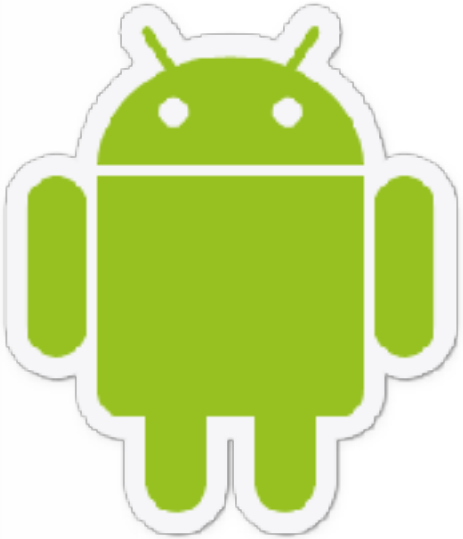
TABEL DATA

DAIMN
AUTOMATION | INDUSTRIAL SERVICE | MONITORING

Smart Factory Seminar 2018



Public profile

Name

Dimas Maryanto

Public email

engineer.dimmaryanto93@outlook.com

You can manage verified email addresses in your email settings.

Bio

Software Engineer

You can @mention other users and organizations to link to them.

URL

http://logs.dimas-maryanto.com

Company

@tabeldatadotcom

You can @mention your company's GitHub organization to link it.

Location

Bandung, Jawa Barat

Profile picture

Upload new picture

- MQTT
  - Connect
  - QoS
  - Payload
  - Publisher
  - Subscribe / Unsubscribe
- Eclipse Mosquitto Broker
  - Messaging
- Architecture


TD
TABEL DATA


DAIMN
AUTOMATION | INDUSTRIAL SERVICE | MONITORING

MQTT


MQTT.ORG

MQTT

MQTT is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium.

MQTT Broker

The broker is primarily responsible for receiving all messages, filtering them, decide who is interested in it and then sending the message to all subscribed clients. It also holds the session of all persisted clients including subscriptions and missed messages (More details). Another responsibility of the broker is the authentication and authorization of clients. And at most of the times a broker is also extensible, which allows to easily integrate custom authentication, authorization and integration into backend systems. Especially the integration is an important aspect, because often the broker is the component, which is directly exposed on the internet and handles a lot of clients and then passes messages along to downstream analyzing and processing systems. As we described in one of our early blog post subscribing to all message is not really an option. All in all the broker is the central hub, which every message needs to pass. Therefore it is important, that it is highly scalable, integratable into backend systems, easy to monitor and of course failure-resistant.

MQTT Client

A MQTT client is any device from a micro controller up to a full fledged server, that has a MQTT library running and is connecting to an MQTT broker over any kind of network. This could be a really small and resource constrained device, that is connected over a wireless network and has a library strapped to the minimum or a typical computer running a graphical MQTT client for testing purposes, basically any device that has a TCP/IP stack and speaks MQTT over it. The client implementation of the MQTT protocol is very straight-forward and really reduced to the essence. That's one aspect, why MQTT is ideally suitable for small devices. MQTT client libraries are available for a huge variety of programming languages, for example Android, Arduino, C, C++, C#, Go, iOS, Java, JavaScript, .NET.

MQTT Quality of Service

- QoS 0 – at most once
- QoS 1 – at least once
- QoS 2

MQTT Payload

actual data to transmit in byte format

MQTT Publisher

After a MQTT client is connected to a broker, it can publish messages. MQTT has a topic-based filtering of the messages on the broker

MQTT Subscribe / Unsubscribe

Publishing messages doesn't make sense if no one ever receives the message, or, in other words, if there are no clients subscribing to any topic. A client needs to send a SUBSCRIBE message to the MQTT broker in order to receive relevant messages. A subscribe message is pretty simple, it just contains a unique packet identifier and a list of subscriptions.

TABEL DATA

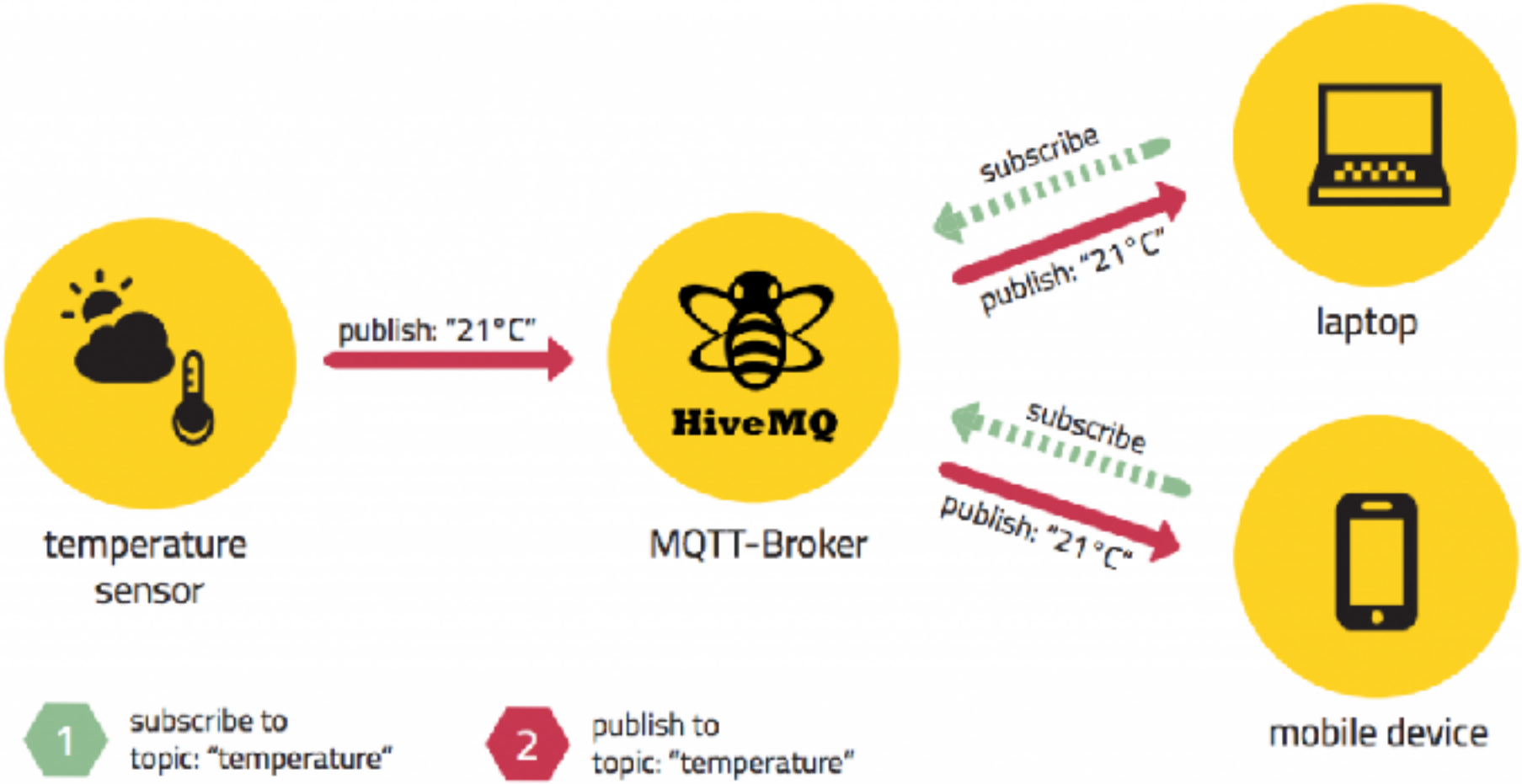
DAIMN
AUTOMATION | INDUSTRIAL SERVICE | MONITORING

Eclipse Mosquitto

MQTT Server

- https://iot.eclipse.org/getting-started
- https://www.cloudmqtt.com/
- https://www.hivemq.com/
- **Mosquitto**

Architecture



Tools development

MQTTBox

TABEL DATA

DAIMN
AUTOMATION | INDUSTRIAL SERVICE | MONITORING

Show Me The Code

Setup Project Android Studio

```
android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.tabeldata.mqtt.demo"
        minSdkVersion 21
        targetSdkVersion 21
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    lintOptions {
        abortOnError false
    }
    dexOptions {
        preDexLibraries = false
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

repositories {
    maven {
        url "https://repo.eclipse.org/content/repositories/paho-snapshots/"
    }
}

dependencies {
    compile 'com.fasterxml.jackson.core:jackson-databind:2.9.3'
    compile 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.0'
    compile 'org.eclipse.paho:org.eclipse.paho.android.service:1.1.1'
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:26.1.0'
    implementation 'com.android.support.constraint:constraint-layout:1.0.2'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.1'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.1'

}
```

Repositories

- https://repo.eclipse.org/content/repositories/paho-snapshots/

Dependencies

- com.fasterxml.jackson.core:jackson-databind:2.9.3
- org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.0
- org.eclipse.paho:org.eclipse.paho.android.service:1.1.1

Access permission

Setup AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.tabeldata.mobile.hmi">

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Demo MQTT Android"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <service android:name="org.eclipse.paho.android.service.MqttService" />
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Connect to Eclipse Mosquitto

```
public MqttAndroidClient mqttAndroidClient;

/**
 * generate client id with org.eclipse.paho.client.mqttv3.MqttClient;
 */
final String clientId = MqttClient.generateClientId();
final String serverUri = "tcp://iot.eclipse.org:1883";

public MqttHelper(Context context) {
    mqttAndroidClient = new MqttAndroidClient(context, serverUri, clientId);
    MqttConnectOptions mqttConnectOptions = new MqttConnectOptions();
    mqttConnectOptions.setAutomaticReconnect(true);
    mqttConnectOptions.setCleanSession(false);

    try {

        mqttAndroidClient.connect(mqttConnectOptions,  userContext: null, new IMqttActionListener() {
            @Override
            public void onSuccess(IMqttToken asyncActionToken) {

                DisconnectedBufferOptions disconnectedBufferOptions = new DisconnectedBufferOptions();
                disconnectedBufferOptions.setBufferEnabled(true);
                disconnectedBufferOptions.setBufferSize(100);
                disconnectedBufferOptions.setPersistBuffer(false);
                disconnectedBufferOptions.setDeleteOldestMessages(false);
                mqttAndroidClient.setBufferOpts(disconnectedBufferOptions);
                subscribeToTopic();
            }

            @Override
            public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
                Log.w( tag: "Mqtt",  msg: "Failed to connect to: " + serverUri + exception.toString());
            }
        });

    } catch (MqttException ex) {
        ex.printStackTrace();
    }
}
```

Publish message to broker

```
final String topicPublisher = "tabeldata/temp-publisher";

/**
 * untuk mempublish message temp
 *
 * @param temperatur
 */
public void publish(Temperatur temperatur) {
    try {
        JSONObject detail = new JSONObject();
        JSONObject value = new JSONObject();
        value.put( name: "temp", new JSONArray(Arrays.asList(temperatur.getTemp())));
        value.put( name: "humidity", new JSONArray(Arrays.asList(temperatur.getHumidity())));

        detail.put( name: "d", value);
        detail.put( name: "ts", DateFormater.toISO8601UTC(new Date()));
        Log.i( tag: "tempPublisher",  msg: "publish: " + detail.toString());

        MqttMessage publishMessage = new MqttMessage();
        publishMessage.setQos(0);
        publishMessage.setPayload(detail.toString().getBytes());
        mqttAndroidClient.publish(
                topicPublisher, publishMessage,  userContext: null, new IMqttActionListener() {
                    @Override
                    public void onSuccess(IMqttToken asyncActionToken) {
                        Log.w( tag: "publish",  msg: "onSuccess: berhasil terkirim");
                    }

                    @Override
                    public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
                        Log.w( tag: "publish",  msg: "onFailure: gagal publish message");
                    }
                });
    } catch (MqttException me) {
        me.printStackTrace();
    } catch (JSONException je) {
        je.printStackTrace();
    }
}
```

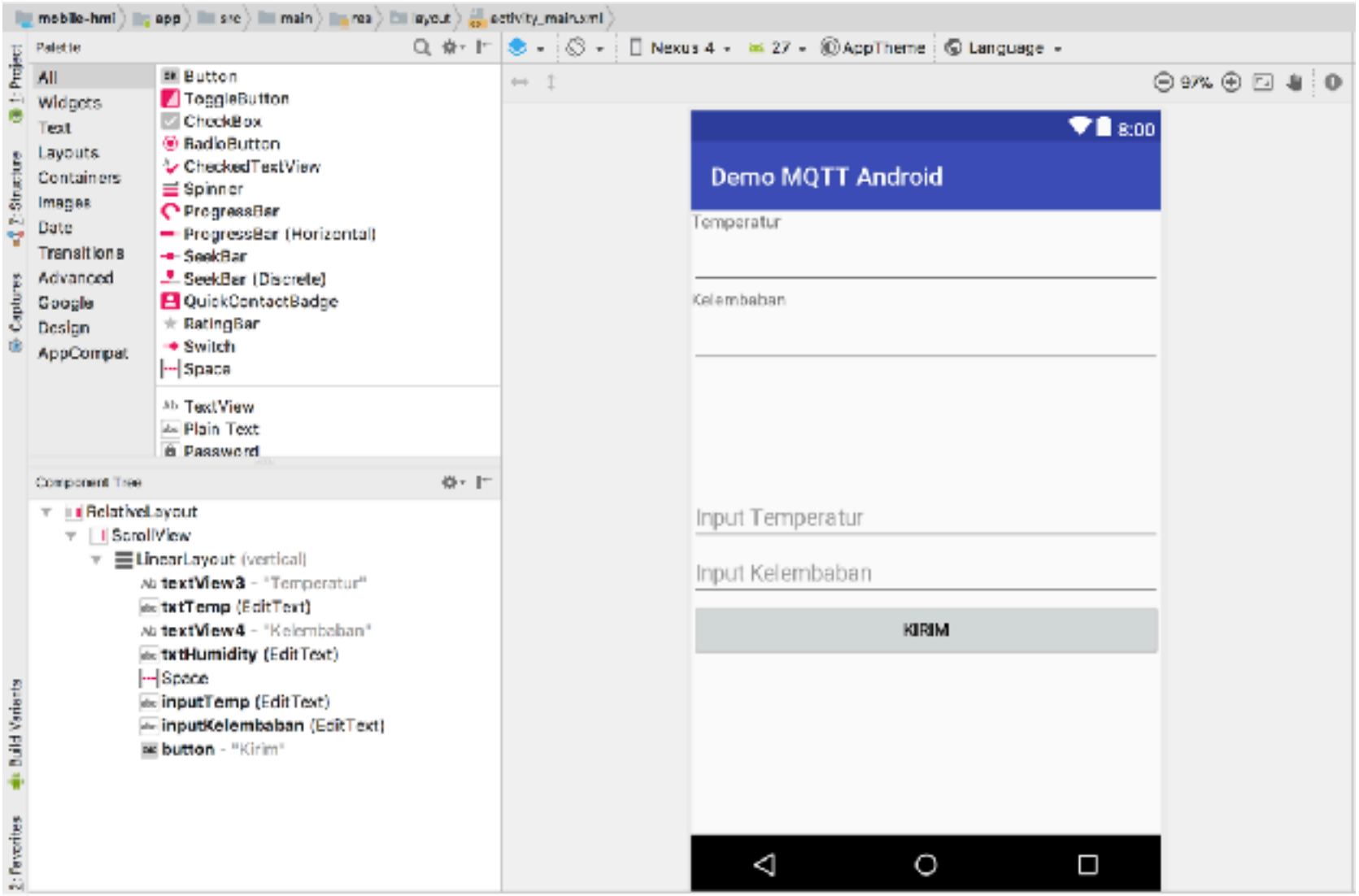
Subscribe topic from Broker

```
final String topicSubscribe = "tabeldata/temp-subcribe";

private void subscribeToTopic() {
    try {
        mqttAndroidClient.subscribe(topicSubscribe,  qos: 0,  userContext: null, new IMqttActionListener() {
            @Override
            public void onSuccess(IMqttToken asyncActionToken) {
                Log.w( tag: "MqttTemp",  msg: "Subscribed!");
            }

            @Override
            public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
                Log.w( tag: "Mqtt",  msg: "Subscribed fail!");
            }
        });
    } catch (MqttException ex) {
        System.err.println("Exceptionst subscribing");
        ex.printStackTrace();
    }
}
```

MQTT Callback Message

```
private void startMqtt() {
    mqttHelper = new MqttHelper(getApplicationContext());
    mqttHelper.setCallback(new MqttCallbackExtended() {

        @Override
        public void messageArrived(String topic, MqttMessage mqttMessage) throws Exception {
            JSONObject json = new JSONObject(mqttMessage.toString());
            if (topic.equalsIgnoreCase(mqttHelper.getTopicTemp())) {
                JSONObject detailTemp = new JSONObject(json.getString( name: "d"));
                temp.setTemp(detailTemp.getJSONArray( name: "temp").getDouble( index: 0));
                temp.setHumidity(detailTemp.getJSONArray( name: "humidity").getDouble( index: 0));

                txtTemp.setText(temp.getTempDecimal());
                txtHumidity.setText(temp.getHumidityDecimal());
            }
        }

        @Override
        public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
            try {
                MqttMessage message = iMqttDeliveryToken.getMessage();
                Log.i( tag: "publisher",  msg: "deliveryComplete: " + new String(message.getPayload()));
                Toast.makeText( context: MainActivity.this,  text: "Pesan terkirim", Toast.LENGTH_SHORT).show();
            } catch (MqttException e) {
                Log.e( tag: "pesan terkirim",  msg: "deliveryComplete: " + e.getMessage());
                e.printStackTrace();
            }
        }

        @Override
        public void connectComplete(boolean b, String s) {
            Toast.makeText( context: MainActivity.this,  text: "Koneksi berhasil", Toast.LENGTH_SHORT).show();
        }

        @Override
        public void connectionLost(Throwable throwable) {
            Log.i( tag: "connection",  msg: "connectionLost: connection terputus!");
            Toast.makeText( context: MainActivity.this,  text: "Koneksi internet terputus", Toast.LENGTH_SHORT).show();
        }
    });
}
```

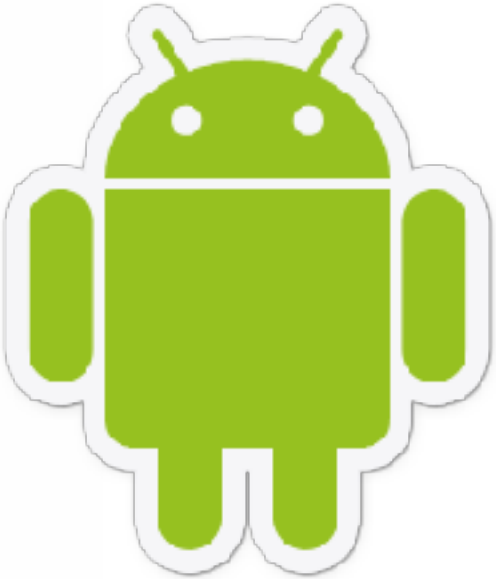
Main Activity Layout



TABEL DATA

DAIMN
AUTOMATION | INDUSTRIAL SERVICE | MONITORING

Demo Application



Source Code Android Project

- https://github.com/tabeldatadotcom/android-mqtt-example

Daftar Pustaka

- https://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe
- https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels
- https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt
- https://www.hivemq.com/wp-content/uploads/pub-sub-mqtt-1024x588.png
- https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment