# Assignment 3 - Report

0713221 黃家綺 / 109550172 何彥寬

## 1. Objective:

- 實現全自動智慧植物系統，讓使用者不用額外耗費心力。
- 利用MCS 讓兩台Raspberry Pi 得以互相溝通。
- Main Raspberry Pi 加上了澆花系統與5050燈板，利用自動分析去開啟關閉它們。
- Second Raspberry Pi 提供手動控制，以及利用 Raindrop Sensor 去判斷天氣為何藉由Line Notify 傳送給使用者天氣狀態，也用一個RGB LED 讓使用者能從它的顏色去判斷現在土壤濕度為何。

## 2. Sensors and Actuators Used:

- Soil Moisture Sensor
- Light Sensor
- I2C LCD
- DHT22
- Buzzer
- RGB LED
- Raindrop Sensor
- 4 Channel Capacitive Touch Module

**Extra:**

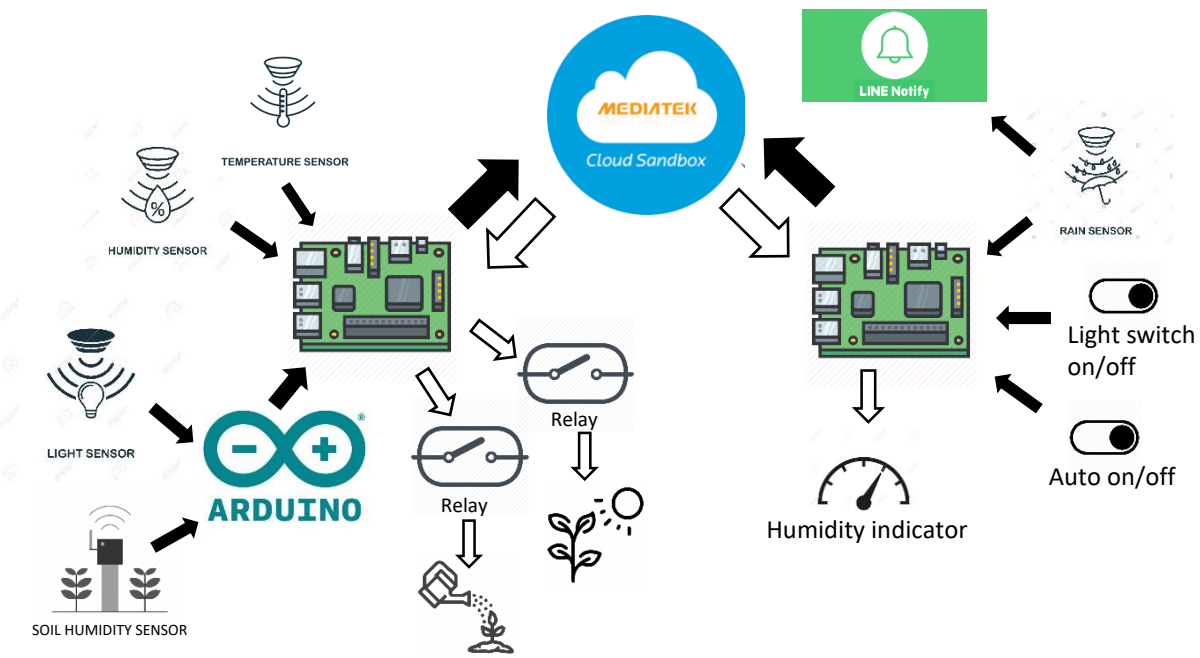
- Arduino
- Light system
  - 12V 5050 12SMD LED White Light
  - Relay Module
  - Transformer
  - Plug
- Watering system
  - Mini Submersible Water Pump
  - Water Pipe
  - Relay Module
  - Bottle with Water

## 3. System Design:

- 在Lab2 的時候我們是用Buzzer 與Email 去提醒主人該澆水了，以及純粹判斷亮度然後用LED 當作電燈去開關。這次我們直接將澆水系統實作出來，利用Mini Submersible Water Pump 抽水再經由水管直接連接到植物，讓它可以由系統去自動照顧植物，而燈也改用 12V 5050 12SMD LED 去提供足夠的光線，也加上了實際時間讓它可以控制在甚麼時段內光線不足才需要開燈，藉此控管植物照光時數，讓植物照著理想狀態生長。

- 另一台Raspberry Pi 用了一個4 Channel Capacitive Touch Module 去提供手動控制燈光，它分成4個選項，手動、自動，以及在手動模式時的開與關，讓人在特殊情況時可以改由自行控制燈的開關。我們也加上了利用 Raindrop Sensor 去判斷天氣為何，再藉由Line Notify 傳送給使用者天氣狀態，也用一個RGB LED 配合另一台裝置上傳到MCS運算後的資料讓使用者能從它的顏色去判斷土壤濕度為何。我們將使用者的選擇與天氣狀況上傳至MCS ，所以我們也能藉由MCS看到現在的狀態，也藉此與另一台裝置溝通。

## 4. Flowchart:



# 5. Code Explanation and The Differences With Example Codes:

**(You can find all the source code in the file named source code or on https://github.com/Kris57880/raspberry_git )**

1) Source Code which Changed or Added:
   - Main Raspberry Pi
     - mcs_get
     - relay (單純連結relay讓它開)
   - Second Raspberry Pi
     - mcs_get_other
     - mcs_upload_other
     - raindrop
     - touch_switch
     - Line_notifier
     - RGB_LED

2) Main Raspberry Pi:
   a) mcs_get:

在Lab3 中，我們對light sensor新增了判斷合適光照時間的function，

```
41  if value == "on" : #manual mode
42      endpoint = "/mcs/v2/devices/" + deviceId_client + "/datachannels/man_ctl/datapoints"
43      url = host + endpoint
44      r = requests.get(url,headers=headers)
45      value = (r.json()["dataChannels"][0]["dataPoints"][0]["values"]["value"])
46      if value : GPIO.output(light,GPIO.HIGH)
47      else :  GPIO.output(light,GPIO.HIGH)
48  else : #auto mode
49      endpoint = "/mcs/v2/devices/" + deviceId_host + "/datachannels/ledswitch/datapoints"
50      url = host + endpoint
51      r = requests.get(url,headers=headers)
52      value = (r.json()["dataChannels"][0]["dataPoints"][0]["values"]["value"])
53      if(value==1):
54          if (now_time >=17 or now_time <= 5):
55              print("light will turn on ")
56              GPIO.output(light,GPIO.HIGH)
57          else :
58              GPIO.output(light,GPIO.LOW)
59      else :
60          if (now_time <=17 and now_time>=5):
61              GPIO.output(light,GPIO.LOW)
```

我們先從mcs client (另一台樹梅派)的資料通道獲取模式設定，若切入手動模式，則再次讀取client 上的手動開關，決定是否開啟燈光，而模式切換與手動開關的code 將在touch_switch 中說明。
當模式切換成自動模式時，他並不會單純的在缺乏光源時開關，我們控制光照時間，保留足夠的黑暗期供植物生長，雖然上面是設定在傍晚17時到凌晨5時亮度不足，開啟燈光，讓其保持長時間日照，但只要調整設定的區間，隨時可以輕鬆調整。

3) Second Raspberry Pi

a) mcs_get_other:

```
host = "http://api.mediatek.com"
headers = {"Content-type": "application/json", "deviceKey": deviceKey}

def rgb():
    endpoint = "/mcs/v2/devices/" + deviceId + "/datachannels/soil_led/datapoints"
    url = host + endpoint
    r = requests.get(url,headers=headers)
    value = (r.json()["dataChannels"][0]["dataPoints"][0]["values"]["value"])
    RGB_LED.display(int(value))

while True:
    rgb()
    time.sleep(5)
```

此程式是由mcs_get 去改，讓它適用於RGB LED 。它僅從MCS 上抓取soil_led 經由判斷式算出的Data channel，再由RGB_LED 根據結果顯示不同顏色。

b) mcs_upload_other:

```
# Set MediaTek Cloud Sand box (MCS) Connection
def post_to_mcs(mode, data):
    payload= {"datapoints":[{"dataChnId": mode,"values":{"value":str(data)}}]}
    headers = {"Content-type": "application/json", "deviceKey": deviceKey}
    not_connected = 1
    while (not_connected):
        try:
            conn = http.client.HTTPConnection("api.mediatek.com:80")
            conn.connect()
            not_connected = 0
        except (http.client.HTTPException, socket.error) as ex:
            print ("Error: %s" % ex)
        time.sleep(1) # sleep 10 seconds
        conn.request("POST", "/mcs/v2/devices/" + deviceId + "/datapoints",
        json.dumps(payload), headers)
        response = conn.getresponse()
        print( response.status, response.reason,# json.dumps(payload),
        time.strftime("%c"))
        data = response.read()
        conn.close()
```

此程式將觸控感應器的資料上傳到MCS 上，與原本的mcs_upload 的post_to_mcs 函式相同，僅Device ID 與 Device Key 改成了第二台Test Device 。post_to_mcs 針對不同的sensor 僅需加入不同引數讓其改變該上傳的Data channel 以及Data 數值，不須針對不同sensor 改變函式內的code 。

c) raindrop:

```python
import time
import RPi.GPIO as GPIO
import mcs_upload_other
import Line_notifier as notifier
status = 0
RAIN = 1
SUN = 0
pin = 40
GPIO.setmode(GPIO.BOARD)
GPIO.setup(pin ,GPIO.IN)
try :
    if GPIO.input(pin)==GPIO.LOW :
        notifier.line_message("It's raining.")
        status = RAIN
        mcs_upload_other.post_to_mcs('rain',1)
        print("It's raining.")

    else :
        notifier.line_message("It's sunny.")
        status = SUN
        mcs_upload_other.post_to_mcs('rain',0)
        print("It's sunny.")

    while True :
        if GPIO.input(pin)==GPIO.LOW:
            if status == SUN :
                notifier.line_message("It's raining.")
                status = RAIN
                mcs_upload_other.post_to_mcs('rain',1)
                print("It's raining.")

        else :
            if status == RAIN :
                notifier.line_message("It's sunny.")
                status = SUN
                mcs_upload_other.post_to_mcs('rain',0)
                print("It's sunny.")
        time.sleep(20)
except KeyboardInterrupt:
    print("close")
```

一開始先判斷初始時有沒有下雨，透過line_ notify 與 mcs_upload_other 通知與上傳資料，並利用status 去記錄狀態，接下來每次在讀取sensor 時，藉由判斷status 讓它在只有狀態改變時才會通知與上傳資料。

d) touch_switch:

```python
while True :
    try:
        if GPIO.input(O1_PIN)==GPIO.HIGH :
            if(count == 1) : continue
            count = 1
            mcs_upload_other.post_to_mcs('mode', 1)
            print("manual mode")
        elif GPIO.input(O2_PIN)==GPIO.HIGH :
            if(count == 2) : continue
            count = 2
            mcs_upload_other.post_to_mcs('mode', 0)
            print("auto mode")
        elif GPIO.input(O3_PIN)==GPIO.HIGH :
            if(count == 3) : continue
            count = 3
            mcs_upload_other.post_to_mcs('man_ctl', 1)
            print("manual on")
        elif GPIO.input(O4_PIN)==GPIO.HIGH :
            if(count == 4) : continue
            count = 4
            mcs_upload_other.post_to_mcs('man_ctl', 0)
            print("manual off")
        time.sleep(1)
    except KeyboardInterrupt:
        print("end")
```

Touch_switch 中，由於sensor 具有四個觸控按鍵，我們將每個按鍵做不同功能，1和2作為模式切換，而3和4作為手動模式中燈的開關。
3與4必須在手動開啟時才會作用，這些數值將會傳送至MCS 供另一台樹莓派讀取，這個過程由於另一台樹莓派的get_mcs 設定成每30秒才會讀取一次Data Channel ，所以反應會較慢，所以touch_switch 有設定time.sleep() 以減少data point負荷，設定1秒讓它同時也不會因為間隔太久導致按下去卻沒有反應的情況發生。
用count 多設一個判斷是為了讓它在按同一個選項時不會重複上傳。

e) Line_notifier

```python
import requests

def line_message(msg):
    token = "6TRvFcSuN4A8jsnHn5eUqjk9efXcq8OTur1HRFD7VBB"
    token2 = "zle1Bi1MvdQEW390Vf5WqM6tLdb0SaOksYONslBh3tM"
    headers = {
        "Authorization": "Bearer " + token,
        "Content-Type" : "application/x-www-form-urlencoded"
   }
    headers2 =  {
        "Authorization": "Bearer " + token2,
        "Content-Type" : "application/x-www-form-urlencoded"
   }
    payload = {'message': msg}
    r = requests.post("https://notify-api.line.me/api/notify",
    headers = headers, params = payload)
    r = requests.post("https://notify-api.line.me/api/notify",
    headers = headers2, params = payload)
    return r.status_code
```

Line_notifier 中，我們將下雨的資訊推播至Line 的聊天室中提醒我們外面正在下雨。我們先從LINE Notify 獲得不同帳號的權杖(token)，由於都是透過request 函式庫向網站請求資料輸入，code 架構與mcs_get 大同小異，不同的是我們必須配合line 的API ，在headers 的部分進行修改，在設計程式架構時我參考之前寫的i2c_lcd 同樣是函式輸入字串作為引數，再由本程式輸出到Line 裡面。

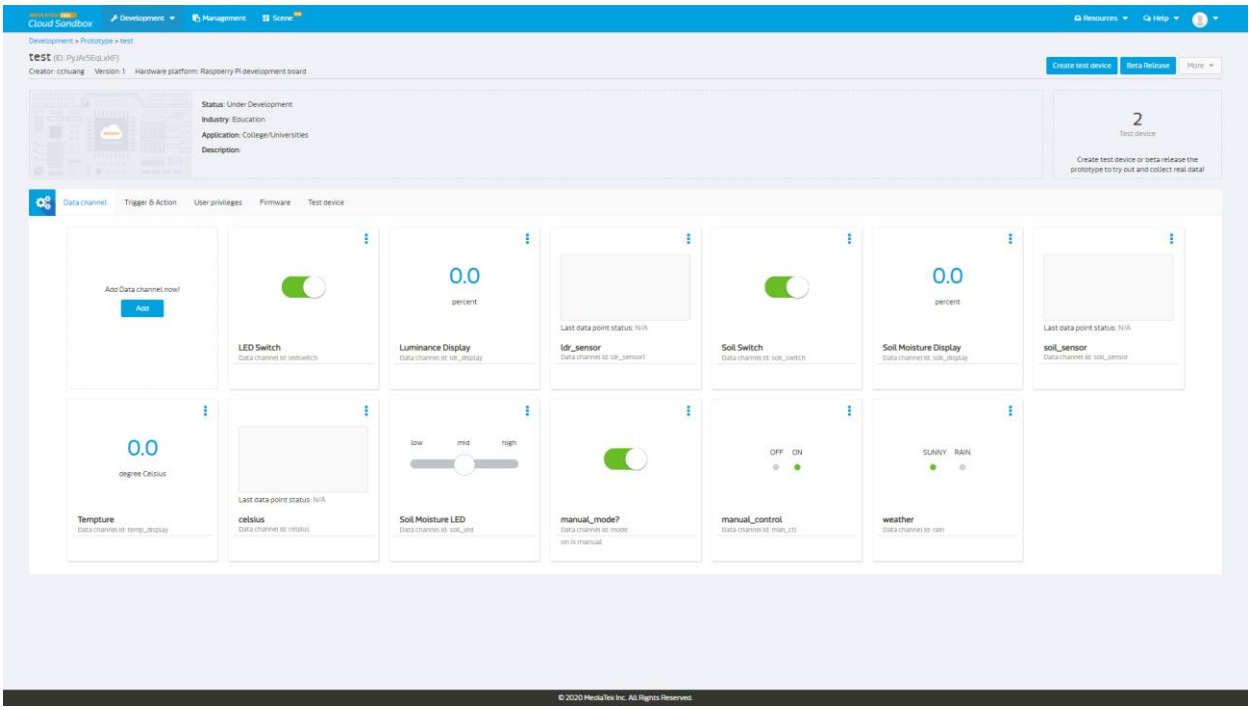
f) RGB_LED:

```
RED_LED_PIN = 13
BLUE_LED_PIN = 19
GREEN_LED_PIN = 26
PWM_FREQ = 200

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(RED_LED_PIN, GPIO.OUT)
GPIO.setup(BLUE_LED_PIN, GPIO.OUT)
GPIO.setup(GREEN_LED_PIN, GPIO.OUT)

red_pwm = GPIO.PWM(RED_LED_PIN, PWM_FREQ)
red_pwm.start(0)
blue_pwm = GPIO.PWM(BLUE_LED_PIN, PWM_FREQ)
blue_pwm.start(0)
green_pwm = GPIO.PWM(GREEN_LED_PIN, PWM_FREQ)
green_pwm.start(0)

def setColor(r=0, g=0, b=0):
    red_pwm.ChangeDutyCycle(100-int(r/255*100))
    blue_pwm.ChangeDutyCycle(100-int(b/255*100))
    green_pwm.ChangeDutyCycle(100-int(g/255*100))

def display(color):
    if color ==  1 :
        for i in range(0,30):
            setColor(255, 0, 0)
            time.sleep(1)
        #red
    elif color == 2 :
        for i in range(0,30):
            setColor(0, 0, 255)
            time.sleep(1)
        #blue
    elif color == 3 :
        for i in range(0,30):
            setColor(0, 255, 0)
            time.sleep(1)
        #green
```

RGB_LED可以透過單一LED顯示不同色彩，非常適合做為顯示土讓濕度的指示燈，這種可以調色的技術是基於PWM，透過調整頻率的方式模擬類比輸出，我們設定基礎頻率是200hz，再由setColor函式進行調光，它控制三種光源的強度以達到混色的效果。RPi.GPIO 的 PWM 工作週期範圍為 0~100，但是平常三原色範圍為 0~255，所以在此進行轉換。如果是對共陽極的全彩 LED 而言，工作週期是代表斷電的時間比例，而 int(r/255*100) 是我們希望通電的時間比例，所以要用 100 減掉後才是應該斷電的時間比例。而相對於共陽極，如果是以 PWM 來控制其亮度，則工作週期越高表示該腳位所代表的顏色就越亮，也就是比例就越高。

我們將get_mcs_other 中的值輸入，並分別以123的數值顯示紅綠藍三種色彩。

4) JavaScript on MCS




a) New Data Channel:

- Soil Moisture LED ( 控制RGB LED 狀態燈）
- Manual mode? （1為手動與0為自動模式）
- Manual control （顯示手動模式下現在電燈控制為開或關）
- Weather （顯示現在為晴天或雨天）

b) Source Code which Changed or Added:

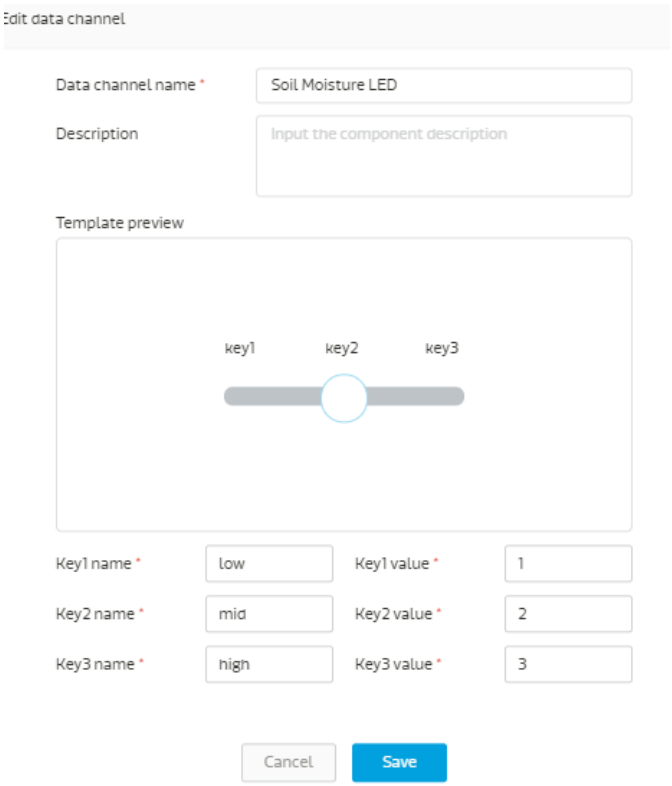
a. Soil sensor

因為新增了Soil Moisture LED 這個Data Channel 所以新增了區分土壤濕度為高中低的code ，並將其傳入Soil Moisture LED Data Channel。

```
var soil_level = context.value;
var percent = (1023-soil_level )/10.23;
if (percent <1 ) percent=percent.toFixed(3);
else percent=percent.toFixed(2);
var sw  = 0;
if (percent <40 ) sw =1 ;
else sw = 0;
var led = 0;
// 1是 low 2是mid 3是high
if (percent > 70 )  led = 3 ;
else if (percent > 50 ) led =2 ;
else led = 1;
return {soil_display : percent, soil_switch:sw, soil_led : led};

```

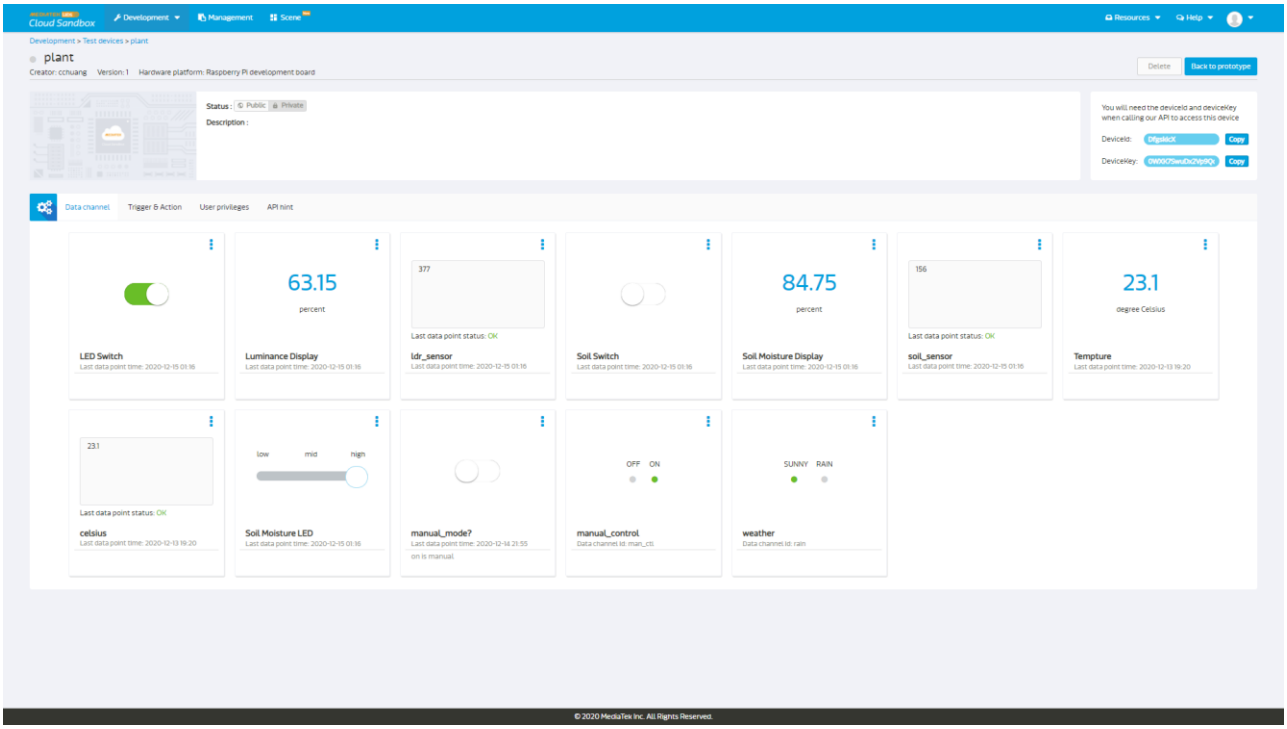
b. Soil Moisture LED

此Data Channel 為Controller 中的Category ，我們設了3個Categories 分別為High 、Mid 、Low ，並賦予他們Key value 分別為3、2、1，以此value 去接收soil sensor 回傳的值以及以此控制狀態燈。

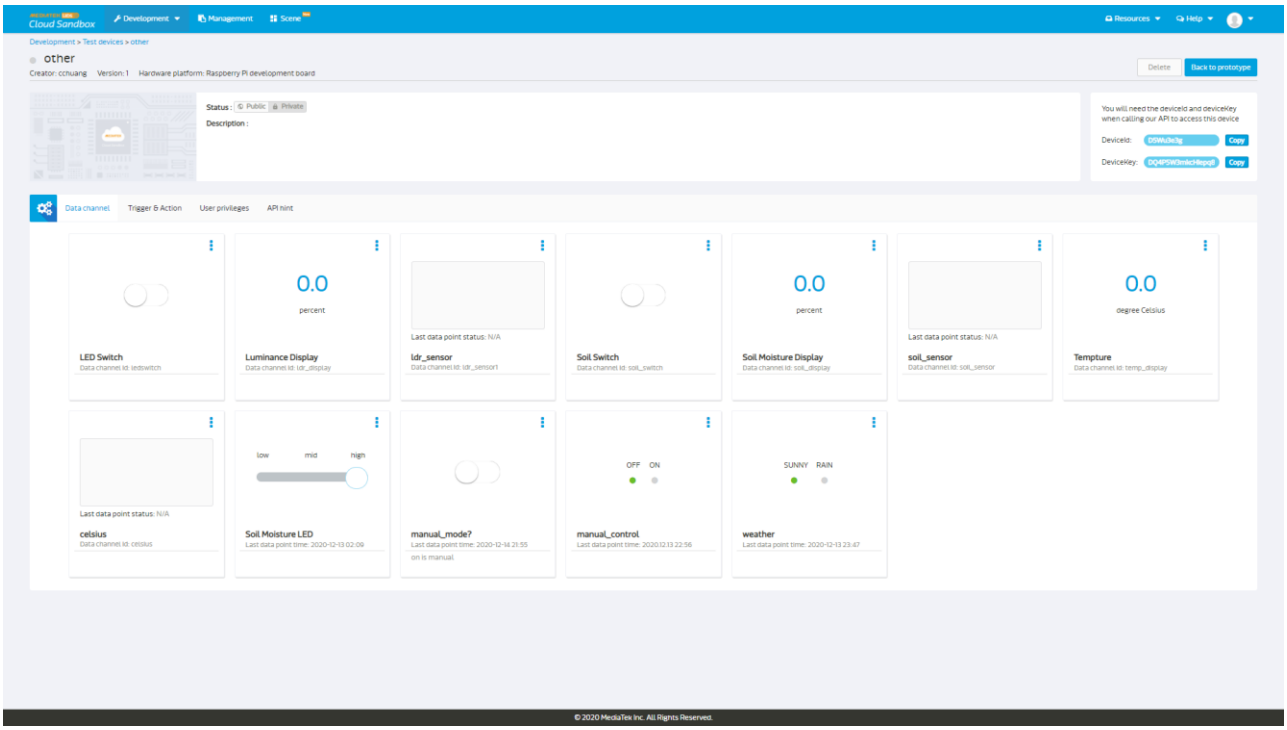


# 6. Test Device 實際數據畫面

- Device 1




- Device 2



## 7. System 實際畫面