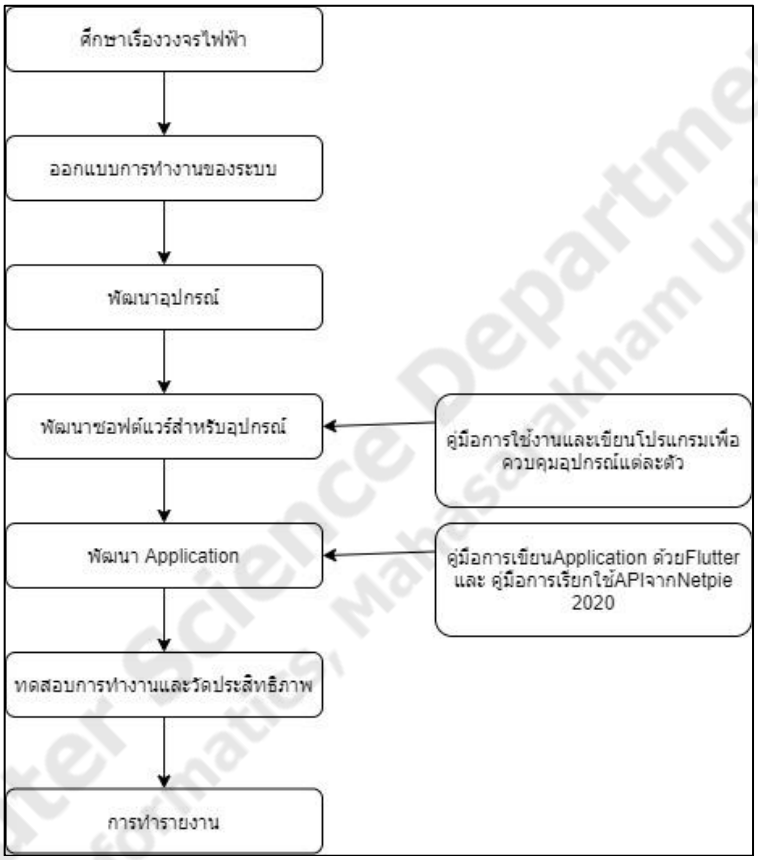
บทที่ 3

ขั้นตอนการดำเนินงาน

3.1 กรอบการดำเนินงาน

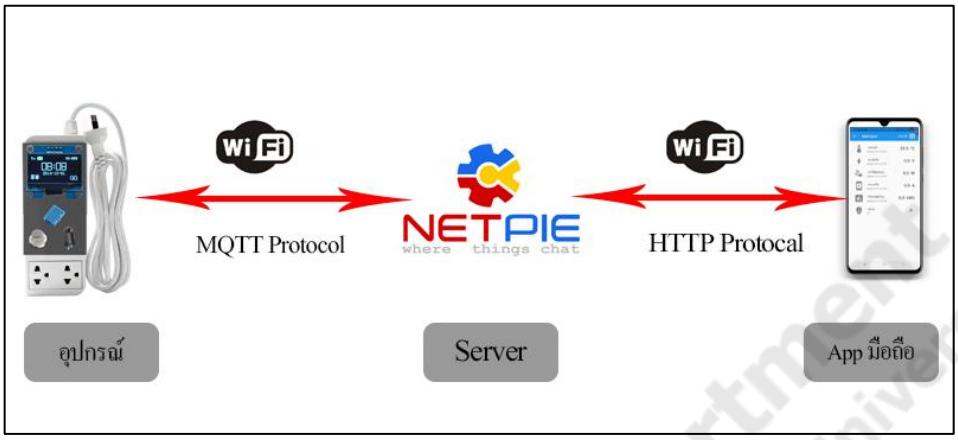


ภาพประกอบที่ 3.1 กรอบการดำเนินงาน

3.1.1 ศึกษาวงจรไฟฟ้า

การจะทำงานเกี่ยวกับไฟฟ้านั้น เราต้องศึกษาและมีความรู้ก่อนจะลงมือทำ เพราะเป็นสิ่งที่อันตรายถ้าหากมีความรู้ไม่เพียงพอนั้น อาจก่อให้เกิดอันตรายและถึงแก่ชีวิตได้ ในงานของเราจะศึกษาเกี่ยวกับวิธีป้องกันอุบัติเหตุจากไฟฟ้าและไฟฟ้ากระแสสลับเป็นหลัก

3.1.2 ออกแบบการทำงานของระบบ



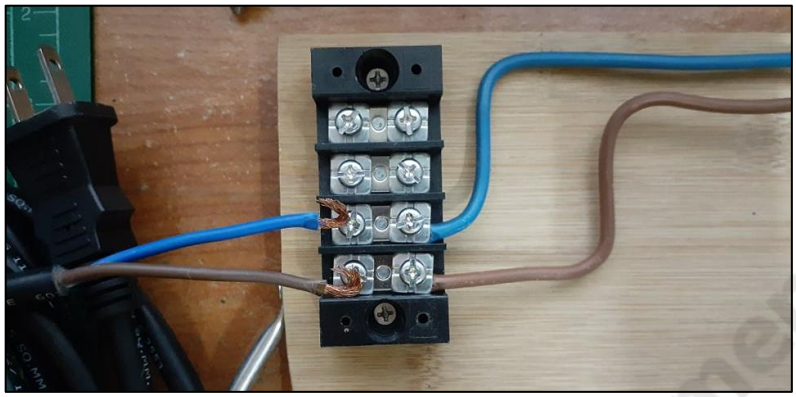
ภาพประกอบที่ 3.2 ออกแบบการทำงานของระบบ

อุปกรณ์จะมี Esp32 เป็นตัวเชื่อมต่อกับ WiFi เพื่อส่งข้อมูลของเซนเซอร์ให้Server ด้วย Protocol MQTT และฝั่ง Application จะรอดึงข้อมูลและส่งข้อมูลไปยังServerโดยใช้ Protocol HTTP เมื่อได้ข้อมูลแล้ว ก็จะนำข้อมูลที่ได้มาแสดงผลบนApplication

3.1.3 พัฒนาอุปกรณ์

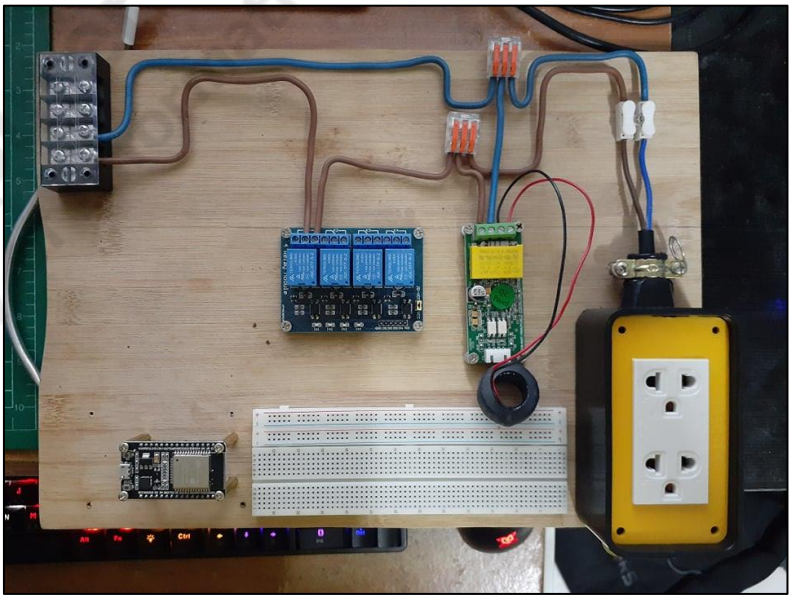
ภาพประกอบที่ 3.3 แผ่นไม้อัด

อุปกรณ์ที่เราพัฒนา เริ่มแรกจะเป็นตัวทดลอง โดยจะใช้แผ่นไม้อัดขนาด 26x36 เซนติเมตร เป็นตัวยึดติดอุปกรณ์เพื่อสะดวกต่อการทดลองงาน และพัฒนางาน



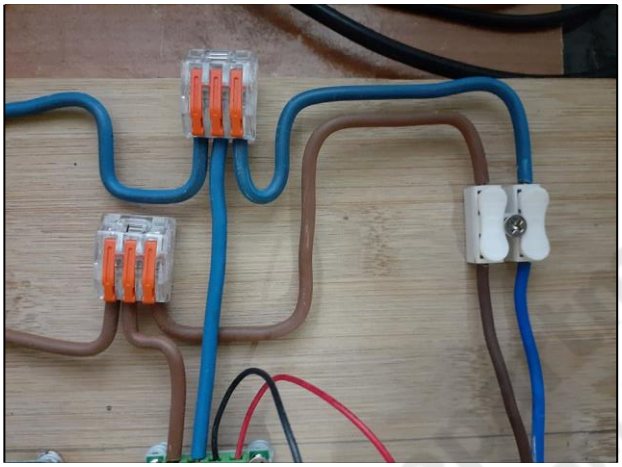
ภาพประกอบที่ 3.4 terminals ขั้วต่อสายไฟ

อุปกรณ์เราต้องต่อไฟเข้า โดยจะต้องเป็นไฟบ้านกระแสสลับ 1เฟส แรงดันประมาณ 220ถึง240VACเท่านั้น ใช้สีของสายไฟตามมาตรฐาน มอก.11-2553 โดยเส้นที่มีไฟ(L) ให้ใช้สายสีน้ำตาล สายนิวทรัล(N) ใช้สายสีฟ้า(น้ำเงิน) การที่เราใช้สีให้ตรงข้อกำหนดนี้ จะทำให้เกิดการต่อสายที่ถูกต้องและไม่สับสนจนเกิดอันตราย จากการพลั้งเผลอหรือเข้าใจผิดตามมาจากคนอื่น ๆ ที่จะมาแก้ไขต่อจากเราและใช้ terminalsขั้วต่อสายไฟ มาต่อสายไฟเพื่อความเป็นระเบียบและง่ายต่อการทำงาน

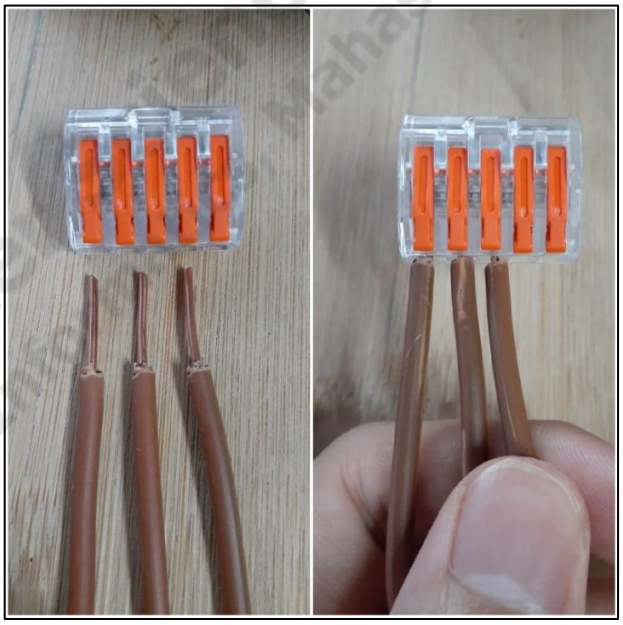


ภาพประกอบที่ 3.5 จัดวางอุปกรณ์

เมื่อต่อไฟเข้าเสร็จแล้ว ให้จัดวางอุปกรณ์วางตามตำแหน่งที่เราออกแบบไว้ และเดินสายไฟเข้าอุปกรณ์ให้เรียบร้อยเพื่อความเป็นระเบียบ

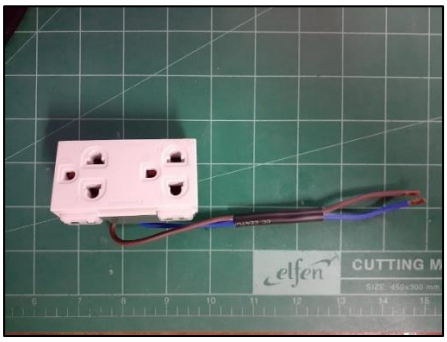


ภาพประกอบที่ 3.6 ตัวต่อสายไฟ

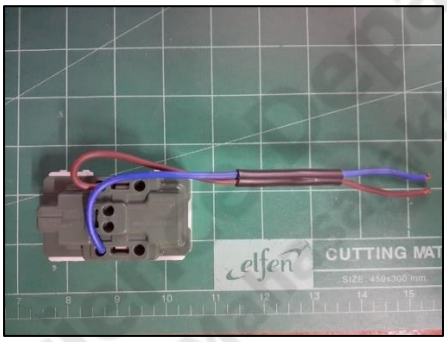


ภาพประกอบที่ 3.7 การต่อสายไฟเข้าด้วยกัน

สำหรับการรวมสายไฟ เราจะใช้ตัวต่อสายไฟมาช่วย เพื่อความเป็นระเบียบและปลอดภัย

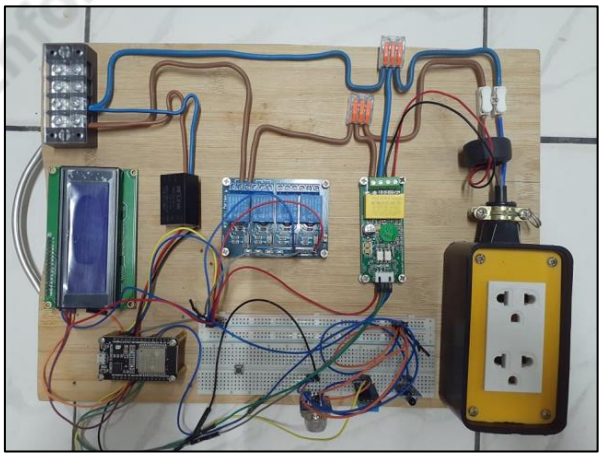


ภาพประกอบที่ 3.8 ด้านหน้าเต้ารับ



ภาพประกอบที่ 3.9 ด้านหลังเต้ารับ

ส่วนเต้ารับก็ต้องเดินสายไฟเข้าให้ถูกวิธี สาย L เสียบช่อง L และสาย N เสียบช่อง N



ภาพประกอบที่ 3.10 ติดตั้งอุปกรณ์และต่อสายไฟเข้าอุปกรณ์

ยึดติดอุปกรณ์ทุกตัวเข้ากับแผ่นไม้ วางตำแหน่งตามที่ถนัดเพื่อความสะดวกต่อทำงาน

3.1.4 พัฒนาอุปกรณ์ให้พร้อมใช้งานจริง

ก่อนหน้านี้เราได้ทำการพัฒนาอุปกรณ์เสร็จเรียบร้อยพร้อมใช้งานแล้ว ต่อไปจะเป็นการนำอุปกรณ์มาใส่กล่องเพื่อใช้งานจริง

ภาพประกอบที่ 3.11 บล็อกยาง 4x8 นิ้ว

อุปกรณ์เราเลือกใช้บล็อกยาง ขนาด 4x8 นิ้ว

ภาพประกอบที่ 3.12 จัดวางอุปกรณ์ลงในบล็อกยาง

จัดวางอุปกรณ์ลงในบล็อกยางตามที่เราออกแบบ

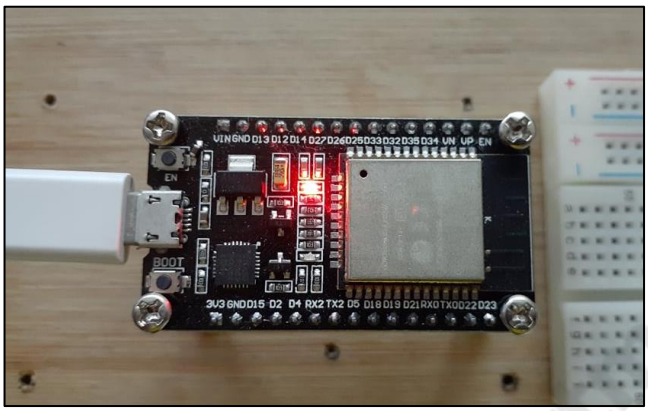
ภาพประกอบที่ 3.13 สายไฟ 1.5 SQ.MM

สายไฟที่ใช้ จะเป็นสายไฟขนาด 1.5 SQ.MM ยาว 5 เมตร รองรับกระแสไฟ 220V/16A 3500 WATTS

ภาพประกอบที่ 3.14 ประกอบอุปกรณ์เข้าด้วยกัน

เมื่อประกอบตัวอุปกรณ์เสร็จ ก็นำสายไฟมาประกอบใส่ให้เรียบร้อย พร้อมใช้งาน

3.1.5 พัฒนาซอฟต์แวร์สำหรับอุปกรณ์



ภาพประกอบที่ 3.15 ESP32 ESP-WROOM-32 Wi-Fi

เริ่มต้นด้วยไมโครคอนโทรลเลอร์ ESP32 หัวใจสำคัญเลยก็คือเชื่อมต่อWiFiเพื่อส่งข้อมูลไฟยังServer โดยเราจะใช้ Library WiFi Manager เข้ามาช่วย

```
#include <WiFi.h>
#include <WiFiManager.h>
#include <Ticker.h>
Ticker ticker;
const int LED = 2;
void tick(){
  digitalWrite(LED,!digitalRead(LED));
}
void configModeCallback(WiFiManager *myWiFiManager){
  Serial.println("Entered config mode");
  Serial.println(WiFi.softAPIP());
  Serial.println(myWiFiManager->getConfigPortalSSID());
  ticker.attach(0.2, tick);
}
void setup() {
  pinMode(LED,OUTPUT);
  Serial.begin(9600);
  ticker.attach(1,tick);
  WiFiManager wm;
  wm.setAPCallback(configModeCallback);
  if( !wm.autoConnect("WiFI_ESP32_Config") ){
    Serial.println("failed to connect and hit timeout");
    ESP.restart();
    delay(1000);
  }
  ticker.detach();
  digitalWrite(LED, LOW);
  delay(100);
  Serial.println("");
  Serial.println("connected...already..WiFi :)");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
void loop() {}
```

ภาพประกอบที่ 3.16 Code WiFi Manager

บรรทัดที่ 1-2 include เรียกใช้ LibraryของWiFi Manager

บรรทัดที่ 3 include เรียกใช้ LibraryของTicker เพื่อทำไฟกระพริบ

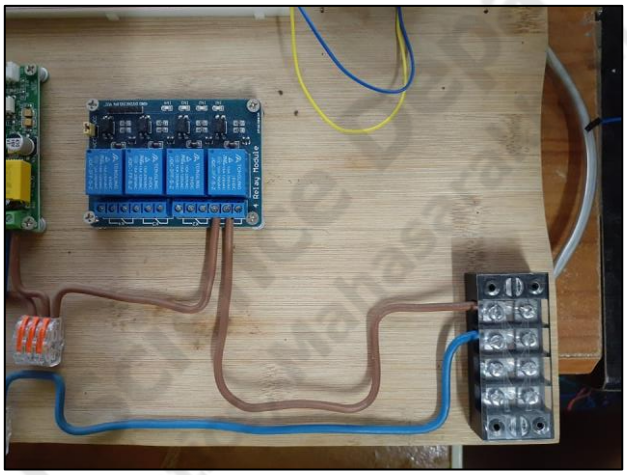
บรรทัดที่ 7 tick() ฟังก์ชันทำไฟกระพริบ

บรรทัดที่ 9 configModeCallback() เช็คว่าWiFiมีการเชื่อมต่อหรือยัง ถ้ายังไม่เชื่อมต่อก็จะเข้าฟังก์ชันนี้ และมีไฟLedกระพริบ

บรรทัดที 20 เรียกใช้ configModeCallback()

บรรทัดที่ 21 ถ้ายังไม่เชื่อมต่อหรือเชื่อมต่อไม่สำเร็จก็จะวนอยู่ที่เดิม จนกว่าจะเชื่อมต่อสำเร็จถึงจะออกเงื่อนไข

บรรทัดที่ 33 แสดง IP address WiFiที่เราเชื่อมต่อสำเร็จ



ภาพประกอบที่ 3.17 การต่อสายไฟเข้า Relay

```
1  #include <Arduino.h>
2  class Relay {
3    private:
4      int pin;
5    public:
6      Relay(int c_pin) {
7        pin = c_pin;
8        pinMode(pin, OUTPUT);
9        digitalWrite(pin, 1); //ปิดใช้งาน Relay
10     }
11     void on() {
12       digitalWrite(pin, 0); //เปิดใช้งาน Relay
13     }
14     void off() {
15       digitalWrite(pin, 1); //ปิดใช้งาน Relay
16     }
17 };
```

ภาพประกอบที่ 3.18 Code ควบคุมRelay

ส่วนของอุปกรณ์

VCC - 5V

GND - GND

IN1 - GPIO13

สายไฟ L ต่อเข้าขา Common(Com) และ Normally Open(NO)

ส่วนของ Code Relay ได้ทำการสร้าง Tabแยกออกมาใช้งานเพื่อไม่ให้ code หน้าหลักดูรกเกินไป

บรรทัดที่ 1 include เรียกใช้Arduino

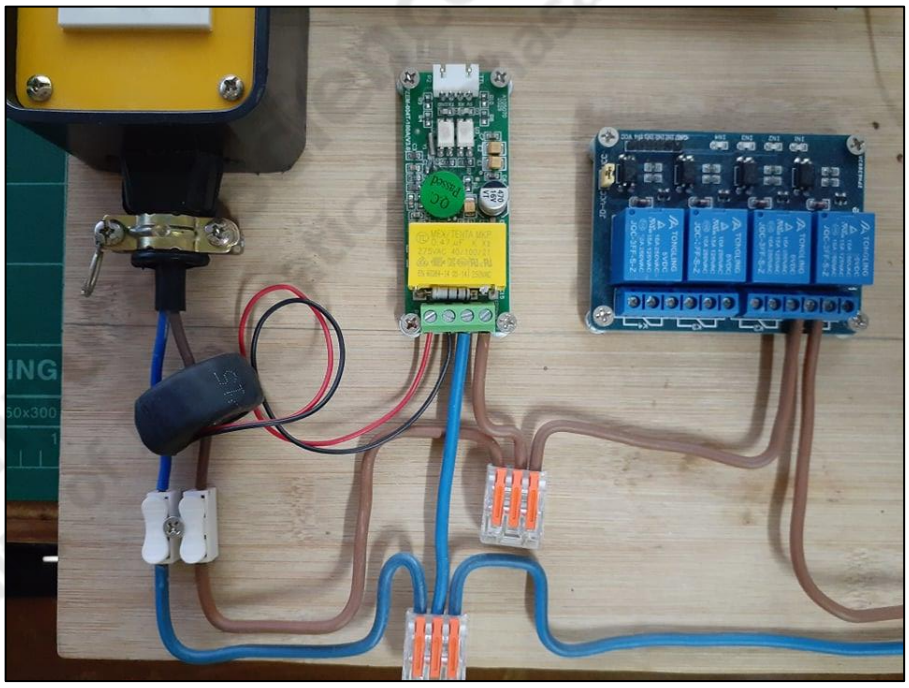
บรรทัดที่ 2 สร้าง class Relay

บรรทัดที่ 6 เรียกใช้ขาที่รับเข้ามาผ่าน constructor

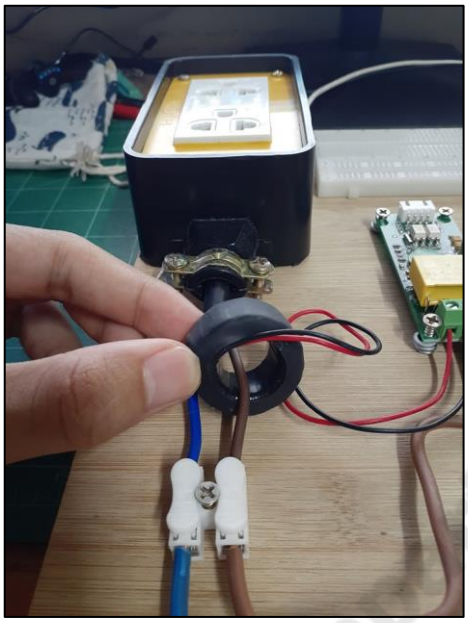
บรรทัดที่ 9 สังปิดRelay เมื่อเริ่มใช้งาน

บรรทัดที่ 11 ฟังก์ชันเปิด Relay

บรรทัดที่ 14 ฟังก์ชันปิด Relay



ภาพประกอบที่ 3.19 การต่อสายไฟเข้า PZEM



ภาพประกอบที่ 3.20 ตัววัดกระแสไฟ

```
#include <Arduino.h>
#include <PZEM004Tv30.h>
HardwareSerial PzemSerial(2);
PZEM004Tv30 pzem(&PzemSerial, 17, 16);
class PZEM {
  private:
    int pin1;
    int pin2;
  public:
    PZEM(int c_pin1, int c_pin2) {
      pin1 = c_pin1;
      pin2 = c_pin2;
      PZEM004Tv30 pzem(&PzemSerial, pin1, pin2);
    }
    float PZEM_voltage() {
      float voltage = pzem.voltage();
      if(isnan(voltage)){
        return 0.0;
      }else{
        return voltage;
      }

    }
    float PZEM_current() {
      float current = pzem.current();
      if(isnan(current)){
        return 0.0;
      }else{
        return current;
      }
    }
```

ภาพประกอบที่ 3.21 Code PZEM 1

```
    float PZEM_power() {
     float power = pzem.power();
      if(isnan(power)){
        return 0.0;
      }else{
        return power;
      }
    }
    float PZEM_energy() {
      float energy = pzem.energy();
      if(isnan(energy)){
        return 0.0;
      }else{
        return energy;
      }
    }
    float PZEM_frequency() {
      float frequency = pzem.frequency();
      if(isnan(frequency)){
        return 0.0;
      }else{
        return frequency;
      }
    }
    float PZEM_pf() {
      float pf = pzem.pf();
      if(isnan(pf)){
        return 0.0;
      }else{
        return pf;
      }
    }
};
```

ภาพประกอบที่ 3.22 Code PZEM 2

ส่วนของอุปกรณ์

5V - 5V

RX – GPIO16

TX - GPIO17

GND – GND

ต่อสายไฟ LและN เข้าที่ตัวอุปกรณ์ และต่อสายไฟของตัววัดกระแสไฟ(ตัวกลมๆดำๆ)เข้ากับตัวอุปกรณ์และตัววัดสายไฟนั้น ต้องเอาสายไฟสอดผ่าน1เส้น เพื่อวัดกระแสไฟฟ้า

ส่วนของ Code PZEM004T ได้ทำการสร้าง Tabแยกออกมาใช้งานเพื่อไม่ให้ code หน้าหลักดูรกเกินไป

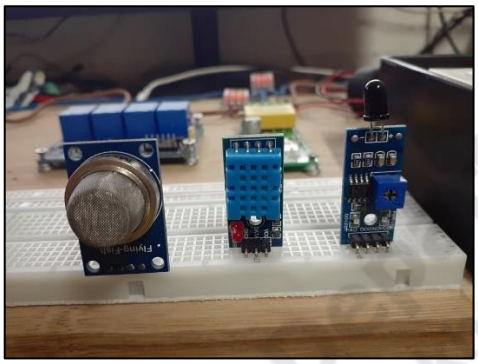
บรรทัดที่ 1 include เรียกใช้Arduino

บรรทัดที่ 2 include เรียกใช้ Library ของ PZEM004Tv30

บรรทัดที่ 3 ประกาศใช้ HardwareSerial2 สำหรับEsp32 ขาของ HardwareSerial2ก็คือ GPIO16(RX) และ GPIO17(TX)

บรรทัดที่ 10 เรียกใช้ขาที่รับเข้ามาผ่าน constructor

บรรทัดที่ 15 ฟังก์ชันอ่านค่าแรงดันไฟฟ้า Returnเป็น Float
บรรทัดที่ 24 ฟังก์ชันอ่านค่ากระแสไฟฟ้าที่ผ่านโหลด Returnเป็น Float
บรรทัดที่ 32 ฟังก์ชันอ่านค่าการใช้พลังงานของโหลด Returnเป็น Float
บรรทัดที่ 40 ฟังก์ชันอ่านค่าพลังงานกิโลวัตต์ต่อชั่วโมง Returnเป็น Float
บรรทัดที่ 48 ฟังก์ชันอ่านความถี่แรงดันไฟฟ้า Returnเป็น Float



ภาพประกอบที่ 3.23 Sensor

```
#include <Arduino.h>
#include "DHT.h"
#define DHTPIN 4
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
class Sensor{
    private:
      int pinMQ; int pinIR;
    public:
    Sensor(char nameSensor, int pin) {
      if (nameSensor == 'M') {
        pinMQ = pin;}
      if (nameSensor == 'D') {
        dht.begin();}
      if (nameSensor == 'I') {
        pinIR = pin;}
    }
    int Value_MQ(){
      return digitalRead(pinMQ);
    }
    float Value_DHT_humidity(){
      float h = dht.readHumidity();
      return h;
    }
    float Value_DHT_temperature(){
     float t = dht.readTemperature();
      return  t;
    }
    float Value_DHT_Fahrenheit(){
      float f = dht.readTemperature(true);
      return  f;
    }
    int Value_IR(){
      return analogRead(pinIR);
    }
};
```

ภาพประกอบที่ 3.24 Code Sensor

ส่วนของSensor MQ-2

VCC – 5V

GND – GND

DO – GPIO5

ส่วนของ Code ได้ทำการสร้าง Tabแยกออกมาใช้งานเพื่อไม่ให้ code หน้าหลักดูรกเกินไป(รวม3 Sensor ใน Class เดียวกัน)

บรรทัดที่ 1 include เรียกใช้Arduino

บรรทัดที่ 11 เรียกใช้ขาที่รับเข้ามาผ่าน constructor

บรรทัดที่ 18 ฟังก์ชันอ่านค่าของSensor Return เป็น int

ส่วนของSensor DHT11

VCC – 5V

GND – GND

DATA – GPIO4

ส่วนของ Code ได้ทำการสร้าง Tabแยกออกมาใช้งานเพื่อไม่ให้ code หน้าหลักดูรกเกินไป(รวม3 Sensor ใน Class เดียวกัน)

บรรทัดที่ 2 include เรียกใช้ Library ของ DHT11

บรรทัดที่ 3-5 ประกาศตัวแปรเก็บขาและTypeของอุปกรณ์ เป็น DHT11

บรรทัดที่ 13 เรียกใช้ขาที่รับเข้ามาผ่าน constructor

บรรทัดที่ 21 ฟังก์ชันอ่านค่าความชื้นของSensor Return เป็น Float

บรรทัดที่ 25 ฟังก์ชันอ่านค่าอุณหภูมิแบบองศาเซลเซียสของSensor Return เป็น Float

บรรทัดที่ 29 ฟังก์ชันอ่านค่าอุณหภูมิแบบองศาฟาเรนไฮต์ของSensor Return เป็น Float

ส่วนของSensor IR

VCC – 5V

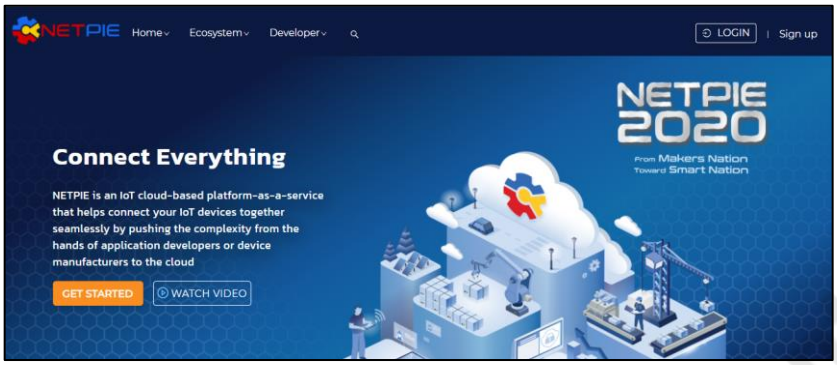
GND – GND

AO – GPIO32

ส่วนของ Code ได้ทำการสร้าง Tabแยกออกมาใช้งานเพื่อไม่ให้ code หน้าหลักดูรกเกินไป(รวม3 Sensor ใน Class เดียวกัน)

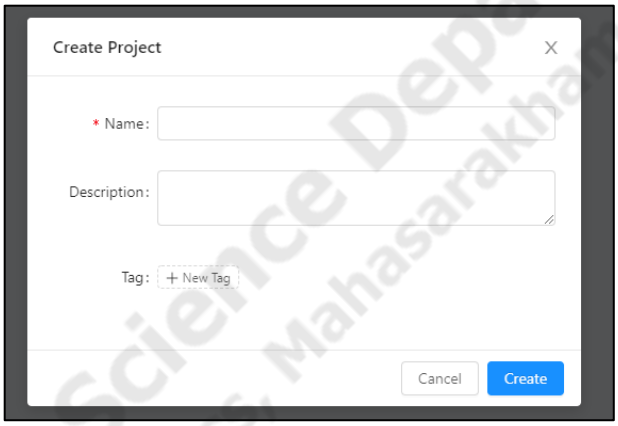
บรรทัดที่ 15 เรียกใช้ขาที่รับเข้ามาผ่าน constructor

บรรทัดที่ 18 ฟังก์ชันอ่านค่าของSensor Return เป็น int



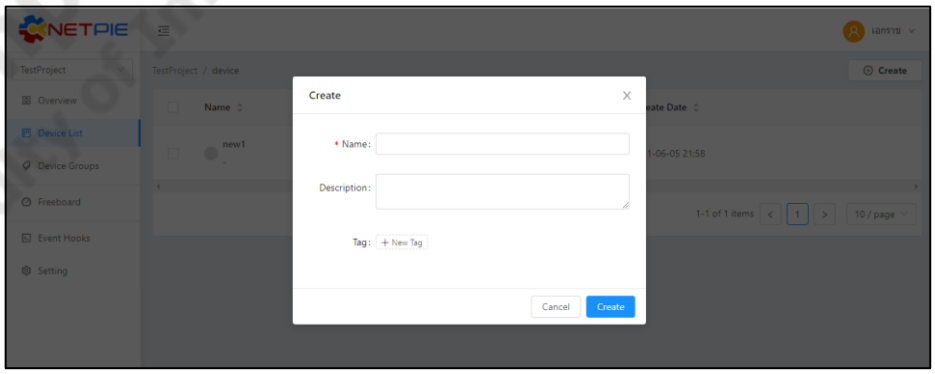
ภาพประกอบที่ 3.25 Web Netpie2020

Login เข้า Netpie2020 เพื่อทำการสร้าง Server ตัวกลางในการสื่อสารของข้อมูล



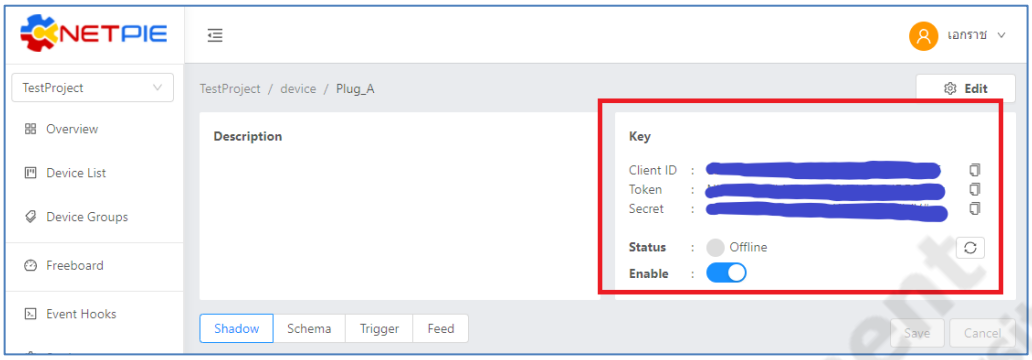
ภาพประกอบที่ 3.26 สร้าง Project

สร้าง Project ของเราในNetpie2020



ภาพประกอบที่ 3.27 สร้างDevice

เมื่อสร้างProjectเสร็จแล้ว ก็มาสร้างDevice ของเราต่อ



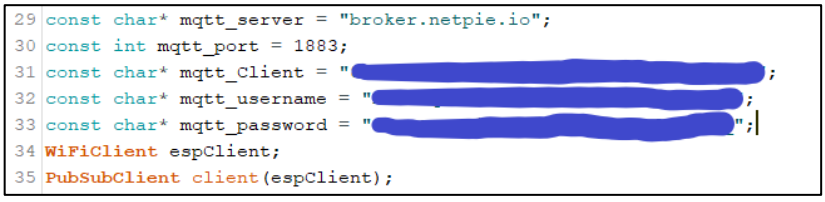
ภาพประกอบที่ 3.28 Keyที่ได้มา

เมื่อสร้าง Device เสร็จ เราจะได้ Key ที่ NETPIE ให้มาเพื่อนำไปใช้งาน

ภาพประกอบที่ 3.29 Code เรียกใช้ไฟล์ Sensor

หน้าCodeหลักจะทำการเชื่อมต่อกับNetPie และเรียกใช้ไฟล์ PZEM , Relay และ Sensor เพื่อส่งข้อมูลขึ้น NetPie

บรรทัดที่ 4 Include PubSubClient ใช้สำหรับให้Esp32เชื่อมต่อและสื่อสารบน Netpie2020(MQTT Protocol)



ภาพประกอบที่ 3.30 Code เชื่อมต่อ Netpie

บรรทัดที่ 21-25 เป็นการประกาศตัวแปรสำหรับเชื่อมต่อ Netpie2020

บรรทัดที่ 23 คือ Client ID ของเราที่ Netpie ให้มา

บรรทัดที่ 24 คือ Token ของเราที่ Netpie ให้มา

บรรทัดที่ 25 คือ Secret ของเราที่ Netpie ให้มา

บรรทัดที่ 26-27 เป็นการกำหนดและเรียกใช้ชุดคำสั่งสำหรับเชื่อมต่อ Netpie2020

```
41 void reconnect() {
42   while (!client.connected()) {
43     Serial.print("Attempting NETPIE2020 connection...");
44     if (client.connect(mqtt_Client, mqtt_username, mqtt_password)) {
45       Serial.println("NETPIE2020 connected");
46       client.subscribe("@msg/RelayStatus");
47     }
48     else {
49       Serial.print("failed, rc=");
50       Serial.print(client.state());
51       Serial.println("try again in 5 seconds");
52       delay(5000);
53     }
54   }
55 }
```

ภาพประกอบที่ 3.31 ฟังก์ชัน reconnect()

บรรทัดที่ 41 ฟังก์ชัน reconnect() เป็นฟังก์ชันการเชื่อมต่อ Netpie2020 หากเชื่อมต่อสำเร็จจะแสดงผลว่า “connected” และหากเชื่อมต่อไม่สำเร็จจะแสดงผลว่า “failed ...” และจะทำการเชื่อมต่อใหม่อัตโนมัติ

```
101 void loop() {
102   //int input = Serial.read();
103   if (!client.connected()) {
104     reconnect();
105   }
106   client.loop();
```

ภาพประกอบที่ 3.32 void loop

ใน void loop จะมีการเช็คการเชื่อมต่ออยู่ตลอดเวลา และถ้าหลุดการเชื่อมต่อก็จะกลับไปเรียกใช้ฟังก์ชัน reconnect เพื่อเชื่อมต่อใหม่อีกครั้ง

บรรทัดที่ 106 ทำให้ esp32 คงสถานการณ์เชื่อมต่อไว้

```
138     String data = "{\"data\": {\"humidity\":" + String(myDHT.Value_DHT_humidity()) +
139                   ", \"temperature\":" + String(myDHT.Value_DHT_temperature()) +
140                   ", \"voltage\":" + String(myPzem.PZEM_voltage()) +
141                   ", \"current\":" + String(myPzem.PZEM_current()) +
142                   ", \"power\":" + String(myPzem.PZEM_power()) +
143                   ", \"energy\":" + String(myPzem.PZEM_energy()) +
144                   ", \"frequency\":" + String(myPzem.PZEM_frequency()) +
145                   ", \"pf\":" + String(myPzem.PZEM_pf()) +
146                   ", \"status_relay\":\"" + String(status_Relay) + "\"}}";
147
148     Serial.println(data);
149     data.toCharArray(msg, (data.length() + 1));
150     client.publish("@shadow/data/update", msg);
```

ภาพประกอบที่ 3.33 Code ส่งข้อมูลผ่าน MQTT

เมื่อเชื่อมต่อNetpie2020 สำเร็จแล้วเราก็จะทำการส่งข้อมูลผ่าน MQTT ไปยัง Device Shadowเพื่อบันทึกเป็นข้อความล่าสุด มีรูปแบบ topic คือ @shadow/data/update ซึ่งรูปแบบข้อมูลต้องเป็นรูปแบบของ JSON เช่น {data:{temp:24}}

บรรทัดที่ 138-146 ทำการดึงค่าข้อมูลมา แล้วมาทำเป็นStringในรูปแบบJSON

บรรทัดที่ 149 นำString ที่ทำขึ้นมามาแปลงเป็นCharArrayก่อน

บรรทัดที่ 150 เป็นการนำค่า msg ที่ได้มา ส่งไปยังDevice Shadow บน Netpie2020

ภาพประกอบที่ 3.34 Device Shadow

Device Shadow กำลังรอรับข้อมูลจากอุปกรณ์ที่เราส่งมา แต่การจะจัดเก็บข้อมูลลง Device Shadow ได้นั้นจำเป็นต้องสร้าง Device Schema ก่อน

ภาพประกอบที่ 3.35 Device Schema

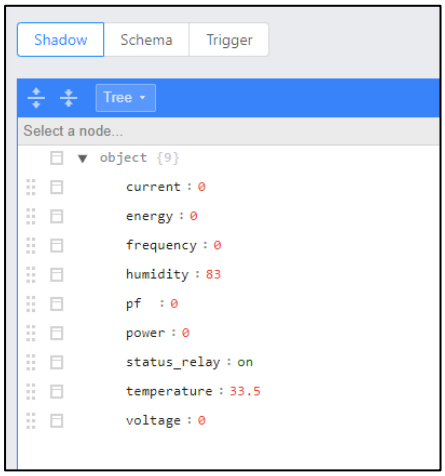
ส่วนของ Device Schema คือ โครงสร้างข้อมูลที่กำหนดไว้เพื่อใช้กำกับ Device Shadow สำหรับ Device ที่ต้องมีการจัดเก็บข้อมูล และต้องเขียนในรูปแบบของJSON

บรรทัดที่ 2 additionalProperties เป็นสถานะการอนุญาตให้บันทึกข้อมูลลง Shadow หรือ Timeseries Database ในกรณีที่ข้อมูลไม่ตรงตามที่กำหนด Properties มี 2 สถานะ คือ true คือ อนุญาตให้บันทึกลง Shadow หรือ Timeseries Database false คือ ไม่อนุญาตให้บันทึกเฉพาะส่วนที่ไม่ตรงตาม Properties อย่างในตัวอย่าง properties 2 ค่าคือ humidity และ temperature ถ้าข้อมูลที่ส่งมาคือ temp, humid, light addtionalProperties = true จะจัดเก็บทั้ง temperature, humidity และ light addtionalProperties = false จะจัดเก็บเพียง temperature, humidity

บรรทัดที่ 3 Properties เริ่มจากกำหนดชื่อฟิลด์(ตัวอย่างคือ “humidity” และ “Temperature”)

บรรทัดที่ 5 Operation สำหรับตั้งค่าการจัดการข้อมูลในฟิลด์นั้นๆ ประกอบไปด้วย store สำหรับตั้งค่าการเก็บข้อมูลลง Timeseries Database ttl คือ ระยะเวลาของการเก็บข้อมูลใน Timeseries Database ซึ่งแต่ละ ข้อมูลมีอายุการเก็บครบตามกำหนดจะถูกลบทิ้งอัตโนมัติ ถ้าต้องการจัดเก็บข้อมูลระบบจำเป็นต้องกำหนดค่านี้ มีหน่วยเป็น ms(มิลลิวินาที), s(วินาที), m(นาที), h(ชั่วโมง), d(วัน), y(ปี) Transform การแปลงข้อมูลก่อนจัดเก็บ expression คือ สูตรหรือวิธีการแปลงข้อมูลก่อนจัดเก็บ ตัวอย่าง แปลงหน่วยอุณหภูมิจากเซลเซียสเป็นฟาเรนไฮต์= ($.temperature*1.8) + 3

บรรทัดที่ 10 Type คือ ชนิดข้อมูลในฟิลด์นั้นๆ ได้แก่ number, string, array, object



ภาพประกอบที่ 3.36 ผลลัพธ์ใน Device Shadow

เมื่อสร้างSchema เสร็จ ข้อมูลที่เราส่งมาจากอุปกรณ์ก็จะแสดงผลลัพธ์ออกทาง Shadow ถือว่าเป็นการส่งข้อมูลสำเร็จ ถ้าอุปกรณ์เราขาดการเชื่อมต่ออินเทอร์เน็ต Shadowจะเก็บบันทึกข้อมูลล่าสุดไว้ให้เราอัตโนมัติ

3.1.6 พัฒนา Application

ภาพประกอบที่ 3.37 Application หน้าแรก

หน้าหลักของ Application ใช้แสดงข้อมูลต่างๆที่เราต้องการนำมาแสดง โดยการดึงข้อมูลผ่าน RESTful API ซึ่งใช้ HTTP Protocol ในการสื่อสาร

```
64  Future<Null> data() async {
65    String deviceAuth = "Device " + clientId + ":" + token;
66    var response = await http.get(
67        Uri.parse("https://api.netpie.io/v2/device/shadow/data"),
68        headers: <String, String>{
69          'Authorization': deviceAuth,
70        });
71    print("update");
72    setState(() {
73      _dataFromAPI = mydataFromJson(response.body);
74      on_off = _dataFromAPI.data.statusRelay;
75      if (this.on_off == "on") {
76        _enableFingerPrint = true;
77        setState(() {});
78      } else {
79        _enableFingerPrint = false;
80        setState(() {});
81      }
82
83    });
84  }
```

ภาพประกอบที่ 3.38 Code RESTful API GET

ส่วนของ Code ในการดึงข้อมูลมาแสดง จะร้องขอข้อมูลโดยใช้ RESTful API แบบ GET บรรทัดที่ 73 เมื่อได้response จากการร้องขอมาแล้ว เราจะส่งresponse.body ไปแปลงข้อมูลใน Modelที่เราสร้าง เพื่อง่ายต่อการเอาข้อมูลไปใช้งาน

```
253
254                     String deviceAuth =
255                         "Device " + clientId + ":" + token;
256                     var response = await http.put(
257                         Uri.parse(
258                             "https://api.netpie.io/v2/device/message?topic=" +
259                                 topic),
260                         headers: <String, String>{
261                           'Authorization': deviceAuth,
262                         },
263                         body: "on");
```

ภาพประกอบที่ 3.39 Code สั่งเปิดเต้ารับ

```
268                     String deviceAuth =
269                         "Device " + clientId + ":" + token;
270                     var response = await http.put(
271                         Uri.parse(
272                             "https://api.netpie.io/v2/device/message?topic=" +
273                                 topic),
274                         headers: <String, String>{
275                           'Authorization': deviceAuth,
276                         },
277                         body: "off");
278                 }
```

ภาพประกอบที่ 3.40 Code สั่งปิดเต้ารับ

ส่วนของ code ปุ่มการเปิดปิดเต้ารับ จะส่งโดยใช้ RESTful API แบบ PUT จะเป็นการ Publishข้อความตามที่เราต้องการ โดยจะใส่ข้อความลงใน body และกำหนดTopic และส่งไปยัง Server Netpie2020 เมื่อส่งไปแล้ว ฝั่งอุปกรณ์ของเราก็ต้องทำการ Subscribe Topicที่เราส่งไป ในที่นี้Topicเราคือ “RelayStatus”

```
41 void reconnect() {
42   while (!client.connected()) {
43     Serial.print("Attempting NETPIE2020 connection...");
44     if (client.connect(mqtt_Client, mqtt_username, mqtt_password)) {
45       Serial.println("NETPIE2020 connected");
46       client.subscribe("@msg/RelayStatus");
47     }
48     else {
49       Serial.print("failed, rc=");
50       Serial.print(client.state());
51       Serial.println("try again in 5 seconds");
52       delay(5000);
53     }
54   }
55 }
56
```

ภาพประกอบที่ 3.41 Code Subscribe Topic

ส่วนของ Code อุปกรณ์ บรรทัดที่ 46 เราจะทำการ Subscribe ไปที่ Topic “RelayStatus” (ติดตามข้อมูลของ “RelayStatus”)

```
57 void callback(char* topic, byte* payload, unsigned int length) {
58   Serial.print("Message arrived [");
59   Serial.print(topic);
60   Serial.print("]: ");
61   String msg;
62   for (int i = 0; i < length; i++) {
63     msg = msg + (char)payload[i];
64   }
65   Serial.println(msg);
66   if (String(topic) == "@msg/RelayStatus") {
67     if (msg == "on") {
68       myRelay.on();
69       status_Relay = "on";
70     }
71     else if (msg == "off") {
72       myRelay.off();
73       status_Relay = "off";
74     }
75   }
76 }
```

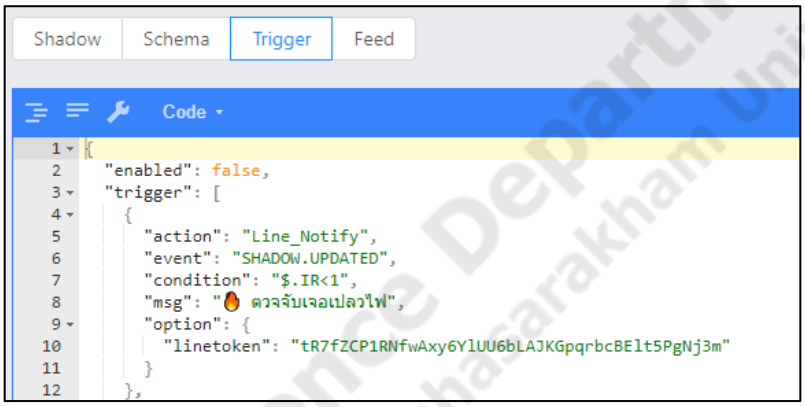
ภาพประกอบที่ 3.42 ฟังก์ชั่น callback

ในส่วนของฟังก์ชั่น callback เป็นฟังก์ชั่นสำหรับการรับ payload หรือข้อความต่างๆที่ถูกส่งมาตาม Topic ที่ อุปกรณ์เราได้ทำการ Subscribe และนำมาแสดงผล โดยข้อความที่ได้รับจะเป็นชนิด byte ไม่สามารถนำความมาใช้ได้โดยตรง จึงสร้างตัวแปรชื่อว่า msg มารอไว้ และใช้ฟังก์ชั่น loop ในการรวมคำจาก byte เป็น String ซึ่ง Topic ที่จะได้รับคือ @msg/RelayStatus และข้อความที่ถูกส่งมาคือ “on” และ “off” เมื่อ อุปกรณ์ ที่ได้รับ Topic มา ถ้าหาก Topic ตรงกันกับ @msg/RelayStatus

ให้ทำตามเงื่อนไขการตรวจสอบข้อความที่ได้รับคือคำว่า “on” ให้ Relayเปิด และหากว่าข้อความที่ได้รับเป็น “off” ให้ Relayปิด

3.1.7 การทำแจ้งเตือน

ใน NETPIE2020 จะมีฟังก์ชันสำหรับทำแจ้งเตือนก็คือ Trigger and Event Hook เป็นระบบที่ผูกกับการเปลี่ยนแปลงข้อมูลของ Device Shadow เข้ากับการกระทำภายนอก (Event Hook) เช่น การตั้งค่าแจ้งเตือนตามสถานะต่างๆ ตามเงื่อนไขการทำงานของ Device ที่ถูกตั้งค่าไว้โดยการใช้งาน Trigger จะ ประกาศในรูปแบบ JSON มีลักษณะดังนี้



ภาพประกอบที่ 3.43 Trigger

บรรทัดที่ 2 enable คือ สถานะเปิด/ปิดหารใช้งาน Trigger

บรรทัดที่ 3 trigger เป็นการตั้งค่าต่างๆของ trigger

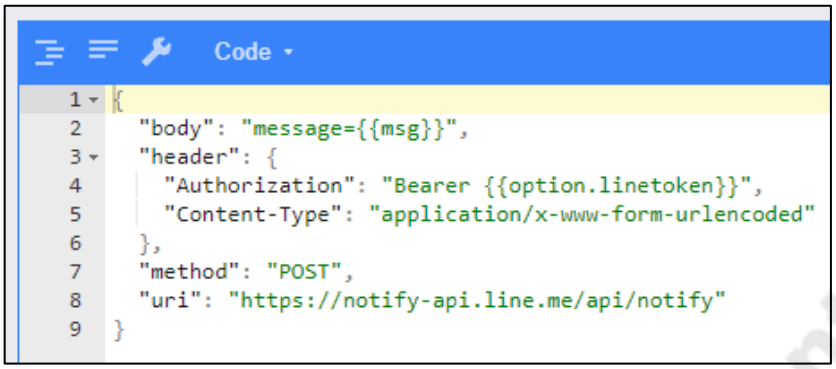
บรรทัดที่ 5 action เมื่อเกิด Trigger จะให้กระทำอะไร โดยระบุชื่อ Event Hook ที่ต้องการทำ

บรรทัดที่ 6 event ประเภทการเปลี่ยนแปลงข้อมูลของ Device Shadow มี 2 ประเภทคือ SHADOW.UPDATED เกิดเมื่อ Device Shadow Data มีการเปลี่ยนแปลงตรงตาม condition ที่กำหนดไว้ (กรณีนี้จำเป็นต้องกำหนด condition ถ้าไม่กำหนดจะไม่เกิด Trigger) DEVICE.STATUSCHANGED เกิดขึ้นเมื่อ Device เปลี่ยนสถานการณ์เชื่อมต่อ Platform จาก Online เป็น Offline หรือ Offline เป็น Online

บรรทัดที่ 7 Condition คือ เงื่อนไขการเปลี่ยนแปลงของ Device Shadow Data จะใช้ในกรณีที่ SHADOW.UPDATED

บรรทัดที่ 8 msg คือ ข้อความที่ต้องการให้แจ้งเตือนกรณีเกิด Trigger

บรรทัดที่ 9 option ใช้สำหรับกำหนดค่าอื่นๆ นอกเหนือจากที่มีระบุไว้ในข้างต้น ในกรณีนี้เราแจ้งเตือนผ่าน LINE Application จึงต้องใส่ LINE TOKEN ของเรา



ภาพประกอบที่ 3.44 Hook

บรรทัดที่ 2 body ส่วนของข้อมูลในที่นี้คือ ข้อความที่จะส่งไปยังปลายทาง

บรรทัดที่ 3 header คือ ข้อมูลเพิ่มเติมที่ต้องการส่งไปยังปลายทาง เช่น Authorization, Content-Type เป็นต้น

บรรทัดที่ 4 method คือ ส่วนที่ก าหนดว่าปลายทางต้องการให้ส่งไปในแบบไหน GET, POST หรือ PUT

บรรทัดที่ 5 uri คือ Endpoint ปลายทางที่กำหนดว่าต้องการให้ส่งไปที่ใด