**ỦY BAN NHÂN DÂN THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC SÀI GÒN**

**KHOA ĐIỆN TỬ VIỄN THÔNG**

**đỌC DỮ LIỆU NHIỆT ĐỘ, ĐỘ ẨM**

**HIỂN THỊ LÊN SERVER NODE.JS**

**ĐỒ ÁN CHUYÊN NGÀNH ĐIỆN TỬ MÁY TÍNH**

**NGÀNH: kỹ thuật điện, điện tử**

**tRÌNH ĐỘ ĐÀO TẠO: ĐẠI HỌC**

**SINH VIÊN THỰC HIỆN GIẢNG VIÊN HƯỚNG DẪN**

**VÕ TRÍ DŨNG THS. nguyễn hữu phúc**

**TP. HỒ CHÍ MINH, THÁNG 5 NĂM 2018**

**ỦY BAN NHÂN DÂN THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC SÀI GÒN**

**KHOA ĐIỆN TỬ VIỄN THÔNG**

**VÕ TRÍ DŨNG**

**đỌC DỮ LIỆU NHIỆT ĐỘ, ĐỘ ẨM**

**HIỂN THỊ LÊN SERVER NODE.JS**

**ĐỒ ÁN CHUYÊN NGÀNH ĐIỆN TỬ MÁY TÍNH**

**NGÀNH: kỹ thuật điện, điện tử**

**tRÌNH ĐỘ ĐÀO TẠO: ĐẠI HỌC**

**GIẢNG VIÊN HƯỚNG DẪN: THS. nguyễn hữu phúc**

**TP. HỒ CHÍ MINH, THÁNG 5 NĂM 2018**

Mục lục

[LỜI NÓI ĐẦU iv](#_Toc514510788)

[DANH MỤC CÁC CHỮ VIẾT TẮT vi](#_Toc514510789)

[DANH MỤC CÁC HÌNH ẢNH vii](#_Toc514510790)

[DANH MỤC CÁC SƠ ĐỒ viii](#_Toc514510791)

[CHƯƠNG 1: TỔNG QUAN HỆ THỐNG GIÁM SÁT NHIỆT ĐỘ VÀ ĐỘ ẨM 1](#_Toc514510792)

[**1.1 Tổng quan hệ thống** 1](#_Toc514510793)

[**1.1.1 Khái niệm về nhiệt độ và độ ẩm** 1](#_Toc514510794)

[**1.1.2 Tổng quan về hệ thống** 1](#_Toc514510795)

[**1.2 Tổng quan linh kiện sử dụng** 2](#_Toc514510796)

[**1.2.1 Board ESP8266 Wifi Uno** 2](#_Toc514510797)

[**1.2.2 Cảm biến DHT11** 4](#_Toc514510798)

[**1.2.3 Màn hình OLED SSD1306** 6](#_Toc514510799)

[**1.3 Các chuẩn giao tiếp được sử dụng** 7](#_Toc514510800)

[**1.3.1 Giao tiếp 1-wire** 7](#_Toc514510801)

[**1.3.2 Giao tiếp I2C** 8](#_Toc514510802)

[CHƯƠNG 2: LẬP TRÌNH CHO ỨNG DỤNG 10](#_Toc514510803)

[**2.1 Phân tích chương trình** 10](#_Toc514510804)

[**2.2 Lập trình cho Server** 10](#_Toc514510805)

[**2.2.1 Lập trình file thực thi App.js** 10](#_Toc514510806)

[**2.2.2 Lập trình file Index.html** 11](#_Toc514510807)

[**2.2.3 Triển khai server lên Heroku** 13](#_Toc514510808)

[**2.3 Lập trình ESP8266** 14](#_Toc514510809)

[**2.3.1 Lưu đồ giải thuật** 14](#_Toc514510810)

[**2.4 Chạy chương trình thực tế** 15](#_Toc514510811)

[CHƯƠNG 3: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 17](#_Toc514510812)

[DANH MỤC TÀI LIỆU THAM KHẢO 18](#_Toc514510813)

[PHỤ LỤC 19](#_Toc514510814)

# LỜI NÓI ĐẦU

Với sự phát triển nhanh chóng của IoT, cùng với nền kinh tế Việt Nam tập trung chủ yếu vào nông nghiệp. Các thiết bị giám sát nhiệt độ, độ ẩm môi trường ngày càng phát triển mạnh, kết hợp với IoT giúp kết nối các thiết bị và con người lại với nhau. Vì vậy, nhu cầu về một thiết bị giám sát các đặt tính môi trường (nhiệt độ, độ ẩm) kết hợp với IoT ngày càng tăng. Đồng thời, thị trường hiện tại cũng xuất hiện rất nhiều sản phẩn tương tự xuất xứ từ Trung Quốc hoặc không rõ nguồn gốc, vì vậy việc an toàn và bảo mật dữ liệu cũng rất được ưu tiên.

Trong quá trình thực hiện đề tài không tránh khỏi những sai sót do kiến thức có giới hạn, cũng như sự thiếu chính xác từ nhiều nguồn tài liệu từ internet, sách, báo…Rất mong nhận được sự đóng góp của quý thầy cô cũng như các bạn để đề tài thực hiện thành công và phát triển hơn nữa.

**LỜI CÁM ƠN**

Tôi xin gửi lời cám ơn sâu sắc tới thầy **Nguyễn Hữu Phúc** đã tận tình giúp đở, chỉ bảo, hướng dẫn nhóm thực hiện trong suốt quá trình thực hiện đề tài này. Trong thời gian làm việc với thầy, tôi không những tiếp thu thêm nhiều kiến thức bổ ích mà còn học tập được tinh thần làm việc, thái độ nghiên cứu khoa học nghiêm túc, hiệu quả của thầy, đây là những điều rất cần thiết cho em trong quá trình học tập và công tác sau này.

Chân thành cảm ơn đến các bạn đã luôn đồng hành, giúp đỡ để tôi có thể thực hiện đề tài này thành công.

Chân thành cảm ơn!

**Võ Trí Dũng**

# DANH MỤC CÁC CHỮ VIẾT TẮT

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IoT |  |  | : | Internet of Thing |
| https |  |  | : | Hypertext Transfer Protocol Secure |
| SPI |  |  | : | Serial Peripheral Interface |
| UART |  |  | : | Universal Asynchronous Receiver-Transmitter |
| OLED |  |  | : | Organic light-emitting diode |
| I2C |  |  | : | Inter-Intergrated Circuit |

# DANH MỤC CÁC HÌNH ẢNH

|  |  |  |
| --- | --- | --- |
| **TT** | **Tên bảng** | **Trang** |
| 1 | Hình 1.1: Sơ đồ chân board ESP8266 Wifi Uno | 3 |
| 2 | Hình 1.2: Cảm biến nhiệt độ , độ ẩm | 4 |
| 3 | Hình 1.3: Màn hình OLED | 6 |
| 4 | Hình 1.4: Quá trình truyền nhận dữ liệu trong chuẩn 1-Wire | 7 |
| 5 | Hinh 2.1 Giao diện web cho người dùng | 12 |
| 6 | Hinh 2.2 Kết quả trên trình duyệt web | 15 |
| 7 | Hinh 2.3 Thông tin trả về từ server | 16 |

**­­­­­**

# DANH MỤC CÁC SƠ ĐỒ

|  |  |  |
| --- | --- | --- |
| **TT** | **Tên sơ đồ** | **Trang** |
| 1 | Sơ đồ 1.1: Sơ đồ kết nối tổng quát các module | 2 |
| 2 | Sơ đồ 1.2: Mô hình giao tiếp I2C | 8 |
| 3 | Sơ đồ 2.1: Lưu đồ giải thuật | 14 |

# CHƯƠNG 1: TỔNG QUAN HỆ THỐNG GIÁM SÁT NHIỆT ĐỘ VÀ ĐỘ ẨM

## **1.1 Tổng quan hệ thống**

### **1.1.1 Khái niệm về nhiệt độ và độ ẩm**

**Nhiệt độ** là đại lượng thể hiện tính chất vật lý nóng, lạnh của vật chất. Nhiệt độ được đo bằng các đơn vị khác nhau và có thể biến đổi bằng các công thức. Trong hệ đo lường quốc tế, nhiệt độ được đo bằng đơn vị Kelvin, ký hiệu là K. Trong đời sống ở Việt Nam và nhiều nước, nó được đo bằng độ C.

**Độ ẩm tương đối** là tỷ số của áp suất hơi nước hiện tại của bất kỳ một hỗn hợp khí nào với hơi nước so với áp suất hơi nước bão hòa tính theo đơn vị là %. Định nghĩa khác của độ ẩm tương đối là tỷ số giữa khối lượng nước trên một thể tích hiện tại so với khối lượng nước trên cùng thể tích đó khi hơi nước bão hòa

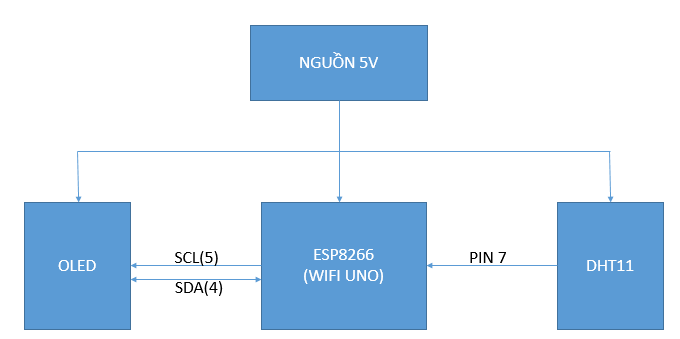
### **1.1.2 Tổng quan về hệ thống**

Hệ thống được phát triển dựa trên nền tảng chip ESP8266. Đây là dòng chip SoC có khả năng kết nối Wifi. ESP8266 sẽ đọc dữ liệu nhiệt độ, độ ẩm từ cảm biến DHT11 thông qua giao thức 1-wire.

Dữ liệu nhận được sẽ được tiến hành so sánh với giá trị những nhằm điều khiển động cơ bật tắt.

Một server Node.js sẽ thực hiện việc cập nhật dữ liệu từ ESP8266 (dữ liệu nhiệt độ, độ ẩm và trạng thái động cơ) đồng thời cho phép người dùng truy xuất dữ liệu này thông qua giao diện web.

**Sơ đồ khối**

****

Sơ đồ 1.1: Sơ đồ kết nối tổng quát các module

Module DHT11 giúp đo nhiệt độ, độ ẩm giao tiếp với Board ESP8266 Wifi Uno thông qua giao tiếp 1-wire tại chân số 7.

Module màn hình OLED SSD1306 hiên thị thông tin nhiệt độ, độ ẩm giao tiếp với board ESP8266 Wifi Uno thông qua giao tiếp I2C chân số 4 (SDA) và chân số 5 (SCL)

## **1.2 Tổng quan linh kiện sử dụng**

### **1.2.1 Board ESP8266 Wifi Uno**

ESP8266 - IoT WiFi Uno với thiết kế tương thích với Arduino Uno, có MCU là module ESP-12E hổ trợ firmware chương trình Arduino giúp dễ dàng lập trình. Đặc biệt mạch có thể được lập trình để chạy các ứng dụng thu thập và điều khiển dữ liệu qua wifi.

Thông số kĩ thuật:

CPU ESP8266

Hỗ trợ kết nối WiFi

Tương thích Arduino UNO

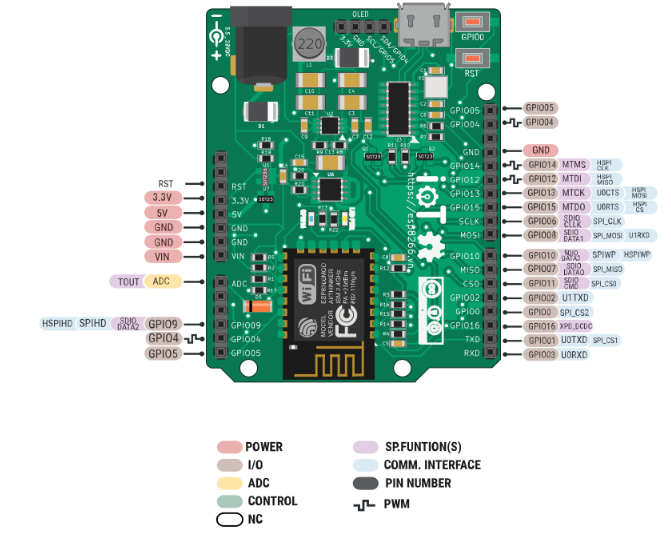
Có thể lập trình được bằng C/C++, Arduino IDE, Micropython, NodeMCU - Lua

Nguồn 9-24V hay 5V từ USB

11 IO, 1 Analog in

4 Mbytes Flash

Module ESP-12F chỉ 3.3VDC ( tối đa 3.6 VDC )

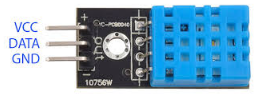
****

Hình 1.1: Sơ đồ chân board ESP8266 Wifi Uno

Với giá thành rẻ, khả năng kết nối Internet thông qua Wifi đồng thời cung cấp các giao tiếp ngoại vi cở bản như I2C, SPI, I/O, UART, PWM,... đáp ứng đủ nhu cầu cho chúng ta phát triển các ứng dụng cơ bản.

### **1.2.2 Cảm biến DHT11**

DHT11 Là cảm biến rất thông dụng hiện nay vì chi phí rẻ và rất dễ lấy dữ liệu thông qua giao tiếp 1-wire (giao tiếp digital 1-wire truyền dữ liệu duy nhất). Cảm biến được tích hợp bộ tiền xử lý tín hiệu giúp dữ liệu nhận về được chính xác mà không cần phải qua bất kỳ tính toán nào.



Hình 1.2: Cảm biến nhiệt độ , độ ẩm

**Thông số kỹ thuật**

Điện áp hoạt động: 3V - 5V (DC)

Dãi độ ẩm hoạt động: 20% - 90% RH, sai số ±5%RH

Dãi nhiệt độ hoạt động: 0°C ~ 50°C, sai số ±2°C

Khoảng cách truyển tối đa: 20m

Cảm biến nhiệt độ, độ ẩm giao tiếp với ESP8266 thông qua giao tiếp 1-wire (chân số 7). Ngoài ra, chúng ta có thể sử dụng module cảm biến DHT22 với giá thành cao hơn nhưng cho khả năng đo chính xác hơn.

**Nguyên lý hoạt động và phương pháp đọc tín hiệu**

Để có thể giao tiếp với DHT11 theo chuẩn 1 chân vi xử lý thực hiện theo bước:

**Bước 1:**Gửi tin hiệu muốn đo (Start) tới DHT11, sau đó DHT11 xác nhận lại.

* Khi đã giao tiếp được với DHT11, cảm biến sẽ gửi lại 5 byte dữ liệu và nhiệt độ đo được.
* MCU thiết lập chân DATA là Output, kéo chân DATA xuống 0 trong khoảng thời gian >18ms. Khi đó DHT11 sẽ hiểu MCU muốn đo giá trị nhiệt độ và độ ẩm.
* MCU đưa chân DATA lên 1, sau đó thiết lập lại là chân đầu vào.
* Sau khoảng 20-40us, DHT11 sẽ kéo chân DATA xuống thấp. Nếu >40us mà
* chân DATA ko được kéo xuống thấp nghĩa là ko giao tiếp được với DHT11.
* Chân DATA sẽ ở mức thấp 80us sau đó nó được DHT11 kéo nên cao trong
* 80us. Bằng việc giám sát chân DATA, MCU có thể biết được có giao tiếp được với DHT11 ko. Nếu tín hiệu đo được DHT11 lên cao, khi đó hoàn thiện quá trình giao tiếp của MCU với DHT.

**Bước 2:** Đọc giá trị trên DHT11

DHT11 sẽ trả giá trị nhiệt độ và độ ẩm về dưới dạng 5 byte. Trong đó:

* Byte 1 : giá trị phần nguyên của độ ẩm (RH%)
* Byte 2 : giá trị phần thập phân của độ ẩm (RH%)
* Byte 3 : giá trị phần nguyên của nhiệt độ (TC)
* Byte 4 : giá trị phần thập phân của nhiệt độ (TC)
* Byte 5 : kiểm tra tổng.

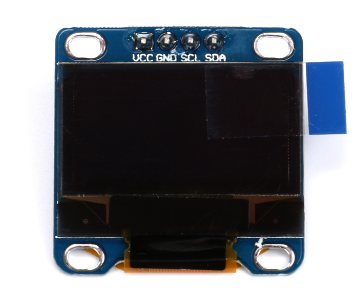
Nếu Byte 5 = (8 bit) (Byte1 +Byte2 +Byte3 + Byte4) thì giá trị độ ẩm và nhiệt độ là chính xác, nếu sai thì kết quả đo không có nghĩa.

**Đọc dữ liệu:**

Sau khi giao tiếp được với DHT11, DHT11 sẽ gửi liên tiếp 40 bit 0 hoặc 1 về MCU, tương ứng chia thành 5 byte kết quả của nhiệt độ và độ ẩm.

Sau khi tín hiệu được đưa về 0, ta đợi chân DATA của MCU được DHT11 kéo lên 1. Nếu chân DATA là 1 trong khoảng 26-28 us thì là 0, còn nếu tồn tại 70us là 1. Do đó trong lập trình ta bắt sườn lên của chân DATA, sau đó delay 50us. Nếu giá trị đo được là 0 thì ta đọc được bit 0, nếu giá trị đo được là 1 thì giá trị đo được là 1. Cứ như thế ta đọc các bit tiếp theo

### **1.2.3 Màn hình OLED SSD1306**

****

Hình 1.3: Màn hình OLED

OLED (viết tắt bởi Organic Light Emitting Diode: Diode phát sáng hữu cơ) đang trở thành đối thủ cạnh tranh cũng như ứng cử viên sáng giá thay thể màn hình LCD.

Màn hình OLED gồm những lớp như tấm nền, Anode, lớp hữu cơ, cathode. Và phát ra ánh sáng theo cách tương tự như đèn LED. Quá trình trên được gọi là phát lân quang điện tử.

Những ưu điểm có thể kể đến trên màn hình OLED là những lớp hữu cơ nhựa mỏng, nhẹ mềm dẻo hơn những lớp tinh thể trên LED hay LCD nhờ vậy mà có thể ứng dụng OLED để chế tạo màn hình gập cuộn được. Độ sáng của OLED cũng tốt hơn LED và không cần đèn nền như trên LCD nên sử dụng pin ít hơn. Góc nhìn cũng cài thiện hơn những công nghệ tiền nhiệm, khoảng 170 độ.

Nhược điểm có thể kể tới là tuổi thọ màn này khá thấp, giá thành sản xuất cao và rất dễ hỏng khi gặp nước. Nên dễ hiểu màn hình này chưa được ứng dụng nhiều.

**Thông số kĩ thuật:**

Điện áp sử dụng: 3V3 đến 5V (DC)

Công suất tiêu thụ: 0.04W

Góc hiển thị: Lớn hơn 160 độ (Em chưa hiểu chỗ này, bác nào giải thích hộ)

Độ phân giải: 128X64 pixel (Điểm ảnh)

Độ rộng màn hình: 0.96inch

Giao tiếp: I2C

Màu: Trắng và Đen

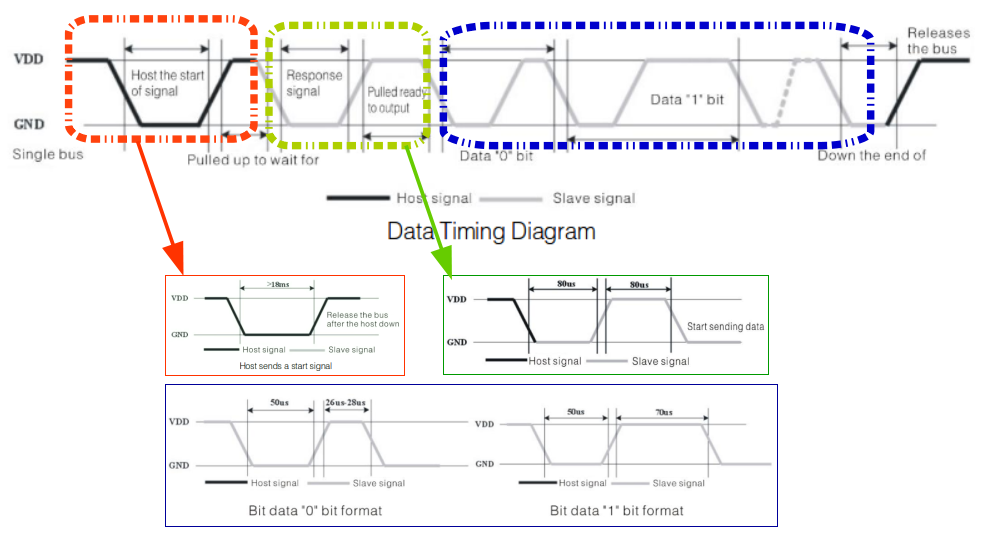
Driver: SSD1306

## **1.3 Các chuẩn giao tiếp được sử dụng**

### **1.3.1 Giao tiếp 1-wire**

OneWire là chuẩn giao tiếp nối tiếp được thiết kế bởi hãng Dallas. Đó là hệ thống bus nhằm kết nối các thiết bị với nhau để truyền hoặc nhận dữ liệu.Trong chuẩn giao tiếp này thường chỉ sử dụng 1 chân đồng thời là vừa là nguồn cung cấp vừa là chân truyền nhận dữ liệu. Cũng giống như các chuẩn giao tiếp khác, OneWire cũng gồm 3 giai đoạn reqquest (hỏi) → respond (đáp) → data reading (truyền nhận dữ liệu).

Hình ảnh mô tả quá trình truyền,nhận dữ liệu của DHT11 như hình bên dưới



Hình 1.4: Quá trình truyền nhận dữ liệu trong chuẩn OneWire

**Tóm tắt**

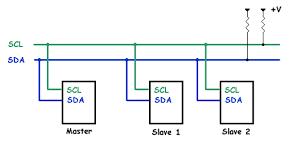
Master (ESP8266) gửi tín hiệu START, DHT11 sẽ chuyển từ chế độ tiết kiệm năng lượng (low-power mode) sang chế độ làm việc bình thường (high-speed mode)

DHT11 nhận được tín hiệu và phản hồi đến master, master nhận tín hiệu và bắt đầu quá trình truyền dữ liệu.

DHT11 sẽ gửi dữ liệu lên bus, mỗi lần gửi là 1 gói 40 bits data.

Khi muốn kết thúc, Master sẽ gửi tín hiệu STOP, kết thúc quá trình truyền nhận dữ liệu

### **1.3.2 Giao tiếp I2C**

****

Sơ đồ 1.2: Mô hình giao tiếp I2C

Ngày nay trong các hệ thống điện tử hiện đại, rất nhiều ICs hay thiết bị ngoại vi cần phải giao tiếp với các ICs hay thiết bị khác – giao tiếp với thế giới bên ngoài. Với mục tiêu đạt được hiệu quả cho phần cứng tốt nhất với mạch điện đơn giản, Phillips đã phát triển một chuẩn giao tiếp nối tiếp 2 dây được gọi là I2C. I2C là tên viết tắt của cụm từ Inter ‐ Intergrated Circuit – Bus giao tiếp giữa các IC với nhau.

I2C mặc dù được phát triển bới Philips, nhưng nó đã được rất nhiều nhà sản xuất IC trên thế giới sử dụng. I2C trở thành một chuẩn công nghiệp cho các giao tiếp điều khiển, có thể kể ra đây một vài tên tuổi ngoài Philips như: Texas Intrument (TI), Maxim‐Dallas, analog Device, National Semiconductor … Bus I2C được sử dụng làm bus giao tiếp ngoại vi cho rất nhiều loại IC khác nhau như các loại Vi điều khiển 8051, PIC, AVR, ARM, chíp nhớ như RAM tĩnh (Static Ram), EEPROM, bộ chuyển đổi tương tự số (ADC), số tương tụ (DAC), IC điểu khiển LCD, LED…

**Đặc điểm giao tiếp I2C**

Một giao tiếp I2C gồm có 2 dây: Serial Data (SDA) và Serial Clock (SCL). SDA là đường truyền dữ liệu 2 hướng, còn SCL là đường truyền xung đồng hồ và chỉ theo một hướng. Như hình vẽ trên, khi một thiết bị ngoại vi kết nối vào đường I2C thì chân SDA của nó sẽ nối với dây SDA của bus, chân SCL sẽ nối với dây SCL.

Mỗi dây SDA hay SCL đều được nối với điện áp dương của nguồn cấp thông qua một điện trở kéo lên (pull‐up resistor). Sự cần thiết của các điện trở kéo này là vì chân giao tiếp I2C của các thiết bị ngoại vi thường là dạng cực máng hở (open‐drain or open‐collector). Giá trị của các điện trở này khác nhau tùy vào từng thiết bị và chuẩn giao tiếp, thường dao động trong khoảng 1KΩ đến 4.7KΩ.

# CHƯƠNG 2: LẬP TRÌNH CHO ỨNG DỤNG

**2.1 Phân tích chương trình**

Hệ thống cần các chức năng sau:

1 Web Server viết bằng Javascript, thực thi bởi Node.js, Server sẽ được triển khai lên Heroku giúp người dùng có thể truy cập thông qua internet.

ESP8266 sau khi kết nối vào mạng WiFi, sẽ tiến hành đọc thông số nhiệt độ, độ ẩm từ cảm biến DHT11 và gởi về Server sau mỗi 2 giây.

Quá trình gởi được thực hiện bởi phương thức GET, ví dụ http://192.168.1.102/update?temp=25&humd=80 với 192.168.1.102 là địa chỉ Web Server, /update là đường dẫn, temp=20 và humd=80 chứa thông tin nhiệt độ 20 độ C và độ ẩm 80%.

Web Server trả về trạng thái HTTP status = 200 (OK), cùng với việc hiển thị ra cửa sổ log giá trị nhiệt độ, độ ẩm.

Người dùng muốn truy câp vào hệ thống cần một thiết bị có kết nối Internet.

## **2.2 Lập trình cho Server**

### **2.2.1 Lập trình file thực thi App.js**

Về phía Web Server, chúng ta cần đảm bảo nó có thể phục vụ cho nhiều Client, với path là:

/update thì sẽ thêm mới dữ liệu để lưu trữ, và in ra màn hình

/get trả về dữ liệu đã lưu trữ định dạng JSON

/ và còn lại thì trả về file index.html

Mảng dữ liệu lưu trữ có định dạng: [{"temp": 25, "humd":80, time: "time"}, ...]

**Mã nguồn server Nodejs sẽ có những lưu ý sau**

|  |  |
| --- | --- |
|  | require dùng để load các thư viện hoặc module cần thiết cho dự án  fs: Module giúp đọc file từ server hoặc upload file lên server  url: Chia nhỏ URL thành những thành phần để dễ dàng truy xuất  http: Phương thức truyền nhận dữ liệu dùng http  querystring: Module giúp chuyển string sang object  db = []: Biến kiểu mảng nhằm chứa dữ liệu nhiệt độ, độ ẩm |
|  | So sánh giá trị pathname để xử lí dữ liệu. Nếu pathname =/update thì sẽ tạo biến newData nhằm lấy dữ liệu client gửi lên thông qua URL, sau đó đẩy dữ liệu vào mảng db thông qua lệnh db.push(newData) , giá trị được hiển thị qua Log khi dùng lệnh console.log(newData). Hàm Date() giúp lấy thời gian hiện tại. |
|  | Trả về định dạng JSON ('Content-Type': 'application/json') của mảng db nếu pathname = /get, sau đó xóa giá trị của mảng. Hàm response.end() sẽ trả về HTTP code (mã 200 là kết quả OK) |
|  | Trả về nội dung của file index.html khi không xảy ra 2 trường hợp đầu. Dùng fs để đọc file index.html và gán nội dung vào content thông qua lệnh fs.readFile('./index.html', function(error, content). Hàm response.writeHead(200, {'Content-Type': 'text/html' }) nhằm khai báo mã HTTP code, định dạng trả về là HTML để đọc file |
|  | Khởi tạo một server HTTP và mở port 8000 và Port môi trường server để các client truy cập |

Dữ liệu nhiệt độ, độ ẩm được gửi lên Server sẽ có dạng sau:

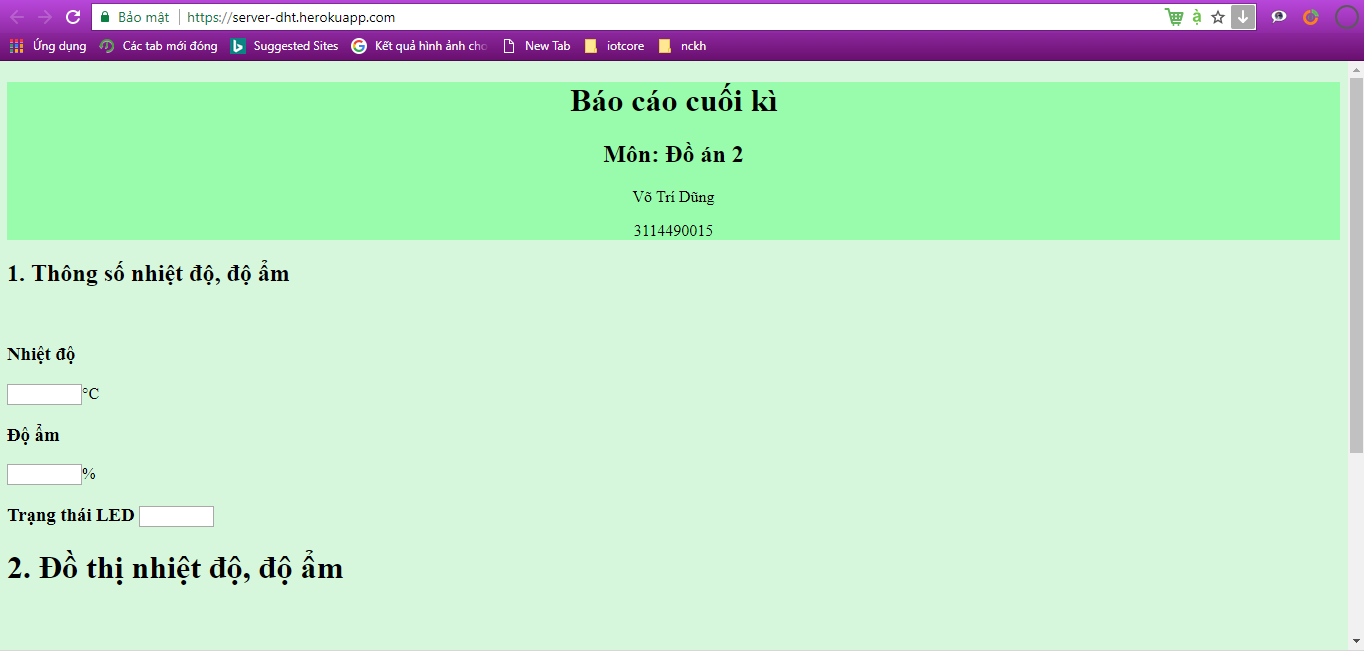
{ temp: '20', humd: '60', time: 2017-08-21T16:56:23.358Z }

{ temp: '20', humd: '60', time: 2017-08-21T16:57:06.277Z }

{ temp: '20', humd: '60', time: 2017-08-21T16:57:17.708Z }

### **2.2.2 Lập trình file Index.html**

Khi người dùng truy cập vào path index.html hoặc các trường hợp khác ngoại trừ (/update và /get) sẽ được server gửi về một file index.html. file này sẽ được trình duyệt web đọc và hiên thị ra giao diện web cho người dùng.



Hình 2.1 Giao diện web cho người dùng

Giao diện chính sẽ bao gồm 2 mục hiển thị dữ liệu:

Hiển thị thông số nhiệt độ, độ ẩm và trạng thái hiện tại của LED.

Hiển thị độ thị nhiệt độ độ ẩm ( thông số nhiệt độ, độ ẩm được cập nhật sau mỗi 2s).

Để hiển thị được đồ thị lên trình duyệt web, cần phải kết hợp canvas js chart bên trong file index.html.

Chúng ta sẽ lấy dữ liệu từ server gửi xuống và vẽ biểu đồ dùng mã Javascrpit, sử dụng tag <scrpit>code JS </scrpit> để chèn nội dung code Javascrpit vào file HTML.

Việc lấy dữ liệu được thực thi bằng hàm httpGetAsync(). Hàm này sử dụng đối tượng XMLHttpRequest để lấy dữ liệu từ server mà không cần phải load lại trang, dữ liệu xmlHttp.responseText lấy từ server tại địa chỉ localhost:8000/get ở định dạng JSON nên cần phải chuyển sang dạng Object bằng hàm JSON.parse().

Sử dụng window.onload = function() để load lại nội dung của graph, các lệnh trong hàm đã được giải thích trong code

### **2.2.3 Triển khai server lên Heroku**

Để người dùng internet có thể truy cập vào server, chúng ta sẽ triển khai toàn bộ server lên Heroku

**Heroku**

Heroku là dịch vụ cung cấp máy chủ miễn phí cho người dùng. Miễn phí sử dụng cùng với vô vàn các addons hỗ trợ cực kỳ hữu ích thì đấy được coi là một trong những dịch vụ hấp dẫn khó cưỡng. Dù miễn phí nhưng nó có thể so sanh với các server trả phí. Heroku hỗ trợ nhiều ngôn ngữ lập trình: NodeJS, Ruby, Python, PHP, Java, Scala, Clojure, Go, Kotlin

Ưu điểm:

* Miễn phí sử dụng.
* Hỗ trợ nhiều ngôn ngữ lập trình.
* Giúp người dùng có thể dễ dàng.
* Hỗ trợ Host và domain miễn phí.

Nhược điểm:

* Sau 2 đến 3 giờ nếu server không có người truy cập thì server sẽ chuyển sang trạng thái ngủ.
* Heroku chỉ cho người dùng 550 giờ mỗi tháng để sử dụng

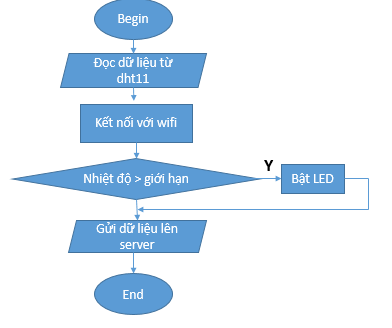
Để có thể chạy ứng dụng trên Heroku, cần sử dụng một file tên Procfile

web: node app.js

Procfile sẽ chỉ định Heroku chạy file thực thi mặc định là file app.js

## **2.3 Lập trình ESP8266**

### **2.3.1 Lưu đồ giải thuật**

****

Sơ đồ 2.1: Lưu đồ giải thuật

Khởi đầu chương trình sẽ kết nối với Wifi thông qua thư viện **ESP8266WiFiMulti.h.**

Ngoài ra, để đọc được cảm biến DHT11 cần phải sử dụng thêm thư viện **DHT.h**

Trong vòng lặp loop, chương trình sẽ thực hiện các nhiệm vụ:

* Đọc dữ liệu nhiệt độ, độ ẩm và so sánh với giá trị nhiệt độ, độ ẩm tối đa. Nếu giá trị này lớn hơn, động cơ sẽ được bật và ngược lại động cơ sẽ tắt.
* ESP8266 sẽ gửi dữ liệu lên server bao gồm: giá trị nhiệt độ, độ ẩm và trạng thái động cơ.

Để có thể kết nối với server http, em sử dụng thư viện **ESP8266HTTPClient.h**

Thông qua kết nối Wifi, thưc viện sẽ cấu hình ESP8266 là một client cho phép truy cập server http thông qua địa chỉ URL.

Địa chỉ URL để cập nhật dữ liệu có cấu trúc sau:

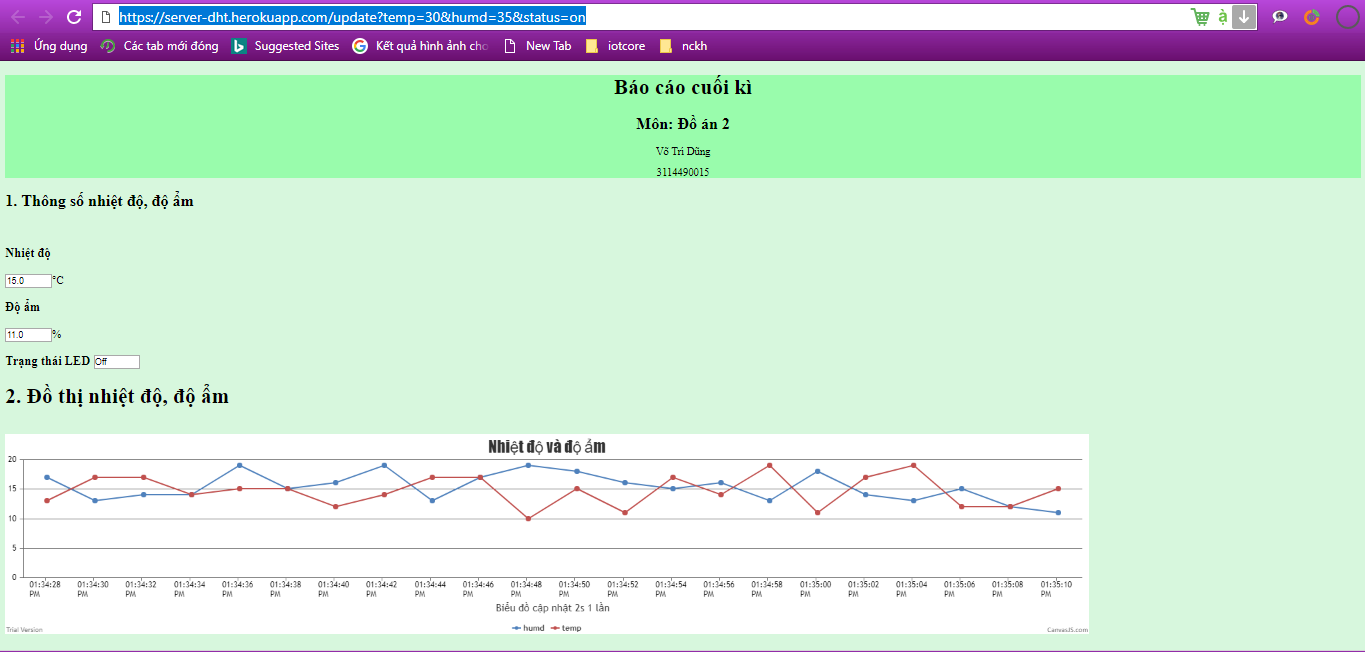
**https://server-dht.herokuapp.com/update?temp=30&humd=35&status=on**

Trong đó:

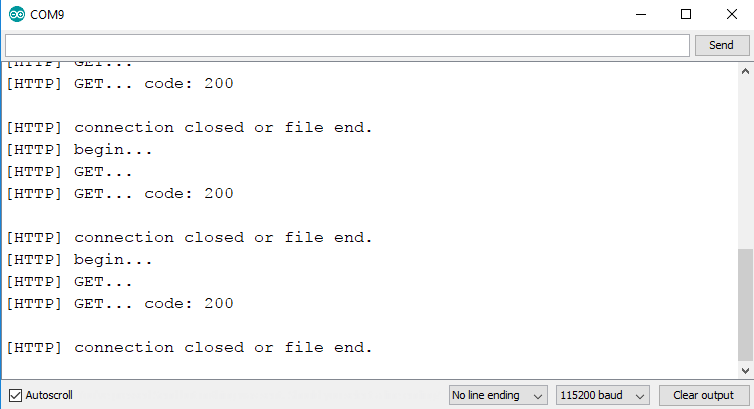
* **https**: là giao thức truyền.
* **server-dht.herokuapp.com:** địa chỉ server.
* **update? :** path, địa chỉ mà client muốn truy xuât.
* **temp=30&humd=35&status=on:** queryy, phần dữ liệu mà client gửi lên.

## **2.4 Chạy chương trình thực tế**

Sau khi tiến hành chạy server trên Heroku đồng thời nạp chương trình vào module ESP8266 thông qua Arduino IDE. Dữ liệu được gửi lên server và người dùng có thể xem trên trình duyetj web như sau:

****

Hình 2.2: Kết quả trên trình duyệt web

****

Hình 2.3: Thông tin trả về từ server

# CHƯƠNG 3: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

**Kết luận**

Sau một thời gian nghiên cứu, đồ án đã được xây dựng hoàn thiện. Hệ thống đáp ứng được nhu cầu cơ bản là đọc đươc dữ liệu từ cảm biến DHT11, người dùng có thể truy cập server bằng tình duyệt và xem được dữ liệu nhiệt độ, độ ẩm.

Hệ thống cung cấp một giao diện người dùng trên web trực quan, ngoài ra, người dùng có thể truy xuất dữ liệu theo định dạng JSON.

**Hạn chế**

Hệ thống chỉ đáp ứng nhu cầu cơ bản là theo dõi dữ liệu nhiệt độ, độ ẩm, ngoài ra chưa có tính năng điều khiển các thiết bị thông qua trình duyệt cũng như chưa có khả năng quản lý thiết bị theo tài khoản người dùng.

Hệ thống chưa áp dụng các cơ chế bảo mật cho dữ liệu.

**Hướng phát triển đề tài**

Phát triển hệ thông quản lý thiết bị theo tài khoản giúp người dùng có thể tạo tài khoản và quản lý các thiết bị của mình.

Ứng dụng cơ chế mã hóa , bảo mật dữ liệu, quản lý truy xuât thông qua Token giúp đảm bảo an toàn cho dữ liệu người dung.

Kết hợp thư viện Wifi manager giúp người dùng có thể cấu hình wifi mà không cần can thiệp vào Code.

# DANH MỤC TÀI LIỆU THAM KHẢO

[1] IoT Maker VN:

https://iotmakervn.github.io/iot-starter-book/ [2] Even’s weblog:

[2] [ESP8266HTTPClient](https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266HTTPClient):

https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266HTTPClient/src/ESP8266HTTPClient.h [4] Xuân Thiếp’s youtube video:

[3] [esp8266-oled-ssd1306](https://github.com/ThingPulse/esp8266-oled-ssd1306)**:**

<https://github.com/ThingPulse/esp8266-oled-ssd1306>

# PHỤ LỤC

**Code cho ESP8266**

#include <Arduino.h>

#include <ESP8266WiFi.h>

#include <ESP8266WiFiMulti.h>

#include <ESP8266HTTPClient.h>

#include "DHT.h"

#define Serial Serial

#define DHTPIN 2

#define DHTTYPE DHT11

int limTemp = 15;

int limHumd = 15;

ESP8266WiFiMulti WiFiMulti;

DHT dht(DHTPIN, DHTTYPE);

void setup() {

pinMode(16, OUTPUT);

Serial.begin(115200);

Serial.println();

Serial.println();

Serial.println();

for(uint8\_t t = 4; t > 0; t--) {

Serial.printf("[SETUP] WAIT %d...\n", t);

Serial.flush();

delay(1000);

}

WiFiMulti.addAP("Wifi", "quenpassroi");

WiFiMulti.addAP("Samsung", "12345678");

}

void loop() {

String ledsta;

float h = random(10,20);//dht.readHumidity();

float t = random(10,20);//dht.readTemperature();

if( t > limTemp ){

digitalWrite(16,HIGH);

ledsta = "On";

}

else{

digitalWrite(16,LOW);

ledsta = "Off";

}

String url = "http://server-dht.herokuapp.com/update?temp=" + String(t, 1) + "&humd=" + String(h, 1)+"&status="+ledsta;

if((WiFiMulti.run() == WL\_CONNECTED)) {

updateServer(url);

}

delay(1000);

}

void updateServer(String str){

HTTPClient http;

Serial.print("[HTTP] begin...\n");

http.begin(str);

Serial.print("[HTTP] GET...\n");

int httpCode = http.GET();

if(httpCode > 0) {

Serial.printf("[HTTP] GET... code: %d\n", httpCode);

if(httpCode == HTTP\_CODE\_OK) {

int len = http.getSize();

uint8\_t buff[128] = { 0 };

WiFiClient \* stream = http.getStreamPtr();

while(http.connected() && (len > 0 || len == -1)) {

size\_t size = stream->available();

if(size) {

int c = stream->readBytes(buff, ((size > sizeof(buff)) ? sizeof(buff) : size));

// write it to Serial

Serial.write(buff, c);

if(len > 0) {

len -= c;

}

}

delay(1);

}

Serial.println();

Serial.print("[HTTP] connection closed or file end.\n");

}

} else {

Serial.printf("[HTTP] GET... failed, error: %s\n", http.errorToString(httpCode).c\_str());

}

http.end();

}

**Code server node.js**

var fs = require('fs');

var url = require('url');

var http = require('http');

var querystring = require('querystring');

var db = []; //database

//---------------------------------------------------------------------------------------------

// function gửi yêu cầu(response) từ phía server hoặc nhận yêu cầu (request) của client gửi lên

function requestHandler(request, response) {

// Giả sử địa chỉ nhận được http://192.168.1.7:8000/update?temp=30&humd=40

var uriData = url.parse(request.url);

var pathname = uriData.pathname; // /update?

var query = uriData.query; // temp=30.5&hum=80

var queryData = querystring.parse(query); // queryData.temp = 30.5, queryData.humd = 40

//-----------------------------------------------------------------------------------------

if (pathname == '/update') {

var newData = {

temp: queryData.temp,

humd: queryData.humd,

ledsta:queryData.status,

time: new Date()

};

db.push(newData);

console.log(newData);

response.end();

//-----------------------------------------------------------------------------------------

} else if (pathname == '/get') {

response.writeHead(200, {

'Content-Type': 'application/json'

});

response.end(JSON.stringify(db));

db = [];

//-----------------------------------------------------------------------------------------

} else {

fs.readFile('./index.html', function(error, content) {

response.writeHead(200, {

'Content-Type': 'text/html'

});

response.end(content);

});

}

//-----------------------------------------------------------------------------------------

}

var server = http.createServer(requestHandler);

server.listen(process.env.PORT || 3000);

console.log('Server listening on port 8000');

**Code html**

<!DOCTYPE html>

<html>

<head>

<style>

body {

background-color: #d7f7dd;

}

center {

background-color: #99fcac;

}

</style>

<meta charset="UTF-8">

<title>DHT11</title>

<!-- Nhúng file Javasript tại đường dẫn src để có thể xây dựng 1 graph -->

<script type="text/javascript" src="https://canvasjs.com/assets/script/canvasjs.min.js"></script>

</head>

<body>

<center style="background-color:f222f1;">

<h1> Báo cáo cuối kì </h1>

<h2> Môn: Đồ án 2 </h2>

<p>Võ Trí Dũng<p>

<p>3114490015</p>

</center>

<h2> 1. Thông số nhiệt độ, độ ẩm</h2><br>

<h3> Nhiệt độ</h3> <input type="text" size="6" id="temp">&#176;C<br>

<h3> Độ ẩm</h3> <input type="text" size="6" id="humd">%<br>

<h3>Trạng thái LED

<input type="text" size="6" id="ledsta"><br>

</h3>

<h1> 2. Đồ thị nhiệt độ, độ ẩm</h1><br>

<!-- thiết lập kích thước cho graph thông qua id ChartContainer đã thiết lập ở trên -->

<div id="ChartContainer" style="height: 300px; width:80%;"></div>

<script type="text/javascript">

function httpGetAsync(theUrl, callback) {

var xmlHttp = new XMLHttpRequest();

xmlHttp.onreadystatechange = function() {

if (xmlHttp.readyState == 4 && xmlHttp.status == 200)

callback(JSON.parse(xmlHttp.responseText));

}

xmlHttp.open("GET", theUrl, true); // true for asynchronous

xmlHttp.send(null);

}

window.onload = function() {

var dataTemp = [];

var dataHumd = [];

var Chart = new CanvasJS.Chart("ChartContainer", {

zoomEnabled: true, // Dùng thuộc tính có thể zoom vào graph

title: {

text: "Nhiệt độ và độ ẩm" // Viết tiêu đề cho graph

},

toolTip: { // Hiển thị cùng lúc 2 trường giá trị nhiệt độ, độ ẩm trên graph

shared: true

},

axisX: {

title: "Biểu đồ cập nhật 2s 1 lần" // Chú thích cho trục X

},

data: [{

// Khai báo các thuộc tính của dataTemp và dataHumd

type: "line", // Chọn kiểu dữ liệu đường

xValueType: "dateTime", // Cài đặt kiểu giá trị tại trục X là thuộc tính thời gian

showInLegend: true, // Hiển thị "temp" ở mục chú thích (legend items)

name: "humd",

dataPoints: dataHumd // Dữ liệu hiển thị sẽ lấy từ dataTemp

},

{

type: "line",

xValueType: "dateTime",

showInLegend: true,

name: "temp",

dataPoints: dataTemp

}

],

});

var yHumdVal = 0; // Biến lưu giá trị độ ẩm (theo trục Y)

var yTempVal = 0; // Biến lưu giá trị nhiệt độ (theo trục Y)

var updateInterval = 2000; // Thời gian cập nhật dữ liệu 2000ms = 2s

var time = new Date(); // Lấy thời gian hiện tại

var updateChart = function() {

httpGetAsync('/get', function(data) {

// Gán giá trị từ localhost:8000/get vào textbox để hiển thị

document.getElementById("temp").value = data[0].temp;

document.getElementById("humd").value = data[0].humd;

document.getElementById("ledsta").value = data[0].ledsta;

// Xuất ra màn hình console trên browser giá trị nhận được từ localhost:8000/get

console.log(data);

// Cập nhật thời gian và lấy giá trị nhiệt độ, độ ẩm từ server

time.setTime(time.getTime() + updateInterval);

yTempVal = parseInt(data[0].temp);

yHumdVal = parseInt(data[0].humd);

dataTemp.push({ // cập nhât dữ liệu mới từ server

x: time.getTime(),

y: yTempVal

});

dataHumd.push({

x: time.getTime(),

y: yHumdVal

});

Chart.render(); // chuyển đổi dữ liệu của của graph thành mô hình đồ họa

});

};

updateChart(); // Chạy lần đầu tiên

setInterval(function() { // Cập nhật lại giá trị graph sau thời gian updateInterval

updateChart()

}, updateInterval);

}

</script>

</body>

</html>

**Code Procfile**

web: node app.js