/data_wind'
BATTERIES_API_URL = 'http://backend:5000/data_batteries'


@app.route('/')
def index():
```

```
    response_solar = requests.get(SOLAR_API_URL)
    data_solar = response_solar.json()
    response_wind = requests.get(WIND_API_URL)
    data_wind = response_wind.json()
    response_batteries = requests.get(BATTERIES_API_URL)
    data_batteries = response_batteries.json()
    # Combinar todos los datos en un solo diccionario
    data = {
        'solar': data_solar,
        'wind': data_wind,
        'batteries': data_batteries
    }
    return render_template('index.html', data=data)


@app.route('/solar_overview')
def solar():
    response = requests.get(SOLAR_API_URL)
    data_solar = response.json()
    return render_template('solar.html', data=data_solar)


@app.route('/wind_overview')
def wind():
    response = requests.get(WIND_API_URL)
    data_wind = response.json()
    return render_template('wind.html', data=data_wind)


@app.route('/batteries_overview')
def batteries():
    response = requests.get(BATTERIES_API_URL)
    data_batteries = response.json()
    return render_template('batteries.html', data=data_batteries)


@app.route('/download_solar')
def download_data_solar():
    # Obtener las fechas desde el frontend (formato: 'YYYY-MM-
DDTHH:MM')
    start_date = request.args.get('start_date')
    end_date = request.args.get('end_date')

    # Convertir las fechas del frontend a objetos datetime
    if start_date:
        start_date = datetime.strptime(start_date, '%Y-%m-%dT%H:%M')
    if end_date:
        end_date = datetime.strptime(end_date, '%Y-%m-%dT%H:%M')

    # Imprimir para depuración
    print(f"Fecha de inicio en el backend (convertida): {start_date}")
    print(f"Fecha de fin en el backend (convertida): {end_date}")

    # Obtener los datos de la API o tu fuente de datos
    response = requests.get(SOLAR_API_URL)
    data = response.json()

    # Filtrar los datos por rango de fechas
```

```
    filtered_data = []
    for d in data:
        try:
            # Convertir el 'timestamp' de los datos a un objeto
datetime usando el formato '%d/%m/%Y %H:%M'
            data_timestamp = datetime.strptime(
                d['timestamp'], '%d/%m/%Y %H:%M')

            # Imprimir para depuración
            print(
                f"Timestamp de los datos convertido a datetime:
{data_timestamp}")

            # Comparar el timestamp con el rango de fechas (start_date
y end_date)
            if start_date <= data_timestamp <= end_date:
                filtered_data.append(d)
        except ValueError as e:
            # Manejar el caso donde el 'timestamp' no tenga el formato
esperado
            print(
                f"Error al convertir el timestamp '{d['timestamp']}' a
datetime: {e}")

    # Si no hay datos en el rango de fechas, devolver un CSV vacío o
mensaje
    if not filtered_data:
        output = io.StringIO()
        writer = csv.writer(output)
        writer.writerow(['timestamp', 'voltage', 'current',
                         'power', 'energy_generated',
'solar_radiation'])
        output.write(
            "No hay datos disponibles para el rango de fechas
seleccionado.\n")
        output.seek(0)
        return send_file(io.BytesIO(output.read().encode('utf-8')),
                         mimetype='text/csv',
                         download_name='data_solar.csv',
                         as_attachment=True)

    # Generar el CSV con los datos filtrados
    output = io.StringIO()
    writer = csv.writer(output)
    writer.writerow(['timestamp', 'voltage', 'current',
                     'power', 'energy_generated', 'solar_radiation'])
    for row in filtered_data:
        writer.writerow([row['timestamp'], row['voltage'],
                         row['current'], row['power'],
row['energy_generated'], row['solar_radiation']])

    output.seek(0)
    return send_file(io.BytesIO(output.read().encode('utf-8')),
                     mimetype='text/csv',
                     download_name='data_solar.csv',
                     as_attachment=True)
```

```
@app.route('/download_wind')
def download_data_wind():
    # Obtener las fechas desde el frontend (formato: 'YYYY-MM-
DDTHH:MM')
    start_date = request.args.get('start_date')
    end_date = request.args.get('end_date')

    # Convertir las fechas del frontend a objetos datetime
    if start_date:
        start_date = datetime.strptime(start_date, '%Y-%m-%dT%H:%M')
    if end_date:
        end_date = datetime.strptime(end_date, '%Y-%m-%dT%H:%M')

    # Imprimir para depuración
    print(f"Fecha de inicio en el backend (convertida): {start_date}")
    print(f"Fecha de fin en el backend (convertida): {end_date}")

    # Obtener los datos de la API o tu fuente de datos
    response = requests.get(WIND_API_URL)
    data = response.json()

    # Filtrar los datos por rango de fechas
    filtered_data = []
    for d in data:
        try:
            # Convertir el 'timestamp' de los datos a un objeto
datetime usando el formato '%d/%m/%Y %H:%M'
            data_timestamp = datetime.strptime(
                d['timestamp'], '%d/%m/%Y %H:%M')

            # Imprimir para depuración
            print(
                f"Timestamp de los datos convertido a datetime:
{data_timestamp}")

            # Comparar el timestamp con el rango de fechas (start_date
y end_date)
            if start_date <= data_timestamp <= end_date:
                filtered_data.append(d)
        except ValueError as e:
            # Manejar el caso donde el 'timestamp' no tenga el formato
esperado
            print(
                f"Error al convertir el timestamp '{d['timestamp']}' a
datetime: {e}")

    # Si no hay datos en el rango de fechas, devolver un CSV vacío o
mensaje
    if not filtered_data:
        output = io.StringIO()
        writer = csv.writer(output)
        writer.writerow(['timestamp', 'voltage', 'current',
                         'power', 'energy_generated', 'wind_speed',
'energy_wh'])
        output.write(
            "No hay datos disponibles para el rango de fechas
seleccionado.\n")
        output.seek(0)
```

```
        return send_file(io.BytesIO(output.read().encode('utf-8')),
                         mimetype='text/csv',
                         download_name='data_wind.csv',
                         as_attachment=True)

    # Generar el CSV con los datos filtrados
    output = io.StringIO()
    writer = csv.writer(output)
    writer.writerow(['timestamp', 'voltage', 'current',
                     'power', 'energy_generated', 'wind_speed',
'energy_wh'])
    for row in filtered_data:
        writer.writerow([row['timestamp'], row['voltage'],
                         row['current'], row['power'],
row['energy_generated'], row['wind_speed'], row['energy_wh']])

    output.seek(0)
    return send_file(io.BytesIO(output.read().encode('utf-8')),
                     mimetype='text/csv',
                     download_name='data_wind.csv',
                     as_attachment=True)


@app.route('/download_batteries')
def download_data_batteries():
    # Obtener las fechas desde el frontend (formato: 'YYYY-MM-
DDTHH:MM')
    start_date = request.args.get('start_date')
    end_date = request.args.get('end_date')

    # Convertir las fechas del frontend a objetos datetime
    if start_date:
        start_date = datetime.strptime(start_date, '%Y-%m-%dT%H:%M')
    if end_date:
        end_date = datetime.strptime(end_date, '%Y-%m-%dT%H:%M')

    # Imprimir para depuración
    print(f"Fecha de inicio en el backend (convertida): {start_date}")
    print(f"Fecha de fin en el backend (convertida): {end_date}")

    # Obtener los datos de la API o tu fuente de datos
    response = requests.get(BATTERIES_API_URL)
    data = response.json()

    # Filtrar los datos por rango de fechas
    filtered_data = []
    for d in data:
        try:
            # Convertir el 'timestamp' de los datos a un objeto
datetime usando el formato '%d/%m/%Y %H:%M'
            data_timestamp = datetime.strptime(
                d['timestamp'], '%d/%m/%Y %H:%M')

            # Imprimir para depuración
            print(
                f"Timestamp de los datos convertido a datetime:
{data_timestamp}")
```

```
            # Comparar el timestamp con el rango de fechas (start_date
y end_date)
            if start_date <= data_timestamp <= end_date:
                filtered_data.append(d)
        except ValueError as e:
            # Manejar el caso donde el 'timestamp' no tenga el formato
esperado
            print(
                f"Error al convertir el timestamp '{d['timestamp']}' a
datetime: {e}")

    # Si no hay datos en el rango de fechas, devolver un CSV vacío o
mensaje
    if not filtered_data:
        output = io.StringIO()
        writer.writerow(['timestamp', 'capacity_solar',
'capacity_generator',
                         'capacity', 'capacity_ah', 'soc', 'soh',
'internal_resistance', 'temperature', 'charge_efficiency',
'discharge_efficiency'])
        output.write(
            "No hay datos disponibles para el rango de fechas
seleccionado.\n")
        output.seek(0)
        return send_file(io.BytesIO(output.read().encode('utf-8')),
                         mimetype='text/csv',
                         download_name='data_batteries.csv',
                         as_attachment=True)

    output = io.StringIO()
    writer = csv.writer(output)
    writer.writerow(['timestamp', 'capacity_solar',
'capacity_generator',
                     'capacity', 'capacity_ah', 'soc', 'soh',
'internal_resistance', 'temperature', 'charge_efficiency',
'discharge_efficiency'])
    for row in filtered_data:
        writer.writerow([row['timestamp'], row['capacity_solar'],
row['capacity_generator'],
                         row['capacity'], row['capacity_ah'],
row['soc'], row['soh'],
                         row['internal_resistance'],
row['temperature'], row['charge_efficiency'],
row['discharge_efficiency']])

    output.seek(0)
    return send_file(io.BytesIO(output.read().encode('utf-8')),
                     mimetype='text/csv',
                     download_name='data_batteries.csv',
                     as_attachment=True)


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000)
```

/docker_tfm/frontend/templates/index.html

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1, width=device-
width">
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/styles_index.css') }}">
    <link rel="stylesheet"
href="https://fonts.googleapis.com/css2?family=Inter:wght@600;700&disp
lay=swap" />
    <!-- Asegúrate de incluir el script para Chart.js -->
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script type="text/javascript">
        // Recargar la página cada 30 segundos
        setInterval(function () {
            location.reload();
        }, 30000); // 30000 ms = 30 segundos
    </script>
</head>

<body>
    <!-- Depuración: Mostrar el contenido de data
    <pre>{{ data | tojson }}</pre> -->

    <div class="dashboard">
        <div class="container">
            <div class="grupo-1">
                <div class="energia-solar">
                    <b class="titulo">Energía Solar</b>
                    <div class="cuadros-energia">
                        <div class="energia-solar">
                            <div class="div">
                                <div class="estado-grupo">
                                    <div class="titulo1">Estado</div>
                                    <b class="variable">
                                        {% if data['solar'][-
1]['power'] and data['solar'][-1]['power'] > 0 %}
                                        On
                                        {% else %}
                                        Off
                                        {% endif %}
                                    </b>
                                </div>
                                <div class="intensidad-grupo">
                                    <div
class="titulo1">Intensidad</div>
                                    <div class="variable-grupo">
                                        <b class="variable1">{{
data['solar'][-1]['current'] }}</b>
                                        <div class="unidad">A</div>
                                    </div>
                                </div>
```

```
                                    <div class="intensidad-grupo">
                                        <div class="titulo1">Voltaje</div>
                                        <div class="variable-grupo">
                                            <b class="variable1">{{
data['solar'][-1]['voltage'] }}</b>
                                            <div class="unidad">V</div>
                                        </div>
                                    </div>
                                </div>
                                <div class="div1">
                                    <div class="potencia">
                                        <div
class="titulo1">Potencia</div>
                                        <div class="variable-grupo2">
                                            <b class="variable3">{{
data['solar'][-1]['power'] }}</b>
                                            <div class="unidad">kWh</div>
                                        </div>
                                    </div>
                                    <div class="potencia">
                                        <div class="titulo1">Energía
Generada</div>
                                        <div class="variable-grupo2">
                                            <b class="variable3">{{
data['solar'][-1]['energy_generated'] }}</b>
                                            <div class="unidad">Wh</div>
                                        </div>
                                    </div>
                                </div>
                                <div class="div2">
                                    <div class="intensidad-grupo">
                                        <div class="titulo1">Energía
generada por hora</div>
                                        <div class="variable-grupo2">
                                            <b class="variable3">{{
data['solar'][-1]['energy_wh'] }}</b>
                                            <div class="unidad">Wh</div>
                                        </div>
                                    </div>
                                    <div class="tendencia">
                                        <img class="icon" alt="tendencia-
icon" src="{% if data['solar'] | length > 1 %}
                {% set current_energy = data['solar'][-1]['energy_wh']
%}
                {% set previous_energy = data['solar'][-
2]['energy_wh'] %}

                {% if current_energy and previous_energy and
current_energy > previous_energy %}
                    /static/images/tendencia_alc.svg
                {% elif current_energy and previous_energy and
current_energy < previous_energy %}
                    /static/images/tendencia_baj.svg
                {% else %}
                    /static/images/tendencia_neutral.svg
                {% endif %}
            {% else %}
                /static/images/tendencia_neutral.svg
```

```
            {% endif %}">

                                    <div class="turbina-elica">
                                        {% if data['solar'] | length >
1 %}
                                        {% set current_energy =
data['solar'][-1]['energy_wh'] %}
                                        {% set previous_energy =
data['solar'][-13]['energy_wh'] %}

                                        {% if current_energy and
previous_energy %}
                                        {% if current_energy !=
previous_energy %}
                                        {{ ((current_energy -
previous_energy) * 100 / previous_energy) | round(2) }}%
                                        {% else %}
                                        Sin cambios
                                        {% endif %}
                                        {% else %}
                                        No hay suficientes datos
                                        {% endif %}
                                        {% else %}
                                        No hay suficientes datos
                                        {% endif %}
                                    </div>
                                </div>
                            </div>
                        </div>
                        <div class="cuadro-imagen">
                            <img class="placa-imagen-icon" alt=""
                                src="{{ url_for('static',
filename='images/placa.svg') }}">

                        </div>
                    </div>
                </div>
                <div class="energia-solar">
                    <b class="titulo">Energía Eólica</b>
                    <div class="cuadros-energia">
                        <div class="energia-solar">
                            <div class="div">
                                <div class="estado-grupo">
                                    <div class="titulo1">Estado</div>
                                    <b class="variable">
                                        {% if data['solar'][-
1]['power'] and data['solar'][-1]['power'] > 0 %}
                                        On
                                        {% else %}
                                        Off
                                        {% endif %}
                                    </b>
                                </div>
                                <div class="intensidad-grupo">
                                    <div
class="titulo1">Intensidad</div>
                                    <div class="variable-grupo">
```

```
                                        <b class="variable1">{{
data['wind'][-1]['current'] }}</b>
                                        <div class="unidad">A</div>
                                    </div>
                                </div>
                                <div class="intensidad-grupo">
                                    <div class="titulo1">Voltaje</div>
                                    <div class="variable-grupo">
                                        <b class="variable1">{{
data['wind'][-1]['voltage'] }}</b>
                                        <div class="unidad">Wh</div>
                                    </div>
                                </div>
                            </div>
                            <div class="div1">
                                <div class="potencia1">
                                    <div
class="titulo1">Potencia</div>
                                    <div class="variable-grupo2">
                                        <b class="variable3">{{
data['wind'][-1]['power'] }}</b>
                                        <div class="unidad">Wh</div>
                                    </div>
                                </div>
                                <div class="potencia1">
                                    <div class="titulo1">Energía
Generada</div>
                                    <div class="variable-grupo2">
                                        <b class="variable3">{{
data['wind'][-1]['energy_generated'] }}</b>
                                        <div class="unidad">Wh</div>
                                    </div>
                                </div>
                            </div>
                            <div class="div2">
                                <div class="grupo1">
                                    <div class="titulo1">Energía
generada por hora</div>
                                    <div class="variable-grupo2">
                                        <b class="variable3">{{
data['wind'][-1]['energy_wh'] }}</b>
                                        <div class="unidad">Wh</div>
                                    </div>
                                </div>
                                <div class="tendencia">
                                    <img class="icon" alt="tendencia-
icon" src="{% if data['wind'] | length > 1 %}
                {% set current_energy = data['wind'][-1]['energy_wh']
%}
                {% set previous_energy = data['wind'][-2]['energy_wh']
%}

                {% if current_energy and previous_energy and
current_energy > previous_energy %}
                    /static/images/tendencia_alc.svg
                {% elif current_energy and previous_energy and
current_energy < previous_energy %}
                    /static/images/tendencia_baj.svg
```

```
                {% else %}
                    /static/images/tendencia_neutral.svg
                {% endif %}
            {% else %}
                /static/images/tendencia_neutral.svg
            {% endif %}">

                                    <div class="turbina-elica">
                                        {% if data['wind'] | length >
1 %}
                                        {% set current_energy =
data['wind'][-1]['energy_wh'] %}
                                        {% set previous_energy =
data['wind'][-13]['energy_wh'] %}

                                        {% if current_energy and
previous_energy %}
                                        {% if current_energy !=
previous_energy %}
                                        {{ ((current_energy -
previous_energy) * 100 / previous_energy) | round(2) }}%
                                        {% else %}
                                        Sin cambios
                                        {% endif %}
                                        {% else %}
                                        No hay suficientes datos
                                        {% endif %}
                                        {% else %}
                                        No hay suficientes datos
                                        {% endif %}
                                    </div>
                                </div>
                            </div>
                        </div>
                        <div class="cuadro-imagen1">
                            <img class="turbina-imagen-icon" alt=""
                                src="{{ url_for('static',
filename='images/turbina.svg') }}">

                        </div>
                    </div>
                </div>
                <div class="localizacion-tiempo">
                    <div class="localizacion">
                        <b class="titulo">Localización</b>
                        <div class="grupo2">
                            <div class="div6">
                                <div class="turbina-
elica">España</div>
                                <b class="lugar">Valencia</b>
                            </div>
                            <div class="div6">
                                <div class="turbina-elica">Hora
CEST</div>
                                <b class="lugar">{{ data['solar'][-
1]['timestamp'].split(' ')[1][:5] }}</b>
                            </div>
                        </div>
```

```
                    </div>
                    <div class="tiempo">
                        <b class="titulo">Tiempo</b>
                        <div class="grupo3">
                            <div class="div1">
                                <div class="div9">
                                    <div class="variable-
titulo">Lunes</div>
                                    <img class="icon2" alt="" src="{{
url_for('static', filename='images/sun.svg') }}">

                                </div>
                                <div class="div10">
                                    <div class="variable-
titulo">Martes</div>
                                    <img class="icon2" alt="" src="{{
url_for('static', filename='images/sun.svg') }}">

                                </div>
                            </div>
                            <div class="div1">
                                <div class="div10">
                                    <div class="variable-
titulo">Miércoles</div>
                                    <img class="icon4" alt=""
                                        src="{{ url_for('static',
filename='images/sun_cloud.svg') }}">

                                </div>
                                <div class="div9">
                                    <div class="variable-
titulo">Jueves</div>
                                    <img class="icon5" alt=""
                                        src="{{ url_for('static',
filename='images/sun_cloud.svg') }}">

                                </div>
                            </div>
                            <div class="div1">
                                <div class="div9">
                                    <div class="variable-
titulo">Viernes</div>
                                    <img class="icon4" alt="" src="{{
url_for('static', filename='images/rain.svg') }}">

                                </div>
                                <div class="div10">
                                    <div class="variable-
titulo">Sábado</div>
                                    <img class="icon5" alt=""
                                        src="{{ url_for('static',
filename='images/thunder.svg') }}">

                                </div>
                            </div>
                        </div>
                    </div>
                </div>
```

```
            </div>
            <div class="grupo-1">
                <div class="gestor-de-baterias">
                    <b class="titulo">Gestor de Baterías</b>
                    <div class="grupo4">
                        <div class="div17">
                            <div class="div18">
                                <div class="grupo5">
                                    <div class="capacidad-de-carga">
                                        <div class="titulo1">Capacidad
de Carga</div>
                                        <div class="variable-grupo2">
                                            <b class="variable3">{{
data['batteries'][-1]['capacity_solar'] }}</b>
                                            <div
class="unidad">kAh</div>
                                        </div>
                                    </div>
                                    <div class="capacidad-de-carga">
                                        <div
class="titulo1">Eficiencia de Carga</div>
                                        <div class="variable-grupo2">
                                            <b class="variable3">{{
data['batteries'][-1]['charge_efficiency'] }}</b>
                                            <div
class="unidad">%</div>
                                        </div>
                                    </div>
                                </div>
                                <b class="titulo21">Batería 1 (Placa
Solar)</b>
                            </div>
                            <div class="div19">
                                <img class="icon-bateria-1" alt=""
                                    src="{{ url_for('static',
filename='images/bateria_1.svg') }}">

                            </div>
                        </div>
                        <div class="div20">
                            <div class="div21">
                                <div class="grupo6">
                                    <div class="capacidad-de-carga1">
                                        <div class="titulo1">Capacidad
de Carga</div>
                                        <div class="variable-grupo2">
                                            <b class="variable3">{{
data['batteries'][-1]['capacity_generator'] }}</b>
                                            <div
class="unidad">kAh</div>
                                        </div>
                                    </div>
                                    <div class="capacidad-de-carga1">
                                        <div
class="titulo1">Eficiencia de Carga</div>
                                        <div class="variable-grupo2">
                                            <b class="variable3">{{
data['batteries'][-1]['charge_efficiency'] }}</b>
```

```
                                        <div
class="unidad">%</div>
                                    </div>
                                </div>
                                <b class="titulo21">Batería 2 (Turbina
Eólica)</b>
                            </div>
                            <div class="div19">
                                <img class="icon-bateria-2" alt=""
                                    src="{{ url_for('static',
filename='images/bateria_2.svg') }}">

                            </div>
                        </div>
                    </div>
                </div>
                <div class="energia-generada-total">
                    <b class="titulo">Energía Generada Total</b>
                    <div class="container1">
                        <div class="gestor-de-baterias">
                            <div class="fecha">
                                {% if data['solar'][-1]['timestamp']
is not none %}
                                <!-- Dividir el timestamp por espacio
para obtener solo la fecha -->
                                {% set date_part = data['solar'][-
1]['timestamp'].split(' ')[0] %}

                                <b class="lugar">{{ date_part }}</b>
                                {% else %}
                                <b class="lugar">Sin fecha
disponible</b>
                                {% endif %}
                            </div>
                            <div class="minuto">
                                <div class="titulo1">Por minuto</div>
                                <div class="variable-grupo14">
                                    <b class="variable3">{{
((data['solar'][-1]['energy_generated'] +
                                        data['wind'][-
1]['energy_generated']) / 5) | round(2) }}</b>
                                    <div class="unidad14">Wh</div>
                                </div>
                            </div>
                            <div class="minuto">
                                <div class="titulo1">Por hora</div>
                                <div class="variable-grupo14">
                                    <b class="variable3">{{
((data['solar'][-1]['energy_wh'] +
                                        data['wind'][-
1]['energy_wh'])/1000) | round(2) }}</b>
                                    <div class="unidad14">kWh</div>
                                </div>
                            </div>
                            <div class="minuto">
                                <div class="titulo1">Por día</div>
                                <div class="variable-grupo14">
```

```
                                    <b class="variable3">
                                        <div class="variable-grupo14">
                                            <b class="variable3">
                                                {% set
total_energy_solar = 0 %}
                                                {% set
total_energy_wind = 0 %}

                                                <!-- Obtener el tamaño
de los últimos 288 elementos o menos, dependiendo de cuántos haya -->
                                                {% set solar_count =
data['solar']|length if data['solar']|length < 288
                                                    else 288 %} {% set
wind_count=data['wind']|length if

data['wind']|length < 288 else 288 %} <!-- Iterar sobre los últimos
                                                    elementos
disponibles de 'solar' -->
                                                {% for item in
data['solar'][-solar_count:] %}

                                                {% set
total_energy_solar = total_energy_solar +

(item['energy_generated']
                                                if
item['energy_generated'] is not none else 0) %}
                                                {% endfor %}

                                                <!-- Iterar sobre
los últimos elementos disponibles de 'wind' -->
                                                {% for item in
data['wind'][-wind_count:] %}

                                                {% set
total_energy_wind = total_energy_wind +

(item['energy_generated'] if

item['energy_generated'] is not none else 0) %}
                                                {% endfor %}

                                                <!-- Mostrar la
suma total redondeada a 2 decimales, dividiendo entre 1000 -->
                                                {{
(total_energy_solar + total_energy_wind)/1000 | round(2) }}

                                            </b>
                                        </div>
                                    </b>
                                    <div class="unidad14">kWh</div>
                                </div>
                            </div>
                        </div>
                        <div class="grafico">
                            <!-- Lugar donde aparecerá el gráfico
generado con Chart.js -->
```

```
                            <canvas id="myChart" style="width:100%;
height:100%;"></canvas>

                        </div>
                    </div>
                </div>
            </div>
        </div>
        <div class="header">
            <img class="logo-icon" alt="" src="{{ url_for('static',
filename='images/logo.svg') }}">

            <div class="nav-botones">
                <div class="general">
                    <div class="turbina-elica">General</div>
                </div>
                <div class="placa-solar" id="placaSolarContainer">
                    <div class="turbina-elica">Placa Solar</div>
                </div>
                <div class="placa-solar" id="turbinaEolicaContainer">
                    <div class="turbina-elica">Turbina Eólica</div>
                </div>
                <div class="placa-solar" id="gestorDeBaterias">
                    <div class="turbina-elica">Gestor de
Baterías</div>
                </div>
            </div>
            <div class="circulos">
                <div class="help">
                    <img class="help-icon" alt="" src="{{
url_for('static', filename='images/help.svg') }}">

                </div>
                <div class="help">
                    <img class="alarm-icon" alt="" src="{{
url_for('static', filename='images/alarm.svg') }}">

                </div>
                <div class="account">
                    <b class="kz">KZ</b>
                </div>
            </div>
        </div>
    </div>

    <script>
        // Asignar los datos de energía solar, eólica y sus timestamps
        const data_solar = {{ data['solar'][-12:] | tojson }};
        const data_wind = {{ data['wind'][-12:] | tojson }};

        // Extraer los timestamps del array de datos de solar
(asumiendo que ambos tienen el mismo número de elementos)
        const timestamps = data_solar.map(d => {
            const date = new Date(d.timestamp);
            const day = String(date.getDate()).padStart(2, '0');
            const month = String(date.getMonth() + 1).padStart(2,
'0');
            const hours = String(date.getHours()).padStart(2, '0');
```

```
            const minutes = String(date.getMinutes()).padStart(2,
'0');
            return `${day}/${month} ${hours}:${minutes}`;
        });

        // Extraer la energía de solar y wind para la gráfica
        const solarData = data_solar.map(d => d.energy_wh || 0);  //
Si energy_wh es null, usar 0
        const windData = data_wind.map(d => d.energy_wh || 0);    //
Lo mismo para wind

        // Calcular la suma de solar y wind
        const totalData = solarData.map((val, i) => val +
windData[i]);

        // Calcular el valor máximo de los datos más 200
        const maxEnergy = Math.max(...solarData, ...windData,
...totalData) + 50;

        // Configuración para la gráfica con Chart.js
        var ctx = document.getElementById('myChart').getContext('2d');

        new Chart(ctx, {
            type: 'line',
            data: {
                labels: timestamps,  // Los timestamps en formato
DD/MM hh:mm
                datasets: [{
                    label: 'Energía Solar',
                    data: solarData,
                    borderColor: '#CFA953', // Color de la línea de
Solar
                    backgroundColor: '#CFA953', // Relleno
transparente
                }, {
                    data: windData,
                    label: 'Energía Eólica',
                    borderColor: '#537ACF', // Color de la línea de
Wind
                    backgroundColor: '#537ACF',
                }, {
                    data: totalData,
                    label: 'Energía Total',
                    borderColor: '#549F58', // Color de la línea de
Total
                    backgroundColor: '#549F58',
                }]
            },
            options: {
                scales: {
                    y: {
                        beginAtZero: true,
                        max: maxEnergy,  // Limitar el eje Y al máximo
de los datos + 200
                        title: {
                            display: true,
                            text: 'Energía (Wh)',  // Etiqueta del eje
Y
```

```
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        },
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    },

                    x: {
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }

                },
                plugins: {
                    legend: {
                        display: true, // Desactivar la leyenda
                        position: 'bottom', // Ubicar la leyenda en la
parte inferior
                        labels: {
                            usePointStyle: true,  // Utilizar círculos
en lugar de rectángulos
                            pointStyle: 'circle', // Definir el estilo
de punto como círculo
                            padding: 24,
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }
                }
            }
        });
    </script>


    <script>
        var placaSolarContainer =
document.getElementById("placaSolarContainer");
        if (placaSolarContainer) {
            placaSolarContainer.addEventListener("click", function (e)
{
```

```
                window.location.href = "/solar_overview";  // Redirige
a solar.html
            });
        }

        var turbinaEolicaContainer =
document.getElementById("turbinaEolicaContainer");
        if (turbinaEolicaContainer) {
            turbinaEolicaContainer.addEventListener("click", function
(e) {
                window.location.href = "/wind_overview";  // Redirige
a wind.html
            });
        }

        var gestorDeBaterias =
document.getElementById("gestorDeBaterias");
        if (gestorDeBaterias) {
            gestorDeBaterias.addEventListener("click", function (e) {
                window.location.href = "/batteries_overview";  //
Redirige a batteries.html
            });
        }
    </script>

</body>

</html>
```

/docker_tfm/frontend/templates/solar.html

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1, width=device-
width">
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/styles_solar.css') }}">
    <link rel="stylesheet"

href="https://fonts.googleapis.com/css2?family=Inter:ital,wght@0,600;0
,700;1,500&display=swap" />
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script type="text/javascript">
        // Recargar la página cada 30 segundos
        setInterval(function () {
            location.reload();
        }, 300000); // 300000 ms = 300 segundos
    </script>
</head>

<body>
```

```
    <div class="placa-solar">
        <div class="container">
            <div class="cuadros">
                <div class="grupo">
                    <div class="placa-solar1">
                        <b class="titulo">Placa Solar</b>
                        <div class="container1">
                            <div class="div">
                                <div class="grupo1">
                                    <div class="titulo1">Energía
generada</div>
                                    <div class="variable-grupo">
                                        <b class="numero">{{ data[-
1]['energy_wh'] }}</b>
                                        <div class="unidad">Wh</div>
                                    </div>
                                </div>
                                <div class="tendencia">
                                    <img class="icon" alt="tendencia-
icon" src="{% if data| length > 1 %}
                {% set current_energy = data[-1]['energy_wh'] %}
                {% set previous_energy = data[-2]['energy_wh'] %}

                {% if current_energy and previous_energy and
current_energy > previous_energy %}
                    /static/images/tendencia_alc.svg
                {% elif current_energy and previous_energy and
current_energy < previous_energy %}
                    /static/images/tendencia_baj.svg
                {% else %}
                    /static/images/tendencia_neutral.svg
                {% endif %}
            {% else %}
                /static/images/tendencia_neutral.svg
            {% endif %}">


                                <div class="variable">{% if data |
length > 1 %}
                                    {% set current_energy = data[-
1]['energy_wh'] %}
                                    {% set previous_energy =
data[-13]['energy_wh'] %}

                                    {% if current_energy and
previous_energy %}
                                    {% if current_energy !=
previous_energy %}
                                    {{ ((current_energy -
previous_energy) * 100 / previous_energy) | round(2) }}%
                                    {% else %}
                                    Sin cambios
                                    {% endif %}
                                    {% else %}
                                    No hay suficientes datos
                                    {% endif %}
                                    {% else %}
```

```
                                        No hay suficientes datos
                                        {% endif %}
                                    </div>
                                </div>
                            </div>
                            <div class="div1">
                                <div class="potencia">
                                    <div
class="titulo1">Potencia</div>
                                    <div class="variable-grupo">
                                        <b class="numero">{{ data[-
1]['power'] }}</b>
                                        <div class="unidad">kWh</div>
                                    </div>
                                </div>
                                <div class="potencia">
                                    <div class="titulo1">Voltaje</div>
                                    <div class="variable-grupo">
                                        <b class="numero">{{ data[-
1]['voltage'] }}</b>
                                        <div class="unidad">V</div>
                                    </div>
                                </div>
                            </div>
                            <div class="div1">
                                <div class="intensidad">
                                    <div
class="titulo1">Intensidad</div>
                                    <div class="variable-grupo">
                                        <b class="numero">{{ data[-
1]['current'] }}</b>
                                        <div class="unidad">A</div>
                                    </div>
                                </div>
                                <div class="intensidad">
                                    <div class="titulo1">Radiación
Solar</div>
                                    <div class="variable-grupo">
                                        <b class="numero">{{ data[-
1]['solar_radiation'] }}</b>
                                        <div class="unidad">W/m²</div>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                    <div class="especificaciones">
                        <b class="titulo">Especificaciones</b>
                        <div class="container2">
                            <div class="div3">
                                <div class="intensidad">
                                    <div class="titulo1">Voltaje
Máximo</div>
                                    <div class="variable-grupo">
                                        <b class="numero" data-
variable="voltage_max"></b>
                                        <div class="unidad">V</div>
                                    </div>
```

```
                                </div>
                                <div class="intensidad">
                                    <div class="titulo1">Intensidad
Máxima</div>
                                    <div class="variable-grupo">
                                        <b class="numero" data-
variable="current_max"></b>
                                        <div class="unidad">A</div>
                                    </div>
                                </div>
                            </div>
                            <div class="div4">
                                <div class="container-escritura">
                                    <div class="voltaje-max1">
                                        <div class="titulo1">Voltaje
Máximo</div>
                                        <div class="variable-grupo7">
                                            <div class="variable1">
                                                <input class="numero7"
data-input="voltage_max_input">
                                            </div>
                                            <div
class="unidad7">V</div>
                                        </div>
                                    </div>
                                    <div class="voltaje-max1">
                                        <div
class="titulo1">Intensidad Máxima</div>
                                        <div class="variable-grupo7">
                                            <div class="variable2">
                                                <input class="numero8"
data-input="current_max_input">
                                            </div>
                                            <div
class="unidad7">A</div>
                                        </div>
                                    </div>
                                </div>
                                <div class="container-2">
                                    <b class="titulo11">Escritura</b>
                                    <div class="boton">
                                        <div class="variable"><button
type="submit">Actualizar</button></div>
                                        <img class="icon1" alt=""
                                            src="{{ url_for('static',
filename='images/update.svg') }}">

                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
                <div class="imagen-fecha">
                    <div class="div5">
                        <img class="esquema-solar-1" alt=""
                            src="{{ url_for('static',
filename='images/esquema_solar.svg') }}">
```

```
                    </div>
                    <div class="div6">
                        <div class="container3">
                            <div class="fecha-inicio">
                                <b class="titulo13">Fecha inicio</b>
                                <div class="selector">
                                    <input type="datetime-local"
id="start_date" name="start_date" required>
                                </div>
                            </div>
                            <div class="fecha-inicio">
                                <b class="titulo14">Fecha fin</b>
                                <div class="selector">
                                    <input type="datetime-local"
id="end_date" name="end_date" required>
                                </div>
                            </div>
                        </div>
                        <div class="boton1">
                            <form action="/download_solar"
method="get" id="download-form">
                                <!-- Inputs ocultos que recibirán los
valores de las fechas seleccionadas -->
                                <input type="hidden"
id="hidden_start_date" name="start_date">
                                <input type="hidden"
id="hidden_end_date" name="end_date">
                                <button type="submit" class="styled-
button">Descargar CSV</button>
                            </form>
                        </div>
                    </div>
                </div>
            </div>
            <div class="graficos">
                <div class="div7">
                    <div class="div8">
                        <b class="titulo">Potencia</b>
                        <div class="grafico-potencia">
                            <canvas id="chart_power"
style="width:100%; height:100%;"></canvas>
                        </div>
                    </div>
                    <div class="div8">
                        <b class="titulo">Energía</b>
                        <div class="grafico-potencia">
                            <canvas id="chart_energy"
style="width:100%; height:100%;"></canvas>
                        </div>
                    </div>
                </div>
                <div class="div7">
                    <div class="div8">
                        <b class="titulo">Voltaje</b>
                        <div class="grafico-potencia">
                            <canvas id="chart_voltage"
style="width:100%; height:100%;"></canvas>
```

```
                        </div>
                    </div>
                    <div class="div8">
                        <b class="titulo">Intensidad</b>
                        <div class="grafico-potencia">
                            <canvas id="chart_current"
style="width:100%; height:100%;"></canvas>
                        </div>
                    </div>
                </div>
            </div>
        </div>
        <div class="header">
            <img class="logo-icon" alt="" src="{{ url_for('static',
filename='images/logo.svg') }}">

            <div class="nav-botones">
                <div class="general" id="generalContainer">
                    <div class="variable">General</div>
                </div>
                <div class="placa-solar2">
                    <div class="variable">Placa Solar</div>
                </div>
                <div class="general" id="turbinaEolicaContainer">
                    <div class="variable">Turbina Eólica</div>
                </div>
                <div class="general" id="gestorDeBaterias">
                    <div class="variable">Gestor de Baterías</div>
                </div>
            </div>
            <div class="circulos">
                <div class="help">
                    <img class="help-icon" alt="" src="{{
url_for('static', filename='images/help.svg') }}">

                </div>
                <div class="help">
                    <img class="alarm-icon" alt="" src="{{
url_for('static', filename='images/alarm.svg') }}">

                </div>
                <div class="account">
                    <b class="kz">KZ</b>
                </div>
            </div>
        </div>
    </div>

    <script>
        var generalContainer =
document.getElementById("generalContainer");
        if (generalContainer) {
            generalContainer.addEventListener("click", function (e) {
                window.location.href = "/";  // Redirige a wind.html
            });
        }
```

```
        var turbinaEolicaContainer =
document.getElementById("turbinaEolicaContainer");
        if (turbinaEolicaContainer) {
            turbinaEolicaContainer.addEventListener("click", function
(e) {
                window.location.href = "/wind_overview";  // Redirige
a wind.html
            });
        }

        var gestorDeBaterias =
document.getElementById("gestorDeBaterias");
        if (gestorDeBaterias) {
            gestorDeBaterias.addEventListener("click", function (e) {
                window.location.href = "/batteries_overview";  //
Redirige a batteries.html
            });
        }
    </script>

    <script>
        // Obtener los límites actuales y mostrarlos
        function getLimits() {
            fetch('http://localhost:5100/limits')
                .then(response => response.json())
                .then(data => {
                    // Seleccionamos todos los elementos con la clase
'numero'
                    const elementosNumero =
document.querySelectorAll('.numero');

                    // Iteramos sobre cada elemento y asignamos el
valor correspondiente basado en 'data-variable'
                    elementosNumero.forEach(element => {
                        const variable = element.getAttribute('data-
variable'); // Obtener el valor del atributo 'data-variable'

                        if (variable === 'voltage_max') {
                            element.textContent = data.voltage_max; //
Asignar voltage_max
                        } else if (variable === 'current_max') {
                            element.textContent = data.current_max; //
Asignar current_max
                        }
                    });

                    // Asignar valores a los elementos de escritura
(para los inputs)
                    document.querySelector('.numero7').textContent =
data.voltage_max;
                    document.querySelector('.numero8').textContent =
data.current_max;
                })
                .catch(error => console.error('Error fetching
limits:', error));
        }

        // Manejar la actualización de los límites
```

```
        document.querySelector('.boton
button').addEventListener('click', function (event) {
            event.preventDefault();
            const limitsData = {};

            // Obtener los valores actuales de los elementos 'numero7'
y 'kz'
            const voltageMax =
document.querySelector('.numero7').value;
            const currentMax =
document.querySelector('.numero8').value;

            if (voltageMax) {
                limitsData.voltage_max = parseFloat(voltageMax);
            }

            if (currentMax) {
                limitsData.current_max = parseFloat(currentMax);
            }
            console.log("Enviando los siguientes datos:", limitsData);
// Añadir esto para verificar los datos

            fetch('http://localhost:5100/limits', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(limitsData)
            })
                .then(response => response.json())
                .then(data => {
                    alert(data.message);
                    getLimits();  // Actualizar los límites mostrados

                    // Borrar los valores de los inputs después de
actualizar
                    document.querySelector('.numero7').value = ""; //
Limpiar el input de voltage_max
                    document.querySelector('.numero8').value = ""; //
Limpiar el input de current_max
                })
                .catch(error => console.error('Error actualizando los
límites:', error));
        });

        // Obtener los límites al cargar la página
        getLimits();
    </script>

    <script>
        // Agregar event listener para el campo de fecha de inicio

document.getElementById('start_date').addEventListener('change',
function () {
            const startDate = this.value;
            console.log("Fecha de inicio seleccionada:", startDate);
        });
```

```
        // Agregar event listener para el campo de fecha de fin
        document.getElementById('end_date').addEventListener('change',
function () {
            const endDate = this.value;
            console.log("Fecha de fin seleccionada:", endDate);
        });

        // Ver las fechas en el formulario cuando se envía
        document.getElementById('download-
form').addEventListener('submit', function (event) {
            const startDate =
document.getElementById('start_date').value;
            const endDate = document.getElementById('end_date').value;

            // Asignar los valores a los inputs ocultos
            document.getElementById('hidden_start_date').value =
startDate;
            document.getElementById('hidden_end_date').value =
endDate;
            // Validar que las fechas no estén vacías
            if (!startDate || !endDate) {
                alert('Por favor, selecciona ambas fechas antes de
descargar el CSV.');
                event.preventDefault();  // Evitar que se envíe el
formulario si están vacías
            }

            // Mostrar las fechas seleccionadas en la consola justo
antes de enviar el formulario
            console.log("Fecha de inicio en el formulario:",
startDate);
            console.log("Fecha de fin en el formulario:", endDate);

        });
    </script>


    <script>
        // Asignar los datos de energía solar, eólica y sus timestamps
        const data_solar = {{ data[-12:] | tojson }};

        // Extraer los timestamps del array de datos de solar
(asumiendo que ambos tienen el mismo número de elementos)
        const timestamps = data_solar.map(d => {
            const date = new Date(d.timestamp);
            const day = String(date.getDate()).padStart(2, '0');
            const month = String(date.getMonth() + 1).padStart(2,
'0');
            const hours = String(date.getHours()).padStart(2, '0');
            const minutes = String(date.getMinutes()).padStart(2,
'0');
            return `${day}/${month} ${hours}:${minutes}`;
        });

        // Extraer la energía de solar y wind para la gráfica
        const powerData = data_solar.map(d => d.power || 0);  // Si
energy_wh es null, usar 0
```

```
        const energyData = data_solar.map(d => d.energy_generated ||
0);    // Lo mismo para wind
        const voltageData = data_solar.map(d => d.voltage || 0);  //
Si energy_wh es null, usar 0
        const currentData = data_solar.map(d => d.current || 0);    //
Lo mismo para wind

        // Calcular el valor máximo de los datos más 200
        const maxPower = Math.max(...powerData) + 0.2;
        const maxEnergy = Math.max(...energyData) + 5;
        const maxVoltage = Math.max(...voltageData) + 15;
        const maxCurrent = Math.max(...currentData) + 0.5;

        // Configuración para la gráfica Potencia
        var power_ctx =
document.getElementById('chart_power').getContext('2d');
        new Chart(power_ctx, {
            type: 'line',
            data: {
                labels: timestamps,  // Los timestamps en formato
DD/MM hh:mm
                datasets: [{
                    label: 'Potencia',
                    data: powerData,
                    borderColor: '#CFA953', // Color de la línea de
Potencia
                    backgroundColor: '#CFA953', // Relleno
transparente
                }]
            },
            options: {
                scales: {
                    y: {
                        beginAtZero: true,
                        max: maxPower,  // Limitar el eje Y al máximo
de los datos + 200
                        title: {
                            display: true,
                            text: 'Potencia (kWh)',  // Etiqueta del
eje Y
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        },
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    },

                    x: {
                        ticks: {
                            color: '#292F32',
```

```
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }

                },
                plugins: {
                    legend: {
                        display: true, // Desactivar la leyenda
                        position: 'bottom', // Ubicar la leyenda en la
parte inferior
                        labels: {
                            usePointStyle: true,  // Utilizar círculos
en lugar de rectángulos
                            pointStyle: 'circle', // Definir el estilo
de punto como círculo
                            padding: 24,
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }
                }
            }
        });

        // Configuración para la gráfica Energía
        var energy_ctx =
document.getElementById('chart_energy').getContext('2d');
        new Chart(energy_ctx, {
            type: 'line',
            data: {
                labels: timestamps,  // Los timestamps en formato
DD/MM hh:mm
                datasets: [{
                    label: 'Energía',
                    data: energyData,
                    borderColor: '#549F58', // Color de la línea de
Energia
                    backgroundColor: '#549F58', // Relleno
transparente
                }]
            },
            options: {
                scales: {
                    y: {
                        beginAtZero: true,
                        max: maxEnergy,  // Limitar el eje Y al máximo
de los datos + 200
                        title: {
                            display: true,
                            text: 'Energía (Wh)',  // Etiqueta del eje
Y
                            color: '#292F32',
```

```
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        },
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    },

                    x: {
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }

                },
                plugins: {
                    legend: {
                        display: true, // Desactivar la leyenda
                        position: 'bottom', // Ubicar la leyenda en la
parte inferior
                        labels: {
                            usePointStyle: true,  // Utilizar círculos
en lugar de rectángulos
                            pointStyle: 'circle', // Definir el estilo
de punto como círculo
                            padding: 24,
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }
                }
            }
        });

        // Configuración para la gráfica Energía
        var voltage_ctx =
document.getElementById('chart_voltage').getContext('2d');
        new Chart(voltage_ctx, {
            type: 'line',
            data: {
                labels: timestamps,  // Los timestamps en formato
DD/MM hh:mm
                datasets: [{
                    label: 'Voltaje',
                    data: voltageData,
```

```
                    borderColor: '#537ACF', // Color de la línea de
Voltaje
                    backgroundColor: '#537ACF', // Relleno
transparente
                }]
            },
            options: {
                scales: {
                    y: {
                        beginAtZero: true,
                        max: maxVoltage,  // Limitar el eje Y al
máximo de los datos + 200
                        title: {
                            display: true,
                            text: 'Voltaje (V)',  // Etiqueta del eje
Y
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        },
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    },

                    x: {
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }

                },
                plugins: {
                    legend: {
                        display: true, // Desactivar la leyenda
                        position: 'bottom', // Ubicar la leyenda en la
parte inferior
                        labels: {
                            usePointStyle: true,  // Utilizar círculos
en lugar de rectángulos
                            pointStyle: 'circle', // Definir el estilo
de punto como círculo
                            padding: 24,
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
```

```
                        }
                    }
                }
            }
        });

        // Configuración para la gráfica Intensidad
        var current_ctx =
document.getElementById('chart_current').getContext('2d');
        new Chart(current_ctx, {
            type: 'line',
            data: {
                labels: timestamps,  // Los timestamps en formato
DD/MM hh:mm
                datasets: [{
                    label: 'Intensidad',
                    data: currentData,
                    borderColor: '#CF5389', // Color de la línea de
Intensidad
                    backgroundColor: '#CF5389', // Relleno
transparente
                }]
            },
            options: {
                scales: {
                    y: {
                        beginAtZero: true,
                        max: maxCurrent,  // Limitar el eje Y al
máximo de los datos + 200
                        title: {
                            display: true,
                            text: 'Intensidad (A)',  // Etiqueta del
eje Y
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        },
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    },

                    x: {
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }
```

```
                },
                plugins: {
                    legend: {
                        display: true, // Desactivar la leyenda
                        position: 'bottom', // Ubicar la leyenda en la
parte inferior
                        labels: {
                            usePointStyle: true,  // Utilizar círculos
en lugar de rectángulos
                            pointStyle: 'circle', // Definir el estilo
de punto como círculo
                            padding: 24,
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }
                }
            }
        });
    </script>

</body>

</html>
```

/docker_tfm/frontend/templates/wind.html

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1, width=device-
width">
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/styles_wind.css') }}">
    <link rel="stylesheet"

href="https://fonts.googleapis.com/css2?family=Inter:ital,wght@0,600;0
,700;1,500&display=swap" />
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script type="text/javascript">
        // Recargar la página cada 30 segundos
        setInterval(function () {
            location.reload();
        }, 300000); // 300000 ms = 300 segundos
    </script>
</head>

<body>
```

```
    <div class="turbina-eolica">
        <div class="container">
            <div class="cuadros">
                <div class="turbina-eolica-fecha">
                    <div class="turbina-eolica1">
                        <b class="titulo">Turbina Eólica</b>
                        <div class="container1">
                            <div class="div">
                                <div class="grupo">
                                    <div class="titulo1">Energía
generada</div>
                                    <div class="variable-grupo">
                                        <b class="variable">{{ data[-
1]['energy_wh'] }}</b>
                                        <div class="unidad">Wh</div>
                                    </div>
                                </div>
                                <div class="tendencia">
                                    <img class="icon" alt="tendencia-
icon" src="{% if data | length > 1 %}
                {% set current_energy = data[-1]['energy_wh'] %}
                {% set previous_energy = data[-2]['energy_wh'] %}

                {% if current_energy and previous_energy and
current_energy > previous_energy %}
                    /static/images/tendencia_alc.svg
                {% elif current_energy and previous_energy and
current_energy < previous_energy %}
                    /static/images/tendencia_baj.svg
                {% else %}
                    /static/images/tendencia_neutral.svg
                {% endif %}
            {% else %}
                /static/images/tendencia_neutral.svg
            {% endif %}">


                                    <div class="controlador-e-
inversor">{% if data | length > 1 %}
                                        {% set current_energy = data[-
1]['energy_wh'] %}
                                        {% set previous_energy =
data[-13]['energy_wh'] %}

                                        {% if current_energy and
previous_energy %}
                                        {% if current_energy !=
previous_energy %}
                                        {{ ((current_energy -
previous_energy) * 100 / previous_energy) | round(2) }}%
                                        {% else %}
                                        Sin cambios
                                        {% endif %}
                                        {% else %}
                                        No hay suficientes datos
                                        {% endif %}
                                        {% else %}
                                        No hay suficientes datos
```

```
                                        {% endif %}</div>
                                </div>
                            </div>
                            <div class="div1">
                                <div class="potencia">
                                    <div
class="titulo2">Potencia</div>
                                    <div class="variable-grupo1">
                                        <b class="variable">{{ data[-
1]['power'] }}</b>
                                        <div class="unidad">Wh</div>
                                    </div>
                                </div>
                                <div class="potencia">
                                    <div class="titulo3">Voltaje</div>
                                    <div class="variable-grupo2">
                                        <b class="variable">{{ data[-
1]['voltage'] }}</b>
                                        <div class="unidad">V</div>
                                    </div>
                                </div>
                            </div>
                            <div class="div1">
                                <div class="potencia">
                                    <div
class="titulo4">Intensidad</div>
                                    <div class="variable-grupo3">
                                        <b class="variable">{{ data[-
1]['current'] }}</b>
                                        <div class="unidad">A</div>
                                    </div>
                                </div>
                                <div class="potencia">
                                    <div class="titulo5">Viento</div>
                                    <div class="variable-grupo4">
                                        <b class="variable">{{ data[-
1]['wind_speed'] }}</b>
                                        <div class="unidad">m/s</div>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                    <div class="fecha">
                        <div class="container2">
                            <div class="fecha-inicio">
                                <b class="titulo6">Fecha inicio</b>
                                <div class="selector">
                                    <input type="datetime-local"
id="start_date" name="start_date" required>
                                </div>
                            </div>
                            <div class="fecha-inicio">
                                <b class="titulo7">Fecha fin</b>
                                <div class="selector">
                                    <input type="datetime-local"
id="end_date" name="end_date" required>
                                </div>
```

```
                            </div>
                        </div>
                        <div class="boton">
                            <form action="/download_wind" method="get"
id="download-form">
                                <!-- Inputs ocultos que recibirán los
valores de las fechas seleccionadas -->
                                <input type="hidden"
id="hidden_start_date" name="start_date">
                                <input type="hidden"
id="hidden_end_date" name="end_date">
                                <button type="submit" class="styled-
button">Descargar CSV</button>
                            </form>
                        </div>
                    </div>
                </div>
                <div class="imagen-especificaciones">
                    <div class="cuadro-imagen">
                        <img class="esquema-turbina-1" alt=""
                            src="{{ url_for('static',
filename='images/esquema_turbina.svg') }}">
                    </div>
                    <div class="container3">
                        <b class="titulo9">Especificaciones</b>
                        <div class="grupo1">
                            <div class="potencia">
                                <div class="titulo10">Voltaje
Máximo</div>
                                <div class="variable-grupo">
                                    <b class="variable" data-
variable="voltage_max"></b>
                                    <div class="unidad">V</div>
                                </div>
                            </div>
                            <div class="div4">
                                <div class="titulo1">Voltaje
Mínimo</div>
                                <div class="variable-grupo">
                                    <b class="variable" data-
variable="voltage_min"></b>
                                    <div class="unidad">V</div>
                                </div>
                            </div>
                            <div class="div4">
                                <div class="titulo1">Intensidad
Máxima</div>
                                <div class="variable-grupo">
                                    <b class="variable" data-
variable="current_max"></b>
                                    <div class="unidad">A</div>
                                </div>
                            </div>
                            <div class="div6">
                                <div class="titulo1">Velocidad de
Corte</div>
                                <div class="variable-grupo">
```

```
                                    <b class="variable" data-
variable="v_cut_in"></b>
                                    <div class="unidad">m/s</div>
                                </div>
                            </div>
                            <div class="div6">
                                <div class="titulo1">Velocidad
Nominal</div>
                                <div class="variable-grupo">
                                    <b class="variable" data-
variable="v_nominal"></b>
                                    <div class="unidad">m/s</div>
                                </div>
                            </div>
                            <div class="div6">
                                <div class="titulo1">Velocidad de
Parada</div>
                                <div class="variable-grupo">
                                    <b class="variable" data-
variable="v_cut_out"></b>
                                    <div class="unidad">m/s</div>
                                </div>
                            </div>
                        </div>
                        <div class="escritura">
                            <div class="grupo1">
                                <div class="div9">
                                    <div class="titulo1">Voltaje
Máximo</div>
                                    <div class="variable-grupo12">
                                        <div class="variable11">
                                            <input class="numero7"
data-input="voltage_max_input">
                                        </div>
                                        <div class="unidad11">V</div>
                                    </div>
                                </div>
                                <div class="div9">
                                    <div class="titulo1">Voltaje
Mínimo</div>
                                    <div class="variable-grupo12">
                                        <div class="variable12">
                                            <input class="numero8"
data-input="voltage_min_input">
                                        </div>
                                        <div class="unidad11">V</div>
                                    </div>
                                </div>
                                <div class="div9">
                                    <div class="titulo1">Intensidad
Máxima</div>
                                    <div class="variable-grupo12">
                                        <div class="variable12">
                                            <input class="numero9"
data-input="current_max_input">
                                        </div>
                                        <div class="unidad11">A</div>
                                    </div>
```

```
                                </div>
                                <div class="div12">
                                    <div class="titulo1">Velocidad de
Corte</div>
                                    <div class="variable-grupo12">
                                        <div class="variable12">
                                            <input class="numero10"
data-input="vc_max_input">
                                        </div>
                                        <div
class="unidad11">m/s</div>
                                    </div>
                                </div>
                                <div class="div12">
                                    <div class="titulo1">Velocidad
Nominal</div>
                                    <div class="variable-grupo12">
                                        <div class="variable12">
                                            <input class="numero11"
data-input="vn_max_input">
                                        </div>
                                        <div
class="unidad11">m/s</div>
                                    </div>
                                </div>
                                <div class="div12">
                                    <div class="titulo1">Velocidad de
Parada</div>
                                    <div class="variable-grupo1">
                                        <div class="variable12">
                                            <input class="numero12"
data-input="vp_max_input">
                                        </div>
                                        <div
class="unidad11">m/s</div>
                                    </div>
                                </div>
                            </div>
                            <div class="grupo2">
                                <b class="titulo22">Escritura</b>
                                <div class="boton1">
                                    <div class="controlador-e-
inversor"><button type="submit">Actualizar</button></div>
                                    <img class="icon4" alt=""
                                        src="{{ url_for('static',
filename='images/update.svg') }}">
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
            <div class="graficos">
                <div class="div15">
                    <div class="div16">
                        <b class="titulo">Potencia</b>
                        <div class="grafico">
```

```
                            <canvas id="chart_power"
style="width:100%; height:100%;"></canvas>
                        </div>
                    </div>
                    <div class="div16">
                        <b class="titulo">Energía</b>
                        <div class="grafico">
                            <canvas id="chart_energy"
style="width:100%; height:100%;"></canvas>
                        </div>
                    </div>
                </div>
                <div class="div15">
                    <div class="div16">
                        <b class="titulo">Voltaje</b>
                        <div class="grafico">
                            <canvas id="chart_voltage"
style="width:100%; height:100%;"></canvas>
                        </div>
                    </div>
                    <div class="div16">
                        <b class="titulo">Intensidad</b>
                        <div class="grafico">
                            <canvas id="chart_current"
style="width:100%; height:100%;"></canvas>
                        </div>
                    </div>
                </div>
            </div>
        </div>
        <div class="header">
            <img class="logo-icon" alt="" src="{{ url_for('static',
filename='images/logo.svg') }}">

            <div class="nav-botones">
                <div class="general" id="generalContainer">
                    <div class="controlador-e-inversor">General</div>
                </div>
                <div class="general" id="placaSolarContainer">
                    <div class="controlador-e-inversor">Placa
Solar</div>
                </div>
                <div class="turbina-eolica2">
                    <div class="controlador-e-inversor">Turbina
Eólica</div>
                </div>
                <div class="general" id="gestorDeBaterias">
                    <div class="controlador-e-inversor">Gestor de
Baterías</div>
                </div>
            </div>
            <div class="circulos">
                <div class="help">
                    <img class="help-icon" alt="" src="{{
url_for('static', filename='images/help.svg') }}">

                </div>
                <div class="help">
```

```
                    <img class="alarm-icon" alt="" src="{{
url_for('static', filename='images/alarm.svg') }}">

                </div>
                <div class="account">
                    <b class="numero">KZ</b>
                </div>
            </div>
        </div>
    </div>



    <script>
        var generalContainer =
document.getElementById("generalContainer");
        if (generalContainer) {
            generalContainer.addEventListener("click", function (e) {
                window.location.href = "/";  // Redirige a wind.html
            });
        }

        var placaSolarContainer =
document.getElementById("placaSolarContainer");
        if (placaSolarContainer) {
            placaSolarContainer.addEventListener("click", function (e)
{
                window.location.href = "/solar_overview";  // Redirige
a wind.html
            });
        }

        var gestorDeBaterias =
document.getElementById("gestorDeBaterias");
        if (gestorDeBaterias) {
            gestorDeBaterias.addEventListener("click", function (e) {
                window.location.href = "/batteries_overview";  //
Redirige a wind.html
            });
        }
    </script>

    <script>
        // Agregar event listener para el campo de fecha de inicio

document.getElementById('start_date').addEventListener('change',
function () {
            const startDate = this.value;
            console.log("Fecha de inicio seleccionada:", startDate);
        });

        // Agregar event listener para el campo de fecha de fin
        document.getElementById('end_date').addEventListener('change',
function () {
            const endDate = this.value;
            console.log("Fecha de fin seleccionada:", endDate);
        });
```

```
        // Ver las fechas en el formulario cuando se envía
        document.getElementById('download-
form').addEventListener('submit', function (event) {
            const startDate =
document.getElementById('start_date').value;
            const endDate = document.getElementById('end_date').value;

            // Asignar los valores a los inputs ocultos
            document.getElementById('hidden_start_date').value =
startDate;
            document.getElementById('hidden_end_date').value =
endDate;
            // Validar que las fechas no estén vacías
            if (!startDate || !endDate) {
                alert('Por favor, selecciona ambas fechas antes de
descargar el CSV.');
                event.preventDefault();  // Evitar que se envíe el
formulario si están vacías
            }

            // Mostrar las fechas seleccionadas en la consola justo
antes de enviar el formulario
            console.log("Fecha de inicio en el formulario:",
startDate);
            console.log("Fecha de fin en el formulario:", endDate);

        });
    </script>

    <script>
        // Obtener los límites actuales y mostrarlos
        function getLimits() {
            fetch('http://localhost:5200/limits')
                .then(response => response.json())
                .then(data => {
                    // Seleccionamos todos los elementos con la clase
'numero'
                    const elementosNumero =
document.querySelectorAll('.variable');

                    // Iteramos sobre cada elemento y asignamos el
valor correspondiente basado en 'data-variable'
                    elementosNumero.forEach(element => {
                        const variable = element.getAttribute('data-
variable'); // Obtener el valor del atributo 'data-variable'

                        if (variable === 'voltage_max') {
                            element.textContent = data.voltage_max; //
Asignar voltage_max
                        } else if (variable === 'voltage_min') {
                            element.textContent = data.voltage_min; //
Asignar current_max
                        }
                        else if (variable === 'current_max') {
                            element.textContent = data.current_max; //
Asignar current_max
                        }
```

```
                        else if (variable === 'v_cut_in') {
                            element.textContent = data.v_cut_in; //
Asignar current_max
                        }
                        else if (variable === 'v_nominal') {
                            element.textContent = data.v_nominal; //
Asignar current_max
                        }
                        else if (variable === 'v_cut_out') {
                            element.textContent = data.v_cut_out; //
Asignar current_max
                        }
                    });

                    // Asignar valores a los elementos de escritura
(para los inputs)
                    document.querySelector('.numero7').textContent =
data.voltage_max;
                    document.querySelector('.numero8').textContent =
data.voltage_min;
                    document.querySelector('.numero9').textContent =
data.current_max;
                    document.querySelector('.numero10').textContent =
data.v_cut_in;
                    document.querySelector('.numero11').textContent =
data.v_nominal;
                    document.querySelector('.numero12').textContent =
data.v_cut_out;
                })
                .catch(error => console.error('Error fetching
limits:', error));
        }

        // Manejar la actualización de los límites
        document.querySelector('.boton1
button').addEventListener('click', function (event) {
            event.preventDefault();
            const limitsData = {};

            // Obtener los valores actuales de los elementos 'numero7'
y 'kz'
            const voltageMax =
document.querySelector('.numero7').value;
            const voltageMin =
document.querySelector('.numero8').value;
            const currentMax =
document.querySelector('.numero9').value;
            const vcMax = document.querySelector('.numero10').value;
            const vnMax = document.querySelector('.numero11').value;
            const vpMax = document.querySelector('.numero12').value;

            if (voltageMax) {
                limitsData.voltage_max = parseFloat(voltageMax);
            }

            if (voltageMin) {
                limitsData.voltage_min = parseFloat(voltageMin);
            }
```

```
            if (currentMax) {
                limitsData.current_max = parseFloat(currentMax);
            }
            if (vcMax) {
                limitsData.v_cut_in = parseFloat(vcMax);
            }
            if (vnMax) {
                limitsData.v_nominal = parseFloat(vnMax);
            }
            if (vpMax) {
                limitsData.v_cut_out = parseFloat(vpMax);
            }
            console.log("Enviando los siguientes datos:", limitsData);
// Añadir esto para verificar los datos

            fetch('http://localhost:5200/limits', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(limitsData)
            })
                .then(response => response.json())
                .then(data => {
                    alert(data.message);
                    getLimits();  // Actualizar los límites mostrados

                    // Borrar los valores de los inputs después de
actualizar
                    document.querySelector('.numero7').value = ""; //
Limpiar el input de voltage_max
                    document.querySelector('.numero8').value = ""; //
Limpiar el input de current_max
                    document.querySelector('.numero9').value = ""; //
Limpiar el input de voltage_max
                    document.querySelector('.numero10').value = ""; //
Limpiar el input de current_max
                    document.querySelector('.numero11').value = ""; //
Limpiar el input de voltage_max
                    document.querySelector('.numero12').value = ""; //
Limpiar el input de current_max
                })
                .catch(error => console.error('Error actualizando los
límites:', error));
        });

        // Obtener los límites al cargar la página
        getLimits();
    </script>

    <script>
        // Asignar los datos de energía solar, eólica y sus timestamps
        const data_wind = {{ data[-12:] | tojson }};

        // Extraer los timestamps del array de datos de solar
(asumiendo que ambos tienen el mismo número de elementos)
        const timestamps = data_wind.map(d => {
            const date = new Date(d.timestamp);
```

```
            const day = String(date.getDate()).padStart(2, '0');
            const month = String(date.getMonth() + 1).padStart(2,
'0');
            const hours = String(date.getHours()).padStart(2, '0');
            const minutes = String(date.getMinutes()).padStart(2,
'0');
            return `${day}/${month} ${hours}:${minutes}`;
        });

        // Extraer la energía de solar y wind para la gráfica
        const powerData = data_wind.map(d => d.power || 0);  // Si
energy_wh es null, usar 0
        const energyData = data_wind.map(d => d.energy_generated ||
0);    // Lo mismo para wind
        const voltageData = data_wind.map(d => d.voltage || 0);  // Si
energy_wh es null, usar 0
        const currentData = data_wind.map(d => d.current || 0);    //
Lo mismo para wind

        // Calcular el valor máximo de los datos más 200
        const maxPower = Math.max(...powerData) + 0.2;
        const maxEnergy = Math.max(...energyData) + 5;
        const maxVoltage = Math.max(...voltageData) + 15;
        const maxCurrent = Math.max(...currentData) + 0.5;

        // Configuración para la gráfica Potencia
        var power_ctx =
document.getElementById('chart_power').getContext('2d');
        new Chart(power_ctx, {
            type: 'line',
            data: {
                labels: timestamps,  // Los timestamps en formato
DD/MM hh:mm
                datasets: [{
                    label: 'Potencia',
                    data: powerData,
                    borderColor: '#CFA953', // Color de la línea de
Potencia
                    backgroundColor: '#CFA953', // Relleno
transparente
                }]
            },
            options: {
                scales: {
                    y: {
                        beginAtZero: true,
                        max: maxPower,  // Limitar el eje Y al máximo
de los datos + 200
                        title: {
                            display: true,
                            text: 'Potencia (kWh)',  // Etiqueta del
eje Y
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        },
```

```
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    },

                    x: {
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }

                },
                plugins: {
                    legend: {
                        display: true, // Desactivar la leyenda
                        position: 'bottom', // Ubicar la leyenda en la
parte inferior
                        labels: {
                            usePointStyle: true,  // Utilizar círculos
en lugar de rectángulos
                            pointStyle: 'circle', // Definir el estilo
de punto como círculo
                            padding: 24,
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }
                }
            }
        });

        // Configuración para la gráfica Energía
        var energy_ctx =
document.getElementById('chart_energy').getContext('2d');
        new Chart(energy_ctx, {
            type: 'line',
            data: {
                labels: timestamps,  // Los timestamps en formato
DD/MM hh:mm
                datasets: [{
                    label: 'Energía',
                    data: energyData,
                    borderColor: '#549F58', // Color de la línea de
Energia
                    backgroundColor: '#549F58', // Relleno
transparente
                }]
```

```
            },
            options: {
                scales: {
                    y: {
                        beginAtZero: true,
                        max: maxEnergy,  // Limitar el eje Y al máximo
de los datos + 200
                        title: {
                            display: true,
                            text: 'Energía (Wh)',  // Etiqueta del eje
Y
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        },
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    },

                    x: {
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }

                },
                plugins: {
                    legend: {
                        display: true, // Desactivar la leyenda
                        position: 'bottom', // Ubicar la leyenda en la
parte inferior
                        labels: {
                            usePointStyle: true,  // Utilizar círculos
en lugar de rectángulos
                            pointStyle: 'circle', // Definir el estilo
de punto como círculo
                            padding: 24,
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }
                }
            }
        });
```

```
        // Configuración para la gráfica Energía
        var voltage_ctx =
document.getElementById('chart_voltage').getContext('2d');
        new Chart(voltage_ctx, {
            type: 'line',
            data: {
                labels: timestamps,  // Los timestamps en formato
DD/MM hh:mm
                datasets: [{
                    label: 'Voltaje',
                    data: voltageData,
                    borderColor: '#537ACF', // Color de la línea de
Voltaje
                    backgroundColor: '#537ACF', // Relleno
transparente
                }]
            },
            options: {
                scales: {
                    y: {
                        beginAtZero: true,
                        max: maxVoltage,  // Limitar el eje Y al
máximo de los datos + 200
                        title: {
                            display: true,
                            text: 'Voltaje (V)',  // Etiqueta del eje
Y
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        },
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    },

                    x: {
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }

                },
                plugins: {
                    legend: {
                        display: true, // Desactivar la leyenda
```

```
                        position: 'bottom', // Ubicar la leyenda en la
parte inferior
                        labels: {
                            usePointStyle: true,  // Utilizar círculos
en lugar de rectángulos
                            pointStyle: 'circle', // Definir el estilo
de punto como círculo
                            padding: 24,
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }
                }
            }
        });

        // Configuración para la gráfica Intensidad
        var current_ctx =
document.getElementById('chart_current').getContext('2d');
        new Chart(current_ctx, {
            type: 'line',
            data: {
                labels: timestamps,  // Los timestamps en formato
DD/MM hh:mm
                datasets: [{
                    label: 'Intensidad',
                    data: currentData,
                    borderColor: '#CF5389', // Color de la línea de
Intensidad
                    backgroundColor: '#CF5389', // Relleno
transparente
                }]
            },
            options: {
                scales: {
                    y: {
                        beginAtZero: true,
                        max: maxCurrent,  // Limitar el eje Y al
máximo de los datos + 200
                        title: {
                            display: true,
                            text: 'Intensidad (A)',  // Etiqueta del
eje Y
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        },
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
```

```
                        }
                    },

                    x: {
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }

                },
                plugins: {
                    legend: {
                        display: true, // Desactivar la leyenda
                        position: 'bottom', // Ubicar la leyenda en la
parte inferior
                        labels: {
                            usePointStyle: true,  // Utilizar círculos
en lugar de rectángulos
                            pointStyle: 'circle', // Definir el estilo
de punto como círculo
                            padding: 24,
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }
                }
            }
        });
    </script>

</body>

</html>
```

/docker_tfm/frontend/templates/batteries.html

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1, width=device-
width">
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/styles_batteries.css') }}">
    <link rel="stylesheet"
```

```
href="https://fonts.googleapis.com/css2?family=Inter:ital,wght@0,600;0
,700;1,500&display=swap" />
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script type="text/javascript">
        // Recargar la página cada 30 segundos
        setInterval(function () {
            location.reload();
        }, 300000); // 300000 ms = 300 segundos
    </script>
</head>

<body>
    <div class="gestor-de-baterias">
        <div class="container">
            <div class="cuadros">
                <div class="grupo">
                    <div class="gestor-de-baterias1">
                        <b class="titulo">Gestor de Baterías</b>
                        <div class="container1">
                            <div class="grupo-1">
                                <div class="div">
                                    <div
class="titulo1">Temperatura</div>
                                    <div class="variable-grupo">
                                        <b class="variable">{{ data[-
1]['temperature'] }}</b>
                                        <div class="unidad">°C</div>
                                    </div>
                                </div>
                                <div class="div1">
                                    <div class="titulo1">Capacidad de
Carga Total</div>
                                    <div class="variable-grupo">
                                        <b class="variable">{{ data[-
1]['capacity_ah'] }}</b>
                                        <div class="unidad">Ah</div>
                                    </div>
                                </div>
                                <div class="div">
                                    <div class="titulo1">Capacidad
Solar</div>
                                    <div class="variable-grupo">
                                        <b class="variable">{{ data[-
1]['capacity_solar'] }}</b>
                                        <div class="unidad">%</div>
                                    </div>
                                </div>
                                <div class="div">
                                    <div class="titulo1">Capacidad
Turbina</div>
                                    <div class="variable-grupo">
                                        <b class="variable">{{ data[-
1]['capacity_generator'] }}</b>
                                        <div class="unidad">%</div>
                                    </div>
                                </div>
                            </div>
```

```
                            <div class="grupo-1">
                                <div class="div">
                                    <div class="titulo1">Estado de
Carga (SOC)</div>
                                    <div class="variable-grupo">
                                        <b class="variable">{{ data[-
1]['soc'] }}</b>
                                        <div class="unidad">%</div>
                                    </div>
                                </div>
                                <div class="div">
                                    <div class="titulo1">Estado de
Salud (SoH)</div>
                                    <div class="variable-grupo">
                                        <b class="variable">{{ data[-
1]['soh'] }}</b>
                                        <div class="unidad">%</div>
                                    </div>
                                </div>
                                <div class="div">
                                    <div class="titulo1">Eficiencia de
Carga</div>
                                    <div class="variable-grupo">
                                        <b class="variable">{{ data[-
1]['charge_efficiency'] }}</b>
                                        <div class="unidad">%</div>
                                    </div>
                                </div>
                                <div class="div">
                                    <div class="titulo1">Eficiencia de
Descarga</div>
                                    <div class="variable-grupo">
                                        <b class="variable">{{ data[-
1]['discharge_efficiency'] }}</b>
                                        <div class="unidad">%</div>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                    <div class="especificaciones-escritura">
                        <b class="titulo9">Especificaciones</b>
                        <div class="container2">
                            <div class="div">
                                <div class="titulo1">Resistencia
Interna Min</div>
                                <div class="variable-grupo">
                                    <b class="variable" data-
variable="internal_resistance_min"></b>
                                    <div class="unidad">mΩ</div>
                                </div>
                            </div>
                            <div class="div">
                                <div class="titulo1">Resistencia
Interna Max</div>
                                <div class="variable-grupo">
                                    <b class="variable" data-
variable="internal_resistance_max"></b>
```

```
                                    <div class="unidad">mΩ</div>
                                </div>
                            </div>
                            <div class="div">
                                <div class="titulo1">Temperatura
Mínima</div>
                                <div class="variable-grupo">
                                    <b class="variable" data-
variable="temperature_min"></b>
                                    <div class="unidad">°C</div>
                                </div>
                            </div>
                            <div class="div">
                                <div class="titulo1">Temperatura
Máxima</div>
                                <div class="variable-grupo">
                                    <b class="variable" data-
variable="temperature_max"></b>
                                    <div class="unidad">°C</div>
                                </div>
                            </div>
                        </div>
                        <div class="escritura">
                            <div class="container3">
                                <div class="container2">
                                    <div class="div12">
                                        <div
class="titulo1">Resistencia Interna Min</div>
                                        <div class="variable-grupo12">
                                            <div class="variable12">
                                                <input class="numero7"
data-input="internal_resistance_min_input">
                                            </div>
                                            <div
class="unidad12">mΩ</div>
                                        </div>
                                    </div>
                                    <div class="div12">
                                        <div
class="titulo1">Resistencia Interna Max</div>
                                        <div class="variable-grupo12">
                                            <div class="variable12">
                                                <input class="numero8"
data-input="internal_resistance_max_input">
                                            </div>
                                            <div
class="unidad12">mΩ</div>
                                        </div>
                                    </div>
                                    <div class="div12">
                                        <div
class="titulo1">Temperatura Mínima</div>
                                        <div class="variable-grupo12">
                                            <div class="variable12">
                                                <input class="numero9"
data-input="temperature_min_input">
                                            </div>
```

```
                                        <div
class="unidad12">°C</div>
                                    </div>
                                </div>
                                <div class="div12">
                                    <div
class="titulo1">Temperatura Máxima</div>
                                    <div class="variable-grupo12">
                                        <div class="variable12">
                                            <input
class="numero10" data-input="temperature_max_input">
                                        </div>
                                        <div
class="unidad12">°C</div>
                                    </div>
                                </div>
                            </div>
                            <div class="escritura1">
                                <b class="titulo18">Escritura</b>
                                <div class="boton">
                                    <div class="turbina-
elica"><button type="submit">Actualizar</button></div>
                                    <img class="icon1" alt=""
                                        src="{{ url_for('static',
filename='images/update.svg') }}">

                                </div>
                            </div>
                        </div>
                    </div>
                </div>
                <div class="imagen-fecha">
                    <div class="cuadro">
                        <img class="esquema-bateria-1" alt=""
                            src="{{ url_for('static',
filename='images/esquema_bateria.svg') }}">

                    </div>
                    <div class="fecha">
                        <div class="container4">
                            <div class="fecha-inicio">
                                <b class="titulo20">Fecha inicio</b>
                                <div class="selector">
                                    <input type="datetime-local"
id="start_date" name="start_date" required>
                                </div>
                            </div>
                            <div class="fecha-inicio">
                                <b class="titulo21">Fecha fin</b>
                                <div class="selector">
                                    <input type="datetime-local"
id="end_date" name="end_date" required>
                                </div>
                            </div>
                        </div>
                        <div class="boton1">
```

```
                            <form action="/download_batteries"
method="get" id="download-form">
                                <!-- Inputs ocultos que recibirán los
valores de las fechas seleccionadas -->
                                <input type="hidden"
id="hidden_start_date" name="start_date">
                                <input type="hidden"
id="hidden_end_date" name="end_date">
                                <button type="submit" class="styled-
button">Descargar CSV</button>
                            </form>
                        </div>
                    </div>
                </div>
            </div>
            <div class="graficos">
                <div class="div16">
                    <div class="div17">
                        <b class="titulo">Capacidad Carga Total</b>
                        <div class="grafico">
                            <canvas id="chart_cap_total"
style="width:100%; height:100%;"></canvas>
                        </div>
                    </div>
                    <div class="div17">
                        <b class="titulo">Temperatura</b>
                        <div class="grafico">
                            <canvas id="chart_temperature"
style="width:100%; height:100%;"></canvas>
                        </div>
                    </div>
                </div>
                <div class="div16">
                    <div class="div17">
                        <b class="titulo">Capacidad Carga Solar</b>
                        <div class="grafico">
                            <canvas id="chart_solar"
style="width:100%; height:100%;"></canvas>
                        </div>
                    </div>
                    <div class="div17">
                        <b class="titulo">Capacidad Carga Turbina</b>
                        <div class="grafico">
                            <canvas id="chart_wind" style="width:100%;
height:100%;"></canvas>
                        </div>
                    </div>
                </div>
            </div>
        </div>
        <div class="header">
            <img class="logo-icon" alt="" src="{{ url_for('static',
filename='images/logo.svg') }}">

            <div class="nav-botones">
                <div class="general" id="generalContainer">
                    <div class="turbina-elica">General</div>
                </div>
```

```
                <div class="general" id="placaSolarContainer">
                    <div class="turbina-elica">Placa Solar</div>
                </div>
                <div class="general" id="turbinaEolicaContainer">
                    <div class="turbina-elica">Turbina Eólica</div>
                </div>
                <div class="gestor-de-baterias2">
                    <div class="turbina-elica">Gestor de
Baterías</div>
                </div>
            </div>
            <div class="circulos">
                <div class="help">
                    <img class="help-icon" alt="" src="{{
url_for('static', filename='images/help.svg') }}">

                </div>
                <div class="help">
                    <img class="alarm-icon" alt="" src="{{
url_for('static', filename='images/alarm.svg') }}">

                </div>
                <div class="account">
                    <b class="numero">KZ</b>
                </div>
            </div>
        </div>
    </div>

    <script>
        // Agregar event listener para el campo de fecha de inicio

document.getElementById('start_date').addEventListener('change',
function () {
            const startDate = this.value;
            console.log("Fecha de inicio seleccionada:", startDate);
        });

        // Agregar event listener para el campo de fecha de fin
        document.getElementById('end_date').addEventListener('change',
function () {
            const endDate = this.value;
            console.log("Fecha de fin seleccionada:", endDate);
        });

        // Ver las fechas en el formulario cuando se envía
        document.getElementById('download-
form').addEventListener('submit', function (event) {
            const startDate =
document.getElementById('start_date').value;
            const endDate = document.getElementById('end_date').value;

            // Asignar los valores a los inputs ocultos
            document.getElementById('hidden_start_date').value =
startDate;
            document.getElementById('hidden_end_date').value =
endDate;
            // Validar que las fechas no estén vacías
```

```
            if (!startDate || !endDate) {
                alert('Por favor, selecciona ambas fechas antes de
descargar el CSV.');
                event.preventDefault();  // Evitar que se envíe el
formulario si están vacías
            }

            // Mostrar las fechas seleccionadas en la consola justo
antes de enviar el formulario
            console.log("Fecha de inicio en el formulario:",
startDate);
            console.log("Fecha de fin en el formulario:", endDate);

        });
    </script>


    <script>
        var generalContainer =
document.getElementById("generalContainer");
        if (generalContainer) {
            generalContainer.addEventListener("click", function (e) {
                window.location.href = "/";  // Redirige a wind.html
            });
        }

        var placaSolarContainer =
document.getElementById("placaSolarContainer");
        if (placaSolarContainer) {
            placaSolarContainer.addEventListener("click", function (e)
{
                window.location.href = "/solar_overview";  // Redirige
a wind.html
            });
        }

        var turbinaEolicaContainer =
document.getElementById("turbinaEolicaContainer");
        if (turbinaEolicaContainer) {
            turbinaEolicaContainer.addEventListener("click", function
(e) {
                window.location.href = "/wind_overview";  // Redirige
a wind.html
            });
        }
    </script>

    <script>
        // Obtener los límites actuales y mostrarlos
        function getLimits() {
            fetch('http://localhost:5300/limits')
                .then(response => response.json())
                .then(data => {
                    // Seleccionamos todos los elementos con la clase
'numero'
                    const elementosNumero =
document.querySelectorAll('.variable');
```

```
                    // Iteramos sobre cada elemento y asignamos el
valor correspondiente basado en 'data-variable'
                    elementosNumero.forEach(element => {
                        const variable = element.getAttribute('data-
variable'); // Obtener el valor del atributo 'data-variable'

                        if (variable === 'internal_resistance_min') {
                            element.textContent =
data.internal_resistance_min; // Asignar voltage_max
                        } else if (variable ===
'internal_resistance_max') {
                            element.textContent =
data.internal_resistance_max; // Asignar current_max
                        }
                        else if (variable === 'temperature_min') {
                            element.textContent =
data.temperature_min; // Asignar current_max
                        }
                        else if (variable === 'temperature_max') {
                            element.textContent =
data.temperature_max; // Asignar current_max
                        }
                    });

                    // Asignar valores a los elementos de escritura
(para los inputs)
                    document.querySelector('.numero7').textContent =
data.internal_resistance_min;
                    document.querySelector('.numero8').textContent =
data.internal_resistance_max;
                    document.querySelector('.numero9').textContent =
data.temperature_min;
                    document.querySelector('.numero10').textContent =
data.temperature_max;
                })
                .catch(error => console.error('Error fetching
limits:', error));
        }

        // Manejar la actualización de los límites
        document.querySelector('.boton
button').addEventListener('click', function (event) {
            event.preventDefault();
            const limitsData = {};

            // Obtener los valores actuales de los elementos 'numero7'
y 'kz'
            const resistanceMin =
document.querySelector('.numero7').value;
            const resistanceMax =
document.querySelector('.numero8').value;
            const temperatureMin =
document.querySelector('.numero9').value;
            const temperatureMax =
document.querySelector('.numero10').value;

            if (resistanceMin) {
```

```
                limitsData.internal_resistance_min =
parseFloat(resistanceMin);
            }

            if (resistanceMax) {
                limitsData.internal_resistance_max =
parseFloat(resistanceMax);
            }

            if (temperatureMin) {
                limitsData.temperature_min =
parseFloat(temperatureMin);
            }
            if (temperatureMax) {
                limitsData.temperature_max =
parseFloat(temperatureMax);
            }
            console.log("Enviando los siguientes datos:", limitsData);
// Añadir esto para verificar los datos

            fetch('http://localhost:5300/limits', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(limitsData)
            })
                .then(response => response.json())
                .then(data => {
                    alert(data.message);
                    getLimits();  // Actualizar los límites mostrados

                    // Borrar los valores de los inputs después de
actualizar
                    document.querySelector('.numero7').value = ""; //
Limpiar el input de voltage_max
                    document.querySelector('.numero8').value = ""; //
Limpiar el input de current_max
                    document.querySelector('.numero9').value = ""; //
Limpiar el input de voltage_max
                    document.querySelector('.numero10').value = ""; //
Limpiar el input de current_max
                })
                .catch(error => console.error('Error actualizando los
límites:', error));
        });

        // Obtener los límites al cargar la página
        getLimits();
    </script>

    <script>
        // Asignar los datos de energía solar, eólica y sus timestamps
        const data_batteries = {{ data[-12:] | tojson }};

        // Extraer los timestamps del array de datos de solar
(asumiendo que ambos tienen el mismo número de elementos)
        const timestamps = data_batteries.map(d => {
```

```
            const date = new Date(d.timestamp);
            const day = String(date.getDate()).padStart(2, '0');
            const month = String(date.getMonth() + 1).padStart(2,
'0');
            const hours = String(date.getHours()).padStart(2, '0');
            const minutes = String(date.getMinutes()).padStart(2,
'0');
            return `${day}/${month} ${hours}:${minutes}`;
        });

        // Extraer la energía de solar y wind para la gráfica
        const cap_totalData = data_batteries.map(d => d.capacity ||
0);  // Si energy_wh es null, usar 0
        const temperatureData = data_batteries.map(d =>
d.energy_generated || 0);    // Lo mismo para wind
        const cap_solarData = data_batteries.map(d => d.capacity_solar
|| 0);  // Si energy_wh es null, usar 0
        const cap_windData = data_batteries.map(d =>
d.capacity_generator || 0);    // Lo mismo para wind

        // Calcular el valor máximo de los datos más 200
        const maxCap_total = Math.max(...cap_totalData) + 2;
        const maxTemperature = Math.max(...temperatureData) + 5;
        const maxCap_solar = Math.max(...cap_solarData) + 10;
        const maxCap_wind = Math.max(...cap_windData) + 10;

        // Configuración para la gráfica Potencia
        var cap_total_ctx =
document.getElementById('chart_cap_total').getContext('2d');
        new Chart(cap_total_ctx, {
            type: 'line',
            data: {
                labels: timestamps,  // Los timestamps en formato
DD/MM hh:mm
                datasets: [{
                    label: 'Capacidad Total',
                    data: cap_totalData,
                    borderColor: '#CFA953', // Color de la línea de
Potencia
                    backgroundColor: '#CFA953', // Relleno
transparente
                }]
            },
            options: {
                scales: {
                    y: {
                        beginAtZero: true,
                        max: maxCap_total,  // Limitar el eje Y al
máximo de los datos + 200
                        title: {
                            display: true,
                            text: 'Capacidad carga (Ah)',  // Etiqueta
del eje Y
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
```

```
                        },
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    },

                    x: {
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }

                },
                plugins: {
                    legend: {
                        display: true, // Desactivar la leyenda
                        position: 'bottom', // Ubicar la leyenda en la
parte inferior
                        labels: {
                            usePointStyle: true,  // Utilizar círculos
en lugar de rectángulos
                            pointStyle: 'circle', // Definir el estilo
de punto como círculo
                            padding: 24,
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }
                }
            }
        });

        // Configuración para la gráfica Energía
        var temperature_ctx =
document.getElementById('chart_temperature').getContext('2d');
        new Chart(temperature_ctx, {
            type: 'line',
            data: {
                labels: timestamps,  // Los timestamps en formato
DD/MM hh:mm
                datasets: [{
                    label: 'Temperatura',
                    data: temperatureData,
                    borderColor: '#549F58', // Color de la línea de
Energia
                    backgroundColor: '#549F58', // Relleno
transparente
```

```
                }]
            },
            options: {
                scales: {
                    y: {
                        beginAtZero: true,
                        max: maxTemperature,  // Limitar el eje Y al
máximo de los datos + 200
                        title: {
                            display: true,
                            text: 'Temperatura (°C)',  // Etiqueta del
eje Y
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        },
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    },

                    x: {
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }

                },
                plugins: {
                    legend: {
                        display: true, // Desactivar la leyenda
                        position: 'bottom', // Ubicar la leyenda en la
parte inferior
                        labels: {
                            usePointStyle: true,  // Utilizar círculos
en lugar de rectángulos
                            pointStyle: 'circle', // Definir el estilo
de punto como círculo
                            padding: 24,
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }
                }
            }
```

```
        });

        // Configuración para la gráfica Energía
        var cap_solar_ctx =
document.getElementById('chart_solar').getContext('2d');
        new Chart(cap_solar_ctx, {
            type: 'line',
            data: {
                labels: timestamps,  // Los timestamps en formato
DD/MM hh:mm
                datasets: [{
                    label: 'Capacidad solar',
                    data: cap_solarData,
                    borderColor: '#537ACF', // Color de la línea de
Voltaje
                    backgroundColor: '#537ACF', // Relleno
transparente
                }]
            },
            options: {
                scales: {
                    y: {
                        beginAtZero: true,
                        max: maxCap_solar,  // Limitar el eje Y al
máximo de los datos + 200
                        title: {
                            display: true,
                            text: 'Capacidad carga (%)',  // Etiqueta
del eje Y
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        },
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    },

                    x: {
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }

                },
                plugins: {
                    legend: {
                        display: true, // Desactivar la leyenda
```

```
                        position: 'bottom', // Ubicar la leyenda en la
parte inferior
                        labels: {
                            usePointStyle: true,  // Utilizar círculos
en lugar de rectángulos
                            pointStyle: 'circle', // Definir el estilo
de punto como círculo
                            padding: 24,
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }
                }
            }
        });

        // Configuración para la gráfica Intensidad
        var cap_wind_ctx =
document.getElementById('chart_wind').getContext('2d');
        new Chart(cap_wind_ctx, {
            type: 'line',
            data: {
                labels: timestamps,  // Los timestamps en formato
DD/MM hh:mm
                datasets: [{
                    label: 'Capacidad turbina',
                    data: cap_windData,
                    borderColor: '#CF5389', // Color de la línea de
Intensidad
                    backgroundColor: '#CF5389', // Relleno
transparente
                }]
            },
            options: {
                scales: {
                    y: {
                        beginAtZero: true,
                        max: maxCap_wind,  // Limitar el eje Y al
máximo de los datos + 200
                        title: {
                            display: true,
                            text: 'Capacidad carga (%)',  // Etiqueta
del eje Y
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        },
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
```

```
                        }
                    },

                    x: {
                        ticks: {
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }

                },
                plugins: {
                    legend: {
                        display: true, // Desactivar la leyenda
                        position: 'bottom', // Ubicar la leyenda en la
parte inferior
                        labels: {
                            usePointStyle: true,  // Utilizar círculos
en lugar de rectángulos
                            pointStyle: 'circle', // Definir el estilo
de punto como círculo
                            padding: 24,
                            color: '#292F32',
                            font: {
                                size: 14,
                                family: 'Inter'
                            }
                        }
                    }
                }
            }
        });
    </script>
</body>

</html>
```