

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

Кафедра Обчислювальної техніки Факультет Інформатики та обчислювальної
техніки

Звіт

до лабораторної роботи №3

з дисципліни

«Інтелектуальні вбудовані системи»

Виконав:

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

Перевірив:

Нікольський С.С.

Оцінка: _____ Дата: _____

Київ 2024

Завдання: Потрібно реалізувати частину Hub. Сервіс який займається накопиченням отриманих даних від Agent/EdgeDataLogic та подальшим зберіганням оброблених даних в БД. Перед збереженням даних потрібно накопичити (наприклад 10 записів), так як такі дані простіше аналізувати і запис в БД множини даних буде швидшим..

Виконання

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

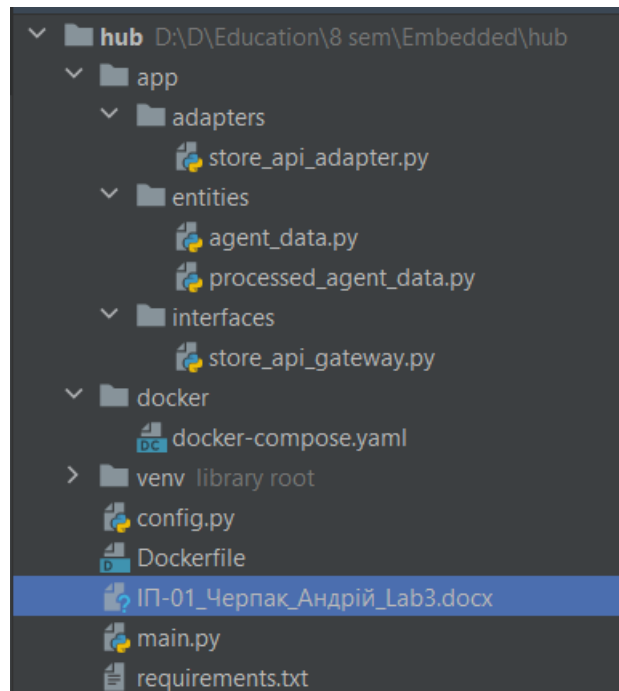


Рисунок 1 – Структура новоствореного проекту

Початкове наповнення файлів повністю відповідатиме коду, наведеному у методичці.

main.py

```
import logging
from typing import List
from fastapi import FastAPI
from redis import Redis
import paho.mqtt.client as mqtt
from app.adapters.store_api_adapter import StoreApiAdapter
from app.entities.processed_agent_data import ProcessedAgentData
from config import STORE_API_BASE_URL, REDIS_HOST, REDIS_PORT, BATCH_SIZE, MQTT_TOPIC, MQTT_BROKER_HOST, MQTT_BROKER_PORT
```

```

# Configure logging settings
logging.basicConfig(
    level=logging.INFO, # Set the log level to INFO (you
    can use logging.DEBUG for more detailed logs)
    format="[%asctime)s] [%levelname)s] [%module)s]
    %(message)s",
    handlers=[
        logging.StreamHandler(), # Output log messages to
the console
        logging.FileHandler("app.log"), # Save log
messages to a file
    ],
)

# Create an instance of the Redis using the configuration
redis_client = Redis(host=REDIS_HOST, port=REDIS_PORT)
# Create an instance of the StoreApiAdapter using the
configuration
store_adapter =
StoreApiAdapter(api_base_url=STORE_API_BASE_URL)
# Create an instance of the AgentMQTTAdapter using the
configuration

# FastAPI
app = FastAPI()

@app.post("/processed_agent_data/")
async def save_processed_agent_data(processed_agent_data:
ProcessedAgentData):
    print(processed_agent_data)
    redis_client.lpush(
        "processed_agent_data",
        processed_agent_data.model_dump_json()
    )
    processed_agent_data_batch: List[ProcessedAgentData]
= []

    if redis_client.llen("processed_agent_data") >=
BATCH_SIZE:
        for _ in range(BATCH_SIZE):
            processed_agent_data =
ProcessedAgentData.model_validate_json(
                redis_client.lpop("processed_agent_data")
            )
            processed_agent_data_batch.append(processed_agent_data)

```

```

store_adapter.save_data(processed_agent_data_batch=processed_agent_data_batch)

    return {"status": "ok"}

# MQTT
client = mqtt.Client()

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        logging.info("Connected to MQTT broker")
        client.subscribe(MQTT_TOPIC)
    else:
        logging.info(f"Failed to connect to MQTT broker with code: {rc}")

def on_message(client, userdata, msg):
    batch_data = None
    try:
        payload: str = msg.payload.decode("utf-8")
        # Create ProcessedAgentData instance with the received data
        processed_agent_data = ProcessedAgentData.model_validate_json(payload, strict=True)
        redis_client.lpush(
            "processed_agent_data",
            processed_agent_data.model_dump_json()
        )

        processed_agent_data_batch: List[ProcessedAgentData] = []
        if redis_client.llen("processed_agent_data") >= BATCH_SIZE:
            for _ in range(BATCH_SIZE):
                processed_agent_data = ProcessedAgentData.model_validate_json(
                    redis_client.lpop("processed_agent_data")
                )
            processed_agent_data_batch.append(processed_agent_data)

        store_adapter.save_data(processed_agent_data_batch=processed_agent_data_batch)

```

```

        return {"status": "ok"}
    except Exception as e:
        logging.info(f"Error processing MQTT message:
{e}")

# Connect
client.on_connect = on_connect
client.on_message = on_message
client.connect(MQTT_BROKER_HOST, MQTT_BROKER_PORT)

# Start
client.loop_start()

```

config.py

```

import os

def try_parse_int(value: str):
    try:
        return int(value)
    except Exception:
        return None

# Configuration for the Store API
STORE_API_HOST = os.environ.get("STORE_API_HOST") or
"localhost"
STORE_API_PORT =
try_parse_int(os.environ.get("STORE_API_PORT")) or 8000
STORE_API_BASE_URL =
f"http://{STORE_API_HOST}:{STORE_API_PORT}"
# Configure for Redis
REDIS_HOST = os.environ.get("REDIS_HOST") or "localhost"
REDIS_PORT = try_parse_int(os.environ.get("REDIS_PORT"))
or 6379
# Configure for hub logic
BATCH_SIZE = try_parse_int(os.environ.get("BATCH_SIZE"))
or 20
# MQTT
MQTT_BROKER_HOST = os.environ.get("MQTT_BROKER_HOST") or
"localhost"
MQTT_BROKER_PORT =
try_parse_int(os.environ.get("MQTT_BROKER_PORT")) or 1883

```

```
MQTT_TOPIC = os.environ.get("MQTT_TOPIC") or  
"processed_agent_data_topic"
```

store_api_adapter.py

```
import json  
import logging  
from typing import List  
  
import pydantic_core  
import requests  
  
from app.entities.processed_agent_data import  
ProcessedAgentData  
from app.interfaces.store_api_gateway import StoreGateway  
  
class StoreApiAdapter(StoreGateway):  
    def __init__(self, api_base_url):  
        self.api_base_url = api_base_url  
  
    def save_data(self, processed_agent_data_batch:  
List[ProcessedAgentData]):  
        # Make a POST request to the Store API endpoint  
        with the processed data  
        pass
```

agent_data.py

```
from datetime import datetime  
from pydantic import BaseModel, field_validator  
  
class AccelerometerData(BaseModel):  
    x: float  
    y: float  
    z: float  
  
class GpsData(BaseModel):  
    latitude: float  
    longitude: float  
  
class AgentData(BaseModel):  
    accelerometer: AccelerometerData  
    gps: GpsData
```

```

timestamp: datetime

@classmethod
@field_validator('timestamp', mode='before')
def parse_timestamp(cls, value):
    # Convert the timestamp to a datetime object
    if isinstance(value, datetime):
        return value
    try:
        return datetime.fromisoformat(value)
    except (TypeError, ValueError):
        raise ValueError("Invalid timestamp format.
Expected ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ).")

```

processed_agent_data.py

```

from pydantic import BaseModel
from app.entities.agent_data import AgentData

class ProcessedAgentData(BaseModel):
    road_state: str
    agent_data: AgentData

    def serialize(self) -> dict:
        return {
            "road_state": self.road_state,
            "agent_data": {
                "user_id": 0,
                "accelerometer": {
                    "x": self.agent_data.accelerometer.x,
                    "y": self.agent_data.accelerometer.y,
                    "z": self.agent_data.accelerometer.z
                },
                "gps": {
                    "latitude":
self.agent_data.gps.latitude,
                    "longitude":
self.agent_data.gps.longitude
                },
                "timestamp":
str(self.agent_data.timestamp)
            }
        }

```

store_api_gateway.py

```
from abc import ABC, abstractmethod
from typing import List
from app.entities.processed_agent_data import
ProcessedAgentData

class StoreGateway(ABC):
    """
    Abstract class representing the Store Gateway
    interface.
    All store gateway adapters must implement these
    methods.
    """

    @abstractmethod
    def save_data(self, processed_agent_data_batch:
List[ProcessedAgentData]) -> bool:
        """
        Method to save the processed agent data in the
        database.
        Parameters:
        processed_agent_data_batch (ProcessedAgentData):
        The processed
        agent data to be saved.
        Returns:
        bool: True if the data is successfully saved,
        False otherwise.
        """
        pass
```

docker-compose.yaml

```
version: "3.9"
name: "road_vision__hub"
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:

postgres_db:
  image: postgres:latest
  container_name: postgres_db
  restart: always
  environment:
    POSTGRES_USER: user
    POSTGRES_PASSWORD: pass
    POSTGRES_DB: test_db
  volumes:
    - postgres_data:/var/lib/postgresql/data
    - ./db/structure.sql:/docker-entrypoint-initdb.d/structure.sql
  ports:
    - "5432:5432"
  networks:
    db_network:

pgadmin:
  container_name: pgadmin4
  image: dpage/pgadmin4
  restart: always
  environment:
    PGADMIN_DEFAULT_EMAIL: admin@admin.com
    PGADMIN_DEFAULT_PASSWORD: root
  volumes:
    - pgadmin-data:/var/lib/pgadmin
  ports:
    - "5050:80"
  networks:
    db_network:

store:
  container_name: store
  build: ../../store
  depends_on:
    - postgres_db
  restart: always
  environment:
    POSTGRES_USER: user
    POSTGRES_PASSWORD: pass
    POSTGRES_DB: test_db
    POSTGRES_HOST: postgres_db
    POSTGRES_PORT: 5432
  ports:
```

```
    - "8000:8000"
  networks:
    db_network:
    hub_store:

redis:
  image: redis:latest
  container_name: redis
  ports:
    - "6379:6379"
  networks:
    hub_redis:

hub:
  container_name: hub
  build: ../
  depends_on:
    - mqtt
    - redis
    - store
  environment:
    STORE_API_HOST: "store"
    STORE_API_PORT: 8000
    REDIS_HOST: "redis"
    REDIS_PORT: 6379
    MQTT_BROKER_HOST: "mqtt"
    MQTT_BROKER_PORT: 1883
    MQTT_TOPIC: "processed_data_topic"
    BATCH_SIZE: 1
  ports:
    - "9000:8000"
  networks:
    mqtt_network:
    hub_store:
    hub_redis:

networks:
  mqtt_network:
  db_network:
  hub_store:
  hub_redis:

volumes:
  postgres_data:
  pgadmin-data:
```

requirements.txt

```
annotated-types==0.6.0
anyio==4.3.0
async-timeout==4.0.3
certifi==2024.2.2
charset-normalizer==3.3.2
click==8.1.7
colorama==0.4.6
fastapi==0.110.0
h11==0.14.0
httpx==0.27.0
httptools==0.6.1
idna==3.6
paho-mqtt==1.6.1
pydantic==2.6.3
pydantic_core==2.16.3
python-dotenv==1.0.1
PyYAML==6.0.1
redis==5.0.2
requests==2.31.0
sniffio==1.3.1
starlette==0.36.3
typing_extensions==4.10.0
urllib3==2.2.1
uvicorn==0.27.1
watchfiles==0.21.0
websockets==12.0
```

Після цього необхідно було імплементувати запит до StoreApi для збереження даних. А саме зробити post запит на ‘{store_host}/processed_agent_data’ з списком елементів ProcessedAgentData.

store_api_adapter.py

```
import json
import logging
from typing import List

import requests

from app.entities.processed_agent_data import ProcessedAgentData
from app.interfaces.store_api_gateway import StoreGateway

class StoreApiAdapter(StoreGateway):
    api_base_url: str

    def __init__(self, api_base_url):
```

```

        self.api_base_url = api_base_url

    def save_data(self, processed_agent_data_batch:
List[ProcessedAgentData]) -> bool:
        """
        Save the processed road data to the Store API.
        Parameters:
            processed_agent_data_batch (dict): Processed
road data to be saved.
        Returns:
            bool: True if the data is successfully saved,
False otherwise.
        """
        # Make a POST request to the Store API endpoint
with the processed data
        try:
            sent_data = [processed_agent_data.serialize()
for processed_agent_data in processed_agent_data_batch]
            response =
requests.post(self.api_base_url+'/processed_agent_data/',
                data=json.dumps(sent_data))
            if response.status_code in (200, 201):
                logging.info("Data sent to Store API")
                return True
            else:
                logging.info("Sending data failed with
status code" + str(response.status_code))
                return False
        except requests.exceptions.RequestException as e:
            logging.info("Request failed with error: " +
str(e))
            return False

```

Форматування коду у word просто жахливе, тому краще відслідковувати зміни по комітам у репозиторії:

<https://github.com/CherpakAndrii/IntellectualEmbeddedSystems/tree/lab-3/hub>

На цьому етапі лабораторну роботу можна вважати виконаною. Піднімемо докер-контекнер та перевіримо результати. Команди для розгортання контейнеру зображено на рисунках 3-6.

```

(venv) PS D:\D\Education\8 sem\Embedded\hub> cd docker
(venv) PS D:\D\Education\8 sem\Embedded\hub\docker> docker-compose up --build
[+] Building 2.4s (10/10)
[+] Building 6.3s (17/17) FINISHED
=> [store internal] load .dockerignore
=> => transferring context: 2B
=> [store internal] load build definition from Dockerfile
=> => transferring dockerfile: 482B
=> [hub internal] load metadata for docker.io/library/python:latest
=> [hub 1/5] FROM docker.io/library/python:latest@sha256:e83d1f4d0c735c7a54fc9dae3cca8c58473e3b3de08fcb7ba3d342ee75cfc09d
=> [store internal] load build context
=> => transferring context: 289.84kB
=> CACHED [hub 2/5] WORKDIR /app
=> CACHED [store 3/5] COPY requirements.txt .
=> CACHED [store 4/5] RUN pip install --no-cache-dir -r requirements.txt
=> CACHED [store 5/5] COPY . .
=> [store] exporting to image
=> => exporting layers
=> => writing image sha256:61a5ac7859fe75910b204875e178e325a1ef3d55179744251a1ad7a0b2d440dc
=> => naming to docker.io/library/road_vision__hub-store
=> [hub internal] load .dockerignore
=> => transferring context: 2B
=> [hub internal] load build definition from Dockerfile
=> => transferring dockerfile: 480B
=> [hub internal] load build context
=> => transferring context: 259.31kB
=> CACHED [hub 3/5] COPY requirements.txt .
=> CACHED [hub 4/5] RUN pip install --no-cache-dir -r requirements.txt
=> [hub 5/5] COPY . .

```

```

=> [hub] exporting to image
=> => exporting layers
=> => writing image sha256:47813c05a32714562fe1df36609c08d73f46a91e6d354acc94ae98f86f5078bf
=> => naming to docker.io/library/road_vision__hub-hub
[+] Running 12/12
✓ Network road_vision__hub_mqtt_network      Created
✓ Network road_vision__hub_hub_store          Created
✓ Network road_vision__hub_hub_redis          Created
✓ Network road_vision__hub_db_network         Created
✓ Volume "road_vision__hub_postgres_data"     Created
✓ Volume "road_vision__hub_pgadmin-data"      Created
✓ Container pgadmin4                          Created

```

```

✓ Container pgadmin4          Created
✓ Container mqtt              Created
✓ Container postgres_db       Created
✓ Container redis              Created
✓ Container store              Created
✓ Container hub                Created
Attaching to hub, mqtt, pgadmin4, postgres_db, redis, store
redis | 1:C 29 Feb 2024 17:50:35.186 # WARNING Memory overcommit must be enabled! Without it, a background save or
d, it can also cause failures without low memory condition, see https://github.com/jemalloc/jemalloc/issues/1328. To fix
nd then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
redis | 1:C 29 Feb 2024 17:50:35.186 * o000o000o000o Redis is starting o000o000o000o
redis | 1:C 29 Feb 2024 17:50:35.186 * Redis version=7.2.4, bits=64, commit=00000000, modified=0, pid=1, just star
redis | 1:C 29 Feb 2024 17:50:35.186 # Warning: no config file specified, using the default config. In order to sp
redis | 1:M 29 Feb 2024 17:50:35.187 * monotonic clock: POSIX clock_gettime
redis | 1:M 29 Feb 2024 17:50:35.188 * Running mode=standalone, port=6379.
redis | 1:M 29 Feb 2024 17:50:35.188 * Server initialized
redis | 1:M 29 Feb 2024 17:50:35.188 * Ready to accept connections tcp
mqtt | 1709229035: mosquitto version 2.0.18 starting
mqtt | 1709229035: Config loaded from /mosquitto/config/mosquitto.conf.
mqtt | 1709229035: Opening ipv4 listen socket on port 1883.
mqtt | 1709229035: Opening ipv6 listen socket on port 1883.
postgres_db | The files belonging to this database system will be owned by user "postgres".
postgres_db | This user must also own the server process.
postgres_db |
postgres_db | The database cluster will be initialized with locale "en_US.utf8".

postgres_db | The default database encoding has accordingly been set to "UTF8".
mqtt | 1709229035: mosquitto version 2.0.18 running
postgres_db | The default text search configuration will be set to "english".
postgres_db |
postgres_db | Data page checksums are disabled.
postgres_db |
postgres_db | fixing permissions on existing directory /var/lib/postgresql/data ... ok
postgres_db | creating subdirectories ... ok
postgres_db | selecting dynamic shared memory implementation ... posix
postgres_db | selecting default max_connections ... 100
postgres_db | selecting default shared_buffers ... 128MB
postgres_db | selecting default time zone ... Etc/UTC
postgres_db | creating configuration files ... ok
postgres_db | running bootstrap script ... ok
postgres_db | performing post-bootstrap initialization ... ok
postgres_db | syncing data to disk ... ok
postgres_db |
postgres_db |
postgres_db | Success. You can now start the database server using:
postgres_db |
postgres_db |     pg_ctl -D /var/lib/postgresql/data -l logfile start
postgres_db |
postgres_db | initdb: warning: enabling "trust" authentication for local connections
postgres_db | initdb: hint: You can change this by editing pg_hba.conf or using the option -A, or --auth-local and --auth-host
postgres_db | waiting for server to start....2024-02-29 17:50:39.157 UTC [48] LOG:  starting PostgreSQL 16.2 (Debian 16.2-1.pg
12.2.0-14) 12.2.0, 64-bit
postgres_db | 2024-02-29 17:50:39.168 UTC [48] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
postgres_db | 2024-02-29 17:50:39.200 UTC [51] LOG:  database system was shut down at 2024-02-29 17:50:37 UTC
postgres_db | 2024-02-29 17:50:39.232 UTC [48] LOG:  database system is ready to accept connections

```

Рисунки 3-6 – Підняття docker-контейнерів з консолі

Тепер перейдемо за посиланням <http://127.0.0.1:8000/docs> та переглянемо автоматично згенеровану з допомогою SwaggerUI документацію сервісу store:

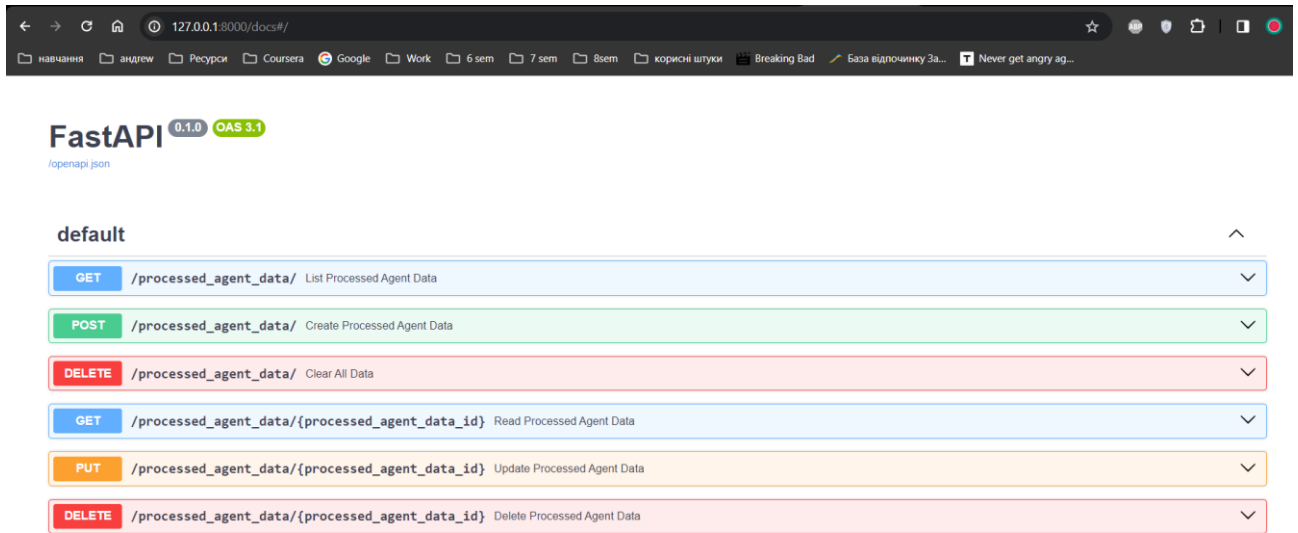


Рисунок 4 – SwaggerUI документація сервісу store

Далі перейдемо за посиланням <http://127.0.0.1:9000/docs> та переглянемо автоматично згенеровану з допомогою SwaggerUI документацію сервісу hub:

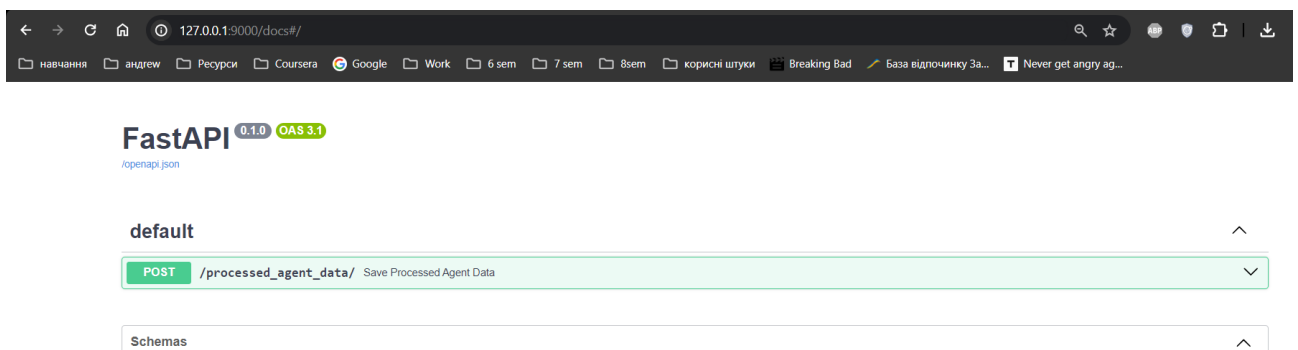


Рисунок 5 – SwaggerUI документація сервісу hub

Так само перейдемо за посиланням <http://127.0.0.1:5050/browser/> та переглянемо стан нашої БД у PgAdmin, попередньо увійшовши в акаунт та під'єднавшись до БД:

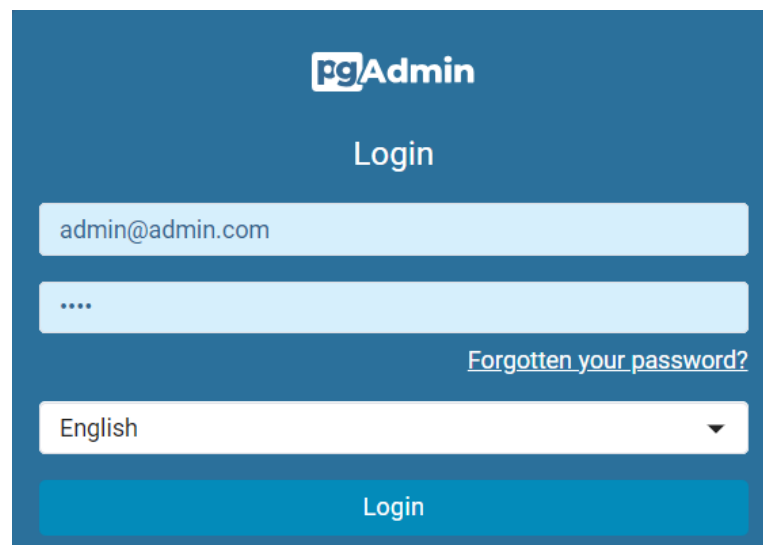


Рисунок 6 – Вхід у PgAdmin

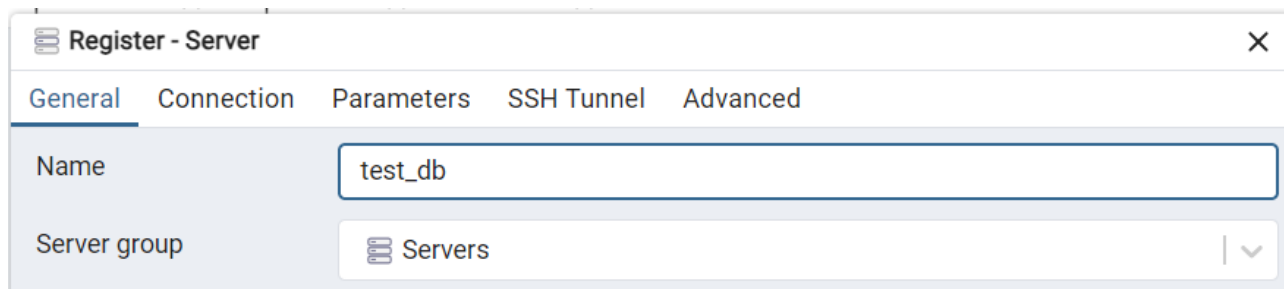


Рисунок 7 – Під'єднання до бази даних у PgAdmin

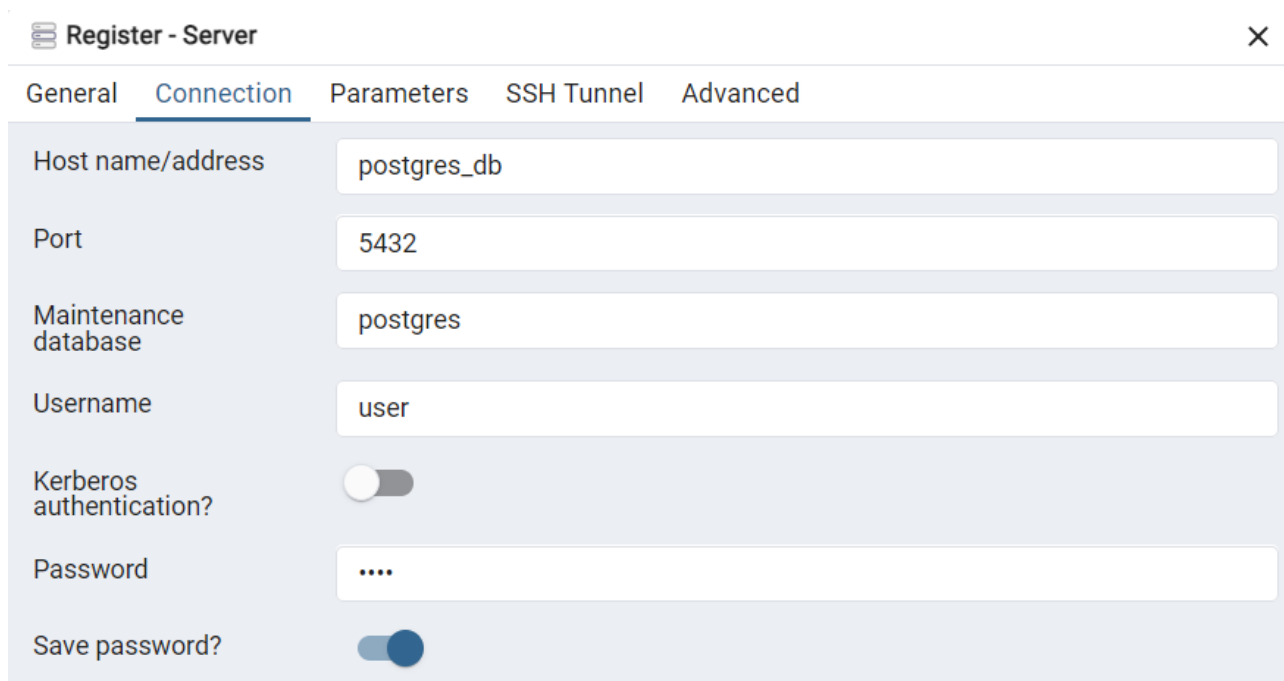


Рисунок 8 – Під'єднання до бази даних у PgAdmin - продовження

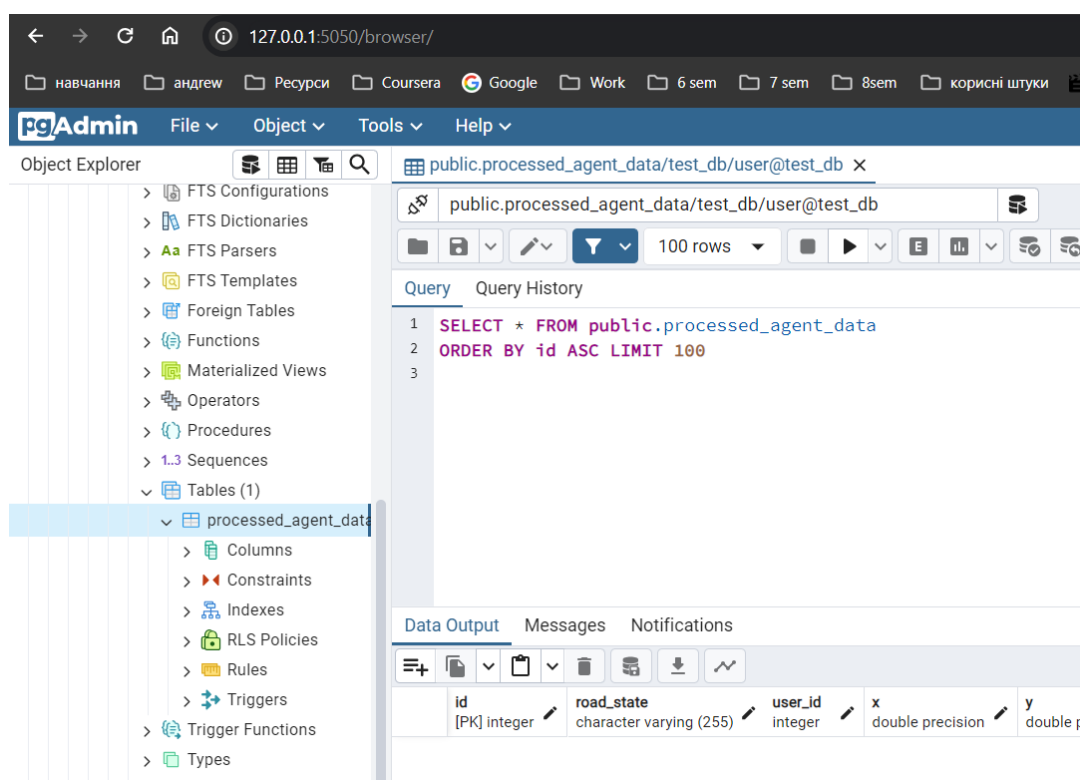


Рисунок 9 – Відслідковування початкового стану бази даних у PgAdmin

Тепер протестуємо систему: спробуємо з допомогою SwaggerUI надіслати 10 запитів на hub, змінюючи координату x, та пересвідчитися, що у store дані запишуться лише після останнього запиту.

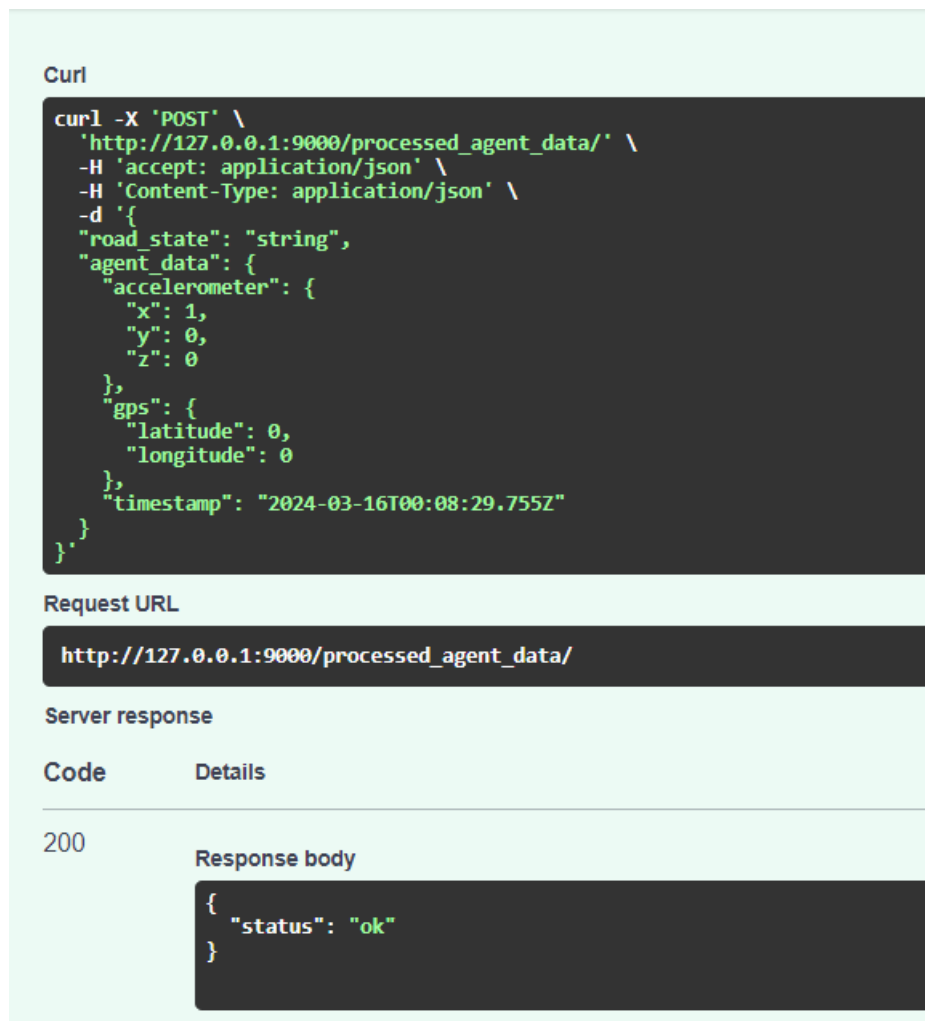


Рисунок 10 – Надсилання даних на hub

Пересвідчимося, що на цьому етапі до бази даних нічого не збереглося:

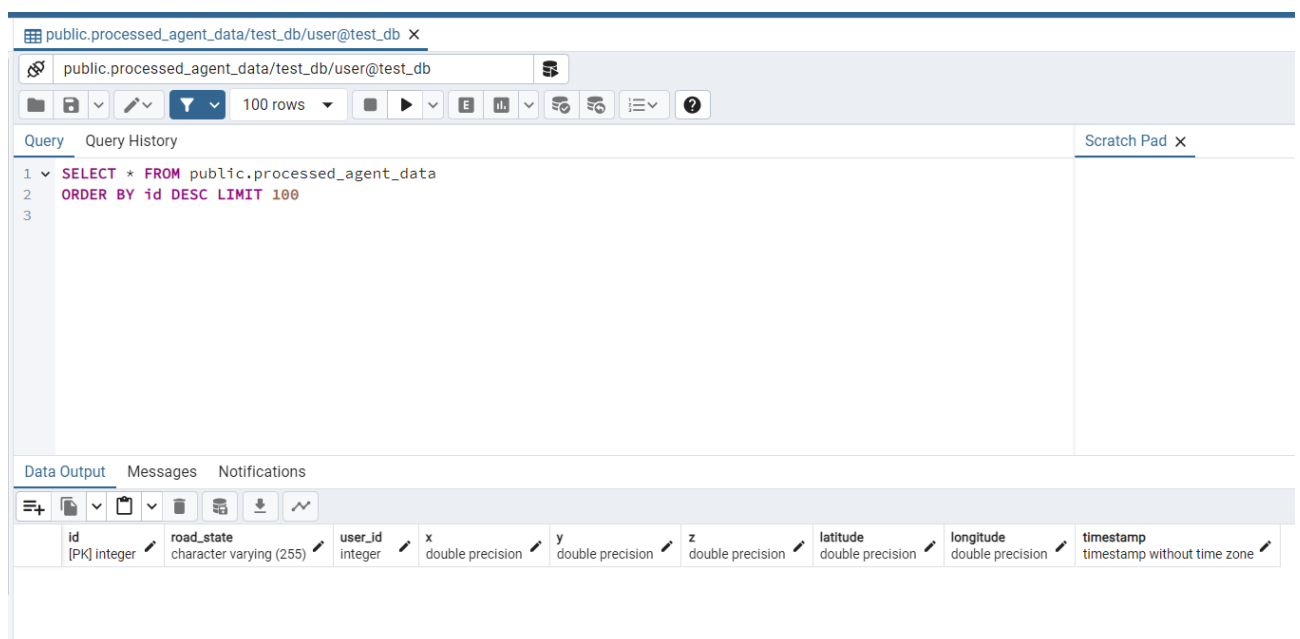


Рисунок 11 – Перевірка стану бази даних після надсилання першого запиту

Надішлемо ще 8 запитів та перевіримо стан бази даних на той момент.

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:9000/processed_agent_data/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "road_state": "string",
    "agent_data": {
      "accelerometer": {
        "x": 9,
        "y": 0,
        "z": 0
      },
      "gps": {
        "latitude": 0,
        "longitude": 0
      },
      "timestamp": "2024-03-16T00:08:29.755Z"
    }
  }'
```

Request URL

http://127.0.0.1:9000/processed_agent_data/

Server response

Code	Details
200	<div>Response body</div> <pre>{ "status": "ok" }</pre>

Рисунок 12 – Надсилання 9ї порції даних на hub

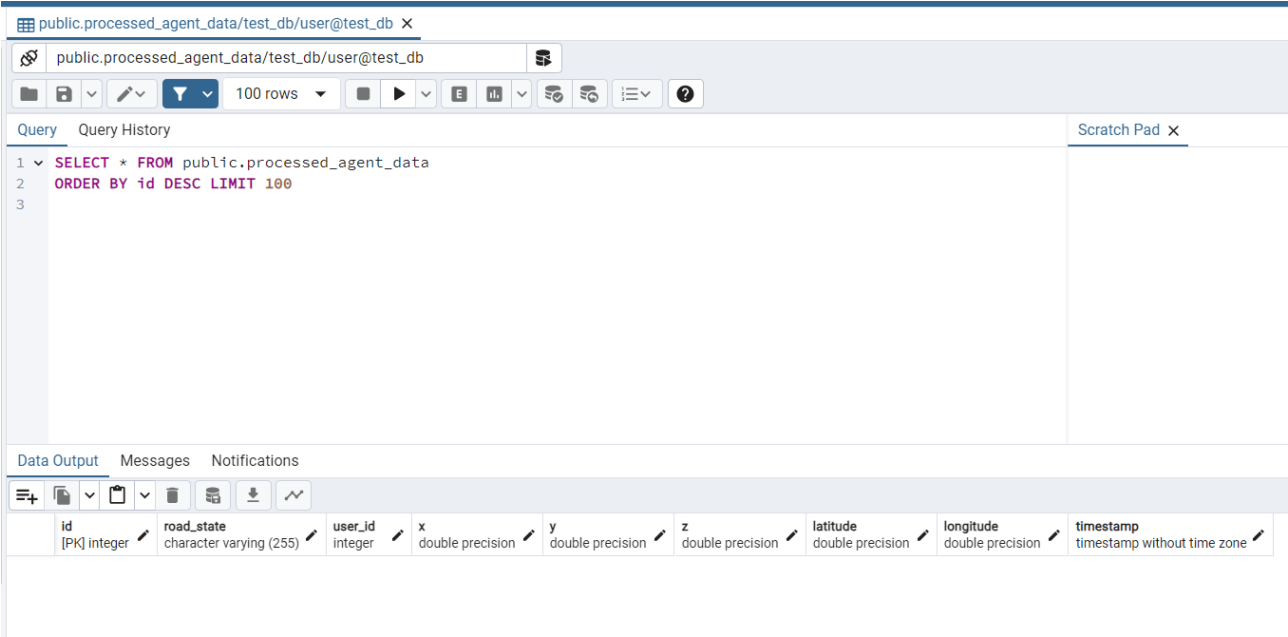


Рисунок 13 – Перевірка стану бази даних після надсилання 9го запиту

Як бачимо, 9 порцій даних було відправлено, але не записано до бази даних. Тепер надішлемо останній запит та пересвідчимося, що усі дані додадуться.

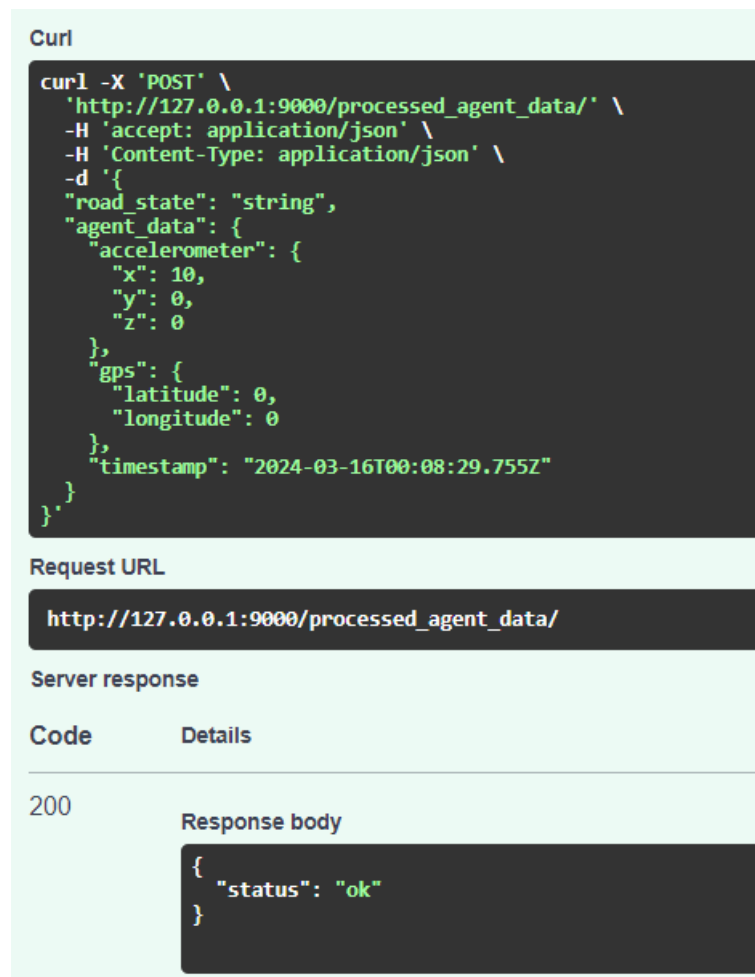


Рисунок 14 – Надсилання 10ї порції даних на hub

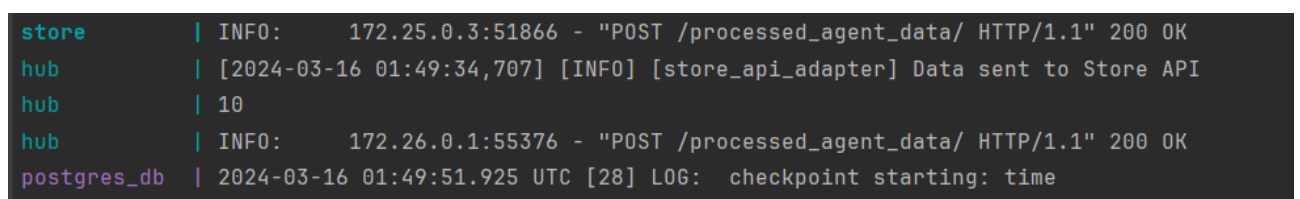


Рисунок 15 – Логи при надсиланні 10ї порції даних

public.processed_agent_data/test_db/user@test_db

Query: `SELECT * FROM public.processed_agent_data ORDER BY id DESC LIMIT 100`

Data Output

	id [PK] integer	road_state character varying (255)	user_id integer	x double precision	y double precision	z double precision	latitude double precision	longitude double precision	timestamp timestamp without time zone
1	10	string	0	1	0	0	0	0	2024-03-16 00:08:29.755
2	9	string	0	2	0	0	0	0	2024-03-16 00:08:29.755
3	8	string	0	3	0	0	0	0	2024-03-16 00:08:29.755
4	7	string	0	4	0	0	0	0	2024-03-16 00:08:29.755
5	6	string	0	5	0	0	0	0	2024-03-16 00:08:29.755
6	5	string	0	6	0	0	0	0	2024-03-16 00:08:29.755
7	4	string	0	7	0	0	0	0	2024-03-16 00:08:29.755
8	3	string	0	8	0	0	0	0	2024-03-16 00:08:29.755
9	2	string	0	9	0	0	0	0	2024-03-16 00:08:29.755
10	1	string	0	10	0	0	0	0	2024-03-16 00:08:29.755

Рисунок 16 – Стан бази даних після отримання пакету даних

Бачимо, що усі дані з'явилися у базі, хоч і записані у оберненому порядку. Отже, лабораторну роботу виконано.

Висновок: під час виконання комп'ютерного практикуму я розібрався з наданою кодовою базою, реалізував логіку збирання даних та надсилання до сховища пакетами, а потім розгорнув необхідні сервіси у Docker та протестував коректність роботи. Весь функціонал відпрацьовував рівно як і очікувалося.