Komunikator MQTT

Protokół MQTT

Message Queuing Telemetry Transport (MQTT) to prosty i lekki protokół transmisji wiadomości i danych. Pierwsza wersja powstała w 1999 roku, protokół został opracowany w firmie IBM oraz Circus Lin, obecnie jest rozwijany przez społeczność OpenSource http://mqtt.org/. Najnowsza wersja 5.0 została opublikowana w 2019 roku.

Właściwości protokołu

- protokół MQTT jest prostym i lekkim protokołem warstwy aplikacji,
- protokół MQTT jest oparty o wzorzec publikacja/subskrypcja,
- jest przeznaczony do transmisji dla urządzeń niewymagających dużej przepustowości,
- poprzez ograniczenie prędkości transmisji, protokół zapewnia większą niezawodność,
- protokół MQTT idealnie sprawdza się przy połączeniach maszyna-maszyna (M2M), w Internecie rzeczy, w urządzeniach mobilnych, oraz tam, gdzie wymagana jest oszczędność przepustowości, oraz energii.

W tabeli 8.1 przedstawiono porównanie protokołów MQTT z HTTP.

	MQTT	**HTTP**
Przesyłana treść	Dane lub krótkie wiadomości	Głównie dokumenty, często bardzo duże
Wzorzec komunikacji	Publikacja / Subskrypcja	Żądanie / Odpowiedź
Złożoność	Mała	Większa
Rozmiar wiadomości	Mały, jeden mały nagłówek	Duży, kilka nagłówków
Jakość usługi	Trzy poziomy	Jeden poziom
Dystrybucja wiadomości	Jeden do wielu (1:0,1,n) Jedna publikacja jest rozsyłana do wielu subskrybentów, albo do żadnego jeśli żaden nie subskrybuje tematu	Jeden do jednego
Biblioteki programistyczne (klienckie)	Małe	Duże

Tabela 8.1. Porównanie protokołów HTTP i MQTT [12]

Łączność z wykorzystaniem protokołu MQTT wymaga Brokera MQTT.

Broker MQTT

Broker MQTT pełni rolę serwera, z którym łączą się klienci. Klienci mogą występować w dwóch rolach, publikatorów i subskrybentów. Publikatory mogą publikować wiadomości w danym temacie a subskrybenci mogą subskrybować tematy. Przesyłanie wiadomości odbywa się za pośrednictwem brokera.

Broker musi być widziany przez komputery klientów. Klienci nie muszą widzieć się wzajemnie. Częstym rozwiązaniem jest broker z publicznym adresem IP i klienci z dostępem do Internetu, bez konieczności posiadania publicznego IP. Klientami mogą być komputery w sieciach lokalnych i urządzenia przenośne. Publikator może przekazać wiadomość subskrybentowi bez znajomości jego adresu IP i wiedzy o jego istnieniu. Broker odbiera wiadomości od klientów publikujących, a następnie rozsyła je do klientów subskrybujących dany temat.

Przykładem zastosowania protokołu MQTT jest Facebook Messenger, za pomocą którego można przesyłać wiadomości między urządzeniami przenośnymi nieposiadającymi publicznych adresów IP.

Funkcje zarządzania połączeniami z brokerem MQTT

- `Connect` – Ustanawia połączenie z brokerem.
- `Subscribe` – Subskrybuje (nasłuchuje) zadany temat na brokerze.
- `Publish` – Publikuje (wysyła) informację na podany temat, poprzez broker, do wszystkich klientów subskrybujących dany temat.
- `Unsubscribe` – Przestaje subskrybować dany temat.
- `Disconnect` – Zamyka połączenie z brokerem.

Powyższe nazwy funkcji pojawiają się w nazwach metod bibliotek programistycznych realizujących funkcje klienta MQTT.

Tematy – Topics

- Publikatory publikują wiadomości na dany temat, subskrybenci subskrybują wybrane tematy.
- Tematy nie muszą być wcześniej tworzone, i mogą mieć dowolną nazwę.
- Tematy mają wielopoziomową strukturę hierarchiczną.
- / (slash) – rozdziela poziomy.
- # (hash character) – multi level wildcard.
- + (plus character) – single level wildcard.
- Przykład: house/+/main-light.
- $SYS/broker/# – komunikaty brokera.
- Symbole + i # można stosować subskrybując tematy, nie można ich stosować publikując.

Jakość usługi – Quality of Service – QoS

Każde połączenie z brokerem musi mieć określoną jakość usługi QoS. Jakość usługi w protokole MQTT, jest to umowa pomiędzy nadawcą (publikatorem), a odbiorcą (subskrybentem) dotycząca gwarancji dostarczenia wiadomości. Gdy klient wysyła/publikuje wiadomość określa poziom QoS tej wiadomości. Gdy broker wysyła daną wiadomość do subskrybentów, obowiązuje QoS określony przez klienta, który opublikował wiadomość.

Zdefiniowane są trzy poziomy jakości usługi określające czy i ile razy wiadomość musi zostać dostarczona do subskrybenta

- `QoS = 0` – At most once (co najwyżej raz) – wysłana wiadomość nie potrzebuje potwierdzenia dostarczenia – wyślij i zapomnij. Jest to najprostsza metoda wysyłania wiadomości. Klient publikuje wiadomość w danym temacie, ale nie otrzymuje potwierdzenia otrzymania wiadomości przez brokera. Jest to odpowiedni wybór w przypadku niezawodnego połączenia klienta z brokerem. `QoS = 0` gwarantuje najniższy koszt pod względem transferu danych.
- `QoS = 1` – At least once (przynajmniej raz) – wysłana wiadomość musi zostać dostarczona przynajmniej raz, potwierdzenie jest wymagane, wiadomość może zostać dostarczona do wielu odbiorców (wiadomość może być odebrana kilkukrotnie, jest wysyłana do subskrybenta, dopóki broker nie otrzyma potwierdzenia odbioru). Można stosować, gdy wiadomość jest idempotentna.
- `QoS = 2` – Exactly once (dokładnie raz) – wysłana wiadomość musi zostać dostarczona dokładnie jeden raz do odbiorcy. Jest to najwyższy poziom QoS, w którym jest wykorzystywana sekwencja czterech komunikatów pomiędzy nadawcą i odbiorcą. QoS = 2 gwarantuje dostarczenie wiadomości dokładnie raz, ale z najwyższym kosztem transferu danych. Należy stosować, gdy wiadomość nie jest idempotentna.

Opcja Retain

- Domyślnie `Retain = false` i broker po przesłaniu opublikowanej wiadomości do wszystkich subskrybentów usuwa ją z pamięci, nowi subskrybenci nie dostaną tej wiadomości.
- Jeśli `Retain = true` broker zachowuje ostatnią wiadomość w temacie i dostarcza ją do nowych subskrybentów, przydatne do przekazania stanu systemu nowym subskrybentom.

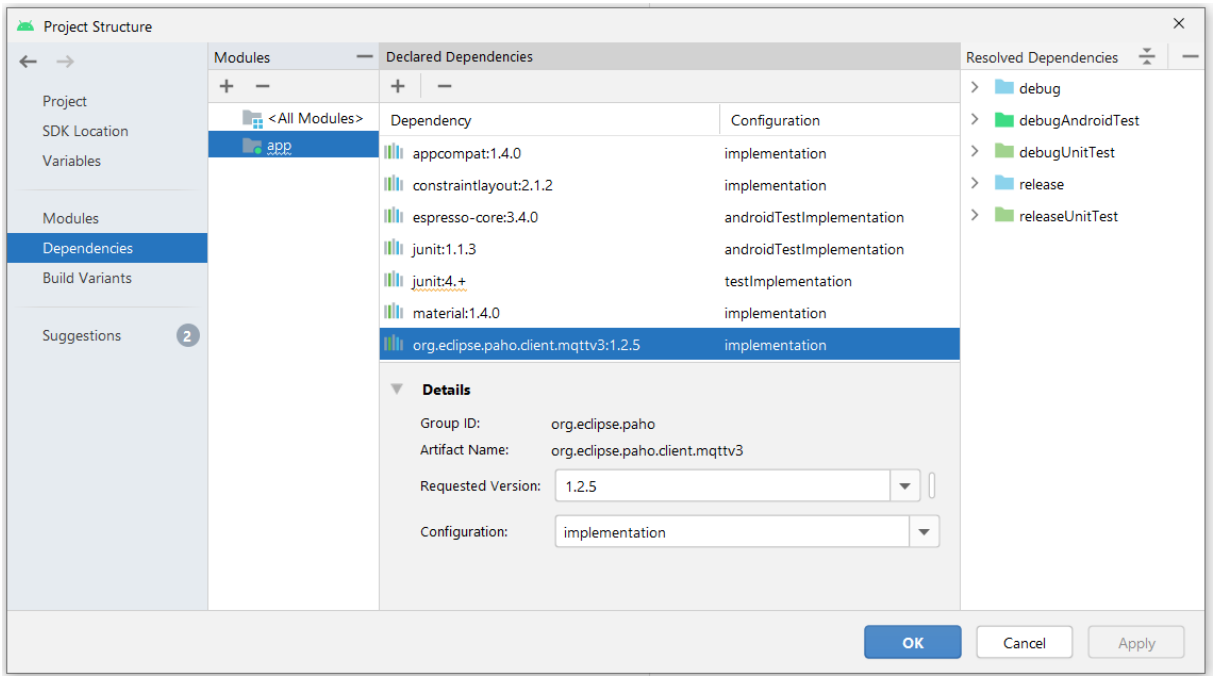
Opcja `Retain = true` powinna być ustawiona, kiedy jakiś czujnik publikuje swój stan, tylko wtedy, gdy ten stan ulega zmianie. Na przykład, czujnik zamkniętych drzwi publikuje stan, tylko wtedy, gdy drzwi są zamykane bądź otwierane. Gdy `Retain = false` nowy subskrybent może długo czekać zanim się dowie, czy drzwi są otwarte, czy zamknięte.

Biblioteka klienta MQTT Paho

Eclipse Paho https://www.eclipse.org/paho/ jest projektem Open Source, którego celem jest tworzenie narzędzi dla systemów Internetu rzeczy. Jednym z nich jest biblioteka klienta MQTT dla aplikacji tworzonych w języku Java https://www.eclipse.org/paho/index.php?page=clients/java/index.php.

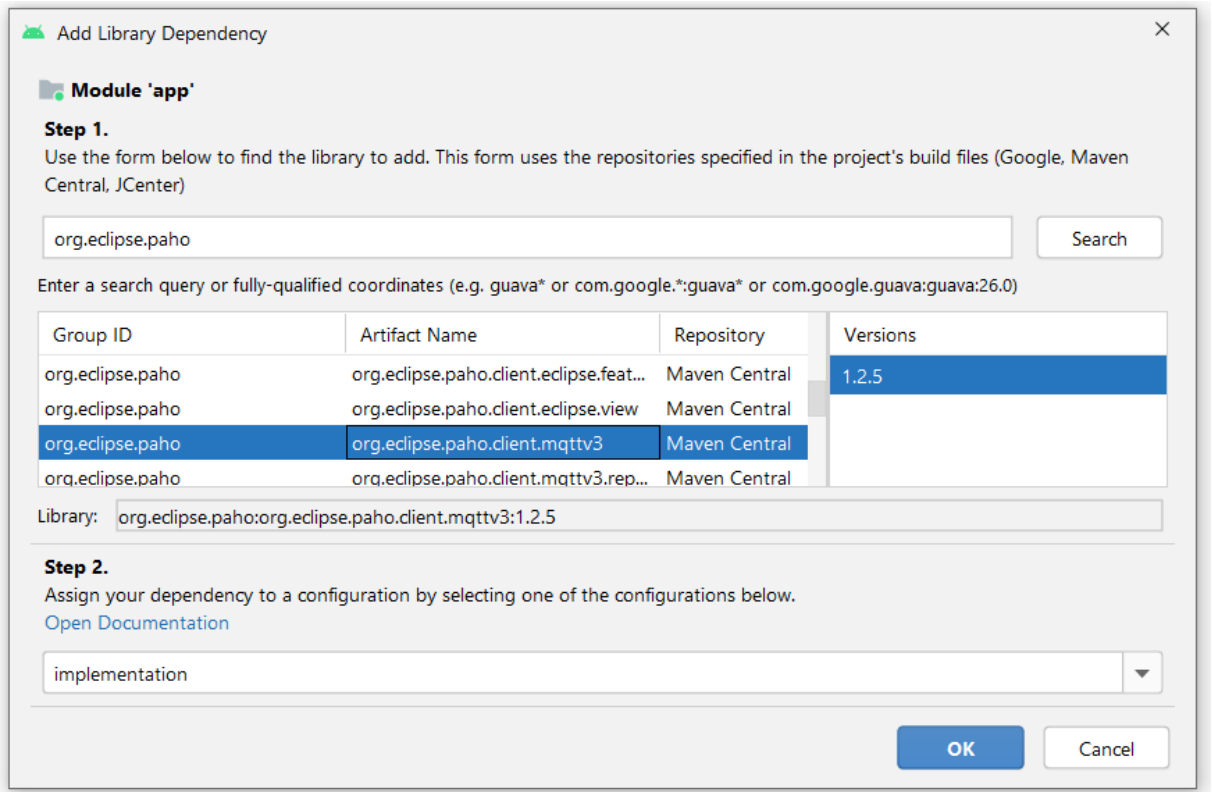
Dodanie do projektu Android Studio biblioteki klienta MQTT Paho

1. Z Menu wybrać File->Project Structure->Dependencies, rysunek 1.



Rys. 1. Okno zależności z zaznaczoną zależnością, którą należy dodać

2. Aby dodać zależność należy kliknąć „+” w zakładce „`Declared Dependecies`”.
3. Dodać zależność wybraną na rysunku 2.
    a. org.eclipse.paho
    b. org.eclipse.paho.client.mqttv3



Rys. 2. Okno dodawania zależności z zaznaczoną zależnością do dodania

Dodanie pozwolenia użycia Internetu w pliku AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
```

Aplikacja prostego komunikatora MQTT

1. Posługując się przykładem z rysunku 3 i listingu 1 stworzyć program prostego komunikatora internetowego, w przykładowym kodzie do publikacji i subskrypcji wykorzystywany jest temat `/paho`.
2. Przetestować aplikację.
3. Po przetestowaniu aplikacji
   a. zmienić subskrybowany temat na `/paho/#`,
   b. zmienić temat publikacji na `/paho/imię autora`,
   c. w małych grupach (2, 3 osobowych) przejść na inne tematy nieznane dla innych grup,
   d. przetestować aplikację ze zmienionymi tematami.

Rys. 3. Przykładowy ekran aplikacji

Listing 1. Przykładowy kod prostej aplikacji klienta MQTT

```
package com.example.paho_1;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Context;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
```

```
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;

import java.nio.charset.StandardCharsets;
import java.util.Random;

public class MainActivity extends AppCompatActivity {

    private MqttClient sampleClient;
    private String broker = "tcp://lukan.sytes.net:1883";
    private String clientId = "titi";
    private String topicSub = "/paho";
    private MqttConnectOptions connOpts;
    private TextView textView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Random random = new Random();
        clientId += random.nextInt();

        textView = findViewById(R.id.textView2);
        MemoryPersistence persistence = new MemoryPersistence();

        try {
            sampleClient = new MqttClient(broker, clientId, persistence);
        } catch (MqttException e) {
            e.printStackTrace();
        }
        sampleClient.setCallback(new MqttCallback() {
            @Override
            public void connectionLost(Throwable cause) {
                try {
                    sampleClient.connect(connOpts);
                } catch (MqttException e) {
                    e.printStackTrace();
                }

                try {
                    sampleClient.subscribe(topicSub);
                    Toast.makeText(getApplicationContext(), "Subskrybcja",
                            Toast.LENGTH_LONG).show();
                } catch (MqttException e) {
                    e.printStackTrace();
                }

            }

            @Override
            public void messageArrived(String topic, MqttMessage message) throws Exception
{
                //Log.v("paho13",message.toString());
                textView.setText(message.getId() +" " + topic + ": " + message.toString() +
                        "\n" + textView.getText());
            }

            @Override
            public void deliveryComplete(IMqttDeliveryToken token) {

            }
        });

        connOpts = new MqttConnectOptions();
```

```
        connOpts.setCleanSession(true);
        connOpts.setAutomaticReconnect(true);

        try {
            sampleClient.connect(connOpts);
        } catch (MqttException e) {
            e.printStackTrace();
        }

        try {
            sampleClient.subscribe(topicSub);
            Toast.makeText(this, "Subskrybcja", Toast.LENGTH_LONG).show();
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    public void onClickButton2(View view){
        MqttMessage message;
        message = new MqttMessage(((TextView)findViewById(R.id.editTextTextPersonName)).
                getText().toString().getBytes(StandardCharsets.UTF_8));

        try {
            sampleClient.publish("/paho", message);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }
}
```