НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

Кафедра Обчислювальної техніки Факультет Інформатики та обчислювальної техніки

Звіт

до лабораторної роботи №1

з дисципліни

«Інтелектуальні вбудовані системи»

Виконав:

Студент IV курсу групи ІП-01 Черпак А.В.

Перевірив:

Нікольський С.С.

Оцінка:_____ Дата:_____

Київ 2024

Завдання: реалізувати програмну частину яка буде читати дані датчиків з файлу (до цього записаний csv file під назвою data.csv) та надсилати їх на Edge. Agent та Edge повинні комунікувати через MQTT.

Виконання

Для початку створимо проект згідно зі вказівками, наведеними у методичці. Отриманий проект матиме структуру, наведену на рисунку 1.

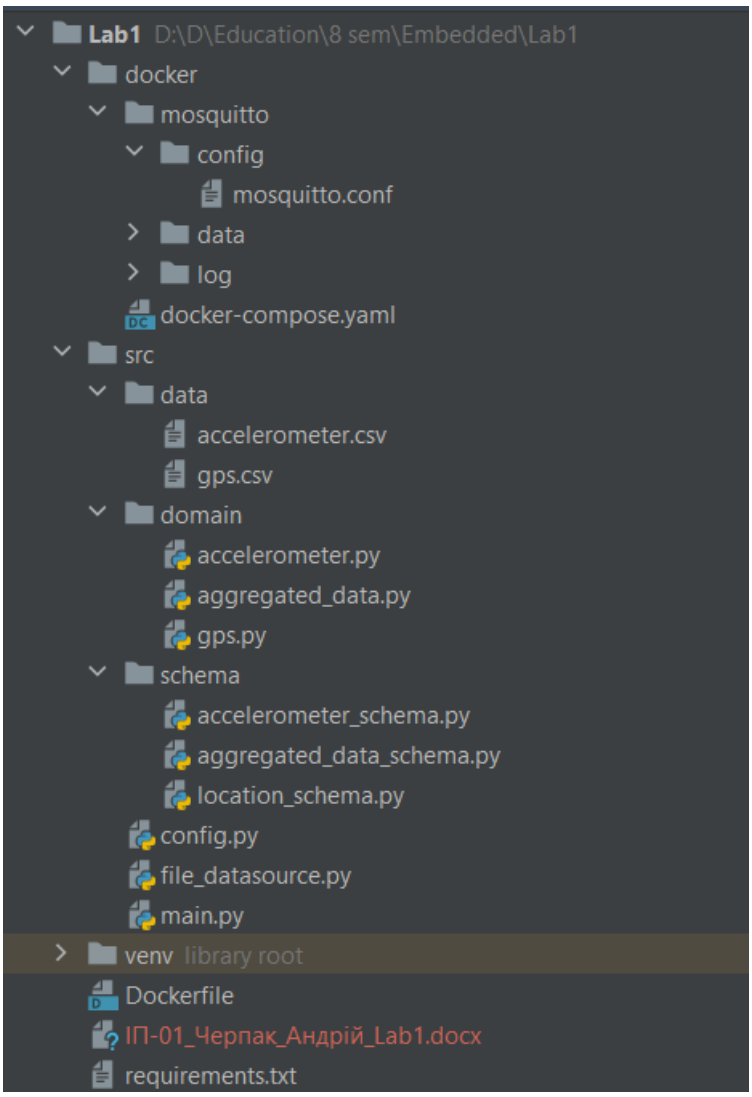

Рисунок 1 – Структура новоствореного проекту

Початкове наповнення файлів повністю відповідатиме коду, наведеному у методичці.

mosquitto.conf

```
persistence true
persistence_location /mosquitto/data/
listener 1883
## Authentication ##
allow_anonymous true
# allow_anonymous false
```

```
# password_file /mosquitto/config/password.txt
## Log ##
log_dest file /mosquitto/log/mosquitto.log
log_dest stdout
# listener 1883
```

docker-compose.yaml

```
version: "3.9"
name: "road_vision"
services:
  mqtt:
    image: eclipse-mosquitto
    container_name: mqtt
    volumes:
    - ./mosquitto:/mosquitto
    - ./mosquitto/data:/mosquitto/data
    - ./mosquitto/log:/mosquitto/log
    ports:
    - 1883:1883
    - 9001:9001
    networks:
      mqtt_network:

  fake_agent:
    container_name: agent
    build: ../
    depends_on:
      - mqtt
    environment:
      MQTT_BROKER_HOST: "mqtt"
      MQTT_BROKER_PORT: 1883
      MQTT_TOPIC: "agent_data_topic"
      DELAY: 0.1
    networks:
      mqtt_network:

networks:
  mqtt_network:
```

accelerometer.py

```
from dataclasses import dataclass


@dataclass
class Accelerometer:
    x: int
```

```
    y: int
    z: int
```

aggregated_data.py

```
from dataclasses import dataclass
from datetime import datetime
from domain.accelerometer import Accelerometer
from domain.gps import Gps

@dataclass
class AggregatedData:
    accelerometer: Accelerometer
    gps: Gps
    time: datetime
```

gps.py

```
from dataclasses import dataclass


@dataclass
class Gps:
    longitude: float
    latitude: float
from marshmallow import Schema, fields
from src.domain.accelerometer import Accelerometer
```

accelerometer_schema.py

```
class AccelerometerSchema(Schema):
    x = fields.Int()
    y = fields.Int()
    z = fields.Int()
```

aggregated_data_schema.py

```
from marshmallow import Schema, fields
from src.schema.accelerometer_schema import
AccelerometerSchema
from src.schema.location_schema import GpsSchema
from src.domain.aggregated_data import AggregatedData


class AggregatedDataSchema(Schema):
    accelerometer = fields.Nested(AccelerometerSchema)
    gps = fields.Nested(GpsSchema)
    time = fields.DateTime('iso')
```

location_schema.py

```
from marshmallow import Schema, fields
from src.domain.gps import Gps
```

```
class GpsSchema(Schema):
    longitude = fields.Number()
    latitude = fields.Number()
```

config.py

```
import os


def try_parse(type, value: str):
    try:
        return type(value)
    except Exception:
        return None

# MQTT config
MQTT_BROKER_HOST = os.environ.get('MQTT_BROKER_HOST') or
'mqtt'
MQTT_BROKER_PORT = try_parse(int,
os.environ.get('MQTT_BROKER_PORT')) or 1883
MQTT_TOPIC = os.environ.get('MQTT_TOPIC') or 'agent'
# Delay for sending data to mqtt in seconds
DELAY = try_parse(float, os.environ.get('DELAY')) or 1
```

file_datasource.py

```
from csv import reader
from datetime import datetime
from domain.aggregated_data import AggregatedData

class FileDatasource:
    def __init__(self, accelerometer_filename: str,
gps_filename: str) -> None:
        pass

    def read(self) -> AggregatedData:
        """Метод повертає дані отримані з датчиків"""

    def startReading(self, *args, **kwargs):
        """Метод повинен викликатись перед початком
читання даних"""

    def stopReading(self, *args, **kwargs):
        """Метод повинен викликатись для закінчення
читання даних"""
```

main.py

```
from paho.mqtt import client as mqtt_client
import json
```

```
import time
from schema.aggregated_data_schema import
AggregatedDataSchema
from file_datasource import FileDatasource
import config


def connect_mqtt(broker, port):
    """Create MQTT client"""
    print(f"CONNECT TO {broker}:{port}")

    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print(f"Connected to MQTT Broker
({broker}:{port})!")
        else:
            print(f"Failed to connect {broker}:{port},
return code %d\n", rc)
            exit(rc) # Stop execution

    client = mqtt_client.Client()
    client.on_connect = on_connect
    client.connect(broker, port)
    client.loop_start()
    return client


def publish(client, topic, datasource, delay):
    datasource.startReading()
    while True:
        time.sleep(delay)
        data = datasource.read()
        msg = AggregatedDataSchema().dumps(data)
        result = client.publish(topic, msg)
        # result: [0, 1]
        status = result[0]
        if status == 0:
            pass
            # print(f"Send `{msg}` to topic `{topic}`")
        else:
            print(f"Failed to send message to topic
{topic}")


def run():
    # Prepare mqtt client
    client = connect_mqtt(config.MQTT_BROKER_HOST,
```

```
config.MQTT_BROKER_PORT)
    # Prepare datasource
    datasource = FileDatasource("data/accelerometer.csv",
"data/gps.csv")
    # Infinity publish data
    publish(client, config.MQTT_TOPIC, datasource,
config.DELAY)


if __name__ == '__main__':
    run()
```

Dockerfile

```
# set base image (host OS)
FROM python:latest
# set the working directory in the container
WORKDIR /usr/agent
# copy the dependencies file to the working directory
COPY requirements.txt .
# install dependencies
RUN pip install -r requirements.txt
# copy the content of the local src directory to the
working directory
COPY src/ .
# command to run on container start
CMD ["python", "main.py"]
```

requirements.txt

```
marshmallow==3.20.2
packaging==23.2
paho-mqtt==1.6.1
```

Після цього необхідно було реалізувати основну логіку читання даних з файлу. Для цього модифікуємо файл file_datasource.py:

```
from csv import reader
from datetime import datetime
from typing import TextIO

from domain.aggregated_data import AggregatedData
import config
from domain.accelerometer import Accelerometer
from domain.gps import Gps


class FileDatasource:
    accelerometer_file: TextIO
    gps_file: TextIO
```

```
    accelerometer_filename: str
    gps_filename: str

    def __init__(self, accelerometer_filename: str,
gps_filename: str) -> None:
        self.accelerometer_filename =
accelerometer_filename
        self.gps_filename = gps_filename

    def read(self) -> AggregatedData:
        """Метод повертає дані отримані з датчиків"""
        self.accelerometer_file, acc_data =
FileDatasource.__get_next_line(self.accelerometer_file,
int)
        self.gps_file, gps_data =
FileDatasource.__get_next_line(self.gps_file, float)

        return AggregatedData(
            Accelerometer(acc_data[0], acc_data[1],
acc_data[2]),
            Gps(gps_data[0], gps_data[1]),
            datetime.now(),
            config.USER_ID,
        )

    def startReading(self, *args, **kwargs):
        """Метод повинен викликатись перед початком
читання даних"""
        self.accelerometer_file =
open(self.accelerometer_filename, 'r')
        self.gps_file = open(self.gps_filename, 'r')

        # skip header
        self.accelerometer_file.readline()
        self.gps_file.readline()

    def stopReading(self, *args, **kwargs):
        """Метод повинен викликатись для закінчення
читання даних"""
        self.accelerometer_file.close()
        self.gps_file.close()

    @staticmethod
    def __get_next_line(file: TextIO, datatype: type) ->
tuple[TextIO, list[float|int]]:
        line = file.readline()
        if not line or len(line) == 0:
```

```
            fl_name = file.name
            file.close()
            file = open(fl_name, 'r')

            # skip header
            file.readline()
            line = file.readline()

        return file, [datatype(num) for num in
line.split(',')]
```

Форматування коду у word просто жахливе, тому краще відслідковувати зміни по комітам у репозиторії https://github.com/CherpakAndrii/Embedded_Lab1

На цьому етапі перше завдання вже виконане. Піднімемо докер-контекнер та перевіримо результати. Команди для розгортання контейнеру зображено на рисунку 2, а на рисунках 3 та 4 – під'єднання з допомогою MQTT Explorer та перевірку результатів відповідно.

```
(venv) PS D:\D\Education\8 sem\Embedded\Lab1> cd docker
(venv) PS D:\D\Education\8 sem\Embedded\Lab1\docker> docker-compose up --build
[+] Building 2.2s (10/10) FINISHED
 => [fake_agent internal] load .dockerignore
 => => transferring context: 2B
 => [fake_agent internal] load build definition from Dockerfile
 => [fake_agent internal] load build context
 => => transferring context: 5.90kB
 => [fake_agent 1/5] FROM docker.io/library/python:latest@sha256:e83d1f4d0c735c7a54fc9dae3cca8c58473e3b3de08fcb7ba3d342ee75cfc09d
 => CACHED [fake_agent 2/5] WORKDIR /usr/agent
 => CACHED [fake_agent 3/5] COPY requirements.txt .
 => CACHED [fake_agent 4/5] RUN pip install -r requirements.txt
 => CACHED [fake_agent 5/5] COPY src/ .
 => [fake_agent] exporting to image
 => => exporting layers
 => => writing image sha256:f93399f03a1f717dd4297575623d47d6af79a6d95058014153908a18f611789c
 => => naming to docker.io/library/road_vision-fake_agent
[+] Running 2/2
 ✓ Container mqtt   Created
 ✓ Container agent  Recreated
Attaching to agent, mqtt
mqtt   | 1708910078: mosquitto version 2.0.18 starting
mqtt   | 1708910078: Config loaded from /mosquitto/config/mosquitto.conf.
mqtt   | 1708910078: Opening ipv4 listen socket on port 1883.
mqtt   | 1708910078: Opening ipv6 listen socket on port 1883.
mqtt   | 1708910078: mosquitto version 2.0.18 running
mqtt   | 1708910079: New connection from 172.20.0.3:47113 on port 1883.
mqtt   | 1708910079: New client connected from 172.20.0.3:47113 as auto-1A18438B-AC84-ED77-4F6A-A3BB88643BD0 (p2, c1, k60).
```

Рисунок 2 – Підняття docker-контейнера з консолі

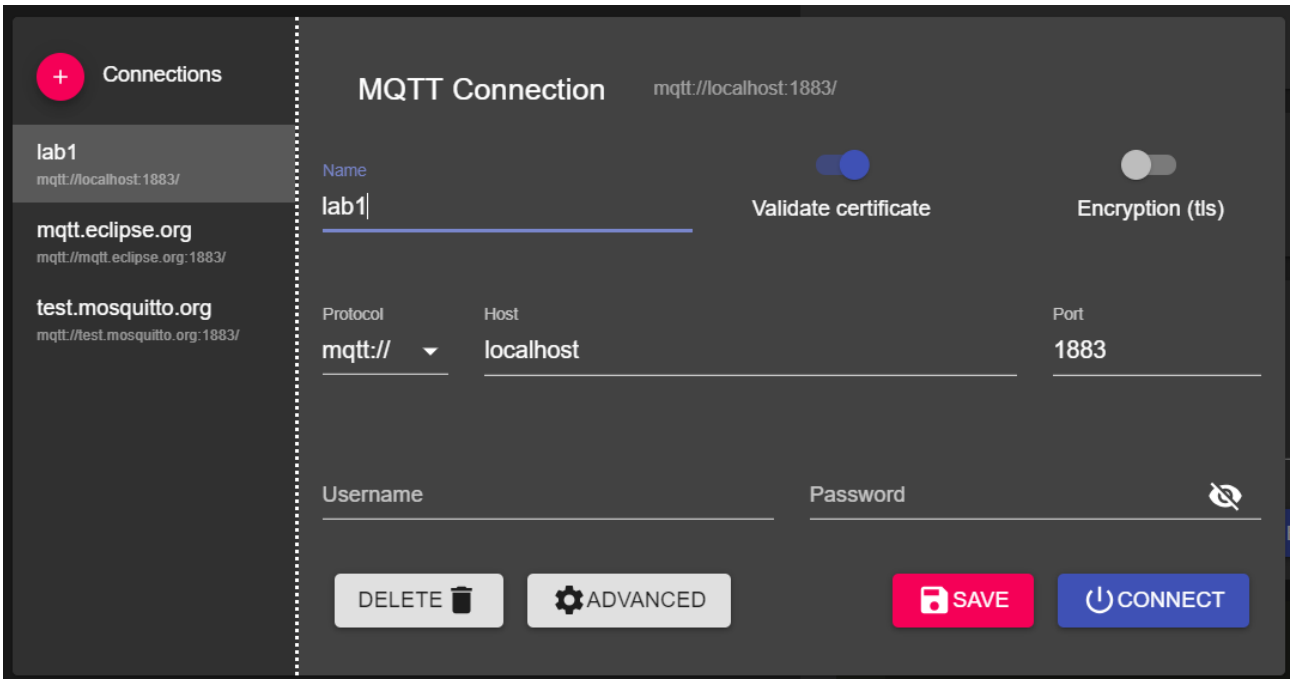

Рисунок 3 – Під’єднання з допомогою MQTT Explorer

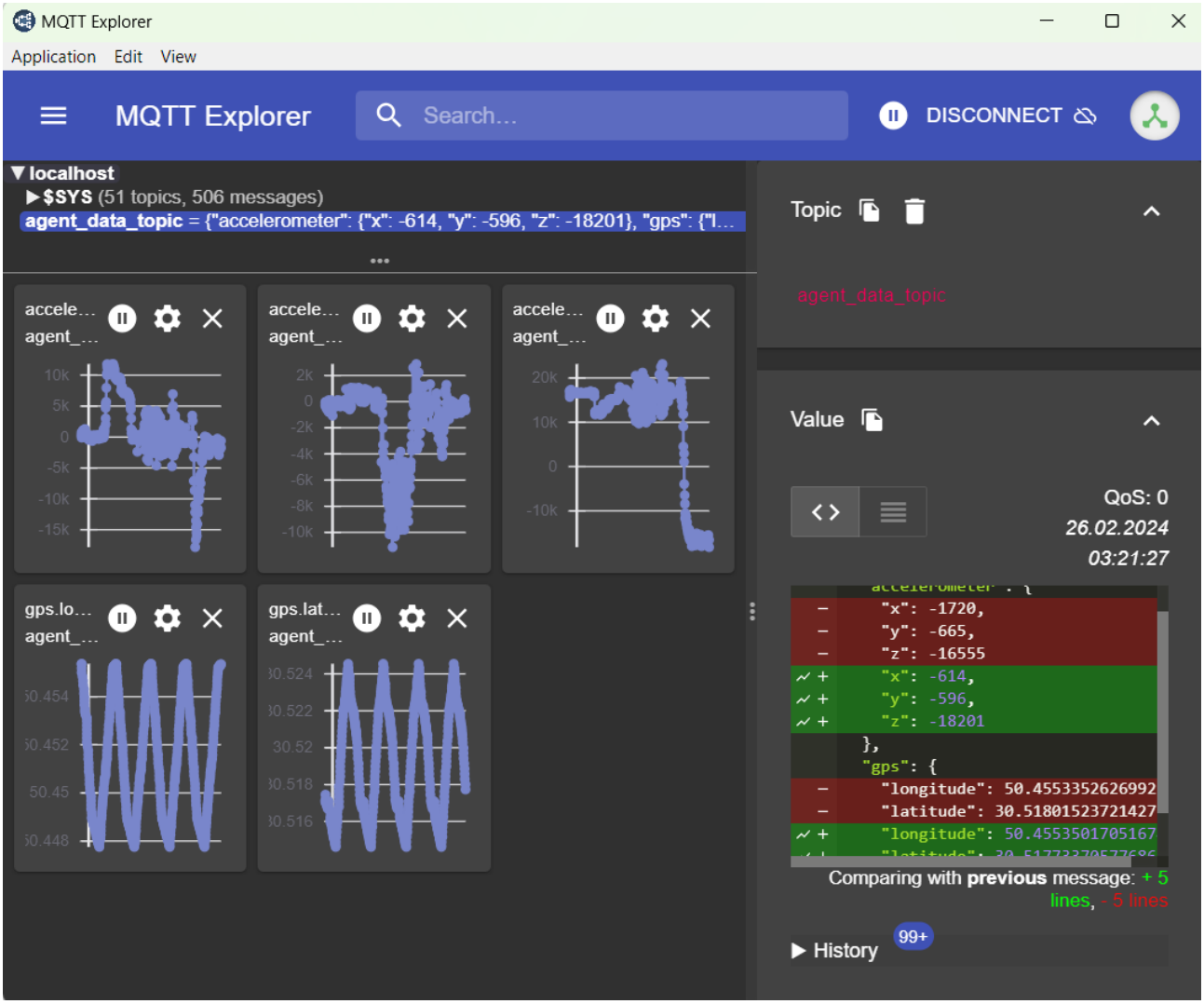

Рисунок 4 – Відслідковування надісланих даних у MQTT Explorer

Тепер перейдемо до наступної частини – додавання нового датчика «parking». Для цього додамо новий файл з даними parking.csv та класи parking.py і parking_schema.py. Нова структура проекту зображена на рисунку 5.

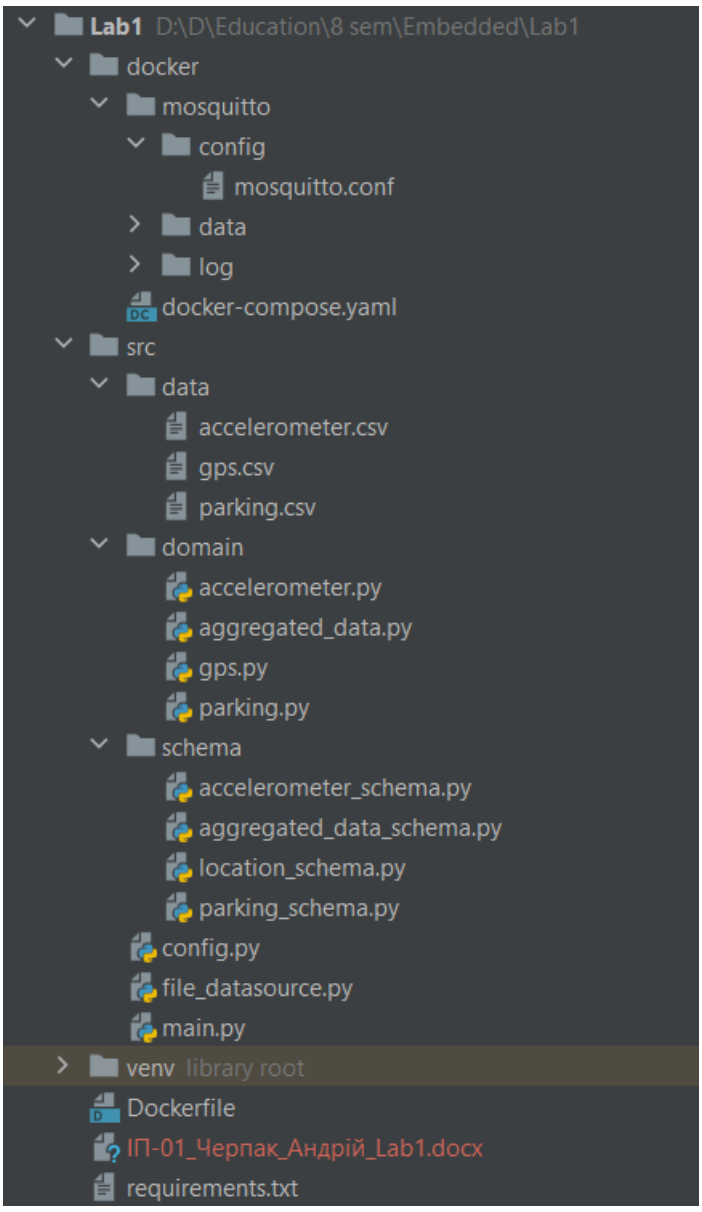

Рисунок 5 – Оновлена структура проекту

Тепер наведемо лістинг коду нових класів:

parking.py

```
from domain.gps import Gps
from dataclasses import dataclass


@dataclass
class Parking:
    empty_count: int
    gps: Gps
```

parking_schema.py

```
from schema.location_schema import GpsSchema
from marshmallow import Schema, fields


class ParkingSchema(Schema):
    empty_count = fields.Number()
    gps = fields.Nested(GpsSchema)
```

Також доведеться модифікувати такі файли, як aggregated_data.py, aggregated_data_schema.py, file_datasource.py та main.py. У перші два необхідно додати нове поле, яке представлятиме дані з нового датчика, у main.py варто буде просто передати у DataSource ще одну назву файлу, а у file_datasource.py – продублювати усю ту ж логіку для новоствореного файлу. Наведемо лістинг коду і цих класів:

aggregated_data.py

```
from dataclasses import dataclass
from datetime import datetime

from domain.accelerometer import Accelerometer
from domain.gps import Gps
from domain.parking import Parking


@dataclass
class AggregatedData:
    accelerometer: Accelerometer
    gps: Gps
    parking: Parking
    timestamp: datetime
    user_id: int
```

aggregated_data_schema.py

```
from marshmallow import Schema, fields

from schema.accelerometer_schema import
AccelerometerSchema
from schema.location_schema import GpsSchema
from schema.parking_schema import ParkingSchema
from domain.aggregated_data import AggregatedData


class AggregatedDataSchema(Schema):
    accelerometer = fields.Nested(AccelerometerSchema)
    gps = fields.Nested(GpsSchema)
```

```
    parking = fields.Nested(ParkingSchema)
    time = fields.DateTime('iso')
```

main.py

```
from paho.mqtt import client as mqtt_client
import json
import time
from schema.aggregated_data_schema import
AggregatedDataSchema
from file_datasource import FileDatasource
import config


def connect_mqtt(broker, port):
    """Create MQTT client"""
    print(f"CONNECT TO {broker}:{port}")

    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print(f"Connected to MQTT Broker
({broker}:{port})!")
        else:
            print(f"Failed to connect {broker}:{port},
return code %d\n", rc)
            exit(rc) # Stop execution

    client = mqtt_client.Client()
    client.on_connect = on_connect
    client.connect(broker, port)
    client.loop_start()
    return client


def publish(client, topic, datasource, delay):
    datasource.startReading()
    while True:
        time.sleep(delay)
        data = datasource.read()
        msg = AggregatedDataSchema().dumps(data)
        result = client.publish(topic, msg)
        # result: [0, 1]
        status = result[0]
        if status == 0:
            pass
            # print(f"Send `{msg}` to topic `{topic}`")
        else:
            print(f"Failed to send message to topic
```

```
{topic}")


def run():
    # Prepare mqtt client
    client = connect_mqtt(config.MQTT_BROKER_HOST,
config.MQTT_BROKER_PORT)
    # Prepare datasource
    datasource = FileDatasource("data/accelerometer.csv",
"data/gps.csv", "data/parking.csv")
    # Infinity publish data
    publish(client, config.MQTT_TOPIC, datasource,
config.DELAY)


if __name__ == '__main__':
    run()
```

file_datasource.py

```
from datetime import datetime
from typing import TextIO

import config
from domain.aggregated_data import AggregatedData
from domain.accelerometer import Accelerometer
from domain.gps import Gps
from domain.parking import Parking


class FileDatasource:
    accelerometer_file: TextIO
    gps_file: TextIO
    parking_file: TextIO

    accelerometer_filename: str
    gps_filename: str
    parking_filename: str

    def __init__(self, accelerometer_filename: str,
gps_filename: str, parking_filename: str) -> None:
        self.accelerometer_filename =
accelerometer_filename
        self.gps_filename = gps_filename
        self.parking_filename = parking_filename

    def read(self) -> AggregatedData:
        """Метод повертає дані отримані з датчиків"""
```

```
        self.accelerometer_file, acc_data =
FileDatasource.__get_next_line(self.accelerometer_file,
int)
        self.gps_file, gps_data =
FileDatasource.__get_next_line(self.gps_file, float)
        self.parking_file, parking_data =
FileDatasource.__get_next_line(self.parking_file, float)

        return AggregatedData(
            Accelerometer(acc_data[0], acc_data[1],
acc_data[2]),
            Gps(gps_data[0], gps_data[1]),
            Parking(int(parking_data[0]),
Gps(parking_data[1], parking_data[2])),
            datetime.now(),
            config.USER_ID,
        )

    def startReading(self, *args, **kwargs):
        """Метод повинен викликатись перед початком
читання даних"""
        self.accelerometer_file =
open(self.accelerometer_filename, 'r')
        self.gps_file = open(self.gps_filename, 'r')
        self.parking_file = open(self.parking_filename,
'r')

        # skip header
        self.accelerometer_file.readline()
        self.gps_file.readline()
        self.parking_file.readline()

    def stopReading(self, *args, **kwargs):
        """Метод повинен викликатись для закінчення
читання даних"""
        self.accelerometer_file.close()
        self.gps_file.close()
        self.parking_file.close()

    @staticmethod
    def __get_next_line(file: TextIO, datatype: type) ->
tuple[TextIO, list[float|int]]:
        line = file.readline()
        if not line or len(line) == 0:
            fl_name = file.name
            file.close()
            file = open(fl_name, 'r')
```

```
            # skip header
            file.readline()
            line = file.readline()

        return file, [datatype(num) for num in
line.split(',')]
```

Чудово, тепер можна знову піднімати докер-контейнер з використанням тієї ж команди і перевіряти результати. Оновлені результати роботи зображено на рисунку 6.

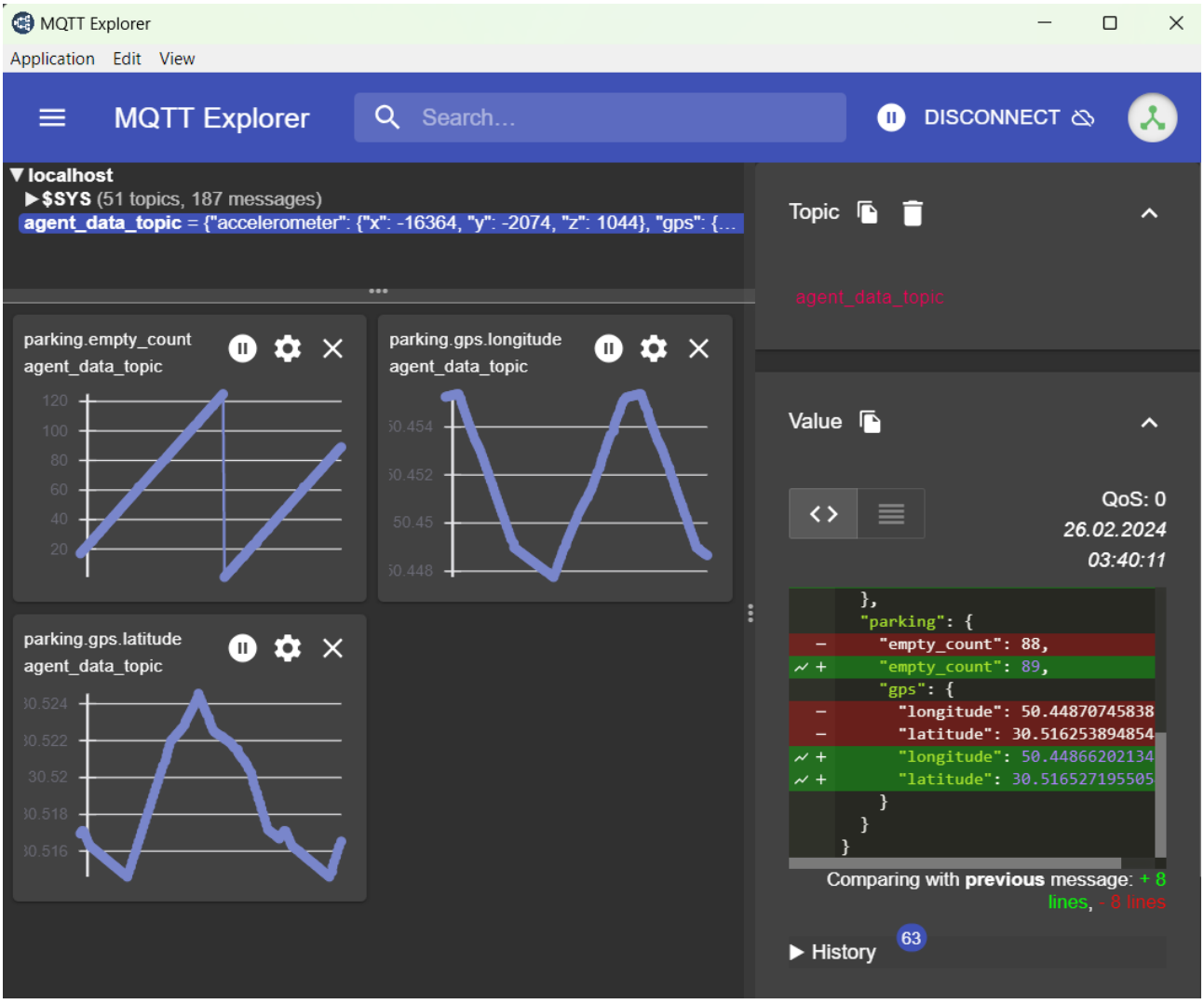

Рисунок 6 – Відслідковування надсилання оновлених даних.

Як бачимо, окрім показів акселерометра та gps тепер Edge модуль отримує і дані з нового сенсора – parking. Отже, новий сенсор успішно додано.

Задля підвищення оцінки було запропоновано реалізувати нескінченний цикл читання з файлу. У мене це було реалізовано ще у початковому варіанті всередині статичного методу __get_next_line(file, datatype) класу FileDatasource. Під час зчитування нової стрічки відбувається перевірка на закінчення файлу (у такому разі стрічка буде порожньою. Якщо файл закінчився – його буде

закрито, а потім відкрито заново, після чого пропущено стрічку з назвами колонок та зчитано наступний рядок. Після цього рядок csv-файлу розділяється на елементи, які приводяться до необхідного типу. В кінці кінців функція повертає файл (або той же, або перевідкритий) та зчитані з нього дані. Ще раз наведемо лістинг цього методу:

```
@staticmethod
def __get_next_line(file: TextIO, datatype: type) -> tuple[TextIO,
list[float|int]]:
    line = file.readline()
    if not line or len(line) == 0:
        fl_name = file.name
        file.close()
        file = open(fl_name, 'r')

        # skip header
        file.readline()
        line = file.readline()

    return file, [datatype(num) for num in line.split(',')]
```

Все, тепер лабораторна робота остаточно виконана :)

Висновок: підчас виконання комп’ютерного практикуму ми ознайомилися з загальною архітектурою системи та структурою проекту fake-agent. Також було отримано та покращено код цього проекту, зокрема, реалізовано клас для читання даних, а також додано новий сенсор. Окрім цього, ми навчилися розгортати окремі компоненти даної мережі з використанням інструменту Docker, а також під’єднуватися до системи з допомогою MQTT Explorer та переглядати передані дані.