WOJSKOWA AKADEMIA TECHNICZNA
im. Jarosława Dąbrowskiego
WYDZIAŁ CYBERNETYKI

Technologie Internetu Rzeczy
Lab. 1

Student
Bartosz MIAZGA

Prowadzący laboratoria:
mgr inż. Tomasz Gutowski

Treść zadania

Zadanie 1

Poniżej przedstawiono skrypt programu publisher.py:

```
import time
import paho.mqtt.client as mqtt
from random import randrange

localhost = '127.0.0.1'
port = 1883

topic_0 = "/sensors/temperature"
topic_1 = "/sensors/blood_pressure"
topic_2 = "/parameters/time_of_sleep"


def on_connect(client, userdata, flags, rc):
    print("Connected with result code: " + str(rc))

def publish(client):
    qos = 0
    while True:
        time.sleep(2)

        if qos == 3:
            qos = 0

        if qos == 0:
            result = client.publish(topic_0, randrange(42), qos)
            print("Temperature info sent")
        elif qos == 1:
            result = client.publish(topic_1, randrange(1000), qos)
            print("Pressure info sent")
        elif qos == 2:
            result = client.publish(topic_2, randrange(12), qos)
            print("Time of sleep info sent")

        status = result[0]

        if status == 0:
            print("Message sent, qos = " + str(qos))
            qos = qos +1
        else:
            print("Massage sending failure")



client = mqtt.Client()
client.on_connect = on_connect
client.connect(localhost,port)
publish(client)

client.disconnect()

```

Poniżej przedstawiono skrypt programu subscriber.py:

```python
import paho.mqtt.client as mqtt
import time
import json
import requests

localhost = '127.0.0.1'
port = 1883

topic_0 = "/sensors/temperature"
topic_1 = "/sensors/blood_pressure"
topic_2 = "/parameters/time_of_sleep"

def on_log(client, userdata, level, buf):
    k = 0

def on_connect(client, userdata, flags, rc):
    print("Connected with result code: " + str(rc))

def on_disconnect(client, userdata, flags, rc=0):
    print("Disconnecting with result code: " + str(rs))

def on_message(client, userdata, msg):
    global m_decode
    topic = msg.topic
    m_decode = str(msg.payload.decode("utf-8","ignore"))
    time.sleep(1)
    if msg.topic == "/sensors/temperature":
        print("temperature: " + m_decode)
    elif msg.topic == "/sensors/blood_pressure":
        print("blood pressure: " + m_decode)
    elif msg.topic == "/sensors/time_of_sleep":
        print("time of sleep: " + m_decode)

client = mqtt.Client()
client.on_connect = on_connect
client.on_disconnect = on_disconnect
client.on_log = on_log
client.on_message = on_message

client.connect(localhost,port)
client.subscribe([(topic_0, 0), (topic_1,1), (topic_2,2)])
client.loop_forever()

```

Efekt działania publisher.py:

```
pi@raspberry:~/mu_code $ python3 publisher.py
Temperature info sent
Message sent, qos = 0
Pressure info sent
Message sent, qos = 1
Time of sleep info sent
Message sent, qos = 2
Temperature info sent
Message sent, qos = 0
Pressure info sent
Message sent, qos = 1
Time of sleep info sent
Message sent, qos = 2
Temperature info sent
Message sent, qos = 0
```

Efekt działania subscriber.py:

```
pi@raspberry:~/mu_code $ python3 subscriber.py
Connected with result code: 0
temperature: 8
blood pressure: 24
temperature: 11
blood pressure: 74
temperature: 5
```

Komentarz:

Komunikacja odbywa się na 3 różne topiki, wybrałem tematykę dotyczącą danych typu „lifestyle" dotyczących użytkownika, takie dane mógłby na przykład wysyłać smartwatch, topiki:

a) sensors/temperaturę
b) sensors/blood_pressure
c) parameters/time_of_sleep

Dla każdego z topików wysyłane są komunikaty o innym Qos, odpowiednio dla:

a) sensors/temperaturę – qos 0
b) sensors/blood_pressure – qos 1
c) parameters/time_of_sleep – qos 2

Stworzyłem 3 subskrypcje, 1 publikację

Zadanie 2

Poniżej przedstawiono skrypt programu zadanie2.py:

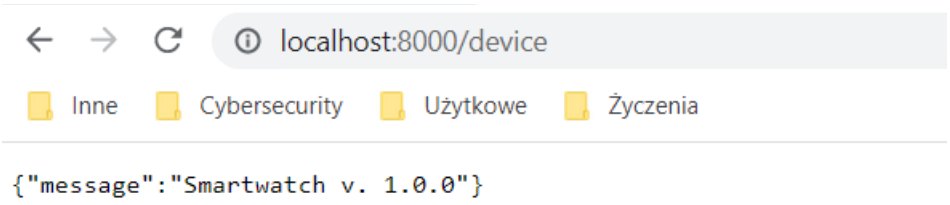
zadanie2.py > ...

```python
from fastapi import FastAPI
from pydantic import BaseModel
from typing import List

app = FastAPI()

records = {
    1: {
        "temperature": 36,
        "blood_pressure": 1000,
        "time_of_sleep": 8

    },
    2: {
        "temperature": 38,
        "blood_pressure": 1100,
        "time_of_sleep": 9

    }
}

class Record(BaseModel):
    temperature: int
    blood_pressure: int
    time_of_sleep: int


@app.get("/device")
def informations_about_device():
    return {"message" : "Smartwatch v. 1.0.0"}

@app.get("/records/{record_id}")
def device_record(record_id: int):
    return {'record': records.get(record_id) }

@app.get("/records")
def all_device_records():
    return {'records': records}

@app.post("/add-record/{record_id}")
def create_record(record_id: int, record: Record):
    if record_id in records:
        return {"Record already exists"}
    else:
        records[record_id] = {"temperature": record.temperature, "blood_pressure": record.blood_pressure, "time_of_sleep": record.time_of_sleep}
        return records[record_id]

@app.post("/change-record/{record_id}")
def change_record(record_id: int, record: Record):
    if record_id not in records:
        return {"Record doesn't exists"}
    else:
        records[record_id] = {"temperature": record.temperature, "blood_pressure": record.blood_pressure, "time_of_sleep": record.time_of_sleep}
        return records[record_id]
```

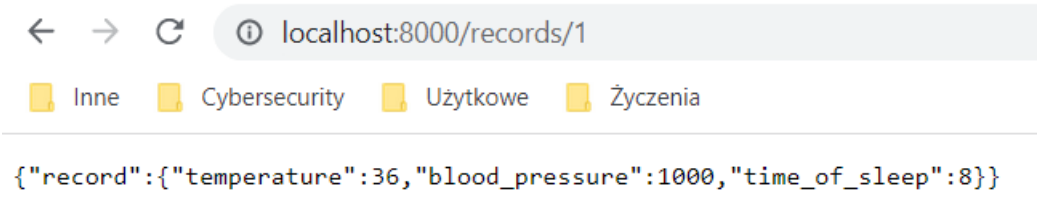
Efekt działania:

Usługi pobierające dane:

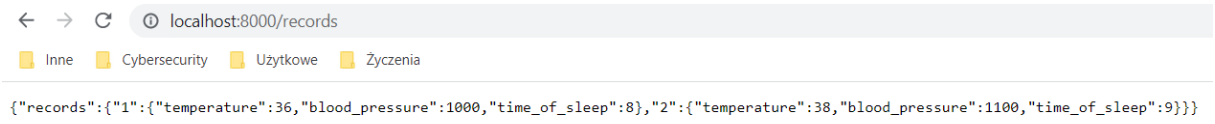
a)Informacje o urządzeniu:



b)Pobranie wskazanego rekordu:

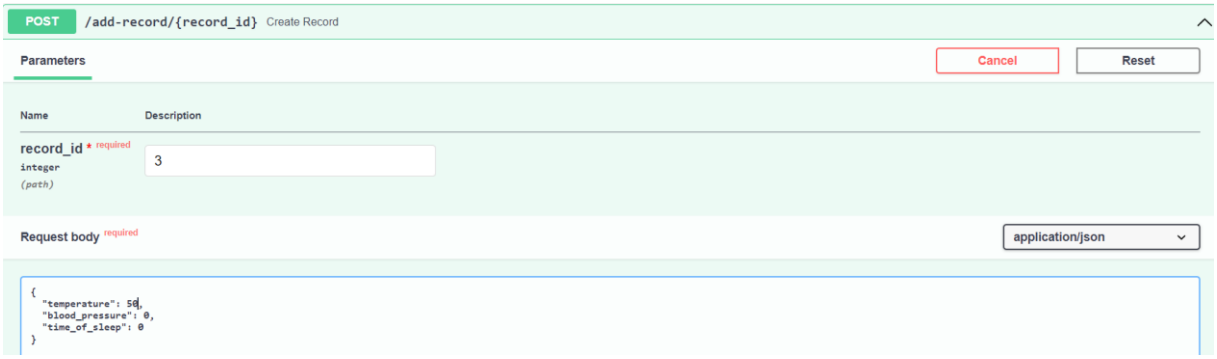


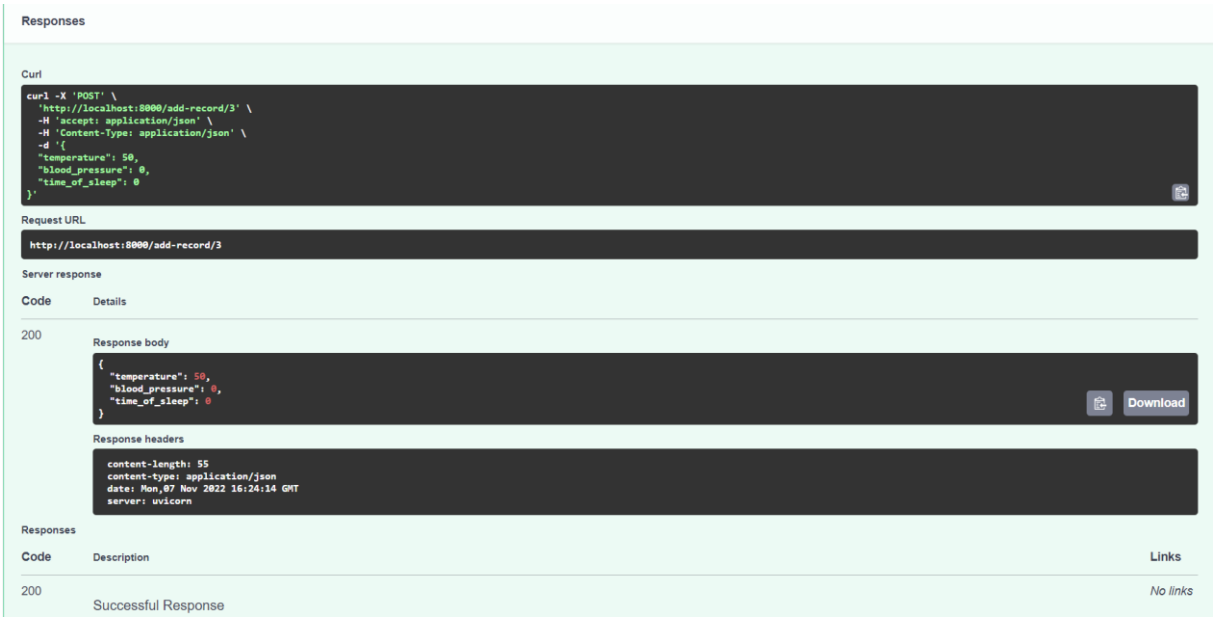
c)Pobranie wszystkich rekordów:



Usługi dodające/modyfikujące dane:

a)Dodawanie nowego rekordu:





b)Modyfikacja rekordu: