WOJSKOWA AKADEMIA TECHNICZNA

im. Jarosława Dąbrowskiego

WYDZIAŁ CYBERNETYKI

STUDIA II°

Temat: **SPRAWOZDANIE Z REALIZACJI ĆWICZENIA LABORATORYJNEGO**

INFORMATYKA

…………………………………………………………………………………

(kierunek studiów)

INŻYNIERIA SYSTEMÓW

…………………………………………………………………………………

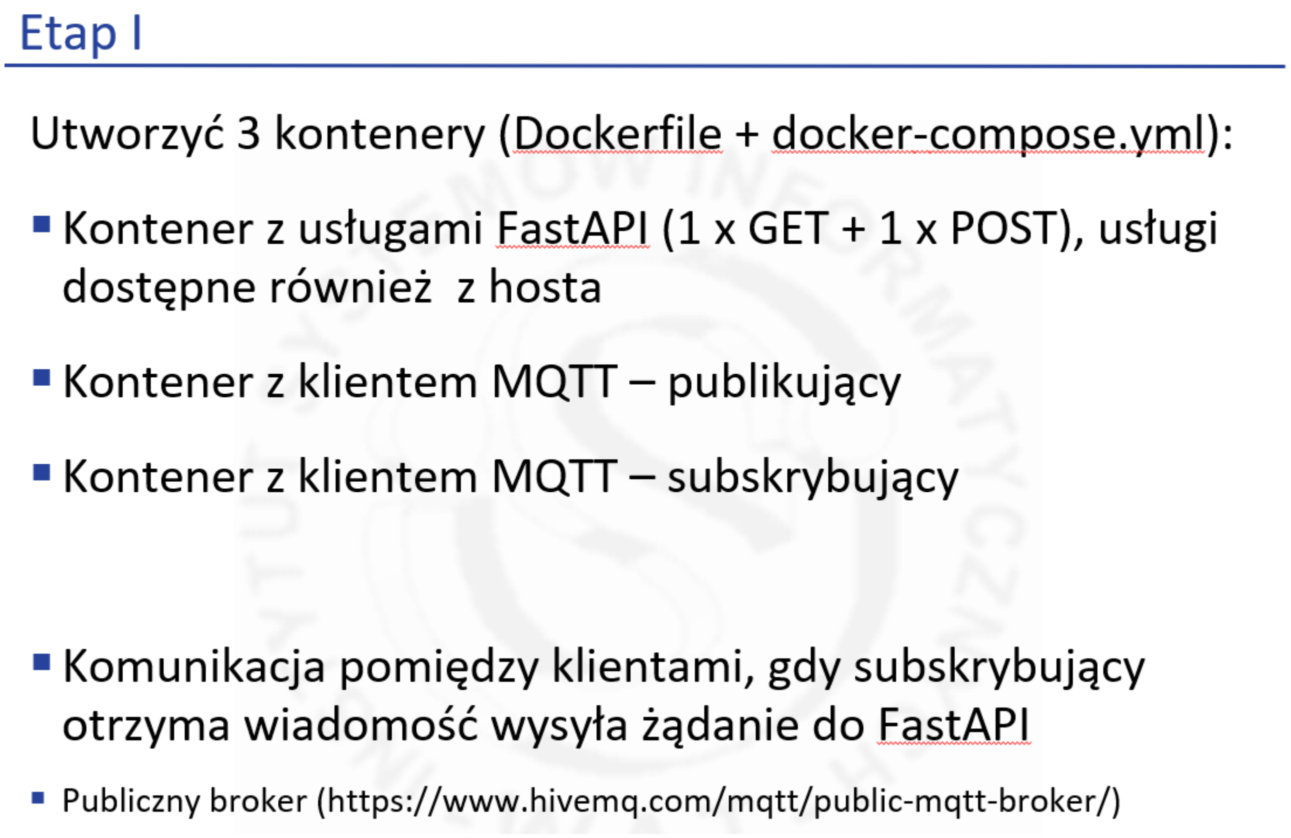
(specjalność)

Wykonał:	Prowadzący:
Radosław Relidzyński	**mgr inż. Tomasz Gutowski**

Warszawa 2023

Treść zadań

I.1. Etap 1

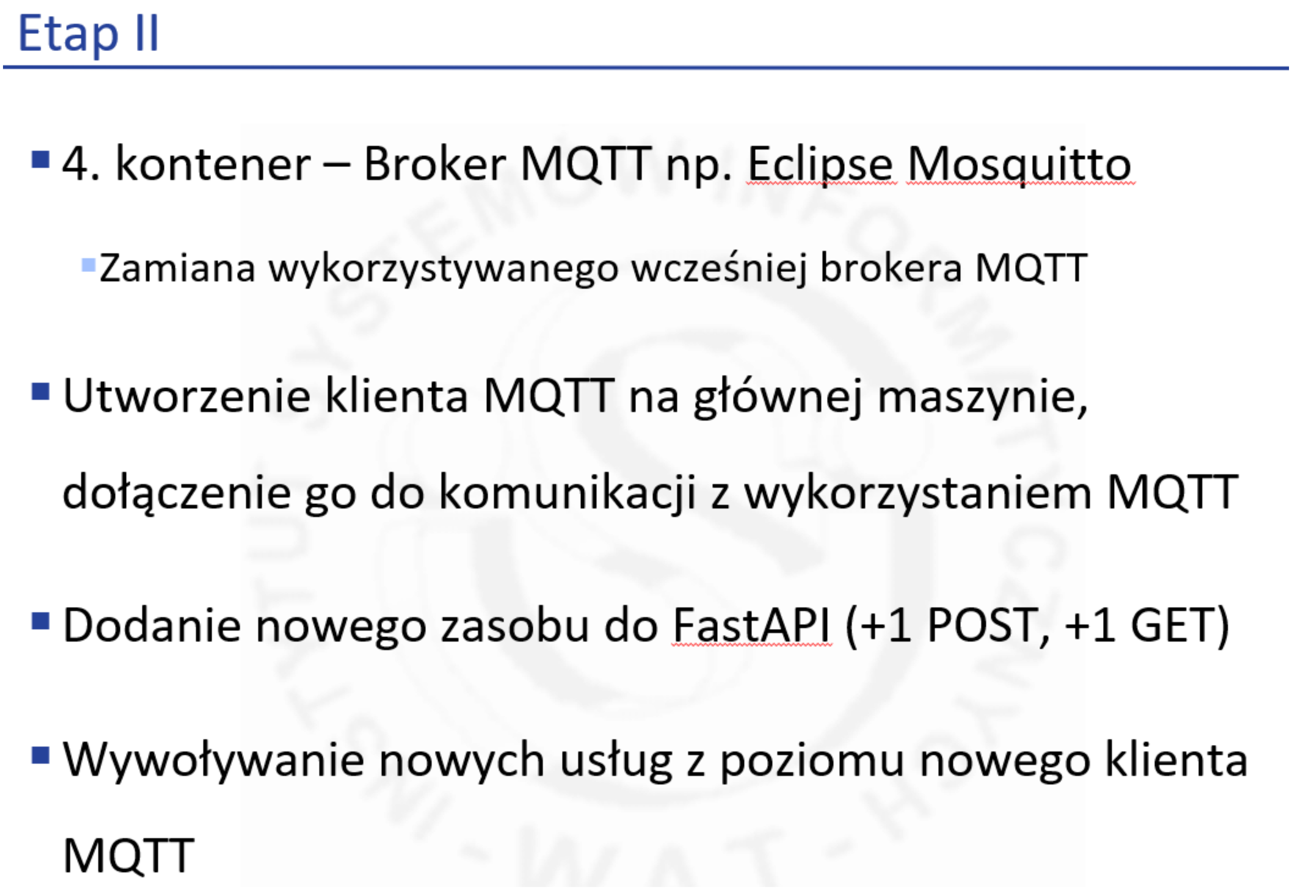


Etap I

Utworzyć 3 kontenery (Dockerfile + docker-compose.yml):

- Kontener z usługami FastAPI (1 x GET + 1 x POST), usługi dostępne również z hosta
- Kontener z klientem MQTT – publikujący
- Kontener z klientem MQTT – subskrybujący
- Komunikacja pomiędzy klientami, gdy subskrybujący otrzyma wiadomość wysyła żądanie do FastAPI
- Publiczny broker (https://www.hivemq.com/mqtt/public-mqtt-broker/)

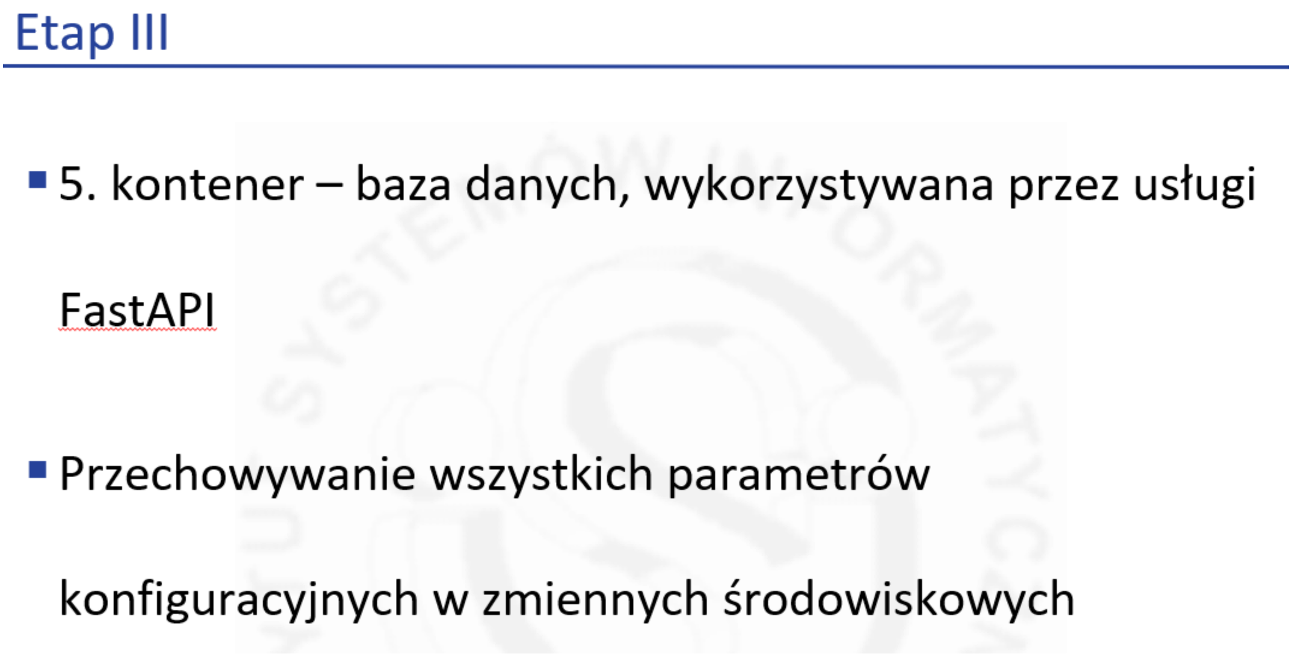
I.2. Etap 2



Etap II

- 4. kontener – Broker MQTT np. Eclipse Mosquitto
  - Zamiana wykorzystywanego wcześniej brokera MQTT
- Utworzenie klienta MQTT na głównej maszynie, dołączenie go do komunikacji z wykorzystaniem MQTT
- Dodanie nowego zasobu do FastAPI (+1 POST, +1 GET)
- Wywoływanie nowych usług z poziomu nowego klienta MQTT

I.3. Etap 3



Etap III

- 5. kontener – baza danych, wykorzystywana przez usługi FastAPI
- Przechowywanie wszystkich parametrów konfiguracyjnych w zmiennych środowiskowych

Rozdział II. Rozwiązanie etapu I

II.1. Tworzenie kontenerów

```
version: '3'

services:
  mqtt_broker:
    container_name: mqttBroker
    build:
      context: mqttBroker
      dockerfile: Dockerfile
    networks:
      - base_network
    ports:
      - "${MQTT_PORT}:${MQTT_PORT}"
      - "${MQTT_WEBSOCKETS_PORT}:${MQTT_WEBSOCKETS_PORT}"
    env_file:
      - .env
#    healthcheck:
#      # test: [ "CMD", "mosquitto_sub", "-h", "localhost", "-p", 
"${MQTT_PORT}", "-t", "healthcheck", "-C", "1" ]
#      test: [ "CMD-SHELL", "ping", "localhost" ]
#      interval: 5s
#      timeout: 3s
#      retries: 2

  postgres_db:
    container_name: postgresDB
    image: postgres:13
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - base_network
    ports:
      - "${POSTGRES_PORT}:${POSTGRES_PORT}"
    healthcheck:
      test: [ "CMD-SHELL", "sh -c 'pg_isready -U ${POSTGRES_USER} -d 
${POSTGRES_DB}'" ]
      interval: 5s
      timeout: 3s
      retries: 2
    env_file:
      - .env

  fastapi_service:
    container_name: fastapi
    build:
      context: fastapiService
      dockerfile: Dockerfile
    networks:
      - base_network
    ports:
      - "${FASTAPI_PORT}:${FASTAPI_PORT}"
    depends_on:
      postgres_db:
        condition: service_healthy
    command: [ "python", "-m", "uvicorn", "fastapi_app:app", "--host", 
"${FASTAPI_HOST}", "--port", "${FASTAPI_PORT}" ]
    env_file:
      - .env
  #    healthcheck:
#      # test: [ "CMD-SHELL", "curl", "--fail",
"${FASTAPI_CONNECTION_PROTOCOL}://localhost:${FASTAPI_PORT}/health" ]
#      test: [ "CMD-SHELL", "ping",
"${FASTAPI_CONNECTION_PROTOCOL}://localhost:${FASTAPI_PORT}/health" ]
#      interval: 5s
#      timeout: 3s
#      retries: 2

  mqtt_subscriber_q0:
    build:
      context: mqttSubscriberQ0
      dockerfile: Dockerfile
    networks:
      - base_network
    depends_on:
      fastapi_service:
        condition: service_started
      mqtt_broker:
        condition: service_started
    env_file:
      - .env

  mqtt_subscriber_q1:
    build:
      context: mqttSubscriberQ1
      dockerfile: Dockerfile
    networks:
      - base_network
    depends_on:
      fastapi_service:
        condition: service_started
      mqtt_broker:
        condition: service_started
    env_file:
      - .env

  mqtt_subscriber_q2:
    build:
      context: mqttSubscriberQ2
      dockerfile: Dockerfile
    networks:
      - base_network
    depends_on:
      fastapi_service:
        condition: service_started
      mqtt_broker:
        condition: service_started
    env_file:
      - .env

  mqtt_app:
    build:
      context: mqttApp
      dockerfile: Dockerfile
    networks:
      - base_network
    depends_on:
      - mqtt_subscriber_q0
      - mqtt_subscriber_q1
      - mqtt_subscriber_q2
    env_file:
      - .env

networks:
  base_network:
    driver: bridge

volumes:
  postgres_data:
```

II.2. Konener fastapi

Dockerfile

```
FROM python

WORKDIR /app

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

COPY . /app/
```

fastapi_app.py

```
import os
import random
import string

from fastapi import FastAPI, HTTPException

import fastapi_database_app as db
from person import Person

app = FastAPI()

db.init_database()


def validate_person_data(person_data):
    return set(Person.__fields__.keys()) == set(person_data.__fields__.keys())


@app.get("/get_first_names")
def get_data1():
    result = db.get_data_from_db()
    print(f'get_first_names, result: {result}')
    return [row['first_name'] for row in result]


@app.get("/get_emails")
def get_data2():
    result = db.get_data_from_db()
    print(f'get_emails, result: {result}')
    return [row['email'] for row in result]


@app.get("/get_all")
```

```
def get_data3():
    result = db.get_data_from_db()
    print(f'get_all, result: {result}')
    return result


@app.post("/add_person/")
def add_person(person: Person):
    if validate_person_data(person):
        result = db.add_data_to_db(person)
        print(f'add_person, result: {result}')
        return result
    else:
        return HTTPException(
            status_code=400,
            detail="Invalid data. Ensure all required fields are provided."
        )


@app.post("/add_random_person")
def add_random_person():
    first_names = ["A", "B", "C"]
    last_names = ["AA", "BB", "CC"]

    first_name = random.choice(string.ascii_uppercase)
    last_name = first_name * 3
    age = random.randint(18, 60)
    email = f"{first_name}.{last_name}@gmail.com"

    person = Person(
        first_name=first_name,
        last_name=last_name,
        age=age,
        email=email,
    )

    # db.add_data_to_db(person)
    return db.add_data_to_db(person)


@app.put("/update_person/")
def update_person(person_to_update: Person):
    if validate_person_data(person_to_update):
        return db.update_db_person(person_to_update)
    else:
        return HTTPException(
            status_code=400,
            detail="Invalid data. Ensure all required fields are provided."
        )

@app.delete("/delete_people")
def delete_people():
    return db.delete_data_from_db()


if __name__ == "__main__":
    FASTAPI_HOST = os.getenv("FASTAPI_HOST", "127.0.0.1")
    FASTAPI_PORT = int(os.getenv("FASTAPI_PORT", 8000))

    db.init_database()
    import uvicorn
    uvicorn.run(app, host=FASTAPI_HOST, port=FASTAPI_PORT)
```

fastapi_database_app.py

```
import os
import psycopg2
from psycopg2.extras import RealDictCursor
from person import Person

POSTGRES_USER = os.getenv("POSTGRES_USER", "postgres")
POSTGRES_PASSWORD = os.getenv("POSTGRES_PASSWORD")
POSTGRES_DB = os.getenv("POSTGRES_DB", "postgres")
POSTGRES_CONNECTION_HOST = os.getenv("POSTGRES_CONNECTION_HOST", "localhost")
POSTGRES_PORT = int(os.getenv("POSTGRES_PORT", 5432))

def get_db_connection():
    conn = psycopg2.connect(
        dbname=POSTGRES_DB,
        user=POSTGRES_USER,
        password=POSTGRES_PASSWORD,
        host=POSTGRES_CONNECTION_HOST,
        port=POSTGRES_PORT,
        cursor_factory=RealDictCursor
    )
    return conn

def run_query(query, params=None):
    conn = get_db_connection()
    cursor = conn.cursor()

    try:
        if params:
            cursor.execute(query, params)
        else:
            cursor.execute(query)

        if query.strip().lower().startswith("select"):
            result = cursor.fetchall()
        else:
            conn.commit()
            result = None

    finally:
        cursor.close()
        conn.close()

    return result

def init_database():
    create_query = """
    CREATE TABLE IF NOT EXISTS people (
        id SERIAL PRIMARY KEY,
        first_name VARCHAR(50) NOT NULL,
        last_name VARCHAR(50) NOT NULL,
        age INT NOT NULL,
        email VARCHAR(50) UNIQUE NOT NULL
    )
    """
    run_query(create_query)
    return "Database initialized"
```

```
def add_data_to_db(person: Person):
    query = "INSERT INTO people (first_name, last_name, age, email) VALUES (%s,
%s, %s, %s) ON CONFLICT DO NOTHING"
    person_data = (person.first_name, person.last_name, person.age,
person.email)
    run_query(query, person_data)
    return "Person added successfully"

def get_data_from_db():
    query = "SELECT * FROM people"
    result = run_query(query)
    return result

def delete_data_from_db():
    query = "DELETE FROM people"
    run_query(query)
    return "People successfully deleted"

def update_db_person(person_to_update: Person):
    query = """
    UPDATE people
    SET age = %s, email = %s
    WHERE first_name = %s AND last_name = %s
    """
    person_data = (person_to_update.age, person_to_update.email,
person_to_update.first_name, person_to_update.last_name)
    run_query(query, person_data)
    return f"Person {person_to_update.first_name} {person_to_update.last_name}
updated successfully"
```

person.py

```
from pydantic import BaseModel


class Person(BaseModel):
    first_name: str
    last_name: str
    age: int
    email: str
```

requirements.txt

```
fastapi
uvicorn
psycopg2
```

II.3. Kontener MQTT publikujący

Dockerfile

```
FROM python

WORKDIR /app

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

```
COPY . /app/

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

mqtt_app.py

```
""" Publisher script that sends values to sensors """
import json
import os
import time

import paho.mqtt.client as mqtt

MQTT_BROCKER_NAME = os.getenv("MQTT_BROCKER_NAME", "mqtt_broker")
MQTT_PORT = int(os.getenv("MQTT_PORT", 1883))
MQTT_KEEPALIVE_TIME = int(os.getenv("MQTT_KEEPALIVE_TIME", 60))

time.sleep(10)

client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)

client.connect(MQTT_BROCKER_NAME, MQTT_PORT, MQTT_KEEPALIVE_TIME)

# qos 0
print("Publishing get_all_people with qos=0")
client.publish("people/get_all_people", payload="get_all_people", qos=0)
time.sleep(1)

# qos 2
print("Publishing add_person with qos=2")
people_data = [
    {"first_name": "Alice", "last_name": "Smith", "age": 25, "email": 
"alice.smith@example.com"},
    {"first_name": "Bob", "last_name": "Brown", "age": 40, "email": 
"bob.brown@example.com"},
    {"first_name": "Cecile", "last_name": "Bracket", "age": 30, "email": 
"cecile.bracket@example.com"},
]
for person_data in people_data:
    client.publish("people/add_person", payload=json.dumps(person_data), qos=2)
    time.sleep(1)

# qos 2
print("Publishing add_person, but random with qos=2")
for person_data in people_data:
    client.publish("people/add_person", payload="add_random_person", qos=2)
    time.sleep(1)

# qos 1
print("Publishing update_person with qos=1")
person_to_update = {"first_name": "John", "last_name": "Doe", "age": 30, 
"email": "john.doe@example.com"}
client.publish("people/update_person", payload=json.dumps(person_to_update), 
qos=1)
time.sleep(1)

# qos 0
print("Publishing get_people, but all with qos=0")
client.publish("people/get_people", payload="get_all_people", qos=0)
time.sleep(1)
```

```
# qos 0
print("Publishing get_people, but for first names with qos=0")
client.publish("people/get_people", payload="get_first_names", qos=0)
time.sleep(1)

client.loop_start()

time.sleep(2)

client.disconnect()

print('Publisher finished')
```

requirements.txt

```
paho-mqtt
```

II.4. Kontener MQTT subskrybujący – na przykładzie Q0

Dockerfile

```
FROM python

WORKDIR /app

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

COPY . /app/

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

fastapi_client.py

```
import os

import requests

FASTAPI_PORT = int(os.getenv("FASTAPI_PORT", 8000))
FASTAPI_CONNECTION_PROTOCOL = os.getenv("FASTAPI_CONNECTION_PROTOCOL", 'http')


class FastapiClient:

    def __init__(self):
        self.url =
f"{FASTAPI_CONNECTION_PROTOCOL}://fastapi_service:{FASTAPI_PORT}/"

    def add_person(self, person_data):
        add_url = self.url + "add_person/"
        return requests.post(add_url, json=person_data)

    def add_random_person(self):
        add_url = self.url + "add_random_person/"
        return requests.post(add_url)

    def update_person(self, person_to_update):
        add_url = self.url + "update_person/"
        return requests.put(add_url, json=person_to_update)
    def show_people(self):
        show_url = self.url + "get_all/"
        return requests.get(show_url)

    def show_people_first_names(self):
        show_url = self.url + "get_first_names/"
        return requests.get(show_url)
```

requirements.txt

```
paho-mqtt
requests
```

subscriber.py

```
import os
import time

import paho.mqtt.client as mqtt

from fastapi_client import FastapiClient

MQTT_BROCKER_NAME = os.getenv("MQTT_BROCKER_NAME", "mqtt_broker")
MQTT_PORT = int(os.getenv("MQTT_PORT", 1883))
MQTT_KEEPALIVE_TIME = int(os.getenv("MQTT_KEEPALIVE_TIME", 60))

time.sleep(2)

fastapi_client = FastapiClient()


def on_connect(client, userdata, flags, reason_code, properties):
    print("SubscriberQ0 connected with result code " + str(reason_code))


def on_message(client, userdata, msg):
    print("SubscriberQ0 " + msg.topic + ": " + str(msg.payload) + ", qos = " +
str(msg.qos))

    if msg.payload.decode('utf-8') == "get_all_people":
        response = fastapi_client.show_people()

    elif msg.payload.decode('utf-8') == "get_first_names":
        response = fastapi_client.show_people_first_names()

    else:
        print(f"Error, wrong get qos 0 message: {msg.payload}")

    print(f"Repsonse: {response}")

    if response.status_code == 200:
        print(f"Response: {response.json()}")
        print()
        print("All people:")
        for person in response.json():
            print(person)
    else:
        print(f"Error {response.status_code}: {response.text}")


client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
```

```
client.on_connect = on_connect
client.on_message = on_message

client.connect(MQTT_BROCKER_NAME, MQTT_PORT, MQTT_KEEPALIVE_TIME)

client.subscribe("people/get_people", qos=0)

try:
    print("SubscriberQ0 working...")
    client.loop_forever()
except KeyboardInterrupt:
    print("SubscriberQ0 shutdown")

client.disconnect()
```

Rozdział III. Rozwiązanie etapu II

III.1. Kontener brockera

Dockerfile

```
FROM eclipse-mosquitto:2.0

COPY mosquitto.conf /mosquitto/config/mosquitto.conf
```

Dockerfile

```
allow_anonymous true
listener 1883
listener 9001
protocol websockets
persistence false
```

III.2. Klient MQTT – nowy klient, na przykładzie Q1

Dockerfile

```
FROM python

WORKDIR /app

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

COPY . /app/

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

fastapi_client.py

```
import os

import requests

FASTAPI_PORT = int(os.getenv("FASTAPI_PORT", 8000))
FASTAPI_CONNECTION_PROTOCOL = os.getenv("FASTAPI_CONNECTION_PROTOCOL", 'http')
```

```
class FastapiClient:

    def __init__(self):
        self.url =
f"{FASTAPI_CONNECTION_PROTOCOL}://fastapi_service:{FASTAPI_PORT}/"

    def add_person(self, person_data):
        add_url = self.url + "add_person/"
        return requests.post(add_url, json=person_data)

    def add_random_person(self):
        add_url = self.url + "add_random_person/"
        return requests.post(add_url)

    def update_person(self, person_to_update):
        add_url = self.url + "update_person/"
        return requests.put(add_url, json=person_to_update)

    def show_people(self):
        show_url = self.url + "get_all/"
        return requests.get(show_url)

    def show_people_first_names(self):
        show_url = self.url + "get_first_names/"
        return requests.get(show_url)
```

requirements.txt

```
paho-mqtt
requests
```

subscriber.py

```
import json
import os
import time

import paho.mqtt.client as mqtt

from fastapi_client import FastapiClient

MQTT_BROCKER_NAME = os.getenv("MQTT_BROCKER_NAME", "mqtt_broker")
MQTT_PORT = int(os.getenv("MQTT_PORT", 1883))
MQTT_KEEPALIVE_TIME = int(os.getenv("MQTT_KEEPALIVE_TIME", 60))

time.sleep(2)

fastapi_client = FastapiClient()


def on_connect(client, userdata, flags, reason_code, properties):
    print("SubscriberQ1 connected with result code " + str(reason_code))


def on_message(client, userdata, msg):
    print("SubscriberQ1 " + msg.topic + ": " + str(msg.payload) + ", qos = " +
str(msg.qos))

    person_to_update = json.loads(msg.payload)

    response = fastapi_client.update_person(person_to_update)
    print(f"Repsonse: {response}")

    if response.status_code == 200:
        print(f"Response: {response.json()}")
    else:
        print(f"Error {response.status_code}: {response.text}")


client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
client.on_connect = on_connect
client.on_message = on_message

client.connect(MQTT_BROCKER_NAME, MQTT_PORT, MQTT_KEEPALIVE_TIME)

client.subscribe("people/update_person", qos=1)

try:
    print("SubscriberQ1 working...")
    client.loop_forever()
except KeyboardInterrupt:
    print("SubscriberQ1 shutdown")

client.disconnect()
```

III.3. Nowe zasoby FastAPI

Dodawanie losowego klienta oraz pobieranie samych imion:

```
@app.get("/get_first_names")
def get_data1():
    result = db.get_data_from_db()
    print(f'get_first_names, result: {result}')
    return [row['first_name'] for row in result]

@app.post("/add_random_person")
def add_random_person():
    first_names = ["A", "B", "C"]
    last_names = ["AA", "BB", "CC"]

    first_name = random.choice(string.ascii_uppercase)
    last_name = first_name * 3
    age = random.randint(18, 60)
    email = f"{first_name}.{last_name}@gmail.com"

    person = Person(
        first_name=first_name,
        last_name=last_name,
        age=age,
        email=email,
    )

    # db.add_data_to_db(person)
    return db.add_data_to_db(person)
```

III.4. Wywołanie nowych usług z MQTT

Wywołanie w mqtt_app.py

```
# qos 2
print("Publishing add_person, but random with qos=2")
for person_data in people_data:
    client.publish("people/add_person", payload="add_random_person", qos=2)
    time.sleep(1)

# qos 0
print("Publishing get_people, but for first names with qos=0")
client.publish("people/get_people", payload="get_first_names", qos=0)
time.sleep(1)
```

Obsługa w subscriberach:

```
# Q2
def on_message(client, userdata, msg):
    print("SubscriberQ2 " + msg.topic + ": " + str(msg.payload) + ", qos = " +
str(msg.qos))

    if msg.payload.decode('utf-8') == "add_random_person":
        response = fastapi_client.add_random_person()

    else:
        person_data = json.loads(msg.payload)
        response = fastapi_client.add_person(person_data)

    print(f"Repsonse: {response}")

    if response.status_code == 200:
        print(f"Response: {response.json()}")
    else:
        print(f"Error {response.status_code}: {response.text}")

# Q0
def on_message(client, userdata, msg):
    print("SubscriberQ0 " + msg.topic + ": " + str(msg.payload) + ", qos = " +
str(msg.qos))

    if msg.payload.decode('utf-8') == "get_all_people":
        response = fastapi_client.show_people()

    elif msg.payload.decode('utf-8') == "get_first_names":
        response = fastapi_client.show_people_first_names()

    else:
        print(f"Error, wrong get qos 0 message: {msg.payload}")

    print(f"Repsonse: {response}")

    if response.status_code == 200:
        print(f"Response: {response.json()}")
        print()
        print("All people:")
        for person in response.json():
            print(person)
    else:
        print(f"Error {response.status_code}: {response.text}")
```

Rozdział IV. Rozwiązanie Etapu III

IV.1. Kontener bazy danych

Fragment z docker-compose.yml

```
  postgres_db:
    container_name: postgresDB
    image: postgres:13
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - base_network
    ports:
      - "${POSTGRES_PORT}:${POSTGRES_PORT}"
    healthcheck:
      test: [ "CMD-SHELL", "sh -c 'pg_isready -U ${POSTGRES_USER} -d
${POSTGRES_DB}'" ]
      interval: 5s
      timeout: 3s
      retries: 2
    env_file:
      - .env
```

IV.2. Przechowywanie wszystkich parametrów konfiguracyjnych w zmiennych środowiskowych

Plik .env z wszystkimi zmiennymi środowiskowymi

```
# General environment file

# FastApi
FASTAPI_HOST=0.0.0.0
FASTAPI_PORT=8000
FASTAPI_CONNECTION_PROTOCOL=http
POSTGRES_CONNECTION_HOST=postgres_db

# MQTT
MQTT_BROCKER_NAME=mqtt_broker
MQTT_PORT=1883
MQTT_WEBSOCKETS_PORT=9001
MQTT_KEEPALIVE_TIME=60

# Postgres
POSTGRES_USER=radek
POSTGRES_PASSWORD=radek123
POSTGRES_DB=radek
POSTGRES_HOST=localhost
POSTGRES_PORT=5432
```

Wykorzystanie ich w docker-compose.yml:

```
  fastapi_service:
    container_name: fastapi
    build:
      context: fastapiService
      dockerfile: Dockerfile
    networks:
      - base_network
    ports:
      - "${FASTAPI_PORT}:${FASTAPI_PORT}"
    depends_on:
      postgres_db:
        condition: service_healthy
    command: [ "python", "-m", "uvicorn", "fastapi_app:app", "--host",
"${FASTAPI_HOST}", "--port", "${FASTAPI_PORT}" ]
    env_file:
      - .env
```

Wykorzystanie ich w skryptach pythona:

```
MQTT_BROCKER_NAME = os.getenv("MQTT_BROCKER_NAME", "mqtt_broker")
MQTT_PORT = int(os.getenv("MQTT_PORT", 1883))
MQTT_KEEPALIVE_TIME = int(os.getenv("MQTT_KEEPALIVE_TIME", 60))
```

Rozdział V. Prezentacja działania

V.1. Kontener brockera

```
2025-01-17 14:00:21 1737118821: mosquitto version 2.0.20 starting
2025-01-17 14:00:21 1737118821: Config loaded from /mosquitto/config/mosquitto.conf.
2025-01-17 14:00:21 1737118821: Opening ipv4 listen socket on port 1883.
2025-01-17 14:00:21 1737118821: Opening ipv6 listen socket on port 1883.
2025-01-17 14:00:21 1737118821: Opening websockets listen socket on port 9001.
2025-01-17 14:00:21 1737118821: mosquitto version 2.0.20 running
2025-01-17 14:00:30 1737118830: New connection from 172.18.0.5:55053 on port 1883.
2025-01-17 14:00:30 1737118830: New client connected from 172.18.0.5:55053 as auto-40B13BB3-02AA-68E3-5E6A-9E2C033C2B22 (p2, c1, k60).
2025-01-17 14:00:30 1737118830: New connection from 172.18.0.7:54249 on port 1883.
2025-01-17 14:00:30 1737118830: New client connected from 172.18.0.7:54249 as auto-F1B0FC4C-E834-B309-E433-9E745E45F204 (p2, c1, k60).
2025-01-17 14:00:30 1737118830: New connection from 172.18.0.6:46423 on port 1883.
2025-01-17 14:00:30 1737118830: New client connected from 172.18.0.6:46423 as auto-B0C9731B-2037-AC53-765F-2FDE5E30A80A (p2, c1, k60).
2025-01-17 14:00:39 1737118839: New connection from 172.18.0.8:45285 on port 1883.
2025-01-17 14:00:39 1737118839: New client connected from 172.18.0.8:45285 as auto-8977504B-25F2-B7AC-BA2A-D4FDB22A5C72 (p2, c1, k60).
2025-01-17 14:00:51 1737118851: Client auto-8977504B-25F2-B7AC-BA2A-D4FDB22A5C72 disconnected.
```

V.2. Kontener bazy danych

```
2025-01-17 14:00:21 2025-01-17 13:00:21.921 UTC [1] LOG:  starting PostgreSQL 13.18 (Debian 13.18-1.pgdg120+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 12.2.0-14) 12.2.0, 64-bit
2025-01-17 14:00:21 2025-01-17 13:00:21.921 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2025-01-17 14:00:21 2025-01-17 13:00:21.921 UTC [1] LOG:  listening on IPv6 address "::", port 5432
2025-01-17 14:00:21 2025-01-17 13:00:21.926 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2025-01-17 14:00:21 2025-01-17 13:00:21.931 UTC [26] LOG:  database system was shut down at 2025-01-17 13:00:09 UTC
2025-01-17 14:00:21 2025-01-17 13:00:21.937 UTC [1] LOG:  database system is ready to accept connections
2025-01-17 14:00:21 
2025-01-17 14:00:21 PostgreSQL Database directory appears to contain a database; Skipping initialization
2025-01-17 14:00:21 
```

V.3. Kontener fastapi

```
2025-01-17 14:00:28 INFO:     Started server process [1]
2025-01-17 14:00:28 INFO:     Waiting for application startup.
2025-01-17 14:00:28 INFO:     Application startup complete.
2025-01-17 14:00:28 INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
2025-01-17 14:00:46 INFO:     172.18.0.6:47882 - "PUT /update_person/ HTTP/1.1" 200 OK
2025-01-17 14:00:47 INFO:     172.18.0.5:42984 - "GET /get_all/ HTTP/1.1" 307 Temporary Redirect
2025-01-17 14:00:47 get_all, result: []
2025-01-17 14:00:47 INFO:     172.18.0.5:42984 - "GET /get_all HTTP/1.1" 200 OK
2025-01-17 14:00:48 INFO:     172.18.0.5:42996 - "GET /get_first_names/ HTTP/1.1" 307 Temporary Redirect
2025-01-17 14:00:48 get_first_names, result: []
2025-01-17 14:00:48 INFO:     172.18.0.5:42996 - "GET /get_first_names HTTP/1.1" 200 OK
2025-01-17 14:00:49 add_person, result: Person added successfully
2025-01-17 14:00:49 INFO:     172.18.0.7:44534 - "POST /add_person/ HTTP/1.1" 200 OK
2025-01-17 14:00:49 add_person, result: Person added successfully
2025-01-17 14:00:49 INFO:     172.18.0.7:44544 - "POST /add_person/ HTTP/1.1" 200 OK
2025-01-17 14:00:49 add_person, result: Person added successfully
```

```
2025-01-17 14:00:49 INFO:     172.18.0.7:44554 - "POST /add_person/ HTTP/1.1" 200 OK
2025-01-17 14:00:49 INFO:     172.18.0.7:44568 - "POST /add_random_person/ HTTP/1.1"
307 Temporary Redirect
2025-01-17 14:00:49 INFO:     172.18.0.7:44568 - "POST /add_random_person HTTP/1.1"
200 OK
2025-01-17 14:00:49 INFO:     172.18.0.7:44570 - "POST /add_random_person/ HTTP/1.1"
307 Temporary Redirect
2025-01-17 14:00:49 INFO:     172.18.0.7:44570 - "POST /add_random_person HTTP/1.1"
200 OK
2025-01-17 14:00:49 INFO:     172.18.0.7:44574 - "POST /add_random_person/ HTTP/1.1"
307 Temporary Redirect
2025-01-17 14:00:49 INFO:     172.18.0.7:44574 - "POST /add_random_person HTTP/1.1"
200 OK
```

V.4. Kontener aplikacji mqtt

```
2025-01-17 14:00:51 Publishing get_all_people with qos=0
2025-01-17 14:00:51 Publishing add_person with qos=2
2025-01-17 14:00:51 Publishing add_person, but random with qos=2
2025-01-17 14:00:51 Publishing update_person with qos=1
2025-01-17 14:00:51 Publishing get_people, but all with qos=0
2025-01-17 14:00:51 Publishing get_people, but for first names with qos=0
2025-01-17 14:00:51 Publisher finished
```

Rozdział VI. Podsumowanie

Podczas ćwiczenia laboratoryjnego udało się zrealizować przy pomocy mqtt oraz fastapi system komunikacji. System został zaprojektowany w sposób modularny, składając się z różnych komponentów.

W ramach ćwiczenia udało się rozszerzyć wiedzę o tym jak działa protokół mqtt, jak działają różne qos, a także jak tworzyć serwer http z obsługą różnych rodzajów zapytań.

Implementacja zawiera:

- Utworzenie 3 topic'ów
- Zastosowanie różnych qos
- Imlementację REST API z wykorzystaniem fast-api
- Komunikację z API z poziomu aplikacji klienckiej (fastapi_client.py)
- Usługi REST API pobierające dane
- Usługę REST API modyfikującą dane
- Usługę REST API usuwającą dane

- Wywołanie usług REST API przez klienta MQTT przetwarzane przez klienta
- Zapisywanie i odczytywanie danych od MQTT z bazy danych
- Zapisywanie i odczytywanie danych od REST API z bazy danych
- Tworzenie własnego brockera MQTT
- Tworzenie niezależnego kontenera z bazą danych
- Tworzenie i wykorzystywanie zmiennych środowiskowych