

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN

____o0o____



VẬT LÝ CHO CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN CUỐI KÌ

GIẢNG VIÊN HƯỚNG DẪN:

thầy CAO XUÂN NAM

THÀNH PHỐ HỒ CHÍ MINH, 8/2022

MUC LUC

1. THÔNG TIN NHÓM.....	4
2. SẢN PHẨM.....	4
1. Tên sản phẩm và ý tưởng.....	4
2. Chức năng sản phẩm.....	4
3. Sơ đồ truyền và nhận dữ liệu.....	5
4. Bản vẽ thiết kế 3D.....	6
Mặt nhìn từ trên xuống:.....	6
Mặt nhìn từ đằng trước:	9
Mặt nhìn từ trái sang:	10
Mặt nhìn từ phải sang:.....	10
Mặt nhìn từ sau:.....	10
Mặt nhìn từ dưới:	11
5. Giao diện web.	11
6. Flow node-red.....	17
a) Login và signup.....	17
b) Các giá trị cảm biến.....	18
3. CÀI ĐẶT ESP32.....	19
Thiết kế mạch.	19
Các thư viện cần thiết.	20
Các biến toàn cục/macros.	20
1) Định nghĩa các chân cắm của từng linh kiện bằng macros.	20
2) Định nghĩa ID channel và API Keys của đọc với ghi cho Server ThinkSpeak.....	20
3) Các chuỗi chứa địa chỉ, requests và cổng ports của các Hosts và Servers và cần thiết để thao tác với dữ liệu.	21
Các hàm.	22
1) <i>setup()</i>	22
2) <i>uploadData()</i>	23
3) <i>updateData()</i>	24
4) <i>printData()</i>	25
5) <i>callback()</i>	25
6) <i>triggerBuzzer()</i>	26

7) <i>eveCheck()</i>	26
8) <i>loop()</i>	27
4. BẢNG PHÂN CÔNG VIỆC.....	30
5. THAM KHẢO	31

1. THÔNG TIN NHÓM.

Nhóm: 8

Lớp: 20CLC03

MSSV	Họ và Tên
20127326	Đỗ Quốc Thắng
20127565	Vũ Đức Xuân Minh
20127635	Lại Minh Thông

2. SẢN PHẨM.

1. Tên sản phẩm và ý tưởng.

Tên sản phẩm: Vòng cổ chăm sóc thú cưng thông minh - **Smet Collar**

Video Demo sản phẩm: <https://youtu.be/cRuoLIfzQFI>

2. Chức năng sản phẩm.

Định vị thú cưng giúp chủ của thú cưng giám sát được những nơi thú cưng mình hay đi tới.

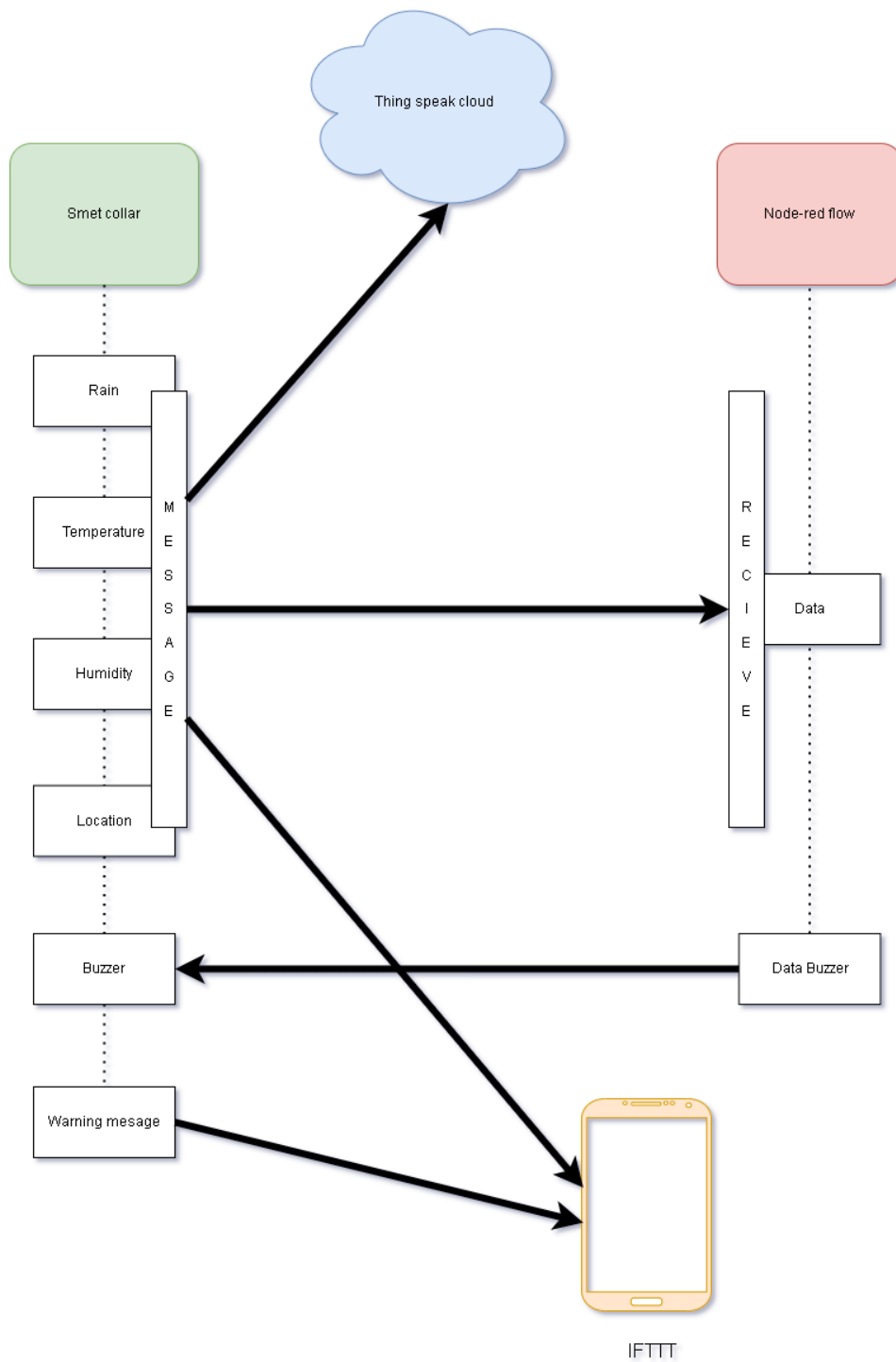
Tiếng còi sẽ được bật thông qua website khi thất lạc (tiếng còi khi bấm lần đầu sẽ phát ra tiếng kêu giúp định vị được thú cưng và bấm lại 1 lần nữa để tắt đi tiếng còi) .

Cảm biến nhiệt độ/độ ẩm trên vòng cổ sẽ thông báo về điện thoại cho người nuôi biết thời tiết nắng đẹp thích hợp để thả thú cưng chơi ngoài trời, dắt đi dạo hoặc mưa để đem thú cưng vào trong nhà.

Các dữ liệu cảm biến nhiệt độ, độ ẩm, mưa và vị trí của thú cưng(kinh độ, vĩ độ) sẽ được gửi về điện thoại.

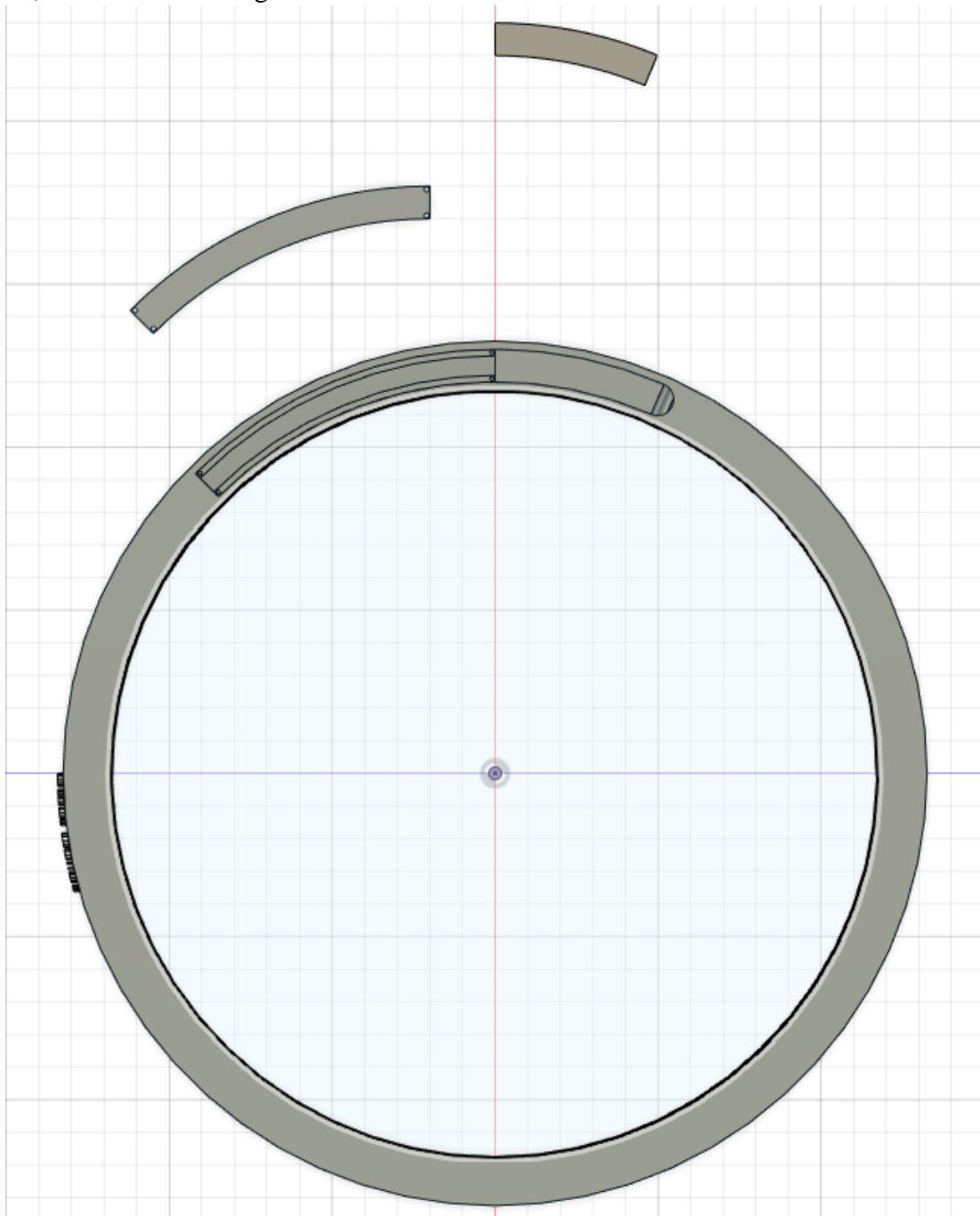
Các dữ liệu cảm biến nhiệt độ, độ ẩm, mưa và vị trí của thú cưng(kinh độ, vĩ độ) sẽ được lưu trữ lại ở trên cloud.

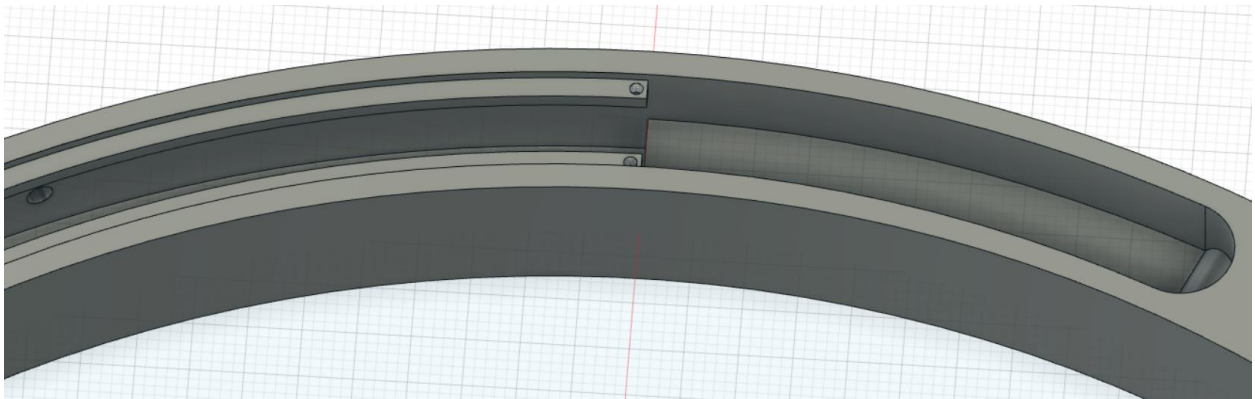
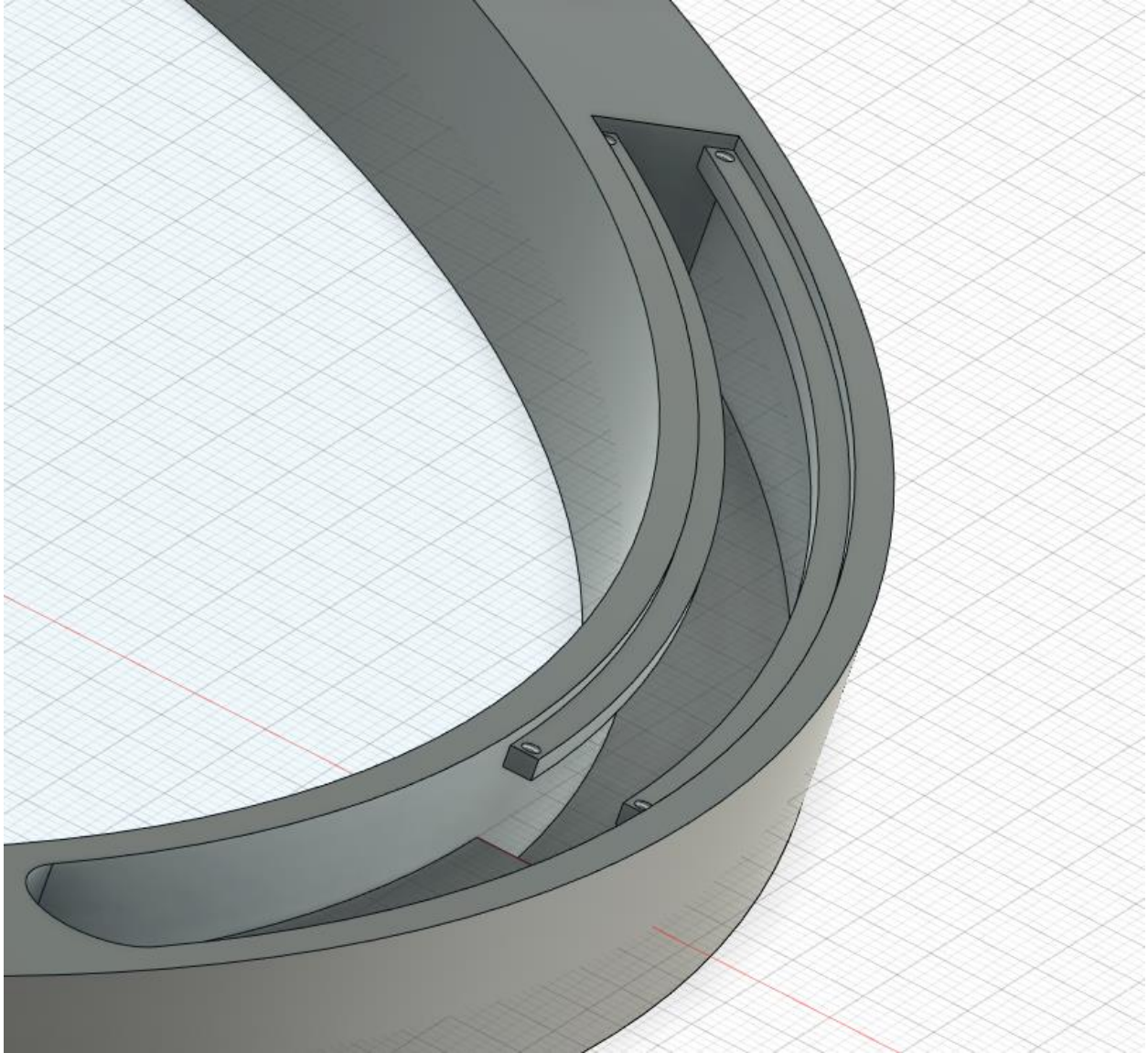
3. Sơ đồ truyền và nhận dữ liệu.

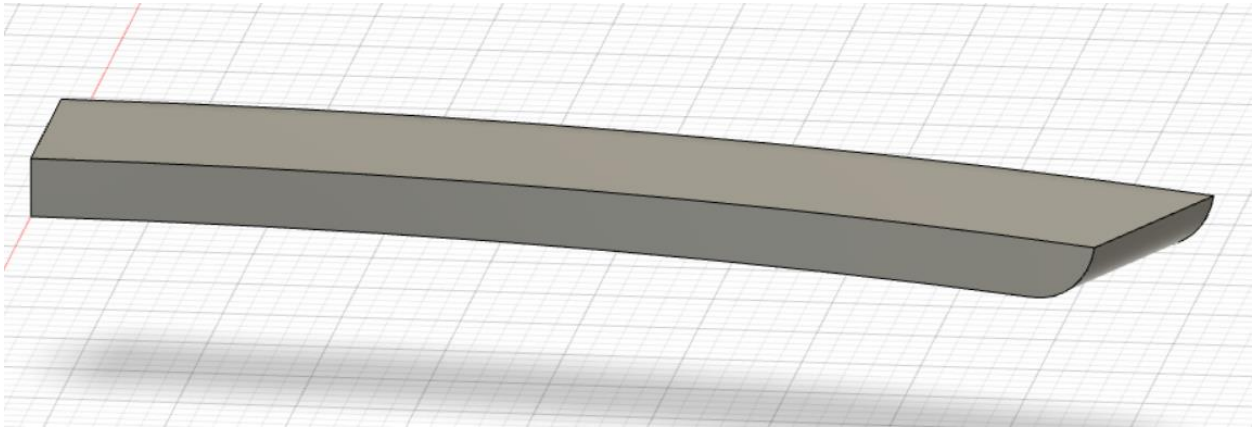
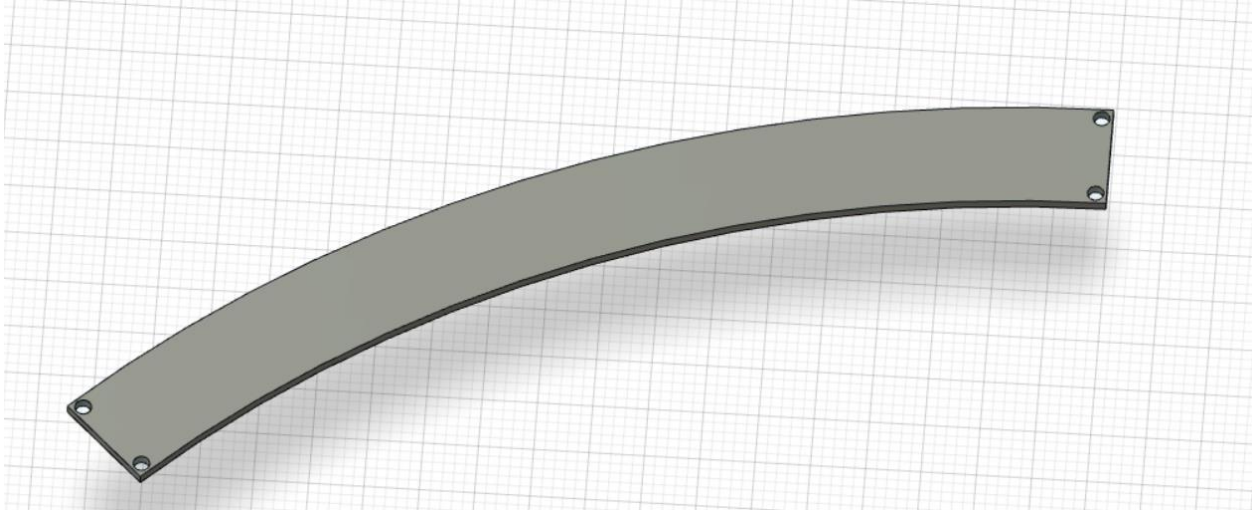


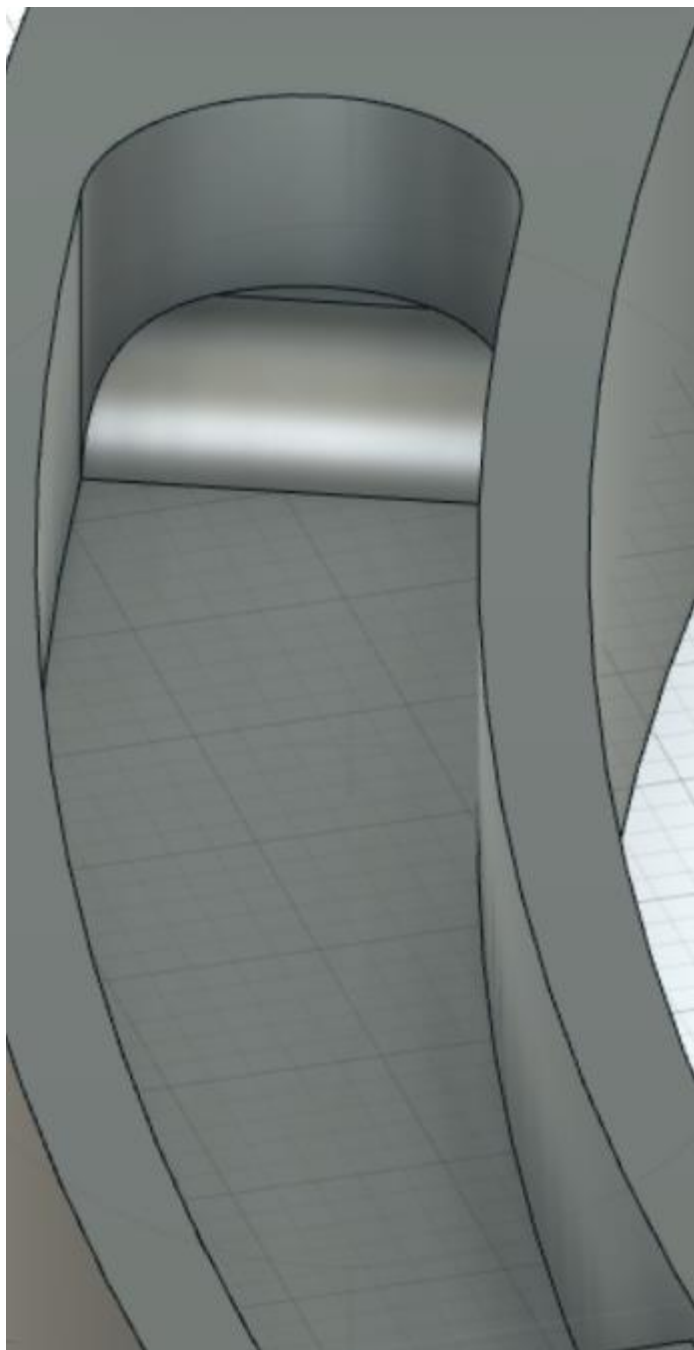
4. Bản vẽ thiết kế 3D.

Mặt nhìn từ trên xuống:

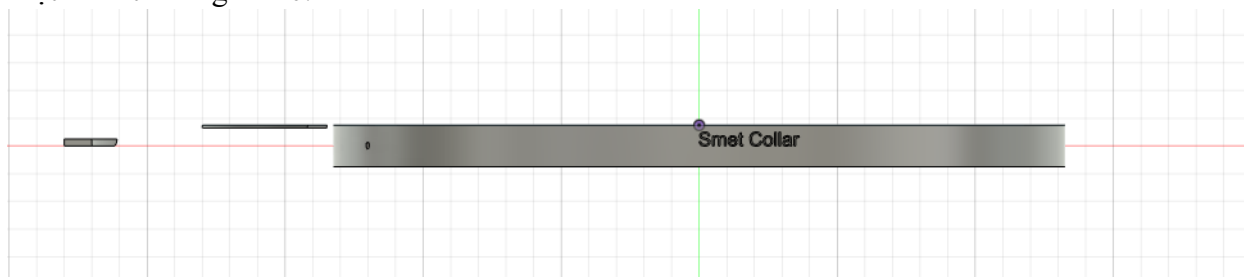




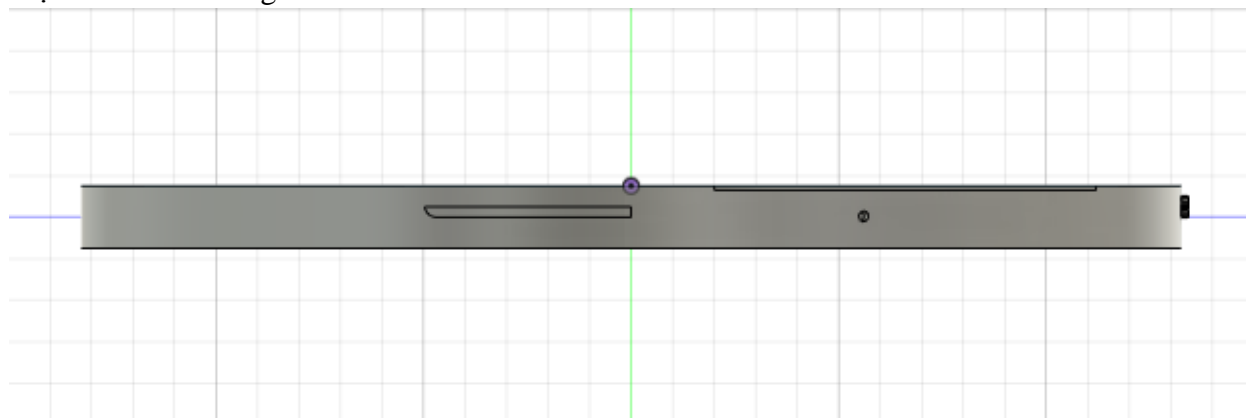




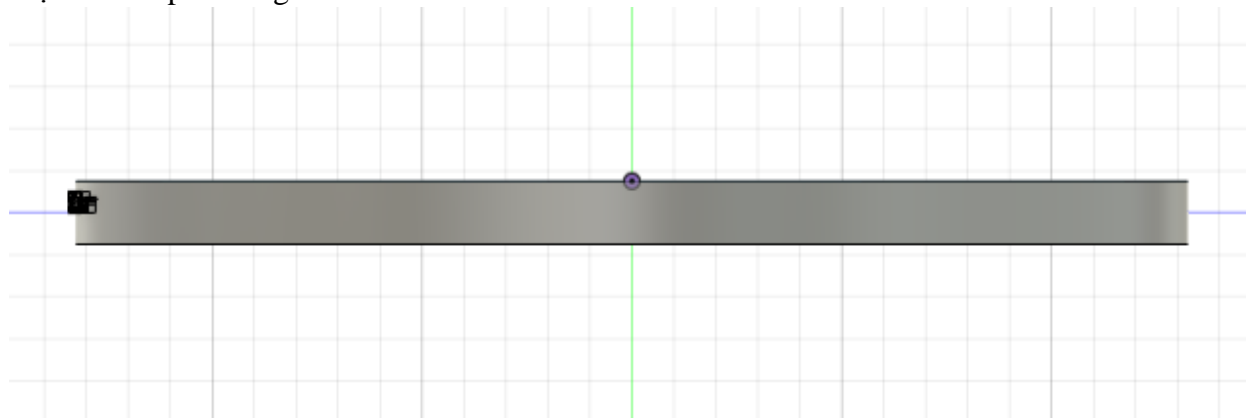
Mặt nhìn từ đằng trước:



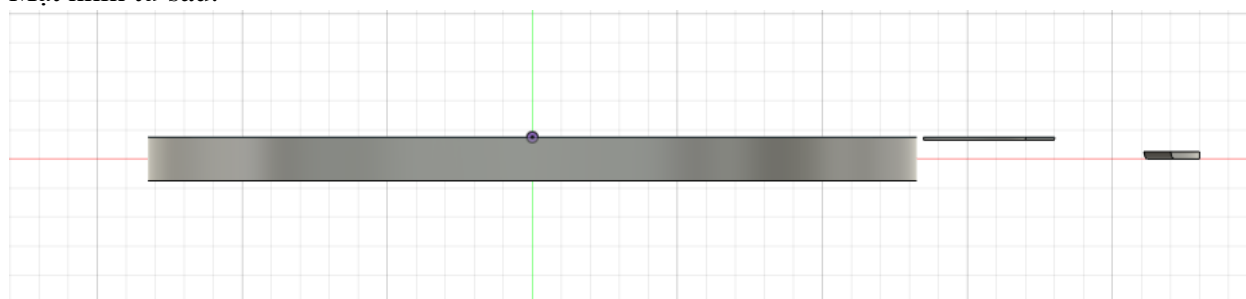
Mặt nhìn từ trái sang:



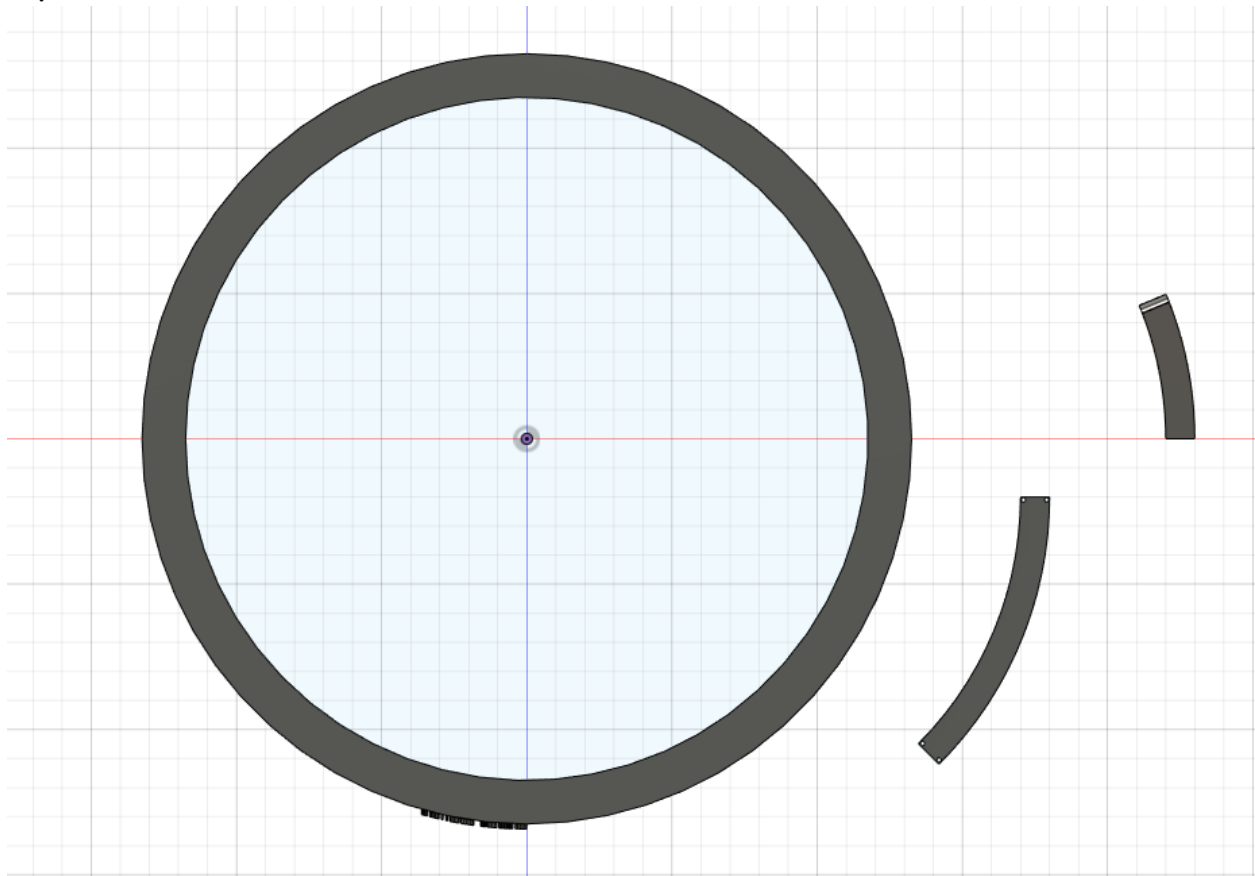
Mặt nhìn từ phải sang:



Mặt nhìn từ sau:

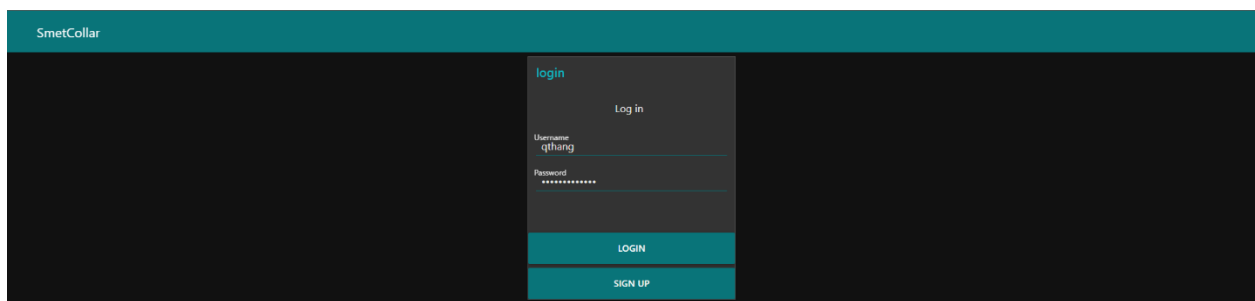


Mặt nhìn từ dưới:

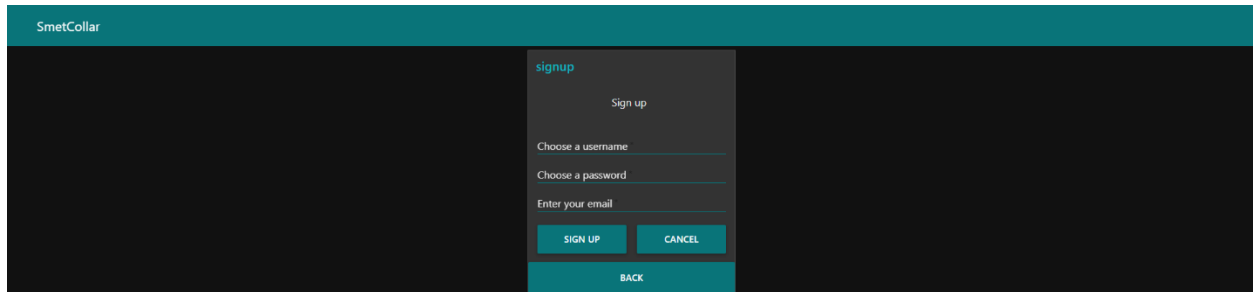


5. Giao diện web.

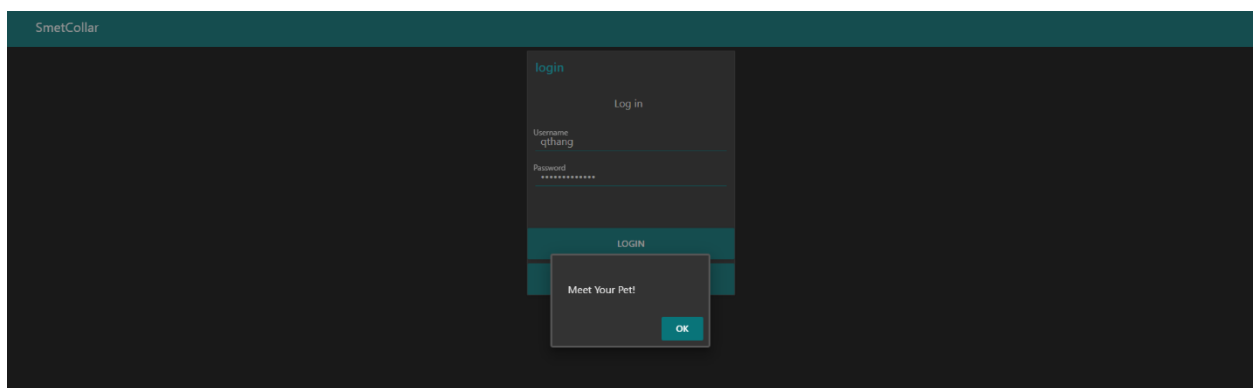
Ban đầu, web sẽ hiện thị màn hình đăng nhập



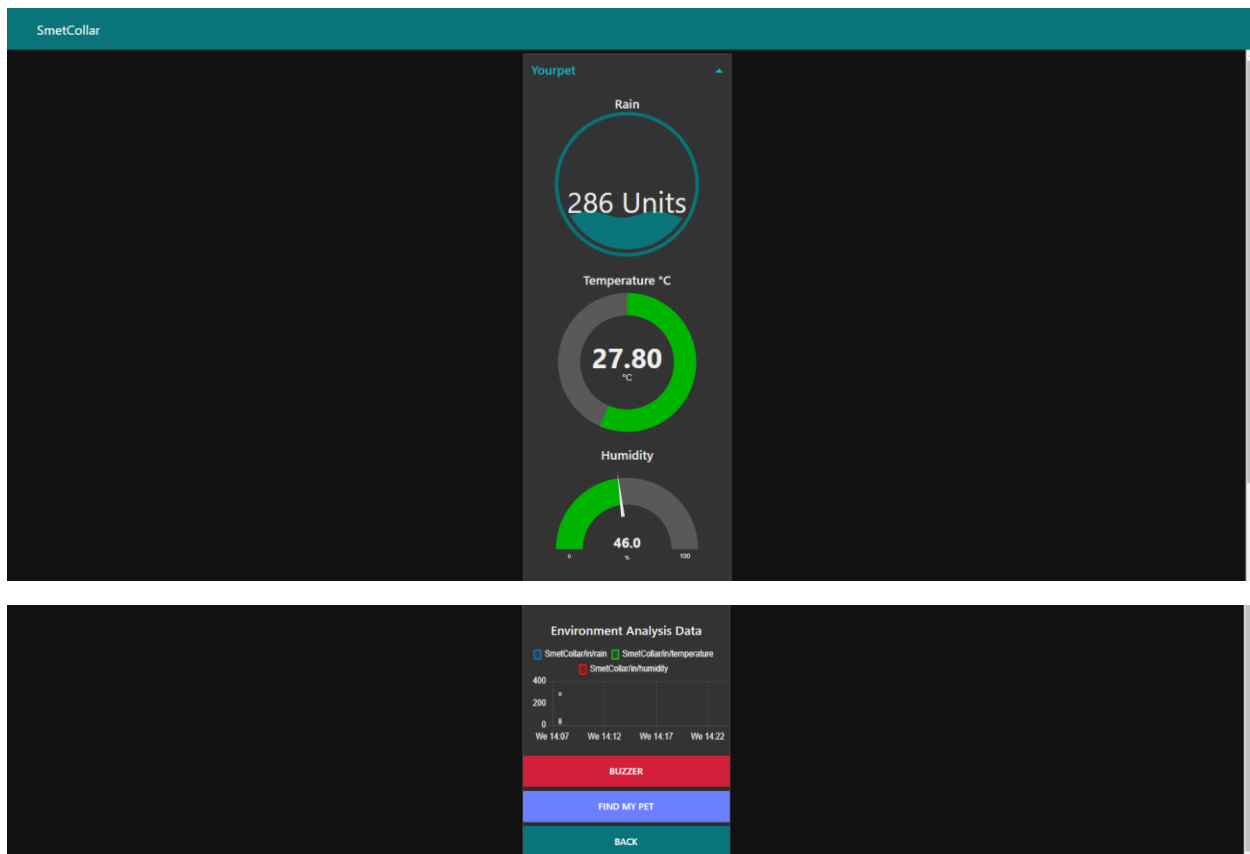
Nếu người dùng chưa có tài khoản, người dùng có thể bấm vào nút sign up để tạo tài khoản đăng nhập. (Do chức năng chưa được tối ưu nên tài khoản và mật khẩu của người dùng chỉ có thể lưu khi còn bật node-red. Nếu tắt node-red thì toàn bộ dữ liệu sẽ bị xóa)

The image shows a web interface for SmetCollar. At the top, there is a teal header with the text "SmetCollar". Below the header, there is a dark grey modal window titled "signup" in teal. Inside the modal, the text "Sign up" is centered. Below it, there are three input fields: "Choose a username", "Choose a password", and "Enter your email". At the bottom of the modal, there are two buttons: "SIGN UP" and "CANCEL", both in teal. Below the modal, there is a teal button labeled "BACK".

Sau khi đăng nhập xong, người dùng sẽ quay lại màn hình login để đăng nhập bằng tài khoản vừa tạo. Sau khi đăng nhập xong sẽ được hiện thông báo sau và được chuyển vào màn hình chính.

The image shows a web interface for SmetCollar. At the top, there is a teal header with the text "SmetCollar". Below the header, there is a dark grey modal window titled "login" in teal. Inside the modal, the text "Log in" is centered. Below it, there are two input fields: "Username" with the value "qthang" and "Password" with masked characters. At the bottom of the modal, there is a teal button labeled "LOGIN". Below the modal, there is a small dark grey box with the text "Meet Your Pet!" and a teal button labeled "OK".

Ở màn hình chính, các dữ liệu như nhiệt độ, độ ẩm, mưa được hiện thị. Ngoài ra còn các nút bấm như *FIND MY PET* và *BUZZER*. Đồng thời khi các giá trị này được gửi từ wokwi qua sẽ được thông báo về điện thoại cũng như gửi lên cloud



Cấu hình gửi thông báo từ dịch vụ của IFTTT như sau:

The screenshot shows the IFTTT app configuration screen for sending a rich notification. The title is "Send a rich notification from the IFTTT app". The configuration fields are:

- Title:** Smet Collar
- Message:** Value1
- Link URL:** http://127.0.0.1:1880/ui
- Image URL:** https://play-lh.googleusercontent.com/ITBX-tCdToFDjp2CBf9UhSqlZFJ5KpZXVM68c2PEQyA-xk6Ua-UxRSkXbUl4V_y_QyX=w240-h480-rw

Each field has an "Add ingredient" button. At the bottom, there is an "Update action" button.

Gửi tin nhắn thông báo đến chủ thú cưng thông qua *value1*.

Tuy nhiên việc gửi thông báo qua dịch vụ IFTTT vẫn chưa hoàn hảo. Vì nhóm vẫn chưa giải quyết được vấn đề làm sao để gửi chuỗi có ký tự khoảng cách thông qua ESP32.

Mặc dù gửi test qua trong web private URL với chuỗi có chứa khoảng cách thì vẫn được. Nhưng khi làm theo cách đã học (dùng hàm *sendRequest()* dùng phương thức *print() GET HTTP*) thì lại chuỗi lại tự độ lược bỏ ký tự khoảng cách.

To trigger an Event with 3 JSON values

Make a POST or GET web request to:

```
https://maker.ifttt.com/trigger/SmetCollar/with/key/jW-xnwHKhkbUasbLtw26-BiYI2v9QmDGCuUzInTsKt
```

With an optional JSON body of:

```
{ "value1" : "a b", "value2" : "", "value3" : "" }
```

The data is completely optional, and you can also pass *value1*, *value2*, and *value3* as query parameters or form variables. This content will be passed on to the action in your Applet.

You can also try it with `curl` from a command line.

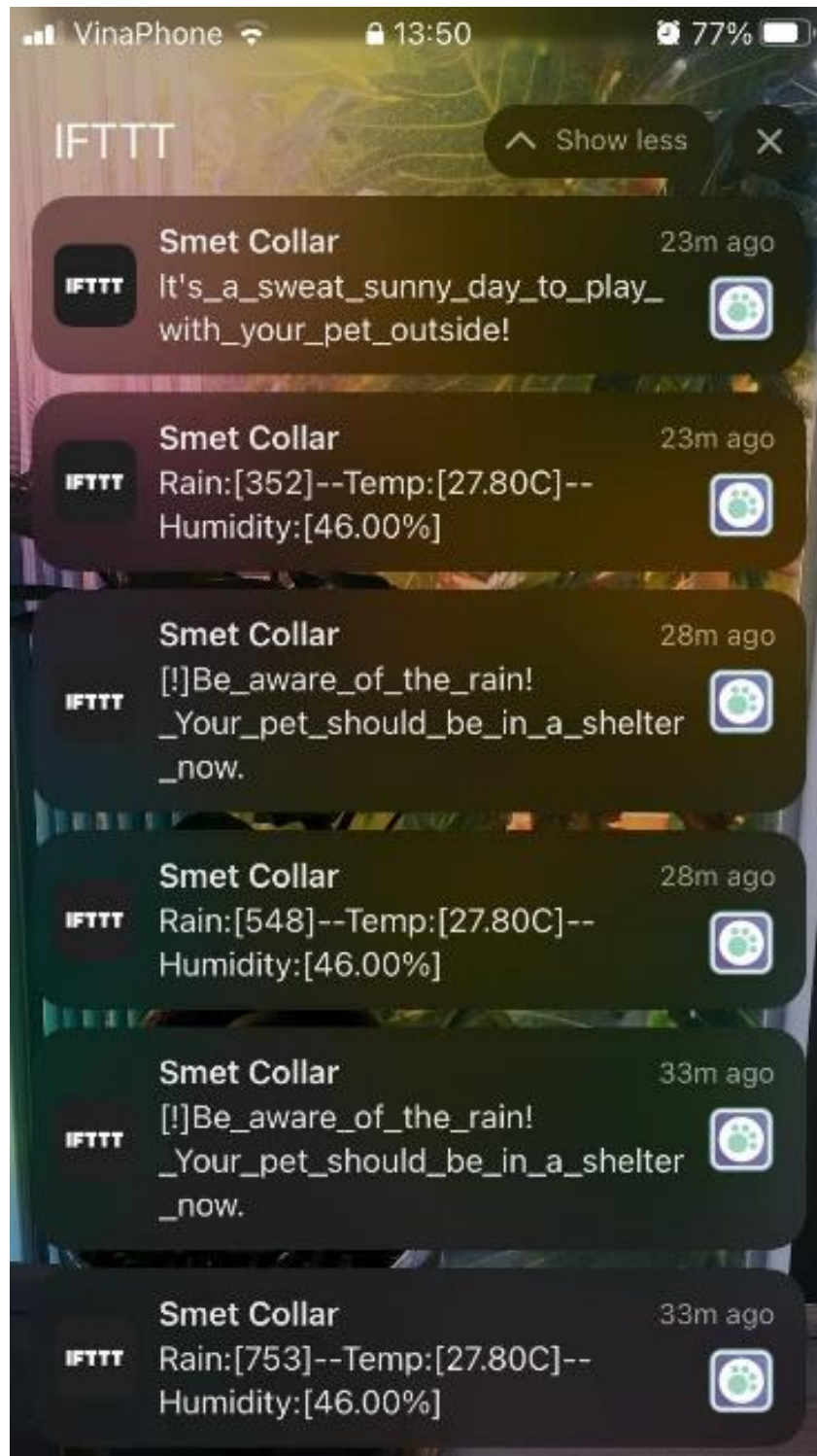
```
curl -X POST -H "Content-Type: application/json" -d '{"value1":"a b b"}' https://maker.ifttt.com/trigger/SmetCollar/with/key/jW-xnwHKhkbUasbLtw26-BiYI2v9QmDGCuUzInTsKt
```

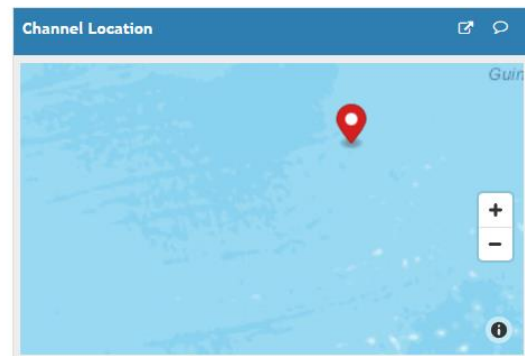
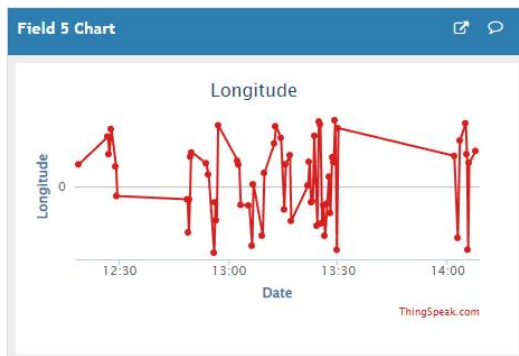
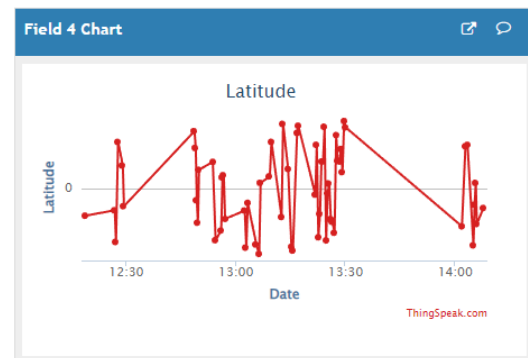
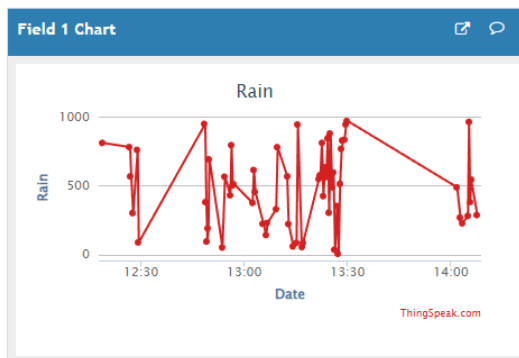
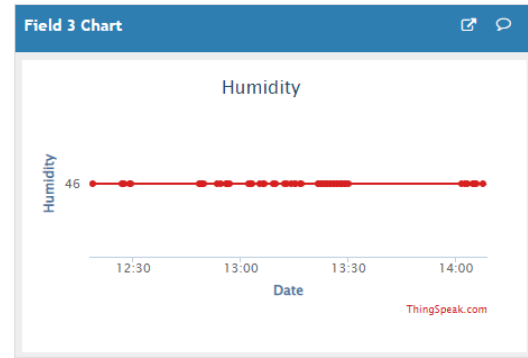
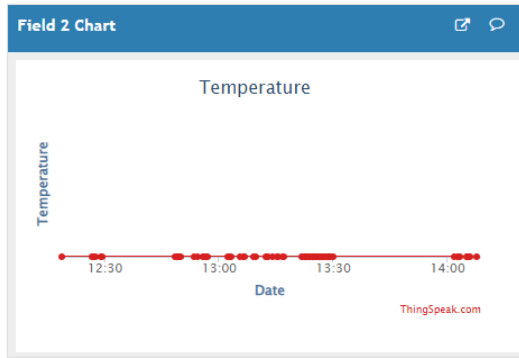
Please read [our FAQ](#) on using Webhooks for more info.

Test It

Nhóm đã có tìm hiểu trên mạng để tìm kiếm nguyên nhân hoặc cách khác nhưng không thành. Vì vậy tạm thời nhóm chỉ cố gắng hoàn chỉnh công việc gửi nhận thông báo nhanh và không bị mất dữ liệu.

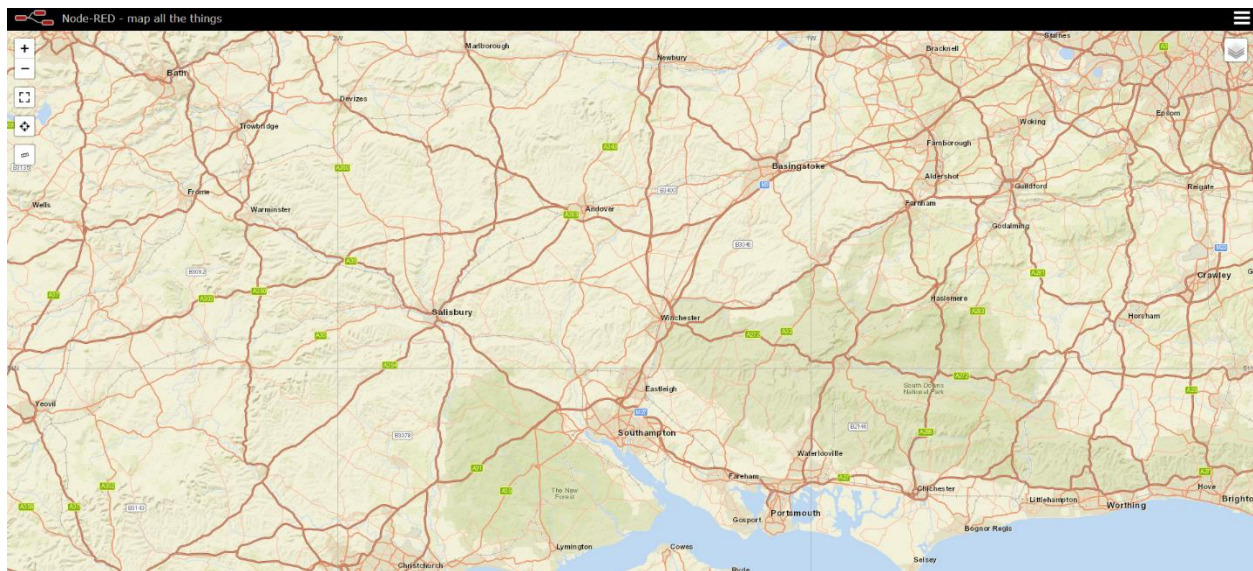
Thông báo trả về sau khi ESP32 xử lý dữ liệu:





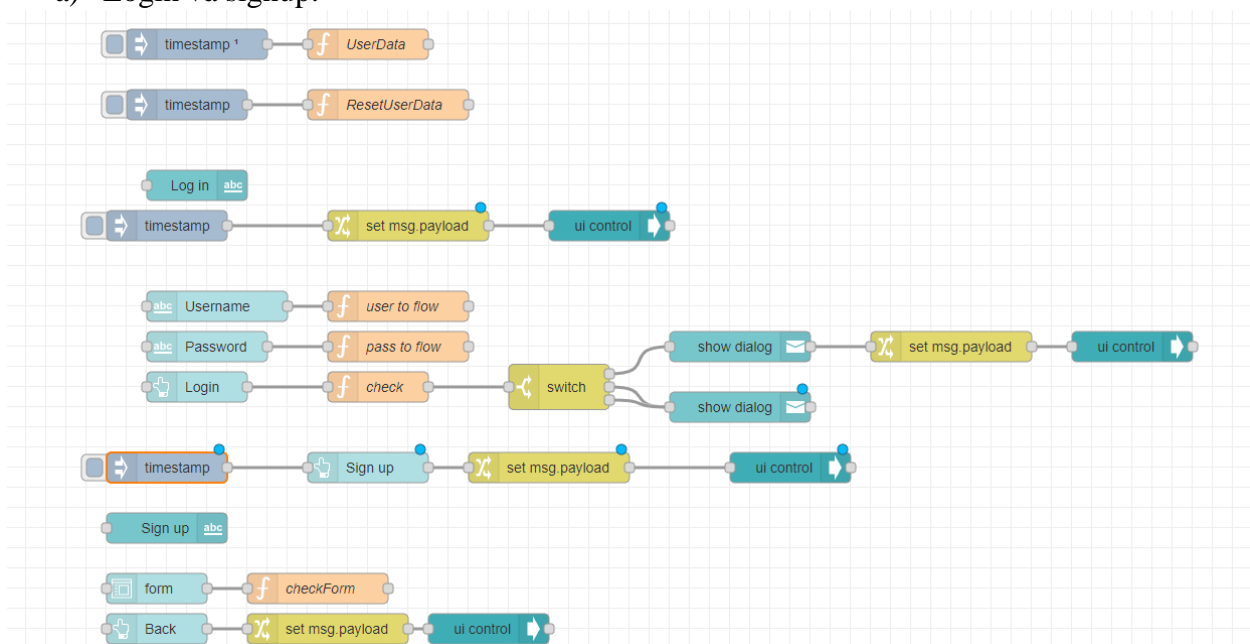
Khi người dùng bấm vào nút **BUZZER**, chuông ở mạch điện sẽ phát ra tiếng kêu để người dùng có thể định vị được thú cưng. Người dùng cần bấm lại 1 lần nữa nếu muốn tắt đi tiếng chuông.

Khi người dùng bấm vào nút **FIND MY PET**, người dùng sẽ được điều hướng tới trang *worldmap* và vị trí thú cưng với tọa độ mới nhất hiện tại được trả về trên bản đồ.



6. Flow node-red.

a) Login và signup.



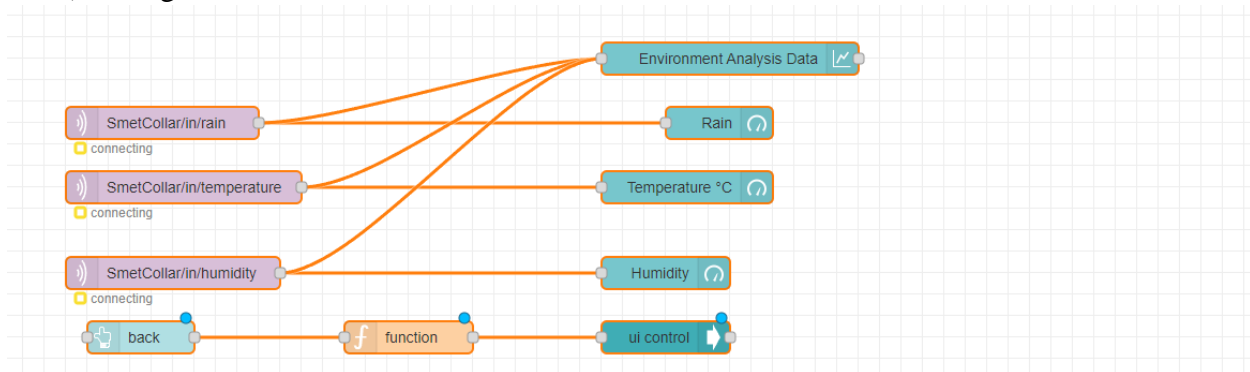
Ở đây, node *Username* sẽ là chỗ giúp cho người dùng có thể nhập vào tên đăng nhập của mình, tương tự với node *Password*, Sau khi người dùng nhập thì sẽ vào node *User to flow* để kiểm tra nếu giá trị khác rỗng thì sẽ set giá trị đó vào *msg.payload* tương tự với *pass to flow*.

Khi bấm nút login, ở node *check* kiểm tra toàn khoản đó đã có và nhập đúng pass thì sẽ trả về 1 chuỗi string và ở node *switch* sẽ kiểm tra chuỗi string đó. Nếu chuỗi string đó “meet your pet” thì sẽ cho người dùng vào trang chính, nếu là “The password is not correct”, hoặc là “Username not exists” thì sẽ thông báo lên qua notification.

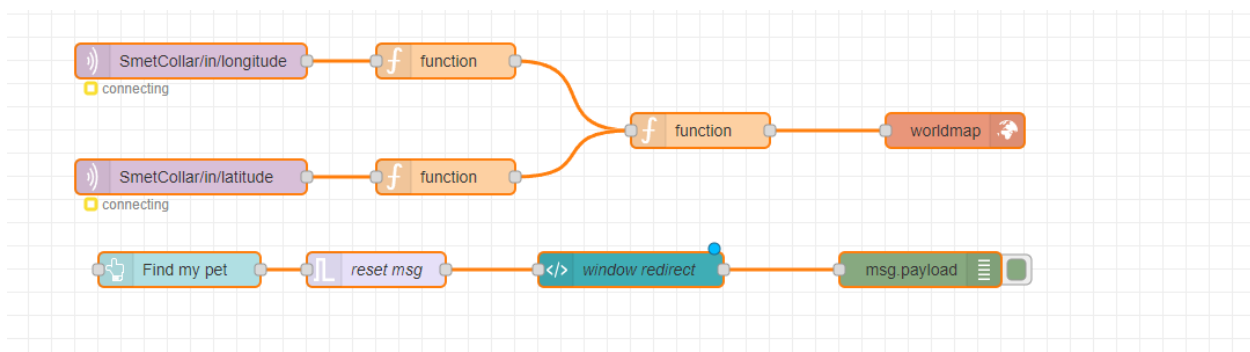
Node *form* là để điền vào 3 giá trị tài khoản, mật khẩu, email để người dùng đăng kí,
Node *checkForm* dùng để kiểm tra xem coi tài khoản có tồn tại hay chưa (do chưa hoàn thiện nên chưa có thể kiểm tra được tài khoản tồn tại)

Node *back* là để quay lại màn hình login từ màn hình signup.

b) Các giá trị cảm biến.



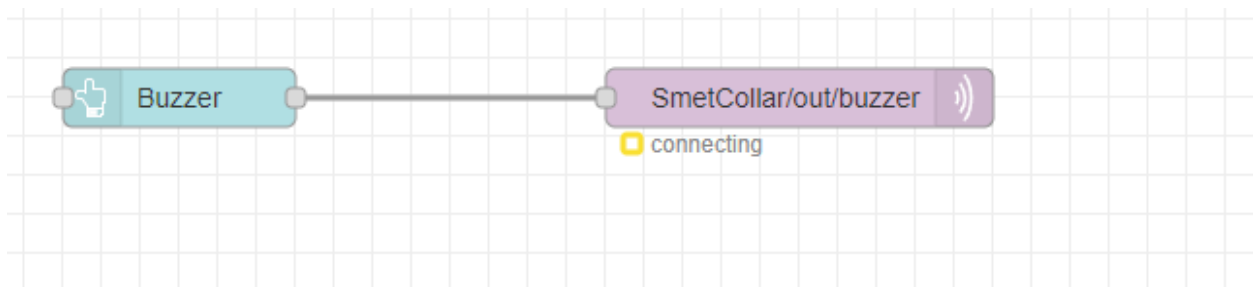
Sử dụng các node *mqtt in* để có truyền các dữ liệu đọc được từ cảm biến ở bên wokwi về lại node-red từ đó hiển thị ra màn hình node-red (value format là *msg.payload*)



Sử dụng 2 node *mqtt in* để lấy các trị vĩ độ và kinh độ từ wokwi về node-red. Sau đó hàm function trước sẽ set tạo flow các giá trị đó, function tiếp theo sẽ get lấy các giá trị đưa thành dạng object cho *msg.payload* rồi truyền vào node *wordmap* [1].

Node *wordmap* được sử dụng từ thư viện *node-red-contrib-web-worldmap* [1].

Ở button *FIND MY PET* sẽ được gán giá trị là *worldmap* nên là khi nào node *window redirect* node này sẽ lấy giá trị đó rồi rồi điều hướng tới <http://127.0.0.1:1880/worldmap/> chứa hình ảnh bản đồ.



Node *BUZZER* ở đây được gán giá trị *BUZZER* sau đó sử dụng *mqtt out* để truyền giá trị qua wokwi từ đó hàm call back sẽ giúp khiến cho chuông kêu.

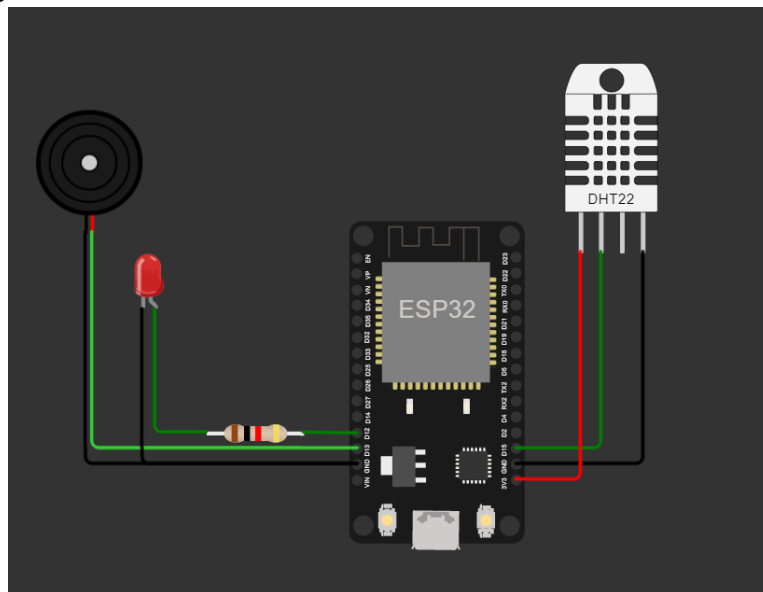
3. CÀI ĐẶT ESP32.

Phần cài đặt ESP32 dưới đây được chạy giả lập trên nền tảng *Wokwi*. Do vậy những linh kiện như GPS NEO6, hay Cảm biến mưa không có sẵn. Nên những giá trị trả về của các cảm biến sẽ được generate ngẫu nhiên trong phạm vi đúng như các giá trị của cảm biến thật.

Đa số các hàm dùng để kết nối lại/kiểm tra kết nối MQTT hay gửi nhận của Servers được sử dụng trong đồ án này đã được học. Vì thế sẽ sử dụng lại và sẽ không giải thích sâu về phần cài đặt chung. Tuy nhiên nếu có sự thay đổi sẽ đưa ra lý do và giải thích rõ.

Mã nguồn: https://github.com/ThongLai/HCMUS_Smet_Collar_IoT

Thiết kế mạch.



- Led tín hiệu: Nối vào chân số 12 và 1 điện trở nhỏ.

- Buzzer: Nối vào chân số 13.
- DHT22: Nguồn VCC sẽ nối vào chân cắm nguồn 3V, Chân tín hiệu SDA sẽ nối vào chân số 15.

Các thư viện cần thiết.

```
#include <WiFi.h>
#include <string.h>
#include "PubSubClient.h"
#include "DHTesp.h"
#include "ThingSpeak.h"
using namespace std;
```

WiFi.h – Giúp thiết lập kết nối WiFi giữa ESP32 với các Servers.

PubSubClient.h – Giúp phân tích trực tuyến dữ liệu và đường ống tích hợp dữ liệu để nhập và phân phối dữ liệu.

DHTesp.h – Đọc nhiệt độ và độ ẩm của cảm biến DHT22

ThingSpeak.h – Giúp giao tiếp với API của ThingSpeak nhanh gọn hơn.

Các biến toàn cục/macros.

Ngoài những biến toàn cục sắp được liệt kê bên dưới. Sẽ có một số biến toàn cục chỉ phục vụ cho 1 số bộ phận hàm nhất định (thường để lưu timestamp tránh dùng câu lệnh delay). Vì vậy để dễ thao tác và lập trình, các biến toàn cục chỉ phục vụ riêng như vậy sẽ để trên các hàm cần sử dụng và sẽ được giải thích trong phần giải thích các hàm đó.

- 1) Định nghĩa các chân cắm của từng linh kiện bằng macros.

```
#define BUTTON 14
#define LED 12
#define BUZZER 13
#define DHT_PIN 15
```

- 2) Định nghĩa ID channel và API Keys của đọc với ghi cho Server ThingSpeak

```
#define CHANNEL_ID 1814183
#define WRITE_API_KEY "NZIGQEYVNFES258N"
#define READ_API_KEY "OWOSMXM9U9Z9FVT2"
```

- 3) Các chuỗi chứa địa chỉ, requests và cổng ports của các Hosts và Servers và cần thiết để thao tác với dữ liệu.

```
const char* ssid = "Wokwi-GUEST";
const char* password = "";

const char* mqttServer = "test.mosquitto.org";

const char * iftttServer = "maker.ifttt.com";
const char * iftttRequest =
"/trigger/SmetCollar/with/key/jW-xnwHKhkbUasbLtw26-
BiYI2v9QWDGCUUzinTsKt?value1=";

const char * thinkSpeakServer = "api.thingspeak.com";

const int PORT = 80;
const int mqttPort = 1883;
```

- 4) Topic đã cài trong MQTT input và output của Node-Red.

```
const char* RAIN_TOPIC = "SmetCollar/in/rain";
const char* TEMP_TOPIC = "SmetCollar/in/temperature";
const char* HUMIDITY_TOPIC = "SmetCollar/in/humidity";
const char* LATITUDE_TOPIC = "SmetCollar/in/longitude";
const char* LONGITUDE_TOPIC = "SmetCollar/in/latitude";

const char* BUZZER_TOPIC = "SmetCollar/out/buzzer";
```

- 5) Tạo lớp WiFiClient cho 3 dịch vụ server sử dụng.

```
WiFiClient espClient;
WiFiClient iftttclient;
WiFiClient TSclient;

PubSubClient mqttClient(espClient);
```

- 6) Nhận dữ liệu lấy từ cảm biến dùng để xử lý và truyền đi qua các server.

```
//DATA
int RAIN = 0;
float TEMP = 0;
float HUMIDITY = 0;
float LATITUDE = 0;
float LONGITUDE = 0;
```

Trong đó RAIN, LATITUDE, LONGITUDE là 3 biến số sẽ phải phát sinh ngẫu nhiên.

Các hàm.

1) *setup()*

```
void setup() {
    // put your setup code here, to run once:
    Serial.begin(115200);
    Serial.println("Connecting to Wifi");

    digitalWrite(LED, HIGH);
    wifiConnect();

    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());

    mqttClient.setServer(mqttServer, mqttPort);
    mqttClient.setCallback(callback);

    ThingSpeak.begin(TSClient);

    dht.setup(DHT_PIN, DHTesp::DHT22);
    pinMode(LED, OUTPUT);
    pinMode(BUZZER, OUTPUT);

    delay(10000);
    digitalWrite(LED, LOW);
}
```

- Phần *setup()* sẽ gọi hàm *wifiConnect()* để connect wifi với ESP32.
- Dùng phương thức *setServer()* để set địa chỉ host và port và *setCallback()* để set hàm được chạy ghi được gọi.
- Khởi tạo lớp *ThingSpeak* với *WiFiClient*.
- Set up các chân linh kiện và cảm biến DHT22.
- Đồng thời sẽ delay 10 giây để các kết nối được ổn định. Trong quá trình này Led tín hiệu sẽ phát sáng cho tới khi kết thúc 10 giây delay.

2) *uploadData()*

```
unsigned int DelayRequired = 10000;
unsigned long PrevMillis = millis();
void uploadData()
{
    if(millis() - PrevMillis > DelayRequired)
    {
        PrevMillis = millis();
        printData();

        //Send to ThingSpeak Cloud
        ThingSpeak.setField(1, RAIN);
        ThingSpeak.setField(2, TEMP);
        ThingSpeak.setField(3, HUMIDITY);
        ThingSpeak.setField(4, LATITUDE);
        ThingSpeak.setField(5, LONGITUDE);

        int sc = ThingSpeak.writeFields(CHANNEL_ID, WRITE_API_KEY);

        if (sc == 200)
            Serial.println("[Sent to Cloud]\n");




        //Send Data through MQTT Server
        mqttClient.publish(RAIN_TOPIC, (to_string(RAIN)).c_str());
        mqttClient.publish(TEMP_TOPIC, (String(TEMP, 2)).c_str());
        mqttClient.publish(HUMIDITY_TOPIC, (String(HUMIDITY, 1)).c_str());
        mqttClient.publish(LATITUDE_TOPIC, String(LATITUDE).c_str());
        mqttClient.publish(LONGITUDE_TOPIC, String(LONGITUDE).c_str());

        Serial.println("[Sent Data]\n");
    }
}
```

Mục tiêu chính: Upload tất cả 5 dữ liệu của cảm biến lên Cloud và MQTT.

- Do dịch vụ ThingSpeak mà nhóm đang sử dụng là bản miễn phí, tức chỉ cho nhận trong khoảng 15s cho 1 dữ liệu từng đợt. Do vậy nên buộc phải delay tốc độ chuyển dữ liệu qua cloud của ThingSpeak (xuống còn 10s vì nhóm thấy đây là thời gian tương đối tốt, dữ liệu không bị mất hay delay nhiều).
- *DelayRequired* sẽ chỉ định khoảng thời gian cần delay trước khi chuyển dữ liệu kế tiếp (10s). *PrevMillis* chứa timestamp của mốc dữ liệu gần nhất được chuyển đi.
- Để gửi dữ liệu 1 thấy hết 5 fields data. Thư viện ThingSpeak cung cấp phương thức cho phép gửi đồng đều như sau:

- Đầu tiên *setField()* cả 5 giá trị cần gửi cho 5 fields.
- Sau đó gọi phương thức *writeFields()* (thay *writeField()* như thông thường chỉ gửi được 1 giá trị) và truyền vào ID của channel và API KEY để write.

<div> <div>  <p>Connect more sensors</p> </div> <div>  <p>Share channel views with friends</p> </div> <div>  <p>Faster update rates</p> </div> </div>		
	FREE For small non-commercial projects	STUDENT For students at degree-granting institutions ⁽¹⁾
Scalable for larger projects	✗ No. Annual usage is capped.	✓
Number of messages	3 million/year (~8,200/day) ⁽²⁾	33 million/year per unit (~90,000/day per unit) ⁽²⁾
Message update interval limit	Every 15 seconds	Every second
Number of channels	4	10 per unit
MATLAB Compute Timeout	20 seconds	20 seconds
Private channel sharing	Limited to 3 shares	Unlimited
Technical Support	Community Support	Community Support
Max image size	✗ Image feature unavailable	5 MB
Messages used per image	✗	100

- Tiếp đến là gửi cho MQTT đến Node-RED. Server của MQTT thực chất không cần delay nhiều vậy vẫn gửi đi và không bị mất dữ liệu. Tuy nhiên nhóm cũng không cần dữ liệu biến động liên tục trên trang web người dùng của Node-RED nên sẽ gửi lên Cloud và gửi lên hiển thị trên Node-RED cùng 1 lúc.
- Gửi dữ liệu dễ dàng qua MQTT nhờ phương thức *publish()* của *WiFiClient*. Tham số là chuỗi topic đã được định nghĩa trước bên node MQTT của Node-RED và giá trị dữ liệu cần gửi đi.

3) *updateData()*

```
void updateData()
{
    TEMP = dht.getTemperature();
    HUMIDITY = dht.getHumidity();
    RAIN = random(1024);
    LATITUDE = random(-9000, 9000)*0.01;
    LONGITUDE = random(-18000, 18000)*0.01;
}
```


Mục tiêu chính: Đây là hàm đóng gói, thực hiện đồng thời cập nhật lại tất cả giá trị của cảm biến.

- RAIN, LATITUDE, LONGITUDE sẽ được phát sinh ngẫu nhiên đúng theo giá trị khoảng của các cảm biến thực tế trả về.

4) *printData()*

```
void printData()
{
    Serial.println("Rain: " + String(RAIN));
    Serial.println("Temp: " + String(TEMP, 2) + "°C");
    Serial.println("Humidity: " + String(HUMIDITY, 2) + "%");
    Serial.println("Langtitude: " + String(LATITUDE));
    Serial.println("Longitude: " + String(LONGITUDE));
}
```

Mục tiêu chính: Đây là hàm đóng gói, thực hiện đồng thời việc in ra Serial các giá trị của cảm biến.

5) *callback()*

```
bool isbuzzerOn = false;
void callback(char* topic, byte* message, unsigned int length) {
    String stTopic = topic;
    String stMessage = (char*)message;

    Serial.println(stTopic);
    Serial.println(stMessage);

    if (stTopic == BUZZER_TOPIC)
        isbuzzerOn = !isbuzzerOn;
}
```

Mục tiêu chính: Sẽ được thực thi khi node MQTT output của Node-RED đưa về dữ liệu. Ở sản phẩm này chỉ xử lý buzzer khi được bấm nút từ Node-RED.

- Mục tiêu của nhóm mong muốn rằng khi nhấn nút *BUZZER* thì còi sẽ vang lên theo nhịp điệu cho đến khi được bấm lần thứ 2 thì dừng. Tuy nhiên nếu xử lý luôn thao tác đó của buzzer trong hàm *callback()* này thì không delay ra nhịp điệu được. Vì vậy hàm này chỉ có nhiệm vụ gửi về tính hiệu nút đã được nhấn cho hàm khác ở vòng *loop()* xử lý.

- Biến kiểu bool *isbuzzerOn* thể hiện trạng thái đã được nhấn “lún” của buzzer. Tức buzzer chỉ ngừng kêu khi trạng thái là False.
- *topic* ở kiểu *char** và *message* kiểu *byte** được gửi từ Node-RED sẽ được ép kiểu sang *StTopic* và *StMessage* ở dạng *String*. (Khác với hàm được học mọi khi là duyệt từng kí tự của *byte** để chuyển sang *String*)
- Sau đó kiểm tra topic được gửi có đúng topic của Buzzer. Nếu có sẽ cho biến *isbuzzerOn* về trạng thái True để còi kêu.

6) *triggerBuzzer()*

```
unsigned long startBuzz = millis();
void triggerBuzzer()
{
    if (millis() - startBuzz > 500)
    {
        startBuzz = millis();
        digitalWrite(LED, HIGH);
        tone(BUZZER, 1000, 100);
        tone(BUZZER, 800, 100);
    }
    digitalWrite(LED, LOW);
    // noTone(BUZZER);
}
```

Mục tiêu chính: Hàm này sẽ hỗ trợ hàm *callback()* để xử lý Buzzer.

- Để buzzer có thể kêu theo nhịp cần có delay vì thế nên sẽ sử dụng biến *startBuzz* để không mắc phải trường hợp Buzzer kêu in ỏi, làm thú cưng hoảng hốt.
- Đồng thời sẽ cho Led tín hiệu chớp tắt liên tục, gây sự chú ý khi đang bị thất lạc thú cưng hoặc vòng cổ.

7) *eveCheck()*

```
String envCheck()
{
    String msg = "";
    if (RAIN >= 1024/3 && RAIN <= 1024/2 && TEMP >= 20 && TEMP <= 33) //Sunny
Day
        msg = "It's_a_sweet_sunny_day_to_play_with_your_pet_outside!";
    else if (RAIN >= 1024/2) //Rainy Day
        msg = "[!]Be_aware_of_the_rain!_Your_pet_should_be_in_a_shelter_now.";
    else if (TEMP <= 10) // Cold day
        msg = "[!]What_a_chilly_day,_your_pet_needs_a_cozy_corner!";
    else if (HUMIDITY >= 60) //The pet is wet
        msg = "[!]Your_pet_is_soaking_wet_right_now!";

    return msg;
}
```

Mục tiêu chính: Kiểm tra và trả về sự kiện thời tiết đặc biệt. Lấy các dữ liệu hiện tại, xử lý và phỏng đoán thời tiết, nhiệt độ ngoài trời để gửi thông báo về điện thoại người chủ.

*Phần này nhóm vẫn chưa hoàn thiện, do chưa biết cách gửi thông báo chuỗi có ký tự khoảng cách thông qua IFTTT. Mặc dù ở trang test của WebHook vẫn cho gửi qua value1 với chuỗi có ký tự trắng. Tuy nhiên khi gửi bằng *GET* HTTP như đã học thì tự động xóa đi hết khoảng trắng(kể cả '\t')

- Tạo biến chứa tin nhắn cần thông báo đến chủ và kiểm tra từng điều kiện như lượng mưa trên 1/3 và dưới 1/2 lượng mưa tối đa thì là trời mây và nhiệt độ không quá nắng từ 20 -> 33 độ C thì mát mẻ phù hợp với các hoạt động ngoài trời với thú cưng.
- Các cảnh báo “xấu” như thú cưng của bạn bị ướt hay trời đang có giông lớn, cần đưa thú cưng vào nơi khô ráo sẽ được trả về với tin nhắn cảnh báo có “[!]” thể hiện thông báo không tốt và cần sự chú ý.

8) *loop()*

```

unsigned long startTimeUpdate = millis();
unsigned long DelayNotify = 3000;
unsigned long startNotify = millis() ;
void loop() {

    if (!mqttClient.connected()){
        mqttReconnect();
    }

    mqttClient.loop();

    if (millis() - startTimeUpdate > 3000)
    {
        startTimeUpdate = millis();

        //Update all data once
        updateData();

        if (millis() - startNotify > DelayNotify)
        {
            startNotify = millis();
            String msg = envCheck();
            if (msg != "")
            {
                sendRequest(iftttServer, iftttRequest, "Rain:[" + String(RAIN) + "]--
Temp:[" + String(TEMP) + "C]--Humidity:[" + String(HUMIDITY) + "%]");
                delay(500);
                sendRequest(iftttServer, iftttRequest, msg);

                Serial.println("[Sent Notifications]\n");
                DelayNotify = 28800000;
            }
            else
                DelayNotify = 3000;
        }
    }

    if (isbuzzerOn)
        triggerBuzzer();

    uploadData();
}

```

- Hàm vòng lặp chính của ESP32. Dùng để gọi các hàm bắt sự kiện gửi tin nhắn, kết quả thông báo với delay đã định trước.
- Đầu tiên cần kiểm tra liên tục Server MQTT còn hoạt động hay không bằng phương thức *connected()* và sẽ kết nối lại bằng hàm *mqttReconnect()*.
- Phần if lớn kế tiếp mục tiêu chính là để cập nhật dữ liệu qua hàm *updateData()*, đồng thời mỗi lần cập nhật dữ liệu mới. Sẽ gọi hàm *venvCheck()* để kiểm tra sự kiện thời tiết đặt biệt, nếu chuỗi trả về trống thì không có sự kiện đặt biệt đối với các thông số dữ liệu hiện tại.
- Tuy nhiên do không muốn mỗi milli giây đều phải cập nhật dữ liệu nên sẽ tạo biến toàn cục *startTimeUpdate* để chứa timestamp của lần cập nhật gần nhất. Với biến này, sẽ cài đặt tốc độ cập nhật dữ liệu sẽ còn xấp xỉ 3s/1 lần.
- Vấn đề của việc gửi thông báo đến cho người dùng là phải gửi đúng thời điểm cần thiết, tránh gửi quá nhiều và liên tục (do nhiệt độ, độ ẩm thay đổi khá chậm nên phải cần sự delay sau khi đã gửi 1 thông báo).
- Hai biến *DelayNotify* chỉ định khoảng thời gian delay và *startNotify* sẽ lưu timestamp lần gửi thông báo gần nhất. Sau đó, nhóm quyết định cài đặt sẽ gửi 8 tiếng/1 lần.

4. BẢNG PHÂN CÔNG VIỆC.

STT	Công việc	Người được phân công
1	Lắp mạch mô phỏng trên nền tảng mô phỏng	Thông, Minh, Thắng
2	Tạo website để có thể kết nối với mạch mô phỏng đã được tạo.	Thắng, Minh
3	Thiết lập hệ thống đăng nhập	Thắng
4	Chỉnh sửa, trau chuốt lại giao diện	Thông
5	Tải những dữ liệu lên cloud và gửi về điện thoại.	Thông
6	Kiểm tra chạy thử	Thông, Thắng, Minh
7	Viết báo cáo phần flow	Thắng, Minh
8	Viết báo cáo phần mã nguồn ESP	Thông
9	Vẽ mô hình 3D	Minh
10	Vẽ flow nhận gửi thông số dữ liệu	Thắng, Thông
11	Tìm hiểu thư viện phục vụ cho đồ án	Thông
12	Lên kế hoạch nội dung thuyết trình, video	Thắng, Minh
13	Chỉnh sửa video demo	Thông

5. THAM KHẢO

File đồ án: https://github.com/ThongLai/HCMUS_Smet_Collar_IoT

Video Demo sản phẩm: <https://youtu.be/cRuoLIzQFI>

Thư viện vẽ và địa vị bản đồ:

[1] <https://flows.nodered.org/node/node-red-contrib-web-worldmap>