**CRESPRIT MQTT Sample App**

**Guideline for Android**

January 06, 2015

**CRESPRIT Inc.**

**Table of Contents**

[1. Overview 3](#_Toc408233614)

[2. Android SampleCode 4](#_Toc408233615)

[3. Node.js Sample Code 10](#_Toc408233616)

[4. 시나리오 14](#_Toc408233617)

# Overview

본 문서는 ALOOH MQTT 메시지 서버를 이용하여 ALOOH 디바이스로부터 전달되는 센서데이터를 모니터링 하거나 ALOOH 디바이스로 메시지 데이터를 전달하기 위한 MQTT 샘플 App에 대한 가이드이다. ALOOH MQTT 메시지 서버를 이용한 모든 데이터는 JSON 형태로 전달된다.

해당 MQTT 샘플 App을 이용하기 위해서는 ALOOH 클라우드 서버에 계정이 생성되어야 하며 안드로이드용 설정 App을 통해 ALOOH 디바이스가 등록이 되어야 한다.

Node.js용 샘플 App에서 로그인과 MQTT 메시지 Publish/Subscribe에 사용되는 인증키 와 FEED ID는 안드로이드 샘플 앱을 통해 확인된 값을 참조하여 적용하여야 한다. 단 FEED ID 값은 ALOOH 클라우드 서버에서도 확인이 가능하다.

# Android SampleCode

1. **패키지 : com.cresprit.mqtt**

* Source
* paho-mqtt-client-1.0.1.jar(MQTT Library)

1. **Source**
2. Manager

* Connection.java : MQTT 라이브러리를 통해 MQTT 서버와 통신
* ConnectionMgr.java : connection 객체 관리 및 서버 URL 정의, 수신 subscribe 메시지 관리
* DeviceListManager.java : ALOOH 서버상에 등록된 디바이스 정보 요청 및 관리
* UserManager.java : 로그인 및 MQTT 인증키 보유

1. Service

* MQTTService.java : subscribe를 위한 서비스

1. Ui

* \*Activity : 각 화면 구성
* \*Adapter : Device및 subscribe 수신 메시지 리스트에 대한 Adapter
* SubscribeCallback : subscribe를 위한 MQTT 라이브러리 인터페이스

1. **Source OverView**
2. Login

* LoginAcitivity에서는 UserManager 클래스의 LoginTask(AsyncTask)에서 통해 ConnectionMgr.SERVER\_API\_LOGIN\_URL(서버 API)를 통해 Login을 시도한다. 서버 API로 통해 전달되는 모든 데이터는 JSON 형식을 따른다.
* 아래 소스코드의 **key**값은 이후 ALOOH 서버 API 및 MQTT Server로 Publish및 Subscribe를 위한 인증키 값으로 사용된다.
* **key**값은 LoginActivity에서 로그인에 대한 결과를 받기 위해 등록한 IUpdateListener의 update() 콜백메소드를 통해 key값을 체크한 뒤 UserManager클래스의 멤버변수 m\_pAuthKey로 저장된다.

**UserManager.java**

…

class LoginTask extends AsyncTask<String, Void, String> {

String key=null;

@Override

protected String doInBackground(String... params1) {

…

request.setURI(new URI(**ConnectionMgr.SERVER\_API\_LOGIN\_URL**));

…

HttpEntity entity = response.getEntity();

String jsonStr = EntityUtils.toString(entity);

try {

JSONObject resObj = new JSONObject(jsonStr);

JSONObject resData = resObj.getJSONObject(JSON\_ID\_DATA);

**key = resData.getString(JSON\_ID\_KEY);**

…

}

@Override

protected void onPostExecute(String \_key) {

if(mListener != null)

**mListener.update(IUpdateListener.REMOVE\_DIALOG, \_key);//콜백호출**

super.onPostExecute(key);

}

…

**LoginActivity.java**

**Protected void onCreate(…){**

…

UserManager userMgr;

userMgr = UserManager.getInstance(LoginActivity.this);

**userMgr.setDialogUpdateListener(listener);//콜백등록**

…

**}**

**private IUpdateListener listener = new IUpdateListener()**

**{**

@Override

public void update(int status, String \_key) {

// TODO Auto-generated method stub

…

**UserManager.getInstance(LoginActivity.this).setAuthKey(\_key);**

…

**};**

1. 장치 리스트

* DeviceListActivity에서는 DeviceListManager 클래스의 DeviceList(AsyncTask)에서 ConnectionMgr.SERVER\_API\_GET\_DEVICE\_LIST\_URL(서버 API)을 통해ALOOH서버로부터 해당 계정에 속한 모든 ALOOH 디바이스를 가져온다.
* 서버 API에서 request 헤더에 “Authorization”을 로그인시에 서버로부터 획득한 UserManager클래스의 m\_pAuthKey 값으로 전달한다.

**DeviceListManager.java**

class DeviceList extends AsyncTask<String, Void, Boolean> {

…

@Override

protected Boolean doInBackground(String... key) {

…

HttpPost request = new HttpPost();

request.setHeader("Content-Type", "application/json");

**request.setHeader("Authorization", "Bearer "+key[0]);**

request.setURI(new URI(**ConnectionMgr.SERVER\_API\_GET\_DEVICE\_LIST\_URL**));…

…

HttpEntity entity = response.getEntity();

String jsonStr = EntityUtils.toString(entity);

try {

JSONObject resObj = new JSONObject(jsonStr);

JSONObject resData = resObj.getJSONObject("data");

JSONArray deviceArray = resData.getJSONArray("devices");

for(int i=0; i<deviceArray.length();i++){

JSONObject device = deviceArray.getJSONObject(i);

String name = device.getString("name");

String feedId = device.getString("**feed\_id**");

String activationCode = device.getString("activation\_code");

String productName = device.getString("product\_name");

deviceList.add(new DeviceInfo(name, feedId, productName, activationCode));

}

…

}

**DeviceListActivity.java**

@Override

protected void onCreate(Bundle savedInstanceState) {

…

device.getDeviceList(UserManager.getInstance(DeviceListActivity.this).getAuthKey());

…

**};**

1. Publish

* PublishActivity에서는 ConnectionMgr 클래스의 getConnectionInstance()메소드를 통해 Publish를 위한 Connection클래스의 오브젝트를 가져와서Connection클래스의 publish()메소드를 통해 publish를 수행한다.
* Connection클래스의 setUsername() 메소드를 통해 로그인시 서버로부터 획득한 UserManager클래스의 m\_pAuthKey값으로 Username을 설정한다.
* DeviceListManager클래스에 저장된 FEED ID(getFeedId())를 매개변수로 하여 publish() 메소드를 호출을 통해 Publish를 수행한다.

**PublishActivity.java**

@Override

protected void onCreate(Bundle savedInstanceState) {…

…

btnPublish.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

message = edtMessage.getText().toString();

…

connection = ConnectionMgr.getConnectionInstance(**ConnectionMgr.MQTT\_SERVER\_URL**);

…

connection**.setUsername**(**UserManager.getInstance(PublishActivity.this).getAuthKey()**);

connection.connect();

connection.setMessage(message);

try {

connection.publish(deviceInfo.getFeedId());

} catch (MqttException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

});

…

}

1. Subscribe

* SubscribeAcitivity에서는 MQTTService.class를 통해 Subscribe를 수행한다.

Subscribe는 데몬 형태로 실시간으로 전달되는 MQTT메시지를 수신하기 위해 안드로이드 Service 컴포넌트를 이용한다.

* MQTTService 클래스에서는 MQTT 서버 URL( **ConnectionMgr.MQTT\_SERVER\_URL)**

을 인자로 하여 Connection클래스의 오브젝트를 생성하며 **UserManager**클래스의 **m\_pAuthKey**값으로 Username인증키를 설정한다.

* DeviceListManager클래스에 저장된 FEED ID(getFeedId())를 매개변수로 하여 Connection클래스의 subscribe() 메소드 호출을 통해 Subscribe를 수행한다.

이후 여타의 MQTT Client로부터 Publish되는 MQTT메시지 중 동일 FEED ID를 가지는 모든 MQTT 메시지를 수신한다.

**SubscribeActivity.java**

@Override

protected void onCreate(Bundle savedInstanceState) {…

…

final Intent intent = new Intent(SubscribeActivity.this, MQTTService.class);

intent.putExtra(DeviceListManager.SELECT\_DEVICE, mSelectedDevice);

startService(intent);

…

}

**MQTTService.java**

public class MQTTService extends Service {

…

@Override

public void onStart(Intent intent, int startId) {

try {

connect = ConnectionMgr.getConnectionInstance(ConnectionMgr.MQTT\_SERVER\_URL);

connect.setUsername(UserManager.getInstance(this).getAuthKey());

deviceInfo = DeviceListManager.getInstance(this).getDeviceList().get(mSelectedDevice);

connect.subscribe(this, deviceInfo.getFeedId());

} catch (MqttException e) {

…

e.printStackTrace();

}…

# Node.js Sample Code

해당 Node.js 파일들은 ALOOH 디바이스상에서 제공하는 IDE(NOIDE)에서 개발, 수정 및 구동을 확인할 수 있다.

1. **Source 구성**
2. Publish.js : Publish를 위한 Node.js 파일
3. Subscribe.js : Subscribe를 위한 Node.js 파일
4. **SourceCode**
5. Publish.js

* 하기 샘플코드는 ALOOH 디바이스의 **A3 Pin(ADC)**에 연결된 **온도센서**로부터 온도데이터를 읽고 ALOOH MQTT 서버로 **JSON형식의 온도데이터를** **Publish** 하는 것을 목적으로 한다.
* 붉은색으로 표기된 인증키(Auth Key)와 FEED ID는 안드로이드 샘플 앱을 통해 사용자에게 할당된 키 값을 확인하여 적용하여야 한다.

var mqtt = require('mqtt'), url = require('url');

var ADC = require('snow/adc');

var adc = new ADC.adc();

**//Auth Key 수정필요**

**var mqtt\_url = url.parse('mqtt://b762f251aba841ceb0cf7d7dcdb41c12:test@api.alooh.io:1883');**

var auth = (mqtt\_url.auth || ':').split(':');

var result;

var dataString;

**adc.open();//ADC 인터페이스로 연결**

**adc.attach(3);//ADC 인터페이스에 대상 Pin(A3)을 설정한다.**

**var client = mqtt.createClient(mqtt\_url.port, mqtt\_url.hostname, {**

**username: auth[0],// b762f251aba841ceb0cf7d7dcdb41c12**

**password: auth[1]**

**});**

setInterval(function(){

**input = adc.read(3, 1000);//A3 pin으로부터 1초에 한번씩 데이터를 읽음**

voltage = (input\*3.3/4096);

temperature = (voltage)\*100;

result = temperature.toFixed(1);

var time = new Date();

var data = { data:

{ data\_streams:

[

{

id: auth[0],

data\_points: [

{

**v: result,**

t: time.toString()

}

]

}

]

}

}

dataString = JSON.stringify(data);

**//FEED ID 수정필요**

**client.publish('54a26953d62e1c0b0a78ee65', dataString, function() {**

console.log('temperature : %j',dataString);

});

}, **1000);//1초 단위 반복**

1. Subscribe.js

* 하기 샘플코드는 ALOOH 디바이스의 GPIO 인터페이스 D3 Pin에 LED를 연결하고

Publish를 수행하는 타 MQTT 클라이언트 장치(Android Sample App)로부터 MQTT 메시지를 받아 LED를 ON, OFF, 점멸 하는 것을 목적으로 한다.

var mqtt = require('mqtt'), url = require('url');

var GPIO = require('snow/gpio');**//GPIO를 위한 모듈**

**//Auth Key 수정필요**

**var mqtt\_url = url.parse('mqtt://b762f251aba841ceb0cf7d7dcdb41c12:test@pino.io:1883');**

var auth = (mqtt\_url.auth || ':').split(':');

**var nGpio = 3;**

var index = 0;

var gpio = new GPIO.gpio();

var intervalHandler;

gpio.open(); **//GPIO D3번 Pin을 연결**

gpio.pinMode(nGpio, 1); **//해당 핀을 출력전용으로 설정(입력: 0)**

function ledStatus(){

if(index < count\*2)

{

if(index == 0 || index%2 == 0)

gpio.digitalWrite(nGpio, 1);**//LED를 켬**

else

gpio.digitalWrite(nGpio, 0);**//LED를 끔**

}

index++;

if(index == (count+2)\*2)

index = 0;

};

function interval(arg)

{

if(arg == 0)

gpio.digitalWrite(nGpio, 1);**//LED를 켬**

else

{

count = arg;

intervalHandler = setInterval(ledStatus, 200);

}

};

var client = mqtt.createClient(mqtt\_url.port, mqtt\_url.hostname, {**//MQTT 클라이언트 생성**

username: auth[0], **// b762f251aba841ceb0cf7d7dcdb41c12**

password: auth[1]

});

**//FEED ID 수정필요**

client.subscribe(**'54a26953d62e1c0b0a78ee65**', function() {**//메시지 수신 Callback**

client.on('message', function(topic, **message**, packet) {

if(message == "on" || message == "ON" || message == "On")

{

gpio.digitalWrite(nGpio,1);**//기존 LED 점멸중인 경우 Interval 함수 중지**

clearInterval(intervalHandler);

}

if(message == "off" || message == "OFF" || message == "Off")

{

gpio.digitalWrite(nGpio,0); **//기존 LED 점멸중인 경우 Interval 함수 중지**

clearInterval(intervalHandler);

}

if(message == "repeat2")

{

clearInterval(intervalHandler); **//기존 LED 점멸중인 경우 Interval 함수 중지**

interval(2);

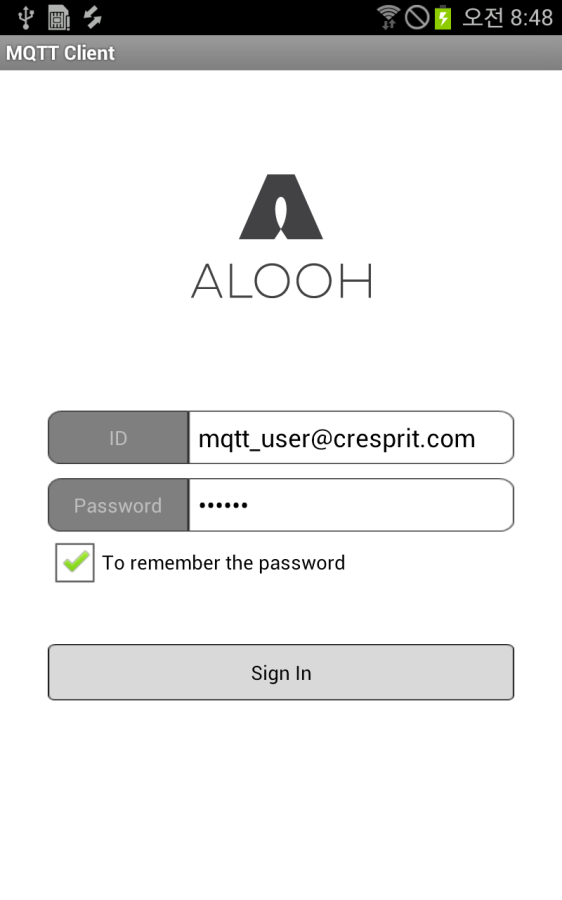
}

});

});

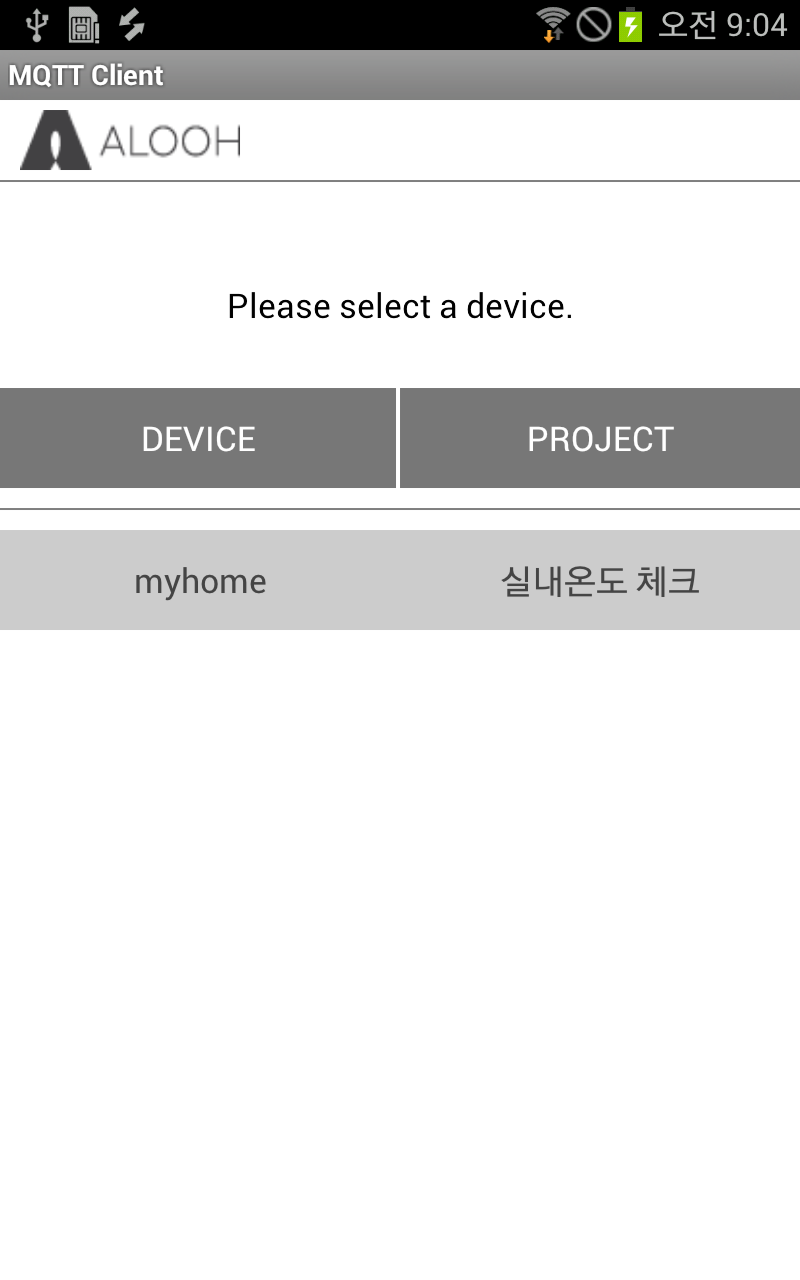
# 시나리오

1. 로그인

* 로그인 화면은 아래와 같이 구성된다. 로그인을 위한 계정은 ALOOH 홈페이지(<http://www.alooh.io/>)나 안드로이드용 ALOOH 디바이스 설정 App을 통해서 생성이 가능하다.

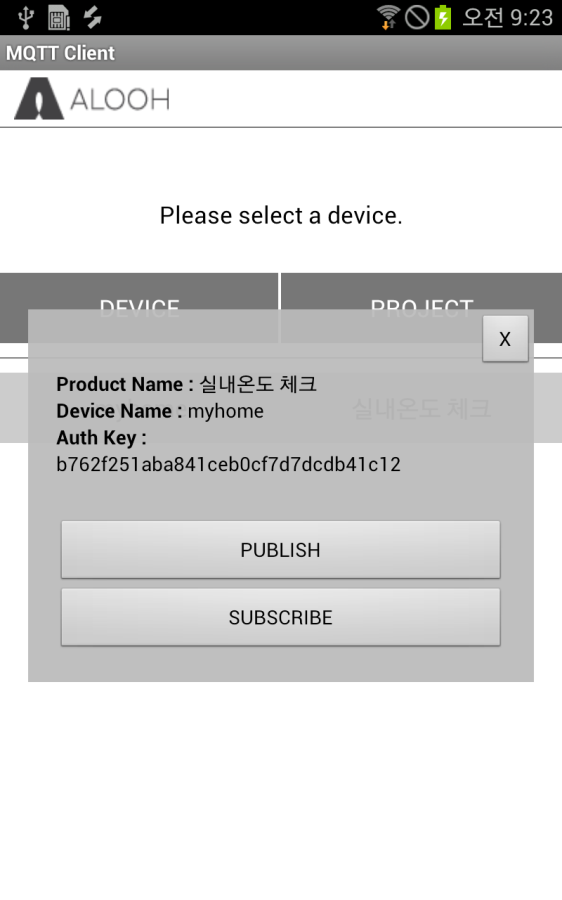
1. ALOOH 디바이스 선택

* 안드로이드 ALOOH 디바이스 설정 App을 통해 등록된 자신이 보유한 모든 ALOOH 디바이스와 디바이스가 속한 프로젝트명을 함께 보여준다. 사용하고자 하는 ALOOH 디바이스를 선택을 한다.



1. Publish/Subscribe 선택화면

* 등록된 장치를 선택하면 Publish및 Subscribe 버튼과 더불어 해당 장치에 할당된 인증키값(AuthKey)이 보여진다. 해당 인증키는 Publish나 Subscribe를 수행할 때 ALOOH MQTT 서버에서 권한을 확인하기 위한 키로 사용된다.
* PUBLISH : ALOOH 클라우드 서버를 통해 생성된 프로젝트에는 TOPIC(FEED) 이라 칭하는 키가 생성이 되는데 특정 TOPIC(FEED)에 대해 Subscribe를 신청한 ALOOH 디바이스에게 메시지를 보내는 역할을 한다. Subscribe를 신청한 여타의 MQTT 클라이언트들도 메시지 수신이 가능하다.
* SUBSCRIBE : 특정 TOPIC(FEED) 에 대해서 ALOOH 디바이스나 여타의 MQTT 클라이언트에서 Publish를 통해 배포되는 메시지를 수신한다.



1. Publish

* 3. Publish/Subscribe 선택화면에서 확인한 인증키 값으로 ALOOH MQTT 서버로 커넥션을 맺고 하기 캡쳐화면에 표시된 TOPIC을 통해 작성한 메시지를 전달한다. 이로 인해 해당 TOPIC으로 Subscribe를 등록한 모든 MQTT 클라이언트는 해당 메시지를 수신하게 된다.
* Publish의 결과에 따라 토스트 팝업 창이 표시된다.



1. Subscribe

* 3. Publish/Subscribe 선택화면에서 확인한 인증키 값으로 ALOOH MQTT 서버로 커넥션을 맺고 하기 캡쳐화면에 표시된 TOPIC을 통해 Subscribe를 등록한다. 이로 인해 ALOOH 디바이스 혹은 여타의 MQTT 클라이언트로부터 해당 TOPIC으로 Publish를 통해 배포되는 메시지를 실시간으로 수신하게 된다

