ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI THỰC HÀNH LAB5

LỚP CE232.N21.1

Giảng viên hướng dẫn: TRẦN HOÀNG LỘC

Sinh viên thực hiện nhóm 7:

Trương Hữu Khang	20520211
Nguyễn Linh Anh Khoa	20520219
Hà Vĩnh Kiện	20520597
Phan Duy Thông	20520789

TP. Hồ Chí Minh 6,2023

A. PHẦN ĐỀ:

Thiết kế một hệ thống IoT thực hiện các chức năng sau:

- NODE ESP32:

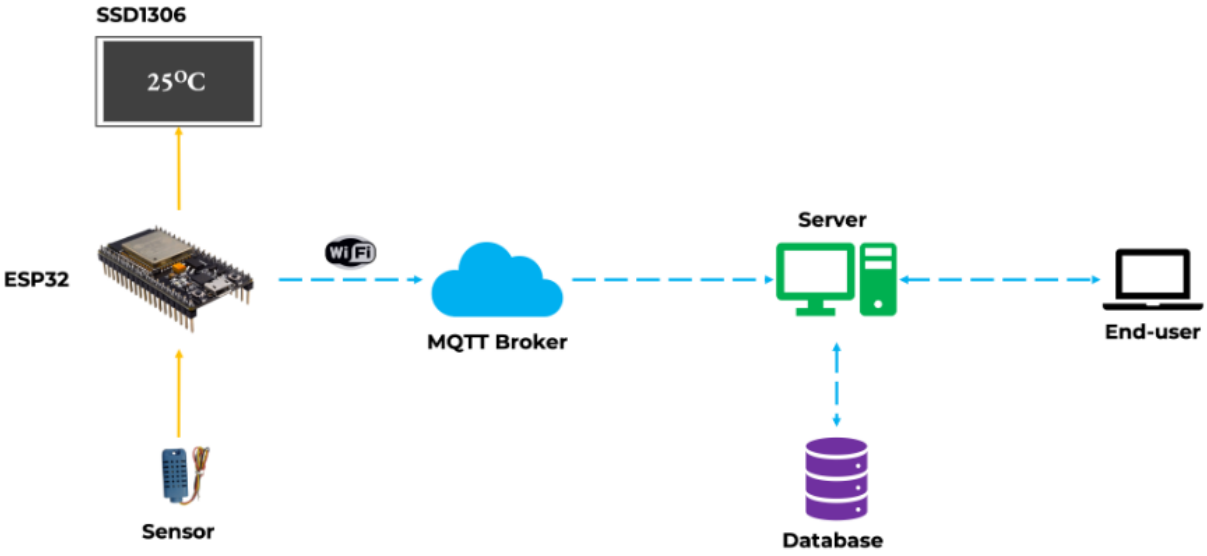
- Lấy các dữ liệu nhiệt độ, độ ẩm, lượng mưa từ các cảm biến (1.5đ)
- Hiển thị dữ liệu lên LCD SSD1306 (1đ)
- Gửi về server với chu kỳ 10s/lần (1.5đ)

- SERVER:

- Nhận dữ liệu (0.5đ)
- Lưu dữ liệu vào database
    - Lưu vào file TXT (1đ)
    - Lưu vào file Excel/CSV/Json (1.5đ)
    - Sử dụng cơ sở dữ liệu như SQL/MySQL/MongoDB/Redis/...(2đ)
- Xây dựng web-app cho phép truy cập local (2đ)

- END-USER:

- Hiển thị dữ liệu để người dùng xem
    - Hiển thị dạng số đơn giản
        - Người dùng tự bấm refresh để xem dữ liệu mới (1đ)
        - Web tự refresh khi có dữ liệu mới (1.5đ)
        - Web tự cập nhật dữ liệu mà không cần refresh (2đ)
    - Hiển thị dạng đồ thị
        - Người dùng tự bấm refresh để xem dữ liệu mới (1.5đ)
        - Web tự refresh khi có dữ liệu mới (2đ)
        - Web tự cập nhật dữ liệu mà không cần refresh (2.5đ)
- Thiết kế website đẹp (Cộng 1đ Bonus)



B. PHẦN BÁO CÁO

1. Node:

Trong phần node sẽ thực hiện hai công đoạn chính là thực hiện kết nối với WiFi và đưa dữ liệu lên MQTT broker sau đó kết nối với các ngoại vi là cảm biến AHT20 và màn hình OLED SSD1306.

Giải thích chi tiết về các giá trị và hàm được sử dụng trong đoạn code này:

```
// CONFIG WIFI AP CONNECT TO MQTT
#define EXAMPLE_ESP_WIFI_SSID "DESKTOP-OI97DKV 6851"
#define EXAMPLE_ESP_WIFI_PASS "40n9X+82"
#define EXAMPLE_ESP_MAXIMUM_RETRY 5
#define CONFIG_BROKER_URL "mqtts://410e88d4c6f74067af21eb2712fbc8a4.s2.eu.hivemq.cloud:8883"

#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA2_PSK

// CONFIG I2C OLED
#define I2C_MASTER_SCL_IO 26      /*!< gpio number for I2C master clock */
#define I2C_MASTER_SDA_IO 25      /*!< gpio number for I2C master data  */
#define I2C_MASTER_NUM I2C_NUM_0  /*!< I2C port number for master dev */
#define I2C_MASTER_FREQ_HZ 100000 /*!< I2C master clock frequency */
#define AHT_ADDR_GND 0x38         /*!< AHT20 I2C address with ADDR pin connected to GND */

/* FreeRTOS event group to signal when we are connected*/
static EventGroupHandle_t s_wifi_event_group;
static EventGroupHandle_t s_mqtt_event_group;
```

Cấu hình tên và mật khẩu của mạng WiFi mà ESP32 sẽ kết nối vào. Nó cũng định nghĩa số lần kết nối tối đa mà ESP32 sẽ thử trước khi báo lỗi.

Đoạn code tiếp theo định nghĩa địa chỉ của broker MQTT mà ESP32 sẽ kết nối đến.

Tiếp theo là cấu hình cho màn hình OLED được kết nối thông qua giao tiếp I2C.

Tạo một event group để theo dõi kết nối WiFi và MQTT.

Định nghĩa một số handle và biến cho các tác vụ và sự kiện.

Định nghĩa một hàm để ghi log lỗi nếu có lỗi xảy ra.

```
static void log_error_if_nonzero(const char *TAG, const char *message, int error_code)···
```

Định nghĩa một hàm xử lý sự kiện MQTT để xử lý các sự kiện từ broker MQTT, bao gồm kết nối, ngắt kết nối, đăng ký đăng ký, hủy đăng ký, đăng bài, và nhận dữ liệu.

```
static void mqtt_event_handler(void *handler_args, esp_event_base_t base, int32_t event_id, void *event_data)···
```

Đoạn code định nghĩa một số bit cho các event group để theo dõi trạng thái của kết nối WiFi và MQTT.

```
static void mqtt_app_start(void)
{
    esp_mqtt_client_config_t mqtt_cfg = {
        .broker.address.uri = CONFIG_BROKER_URL,
        // .credentials.username = "HbI4VBzXtoM1fq7UT9KjSQeyTRPMreZXfyRYZeLgsn7vE2dl0OuIsx49cIMtMBgP",
        .credentials.username = "esp32",
        .credentials.authentication.password = "3W645!r7",
        .credentials.client_id = "esp32_node",
    };

    client = esp_mqtt_client_init(&mqtt_cfg);
    /* The last argument may be used to pass data to the event handler, in this example mqtt_event_handler */
    esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler, NULL);
    esp_mqtt_client_start(client);
}
```

Hàm mqtt_app_start(): Đây là hàm để khởi tạo kết nối đến broker MQTT. Nó tạo một cấu trúc esp_mqtt_client_config_t để định cấu hình cho kết nối MQTT. Trong đó:

CONFIG_BROKER_URL là địa chỉ của broker MQTT, đã được định nghĩa trong tệp cấu hình.

.credentials.username là tên người dùng được sử dụng để xác thực kết nối MQTT.

.credentials.authentication.password là mật khẩu được sử dụng để xác thực kết nối MQTT.

.credentials.client_id là ID của thiết bị được kết nối vào broker MQTT.

Sau đó, hàm này sử dụng esp_mqtt_client_init() để khởi tạo client MQTT và esp_mqtt_client_start() để bắt đầu kết nối với broker MQTT.

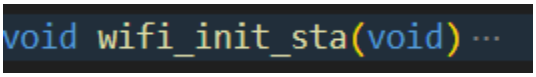
```
static void WiFi_event_handler(void *arg, esp_event_base_t event_base,
                               int32_t event_id, void *event_data)
```

Hàm WiFi_event_handler(): Đây là hàm xử lý các sự kiện liên quan đến kết nối WiFi của thiết bị ESP32. Nó xác định các cách xử lý cho các sự kiện khác nhau, bao gồm:

Sự kiện WIFI_EVENT_STA_START: Khi thiết bị được khởi động và sẵn sàng để kết nối vào mạng WiFi, sự kiện này được gửi để bắt đầu quá trình kết nối.

Sự kiện WIFI_EVENT_STA_DISCONNECTED: Khi kết nối WiFi của thiết bị bị ngắt kết nối, sự kiện này được gửi để xử lý việc kết nối lại hoặc báo lỗi nếu đã thử kết nối quá số lần cho phép.

Sự kiện IP_EVENT_STA_GOT_IP: Khi thiết bị đã kết nối thành công và có địa chỉ IP, sự kiện này được gửi để xác định địa chỉ IP của thiết bị và báo hiệu cho hệ thống rằng thiết bị đã kết nối thành công.



Hàm wifi_init_sta(): Đây là hàm để khởi tạo kết nối WiFi của thiết bị ESP32. Nó sử dụng các API của ESP-IDF để cấu hình mạng WiFi, bao gồm:

esp_netif_init(): Khởi tạo giao diện mạng.

esp_event_loop_create_default(): Khởi tạo vòng lặp sự kiện.

esp_netif_create_default_wifi_sta(): Tạo giao diện mạng WiFi.

esp_wifi_init(): Khởi tạo wifi với cấu hình mặc định.

esp_wifi_set_mode(): Thiết lập chế độ WiFi là WIFI_MODE_STA (chế độ kết nối vào mạng).

esp_wifi_set_config(): Thiết lập cấu hình WiFi cho giao diện mạng được tạo ra ở trên.

esp_wifi_start(): Bắt đầu kết nối WiFi.

Sau đó, hàm này sử dụng xEventGroupWaitBits() để chờ đợi kết nối WiFi được thiết lập hoặc báo lỗi nếu kết nối không thành công.

Chú ý rằng đoạn mã này sử dụng một số hằng số được định nghĩa trong tệp cấu hình, bao gồm tên và mật khẩu của mạng WiFi (EXAMPLE_ESP_WIFI_SSID và EXAMPLE_ESP_WIFI_PASS) và số lần thử kết nối lại tối đa (EXAMPLE_ESP_MAXIMUM_RETRY).

```c
// i2c master initialization
static esp_err_t i2c_master_init(void)
{
    int i2c_master_port = I2C_NUM_0;
    i2c_config_t conf = {
        .mode = I2C_MODE_MASTER,
        .sda_io_num = I2C_MASTER_SDA_IO,
        .sda_pullup_en = GPIO_PULLUP_ENABLE,
        .scl_io_num = I2C_MASTER_SCL_IO,
        .scl_pullup_en = GPIO_PULLUP_ENABLE,
        .master.clk_speed = I2C_MASTER_FREQ_HZ,
        // .clk_flags = 0,          /*!< Optional, you can use I2C_SCLK_SRC_FLAG_* flags to choose i2c source clock here. */
    };
    esp_err_t err = i2c_param_config(i2c_master_port, &conf);
    if (err != ESP_OK)
    {
        return err;
    }
    return i2c_driver_install(i2c_master_port, conf.mode, 0, 0, 0);
}
```

Đoạn code này khởi tạo giao diện I2C master và sử dụng nó để giao tiếp với một màn hình OLED SSD1306 và cảm biến nhiệt độ và độ ẩm AHT20. Hàm i2c_master_init thiết lập cấu hình I2C và cài đặt trình điều khiển I2C.

```c
void ssd1306_show_sensor_data(ssd1306_handle_t *dev, aht_t *aht)
{
    float temperature, humidity;
    uint8_t str[64];
    i2cdev_init();
    esp_err_t res = aht_get_data(&aht20, &temperature, &humidity);
    if (res == ESP_OK)
        ESP_LOGI(TAG_I2C, "Temperature: %.1f°C, Humidity: %.2f%%", temperature, humidity);
    else
        ESP_LOGE(TAG_I2C, "Error reading data: %d (%s)", res, esp_err_to_name(res));
    i2cdev_done();

    i2c_master_init();
    sprintf((char *)str, "Temperature: %.2f", temperature);
    ssd1306_draw_string(ssd1306, 0, 0, (uint8_t *)str, 12, 1);
    sprintf((char *)str, "Humidity: %.2f", humidity);
    ssd1306_draw_string(ssd1306, 0, 16, (uint8_t *)str, 12, 1);
    ssd1306_refresh_gram(ssd1306);
    i2c_driver_delete(I2C_NUM_0);
    sprintf((char *)str, "%.2f,%.2f", temperature, humidity);
    esp_mqtt_client_publish(client, "temperature_humidity/data", (char *)str, 0, 1, 0);
}
```

Hàm ssd1306_show_sensor_data đọc dữ liệu nhiệt độ và độ ẩm từ cảm biến AHT20, hiển thị nó trên màn hình OLED SSD1306, và đăng nó lên một chủ đề MQTT.

```c
void app_main(void)···
```

Hàm app_main khởi tạo cảm biến AHT20 và màn hình OLED SSD1306, thiết lập kết nối Wi-Fi và MQTT, và định kỳ đọc và hiển thị dữ liệu cảm biến.

2. 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["CE232_Lab"]
mycol = mydb["Lab5_Nhom7"]
```

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".

```python
def on_message(client_mqtt, userdata, message):
    print("received message =", str(message.payload.decode("utf-8")))
    doc = {}
    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 khi chương trình nhận được dữ liệu từ chủ đề "temperature_humidity/data". Đầu vào của hàm on_message bao gồm client_mqtt, userdata và message. Hàm này được thiết lập để xử lý dữ liệu nhận được từ message.payload.

Trong hàm on_message, chương trình lấy dữ liệu từ message.payload và chuyển đổi nhiệt độ và độ ẩm từ chuỗi sang số. Sau đó, chương trình tạo một dictionary mới và lưu trữ nhiệt độ, độ ẩm, timestamp và múi giờ hiện tại vào dictionary. Cuối cùng, chương trình lưu trữ dictionary này vào bảng "Lab5_Nhom7" trong cơ sở dữ liệu MongoDB.

Sử dụng client_mqtt.loop_forever() để cho phép chương trình lắng nghe các tin nhắn từ broker MQTT liên tục.

```python
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
temperature = 0
humidity = 0
```

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

```
# Trang chủ của ứng dụng web
@app.route('/')
def index():
    return render_template('index.html')

# 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")
    # Trả về dữ liệu dưới dạng JSON
    return jsonify({'temperature': temperature, 'humidity': humidity})
```

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 temperature và humidity.

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

```
# Hàm xử lý khi nhận được dữ liệu từ MQTT broker
def on_message(client, userdata, message):
    global temperature, humidity
    print("Received message: ", str(message.payload.decode("utf-8")))
    # Lấy dữ liệu từ message payload
    data = message.payload.decode()
    temperature, humidity = data.split(',')
    temperature = float(temperature)
    humidity = float(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 temperature và humidity.

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

```
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;
}
```

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

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

    <div class="content">
        <h2>Giảng viên Trần Hoàng Lộc</h2>
        <h3>Sinh viên thực hiện: nhóm 7</h3>

        <ul>
            <li>Trương Hữu Khang</li>
            <li>Nguyễn Linh Anh Khoa</li>
            <li>Hà Vĩnh Kiện</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.

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

```css
.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>
```

```
<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 Trần Hoàng Lộc</h2>
    <h3>Sinh viên thực hiện: nhóm 7</h3>

    <ul>
        <li>Trương Hữu Khang</li>
        <li>Nguyễn Linh Anh Khoa</li>
        <li>Hà Vĩnh Kiện</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.

```
$(document).ready(function () {
    // Gọi API để lấy dữ liệu nhiệt độ và độ ẩm mới nhất
    $.getJSON('/api/data', function (data) {
        // Cập nhật dữ liệu lên biểu đồ
        myChart.data.labels.push(new Date().toLocaleTimeString())
        myChart.data.datasets[0].data.push(data.temperature)
        myChart.data.datasets[1].data.push(data.humidity)
        myChart.update()
    })
    // Cập nhật dữ liệu mỗi 5 giây
    setInterval(function () {
        $.getJSON('/api/data', function (data) {
            myChart.data.labels.push(new Date().toLocaleTimeString())
            myChart.data.datasets[0].data.push(data.temperature)
            myChart.data.datasets[1].data.push(data.humidity)
            myChart.update()
        })
    }, 10000)
})
```

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.

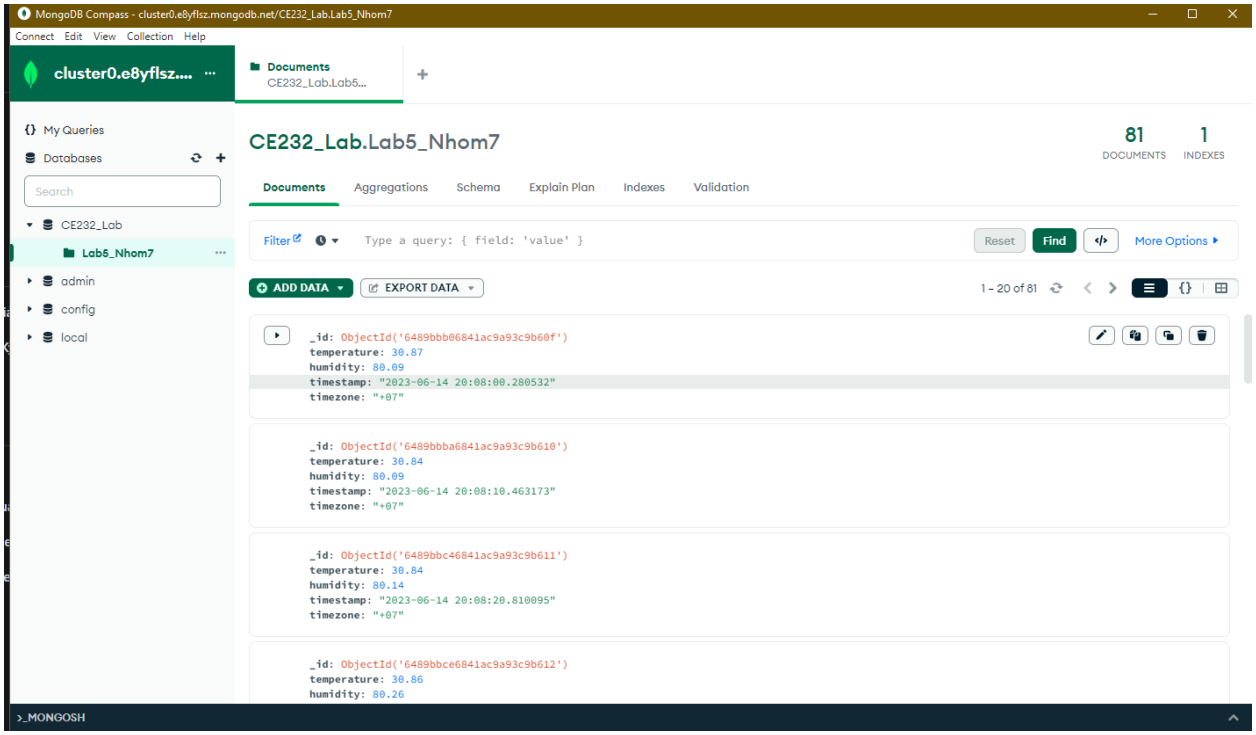
Cuối cùng, hàm setInterval() được sử dụng để cập nhật dữ liệu và vẽ lại biểu đồ mỗi 10 giây bằng cách gọi lại hàm lấy dữ liệu và cập nhật biểu đồ.

HOẠT ĐỘNG CỦA HỆ THỐNG

Hệ thống IoT này bao gồm một node ESP32 được cài đặt các cảm biến để lấy dữ liệu nhiệt độ, độ ẩm, hiển thị dữ liệu lên LCD SSD1306 và gửi dữ liệu về server với chu kỳ 10s/lần. Server nhận dữ liệu từ node ESP32 và lưu trữ dữ liệu vào cơ sở dữ liệu MongoDB, file Json. Một web-app được xây dựng để truy cập local cho phép người dùng

xem dữ liệu. Người dùng có thể xem dữ liệu dưới dạng đồ thị. Web-app có thể tự cập nhật dữ liệu mà không cần refresh.

MongoDB



WebServer

Video demo: https://youtu.be/zofdTVI-3A4

Source code: https://github.com/otaros/CE232_Lab/tree/main/Lab5