

BÁO CÁO THỰC HÀNH

CÔNG NGHỆ INTERNET OF THINGS HIỆN ĐẠI

Lab 3

THỰC HÀNH: HTTP MQTT in IoT Platform

GVHD: **Phan Trung Phát**

Lớp: **NT532.021**

Họ và tên	MSSV
Nguyễn Thành Đăng	21520683
Nguyễn Trần Bảo Quốc	21520421

ĐÁNH GIÁ KHÁC (*):

Nội dung	Kết quả
Tổng thời gian thực hiện bài thực hành trung bình (1)	2 tuần
Link Video thực hiện (2) (nếu có)	Bài 1 Bài 2 Bài 3-Rasp Bài 3-HiveMQ Bài 4 Bài 5 Drive tổng hợp tất cả các file liên quan
Ý kiến (3) (nếu có) + Khó khăn + Đề xuất ...	
Điểm tự đánh giá (4)	10/10

(*): phần (1) và (4) bắt buộc thực hiện.

Phần bên dưới là báo cáo chi tiết của nhóm/cá nhân thực hiện.



LƯU HÀNH NỘI BỘ

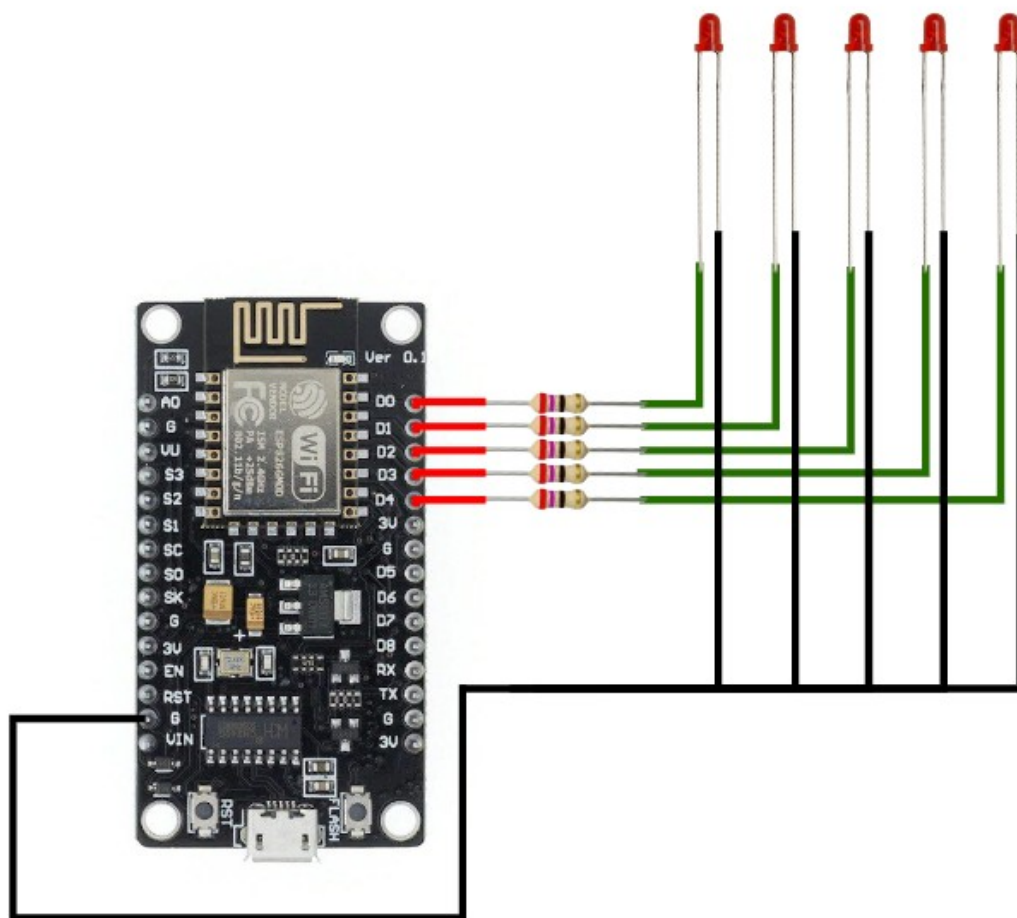


Câu hỏi 1. Lấy ý tưởng từ bài 5 - bài thực hành số 1 “thử tài đoán số”, xây dựng kịch bản gồm 1 Wemos D1, 5 đèn LED. HTTP Server được dựng trên Wemos D1. Trò chơi được dừng lại khi người chơi không còn điểm. Bắt đầu với 5 đèn LED sáng từ trái sang phải và ngược lại, sau đó trò chơi sẽ ngẫu nhiên 1 số lượng đèn bất kỳ. Đèn được hiển thị trong khoảng thời gian là 2 giây sau đó tắt đi, vị trí của các đèn được thiết đặt một cách ngẫu nhiên. Người chơi cần nhanh trí đếm số lượng đèn và % 3, thực hiện chọn nút tương ứng trên giao diện website trong khoảng 2 giây tiếp theo. Nếu người dùng chọn đúng thì sẽ được cộng điểm và bị trừ điểm nếu chọn sai hoặc hết thời gian. Điểm số và tình trạng của trò chơi sẽ được hiển thị qua giao diện website.

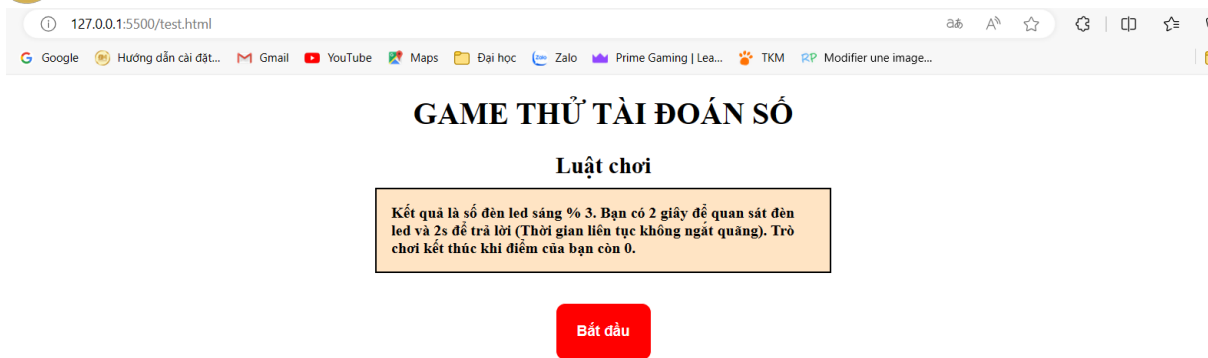
1. Minh chứng:

Link demo bài 1: [Tại đây](#)

Để thuận tiện cho tiến độ thực hiện bài lab nên nhóm đã sử dụng **ESP8266** thay cho **WemosD1** (2 thiết bị tương đồng).



Hình 1. Mô hình nối dây bài 1

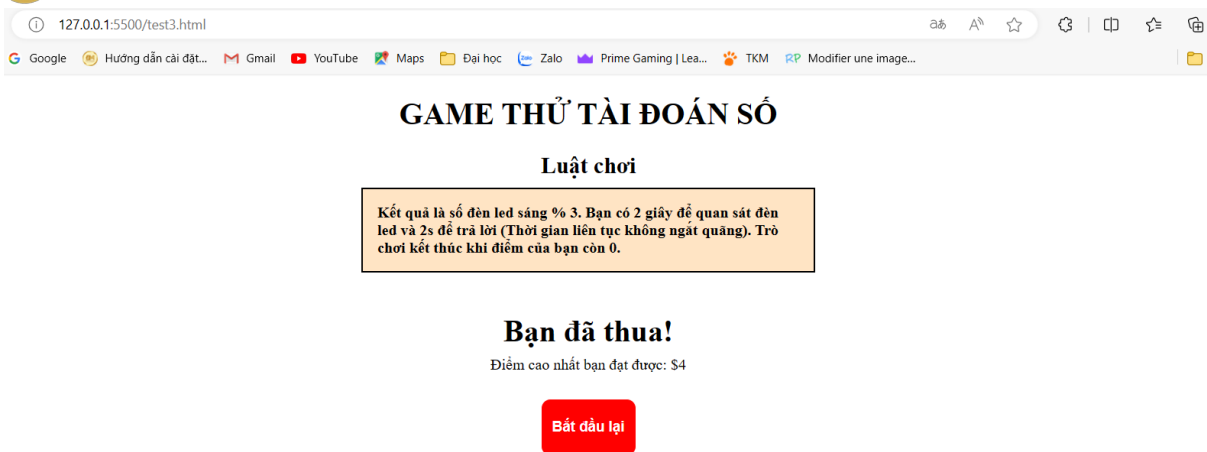


14

Hình 2. Giao diện home của game xem trên localhost



Hình 3. Giao diện game quan sát trên localhost



14

Hình 4. Giao diện khi thất bại xem trên localhost (do giao diện html nên chưa có data truyền vào)

Code của bài khá dài do có html nên em không thêm vào báo cáo(thầy có thể xem ở file ino kèm theo).

2. Giải thích:

```
5 ESP8266WebServer server(80);
6 const char* ssid = "TheLight";
7 const char* password = "20022003";
8
9 int led[] = {16, 5, 4, 0, 2};
10
11 int guessedNumber = -1;
12 int score = 0;
13 int maxScore = 0;
14 int question = 0;
15 int questionComp = 0;
16 bool isStart = false;
17 bool isFirst = true;
18 int interval= 3000;
19 unsigned long previousMillis = 0;
20 unsigned long currentTime = 0;
21
```

Hình 5. Khai báo tham số

Đầu tiên là khai báo thư viện Wifi và WebServer cho ESP. WebServer này được chạy trên port 80 (Line 5).

Line 6, line 7 là thông tin kết nối của wifi.

Line 9 khai báo chân Led của ESP ↔ chân D0, D1, D2, D3 và D4.

Line 11 → Line 20 khai báo các biến toàn cục cho chương trình:

- **guessedNumber** là đáp án người chơi lựa chọn (Question % 3).



- **score** là điểm của người chơi.
- **maxScore** là điểm cao nhất mà người chơi nhận được.
- **question** là số đèn sáng được random. Còn **questionComp** là số đèn sáng được so sánh với đáp án người chơi.
- **isStart** là biến để kiểm tra xem trò chơi đã bắt đầu chưa (phục vụ việc đèn sáng từ trái sang phải và ngược lại mỗi khi game mới bắt đầu). Biến **isFirst** kiểm tra xem game đã trải qua lượt đầu tiên chưa (do lượt đầu tiên và các lượt khác có xử lý khác nhau).
- **Interval**, **previousMillis**, **currentTime** là các biến hỗ trợ việc đèn sáng 2s thì tắt đi (**Không sử dụng delay** vì khi delay(2000) thì tốc độ chương trình bị giảm xuống không tối ưu do xử lý bất đồng bộ).

```
23 void swap(int *a, int *b) {
24     int temp = *a;
25     *a = *b;
26     *b = temp;
27 }
28
29 void randomArray(int arr[], int size) {
30     srand(time(NULL));
31
32     for (int i = size - 1; i > 0; --i) {
33         int j = rand() % (i + 1);
34         swap(&arr[i], &arr[j]);
35     }
36 }
```

Hình 6. Random mảng led

Random mảng led để khi sáng đèn cho người chơi quan sát, vị trí sáng đèn được random (thứ tự chân đèn phát sáng bị thay đổi trong mảng).

```
37
38 String html_Home_Page()
39 {
40     return "<html> <head> <meta charset='utf-8'> </head> <style> .btn{ margin: 10px; width: 100px; height: 60px; border-r
41 }
42 String html_Game_Page(int sc)
43 {
44     String html = "<html> <head> <meta charset='utf-8'> </head> <style> .btn{ margin: 10px; width: 100px; height: 60px; b
45     html+= String(sc) + "</h2> </div> <div style='display: flex; flex-wrap: wrap; justify-content: center;'> <h3 class='
46
47     String htmlOver = "<html> <head> <meta charset='utf-8'> </head> <style> .btn{ margin: 10px; width: 100px; height: 60p
48     htmlOver += String(maxScore) + "</p> </div> <div style='display: flex; justify-content: center;'> <Button onclick='
49     if ( isFirst || sc > 0)
50         return html;
51     score = 0;
52     guessedNumber = -1;
53     isFirst = true;
54     isStart = false;
55     question = 0;
56     questionComp = 0;
57     return htmlOver;
58 }
```

Hình 7. Giao diện trò chơi



Thầy có thể xem html rõ hơn trong file html đính kèm.

html_Home_Page là giao diện bắt đầu ở *Hình 2* của báo cáo này.

html_Game_Page sẽ có 2 giao diện là đang chơi (*Hình 3*) và giao diện thua cuộc (*Hình 4*). Khi điểm lớn hơn 0 hoặc là lần đầu tiên thì nó sẽ hiển thị giao diện đang chơi và ngược lại sẽ là giao diện thua cuộc và reset các biến về lúc khởi tạo.

14

```
108 void setup() {
109
110     pinMode(16, OUTPUT);
111     pinMode(5, OUTPUT);
112     pinMode(4, OUTPUT);
113     pinMode(0, OUTPUT);
114     pinMode(2, OUTPUT);
115
116     Serial.begin(9600);
117     WiFi.begin(ssid, password);
118
119     Serial.println("");
120     while (WiFi.status() != WL_CONNECTED) {
121         delay(500);
122         Serial.print(".");
123     }
124     Serial.print("Connected to "); Serial.println(ssid);
125     Serial.print("IP address: "); Serial.println(WiFi.localIP());
126     server.on("/", handleRoot);
127     server.on("/guess", handleGuess);
128     server.begin();
129 }
130
```

Hình 8. Hàm setup

Khai báo các chân led, thiết lập kết nối wifi (*Line 117*), định nghĩa route cho Webserver (*Line 126, Line 127*) và Chạy Webserver ở *Line 128*.

- Với đường dẫn mặc định thì thực thi **handleRoot**.
- Với đường dẫn *'/guess'* thì thực hiện **handleGuess**.

```
181 void loop() {
182     gameStart(!isStart);
183     server.handleClient();
184
185     currentTime = millis();
186     if (currentTime - previousMillis <= interval) {
187         for(int i = 0; i < question; i++)
188         {
189             digitalWrite(led[i], HIGH);
190         }
191     }
192     else
193     {
194         for(int i = 0; i < 5; i++)
195         {
196             digitalWrite(led[i], LOW);
197         }
198     }
199 }
200
```

Hình 9. Hàm loop

Hàm **gameStart** là kiểm tra xem đây có phải là game mới bắt đầu không để nó có thể thực hiện việc sáng đèn từ trái sang phải và ngược lại.



Hàm **handleClient** là hàm được cung cấp bởi WebServer nó lắng nghe và thực hiện các handle được khai báo trong Route của nó.

Từ *line 185 đến line 198* là thực hiện sáng led 2s. (Nằm trong loop nhưng không sử dụng delay làm ảnh hưởng tới hiệu suất của ứng dụng).

```
void handleRoot() {  
    server.send(200, "text/html", html_Home_Page());  
}
```

Hình 10. Hàm *handleRoot* chỉ là hiển thị thành công trang *html_Home_Page*

```
65 void handleGuess() {  
66     digitalWrite(16, LOW);  
67     digitalWrite(5, LOW);  
68     digitalWrite(4, LOW);  
69     digitalWrite(0, LOW);  
70     digitalWrite(2, LOW);  
71     question = random(1,6);  
72     randomArray(led, 5);  
73  
74     previousMillis = currentTime;  
75  
76     guessedNumber = server.arg("answer").toInt();  
77     if (isFirst)  
78     {  
79         server.send(200, "text/html", html_Game_Page(0) );  
80         isFirst = false;  
81         score = 0;  
82         questionComp = question;  
83         Serial.println(question);  
84         return;  
85     }  
86     if ( !(guessedNumber < 0 || guessedNumber > 2) && !isFirst)  
87     {  
88         isFirst = false;  
89         if (guessedNumber == questionComp % 3 ) {  
90             score++;  
91             maxScore = score;  
92             server.send(200, "text/html", html_Game_Page(score) );  
93         }  
94         else {  
95             score--; // Trừ điểm nếu đoán sai  
96             server.send(200, "text/html", html_Game_Page(score) );  
97         }  
98     } if (guessedNumber == 3)  
99     {  
100         score--; // Trừ đi (const char [10])"text/html"  
101         server.send(200, "text/html", html_Game_Page(score) );  
102     }  
103  
104     questionComp = question;  
105     Serial.println(question);  
106 }  
107
```

Hình 11. Hàm *handleGuess*

Đầu tiên là tắt hết các led trước đó (Đảm bảo không có vấn đề xung đột luồng led sáng).

Gán *currentTime* cho *previousMillis* (do đã tiến hành thực thi câu hỏi khác).



guessedNumber = server.arg("answer").toInt() là để lấy kết quả mà người chơi đã dự đoán trong *url parameters*.

```
<button onclick="HREF('/guess?answer=0')" class="btn">0</button>
<button onclick="HREF('/guess?answer=1')" class="btn">1</button>
<button onclick="HREF('/guess?answer=2')" class="btn">2</button>
```

Hình 12. Href của cái nút kết quả

Kiểm tra nếu là lần đầu tiên thì chỉ cần lưu lại question vào **questionComp** (do lúc này game vừa mới chạy, người chơi chưa hề chọn đáp án, url lúc này vẫn là *"/guess"*).

Nếu đây là lượt chơi thứ *n* thì kiểm tra kết quả lấy từ url parameters đã chính xác chưa, nếu chưa chính xác thì thực hiện trừ điểm và ngược lại (giao diện thua hay thắng đã được định nghĩa trong hàm **html_Game_Page**).

```
21 <script>
25   function countdown() {
27       let id = setInterval(() => {
28           const qs = document.getElementById('qs');
29           const tl = document.getElementById('tl');
30           qs.style.color = "red";
31           if (x < 2 && x > -0.99)
32           {
33               tl.innerHTML = "Thời gian lựa chọn: " + Math.round(x*10)/10;
34               qs.style.color = "black";
35               tl.style.color = "red";
36               qs.innerHTML = "Thời gian quan sát: 0"
37           }
38           else qs.innerHTML = "Thời gian quan sát: " + Math.round((x - 2)*10)/10;
39           if (x < 0) {
40               clearInterval(id);
41               tl.style.color = "black";
42               qs.style.color = "black";
43               qs.innerHTML = "Thời gian quan sát: 0";
44           }
45           x -= 0.1;
46           if (x <= 0) window.location.href = "/guess";
47       }, 100);
48   }
49   function HREF(par) {
50       window.location.href = par;
51   }
52 </script>
```

Hình 13. Hàm đếm ngược timeout

Timeout được em xử lý bằng JavaScript đặt trong **html_Game_Page**. Khi kết thúc 4s, nó tự động chuyển sang *"/guess"*, khi này chương trình trên ESP sẽ không phân tích được answer từ url parameter nên mặc định nó sẽ xử lý theo hướng là người chơi chọn sai.

Câu hỏi 2. Xây dựng kịch bản thiết bị gồm 1 Wemos D1, 1 cảm biến ánh sáng, 1 cảm biến khoảng cách và 3 đèn LED. Trong đó, máy tính cá nhân như một Server, sử dụng một ngôn ngữ lập trình bất kỳ để xây dựng các RESTful API (chủ yếu sử dụng hai phương thức GET và POST). Sử dụng Wemos D1 như một HTTP Client, định kỳ mỗi 5s Wemos D1 thu thập giá trị cảm biến ánh sáng và cảm biến khoảng cách, gửi các giá trị cảm biến với định

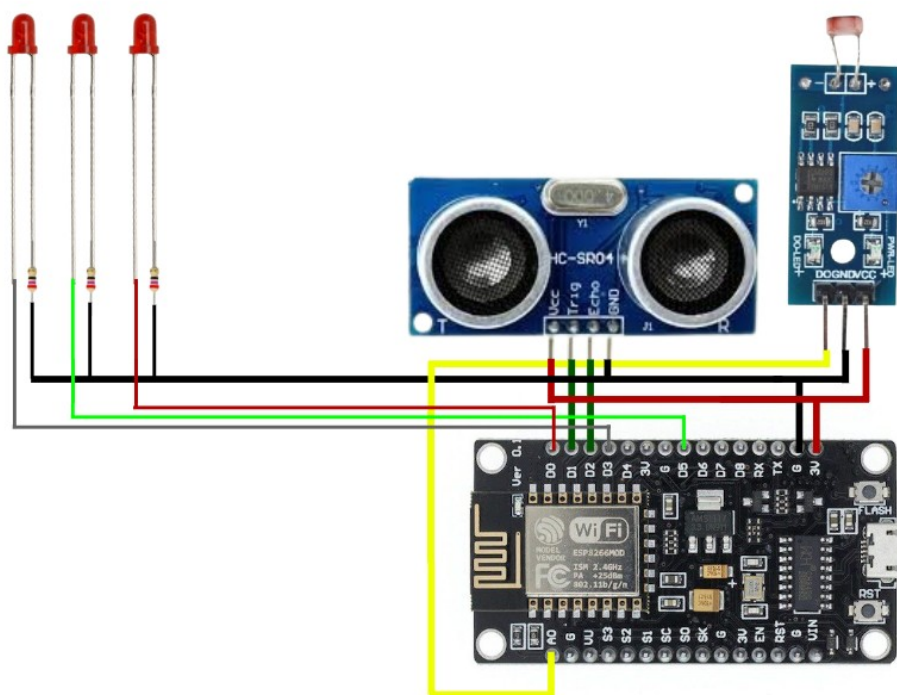


dạng JSON trong payload của POST API ở trên. Đồng thời, Server sẽ trả về số lượng đèn sáng tùy theo cường độ ánh sáng. Nếu có người tiếp cận dựa vào giá trị của cảm biến khoảng cách thì trả về cho người dùng số lượng đèn sáng để cung cấp đủ ánh sáng. Nếu không có người tiếp cận thì tắt các đèn. Khi máy tính nhận được thông điệp thì xuất ra command line hoặc bất kỳ trình dạng nào có thể quan sát được. Các nội dung gửi đi và về đều ở dạng JSON.

Lưu ý: đèn được sáng theo chiều từ trái sang phải và ngược lại.

1. Minh chứng:

Link demo bài 2: [Tại đây](#)



Hình 14. Mô hình nối dây bài 2

```
Received sensor data: {  
  error: true,  
  message: 'This is a message of API',  
  data: { light: 190, distance: 2.753978014 }  
}
```

Hình 15. Json nhận được từ ESP

```
{ num_lights: 1 }
```

Hình 16. Server phân tích dữ liệu và gửi trả



```
num_lights":0
"num_lights":1
"num_lights":0
"num_lights":0
"num_lights":0
"num_lights":0
"num_lights":0
"num_lights":1
"num_lights":1
"num_lights":1
"num_lights":1
```

Hình 17. Json nhận được ở ESP

2. Giải thích:

```
1 #include <ESP8266WiFi.h>
2 #include <ArduinoJson.h>
3
4 const char* ssid = "TheLight";
5 const char* password = "20022003";
6 const char* server_ip = "192.168.120.187";
7 const int server_port = 3000;
8 const unsigned long post_interval = 5000;
9
10 WiFiClient client;
11 const int trig = 4;
12 const int echo = 5;
13 int led[3] = {0,16,14};
14 const int lightSensor = A0;
15 int lastPostTime = 0;
16 int currentLedOn = 0;
```

Hình 18. Khai báo các thông số mặc định

Thư viện: Khai báo bao gồm thư viện wifi của ESP, thư viện ArduinoJson để phân tích Json.

Môi trường: Khai báo các thông tin kết nối wifi, port của server, địa chỉ ip của server.

Biến: Khai báo khoảng thời gian gửi data (**post_interval**), WiFiClient.

- Các chân tín hiệu của cảm biến khoảng cách (Line 11, Line 12)
- Chân led (0,16,14) tương đương D3, D5 và D0.
- Chân A0 của cảm biến ánh sáng (quang trở).
- lastPostTime để kiểm soát thời gian gửi data (không đặt trong loop để tránh hiệu suất ứng dụng bị giảm).
- **currentLedOn** là lưu lại số đèn led đang sáng (khi server gửi chỉ định led sáng, nếu nó vẫn giữ nguyên với state hiện tại thì không thực thi còn khác với currentLedOn này thì nó sẽ thực hiện thay đổi số led sáng).



```
18 void setup() {
19     Serial.begin(9600);
20
21     Serial.println("Đang kết nối tới WiFi...");
22     WiFi.begin(ssid, password);
23     while (WiFi.status() != WL_CONNECTED) {
24         delay(500);
25         Serial.print(".");
26     }
27     Serial.println("Đã kết nối được tới WiFi");
28
29     pinMode(trig, OUTPUT);
30     pinMode(echo, INPUT);
31     pinMode(lightSensor, INPUT);
32     for(int i = 0; i < 3; i++)
33     {
34         pinMode(led[i], OUTPUT);
35     }
36 }
```

Hình 19. Hàm setup

Kết nối đến wifi được khai báo và thực hiện khai báo mặc định tín hiệu các chân led và chân tín hiệu của các sensor.

```
38 void loop() {
39     if (millis() - lastPostTime >= post_interval) {
40         postSensorData();
41         lastPostTime = millis();
42     }
43     delay(400);
44     getLights();
45 }
```

Hình 20. Trong loop

Thực hiện postSensorData trong một khoảng time interval. Sau đó thực hiện nhận lại chỉ thị số đèn sáng được xử lý tại server.

```
48 void postSensorData() {
49     float lightIntensity = analogRead(A0);
50     float distance = getDistance();
51
52     StaticJsonDocument<200> sensorData;
53     StaticJsonDocument<200> jsonData;
54
55     jsonData["error"] = true;
56     jsonData["message"] = "This is a message of API";
57
58     sensorData["light"] = lightIntensity;
59     sensorData["distance"] = distance;
60     jsonData["data"] = sensorData;
61     char jsonBuffer[256];
62     serializeJson(jsonData, jsonBuffer);
63
64     if (client.connect(server_ip, server_port)) {
65         client.println("POST /sensors HTTP/1.1");
66         client.println("Host: " + String(server_ip));
67         client.println("Content-Type: application/json");
68         client.print("Content-Length: ");
69         client.println(strlen(jsonBuffer));
70         client.println();
71         client.println(jsonBuffer);
72     } else {
73         Serial.println("Không thể kết nối tới server");
74         delay(1000);
75     }
76 }
```

Activate Windows

Hình 21. Hàm postSensorData

Thu dữ liệu mức độ ánh sáng vào biến **lightIntensity**, khoảng cách thu từ cảm biến khoảng cách vào biến **distance**. (hàm



getDistance như bên dưới)

```
119 float getDistance()
120 {
121     unsigned long duration;
122     float distance;
123     digitalWrite(trig,0);
124     delayMicroseconds(2);
125     digitalWrite(trig,1);
126     delayMicroseconds(5);
127     digitalWrite(trig,0);
128     duration = pulseIn(echo,HIGH);
129     distance = float(duration/2/29.412);
130     return distance;
131 }
```

Tạo một **sensorData** để lưu json data trong Json có trường là “data”. Biến **jsonData** là Json cuối cùng mà ESP gửi đến server.

Line 55 → Line 60 thực hiện lưu dữ liệu vào jsonData theo format bên dưới,

Lưu ý: Định dạng của HTTP API:

```
{
  "error": true,
  "message": "this is a message of API",
  "data": {
    "temperature": value,
    "light": value,
  }
}
```

Line 64 – Line 71 thực hiện gửi jsonData đến server (các thông số server đã khai báo toàn cục). Thông tin gói tin bao gồm method (ở đây là **POST**), đường dẫn api (ở đây là /sensors), dạng HTTP, Length và payload chính là jsonData.

```
78 void getLights() {
79     client.print(String("GET /lights HTTP/1.1\r\n") +
80         "Host: " + String(server_ip) + "\r\n" +
81         "Connection: close\r\n\r\n");
82     String response = "";
83     while (client.available()) {
84         response = client.readStringUntil('\r');
85     }
86     int startPos = response.indexOf('{');
87     int endPos = response.indexOf('}', startPos);
88     String jsonContent = response.substring(startPos, endPos);
89     Serial.println(jsonContent);
90     DynamicJsonDocument doc(256);
91     DeserializationError error = deserializeJson(doc, jsonContent);
92     int numLights = doc["num_lights"];
93     if(currentLedOn!= numLights)
94     {
95         for(int i =0; i<3; i++)
96         {
97             digitalWrite(led[i], LOW);
98         }
99         for(int i =0; i<numLights; i++)
100         {
101             digitalWrite(led[i], HIGH);
102         }
103     }
```



Hình 22. Ở hàm `getLight`

Đầu tiên khai báo header kết nối đến server (ở đây method là **GET**).

Đọc dữ liệu mà server gửi trả (*Line 82 → Line 85*)

Cắt ra phần json bằng thuật toán cắt string trong '{' và '}'. Hoàn toàn có thể dùng các phương thức lấy payload của thư viện cung cấp cũng được.

Lấy ra giá trị numlights trong trường num_lights của json nhận được (*Line 92*). Sau đó xử lý đèn sáng (Nếu đèn sáng thay đổi với state hiện tại thì nó sẽ chỉnh sửa số đèn sáng còn không thì giữ nguyên). Hoặc hoàn toàn không cần sử dụng **if** (hiệu suất bị giảm hơn 1 tỷ do lúc nào cũng phải chạy liên tục thiết lập led sáng).

```
1  const express = require('express');
2  const bodyParser = require('body-parser');
3
4  const app = express();
5  const PORT = 3000;
6
7  app.use(bodyParser.json());
8
9  let sensorData = { light: 0, distance: 0 };
```

Hình 23. Tại phía server

Sử dụng express để code server. Đầu tiên là khai báo thư viện và biến toàn cục lưu dữ liệu thu được (light và distance). BodyParser giúp hỗ trợ đọc dữ liệu json.

```
11 app.post('/sensors', (req, res) => {
12   sensorData = req.body;
13   console.log('Received sensor data:', sensorData);
14   res.send(numLights);
15 });
```

Hình 24. Method POST tại đường dẫn `/sensors`

Lưu body của request vào biến toàn cục sensorData.

```
17 app.get('/lights', (req, res) => {
18   const numLights = calculateLights(sensorData.data.light, sensorData.data.distance);
19   res.json({ num_lights: numLights });
20   console.log({ num_lights: numLights });
21 });
22 function calculateLights(lightIntensity, distance) {
23   if (distance > 10)
24   {
25     return 0;
26   } else
27   {
28     if (lightIntensity < 100)
29       return 0;
30     if (lightIntensity < 300)
31       return 1;
32     if (lightIntensity < 500)
33       return 2;
34     else return 3
35   }
36 }
```




Hình 25. Method GET tại đường dẫn /lights và hàm xử lý dữ liệu

Kiểm tra nếu distance nhận được nằm ngoài 10cm thì nó sẽ không thực hiện sáng đèn dù cho mức sáng như thế nào đi nữa. Nếu distance trong khoảng 10cm thì nó sẽ thực hiện kiểm tra mức độ ánh sáng để bật số đèn led tương ứng:

- Mức ánh sáng < 100 thì 0 có đèn sáng (dùng quang trở nên value nhận được càng thấp thì độ sáng càng cao).
- Mức ánh sáng < 300 thì sẽ bật 1 đèn sáng.
- Mức ánh sáng < 500 thì sẽ bật 2 đèn sáng.
- Còn lại sẽ bật 3 đèn sáng.

```
37 app.listen(PORT, () => {  
38   console.log(`Server is running on port ${PORT}`);  
39 });  
40
```

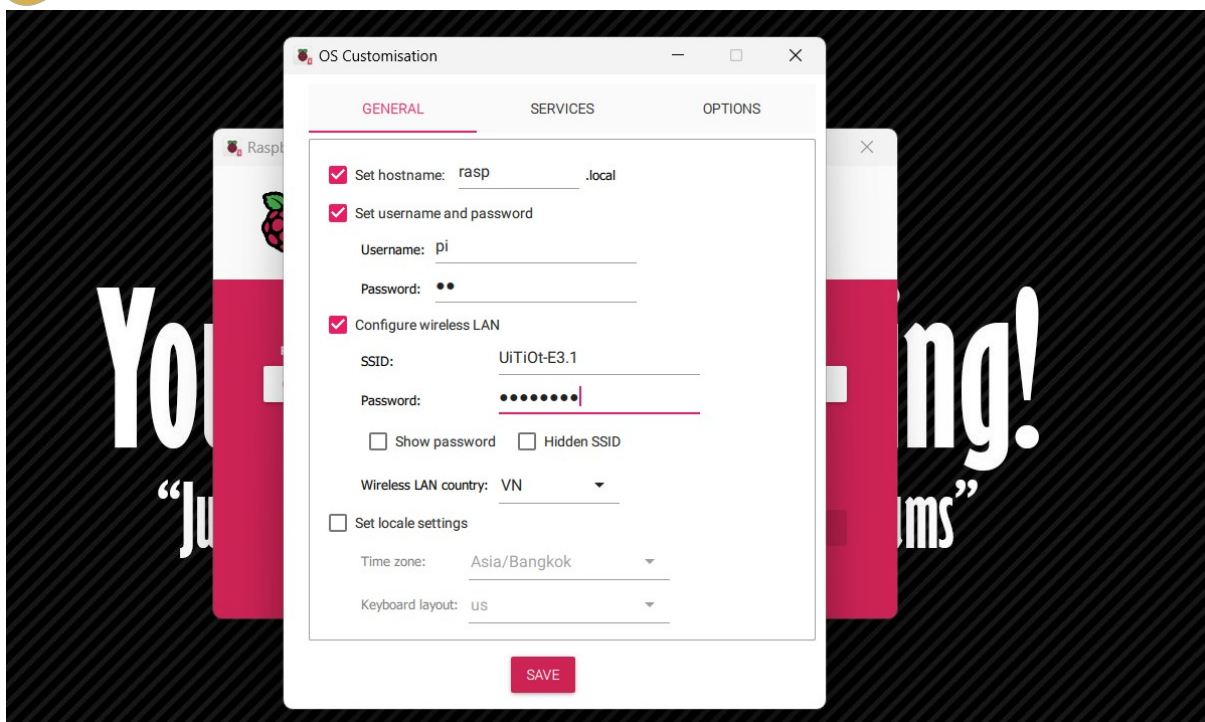
Hình 26. Khởi chạy server với PORT được khai báo trước đó.

Câu hỏi 3. SSH hoặc VNC vào Raspberry Pi được cung cấp sẵn để tiến hành kiểm tra hoạt động của MQTT Broker. Sử dụng một phần mềm MQTT Client bất kỳ (ví dụ như MQTTLens) để tiến hành publish vào topic “cq21/Y/nhomX/check”, với X là số thứ tự nhóm. Trên Client 1 subscribe vào topic trên để nhận thông điệp được gửi từ Client 2.

1. Cài đặt MQTT Broken:

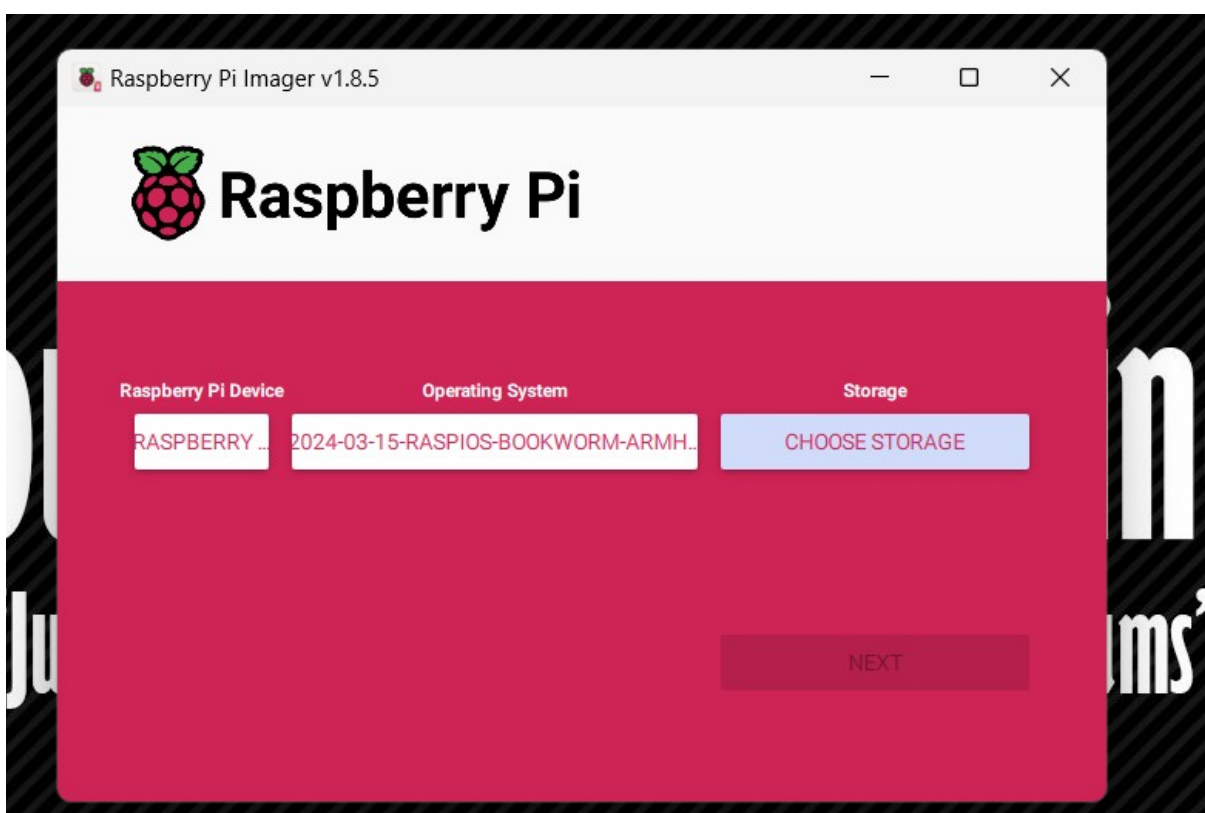
Link minh chứng (demo) bài 3: [Tại đây](#)

Bước 1: Tùy chỉnh các thiết lập cơ bản cho Raspberry pi Image (bao gồm wifi, tài khoản đăng nhập).



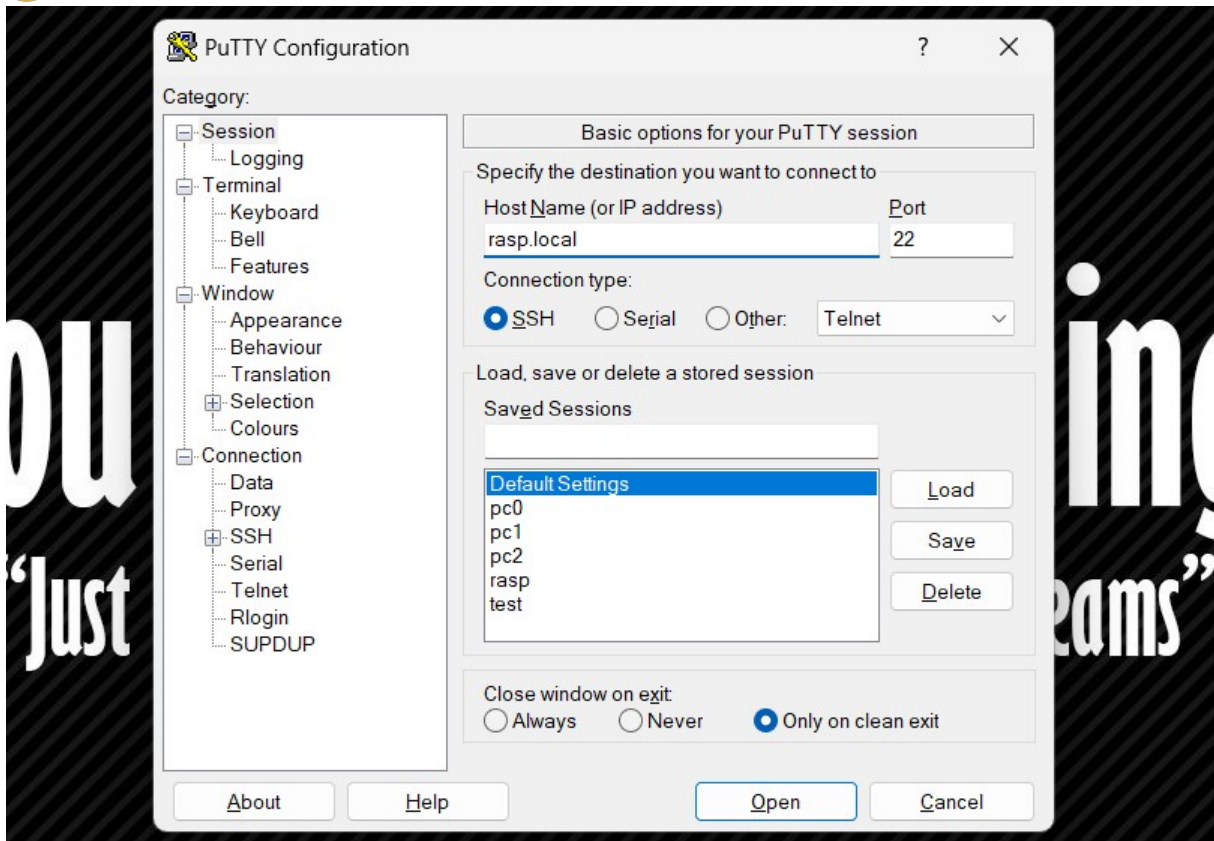
Hình 27. Thiết lập cài đặt cơ bản trước khi nạp code cho thẻ nhớ SSD

Bước 2: Chọn loại rasp pi (ở đây là pi 4), chọn image phù hợp và thẻ nhớ nạp image.



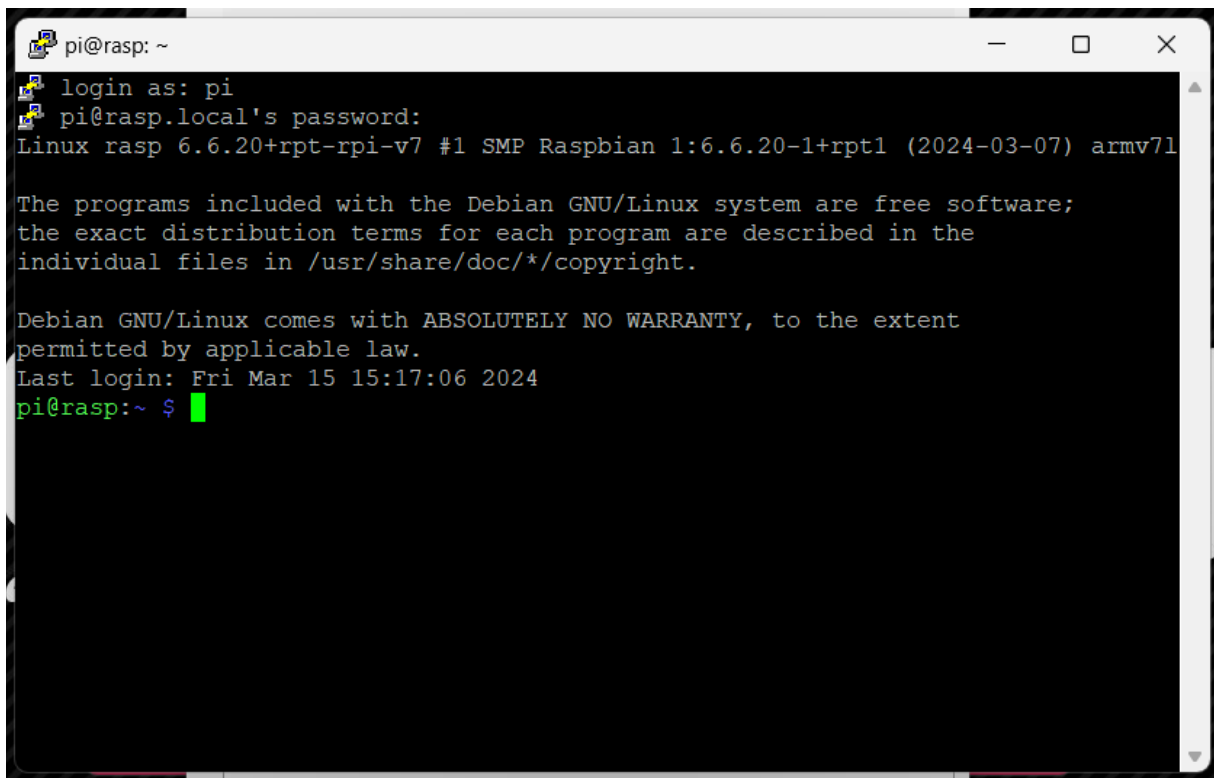
Hình 28. Tùy chỉnh file image và vị trí image

Bước 3 : Tiến hành SSH vào raspberry pi bằng hostname được đặt ở bước 1.



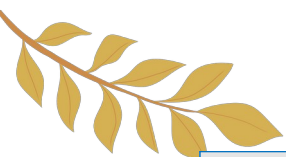
Hình 29. Tiến hành SSH

Bước 4 : Đăng nhập vào raspberry pi bằng tài khoản đã thiết lập ở bước 1.



Hình 30. Đăng nhập

Bước 5 : Tiến hành cập nhật và nâng cấp các gói apt.



```
$ sudo apt update && sudo apt upgrade
```

```
Preparing to unpack .../19-rpi-eeeprom-22.0-1_all.deb ...
Unpacking rpi-eeeprom (22.0-1) over (21.4-1) ...
Selecting previously unselected package rtimuccli.
Preparing to unpack .../20-rtimuccli_0.1_armhf.deb ...
Unpacking rtimuccli (0.1) ...
Selecting previously unselected package python3-sense-emu.
Preparing to unpack .../21-python3-sense-emu_1.2.2_all.deb ...
Unpacking python3-sense-emu (1.2.2) ...
Selecting previously unselected package sense-emu-tools.
Preparing to unpack .../22-sense-emu-tools_1.2.2_all.deb ...
Unpacking sense-emu-tools (1.2.2) ...
Preparing to unpack .../23-scratch3_3.30.9_armhf.deb ...
Unpacking scratch3 (3.30.9) over (3.30.8) ...
Setting up rp-prefapps (0.55) ...
Setting up bsdextrautils (2.38.1-5+deb12u1) ...
Setting up firefox (125.0.1-1+rpt4) ...
Setting up python3-sense-emu (1.2.2) ...
Setting up pi-bluetooth (0.11.20) ...
Setting up python3-pycryptodome (3.11.0+dfsg1-4) ...
Setting up raspi-utils (20240402-1) ...
Setting up gui-pkinst (0.15) ...
Setting up eject (2.38.1-5+deb12u1) ...
Setting up sense-emu-tools (1.2.2) ...
Setting up pipanel (1:1.42) ...
Setting up rkill (2.38.1-5+deb12u1) ...
Setting up piwiz (0.68) ...
Setting up arandr (0.1.11-1+rpt25) ...
Setting up libfdisk1:armhf (2.38.1-5+deb12u1) ...
Setting up rtimuccli (0.1) ...
Setting up mount (2.38.1-5+deb12u1) ...
Setting up libwf-utils0:armhf (0.7.5-1-bp011+1-rpt35) ...
Setting up rpi-eeeprom (22.0-1) ...
Setting up piclone (0.29) ...
Setting up cc-gui (1.79) ...
Setting up fdisk (2.38.1-5+deb12u1) ...
Setting up wayfire (0.7.5-1-bp011+1-rpt35) ...
Setting up raspberrypi-ui-mods (1.20240315) ...
Processing triggers for libgl1:armhf (2.38.1-5+deb12u1) ...
Processing triggers for libc-bin (2.36-9+rpt2+deb12u4) ...
Processing triggers for man-db (2.11.2-2) ...
Processing triggers for shared-mime-info (2.2-1) ...
Processing triggers for mailcap (3.70+nmul) ...
Processing triggers for desktop-file-utils (0.26-1) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
Processing triggers for gnome-menus (3.36.0-1.1) ...
Setting up scratch3 (3.30.9) ...
pi@rasp:~$
```

14

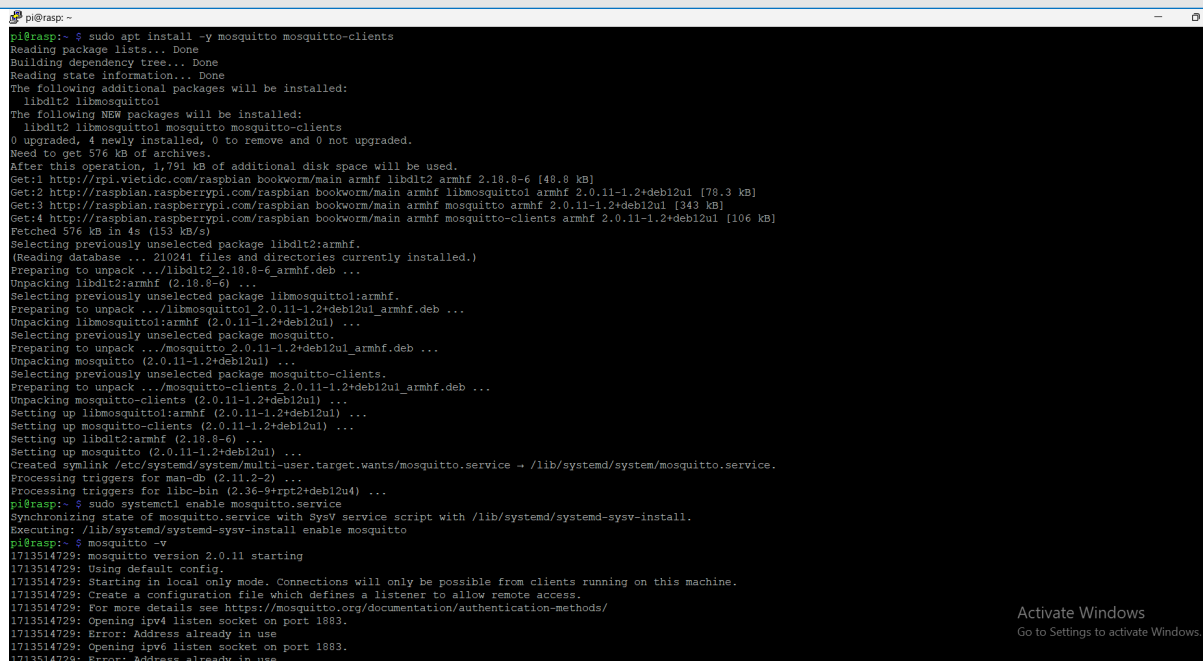
Hình 31. Cập nhật và nâng cấp apt

Bước 6 : Cài đặt mosquito

```
$ sudo apt install -y mosquitto mosquitto-clients
//tải mosquitto mqtt

$ sudo systemctl enable mosquitto.service
//Tự động bật khi Raspbi mở

$ mosquitto -v
//Test xem thành công chưa
```

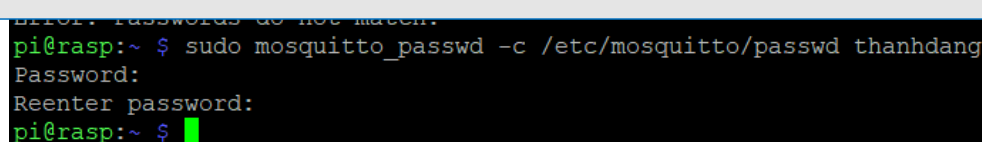


```
pi@rasp:~$ sudo apt install -y mosquitto mosquitto-clients
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libldt2 libmosquitto
The following NEW packages will be installed:
  libldt2 libmosquitto mosquitto mosquitto-clients
0 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 576 kB of archives.
After this operation, 1,791 kB of additional disk space will be used.
Get:1 http://ftp.vietdc.com/raspbian bookworm/main armhf libldt2 armhf 2.18.8-6 [48.8 kB]
Get:2 http://raspbian.raspberrypi.com/raspbian bookworm/main armhf libmosquitto armhf 2.0.11-1.2+deb12u1 [78.3 kB]
Get:3 http://raspbian.raspberrypi.com/raspbian bookworm/main armhf mosquitto armhf 2.0.11-1.2+deb12u1 [343 kB]
Get:4 http://raspbian.raspberrypi.com/raspbian bookworm/main armhf mosquitto-clients armhf 2.0.11-1.2+deb12u1 [106 kB]
Fetched 576 kB in 4s (153 kB/s)
Selecting previously unselected package libldt2:armhf.
(Reading database ... 210241 files and directories currently installed.)
Preparing to unpack .../libldt2_2.18.8-6_armhf.deb ...
Unpacking libldt2:armhf (2.18.8-6) ...
Selecting previously unselected package libmosquitto:armhf.
Preparing to unpack .../libmosquitto_2.0.11-1.2+deb12u1_armhf.deb ...
Unpacking libmosquitto:armhf (2.0.11-1.2+deb12u1) ...
Selecting previously unselected package mosquitto.
Preparing to unpack .../mosquitto_2.0.11-1.2+deb12u1_armhf.deb ...
Unpacking mosquitto (2.0.11-1.2+deb12u1) ...
Selecting previously unselected package mosquitto-clients.
Preparing to unpack .../mosquitto-clients_2.0.11-1.2+deb12u1_armhf.deb ...
Unpacking mosquitto-clients (2.0.11-1.2+deb12u1) ...
Setting up libmosquitto:armhf (2.0.11-1.2+deb12u1) ...
Setting up mosquitto-clients (2.0.11-1.2+deb12u1) ...
Setting up libldt2:armhf (2.18.8-6) ...
Setting up mosquitto (2.0.11-1.2+deb12u1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/mosquitto.service → /lib/systemd/system/mosquitto.service.
Processing triggers for man-db (2.11.2-2) ...
Processing triggers for libc-bin (2.36-9+rpt2+deb12u4) ...
pi@rasp:~$ sudo systemctl enable mosquitto.service
Synchronizing state of mosquitto.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable mosquitto
pi@rasp:~$ mosquitto -v
1713514729: mosquitto version 2.0.11 starting
1713514729: Using default config.
1713514729: Starting in local only mode. Connections will only be possible from clients running on this machine.
1713514729: Create a configuration file which defines a listener to allow remote access.
1713514729: For more details see https://mosquitto.org/documentation/authentication-methods/
1713514729: Opening ipv4 listen socket on port 1883.
1713514729: Error: Address already in use
1713514729: Opening ipv6 listen socket on port 1883.
1713514729: Error: Address already in use
```

Hình 32. Cài đặt mosquitto mqtt

Bước 7 : Thiết lập tài khoản cho mqtt broker (ở đây username là thanhđang, password là 123).

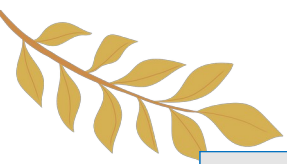
```
sudo mosquitto_passwd -c /etc/mosquitto/passwd thanhđang
mk: 123
```



```
pi@rasp:~$ sudo mosquitto_passwd -c /etc/mosquitto/passwd thanhđang
Password:
Reenter password:
pi@rasp:~$
```

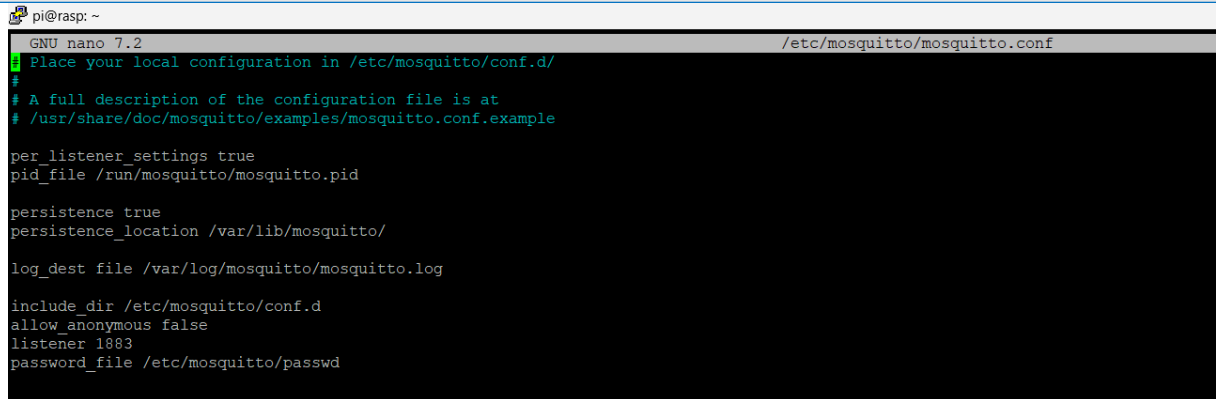
Hình 33. Thiết lập tài khoản cho mqtt broker

Bước 8: Enable tài khoản đó hoạt động trong mqtt broker



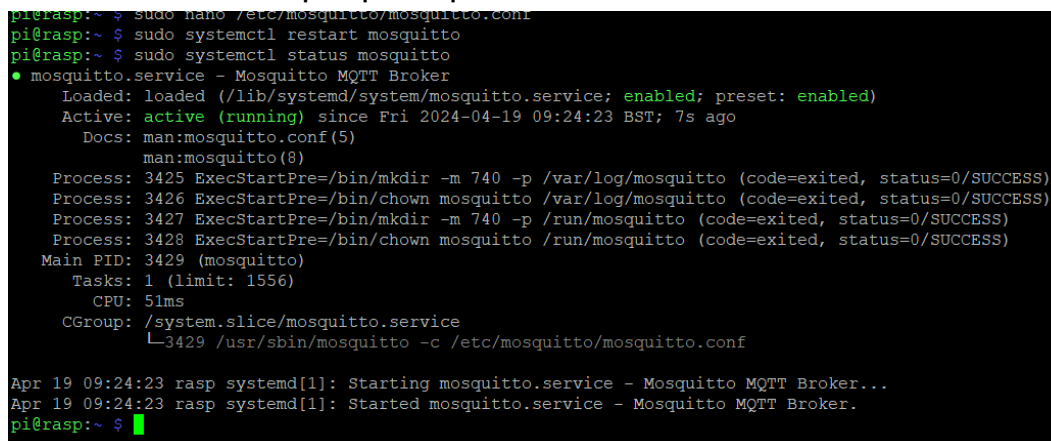
```
per_listener_settings true
//đặt ở hàng đầu tiên

allow_anonymous false
listener 1883
password_file /etc/mosquitto/passwd
//Đặt ở cuối
//Cụ thể'giống hình dưới
```



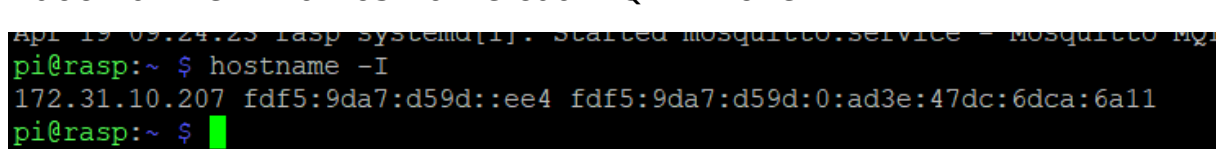
Hình 34. Kích hoạt hiệu lực tài khoản vào MQTT broker

Bước 9: Restart lại dịch vụ



Hình 35. Kết quả sau khi restart lại dịch vụ

Bước 10: Kiểm tra hostname của MQTT Broker



Hình 36. Hostname của MQTT Broker

Như vậy là đã thiết lập thành công MQTT Broker trên Raspberry thật.

Cú pháp **subscribe** 1 topic:

```
mosquitto_sub -h 172.31.10.207 -p 1883 -u thanh dang -P 123 -t "cq21/NT532.021.1/nhom2/check"
```

Trong đó “172.31.10.207” là hostname ở hình 36, “thanh dang”, “123” là username và password đã tạo ở hình 33, còn “cq21/NT532.021.1/nhom2/check” là topic muốn subscribe. Cú pháp để **publish**:

```
mosquitto_pub -h 172.31.10.207 -p 1883 -u thanh dang -P 123 -t "cq21/NT532.021.1/nhom2/check" -m "Hello"
```

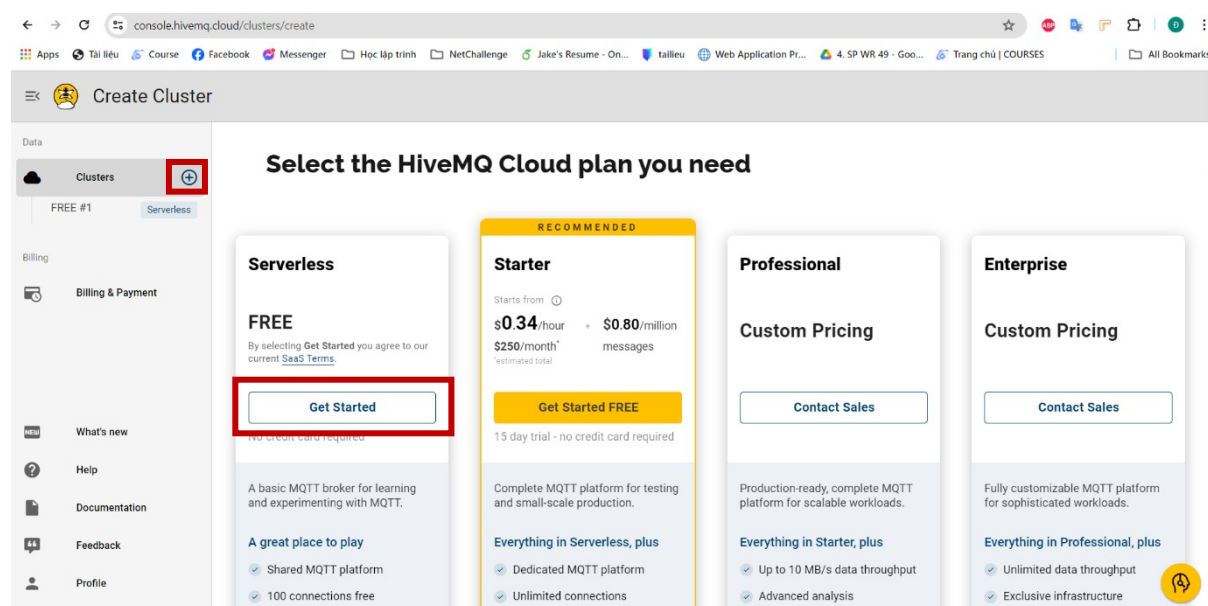
Cú pháp khá tương tự với subscribe. “Hello” là message publish đến Topic.

2. Cài đặt trên Cloud (HiveMQ)

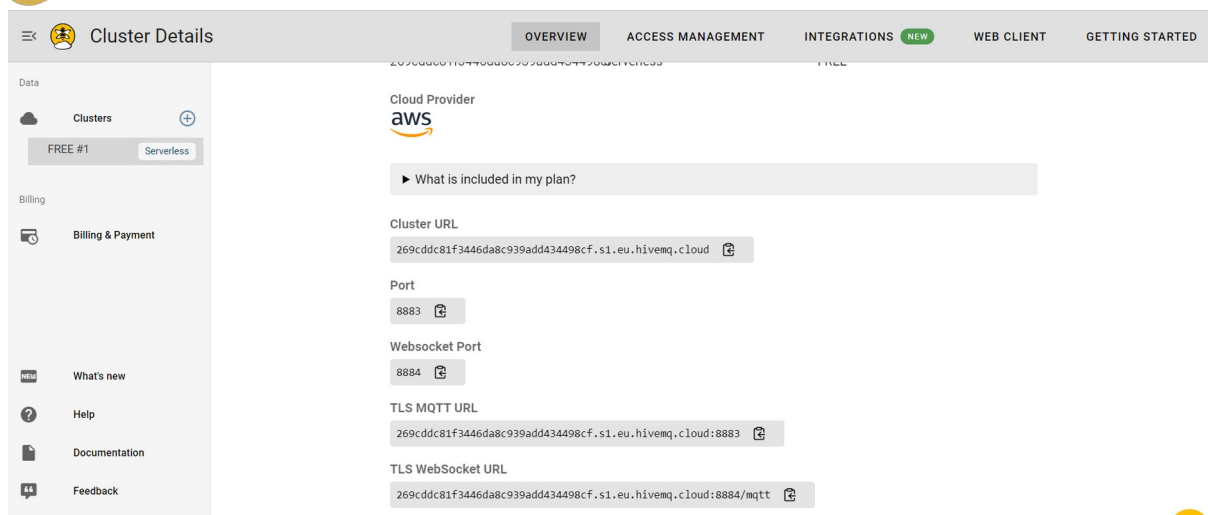
Link minh chứng (demo) bài 3: [Tại đây](#)

Bước 1 : Tạo tài khoản trên HiveMQ (có thể dùng tài khoản google).

Bước 2: Tạo một cluster (Ở đây mức độ test nên em dùng Free)



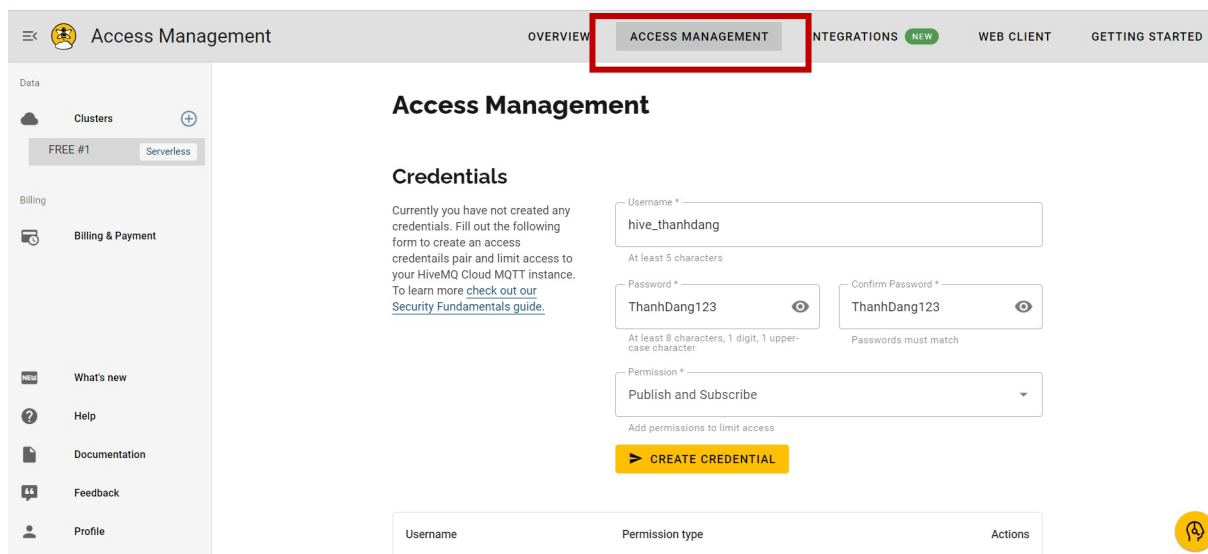
Hình 37. Tạo Cluster



14

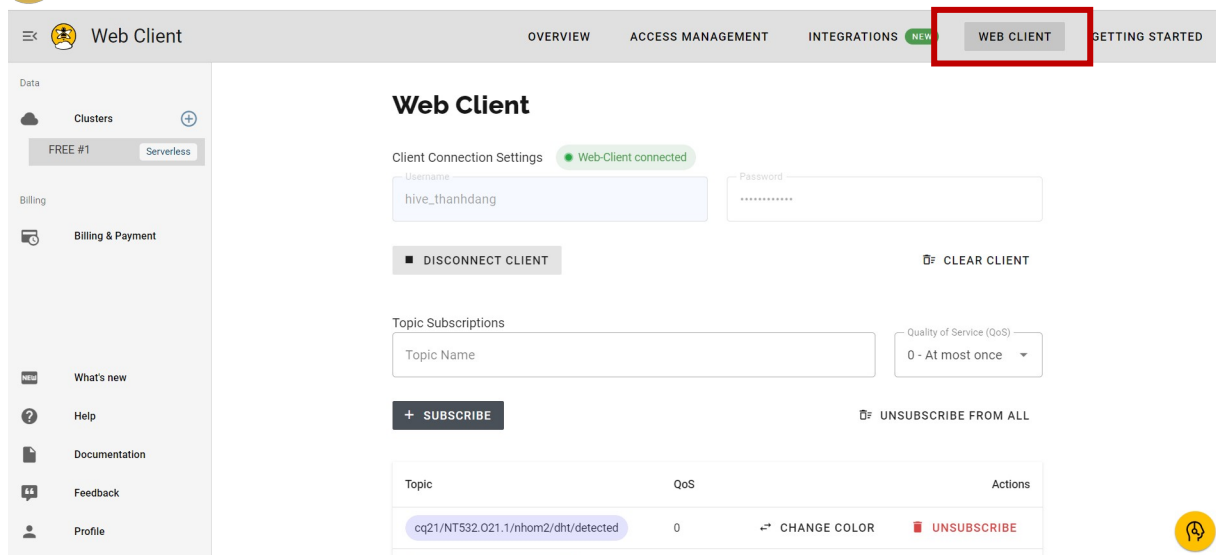
Hình 38. Thông tin của Cluster được tạo(bao gồm url, port để kết nối)

Bước 3: Tạo thông tin tài khoản kết nối đến MQTT Broker. Ở đây username là **hive_thanh dang**, password là **ThanhDang123** trong tab "ACES MANAGEMENT".



Hình 39. Tạo account để kết nối đến mqtt

Bước 4: Kết nối vào MQTT đó thông qua giao diện WEB để có thể subscribe hay publish trong tab "WEB CLIENT"



14

Hình 40. Quản lý subscribe và publish.

```
7 const mqttBrokerUrl = 'mqtt://269cddc81f3446da8c939add434498cf.s1.eu.hivemq.cloud'; // URL của MQTT broker
8 const mqttUsername = 'hive_thanhhdang'; // Tên đăng nhập MQTT
9 const mqttPassword = 'ThanhDang123'; // Mật khẩu MQTT
10 const mqttTopic = 'cq21/NT532.021.1/nhom2/check'; // Topic muốn subscribe vào
```

Hình 41. Khai báo thông tin kết nối MQTT trong express (thực hiện backend)

```
9 // MQTT broker
10 const char *mqtt_broker = "269cddc81f3446da8c939add434498cf.s1.eu.hivemq.cloud";
11 const char *topic = "cq21/NT532.021.1/nhom2/led";
12 const char *mqtt_username = "hive_thanhhdang";
13 const char *mqtt_password = "ThanhDang123";
14 const int mqtt_port = 8883;
```

Hình 42. Khai báo thông tin kết nối trong C (thực hiện nạp cho ESP)

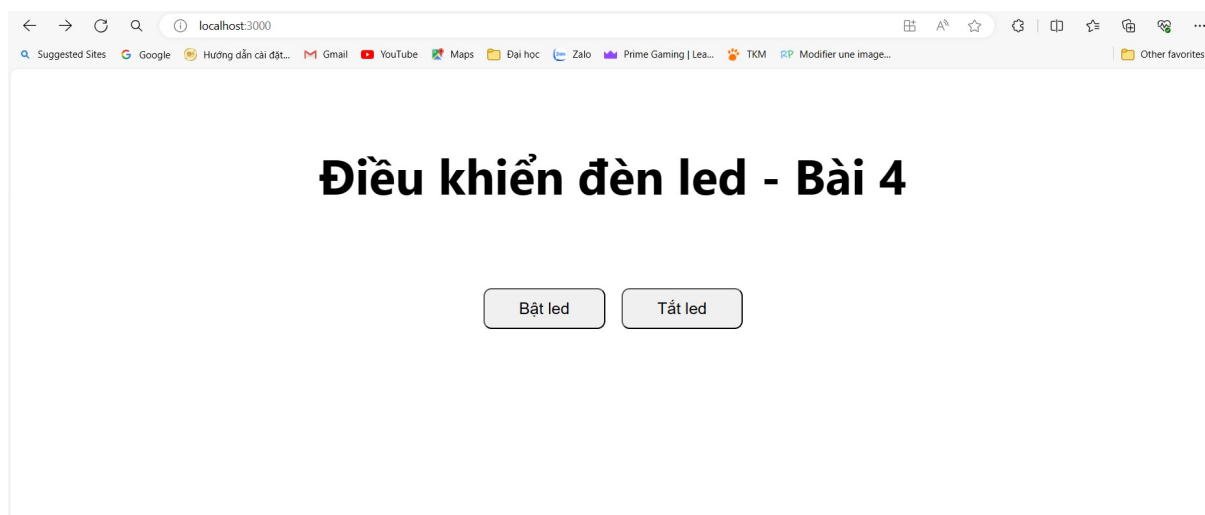
Câu hỏi 4: Xây dựng kịch bản 1 Wemos D1, 1 Raspberry Pi, 1 máy tính cá nhân và 1 đèn LED. Raspberry Pi đóng vai trò là MQTT Broker. Triển khai MQTT Client trên Wemos D1 và máy tính cá nhân. Tại máy tính cá nhân người dùng thông qua topic “cq21/Y/nhomX/led”, tiến hành publish các trạng thái của đèn LED. Tại Wemos D1, subscribe vào topic “cq21/Y/nhomX/led” để lắng nghe trạng thái của đèn, từ đó điều khiển bật hoặc tắt đèn LED.

1. Minh chứng:

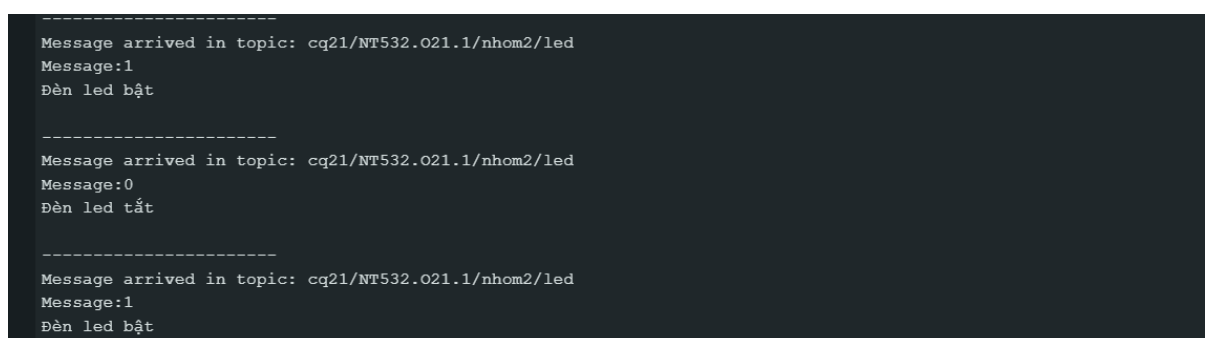
Link minh chứng (demo) bài 4: [Tại đây](#)



Hình 43. Mô hình bài 4



Hình 44. Giao diện điều khiển bật tắt đèn led



Hình 45. Serial trên ESP



```
Received message from topic: cq21/NT532.021.1/nhom2/led
Message: 1
Received message from topic: cq21/NT532.021.1/nhom2/led
Message: 0
Received message from topic: cq21/NT532.021.1/nhom2/led
Message: 1
Received message from topic: cq21/NT532.021.1/nhom2/led
Message: 0
Received message from topic: cq21/NT532.021.1/nhom2/led
Message: 1
Received message from topic: cq21/NT532.021.1/nhom2/led
Message: 0
Received message from topic: cq21/NT532.021.1/nhom2/led
Message: 1
Received message from topic: cq21/NT532.021.1/nhom2/led
Message: 0
Received message from topic: cq21/NT532.021.1/nhom2/led
Message: 1
Received message from topic: cq21/NT532.021.1/nhom2/led
Message: 0
```

Hình 46. Console trên backend viết bằng express

2. Giải thích:

Trong bài này em sẽ sử dụng MQTT trên HiveMQ. Cách tạo giống mục 3.2

```
6  const char *ssid = "TheLight"; // Tên WiFi của bạn
7  const char *password = "20022003"; // Mật khẩu WiFi của bạn
8
9  // MQTT Broker
10 const char *mqtt_broker = "269cddc81f3446da8c939add434498cf.s1.eu.hivemq.cloud";
11 const char *topic = "cq21/NT532.021.1/nhom2/led";
12 const char *mqtt_username = "hive_thanhhdang";
13 const char *mqtt_password = "ThanhDang123";
14 const int mqtt_port = 8883;
15
```

Hình 47. Thông tin kết nối giống

```
16 const int led = 16;
17
18 WiFiClientSecure espClient;
19 PubSubClient client(espClient);
20
```

Hình 48. Khai báo các biến toàn cục cho chương trình

Ở đây em sử dụng chân led 16 tương ứng với D0 ở ESP, **WiFiClientSecure** (do yêu cầu kết nối của HiveMQ là phải được mã hóa. **PubSubClient** là khai báo client ở dạng Publish hoặc subscribe với định dạng kết nối là espClient.



```
21 void setup() {
22   Serial.begin(9600);
23   WiFi.begin(ssid, password);
24   Serial.println("Connecting to WiFi...");
25   while (WiFi.status() != WL_CONNECTED) {
26     delay(500);
27     Serial.print(".");
28   }
29   Serial.println("\nConnected to the WiFi network");
30   espClient.setInsecure();
31   client.setServer(mqtt_broker, mqtt_port);
32   client.setCallback(callback);
33
34   // Connect to MQTT broker
35   while (!client.connected()) {
36     if (client.connect("ESP8266Client", mqtt_username, mqtt_password)) {
37       Serial.println("Connected to MQTT broker");
38       client.publish(topic, "Connected, you can control now. \n'1' or 'on' to Led on \n'0' or 'off' to Led off ");
39       client.subscribe(topic);
40     } else {
41       Serial.print("Failed to connect to MQTT broker, rc =");
42       Serial.print(client.state());
43       Serial.println(" retrying in 5 seconds");
44       delay(5000);
45     }
46   }
47   pinMode(led, OUTPUT);
48
49 }
```

Hình 49. Hàm setup

Thực hiện kết nối đến wifi

Kết nối đến mqtt broker:

- Khi kết nối thành công thì tự động subscribe vào topic đã được khai báo toàn cục trước đó, cụ thể là **cq21/NT532.O21.1/nhom2/led**. (Line 35-Line39)
- Khi kết nối thất bại thì sẽ thông báo lỗi và delay 5s để kết nối lại. (Line 41-44)

Thiết lập pinMode cho led. (Line 47).

```
51 void callback(char *topic, byte *payload, unsigned int length) {
52   Serial.print("Message arrived in topic: ");
53   Serial.println(topic);
54   Serial.print("Message:");
55   for (int i = 0; i < length; i++) {
56     Serial.print((char)payload[i]);
57   }
58   Serial.println();
59   if ((char)payload[0] == '1' || ((char)payload[0] == 'o' && (char)payload[1] == 'n') )
60   {
61     Serial.println("Đèn led bật");
62     digitalWrite(led,HIGH);
63   }
64   else
65   {
66     Serial.println("Đèn led tắt");
67     digitalWrite(led,LOW);
68   }
69   Serial.println();
70   Serial.println("-----");
71 }
72
73 void loop() {
74   client.loop();
75 }
```

Hình 50. Hàm callback và loop



Hàm **callback** mục đích để xử lý dữ liệu nhận được từ MQTT. Nếu dữ liệu nhận được là '1' hoặc 'on' thì sẽ bật led còn nếu không thì sẽ tắt led (Line 59-67).

Hàm **loop** thì chỉ việc thực thi liên tục kết nối MQTT.

```
1  const express = require('express');
2  const mqtt = require('mqtt');
3  const app = express();
4
5  app.use((req, res, next) => {
6    res.setHeader('Access-Control-Allow-Origin', 'http://localhost:3000');
7    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE');
8    res.setHeader('Access-Control-Allow-Headers', 'Content-Type');
9    next(); //Loại bỏ CORS do chạy 2 package trên cùng host
10 });
11 // Thông tin MQTT broker
12 const mqttBrokerUrl = 'mqtts://269cddc81f3446da8c939add434498cf.s1.eu.hivemq.cloud'; // URL của MQTT broker
13 const mqttUsername = 'hive_thanhhdang'; // Tên đăng nhập MQTT
14 const mqttPassword = 'ThanhDang123'; // Mật khẩu MQTT
15 const mqttTopic = 'cq21/NT532.021.1/nhom2/led'; // Topic muốn subscribe vào
16
17 // Kết nối tới MQTT broker sử dụng xác thực username và password đã tạo
18 const mqttClient = mqtt.connect(mqttBrokerUrl, {
19   username: mqttUsername,
20   password: mqttPassword
21 });
22
23 mqttClient.on('connect', () => {
24   console.log('Connected to MQTT broker');
25   // Đăng ký (subscribe) vào topic
26   mqttClient.subscribe(mqttTopic, (err) => {
27     if (err) {
28       console.error('Error subscribing to topic:', err);
29     } else {
30       console.log('Subscribed to topic:', mqttTopic);
31     }
32   });
33 });
34
35 // Xử lý tin nhắn khi nhận được từ topic đã subscribe
36 mqttClient.on('message', (topic, message) => {
37   console.log('Received message from topic:', topic);
38   console.log('Message:', message.toString());
39 });
40
41 // Route để kiểm tra kết nối
42 app.get('/', (req, res) => {
43   res.send('Express server is running');
44 });
45
46 app.post('/control/:id', (req, res) => {
47   const { id } = req.params; // Lấy giá trị của tham số id từ URL
48   mqttClient.publish(mqttTopic, id); // Gửi giá trị của id tới MQTT topic
49   res.send(`${(id==1)?'On':'Off'}`); // Phản hồi thành công cho yêu cầu
50 });
51
52 // Khởi động server
53 app.listen(8000, () => {
54   console.log('Server is running on port 8000');
55 });
56
```

Hình 51. Backend có comment để giải thích code

Chú ý là ở method post (Line 46-Line 50) nếu post có id là 1 thì backend publish một message có nội dung là id (id =1 thì bật mà id =0 thì tắt, có giải thích ở phần ino).

Trên frontend:

```

8  const handleClick = (x) => {
9    fetch(`http://localhost:8000/control/${x}`, {
10     method: 'POST',
11     headers: {
12       'Content-Type': 'application/json'
13     },
14     //body: JSON.stringify({ /* Dữ liệu bạn muốn gửi đi */ })
15   })
16   console.log(x + ": OK")
17 };

```

Hình 52. Sự kiện để post đến api control/id

```

<button className='btn' onClick={()=>handleClick(1)}>Bật led</button>
<button className='btn' onClick={()=>handleClick(0)}>Tắt led</button>

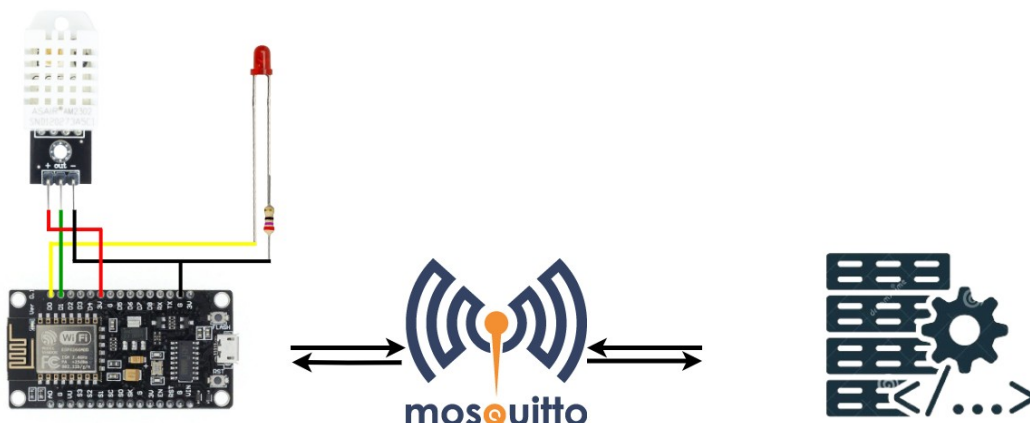
```

Hình 53. Button sẽ điều hướng tới handle bật hoặc tắt

Câu hỏi 5: . Xây dựng kịch bản gồm 1 Wemos D1, 1 Raspberry Pi, 1 máy tính cá nhân, 1 cảm biến nhiệt độ, độ ẩm DHT22 và 1 đèn LED. Raspberry Pi đóng vai trò là MQTT Broker. Tại máy tính cá nhân, sử dụng một ngôn ngữ lập trình bất kỳ để subscribe vào lắng nghe dữ liệu cảm biến từ Wemos D1 thông qua topic “cq21/Y/nhomX/dht/value”. Cho khoảng giá trị bình thường của nhiệt độ là từ 25 oC - 27oC hoặc độ ẩm 40% - 70%, nếu trong quá trình đo đạc phát hiện giá trị bất thường thì máy tính cá nhân publish tín hiệu bất thường vào topic “cq21/Y/nhomX/dht/detected”. Wemos D1 subscribe vào topic “cq21/Y/nhomX/dht/detected”, nếu nhận giá trị phát hiện bất thường thì chớp tắt đèn một cách liên tục để báo hiệu.

1. Minh chứng:

Link minh chứng (demo) bài 5: [Tại đây](#)



BACKEND DEVELOPMENT

Hình 54. Sơ đồ bài 5



```
Message: {"temper":30,"humid":57.5,"state":true}
{ temper: 30, humid: 57.5, state: true }
Received message from topic: cq21/NT532.021.1/nhom2/dht/value
Message: {"temper":29.89999962,"humid":57.59999847,"state":false}
{ temper: 29.89999962, humid: 57.59999847, state: false }
Cảnh báo bất thường!
Received message from topic: cq21/NT532.021.1/nhom2/dht/detected
Message: {"temper": 29.89999962,"humid": 57.59999847,"state":true}
{ temper: 29.89999962, humid: 57.59999847, state: true }
```

Hình 55. Data nhận được trên backend

```
00:51:20.364 -> Message arrived in topic: cq21/NT532.021.1/nhom2/dht/value
00:51:20.398 -> Message: {"temper":29.89999962,"humid":57.59999847,"state":false}
00:51:20.489 ->
00:51:20.534 -> -----
00:51:21.030 -> Message arrived in topic: cq21/NT532.021.1/nhom2/dht/detected
00:51:21.030 -> Message: {"temper": 29.89999962,"humid": 57.59999847,"state":true}
00:51:21.131 ->
00:51:21.131 -> -----
00:51:26.583 -> Message arrived in topic: cq21/NT532.021.1/nhom2/dht/value
00:51:26.969 -> Message: {"temper":29.60000038,"humid":58.5,"state":true}
00:51:26.970 ->
```

Hình 56. Tín hiệu nhận được từ backend ở ESP

2. Giải thích

```
5 #include "DHT.h"
```

Hình 57. Thư viện DHT (được cài đặt từ github trên google)

```
23 DHT dht(DHTPIN, DHTTYPE);
24 const int led = 16;
25 bool state = false;
26 int lastPostTime = 0;
27 const unsigned long post_interval = 5000;
28
```

Hình 58. Khai báo biến toàn cục

```
13 const char *topic = "cq21/NT532.021.1/nhom2/dht/value";
14 const char *topicrec = "cq21/NT532.021.1/nhom2/dht/detected";
```

Hình 59. Tên topic

Trên ESP các thông tin kết nối cơ bản giống với bài 4. Tuy nhiên ở biến toàn cục có khai báo thông tin chân tín hiệu của DHT (đã bao gồm khai báo thư viện DHT). Biến **lastPostTime** và **post_interval** là 2 biến phục vụ cho việc cứ 5s sẽ gửi data đến server 1 lần, biến **state** là biến xem data có đang bất thường không. Và sẽ có 2 topic cho 2 mục đích khác nhau.

```
50 while (!client.connected()) {
51     if (client.connect("ESP8266Client", mqtt_username, mqtt_password)) {
52         Serial.println("Connected to MQTT broker");
53         client.subscribe(topic);
54         client.subscribe(topicrec);
55     } else {
56         Serial.print("Failed to connect to MQTT broker, rc =");
57         Serial.print(client.state());
58         Serial.println(" retrying in 5 seconds");
59         delay(5000);
60     }
61 }
62 dht.begin();
63 pinMode(led, OUTPUT);
```




Hình 60. Xử lý trong hàm setup

Trong setup thì tương tự bài 4, tuy nhiên ở bài 5 chúng ta sẽ subscribe vào 2 topic. **Topic** là nơi gửi message của DHT, còn **topicrec** là nơi nhận message xử lý từ server (Line 53 và Line 54). Bắt đầu chạy dht (Line 62).

14

```
67 void callback(char *topicrec, byte *payload, unsigned int length) {
68     Serial.print("Message arrived in topic: ");
69     Serial.println(topicrec);
70     Serial.print("Message:");
71     for (int i = 0; i < length; i++) {
72         Serial.print((char)payload[i]);
73     }
74     Serial.println();
75     String data = "";
76     for (int i = 0; i < length; i++) {
77         data.concat((char)payload[i]);
78     }
79     DynamicJsonDocument doc(256);
80     DeserializationError error = deserializeJson(doc, data);
81     state = doc["state"];
82
83     Serial.println();
84     Serial.println("-----");
85 }
```

Hình 61. Trong hàm callback

Tiến hành nhận json gửi trả từ server (**topicrec**). Đọc và lấy ra giá trị của trường state để gán vào biến **state**.

```
87 void loop() {
88     client.loop();
89     if (state)
90     {
91         digitalWrite(led,HIGH);
92         delay(500);
93     }
94     digitalWrite(led,LOW);
95     delay(150);
96
97     float h = dht.readHumidity();
98     float t = dht.readTemperature();
99
100     StaticJsonDocument<200> sensorData;
101
102     sensorData["temper"] = t;
103     sensorData["humid"] = h;
104     sensorData["state"] = state;
105     char jsonBuffer[256];
106     serializeJson(sensorData, jsonBuffer);
107
108     if (millis() - lastPostTime >= post_interval) {
109         client.publish(topic, jsonBuffer);
110         lastPostTime = millis();
111     }
112 }
```

Hình 62. Trong hàm loop

Kiểm tra trạng thái state, nếu **state = true** thì tiến hành cảnh báo bằng chop tắt và nếu **false** thì ngược lại (Line 89 – Line 95).

Xử lý 5s gửi message (dạng json) 1 lần (Line 100 – Line 111).