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

* 1. Etap 1

A screenshot of a computer

Description automatically generated

* 1. Etap 2

A screenshot of a computer

Description automatically generated

* 1. Etap 3

A screenshot of a computer

Description automatically generated

1. Rozwiązanie etapu I
   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:

* 1. 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

* 1. 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

* 1. 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()

1. Rozwiązanie etapu II
   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

* 1. 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()

* 1. 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)

* 1. 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}")

1. Rozwiązanie Etapu III
   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

* 1. 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))

1. Prezentacja działania
   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.

* 1. 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

* 1. 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

* 1. 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

1. 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