Next Generation Meta Operating System

D2 MVP Documentation

Document Identification			
Status	Final	Due Date	25/10/2024
Version	1.0	Submission Date	25/10/2024

Related WP	WP2	Document Reference	D2.0
Related Deliverable(s)	D1 - Detailed Desgin	Dissemination Level (*)	PU
Lead Participant	MAR	Lead Author	Amjad Majid
Contributors	Favian Gozali	Reviewers	Favian Gozali
			Massimo Neri

Keywords:

(*) Dissemination level: **(PU)** Public, fully open, e.g., web (Deliverables flagged as public will be automatically published in CORDIS project's page). **(SEN)** Sensitive, limited under the conditions of the Grant Agreement. **(Classified EU-R)** EU RESTRICTED under the Commission Decision No2015/444. **(Classified EU-C)** EU CONFIDENTIAL under the Commission Decision No2015/444. **(Classified EU-S)** EU SECRET under the Commission Decision No2015/444.

Energy, IoT, data processing, data visualization, minimum viable product

Document Information

List of Contributors	
Name	Partner
Amjad Majid	Martel Innovate
Favian Gozali	Martel Innovate

Document History			
Version	Date	Change editors	Changes
0.1	20/10/2024	Favian Gozali (MAR)	Initial Content
0.5	22/10/2024	Amjad Majid (MAR)	Improved Content and layout
1.0	25/10/2024	Favian Gozali (MAR) Amjad Majid (MAR)	Final Version

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Favian Gozali (MAR)	24/10/2024
Quality manager	Amjad Majid (MAR)	
Project Coordinator	Albert Seubers (MAR)	
Technical Manager	Amjad Majid (MAR)	

Table of Contents

List of Figures

List of Acronyms

Abbreviation / acronym	Description
AI	Artificial Intelligence
Dx.y	Deliverable number y belonging to WP x
EC	European Commission
EV	Electic Vehicle
HQ	Headquarter
IoT	Internet of Things
kWh	Kilowatt-hour
ML	Machine Learning
MVP	Minimum Viable Product
OSS	One Stop Shop
PV	photovoltaic
WP	Work Package

Executive Summary

Eros4NRG is a platform designed to monitor and analyze energy production and consumption, enabling more efficient decision-making through predictive analytics. It acts as a one-stop-shop (OSS) for energy stakeholders by integrating data from electric vehicles (EVs), EV charging stations, photovoltaic (PV) plants, and smart headquarters. The platform enhances trust and transparency in energy data, while addressing key challenges such as: (i) Unreliable data from IoT sources impacting machine learning operations; (ii) poor organization of static energy data leading to information loss; and (iii) the need for user-friendly AI/ML insights for non-technical stakeholders.

Eros4NRG's Minimum Viable Product (MVP) includes four key modules:

1. *Data Gathering Module:* Continuously collects raw data from sources like EV fleets and PV systems, storing it in a MinIO data lake for efficient, scalable processing.
2. *Data Processing Module:* Transforms raw data into structured formats in PostgreSQL for querying, visualization, and AI/ML model training, ensuring data quality and reliability.
3. *Data Visualization Module:* Provides an intuitive dashboard for real-time and historical data insights, offering metrics like battery status, charging sessions, and energy consumption to support informed decision-making.
4. *Data Catalogue Module:* Offers an API interface for querying structured data, enabling users to retrieve and analyze both real-time and historical data easily.

This document also outlines the next steps required to transition from the MVP to a fully functional product:

1. *AI-Driven Predictive Analytics:* Integration of AI models to forecast energy consumption, optimize EV battery health, and improve charging station usage.
2. *Zero-Trust Security Module:* Implementation of zero-trust architecture to enforce strict access controls and protect sensitive data.
3. *Enhanced Dashboard:* Expansion of the dashboard for more detailed insights, customizable views, and real-time alerts.
4. *Improved Data Cleaning Module:* Enhanced data cleaning processes to ensure higher data accuracy and model performance.

The MVP of Eros4NRG establishes a solid foundation for monitoring and optimizing energy systems through advanced data integration, processing, and visualization. With its robust architecture and modular approach, the platform is positioned to address the evolving needs of energy stakeholders by ensuring data accuracy, security, and usability. Moving forward, Eros4NRG will continue to expand its capabilities, incorporating AI-driven insights to predict energy trends, fortifying its security with zero-trust principles, and enhancing the user interface for deeper insights and more intuitive decision-making.

1 Introduction

Eros4NRG is a comprehensive platform for monitoring and analysing energy production and consumption. Through data-driven predictive analytics, Eros4NRG enables end users to make informed decisions about their energy operations, leading to more efficient business operations. This platform will also serve as a one-stop shop (OSS) for energy stakeholders. To realise this vision, we plan to involve in the development process two main stakeholder groups:

- **EMOTION** main entities that produce the data assets which will be exploited by Eros4NRG are EVs (Electric Vehicles) and EV charging stations.
- **ASM Terni** main entities that produce data that will be exploited and later processed by Eros4NRG are PVs (photovoltaic plants) and ASM Smart HQs.

In addition to enegy data analytics, Eros4NRG aims to enhance data trust & transparency by analysing and securing IoT and edge data. Also, establishing cutting-edge data storage solutions that enhance semantic connections between dynamic information and metadata. Finally, Eros4NRG aims to enhance the Nemo Project Smart Energy Trial by acting as a native extention to it. To realize this vision, Eros4NRG will address the following key challenges:

- **Unreliable Energy IoT/Data Sources:** Addressing the impact of Inaccurate data in energy ecosystems (Buildings, PVs, EVs) on ML operations and training.
- **Lack of Organization and Management of Energy Data:** Tackling overlooked static data elements like energy entity attributes, environment, geolocation, and device fleets, leading to information loss and diminished analytics quality.
- **Involving Stakeholders in Cross-Value Chain Services:** Enhancing AI/ML model explainability and translating machine learning predictions into user-friendly language to make them comprehensible to non-technical stakeholders.
- **Deliver a one-stop-shop (OSS) for predictive energy analytics**: An OSS has the potential to be a centralised platform that offers all the necessary tools and resources for utilising predictive analytics in the energy sector. This platform will include features such as data collection, analysis, modelling, and forecasting, as well as tools for visualisation and interpretation of results.

1.1 Purpose of the document

This document details the minimum viable product (MVP) of the Eros4NRG platform. It outlines the key components and functionalities developed so far, with a focus on the following four main modules:

1. Data Gathering Module: Responsible for collecting raw data from various sources.
2. Data Processing Module: Managing data preprocessing, storage, and structuring for analysis.
3. Data visualization Module: Provides intuitive dashboards and reports for visualizing and interpreting the processed data.
4. Data Catalogue Module: Enables querying and managing the clearned data stored in the clean data bucket (the PostgresSQL database).

This document also highlights the architecture of the platform and outlines the next steps for further development and improvements.

1.2 Relation to other project work

This is the second deliverable that presents the MVP of Eros4NRG following the first deliverable that detailed the architecture design of Eros4NRG.

1.3 Structure of the document

This document is organized into two main chapters:

- **Chapter 2: Minimum Viable Product (MVP)**

  This chapter details the MVP of the Eros4NRG platform, structured into four sub-sections:

  - *Data Gathering Module:* Discusses the process of collecting raw data from EMOTION and ASM Terni systems.
  - *Data Processing Module:* Describes the extraction, structuring, and storage of data in PostgreSQL after processing.
  - *Data Visualization Module:* Covers the dashboard for visualizing key energy metrics and data analysis.
  - *Data Catalogue Module:* Explains how users can query and interact with the cleaned data stored in the PostgreSQL database.

- **Chapter 3: Next Steps**
  This chapter outlines the future steps required to advance the Eros4NRG platform, including integrating AI-driven analytics, improving security measures, and enhancing the user interface for better decision-making capabilities.

2 Eros4NRG Minimum Viable Product

Eros4NRG relies on a comprehensive set of both functional and non-functional requirements to shape its system architecture. This architecture is designed to be service-oriented and seamlessly compatible with modern cloud-native infrastructures such as MinIO, Postgres, and Docker. At the heart of the Eros4NRG platform lie its core components, which include:

- **Zero Trust Data Services**: This component plays a pivotal role in Eros4NRG's operations by extracting crucial information from ASM Terni and EMOTION IoT/Edge Fleets. Its duties encompass not only metadata decomposition, but also ensuring that data is appropriately structured and prepared for further analysis.
- **Collaborative Data Enclaves**: This section houses the storage components and data models that harmonize with the data assets.
- **One-Stop-Shop for Energy (OSS 4 Energy):** This upstream component serves as a dashboard, showcasing the results of AI/ML models that deliver predictive analytics within the energy sector. It also facilitates advanced visualizations, reports, and queries for enhanced insight and decision-making

The MVP consists of four main modules: the data gathering module, the data processing module, the data visualisation module, and the data catalogue module.

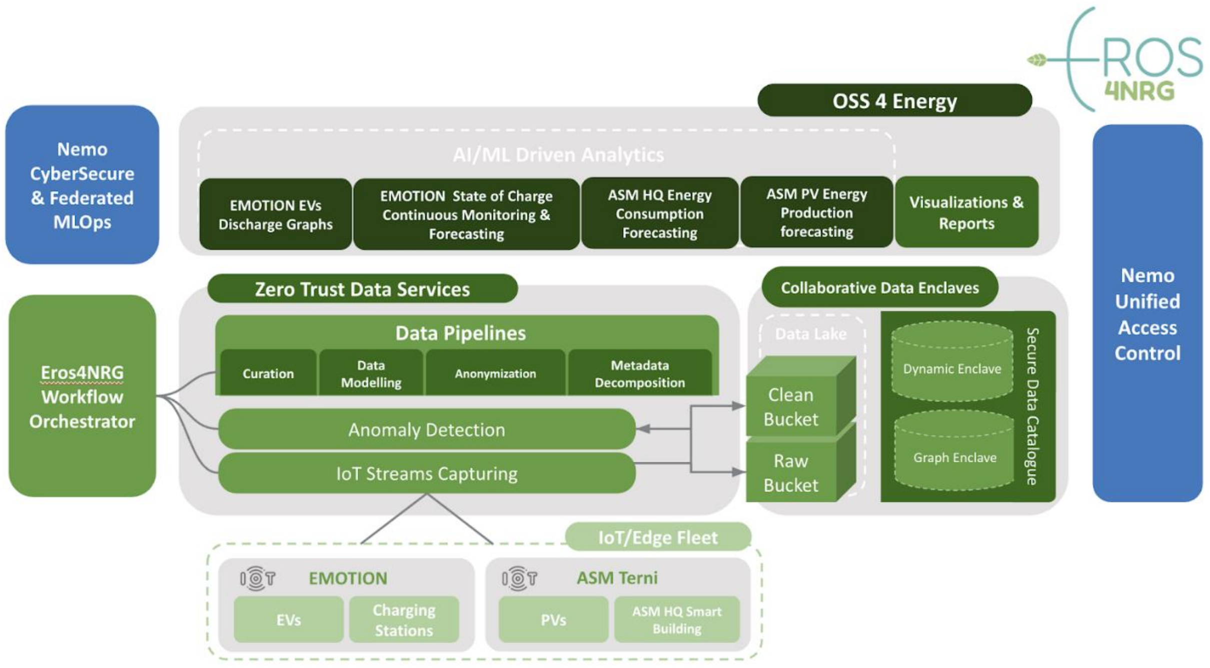


Figure 1: Eros4NRG's overall architecture

2.1 Data Gathering Module

Figure 2 illustrates the data gathering module. It is responsible for capturing raw data from EMOTION and ASM Terni systems via API calls. This continuous stream of data enables the platform to monitor and analyze energy consumption and production, as well as EV battery performance in real time. The module starts by sending API requests to the EMOTION and ASM Terni systems. The collected data is fetched in chronological order, starting from the earliest available records to ensure the entire dataset is retrieved without missing any historical context. After the initial data collection, the system checks for new data entries every five minutes to maintain an up-to-date dataset.

The raw data is stored in a MinIO bucket acting as a data lake. MinIO is selected for its ability to efficiently manage large volumes of unstructured data, offering high scalability and rapid retrieval. This data is crucial for feeding downstream processes, including data preprocessing and anomaly detection. The MinIO architecture allows Eros4NRG to store data as soon as it is received, providing a streamlined approach to handling large data volumes from distributed sources such as PV plants and EV charging stations.

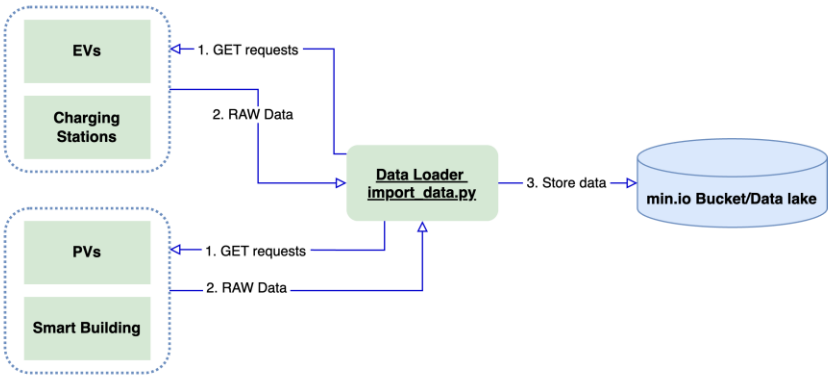


Figure 2: The data gathering module

2.2 Data Processing Module

The Data Processing Module is central to transforming the raw data into a structured format suitable for further analysis and querying. As illustrated in Figure 3, this module extracts the stored data from the MinIO bucket, applies preprocessing steps, and stores it in a relational database for efficient access.

The module starts by reading data from the MinIO bucket. It decomposes the data by separating metadata from actual data points. The processed data is then formatted as a DataFrame and stored in a PostgreSQL database. PostgreSQL, being a highly scalable and reliable relational database system, is used to facilitate easier query logic, especially for data visualization and data cataloging. This allows users to perform complex queries and retrieve specific data insights efficiently. Once data has been successfully transferred to PostgreSQL, it is deleted from MinIO to avoid data duplication and ensure storage optimization. This module not only enables the transformation of raw data into structured information but also lays the groundwork for advanced querying and AI/ML model training.

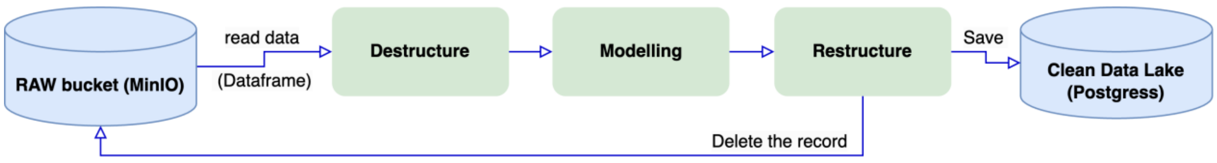


Figure 3: The data processing module

2.3 Data Visualization Module

The Eros4NRG dashboard, as shown in Figure 4, provides an intuitive interface for visualizing data stored in the PostgreSQL database. Users can select specific data providers and zoom in on the data of interest. For example, Figure 4 depicts data related to a vehicle from the EMOTION car fleet, identified as car ID 7. The displayed panels offer key metrics, such as the number of charges, battery percentage, charging status, and total kilometers driven. These visual insights help users understand trends, performance, and the overall condition of the vehicle fleet.

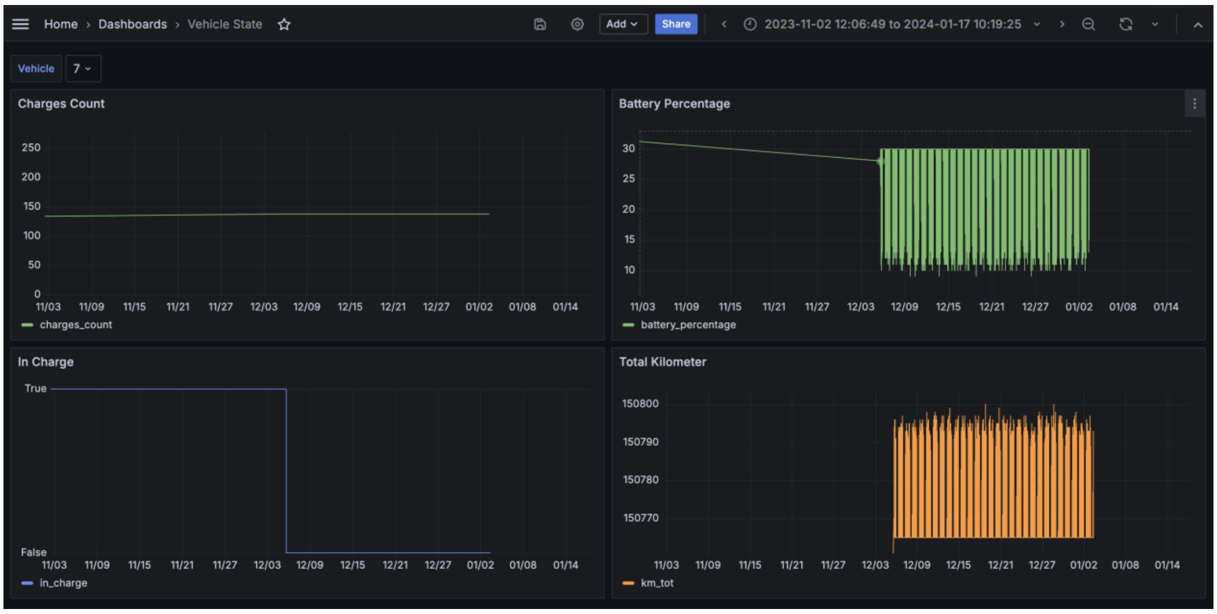


Figure 4: Eros4NRG Dashboard

The dashboard also allows for more granular data investigation over specific time frames. For instance, Figure 5 showcases data from vehicle ID 3, displaying battery percentage, charge amount, and total kilometers traveled. The graph reveals an anomaly in the kWh charged—it remains stagnant despite the vehicle covering more kilometers recently. This inconsistency, especially when the vehicle was charged at a low battery level, suggests potential issues with the sensor capturing the total kWh charged. This information is valuable for energy stakeholders, who can further investigate sensor accuracy and energy efficiency.

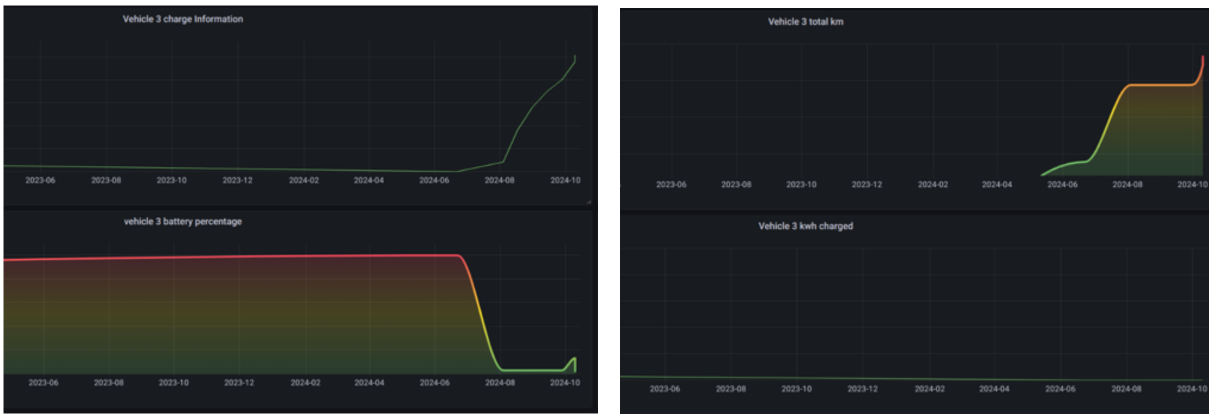


Figure 5: Information regarding vehicle 3

Document name:	D2 MVP Documentation					**Page:**	12 of 24
Reference:	D2.0	**Dissemination:**	PU	**Version:**	1.0	**Status:**	Final

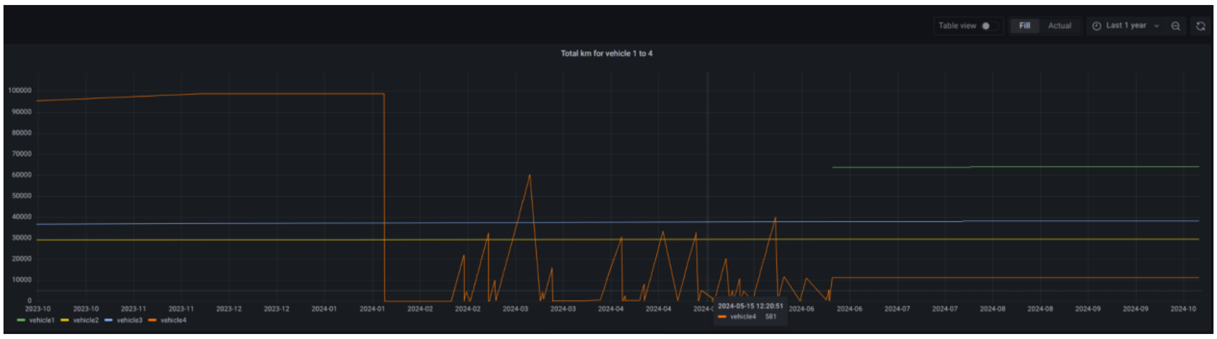


Figure 6 illustrates the total kilometers recorded between charges for vehicles 1, 2, 3, and 4. The graph indicates relatively consistent usage patterns for vehicles 1, 2, and 3. However, vehicle 4 exhibits significant variability in its usage, particularly from January to July 2024. This noticeable change invites energy stakeholders to investigate events in December 2023, as the vehicle showed higher mileage before that period.

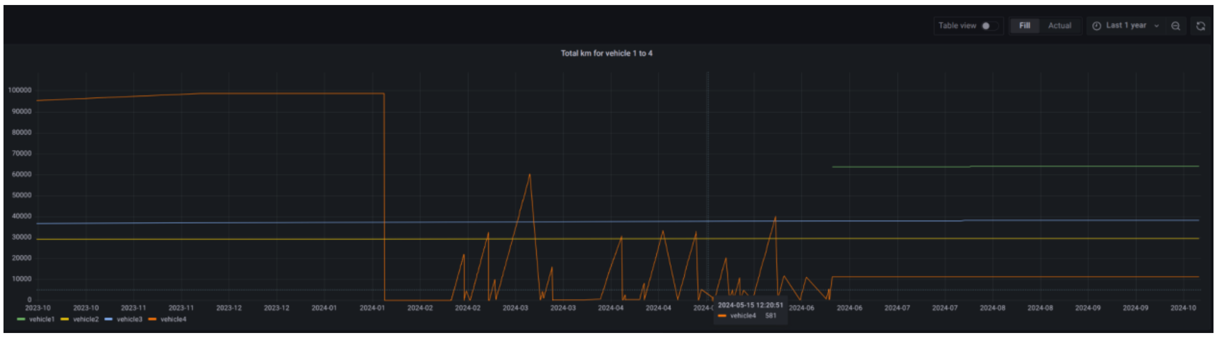


Figure 6: total kilometer for vechicle 1 to 4

Figure 7 presents a comparison of battery percentages recorded between charges for vehicles 1, 3, 8, and 9. The graph highlights that vehicles 1 and 3 were infrequently used from October to December 2023. In contrast, vehicle 8 showed consistent usage, as indicated by the progressively lower battery percentages with each charge. However, a noticeable shift occurred in June 2024, when vehicles 1 and 3 experienced a sharp drop in battery percentage, suggesting significant usage during that period. By July 2024, this trend extended to all four vehicles, all of which experienced substantial declines in battery percentage, continuing up to recent times.

Additionally, the data shows an anomaly for vehicle 8 in October 2023, where the recorded battery percentage spiked to an abnormal value of approximately 2264%, far exceeding the 100% limit. This anomaly may be due to a sensor error or a misrecorded value. Energy stakeholders can use this insight to recommend a maintenance check for vehicle 8 to address potential sensor issues.

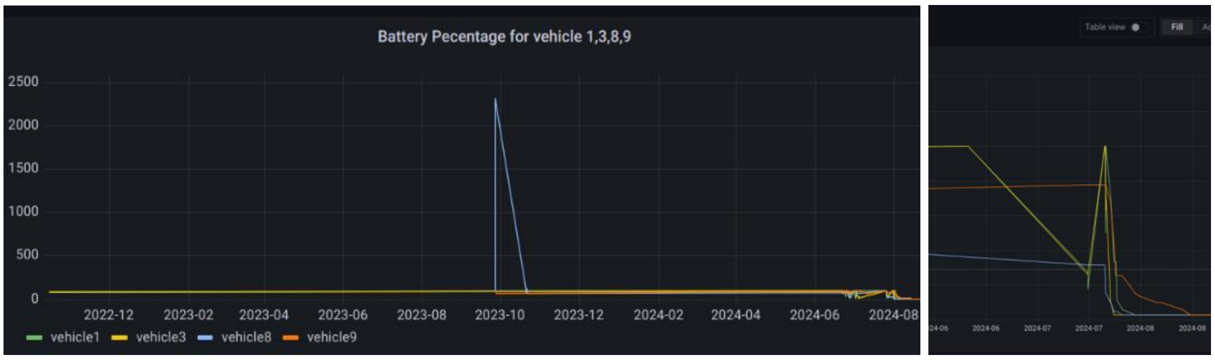
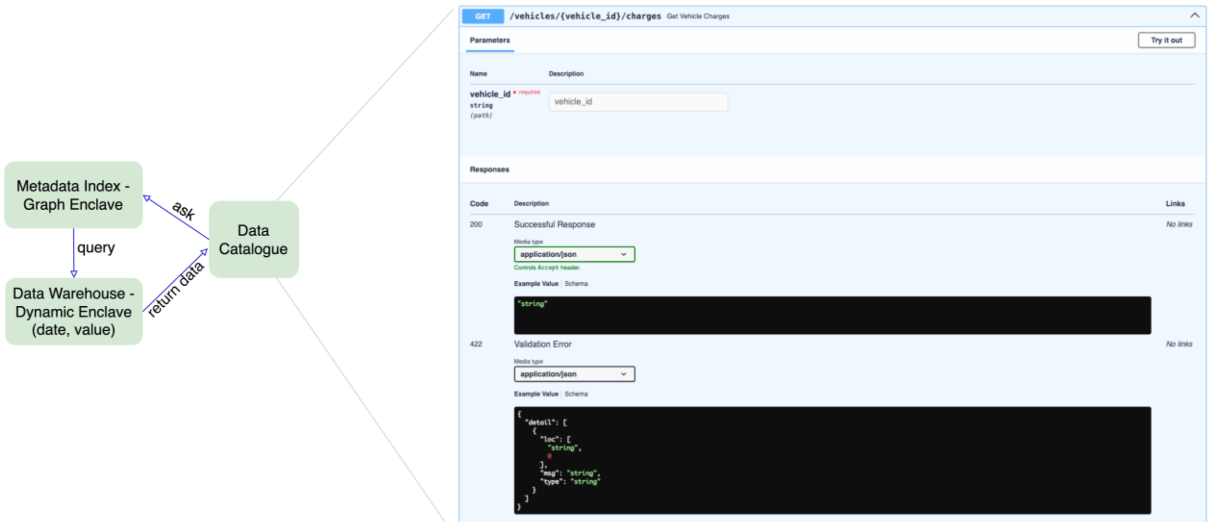


Figure 7: Battery percentage for vehicle 1,3,8,9

Document name:	D2 MVP Documentation					**Page:**	13 of 24
Reference:	D2.0	**Dissemination:**	PU	**Version:**	1.0	**Status:**	Final

2.4 Data Catalogue

The Data Catalogue is an access point created using FastAPI to enable users to query the cleaned data stored in the postgres database. It creates a request into the metadata table asking for specific information the user needs which then queries the history table that then returns the information asked by the user. Alternatively, it can also provide direct information from the database if the user already has the history ID that they want presented. Currently there are 3 endpoints, 2 of which are for accessing the metadata and 1 for directly accessing history information.


Metadata Index - Graph Enclave
ask
query
Data Catalogue
Data Warehouse - Dynamic Enclave (date, value)
return data
GET
Try it out

```
/vehicles/{vehicle_id}/charges  Get Vehicle Charges

Parameters

Name                    Description
vehicle_id * required   vehicle_id
string
(path)

Responses

Code   Description                                   Links
200    Successful Response                           No links
       Media type
       application/json
       Controls Accept header.
       Example Value | Schema
       "string"
422    Validation Error                              No links
       Media type
       application/json
       Example Value | Schema
       {
         "detail": [
           {
             "loc": [
               "string",
               0
             ],
             "msg": "string",
             "type": "string"
           }
         ]
       }
```

Figure 8: Data Catalogue

2.5 User manual and relevant libraries

To run the data gathering, processing, and visualization module, follow the following commands.

1. Clone the repository

```
Git clone https://github.com/Eros4NRG/Eros4NRG
```

2. Move inside the Eros4NRG folder

```
cd Eros4NRG
```

3. run the following docker-compose command in the terminal

```
docker-compose -f <<the .yaml filename>> up -d -build
```

4. Docker command to access individual services:

```
docker exec -it <<service name>> bash
```

5. Docker command to access log data for each service:

```
docker logs <<service name>>
```

6. Docker compose command to shut down the project

```
docker-compose -f <<the .yaml filename>> down -v
```

3 Next Steps

Building on the MVP, Eros4NRG will undergo several enhancements and additions to maximize its potential and improve user experience. The next steps in development include:

1. **AI-Driven Predictive Analytics:** Eros4NRG will integrate sophisticated AI models to predict future energy production and consumption trends, forecast EV battery health, and optimize charging station usage. This will enable stakeholders to proactively plan for energy demands and resource allocation. AI techniques such as Neural Networks (LSTMs), Time-Series Analysis, and Advanced Statistical Models will be employed to deliver accurate forecasts and data-driven insights.
2. **Zero-Trust Security Module:** To enhance data protection, Eros4NRG will implement a zero-trust security architecture, ensuring strict role-based and resource-based access control across the platform. This will safeguard sensitive energy and IoT data from unauthorized access, ensuring compliance with security standards and protecting stakeholder data.
3. **Dashboard Enhancements:** The data visualization dashboard will be expanded to provide a richer user experience, enabling users to perform more complex queries and view more detailed reports. The focus will be on improving usability by providing customizable views, enhanced filters, and real-time alerts for anomalies or critical energy events.
4. **Improved Data Cleaning Module:** Enhancements to the data cleaning module will ensure more robust data processing by automating the detection and resolution of inconsistencies, duplicates, and outliers. This will further increase the accuracy of AI models and improve the overall reliability of the platform's data outputs.

4 Conclusions

This document has outlined the Minimum Viable Product (MVP) of the Eros4NRG platform, highlighting the key components necessary for monitoring and analyzing energy production and consumption. The MVP addresses several challenges, including unreliable IoT data sources, lack of proper energy data management, and the need for clearer AI/ML insights for stakeholders. However, further work is required to fully address data organization and management and ensure seamless cross-value chain service integration. The four core modules—Data Gathering, Data Processing, Data Visualization, and Data Catalogue—establish a strong foundation for Eros4NRG's operations. These modules will serve as the input for future deliverables focused on enhancing predictive analytics capabilities and refining user interaction with energy data.

The next steps in the project involve integrating AI-driven predictive analytics, implementing zero-trust security models, and expanding the dashboard’s functionality for improved user decision-making. These enhancements are aligned with the overall project roadmap, ensuring that Eros4NRG evolves to meet both the technical and operational needs of energy stakeholders.

5 Annexes

5.1 API Documentation

Eros4NRG provides an API module (based on FastAPI) that allows stakeholders to retrieve information directly from the Postgres database in a json format. Currently there are 3 endpoints that are provided

Metadata information regarding vehicles	
Description	List out available vehicle IDs and provides extracted metadata information regarding specific vehicles which is a list of charge history IDs
HTTP Method	GET
Endpoint	1. `/vehicle` 2. `/vehicle=${vehicle_id}`
Parameters	Vehicle_id = the specific vehicle ID that the stakeholders would like metadata information from
Output	1. [1,2,3,4,5,6...] 2. [657841,324652, 626743,...]

Table 9: Vehicle Metadata endpoints

Information regarding vehicle charge history	
Description	Provides detailed information regarding specific vehicles charge history
HTTP Method	GET
Endpoint	`/charges=${charge_id}`
Parameters	charge_id = the specific charge history ID that the stakeholders would like more detailed information from
Output	[Id Timestamp Charge_count Kwh_charged ...]

Table 10: Vehicle Charge History endpoint

5.2 Eros4NRG Code Source

IoT Data Capturing and Storage Code

This section includes the Python code responsible for fetching vehicle state data from APIs, storing it in MinIO, and ensuring continuous data capture and storage.

```
import click
import requests
import json
from minio import Minio
from minio.error import S3Error
import io
import os
import time

# MinIO Configuration from environment variables
MINIO_URL = os.getenv('MINIO_URL')
ACCESS_KEY = os.getenv('ACCESS_KEY')
SECRET_KEY = os.getenv('SECRET_KEY')
BUCKET_NAME = os.getenv('BUCKET_NAME')

# Initialize MinIO client
client = Minio(
    MINIO_URL,
    access_key=ACCESS_KEY,
    secret_key=SECRET_KEY,
    secure=False  # Set to True if using HTTPS
)

# Create the bucket if it doesn't exist
if not client.bucket_exists(BUCKET_NAME):
    client.make_bucket(BUCKET_NAME)
else:
    print(f"Bucket '{BUCKET_NAME}' already exists.")

@click.command()
@click.option('--url', required=True, help="The data source URL")
@click.option('--drct', required=True, help="A directory name for the data to be stored")
def fetch_and_store_vehicle_states(url, drct):
    """Fetches vehicle states data from the given URL and stores it in the specified MinIO bucket."""

    page_num = 1
    while True:
        response = requests.get(url)
        response.raise_for_status()  # Raise an error if the request fails
        data = response.json()

        # Prepare the data to be uploaded to MinIO (as JSON string)
        json_data = json.dumps(data['results']).encode('utf-8')  # Convert to bytes

        # Use an in-memory bytes buffer to simulate a file
        data_stream = io.BytesIO(json_data)

        # Generate object name with the 'drct/' folder prefix
        object_name = f'{drct}/page_{page_num}.json'

        # Upload data to MinIO
        client.put_object(
            BUCKET_NAME,
            object_name,
            data_stream,
            length=len(json_data),
            content_type='application/json'
        )

        print(f'Uploaded {object_name} to bucket {BUCKET_NAME}.')

        # Check for the next page
        url = data.get('next')  # Set 'url' to the next page
        if not url:  # If there is no 'next' page
            print("No more pages to fetch, waiting for 5 minutes before checking again...")
            time.sleep(300)  # Sleep for 5 minutes (300 seconds)
            url = data.get('self', None)  # Retry the same URL or initial URL after sleep
            continue  # Continue checking after sleep

        page_num += 1

if __name__ == '__main__':
    # Fetch the data from the API and store it in the MinIO bucket
    fetch_and_store_vehicle_states()
```

Code for Reading and Writing Data Between MinIO and PostgreSQL

This part of the code reads data stored in MinIO and writes it into the PostgreSQL database after processing. It also handles cleaning up the data from MinIO after successful transfer.

```
import os
import json
import io
import pandas as pd
from minio import Minio
from sqlalchemy import create_engine

# MinIO Configuration from environment variables
MINIO_URL = os.getenv('MINIO_URL')
ACCESS_KEY = os.getenv('ACCESS_KEY')
SECRET_KEY = os.getenv('SECRET_KEY')
BUCKET_NAME = os.getenv('BUCKET_NAME')

# PostgreSQL Configuration from environment variables
POSTGRES_HOST = os.getenv('POSTGRES_HOST')
POSTGRES_DB = os.getenv('POSTGRES_DB')
POSTGRES_USER = os.getenv('POSTGRES_USER')
POSTGRES_PASSWORD = os.getenv('POSTGRES_PASSWORD')

# Initialize MinIO client
client = Minio(
```

```python
    MINIO_URL,
    access_key=ACCESS_KEY,
    secret_key=SECRET_KEY,
    secure=False  # Set to True if using HTTPS
)

# Initialize PostgreSQL engine
engine =
create_engine(f"postgresql://{POSTGRES_USER}:{POSTGRES_PASSWORD}@{POSTGRES_HOST}/{POSTGRES_DB}")

def read_minio_and_store_in_postgres(drct, table_name):
    """Reads data from MinIO and stores it in PostgreSQL, then deletes the data from MinIO."""

    # List objects in the MinIO bucket directory
    objects = client.list_objects(BUCKET_NAME, prefix=f"{drct}/", recursive=True)

    for obj in objects:
        # Read the object from MinIO
        response = client.get_object(BUCKET_NAME, obj.object_name)
        data = json.loads(response.read())
        response.close()
        response.release_conn()

        # Convert the data into a pandas DataFrame
        df = pd.DataFrame(data)

        # Store the data in PostgreSQL
        store_in_postgres(df, table_name)
        print(f"Data from {obj.object_name} has been stored in PostgreSQL.")

        # Delete the object from MinIO after storing it in PostgreSQL
        delete_from_minio(BUCKET_NAME, obj.object_name)

def store_in_postgres(df, table_name):
    """Stores the given DataFrame in the specified PostgreSQL table."""
    try:
        # Save DataFrame to PostgreSQL
        df.to_sql(table_name, engine, if_exists='append', index=False)
        print(f"Data stored in PostgreSQL table '{table_name}' successfully.")
    except Exception as e:
        print(f"Error storing data in PostgreSQL: {e}")

def delete_from_minio(bucket_name, object_name):
    """Deletes the object from MinIO."""
    try:
        client.remove_object(bucket_name, object_name)
        print(f"Deleted {object_name} from MinIO bucket {bucket_name}.")
    except Exception as e:
        print(f"Error deleting {object_name} from MinIO: {e}")

if __name__ == '__main__':
    # Reading data from MinIO and store it in Postgres, then delete from MinIO
    read_minio_and_store_in_postgres(drct="vehicle-data", table_name="vehicle_states")
```

Docker Compose for Running Eros4NRG Services

The Docker Compose file manages the various services and containers, including MinIO, PostgreSQL, Grafana, and the Python application for data storage and processing.

```yaml
services:
  minio:
    image: minio/minio
    container_name: minio
    ports:
      - "9000:9000"  # MinIO web interface
      - "9001:9001"  # MinIO admin console
    environment:
      MINIO_ROOT_USER: admin
      MINIO_ROOT_PASSWORD: eros4nrg
    volumes:
      - ./minio_data:/data  # Persist MinIO data locally
    command: server /data --console-address ":9001"
    networks:
      - app-network

  postgres:
    image: postgres:13
    container_name: postgres
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres_password
      POSTGRES_DB: emotion_data
    volumes:
      - ./postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    depends_on:
      - minio  # Ensure MinIO is running before PostgreSQL
    networks:
      - app-network

  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    ports:
      - "3000:3000"  # Grafana web interface
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=admin
    depends_on:
      - postgres  # Ensure PostgreSQL is running before Grafana
    networks:
      - app-network

  python-app:
    build: ./app  # Build the Python app from the 'app' folder
    container_name: python-app
    depends_on:
      - minio
      - postgres
    environment:
      MINIO_URL: minio:9000  # Service name of MinIO container
      ACCESS_KEY: admin
      SECRET_KEY: eros4nrg
      BUCKET_NAME: emotion-data
      POSTGRES_HOST: postgres
      POSTGRES_DB: emotion_data
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres_password
    volumes:
      - ./app:/usr/src/app  # Mount the app folder for live code changes
    networks:
      - app-network
    command: >
      sh -c "python iot_data_storage.py --url https://emotion-projects.eu/api/vehicle-states/?ordering=timestamp --drct vehicle-data
&
             python iot_data_storage.py --url https://emotion-projects.eu/api/tower-states/?ordering=timestamp  --drct tower-states"

networks:
  app-network:
    driver: bridge
```

Eros4NRG Data Catalogue GUI

This flask-based code set up the graphical user interface (GUI) for Eros4NRG to enable end user to query clean data from the the PostgresSQL database (the clean data bucket).

```
import pandas as pd
import data_catalogue.util as dc_util

from fastapi import FastAPI

app = FastAPI()


@app.get("/vehicles")
async def get_vehicles():
    return dc_util.get_all_vehicles()


@app.get("/vehicles/{vehicle_id}/charges")
async def get_vehicle_charges(vehicle_id: str):
    return dc_util.get_all_vehicle_charges(vehicle_id)


@app.get("/charges/{charge_id}")
async def get_vehicle_charges(charge_id: str):
    return dc_util.get_charge_information(charge_id)
```

Eros4NRG Dockerfile

This Dockerfile defines the build environment for the Python application responsible for storing IoT data, including the required Python packages and application code.

```
# Use an official Python image
FROM python:3.11-slim

# Set the working directory in the container
WORKDIR /usr/src/app

# Copy the requirements file into the container
COPY requirements.txt .

# Install the required packages
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code
COPY . .

# Set the command to run the application
CMD ["python", "./iot_data_storage.py"]
```