

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:

```
2 import time
3 import paho.mqtt.client as mqtt
4 from random import randrange
5
6 localhost = '127.0.0.1'
7 port = 1883
8
9 topic_0 = "/sensors/temperature"
10 topic_1 = "/sensors/blood_pressure"
11 topic_2 = "/parameters/time_of_sleep"
12
13
14 def on_connect(client, userdata, flags, rc):
15     print("Connected with result code: " + str(rc))
16
17 def publish(client):
18     qos = 0
19     while True:
20         time.sleep(2)
21
22         if qos == 3:
23             qos = 0
24
25         if qos == 0:
26             result = client.publish(topic_0, randrange(42), qos)
27             print("Temperature info sent")
28         elif qos == 1:
29             result = client.publish(topic_1, randrange(1000), qos)
30             print("Pressure info sent")
31         elif qos == 2:
32             result = client.publish(topic_2, randrange(12), qos)
33             print("Time of sleep info sent")
34
35         status = result[0]
36
37         if status == 0:
38             print("Message sent, qos = " + str(qos))
39             qos = qos + 1
40         else:
41             print("Message sending failure")
42
43
44
45 client = mqtt.Client()
46 client.on_connect = on_connect
47 client.connect(localhost, port)
48 publish(client)
49
50 client.disconnect()
51
```

Poniżej przedstawiono skrypt programu subscriber.py:

```
1 import paho.mqtt.client as mqtt
2 import time
3 import json
4 import requests
5
6 localhost = '127.0.0.1'
7 port = 1883
8
9 topic_0 = "/sensors/temperature"
10 topic_1 = "/sensors/blood_pressure"
11 topic_2 = "/parameters/time_of_sleep"
12
13 def on_log(client, userdata, level, buf):
14     k = 0
15
16 def on_connect(client, userdata, flags, rc):
17     print("Connected with result code: " + str(rc))
18
19 def on_disconnect(client, userdata, flags, rc=0):
20     print("Disconnecting with result code: " + str(rs))
21
22 def on_message(client, userdata, msg):
23     global m_decode
24     topic = msg.topic
25     m_decode = str(msg.payload.decode("utf-8","ignore"))
26     time.sleep(1)
27     if msg.topic == "/sensors/temperature":
28         print("temperature: " + m_decode)
29
30     elif msg.topic == "/sensors/blood_pressure":
31         print("blood pressure: " + m_decode)
32     elif msg.topic == "/sensors/time_of_sleep":
33         print("time of sleep: " + m_decode)
34
35 client = mqtt.Client()
36 client.on_connect = on_connect
37 client.on_disconnect = on_disconnect
38 client.on_log = on_log
39 client.on_message = on_message
40
41 client.connect(localhost,port)
42 client.subscribe([(topic_0, 0), (topic_1,1), (topic_2,2)])
43 client.loop_forever()
```

Efekt działania publisher.py:

```
pi@raspberrypi:~/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@raspberrypi:~/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/temperature
- b) sensors/blood_pressure
- c) parameters/time_of_sleep

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

- a) sensors/temperature – 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 > ...

```
1  from fastapi import FastAPI
2  from pydantic import BaseModel
3  from typing import List
4
5  app = FastAPI()
6
7  records = {
8      1: {
9          "temperature": 36,
10         "blood_pressure": 1000,
11         "time_of_sleep": 8
12     },
13     },
14     2: {
15         "temperature": 38,
16         "blood_pressure": 1100,
17         "time_of_sleep": 9
18     }
19 }
20
21
22 class Record(BaseModel):
23     temperature: int
24     blood_pressure: int
25     time_of_sleep: int
26
27
28 @app.get("/device")
29 def informations_about_device():
30     return {"message": "Smartwatch v. 1.0.0"}
31
32 @app.get("/records/{record_id}")
33 def device_record(record_id: int):
34     return {'record': records.get(record_id) }
35
36 @app.get("/records")
37 def all_device_records():
```

```
38     return {'records': records}
39
40 @app.post("/add-record/{record_id}")
41 def create_record(record_id: int, record: Record):
42     if record_id in records:
43         return {"Record already exists"}
44     else:
45         records[record_id] = {"temperature": record.temperature, "blood_pressure": record.blood_pressure, "time_of_sleep": record.time_of_sleep}
46         return records[record_id]
47
48 @app.post("/change-record/{record_id}")
49 def change_record(record_id: int, record: Record):
50     if record_id not in records:
51         return {"Record doesn't exists"}
52     else:
53         records[record_id] = {"temperature": record.temperature, "blood_pressure": record.blood_pressure, "time_of_sleep": record.time_of_sleep}
54         return records[record_id]
55
```

Efekt działania:

Usługi pobierające dane:

a) Informacje o urządzeniu:

← → ↻ ⓘ localhost:8000/device

Inne Cybersecurity Użytkowe Życzenia

```
{"message": "Smartwatch v. 1.0.0"}
```

b) Pobranie wskazanego rekordu:

← → ↻ ⓘ localhost:8000/records/1

Inne Cybersecurity Użytkowe Życzenia

```
{"record": {"temperature": 36, "blood_pressure": 100, "time_of_sleep": 8}}
```

c) Pobranie wszystkich rekordów:

← → ↻ ⓘ localhost:8000/records

Inne Cybersecurity Użytkowe Życzenia

```
{"records": {"1": {"temperature": 36, "blood_pressure": 100, "time_of_sleep": 8}, "2": {"temperature": 38, "blood_pressure": 110, "time_of_sleep": 9}}}
```

Usługi dodające/modyfikujące dane:

a) Dodawanie nowego rekordu:

POST /add-record/{record_id} Create Record

Parameters

record_id * required
integer
(path)

3

Request body required

application/json

```
{  "temperature": 50,  "blood_pressure": 0,  "time_of_sleep": 0}
```

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8000/add-record/3' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "temperature": 50,
    "blood_pressure": 0,
    "time_of_sleep": 0
  }'
```

Request URL

http://localhost:8000/add-record/3

Server response

Code	Details
200	<p>Response body</p> <pre>{ "temperature": 50, "blood_pressure": 0, "time_of_sleep": 0 }</pre> <p>Response headers</p> <pre>content-length: 55 content-type: application/json date: Mon, 07 Nov 2022 16:24:14 GMT server: uvicorn</pre>

Responses

Code	Description	Links
200	Successful Response	No links

b)Modyfikacja rekordu:

POST /change-record/{record_id} Change Record

Parameters

Cancel Reset

Name	Description
record_id * required	
integer	
(path)	

Request body required

application/json

```
{
  "temperature": 20,
  "blood_pressure": 0,
  "time_of_sleep": 0
}
```

Curl

```
curl -X 'POST' \
  'http://localhost:8000/change-record/3' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "temperature": 20,
    "blood_pressure": 0,
    "time_of_sleep": 0
  }'
```

Request URL

http://localhost:8000/change-record/3

Server response

Code	Details
200	<p>Response body</p> <pre>{ "temperature": 20, "blood_pressure": 0, "time_of_sleep": 0 }</pre> <p>Response headers</p> <pre>content-length: 55 content-type: application/json date: Mon, 07 Nov 2022 16:25:18 GMT server: uvicorn</pre>

Responses

Code	Description	Links
200	Successful Response	No links