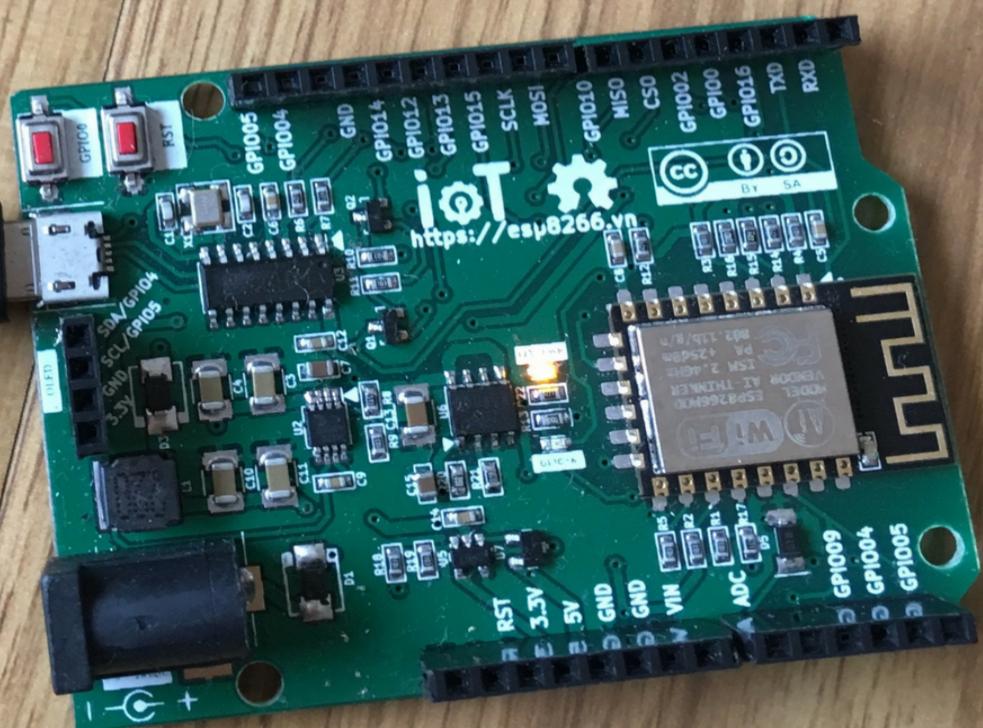


BUILD AMAZING INTERNET
OF THINGS PROJECTS

INTERNET OF THINGS

Cho người mới bắt đầu



IOT ACADEMY

INTERNET OF THINGS (IOT)

cho người mới bắt đầu

IoT Academy

MỤC LỤC

Lời mở đầu	1
Thuật ngữ hay sử dụng	1
Giải thích code trong bài	1
Giới thiệu nội dung	2
Ai có thể sử dụng?	2
Mục tiêu mang lại cho người đọc	3
Chuẩn bị	3
Kiến thức cơ bản	4
Internet Of Things (IoT)	5
Internet of Things (IoT) là gì?	5
Những ứng dụng thực tế trong cuộc sống	7
ESP8266	8
Sơ đồ chân	8
Thông số phần cứng	9
SDK hỗ trợ chính thức từ hãng	9
ESP8285	10
Module và Board mạch phát triển	10
Arduino là gì?	12
Một số đặc điểm của Arduino	12
Các lợi ích khi sử dụng Arduino	13
Cộng đồng Arduino trên thế giới	13
Arduino cho ESP8266 & board mạch ESP8266 WiFi Uno	13
IoT Starter Kit	15
Hình ảnh	15
Breadboard	16
Thông tin	16
Node.js	18
Lý do sử dụng Node.js trong cuốn sách này	18
Cuốn sách này có hướng dẫn Javascript?	19
Sublime Text	20
Cài đặt và chuẩn bị	21
Arduino IDE	21
Cài đặt thư viện Arduino	22
USB CDC driver	23
Chọn Board ESP8266 WiFi Uno trong Arduino IDE	25
Serial Terminal	26
Node.js	26
Sublime Text	26
Git	26
Tổng kết	27

Hello World	28
Chớp tắt bóng LED	29
Kiến thức	29
Đầu nối	30
Mã nguồn chớp tắt dùng Delay.....	30
Mã nguồn chớp tắt dùng định thời	31
Digital IO	31
Tổng kết.....	32
Nút nhấn	33
Kiến thức	33
Mã nguồn dùng hỏi vòng.....	33
Mã nguồn dùng ngắn.....	34
Các khái niệm.....	35
OLED	36
Màn hình OLED.....	36
Màn hình OLED SSD1306	36
Giao tiếp I2C	36
Hiển thị màn hình OLED với ESP8266.....	37
Tổng kết.....	39
ESP8266 WiFi	40
Chế độ WiFi Station	42
Kiến thức	42
Kết nối vào mạng WiFi nội bộ	42
Sử dụng WiFiMulti	43
HTTP Client	45
Giao thức HTTP	45
JSON	48
Ứng dụng xem giá Bitcoin	49
Chế độ WiFi Access Point	51
ESP8266 hoạt động ở chế độ Access Point	51
Tạo ra một mạng WiFi sử dụng ESP8266.....	51
Web Server	52
Web Server là gì?	52
HTML - Javascript - CSS	52
Ứng dụng điều khiển đèn LED thông qua Webserver	54
Code	54
Trao đổi dữ liệu giữa 2 ESP8266	56
Yêu cầu	56
Hướng dẫn thực hiện.....	56
Những vướng mắc thường gặp	56
Tổng kết	57
Dự án đọc cảm biến DHT11 và gởi về Server	58
Thiết kế ứng dụng.....	59
Yêu cầu	60

Phân tích	60
Kiến thức	60
Thực hiện.....	62
Server Nodejs	63
Code ESP8266.....	69
Chuẩn bị.....	69
Tổng kết.....	71
Ứng dụng mở rộng	72
Dùng ESP8266 như 1 Web Server	72
Các chế độ cấu hình WiFi	73
Smartconfig.....	74
Kiến thức.....	74
Code	75
WPS.....	77
WPS là gì?.....	77
Thực hiện WPS với ESP8266	77
Code.....	77
Wifi Manager.....	79
Hoạt động của WifiManager	79
Chuẩn bị	79
Code	79
MQTT	81
MQTT là gì?	82
Publish, subscribe	82
QoS	82
Retain	83
LWT	83
MQTT Client.....	84
MQTT Broker.....	85
Ứng dụng thực tế	86
Websocket.....	87
Websocket là gì?.....	88
Ưu điểm	88
Nhược điểm	88
Ứng dụng Websocket	89
Yêu cầu	89
Chuẩn bị.....	89
Đoạn code Javascript để tạo kết nối Web Socket	89
Nhúng file HTML chứa đoạn code JS vào ESP8266	89
Cheatsheet.....	90
C - Cheatsheet.....	91
Arduino - Cheatsheet.....	92
Tổng kết.....	95

LỜI MỞ ĐẦU

Internet Of Things (IoT) – **Internet vạn vật** dường như đang đứng trước một bước ngoặt để đi đến giai đoạn tiếp theo cho một thế giới hiện đại, văn minh. Đó là viễn cảnh mà mọi vật đều có thể kết nối với nhau thông qua Internet không dây. Các doanh nghiệp đang có xu hướng ứng dụng sản phẩm công nghệ IoT vào sản xuất ngày càng nhiều bởi thị trường sáng tạo tiềm năng và chi phí sản xuất ngày càng thấp.

Chứng kiến sự phát triển như vũ bão của các sản phẩm ứng dụng công nghệ IoT và thị trường công nghệ Start up tiềm năng đang ngày càng sôi động hơn bao giờ hết, quyển sách này cung cấp các nội dung về IoT với triết lí Không chỉ là thực tế – không rời rạc, hướng đến những người trẻ tuổi đã, đang và muốn tập trung năng lực của mình cho không gian Internet Of Things. Mong muốn cho ra đời những sản phẩm độc đáo, sáng tạo, ngày càng hoàn thiện và đồng bộ để có thể đáp ứng nhu cầu của cuộc sống. Nội dung được thiết kế một cách cơ bản giúp học viên có cái nhìn tổng quan về việc xây dựng hệ thống, sản xuất thiết bị và dễ dàng tham gia vào lĩnh vực IoT mới mẻ.

THUẬT NGỮ HAY SỬ DỤNG

- **IoT** Internet Of Things hay internet vạn vật
- **ESP8266** chip khả trình tích hợp thu phát WiFi
- **Git** trình quản lý phiên bản
- **Github** mạng xã hội lập trình viên
- **IDE** Integrated Development Environment - môi trường phát triển tích hợp
- **Compiler** trình biên dịch
- **Library** thư viện
- **Logic Level** mức điện áp để chip hiểu được (1 hay 0)

GIẢI THÍCH CODE TRONG BÀI

```
void setup()
{
    //comment ①
    int a = 1;
    a++; ②
}
```

① Dòng này giản thícch đây là comment

② Dòng này giải thícch biến **a** tăng thêm 1 đơn vị

GIỚI THIỆU NỘI DUNG

Nội dung quyển sách này sẽ hướng dẫn chi tiết cho người đọc lập trình ứng dụng cho Chip WiFi ESP8266 kết nối với Server, gởi dữ liệu, và thực thi các lệnh từ Server. Từ đó người đọc có thể có những kiến thức cơ bản, dùng nó để xây dựng các ứng dụng Internet Of Things thực tế.

Toàn bộ nội dung biên tập sử dụng phần cứng là System On Chip (SoC) **ESP8266** - có khả năng kết nối WiFi và lập trình được giá rẻ, phổ biến trên thế giới. Board mạch sử dụng là board phần cứng mở [IoT WiFi Uno](#) có sơ đồ chân tương thích với các board Arduino Uno.

Phần mềm sử dụng lập trình máy tính là [Arduino](#), ngôn ngữ lập trình [C/C++](#)

Một chút về Server sử dụng [NodeJS](#) với ngôn ngữ lập trình [Javascript](#)

Bạn sẽ cần tìm hiểu một số công cụ và khái niệm được sử dụng trong quyển sách này như sau:

- **Git** Trình quản lý phiên bản sử dụng rất rộng rãi trên thế giới, [Github](#) là một mạng xã hội cho lập trình viên dựa trên Git. Git giúp bạn quản lý được mã nguồn, làm việc nhóm, xử lý các thao tác hợp nhất, lịch sử mã nguồn ... Có thể trong quá trình làm việc với quyển sách này, bạn sẽ cần sử dụng các thư viện mã nguồn mở cho Arduino từ Github, nên việc cài đặt và sử dụng công cụ khá cần thiết cho việc đó. Chưa kể, nó sẽ giúp bạn quản lý mã nguồn và dự án ngày càng chuyên nghiệp hơn.
- **Sublime Text** là một trình soạn thảo phổ biến, nhanh, nhẹ và nhiều tính năng hay. Sử dụng để lập trình Javascript (NodeJS)
- Code formater
- Editorconfig



Bạn có thể sử dụng bất kỳ board ESP8266 nào khác trên thị trường cho cuốn sách này, ví dụ như: [NODEMCU](#), [Wemos](#), ...



Ngoài nội dung trong cuốn sách này, thì còn nhiều dự án mẫu cũng như các hướng dẫn trực tuyến tại [arduino.esp8266.vn](#)



Tất cả các phần Code đều không giải thích rõ chi tiết API cho mỗi tính năng. Mà thay vào đó được cung cấp tại phụ lục Cheat Sheet (Arduino và C)

AI CÓ THỂ SỬ DỤNG?

- Các lập trình viên phần mềm/Mobile App, Web App... muốn tham gia làm sản phẩm IoT
- Sinh viên muốn nâng cao kỹ năng, bổ sung kiến thức

- Cá nhân muốn tự mình làm các sản phẩm phục vụ cuộc sống - phục vụ công việc
- Startup Tech không chuyên về phần cứng hoặc phần mềm

MỤC TIÊU MANG LẠI CHO NGƯỜI ĐỌC

- Giúp cho người không chuyên về phần cứng tiếp cận để làm sản phẩm IoT dễ dàng
- Có thể tự phát triển hệ thống tích hợp cho sản phẩm IoT
- Hiểu biết về quy trình tạo ra sản phẩm phần cứng, đi vào mảng sản xuất thiết bị
- Tránh những sai sót không đáng có khi phát triển và thiết kế hệ thống

CHUẨN BỊ

- 1 bộ IoT Starter Kit bao gồm các cảm biến, cơ cấu + ESP8266 WiFi Uno
- 1 máy tính cá nhân (Windows, MacOS hoặc Linux)
- Tài liệu hướng dẫn nội bộ, được phát qua mỗi bài học
- Arduino ESP8266 Cheatsheet (Mục lục cuối quyển sách này)

KIẾN THỨC CƠ BẢN

Chúng ta sẽ bắt đầu bằng việc tìm hiểu tổng quan về hệ thống IoT, tổng quan về dòng chip **ESP8266**, rồi đến việc cài đặt công cụ phát triển **Arduino** trên máy tính của bạn. Tiếp đến là việc biên dịch các dự án mẫu, lựa chọn trình thư viện, trình soạn thảo sẽ làm việc. Kết thúc chương này chúng ta nên có được cái nhìn tổng quát về hệ thống IoT, làm thế nào và sử dụng công cụ gì để lập trình ứng dụng với ESP8266.

Điểm qua phần này như sau:

- IoT và ứng dụng thực tế
- Tìm hiểu về chip WiFi khả trình **ESP8266**
- Arduino IDE và sử dụng Arduino với ESP8266
- Starter Kit bộ công cụ khởi động việc học lập trình IoT
- Node.js - Javascript ngôn ngữ lập trình Server Side
- Cài đặt tất cả các công cụ

Với những ai đã từng hiểu rõ ESP8266, đã từng làm về hệ thống IoT, đã chuyên nghiệp trong lập trình C/C++ có thể bỏ qua chương này.

INTERNET OF THINGS (IOT)

INTERNET OF THINGS (IOT) LÀ GÌ?

Internet of Things (IoT) - Mạng lưới vạn vật kết nối Internet là một kịch bản của thế giới, khi mà mỗi đồ vật, con người được cung cấp một định danh của riêng mình, và tất cả có khả năng truyền tải, trao đổi thông tin, dữ liệu qua một mạng duy nhất mà không cần đến sự tương tác trực tiếp giữa người với người, hay người với máy tính. IoT đã phát triển từ sự hội tụ của công nghệ không dây, công nghệ vi cơ điện tử và Internet[1]. Nói đơn giản là một tập hợp các thiết bị có khả năng kết nối với nhau, với Internet và với thế giới bên ngoài để thực hiện một công việc nào đó. Link: vi.wikipedia.org/wiki/Mạng_lưới_vạn_vật_kết_nối_Internet

– Wikipedia

Internet of things (IoT) dùng để chỉ các đối tượng có thể được nhận biết cũng như chỉ sự tồn tại của chúng trong một kiến trúc tổng hòa mang tính kết nối: Mạng lưới vạn vật kết nối Internet, hay gọi đơn giản hơn là **Things**.

IoT có thể là bộ cảm ứng được lắp ráp trong một chiếc tủ lạnh để ghi lại nhiệt độ, là một trái tim được cấy ghép trong cơ thể con người,... Hiểu đơn giản, IoT có thể khiến mọi vật giờ đây có thể giao tiếp với nhau dễ dàng hơn và ưu điểm lớn nhất của "**Thông minh**" là khả năng phòng ngừa và cảnh báo tại bất kì đâu.

Cụm từ Internet of things được đưa ra bởi Kevin Ashton vào năm 1999, tiếp sau đó nó cũng được dùng nhiều trong các ấn phẩm đến từ các hãng và nhà phân tích. Họ cho rằng IoT là một hệ thống phức tạp, bởi nó là một lượng lớn các đường liên kết giữa máy móc, thiết bị và dịch vụ với nhau. Ban đầu, IoT không mang ý nghĩa tự động và thông minh. Về sau, người ta đã nghĩ đến khả năng kết hợp giữa hai khái niệm IoT - Autonomous control lại với nhau. Nó có thể quan sát sự thay đổi và phản hồi với môi trường xung quanh, cũng có thể tự điều khiển bản thân mà không cần kết nối mạng. Việc tích hợp trí thông minh vào IoT còn có thể giúp các thiết bị, máy móc, phần mềm thu thập và phân tích các dữ liệu điện tử của con người khi chúng ta tương tác với chúng. Xu hướng tất yếu trong tương lai, con người có thể giao tiếp với máy móc chỉ qua mạng internet không dây mà không cần thêm bất cứ hình thức trung gian nào khác.

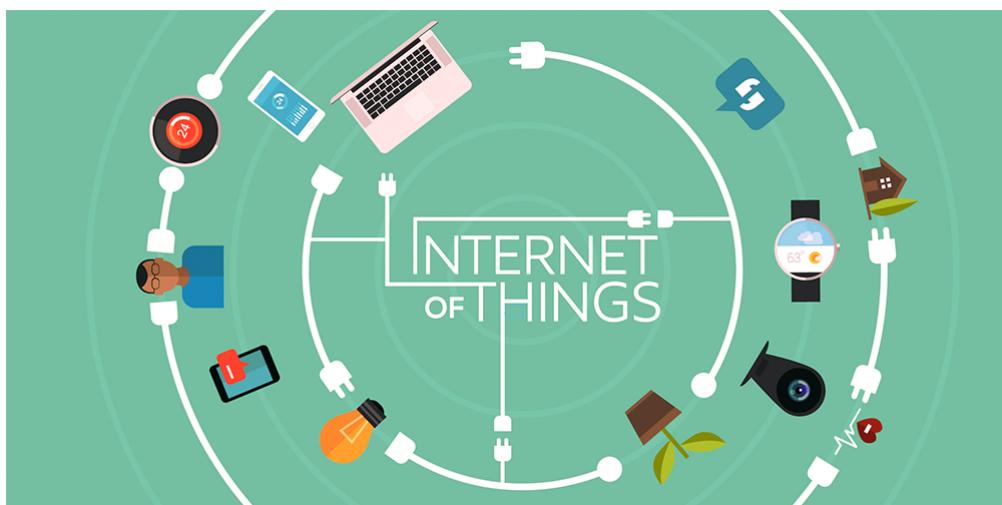
Câu hỏi đặt ra là, điều gì giúp IoT "thông minh" và "hiểu" con người? Ban đầu, người ta cho rằng Internet của vạn vật chủ yếu xoay quanh giao tiếp M2M (các thiết bị kết nối với nhau thông qua một thiết bị khác điều khiển). Nhưng khi hướng đến sự "thông minh hóa", đó không chỉ là giao tiếp giữa M2M nữa mà cần phải đề cập đến các cảm biến (sensor). Và cũng đừng lầm tưởng rằng Sensor là

một cỗ máy hoạt động dưới sự vận hành của các thiết bị khác mà thực chất, nó tương tự như đôi mắt và đôi tai của loài người với sự ghi nhận liên tục những đo lường, định lượng, thu thập dữ liệu từ thế giới bên ngoài. Suy cho cùng, Internet of things đem đến sự kết nối giữa máy móc và cảm biến, và nhờ đến dữ liệu điện toán đám mây để mã hóa dữ liệu. Những ứng dụng điện toán đám mây là mắt xích quan trọng giúp cho Internet of things có thể hoạt động nhờ sự phân tích, xử lý và sử dụng dữ liệu mà các cảm biến thu thập được.

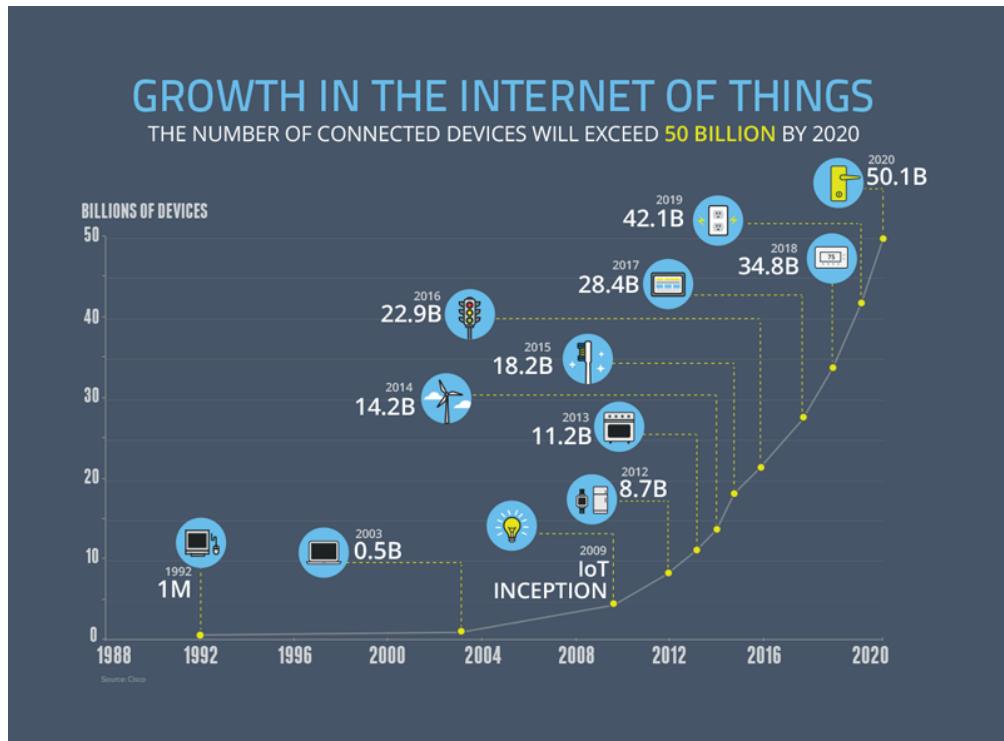
Tình hình trên thế giới hiện nay, tác động của IoT rất đa dạng và tích cực ở nhiều lĩnh vực: quản lý hạ tầng, y tế, xây dựng và tự động hóa, giao thông.... John Chambers (CEO của Cisco) đã công bố: Cho đến năm 2024 sẽ có 500 tỷ thiết bị được kết nối. Thực tế, con số này lớn hơn gần 100 lần số người trên Trái đất, điều đó cho thấy "vạn vật" nhiều hơn con người rất nhiều. Chúng ta đều biết ứng dụng IoT có thể "nói chuyện" với con người thông qua bàn phím, thiết bị cũng được thiết kế ngày càng hoàn thiện với nhiều cảm biến hơn để có thể giao tiếp một cách nhanh nhất và chính xác nhất với con người, thu thập dữ liệu đơn giản từ mỗi người chúng ta. Nhưng quan trọng nhất, tuy giao tiếp với con người nhưng ứng dụng IoT không phải là con người.

Người ta cho rằng, IoT là chìa khóa của sự thành công, là bước ngoặt và cơ hội lớn của tương lai. Để không bị tụt lại phía sau, các chính phủ và doanh nghiệp cần có sự đổi mới và đầu tư mạnh tay hơn để phát triển các sản phẩm ứng dụng công nghệ Internet of things.

Các hashtag: #IoT #InternetOfThings



Hình 1. Hình minh họa



Hình 2. Sự phát triển của iot dự đoán đến năm 2020

► <https://www.youtube.com/watch?v=M8FKl0kRuVM> (YouTube video)

Video khái niệm về IoT

NHỮNG ỨNG DỤNG THỰC TẾ TRONG CUỘC SỐNG

► <https://www.youtube.com/watch?v=NjYTzvAVozo> (YouTube video)

Smarthome

► <https://www.youtube.com/watch?v=ZteX4BI46nc> (YouTube video)

Smart wifi video car

► <https://www.youtube.com/watch?v=c6MsG3olehY> (YouTube video)

Smart light

► <https://www.youtube.com/watch?v=Br5aJa6MkBc> (YouTube video)

Smart city

ESP8266

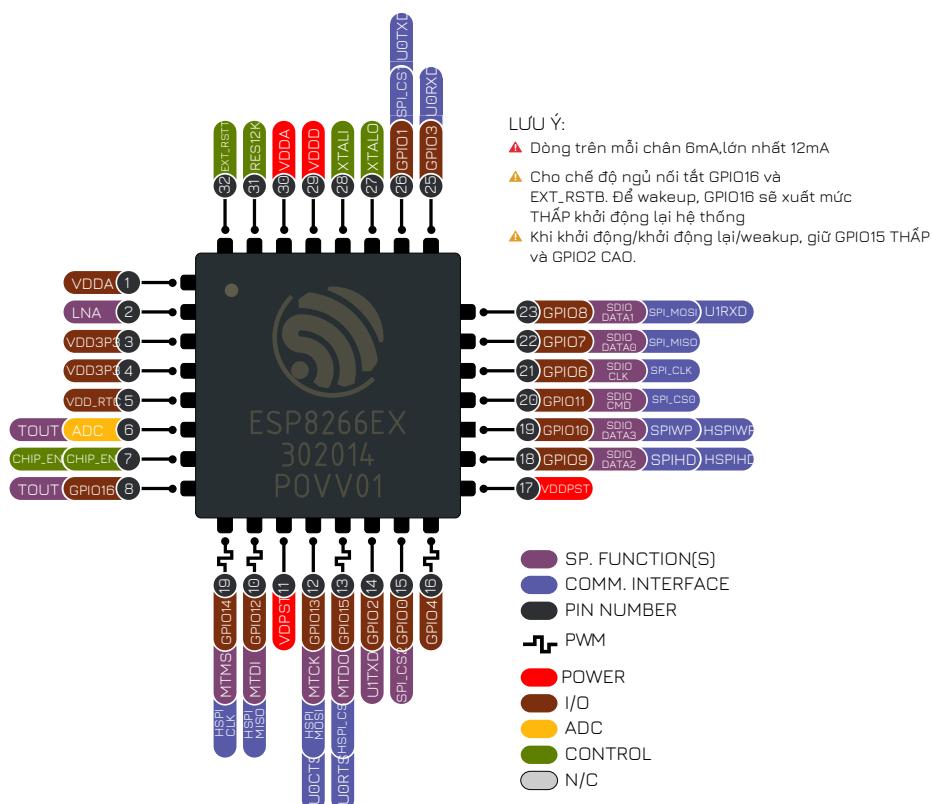
ESP8266 là dòng chip tích hợp Wi-Fi 2.4Ghz có thể lập trình được, rẻ tiền được sản xuất bởi một công ty bán dẫn Trung Quốc: Espressif Systems.

Được phát hành đầu tiên vào tháng 8 năm 2014, đóng gói đưa ra thị trường dạng Mô dun ESP-01, được sản xuất bởi bên thứ 3: AI-Thinker. Có khả năng kết nối Internet qua mạng Wi-Fi một cách nhanh chóng và sử dụng rất ít linh kiện đi kèm. Với giá cả có thể nói là rất rẻ so với tính năng và khả năng ESP8266 có thể làm được.

ESP8266 có một cộng đồng các nhà phát triển trên thế giới rất lớn, cung cấp nhiều Module lập trình mã mở giúp nhiều người có thể tiếp cận và xây dựng ứng dụng rất nhanh.

Hiện nay tất cả các dòng chip ESP8266 trên thị trường đều mang nhãn ESP8266EX, là phiên bản nâng cấp của ESP8266

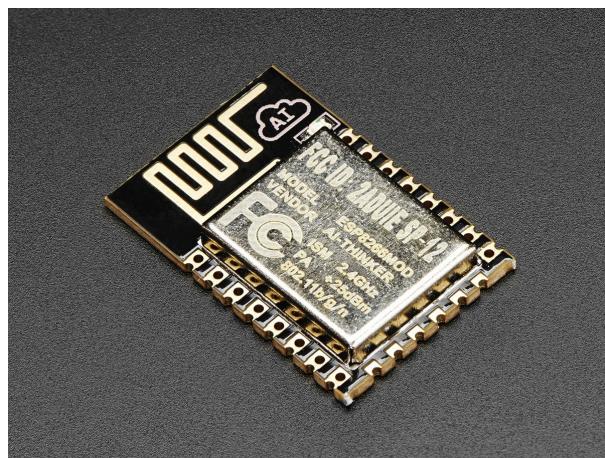
SƠ ĐỒ CHÂN



Hình 3. Sơ đồ chân ESP8266EX

THÔNG SỐ PHẦN CỨNG

- 32-bit RISC CPU : Tensilica Xtensa LX106 running at 80 MHz
- Hỗ trợ Flash ngoài từ 512KiB đến 4MiB
- 64KBytes RAM thực thi lệnh
- 96KBytes RAM dữ liệu
- 64KBytes boot ROM
- Chuẩn wifi EEE 802.11 b/g/n, Wi-Fi 2.4 GHz
 - Tích hợp TR switch, balun, LNA, khuếch đại công suất và matching network
 - Hỗ trợ WEP, WPA/WPA2, Open network
- Tích hợp giao thức TCP/IP
- Hỗ trợ nhiều loại anten
- 16 chân GPIO
- Hỗ trợ SDIO 2.0, UART, SPI, I²C, PWM,I²S với DMA
- 1 ADC 10-bit
- Dải nhiệt độ hoạt động rộng : -40C ~ 125C



Hình 4. Một module tích hợp phổ biến (Module ESP12E)

SDK HỖ TRỢ CHÍNH THỨC TỪ HÃNG

Espressif hiện đã hỗ trợ 3 nền tảng SDK (Software Development Kit - Gói phát triển phần mềm) độc lập, là: **NONOS SDK**, **RTOS SDK** và **Arduino**. Cả 3 đều có những ưu điểm riêng phù hợp với từng ứng dụng nhất định, và sử dụng chung nhiều các hàm điều khiển phần cứng. Hiện nay **Arduino** đang được sử dụng rộng rãi bởi tính dễ sử dụng, kiến trúc phần mềm tốt và tận dụng được nhiều thư viện.

cộng đồng

ESP8266 NONOS SDK

Hiện nay, **NONOS SDK** phiên bản từ **2.0.0** trở lên đã ổn định và cung cấp gần như là đầy đủ tất cả các tính năng mà ESP8266 có thể thực hiện:

- Các API cho Timer, System, Wifi, đọc ghi SPI Flash, Sleep và các Module phần cứng: GPIO, SPI, I²C, PWM, I²S với DMA.
- **Smartconfig**: Hỗ trợ cấu hình thông số Wi-Fi cho ESP8266 nhanh chóng.
- **Sniffer** API: Bắt các gói tin trong mạng không dây 2.4Ghz.
- **SNTP** API: Đồng bộ thời gian với Máy chủ thời gian.
- **WPA2 Enterprise** API: Cung cấp việc quản lý kết nối Wi-Fi bằng tài khoản sử dụng các máy chủ RADIUS.
- **TCP/UDP** API: Cho kết nối internet, và hỗ trợ các Module dựa trên, như: HTTP, MQTT, CoAP.
- **mDNS** API: Giúp tìm ra IP của thiết bị trong mạng nội bộ bằng tên [hostname].
- **MESH** API: Liên kết các module ESP8266 với cấu trúc mạng MESH
- **FOTA** API: Firmware Over The Air - cập nhật firmware từ xa cho thiết bị.
- **ESP-Now** API: Sử dụng các gói tin Wireless 2.4GHz trao đổi trực tiếp với ESP8266 khác mà không cần kết nối tới Access Point.
- **Simple Pair** API: Thiết lập kết nối bảo mật giữa 2 thiết bị tự động.

ESP8266 RTOS SDK

RTOS SDK sử dụng **FreeRTOS** làm nền tảng, đồng thời hầu hết các API của **NON OS SDK** đều có thể sử dụng với **RTOS SDK**.

ESP8285

ESP8285 là một phiên bản khác sau này của ESP8266EX, giống hoàn toàn ESP8266EX ngoại trừ việc thay vì dùng SPI FLASH bên ngoài thì ESP8285 tích hợp 1MiB Flash bên trong giúp giảm diện tích phần cứng và đơn giản hóa quá trình sản xuất.

MODULE VÀ BOARD MẠCH PHÁT TRIỂN

ESP8266 cần ít nhất thêm 7 linh kiện nữa mới có thể hoạt động, trong đó phần khó nhất là Antena.

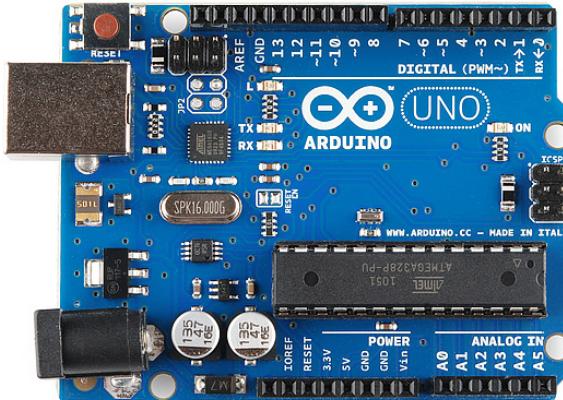
Đòi hỏi phải được sản xuất, kiểm tra với các thiết bị hiện đại. Do đó, trên thị trường xuất hiện nhiều Module và Board mạch phát triển đảm đương hết để người dùng đơn giản nhất trong việc phát triển ứng dụng. Một số Module và Board phát triển phổ biến:

Bảng 1. Một số module ESP8266 trên thị trường

Tên	Số chân	Pitch	LEDs	Antenna	Shielded	Dimensions
ESP-01	6	0.1"	Yes	PCB	No	14.3 × 24.8
ESP-02	6	0.1"	No	U-FL	No	14.2 × 14.2
ESP-03	10	2mm	No	Ceramic	No	17.3 × 12.1
ESP-04	10	2mm	No	None	No	14.7 × 12.1
ESP-05	3	0.1"	No	U-FL	No	14.2 × 14.2
ESP-06	11	misc	No	None	Yes	14.2 × 14.7
ESP-07	14	2mm	Yes	Ceramic+U-FL	Yes	20.0 × 16.0
ESP-08	10	2mm	No	None	Yes	17.0 × 16.0
ESP-09	10	misc	No	None	No	10.0 × 10.0
ESP-10	3	2mm	No	None	No	14.2 × 10.0
ESP-11	6	0.05"	No	Ceramic	No	17.3 × 12.1
ESP-12	14	2mm	Yes	PCB	Yes	24.0 × 16.0
ESP-12E	20	2mm	Yes	PCB	Yes	24.0 × 16.0
ESP-12F	20	2mm	Yes	PCB	Yes	24.0 × 16.0
ESP-13	16	1.5mm	No	PCB	Yes	18.0 × 20.0
ESP-14	22	2mm	No	PCB	Yes	24.3 × 16.2

Xem thêm esp8266.vn/introduction/esp-module/

ARDUINO LÀ GÌ?



Hình 5. Board mạch Arduino

Arduino là một IDE tích hợp sẵn editor, compiler, programmer và đi kèm với nó là các firmware có bootloader, các bộ thư viện được xây dựng sẵn và dễ dàng tích hợp. Ngôn ngữ sử dụng là C/C. **Tất cả** đều **opensource** và được đóng góp, phát triển hàng ngày bởi cộng đồng. Triết lý thiết kế và sử dụng của Arduino giúp cho người mới, không chuyên rất dễ tiếp cận, các công ty, hardware dễ dàng tích hợp. Tuy nhiên, với trình biên dịch C/C và các thư viện chất lượng được xây dựng bên dưới thì mức độ phổ biến ngày càng tăng và hiệu năng thì không hề thua kém các trình biên dịch chuyên nghiệp cho chip khác.

Đại diện cho **Arduino** ban đầu là chip AVR, nhưng sau này có rất nhiều nhà sản xuất sử dụng các chip khác nhau như **ARM**, **PIC**, **STM32** gần đây nhất là **ESP8266**, **ESP32**, và **RISCV** với năng lực phần cứng và phần mềm đi kèm mạnh mẽ hơn nhiều.

MỘT SỐ ĐẶC ĐIỂM CỦA ARDUINO

- Arduino che dấu đi sự phức tạp của điện tử bằng cách khái niệm đơn giản, che đi sự phức tạp của phần mềm bằng cách thủ tục ngắn gọn. Việc setup output cho 1 MCU bằng cách setup thanh ghi rõ ràng phức tạp đến độ người chuyên cũng phải lật datasheet ra xem, nhưng với Arduino thì chỉ cần gọi 1 hàm.
- Bởi vì tính phổ biến và dễ dùng, với các thư viện được tích hợp sẵn. Bạn chỉ cần quan tâm đến tính năng sản phẩm mà bỏ qua các tiểu tiết (protocol, datasheet ...) Nên giúp các newbie không chuyên dễ dàng tiếp cận và làm ra các sản phẩm tuyệt vời mà không cần phải biết nhiều về điện tử.
- Chính vì không quan tâm nhiều đến cách thức hoạt động của các Module đi kèm, nên đa phần người dùng sẽ khó xử lý được khi có các vấn đề phát sinh ngoài tầm của thư viện.
- Các module prototype làm sẵn cho Arduino có độ bền không cao, mục tiêu đơn giản hóa quá trình làm sản phẩm.

CÁC LỢI ÍCH KHI SỬ DỤNG ARDUINO

- Thiết kế IDE tốt, có thể dễ dàng tích hợp nhiều loại compiler, nhiều loại hardware mà không hề giảm hiệu năng. Ví dụ: Arduino gốc cho AVR, nhưng có nhiều phiên bản cho STM32, PIC32, ESP8266, ESP32... tận dụng tối đa các thư viện sẵn có.
- Các thư viện được viết dựa trên lớp API trên cùng, nên đa số các thư viện cho Arduino có thể dùng được cho tất cả các chip. Điển hình là Arduino cho ESP8266 có thể tận dụng trên 90% các thư viện cho Arduino khác
- Trình biên dịch cho Arduino là C/C++, bạn có biết là khi biên dịch ESP8266 non-os SDK và ESP8266 Arduino cùng dùng chung trình biên dịch? Vậy thì hiệu năng không hề thua kém
- Các tổ chức các thư viện C/C++ theo dạng OOP giúp phân lớp, kế thừa và quản lý cực kỳ tốt cho các ứng dụng lớn. Các MCU ngày càng mạnh mẽ, và ứng dụng cho nó sẽ ngày càng lớn. Các mô hình quản lý code đơn giản trước đây (thuần C) sẽ khó.
- Các project cho Arduino đều opensource, bạn dễ dàng lấy nó và đưa vào sản phẩm production với chất lượng tốt và học hỏi được nhiều từ cách thức thiết kế chương trình của các bậc thầy.
- Arduino chú trọng tính đa nền tảng, module hóa cao, phù hợp với các ứng dụng từ phức tạp tới cực kỳ phức tạp. Các ứng dụng kiểu này rất phổ biến trong thực tế. Nếu bạn không dùng C++, hoặc arduino mà gặp vấn đề về overcontrol thì nên thử qua Arduino.
- Bạn sẽ tiết kiệm được rất nhiều thời gian cho việc tập trung vào tính năng sản phẩm đấy. Thời buổi này, thời gian là tiền và có quá nhiều thứ để học, làm thì nên ưu tiên đúng chỗ.

CỘNG ĐỒNG ARDUINO TRÊN THẾ GIỚI

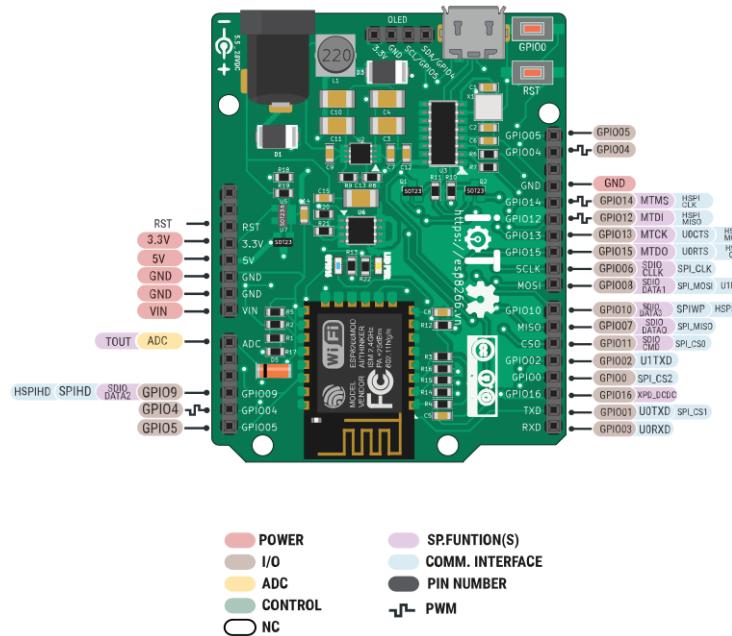
- [Arduino chính thức\(IDE & AVR/ARM/x86 Board\)](http://www.arduino.cc) www.arduino.cc
- [Arduino cho ESP8266](https://github.com/esp8266/Arduino) github.com/esp8266/Arduino
- [Arduino cho ESP32](https://github.com/espressif/arduino-esp32) github.com/espressif/arduino-esp32
- [Arduino cho PIC32](http://chipkit.net/) chipkit.net/
- [Arduino cho STM32](http://www.stm32duino.com/) www.stm32duino.com/
- *[Các dự án Arduino](http://www.hackster.io/arduino) www.hackster.io/arduino

ARDUINO CHO ESP8266 & BOARD MẠCH WiFi UNO

Board mạch **ESP8266 WiFi Uno** là một dự án mã nguồn mở giúp hỗ trợ môi trường phát triển Arduino cho ESP8266. Giúp bạn có thể viết 1 Sketches sử dụng các thư viện và hàm tương tự của Arduino, có thể chạy trực tiếp trên ESP8266 mà không cần bất kỳ Vi điều khiển nào khác.

ESP8266 Arduino core đi kèm với thư viện kết nối WiFi hỗ trợ TCP, UDP và các ứng dụng HTTP, mDNS, SSDP, DNS Servers. Ngoài ra còn có thể thực hiện cập nhật OTA, sử dụng Filesystem dùng bộ nhớ Flash hay thẻ SD, điều khiển servos, ngoại vi SPI, I2C.

Link: github.com/iotmakervn/iot-wifi-uno-hw



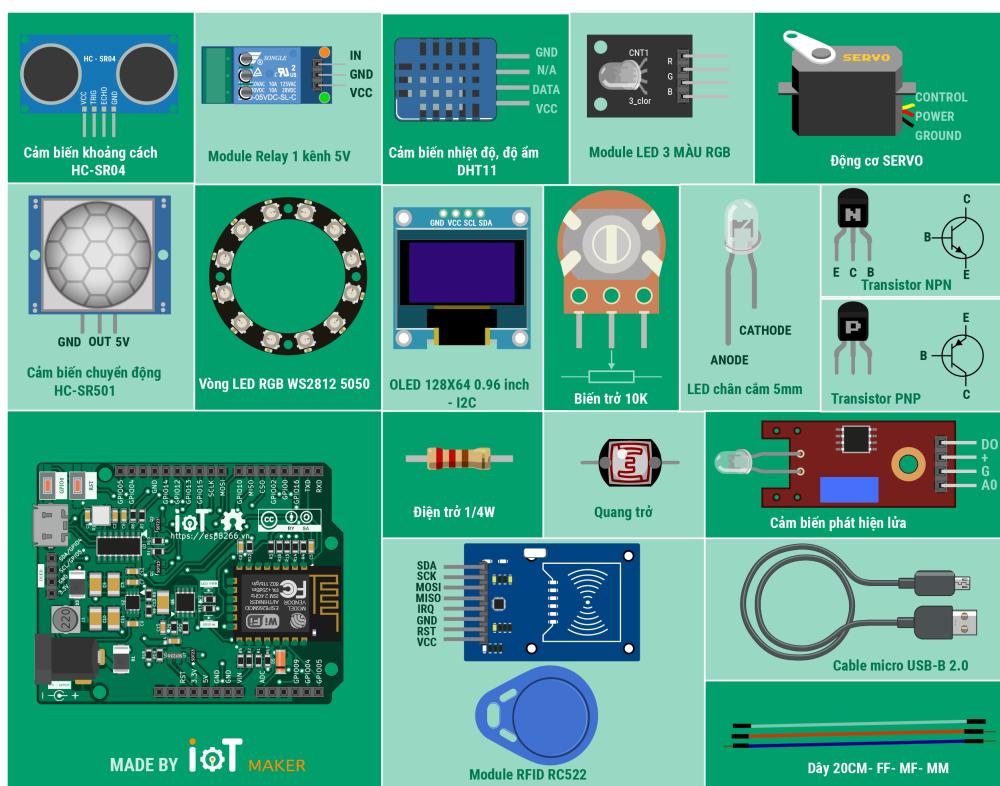
Hình 6. PINOUT

IOT STARTER KIT

HÌNH ẢNH



Hình 7. Hình ảnh thực tế



Hình 8. Các vật tư

BREADBOARD

Breadboard dùng để kết nối các đường mạch điện tử một cách nhanh chóng bằng các dây có 1 đầu M (Male)



Hình 9. Breadboard

THÔNG TIN

IoT Starter Kit - Wifi Uno là bộ dụng cụ các sản phẩm điện tử dùng trong các chương trình giảng dạy STEM (Science-Technology-Engineering-Math) và phát triển ứng dụng IoT.

Bộ Kit sử dụng Board ESP8266 làm bộ điều khiển, tương thích với hầu hết các thư viện cho Arduino, mạnh mẽ với nhiều ứng dụng mẫu và dễ dàng bắt đầu với người mới.

Những linh kiện trong bộ Starter Kit- Wifi Uno và chức năng của các linh kiện : .Các vật tư trong bộ IoT Staterkit

STT	Tên	Số lượng	Chức năng
1	Esp8266 Wifi-Uno	1	Nhận, xử lý, điều khiển thiết bị
2	OLED LCD 128X640.96 inch	1	Hiển thị thông tin người dùng yêu cầu
3	Breadboard 830 Point	1	Linh kiện giúp kết nối các thiết bị (kích thước lớn)
4	Cảm biến chuyển động	1	Phát hiện chuyển động nếu trong vùng hoạt động của cảm biến
5	Cảm biến khoảng cách	1	Xác định khoảng cách đến vật cản
6	Cảm biến nhiệt độ, độ ẩm DHT11	1	Đo nhiệt độ, độ ẩm của môi trường
7	Cảm biến phát hiện lửa	1	Phát hiện có lửa hay không
8	Module RGB LED 3 màu	1	Hiển thị nhiều màu sắc bằng LED

NODE.JS

Node.js là một Javascript Run time Cross Platform (chạy đa hệ điều hành) được xây dựng dựa trên mã nguồn mở Google's V8 JavaScript engine cho Chrome (Browser). Node.js cho phép các lập trình viên có thể xây dựng ứng dụng Server Side, truy cập vào tài nguyên hệ thống và thực hiện được phần lớn các tác vụ hệ điều hành có thể thực hiện bằng ngôn ngữ Javascript, hoặc liên kết C++.

Hiện nay trên thế giới đã có nhiều Công ty ứng dụng Node.js xây dựng các hệ thống production lớn, như Paypal, hoặc các microservice dựa trên Node.js cũng được triển khai ở đa số các hãng hàng đầu về công nghệ.

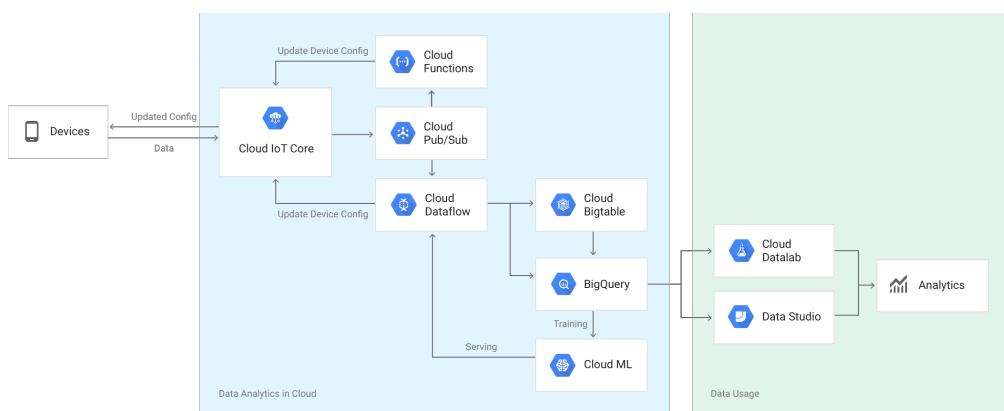
Nền tảng Cloud của gần như tất cả các ông lớn hiện nay đều hỗ trợ thực thi Node.js, điển hình như Amazon Lambda, Google Script, IBM Blumix, Microsoft Azure ...

Ngôn ngữ lập trình Javascript được cải tiến liên tục, hiện nay là EcmaScript 6 (ES5, ES2015) và đang được cải tiến rất nhanh, với nhiều ưu điểm như dễ học, xúc tích, OOP...

Một lý do Javascript được ưa chuộng nữa là đa phần các lập trình viên viết Web, Mobile đều biết, và giờ đây, nhờ Node.js mà họ có thể triển khai các ứng dụng Server Side bằng Javascript, mà không cần dùng ngôn ngữ nào khác (như trước kia phải cần Java, PHP ...)

LÝ DO SỬ DỤNG NODE.JS TRONG CUỐN SÁCH NÀY

Một hệ thống Internet Of Things đầy đủ phức tạp, bao gồm thiết bị, Server xử lý kết nối, Server dữ liệu (Database), các hệ thống cân bằng tải, các hệ thống phân thích, báo cáo dữ liệu, trí tuệ nhân tạo. Mô hình ví dụ của Google IoT Core



Hình 10. Google IoT Core Diagram

Server là một thành phần không thể thiếu trong hệ thống IoT. Với nhiều ưu điểm của Node.js thì nó rất phù hợp trong việc phát triển các Server cho IoT trong tương lai. Ngoài ra, Node.js được cộng đồng hỗ trợ rất nhiều, và không khó để tìm thấy 1 package cần thiết, tiết kiệm rất nhiều thời gian phát triển ứng dụng.

CUỐN SÁCH NÀY CÓ HƯỚNG DẪN JAVASCRIPT?

Không, nhưng bạn đừng vội thất vọng, các ứng dụng Node.js sử dụng để thực hiện các bài tập trong cuốn sách này khá đơn giản và **ít** mã nguồn, đủ cho bạn vẫn hiểu dù cho trước đây chưa bao giờ lập trình Javascript.

SUBLIME TEXT

Nếu ở phần Chip, lập trình cho ESP8266 bạn đã có Arduino IDE, bao gồm cả trình soạn thảo. Nhưng với Node.js thì bạn cần 1 trình soạn thảo khác. Ngoài Sublime Text, thì bạn có thể lựa chọn các trình soạn thảo phổ biến hiện nay, như Atom, Visual Code, nhưng đừng dại sử dụng Nodepad, mặc dù là vẫn được.

Sublime Text là một trình soạn thảo được nhiều lập trình viên ưu thích hàng đầu hiện nay, bởi nhiều lý do, trong đó, tốc độ là quan trọng nhất. Nó thực sự nhanh, nhanh gần như số 1 trong số các trình soạn thảo hiện nay. Ngoài ra nó miễn phí (lâu lâu nhắc mua, khung thoại mà nhiều lập trình viên bảo thiếu thì buồn).

```

var hello = (text) => {
  var hello = (text) => {
    console.log(text);
  }
  hello('world');
}

```

Line 5, Column 16 UTF-8 Tab Size: 4 JavaScript

Hình 11. Sublime Text

CÀI ĐẶT VÀ CHUẨN BỊ

ARDUINO IDE

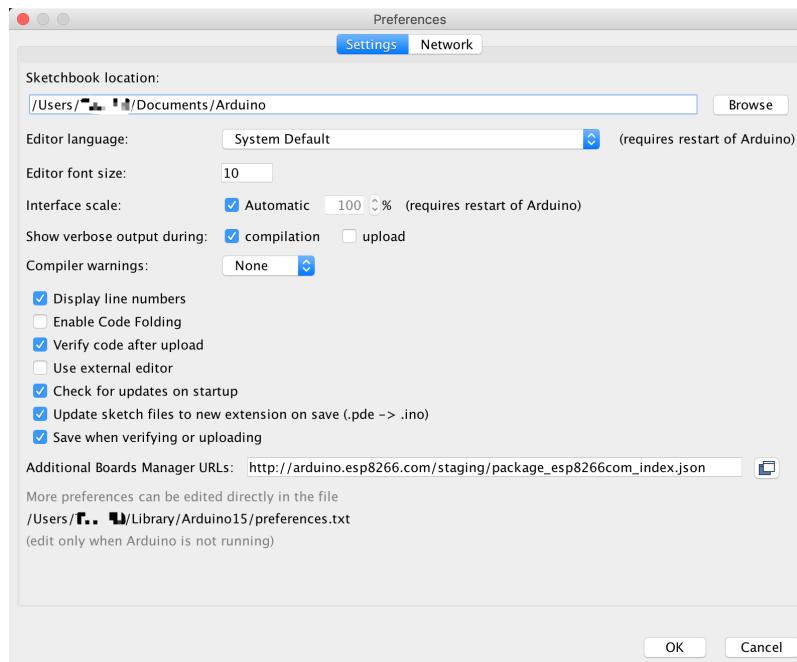
Bước 1: Cài đặt Arduino IDE.

Download và cài đặt arduino từ trang chủ của arduino. Link download: www.arduino.cc/en/Main/Software.

Tùy hệ điều hành mà chọn gói cài đặt thích hợp.

Bước 2: Cài đặt bộ công cụ, trình biên dịch, SDK hỗ trợ chip ESP8266 trong Arudino IDE.

Với bộ công cụ này, chúng ta có thể dễ dàng lập trình, biên dịch và sử dụng các thư viện dành cho ESP8266 trực tiếp trên Arduino IDE. Mở Arduino IDE, trên thanh Menu chọn **File → Preferences**, trong tab **settings** chọn các tùy chọn như hình dưới:



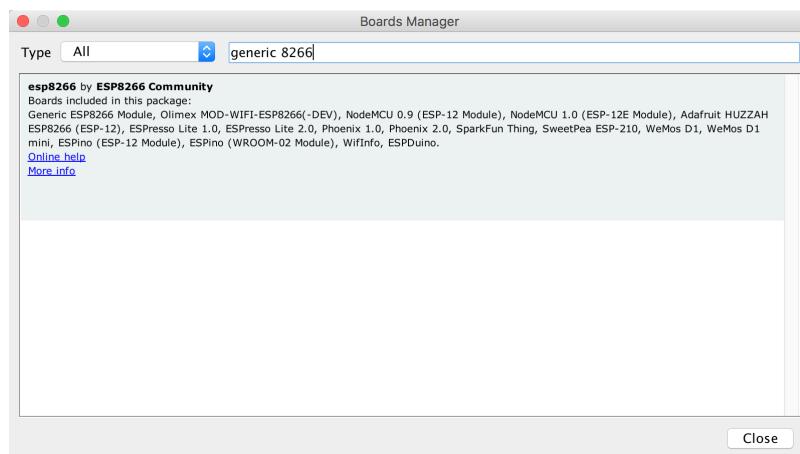
Hình 12. Thêm file thông tin board ESP8266

Sketchbook location là đường dẫn mà bạn muốn lưu Sketch (file chương trình), trên các hệ điều hành Unix liked đường dẫn mặc định là: `/home/name_your_computer/Arduino`. Đây cũng sẽ là vị trí lưu những thư viện mà chúng ta sẽ thêm vào sau này.

Mục **Additional Board Manager URLs** field nhập đường dẫn http://arduino.esp8266.com/stable/package_esp8266com_index.json.

Bước 3: Cài đặt board ESP8266.

Mở **Boards Manager** ở mục **Tools** trên thanh menu-bar → tìm board cần sử dụng với keyword **Generic 8266** → chọn board cần cài đặt như hình và nhấn vào **install**.

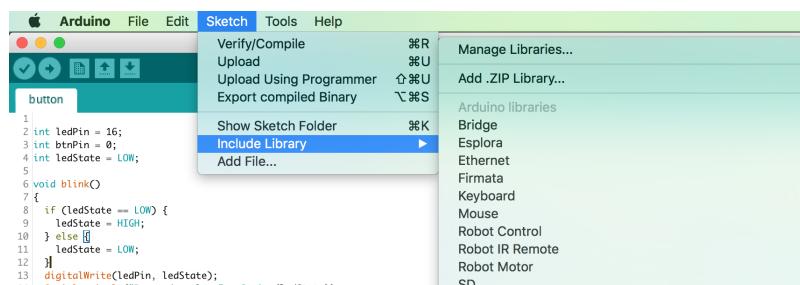


Hình 13. Cài đặt board ESP8266

CÀI ĐẶT THƯ VIỆN ARDUINO

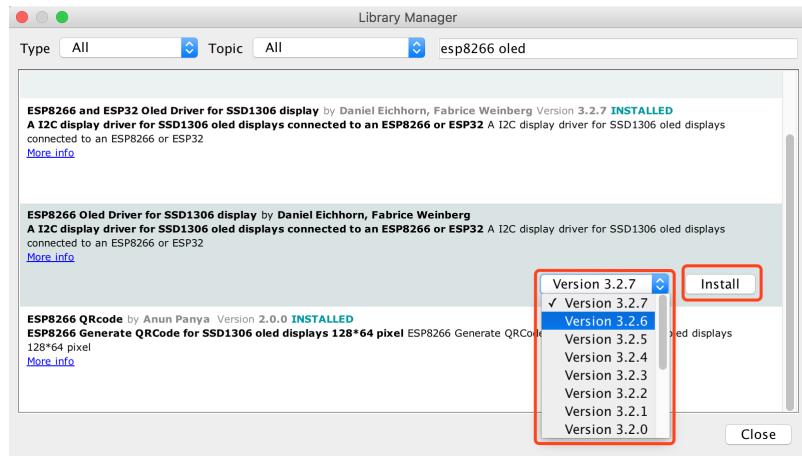
Một số thư viện do các nhà phát triển khác công bố và được tự do sử dụng có thể cài đặt trực tiếp bằng công cụ Library Manager của Arduino.

Khởi động arduino IDE và chọn mục **Sketch** ⇒ **include library** ⇒ **Manage libraries**:



Hình 14. Khởi động Library Manager

Trong mục **library manager** nhập nội dung thư viện cần tìm tại hộp thoại text box, chọn phiên bản, rồi nhấn **install**, Những thư viện đã được cài đặt sẽ có text hiển thị **INSTALLED** ở đầu mỗi thư viện. Ví dụ tìm thư viện OLED liên quan đến ESP8266:



Hình 15. Cài đặt thư viện

USB CDC DRIVER.

Board ESP8266 WiFi Uno được kết nối với máy tính qua cổng USB MicroB và sử dụng chip **CH340** để chuyển đổi USB sang UART. Vì vậy cần cài USB driver để máy tính và board có thể giao tiếp với nhau.

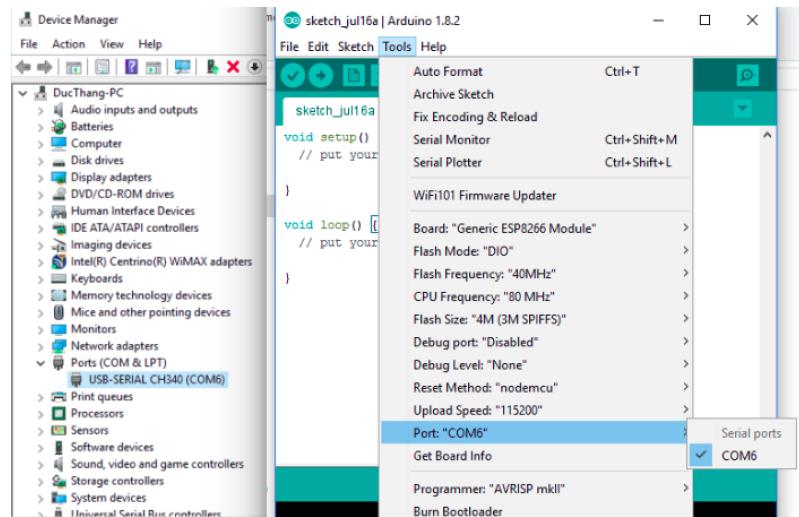
Thực hiện kết nối cable USB với board, đảm bảm đèn LED khoanh tròn sáng như ở hình dưới:



Hình 16. Connect USB

WINDOWS & LINUX

Tải bản cài đặt USB driver cho Windows www.wch.cn/download/CH341SER_ZIP.html và cho Linux www.wch.cn/download/CH341PAR_LINUX_ZIP.html. Làm theo các yêu cầu cài đặt. Sau khi cài đặt, kết quả hiển thị trên Arduino như hình



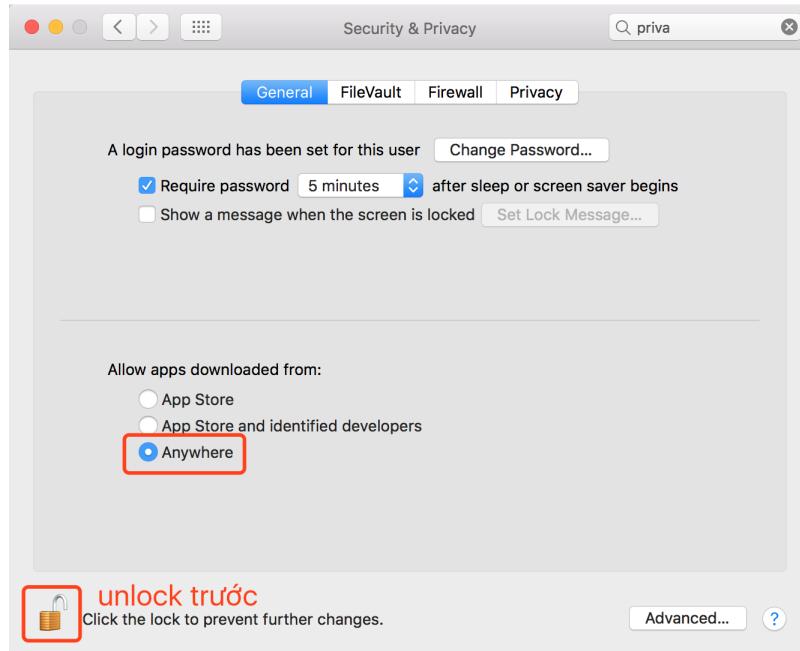
Hình 17. Kết nối thành công

MAC OS

Tải bản cài đặt: arduino.esp8266.vn/_static/download/CH34x_Install_V1.3.pkg

Đối với Mac OS Sierra trở về sau nếu gặp vấn đề bị RESET máy thì xử lý như sau:

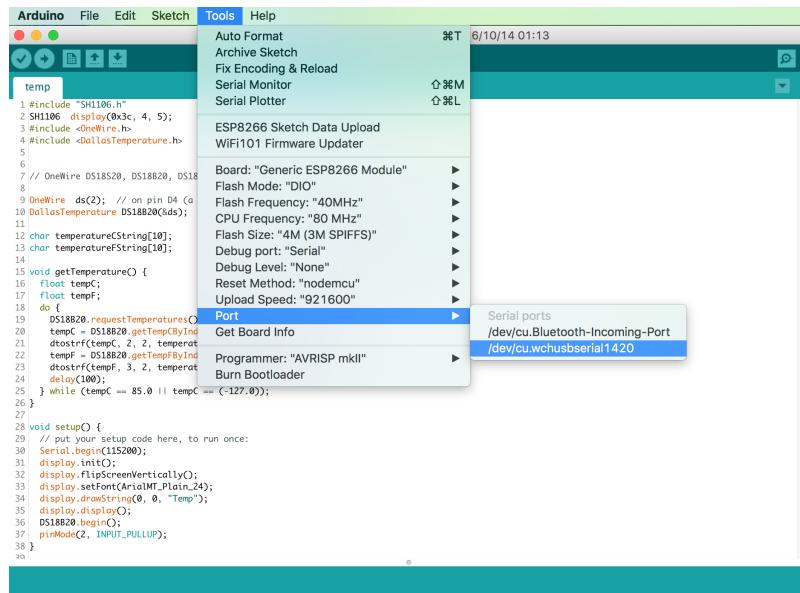
- Mở ứng dụng "Terminal" **cmd + space** → Enter Terminal
- Xóa driver: **sudo rm -rf /System/Library/Extensions/usb.kext**
- Với một số máy, bạn phải thực thi thêm **sudo rm -rf /Library/Extensions/usbserial.kext**
- Nếu không thể thực hiện được lệnh trên, bạn cần phải thay đổi **Security and Privacy** trong phần **System Preference**. Chọn **Allow Apps Downloaded From** từ **Mac App Store and Identified Developers** sang **Anywhere** - Và tải **CH34x_Install_V1.3.pkg** về cài đặt lại



Hình 18. Lựa chọn Allow Apps Downloaded From Anywhere

CHỌN BOARD ESP8266 WiFi UNO TRONG ARDUINO IDE

Sau khi kết nối và cài đặt xong, sẽ xuất hiện cổng COM ảo trên máy tính (Tùy từng loại hệ điều hành mà có những tên cổng như: **COM1, COM2 ...** đối với Windows, **/dev/tty.wchusbserial1420** trên Mac OS, **/dev/ttys0** trên Linux) Mở Arduino IDE và lựa chọn (tham khảo cấu hình kết nối như hình dưới):



Hình 19. Cấu hình Board ESP8266 WiFi Uno

- Board: **Generic ESP8266 Module**
- Flash Size: **4M (3M SPIFFS)**
- Port: chọn cổng khi gắn thiết bị vào sẽ thấy xuất hiện

- Upload speed: Chọn cao nhất, nếu nạp không được chọn thấp dần

SERIAL TERMINAL

SỬ DỤNG ARDUINO IDE SERIAL MONITOR

MỘT SỐ PHẦN MỀM MIỄN PHÍ KHÁC

NODE.JS

Tải và cài đặt Node.js tại: nodejs.org/en/download/

SUBLIME TEXT

Tải và cài đặt tại: www.sublimetext.com/

GIT

Một công cụ hỗ trợ khác bạn cũng nên cài đặt và tập sử dụng, nó không giúp bạn trở thành 1 lập trình viên, nhưng nó giúp 1 lập trình viên trở nên chuyên nghiệp và làm việc hiệu quả: git-scm.com/

TỔNG KẾT

Tới lúc này, bạn có thể đã có cái nhìn tổng quan về hệ sinh thái, công cụ và phương thức làm việc với ESP8266 cũng như tổng quan về hệ thống IoT. Đồng thời đã có thể bắt đầu việc phát triển ứng dụng cho ESP8266 ngay lập tức. Tiếp theo chúng ta sẽ tìm hiểu những kiến thức cơ bản và cần thiết nhất để khởi động ứng dụng cho ESP8266.

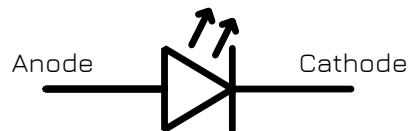
HELLO WORLD

Bất kỳ một chương trình học nào cũng cần nên bắt đầu một cách từ từ. Bởi vì thời điểm này chúng ta đều mới bắt đầu, nhiều khái niệm, kiến thức về lĩnh vực này gần như không có nhiều. Helloworld giúp các bạn có thể nắm được các kiến thức cơ bản, làm sao để biên dịch, nạp được chương trình. Làm sao để sử dụng các thư viện công cộng. Cũng như nắm được một số kiến thức về kiến trúc chương trình Arduino.

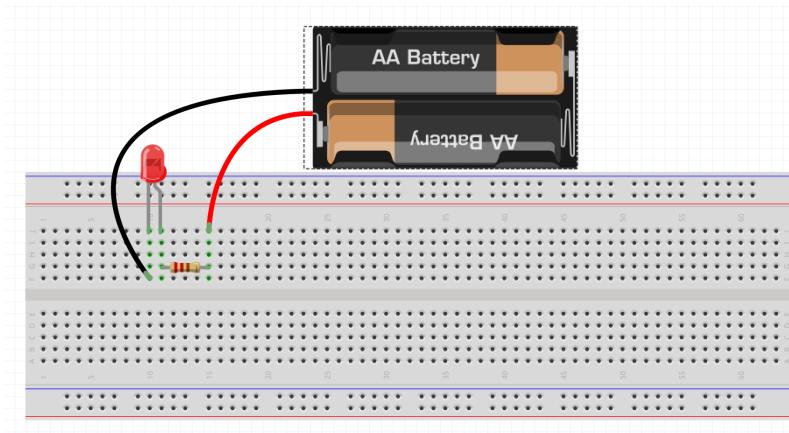
CHỐP TẮT BÓNG LED

KIẾN THỨC

Đèn LED viết tắt (Light Emitting Diodes) - là bóng bán dẫn có thể phát sáng với màu sắc khác nhau tùy thuộc vào chất liệu bán dẫn. Để điều khiển được bóng LED cần cung cấp mức điện áp chênh lệch giữa cực âm và cực dương của bóng LED cao hơn mức điện áp V_f (datasheet), thường là 3.2v, và dòng điện nhỏ hơn mức chịu đựng của nó, thường là 15mA

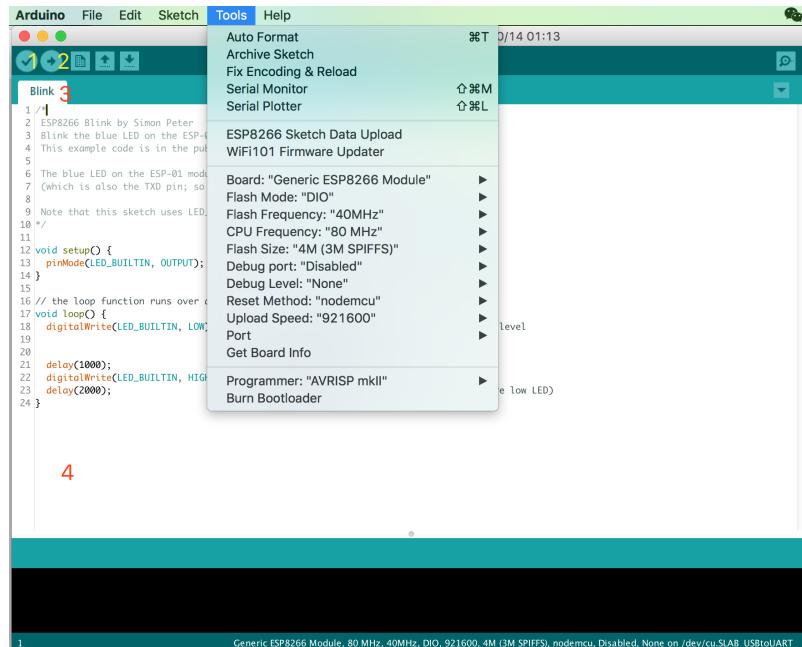


Hình 20. Ký hiệu LED trên mạch điện (Cathode+, Anode-)



Hình 21. Mạch có thể chạy được như sau

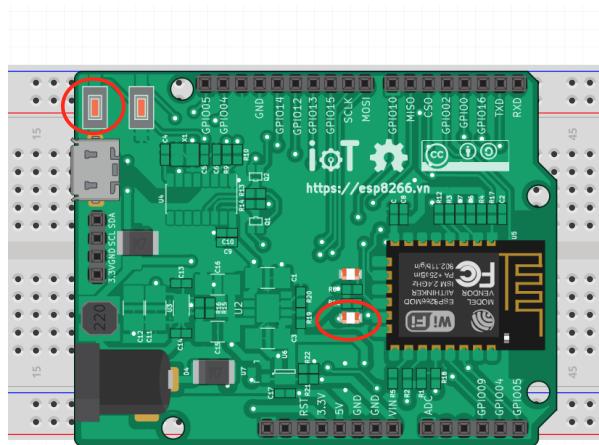
Arduino IDE



Hình 22. Arduini IDE

- ① Biên dịch chương trình (kiểm tra có lỗi hay không)
- ② Biên dịch và nạp chương trình
- ③ Tab tên file
- ④ Khu vực nội dung file **ino**

ĐẦU NỐI



Hình 23. Mạch ESP8266 WiFi Uno có đầu nối sǎn LED vào Pin 16, và nút nhá̉t vào Pin 0

MÃ NGUỒN CHỐP TẮT DÙNG DELAY

```

int pin_led = 16;
/* hàm này được gọi 1 lần duy nhất khi khởi động */
void setup() {
    pinMode(pin_led, OUTPUT);      // cấu hình pin 16 là ngõ ra
}

/* hàm loop sẽ được gọi liên tục */
void loop() {
    digitalWrite(pin_led, HIGH); // tắt LED (HIGH - có nghĩa là mức cao)
    delay(1000);               // chờ 1 giây
    digitalWrite(pin_led, LOW); // bật LED bởi mức điện áp LOW
    delay(1000);               // chờ 1 giây
}

```

MÃ NGUỒN CHỐP TẮT DÙNG ĐỊNH THỜI

```

int ledPin = 16;
int ledState = LOW;
unsigned long previousMillis = 0;
const long interval = 1000;

void setup() {
    pinMode(ledPin, OUTPUT);
}

void loop() {
    unsigned long currentMillis = millis();
    if (currentMillis - previousMillis >= interval) {
        previousMillis = currentMillis;
        if (ledState == LOW)
            ledState = HIGH; // Đổi trạng thái
        else
            ledState = LOW; // Đổi trạng thái
        digitalWrite(ledPin, ledState);
    }
}

```

► <https://www.youtube.com/watch?v=8jM-JTFeAlg> (YouTube video)

Video kết quả

DIGITAL IO

Tên Pin trong Arduino (Pin number) giống với thứ tự chân của ESP8266. `pinMode`, `digitalRead`, và `digitalWrite` đều sử dụng Pin Number như nhau, ví dụ như đọc GPIO2, gọi hàm `digitalRead(2)`.

Chân `GPIO0..15` có thể là `INPUT`, `OUTPUT`, hay `INPUT_PULLUP`. Chân `GPIO16` có thể là `INPUT`, `OUTPUT` hay `INPUT_PULLDOWN_16`. Khi khởi động, tất cả các chân sẽ được cấu hình là `INPUT`.

Mỗi chân có thể phục vụ cho một tính năng nào đó, ví dụ `Serial`, `I2C`, `SPI`. Và tính năng đó sẽ được cấu hình đúng khi sử dụng thư viện. Hình bên dưới thể hiện sơ đồ chân đối với module ESP-12 phổ biến.

`GPIO6` và `GPIO11` không được thể hiện bởi vì nó được sử dụng cho việc kết nối với Flash. Việc sử dụng 2

chân này có thể gây lỗi chương trình.



Một số board và module khác (ví dụ ESP-12ED, NodeMCU 1.0) không có GPIO9 và GPIO11, họ sử dụng với chế độ DIO cho Flash, trong khi ESP12 chúng ta nói bên trên sử dụng chế độ QIO

Ngắt GPIO hỗ trợ thông qua các hàm `attachInterrupt`, `detachInterrupt` Ngắt GPIO có thể gán cho bất kỳ GPIO nào, ngoại trừ GPIO16. Đều hỗ trợ các ngắt tiêu chuẩn của Arduino như: `CHANGE`, `RISING`, `FALLING`.

TỔNG KẾT

Các ứng dụng mở rộng

- Fading LED (sáng dần hay tắt dần)
- Chớp tắt LED dùng Ticker

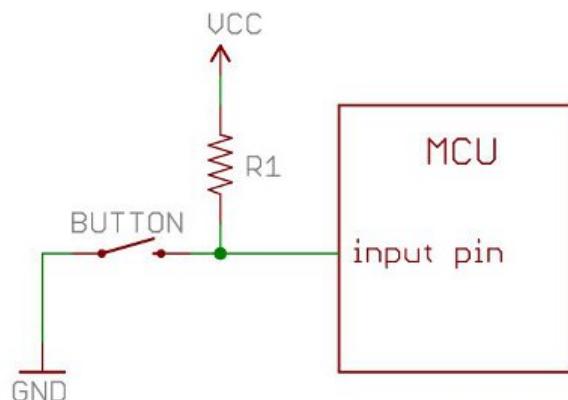
NÚT NHẤN

KIẾN THỨC

Nút nhấn sẽ giúp việc ESP8266 khởi động một hành động nào đó khi cần thiết. Trong nhiều ứng dụng chúng ta hầu như đều cần những kích hoạt từ bên ngoài. Xuyên suốt cuốn sách này, sẽ dùng nút nhấn để kích hoạt chạy các ứng dụng mẫu cũng như đèn LED để thông báo các trạng thái. Trong phần này, nhấn nút đèn LED sẽ chuyển trạng thái (từ sáng → tắt và ngược lại).

Đây là ví dụ đơn giản, trong thực tế việc xử lý nút nhấn khá phiền phức. Bởi vì nút nhấn vật lý khi được nhấn sẽ tạo ra hàng loạt các xung lên xuống (nhiều, bouncing...). Thường thì chỉ cần đảm bảo mức Logic của chân đo được đã được giữ ổn định trong khoảng 100 mili giây là được xem đã ổn định.

Ngoài cách dùng ngắn để xác định nút nhấn có được nhấn hay không - cách này sẽ tiết kiệm tài nguyên tính toán của CPU, nó chỉ được gọi khi có sự kiện xảy ra, thì còn một cách nữa là hỏi vòng: Cách này đổi hỏi CPU liên tục kiểm tra xem mức Logic của nút nhấn. Đồng thời việc đáp ứng cũng không nhanh bằng sử dụng ngắn.



Yêu cầu: Nhấn nút (GPIO0) thì chớp tắt đèn LED (GPIO6) và in ra cổng Serial



Mạch ESP8266 WiFi Uno đấu sẵn nút nhấn vào GPIO0

MÃ NGUỒN DÙNG HỎI VÒNG

```

int ledPin = 16;           // LED nối vào chân 16
int btnPin = 0;            // Nút nhấn nối vào chân 0
int ledState = LOW;

void blink()
{
    if (ledState == LOW) {
        ledState = HIGH;
    } else {
        ledState = LOW;
    }
    digitalWrite(ledPin, ledState); //Đảo trạng thái LED & in ra serial
    Serial.println("Pressed, value=" + String(ledState));
}

bool isPressed()
{
    return (digitalRead(btnPin) == 0);
}
void setup()
{
    pinMode(ledPin, OUTPUT);      // Cấu hình LED là ngõ ra
    pinMode(btnPin, INPUT_PULLUP); // Cấu hình nút nhấn là ngõ vào pull-up

    Serial.begin(115200);
}

void loop()
{
    if (isPressed()) {
        blink();
    }
}

```

MÃ NGUỒN DÙNG NGẮT

```

int ledPin = 16;           // LED nối vào chân 16
int btnPin = 0;            // Nút nhấn nối vào chân 0
int ledState = LOW;

void blink()
{
    if (ledState == LOW) {
        ledState = HIGH;
    } else {
        ledState = LOW;
    }
    digitalWrite(ledPin, ledState);
    Serial.println("Pressed, value=" + String(ledState));
}

void setup()
{
    pinMode(ledPin, OUTPUT);      // sets the digital pin as output
    pinMode(btnPin, INPUT_PULLUP); // Cấu hình nút nhấn là ngõ vào pull-up
    attachInterrupt(btnPin, blink, FALLING); //cài đặt ngắt cho chân LED
    Serial.begin(115200);
}

void loop()
{
    //Không phải làm gì
}

```

► https://www.youtube.com/watch?v=wE-yc_CGpfE (YouTube video)

Video kết quả

CÁC KHÁI NIỆM

- Ngắt
- pull-up

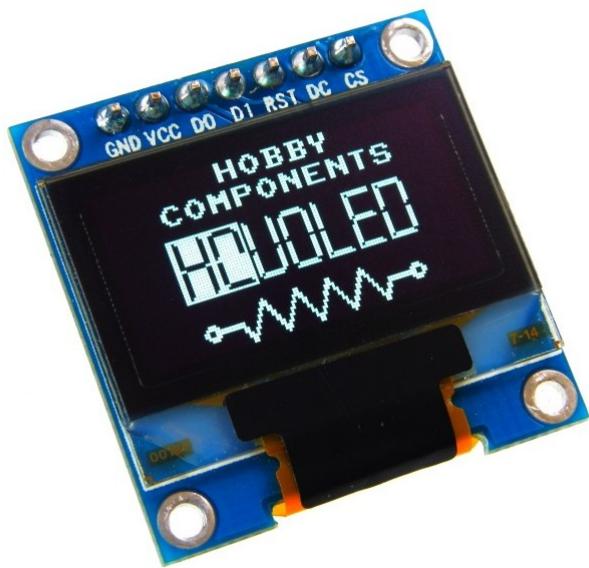
OLED

MÀN HÌNH OLED

OLED (Organic Light Emitting Diode) là loại màn hình hiển thị bao gồm một lớp vật liệu hữu cơ với chủ yếu là cacbon nằm giữa hai điện cực anot và catot sẽ tự động phát sáng mỗi khi có dòng điện chạy qua. OLED sử dụng điốt phát quang hữu cơ, chính vì thế nó không cần tới đèn nền chiếu sáng, do đó có lợi thế về kích thước cũng như tiết kiệm điện hơn so với các loại LCD. Và độ sáng tương đối tốt ở môi trường sáng tự nhiên

MÀN HÌNH OLED SSD1306

Là màn hình loại nhỏ, kích thước tầm 0.96 inch cho tới 1.25 inch, được dùng khá rộng rãi trong các sản phẩm điện tử. Tấm nền được điều khiển bằng chip driver SSD1306. Chip này giao tiếp với các bộ điều khiển/MCU khác bằng giao tiếp LCD



Hình 24. OLED SSD1306

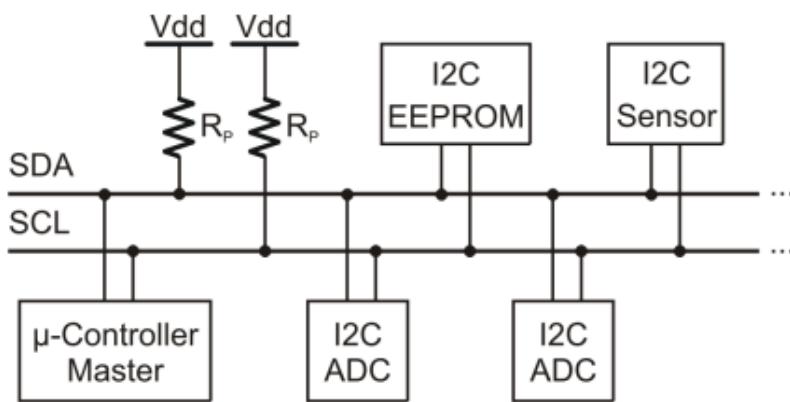
GIAO TIẾP I2C

I2C (Inter-Integrated Circuit) là một loại bus nối tiếp được phát triển bởi hãng Philips nhằm truyền nhận dữ liệu giữa các IC. I2C sử dụng 2 đường truyền tín hiệu, 1 đường xung nhịp đồng hồ (SCL) do

Master phát đi và 1 đường truyền dữ liệu theo 2 hướng (SDA)

I2C Clock

```
Failed to generate image: Could not find the 'WaveDromEditor' executable in PATH; add it to the PATH or specify its
location using the 'WaveDromEditor' document attribute
{
  signal:
  [
    {
      name: "SCL", wave: "hn....."},
      {},
      { name: "SDA", wave: "h0=====01", data: 'A0 A1 A2 D0 D1 D2 D3 D4', node: '..A..B....C..' },
      {
        node: '..a..b....c..' },
    ],
    edge: [
      'A-a', 'B-b', 'a<->b Address',
      'C-c', 'b<->c Data',
    ]
}
```



Hình 25. Mô hình mạng I2C

Mạch vật lý I2C là mạch cực thu hồi, do đó để mạng I2C có thể hoạt động được, cần tối thiểu 2 cặp điện trở pull-up như trên hình. Thông thường 4k7, hoặc 1k2. Tùy thuộc vào tốc độ truyền và khoảng cách truyền.

HIỂN THỊ MÀN HÌNH OLED VỚI ESP8266

Bước 1: Đấu nối nối chân GPIO4 của ESP8266 với chân SDA của OLED, chân GPIO5 với SCL. Cấp nguồn 3v3 vào VCC và đấu GND cho OLED. Tuy nhiên với board ESP8266 IoT Uno thì phần đấu nối đã ra sẵn header, bạn chỉ cần cắm OLED vào như hình



Hình 26. ESP8266 với OLED

Bước 2: Cài đặt thư viện [ESP8266 and ESP32 OLED driver for SSD1306 display](#)

Bước 3: Lập trình Chúng ta sẽ thực hiện hiển thị giả lập đồng hồ trên màn hình OLED

```
#include <Wire.h>
#include "SSD1306.h"

SSD1306 display(0x3c, 4, 5);
int thoi_gian = 0;
void setup()
{
    Serial.begin(115200);
    display.init();
    //display.flipScreenVertically(); //đảo chiều
    display.setFont(ArialMT_Plain_10);
    display.drawString(0, 0, "Hello world");
    display.display();
    delay(1000);
    display.clear();
}

void loop()
{
    int gio, phut, giay;

    delay(1000);
    thoi_gian++;

    gio = thoi_gian/3600;
    phut = (thoi_gian%3600)/60;
    giay = thoi_gian % 60;
    display.clear();
    display.drawString(0, 0, String(gio) + ":" + String(phut) + ":" + String(giay));
    display.display();
}
```

TỔNG KẾT

Tổng kết chương

ESP8266 WIFI

Kết nối WiFi chính điểm mạnh nhất của chip ESP8266, nó có thể kết nối đến các Router sẵn có trong gia đình, các Access Point với các tiêu chuẩn kết nối thông dụng hiện nay ở tần số 2.4GHz - ở chế độ STA. Ngoài ra, ESP8266 còn hỗ trợ chế độ AP (Access Point), tức là nó có thể khởi động một (hoặc nhiều) Access Point và cho phép các Client khác có thể kết nối vào, hoặc chạy đồng thời cả chế độ STA và AP.

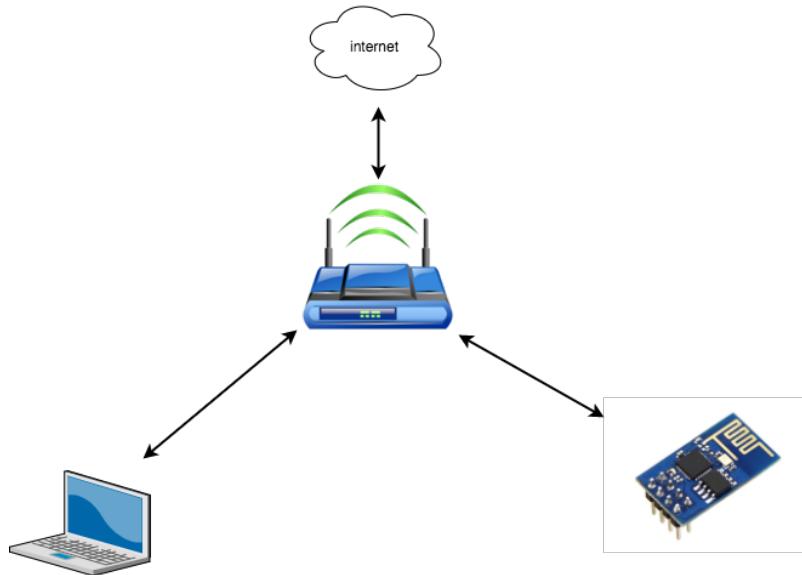
Trong đa phần các ứng dụng thì chế độ STA được sử dụng rất nhiều, nó giúp thiết bị kết nối đến mạng WiFi cục bộ, có internet để kết nối về Server và gửi dữ liệu. Một số trường hợp khác thì chế độ AP được sử dụng để trao đổi dữ liệu với ESP8266 và máy tính (hoặc thiết bị có hỗ trợ trình duyệt). Ví dụ như điều khiển đóng tắt đèn thông qua Web Server chạy trên ESP8266.

WiFi Access Point là một thiết bị xử lý kết nối trung tâm và phân phối các luồng dữ liệu. Như là việc xử lý các gói tin IP để định địa chỉ mạng LAN, định tuyến các gói tin từ Internet về các máy trạm (Station).



Hình 27. WiFi Access Point

Thiết bị kết nối đến Access Point được gọi là Station, các máy tính Laptop, máy tính có card WiFi khi kết nối vào Access Point thì đều được gọi là Station



Hình 28. Mạng WiFi

Các Station khi muốn kết nối vào Access Point thì cần xác định thông qua **BSSID**, thông thường chúng ta hay gọi là **SSID** - hay mạng WiFi. Bạn có thể dễ dàng xem danh sách SSID xung quanh mình khi scan wifi trên máy tính để kết nối mạng Internet.

Trong phần này chúng ta sẽ tìm hiểu về các chế độ WiFi của ESP8266

- Chế độ **Station - STA** kết nối tới Access Point sẵn có
- Sử dụng **HTTPClient** để gởi và lấy dữ liệu từ Internet
- Chế độ **Access Point - AP** cho phép Client khác kết nối vào
- **Web Server** chạy trên ESP8266, dùng để bật tắt đèn LED

CHẾ ĐỘ WIFI STATION

KIẾN THỨC

Để kết nối được vào mạng Internet, thì đầu tiên ESP8266 phải kết nối vào mạng WiFi nội bộ, và mạng WiFi nội bộ phải có kết nối WAN Internet. Đa phần các Modem hiện nay đều tích hợp luôn cả WiFi Access Point, do đó khá dễ dàng trong việc triển khai các ứng dụng IoT.

Khi muốn kết nối vào mạng WiFi cục bộ thì ESP8266 cần phải hoạt động ở chế độ Station (STA), đồng thời nó phải được cung cấp tên (SSID) và mật khẩu mạng WiFi.

Mỗi Access Point đều yêu cầu một phương thức mã hóa để Station sử dụng để tạo kết nối - ví dụ **WEP**, **WPA2**, tuy nhiên chúng ta có lẽ không cần quan tâm nhiều, vì ESP8266 sẽ tự động thực hiện các thao tác lựa chọn phương thức mã hóa.

Khi kết nối thành công vào mạng WiFi thì ESP8266 sẽ khởi động DHCP Client (mặc định) để xin cấp phát địa chỉ IP trước khi bắt đầu các kết nối IP. Do đó, nếu như vì lý do gì đó, mà Access Point của bạn không có DHCP Server để cấp phát IP thì bạn phải cấu hình IP tĩnh cho ESP8266.

KẾT NỐI VÀO MẠNG WIFI NỘI BỘ

Với đoạn code này, nếu bạn cung cấp đúng **SSID** và **PASSWORD**, đồng thời Access Point hoạt động thì thiết bị sẽ kết nối và in ra Serial Terminal địa chỉ IP của ESP8266 trong mạng LAN

```
#include <ESP8266WiFi.h>

const char* ssid      = "your-ssid";
const char* password = "your-password";

void setup() {
    Serial.begin(115200);
    delay(10);
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
}

void loop() {
```

SỬ DỤNG WiFiMulti

Tuy nhiên, đôi lúc ứng dụng bạn cần **nồi đồng cối đá**, thì có 2-3 mạng WiFi để backup là bình thường, class WiFiMulti sẽ giúp bạn điều đó. Cùng với một hàm monitor đơn giản để báo cho các chức năng khác biết khi mạng đã được thiết lập.

```
#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>

ESP8266WiFiMulti wifiMulti;
boolean connectioWasAlive = true;

void setup()
{
    Serial.begin(115200);
    Serial.println();
    wifiMulti.addAP("primary-network-name", "pass-to-primary-network");
    wifiMulti.addAP("secondary-network-name", "pass-to-secondary-network");
    wifiMulti.addAP("tertiary-network-name", "pass-to-tertiary-network");
}

void monitorWiFi()
{
    if (wifiMulti.run() != WL_CONNECTED)
    {
        if (connectioWasAlive == true)
        {
            connectioWasAlive = false;
            Serial.print("Looking for WiFi ");
        }
        Serial.print(".");
        delay(500);
    }
    else if (connectioWasAlive == false)
    {
        connectioWasAlive = true;
        Serial.printf(" connected to %s\n", WiFi.SSID().c_str());
    }
}

void loop()
{
    monitorWiFi();
}
```

HTTP CLIENT

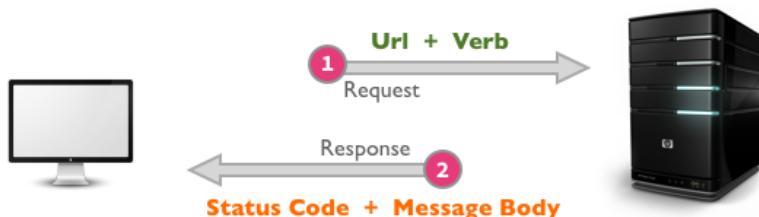
GIAO THỨC HTTP

HTTP - Hypertext Transfer Protocol (giao thức truyền dẫn siêu văn bản), là giao thức để truyền dữ liệu giữa các máy tính qua [www](#) (World Wide Web), với dữ liệu có thể là dạng text, file, ảnh, hoặc video.

HTTP được thiết kế để trao đổi dữ liệu giữa Client và Server trên nền TCP/IP, nó vận hành theo cơ chế **yêu cầu/trả lời, stateless - không lưu trữ trạng thái**. Trình duyệt Web chính là Client, và một máy chủ chứa Web Site là Server. Client sẽ kết nối tới Server, gửi dữ liệu đến server bao gồm các thông tin header. Server nhận được thông tin và căn cứ trên đó gửi phản hồi lại cho Client. Đồng thời đóng kết nối.

Một ví dụ điển hình là khi bạn gõ địa chỉ vào thanh địa chỉ của trình duyệt và nhấn [Enter](#), thì ngay lập tức Web Client sẽ thực hiện việc gửi yêu cầu tới Web Server có địa chỉ mà bạn vừa gõ. Web Server sẽ trả lời bằng nội dung Web Site mà bạn cần xem.

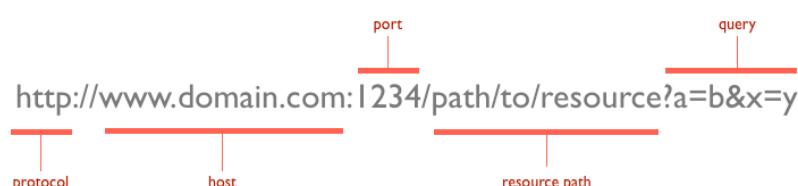
Trong giao thức HTTP, việc thiết lập kết nối chỉ có thể xuất phát từ phía client (lúc này có thể gọi là HTTP Client). Khi client gửi yêu cầu, cùng với URL và payload (dữ liệu muốn lấy) tới server. Server (HTTP Server) lắng nghe mọi yêu cầu từ phía client và trả lời các yêu cầu ấy, và khi trả lời xong kết nối được chấm dứt.



Hình 29. Cách thức HTTP hoạt động

Khi nhắc tới HTTP thì Hyperlink, hay URL (Uniform Resource Locator) là những khái niệm được thấy hàng ngày

URL được dùng để định dạng địa chỉ Website, chứa các thông tin yêu cầu từ client và server dựa vào đó xử lý, cấu trúc của nó như hình



Hình 30. Cấu trúc 1 URL

Ví dụ bạn có thể gửi thông tin về nhiệt độ đến server thông qua đường dẫn:

http://esp8266.vn/log.php?nhiet_do=30

Cấu trúc 1 URL

scheme	host	port	path	query	fragment
http://	server.com:	8080	/path/to/log.php	?nhiet_do=30&do_am=80	#test

- ① **scheme** xác định giao thức truyỀn tới server, nếu là **https** thì sẽ được mã hóa
- ② **host** địa chỉ server
- ③ **port** port server dùng để phục vụ, nếu ko có thì mặc định là **80** cho web
- ④ **path** thông tin client muốn truy suẤt
- ⑤ **query** thông tin client muốn gửi lên
- ⑥ **fragment** thuộc tính này giúp browser đi đến vị trí của trang

Với đường dẫn như trên, khi bạn gõ vào trình duyệt, thì quá browser sẽ thực hiện kết nối và gửi dữ liệu như sau. Bạn có thể sử dụng curl để xem chi tiết dữ liệu raw truyền nhận **curl -v**

http://esp8266.vn/log.php?nhiet_do=30

```
GET /log.php?nhiet_do=30 HTTP/1.1
Host: esp8266.vn
User-Agent: curl/7.49.1
Accept: */*
```

Dữ liệu trả về

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive

data
```

GIAO THỨC HTTP ĐỊNH NGHĨA MỘT SỐ PHƯƠNG THỨC (METHOD) TRUYỀN ĐẾN SERVER:

- **GET** là phương thức yêu cầu dữ liệu đơn giản và thường sử dụng nhất của HTTP. Phương thức GET yêu cầu server chỉ trả về dữ liệu bằng việc cung cấp các thông tin truy vấn trên URL, thông thường Server căn cứ vào thông tin truy vấn đó trả về dữ liệu mà không thay đổi nó. **path** và **query** trong URL chứa thông tin truy vấn.
- **POST** tương tự như **GET**, nhưng **POST** có thể gửi dữ liệu về Server.
- **PUT** là phương thức yêu cầu tạo mới một dữ liệu, giống **POST** nhưng đánh dấu cho Server biết, nếu dữ liệu không tồn tại trong cơ sở dữ liệu thì tạo mới, hoặc sửa đổi nó.
- **DELETE** Tương tự như **GET**, nhưng báo cho Server biết về việc xóa dữ liệu thông qua URL

Các phương thức thông thường chỉ dùng **GET** và **POST**, các phương thức còn lại thường sử dụng trong API server (RESTful). Một số điểm khác biệt giữa **POST** và **GET**

- **GET** có thể bị cache (lưu trữ ở trình duyệt và sử dụng lại sau đó), nội dung request có thể lưu trữ ở lịch sử trình duyệt, có thể được đánh dấu (bookmark)
- **GET** không nên sử dụng để gửi các dữ liệu nhạy cảm
- **GET** bị giới hạn độ lớn dữ liệu cần gửi
- **GET** chỉ nên dùng để lấy dữ liệu về
- **POST** không bị cache, không tồn tại dữ liệu gửi trong lịch sử trình duyệt, không thể đánh dấu (bookmark)
- **POST** không giới hạn bởi độ lớn dữ liệu cần gửi

HTTP HEADER & STATUS CODE

Dữ liệu trả về bao giờ cũng có phần thông tin header với dòng đầu tiên chưa Status Code **HTTP/1.1 200 OK** có nghĩa là status code = 200, request được trả về phù hợp. Theo sau đó là các cặp header chứa thông tin Server muốn trao đổi với Client, mà nếu là trình duyệt thì nó bị ẩn đi (người dùng bình thường không thể thấy). Các cặp header này định dạng theo kiểu **name: value** và kết thúc bằng ký tự xuống dòng không thấy bằng mắt thường (**0x0D 0x0A** hay **\r\n**). Trong ví dụ trên, thông tin header **Content-Type: text/html; charset=utf-8** báo cho trình duyệt biết rằng định dạng dữ liệu gửi về là dạng text, mã hóa utf-8. **Transfer-Encoding: chunked** chiều dài dữ liệu không được biết trước và gửi cho tới khi server đóng kết nối.

Một số HTTP status code thường thấy:

- **1xx**: mã trạng thái thông tin. Bản chất của mã trạng thái này, chỉ để thông báo với client rằng request đã được chấp nhận. Các mã trạng thái thông tin này được quy định trong **HTTP/1.1**, còn phiên bản **HTTP/1.0** hay trước đó thì không có, có thể bỏ qua phần mã trạng thái này. Mã hay gấp nhất:
 - **100 Continue**: thông báo cho client biết là có thể gửi tiếp phần request còn lại nếu còn hoặc kết thúc nếu đã hết. Nếu trong request POST, phần thân request lớn sẽ bị server từ chối, và để giải quyết điều này thì client phải gửi **Expect: 100-continue** theo sau phần header ban đầu
- **2xx**: nhóm mã trạng thái này thông báo với client rằng request đã được nhận, hiểu và xử lý thành công. Với một số mã thường thấy:
 - **200 OK**: thông báo cho client biết là request đã gửi thành công. Có thể thấy mã trạng thái này trong các phương thức **GET, HEAD, POST, TRACE**
 - **201 Created**: cho biết request đã xử lý thành công và tài nguyên đã được khởi tạo. Được sử dụng để xác nhận sự thành công của một request **PUT** hoặc **POST**

- **204 No Content:** thông báo không có phần thân message trong response
- **3xx:** nhóm mã trạng thái thông báo client còn phải thực hiện thêm hành động nữa, để hoàn thành request. Và mã trạng thái thường gặp trong nhóm:
 - **301 Moved Permanently:** thông báo tài nguyên được yêu cầu đã được chuyển hướng sang một URL mới, và server sẽ gửi URL mới này trong response cho client biết
- **4xx** nhóm mã thông báo các lỗi từ phía client: Được sử dụng khi server cho rằng phía client đang xảy ra lỗi, với một request, hoặc tài nguyên không hợp lệ, hoặc một request không đúng. Các mã thông dụng:
 - **400 Bad Request:** thông báo request đã gửi là sai
 - **401 Unauthorized:** chỉ ra rằng request cần được xác thực. Client có thể gửi lại request với header đã được xác thực. Trường hợp đã đính kèm header xác thực nhưng vẫn nhận được thông báo này tức là header xác thực chưa hợp lệ
 - **403 Forbidden:** server từ chối quyền truy cập của client
 - **404 Not Found:** thông báo tài nguyên không hợp lệ và tồn tại trên server
 - **409 Conflict:** server không thể hoàn thành yêu cầu vì client cố chỉnh sửa tài nguyên mới hơn so với **timestamp** của client. Xung đột xảy ra chủ yếu trong các request **PUT** trong quá trình hợp tác chỉnh sửa tài nguyên
- **5xx:** nhóm lệnh thông báo server đang ở trong tình trạng lỗi hoặc không có khả năng thực hiện yêu cầu. Một số mã thường gặp:
 - **500 Internal Server Error:** cho biết là không thể thực hiện request của client
 - **501 Not Implemented:** thông báo server không ra phương thức được yêu cầu hoặc không có khả năng hỗ trợ chức năng mà client đã request
 - **503 Service Unavailable:** Xảy ra khi hệ thống của server đang bão dưỡng hoặc quá tải

JSON

JSON (JavaScript Object Notation) là 1 định dạng hoán vị dữ liệu nhanh. Chúng dễ dàng cho chúng ta đọc và viết. Dễ dàng cho thiết bị phân tích và phát sinh. Chúng là cơ sở dựa trên tập hợp của Ngôn Ngữ Lập Trình JavaScript. JSON là 1 định dạng kiểu text mà hoàn toàn độc lập với các ngôn ngữ hoán chính, thuộc họ hàng với các ngôn ngữ họ hàng C, gồm có C, C++, C#, Java, JavaScript, Perl, Python, và nhiều ngôn ngữ khác. Những đặc tính đó đã tạo nên JSON 1 ngôn ngữ hoán vị dữ liệu lý tưởng.

JSON được xây dựng trên 2 cấu trúc:

Là tập hợp của các cặp tên và giá trị name-value. Trong những ngôn ngữ khác nhau, đây được nhận thấy như là 1 đối tượng (object), sự ghi (record), cấu trúc (struct), từ điển (dictionary), bảng băm (hash table), danh sách khoá (keyed list), hay mảng liên hợp. Là 1 tập hợp các giá trị đã được sắp xếp. Trong

hầu hết các ngôn ngữ, this được nhận thấy như là 1 mảng, véc tơ, tập hợp hay là 1 dãy sequence. Đây là 1 cấu trúc dữ liệu phổ dụng. Hầu như tất cả các ngôn ngữ lập trình hiện đại đều hỗ trợ chúng trong 1 hình thức nào đó. Chúng tạo nên ý nghĩa của 1 định dạng hoán vị dữ liệu với các ngôn ngữ lập trình cũng đã được cơ sở hoá trên cấu trúc này.

Cú pháp

- Dữ liệu nằm trong các cặp name/value
- Các dữ liệu được ngăn cách bởi dấu phẩy ,
- Các đối tượng (name/value) nằm giữa hai dấu ngoặc kép "
- Tất cả các đối tượng nằm bên trong hai dấu ngoặc nhọn {}
- Dữ liệu của JSON được viết theo từng cặp name/value. Một cặp name/value bao gồm trường name (nằm trong hai dấu ngoặc kép ", theo sau là dấu hai chấm :, và sau cùng là trường value (cũng được nằm trong hai dấu ngoặc kép ". Ví dụ: "name":"John"

```
{  
    "username" : "your-user-name",  
    "email" : "your-email@email.com",  
    "website" : "iota.edu.vn",  
    "title" : "IoT Stater Course"  
}
```

ỨNG DỤNG XEM GIÁ BITCOIN

Một ứng dụng đơn giản sử dụng giao thức HTTP để lấy tỉ giá Bitcoin (BTC)/USD từ các trang Web giao dịch, hiển thị lên màn hình OLED.

Chúng ta có rất nhiều nguồn lấy tỉ giá, một trong số đó là coinmarketcap.com/api/. Với tài liệu được cung cấp, và nhu cầu là chỉ lấy tỉ giá Bitcoin/USD, chúng ta chỉ cần ESP8266 gửi 1 HTTP Request đến api.coinmarketcap.com/v1/ticker/bitcoin/ thì sẽ nhận được một chuỗi JSON dạng như:

```
[  
  {  
    "id": "bitcoin",  
    "name": "Bitcoin",  
    "symbol": "BTC",  
    "rank": "1",  
    "price_usd": "4121.82",  
    "price_btc": "1.0",  
    "24h_volume_usd": "2070600000.0",  
    "market_cap_usd": "68082264895.0",  
    "available_supply": "16517525.0",  
    "total_supply": "16517525.0",  
    "percent_change_1h": "0.38",  
    "percent_change_24h": "0.97",  
    "percent_change_7d": "2.28",  
    "last_updated": "1503240569"  
  }  
]
```

và giá trị của trường `price_usd` chính là giá trị chúng ta muốn hiển thị.



Đa số các dịch vụ Web hiện nay đều sử dụng giao thức bảo mật **HTTPS**, cũng là HTTP, nhưng quá trình truyền nhận được mã hóa dữ liệu, thực hiện xác thực trước khi gửi giữa Client và Server.

CHẾ ĐỘ WIFI ACCESS POINT

ESP8266 HOẠT ĐỘNG Ở CHẾ ĐỘ ACCESS POINT

ESP8266 có khả năng cho phép các thiết bị khác (Station - STA) truy cập vào và hoạt động như là 1 Access Point, có thể tự thiết lập 1 mạng WiFi nội bộ, với khả năng khởi động DHCP Client và cung cấp được IP cho các Client kết nối tới. Do giới hạn về RAM, nên số lượng tối đa các STA có thể kết nối đến một ESP8266 hiện tại là 5.

TẠO RA MỘT MẠNG WIFI SỬ DỤNG ESP8266

Với đoạn code này, bạn có thể tạo ra một mạng WiFi cục bộ có SSID là **AP-XXXXXX** và có thể dùng máy tính để kết nối trực tiếp vào với password là **password**

```
#include <ESP8266WiFi.h>

const char *password = "password";

void setup() {
    Serial.begin(115200);
    Serial.print("Configuring access point...");
    char ssid[64];
    sprintf(ssid, "AP-%06X", ESP.getChipId());
    WiFi.softAP(ssid, password);

    IPAddress myIP = WiFi.softAPIP();
    Serial.print("AP IP address: ");
    Serial.println(myIP);
}

void loop() {
```

WEB SERVER

WEB SERVER LÀ GÌ?

Web Server là một máy chủ Web mà khi có bất kỳ một Web Client nào (chẳng hạn Web Browser) truy cập vào, thì nó sẽ căn cứ trên các thông tin yêu cầu truy cập để xử lý, và phản hồi lại nội dung. Đa phần các nội dung Web Server phục vụ là HTML, Javascript, CSS, JSON và bao gồm cả các dữ liệu Binary.

Mặc định các Web Server phục vụ trên Port 80, và 443 cho dịch vụ Web có bảo mật **HTTPS**

HTML - JAVASCRIPT - CSS

HTML, Javascript và CSS là ba ngôn ngữ để xây dựng và phát triển Web. Những hiểu biết cơ bản về chúng sẽ tạo điều kiện thuận lợi cho các quá trình tiếp theo sau được dễ dàng hơn.

HTML

Viết đầu tiên là **Hyper Text Markup Language** - ngôn ngữ đánh dấu siêu văn bản dùng để cấu trúc nội dung của một trang Web, ví dụ như: chỉ định các đoạn văn bản, tiêu đề, bảng dữ liệu, hoặc nhúng hình ảnh hoặc video vào Web. Mỗi trang web chứa một loạt các liên kết đến các trang khác được gọi là **hyperlinks** (siêu liên kết). Mỗi trang được tạo ra từ nhiều **tag** (thẻ) khác nhau, với cấu trúc một **tag** như sau

Cấu trúc 1 tag

```
<tagname> nội dung tag...</tagname>
```

- Các tag thường đi theo cặp, bắt đầu bởi **<tagname>** và kết thúc bằng **</tagname>**



Tag kết thúc phải có dấu gạch chéo / phía trước tên thẻ.

Trang HTML cơ bản có thể được thấy như sau:

```
<!DOCTYPE html>
<html>
<head>
    <title>Page Title</title>
</head>
<body>

    <h1>This is a Heading</h1>
    <p>This is a paragraph.</p>

</body>
</html>
```

Với một số tag cơ bản như sau:

- `<!DOCTYPE html>` cho biết là HTML5
- `<html>` root của trang HTML
- `<head>` chứa thông tin về tài liệu
- `<title>` phần tiêu đề trang
- `<body>` phần chứa nội dung trang hiển thị
- `<h1>` nơi chứa phần tiêu đề chính
- `<p>` phần ghi các đoạn văn bản

JAVASCRIPT

Javascript là một ngôn ngữ được thiết kế chủ yếu để thêm tương tác vào các trang Web, và tạo ra các ứng dụng Web.

Các chương trình Javascript có thể được nhúng trực tiếp vào HTML của Web. Và tùy vào mục đích cụ thể, script có thể chạy khi mở trang Web, nhấp chuột, gõ phím, gửi biểu mẫu, cập nhật dữ liệu, giao tiếp với cơ sở dữ liệu,...

Để nhúng chương trình viết bằng Javascript vào trang HTML, chỉ cần thêm tag `<script>` và thuộc tính `type`. Có thể thêm phần này ở phần `<head>` hoặc phần `<body>` của HTML. Ví dụ sau đây minh họa việc thêm một chương trình Javascript vào phần thân (`<body>`) của HTML:

```
<!DOCTYPE html>
<html>
<head>
    <title>My first JavaScript page</title>
</head>
<body>

    <script type="text/javascript">
        // chương trình Javascript được viết ở đây
    </script>

</body>
</html>
```

CSS

CSS là từ viết tắt của **Cascading Style Sheets**, là một ngôn ngữ được thiết kế để xử lý giao diện Web, giúp các trang Web được đẹp hơn. CSS có thể kiểm soát được màu sắc của văn bản, phong chữ, kích cỡ chữ, khoảng cách giữa các đoạn văn, hình nền hoặc màu nền, và nhiều hiệu ứng khác nữa.

Ví dụ : sử dụng phương thức Encoding UTF-8 ở phần `<head>` trong trang HTML

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title> Sử dụng phương thức Encoding UTF-8</title>
</head>
<body>

</body>
</html>
```

ỨNG DỤNG ĐIỀU KHIỂN ĐÈN LED QUA WEB SERVER

ESP8266 hoàn toàn có thể thực hiện vai trò Web Server để phục vụ cho một vài kết nối đến, tận dụng giao diện Web để điều khiển, cấu hình cho nó.

Ứng dụng này sẽ sử dụng giao diện Web phục vụ bởi ESP8266 và điều khiển chính đèn LED trên board.

CODE

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>

const char* ssid = ".....";
const char* password = ".....";
const int led = 16;
const char *html = \
"<html> \
<head> \
<title>ESP8266 Webserver</title> \
</head> \
<body> \
<a href=\"/on\">ON</a> \
<a href=\"/off\">OFF</a> \
</body> \
</html>";

ESP8266WebServer server(80);

void handleOn() {
    digitalWrite(led, 0);
    server.sendHeader("Location", "/");
    server.send(301);
}

void handleOff() {
    digitalWrite(led, 1);
    server.sendHeader("Location", "/");
    server.send(301);
}

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

void setup(void){
    pinMode(led, OUTPUT);
    digitalWrite(led, 0);
    Serial.begin(115200);
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.print("Connected to ");
    Serial.println(ssid);
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());

    server.on("/", handleRoot);
    server.on("/on", handleOn);
    server.on("/off", handleOff);
    server.begin();
    Serial.println("HTTP server started");
}

void loop(void){
    server.handleClient();
}
```

TRAO ĐỔI DỮ LIỆU GIỮA 2 ESP8266

YÊU CẦU

Ứng dụng kiến thức đã học thực hiện việc

HƯỚNG DẪN THỰC HIỆN

NHỮNG VƯƠNG MẮC THƯỜNG GẶP

TỔNG KẾT

Tổng kết chương

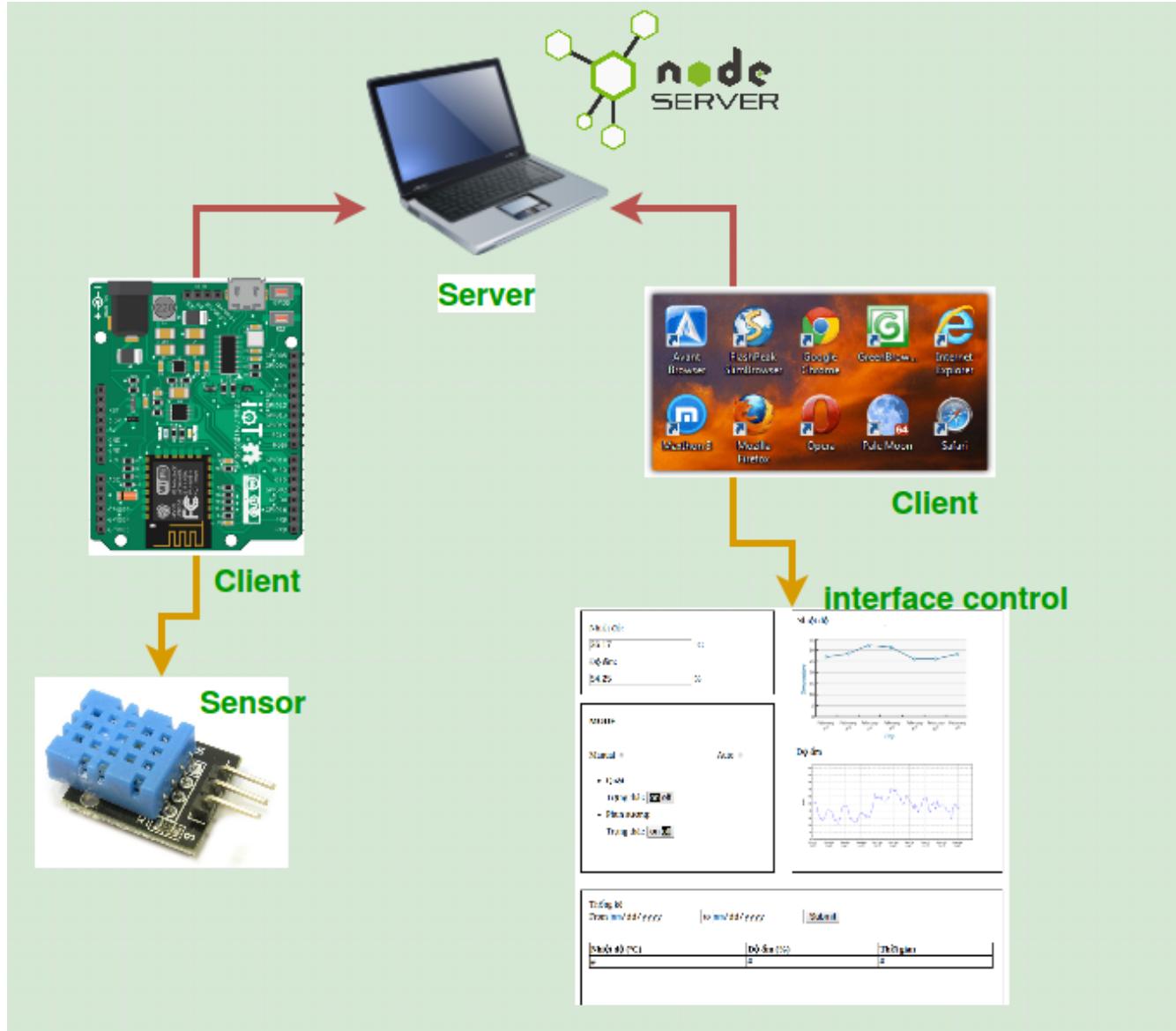
DỰ ÁN ĐỌC CẢM BIẾN DHT11 VÀ GỬI VỀ SERVER

Trong bài này chúng ta sẽ xây dựng ứng dụng dùng cảm biến DHT11 để thu thập nhiệt độ, độ ẩm của môi trường. Thông tin về nhiệt độ và độ ẩm sẽ được hiển thị trên máy tính và hiển thị trên trình duyệt web bằng cách truy cập vào 1 địa chỉ URL được chỉ định. Một số kiến thức cần thiết :

- **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
- **DHT11** là một cảm biến có khả năng đo nhiệt độ và độ ẩm không khí với độ chính xác vừa phải, giá cả phải chăng. Có thể lấy dữ liệu đo được của cảm biến bằng giao thức OneWire.

THIẾT KẾ ỨNG DỤNG

Hình ảnh bên dưới mô tả tổng quan dự án



Hình 31. Tổng quan mô hình của dự án

Trong thực tế, khi thiết kế ứng dụng, người dùng cần một giao diện giám sát và điều khiển thân thiện, đồng thời có thể phát triển thêm các tính năng như hiển thị kết quả dưới dạng đồ thị (chart), lưu trữ dữ liệu theo thời gian chỉ định hay điều khiển trạng thái các thiết bị chỉ với 1 click chuột trên máy tính. Các dự án với mô hình phức tạp sẽ cần quản lý các kết nối cũng như dữ liệu của các thiết bị...

Chúng ta sẽ giải quyết những vấn đề trên thông qua ứng dụng đọc nhiệt độ, độ ẩm của môi trường và gửi về server. Đây là một ứng dụng khá đơn giản, hữu ích và dễ làm. Thông qua phần này chúng ta có thể xây dựng được một ứng dụng IoT thực tế, nắm bắt được các kiến thức cơ bản về thu thập dữ liệu, xây dựng thiết bị và server.

YÊU CẦU

- Dùng cảm biến DHT11 để thu thập nhiệt độ, độ ẩm của môi trường và kết nối với board mạch ESP8266
- Board mạch ESP8266 sẽ kết nối không dây đến mạng WiFi và gửi dữ liệu về HTTP Server
- Phần cơ bản: HTTP Server hiển thị dữ liệu nhiệt độ, độ ẩm ra màn hình Log trên máy tính
- Phần nâng cao: HTTP Server lưu trữ dữ liệu, và cung cấp file HTML cho người dùng có thể xem qua Browser

PHÂN TÍCH

- Chúng ta cần 1 Web Server viết bằng Javascript, thực thi bởi Node.js, lắng nghe ở Port được chỉ định trên máy tính cá nhân. Ở đây là port 8000
- Máy tính phải có kết nối cùng mạng WiFi nội bộ với ESP8266 và cần biết địa chỉ IP của máy tính để ESP8266 có thể truy cập, ví dụ IP là **192.168.1.102**
- ESP8266 sau khi kết nối vào mạng WiFi nội bộ, sẽ tiến hành đọc thông số nhiệt độ, độ ẩm từ cảm biến DHT11 và gửi về Server sau mỗi 2 giây.
- Quá trình gửi 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.
- Ở phần nâng cao: – Web Server lưu trữ dữ liệu nhiệt độ, độ ẩm trong mảng, chứa ở bộ nhớ RAM – Web Server còn cung cấp 1 file **index.html** chứa mã Javascript có thể yêu cầu lấy dữ liệu nhiệt độ, độ ẩm lưu trong RAM, và hiển thị lên biểu đồ

KIẾN THỨC

Sẽ dễ dàng hơn nếu chúng ta có những kiến thức cơ bản về

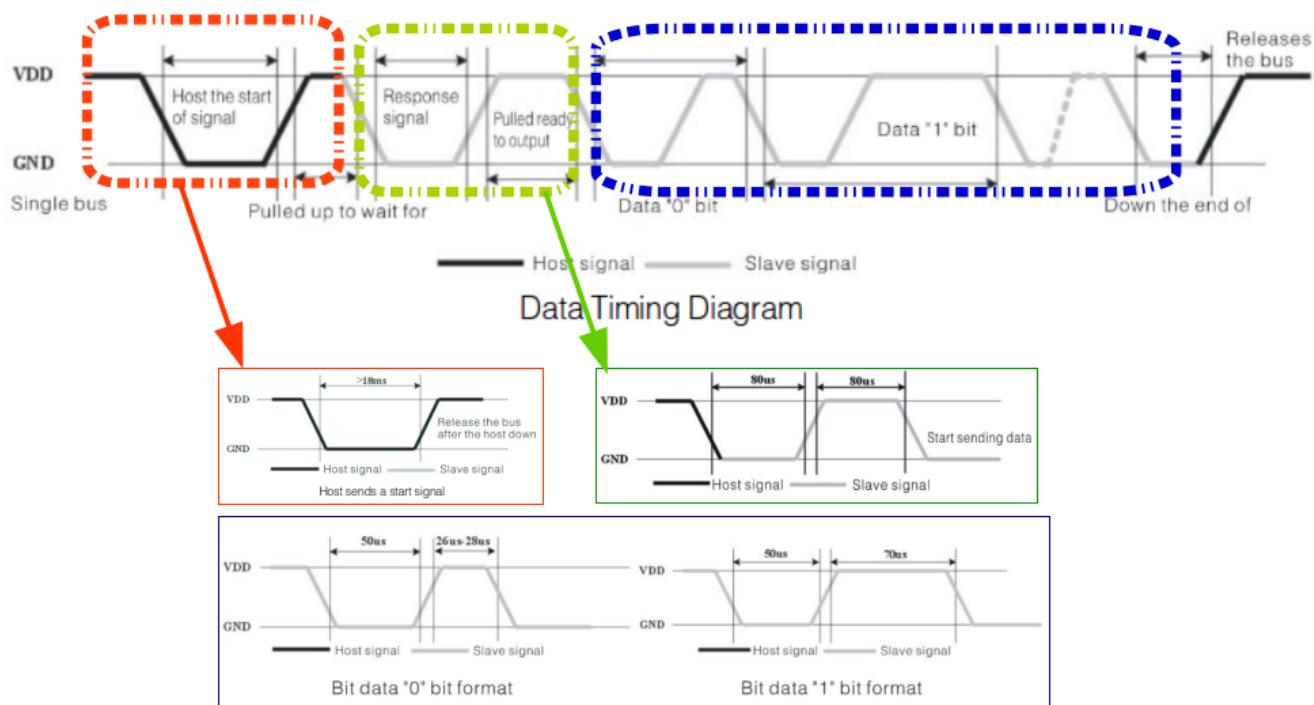
- Chuẩn truyền dữ liệu OneWire giữa các IC
- Ngôn ngữ Javascript để xây dựng server bằng cách dùng Node.js
- Ngôn ngữ HTML để xây dựng 1 trang html đơn giản nhằm hiển thị dữ liệu

Tuy nhiên cũng đừng quá lo lắng nếu bạn chưa từng dùng những thứ này, chúng ta sẽ hiểu nó khi đọc các phần tiếp theo.

Cảm biến DHT 11 và chuẩn dữ liệu OneWire

- DHT11 là cảm biến có chức năng đo nhiệt độ, độ ẩm của môi trường, được dùng khá phổ biến vì giá thành thấp và độ ổn định cao. Cảm biến sử dụng chuẩn truyền dữ liệu OneWire. Thông tin chi tiết về DHT11 có thể xem tại [Datasheet](#)
- 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 request (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 32. 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

Chi tiết về chuẩn OneWire xem tại maximintegrated.com

Ngôn ngữ HTML

Một trong những địa chỉ web để học HTML cho người mới bắt đầu là [w3school.com/HTML](https://www.w3schools.com/HTML), lưu ý rằng chúng ta sẽ không đi quá sâu vào việc học HTML, bởi việc này có thể ảnh hưởng đến tiến độ thực hiện của project, tại thời điểm này chúng ta chỉ cần học đủ để xây dựng project hoàn chỉnh.

Node.js và Javascript

Để tạo server dùng Node.js cần trang bị một số kiến thức cơ bản về Javascript và Node.js, để học Javascript chúng ta có thể truy cập địa chỉ URL w3school.com/Javascript, với Node.js thì codeschool.com thật sự hữu ích với người mới bắt đầu.

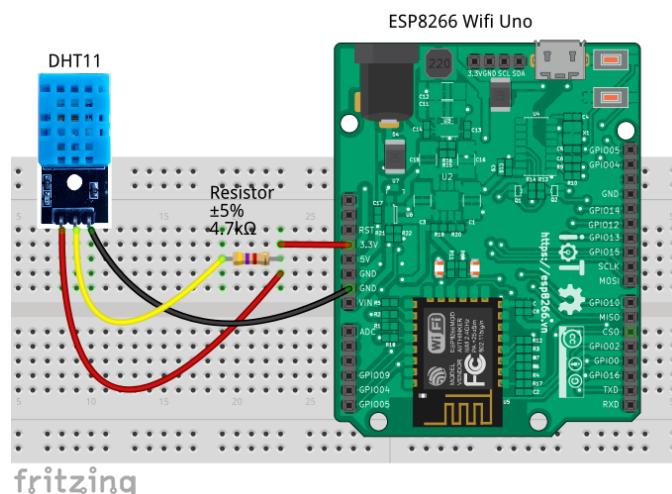
THỰC HIỆN

Linh kiện cần có

- Cảm biến DHT11
 - Board ESP8266 WiFi Uno
 - Dây nối male-female header
 - Điện trở 5K Ohm
 - Cable kết nối giữa board ESP8266 và máy tính

Đầu nối

Kết nối sơ đồ mạch điện như hình bên dưới



Hình 33. Kết nối DHT11 và ESP8266 WiFi Uno

SERVER NODEJS

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
- ***** Còn lại thì trả về file **index.html**
- Mảng dữ liệu lưu trữ có định dạng: `[{"temp": 25, "humd": 80, time: "time"}, ...]`

Mã nguồn file **server.js**

```

//-----①-----
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') {               ②
        var newData = {
            temp: queryData.temp,
            humd: queryData.humd,
            time: new Date()
        };
        db.push(newData);
        console.log(newData);
        response.end();
    }
    //-----③-----
} else if (pathname == '/get') {
    response.writeHead(200, {
        'Content-Type': 'application/json'
    });
    response.end(JSON.stringify(db));
    db = [];
}
//-----④-----
} else {                                     ④
    fs.readFile('./index.html', function(error, content) {
        response.writeHead(200, {
            'Content-Type': 'text/html'
        });
        response.end(content);
    });
}
//-----⑤-----
}

var server = http.createServer(requestHandler);
server.listen(8000);
console.log('Server listening on port 8000');

```

Giải thích mã nguồn

① 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 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

- ② So sánh giá trị pathname để xử lý 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.
- ③ 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)
- ④ Trả về nội dung của file index.html khi không xảy ra 2 trường hợp <2> và <3>. 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
- ⑤ Khởi tạo một server HTTP và mở port 8000 để các client truy cập

Bạn có thể truy cập đến đường dẫn của file server.js và thực thi đoạn code trên với dòng lệnh `node server.js`, sau đó thử truy cập vào `localhost:8000` để xem trang `index.html`. Hoặc truy cập vào `localhost:8000/update?temp=20&humd=60` để xem màn hình Log in ra kết quả nhiệt độ và độ ẩm.

```
{ 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 }
```

Ở phần cơ bản chúng ta chưa cần phải quan tâm đến file `index.html` và đoạn code Javascript trong đó. Cũng nhưng chưa quan tâm tới việc xử lý khi đường dẫn là `/get` hoặc `/*`, mà chỉ quan tâm duy nhất khi nhận được với đường dẫn `/update`.

Khi đã hoàn thành phần cơ bản chúng ta sẽ đi đến một ứng dụng khá phổ biến, người dùng cần hiển thị các dữ liệu thu thập một cách trực quan thông qua trình duyệt Web. Vì vậy chúng ta sẽ làm 1 file `index.html` chứa mã nguồn Javascript có thể yêu cầu Server trả về dữ liệu mỗi giây để hiển thị lên 1 biểu đồ canvas.

Mã nguồn file `index.html`

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>DHT11</title>
</head>
<!--
<link href="https://fonts.googleapis.com/css?family=Athiti|Saira+Semi+Condensed" rel="stylesheet">
<body style="font-family:'Saira Semi Condensed',sans-serif">
    <h1> 1. Cập nhật giá trị nhiệt độ, độ ẩm ở textbox</h1><br>
    <h2> Temperature</h2> <input type="text" size="6" id="temp">&#176;C<br>
    <h2> Humidity</h2> <input type="text" size="6" id="humd">%<br>
    <h1> 2. Đồ thị nhiệt độ, độ ẩm</h1><br>
<!--
<script type="text/javascript">
    function httpGetAsync(theUrl, callback) {
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function() {
```

①

②

```

        if (xmlHttp.readyState == 4 && xmlHttp.status == 200)                      ③
            callback(JSON.parse(xmlHttp.responseText));
    }
    xmlHttp.open("GET", theUrl, true); // true for asynchronous
    xmlHttp.send(null);
}

//----- -->
window.onload = function() {
    var dataTemp = [];
    var dataHumd = [];                                         ④

    var Chart = new CanvasJS.Chart("ChartContainer", {
        zoomEnabled: true,                                     // Dùng thuộc tính có thể zoom vào graph
        title: {
            text: "Temprature & Humidity"                // Viết tiêu đề cho graph
        },
        toolTip: {                                              // Hiển thị cùng lúc 2 trường giá trị nhiệt độ, độ ẩm trên graph
            shared: true
        },
        axisX: {
            title: "chart updates every 2 secs" // 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: "temp",                               // Tên dữ liệu hiển thị sẽ lấy từ dataTemp
                dataPoints: dataTemp
            },
            {
                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 "humd" ở mục chú thích (legend items)
                name: "humd",                               // Tên dữ liệu hiển thị sẽ lấy từ dataHumd
                dataPoints: dataHumd
            }
        ],
    });
    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 dataLength = 100;                                     // Số lượng điểm dữ liệu của nhiệt độ, độ ẩm có thể nhìn thấy được
    var time = new Date();                                    // Tạo biến time, set giá trị ban đầu và cập nhật thời gian cho trục X

    // starting at 11.55 am
    time.setHours(11);
    time.setMinutes(55);
    time.setSeconds(00);
    time.setMilliseconds(00);

    var updateChart = function(count) {
        console.log(count);
        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;

            // Xuất ra màn hình console trên browser giá trị nhận được từ localhost:8000/get
            console.log(data);
            for (var j = 0; j < count; j++) {

                // 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({
            x: time.getTime(),
            y: yTempVal
        });
        dataHumd.push({
            x: time.getTime(),
            y: yHumdVal
        });
    };
    Chart.render(); // chuyển đổi dữ liệu của graph thành mô hình đồ họa
};

updateChart(100); // Khởi tạo giá trị đầu tiên cho dữ liệu,
setInterval(function() { // Cập nhật lại giá trị graph sau thời gian updateInterval
    updateChart(dataLength)
}, updateInterval);
}

</script>

<!-- Nhúng file Javascript tại đường dẫn src để có thể xây dựng 1 graph --&gt;
&lt;script type="text/javascript" src="https://canvasjs.com/assets/script/canvasjs.min.js"&gt;&lt;/script&gt;

<!-- thiết lập kích thước cho graph thông qua id ChartContainer đã thiết lập ở trên --&gt;
&lt;div id="ChartContainer" style="height: 300px; width:80%;"&gt;&lt;/div&gt;
&lt;/body&gt;

&lt;/html&gt;
</pre>

```

Giải thích mã nguồn

① Những tag cơ bản khi khởi tạo 1 trang HTML

- <!DOCTYPE html> cho trình duyệt biết phiên bản HTML được sử dụng
- <html> cho trình duyệt biết đây là văn bản HTML
- <head> khai báo thông tin cho trang HTML
- <meta charset="UTF-8"> cung cấp dữ liệu về văn bản HTML, dạng mã hóa charset="UTF-8"
- <title>DHT11</title> tiêu đề của trang

② Tạo 2 textbox tại tag tiêu đề phụ <h1> để hiển thị nhiệt độ, độ ẩm. Mỗi textbox sẽ có 1 id để cập nhật giá nhiệt độ, độ ẩm từ server. Mã định dạng °C là của kí tự độ C

<3><4> Sử dụng CanvasJS Chart để vẽ biểu đồ nhiệt độ, độ ẩm.

Phân tích

- 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()`.



Cần phải có dữ liệu từ ESP8266 gửi lên server thì tại địa chỉ localhost:8000/get mới có dữ liệu.



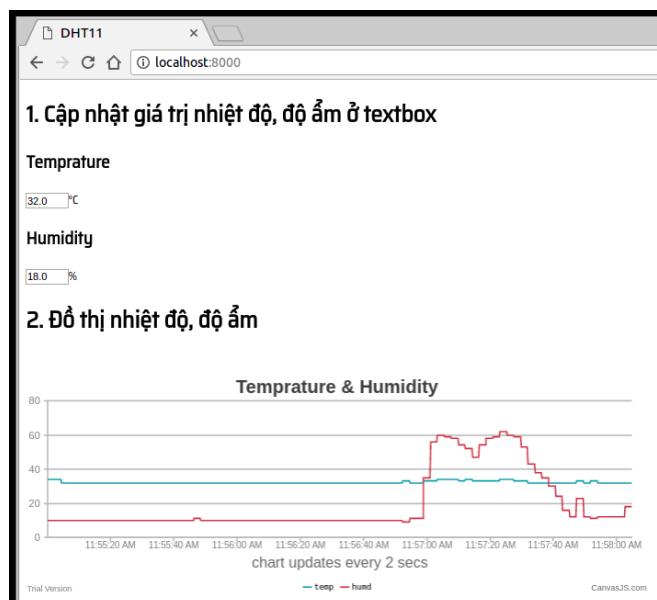
Hình 34. Hình ảnh dữ liệu từ địa chỉ `localhost:8000/get` ở định dạng JSON

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



Truy cập vào trang canvasjs.com/javascript-charts/ để lựa chọn và xây dựng những đồ thị phù hợp với mục đích của bạn.

Hình ảnh trang HTML sau khi xây dựng



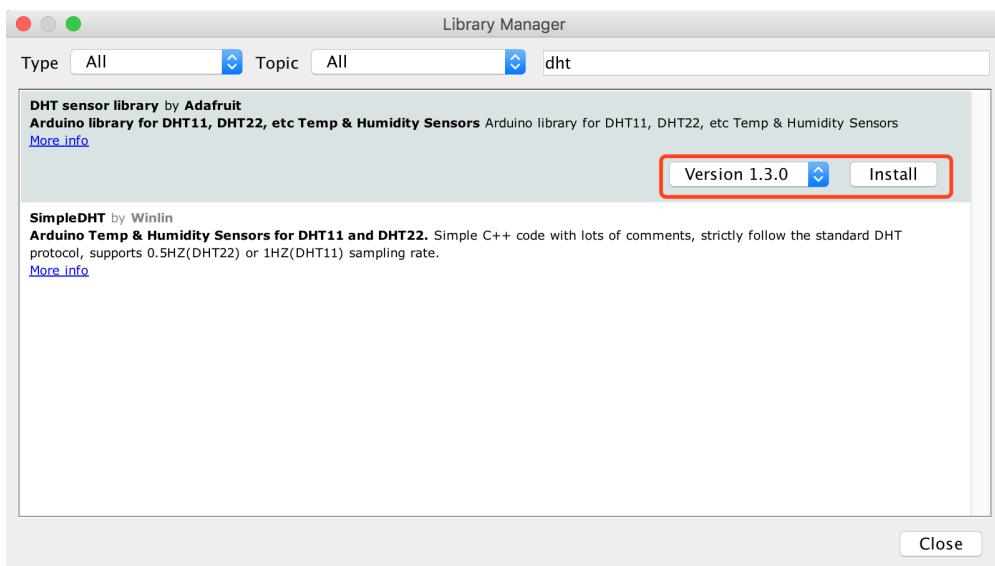
Hình 35. Hình ảnh giao diện HTML

CODE ESP8266

ESP8266 sử dụng thư viện `HTTPClient` để kết nối tới Web Server và lấy dữ liệu nhiệt độ, độ ẩm thông qua phương thức GET với query là `temp` và `humd`.

CHUẨN BỊ

- Cung cấp SSID và PASSWORD WiFi cho board mạch ESP8266 để kết nối vào mạng nội bộ với Web Server.
- Cung cấp địa chỉ IP, port của Web Server.
- Thư viện hỗ trợ lấy dữ liệu của DHT11. Dựa theo chuẩn truyền nhận 1 wire và sự phổ biến của dòng sensor DHTXX (DHT11, DHT22,...), có rất nhiều thư viện được xây dựng lên để việc lập trình với DHT11 trở nên dễ dàng hơn. Trong bài này chúng ta sẽ cài đặt và sử dụng thư viện **DHT sensor library** của Adafruit



Hình 36. Hình ảnh thư viện DHT sensor library

Mã nguồn ESP8266

```

#include <DHT.h>           // Khai báo sử dụng thư viện DHT
#include <ESP8266WiFi.h>    // Khai báo sử dụng thư viện ESP8266WiFi.h để thiết lập chế độ HTTP client cho ESP8266
#define DHTPIN 4            // Chân dữ liệu của DHT11 kết nối với GPIO4 của ESP8266
#define DHTTYPE DHT11        // Loại DHT được sử dụng

DHT dht(DHTPIN, DHTTYPE); // Tạo 1 biến client thuộc kiểu WiFiClient
WiFiClient client;        // WiFi.begin(ssid, password); // Tên mạng Wifi được chỉ định sẽ kết nối (SSID)
const char* ssid = "YOUR-WIFI-SSID"; // Password của mạng Wifi được chỉ định sẽ kết nối
const char* password = "YOUR-WIFI-PASS"; // Địa chỉ IP của máy khi truy cập cùng mạng WiFi
const char* server = "192.168.1.100"; // Port của server đã mở
const int port = 8000;           // Biến cập nhật dữ liệu sau mỗi 2s
const int sendingInterval = 2 * 1000;

void setup() {
  Serial.begin(115200);          // WiFi.begin(ssid, password); // Khởi tạo DHT11 để truyền nhận dữ liệu
  dht.begin();                  // WiFi.begin(ssid, password);
  Serial.println("Connecting"); // WiFi.begin(ssid, password);

  // Thiết lập ESP8266 là Station và kết nối đến Wifi. in ra dấu `.` trên terminal nếu chưa được kết nối
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(100);
  }
  Serial.println("\r\nWiFi connected");
}

void loop() {

// Đọc giá trị nhiệt độ (độ C), độ ẩm. Xuất ra thông báo lỗi và thoát ra nếu dữ liệu không phải là số
float temp = dht.readTemperature();
float humi = dht.readHumidity();
if (isnan(temp) || isnan(humi)) {
  Serial.println("Failed to read from DHT sensor!");
  return;
}

if (client.connect(server, port)) { // WiFi.begin(ssid, password); // Khởi tạo kết nối đến server thông qua IP và PORT đã mở
//-----
  String req_uri = "/update?temp=" + String(temp, 1) + "&humd=" + String(humi, 1);
  client.print("GET " + req_uri + " HTTP/1.1\r\n" + "Host: " + server + "\r\n" + "Connection: close\r\n" + "Content-Length: 0\r\n" + "\r\n");
//-----

// temp, humi chuyển từ định dạng float sang định dạng string và in ra màn hình serial      // terminal trên Arduino.
  Serial.printf("Nhiet do %s - Do am %s\r\n", String(temp, 1).c_str(), String(humi, 1).c_str());
}
client.stop(); // Ngắt kết nối đến server

delay(sendingInterval);
}

```

① ESP8266 sẽ gửi dữ liệu lên server sau khi kết nối thành công đến server thông qua lệnh `client.print()`. Nội dung gửi :

- GET /update?temp=30.6&humd=60 HTTP/1.1 với :
 - GET là phương thức gửi dữ liệu.
 - /update?temp=30.6&humd=60 là nội dung cần gửi bao gồm cả pathname và dữ liệu.
 - HTTP/1.1 khai báo sử dụng giao thức HTTP version 1.1 để có thể tạo 1 HTTP request

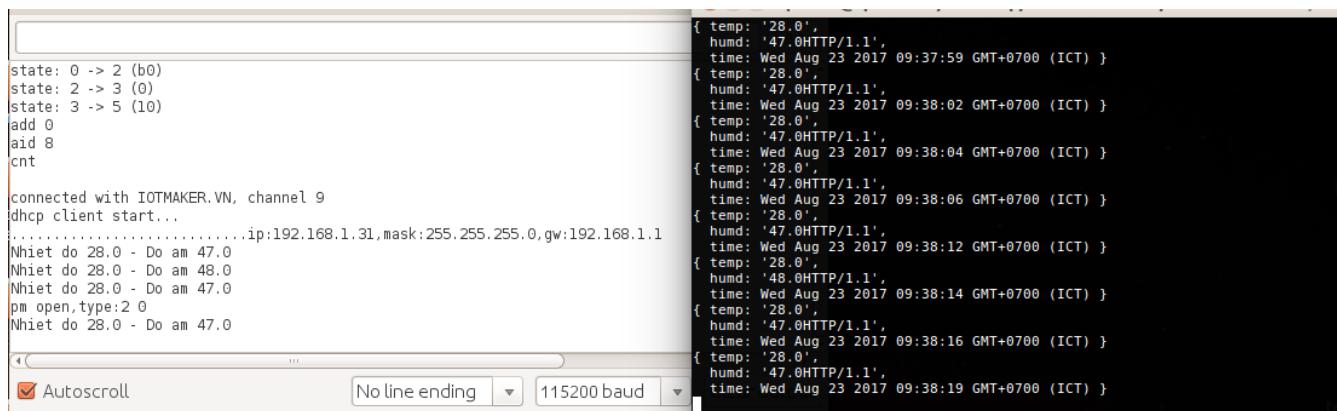
- Host: 192.168.1.7:8000 thông tin về địa chỉ IP và port của server
- Connection, Content-Length

Có thể xem thông tin của quá trình truyền nhận dữ liệu với lệnh `curl -v`

`http://192.168.1.7:8000/update?temp=28.0&humd=45.0`. Thông tin hiển thị như bên dưới

```
name@yourname:~$ curl -v http://192.168.1.7:8000/update?temp=28.0&humd=45.0
[1] 9277
name@yourname:~$ * Trying 192.168.1.7...
* Connected to 192.168.1.7 (192.168.1.7) port 8000 (#0)
> GET /update?temp=28.0 HTTP/1.1    // > Thông tin gửi từ ESP8266
> Host: 192.168.1.7:8000
> User-Agent: curl/7.47.0
> Accept: */*
>                                         // < Thông tin gửi từ server
< HTTP/1.1 200 OK
< Date: Wed, 23 Aug 2017 17:22:49 GMT
< Connection: keep-alive
< Content-Length: 0
<
* Connection #0 to host 192.168.1.7 left intact
```

Kết quả hiển thị trên Arduino IDE và màn hình Log của máy tính



Hình 37. Hình ảnh trên Arduino và terminal sau khi kết nối

TỔNG KẾT

Sau khi hoàn thành dự án, chúng ta đã có cái nhìn tổng quan về trình tự các bước để xây dựng một dự án hoàn chỉnh trong việc thu thập dữ liệu của cảm biến cũng như xây dựng server để quản lý các máy khách. Từ đó là bước đệm để giúp bạn phát triển thêm nhiều các dự án thu thập và xử lý dữ liệu trong tương lai.

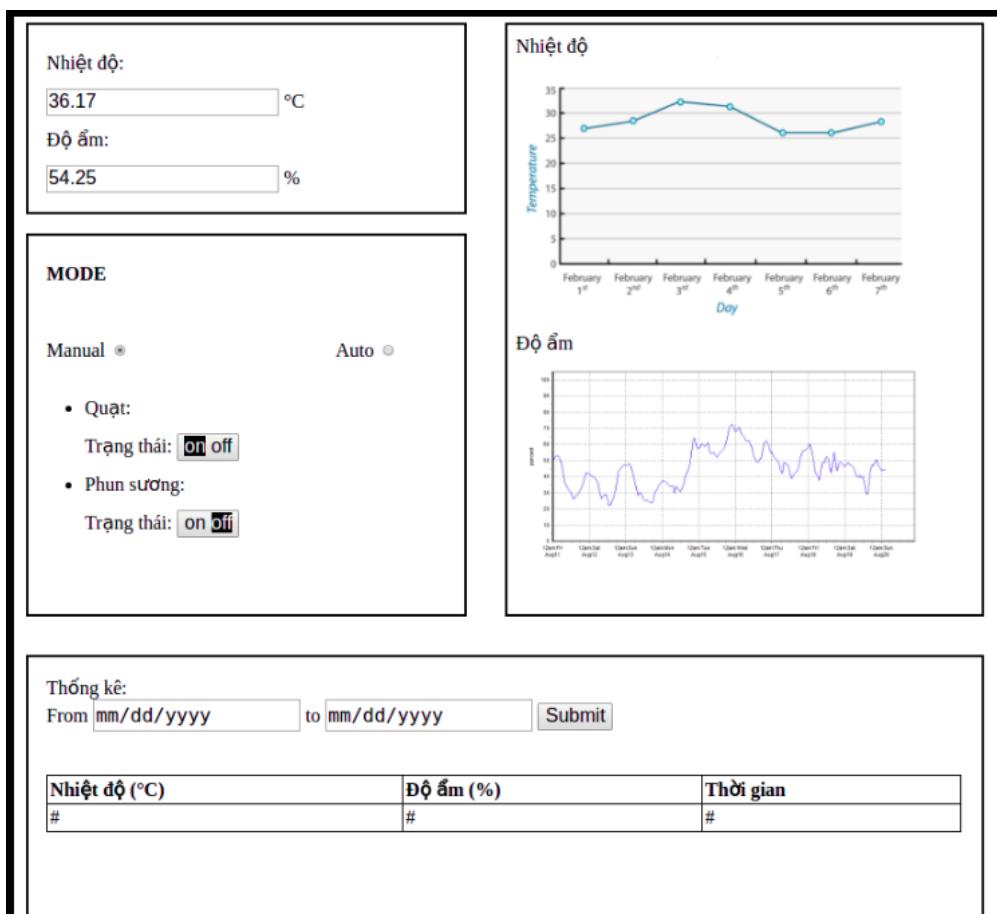
ỨNG DỤNG MỞ RỘNG

DÙNG ESP8266 NHƯ 1 WEB SERVER

Xây dựng 1 dự án giám sát và điều khiển nhiệt độ, độ ẩm hiển thị trên web với giao diện điều khiển :

- Hiển thị giá trị nhiệt độ, độ ẩm.
- Hiển thị chart nhiệt độ, độ ẩm theo thời gian.
- Có 2 chế độ Auto và Manual.
 - Với chế độ Auto, nhiệt độ > 35°C sẽ tự động bật quạt, độ ẩm > 50% sẽ bật máy phun sương.
 - Với chế độ manual có thể điều khiển quạt và máy phun sương bằng các nút nhấn ở ON/OFF
- Có tùy chọn hiển thị lịch sử nhiệt độ, độ ẩm theo thời gian từ ngày aa/bb/cccc đến ngày xx/yy/zzzz

Hình ảnh thiết kế giao diện như bên dưới:



Hình 38. Giao diện điều khiển trên trang HTML

CÁC CHẾ ĐỘ CẤU HÌNH WIFI

Thông thường, khi bắt đầu kết nối wifi cho ESP8266, ta phải cấu hình cho thiết bị các thông số của Access Point cũng như SSID và password nếu mạng wifi được thiết lập các bảo mật như WEP/WPA/WPA2. Tuy nhiên, các ứng dụng nhúng sử dụng Wi-fi thường không chú trọng đến giao diện người dùng (user interface), ví dụ như bàn phím hay touchscreen... Vì thế, mỗi khi muốn kết nối thiết bị ESP với một AP nào đó, bạn cần phải có một máy tính đã cài đặt sẵn chương trình biên dịch để có thể cấu hình thông số cho thiết bị rồi nạp lại code xuống ESP thông qua cáp USB (mà không phải lúc nào cũng có sẵn).

Điều này làm cho việc kết nối wifi trở nên khá bất tiện và phức tạp. Do vậy ESP8266 cung cấp các phương pháp thay thế khác giúp đơn giản hóa việc kết nối trạm ESP (chế độ Station) với một điểm truy cập. Đó là kết nối bằng SmartConfig, WPS hay Wifi Manager.

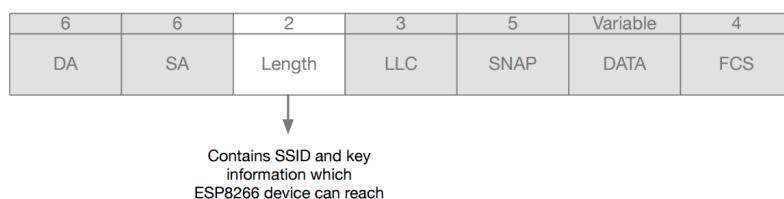
SMARTCONFIG

KIẾN THỨC

SmartConfig là công nghệ được tạo ra nhằm cấu hình cho các thiết bị cho phép kết nối với mạng wifi một cách dễ dàng nhất bằng smart phone. Nói một cách đơn giản, để kết nối wifi cho thiết bị ESP8266, ta chỉ cần cung cấp thông tin mạng wifi (bao gồm SSID và password) cho ESP thông qua 1 ứng dụng trên smart phone.

Ban đầu, thiết bị ESP chưa được kết nối vào wifi nên ứng dụng này không thể trực tiếp gửi các thông tin của mạng wifi cho thiết bị. Thay vào đó, ứng dụng SmartConfig sẽ mã hóa các thông tin này vào trong trường Length của gói tin UDP, rồi gửi gói tin này cho các Access Point thông qua giao thức ESP-TOUCH. Gói tin sẽ được truyền lặp đi lặp lại theo chu kỳ trong mạng. Thiết bị ESP (lúc này đang ở chế độ sniffer) sẽ theo dõi các gói tin được truyền trong mạng, nó sẽ phát hiện và giải mã thông tin từ các gói tin này để có thể kết nối vào mạng wifi. Sau khi cấu hình xong, kết quả trả về từ ứng dụng là địa chỉ IP của thiết bị ESP vừa kết nối.

Như vậy, ta có thể dễ dàng kết nối wifi cho thiết bị ESP8266 mà không cần phải khai báo lại thông số trong chương trình hay nạp lại firmware.



Hình 39. Gói tin UDP

Lưu ý:

- Khoảng cách giữa thiết bị và router càng xa thì thời gian kết nối sẽ càng lớn.
- Nếu thiết bị không thể kết nối với router trong khoảng thời gian quy định thì ứng dụng sẽ trả về thông báo cấu hình thất bại. Người dùng có thể cài đặt thời gian timeout này thông qua lệnh `esptouch_set_timeout(uint8 time_s)`
- Trong quá trình cấu hình kết nối thiết bị bằng SmartConfig, thiết bị nên tắt các chế độ Station và Soft-AP.
- Người dùng có thể cấu hình cho nhiều thiết bị kết nối chung vào một router cùng lúc.
- ESP Touch hiện nay hỗ trợ đối với Access Point chuẩn 802.11n 2.4Ghz

CODE

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <Ticker.h>
#include <time.h>

#define PIN_LED 16
#define PIN_BUTTON 0

#define LED_ON() digitalWrite(PIN_LED, HIGH)
#define LED_OFF() digitalWrite(PIN_LED, LOW)
#define LED_TOGGLE() digitalWrite(PIN_LED, digitalRead(PIN_LED) ^ 0x01)

Ticker ticker;

/// Xét trạng thái hiện tại của button
bool longPress()
{
    static int lastPress = 0;
    if (millis() - lastPress > 3000 && digitalRead(PIN_BUTTON) == 0) { // Nếu button được nhấn và giữ trong 3s thì
        return true; // trả về "true".
    } else if (digitalRead(PIN_BUTTON) == 1) { // Nếu button không được nhấn hay nhấn không giữ
        // 3s
        lastPress = millis(); // gán biến lastPress bằng thời điểm hiện tại
    }
    return false; // gọi hàm, và trả về "false"
}

void tick()
{
    int state = digitalRead(PIN_LED); // Lấy trạng thái hiện tại của LED (GPIO16)
    digitalWrite(PIN_LED, !state); // Đảo trạng thái LED.
}

bool in_smartconfig = false; // biến trạng thái kiểm tra thiết bị có đang trong smartconfig hay không.

/// Vào chế độ Smartconfig
void enter_smartconfig()
{
    if (in_smartconfig == false) { // Nếu thiết bị không ở trong chế độ smartconfig
        in_smartconfig = true; // Gán biến trạng thái là đang trong chế độ smartconfig
        ticker.attach(0.1, tick); // Gọi hàm tick sau 0.1s. (Đảo trạng thái led)
        WiFi.beginSmartConfig(); // Bắt đầu chế độ smartconfig
    }
}

// Thoát chế độ smartconfig
void exit_smart()
{
    ticker.detach(); // Bật LED
    LED_ON(); // Gán biến trạng thái trở về ban đầu.
    in_smartconfig = false;
}

void setup() {
    Serial.begin(115200); // Cài đặt các thông số ban đầu
    Serial.setDebugOutput(true); // tốc độ baud = 115200
    // hiển thị các thông tin debug hệ thống lên màn hình qua serial

    pinMode(PIN_LED, OUTPUT); //Cấu hình GPIO cho các chân LED và button
    pinMode(PIN_BUTTON, INPUT); // LED là chân output và button là input
    ticker.attach(1, tick); // Gọi hàm tick sau 1s
    Serial.println("Setup done"); // Thông báo "Setup done" ra màn hình
```

```
}

/// Chương trình chính
void loop() {
    if (longPress()) { // Gọi hàm longPress kiểm tra trạng thái button
        enter_smartconfig(); // Nếu button được nhấn giữ trong 3s thì vào trạng thái smartconfig và
        Serial.println("Enter smartconfig"); // in thông báo "Enter smartconfig" ra màn hình
    }
    if (WiFi.status() == WL_CONNECTED && in_smartconfig && WiFi.smartConfigDone()) { // Kiểm tra trạng thái kết nối wifi,
        các thông số cấu hình cũng như trạng thái smartconfig
        exit_smart(); // khi thiết bị đã kết nối wifi thành công, thoát trạng thái smartconfig
        Serial.println("Connected, Exit smartconfig"); // thông báo ra màn hình.
    }
    if (WiFi.status() == WL_CONNECTED) {
        //do something
    }
}
```

WPS

WPS LÀ GÌ?

Nếu đã từng cấu hình cho một router wifi, sẽ gặp qua các thuật ngữ WPS trong các menu cấu hình của router. Hoặc từng nhìn thấy một nút nhấn trên các router với chữ viết bên cạnh WPS. Vậy WPS là gì? Quá trình thực hiện kết nối như thế nào? Cũng như thực hiện WPS với ESP8266, là những những nội dung sẽ được nói đến ở phần này.

WPS là từ viết tắt của Wifi Protected Setup, một phương thức giúp việc kết nối với mạng không dây giữa router và thiết bị kết nối không dây một cách nhanh chóng và dễ dàng, thay vì làm một cách thủ công: tìm mạng wifi cần kết nối và nhập mật khẩu để vào mạng wifi. WPS chỉ hoạt động khi cả hai thiết bị là router và thiết bị cần kết nối đến router có hỗ trợ chuẩn bảo mật cá nhân WPA/WPA2.

WPS có ba chế độ hoạt động: chế độ kết nối với mã PIN, chế độ kết nối bằng nút nhấn, và chế độ kết nối NFC - Near Field Communication (chưa phổ biến). Một trong những chế độ phổ biến và sẽ thực hiện trong phần này là chế độ kết nối bằng nút nhấn.

Ở chế độ kết nối bằng nút nhấn, điều trước tiên cần thực hiện:

- Nhấn nút WPS trên router, để giúp router vào chế độ bảo mật đặc biệt, ở chế độ này router sẽ cho phép các yêu cầu kết nối đến router từ các thiết bị WPS (các thiết bị có hỗ trợ WPS).
- Tiếp theo là nhấn nút nhấn ở thiết bị WPS. Nút nhấn này giúp thiết bị WPS kết nối đến router, việc kết nối này có thể thất bại nếu quá thời gian. Thời gian này được nhà sản xuất các thiết bị hỗ trợ chế độ này quy định, khoảng từ 1 phút đến 5 phút.

THỰC HIỆN WPS VỚI ESP8266

ESP8266 hỗ trợ hàm `WiFi.beginWPSCConfig()` trong thư viện `ESP8266WiFi`. Với hàm này giúp ESP8266 vào chế độ cấu hình với WPS và kết nối đến mạng wifi của router. Ví dụ này ESP8266 sẽ được đưa sẵn vào chế độ WPS, mà không cần thêm nút nhấn nào.

- WPS chỉ có thể thực hiện khi ESP8266 ở chế độ STA (Station)
- Router phải ở trong chế độ WPS trước

CODE

```
#include <ESP8266WiFi.h>

void setup()
{
    Serial.begin(115200);
    WiFi.mode(WIFI_STA);
    WiFi.begin("", "");
    delay(4000);
    // Check if WiFi is already connected and if not, begin the WPS process.
    while (WiFi.status() != WL_CONNECTED)
    {
        Serial.println("\nAttempting connection ...");
        WiFi.beginWPSConfig();
        delay(6000);
    }
    Serial.println("\nConnection already established.");
    Serial.println(WiFi.localIP());
    Serial.println(WiFi.SSID());
    Serial.println(WiFi.macAddress());
}

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

WIFI MANAGER

WiFiManager là một phương pháp khác giúp ESP8266 có thể vào được wifi mà không cần phải chỉnh sửa lại chương trình và nạp lại chương trình cho chip ESP8266. Với tính năng này, việc cấu hình wifi cho ESP8266 cũng hết sức dễ dàng, với những thiết bị quen thuộc (điện thoại thông minh, laptop) thông qua giao diện web.

HOẠT ĐỘNG CỦA WIFI MANAGER

- Khi ESP8266 khởi động, ESP8266 sẽ vào chế độ STA và sẽ tự động kết nối đến một AP với các thông tin kết nối này đã được lưu vào ESP8266 ở lần kết nối trước đó.
- Nếu như kết nối không thành công (có thể là AP lần trước không còn nữa, hoặc chưa có thông tin của bất cứ AP nào trong ESP8266), thì ESP8266 lúc này sẽ vào chế độ AP với một DNS (nếu không thiết lập thì DNS sẽ là tên mặc định của ESP8266, hoặc có thể đặt lại) và khởi động Web Server (với địa chỉ mặc định là 192.168.4.1)
- Sử dụng các thiết bị có hỗ trợ wifi, và có trình duyệt web (điện thoại thông minh, laptop, máy tính bảng...) để kết nối đến AP của ESP8266 vừa mới tạo ra. Có thể thấy một giao diện (với tên AP của ESP8266 là mặc định và không cần đặt mật khẩu cho ESP8266 AP) tương tự như sau :
 - Sau khi vào được giao diện của ESP8266 AP, chọn mục cấu hình cho wifi cho ESP8266 (như ví dụ trên là Configure WiFi hoặc Configure WiFi [No Scan]), có thể sẽ thấy giao diện tiếp theo tương tự như sau :
 - Chọn mạng wifi cần kết nối và nhập mật khẩu để vào wifi.
 - Nếu ESP8266 kết nối thành công, ta sẽ không thấy tên của ESP8266 AP nữa. Nếu chưa thành công thì chỉ cần kết nối lại ESP8266 AP và cấu hình lại.

CHUẨN BỊ

- Cài đặt thư viện: github.com/tzapu/WiFiManager

CODE

```

#include <ESP8266WiFi.h>           //https://github.com/esp8266/Arduino

//needed for library
#include <DNSServer.h>
#include <ESP8266WebServer.h>
#include "WiFiManager.h"           //https://github.com/tzapu/WiFiManager

void configModeCallback (WiFiManager *myWiFiManager)
{
    Serial.println("Entered config mode");
    Serial.println(WiFi.softAPIP());
    //if you used auto generated SSID, print it
    Serial.println(myWiFiManager->getConfigPortalSSID());
}

void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);

    //WiFiManager
    //Local intialization. Once its business is done, there is no need to keep it around
    WiFiManager wifiManager;
    //reset settings - for testing
    //wifiManager.resetSettings();

    //set callback that gets called when connecting to previous WiFi fails, and enters Access Point mode
    wifiManager.setAPCallback(configModeCallback);

    //fetches ssid and pass and tries to connect
    //if it does not connect it starts an access point with the specified name
    //here "AutoConnectAP"
    //and goes into a blocking loop awaiting configuration
    if (!wifiManager.autoConnect())
    {
        Serial.println("failed to connect and hit timeout");
        //reset and try again, or maybe put it to deep sleep
        ESP.reset();
        delay(1000);
    }

    //if you get here you have connected to the WiFi
    Serial.println("connected...yeey :)");
}

void loop()
{
    // put your main code here, to run repeatedly:
}

```

MQTT

MQTT LÀ GÌ?



MQTT (Message Queuing Telemetry Transport) là một giao thức gởi dạng publish/subscribe sử dụng cho các thiết bị Internet of Things với băng thông thấp, độ tin cậy cao và khả năng được sử dụng trong mạng lưới không ổn định.

Bởi vì giao thức này sử dụng băng thông thấp trong môi trường có độ trễ cao nên nó là một giao thức lý tưởng cho các ứng dụng M2M

MQTT cũng là giao thức sử dụng trong Facebook Messenger

Và MQTT là gì? Để có một cái nhìn toàn diện hoặc định nghĩa chi tiết, chỉ cần google "what is mqtt", "mqtt slides" ... Ở đây chúng ta chỉ nói ngắn gọn thôi, đủ để hiểu giao thức MQTT, bao gồm các định nghĩa "subscribe", "publish", "qos", "retain", "last will and testament (lwt)"

PUBLISH, SUBSCRIBE

Trong một hệ thống sử dụng giao thức MQTT, nhiều node trạm (gọi là mqtt client - gọi tắt là client) kết nối tới một MQTT server (gọi là broker). Mỗi client sẽ đăng ký một vài kênh (topic), ví dụ như "/client1/channel1", "/client1/channel2". Quá trình đăng ký này gọi là "**subscribe**", giống như chúng ta đăng ký nhận tin trên một kênh Youtube vậy. Mỗi client sẽ nhận được dữ liệu khi bất kỳ trạm nào khác gởi dữ liệu vào kênh đã đăng ký. Khi một client gởi dữ liệu tới kênh đó, gọi là "**publish**".

QoS

Ở đây có 3 tùy chọn **QoS** (Qualities of service) khi "publish" và "subscribe":

- **QoS0** Broker/client sẽ gởi dữ liệu đúng 1 lần, quá trình gởi được xác nhận bởi chỉ giao thức TCP/IP, giống kiểu đem con bỏ chợ.
- **QoS1** Broker/client sẽ gởi dữ liệu với ít nhất 1 lần xác nhận từ đầu kia, nghĩa là có thể có nhiều hơn 1 lần xác nhận đã nhận được dữ liệu.
- **QoS2** Broker/client đảm bảo khi gởi dữ liệu thì phía nhận chỉ nhận được đúng 1 lần, quá trình này phải trải qua 4 bước bắt tay.

Xem thêm QoS: code.google.com/p/mqtt4erl/wiki/QualityOfServiceUseCases

Một gói tin có thể được gởi ở bất kỳ QoS nào, và các client cũng có thể subscribe với bất kỳ yêu cầu QoS nào. Có nghĩa là client sẽ lựa chọn QoS tối đa mà nó có để nhận tin. Ví dụ, nếu 1 gói dữ liệu được

publish với QoS2, và client subscribe với QoS0, thì gói dữ liệu được nhận về client này sẽ được broker gửi với QoS0, và 1 client khác đăng ký cùng kênh này với QoS 2, thì nó sẽ được Broker gửi dữ liệu với QoS2.

Một ví dụ khác, nếu 1 client subscribe với QoS2 và gói dữ liệu gửi vào kênh đó publish với QoS0 thì client đó sẽ được Broker gửi dữ liệu với QoS0. QoS càng cao thì càng đáng tin cậy, đồng thời độ trễ và băng thông đòi hỏi cũng cao hơn.

RETAIN

Nếu RETAIN được set bằng 1, khi gói tin được publish từ Client, Broker **PHẢI** lưu trữ lại gói tin với QoS, và nó sẽ được gửi đến bất kỳ Client nào subscribe cùng kênh trong tương lai. Khi một Client kết nối tới Broker và subscribe, nó sẽ nhận được gói tin cuối cùng có RETAIN = 1 với bất kỳ topic nào mà nó đăng ký trùng. Tuy nhiên, nếu Broker nhận được gói tin mà có QoS = 0 và RETAIN = 1, nó sẽ huỷ tất cả các gói tin có RETAIN = 1 trước đó. Và phải lưu gói tin này lại, nhưng hoàn toàn có thể huỷ bất kỳ lúc nào.

Khi publish một gói dữ liệu đến Client, Broker phải set RETAIN = 1 nếu gói được gửi như là kết quả của việc subscribe mới của Client (giống như tin nhắn ACK báo subscribe thành công). RETAIN phải bằng 0 nếu không quan tâm tới kết quả của việc subscribe.

LWT

Gói tin LWT (last will and testament) không thực sự biết được Client có trực tuyến hay không, cái này do gói tin KeepAlive đảm nhận. Tuy nhiên gói tin LWT như là thông tin điều gì sẽ xảy đến sau khi thiết bị ngoại tuyến.

Một ví dụ

Tôi có 1 cảm biến, nó gửi những dữ liệu quan trọng và rất không thường xuyên. Nó có đăng ký trước với Broker một tin nhắn lwt ở topic `/node/gone-offline` với tin nhắn `id` của nó. Và tôi cũng đăng ký theo dõi topic `/node/gone-offline`, sẽ gửi SMS tới điện thoại mỗi khi nhận được tin nhắn nào ở kênh mà tôi theo dõi. Trong quá trình hoạt động, cảm biến luôn giữ kết nối với Broker bởi việc luôn gửi gói tin keepAlive. Nhưng nếu vì lý do gì đó, cảm biến này chuyển sang ngoại tuyến, kết nối tới Broker timeout do Broker không còn nhận được gói keepAlive. Lúc này, do cảm biến của tôi đã đăng ký LWT, do vậy broker sẽ đóng kết nối của Cảm biến, đồng thời sẽ publish một gói tin là Id của cảm biến vào kênh `/node/gone-offline`, dĩ nhiên là tôi cũng sẽ nhận được tin nhắn báo cái Cảm biến yêu quý của mình đã ngoại tuyến.

Ngắn gọn

Ngoài việc đóng kết nối của Client đã ngoại tuyến, gói tin LWT có thể được định nghĩa trước và được gửi bởi Broker tới kênh nào đó khi thiết bị đăng ký LWT ngoại tuyến.

MQTT CLIENT

MQTT BROKER

ỨNG DỤNG THỰC TẾ

WEBSOCKET

WEBSOCKET LÀ GÌ?

WebSoket là công nghệ hỗ trợ giao tiếp hai chiều giữa client và server bằng cách sử dụng một TCP socket để tạo một kết nối liên tục, hiệu quả và ít tốn kém. Mặc dù được thiết kế để chuyên dụng cho các ứng dụng web, lập trình viên vẫn có thể đưa chúng vào bất kỳ loại ứng dụng nào.

WebSockets mới xuất hiện trong HTML5, cho phép các kênh giao tiếp song song hai chiều và hiện đã được hỗ trợ trong nhiều trình duyệt (Firefox, Google Chrome và Safari). Kết nối được mở thông qua một HTTP request (yêu cầu HTTP), với những header đặc biệt thông báo cho Server (có hỗ trợ) chuyển sang kết nối Websocket. Kết nối này được duy trì để bạn có thể gửi và nhận dữ liệu bằng một cách liên tục, không đứt quãng, và không cần bất kỳ HTTP header (overhead) nào nữa.

ƯU ĐIỂM

WebSockets cung cấp khả năng giao tiếp hai chiều với kết nối được duy trì, có độ trễ thấp, giúp Server dễ dàng giao tiếp với Client.

NHƯỢC ĐIỂM

ỨNG DỤNG WEBSOCKET

Trong phần này, chúng ta sẽ thiết lập ứng dụng sử dụng Websocket để cập nhật trạng thái nút nhấn thời gian thực.

YÊU CẦU

Khởi động 1 Webserver (có hỗ trợ Websocket) trên chip ESP8266, khi truy cập vào địa chỉ IP của ESP8266 sẽ trả về 1 file HTML, chứa nội dung của đoạn Javascript thiết lập kết nối Websocket đến ESP8266, và lắng nghe các gói tin từ ESP8266. Khi nhấn nút trên board ESP8266 sẽ gửi nội dung trạng thái (đối lập) về Web Browser và thanh đổi nội dung hiển thị. Đồng thời khi nhấn nút trên trình duyệt sẽ thay đổi trạng thái đèn LED trên board ESP8266

CHUẨN BỊ

- Cài đặt thư viện github.com/me-no-dev/ESPAsyncWebServer

ĐOẠN CODE JAVASCRIPT ĐỂ TẠO KẾT NỐI WEB SOCKET

NHỮNG FILE HTML CHỨA ĐOẠN CODE JS VÀO ESP8266

Unresolved directive in index.adoc - include::doc/08-fota/00-index.adoc[]

CHEATSHEET

C - CHEATSHEET

```
/* Số và hệ số */
int binary_number = 0b11111111;
int hex_number = 0xFF;
int dec_nubmer = 200;
int octa_number = 0377;
float float_number = 88.0f;
float double_number = 88.0;
int negative_number = -1;
int positive_number = 1; // or +1

/* Đặt tên biến */
int x; // một biến
int x = 1; // biến được khai báo và khởi tạo
float x, y, z; // nhiều biến cùng kiểu dữ liệu
const int x = 88; // biến tĩnh, không ghi được
int tenBien1ok; // đặt tên biến này đúng
int ten_bien_nay_ok;

/* Đặt tên sai */
int 2001_tensai; // vì số ở đầu
int ten-sai; // dấu '-' không phải là alphanumeric
int while; // sai, vì dùng từ khóa vòng lặp while
```

```
/* Số và hệ số */
int binary_number = 0b11111111;
int hex_number = 0xFF;
int dec_nubmer = 200;
int octa_number = 0377;
float float_number = 88.0f;
float double_number = 88.0;
int negative_number = -1;
int positive_number = 1; // or +1

/* Đặt tên biến */
int x; // một biến
int x = 1; // biến được khai báo và khởi tạo
float x, y, z; // nhiều biến cùng kiểu dữ liệu
const int x = 88; // biến tĩnh, không ghi được
int tenBien1ok; // đặt tên biến này đúng
int ten_bien_nay_ok;

/* Đặt tên sai */
int 2001_tensai; // vì số ở đầu
int ten-sai; // dấu '-' không phải là alphanumeric
int while; // sai, vì dùng từ khóa vòng lặp while
```

```
/* Số và hệ số */
int binary_number = 0b11111111;
int hex_number = 0xFF;
int dec_nubmer = 200;
int octa_number = 0377;
float float_number = 88.0f;
float double_number = 88.0;
int negative_number = -1;
int positive_number = 1; // or +1

/* Đặt tên biến */
int x; // một biến
int x = 1; // biến được khai báo và khởi tạo
float x, y, z; // nhiều biến cùng kiểu dữ liệu
const int x = 88; // biến tĩnh, không ghi được
int tenBien1ok; // đặt tên biến này đúng
int ten_bien_nay_ok;

/* Đặt tên sai */
int 2001_tensai; // vì số ở đầu
int ten-sai; // dấu '-' không phải là alphanumeric
int while; // sai, vì dùng từ khóa vòng lặp while
```

```
/* Số và hệ số */
int binary_number = 0b11111111;
int hex_number = 0xFF;
int dec_nubmer = 200;
int octa_number = 0377;
float float_number = 88.0f;
float double_number = 88.0;
int negative_number = -1;
int positive_number = 1; // or +1

/* Đặt tên biến */
int x; // một biến
int x = 1; // biến được khai báo và khởi tạo
float x, y, z; // nhiều biến cùng kiểu dữ liệu
const int x = 88; // biến tĩnh, không ghi được
int tenBien1ok; // đặt tên biến này đúng
int ten_bien_nay_ok;

/* Đặt tên sai */
int 2001_tensai; // vì số ở đầu
int ten-sai; // dấu '-' không phải là alphanumeric
int while; // sai, vì dùng từ khóa vòng lặp while
```

ARDUINO - CHEATSHEET

```

/* Basic Program Structure */

void setup() {
    /* Runs once when sketch starts */
}

void loop() {
    /* Runs repeatedly */
}

/* condition */
if (x < 5) { ... }
else if (x > 10) { ... } //option
else { ... }           //option

/* loop */
while (x < 5) { ... }

do { ... } while(x < 0);

for (int i = 0; i < 10; i++) { ... }

break;    // Exit a loop immediately
continue; // Go to next iteration

switch (var) {
case 1:
    ...
break;
case 2:
    ...
break;
default:
    ...
}

/* Function Definitions */
/* <ret. type> <name>(<params>) { ... } */
int func_name(int x) { return x*2; }

return x; // x must match return type
return;   // For void return type

/* Include */
#include <stdio.h>          //standard library
#include "your-library.h" //Custom library

/* Strings */
/* Includes \0 null termination */
char str1[8] = {'A','r','d','u','i','n','o','\0'};

/* Compiler adds null termination */
char str2[8] = {'A','r','d','u','i','n','o'};
char str3[] = "Arduino";
char str4[8] = "Arduino";

/*Arrays*/
int myPins[] = {2, 4, 8, 3, 6}; // Array of 6 ints
int myInts[6];
myInts[0] = 42; // Assigning first
myInts[6] = 12; // ERROR! Indexes, are 0 though 5

/*Qualifiers*/
static      persists between calls
volatile    in RAM (nice for ISR)
const       read-only
 PROGMEM  in flash

```

```

/* General Operators */
= assignment
+ add
- subtract
* multiply
/ divide
% modulo
== equal to
!= not equal to
< less than
> greater than
<= less than or equal to
>= greater than or equal to
&& and
|| or
! not

/* Compound Operators */
++ increment
-- decrement
+= compound addition
-= compound subtraction
*= compound multiplication
/= compound division
&= compound bitwise and
|= compound bitwise or

/* Bitwise Operators */
& bitwise and | bitwise or
^ bitwise xor ~ bitwise not
<< shift left >> shift right
/* Pointer Access */
& reference: get a pointer
* dereference: follow a pointer

/* Numeric Constants */
123          decimal
0b0111011 binary
0173         octal - base 8
0x7B         hexadecimal base 16
123U         force unsigned
123L         force long
123UL        force unsigned long
123.0        force floating point
1.23e6       1.23*10^6 = 1230000

/* Data types */
boolean      true | false
char         -128      - 127, 'a' '$' etc.
unsigned char 0          - 255
byte          0          - 255
int          -32768     - 32767
unsigned int 0          - 65535
word          0          - 65535
long          -2147483648 - 2147483647
unsigned long 0          - 4294967295
float         -3.4028e+38 - 3.4028e+38
double        -3.4028e+38 - 3.4028e+38
void          i.e., no return value

/* Variable declare */
int          a;
int          a = 0b0111011, b = 0123, c = 1, d;
float        fa = 1.0f;
double       da = 1.0;
char          *pointer;
char          *other_pointer = NULL;

```

```

/*
 * BUILT-IN FUNCTIONS
 * Pin Input/Output
 * Digital I/O - pins 0-13 A0-A5
 */
pinMode(pin,
[INPUT, OUTPUT, INPUT_PULLUP])
int digitalRead(pin)
digitalWrite(pin, [HIGH, LOW])

/*Analog In - pins A0-A5*/
int analogRead(pin)
analogReference(
[DEFAULT, INTERNAL, EXTERNAL])

/*PWM Out - pins 3 5 6 9 10 11 */
analogWrite(pin, value)

/*Advanced I/O*/
tone(pin, freq_Hz)
tone(pin, freq_Hz, duration_ms)
noTone(pin)
shiftOut(dataPin, clockPin,
[MSBFIRST, LSBFIRST], value)
unsigned long pulseIn(pin,
[HIGH, LOW])

/*Time*/
unsigned long millis() // Overflows at 50 days
unsigned long micros() // Overflows at 70 minutes
delay(ms)
delayMicroseconds(usec)

/*Math*/
min(x, y)    max(x, y)
abs(x)        sin(rad)
cos(rad)      tan(rad)
sqrt(x)       pow(base, exponent)
constrain(x, minval, maxval)
map(val, fromL, fromH, toL, toH)

/*Random Numbers*/
randomSeed(seed) // long or int
long random(max) // 0 to max-1
long random(min, max)

/*Bits and Bytes*/
lowByte(x)
highByte(x)
bitRead(x, bitn)
bitWrite(x, bitn, bit)
bitSet(x, bitn)
bitClear(x, bitn)
bit(bitn) // bitn: 0=LSB 7=MSB

/*Type Conversions*/
char(val)   byte(val)
int(val)    word(val)
long(val)   float(val)

/*External Interrupts*/
attachInterrupt(interrupt, func,
[LOW, CHANGE, RISING, FALLING])
detachInterrupt(interrupt)
interrupts()
noInterrupts()

/* Serial - comm. with PC or via RX/TX */
begin(long speed) // Up to 115200
end()
int available() // #bytes available
int read() // -1 if none available
int peek() // Read w/o removing
flush()
print(data)
println(data)
write(byte)
write(char * string)
write(byte * data, size)
SerialEvent() // Called if data ready

/* Servo.h - control servo motors */
attach(pin, [min_uS, max_uS])
write(angle) // 0 to 180
writeMicroseconds(uS) // 1000-2000; 1500 is midpoint
int read() // 0 to 180
bool attached()
detach()

/*Wire.h - I2C communication*/
begin() // Join a master

begin(addr) // Join a slave @ addr
requestFrom(address, count)
beginTransmission(addr) // Step 1
send(byte) // Step 2
send(char * string)
send(byte * data, size)
endTransmission() // Step 3
int available() // #bytes available
byte receive() // Get next byte
onReceive(handler)
onRequest(handler)

```

LIBRARIES/
TỔNG KẾT