ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI THỰC HÀNH LAB 5, 6
LỚP CE339.N21

Giảng viên hướng dẫn: NGUYỄN DUY XUÂN BÁCH

Sinh viên thực hiện nhóm 11:

Trương Hữu Khang	20520211
Nguyễn Linh Anh Khoa	20520219
Phan Duy Thông	20520789

TP. Hồ Chí Minh 6,2023

A. YÊU CẦU

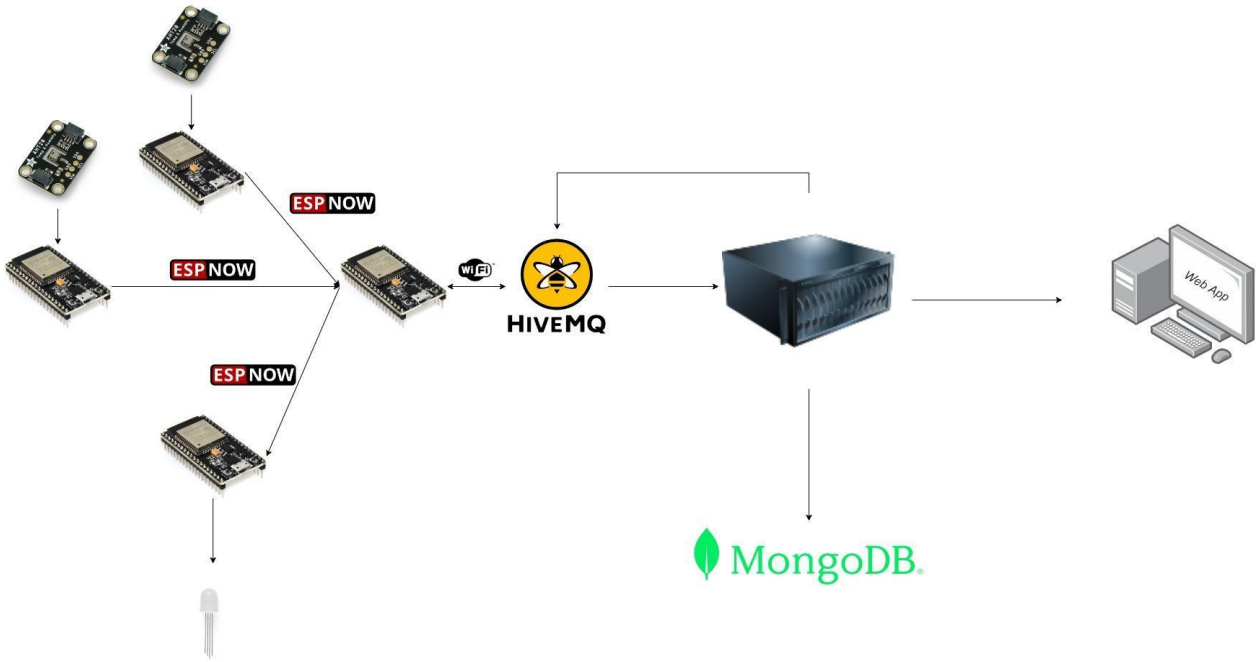
- Tự chọn nội dung/ứng dụng.
- Yêu cầu đảm bảo đủ các yếu tố theo mô hình sau:
- End-device: 3 nodes WiFi.
- Trong 3 nodes phải có đủ: node cảm biến, node điều khiển.
- Các node này giao tiếp với Gateway/Fog Computing.
- Gateway/Fog Computing giao tiếp với IoT Cloud sử dụng giao thức MQTT
- IoT Cloud cần được build với hệ cơ sở dữ liệu NoSQL.
- Tích hợp thêm AI để xử lý dữ liệu.

B. THỰC HIỆN:

ĐỀ TÀI: HỆ THỐNG GIÁM SÁT VÀ ĐIỀU KHIỂN NHIỆT ĐỘ TRONG NHÀ

- Trong bài lab này em sử dụng board ESP32 WiFi.
- Các board ESP32 WiFi node và gateway kết nối với nhau qua ESPNOW
- Broker sử dụng https://www.hivemq.com/
- Cơ sở dữ liệu NoSQL ở MongoDB

1. Mô hình hệ thống



2. Các kết nối

a. Kết nối Node với Gateway thông qua ESPNOW

- Node cảm biến nhiệt độ và độ ẩm: Sử dụng ESP32 kết nối với cảm biến nhiệt độ và độ ẩm AHT20.
- Node điều khiển: Sử dụng ESP32 để điều khiển thiết bị điều khiển nhiệt độ như điều khiển quạt tản nhiệt hoặc bật/tắt hệ thống điều hòa servo.
- GateWay kết nối với các node thông qua ESPNOW.
- Để kết nối hai ESP32 với nhau, đầu tiên chúng ta cần cài đặt thư viện ESP-NOW trên cả hai ESP32. Sau đó, chúng ta sẽ cấu hình một ESP32 làm Master (gateway) và một ESP32 làm Slave (node).
- ESP32 Master sẽ được cấu hình để gửi dữ liệu đến ESP32 Slave. ESP32 Master sẽ tạo ra một cấu trúc dữ liệu (struct) chứa dữ liệu cần gửi, một địa chỉ MAC của ESP32 Slave và một số các thông số khác. Sau đó, ESP32 Master sẽ gửi cấu trúc dữ liệu này đến ESP32 Slave thông qua ESP-NOW.
- ESP32 Slave sẽ được cấu hình để nhận dữ liệu từ ESP32 Master. ESP32 Slave sẽ sử dụng hàm "esp_now_register_recv_cb()" để đăng ký một hàm xử lý dữ liệu (callback function) để xử lý dữ liệu nhận được từ ESP32 Master. Khi ESP32 Slave nhận được dữ liệu từ ESP32 Master, hàm xử lý dữ liệu sẽ được gọi để xử lý dữ liệu đó.
- Sau khi đăng ký hàm xử lý dữ liệu và nhận được dữ liệu từ ESP32 Master, ESP32 Slave có thể sử dụng dữ liệu đó để thực hiện các tác vụ cần thiết.

Cụ thể:

NODE CẢM BIẾN

Hàm pair_task được định nghĩa để gửi thông điệp ESP-NOW cho gateway để ghép đôi.

```
void pair_task(void *pvParameters)
{
    /* setup-code */
    esp_now_msg_t msg;
    msg.type = PAIRING;
    memcpy(msg.data, mac, 6);
    for (;;)
    {
        if (xEventGroupGetBits(pair_status) & PAIRED)
        {
            // do nothing
        }
        else
        {
            espnow_send_msg(msg, broadcast_mac);
        }
        vTaskDelay(pdTICKS_TO_MS(1000));
    }
}
```

Hàm sensor_getdata được định nghĩa để lấy dữ liệu từ cảm biến và gửi nó đến gateway thông qua giao thức ESP-NOW.

```
void sensor_getdata(void *pvParameters)
{
    /* setup-code */
    sensors_event_t humidity, temp;
    esp_now_msg_t msg;
    msg.type = DATA;
    // sprintf((char *)msg.data, "Hello world");
    DynamicJsonDocument doc(64);
    doc["node"] = ID;
    for (;;)
    {
        /* loop */
        xEventGroupWaitBits(pair_status, PAIRED, pdFALSE, pdFALSE, portMAX_DELAY);
        aht.getEvent(&humidity, &temp);
        Serial.printf("Temp: %f Hump: %f\n", temp.temperature, humidity.relative_hum
        doc["temp"] = temp.temperature;
        doc["humidity"] = humidity.relative_humidity;
        doc.as<String>().toCharArray((char *)msg.data, sizeof(msg.data));
        espnow_send_msg(msg, gateway_mac);
        vTaskDelay(pdTICKS_TO_MS(5000));
    }
}
```

Hàm espnow_send_msg được định nghĩa để gửi thông điệp ESP-NOW đến một địa chỉ MAC được cung cấp.

```
void espnow_send_msg(esp_now_msg_t msg, uint8_t *mac)
{
    uint8_t packet_size = sizeof(esp_now_msg_t);
    uint8_t msg_data[packet_size];
    memcpy(&msg_data[0], &msg, sizeof(esp_now_msg_t));

    esp_err_t status = esp_now_send(mac, msg_data, sizeof(esp_now_msg_t));
    if (ESP_OK != status)
    {
        Serial.println("Error sending message");
    }
}
```

Hàm msg_send_cb được định nghĩa để xử lý kết quả khi một thông điệp ESP-NOW được gửi đi.

```
void msg_send_cb(const uint8_t *mac, esp_now_send_status_t sendStatus)
{

    switch (sendStatus)
    {
    case ESP_NOW_SEND_SUCCESS:
        /* code here */
        Serial.println("Send success");
        break;

    case ESP_NOW_SEND_FAIL:
        /* code here */
        Serial.println("Send fail");
        break;

    default:
        break;
    }
}
```

Hàm msg_recv_cb được định nghĩa để xử lý thông điệp ESP-NOW được nhận.

```
void msg_recv_cb(const uint8_t *mac_addr, const uint8_t *data, int len)
{
    /* code here */
    Serial.println("Message received");
    esp_now_msg_t msg;
    memcpy(&msg, data, sizeof(esp_now_msg_t));
    Serial.printf("ID: %d\n", msg.id);
    switch (msg.type)
    {
    case PAIRING:
        break;
    case PAIRING_RESPONSE:
        if (msg.id == 0)
        {
            memcpy(gateway_mac, msg.data, 6);
            esp_now_peer_info_t peer_info;
            peer_info.channel = 0;
            memcpy(peer_info.peer_addr, gateway_mac, 6);
            peer_info.encrypt = false;
            peer_info.ifidx = WIFI_IF_STA;

            if (esp_now_add_peer(&peer_info) != ESP_OK)
            {
                Serial.println("Could not add peer");
            }
            else
            {
                xEventGroupSetBits(pair_status, PAIRED);
            }
        }
        break;
    case DATA:
        break;
```

Hàm espnow_init được định nghĩa để khởi tạo giao thức ESP-NOW.

```
void espnow_init(void)
{
    // Puts ESP in STATION MODE
    Wire.begin(SDA_GPIO_PIN, SCL_GPIO_PIN);
    aht.begin();
    WiFi.mode(WIFI_STA);

    if (esp_now_init() != 0)
    {
        return;
    }
    else
    {
        Serial.println("ESPNow Init Success");
    }
    esp_now_peer_info_t peer_info;
    peer_info.channel = 0;
    peer_info.ifidx = WIFI_IF_STA;
    memcpy(peer_info.peer_addr, broadcast_mac, 6);
    peer_info.encrypt = false;
    if (esp_now_add_peer(&peer_info) != ESP_OK)
    {
        Serial.println("Could not add peer");
    }
    // Set up callback
    if (esp_now_register_recv_cb(msg_recv_cb) != ESP_OK)
    {
        Serial.println("Could not register callback");
    }

    if (esp_now_register_send_cb(msg_send_cb) != ESP_OK)
    {
        Serial.println("Could not register callback");
    }
}
```

NODE ĐIỀU KHIỂN

Hàm messageHandler được định nghĩa để xử lý thông điệp nhận được từ broker MQTT. Trong hàm này, dữ liệu được giải mã từ JSON và lưu trữ vào biến node 1 và node 2 tương ứng với nút cảm biến 1 và 2.

Biến node1 và node 2 được sử dụng để lưu trữ giá trị nhiệt độ của các nút cảm biến tương ứng.

```
float node1 = 0,node2 = 0;
```

Trong hàm setup, ESP32 được kết nối đến WiFi và broker MQTT. Hàm client.connect được gọi để kết nối đến broker MQTT, và hàm client.subscribe được gọi để đăng ký để nhận các thông điệp từ chủ đề "temperature_humidity/data".

```
void setup()
{
    // put your setup code here, to run once:
    Serial.begin(9600);
    pinMode(LED_BUILTIN, OUTPUT);
    WiFi.mode(WIFI_AP_STA);

    WiFi.begin(SSID, PASSWORD);
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(100);
    }
    wifiClient.setInsecure();
    client.setServer(MQTT_SERVER, MQTT_PORT);
    String clientId = "ESP32";
    clientId += String(random(0xffff), HEX);

    if (client.connect(clientId.c_str(), MQTT_USER, MQTT_PASSWORD))
    {
        Serial.println("connected");
    }

    if (!client.connected())
    {
        Serial.println("Connection failed");
        return;
    }
    client.setCallback(messageHandler);
    client.subscribe("temperature_humidity/data");
}
```

Trong hàm loop, hàm client.loop được gọi để giữ kết nối với broker MQTT. Nếu giá trị nhiệt độ của bất kỳ nút cảm biến nào vượt quá 35 độ C, đèn LED sẽ được bật lên. Nếu không, đèn LED sẽ tắt.

```ino
void loop()
{
    // put your main code here, to run repeatedly:
    client.loop();
}

void messageHandler(char *topic, byte *payload, unsigned int length)
{
    Serial.printf("Node 1: %f, Node 2: %f\n", node1, node2);
    DynamicJsonDocument doc(64);
    deserializeJson(doc, payload);
    Serial.println("Message arrived");
    switch (doc["node"].as<int>())
    {
    case 1:
        /* code */
        node1 = doc["temp"].as<float>();
        break;
    case 2:
        /* code */
        node2 = doc["temp"].as<float>();
        break;
    default:
        break;
    }
    if(node1 > 35 || node2 > 35){
        digitalWrite(LED_BUILTIN, HIGH);
    }else{
        digitalWrite(LED_BUILTIN, LOW);
    }
}
```

b. Kết nối GateWay với MQTT broker

Để kết nối GateWay với MQTT broker ta cần thiết lập:

- Khai báo các thư viện cần thiết và các biến, hằng số.
- Khởi tạo kết nối Wi-Fi và kết nối đến MQTT broker.
- Thiết lập callback function để xử lý các thông điệp được nhận từ MQTT broker.

- Trong vòng lặp chính, kiểm tra kết nối đến MQTT broker và thực hiện các bước sau:
    - Nếu không kết nối được đến MQTT broker, thực hiện kết nối lại.
    - Nếu đã đến thời điểm gửi dữ liệu, thực hiện gửi dữ liệu về tình trạng của thiết bị lên MQTT broker.
    - Thực hiện lắng nghe các thông điệp từ MQTT broker và xử lý tương ứng với các yêu cầu điều khiển thiết bị.
- Lặp lại các bước từ 4 đến khi chương trình kết thúc hoặc bị ngắt kết nối đến MQTT broker.

Cụ thể

Đoạn mã bắt đầu bằng việc import các thư viện và khai báo các hằng số và biến cần thiết.

```
#define MQTT_SERVER "410e88d4c6f74067af21eb2712fbc8a4.s2.eu.hivemq.cloud"
#define MQTT_PORT 8883
#define MQTT_USER "esp32"
#define MQTT_PASSWORD "3W645!r7"

const char *SSID = "DESKTOP-OI97DKV 6851";
const char *PASSWORD = "40n9X+82";

WiFiClientSecure wifiClient;
PubSubClient client(wifiClient);

uint8_t broadcast_mac[] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
uint8_t mac[6];
typedef struct esp_now_msg_t {
  uint8_t data[128];
  msg_type type;
  uint8_t id = 0;
} esp_now_msg_t;
```

Hàm espnow_send_msg được định nghĩa để gửi thông điệp ESP-NOW đến một địa chỉ MAC được cung cấp.

```
void espnow_send_msg(esp_now_msg_t msg, uint8_t *mac) {
  uint8_t packet_size = sizeof(esp_now_msg_t);
  uint8_t msg_data[packet_size];
  memcpy(&msg_data[0], &msg, sizeof(esp_now_msg_t));

  esp_err_t status = esp_now_send(mac, msg_data, sizeof(esp_now_msg_t));
  if (ESP_OK != status) {
    Serial.println("Error sending message");
  }
}
```

Hàm msg_send_cb được định nghĩa để xử lý kết quả khi một thông điệp ESP-NOW được gửi đi.

```
void msg_send_cb(const uint8_t *mac, esp_now_send_status_t sendStatus) {

  switch (sendStatus) {
    case ESP_NOW_SEND_SUCCESS:
      /* code here */
      Serial.println("Message sent");
      break;

    case ESP_NOW_SEND_FAIL:
      /* code here */
      Serial.println("Message failed to send");
      break;

    default:
      break;
  }
}
```

Hàm msg_recv_cb được định nghĩa để xử lý thông điệp ESP-NOW được nhận.

```
void msg_recv_cb(const uint8_t *mac_addr, const uint8_t *data, int len) {
  /* code here */
  Serial.println("Message received");
  esp_now_msg_t msg;
  esp_now_msg_t reponse;
  memcpy(&msg, data, sizeof(esp_now_msg_t));
  Serial.printf("MAC: %02x:%02x:%02x:%02x:%02x:%02x\n", mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
  Serial.printf("ID: %d\n", msg.id);
  Serial.printf("Type: %d\n", msg.type);
  // Serial.printf("MAC: %02x:%02x:%02x:%02x:%02x:%02x\n", msg.data[0], msg.data[1], msg.data[2], msg.data[3], msg.data[4], msg.data[5]);
  if (msg.id != 0)  // id gateway = 0
  {
    switch (msg.type) {
      case PAIRING:
        /* code */
        esp_now_peer_info peer_info;
        memcpy(peer_info.peer_addr, mac_addr, 6);
        peer_info.ifidx = WIFI_IF_STA;
        peer_info.channel = 0;
        peer_info.encrypt = false;
        esp_now_add_peer(&peer_info);

        reponse.type = PAIRING_RESPONSE;
        memcpy(reponse.data, mac, 6);

        espnow_send_msg(reponse, peer_info.peer_addr);
        break;
      case PAIRING_RESPONSE:
        /* code */
        break;
      case DATA:
        /* code */
        Serial.printf("Data: %s\n", msg.data);
        client.publish("temperature_humidity/data", (char *)msg.data);
        break;
      default:
        break;
    }
```

Hàm espnow_init được định nghĩa để khởi tạo giao thức ESP-NOW.

```
void espnow_init(void)
{
  // Puts ESP in STATION MODE
  WiFi.mode(WIFI_AP_STA);

  if (esp_now_init() != 0)
  {
    return;
  }

  esp_now_peer_info_t peer_info;
  peer_info.channel = 0;
  peer_info.ifidx = WIFI_IF_STA;
  memcpy(peer_info.peer_addr, broadcast_mac, 6);
  peer_info.encrypt = false;
  if (esp_now_add_peer(&peer_info) != ESP_OK)
  {
    Serial.println("Could not add peer");
  }
  // Set up callback
  if (esp_now_register_recv_cb(msg_recv_cb) != ESP_OK)
  {
    Serial.println("Could not register callback");
  }

  if (esp_now_register_send_cb(msg_send_cb) != ESP_OK)
  {
    Serial.println("Could not register send callback");
  }
}
```

Trong hàm setup, ESP32 được kết nối đến WiFi và broker MQTT. Các hàm esp_now_init và esp_now_register_recv_cb được gọi để khởi tạo giao thức ESP-NOW và đăng ký hàm xử lý thông điệp nhận được.

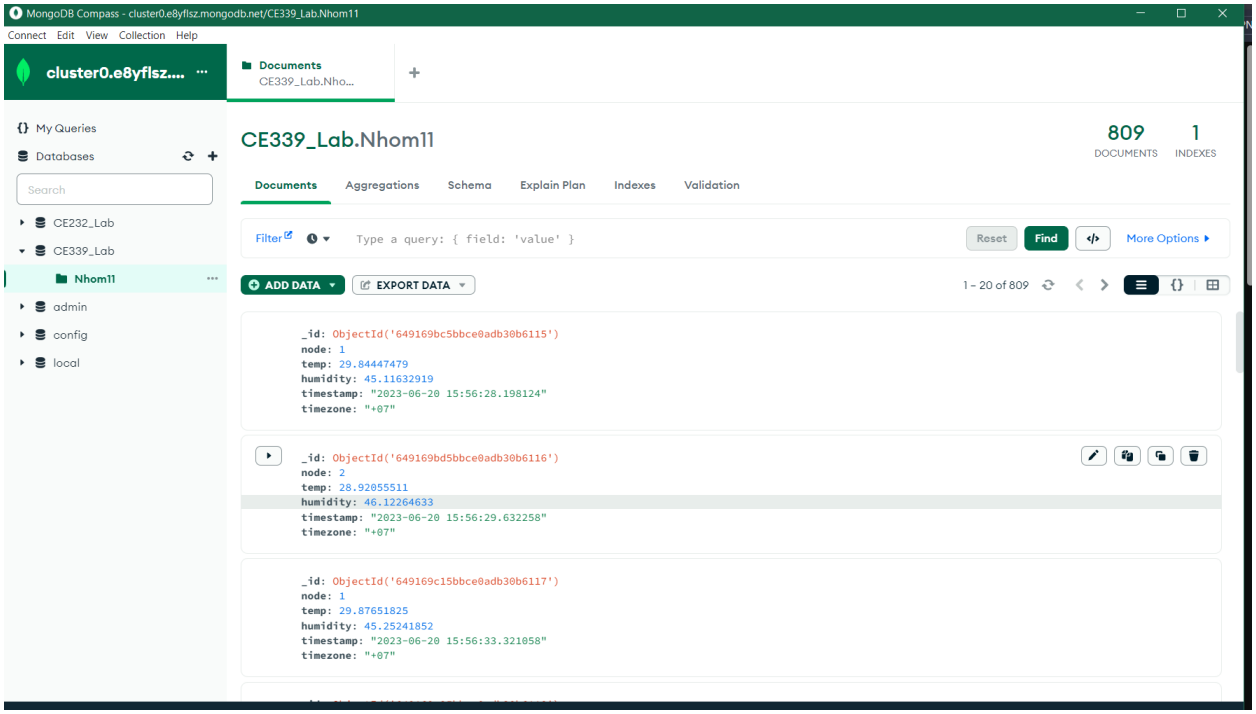
```
void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600);
  esp_read_mac(mac, ESP_MAC_WIFI_STA);

  // Connect to WiFi
  WiFi.mode(WIFI_AP_STA);

  WiFi.begin(SSID, PASSWORD);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(100);
  }
  wifiClient.setInsecure();
  client.setServer(MQTT_SERVER, MQTT_PORT);
  String clientId = "ESP32";
  clientId += String(random(0xffff), HEX);

  if (client.connect(clientId.c_str(), MQTT_USER, MQTT_PASSWORD))
  {
    Serial.println("connected");
  }

  if (!client.connected())
  {
    Serial.println("Connection failed");
    return;
  }

  espnow_init();
}
```

Trong hàm loop, không có gì được thực hiện và chương trình chỉ đợi để nhận thông điệp ESP-NOW hoặc kết nối với broker MQTT.

3. Cơ sở dữ liệu

- MongoDB là một cơ sở dữ liệu phi quan hệ (non-relational database) phổ biến, được phát triển bởi MongoDB Inc. Nó được thiết kế để lưu trữ dữ liệu dưới dạng tài liệu (document), sử dụng định dạng JSON (JavaScript Object Notation). MongoDB sử dụng mô hình dữ liệu linh hoạt và có thể mở rộng để xử lý các tải đồng thời lớn.
- MongoDB có một số đặc điểm chính sau:
    - Cấu trúc dữ liệu tài liệu (Document-oriented): MongoDB lưu trữ dữ liệu dưới dạng tài liệu, cho phép lưu trữ dữ liệu có cấu trúc và không có cấu trúc.
    - Tính mở rộng dễ dàng: MongoDB có thể mở rộng dễ dàng để xử lý các tải đồng thời lớn bằng việc thêm các nút (node) máy tính vào hệ thống.
    - Các truy vấn phức tạp: MongoDB hỗ trợ các truy vấn phức tạp và các phép toán trên dữ liệu, cho phép truy vấn dữ liệu nhanh chóng và hiệu quả.
    - Các tính năng mạnh mẽ: MongoDB hỗ trợ các tính năng như replica set (bản sao dữ liệu), sharding (phân tán dữ liệu), index (chỉ mục), và MapReduce (xử lý dữ liệu phân tán).

MongoDB được sử dụng trong nhiều ứng dụng, bao gồm các ứng dụng web, phân tích dữ liệu, IoT, và các ứng dụng Big Data. Các công ty lớn như Google, Adobe, eBay và Cisco đều sử dụng MongoDB để lưu trữ và xử lý dữ liệu của mình.



4. Server

1. Data processing

- Import các thư viện cần thiết

```
import os
import json
from datetime import datetime
import paho.mqtt.client as paho
from paho import mqtt
from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi
```

Trong đó:

- Thư viện os: Được sử dụng để thiết lập múi giờ.
- Thư viện json: Không được sử dụng trong chương trình này.
- Thư viện datetime: Được sử dụng để lấy thời gian hiện tại và chuyển đổi giữa các múi giờ.

- Thư viện paho-mqtt: Được sử dụng để kết nối và đăng ký theo dõi dữ liệu từ broker MQTT.
- Thư viện pymongo: Được sử dụng để kết nối và lưu trữ dữ liệu vào cơ sở dữ liệu MongoDB.
- Thiết lập môi trường múi giờ thành "Asia/Bangkok".

```
# set time zone to Bangkok/Hanoi/Jakarta
os.environ['TZ'] = 'Asia/Bangkok'
```

- Điều này sẽ đảm bảo rằng thời gian được lấy và lưu trữ theo múi giờ của Bangkok/Hanoi/Jakarta.
- Kết nối tới MongoDB Atlas bằng địa chỉ URI và tạo một đối tượng MongoClient. Chọn phiên bản API ServerApi("1") để sử dụng tính năng mới nhất.

```
# Create a new client and connect to the server
uri = "mongodb+srv://otaros0:CP5OhD7CRWovQghi@cluster0.e8yflsz.mongodb.net/?retryWrites=true&w=majority"
client = MongoClient(uri, server_api=ServerApi("1"))

mydb = client["CE339_Lab"]
mycol = mydb["Nhom11"]
```

- Đoạn mã này thiết lập kết nối đến cơ sở dữ liệu MongoDB Atlas sử dụng địa chỉ URI và tạo một đối tượng MongoClient từ đó. Sau đó, chương trình chọn cơ sở dữ liệu "CE232_Lab" và bảng "Lab5_Nhom7" để lưu trữ dữ liệu.
- Kết nối tới broker MQTT bằng địa chỉ của broker, đăng nhập bằng tên người dùng và mật khẩu, và đăng ký theo dõi chủ đề "temperature_humidity/data".

```
# Connect to MQTT broker with username and password if needed
broker_address = "410e88d4c6f74067af21eb2712fbc8a4.s2.eu.hivemq.cloud"
client_mqtt = paho.Client(client_id="", userdata=None, protocol=paho.MQTTv5)
client_mqtt.on_message = on_message
client_mqtt.tls_set(tls_version=mqtt.client.ssl.PROTOCOL_TLS)
client_mqtt.username_pw_set(username="dataprocess", password="&447s92E")
client_mqtt.connect(broker_address, port=8883)
client_mqtt.subscribe("temperature_humidity/data")
```

- Đoạn mã này kết nối đến broker MQTT sử dụng địa chỉ broker_address và thiết lập đăng nhập bằng tên người dùng và mật khẩu. Sau đó, chương trình đăng ký theo dõi chủ đề "temperature_humidity/data" để nhận dữ liệu.
- Định nghĩa hàm on_message để xử lý dữ liệu nhận được từ chủ đề "temperature_humidity/data".

```
def on_message(client_mqtt, userdata, message):
    print("received message =", str(message.payload.decode("utf-8")))
    doc = json.loads(str(message.payload.decode("utf-8")))
    # data = message.payload.decode()
    # temperature, humidity = data.split(',')
    # temperature = float(temperature)
    # humidity = float(humidity)
    # doc['temperature'] = temperature
    # doc['humidity'] = humidity
    doc['timestamp'] = str(datetime.now())
    doc['timezone'] = str(datetime.now().astimezone().tzinfo)
    mycol.insert_one(doc)
```

- Hàm on_message được gọi mỗi khi client nhận được một thông điệp từ broker MQTT. Hàm này có ba tham số:
    - client_mqtt: đại diện cho client MQTT.
    - userdata: dữ liệu người dùng được truyền vào khi khởi tạo client MQTT, có thể là bất kỳ đối tượng Python nào.

    - message: đại diện cho thông điệp nhận được từ broker MQTT, bao gồm các thông tin như chủ đề (topic) của thông điệp, nội dung của thông điệp và các thuộc tính khác.
- Trong hàm on_message, đầu tiên chúng ta in ra nội dung của thông điệp nhận được bằng cách sử dụng phương thức decode để chuyển đổi nội dung từ định dạng bytes sang định dạng chuỗi.
- Sau đó, chúng ta phân tích nội dung của thông điệp bằng cách sử dụng phương thức json.loads để chuyển đổi chuỗi JSON thành một đối tượng Python.
- Tiếp theo, chúng ta thêm hai thuộc tính mới vào đối tượng Python này:
- 'timestamp': thời điểm nhận được thông điệp, được chuyển đổi sang định dạng chuỗi bằng phương thức str(datetime.now()).
- 'timezone': múi giờ hiện tại, được chuyển đổi sang định dạng chuỗi bằng phương thức str(datetime.now().astimezone().tzinfo).
- Sau đó, chúng ta sử dụng phương thức insert_one của đối tượng mycol để lưu trữ đối tượng Python này vào một bộ sưu tập (collection) trong cơ sở dữ liệu MongoDB. Cụ thể, bộ sưu tập này được đại diện bởi biến mycol.
- Sử dụng client_mqtt.loop_forever() để cho phép chương trình lắng nghe các tin nhắn từ broker MQTT liên tục.

```
client_mqtt.loop_forever()
```

- Cuối cùng, chương trình sử dụng client_mqtt.loop_forever() để cho phép chương trình lắng nghe các tin nhắn từ broker MQTT liên tục.

2. WebServer

app.py

Sử dụng Flask framework của Python để thực hiện ứng dụng web server. Ứng dụng cho phép lấy dữ liệu nhiệt độ và độ ẩm từ MQTT broker và hiển thị trên trang web bằng thư viện Chart.js để vẽ biểu đồ

Đầu tiên, chúng ta import các module cần thiết:

```
from flask import Flask, jsonify, render_template
import paho.mqtt.client as paho
from paho import mqtt
import json
```

Tiếp theo, chúng ta khởi tạo Flask app:

```
app = Flask(__name__)
```

Sau đó, chúng ta kết nối tới MQTT broker:

```
# Kết nối tới MQTT broker
broker_address = "410e88d4c6f74067af21eb2712fbc8a4.s2.eu.hivemq.cloud"
# broker_address = "broker.hivemq.com"
client = paho.Client(client_id="webhost", userdata=None, protocol=paho.MQT
client.tls_set(tls_version=mqtt.client.ssl.PROTOCOL_TLS)
client.on_message = on_message # Thiết lập callback function cho MQTT clie
client.username_pw_set("webserver", "47yX37^4")
client.connect(broker_address, port = 8883)
```

Tiếp theo, chúng ta khởi tạo biến temperature và humidity với giá trị mặc định là 0:

```
# Khởi tạo temperature và humidity

temperature1 = 0
humidity1 = 0

temperature2 = 0
humidity2 = 0
```

Chúng ta cũng định nghĩa hai route cho ứng dụng web server:

```
# API trả về dữ liệu nhiệt độ và độ ẩm mới nhất
@app.route('/api/data')
def get_data():
    # Gửi yêu cầu lấy dữ liệu mới nhất tới MQTT broker
    client.publish("temperature_humidity/get_data", "get_data")
    print(temperature1, humidity1, temperature2, humidity2, sep=' - ')
    # Trả về dữ liệu dưới dạng JSON
    return jsonify({'temperature1': temperature1, 'humidity1': humidity1 , 'temperature2': temperature2, 'humidity2': humidity2})
```

Route index() trả về trang chủ của ứng dụng web, được render từ file HTML index.html.

Route get_data() trả về dữ liệu nhiệt độ và độ ẩm mới nhất. Để lấy dữ liệu này, chúng ta gửi yêu cầu tới MQTT broker thông qua topic temperature_humidity/get_data, và trả về dữ liệu đã lưu trữ trong biến temperature1, humidity1, temperature2, humidity2 .

Chúng ta cũng định nghĩa hàm on_message() để xử lý dữ liệu nhận được từ MQTT broker:

```python
# Hàm xử lý khi nhận được dữ liệu từ MQTT broker
def on_message(client, userdata, message):
    global temperature1, humidity1, temperature2, humidity2
    print("Received message: ", str(message.payload.decode("utf-8")))
    # Lấy dữ liệu từ message payload
    data = json.loads(str(message.payload.decode("utf-8")))
    print(type(data["node"]))
    if data["node"] == 1:
        temperature1 = data["temp"]
        humidity1 = data["humidity"]
    elif data["node"] == 2:
        temperature2 = data["temp"]
        humidity2 = data["humidity"]
```

Hàm này sẽ được gọi mỗi khi nhận được dữ liệu mới từ MQTT broker.

Dữ liệu này được lấy từ message payload và được lưu trữ trong biến temp và humidity.

Cuối cùng, chúng ta start MQTT client và Flask app:

```python
client.subscribe("temperature_humidity/data")

if __name__ == '__main__':
    # Start MQTT client
    client.loop_start()
    # Start Flask app
    app.run()
```

Sau đó, chúng ta subscribe tới topic temperature_humidity/data để nhận dữ liệu từ MQTT broker.

Cuối cùng, chúng ta start cả MQTT client và Flask app bằng cách sử dụng client.loop_start() và app.run().

Trên trang web, chúng ta sử dụng thư viện Chart.js để hiển thị dữ liệu nhiệt độ và độ ẩm dưới dạng biểu đồ đường. Biểu đồ này sẽ được cập nhật mỗi 10 giây bằng cách gọi API /api/data của Flask app.

Index.html

Phần <head> của tài liệu HTML này chứa đường dẫn tới các thư viện JavaScript và CSS được sử dụng trong trang web này. Cụ thể là thư viện jQuery và thư viện Chart.js.

```
<head>
    <meta charset="utf-8" />
    <title>Display Data</title>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <style>
        @import url('https://fonts.googleapis.com/css2?family=Roboto&display=swap');
        @import url('https://fonts.cdnfonts.com/css/sf-pro-display');
```

Phần <body> bao gồm các thẻ HTML để hiển thị tiêu đề, biểu đồ và danh sách các sinh viên thực hiện bài lab.

```
body {
    font-family: 'Roboto', sans-serif;
    background-color: #d2e9e9;
    box-sizing: border-box;
    margin: 20px auto;
}
```

```
<div class="content">
    <h2>Giảng viên hướng dẫn: Nguyễn Duy Xuân Bách</h2>
    <h3>Sinh viên thực hiện: nhóm 7</h3>

    <ul>
        <li>Trương Hữu Khang</li>
        <li>Nguyễn Linh Anh Khoa</li>
        <li>Phan Duy Thông</li>
    </ul>
</div>
```

Trong phần , tiêu đề chính của trang web được đặt trong một thẻ <h1> và được thiết lập kiểu chữ bằng CSS.

```
<div class="title">
    <h1>Bài thực hành Lab 5</h1>
</div>
```

```
.title {
    font-family: 'SF Pro Display', sans-serif;
    font-size: 20px;
    background: #4a30cf;
    background: linear-gradient(to top, #4a30cf 0%, #cf1d23 100%);
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
    text-align: center;
    font-weight: bolder;
    text-transform: uppercase;
}
```

Phần biểu đồ được đặt trong một thẻ và sử dụng thẻ <canvas> để vẽ biểu đồ. Biểu đồ sử dụng thư viện Chart.js để vẽ và được thiết lập kiểu dữ liệu để hiển thị là biểu đồ dạng đường.

```html
<div class="chart">
    <canvas id="myChart"></canvas>
</div>
```

```html
<script>
    var ctx = document.getElementById('myChart').getContext('2d')
    var myChart = new Chart(ctx, {
        type: 'line',
        data: {
            labels: [],
            datasets: [
                {
                    label: 'Temperature',
                    data: [],
                    backgroundColor: 'rgba(255, 99, 132, 0.2)',
                    borderColor: 'rgba(255, 99, 132, 1)',
                    borderWidth: 1,
                },
                {
                    label: 'Humidity',
                    data: [],
                    backgroundColor: 'rgba(54, 162, 235, 0.2)',
                    borderColor: 'rgba(54, 162, 235, 1)',
                    borderWidth: 1,
                },
            ],
        },
        options: {
            scales: {
                yAxes: [
                    {
                        ticks: {
                            beginAtZero: true,
                        },
                    },
                ],
            },
        },
    })
```

Phần chứa thông tin về giảng viên và danh sách các sinh viên thực hiện bài lab.

```html
<div class="content">
    <h2>Giảng viên hướng dẫn: Nguyễn Duy Xuân Bách</h2>
    <h3>Sinh viên thực hiện: nhóm 7</h3>

    <ul>
        <li>Trương Hữu Khang</li>
        <li>Nguyễn Linh Anh Khoa</li>
        <li>Phan Duy Thông</li>
    </ul>
</div>
```

```css
.content {
    background-color: white;
    text-align: left;
    margin: 0 20px;
    border-radius: 10px;
    padding: 20px;
}

ul {
    list-style-type: none;
}
```

Phần JavaScript được sử dụng để gọi API và cập nhật dữ liệu lên biểu đồ. Đầu tiên, một đối tượng Chart được tạo ra để định nghĩa biểu đồ và được thiết lập với hai đối tượng Dataset để hiển thị dữ liệu nhiệt độ và độ ẩm.

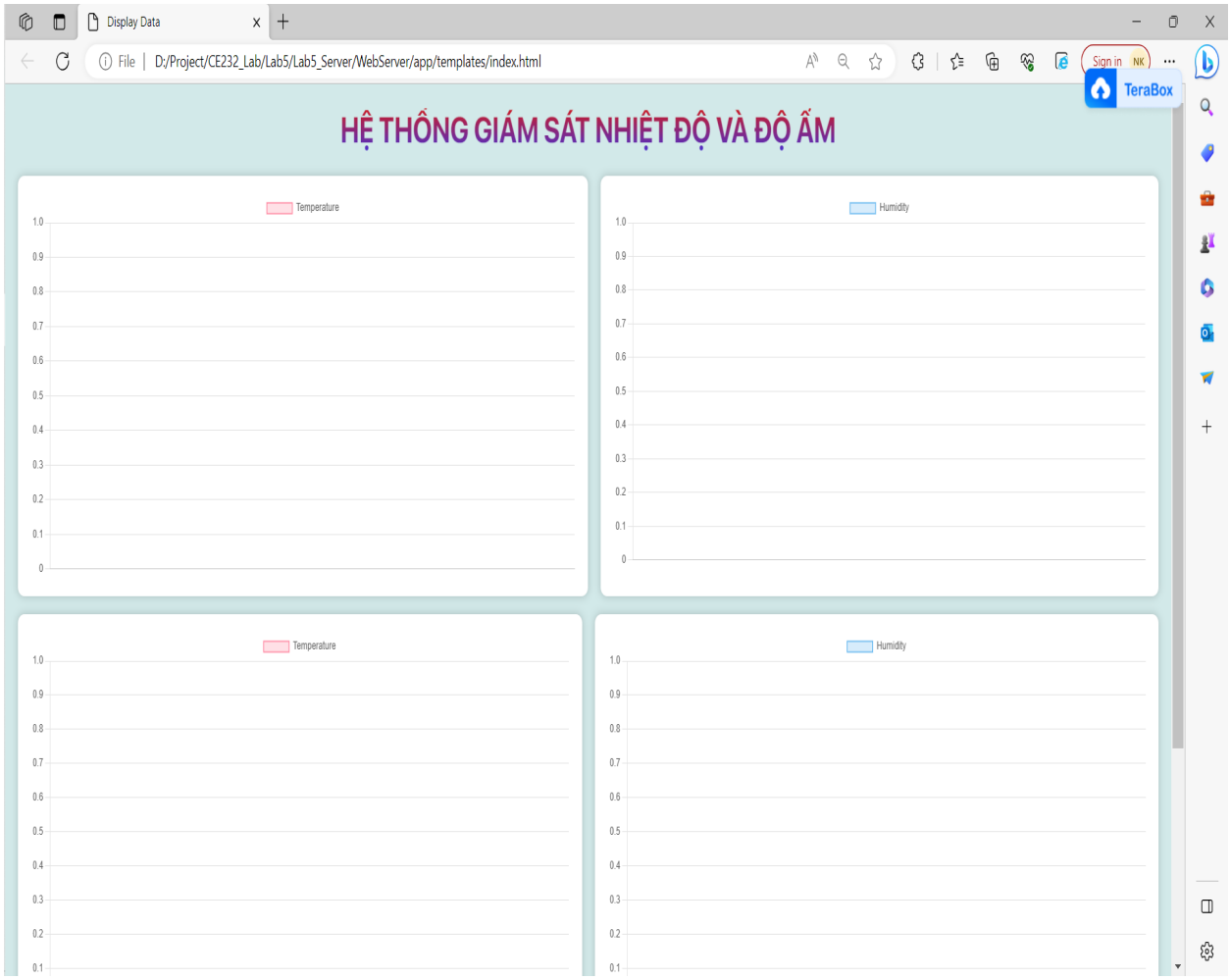
```
// Cập nhật dữ liệu mỗi 10 giây
setInterval(function () {
  $.getJSON("/api/data", function (data) {
    // Cập nhật dữ liệu của biểu đồ1
    tempChart1.data.labels.push(new Date().toLocaleTimeString());
    tempChart1.data.datasets[0].data.push(data.temperature1);
    if (tempChart1.data.labels.length > 20) {
      // Xóa điểm dữ liệu cũ nếu số lượng điểm vượt quá 20
      tempChart1.data.labels.shift();
      tempChart1.data.datasets[0].data.shift();
    }
    tempChart1.update();

    humChart1.data.labels.push(new Date().toLocaleTimeString());
    humChart1.data.datasets[0].data.push(data.humidity1);
    if (humChart1.data.labels.length > 20) {
      humChart1.data.labels.shift();
      humChart1.data.datasets[0].data.shift();
    }
    humChart1.update();
```

Sau đó, hàm $(document).ready() được sử dụng để đảm bảo rằng đoạn mã JavaScript chỉ được thực thi sau khi trang web đã được tải hoàn toàn. Hàm $.getJSON() được sử dụng để gọi API để lấy dữ liệu nhiệt độ và độ ẩm mới nhất từ máy chủ và sau đó cập nhật dữ liệu lên biểu đồ bằng cách thêm thẻ thời gian mới nhất và giá trị nhiệt độ và độ ẩm tương ứng vào các mảng dữ liệu của các đối tượng Dataset được định nghĩa trong biểu đồ.

Cuối cùng, hàm setInterval() được sử dụng để cập nhật dữ liệu và vẽ lại biểu đồ mỗi 10 giây bằng cách gọi lại hàm lấy dữ liệu và cập nhật biểu đồ.

5. Web Server

Để dễ dàng theo dõi hoạt động của hệ thống ta sẽ quan sát trên web server được xây dựng bằng framework Flask ở trên



6. Hoạt động của hệ thống

Mô hình hoạt động của hệ thống trên có thể diễn ra theo các bước sau:

- Các node cảm biến thu thập dữ liệu từ môi trường và gửi dữ liệu đến node điều khiển thông qua kết nối WiFi.
- Node điều khiển tiếp nhận và xử lý dữ liệu từ các node cảm biến, sau đó gửi dữ liệu về Gateway/Fog Computing thông qua kết nối WiFi.
- Gateway/Fog Computing nhận dữ liệu từ các node điều khiển và gửi dữ liệu đến IoT Cloud thông qua giao thức MQTT.

- IoT Cloud lưu trữ dữ liệu dưới dạng tài liệu (document) sử dụng hệ cơ sở dữ liệu NoSQL. Dữ liệu này được lưu trữ và sẵn sàng được truy xuất bởi các ứng dụng và các công cụ phân tích dữ liệu.
- Webserver dùng để hiển thị và giám sát

Video demo: https://youtu.be/POULOGFyrfY

Source code: https://github.com/otaros/CE339_Lab/tree/main/Lab5-6