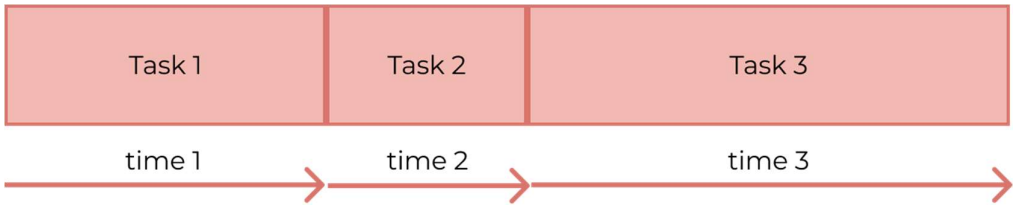
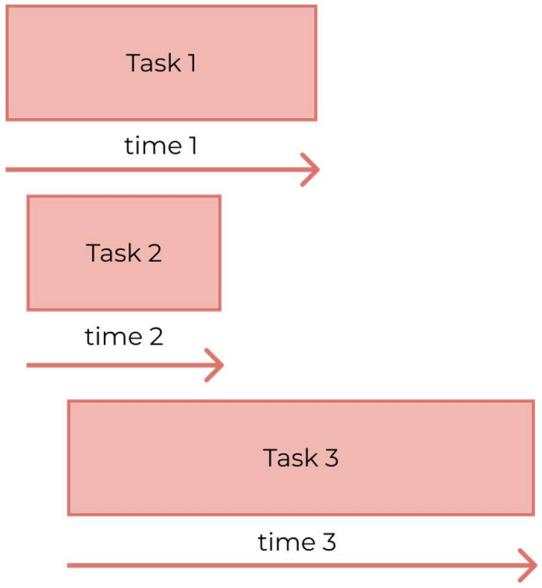


Hình 10. Minh họa xử lý đồng bộ

Khác với xử lý đồng bộ là xử lý bất đồng bộ (Asynchronous). Trong mô hình này, các công việc có thể được thực hiện cùng một lúc. Do vậy, công việc sau không phải chờ đợi công việc trước nữa. Do đó, sẽ có những trường hợp công việc sau kết thúc trước, nó có thể sẽ cho ra kết quả trong khi công việc trước đó còn đang thực thi nên kết quả của chương trình có thể sẽ không theo đúng thứ tự trực quan của nó. Tuy nhiên, do hạn chế tối đa việc "chờ đời" nên tổng thời gian thực hiện cả chương trình sẽ được rút ngắn một cách đáng kể. Đối với lập trình server, việc xử lý bất đồng bộ không chỉ tăng thời gian đáp ứng mà còn khai thác được khả năng xử lý song song, giúp server có thể đáp ứng nhiều hơn một kết nối trong cùng một lúc.



Hình 11. Minh họa xử lý bất đồng bộ

Từ các ưu điểm trên, nhóm quyết định thiết lập một máy chủ HTTP bất đồng bộ trên ESP8266 để chạy nền tảng web sử dụng thư viện ESPAsyncWebServer [2], thư viện hỗ trợ lập trình máy chủ HTTP không đồng bộ và máy chủ WebSocket.

3.4. Các ngôn ngữ, công nghệ web khác

Các ngôn ngữ và công nghệ web được sử dụng trong đề tài này gồm có HTML, CSS, JavaScript và AJAX.

HTML là chữ viết tắt cho cụm từ HyperText Markup Language (có nghĩa là ngôn ngữ đánh dấu siêu văn bản) là một ngôn ngữ đánh dấu được thiết kế ra để tạo nên các trang web hiện đang được sử dụng để tạo ra tất cả các website trên thế giới. HTML mô tả cấu trúc của văn

bản hay nội dung trang web, bênh cạnh đó còn có thể giúp người lập trình quy định màu sắc, thiết kế của các thành phần có trong trang web

CSS hay Cascading Style Sheet language, ngôn ngữ tạo phong cách cho trang web, được dùng để tạo phong cách và định kiểu cho những yếu tố được viết dưới dạng ngôn ngữ đánh dấu, như là HTML. CSS có chức năng điều khiển định dạng của nhiều trang web cùng lúc để tiết kiệm công sức cho người viết web, ngoài ra còn giúp tách biệt phần quy định cách hiển thị của trang web với nội dung chính của trang. Phương thức hoạt động của CSS là sẽ tìm kiếm các vùng được chọn, có thể là tên một thẻ HTML, tên một ID, class hay nhiều kiểu khác và sau đó sẽ áp dụng các thuộc tính cần thay đổi lên vùng chọn đó.

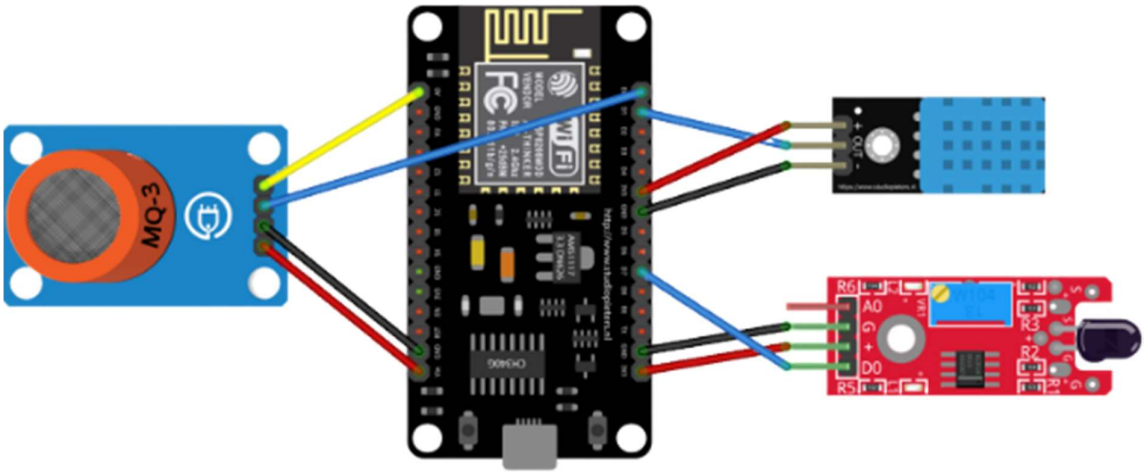
JavaScript là một ngôn ngữ lập trình hoặc ngôn ngữ kịch bản cho phép triển khai các chức năng phức tạp trên trang ví dụ như trình chiếu ảnh, đồ họa hoạt hình hay hình thức tương tác, v.v. Mô hình đối tượng tài liệu (DOM) cho phép tạo và kiểm soát nội dung của trang web động với JS, có thể thay đổi nội dung trang HTML, có thể hiển thị / ẩn các phần tử trang HTML, có thể thay đổi các thuộc tính HTML của phần tử HTML, v.v. mà không yêu cầu người dùng tải lại trang web theo cách thủ công.

AJAX (Asynchronous JavaScript and XML) là một bộ các kỹ thuật thiết kế web giúp cho các ứng dụng web hoạt động bất đồng bộ. Và trong đó, yếu tố quyết định của AJAX là đối tượng XMLHttpRequest. Tất cả các trình duyết hiện tại đều hỗ trợ đối tượng XMLHttpRequest. Đối tượng XMLHttpRequest cho phép các trang web được cập nhật không đồng bộ bằng cách trao đổi dữ liệu với máy chủ web từ phía sau và điều này có nghĩa là có thể cập nhật các phần của trang web mà không cần tải lại toàn bộ trang

PHẦN III. THIẾT KẾ VÀ CÀI ĐẶT

1. Khối điều khiển và cảm biến

Khối điều khiển và cảm biến gồm có 4 linh kiện chính: ESP8266, DHT11, MQ-3 và cảm biến lửa. Trong đó ESP8266 là mạch xử lý chính, giữ nhiệm vụ cấp nguồn, điều khiển và nhận dữ liệu từ các cảm biến kể trên. Nguồn cấp cho ESP8266 sẽ được cắm trực tiếp vào cổng URAT vào laptop (5V), và các cảm biến sẽ được cấp nguồn thông qua chân của ESP8266. Dữ liệu sẽ được đọc từ các chân digital và analog có kết nối với cảm biến. Sơ đồ nối chân của các linh kiện trong khối điều khiển được minh họa trong hình bên dưới:



Hình 12. Sơ đồ nối chân của khối điều khiển và cảm biến

Bảng 2. Bảng thông số nối chân của khối điều khiển và cảm biến

STT	Sensor	Sensor pin	ESP8266 Pin
1	DHT11	VCC	3.3V
		GND	GND
		D0	D1
2	Flame Sensor	VCC	3.3V
		GND	GND
		D0	D7
3	MQ-3	VCC	Vin (5V)
		GND	GND
		A0	A0
		D0	D0

2. LẬP TRÌNH VÀ CÀI ĐẶT

2.1. Tập tin JavaScript dùng để tự động cập nhật các chỉ số môi trường

Dưới đây là một phần của mã lệnh trong tập tin JavaScript dùng để cập nhật các chỉ số nhiệt độ, độ ẩm và nồng độ cồn một cách tự động mà không cần phải chạy lại toàn bộ trang web.

Khai báo biến time lưu trữ thời gian tự động cập nhật và biến flame dùng để lưu trữ trạng thái tín hiệu báo cháy.

```
var time = 1000; // 1000 millis seconds = 1 second
var flame = 1; // flame: 1-Fire, 0-No flame
```

Đoạn mã chương trình sau đây sẽ mô tả việc tự động cập nhật chỉ số nhiệt độ, làm tương tự với chỉ số độ ẩm và nồng độ cồn. Sử dụng phương thức setInterval(F(), T) dùng để thiết lập độ trễ cho hàm F nằm ở tham số đầu tiên mộ khoản thời gian bằng T millisecond. Nghĩa là hàm function() ở bên dưới sẽ được thực thi liên tục trong mỗi giây.

```
setInterval(function() {
```

Khởi tạo một đối tượng XMLHttpRequest() để chuẩn bị gửi yêu đến server.

```
  var xhttp = new XMLHttpRequest();
```

Sau khi nhận được phản hồi từ server, tự động cập nhật giá trị của phần tử có id là "temperature" trong tập tin HTML lại cho phù hợp.

```
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200)
      document.getElementById("temperature").innerHTML =
this.responseText;
  };
```

Gửi một yêu cầu GET với URL "/temperature" để đọc giá trị nhiệt độ mới nhất.

```
  xhttp.open("GET", "/temperature", true);
  xhttp.send();
}, time);
```

Tương tự như trên, đoạn mã dưới đây dùng để tự động hiển thị cảnh báo cháy ra màng hình browser mỗi khi có tín hiệu phát hiện được lửa.

```
setInterval(function() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200)
    {
      var newF = this.responseText;
      if (newF == 0 && newF != flame)
        document.getElementById('flame-
modal').style.display = "flex";
      flame = newF;
    }
  };
  xhttp.open("GET", "/flame", true);
  xhttp.send();
}, time);
```

2.2. ESP8266 sketch

Thêm các thư viện cần thiết

```
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <FS.h>
#include <DHT.h>
#include <Wire.h>
```

Nhập thông số mạng WiFi để ESP8266 có thể kết nối được với Internet

```
const char* ssid = ""; // WiFi SSID
const char* password = ""; // WiFi password
```

Định nghĩa biến

```
AsyncWebServer server(80);
```

Định nghĩa các biến cần thiết

```
#define dhtPin D1
#define dhtType DHT11
#define flamePin D7
DHT dht(dhtPin, dhtType);
float t = 0.0; // temperature
float h = 0.0; // humidity
float a = 0.0; // alcohol
int f = HIGH; // flame: HIGH: no flame; LOW: fire!
```

Định nghĩa các thông số liên quan đến kênh ThingSpeak được sử dụng

```
const int channelID = 1063062;
String writeAPIKey = "RAOKCHK6EQOSPSTR"; // write API key
for your ThingSpeak Channel
const char* thingspeak_server = "api.thingspeak.com";
WiFiClient client;
```

Hàm processor() có chức năng xử lý việc gán giá trị thu được từ cảm biến vào các placeholder trong tệp HTML. Placeholder được định nghĩa trong hai dấu phần trăm, ví dụ: %TEMPERATURE%

```
String processor(const String& var) {
  if (var == "TEMPERATURE") return String(t);
  else
    if (var == "HUMIDITY") return String(h);
    else
      if (var == "ALCOHOL") return String(a);
      else
        if (var == "FLAME") return String(f);
  return String();
}
```

Hàm setup(), bắt đầu khởi tạo Serial Monitor và thiết lập các cảm biến, GPIO làm đầu đọc dữ liệu.

```
void setup() {
  Serial.begin(115200);

  // Initialize sensor
  dht.begin();
  pinMode(flamePin, INPUT);
```

Khởi tạo SPIFFS

```
  // Initialize SPIFFS
  if (!SPIFFS.begin())
  {
```

```
    Serial.println("Error SPIFFS");
    return;
  }
```

Kết nối WiFi cho ESP8266

```
  // Connect to WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.println("Connecting");
    delay(5000);
  }
  Serial.println("Connected");
  Serial.println(WiFi.localIP());
```

Sử dụng thư viện ESPAsyncWebServer để cấu hình các đường dẫn (route) để máy chủ lắng nghe và xử lý các HTTP Request. Sử dụng phương thức on cho đối tượng server như sau:

```
  // Route for root / web page
  server.on("/", HTTP_GET, [](AsyncWebServerRequest
*request){
      request->send(SPIFFS, "/index.html", String(), false,
processor);
    });
```

Khi máy chủ nhận được yêu cầu trên đường dẫn gốc "/", máy sẽ gửi tập tin index.html đã được lưu trong hệ thống tập tin flash của ESP8266 đến máy client. Tham số cuối của hàm send() là hàm processor giúp ta có thể thay thế các placholder tương ứng trong các tệp HTML. Làm tương tự khi máy chủ nhận được yêu cầu trên các đường dẫn khác:

```
  server.on("/index.html", HTTP_GET,
[](AsyncWebServerRequest *request) {
      request->send(SPIFFS, "/index.html", String(), false,
processor);
    });
  server.on("/content.html", HTTP_GET,
[](AsyncWebServerRequest *request) {
      request->send(SPIFFS, "/content.html", String(),
false, processor);
    });
  server.on("/chart.html", HTTP_GET,
[](AsyncWebServerRequest *request) {
      request->send(SPIFFS, "/chart.html", String(), false,
processor);
    });
```

Do các trang html có sử dụng tham chiếu đến tập tin CSS và JavaScript, nên máy khác cũng sẽ đưa ra yêu cầu tập tin đó. Khi đó, máy chủ sẽ gửi tập tin tương ứng đến máy khách:

```
  server.on("/css/styles.css", HTTP_GET,
[](AsyncWebServerRequest *request) {
      request->send(SPIFFS, "/css/styles.css", "text/css");
    });
  server.on("/css/index.css", HTTP_GET,
[](AsyncWebServerRequest *request) {
      request->send(SPIFFS, "/css/index.css", "text/css");
    });
  server.on("/css/content.css", HTTP_GET,
[](AsyncWebServerRequest *request) {
      request->send(SPIFFS, "/css/content.css",
"text/css");
    });
  server.on("/css/chart.css", HTTP_GET,
[](AsyncWebServerRequest *request) {
      request->send(SPIFFS, "/css/chart.css", "text/css");
    });

  server.on("/script.js", HTTP_GET,
[](AsyncWebServerRequest *request) {
      request->send(SPIFFS, "/script.js",
"text/javascript");
    });
```

Trong tập tin JavaScript có sử dụng một đoạn mã yêu cầu các thông số nhiệt độ, độ ẩm, độ cồn và tín hiệu có lửa trên các tuyến /temperature, /humidity, /alcohol và /flame mỗi 1 giây. Khi nhận được các yêu cầu như vậy, server sẽ gửi các giá trị cảm biến đã được lưu trong các biến t, h, a và f đã được định nghĩa và cập nhật giá trị thường xuyên dưới dạng plain text thông qua phương thức c_str().

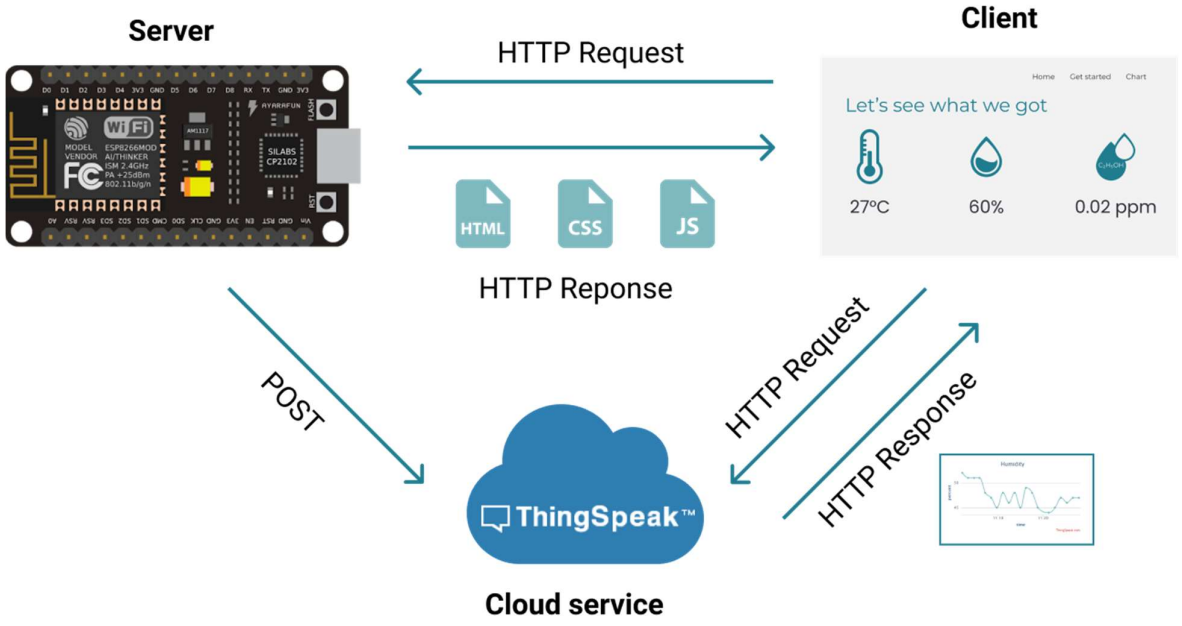
```
  // Response to XMLHttpRequest
  server.on("/temperature", HTTP_GET,
[](AsyncWebServerRequest *request){
      request->send_P(200, "text/plain",
String(t).c_str());
    });
  server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest
*request){
      request->send_P(200, "text/plain",
String(h).c_str());
    });
  server.on("/alcohol", HTTP_GET, [](AsyncWebServerRequest
*request){
      request->send_P(200, "text/plain",
String(a).c_str());
    });
  server.on("/flame", HTTP_GET, [](AsyncWebServerRequest
*request) {
```

```
      a = newA;
      Serial.print(t); Serial.print(" "); Serial.print(h);
Serial.print(" "); Serial.println(a);
```

Kết nối với ThingSpeak và tiến hành gửi dữ liệu.

```
      if (client.connect(thingspeak_server, 80))
      {
        String str = "field1=" + String(t, 2) + "&field2="
+ String(h, 2) + "&field3=" + String(a, 2);
        client.print("POST /update HTTP/1.1\n");
        client.print("Host: api.thingspeak.com\n");
        client.print("Connection: close\n");
        client.print("X-THINGSPEAKAPIKEY: " + writeAPIKey +
"\n");
        client.print("Content-Type: application/x-www-form-
urlencoded\n");
        client.print("Content-Length: ");
        client.print(str.length());
        client.print("\n\n");
        client.print(str);
        client.print("\n\n");
      }
      client.stop();
    }
```

3. MÔ HÌNH HỆ THỐNG VÀ NGUYÊN LÝ HOẠT ĐỘNG



Hình 13. Minh họa nguyên lý hoạt động của hệ thống

Ngoài vai trò là chip xử lý chính, giữ nhiệm vụ điều khiển và thu thập dữ liệu từ các cảm biến, ESP8266 còn là một server HTTP bất đồng bộ. Khi có yêu cầu từ máy khách, server