

BUILD AMAZING INTERNET
OF THINGS PROJECTS

INTERNET OF THINGS

Cho người mới bắt đầu



IOT MAKER VIETNAM

Internet Of Things (IoT)

cho người mới bắt đầu

IoT Maker Việt Nam

Mục lục

Lời mở đầu	1
Đôi lời về tác giả	1
Thuật ngữ hay sử dụng	1
Giải thích code trong bài	2
Giới thiệu nội dung	3
Ai có thể sử dụng?	4
Mục tiêu mang lại cho người đọc	4
Chuẩn bị	4
Kiến thức cơ bản	5
Internet Of Things (IoT)	6
Internet of Things (IoT) là gì?	6
Hệ thống Internet of Things (IoT)	7
Những ứng dụng thực tế trong cuộc sống	10
ESP8266	11
Sơ đồ chân	11
Thông số phần cứng	12
SDK hỗ trợ chính thức từ hãng	12
ESP8285	13
Module và Board mạch phát triển	14
Board mạch phát triển ESP8266	14
Arduino là gì?	16
Một số đặc điểm của Arduino	16
Các lợi ích khi sử dụng Arduino	17
Cộng đồng Arduino trên thế giới	17
Arduino cho ESP8266 & board mạch ESP8266 WiFi Uno	17
Node.js	19
Lý do sử dụng Node.js trong cuốn sách này	19
Cuốn sách này có hướng dẫn Node.js ?	20
Sublime Text	21
Cài đặt và chuẩn bị	22
Arduino IDE	22
Cài đặt thư viện Arduino	23
USB CDC driver	24
Chọn Board ESP8266 WiFi Uno trong Arduino IDE	26
Nạp chương trình xuống board dùng Arduino IDE	27
Xuất firmware binary trong Arduino IDE	27
Serial Terminal	28
Node.js	29
Sublime Text	29
Git	29

Tổng kết.....	30
Hello World	31
Chớp tắt bóng LED	32
Kiến thức	32
Đấu nối.....	33
Mã nguồn chớp tắt dùng Delay.....	34
Mã nguồn chớp tắt dùng định thời.....	34
Digital IO.....	34
Tổng kết.....	35
Kiến thức	35
Mã nguồn dùng hồi vòng	36
Mã nguồn dùng ngắt.....	36
Các khái niệm.....	37
OLED	38
Màn hình OLED.....	38
Màn hình OLED SSD1306	38
Giao tiếp I2C	38
Hiển thị màn hình OLED với ESP8266.....	39
Tổng kết	42
ESP8266 WiFi	43
Chế độ WiFi Station	45
Kiến thức	45
Kết nối vào mạng WiFi nội bộ	45
Sử dụng WiFiMulti	46
HTTP Client	48
Giao thức HTTP	48
JSON	51
Ứng dụng xem giá Bitcoin	52
Chế độ WiFi Access Point.....	56
ESP8266 hoạt động ở chế độ Access Point	56
Khởi tạo mạng WiFi sử dụng ESP8266	57
Web Server.....	58
Web Server là gì?	58
HTML - Javascript - CSS	58
Ứng dụng điều khiển đèn LED thông qua Webserver.....	61
ESP8266 Web Server	61
Kết hợp WiFi AP và Web Server.....	62
Trao đổi dữ liệu giữa 2 ESP8266	63
Yêu cầu	63
Hướng dẫn thực hiện.....	63
Code	63
Tổng kết.....	66
Dự án đọc cảm biến DHT11 và gửi về Server	67
Thiết kế ứng dụng.....	68

Yêu cầu	69
Phân tích	69
Kiến trúc	69
Thực hiện	71
Server Nodejs.....	72
Code ESP8266	77
Chuẩn bị	77
Ứng dụng mở rộng.....	80
Dùng ESP8266 như 1 Web Server	80
Tổng kết	81
Các chế độ cấu hình WiFi.....	82
Smartconfig	83
Kiến trúc	83
Thực hiện SmartConfig với ESP8266	84
Code	85
WPS	87
WPS là gì?.....	87
Thực hiện WPS với ESP8266	88
Code	88
Wifi Manager.....	89
Hoạt động cơ bản WifiManager.....	89
Chuẩn bị	91
Code.....	91
Mở rộng	92
Tổng kết.....	93
MQTT	94
Publish, subscribe	94
QoS	94
Retain	95
LWT	95
MQTT Client.....	96
MQTT Lens.....	96
MQTT.js.....	99
ESP8266 MQTT Client	102
MQTT Broker	109
MOSCA	109
Một số MQTT Broker sử dụng cho sản phẩm thực tế	113
Tổng kết.....	114
Websocket	115
Ưu điểm	115
Nhược điểm	115
Sử dụng ESP8266 như Websocket Server	116
Yêu cầu	116
Chuẩn bị	116

Đoạn code Javascript để tạo kết nối Web Socket	116
Nhúng file HTML chứa đoạn code JS vào ESP8266	117
Chương trình hoàn chỉnh cho ESP8266	118
Kết quả	120
Video kết quả	121
Sử dụng ESP8266 như Websocket Client	122
Javascript Websocket Client trên trình duyệt	122
Node.js Websocket Server	123
ESP8266 Websocket Client	124
Tổng kết	127
Firmware update over the air (FOTA)	128
Cập nhật firmware từ xa	129
Bảo mật	129
An toàn	130
Yêu cầu cần bản	130
Update process - memory view	130
OTA sử dụng Arduino IDE	132
Bước 1: nạp firmware hỗ trợ OTA thông qua cổng Serial	132
Bước 2: Lựa chọn cổng nạp thông qua OTA	134
Bước 3: Sửa firmware mới và nạp lại thông qua WiFi	135
Sử dụng mật khẩu	136
Những sự cố thường gặp	137
Cập nhật Firmware dùng Web Browser	138
Thực hiện	138
Bảo mật	141
HTTP Server	142
ESP8266 ESPhttpUpdate	142
Node.js OTA Server	144
Cheatsheet	146
Arduino - ESP8266 Cheatsheet	147
C - Cheatsheet	151
Lời kết	154
Các thành viên tham gia đóng góp	154
Lời kết	154
Giấy phép sử dụng tài liệu	155

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.

Chúng 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.

Đôi lời về tác giả

Chủ biên của cuốn sách là ông [Phạm Minh Tuấn\(TuanPM\)](#), có nhiều năm kinh nghiệm làm việc trong mảng IoT và phát triển các thư viện mã nguồn mở cho cộng đồng. Tác giả xây dựng cuốn sách này với mục đích đóng góp 1 phần nhỏ những kiến thức của mình vào sự phát triển của ngành công nghiệp vẫn còn mới mẻ nhưng rất tiềm năng này.

Thuật ngữ hay sử dụng

- **IoT** - Internet Of Things hay internet vạn vật.
- **ESP8266** - Chip xử lí tích hợp thu phát WiFi.
- **Git** - Trình quản lý phiên bản.
- **Github** - Mạng xã hội dành cho lập trình viên.
- **IDE** - Viết tắt của Integrated Development Enviroment - môi trường phát triển tích hợp.
- **Compiler** - Trình biên dịch.
- **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 giải thích đây là comment (chú thích).

② Dòng này giải thích biến `a` tăng thêm 1 đơn vị.

IOTMAKER.VN

Giới thiệu nội dung

Nội dung quyển sách này bao gồm các hướng dẫn chi tiết cho người đọc lập trình ứng dụng IoT sử dụng Chip WiFi phổ biến hiện nay là ESP8266 để kết nối với Server, gửi, nhận dữ liệu và thực thi các lệnh từ Server. Internet Of Things dựa và các kết nối Internet khá nhiều, do vậy các nội dung cũng tập trung nhiều vào các giao thức (prototcol), các phương pháp quản lý cũng như cấu hình kết nối.

Phần cứng sử dụng chính là System On Chip (SoC) **ESP8266** - có khả năng kết nối WiFi và lập trình được với giá thành rẻ và 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 trên máy tính cho Chip ESP8266 là [Arduino](#), ngôn ngữ lập trình **C/C++**.

Các phần liên quan đến Server chạy trên máy tính sử dụng [NodeJS](#) với ngôn ngữ lập trình **Javascript**.

Ngoài ra, bạn sẽ cần tìm hiểu một số công cụ và khái niệm thường xuyên 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 - Dùng để định dạng mã nguồn phù hợp, dễ đọc, dễ sửa chữa.
- Editorconfig - Là một công cụ cộng thêm vào cho các Editor, giúp việc đồng bộ hóa các tiêu chuẩn như Indent, Align, Space ... để đảm bảo code khi được mở ở các Editor không bị thay đổi.



Tuy phần cứng chính thức sử dụng là board mạch phần cứng mở [IoT WiFi Uno](#), nhưng bạn hoàn toàn có thể sử dụng bất kỳ board ESP8266 nào khác trên thị trường cho cuốn sách này, ví dụ như: [NODEMCU](#), [Wemos](#), ...



Các nội dung trong quyển sách này tập trung vào hướng dẫn các giao thức, cách thức làm việc với có hệ thống với board ESP8266, Server ..., còn các dự án mẫu, hướng dẫn chi tiết có thể tìm thêm 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 sai hệ thống.

Chuẩn bị

- Ít nhất bạn cần 1 board mạch ESP8266 lập trình được, tốt nhất nên sử dụng các board mạch tương thích với Arduino IDE (đã có các module nạp cho chip).
- Nên có thêm các module khác như cảm biến, động cơ để thực hành, một bộ StarterKit là phù hợp.
- 1 máy tính cá nhân (Windows, MacOS hoặc Linux).
- C & Arduino ESP8266 Cheatsheet (Mục lục cuối quyển sách này).

Kiến thức cơ bản

Trong phần này, 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 sẽ 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 **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](https://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](#)

Hệ thống Internet of Things (IoT)

Hệ thống IoT cho phép người dùng tiến sâu hơn vào việc tự động hóa, phân tích, tích hợp. Giúp cho việc cải thiện tầm nhìn, tính chính xác, nâng tầm các công nghệ về cảm biến, kết nối, robot để đạt hiệu quả cao nhất.

Các hệ thống IoT phát triển, khai thác các tiến bộ của phần mềm, giảm giá thành khi xây dựng phần cứng và tận dụng các công nghệ hiện đại. Những cải tiến này làm thay đổi cách vận hành của quá trình sản xuất sản phẩm, dịch vụ, xã hội, kinh tế và ảnh hưởng đến cả chính trị

Những điểm mấu chốt của IoT

Những vấn đề quan trọng nhất của hệ thống IoT bao gồm trí thông minh nhân tạo, kết nối, cảm biến và các thiết bị nhỏ nhưng mang tính cơ động cao, chúng được mô tả sơ lược như bên dưới:

- **AI (Artificial Intelligence)** - Hệ thống IoT về cơ bản được hiểu là làm cho mọi thiết bị trở nên **thông minh**, nghĩa là nó giúp nâng cao mọi khía cạnh của cuộc sống bằng những dữ liệu thu thập được, thông qua các thuật toán tính toán nhân tạo và kết nối mạng. Một ví dụ đơn giản như hộp đựng gạo của bạn, khi biết rằng gạo sắp hết, hệ thống tự động đặt một đơn hàng mới cho nhà cung cấp.
- **Connectivity** - Là một đặc trưng cơ bản của IoT, hiện nay các mạng thiết bị đang trở nên phổ

biến, nhiều mạng thiết bị ngày càng nhỏ hơn, rẻ hơn và được phát triển phù hợp với thực tế cũng như nhu cầu của người dùng .

- **Sensors** - IoT sẽ mất đi sự quan trọng của mình nếu không có sensors. Các cảm biến hoạt động giống như một công cụ giúp IoT chuyển từ mạng lưới các thiết bị thụ động sang mạng lưới các thiết bị tích cực, đồng thời có thể tương tác với thế giới thực.
- **Active Engagement** Ngày nay, phần lớn các tương tác của những công nghệ kết nối xảy ra 1 cách thụ động. IoT được cho là sẽ đem đến những hệ thống mang tính tích cực về nội dung, sản phẩm cũng như các dịch vụ gắn kết.
- **Small Devices** - Như đã được dự đoán từ trước, các thiết bị ngày càng được tối ưu với mục đích nâng cao độ chính xác, khả năng mở rộng cũng như tính linh hoạt. Nó được thiết kế ngày càng nhỏ hơn, rẻ hơn và mạnh mẽ hơn theo thời gian.

IoT – Những lợi ích mang lại

Những lợi ích mà IoT đem lại được dàn trải hầu hết đến các tất cả các lĩnh vực trong đời sống, kinh doanh... Dưới đây liệt kê ngắn gọn một số tính năng hữu ích của IoT:

- **Cải thiện việc gắn kết khách hàng** - Hệ thống IoT giúp phân tích các điểm mù hiện tại, tìm ra những sai sót về độ chính xác. IoT thay đổi điều này để mang lại nhiều sự gắn kết hơn và hiệu quả hơn với người dùng. Một ứng dụng tại các cửa hàng, dịch vụ iBeacon giúp tăng số lượng sản phẩm tới người tiêu dùng bằng cách chỉ dẫn người dùng tới khu vực cụ thể trong cửa hàng và đưa ra các gợi ý về sản phẩm. Chúng cung cấp các thông tin chi tiết, các đánh giá về sản phẩm, ...Bên cạnh đó chúng cũng có khả năng cho phép người dùng chia sẻ các sản phẩm qua mạng xã hội ...
- **Tối ưu hóa công nghệ** - giúp nâng cao trải nghiệm của khách hàng cũng như cải thiện việc sử dụng thiết bị và hỗ trợ cải tiến công nghệ.
- **Giảm sự hao phí** - IoT giúp việc quản lý tài nguyên ở các lĩnh vực được cải thiện 1 cách rõ ràng. Các phân tích hiện tại thường cung cấp cho chúng ta cái nhìn ở khía cạnh bên ngoài, trong khi IoT cung cấp các dữ liệu, thông tin thực tế để quản lý tài nguyên một cách hiệu quả hơn.
- **Tăng cường việc thu thập dữ liệu** - Thông thường, việc thu thập dữ liệu bị hạn chế do thiết kế hệ thống mang tính thụ động. IoT phá vỡ sự ràng buộc, giới hạn của thiết kế và tạo ra 1 hình ảnh chính xác của tất cả mọi thứ.

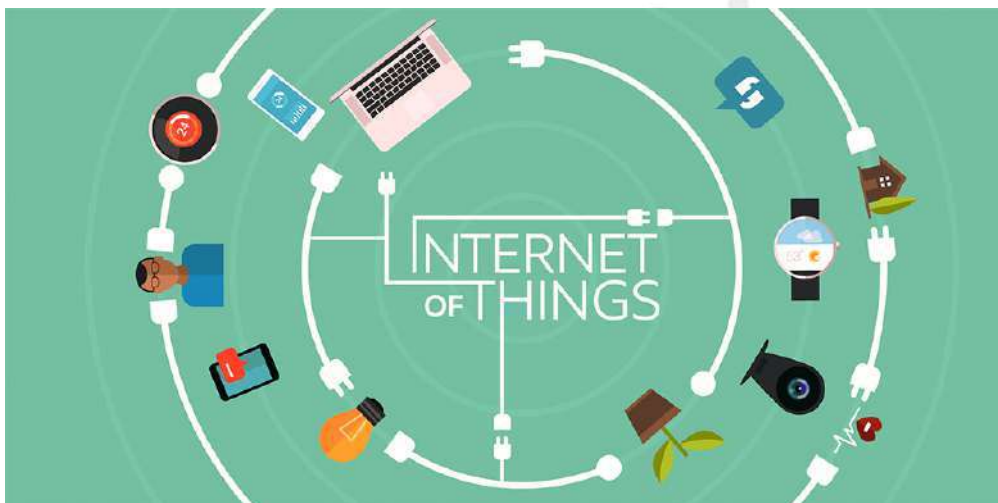
IoT – Những thách thức gặp phải

Mặc dù IoT mang lại khá nhiều lợi ích ấn tượng, nó cũng gặp phải những thách thức đáng kể. Dưới đây là 1 số vấn đề chính của IoT :

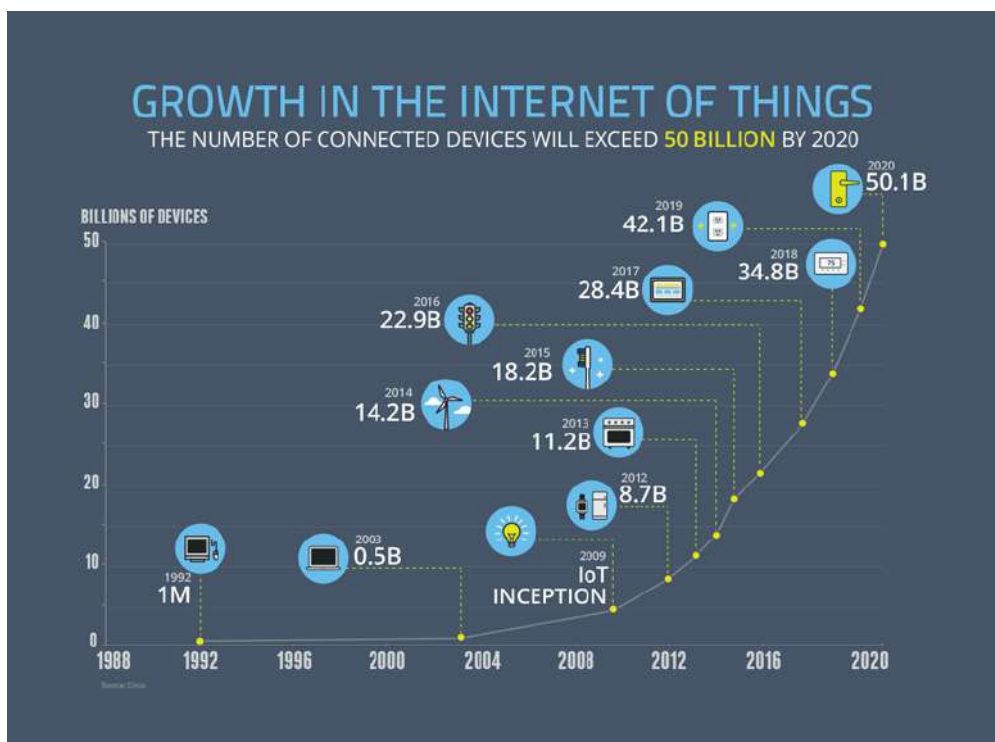
- **Kiểm soát an ninh** - IoT tạo ra 1 hệ sinh thái mà ở đó các thiết bị kết nối liên tục và giao tiếp với nhau qua mạng lưới các kết nối. Tuy nhiên, hệ thống thường chưa chú trọng đến các biện pháp

an ninh nhằm bảo mật thông tin, dẫn đến nó có thể gặp phải các cuộc tấn công nhằm lấy cắp thông tin của người dùng.

- **Tính bảo mật** - Do tính bảo mật chưa cao cộng với bản chất của IoT là không cần nhiều sự tương tác của con người nên các kẻ tấn công có thể cung cấp các thông tin người dùng giả mạo.
- **Tính phức tạp** - Một số hệ thống IoT có độ phức tạp về thiết kế và triển khai ứng dụng cũng như khó khăn trong việc bảo trì, nâng cấp hệ thống do sử dụng nhiều công nghệ còn khá mới mẻ.
- **Tính linh hoạt** - Có nhiều sự lo ngại khi đề cập đến tính linh hoạt của hệ thống IoT khi tích hợp với các hệ thống khác bởi các hệ thống khi kết hợp có thể xảy ra xung đột và các tính năng sẽ bị khóa lẫn nhau.
- **Tuân thủ các tiêu chuẩn** - Giống như các công nghệ khác trong lĩnh vực thương mại, IoT cũng phải tuân thủ các tiêu chuẩn, quy định đã đặt ra trước đó. Tính phức tạp của IoT làm cho việc tuân thủ các tiêu chuẩn là một thử thách thực sự



Hình 1. Hình minh họa



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

Những ứng dụng thực tế trong cuộc sống

Những ứng dụng của IoT vào các lĩnh vực trong đời sống là vô cùng phong phú và đa dạng. Chúng ta sẽ cùng điểm qua một số ứng dụng điển hình đã mang lại "tiếng tăm" cho IoT:

- **Smart Home** - Theo thống kê, smart home là ứng dụng liên quan đến IoT được tìm kiếm nhiều nhất trên Google. Smart Home là 1 ngôi nhà với rất nhiều tính năng tự động như bật máy điều hòa không khí khi bạn sắp về tới nhà, tắt đèn ngay khi bạn rời khỏi nhà, mở khóa khi người thân trong gia đình đang ở cửa nhà, mở garage khi bạn lái xe đi làm về ... còn rất nhiều những tính năng giúp nâng cao chất lượng cuộc sống khi sử dụng smart home.
- **Vật dụng mang theo trên người** - Có thể kể đến một số thiết bị như **Dashbon Mask**, đây là 1 chiếc smart headphone giúp bạn vừa có thể nghe nhạc với âm thanh có độ trung thực cao vừa có thể xem phim HD với máy chiếu ảo, hoặc **AMPL SmartBag** ba lô có pin dự phòng có thể sạc điện cho các thiết bị di động, kể cả máy tính.
- **Connected cars** - Giúp nâng cao những trải nghiệm cho người dùng xe ô tô, 1 chiếc **Connected car** có thể tối ưu các hoạt động của nó như thông báo khi hết nhiên liệu, đưa ra các cảnh báo khi có vật tới gần hoặc mới đây nhất là xe điện tự lái của hãng Tesla...

ESP8266

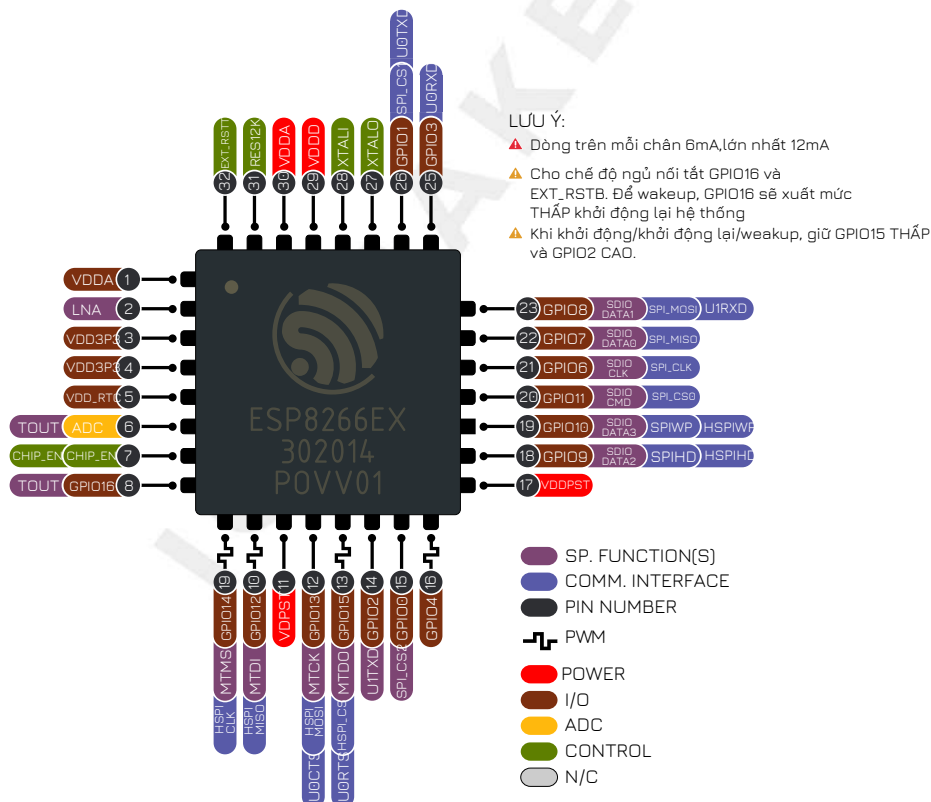
ESP8266 là dòng chip tích hợp Wi-Fi 2.4Ghz có thể lập trình được, rẻ tiền được sản xuất bởi một công ty bán dẫn Trung Quốc: Espressif Systems.

Được phát hành đầu tiên vào tháng 8 năm 2014, đóng gói đưa ra thị trường dạng Mô đun 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 nguồn 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

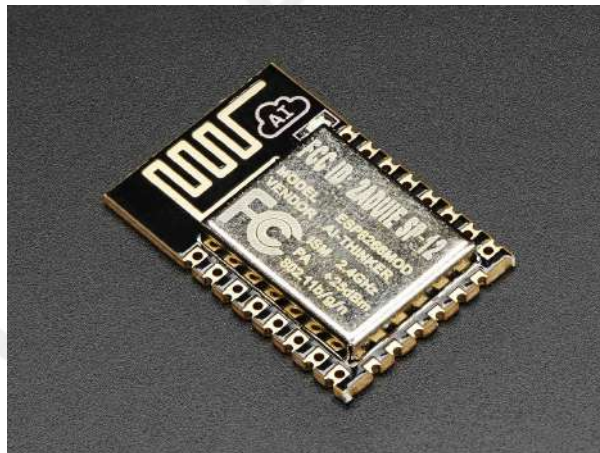
Sơ đồ chân



Hình 3. Sơ đồ chân ESP8266EX

Thông số phần cứng

- 32-bit RISC CPU : Tensilica Xtensa LX106 chạy ở xung nhịp 80 MHz
- Hỗ trợ Flash ngoài từ 512KiB đến 4MiB
- 64KBytes RAM thực thi lệnh
- 96KBytes RAM dữ liệu
- 64KBytes boot ROM
- Chuẩn wifi IEEE 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 các giao thức 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

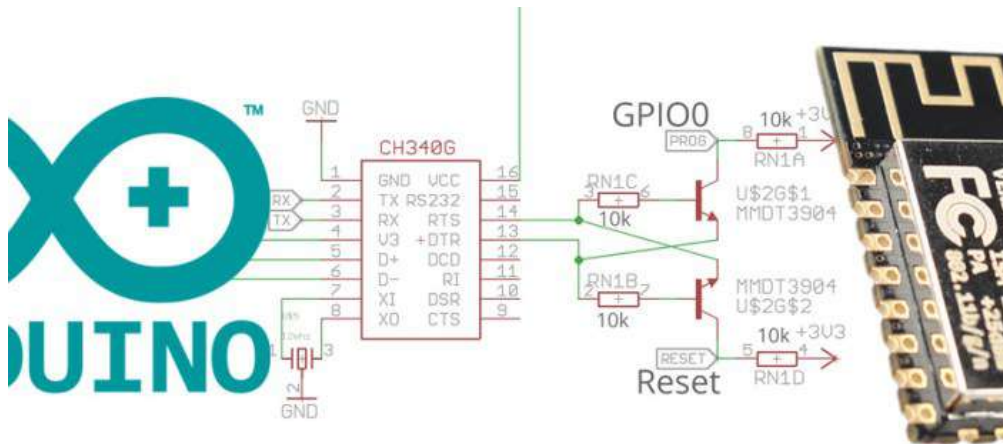
ESP8266 cần ít nhất thêm 7 linh kiện nữa mới có thể hoạt động, trong đó phần khó nhất là Antena. Đòi hỏi phải được sản xuất, kiểm tra với các thiết bị hiện đại. Do đó, trên thị trường xuất hiện nhiều Module và Board mạch phát triển đảm đương hết để người dùng đơn giản nhất trong việc phát triển ứng dụng. 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

Board mạch phát triển ESP8266

Module ESP8266 chỉ bao gồm Chip ESP8266 và các linh kiện giúp chip có thể hoạt động được, tuy nhiên, trong quá trình phát triển sản phẩm, chúng ta cần phải **nạp** chương trình cho chip trước khi đưa vào hoạt động thực tế. Quá trình này là quá trình gửi dữ liệu Binary (đã biên dịch trên máy tính) xuống bộ nhớ Flash của ESP8266. Để đưa ESP8266 vào chế độ **Nạp** (Program) thì cần phải đặt mức logic 0 (0V - GND) vào chân **GPIO0**, đồng thời RESET chip. Rồi sau đó có thể dùng các công cụ nạp để gửi Firmware từ máy tính xuống.



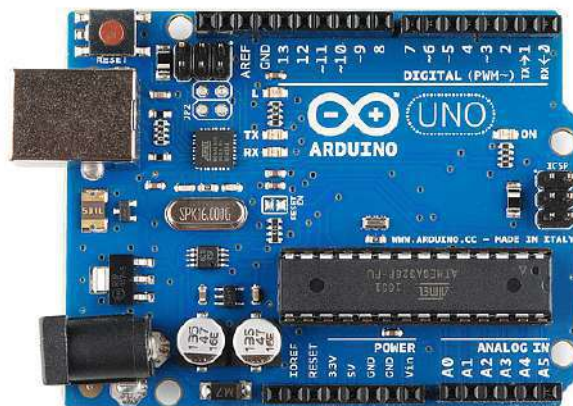
Hình 5. Một mạch nạp tự động sử dụng chip USB CDC

Hiện nay các Board mạch phát triển đều tích hợp các mạch nạp tự động, nghĩa là phần mềm sẽ tự động điều chỉnh các chân DTR và RTS của chip USB CDC, đưa ESP8266 vào chế độ nạp, sau đó sẽ gửi firmware xuống. Arduino IDE cũng vậy, nó sẽ điều chỉnh dựa trên việc khai báo Board mạch sử dụng.



Nếu bạn là người mới bắt đầu và chưa rõ về phần cứng, thì tốt nhất nên sử dụng một Board mạch phát triển sẵn có các chế độ nạp tự động. Khi bạn chuyển sang sản xuất phần cứng cho các ứng dụng cụ thể, thì có thể tách rời phần nạp tự động này ra để tiết giảm chi phí. Các mạch điện này đều được công bố rộng rãi.

Arduino là gì?



Hình 6. 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 **RISC-V** 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 các khái niệm đơn giản, che đi sự phức tạp của phần mềm bằng các 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
- Cách 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 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\(IDE & AVR/ARM/x86 Board\) www.arduino.cc](http://www.arduino.cc)
- [Arduino cho ESP8266 github.com/esp8266/Arduino](https://github.com/esp8266/Arduino)
- [Arduino cho ESP32 github.com/espressif/arduino-esp32](https://github.com/espressif/arduino-esp32)
- [Arduino cho PIC32 chipkit.net/](http://chipkit.net/)
- [Arduino cho STM32 www.stm32duino.com/](http://www.stm32duino.com/)
- [*Các dự án Arduino www.hackster.io/arduino](http://www.hackster.io/arduino)

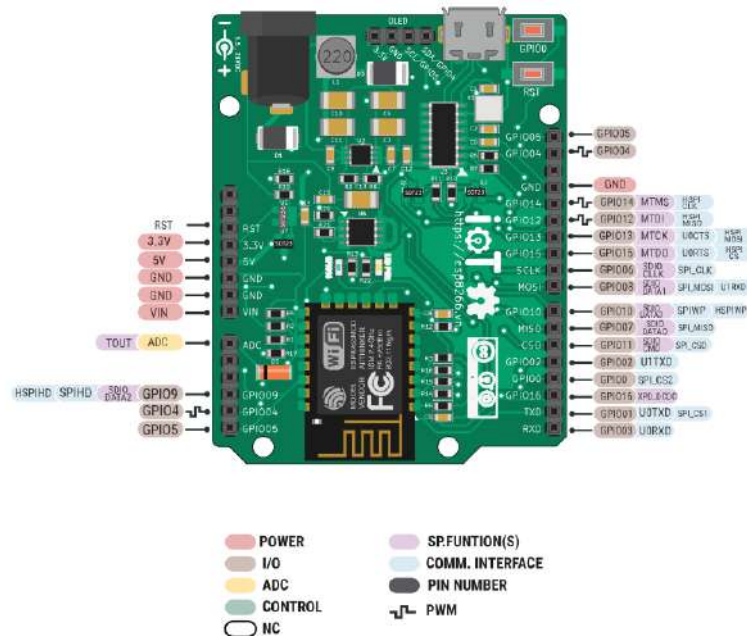
Arduino cho ESP8266 & board mạch ESP8266 WiFi Uno

[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 Sketch 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 7. PINOUT

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.

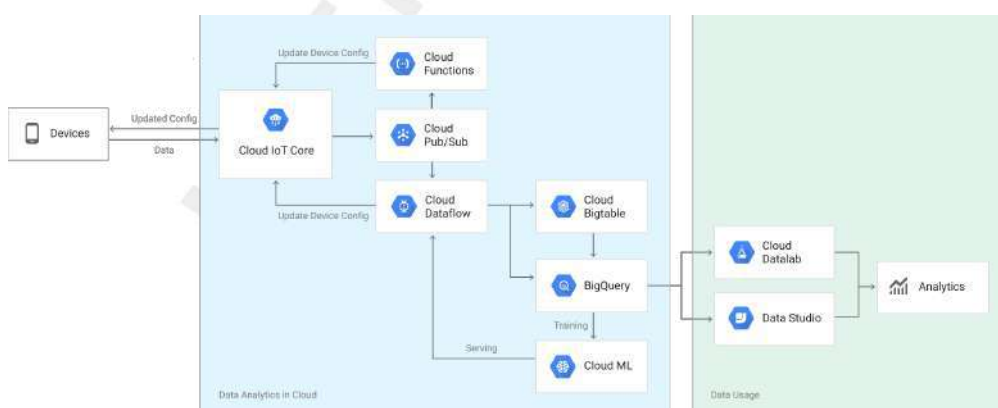
Nền tảng Cloud của gần như tất cả các nhà phát triển lớn hiện nay đều hỗ trợ thực thi Node.js, điển hình như Amazon Lambda, Google Script, IBM Blumix, Microsoft Azure ...

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 Node.js 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 khá 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 tích, báo cáo dữ liệu, trí tuệ nhân tạo. Mô hình ví dụ của Google IoT Core



Hình 8. 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 Node.js ?

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 với Node.js.

IOTMAKER.VN

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, bạn có thể lựa chọn các trình soạn thảo phổ biến hiện nay như **Atom**, **Visual Code**, nhưng đừng dại sử dụng Notepad, 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 là 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).



Hình 9. Sublime Text

Cài đặt và chuẩn bị

Arduino IDE

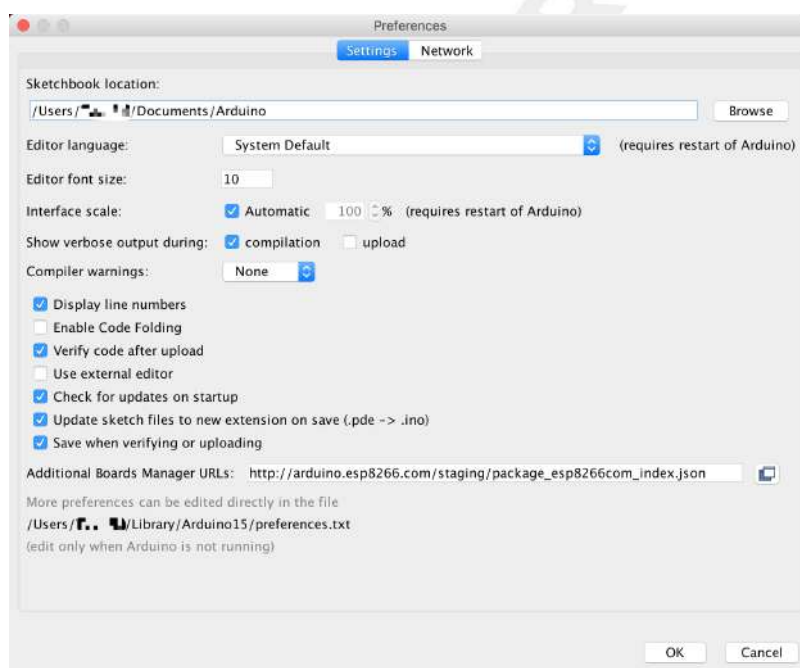
Bước 1: Cài đặt Arduino IDE.

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 Arduino 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 10. 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 like 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.

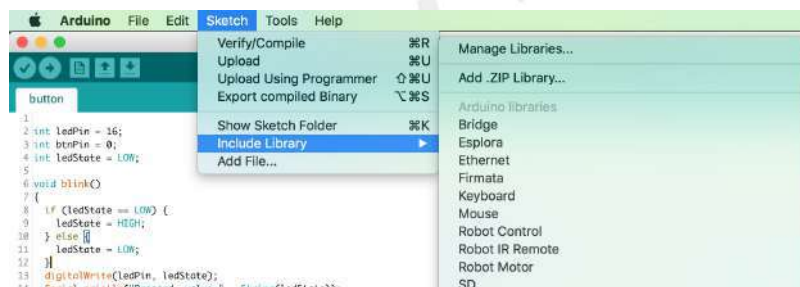


Hình 11. Cài đặt board ESP8266

Cài đặt thư viện Arduino

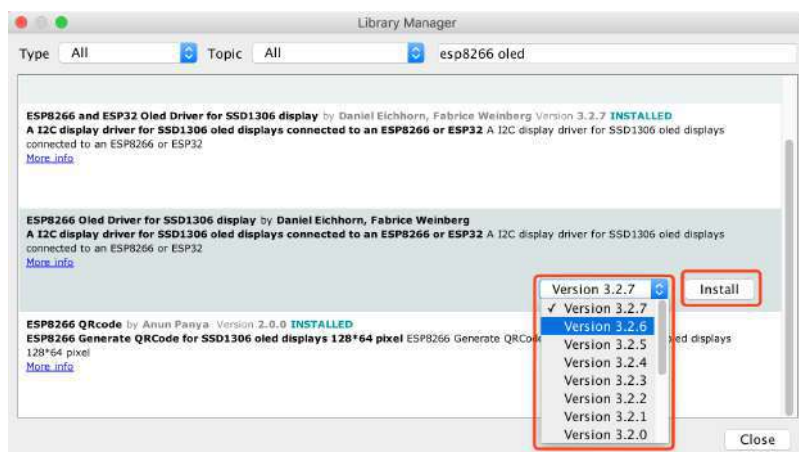
Một số thư viện do các nhà phát triển khác công bố và được tự do sử dụng có thể cài đặt trực tiếp bằng công cụ Library Manager của Arduino.

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



Hình 12. 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 13. 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.

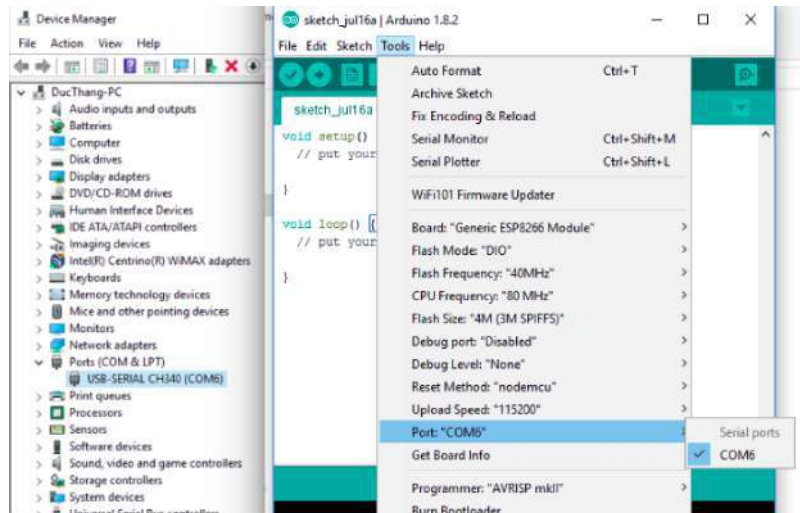
Thực hiện kết nối cable USB với board, đảm bảo đèn LED khoanh tròn sáng như ở hình dưới:



Hình 14. 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:



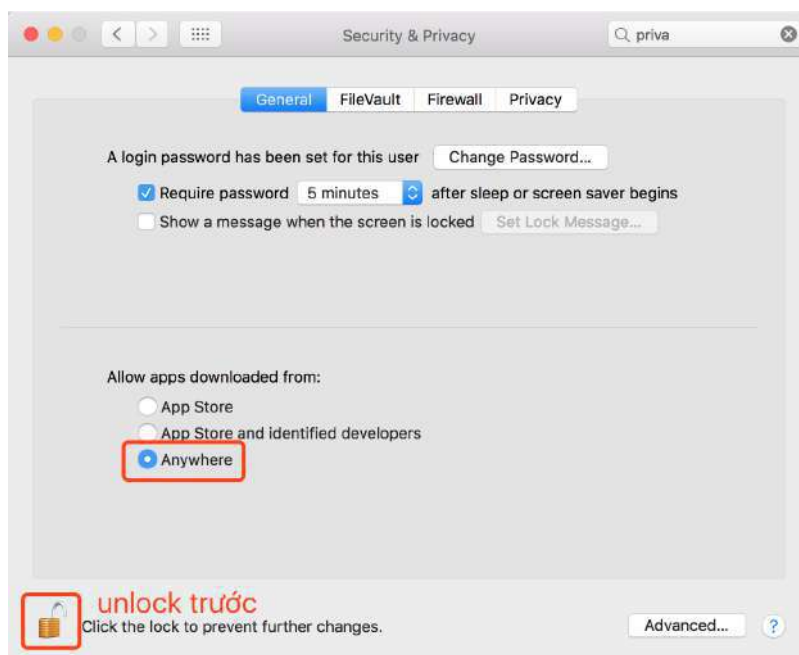
Hình 15. Kết nối thành công

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:

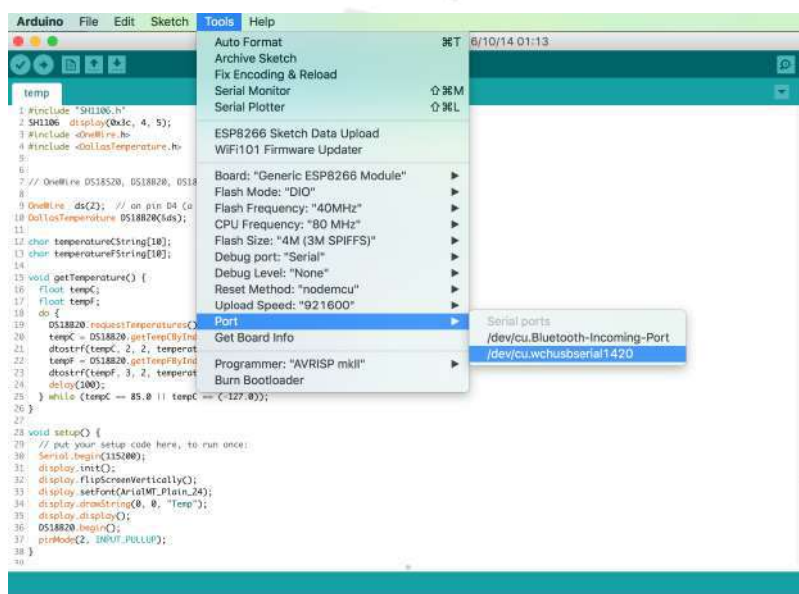
- Mở ứng dụng "Terminal" **cmd + space** -> Enter Terminal
- Xóa driver: **sudo rm -rf /System/Library/Extensions/usb.kext**
- Với một số máy, bạn phải thực thi thêm **sudo rm -rf /Library/Extensions/usbserial.kext**
- Nếu không thể thực hiện được lệnh trên, bạn cần phải thay đổi **Security and Privacy** 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 16. 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/ttyUSB0** trên Linux) Mở Arduino IDE và lựa chọn (tham khảo cấu hình kết nối như hình dưới):



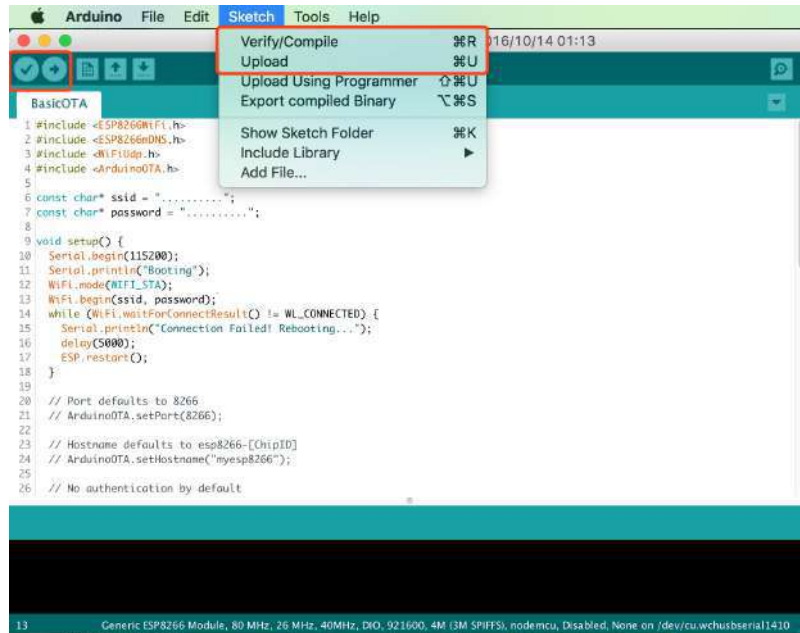
Hình 17. Cấu hình Board ESP8266 WiFi Uno

- 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.

Nạp chương trình xuống board dùng Arduino IDE

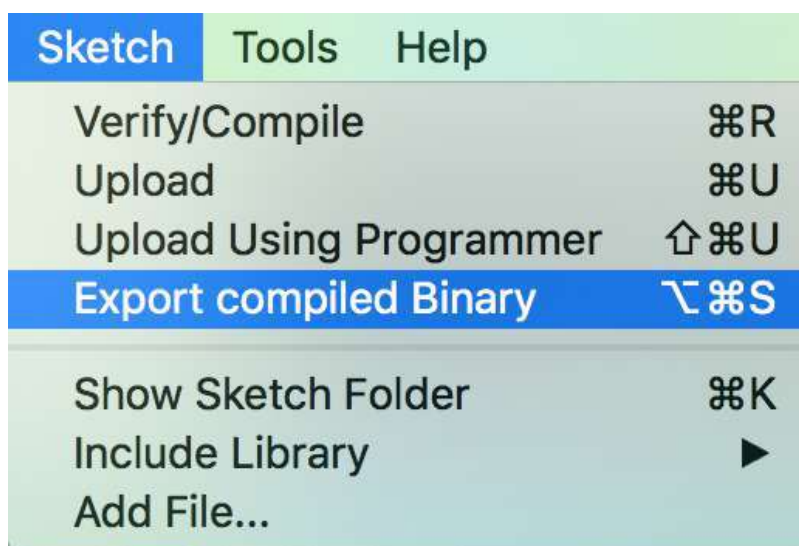
Trên giao diện Arduino có 2 nút, ngoài cùng bên trái là nút **Verify**, để biên dịch chương trình, tương đương với **Sketch > Verify/Compile**, nút tiếp theo là **Upload**, tương đương **Sketch > Upload**. Khi đã lựa chọn board phù hợp, chương trình không có lỗi, thì nhấn **Upload** sẽ nạp chương trình vào board và thực thi sau đó.



Hình 18. Nạp chương trình

Xuất firmware binary trong Arduino IDE

Với bất kỳ tình huống nào cần file Binary, bạn có thể được xuất ra bằng cách **Sketch > Export compiled Binary**, và file .bin sẽ nằm trong thư mục của Sketch.



Hình 19. Xuất file Binary

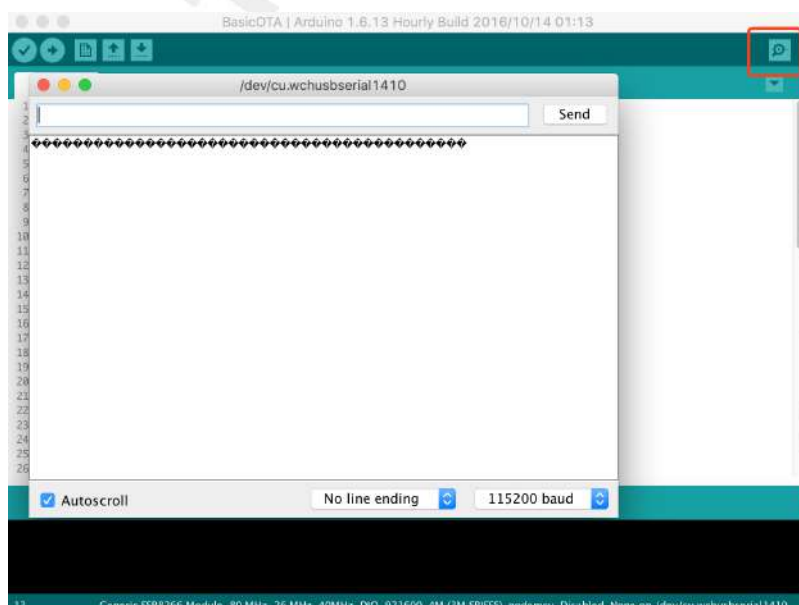
Serial Terminal

Có nhiều ứng dụng miễn phí để tương tác với cổng Serial trên máy tính:

- Windows: PuTTY, realterm
- Linux: minicom, screen
- MacOS: minicom, screen

Sử dụng Arduino IDE Serial Monitor

Arduino có tích hợp sẵn Serial Monitor, khi chọn đúng cổng Serial, thì có thể nhấn biểu tượng Serial trên IDE để mở:



Hình 20. Arduino IDE Serial Monitor

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/

IOTMAKER.VN

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. Các công cụ được lựa chọn đều là đa nền tảng, dễ dàng được sử dụng cho các hệ điều hành Mac OS, Windows, hay Linux

IOTMAKER.VN

Hello World

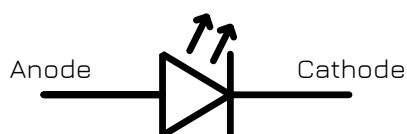
Bất kỳ một chương trình học nào cũng cần nên bắt đầu một cách từ từ. Bởi vì thời điểm này chúng ta đều mới bắt đầu, nhiều khái niệm, kiến thức về lĩnh vực này gần như không có nhiều. **Helloworld** giúp các bạn có thể nắm được các kiến thức cơ bản, làm sao để biên dịch, nạp được chương trình. Làm sao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

IOTMAKER.VN

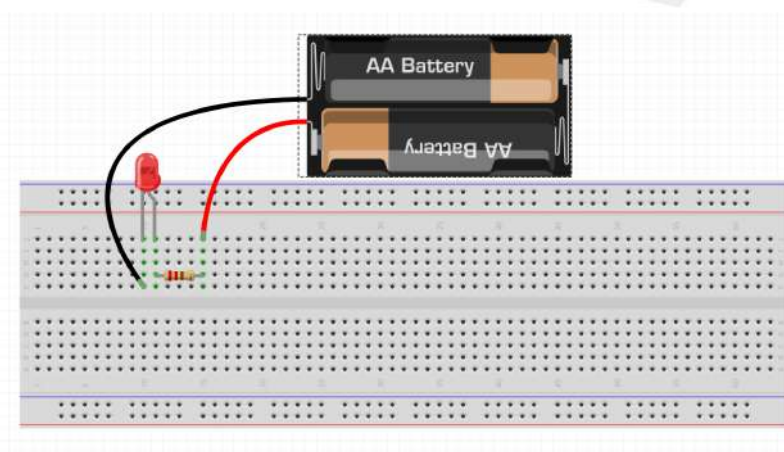
Chớp tắt bóng LED

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.

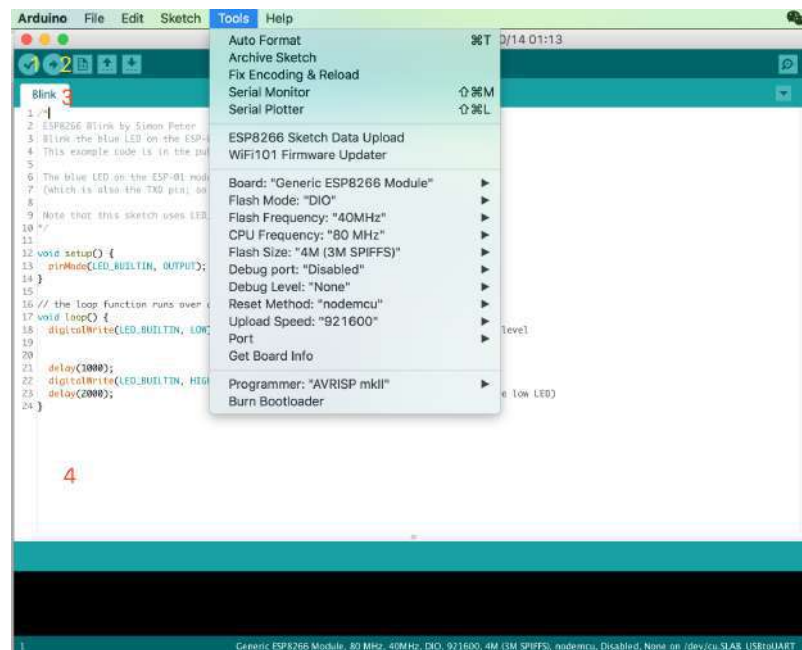


Hình 21. Ký hiệu LED trên mạch điện (Cathode+, Anode-)



Hình 22. Mạch có thể chạy được như sau

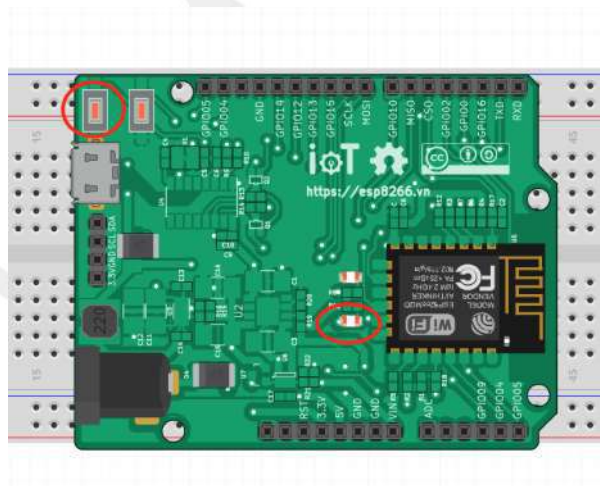
Arduino IDE



Hình 23. Arduino 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**.

Đấu nối



Hình 24. Mạch ESP8266 WiFi Uno có đấu nối sẵn LED vào Pin 16, và nút nhất vào Pin 0

Với mã nguồn bên dưới, sau khi kiểm tra chương trình, bạn cần chắc chắn đã **Chọn Board ESP8266 WiFi Uno** trong **Arduino IDE** và **Nạp chương trình xuống board dùng Arduino IDE**

Mã nguồn chớp 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 chớp 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);
    }
}
```

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)`.

Các chân GPIO0 đến GPIO15 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.

`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 và đều hỗ trợ các ngắt tiêu chuẩn của Arduino như: `CHANGE`, `RISING`, `FALLING`.

Tổng kết

Các ứng dụng mở rộng

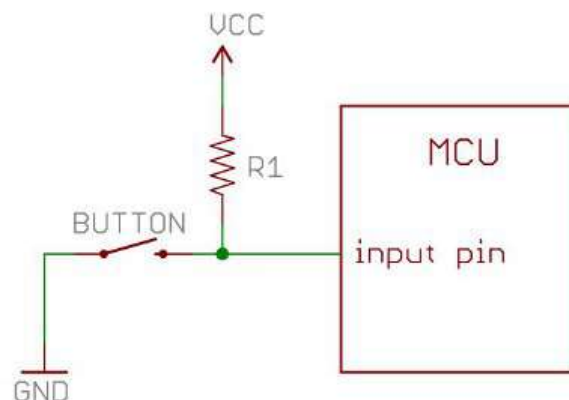
- Fading LED (sáng dần hay tắt dần)
- Chớp tắt LED dùng Ticker == Nút nhấn

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 phức tạp. 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 ngắt 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 đòi hỏi CPU liên tục kiểm tra xem mức Logic của nút nhấn. Đồng thời việc đáp ứng cũng không nhanh bằng sử dụng ngắt.





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

Thực hiện sau khi kiểm tra mã nguồn:

- [Chọn Board ESP8266 WiFi Uno trong Arduino IDE](#)
- [Nạp chương trình xuống board dùng Arduino IDE](#)

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
}
```

Các khái niệm

- **Ngắt** Ngắt là một khái niệm liên quan nhiều đến phần cứng, một sự kiện nào đó xảy ra, bắt buộc CPU phải dừng các tác vụ bình thường khác đang thực thi để thực hiện tác vụ Ngắt. Ví dụ, cấu hình ngắt khi có thay đổi mức logic từ 1 về 0 (cạnh xuống) của GPIO, thì khi mức logic thay đổi trên GPIO đó, CPU sẽ ngay lập tức dừng và lưu các trạng thái tại chương trình chính và nhảy vào hàm ngắt để thực thi các lệnh trong đó.
- **pull-up/pull-down** Đa số các chân GPIO của Chip đều có thể có 3 trạng thái, trạng thái là ngõ ra mức cao (logic 1), trạng thái là ngõ ra mức thấp (logic 0) và trạng thái ngõ vào (input). Ở trạng thái Input, thì các GPIO được cấu hình trở kháng cao (Hi-Z), hay còn gọi là trạng thái cách ly, không cho dòng điện đi qua, nhưng vẫn cảm nhận được điện áp. Ở trạng thái Hi-Z, nếu không xác định mức logic trước cho GPIO thì GPIO này bị thả trôi, nghĩa là rất dễ ảnh hưởng bởi môi trường, khi đọc về sẽ không đoán định được mức Logic. Pull-up là nối 1 điện trở với GPIO này lên mức logic 1, xác định trước 1 điện áp cho nó để đảm bảo không có tác động điện nào thì nó là mức logic 1. Tương tự, pull-down xác định trước mức logic 0 cho GPIO.

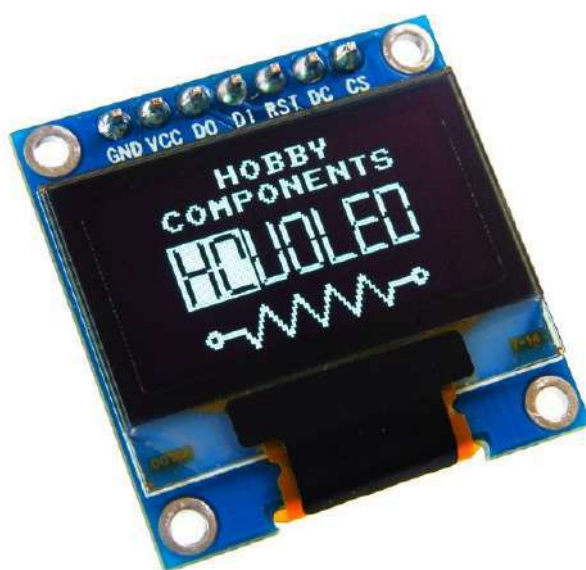
OLED

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 anốt và catốt sẽ tự động phát sáng mỗi khi có dòng điện chạy qua. OLED sử dụng đi-ốt 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

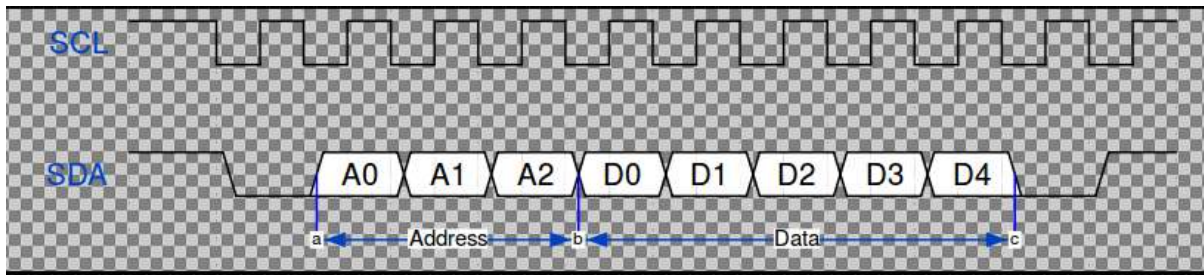


Hình 25. OLED SSD1306

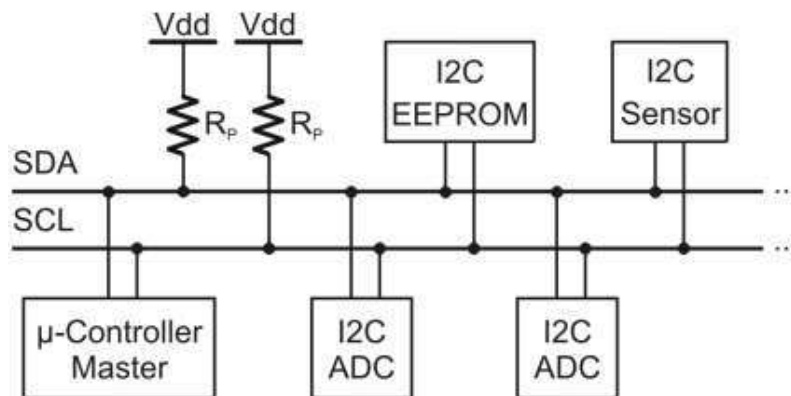
Giao tiếp I2C

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)



Hình 26. I2C Clock



Hình 27. Mô hình mạng I2C

Mạch vật lý I2C là mạch cực thu hở, 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: Đầu 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 đầu nối đã ra sẵn header, bạn chỉ cần cắm OLED vào như hình



Hình 28. ESP8266 với OLED

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

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();
}
```

Thực hiện sau khi kiểm tra mã nguồn:

- [Chọn Board ESP8266 WiFi Uno trong Arduino IDE](#)
- [Nạp chương trình xuống board dùng Arduino IDE](#)

Tổng kết

Thông qua các ví dụ mẫu về điều khiển đèn LED, nút nhấn và OLED, chúng ta có thể triển khai thêm nhiều ví dụ khác sử dụng GPIO của ESP8266, cũng như hiểu rõ hơn về cách làm việc của Arduino IDE, cách nạp chương trình, sử dụng thư viện ...

Một trong những ứng dụng có thể sử dụng các kiến thức này như là trò chơi Flappy Bird, bạn có thể thử nghiệm tại đây: <https://github.com/esp8266vn/arduino-flappybird>



Hình 29. Arduino FlappyBird

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.

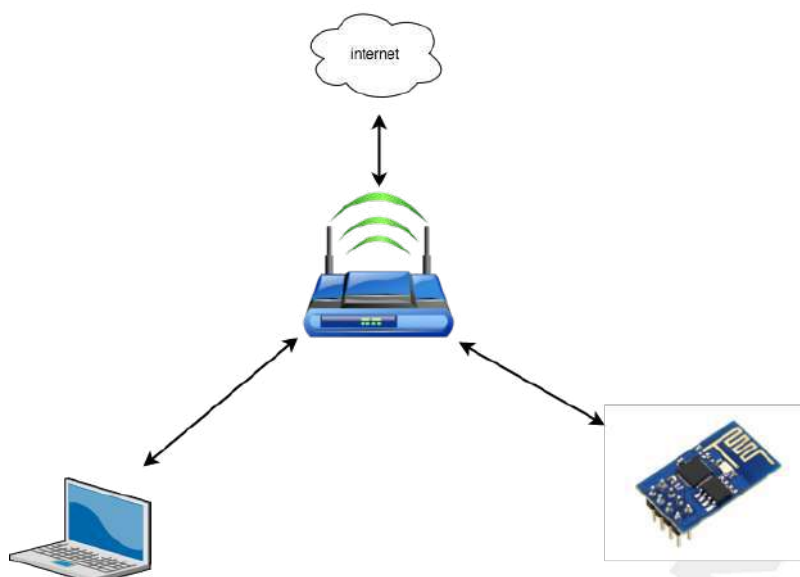
Trong đa phần các ứng dụng thì chế độ STA được sử dụng rất nhiều, nó giúp thiết bị kết nối đến mạng WiFi cục bộ, có internet để kết nối đến Server và gửi dữ liệu. Một số trường hợp khác thì chế độ AP được sử dụng để trao đổi dữ liệu với ESP8266 và máy tính (hoặc thiết bị có hỗ trợ trình duyệt). Ví dụ 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 30. 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 31. 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 gửi 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 nhằm tạo kết nối - ví dụ các phương thức 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() {

    // Thiết lập truyền dữ liệu nối tiếp ở tốc độ 115200 bits/s
    Serial.begin(115200);
    delay(10);
    Serial.print("Connecting to ");

    // In ra tên mạng wifi sẽ kết nối đến
    Serial.println(ssid);

    // Thiết lập ESP8266 ở chế độ station và kết nối đến mạng wifi đã chỉ định
    WiFi.begin(ssid, password);
    // Đoạn code in ra dấu . nếu ESP8266 chưa được kết nối
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    // In ra dòng "WiFi connected" và địa chỉ IP của ESP8266
    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 **nổi 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>
// Khai báo biến wifiMulti thuộc class ESP8266WiFiMulti để sử dụng các chức năng của class này.
ESP8266WiFiMulti wifiMulti;
// Biến connectioWasAlive nhằm kiểm tra kết nối của ESP8266 đến mạng wifi.
boolean connectioWasAlive = true;

void setup()
{
  Serial.begin(115200);
  Serial.println();
  // Add vào các mạng wifi mà ESP8266 được chỉ định sẽ kết nối
  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()
{
  // Kiểm tra nếu chưa kết nối đến 1 mạng wifi nào sẽ cài đặt connectioWasAlive = false
  // đồng thời in ra dấu "." sau mỗi 500ms nếu chưa được kết nối.
  if (wifiMulti.run() != WL_CONNECTED)
  {
    if (connectioWasAlive == true)
    {
      connectioWasAlive = false;
      Serial.print("Looking for WiFi ");
    }
    Serial.print(".");
    delay(500);
  }
  // Nếu đã kết nối đến 1 trong các mạng wifi sẽ in ra tên mạng wifi và set connectioWasAlive = true
  // để khi mất kết nối chương trình sẽ vào phần if (connectioWasAlive == true) nhằm thông báo đang
  // tìm kiếm mạng wifi
  else if (connectioWasAlive == false)
  {
    connectioWasAlive = true;
    Serial.printf(" connected to %s\n", WiFi.SSID().c_str());
  }
}

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

Thực hiện sau khi kiểm tra mã nguồn:

- Chọn Board ESP8266 WiFi Uno trong Arduino IDE
- Nạp chương trình xuống board dùng Arduino IDE

HTTP Client

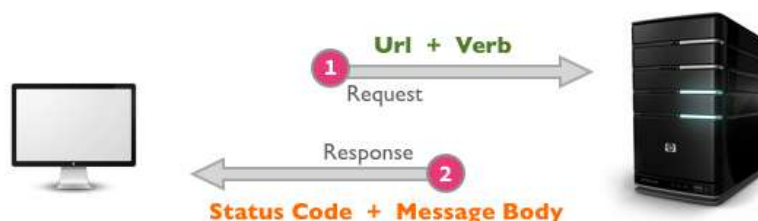
Giao thức HTTP

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.

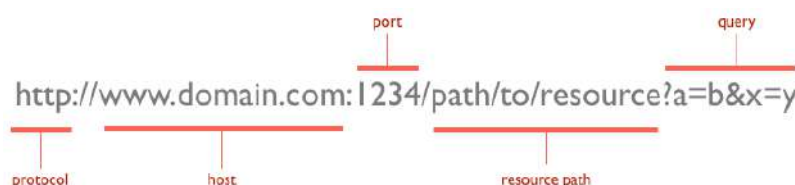
Trong giao thức HTTP, việc thiết lập kết nối chỉ có thể xuất phát từ phía client (lúc này có thể gọi là HTTP Client). Khi client gửi yêu cầu, cùng với **URL** và payload (dữ liệu muốn lấy) tới server. Server (HTTP Server) lắng nghe mọi yêu cầu từ phía client và trả lời các yêu cầu ấy. Khi trả lời xong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



Hình 32. 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 33. 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 trình duyệt sẽ thực hiện kết nối và gửi dữ liệu như sau.

```
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
```



Trên linux, bạn có thể sử dụng curl để xem chi tiết dữ liệu raw truyền nhận, gõ lệnh **curl -v http://esp8266.vn/log.php?nhiet_do=30** trên terminal

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 catch, 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 chứa 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. Các mã hay gặp:
 - **100 Continue**: Thông báo cho Client biết là có thể gửi tiếp phần request còn lại nếu còn, kết thúc nếu đã hết. Nếu trong request POST, phần thân request lớn sẽ bị server từ chối vì 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 để hoàn thành request. 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 không 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 có 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 bảo dưỡng hoặc quá tải.

JSON

JSON (JavaScript Object Notation) là 1 định dạng trao đổi dữ liệu để giúp việc đọc và viết dữ liệu trở nên dễ dàng hơn, máy tính cũng sẽ dễ phân tích và tạo ra JSON. 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 hoàn chỉnh, thuộc họ hàng với các ngôn ngữ trong họ hàng của 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 có thể là 1 object, record, struct, dictionary, hash table, keyed list hay associative array.
- Là 1 tập hợp các giá trị đã được sắp xếp. Trong hầu hết các ngôn ngữ, dữ liệu này được xem như array, véc tơ, list hay 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 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 www.cryptocompare.com/. 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 min-api.cryptocompare.com/data/price?fsym=BTC&tsyms=USD thì sẽ nhận được một chuỗi JSON dạng như:

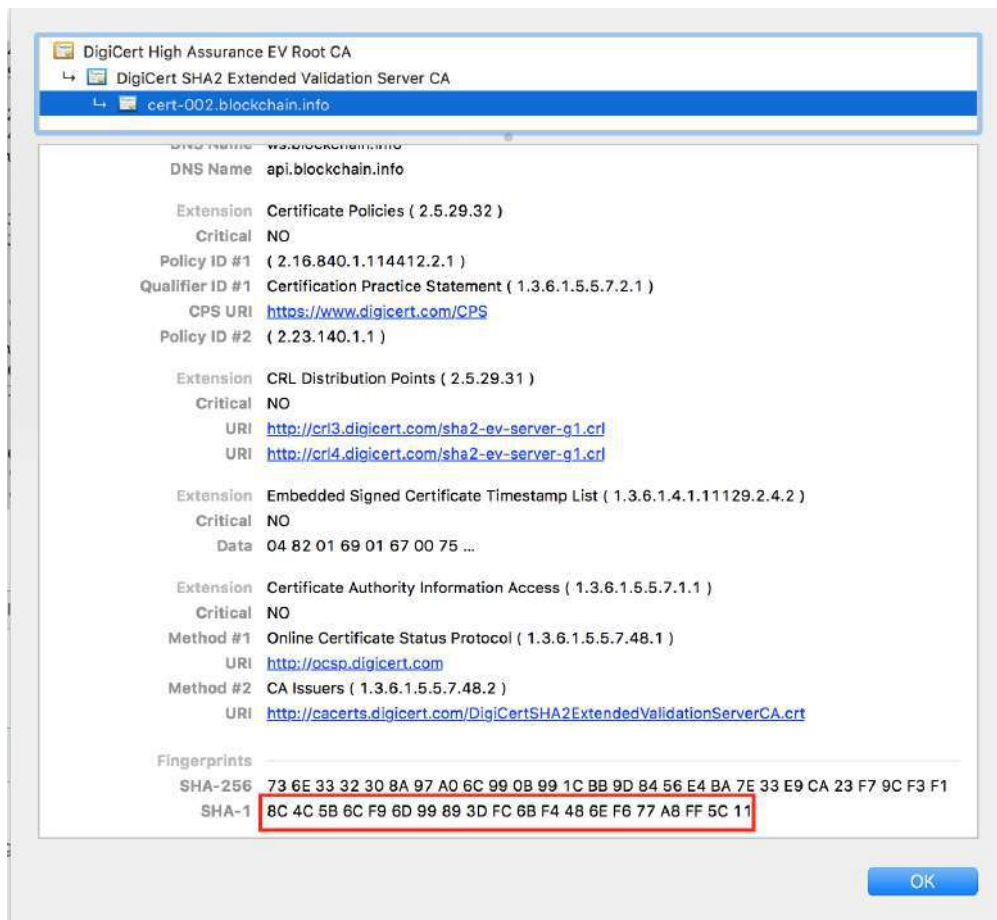
```
{"USD":4731.44}
```

và giá trị của trường **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**, về cơ bản nó cũng là HTTP, nhưng quá trình truyền nhận được mã hóa dữ liệu, thực hiện xác thực trước khi gửi giữa Client và Server.

Khi dùng HTTPS, chúng ta cần cung cấp SHA1 Fingerprint để Client có thể xác thực server. Bạn có thể dùng trình duyệt để truy cập trước để lấy. Nếu là **Chrome**, sau khi truy cập vào địa chỉ [min-api.cryptocompare.com/data/price?fsym=BTC&tsyms=USD](https://api.cryptocompare.com/data/price?fsym=BTC&tsyms=USD), Nhấn Ctrl+Shift+I (Shift + ⌘ + I với MacOS) và đi đến **Security > View Certificate > Details > Thumbprint**. Bạn sẽ thấy hình như bên dưới, và copy nó.



Hình 34. HTTPS Fingerprint

Đây là đoạn Code lấy giá Bitcoin, cứ mỗi 5 giây, ESP8266 sẽ kết nối đến Cryptocompare server để lấy thông tin và hiển thị lên OLED.

```

#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <ArduinoJson.h> //https://github.com/bblanchon/ArduinoJson
#include "SSD1306.h" //https://github.com/squix78/esp8266-oled-ssd1306

const char* ssid = ".....";
const char* password = ".....";
SSD1306 display(0x3c, 4, 5);
/* xem thêm https://www.cryptocompare.com/api/ */
const char* host = "https://min-api.cryptocompare.com/data/price?fsym=BTC&tsyms=USD";
/*SHA1 fingerprint*/
const char* fingerprint = "61 DE E9 FF BB 6B AD AA E4 9A 38 95 DC EC 74 2C 61 4B 7D 07";

void getBitcoin()
{
    HTTPClient http;
    Serial.print("connecting to ");
    Serial.println(host);

    http.begin(host, fingerprint);
    int httpCode = http.GET();
    if (httpCode == HTTP_CODE_OK) {
        String payload = http.getString();
        Serial.println(payload);
        StaticJsonBuffer<512> jsonBuffer;
        JsonObject root = jsonBuffer.parseObject(payload);
        if (!root.success()) {
            Serial.println("parseObject() failed");
            return;
        }

        double priceUSD = root["USD"];
        display.clear();
        display.drawString(0, 0, "Bitcoin price");
        display.drawString(0, 18, String(priceUSD));
        display.display();
        Serial.println(priceUSD);
    }
    http.end();
}

void setup() {
    Serial.begin(115200);
    display.init();
    display.clear();
    display.setFont(ArialMT_Plain_16);
    display.drawString(0, 0, "Connecting to");
    display.drawString(0, 18, ssid);
    display.display();
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    display.clear();
    display.drawString(0, 0, "Connected");
    display.display();
}

void loop() {
    if (WiFi.status() == WL_CONNECTED) {
        getBitcoin();
    }
    delay(5000);
}

```

Thực hiện sau khi kiểm tra mã nguồn:

- Chọn Board ESP8266 WiFi Uno trong Arduino IDE
- Nạp chương trình xuống board dùng Arduino IDE



Hình 35. Kết quả lấy giá Bitcoin và hiển thị mỗi 5 giây

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.

Khởi tạo mạng

Đầu tiên bạn cần include `<ESP8266WiFi.h>`, thư viện này chứa hàm `softAP` dùng để cấu hình Access Point mềm (soft AP) để khởi tạo một mạng WiFi.

Một mạng WiFi đơn giản nhất chỉ cần cung cấp tên SSID và không mật khẩu `WiFi.softAP(ssid)`. Phức tạp hơn, bạn cung cấp mật khẩu cho mạng WiFi `WiFi.softAP(ssid, password)`, hoặc chi tiết `WiFi.softAP(ssid, password, channel, hidden)` khi cung cấp chính xác kênh truyền (1..13), mặc định 1 và ẩn nó đi, không hiển thị ra khi `hidden = true`

Nhớ rằng `ssid` sử dụng chuỗi ký tự không quá 63, và mật khẩu (có thể không cần) với tối thiểu 8 ký tự cho mạng WPA2-PSK

Hàm `softAP` sẽ trả về `true` nếu khởi tạo thành công mạng WiFi



Lưu ý rằng, mạng WiFi khởi tạo bởi hàm `softAP` sẽ sử dụng địa chỉ IP mặc định là `192.168.4.1`, và chạy 1 DHCP Server cung cấp dải IP cho client kết nối tới là `192.168.1.x`. Bạn có thể thay đổi địa chỉ IP mặc định này bằng hàm `softAPConfig`. Ngoài ra, ESP8266 có thể chạy được song song 2 chế độ Station và Access Point, nhưng lưu ý, chỉ được 1 channel, và channel của softAP sử dụng bởi channel của Station.

`WiFi.softAPConfig(local_ip, gateway, subnet)` dùng để cấu hình IP cho Access Point

Cấu hình địa chỉ IP cho ESP8266 AP là `192.168.4.22`

```
IPAddress local_IP(192,168,4,22);
IPAddress gateway(192,168,4,9);
IPAddress subnet(255,255,255,0);
WiFi.softAPConfig(local_IP, gateway, subnet)
```

`WiFi.softAPgetStationNum()` sẽ trả về số lượng client đang kết nối tới Access Point

Khởi tạo 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() {
  Serial.printf("Stations connected = %d\n", WiFi.softAPgetStationNum());
  delay(3000);
}
```

Thực hiện sau khi kiểm tra mã nguồn:

- [Chọn Board ESP8266 WiFi Uno trong Arduino IDE](#)
- [Nạp chương trình xuống board dùng Arduino IDE](#)



Hình 36. Khi khởi động 1 WiFiAccesspoint

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 đầy đủ 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 nên được viết ở đây
    // ngay trước </body>
  </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.

Một đoạn CSS bao gồm 4 phần như thế này:

```

vùng-chọn {
  thuộc-tính-a: giá-trị-x;
  thuộc-tính-b: giá-trị-y;
  .....
}

```

Ví dụ về sử dụng CSS trong HTML

```

<!DOCTYPE html>
<html>
<head>
  <title>My first JavaScript page</title>
  <style>
    .class-select {
      background-color: red;
    }

    #id-select {
      background-color: green;
    }

    div {
      color: blue;
    }
  </style>
</head>
<body>
  <div id="id-select"> css select by id </div>
  <div class="class-select"> css select by class </div>
</body>
</html>

```

Với đoạn HTML bên trên, thì màu nền của tag div có id = **id-select** sẽ có màu xanh lá cây, tag div có class = **class-select** sẽ có màu đỏ, và chữ của tất cả những thẻ div có màu xanh nước biển.

Ứng dụng điều khiển đèn LED thông qua Webserver

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.

ESP8266 Web Server

Với ứng dụng này, ESP8266 sẽ khởi tạo 1 Web Server, khi có bất kỳ client nào kết nối tới (Web Browser) thì ESP8266 sẽ gửi về 1 trang HTML với các thông tin để Client có thể điều khiển chập tắt đèn LED của board.

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

const char* ssid = ".....";
const char* password = ".....";
const int led = 16; //LED pin = gpio16
/* HTML sẽ được gửi xuống client */
const char *html = \
"<html>\n\
  <head>\n\
    <title>ESP8266 Webserver</title>\n\
  </head>\n\
  <body>\n\
    <a href=\"/on\">ON</a>\n\
    <a href=\"/off\">OFF</a>\n\
  </body>\n\
</html>";

/* Web Server lắng nghe ở port 80 */
ESP8266WebServer server(80);

/* hàm này được gọi khi trình duyệt truy vấn đến '/on'
 * sẽ bật đèn LED (0 = on), sau đó chuyển hướng trình duyệt
 * về lại trang chủ '/'
 */
void handleOn() {
  digitalWrite(led, 0);
  server.sendHeader("Location", "/");
  server.send(301);
}

/* hàm này được gọi khi trình duyệt truy vấn đến '/off'
 * sẽ tắt đèn LED (1 = off), sau đó chuyển hướng trình duyệt
 * về lại trang chủ '/'
 */
void handleOff() {
  digitalWrite(led, 1);
  server.sendHeader("Location", "/");
  server.send(301);
}

/* hàm này được gọi khi trình duyệt truy vấn đến trang chủ '/'
 * sẽ gửi dữ liệu HTML, cung cấp các thông tin để bật, tắt LED
 */
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: ");

  /* Chúng ta có thể biết IP của ESP8266
   * để kết nối tới nhờ gọi hàm này
   */
  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();
}
```

Thực hiện sau khi kiểm tra mã nguồn:

- [Chọn Board ESP8266 WiFi Uno trong Arduino IDE](#)
- [Nạp chương trình xuống board dùng Arduino IDE](#)

Kết hợp WiFi AP và Web Server

Trao đổi dữ liệu giữa 2 ESP8266

Thông thường, muốn hai hay nhiều ESP8266 có thể liên lạc, trao đổi dữ liệu với nhau sẽ cần đến một router hay access-point, các module ESP8266 này sẽ kết nối vào Access Point rồi sau đó giao tiếp với nhau.

Có một số ứng dụng đơn giản để kết nối 2 ESP8266 với nhau mà không cần Access Point, chúng ta có thể khởi tạo 1 board hoạt động như là WiFi Access Point, đồng thời khởi tạo 1 TCP Server. Board khác hoạt động như 1 WiFi client thông thường, kết nối vào mạng WiFi đã được tạo, và khởi động 1 TCP Client kết nối vào TCP Server kia.

Yêu cầu

- Không cần bất kỳ một Router, hay Access Point nào, thực hiện việc kết nối giao tiếp giữa 2 ESP8266 thông qua mạng WiFi, 2 Board này sẽ truyền dữ liệu với nhau mỗi giây, và hiển thị lên Serial Terminal

Hướng dẫn thực hiện

Để trao đổi dữ liệu giữa 2 ESP8266 cần đáp ứng các điều kiện sau:

- 1 ESP8266 sẽ khởi động SoftAP để tạo mạng WiFi, đồng thời khởi động 1 TCP Server để làm Server.
- 1 ESP8266 phải là client với chế độ WiFi Station và kết nối đến ESP8266 server đã tạo ở trên.
- Sau mỗi giây, Board ESP8266 này sẽ gửi dữ liệu vào Board kia, board nhận được dữ liệu sẽ in ra cổng Serial và gửi ngược lại.

Code

P2P server

```
#include <ESP8266WiFi.h>

#define PORT 23
// Giới hạn số lượng clients kết nối
#define MAX_CLIENTS 3

// Tên và mật khẩu của ESP8266 AP sẽ tạo
const char *ssid = "yourSSID";
const char *password = "yourPassword";

//khởi tạo IP address
IPAddress local_IP(192, 168, 4, 1);
IPAddress gateway(192, 168, 4, 1);
IPAddress subnet(255, 255, 255, 0);
```



```

// Khoi tao port de clients ket noi.
WiFiServer server(PORT);
WiFiClient clients[MAX_CLIENTS];

void setup() {
  Serial.begin(115200);
  Serial.println();
  Serial.print("Setting soft-AP configuration ... ");

  //Cau hinh acces point, cai dat soft AP de client ket noi vao.
  WiFi.softAPConfig(local_IP, gateway, subnet);
  WiFi.softAP(ssid, password);

  //In ra local_IP cua AP.
  Serial.print("AP IP address: ");
  Serial.println(WiFi.softAPIP());
  Serial.println("Telnet server started");
  server.begin();
}

void loop() {
  uint8_t i;
  // kiem tra co client moi ket noi khong
  if (server.hasClient()) {
    for (i = 0; i < MAX_CLIENTS; i++) {
      if (!clients[i] || !clients[i].connected())
      { if (clients[i]) clients[i].stop();
        clients[i] = server.available();
        Serial.print("New client: "); Serial1.print(i);
        continue;
      }
    }
    WiFiClient serverClient = server.available();
    serverClient.stop();
  }
  // Kiem tra neu so client ket noi MAX_CLIENTS
  // co client, client duoc ket noi va o trang thai available
  // doc du lieu tu client, va gui lai du lieu cho client do.
  for (i = 0; i < MAX_CLIENTS; i++) {
    if (clients[i] && clients[i].connected()) {
      if (clients[i].available()) {
        String line = clients[i].readStringUntil('\r');
        Serial.print("Server receive from Client:");
        Serial.println(line);

        //Gui thong tin hoai dap cho client
        String resp = String(line.reserve(line.length() - 1));
        Serial.print(" Then, response back to client:");
        Serial.println(resp);
        clients[i].write(resp.c_str());
        Serial.println();
      }
    }
  }
}

```

P2P client

```

#include "ESP8266WiFi.h"
// Ten va mat khau cua ESP8266 AP lam server se vao
const char *ssid = "yourSSID";
const char *password = "yourPassword";

IPAddress server_ip(192, 168, 4, 1);
#define PORT 23
// port 23 la port cua esp8226 lam AP da khoi tao.
WiFiClient client;

void setup() {
  uint8_t i = 0;
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  Serial.print("\nConnecting to ");
  Serial.println(ssid);

  // Kiem tra tinh trang ket noi, neu chua ket noi duoc
  // se in chuoai "connecting..." tren man hinh serial terminal.
  while (WiFi.status() != WL_CONNECTED) {
    Serial.println("Connecting...");
    delay(500);
  }
}

unsigned long previousMillis = 0;
void loop() {
  if (WiFi.status() == WL_CONNECTED) {
    // Kiem tra neu client(STA) chua duoc ket noi.
    // Kiem tra tiep tục neu khong duoc ket noi den IP va PORT cua server(AP)
    // thi in ra serial terminal chuoai "connection failed".
    while (!client.connected()) {
      if (!client.connect(server_ip, PORT)) {
        Serial.println("connection failed");
        delay(1000);
        return;
      }
    }
    // Neu client(STA) duoc ket noi thi se doc du lieu tu server(AP)
    // den khi gap ki tu \r va in ra serial terminal du lieu nhan duoc.
    while (client.available()) {
      String line = client.readStringUntil('\r');
      Serial.print("Client receive from Server:");
      Serial.println(line);
    }
    //Send PING to server every 1000ms.
    unsigned long currentMillis = millis();
    if (currentMillis - previousMillis >= 1000) {
      previousMillis = currentMillis;
      client.write("PING\r");
    }
  }
}

```

Thực hiện sau khi kiểm tra mã nguồn:

- Chọn Board ESP8266 WiFi Uno trong Arduino IDE
- Nạp chương trình xuống board dùng Arduino IDE

Tổng kết

Trong phần này, chúng ta đã nắm rõ các chế độ hoạt động của ESP8266, các giao thức truyền TCP/IP, HTTP và các định dạng HTML, CSS, Javascript. Ngoài các ví dụ thực tiễn sử dụng ESP8266 như HTTP Client ở hướng dẫn trên, bạn có thể sử dụng HTTPClient để kết nối đến các Server tự tạo, gửi dữ liệu cảm biến đến Server, cũng như lấy dữ liệu từ Server để thực thi các tác vụ.

Chế độ WiFi Access Point và Web Server chạy trên ESP8266 thường sử dụng để cấu hình các thông số cho sản phẩm, sử dụng giao diện Web có ở bất kỳ máy tính nào để cung cấp các thông số phức tạp cho ứng dụng một cách dễ dàng.

IOTMAKER.VN

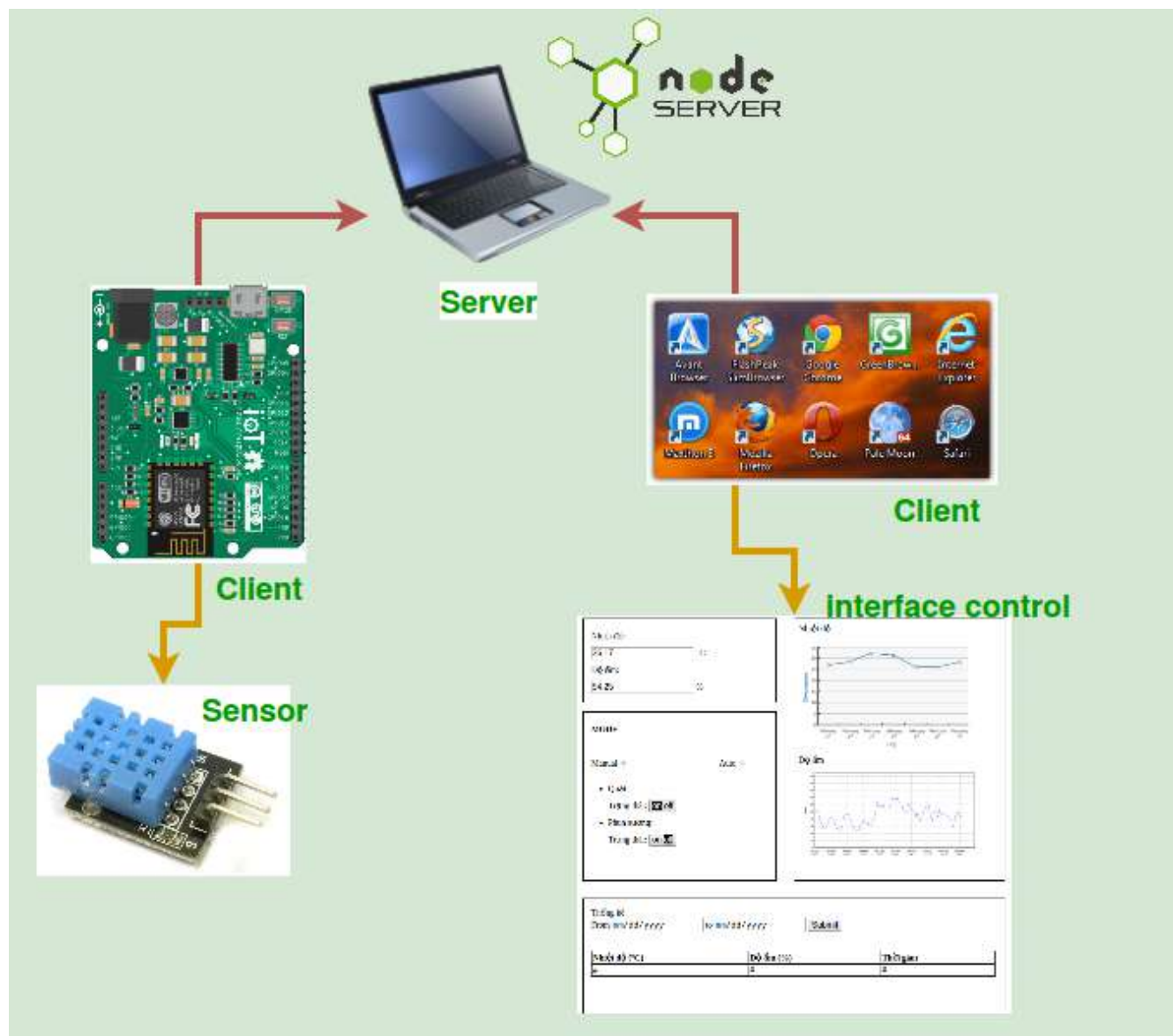
Dự án đọc cảm biến DHT11 và gửi về Server

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 37. 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

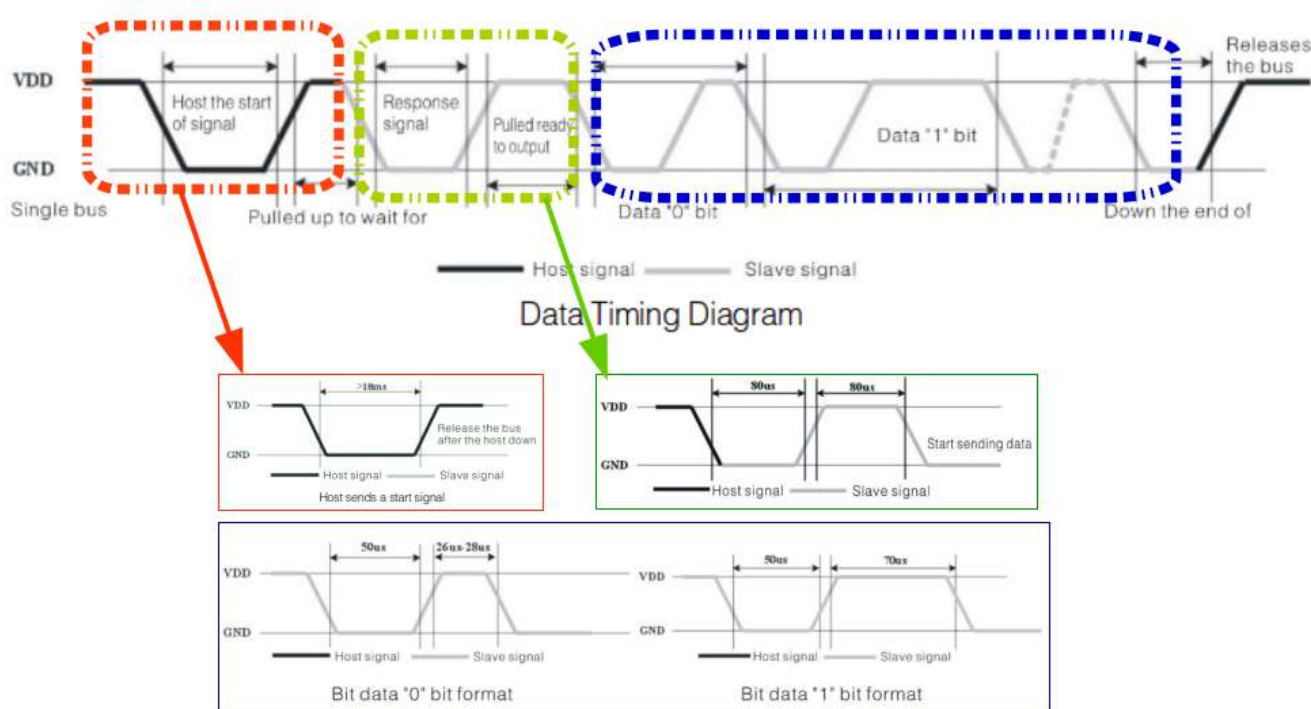
- Chuẩn truyền dữ liệu OneWire giữa các IC
- Ngôn ngữ Javascript để xây dựng server bằng cách dùng Node.js
- Ngôn ngữ HTML để xây dựng 1 trang html đơn giản nhằm hiển thị dữ liệu

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 38. 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.w3school.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](https://www.w3school.com/javascript), với Node.js thì [codeschool.com](https://www.codeschool.com) thật sự hữu ích với người mới bắt đầu.

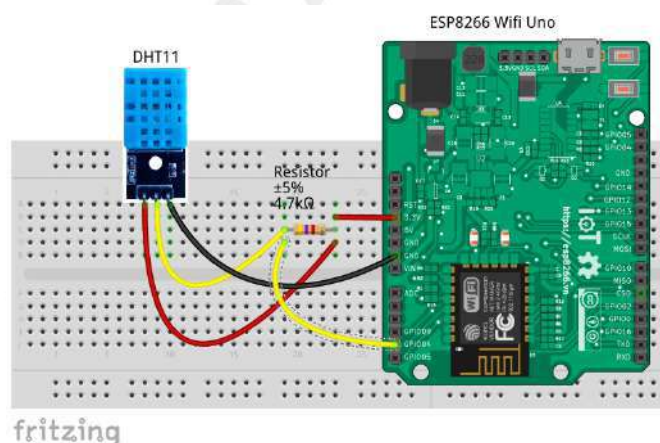
Thực hiện

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

Đấu nối

Kết nối sơ đồ mạch điện như hình bên dưới



Hình 39. 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
- **/** và còn lại thì trả về file **index.html**
- Mảng dữ liệu lưu trữ có định dạng: `[{"temp": 25, "humd": 80, "time": "time"}, ...]`

Mã nguồn file **server.js**

```
//-----
var fs = require('fs');
var url = require('url');
var http = require('http');①
var querystring = require('querystring');
var db = []; //database
//-----
// function gửi yêu cầu(response) từ phía server hoặc nhận yêu cầu (request) của client gửi lên
function requestHandler(request, response) {

    // Giả sử địa chỉ nhận được http://192.168.1.7:8000/update?temp=30&humd=40
    var uriData = url.parse(request.url);
    var pathname = uriData.pathname;           // /update?
    var query = uriData.query;                 // temp=30.5&hum=80
    var queryData = querystring.parse(query); // queryData.temp = 30.5, queryData.humd = 40
    //-----
    if (pathname == '/update') {
        var newData = {
            temp: queryData.temp,
            humd: queryData.humd,
            time: new Date()②
        };
        db.push(newData);
        console.log(newData);
        response.end();
    }
    //-----
    } else if (pathname == '/get') {③
        response.writeHead(200, {
            'Content-Type': 'application/json'
        });
        response.end(JSON.stringify(db));
        db = [];
    }
    //-----
    } else { ④
        fs.readFile('./index.html', function(error, content) {
            response.writeHead(200, {
                'Content-Type': 'text/html'
            });
            response.end(content);
        });
    }
    //-----
}

var server = http.createServer(requestHandler);
server.listen(8000); ⑤
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 thành phần để dễ dàng truy xuất
 - **http**: Phương thức truyền nhận dữ liệu dùng http
 - **querystring**: Module giúp chuyển string sang object
 - **db = []**: Biến kiểu mảng nhằm chứa dữ liệu nhiệt độ, độ ẩm
- 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>①
<html>
```

```

<head>
  <meta charset="UTF-8">
  <title>DHT11</title>
  <!-- Những file Javascript tại đường dẫn src để có thể xây dựng 1 graph -->
  <script type="text/javascript" src="https://canvasjs.com/assets/script/canvasjs.min.js"></script>②
</head>
<body>
  <h1> 1. THÔNG SỐ NHIỆT ĐỘ, ĐỘ ẨM</h><br> ③
    <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 THI</h1><br>
  <!-- thiết lập kích thước cho graph thông qua id ChartContainer đã thiết lập ở trên -->
  <div id="ChartContainer" style="height: 300px; width:80%;"></div>
  <script type="text/javascript">
    function httpGetAsync(theUrl, callback) { ④
      var xmlHttp = new XMLHttpRequest();
      xmlHttp.onreadystatechange = function() {
        if (xmlHttp.readyState == 4 & xmlHttp.status == 200)
          callback(JSON.parse(xmlHttp.responseText));
      }
      xmlHttp.open("GET", theUrl, true); // true for asynchronous
      xmlHttp.send(null);
    }

    window.onload = function() {
      var dataTemp = [];
      var dataHumd = [];

      var Chart = new CanvasJS.Chart("ChartContainer", {
        zoomEnabled: true, // Dùng thuộc tính có thể zoom vào graph
        title: {
          text: "Temperature & Humidity" // Viết tiêu đề cho graph
        },
        tooltip: { // Hiện 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, // Hiện thị "temp" ở mục chú thích (legend items)
          name: "temp",
          dataPoints: dataTemp // Dữ liệu hiển thị sẽ lấy từ dataTemp
        },
        {
          type: "line",
          xValueType: "dateTime",
          showInLegend: true,
          name: "humd",
          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 time = new Date(); // Lấy thời gian hiện tại

    var updateChart = function() {
      httpGetAsync('/get', function(data) {

        // Gán giá trị từ localhost:8000/get vào textbox để hiển thị
        document.getElementById("temp").value = data[0].temp;
        document.getElementById("humd").value = data[0].humd;
      });
    };
  </script>

```

```

        // Xuất ra màn hình console trên browser giá trị nhận được từ localhost:8000/get
        console.log(data);
        // Cập nhật thời gian và lấy giá trị nhiệt độ, độ ẩm từ server
        time.setTime(time.getTime() + updateInterval);
        yTempVal = parseInt(data[0].temp);
        yHumdVal = parseInt(data[0].humd);
        dataTemp.push({ // cập nhật dữ liệu mới từ server
            x: time.getTime(),
            y: yTempVal
        });
        dataHumd.push({
            x: time.getTime(),
            y: yHumdVal
        });
        Chart.render(); // chuyển đổi dữ liệu của của graph thành mô hình đồ họa
    });
};
updateChart(); // Chạy lần đầu tiên
setInterval(function() { // Cập nhật lại giá trị graph sau thời gian updateInterval
    updateChart()
}, updateInterval);
}
</script>
</body>
</html>

```

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 Sử dụng CanvasJS Chart để vẽ biểu đồ nhiệt độ, độ ẩm.

③ 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

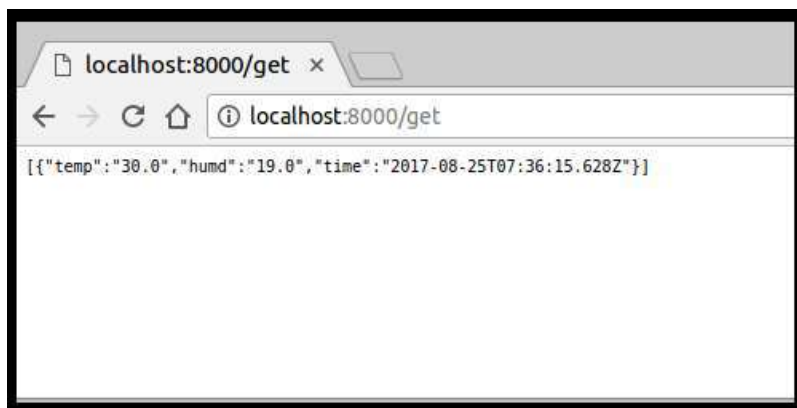
④ Hàm giúp lấy dữ liệu từ server

Phân tích phần vẽ biểu đồ

- Chúng ta sẽ lấy dữ liệu từ server gửi xuống và vẽ biểu đồ dùng mã Javascript, sử dụng tag `<script>code JS </script>` để chèn nội dung code Javascript 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 `xmlHttpRequest.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.



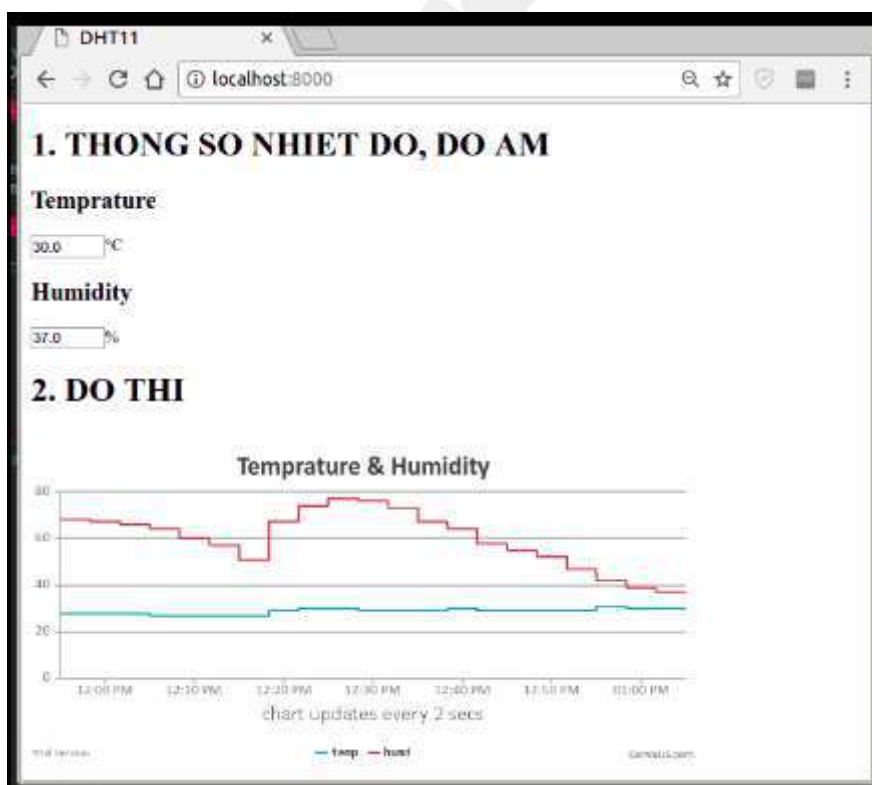
Hình 40. Hình ảnh dữ liệu tại địa chỉ localhost:8000/get

- 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



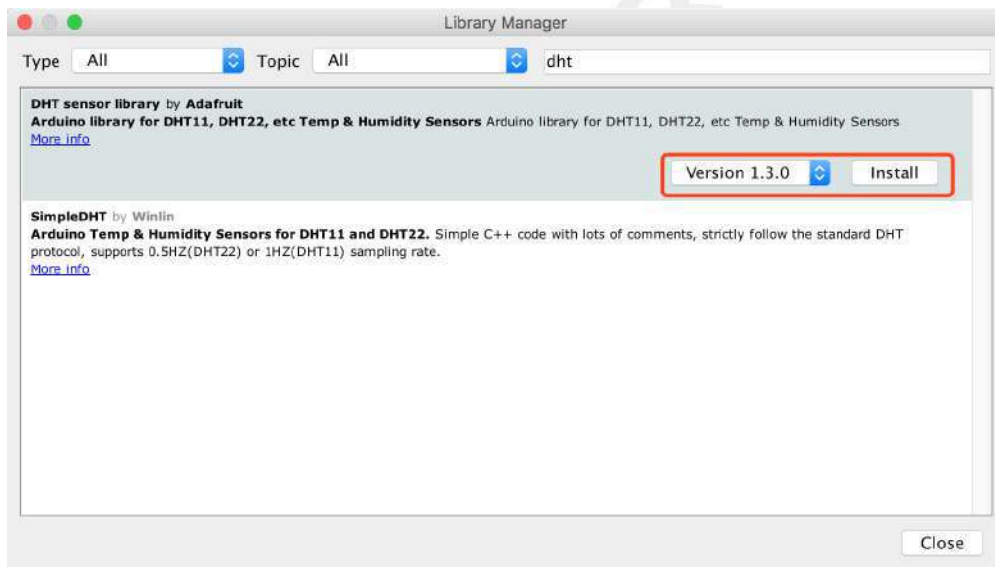
Hình 41. Hình ảnh giao diện HTML

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 42. 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);
WiFiClient client;         // Tạo 1 biến client thuộc kiểu WiFiClient
const char* ssid = "YOUR-WIFI-SSID"; // Tên mạng Wifi được chỉ định sẽ kết nối (SSID)
const char* password = "YOUR-WIFI-PASS"; // Password của mạng Wifi được chỉ định sẽ kết nối
const char* server = "Your-local-IP"; // Địa chỉ IP của máy khi truy cập cùng mạng Wifi
const int port = 8000;     // Port của server đã mở
const int sendingInterval = 2 * 1000; // Biến cập nhật dữ liệu sau mỗi 2s

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

  // 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)) { // 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\n" + "Host: " + server + "\n" + "Connection: close\n" + "Content-Length:
0\n" + "\n\n"); ①
    //-----

    // temp, humi chuyển từ định dạng float sang định dạng string và in ra màn hình serial // terminal trên Arduino.
    Serial.printf("Nhiệt độ %s - Độ ẩm %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**

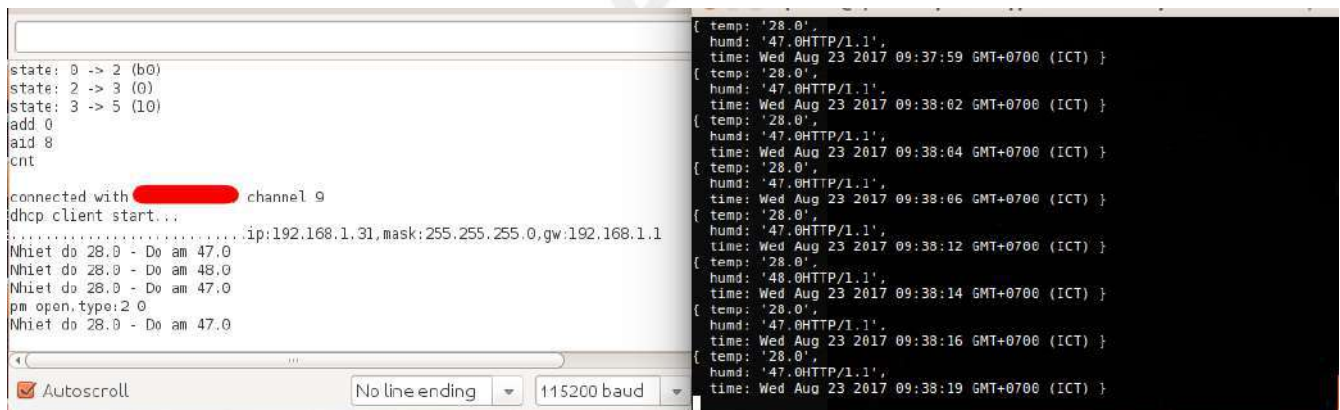
Thực hiện sau khi kiểm tra mã nguồn:

- Chọn Board ESP8266 WiFi Uno trong Arduino IDE
- Nạp chương trình xuống board dùng Arduino IDE

Có thể xem thông các 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 43. Hình ảnh trên Arduino và terminal sau khi kết nối

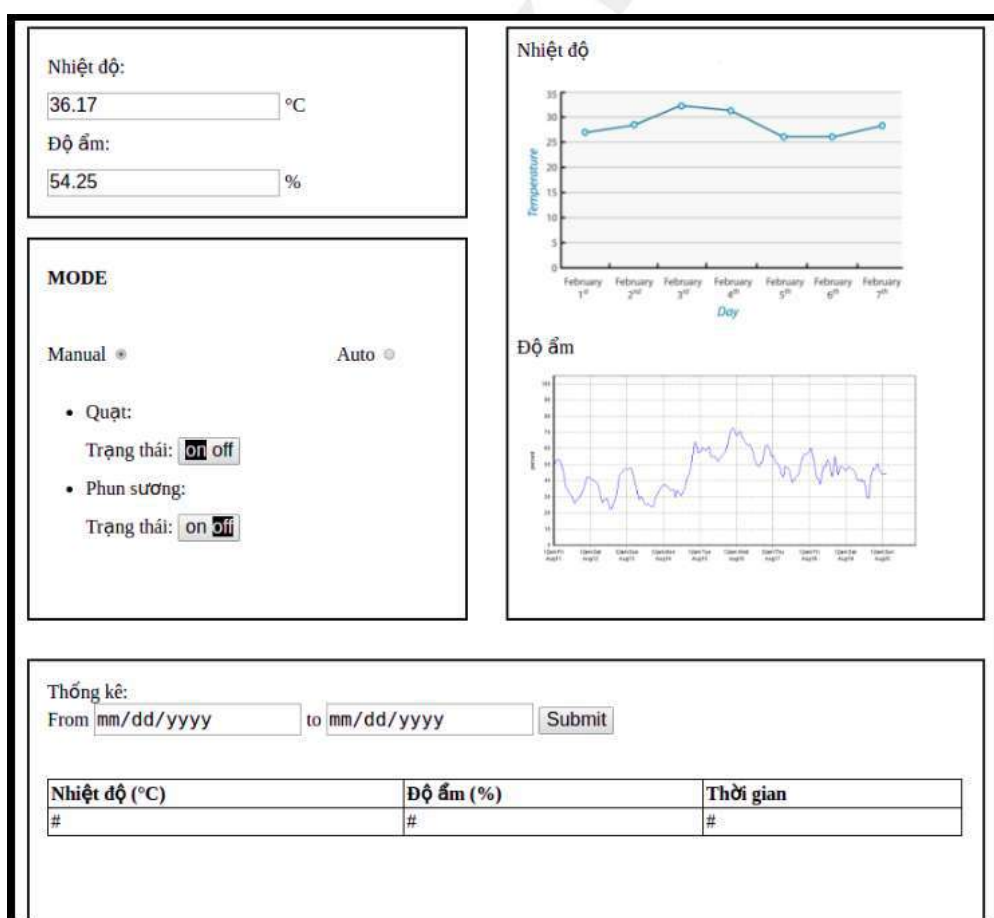
Ứng dụng mở rộng

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 44. Giao diện điều khiển trên trang HTML

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 lí các máy khách. Từ đó là bước đệm để giúp bạn phát triển thêm nhiều các dự án thu thập và xử lí dữ liệu trong tương lai.

IOTMAKER.VN

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 ít chú trọng đến giao diện người dùng (user interface), không có bàn phím hay touchscreen,... để giao tiếp. Vì thế, mỗi khi muốn kết nối thiết bị ESP với một Access Point nào đó, bạn cần phải có một máy tính đã cài đặt sẵn phần mềm biên dịch, tiếp theo là viết code cấu hình lại thông số wifi cho thiết bị, sau đó nạp code cho thiết bị thông qua một cable USB.

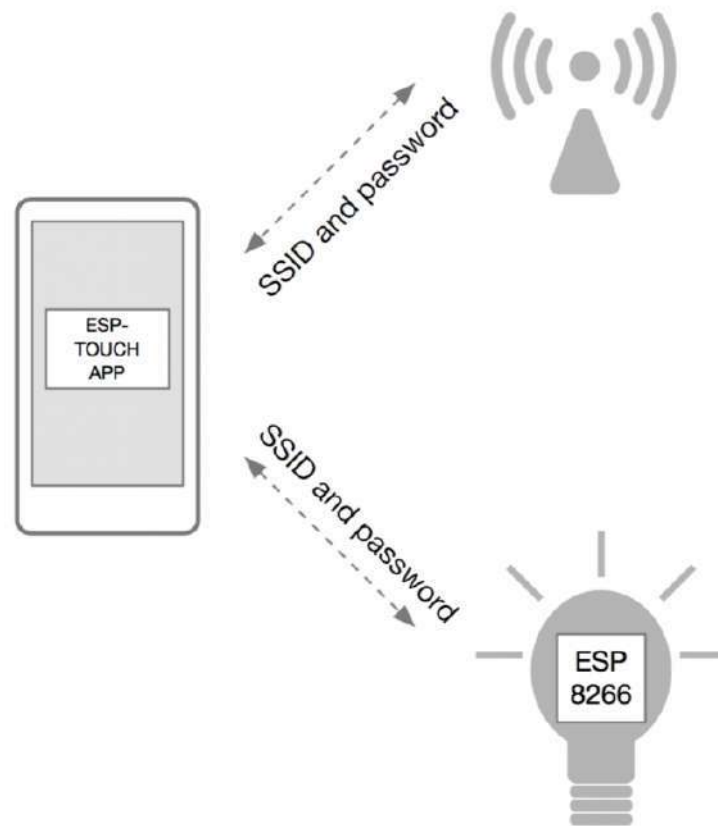
Điều này làm cho việc kết nối wifi trở nên khá bất tiện và phức tạp. Do vậy ESP8266 cung cấp các phương pháp thay thế khác giúp đơn giản hóa việc kết nối trạm ESP (chế độ Station) với một điểm truy cập. Đó là kết nối bằng [SmartConfig](#), [WPS](#) hay [Wifi Manager](#).

IOTMAKER.VN

Smartconfig

Kiến thức

SmartConfig là một giao thức được tạo ra nhằm cấu hình cho các thiết bị 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.



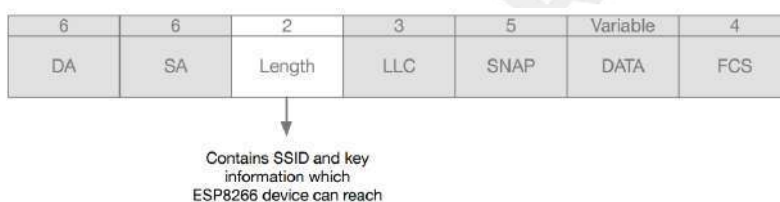
Hình 45. SmartConfig với ESP8266

Chúng ta nên biết rằng, khi 1 điện thoại thông minh đã kết nối vào mạng WiFi có mật khẩu, thì toàn bộ dữ liệu trao đổi giữa Điện thoại và đầu mối khác trong mạng sẽ được mã hóa. Nghĩa là các thiết bị chưa được kết nối mạng và không có mật khẩu thì không thể giải mã được dữ liệu. Vậy làm thế nào để Ứng dụng trên điện thoại gửi thông tin kết nối này đến 1 thiết bị khác chưa hề kết nối mạng. Để làm được điều này, thì nhờ vào 2 đặc điểm sau:

- ESP8266 có khả năng **lắng nghe** tất cả các gói tin không dây WiFi xung quanh nó, bao gồm cả các gói tin đã được mã hóa.
- Các gói tin gửi trong mạng WiFi được mã hóa và không thể đọc được nội dung, tuy nhiên độ dài gói tin là một hằng số. Ví dụ, gói tin A chưa mã hóa có chiều dài là x , khi mã hóa gói tin A thành gói tin B, thì gói tin B sẽ có chiều dài là $x + n$, thì n là hằng số.

Cách thức để giao thức ESPTOUCH thực hiện việc gửi thông tin SSID và mật khẩu cho thiết bị như sau:

- ESP8266 sẽ vào chế độ lắng nghe, lần lượt từng kênh.
- Điện thoại phải kết nối vào mạng WiFi được mã hóa.
- Ứng dụng trên điện thoại sẽ tiến hành gửi các gói tin với nội dung bất kỳ, nhưng có độ dài n theo từng ký tự của SSID và mật khẩu. Ví dụ, ssid của mạng là mynetwork thì sẽ có ký tự m, với ký tự $\text{ascii} = 109$, Ứng dụng sẽ gửi gói tin có độ dài 109 với nội dung bất kỳ, và lặp lại cho đến hết ký tự k, cũng như mật khẩu, và các ký tự khác như CRC.
- Có thể giao thức ESPTOUCH sẽ mã hóa cả các thông số gửi đi, nhưng vẫn giữ nguyên tắc như trên.
- ESP8266 sẽ phát hiện ra các gói tin với độ dài thay đổi này và ghép nối lại thành SSID và password để kết nối vào mạng.
- Khi ESP8266 kết nối thành công đến mạng, ESP8266 sẽ kết nối đến IP của Điện thoại, được cung cấp thông qua ESPTOUCH, và gửi thông tin kết nối thành công đến ứng dụng trên điện thoại.



Hình 46. Gói tin UDP

Lưu ý:

- Khoảng cách giữa thiết bị và router càng xa thì thời gian kết nối sẽ càng lâu.
- 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 phải được cài đặt ở chế độ Station.
- 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

Thực hiện SmartConfig với ESP8266

Trong ví dụ dưới đây, chúng ta sẽ tiến hành kết nối wifi cho board ESP8266 bằng SmartConfig. Sử dụng ứng dụng ESP8266 SmartConfig (Android). Bạn có thể dễ dàng tìm thấy ứng dụng này cũng như các ứng dụng tương tự trên Play Store (Android) hay iTunes (iOS) để thực hiện việc kết nối bằng

SmartConfig này.

Trước tiên, ta sẽ nạp chương trình cho ESP8266. Điểm mấu chốt trong chương trình này chính là hàm `WiFi.beginSmartConfig()` được cung cấp trong thư viện `ESP8266WiFi`. Hàm này cho phép thiết bị khởi động chế độ SmartConfig, thu thập các thông tin từ các gói tin và giải mã chúng để có thể kết nối vào mạng Wifi.

Sau khi nạp xong chương trình, ta nhấn giữ button (GPIO0) trong 3s để thiết bị đi vào chế độ smartconfig. (Lúc này bạn sẽ thấy led trên board nhấp nháy nhanh hơn). Dùng smart phone của bạn truy cập vào wifi muốn kết nối, sau đó mở ứng dụng smartconfig và nhập các thông tin SSID và PASSWORD (nếu có) của wifi. Nhấn **CONFIRM** để xác nhận.

► <https://www.youtube.com/watch?v=-RqMKvMLPi0> (YouTube video)

Video Demo

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;

/* Hàm kiểm tra trạng thái 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;
    } else if (digitalRead(PIN_BUTTON) == 1) { // Nếu button không được nhấn và giữ đủ 3s thì
        lastPress = millis(); // gán biến lastPress bằng thời điểm khi gọi hàm
    }
    return 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 chế độ smartconfig hay không.

/* Vào chế độ Smartconfig*/
void enter_smartconfig()
{
    if (in_smartconfig == false) { // Kiểm tra biến trạng thái, nếu không ở chế độ smartconfig thì
        in_smartconfig = true; // Gán biến trạng thái bằng "true", nghĩa là đang trong smartconfig
        ticker.attach(0.1, tick); // Nhấp nháy led chu kì 0.1s.
    }
}
```

```

    WiFi.mode(WIFI_STA);           // Thiết lập kết nối cho thiết bị ở chế độ Station mode
    WiFi.beginSmartConfig();        // Bắt đầu chế độ smartconfig
    Serial.println("Enter smartconfig"); // In thông báo "Enter smartconfig" ra màn hình
}
}

/* Thoát chế độ smartconfig*/
void exit_smart()
{
    ticker.detach();               // Ngừng nháy led
    LED_ON();                      // Bật LED
    in_smartconfig = false;        // Gán biến trạng thái trở về ban đầu.
    Serial.println("Connected, Exit smartconfig"); // In thông báo ra màn hình.
}

/* Cài đặt các thông số ban đầu*/
void setup() {
    Serial.begin(115200);           // Tốc độ baud = 115200
    Serial.setDebugOutput(true);    // 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);     // Chớp tắt led chu kì 1s
    Serial.println("Setup done");    // In thông báo đã cài đặt xong
}

/* 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
    }
    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 hết nối wifi thành công, thoát chế độ smartconfig
    }
    if (WiFi.status() == WL_CONNECTED) {
        //Chương trình của bạn khi thiết bị đã được kết nối wifi
    }
}
}

```

Thực hiện sau khi kiểm tra mã nguồn:

- Chọn Board ESP8266 WiFi Uno trong Arduino IDE
- Nạp chương trình xuống board dùng Arduino IDE

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.



Hình 47. Nút nhấn WPS trên router

Thực hiện WPS với ESP8266

ESP8266 hỗ trợ hàm `WiFi.beginWPSConfig()` 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()
{
    // Cài đặt các thông số ban đầu
    Serial.begin(115200);
    WiFi.mode(WIFI_STA);
    // Kết nối với AP cũ đã vào trước đó, SSID và password được lưu trong bộ nhớ flash của thiết bị
    WiFi.begin("", "");
    delay(4000);

    // Kiểm tra xem wifi đã được kết nối chưa, nếu chưa, bắt đầu kết nối bằng WPS
    // Lưu ý, cần phải đảm bảo rằng Router của bạn đang ở trong trạng thái WPS.
    while (WiFi.status() != WL_CONNECTED)
    {
        Serial.println("\nAttempting connection ...");
        WiFi.beginWPSConfig();
        delay(6000);
    }
    // Khi kết nối thành công, in thông báo ra màn hình cùng với các thông số của Wifi vừa kết nối.
    Serial.println("\nConnection already established.");
    Serial.println(WiFi.localIP());
    Serial.println(WiFi.SSID());
    Serial.println(WiFi.macAddress());
}

void loop()
{
    // chương trình chính
}
```

Thực hiện sau khi kiểm tra mã nguồn:

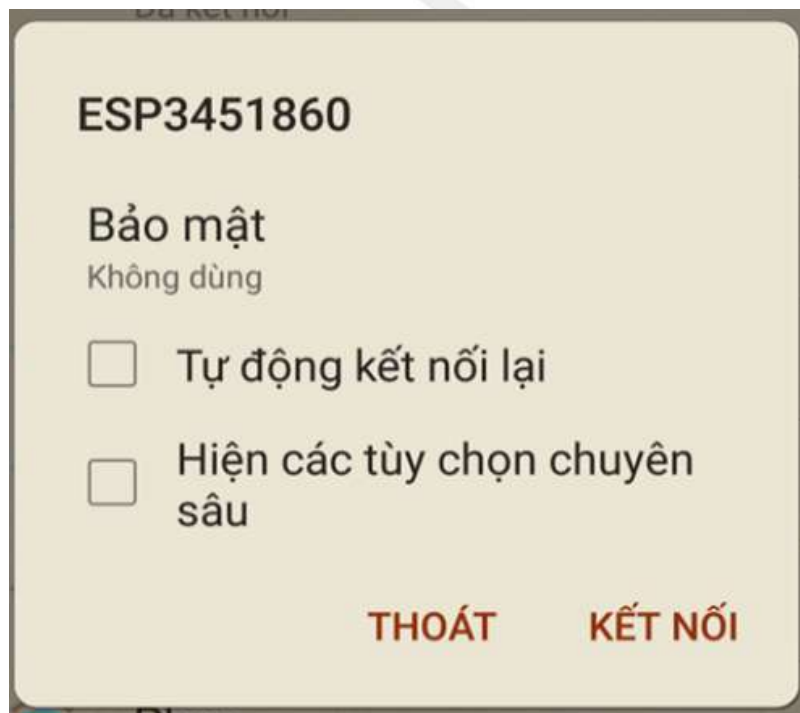
- Chọn Board ESP8266 WiFi Uno trong Arduino IDE
- Nạp chương trình xuống board dùng Arduino IDE

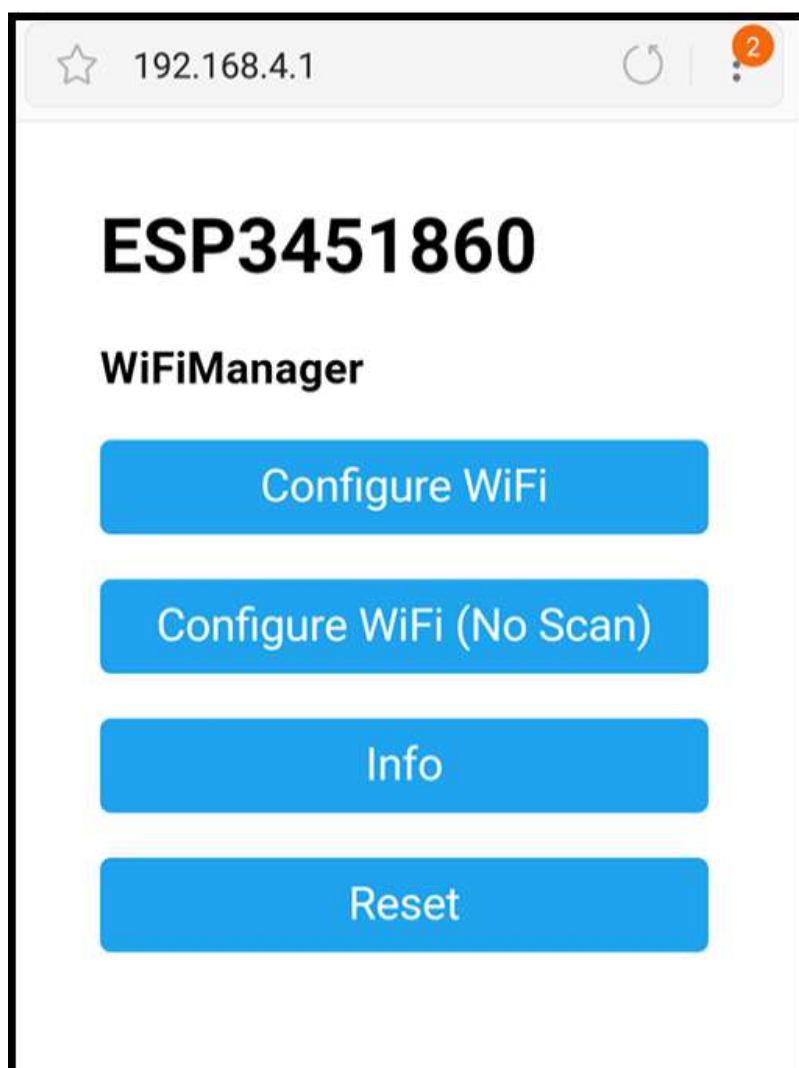
Wifi Manager

WifiManager là một thư viện cấu hình ESP8266 kết nối vào mạng WiFi cục bộ sử dụng giao diện Web. Bằng cách khởi động 1 mạng WiFi riêng với Captive Portal, ESP8266 sẽ cho phép các thiết bị khác như máy tính, điện thoại thông minh kết nối vào, đồng thời chuyển hướng mọi kết nối đến giao diện Web do ESP8266 tạo nên. Trên giao diện này, sẽ cung cấp các trường để người dùng có thể dễ dàng quét mạng xung quanh, chọn mạng WiFi, nhập mật khẩu, lưu cấu hình.

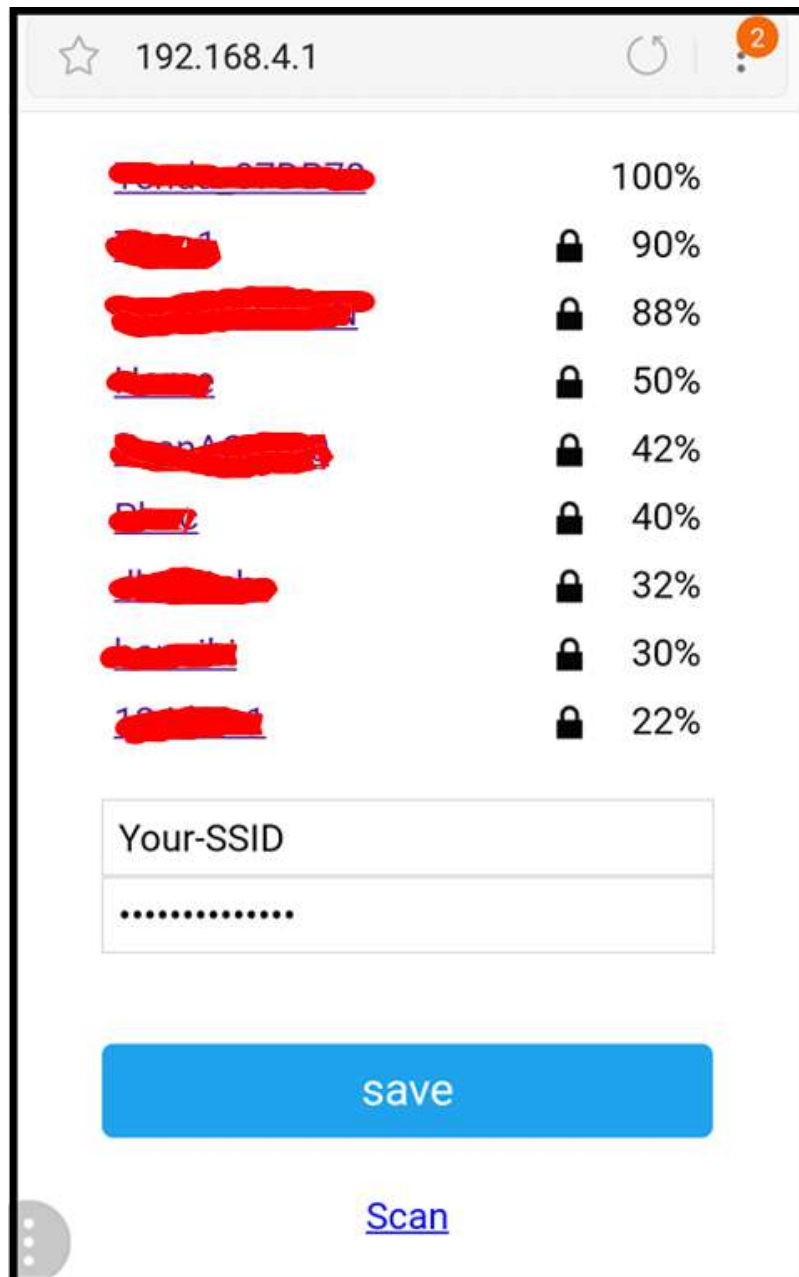
Hoạt động cơ bản WifiManager

- Khi ESP8266 khởi động, ESP8266 sẽ vào chế độ STATION và sẽ tự động kết nối đến một Access Point với các thông tin kết nối đã được lưu vào ESP8266 ở lần kết nối thành công trước đó.
- Nếu như kết nối không thành công (có thể là Access Point lần trước không còn nữa, hay sai mật khẩu, hoặc chưa có thông tin của bất cứ Access Point nào trong ESP8266), lúc này ESP8266 sẽ vào chế độ AP với một DNS trở về chính nó (có thể thiết lập DNS trở về địa chỉ khác) 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 cài đặt mật khẩu cho ESP8266 AP) tương tự như sau :





- Sau khi vào được giao diện option của ESP8266 AP ở địa chỉ 192.168.4.1, 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, xem thêm [Cài đặt thư viện Arduino](#)

Code

```

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

//các thư viện cần thiết
#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());
    Serial.println(myWiFiManager->getConfigPortalSSID());
}

// Cài đặt thông số ban đầu
void setup()
{
    Serial.begin(115200);

    //Khai báo wifiManager thuộc class WiFiManager, được quy định trong file WiFiManager.h
    WiFiManager wifiManager;
    //có thể reset các cài đặt cũ bằng cách gọi hàm:
    //wifiManager.resetSettings();

    //Cài đặt callback, khi kết nối với wifi cũ thất bại, thiết bị sẽ gọi hàm callback
    //và khởi động chế độ AP với SSID được cài tự động là "ESP+chipID"
    wifiManager.setAPCallback(configModeCallback);
    if (!wifiManager.autoConnect())
    {
        Serial.println("failed to connect and hit timeout");
        //Nếu kết nối thất bại, thử kết nối lại bằng cách reset thiết bị
        ESP.reset();
        delay(1000);
    }
    //Nếu kết nối wifi thành công, in thông báo ra màn hình
    Serial.println("connected...yeey :)");
}

void loop()
{
    // Chương trình chính
}

```

Thực hiện sau khi kiểm tra mã nguồn:

- Chọn Board ESP8266 WiFi Uno trong Arduino IDE
- Nạp chương trình xuống board dùng Arduino IDE

Mở rộng

Ngoài các chế độ cơ bản, thì thư viện WiFiManager còn nhiều tính năng hữu ích khác như **startConfigPortal** để khởi động cấu hình khi cần (ví dụ nhấn nút để cấu hình), bổ sung các trường tùy chọn trên giao diện Web, tùy chọn lại giao diện ...

Các thông tin API và ví dụ bạn có thể dễ dàng tìm thấy tại github.com/tzapu/WiFiManager

Tổng kết

Để triển khai một ứng dụng IoT thực tế thì đòi hỏi rất nhiều vấn đề, một trong số những điều quan trọng là dễ dùng, dễ cấu hình cho người sử dụng và phải bảo mật trong quá trình cung cấp thông tin cho thiết bị. Tùy thuộc vào nhu cầu phát triển sản phẩm và tính năng của sản phẩm mà bạn có thể lựa chọn cho mình phương pháp cấu hình phù hợp. Ví dụ, nếu thiết bị có nút nhấn và có phần mềm trên điện thoại, thì SmartConfig và WPS là một sự lựa chọn. Nếu là 1 bóng đèn thông minh không có gì cả, thì WiFiManager lại hữu hiệu.

IOTMAKER.VN

MQTT



MQTT (Message Queuing Telemetry Transport) là một giao thức gửi 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 gửi dữ liệu vào kênh đã đăng ký. Khi một Client gửi 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 gửi dữ liệu đúng 1 lần, quá trình gửi được xác nhận bởi chỉ giao thức TCP/IP, giống kiểu đem con bỏ chợ.
- **QoS1** Broker/Client sẽ gửi 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 gửi 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 gửi 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 tôi 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

Như chúng ta đã tìm hiểu ở phần trước, 2 thành phần publisher và subscriber là đặc trưng tạo nên giao thức MQTT. Các MQTT Client không kết nối trực tiếp với nhau, mọi gói dữ liệu được gửi đi đều thông qua MQTT Broker. Để có thể triển khai các ứng dụng của MQTT Client, chúng ta cần MQTT Broker (sẽ được trình bày trong phần sau). Ở phần này chúng ta sẽ làm quen với giao thức MQTT bằng các ví dụ sử dụng MQTT Client thông dụng và các dịch vụ MQTT Broker miễn phí và phổ biến, 2 trong số chúng là test.mosquitto.org và cloudmqtt.com

MQTT Lens

Thông tin

MQTT Lens là một tiện ích mở rộng của Chrome (Chrome Extension), nó sử dụng trình duyệt Chrome để kết nối đến MQTT Broker cũng như test các tính năng publish/subscribe của giao thức MQTT. Đây là một công cụ rất hữu ích để kiểm tra kết nối đến MQTT Broker và kiểm tra việc gửi và nhận gói tin.

Một số thông tin của MQTT Lens được trình bày ở bảng dưới.

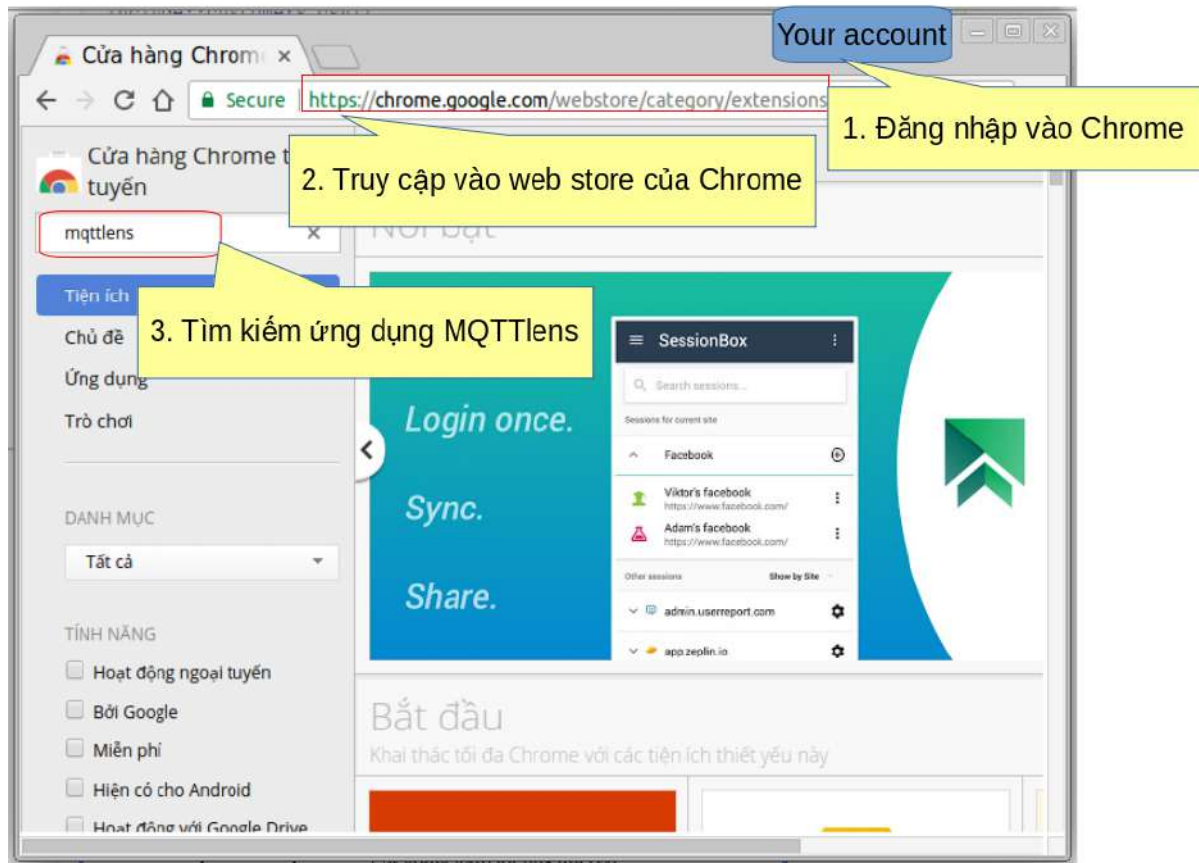
Bảng 2. Short Profile

Type	Chrome App
License	MIT
Operating Systems	Windows, Linux & MacOSX
Website	"https

Kết nối

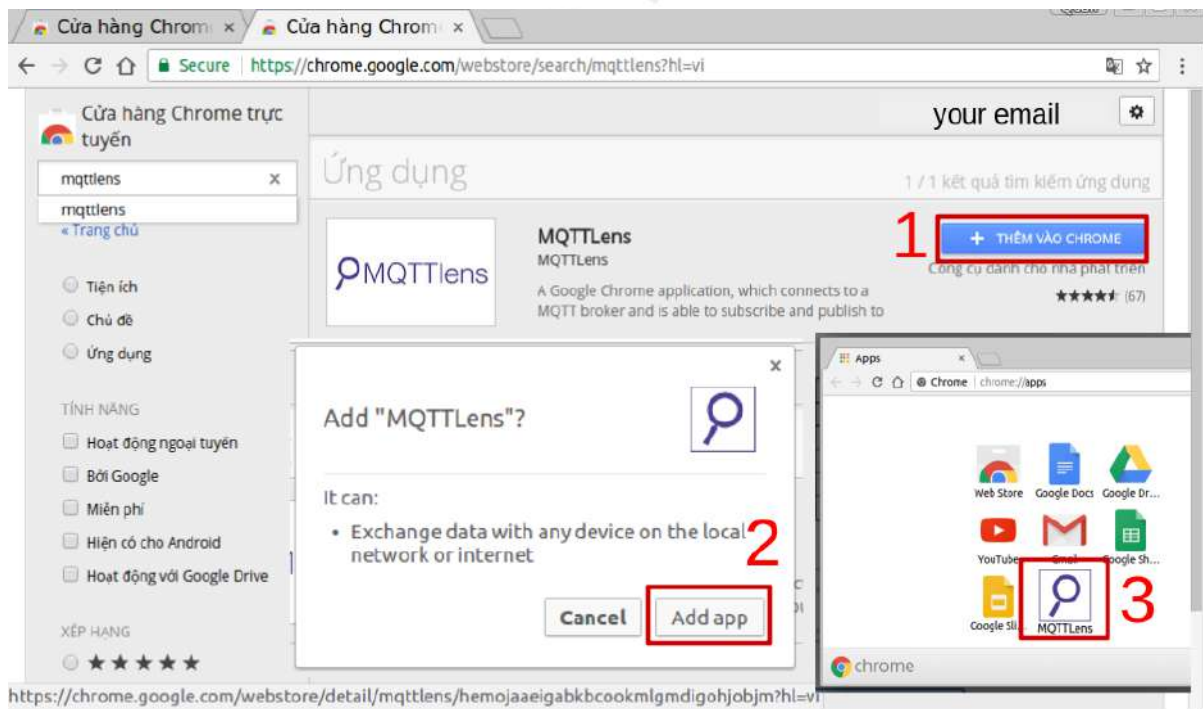
Chúng ta sẽ sử dụng công cụ này với dịch vụ MQTT Broker tại iot.eclipse.org được trình bày như các bước bên dưới:

- Bước 1: Cài đặt trình duyệt Chrome, thực hiện đăng nhập tài khoản của bạn vào chrome, truy cập vào địa chỉ chrome.google.com/webstore/category/extensions và gõ mqttlens vào mục tìm kiếm tiện ích như hình bên dưới.



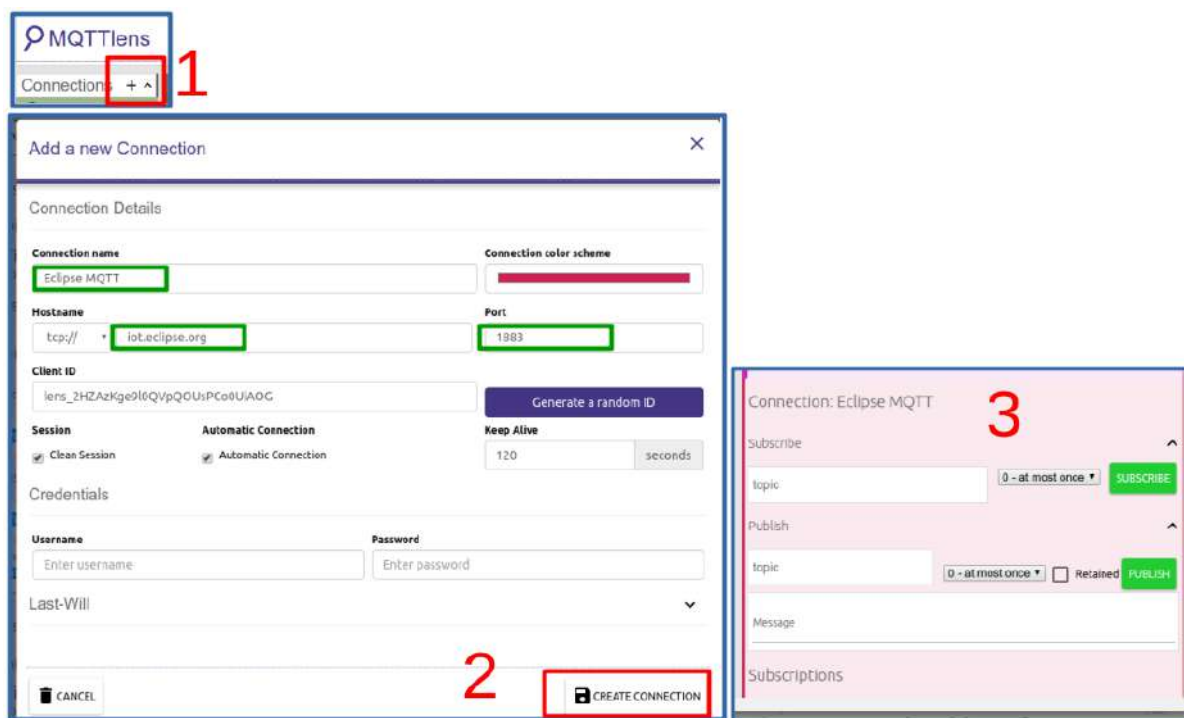
Hình 48. Hình ảnh tìm kiếm tiện ích mqttlens trên chrome store

- Bước 2: Thêm và khởi chạy MQTT lens



Hình 49. Hình ảnh các bước thêm và khởi chạy tiện ích MQTT lens

- Bước 3 : Tạo 1 MQTT Client kết nối đến MQTT Broker eclipse như các bước bên dưới.

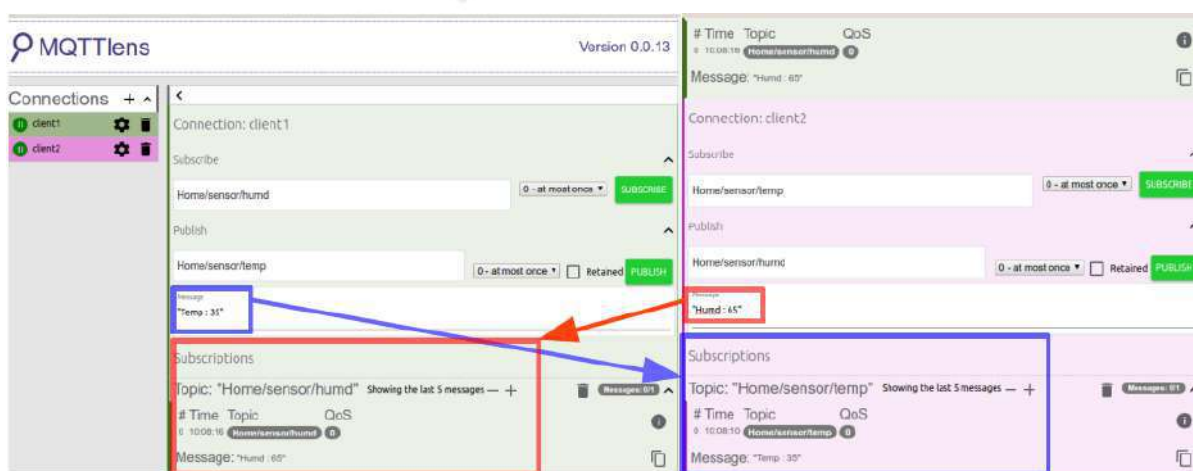


Hình 50. Hình ảnh tạo 1 MQTT client

Giải thích

Chúng ta sẽ tạo 1 connection có tên eclipse MQTT với host name của MQTT Broker là iot.eclipse.org, Broker này sẽ giúp trao đổi dữ liệu của các Client với nhau và lắng nghe các Client ở port 1883 (port sử dụng giao thức MQTT và không mã hóa dữ liệu, các port khác tham khảo tại test.mosquitto.org) Ở connection này sẽ đăng kí nhận gói tin tại topic [Home/garden/sensor/#](#) (kí tự # cho phép subscribe các topic [Home/garden/sensor/1](#), [Home/garden/sensor/2](#) vv...). Tiếp theo chúng ta sẽ publish 1 gói dữ liệu với nội dung "Temp in garden: 27degree Celcius " tại topic [Home/garden/sensor/1](#).

Kết quả: Tại mục subscriptions, chúng ta sẽ nhận được gói dữ liệu đã publish do đã subscribe topic [Home/garden/sensor/#](#) như hình bên dưới.



Hình 51. Hình ảnh dữ liệu nhận được sau khi publish gói tin

Mở rộng

Tạo nhiều connection để subscribe và publish các gói tin với MQTT Broker iot.eclipse.org đồng thời test các gói tin với QoS và LWT

MQTT.js

MQTT.js là một thư viện MQTT client, được viết bằng ngôn ngữ JavaScript trên nền tảng Node.js và hỗ trợ MQTT Over Websocket (MOW).

MQTT.js là dự án mã nguồn mở (open source), bạn có thể tải MQTT.js bản cập nhật mới nhất tại github.com/mqttjs/MQTT.js

Cài đặt

Trước tiên ta cần kiểm tra hệ điều hành đã hỗ trợ Node.js trước khi cài đặt MQTT.js. Nếu chưa thì có thể tham khảo cách cài đặt tại nodejs.org/en/

Khởi tạo một dự án Node.js. Để dễ quản lý, có thể tạo một thư mục riêng, ví dụ **mqtt-client** và một file javascript trong đó, ví dụ như **client-a.js**. Đi đến thư mục này và mở terminal (linux OS) hoặc Command Prompt (trên Windows OS) và dùng lệnh:

```
npm init
```

Khi chạy lệnh này, bạn cũng cần phải khai báo thêm một số thông tin cho dự án như tên, phiên bản, keywords, tác giả,... Sau khi tạo xong, trong thư mục vừa tạo sẽ xuất hiện một file là **package.json** với nội dung là các phần đã khai báo. File này cũng chứa thuộc tính dùng để lưu trữ các package chúng ta đã cài đặt.

Tiếp theo chúng ta sẽ cài **MQTT.js**, sử dụng lệnh:

```
npm install mqtt --save
```

Sau khi cài đặt xong, bạn có thể sử dụng module **mqtt** để thực hiện việc kết nối MQTT Client với Broker, publish message hay subscribe topic. Tất nhiên, toàn bộ các file liên quan đến thư viện sẽ nằm trong thư mục **node_modules**, trong thư mục dự án.

Để hiểu rõ hơn cách hoạt động của MQTT.js, chúng ta sẽ tạo ra thêm 1 số file mã nguồn Javascript (file .js) là **client-a.js** và **client-b.js** thực hiện subscribe và publish các gói tin.

Nội dung thực hiện

2 Client này sẽ kết nối vào cùng 1 MQTT Broker. Client A sẽ subscribe kênh **/client-a/sub**, nếu nhận bất kỳ dữ liệu nào được publish vào kênh này, client A sẽ public dữ liệu **Hello from client A** vào kênh **/client-b/sub** và đóng kết nối, kết thúc. Client B sẽ subscribe kênh **/client-b/sub**, nếu nhận bất kỳ dữ liệu nào được public vào kênh này, client B sẽ đóng kết nối và kết thúc. Ngay khi kết nối vào Broker, client B sẽ public 1 gói tin **Hello from client B** vào kênh **/client-a/sub**

Mã nguồn của client A

client-a.js

```
// tạo biến mqtt sử dụng các chức năng của module mqtt
var mqtt = require('mqtt')
// tạo biến client sử dụng thuộc tính connect để kết nối đến broket MQTT với hostname mqtt://iot.eclipse.org
var client = mqtt.connect('mqtt://iot.eclipse.org')
// function có chức năng subscribe 1 topic nếu đã kết nối thành công đến broker
client.on('connect', function() {
  console.log('Client A connected')
  // client subscribe topic /client-a/sub
  client.subscribe('/client-a/sub')
})
// function có chức năng gửi 1 gói tin đến đến topic đã đăng kí
client.on('message', function(topic, message) {
  // in ra màn hình console 1 message ở định dạng string
  console.log(message.toString())
  // publish gói tin 'Hello from client A' đến topic /client-b/sub
  client.publish('/client-b/sub', 'Hello from client A')
  // đóng kết nối của client
  client.end()
})
console.log('Client A started')
```

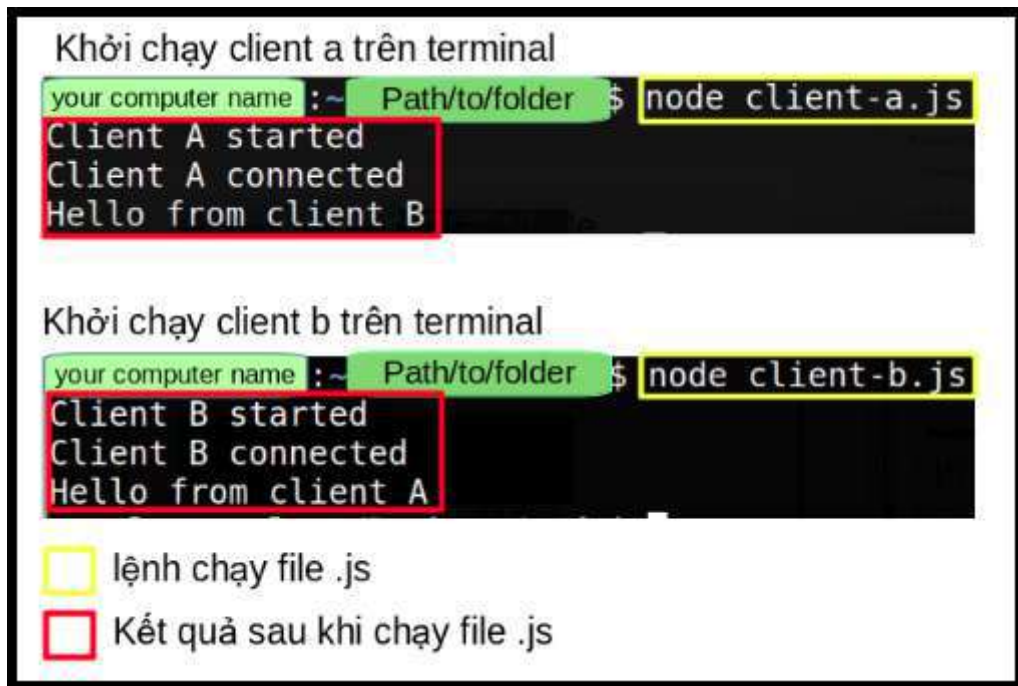
Mã nguồn của client B

client-b.js

```
// tạo biến mqtt sử dụng các chức năng của module mqtt
var mqtt = require('mqtt')
// tạo biến client sử dụng thuộc tính connect để kết nối đến broket MQTT với hostname mqtt://iot.eclipse.org
var client = mqtt.connect('mqtt://iot.eclipse.org')
// function có chức năng subscribe 1 topic nếu đã kết nối thành công đến broker
client.on('connect', function() {
  console.log('Client B connected')
  // client subscribe topic /client-b/sub
  client.subscribe('/client-b/sub')
  // publish gói tin 'Hello from client B' đến topic /client-a/sub
  client.publish('/client-a/sub', 'Hello from client B')
})

client.on('message', function(topic, message) {
  // in ra màn hình console 1 message ở định dạng string
  console.log(message.toString())
  // đóng kết nối của client
  client.end()
})
console.log('Client B started')
```

Kết quả hiển thị như hình bên dưới:



Hình 52. Hình ảnh kết quả khi khởi chạy các MQTT client

Ngoài ra, MQTT.js còn cung cấp thêm các lệnh để có thể tương tác với Broker thông qua terminal. Để làm được điều này, chúng ta cài đặt MQTT.js như một module toàn cục bằng cách sử dụng lệnh:

```
npm install mqtt -g.
```

Bạn có thể kiểm tra bằng cách mở 2 màn hình terminal, ở màn hình 1 (tạm gọi là subscriber) sẽ subscribe vào topic tên là "topicA" bằng lệnh:

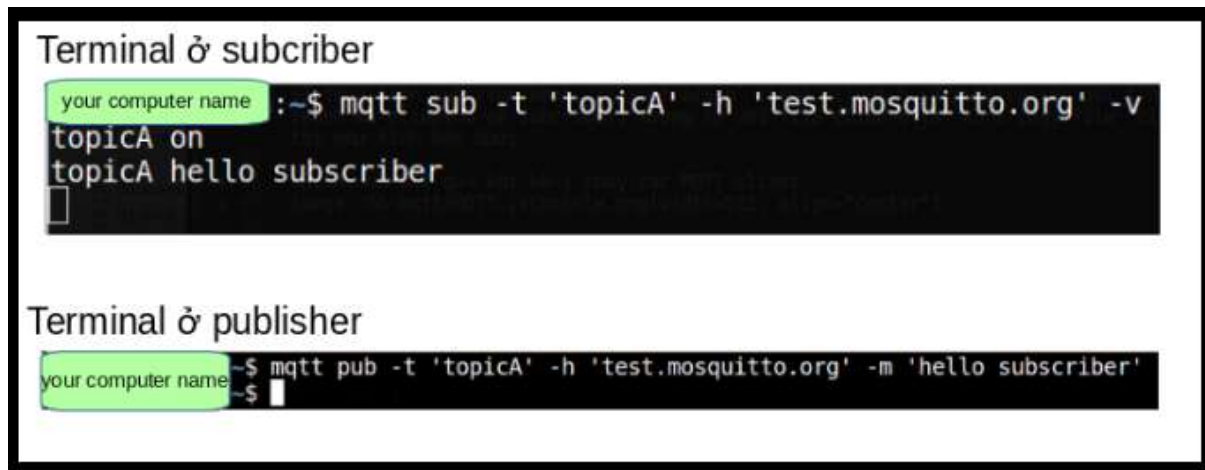
```
mqtt sub -t 'topicA' -h 'test.mosquitto.org' -v
```

Ở terminal thứ 2 (tạm gọi là publisher) thực hiện publish một tin nhắn với nội dung "hello subscriber" tới "topicA":

```
mqtt pub -t 'topicA' -h 'test.mosquitto.org' -m 'hello subscriber'
```

Các options:

- **-t** : MQTT topic, nơi sẽ thực hiện pushlish 1 message.
- **-h** : Xác định máy chủ sẽ kết nối đến.
- **-m** : Gửi 1 message dùng command line.
- **-v** : verbose, option cho phép ghi lại nhật kí hoạt động của các tập tin trong file cấu hình.



Hình 53. Hình ảnh message được publish dùng command line



Để xem thêm các API khác trong MQTT.js, bạn có thể sử dụng lệnh: `mqtt help [command]`.

ESP8266 MQTT Client

Thực tế có khá nhiều thư viện MQTT cho ESP8266 trên Arduino, ở đây chúng ta chỉ đề cập đến 2 thư viện phổ biến là `PubSubClient` và `ESP8266MQTTClient`

PubSubClient

Trong phần này chúng ta sẽ thực hiện kết nối board ESP8266 WiFi Uno đến 1 broker sử dụng thư viện PubSubClient.

- **Bước 1** : Download thư viện PubSubClient tại đường dẫn github.com/knolleary/pubsubclient và add vào chương trình Arduino. Ngoài ra có thể import thư viện này trong Arduino bằng cách tìm kiếm thư viện với từ khóa `PubSubClient`, chọn thư viện PubSubClient của tác giả Nick O'Leary và nhấn install.
- **Bước 2** : Viết và nạp chương trình cho ESP8266. Mã nguồn được trình bày ở phía dưới

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

const char* ssid = ".....";
const char* password = ".....";
const char* mqtt_server = "broker.mqtt-dashboard.com";

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  pinMode(16, OUTPUT);
  Serial.begin(115200);
  // hàm thực hiện chức năng kết nối Wifi và in ra địa chỉ IP của ESP8266
  setup_wifi();
  // cài đặt server là broker.mqtt-dashboard.com và lắng nghe client ở port 1883
```



```

client.setServer(mqtt_server, 1883);
// gọi hàm callback để thực hiện các chức năng publish/subscribe
client.setCallback(callback);
// gọi hàm reconnect() để thực hiện kết nối lại với server khi bị mất kết nối
reconnect();
}

void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  // kết nối đến mạng Wifi
  WiFi.begin(ssid, password);
  // in ra dấu . nếu chưa kết nối được đến mạng Wifi
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // in ra thông báo đã kết nối và địa chỉ IP của ESP8266
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int length) {
  // in ra tên của topic và nội dung nhận được từ kênh MQTT lens đã publish
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  // kiểm tra nếu dữ liệu nhận được từ topic ESP8266/LED_GPIO16/status là chuỗi "on"
  // sẽ bật led GPIO16, nếu là chuỗi "off" sẽ tắt led GPIO16
  if ((char)payload[0] == 'o' && (char)payload[1] == 'n') //on
    digitalWrite(16, LOW);
  else if ((char)payload[0] == 'o' && (char)payload[1] == 'f' && (char)payload[2] == 'f') //off
    digitalWrite(16, HIGH);
  Serial.println();
}

void reconnect() {
  // lặp cho đến khi được kết nối trở lại
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP8266")) {
      Serial.println("connected");
      // publish gói tin "Connected!" đến topic ESP8266/connection/board
      client.publish("ESP8266/connection/board", "Connected!");
      // đăng ký nhận gói tin tại topic ESP8266/LED_GPIO16/status
      client.subscribe("ESP8266/LED_GPIO16/status");
    } else {
      // in ra màn hình trạng thái của client khi không kết nối được với MQTT broker
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // delay 5s trước khi thử lại
      delay(5000);
    }
  }
}

void loop() {
  // kiểm tra nếu ESP8266 chưa kết nối được thì sẽ thực hiện kết nối lại
  if (!client.connected()) {

```



```

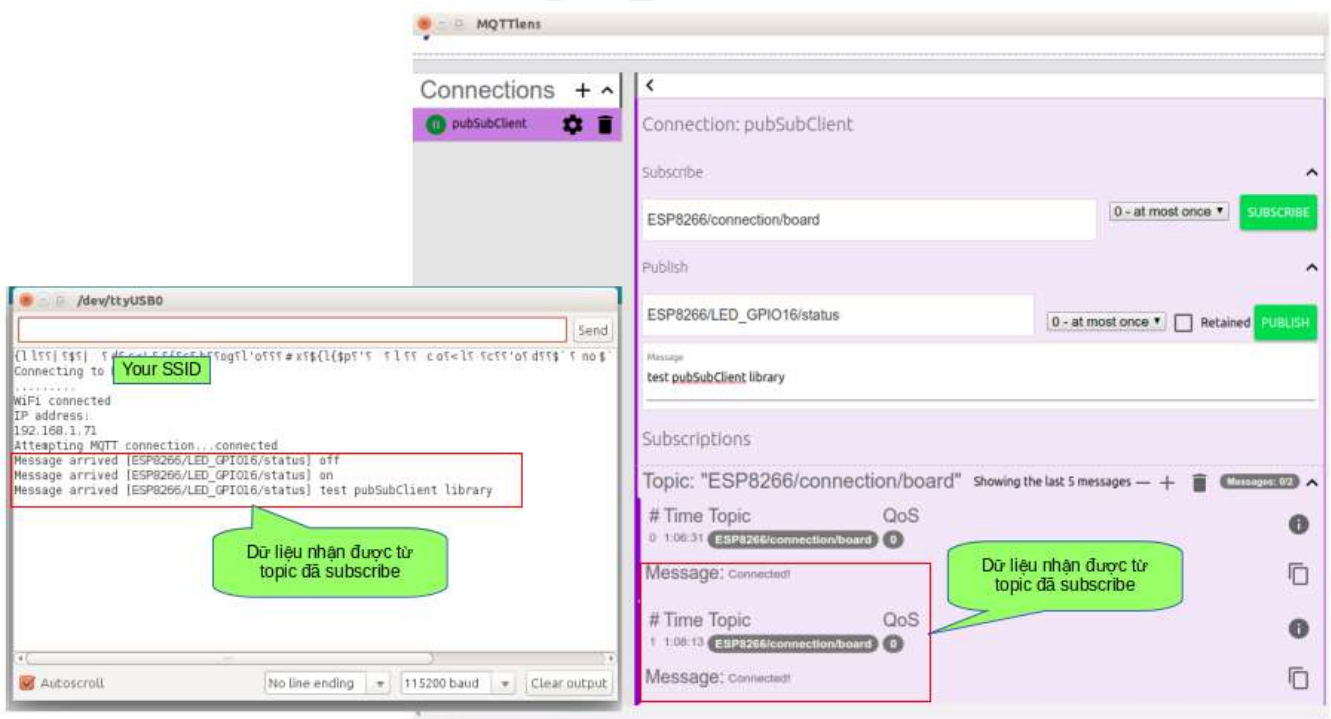
    reconnect();
  }
  client.loop();
}

```

Giải thích mã nguồn:

Chúng ta sẽ tạo một biến `espClient` thuộc lớp `WiFiClient`, biến này được khai báo là MQTT Client và sử dụng các thuộc tính của thư viện `PubSubClient`. Tại hàm `setup()` sẽ thiết lập ESP8266 ở chế độ station, kết nối đến mạng wifi. Bên cạnh đó hàm `setup()` cũng sẽ thực hiện chức năng tự động kết nối lại với MQTT Broker khi xảy ra mất kết nối đồng thời thực hiện các chức năng publish, subscribe của 1 MQTT Client thông qua hàm `reconnect()`. Hàm `callback()` có nhiệm vụ lấy dữ liệu của các publisher khi publish 1 message sau đó gửi đến các client đã subscribe topic đó và kiểm tra nội dung của message để điều khiển led ở GPIO16. Hàm `loop()` có chức năng kết nối Client là ESP8266 với Broker, thực hiện chức năng publish 1 message và subscribe topic. `client.loop()` sẽ kiểm tra thời gian kết nối của Client với gói `KEEP_ALIVE` để đưa ra các thông tin về trạng thái kết nối của ESP8266 đồng thời lấy dữ liệu của message từ buffer để gửi đến các Client đã subscribe topic.

- **Bước 3 :** Mở MQTT lens trên trình duyệt Chrome, tạo 1 connection với host name `broker.mqtt-dashboard.com`, sử dụng port 1883. Thực hiện subscribe topic `ESP8266/connection/board`. Sau khi nhấn nút subscribe trên MQTT lens sẽ xuất hiện 1 message gửi từ esp8288 với nội dung `connected`. Thực hiện pushlish các message vào topic `ESP8266/LED_GPIO16/status`. Nếu pushlish message với nội dung `on`, led GPIO16 trên board sẽ sáng, pushlish message `off` led GPIO16 trên board sẽ tắt. Các message với nội dung khác thì vẫn sẽ hiển thị dữ liệu nhận được trên serial terminal của Arduino nhưng sẽ không có tác dụng điều khiển led GPIO16. Kết quả hiển thị như hình bên dưới:



Hình 54. Kết quả hiển thị trên serial terminal và MQTT lens khi sử dụng thư viện pubsubClient

ESP8266MQTTClient

Tiếp theo, chúng ta sẽ tìm hiểu cách sử dụng thư viện ESP8266MQTTClient, thư viện được cộng đồng developer đánh giá là ổn định để sử dụng hơn so với thư viện PubSubClient thông qua 1 ứng dụng điều khiển led trên board ESP8266 WiFi Uno bằng 1 ứng dụng trên điện thoại smartphone.

- **Bước 1** : Download thư viện **ESP8266MQTTClient** tại đường dẫn github.com/tuanpmt/ESP8266MQTTClient và add vào chương trình Arduino. Ngoài ra có thể import thư viện này trong Arduino bằng cách tìm kiếm thư viện với từ khóa **ESP8266MQTT**, chọn thư viện của tác giả Tuan PM, version 1.3 và nhấn install.
- **Bước 2** : Viết và nạp chương trình cho ESP8266. Mã nguồn được trình bày ở phía dưới.

```
#include <ESP8266MQTTClient.h>
#include <ESP8266WiFi.h>

#define ledPin 16 //Led on board ESP8266 WiFi Uno

MQTTClient mqtt;

const char* ssid = "Your SSID";
const char* password = "Your password";

void setup() {
    Serial.begin(115200);
    pinMode(ledPin, OUTPUT);

    // OFF led GPIO16 khi bắt đầu chương trình
    digitalWrite(ledPin, HIGH);

    // Thiết lập ESP8266 ở chế độ STA và kết nối Wifi
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    Serial.print("\nConnecting to ");
    Serial.println(ssid);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());

    // Kết nối đến server MQTT, in ra id của topic "esp8266/GPIO16" đồng thời đăng kí nhận message với gói QoS 0,
    // và subscribe các topic "hello/esp8266"
    mqtt.onConnect([]() {
        Serial.printf("MQTT: Connected\r\n");
        Serial.printf("Subscribe id: %d\r\n", mqtt.subscribe("esp8266/GPIO16", 0));
        mqtt.subscribe("esp8266/GPIO16", 0);
    });

    // Thực hiện chức năng subscribe topic và publish các message
    mqtt.onSubscribe([](int sub_id) {
        //in ra id của các topic đã subscribe là "hello/esp8266" và "MQTTlens/test/#"
        Serial.printf("Subscribe topic id: %d ok\r\n", sub_id);

        //publish message có nội dung hello app đến topic Broker/app với gói QoS 0 và cờ retain 0
        mqtt.publish("Broker/app", "hello app", 0, 0);
    });
}
```

```
// Xử lý dữ liệu nhận được của các topic đã subscribe
mqtt.onData([])(String topic, String data, bool cont) {
    Serial.printf("Data received, topic: %s, data: %s\r\n", topic.c_str(), data.c_str());

    // Nếu chuỗi nhận được là 'on' sẽ ON led trên board ESP8266 WiFi Uno, chuỗi 'off' sẽ OFF led
    if (topic == "esp8266/GPIO16" && data[0] == 'o' && data[1] == 'n' && data[2] == '\0') {
        digitalWrite(ledPin, LOW);
        Serial.println("Turn on the led on board");
    } else if (topic == "esp8266/GPIO16" && data[0] == 'o' && data[1] == 'f' && data[2] == 'f' && data[3] == '\0' ) {
        digitalWrite(ledPin, HIGH);
        Serial.println("Turn off the led on board");
    }
};

// khởi tạo broker MQTT là iot.eclipse.org sử dụng phương thức websocket và lắng nghe client ở port 80
mqtt.begin("ws://iot.eclipse.org:80/ws");

void loop() {
    // Hàm khởi tạo MQTT, kiểm tra và xử lý các dữ liệu từ các topic, kiểm tra các thuộc tính của giao
    // thức như gói keep-a-live, gói tin QoS, id của topic...,
    mqtt.handle();
}
```

Giải thích mã nguồn

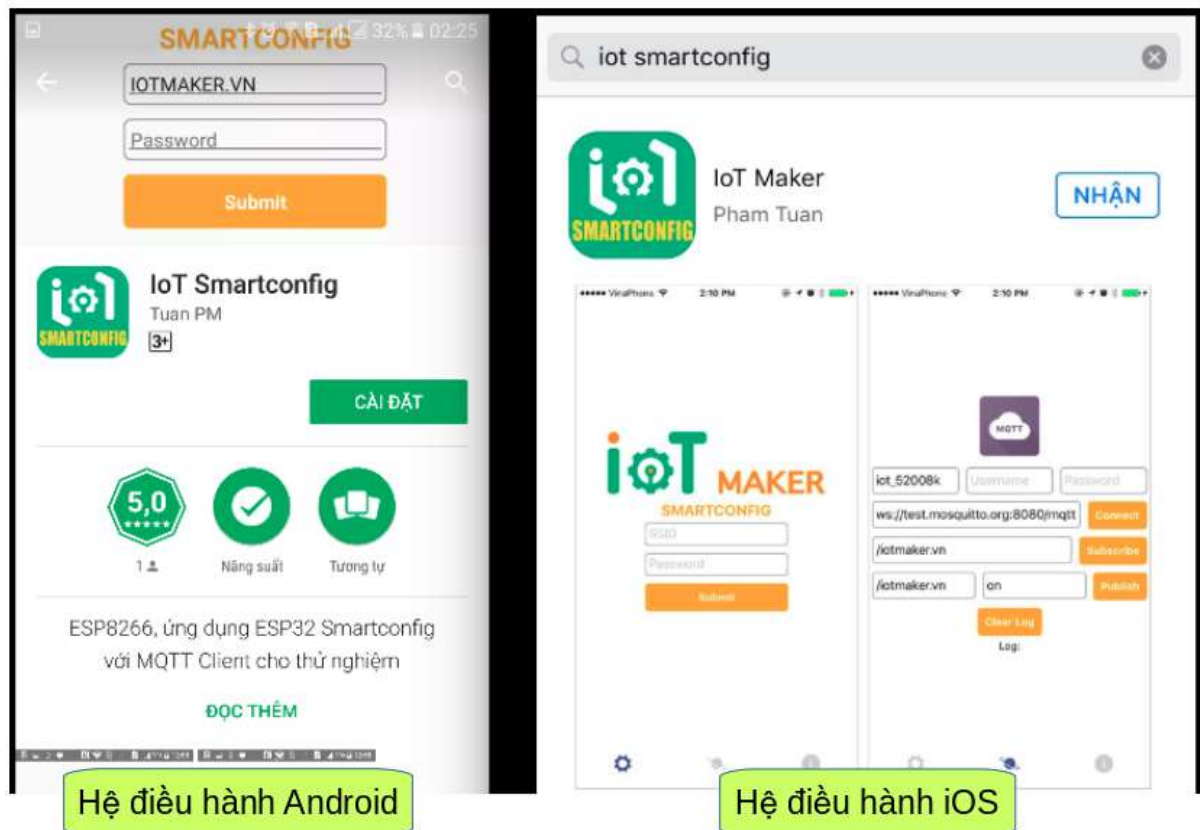
Tương tự như mã nguồn của chương trình sử dụng thư viện pubsubClient, chúng ta cũng sẽ khởi tạo ESP8266 là MQTT Client trong class MQTT của thư viện ESP8266MQTTClient. Cài đặt ESP8266 ở chế độ Station và kết nối đến network wifi. Chức năng của các hàm trong thư viện đã được giải thích ở file mã nguồn, ở hàm `mqtt.onConnect()` chúng ta sẽ subscribe topic là `esp8266/GPIO16`. Hàm `mqtt.onSubscribe()` sẽ thực hiện publish các message ở topic đã chỉ định là `Broker/app`. Hàm `mqtt.onData()` sẽ nhận, kiểm tra và xử lý dữ liệu nhận được từ topic đã subscribe. Ở đây ta sẽ dùng 1 public MQTT Broker là `iot.eclipse.org`, sử dụng phương thức Websocket là lắng nghe các MQTT Client ở port 80, đây là port mặc định khi sử dụng Websocket. Việc gửi nhận dữ liệu bằng phương thức Websocket sẽ giúp giảm băng thông và độ trễ khi truyền nhận dữ liệu thông qua giao thức MQTT. Chi tiết về Websocket chúng ta sẽ được học ở các bài học sau. Ở `loop()` chúng ta chỉ cần gọi hàm `handle()` để khởi tạo và kiểm tra các thuộc tính của giao thức cũng như xử lý, truyền và nhận dữ liệu từ các topic đã subscribe và public.



Để tìm hiểu chi tiết file cấu hình của thư viện, có thể xem tại github.com/tuanpmt/ESP8266MQTTClient/tree/master/src

- **Bước 3** : Cài đặt và sử dụng ứng dụng trên điện thoại để điều khiển led GPIO16.

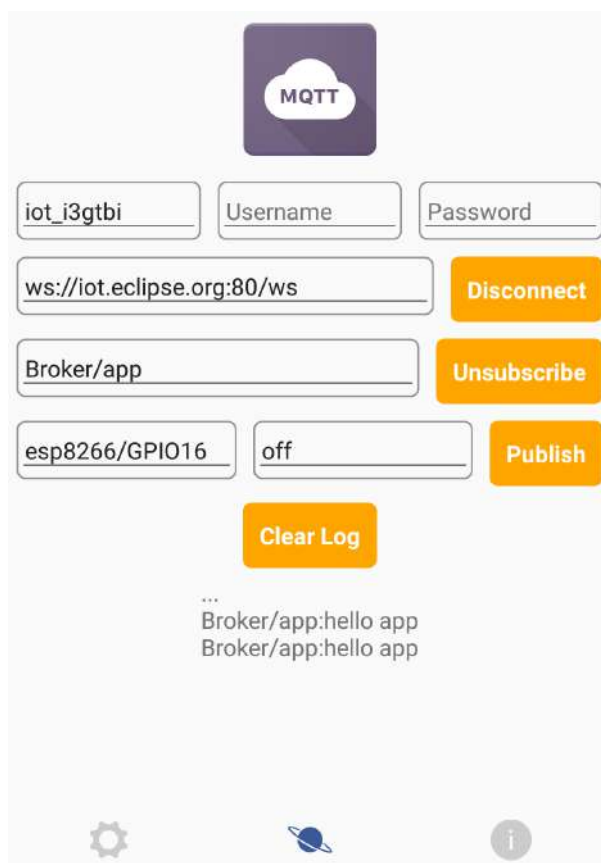
Truy cập vào [App Store](#) trên hệ điều hành iOS hoặc [CH Play](#) trên hệ điều hành Android. nhập từ khóa [IoT Smartconfig](#) và cài đặt ứng dụng [IoT Smartconfig](#) của developer Tuan PM. Hình ảnh ứng dụng hiển thị như bên dưới:



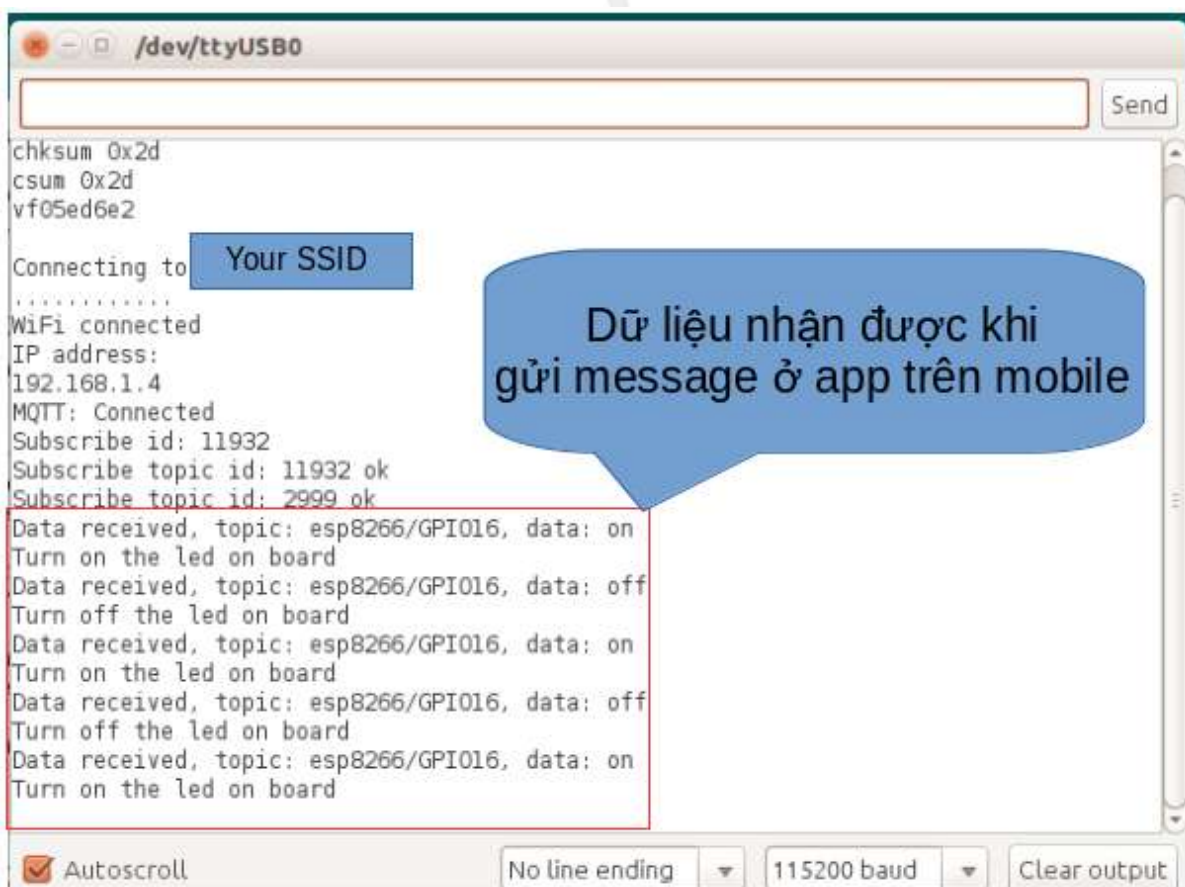
Hình 55. Hình ảnh ứng dụng IoT Smartconfig trên hệ điều hành iOS và Android

Ứng dụng này sử dụng với ESP8266 và ESP32, ngoài chức năng cơ bản là publish, subscribe của giao thức MQTT, ứng dụng còn có chức năng smartconfig để ESP8266 và ESP32 có thể dễ dàng thiết lập kết nối với các network wifi khác nhau một cách thuận tiện và nhanh chóng mà không phải nạp lại mã nguồn.

Tiếp theo, trượt ứng dụng qua phần MQTT, nhấn vào nút **connect** để kết nối đến server MQTT Broker <ws://iot.eclipse.org:80/ws>. Thực hiện subscribe topic **Broker/app** và publish message vào topic **esp8266/GPIO16**. Nếu publish message **on** vào **esp8266/GPIO16** thì led trên board ESP8266 WiFi Uno sẽ sáng, gửi **off** sẽ tắt led, đồng thời khi ESP8266 publish các message ở topic **Broker/app** thì nội dung các message sẽ được hiển thị trên ứng dụng. Kết quả hiển thị như hình bên dưới:



Hình 56. Hình ảnh subscribe topic và publish các message trên ứng dụng



Hình 57. Hình ảnh trên Serial terminal của Arduino

MQTT Broker

Ở phần trước chúng ta sử dụng các dịch vụ MQTT Broker miễn phí để thử nghiệm, tuy nhiên ta có thể sẽ phải trả phí dịch vụ với những ứng dụng lớn cần băng thông rộng và tốc độ đáp ứng nhanh, cộng với việc dữ liệu có thể bị tấn công do độ bảo mật thông tin chưa cao. Do đó, ở phần này, chúng ta sẽ tự mình xây dựng 1 MQTT Broker. Việc tự thiết lập 1 MQTT broker giúp chúng ta có thể sử dụng giao thức MQTT trên máy local mà không cần kết nối đến các dịch vụ MQTT Broker ở mạng internet. Quá trình truyền, nhận và xử lý dữ liệu diễn ra 1 cách nhanh chóng cũng như bảo mật thông tin của người dùng. Tuy nhiên, để tạo được 1 MQTT Broker với đầy đủ tính năng của giao thức MQTT đòi hỏi chúng ta phải có kiến thức tốt về giao thức MQTT cũng như các ngôn ngữ lập trình hỗ trợ cho việc xây dựng nó. Để bắt đầu, ta sẽ tạo ra 1 MQTT Broker đơn giản bằng cách dùng 1 module hỗ trợ sẵn có đó là **Mosca**.

MOSCA

Mosca là 1 trong số rất nhiều server MQTT Broker của giao thức MQTT. Có thể kể đến các server khác như HiveMQ, Apache Apollo, Mosquitto, Mongoose. Mosca có 1 số đặc điểm như sau:

- Nó là 1 Node.js Broker, được viết bằng ngôn ngữ JavaScript vì vậy để có thể xây dựng MQTT Broker, chúng ta cần Node.js để chạy. Mosca có thể nhúng vào ứng dụng của bạn nếu ứng dụng này được viết bằng Node.js
- Mosca là 1 multi-transport MQTT Broker, có nghĩa là nó hỗ trợ tất cả các chức năng publish, subscribe của các broker khác. Danh sách các publish/subscribe broker được hỗ trợ bao gồm RabbitMQ, Redis, Mosquitto, ZeroMQ. Ở phần này chúng ta sẽ tạo ra 1 MQTT Broker đơn giản dùng Mosca với sự hỗ trợ của cơ sở dữ liệu **Mongodb**

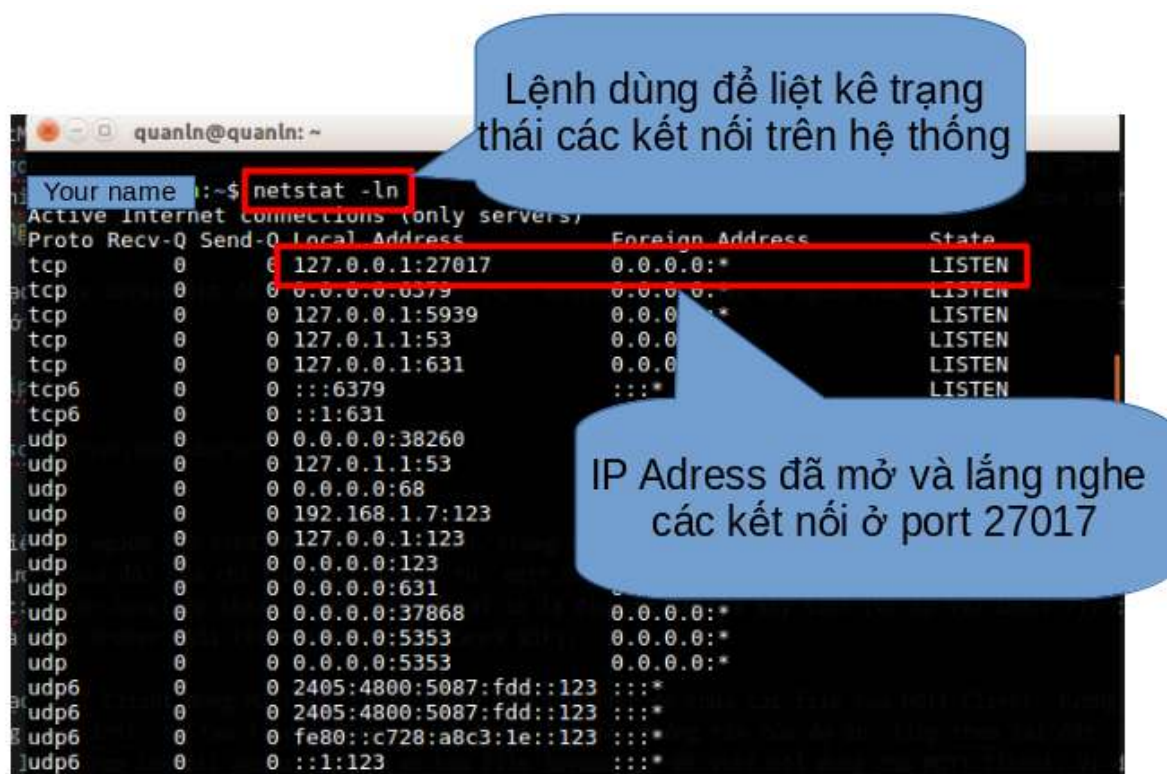
Mục tiêu

- Chúng ta sẽ tạo 1 MQTT Client là ESP8266 và 1 MQTT Client trên máy tính sử dụng MQTT.js nhằm kết nối đến MQTT Broker, subscribe topic và publish các message.
- Dùng Mosca tạo 1 MQTT Broker trên máy tính cá nhân nhằm broadcast messages (truyền bá các gói tin) đến các MQTT Client.

Các bước thực hiện

- **Bước 1** : Trước tiên, chúng ta nên tạo 1 folder để thiết lập 1 MQTT Broker trên máy local. Đi đến folder này, tạo file package.js bằng lệnh **npm init** và thiết lập các thông tin của dự án. Tiếp theo, cài đặt module mosca bằng lệnh **npm install mosca --save**. Để lắng nghe các kết nối đến từ client cũng như lưu trữ dữ liệu về thông tin kết nối và nội dung các message ta cần công cụ hỗ trợ đó là **MongoDB**, bạn cũng có thể chọn Redis, Mosquitto, RabbitMQ... và tìm hiểu thêm về điểm

mạnh, yếu của các cơ sở dữ liệu này. Để cài đặt MongoDB, chúng ta sẽ truy cập vào địa chỉ docs.mongodb.com/manual/administration/install-community/, tùy theo hệ điều hành để chọn gói cài đặt thích hợp. Sau khi cài đặt xong, chúng ta sẽ mở port 27017 (port mặc định khi dùng mongodb, chúng ta có thể điều chỉnh port ở file cấu hình của mongodb) để lắng nghe các kết nối từ client thông qua lệnh `sudo service mongod start`. Trên hệ điều hành Linux, có thể kiểm tra các kết nối trên hệ thống bằng lệnh `Netstat` như hình dưới:



Hình 58. Hình ảnh port 27017 đã mở thành công và lắng nghe các kết nối

- **Bước 2 :** Tạo file Javascript để viết mã nguồn cho MQTT Broker. Ví dụ về mã nguồn của file `serverMosca.js` được viết bên dưới:

Mã nguồn file `serverMosca.js`

```

var mosca = require('mosca'); // Khai báo biến mosca sử dụng các thuộc tính của module mosca
// Sử dụng thư viện ascoltatore nhằm hỗ trợ publish message, subscribe topic đến từ các Broker/Protocol
var ascoltatore = {

  type: 'mongo',
  url: 'mongodb://localhost:27017/mqtt', // url: địa chỉ url của mongodb, server sẽ lắng nghe các client ở địa
                                         // chỉ localhost:27017

  pubsubCollection: 'ascoltatori',      // pubsubCollection: Nơi để lưu trữ các message của mongodb
  mongo: {}                             // mongo: Cài đặt dành cho kết nối của mongo. Không sử dụng
};
var settings = {
  port: 1883, // port kết nối đến server
  backend: ascoltatore // ascoltatore sẽ được gọi và thực thi khi tạo server được tạo để thiết lập các kết nối
};

// Lệnh tạo server sử dụng mosca
var server = new mosca.Server(settings);

// Thực hiện hàm setup, in ra màn hình console nếu có sự kiện ready của server
server.on('ready', setup);
function setup() {
  console.log('Mosca server is up and running');
}

// In ra dòng chữ client connected và id của client khi có sự kiện client kết nối thành công đến server
server.on('clientConnected', function(client) {
  console.log('client connected', client.id);
});

// In ra dòng chữ client disconnected và id của client khi có sự kiện client ngắt kết nối với server
server.on('clientDisconnected', function(client) {
  console.log('client disconnected', client.id);
});

// In ra message của client gửi ở dạng string khi có sự kiện client publish 1 message
server.on('published', function(packet, client) {
  console.log('Published', packet.payload.toString());
});

```

- **Bước 3** : Viết mã nguồn cho ESP8266. Để nhanh chóng, chúng ta sẽ dùng mã nguồn của thư viện ESP8266MQTTClient đã viết ở mục trước. Sửa đổi địa chỉ của MQTT Broker từ `mqtt.begin("mqtt://iot.eclipse.org:1883");` thành `mqtt.begin("mqtt://your-local-ip:1883");` với your-local-ip là địa chỉ IP của máy tính (ví dụ 192.168.1.7), chú ý rằng ESP8266 và MQTT Broker phải kết nối chung 1 network WiFi.
- **Bước 4** : Tạo MQTT Client dùng MQTT.js. Chúng ta sẽ tạo 1 folder để chứa các file của MQTT Client. Tương tự như bước 1, dùng `npm init` để tạo file package.js và thiết lập các thông tin của dự án. Tiếp theo cài đặt module mqtt bằng lệnh `npm install mqtt --save` và tạo file Javascript để viết nội dung cho MQTT Client. Ví dụ về mã nguồn file `moscaClient.js` được trình bày bên dưới:

Mã nguồn file moscaClient.js


```
// Khai báo biến mqtt để sử dụng các thuộc tính thuộc module mqtt
var mqtt = require('mqtt');
// Tạo 1 kết nối đến địa chỉ 192.168.1.7 port 1883 thông qua giao thức MQTT
var client = mqtt.connect('mqtt://192.168.1.7:1883');
// Khi có sự kiện connect đến server, client sẽ subscribe topic MQTTlens/test/3 và
// publish 1 message "on" vào topic hello/world để ON led ở board ESP8266 WiFi Uno
client.on('connect', function () {
  client.subscribe('Broker/app');
  client.publish('esp8266/GPI016', 'on');
});
// Khi có message gửi đến client, client sẽ chuyển đổi dữ liệu từ Buffer sang dạng String và in ra màn
// hình console dữ liệu nhận được và đóng kết nối.
client.on('message', function (topic, message) {
  // message is Buffer
  console.log(message.toString());
  //client.end();
});
```

Kết quả

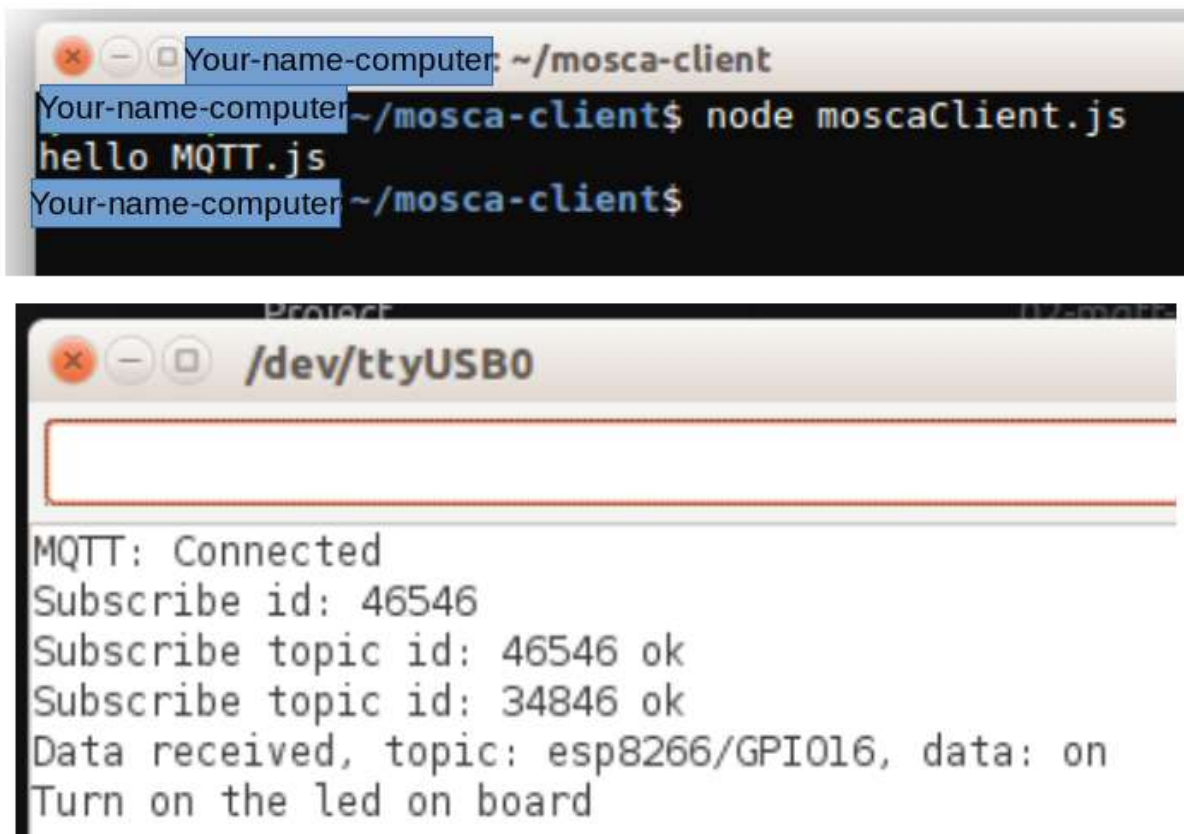
Trên terminal, đi đến thư mục chứa file `moscaServer.js` và khởi chạy server bằng lệnh `node moscaServer.js`. Server sẽ khởi động và lắng nghe các kết nối đến từ các MQTT Client. Tiếp theo, nạp chương trình trên Arduino cho ESP8266, sau đó khởi chạy MQTT Client trên máy tính bằng lệnh `node moscaClient.js`. Khi có các sự kiện kết nối, ngắt kết nối, publish 1 message hay subscribe 1 topic đến từ các client thì bên phía server đều sẽ hiển thị nội dung và thông tin. Các terminal hiển thị kết quả như hình bên dưới:

```

Your-name-computer ~/mosca-server
Your-name-computer$ cd mosca-server/
Your-name-computer$ node moscaServer.js
Mosca server is up and running ← Khởi chạy server
client connected ESP_1868452
Published ESP_1868452
Published {"clientId":"ESP_1868452","topic":"esp8266/GPI016"}
Published {"clientId":"ESP_1868452","topic":"hello/world"}
Published hello app
Published hello app ← Thông tin của ESP8266 khi kết nối
client connected mqttjs_208d6864
Published mqttjs_208d6864
Published on
Published {"clientId":"mqttjs_208d6864","topic":"Broker/app"} ← Thông tin của MQTT.js khi kết nối

```

Hình 59. Hình ảnh thông tin nhận được từ các client ở server mosca



```
Your-name-computer: ~/mosca-client
Your-name-computer:~/mosca-client$ node moscaClient.js
hello MQTT.js
Your-name-computer:~/mosca-client$

/dev/ttyUSB0

MQTT: Connected
Subscribe id: 46546
Subscribe topic id: 46546 ok
Subscribe topic id: 34846 ok
Data received, topic: esp8266/GPI016, data: on
Turn on the led on board
```

Hình 60. Hình ảnh thông tin nhận được ở client mqtt.js và client ESP8266

Một số MQTT Broker sử dụng cho sản phẩm thực tế

Mosquitto

Mosquitto là 1 MQTT Broker viết bằng ngôn ngữ lập trình C. Một số đặc điểm nổi bật của mosquitto là tốc độ truyền nhận và xử lý dữ liệu nhanh, độ ổn định cao, được sử dụng rộng rãi và phù hợp với những ứng dụng embedded. Thích hợp cho các hệ thống nhỏ chạy trên máy local như Raspberry Pi, bên cạnh đó Mosquitto cũng được hỗ trợ các giao thức TLS/SSL (các giao thức nhằm xác thực server và client, mã hóa các message để bảo mật dữ liệu).

Một số nhược điểm của mosquitto là khó thiết kế khi làm những ứng dụng lớn và ít phương thức xác thực thiết bị nên khả năng bảo mật vẫn chưa tối ưu.

EMQ

EMQ (Erlang MQTT Broker) là một MQTT Broker được viết bằng ngôn ngữ lập trình Erlang. Ưu điểm của EMQ là tính ổn định cao, thích hợp để thiết kế các hệ thống lớn do khả năng mở rộng ứng dụng dễ dàng cũng như khá dễ để cài đặt. Ngoài ra EMQ còn hỗ trợ nhiều phương thức xác thực người dùng, phát triển và cập nhật tính năng liên tục bởi cộng đồng developer. Tuy nhiên điểm yếu của MQTT broker này là khó đối với những người mới bắt đầu. Thông tin về EMQ có thể xem tại trang [emqttd-docs.readthedocs.io/en/latest/#](https://docs.readthedocs.io/en/latest/#)

Tổng kết

Từ những nội dung đã trình bày ở trên, chúng ta phần nào hiểu rõ về cách thức hoạt động của giao thức MQTT cũng như vai trò của nó trong các ứng dụng IoT. Những nội dung được trình bày ở phần này chỉ là phần cơ bản của giao thức, vì đây là giao thức quan trọng và thường sử dụng trong IoT nên chúng ta hãy dành nhiều thời gian hơn để nghiên cứu thêm về hoạt động của các gói QoS, Keep alive, cũng như các vấn đề chứng thực tài khoản, vấn đề bảo mật dữ liệu khi sử dụng MQTT.

IOTMAKER.VN

Websocket

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 sử dụng cho các ứng dụng web, lập trình viên vẫn có thể đưa chúng vào bất kì 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. 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 một cách liên tục, không đứt quãng, và không cần bất kỳ HTTP header (overhead) nào nữa.

Websocket hỗ trợ cho các trình duyệt phổ biến hiện nay như: Google Chrome, Microsoft Edge, Internet Explorer, Firefox, Safari và Opera.

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. Do đó, websocket sẽ phù hợp cho các ứng dụng real-time, người dùng sẽ không mất thời gian phải reload lại trình duyệt để cập nhật thông tin mới nhất như khi sử dụng giao thức HTTP.

Nhược điểm

Giao thức Websocket chưa được tất cả các trình duyệt đã có hiện nay hỗ trợ. Websocket cũng đòi hỏi các ứng dụng web trên server để hỗ trợ nó.

Sử dụng ESP8266 như WebSocket Server

Trong phần này, chúng ta sẽ thiết lập ứng dụng sử dụng ESP8266 như 1 WebSocket Server và Trình duyệt như là một Web Socket Client để cập nhật trạng thái nút nhấn, cũng như điều khiển đèn LED trên board thời gian thực thông qua Trình duyệt.

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 bao gồm nội dung của đoạn Javascript thiết lập kết nối WebSocket đến ESP8266 đồng thời lắng nghe các gói tin từ ESP8266 Server.
- Khi nhấn nút trên board ESP8266 sẽ gửi nội dung trạng thái nút nhấn đến Web Browser hiển thị dạng hộp kiểm (checkbox), nhấn nút là có kiểm, không nhấn nút là không có kiểm.
- Đồng thời khi nhấn hộp kiểm trên trình duyệt sẽ thay đổi trạng thái đèn LED trên board ESP8266.

Chuẩn bị

Cài đặt thư viện, xem thêm [Cài đặt thư viện Arduino](#):

- github.com/me-no-dev/ESPAsyncWebServer
- github.com/me-no-dev/ESPAsyncTCP

Giới thiệu về thư viện ESPAsyncWebServer

Thư viện ESPAsyncWebServer dùng cho việc thiết lập HTTP server và websocket server cho module ESP8266, và xử lý các sự kiện trên server-client.

Để các chương trình dùng thư viện ESPAsyncWebserver hoạt động, ta cần dùng thêm thư viện ESPAsyncTCP.

Đoạn code Javascript để tạo kết nối Web Socket

```
//Trình tự mở một websocket cơ bản:

var ws = new WebSocket('ws://domain.com:8000/'); // mở 1 websocket
ws.onopen = function() //
{
    // sự kiện khi websocket được mở thành công
};

ws.onmessage = function(evt)
{
    // sự kiện xảy ra khi client nhận dữ liệu từ server
};
ws.onclose = function() {
    // sự kiện khi websocket bị đóng
};
```

Nhúng file HTML chứa đoạn code JS vào ESP8266

```

<!DOCTYPE HTML>
<html>
<head>
  <title>ESP8266 WebSocket</title>
</head>
<body>
  <div> Webscoket status <span id="status" style="font-weight: bold;"> disconnected </span> </div>
  <div> ESP8266 Button Status <input type="checkbox" id="btn" name="btn" /> </div>
  <div> Control LED <input type="checkbox" id="led" name="led" disabled="true" /> </div>

  <script type="text/javascript">
    var button = document.getElementById('btn');
    var led = document.getElementById('led');
    var url = window.location.host; // hàm trả về url của trang hiện tại kèm theo port
    var ws = new WebSocket('ws://' + url + '/ws'); // mở 1 websocket với port 8000
    ws.onopen = function() //khi websocket được mở thì hàm này sẽ được thực hiện
    {
      document.getElementById('status').innerHTML = 'Connected';
      led.disabled = false; //khi websocket được mở, mới cho phép
    };

    ws.onmessage = function(evt) // sự kiện xảy ra khi client nhận dữ liệu từ server
    {
      if(evt.data == 'BTN_PRESSED') {
        button.checked = true;
      } else if(evt.data == 'BTN_RELEASE') {
        button.checked = false;
      }
    };

    ws.onclose = function() { // hàm này sẽ được thực hiện khi đóng websocket
      led.disabled = true;
      document.getElementById('status').innerHTML = 'Disconnected';
    };

    led.onchange = function() { // thực hiện thay đổi bật/tắt led
      var status = 'LED_OFF';
      if (led.checked) {
        status = 'LED_ON';
      }
      ws.send(status)
    }

  </script>
</body>
</html>

```

Chương trình hoàn chỉnh cho ESP8266

```

#include <ESP8266WiFi.h>
#include <ESPAsyncWebServer.h>

const char* ssid = "*****";
const char* password = "*****";
const int LED = 16;
const int BTN = 0;

// để đưa đoạn code HTML vào chương trình Arduino, cần chuyển đổi code HTML sang dạng char

const char index_html[] PROGMEM = "
<!DOCTYPE HTML>
<html>
<head>
  <title>ESP8266 WebSocket</title>

```

```

"</head>"
"<body>"
"  <div> Webscoket status <span id=\"status\" style=\"font-weight: bold;\">> disconnected </span> </div>"
"  <div> ESP8266 Button Status <input type=\"checkbox\" id=\"btn\" name=\"btn\" /> </div>"
"  <div> Control LED <input type=\"checkbox\" id=\"led\" name=\"led\" disabled=\"true\" /> </div>"
"  <script type=\"text/javascript\">"
"    var button = document.getElementById('btn');"
"    var led = document.getElementById('led');"
"    var status = document.getElementById('status');"
"    var url = window.location.host;"
"    var ws = new WebSocket('ws://' + url + '/ws');"
"    ws.onopen = function()"
"    {"
"      status.text = 'Connected';"
"      led.disabled = false;"
"    };"
"    ws.onmessage = function(evt)"
"    {"
"      if(evt.data == 'BTN_PRESSED') {"
"        button.checked = true;"
"      } else if(evt.data == 'BTN_RELEASE') {"
"        button.checked = false;"
"      }"
"    };"
"    ws.onclose = function() {"
"      led.disabled = true;"
"      status.text = 'Disconnected';"
"    };"
"    led.onchange = function() {"
"      var status = 'LED_OFF';"
"      if (led.checked) {"
"        status = 'LED_ON';"
"      }"
"      ws.send(status)"
"    }"
"  </script>"
"</body>"
"</html>";

AsyncWebServer server(8000);
AsyncWebSocket ws("/ws");

// Hàm xử lý sự kiện trên Server khi client là browser phát sự kiện
void onWsEvent(AsyncWebSocket * server, AsyncWebSocketClient * client, AwsEventType type, void * arg, uint8_t *data,
size_t len) {
  if (type == WS_EVT_DATA && len > 0) { // type: loại sự kiện mà server nhận được. Nếu sự kiện nhận được là từ
websocket thì bắt đầu xử lý
    data[len] = 0;
    String data_str = String((char*)data); // ép kiểu, đổi từ kiểu char sang String
    if (data_str == "LED_ON") {
      digitalWrite(LED, 0); // Khi client phát sự kiện "LED_ON" thì server sẽ bật LED
    } else if (data_str == "LED_OFF") {
      digitalWrite(LED, 1); // Khi client phát sự kiện "LED_OFF" thì server sẽ tắt LED
    }
  }
}

void setup()
{
  pinMode(LED, OUTPUT);
  pinMode(BTN, INPUT);
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  WiFi.mode(WIFI_AP_STA);
  WiFi.begin(ssid, password);
  if (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.printf("STA: Failed!\n");
  }
}

```



```

WiFi.disconnect(false);
delay(1000);
WiFi.begin(ssid, password);
}

ws.onEvent(onWsEvent); // gọi hàm onWsEvent
server.addHandler(ws);
server.on("/", HTTP_GET, [](AsyncWebServerRequest * request) {

    request->send_P(200, "text/html", index_html); // trả về file index.html trên giao diện browser khi browser truy cập
    vào IP của server
});
server.begin(); // khởi động server
}

void loop()
{
    static bool isPressed = false;
    if (!isPressed && digitalRead(BTN) == 0) { //Nhấn nút nhấn GPIO0
        isPressed = true;
        ws.textAll("BTN_PRESSED");
    } else if (isPressed && digitalRead(BTN)) { //Nhả nút nhấn GPIO0
        isPressed = false;
        ws.textAll("BTN_RELEASE");
    }
}
}

```

Thực hiện sau khi kiểm tra mã nguồn:

- [Chọn Board ESP8266 WiFi Uno trong Arduino IDE](#)
- [Nạp chương trình xuống board dùng Arduino IDE](#)

Kết quả

Sau khi biên dịch xong code trên Arduino, ta vào browser, truy cập vào địa chỉ IP của ESP8266 đã trả về trên Serial Monitor cùng với port đã thiết lập trên server, ở trường hợp này là 192.168.1.65:8000



Hình 61. Cửa sổ trình duyệt có thể điều khiển ESP8266 thông qua Web Socket

Video kết quả

► <https://www.youtube.com/watch?v=pN3YSLiWbHk> (YouTube video)

IOTMAKER.VN

Sử dụng ESP8266 như WebSocket Client

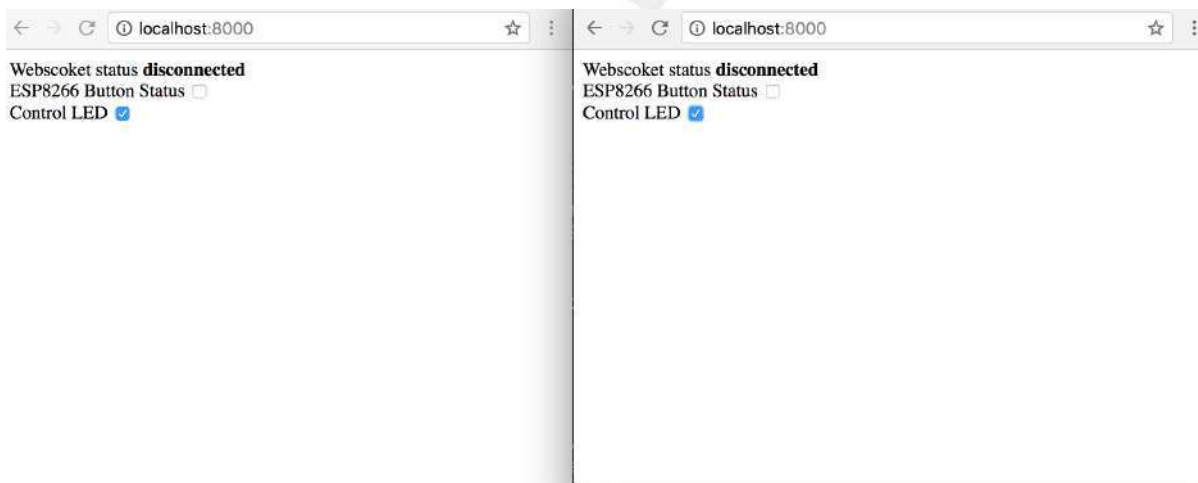
Trong một số ứng dụng khác, chúng ta có 1 Server WebSocket để thực hiện các tác vụ thời gian thực như Ứng dụng điện thoại, trình duyệt Web. Thì ESP8266 có thể kết nối trực tiếp vào các server này như 1 WebSocket Client để tiếp nhận, hoặc gửi thông tin thông qua WebSocket.

Một số dịch vụ sử dụng WebSocket điển hình như dịch vụ tin nhắn [Slack](#), dịch vụ cơ sở dữ liệu thời gian thực [Firebase](#)

Ở phần này, chúng ta sẽ sử dụng Node.js để tự xây dựng 1 Web server, vừa đóng vai trò là 1 WebSocket Server. Có những tính năng:

- Có thể cung cấp file [index.html](#) chứa các đoạn mã javascript tạo kết nối WebSocket giữa trình duyệt với Server, giống như phần [Server Nodejs](#)
- Cho phép kết nối WebSocket đến, bao gồm từ trình duyệt, hay từ ESP8266
- Server sẽ broadcast tất cả các gói tin từ bất kỳ 1 client nào gửi đến, tới tất cả các client còn lại.

Với tính năng như trên thì bạn có thể hình dung như sau: Nếu 1 cửa sổ trình duyệt có kết nối WebSocket đến Server, khi nhấn 1 nút kiểm, thì sẽ gửi về server trạng thái của nút kiểm đó. Ví dụ [LED_ON](#), server nhận được sẽ gửi dữ liệu [LED_ON](#) đến các trình duyệt còn lại (hoặc bao gồm cả ESP8266), và trình duyệt còn lại sẽ hiển thị trạng thái nút kiểm này đang bật.



Hình 62. Hai cửa sổ trình duyệt sẽ hiển thị trạng thái nút kiểm giống nhau khi click thay đổi

Javascript WebSocket Client trên trình duyệt

Với file [index.html](#) có chứa mã nguồn Javascript tạo kết nối đến WebSocket, để cùng thư mục với file [server.js](#)

index.html

```
<!DOCTYPE HTML>
<html>
<head>
  <title>ESP8266 WebSocket</title>
</head>
<body>
  <div> Webscoket status <span id="status" style="font-weight: bold;"> disconnected </span> </div>
  <div> ESP8266 Button Status <input type="checkbox" id="btn" name="btn" /> </div>
  <div> Control LED <input type="checkbox" id="led" name="led" disabled="true" /> </div>

  <script type="text/javascript">
    var button = document.getElementById('btn');
    var led = document.getElementById('led');
    var url = window.location.host; // hàm trả về url của trang hiện tại kèm theo port
    var ws = new WebSocket('ws://' + url + '/ws'); // mở 1 websocket với port 8000
    console.log('connecting...')
    ws.onopen = function() //khi websocket được mở thì hàm này sẽ được thực hiện
    {
      document.getElementById('status').innerHTML = 'Connected';
      led.disabled = false; //khi websocket được mở, mới cho phép
      console.log('connected...')
    };

    ws.onmessage = function(evt) // sự kiện xảy ra khi client nhận dữ liệu từ server
    {
      console.log(evt.data)
      if(evt.data == 'BTN_PRESSED') {
        button.checked = true;
      } else if(evt.data == 'BTN_RELEASE') {
        button.checked = false;
      } else if(evt.data == 'LED_OFF') {
        led.checked = false;
      } else if(evt.data == 'LED_ON') {
        led.checked = true;
      }
    };
    ws.onclose = function() { // hàm này sẽ được thực hiện khi đóng websocket
      led.disabled = true;
      document.getElementById('status').innerHTML = 'Connected';
    };

    led.onchange = function() { // thực hiện thay đổi bật/tắt led
      var led_status = 'LED_OFF';
      if (led.checked) {
        led_status = 'LED_ON';
      }
      ws.send(led_status)
    }
  </script>
</body>
</html>
```

Node.js WebSocket Server

Trong phần này chúng ta cần dùng thư viện WebSocket [ws github.com/websockets/ws](https://github.com/websockets/ws). Bạn có thể cài đặt bằng cách thực thi lệnh:

```
npm install ws
```

server.js

```

var fs = require('fs');
var url = require('url');
var http = require('http');
var WebSocket = require('ws');

// function gửi yêu cầu(response) từ phía server hoặc nhận yêu cầu (request) của client gửi lên
function requestHandler(request, response) {
  fs.readFile('./index.html', function(error, content) {
    response.writeHead(200, {
      'Content-Type': 'text/html'
    });
    response.end(content);
  });
}

// create http server
var server = http.createServer(requestHandler);
var ws = new WebSocket.Server({ server });
var clients = [];

function broadcast(socket, data) {
  console.log(clients.length);
  for(var i=0; i<clients.length; i++) {
    if(clients[i] != socket) {
      clients[i].send(data);
    }
  }
}

ws.on('connection', function(socket, req) {
  clients.push(socket);

  socket.on('message', function(message) {
    console.log('received: %s', message);
    broadcast(socket, message);
  });

  socket.on('close', function() {
    var index = clients.indexOf(socket);
    clients.splice(index, 1);
    console.log('disconnected');
  });
});

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

```

ESP8266 Websocket Client

github.com/Links2004/arduinoWebSockets

```
#include <Arduino.h>

#include <ESP8266WiFi.h>
#include <WebSocketsClient.h> //https://github.com/Links2004/arduinoWebSockets
WebSocketsClient webSocket;

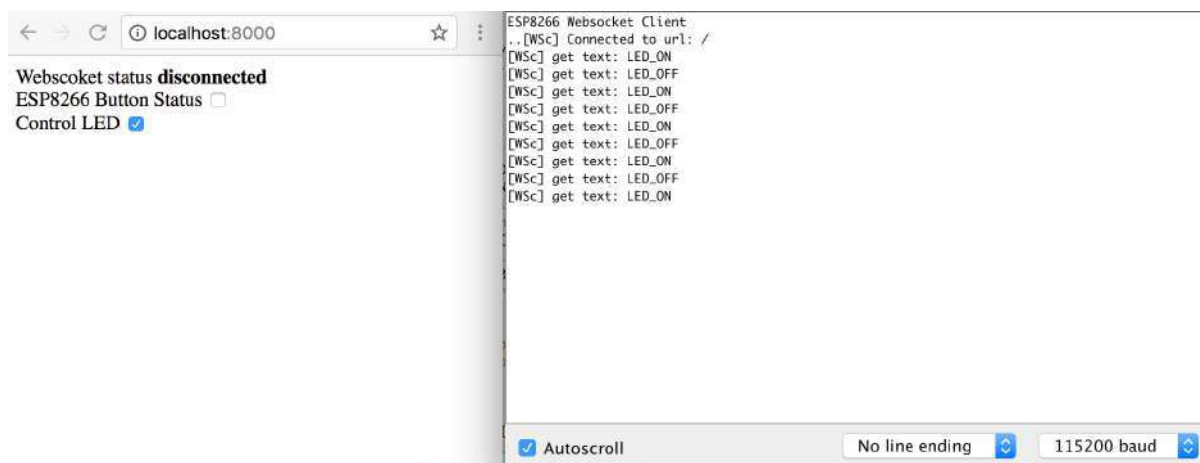
const char* ssid = "...";
const char* password = "...";
const int LED = 16;
const int BTN = 0;

void websocketEvent(WStype_t type, uint8_t * payload, size_t length) {
    switch (type) {
        case WStype_DISCONNECTED:
            Serial.printf("[WSc] Disconnected!\n");
            break;
        case WStype_CONNECTED:
            {
                Serial.printf("[WSc] Connected to url: %s\n", payload);
            }
            break;
        case WStype_TEXT:
            Serial.printf("[WSc] get text: %s\n", payload);
            if(strcmp((char*)payload, "LED_ON") == 0) {
                digitalWrite(LED, 0); // Khi client phát sự kiện "LED_ON" thì server sẽ bật LED
            } else if(strcmp((char*)payload, "LED_OFF") == 0) {
                digitalWrite(LED, 1); // Khi client phát sự kiện "LED_OFF" thì server sẽ tắt LED
            }
            break;
        case WStype_BIN:
            Serial.printf("[WSc] get binary length: %u\n", length);
            break;
    }
}

void setup() {
    pinMode(LED, OUTPUT);
    pinMode(BTN, INPUT);
    Serial.begin(115200);
    Serial.println("ESP8266 WebSocket Client");
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    webSocket.begin("192.168.0.106", 8000);
    webSocket.onEvent(websocketEvent);
}

void loop() {
    webSocket.loop();
    static bool isPressed = false;
    if (!isPressed && digitalRead(BTN) == 0) { //Nhấn nút nhấn GPIO0
        isPressed = true;
        webSocket.sendTXT("BTN_PRESSED");
    } else if (isPressed && digitalRead(BTN)) { //Nhả nút nhấn GPIO0
        isPressed = false;
        webSocket.sendTXT("BTN_RELEASE");
    }
}
```



Hình 63. Click nút kiểm sẽ thay đổi trạng thái LED trên board

Tổng kết

Việc sử dụng giao thức websocket sẽ có nhiều lợi ích cho các kết nối 2 chiều, luôn được duy trì và có độ trễ thấp.

IOTMAKER.VN

Firmware update over the air (FOTA)

Các phương pháp phát triển phần mềm và sản phẩm phổ biến hiện nay, thì xuất bản kết quả từng giai đoạn thường mang lại hiệu quả cao, sản phẩm có thể đến tay người dùng sớm, nhận được phản hồi sớm từ khách hàng, và được điều chỉnh để hợp lý hơn. Chính việc phát hành sản phẩm sớm thường sẽ ít tính năng và cần cập nhật thêm tính năng, nâng cao chất lượng sản phẩm trong tương lai.

Cập nhật ứng dụng từ xa trên các phần mềm điện thoại, máy tính đã rất phổ biến. Các phần mềm được cập nhật hàng tuần, tháng... để sửa lỗi, nâng cấp tính năng.

Đối với các sản phẩm phần cứng cũng tương tự, chúng ta nên bổ sung các tính năng cập nhật từ xa ngay từ giai đoạn phát triển sản phẩm. Ngoài việc giúp nâng cấp các tính năng trong tương lai một cách dễ dàng, thì vấn đề sửa lỗi, nâng cấp hệ thống từ xa sẽ giúp tiết kiệm được rất nhiều chi phí và nguồn lực.

Trong phần này, chúng ta sẽ tìm hiểu các phương pháp cập nhật từ xa cho ESP8266, làm sao để nạp Firmware không dây cho module, làm sao để ESP8266 có thể tự tải Firmware về, làm sao để ESP8266 có thể tự khởi động 1 HTTP Server để có giao diện Web upload firmware lên chip.

Cập nhật firmware từ xa

Cập nhật firmware OTA (Over the Air) là tiến trình tải firmware mới vào ESP8266 module thay vì sử dụng cổng Serial. Tính năng này thực sự rất hữu dụng trong nhiều trường hợp giới hạn về kết nối vật lý đến ESP Module.

OTA có thể thực hiện với:

- Arduino IDE
- Web Browser
- HTTP Server

Sử dụng OTA với tùy chọn dùng Arduino IDE trong quá trình phát triển, thử nghiệm, 2 tùy chọn còn lại phù hợp cho việc triển khai ứng dụng thực tế, cung cấp tính năng cập nhật OTA thông qua web hay sử dụng HTTP Server.

Trong tất cả các trường hợp, thì Firmware hỗ trợ OTA phải được nạp lần đầu tiên qua cổng Serial, nếu mọi thứ hoạt động trơn tru, logic ứng dụng OTA hoạt động đúng thì có thể thực hiện việc cập nhật firmware thông qua OTA.

Sẽ không có đảm bảo an ninh đối với quá trình cập nhật OTA bị hack. Nó phụ thuộc vào nhà phát triển đảm bảo việc cập nhật được phép từ nguồn hợp pháp, đáng tin cậy. Khi cập nhật hoàn tất, ESP8266 sẽ khởi động lại và thực thi code mới. Nhà phát triển phải đảm bảo ứng dụng thực trên module phải được tắt và khởi động lại 1 cách an toàn. Nội dung bên dưới cung cấp bổ sung các thông tin về an ninh, và an toàn cho tiến trình cập nhật OTA.

Bảo mật

Khi ESP8266 được phép thực thi OTA, có nghĩa nó được kết nối mạng không dây và có khả năng được cập nhật Sketch mới. Cho nên khả năng ESP8266 bị tấn công sẽ nhiều hơn và bị nạp bởi mã thực thi khác là rất cao. Để giảm khả năng bị tấn công cần xem xét bảo vệ cập nhật của bạn với một mật khẩu, cổng sử dụng cố định khác biệt, v.v...

Kiểm tra những tính năng được cung cấp bởi thư viện ArduinoOTA thường xuyên, có thể được nâng cấp khả năng bảo vệ an toàn:

```
void setPort(uint16_t port);  
void setHostname(const char* hostname);  
void setPassword(const char* password);
```

Một số chức năng bảo vệ đã được xây dựng trong và không yêu cầu bất kỳ mã hóa nào cho nhà phát triển. ArduinoOTA và espota.py sử dụng Digest-MD5 để chứng thực việc tải firmware lên. Đơn giản là đảm bảo tính toàn vẹn của firmware bằng việc tính MD5.

Hãy phân tích rủi ro cho riêng ứng dụng của bạn và tùy thuộc vào ứng dụng mà quyết định những chức năng cũng như thư viện để thực hiện. Nếu cần thiết, có thể xem xét việc thực hiện các phương thức bảo vệ khỏi bị hack, ví dụ như cập nhật OTA chỉ cho tải lên chỉ theo lịch trình cụ thể, kích hoạt OTA chỉ được người dùng nhấn nút chuyên dụng “Cập nhật”, v.v...

An toàn

Quá trình OTA tiêu tốn nguồn tài nguyên và băng thông của ESP8266 khi tải lên. Sau đó, ESP8266 được khởi động lại và một Sketch mới được thực thi. Cần phân tích và kiểm tra ảnh hưởng của quá trình này tới các chức năng cũ và sketch mới của ESP module.

Nếu ESP được đặt ở xa và điều khiển một vài thiết bị, ta nên chú ý tới hoạt động của thiết bị nếu thiết bị ngừng hoạt động đột xuất do quá trình cập nhật. Do đó, ta cần phải xác định được trạng thái làm việc an toàn của thiết bị trước quá trình cập nhật. Ví dụ, module được dùng để điều khiển hệ thống tưới nước tự động trong vườn. Nếu trong quá trình hoạt động mà hệ thống điều khiển bị tắt đột ngột và các van bị mở, thì cả vườn sẽ bị ngập nước.

Các hàm sau đây được cung cấp bởi thư viện ArduinoOTA và được dùng để xử lý ứng dụng trong quá trình cập nhật OTA hoặc để xử lý khi OTA gặp lỗi:

```
void onStart(OTA_CALLBACK(fn));  
void onEnd(OTA_CALLBACK(fn));  
void onProgress(OTA_CALLBACK_PROGRESS(fn));  
void onError(OTA_CALLBACK_ERROR (fn));
```

Yêu cầu căn bản

Bộ nhớ Flash phải có đủ dung lượng để lưu cả sketch cũ (đang vận hành trên hệ thống) và sketch mới (cập nhật OTA).

Hệ thống File và EEPROM cũng cần dung lượng để lưu trữ.

Hàm `ESP.getFreeSketchSpace()` được dùng để kiểm tra dung lượng trống cho sketch mới.

Update process - memory view

- Sketch mới sẽ được chứa trong dung lượng trống giữa sketch cũ và spiff.
- Trong lần reboot tiếp theo thì “eboot” bootloader kiểm tra các câu lệnh.
- Sketch mới sẽ được copy.
- Sketch mới khởi động.



OTA sử dụng Arduino IDE

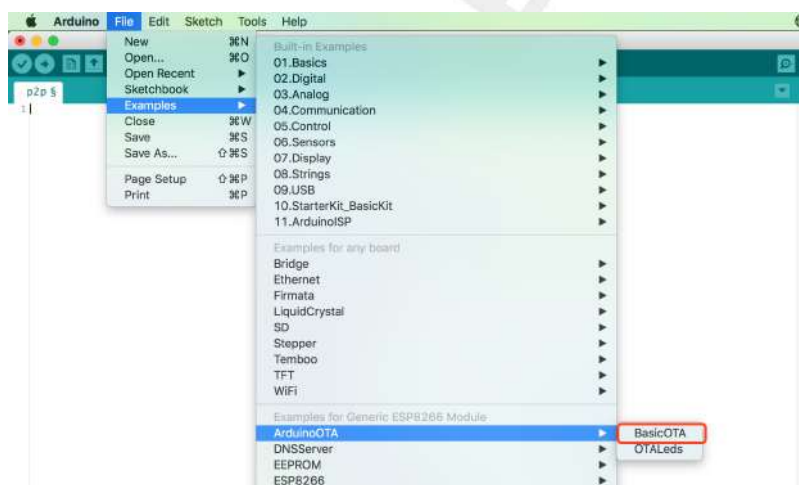
Thực hiện OTA với Arduino IDE chỉ nên thực hiện khi:

- Sử dụng nạp firmware cho module ESP8266 thông qua WiFi mà không dùng cổng Serial
- ESP8266 và máy tính chạy Arduino IDE sử dụng chung mạng WiFi nội bộ
- ESP8266 đã kết nối thành công vào mạng WiFi
- Nạp firmware đã hỗ trợ OTA thông qua cổng Serial trước đó

Trước khi bắt đầu, cần phải chắc chắn Arduino IDE đã được cài đặt phiên bản mới nhất, bao gồm gói ESP8266 cho Arduino, và Python 2.7

Bước 1: nạp firmware hỗ trợ OTA thông qua cổng Serial

Mở sketch ví dụ mẫu **BasicOTA.ino**. Vào File > Examples > ArduinoOTA.



Hình 64. Mở sketch ví dụ mẫu OTA

Cung cấp chính xác SSID và mật khẩu mạng WiFi đang dùng để ESP8266 có thể kết nối vào

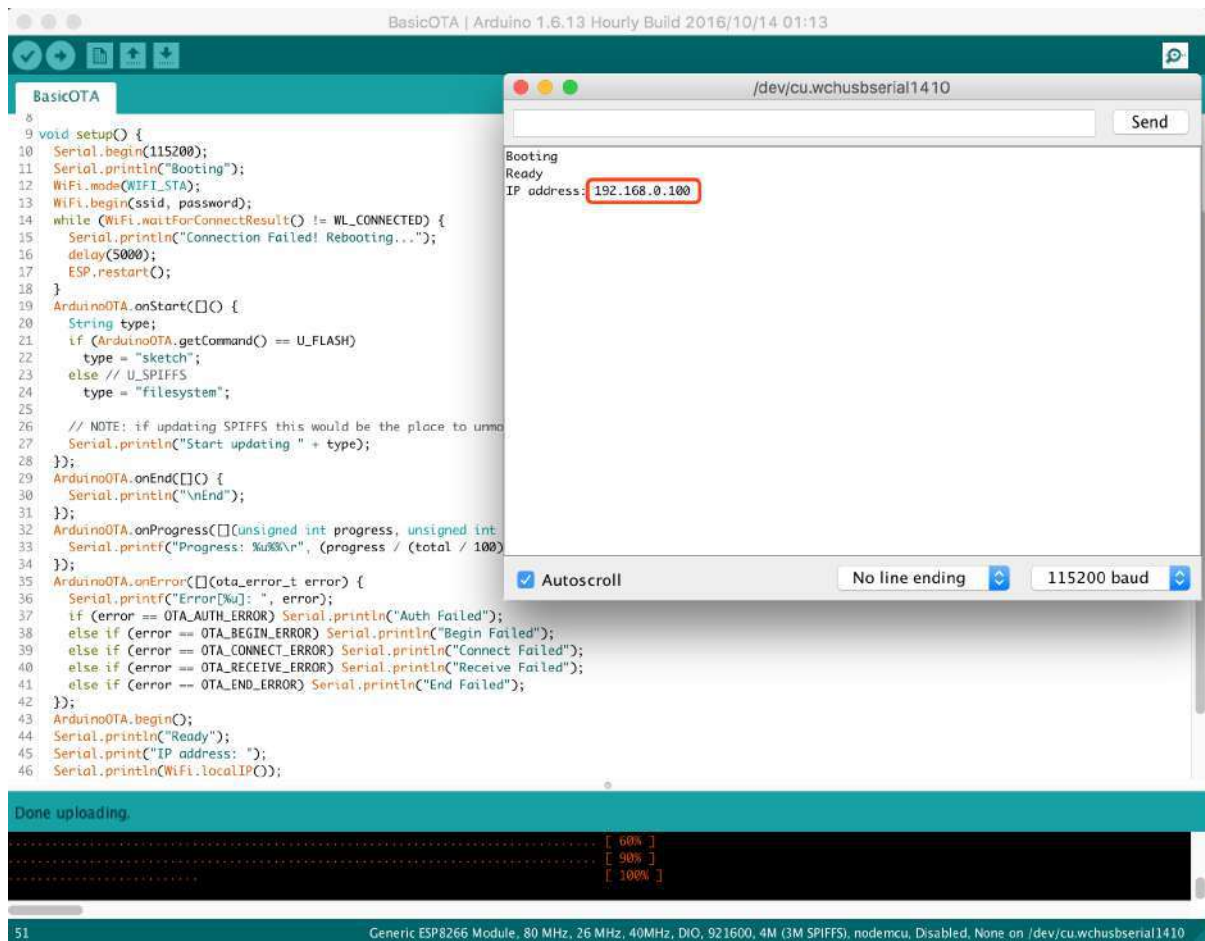
Cung cấp SSID và password

```
const char* ssid = ".....";
const char* password = ".....";
```



Tùy vào phiên bản và board ESP bạn sử dụng, bạn có thể thấy “Upload Using:” trong menu. Lựa chọn này sẽ không hoạt động và không ảnh hưởng tới lựa chọn của bạn. Chức năng này tương thích với các phiên bản OTA cũ và bị gỡ bỏ ở platform package version 2.2.0.

Upload sketch **Ctrl+U** Khi hoàn thành, mở Serial Monitor **Ctrl+Shift+M** (xem [Sử dụng Arduino IDE Serial Monitor](#)) và kiểm tra module đã truy cập được WIFI chưa



Hình 65. Kiểm tra module ESP8266 đã truy cập được mạng WIFI nội bộ chưa



ESP module nên được reset sau khi Upload xong firmware

```

#include <ESP8266WiFi.h>
#include <ESP8266mDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>

const char* ssid = "...";
const char* password = "...";

void setup() {
  Serial.begin(115200);
  Serial.println("Booting");
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("Connection Failed! Rebooting...");
    delay(5000);
    ESP.restart();
  }
  ArduinoOTA.onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH)
      type = "sketch";
    else // U_SPIFFS
      type = "filesystem";

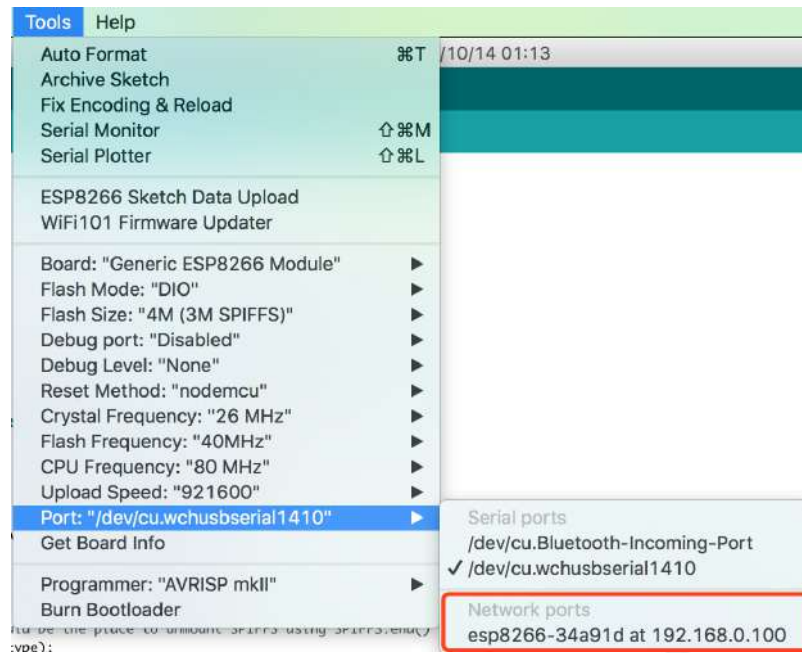
    // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end()
    Serial.println("Start updating " + type);
  });
  ArduinoOTA.onEnd([]() {
    Serial.println("\nEnd");
  });
  ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
  });
  ArduinoOTA.onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
    else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
    else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
    else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
    else if (error == OTA_END_ERROR) Serial.println("End Failed");
  });
  ArduinoOTA.setPassword((const char *)"ota-pass");
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  ArduinoOTA.handle();
}

```

Bước 2: Lựa chọn cổng nạp thông qua OTA

Khi module kết nối tới mạng WiFi thành công, sau vài giây, cổng esp8266-ota sẽ xuất hiện trên Arduino IDE. Lúc này bạn hoàn toàn có thể bỏ kết nối Serial từ board mạch đến máy tính. Arduino IDE có thể nạp firmware mới thông qua WiFi. Chọn port với địa chỉ IP hiện trên cửa sổ Serial Monitor ở bước trước.



Hình 66. Chọn cổng nạp từ Network

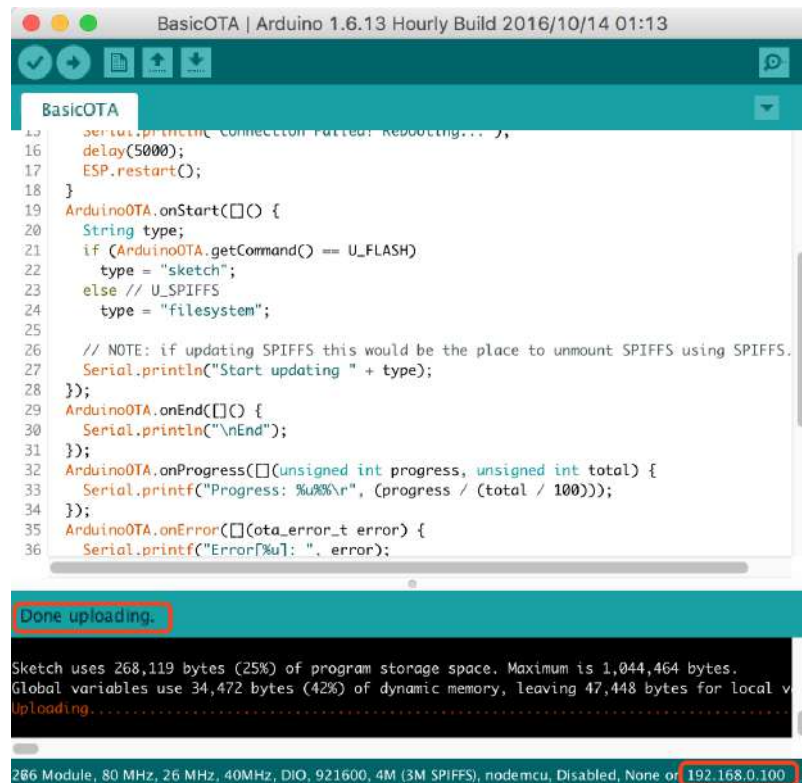


Nếu cổng OTA không hiện lên, bạn cần thoát Arduino IDE, và mở lại. Kiểm tra lại port OTA. Nếu vẫn tiếp tục không hiển thị cổng OTA, kiểm tra tường lửa của máy và các cài đặt trên router.

Bước 3: Sửa firmware mới và nạp lại thông qua WiFi

Sau khi đã chọn đúng cổng nạp OTA, bạn hoàn toàn có thể sửa lại firmware mới và nạp thông qua WiFi, tuy nhiên cần lưu ý như sau:

- Firmware mới phải đảm bảo kết nối đến WiFi không bị mất (ví dụ cấp sai mật khẩu)
- Firmware mới phải có các hàm khởi tạo và xử lý OTA như [Bước 1: nạp firmware hỗ trợ OTA thông qua cổng Serial](#)



Hình 67. Nạp firmware thành công thông qua OTA

Sử dụng mật khẩu

Bảo vệ quá trình upload OTA với password là một quá trình khá đơn giản. Những việc bạn cần làm là bổ sung đoạn mã nguồn:

```
ArduinoOTA.setPassword((const char *)"your-password");
```

Sau đó upload lại sketch một lần nữa (dùng OTA). Sau khi biên dịch và upload xong, cửa sổ sẽ hiện lên yêu cầu nhập password:



Hình 68. Password cho OTA

Nhập password, nếu đúng, kết quả là thông báo **Authenticating...OK** và quá trình nạp diễn ra bình thường.

Các lần nạp sau Arduino IDE sẽ nhớ mật khẩu và không hỏi lại, trừ khi bạn thay đổi mật khẩu OTA, và các bước xác thực không thành công, Arduino IDE sẽ hỏi lại bạn.

Cần lưu ý là password có thể dễ dàng thấy được, nếu IDE không được đóng sau lần upload cuối

cùng. Việc này có thể được thực hiện bằng cách cho phép **Show verbose output during: upload** trong **File > Preferences** và upload lên module.

Những sự cố thường gặp

Nếu việc cập nhật OTA thất bại, bước đầu tiên bạn cần làm là kiểm tra phần báo lỗi hiện trên cửa sổ Log của Arduino IDE. Nếu việc này không giúp được bạn, hãy upload lại khi kiểm tra các thông tin của ESP hiện trên serial port.

Các nguyên nhân phổ biến gây lỗi OTA như sau:

- Không đủ dung lượng bộ nhớ trên chip (ví dụ như ESP01 với 512KB bộ nhớ flash không đủ cho OTA).
- Khu vực dữ liệu cho SPIFFS quá nhiều, không còn đủ để chứa firmware, trong trường hợp bạn có 4MB Flash thì trường hợp này không xảy ra.
- Khu vực bộ nhớ chứa firmware quá ít, tối thiểu là 512KB
- Không reset module ESP sau lần upload đầu dùng Serial Port.

Cập nhật Firmware dùng Web Browser

Khi thực hiện cập nhật firmware dùng Web Browser, ESP8266 sẽ khởi động 1 HTTP Server, với 1 form upload. Khi truy cập đúng địa chỉ của nó, bạn sẽ được cung cấp 1 giao diện để chọn file binary, và upload lên Chip. Việc này hữu dụng khi không dùng Arduino IDE cho việc cập nhật, sử dụng luôn trình duyệt sẵn có. Hoặc tích hợp vào 1 ứng dụng mà bạn có thể muốn cập nhật nó trong tương.

Cập nhật với web browser được thực hiện bằng thư viện `ESP8266HTTPUpdateServer` cùng với 2 thư viện khác `ESP8266WebServer` và `ESP8266mDNS` cho việc nhận diện ESP8266 trong mạng nội bộ.

Thực hiện

Mở ví dụ: `File > Examples > ESP8266HTTPUpdateServer > WebUpdater`

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <ESP8266HTTPUpdateServer.h>

const char* host = "esp8266-webupdate";
const char* ssid = "...";
const char* password = "...";

ESP8266WebServer httpServer(80);
ESP8266HTTPUpdateServer httpUpdater;

void setup(void){
    Serial.begin(115200);
    Serial.println();
    Serial.println("Booting Sketch...");
    WiFi.mode(WIFI_AP_STA);
    WiFi.begin(ssid, password);

    while(WiFi.waitForConnectResult() != WL_CONNECTED){
        WiFi.begin(ssid, password);
        Serial.println("WiFi failed, retrying.");
    }

    MDNS.begin(host);
    httpUpdater.setup(&httpServer);
    httpServer.begin();

    MDNS.addService("http", "tcp", 80);
    Serial.printf("HTTPUpdateServer ready! Open http://%s.local/update in your browser\n", host);
}

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

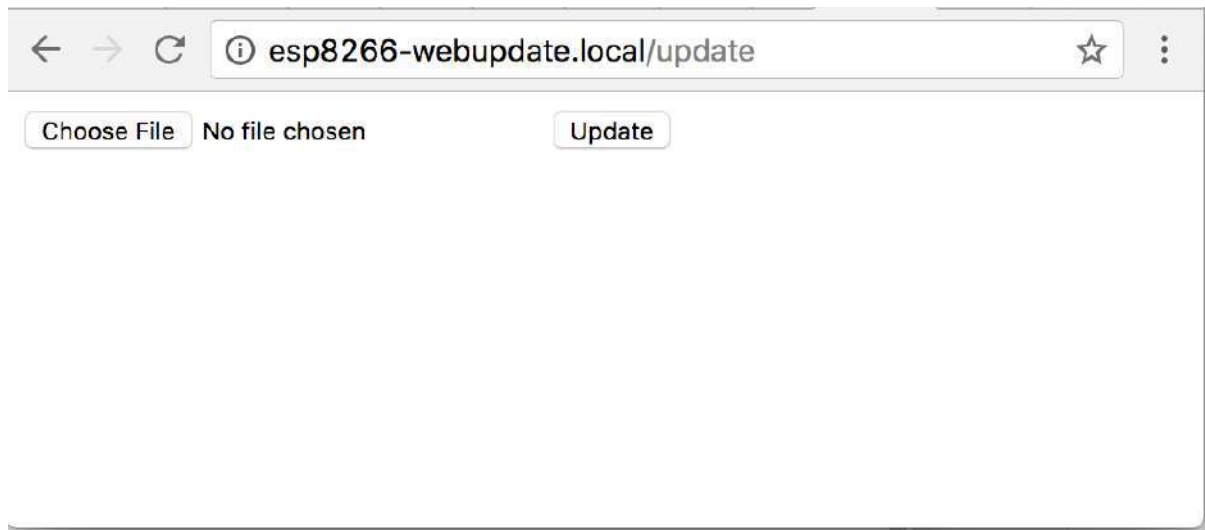
Cung cấp đúng SSID và mật khẩu mạng WiFi máy tính bạn đang dùng, nạp Firmware `WebUpdater` vào ESP8266 [Chọn Board ESP8266 WiFi Uno trong Arduino IDE](#) và [Nạp chương trình xuống board dùng Arduino IDE](#)



Lưu ý là máy tính phải sử dụng mạng WiFi cùng với ESP8266

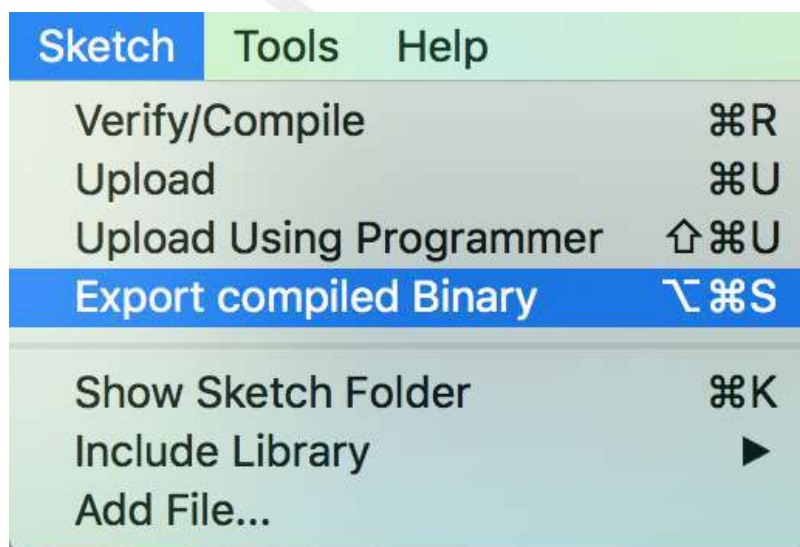
Khi bạn không thể truy cập vào module ESP8266 theo công Serial, thì để nhận diện được địa chỉ IP của module trong mạng LAN, bạn cần chạy dịch vụ mDNS trên máy tính. Dịch vụ này sẵn có trong MacOS, tuy nhiên, với Linux thì bạn cần cài đặt Avahi: avahi.org/, còn Windows thì Bonjour: support.apple.com/downloads/bonjour_for_windows

Với dịch vụ mDNS chạy trên máy tính, bạn dễ dàng truy cập vào ESP8266 theo đường dẫn esp8266-webupdate.local/update



Hình 69. Giao diện Web cập nhật firmware

Bằng cách chọn file và nhấn cập nhật, ESP8266 sẽ tiến hành cập nhật firmware mới do bạn gửi lên. File này có thể được xuất ra bằng cách **Sketch > Export compiled Binary**, và file .bin sẽ nằm trong thư mục của Sketch

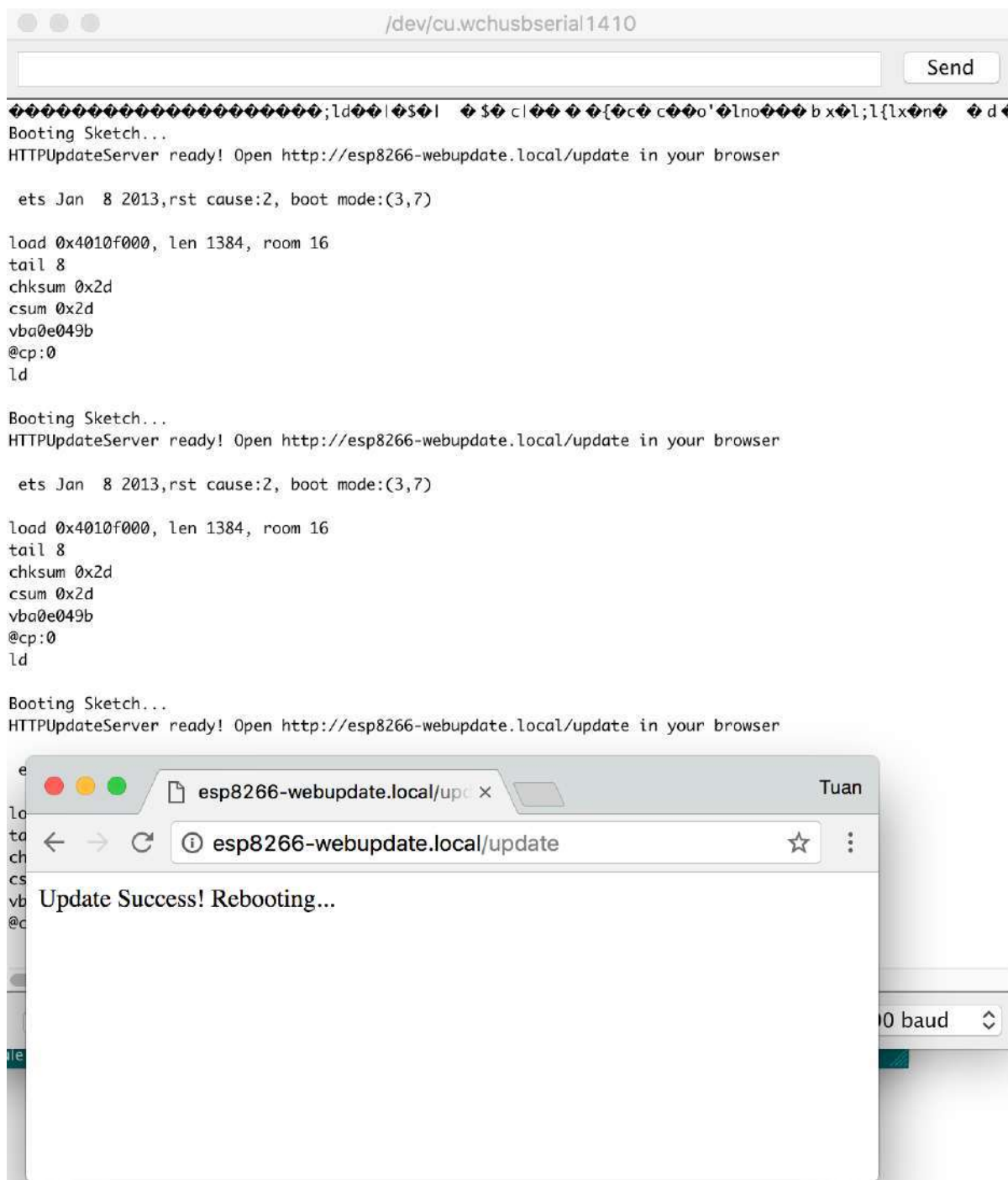


Hình 70. Xuất file Binary



Khi đã nhập esp8266-webupdate.local/update mà không thực hiện được, hãy thay esp8266-webupdate với địa chỉ IP của module (bạn có thể xem trong modem/router, hay Serial Terminal). Ví dụ, nếu IP của module là **192.168.1.100** thì URL phải là 192.168.1.100/update. Phương pháp này hữu hiệu trong trường hợp host software cài đặt ở bước 1 không hoạt động.

Nếu các bước diễn ra thành công tốt đẹp, thì trên trình duyệt và cửa sổ Serial Terminal (nếu mở) như hình



Hình 71. Thực hiện thành công cập nhật Firmware sử dụng WebUpdater

Bảo mật

Nếu bổ sung WebUpdater vào sản phẩm của mình, dĩ nhiên bạn sẽ không muốn người khác tự do đưa vào thiết bị 1 firmware khác. Hãy sử dụng hàm `httpUpdater.setup(&httpServer, update_path, update_username, update_password);` với các thông số username, password mà bắt buộc bạn phải nhập đúng mới được phép upload firmware mới.

Mở ví dụ : [File](#) > [Examples](#) > [ESP8266HTTPUpdateServer](#) > [SecureWebUpdater](#) để xem chi tiết

IOTMAKER.VN

HTTP Server

Với 2 phương pháp trước, bạn dễ dàng cập nhật Firmware thông qua mạng WiFi nội bộ. Tuy nhiên, khi triển khai ứng dụng thực tế, chúng ta sẽ cần cập nhật Firmware từ xa thông qua Internet, và cần 1 Server để lưu trữ firmware, quản lý các phiên bản.

Một kịch bản phổ thông thường được làm nhất đó là:

- Khi ESP8266 khởi động (khoảng sau 1 khoảng thời gian - ví dụ như 1 ngày), nó sẽ kết nối đến Server, cung cấp thông tin phiên bản hiện có của nó
- Server khi nhận thấy phiên bản hiện tại cần phải được nâng cấp, nó sẽ trả về firmware mới
- Nếu phiên bản hiện tại của ESP8266 không cần phải cập nhật, nó sẽ trả về mã 304.

Để thực hiện được điều này, chúng ta cần thực hiện trên cả ESP8266 và trên Server side. Thử nghiệm trong mục này, chúng ta sẽ dùng Node.js làm server. Bạn hoàn toàn có thể thực thi đoạn code Server này và gán cho nó domain để có thể truy cập từ bất kỳ đâu.

ESP8266 ESPhttpUpdate

Bằng cách thực thi `ESPhttpUpdate.update("your-domain.com", 8000, "/firmware.bin");`, ESP8266 sẽ tự động kết nối tới server ở địa chỉ `your-domain.com:8000/firmware.bin` để tải phiên bản firmware mới về. Mã HTTP Status Code:

- (Mã) 200: Nếu tồn tại firmware mới, và nội dung file sẽ được gửi kèm sau đó
- (Mã) 304: Thông báo là không có bản update mới.

Đoạn mã có thể dễ dàng tìm thấy ở [File > Examples > ESPhttpUpdate > httpUpdate](#)

Bạn cần cung cấp SSID, mật khẩu WiFi chính xác, thực hiện [Chọn Board ESP8266 WiFi Uno trong Arduino IDE](#) và [Nạp chương trình xuống board dùng Arduino IDE](#)

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <ESP8266httpUpdate.h>

const char* ssid = "...";
const char* password = "...";
const char *currentVersion = "1.0"; ①
const char *serverUrl = "http://192.168.0.106:8000/firmware.bin"; ②
void setup() {

    Serial.begin(115200);
    Serial.println();
    Serial.println();
    Serial.print("ESP8266 http update, current version: ");
    Serial.println(currentVersion);

    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    t_httpUpdate_return ret = ESPhttpUpdate.update(serverUrl, currentVersion);
    switch (ret) {
        case HTTP_UPDATE_FAILED:
            Serial.printf("HTTP_UPDATE_FAILED Error (%d): %s", ESPhttpUpdate.getLastError(),
                ESPhttpUpdate.getLastErrorMessage().c_str());
            break;

        case HTTP_UPDATE_NO_UPDATES:
            Serial.println("HTTP_UPDATE_NO_UPDATES");
            break;
        case HTTP_UPDATE_OK:
            Serial.println("HTTP_UPDATE_OK");
            break;
    }
}

void loop() {
}
```

- ① Phiên bản firmware của bạn, giả sử bạn thay đổi là 2.0 và đặt lên server, sau đó bạn đổi lại 1.0 và nạp vào board
- ② Đường dẫn đến firmware của bạn, là địa chỉ web, ip, hay domain

Node.js OTA Server

Khi ESP8266 kết nối tới Web Server, thì nó sẽ cung cấp các thông tin Header để Server căn cứ vào đó đánh giá firmware có cần phải cập nhật hay không. Ví dụ về các header

```
[HTTP_USER_AGENT] => ESP8266-http-Update
[HTTP_X_ESP8266_STA_MAC] => 18:FE:AA:AA:AA:AA
[HTTP_X_ESP8266_AP_MAC] => 1A:FE:AA:AA:AA:AA
[HTTP_X_ESP8266_FREE_SPACE] => 671744
[HTTP_X_ESP8266_SKETCH_SIZE] => 373940
[HTTP_X_ESP8266_SKETCH_MD5] => a56f8ef78a0bebd812f62067daf1408a
[HTTP_X_ESP8266_CHIP_SIZE] => 4194304
[HTTP_X_ESP8266_SDK_VERSION] => 1.3.0
[HTTP_X_ESP8266_VERSION] => 1.0
```

Dựa trên kiến thức phần [Server Nodejs](#), chúng ta xây dựng 1 OTA Server dùng Node.js như sau

server.js

```
var fs = require('fs');
var url = require('url');
var http = require('http');
var querystring = require('querystring');
var crypto = require('crypto');
// function gửi yêu cầu(response) từ phía server hoặc nhận yêu cầu (request) của client gửi lên
function requestHandler(request, response) {

    // Giải sử địa chỉ yêu cầu firmware http://192.168.1.7:8000/firmware.bin
    var uriData = url.parse(request.url);
    var pathname = uriData.pathname;           // /firmware.bin

    if (pathname == '/firmware.bin') {
        var ver = request.headers['x-esp8266-version'];
        console.log('Client request update, version ', ver);
        if(ver == '1.0') {
            console.log('Send firmware 2.0 to client');
            fs.readFile('./esp8266-firmware-2.0.bin', function(error, content) {
                response.writeHead(200, {
                    'Content-Type': 'binary/octet-stream',
                    'Content-Length': Buffer.byteLength(content),
                    'x-MD5': crypto.createHash('md5').update(content).digest("hex")
                });
                response.end(content);
            });
        } else {
            response.statusCode = 304;
            response.end();
        }
    }
}

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

Thực thi `node server.js` để khởi động Server



Khi bạn làm việc với các ngôn ngữ lập trình khác, luôn phải đảm bảo khi gửi về client cần có đầy đủ thông tin header **Content-Length** và **x-MD5**



Bạn cần file **esp8266-firmware-2.0.bin** ở mục **ESP8266 ESPhttpUpdate** biên dịch với **currentVersion = "2.0"** đặt cùng thư mục với file **server.js**

Nếu bạn thực hiện đầy đủ các bước như trên, kết quả thực thi sẽ như hình

```
ota-server — node server.js — 80x34
Server listening on port 8000
Client request update, version 1.0
Send firmware 2.0 to client
Client request update, version 2.0
...
data_request = ESPhttpUpdate.update(serverURL, currentVersion){
  ...
}
HTTP_UPDATE_FAILED
...
HTTP_UPDATE_NO_UPDATES
...
HTTP_UPDATE_OK
...

/dev/cu.wchusbserial1410
ESP8266 http update, current version: 1.0
ets Jan 8 2013, rst cause:2, boot mode:(3,7)
Load 0x4010f000, len 1384, room 16
tail 8
chksum 0x2d
csum 0x2d
vba0e049b
esp:0
ld
ESP8266 http update, current version: 2.0
.....HTTP_UPDATE_NO_UPDATES
```

Hình 72. Kết quả thực hiện OTA sử dụng HTTP Server

Cheatsheet

Phần này cung cấp thông tin cho việc tra cứu nhanh các hàm có thể sử dụng với Arduino, ESP8266 và ngôn ngữ lập trình C. Có rất nhiều thư viện cho Arduino cung cấp các tính năng hữu ích thông qua các API cho phép người sử dụng gọi, nội dung tóm lược ở đây chỉ đề cập tới các thư viện thường xuyên được sử dụng nhất. Nếu bạn sử dụng các thư viện khác, hoàn toàn có thể tra cứu dễ dàng trong tài liệu sử dụng của thư viện đó. Thông thường các thư viện được phát hành nguồn mở trên [Github](#) sẽ có file **README.md** cung cấp đầy đủ các thông tin.

IOTMAKER.VN

Arduino - ESP8266 Cheatsheet

```

/* CẤU TRÚC CƠ BẢN CỦA 1 SKETCH */
void setup() {
  /*
   Hàm được gọi khi bắt đầu sketch. Dùng để khởi tạo
   biến, cấu hình chân, khởi tạo thư viện...
   Code trong setup chỉ chạy 1 lần (khi khởi động hoặc
   reset board)
  */
}
void loop() {
  // Nội dung trong loop() lặp lại liên tục
}

/* LỆNH RẼ NHÁNH */
if (x < 5)      // thực thi code nếu x<5
{ code }
else if (x > 10) // thực thi code nếu x>10
{ code }
else { code }  // thực thi code các trường hợp còn lại

switch (var) { // thực thi case có giá trị var
case 1:
...
break;
case 2:
...
break;
default:
...
}

/* CÁC KIỂU VÒNG LẶP */
/* Thực hiện code nếu x<5 */
while (x < 5) { code };
/* Thực hiện code, so sánh, nếu x<0 tiếp tục thực hiện
code */
do { code } while(x < 0);
/* Khởi tạo i, thực hiện code và tăng i nếu i < 10 */
for (int i = 0; i < 10; i++) { code };
/* Thoát ra vòng lặp (for, while, do-while) ngay lập
tức */
break;
/* Đi đến chu kì lặp tiếp theo của vòng lặp hiện tại */
continue;

/* CÁC ĐỊNH NGHĨA VỀ HÀM*/
<ret. type> <name>(<params>) { ... }
int func_name(int x) { return x*2; }
return x; // x phải trùng khớp với kiểu trả về của hàm
return;   // loại return dành cho hàm void

/* INCLUDE */
/* include thư viện chuẩn */
#include <stdio.h>
/* include thư viện tạo bởi người dùng */
#include "your-library.h"

```

```

/* DỮ LIỆU KIỂU CHUỖI */
/* Chuỗi bao gồm kí tự kết thúc chuỗi \0 (null) */
char str1[8] = {'A','r','d','u','i','n','o','\0'};
/* Trình biên dịch tự động thêm kí tự \0 vào cuối
chuỗi */
char str2[8] = {'A','r','d','u','i','n','o'};
/* Khai báo chuỗi ,không khai báo số phần tử và gán giá
trị chuỗi*/
char str3[] = "Arduino";
/* Khai báo và gán giá trị cho chuỗi */
char str4[8] = "Arduino";

/* DỮ LIỆU KIỂU MẢNG */
/* Khai báo mảng kiểu int có 6 phần tử và gán giá trị
cho mỗi phần tử */
int myPins[] = {2, 4, 8, 3, 6};
/* Mảng kiểu int 6 phần tử và không gán giá trị */
int myInts[6];
myInts[0] = 42; // Gán giá trị 42 cho phần tử đầu tiên
myInts[6] = 12; // LỖI ! chỉ số của mảng chỉ từ 0 đến 5

/*Qualifiers*/
static // Không thay đổi giá trị ở các lần gọi
volatile // In RAM (Thường dùng trong ngắt)
const // Không đổi (chỉ đọc)
PROGMEM /* Cho phép lưu trữ dữ liệu trong bộ nhớ
FLASH thay vì SRAM */

/* CÁC TOÁN TỬ, PHÉP TOÁN THƯỜNG DÙNG */
/* Các toán tử thường dùng */
= toán tử bằng
+ toán tử cộng
- toán tử trừ
* toán tử nhân
/ toán tử chia lấy phần nguyên
% toán tử chia lấy phần dư
== phép so sánh bằng
!= phép so sánh không không bằng (khác)
< phép so sánh nhỏ hơn
> phép so sánh lớn hơn
<= phép so sánh nhỏ hơn hoặc bằng
>= phép so sánh lớn hơn hoặc bằng
&& phép toán logic (AND)
|| phép toán logic (OR)
! phép toán logic (NOT)

/* Các toán tử hợp nhất */
++ tăng 1 đơn vị
-- giảm 1 đơn vị
+= phép toán cộng và gán giá trị
ex: x = 5; x+= 1; //x = 6
-= phép toán trừ và gán giá trị
ex: x = 5; x-= 1; //x = 4
*= phép toán nhân và gán giá trị
ex: x = 5; x*= 3; //x = 15
/= phép toán chia lấy phần nguyên và gán giá trị
ex: x = 6; x/= 2; //x = 3

```

```

ex: x = 6; x/= 2; //x = 3
&= phép toán logic AND và gán giá trị
ex: x = 0b1010; x&= 0110; //x =0b0010
|= phép toán logic OR và gán giá trị
ex: x = 0b1010; x|= 0110; //x =0b1110

/* Các toán tử trên bit */
& and ^ xor
<< dịch trái >> dịch phải
| or ~ not

/* Thực thi với con trỏ */
&reference: // lấy địa chỉ của biến mà con trỏ trỏ tới
*dereference:// lấy giá trị của biến mà con trỏ trỏ tới

/* HẰNG SỐ VÀ KIỂU DỮ LIỆU */
123 Số thập phân
0b0111 Số nhị phân
0173 Số Octal - base 8
0x7B Số thập lục phân base 16
123U Số nguyên không dấu
123L Số nguyên có dấu 4 bytes
123UL Số nguyên không dấu 4bytes
123.0 Số thực
1.23e6 Số thực dùng cơ số mũ ex: 1.23*10^3 = 1230

/*PHẠM VI CỦA KIỂU DỮ LIỆU*/
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

/* KHAI BÁO BIẾN */
int a;
int a = 0b01111011, 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
 */
/* Thiết lập cấu hình chân */
pinMode(pin,[INPUT, OUTPUT, INPUT_PULLUP])
/* Đọc giá trị của pin_6 và gán cho a */
int a = digitalRead(pin_6);
/* Cài đặt mức HIGH/LOW cho pin_5 */
digitalWrite(pin_5, [HIGH, LOW])
/* Đọc giá trị của pin và gán cho a, pin từ A0-->A5 */
int a = analogRead(pin)

/* PWM ngõ ra - pins 3 5 6 9 10 11
 * ESP8266: pin 0..16, range = 0..1023, 1KHz default
 */
/* Đặt giá trị PWM cho pin */

```

```

analogWrite(pin, value)
/* ESP8266: thay đổi RANGE PWM output */
analogWriteRange(new_range)
/* ESP8266: Tần số PWM output */
analogWriteFreq(new_frequency)

/* ADVANCED I/O */
/* Tạo sóng vuông tần số freq_Hz với duty cycle=50% */
tone(pin, freq_Hz)
/* Tạo sóng vuông tần số freq_Hz, duration mili giây */
tone(pin, freq_Hz, duration_ms)
/* Ngừng việc tạo sóng vuông khi dùng tone() */
noTone(pin)
/* Dịch 1 byte, mỗi lần dịch 1 bit, dịch từ bit cao */
shiftOut(dataPin, clockPin,[MSBFIRST, LSBFIRST], value)
/* Trả về (ms) của xung HIGH/LOW trên chân pin */
unsigned long pulseIn(pin,[HIGH, LOW])

/* CHỨC NĂNG NGẮT */
/* Thiết lập chức năng ngắt ở các chân digital */
attachInterrupt(interrupt, func, mode)
/*
interrupt: số ngắt (thường là chân sử dụng chức năng ngắt)
func : hàm sẽ được gọi khi ngắt xảy ra (lưu ý : hàm không có tham số đầu vào cũng như kiểu trả về)
mode : gồm các chế độ LOW,CHANGE, RISING, FALLING. Ngắt sẽ được kích hoạt khi chân ngắt ở mode tương ứng
*/
/* Vô hiệu hóa ngắt interrupt */
detachInterrupt(interrupt)
/* Vô hiệu hóa tất cả các ngắt */
noInterrupts()
/* Cho phép tái ngắt sau khi dùng noInterrupts() */
interrupts()

/*****
 *          THƯ VIỆN PHỔ BIẾN
 *****/

/*****
 *          Serial
 *          *Thư viện giao tiếp với PC hoặc thông qua RX/TX*
 *****/
/* Thiết lập giao tiếp serial-UART với tốc độ speed */
begin(long speed)
/* Vô hiệu hóa giao tiếp serial */
end()
/* Trả về số bytes có sẵn để đọc */
int available()
/* Đọc dữ liệu đến từ serial (trả về byte đầu tiên của dữ liệu từ serial hoặc -1 nếu dữ liệu không có */
int read()
/* Chờ quá trình truyền dữ liệu serial hoàn tất */
flush()
/* In ra serial port dữ liệu data (với bất kì kiểu dữ liệu nào được thiết lập */
print(data)
/* Tương tự như print(data) nhưng sau khi in ra serial -port, con trỏ sẽ xuống dòng tiếp theo */
println(data)
/* Gửi dữ liệu value/string/array đến serial port */
write(byte)
/* Hàm được gọi khi có dữ liệu đến từ chân RX */
SerialEvent()

```

```

/*****
 *                               *
 *   Thư viện hỗ trợ điều khiển động cơ servo *
 *****/
/*
Thiết lập chân kết nối với servo và độ rộng xung
pin : Chân kết nối với servo
[min_uS, max_uS] : Độ rộng xung tính theo us tương ứng
với góc xoay từ 0 đến 180
*/
attach(pin, [min_uS, max_uS])
/* Ghi dữ liệu góc xoay cho động cơ angle từ 0~180 */
write(angle)
/* Viết giá trị để điều khiển góc quay cho servo, giá
trị từ 700 ~ 2300 */
writeMicroseconds(uS)
/* Đọc giá trị góc xoay (0 đến 180 độ) */
int read()
/* Trả về true nếu biến servo đã kết nối đến pin */
bool attached()
/* Gỡ bỏ biến servo ra khỏi chân đã kết nối */
detach()

/*****
 *                               *
 *                               *
 *   Dùng trong giao tiếp I2C *
 *****/
/* Master khởi tạo thư viện Wire với giao tiếp I2C */
begin()
/* Slave tham gia vào kết nối i2c, addr là 7 bits địa
chỉ của slave */
begin(addr)
/*
Master yêu cầu 1 số byte từ slave:
address: 7bits địa chỉ của slave.
count: Số lượng byte master yêu cầu
stop: Kiểu boolean, nếu true, master tín hiệu stop sau
khi yêu cầu và giải phóng bus I2C, nếu false, master
gửi yêu cầu restart để giữ kết nối
*/
requestFrom(address, count, stop)
/* Gửi tín hiệu bắt đầu, truyền dữ liệu đến slave có
địa chỉ addr */
beginTransmission(addr)
/* Gửi dữ liệu (1 byte) đến slave */
send(byte)
/* Gửi dữ liệu (string) đến slave */
send(char * string)
/* Gửi dữ liệu (1 mảng) với số byte là size */
send(byte * data, size)
/* Gửi tín hiệu kết thúc truyền dữ liệu tới slave */
endTransmission()
/* Trả về số byte available sau khi đọc bởi read() */
int available()
/* truy xuất đến 1 byte đã truyền từ slave đến master
hoặc truyền ở chiều ngược lại khi nhận được
requestFrom. Trả về byte tiếp theo đã nhận được */
byte receive()
/* Hàm handler sẽ được gọi khi slave nhận dữ liệu */
onReceive(handler)
/* Handler sẽ được gọi khi master yêu cầu dữ liệu */
onRequest(handler)

/*****
 *                               *
 *                               *
 *   ESP8266 *
 *****/

```

```

/* Dùng khi chương trình cần thực thi nhiều tác vụ cùng
một lúc, thư viện hỗ trợ <Scheduler.h> */
void yield();
/* Reset chip ESP */
void ESP.reset();
/* Trả về kích thước vùng nhớ trống ở heap */
uint32_t ESP.getFreeHeap();
/* Trả về ID của chip ESP */
uint32_t ESP.getChipId();

/* CHẾ ĐỘ CẤU HÌNH WIFI STATION */
/* Thiết lập chế độ station */
WiFi.mode(WIFI_STA);
/* Kết nối đến AP */
WiFi.begin(ssid, password);
/* Ngắt kết nối đến network wifi hiện tại */
bool WiFi.disconnect();
/* Trả về địa chỉ IP của station */
WiFi.localIP();
/* Trả về trạng thái khi kết nối đến AP */
WiFi.status();
/* Trả về tên của network Wifi đã kết nối */
WiFi.SSID();
/* Trả về cường độ của Wifi (đơn vị dBm) */
WiFi.RSSI();
/* Bắt đầu thiết lập chế độ WPS */
WiFi.beginWPSConfig();
/* Bắt đầu thiết lập chế độ smart config */
WiFi.beginSmartConfig();

/* CHẾ ĐỘ CẤU HÌNH WIFI SOFT ACCESS POINT (AP) */
/* Khởi tạo 1 AP với tên và password */
WiFi.softAP(ssid, password);
/* Khởi tạo 1 AP và thiết lập cấu hình cho AP gồm địa
chỉ IP, gateway và subnet */
WiFi.softAPConfig(local_ip, gateway, subnet);
/* Trả về số station đã kết nối đến AP */
WiFi.softAPgetStationNum();
/* Ngắt kết nối của các station */
WiFi.softAPdisconnect(wifioff);
/* Trả về địa chỉ IP của AP */
WiFi.softAPIP();
/* Trả về địa chỉ MAC của AP */
WiFi.softAPmacAddress(mac);

/* WIFI FEATURES */
/* SCAN */
/* Thiết lập chế độ Station */
WiFi.mode(WIFI_STA);
/* Scan và trả về số lượng network available */
WiFi.scanNetworks();
/* Trả về tên của network (kiểu string) ở vị trí num */
WiFi.SSID(num).c_str();
/* Trả về thông tin của tất cả các network */
WiFi.getNetworkInfo(networkItem, &ssid,
&encryptionType, &RSSI, &BSSID, &channel, &isHidden)

/* DIAGNOSTICS */
/*
* Mục đích : chuẩn đoán, cung cấp thông tin và khắc
* phục sự cố khi không kết nối đến net work Wifi
*/
/* In ra serial các chuẩn đoán thông tin của network */
WiFi.printDiag(Serial);
/* Enable chế độ debug */
Serial.setDebugOutput(true);

```

```

/* WEBSERVER */
/* Máy chủ TCP tại port 80 sẽ phản hồi các HTTP request
được gửi lên từ client */
ESP8266WebServer server(80);
/* Server bắt đầu lắng nghe các client */
server.begin();
/* Viết dữ liệu đến các client */
server.write(buf, len)
/* Khởi tạo server ở địa chỉ URL, handleRoot là nội
dung hoặc hàm sẽ thực hiện */
server.on("URL", handleRoot);
/* Server tương tác với client để gửi, nhận dữ liệu */
server.handleClient();
/* Trả về 1 nếu biến val có tồn tại trên server */
server.hasArg(val)
/* Lấy giá trị của biến val trên server */
server.arg(val);
/* Gửi yêu cầu cập nhật dữ liệu từ server:
code : HTTP code
text/html : Định dạng trả về là file HTML
content: Nội dung sẽ trả về từ phía server */
server.send(code, "text/html", content);

/* MQTT */
/*
Các thư viện hỗ trợ giao thức MQTT dành cho ESP8266
trên arduino thường sử dụng là ESP8266MQTTClient
và PubSubClient. Phần này giải thích các hàm của các
thư viện
*/
/* Các hàm của thư viện ESP8266MQTTClient */
/* Khai báo 1 biến mqtt thuộc class MQTTClient */
MQTTClient mqtt;
/* Lấy dữ liệu nhận được từ topic đã subscribe */
mqtt.onData([](String topic, String data, bool cont)
/* Hủy subscribe topic /qos0 */

```

```

mqtt.unsubscribe("/qos0");
/* Thực hiện subscribe topic và publish các message */
mqtt.onSubscribe([](int sub_id)
/* Publish 1 message "qos0" đến topic /qos0 với QoS =0,
và Retain = 0 */
mqtt.publish("/qos0", "qos0", 0, 0);
/* Kết nối đến server MQTT */
mqtt.onConnect([]()
/* Subscribe topic và nhận message tại topic /qos0 */
mqtt.subscribe("/qos0", 0)
/* Bắt đầu truyền nhận dữ liệu với broker MQTT có url
mqtt://test.mosquitto.org tại port 1883 */
mqtt.begin("mqtt://test.mosquitto.org:1883")
/* Hàm được gọi trong loop() nhằm khởi tạo MQTT, kiểm
tra xử lý các dữ liệu từ các topic, kiểm tra các thuộc
tính của giao thức như gói keep-a-live, QoS... */
mqtt.handle();

/* Các hàm của thư viện PubSubClient */
/* Khai báo biến espClient thuộc đối tượng client trong
class PubSubClient */
PubSubClient client(espClient);
/* Publish gói tin "Connected!" đến topic ESP8266 */
client.publish("ESP8266", "Connected!");
/* Subscribe để nhận các message từ topic ESP8266 */
client.subscribe("ESP8266");
/* Cài đặt server lắng nghe client ở port 1883 */
client.setServer(url_server, 1883);
/* Gọi hàm callback để thực hiện các chức năng
publish/subscribe */
client.setCallback(callback);
/* Hàm được gọi trong loop() nhằm khởi tạo MQTT, kiểm
tra
xử lý các dữ liệu từ các topic, kiểm tra các thuộc tính
của giao thức như gói keep-a-live, gói tin QoS... */
client.loop();

```

C - Cheatsheet

```

/* CẤU TRÚC CƠ BẢN */
Viết chú thích trên 1 dòng dùng //
    ex: x++ ; // tăng x 1 đơn vị
/* */ Viết chú thích trên nhiều dòng.
ex : /*****
    * Chú thích được viết *
    * trên nhiều dòng *
    *****/

/* CẤU TRÚC 1 CHƯƠNG TRÌNH */
#include <stdio.h> //include thư viện chuẩn của C
#include "iLib.h" //include thư viện tạo bởi người dùng
int global_var; //biến được dùng trong chương trình
/* Khai báo hàm bắt đầu của 1 chương trình C với kiểu
trả về là integer. Đối số arg kiểu int được truyền vào
hàm */
int main (int arg){
    float local_var ; // Biến chỉ được dùng trong hàm main
    Lệnh 1
    ...
    Lệnh n ;
    return 0; //chương trình thực hiện thành công và thoát
}

/*KIỂU DỮ LIỆU VÀ PHẠM VI */
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

/* ĐẶT TÊN BIẾN */
/* Đặt tên đúng */
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

/* HẰNG SỐ VÀ KIỂU DỮ LIỆU */
123      Số thập phân
0b0111    Số nhị phân
0173     Số Octal - base 8
0x7B     Số thập lục phân base 16
123UL    Số nguyên không dấu

```

```

123L      Số nguyên có dấu 4 bytes
123UL     Số nguyên không dấu 4bytes
123.0     Số thực
1.23e6    Số thực dùng cơ số mũ ex: 1.23*10^3 = 1230
/* định nghĩa hằng số a kiểu nguyên, có giá trị là 1 */
const int a = 1;
/* Định nghĩa hằng số x kiểu thực, có giá trị là 4.0 */
const float x = 4;
/* Định nghĩa hằng số c kiểu integer có giá trị 49 */
const c = '1'; // Ký tự 1 trong mã ASCII là 49
/* Định nghĩa str là hằng số kiểu con trỏ, trỏ tới
chuỗi "Cheasheet C" */
const char * str = "Cheasheet C";

/* KHAI BÁO BIẾN */
/* Khai báo biến a kiểu nguyên và không gán giá trị */
int a;
/* khai báo a kiểu binary, b kiểu base8, c kiểu số
nguyên, d kiểu số nguyên và không gán giá trị */
int a = 0b01111011, b = 0123, c = 1, d;
/* Khai báo biến fa thuộc kiểu số thực float */
float fa = 1.0f;
/* Khai báo biến da thuộc kiểu số thực double */
double da = 1.0;
/* Khai báo biến con trỏ và trỏ đến 1 vùng nhớ không
xác định */
char *pointer;
/* Khai báo biến con trỏ và trỏ về NULL (0)*/
char *other_pointer = NULL;

/* CHUỖI KÍ TỰ */
/* Chuỗi bao gồm kí tự kết thúc chuỗi \0 (null) */
char str1[8] = {'A','r','d','u','i','n','o','\0'};
/* Trình biên dịch tự động thêm kí tự \0 vào cuối
chuỗi */
char str2[8] = {'A','r','d','u','i','n','o'};
/* Khai báo chuỗi ,không khai báo số phần tử và gán giá
trị chuỗi */
char str3[] = "Arduino";
/* Khai báo và gán giá trị cho chuỗi */
char str4[8] = "Arduino";

/* Các hàm xử lí chuỗi thường dùng */
/* Nối các kí tự từ chuỗi source tiếp vào vị trí cuối
của chuỗi dest */
strcat(dest, source)
/* Tìm vị trí xuất hiện đầu tiên của kí tự c trong
source, trả về con trỏ chỉ tới vị trí đó hoặc null nếu
không tìm thấy c trong source */
strchr(source, c)
/* Hàm trả về độ dài của chuỗi st */
strlen(st)
/* copy và thay các kí tự của chuỗi soure vào dest */
strcpy(dest, source)
/* chép kí tự từ đầu đến n từ chuỗi source vào dest */
strncpy(dest, source, n)

```



```

/* MẢNG */
/* Khai báo mảng 1 chiều 6 phần tử kiểu integer và gán giá trị cho mỗi phần tử */
int myPins[] = {2, 4, 8, 3, 6};
/* Khai báo mảng 1 chiều 6 phần tử kiểu integer và không gán giá trị */
int myInts[6];
myInts[0] = 42; // Gán giá trị 42 cho phần tử đầu tiên
myInts[6] = 12; // LỖI ! chỉ số của mảng chỉ từ 0 đến 5
/* Lấy giá trị của phần tử thứ 3 trong mảng myInts */
int c = myInts[2]; // Có thể dùng *(myInts + 2)
/* Lấy địa chỉ của phần tử thứ 3 trong mảng myInts */
int c = &myInts[2]; // Có thể dùng int c = myInts + int
/* Trả về chiều dài của mảng myInts */
int length = sizeof(myInts) / sizeof(myInts[0]);
/* Khai báo 2 mảng kiểu float, arr1 có 5 phần tử, arr2 có 10 phần tử */
float arr1[5], arr2[10];
/* Khai báo mảng số nguyên arr có 2 dòng, 5 cột. Tổng cộng có 10 phần tử */
int a[2][5];

/* KHỞI LỆNH VÀ CÁC LỆNH DÙNG TRONG VÒNG LẶP */
{} // bao gồm nhiều lệnh, thường được sử dụng trong hàm
/* Goto : chương trình sẽ nhảy đến nhãn (nhãn phải có mặt trong câu lệnh chứa goto) */
goto nhãn;
/* Continue : Chỉ dùng trong các lệnh có vòng lặp sẽ chuyển qua chu kỳ mới của vòng lặp trong cùng nhất */
continue; /*
/* Break : Dùng với các vòng lặp thoát khỏi vòng lặp trong cùng nhất, hoặc dùng trong cấu trúc switch..case để thoát ra khỏi case tương ứng */
break; /*
/* Return */
/* Dùng cho hàm không có kiểu trả về (void) */
return;
/* Value có thể là hằng số, biến, biểu thức hoặc gọi đến 1 hàm khác để trả về giá trị */
return <value>;

/* LỆNH Rẽ NHÁNH */
if (x < 5) // thực thi code nếu x<5
{ code }
else if (x > 10) // thực thi code nếu x>10
{ code }
else { code } // thực thi code các trường hợp còn lại

switch (var) { // thực thi case có giá trị var
case 1:
...
break;
case 2:
...
break;
default:
...
}

/* CÁC KIỂU VÒNG LẶP */
/* While: Thực hiện code nếu x<5 */
while (x < 5) { code };
/* Do-While : Thực hiện code, so sánh, nếu x<0 tiếp tục thực hiện code */
do { code } while(x < 0);

```

```

/* for : Khởi tạo và gán giá trị cho i, thực hiện code tăng i nếu i < 10 */
for (int i = 0; i < 10; i++) { code };

/* PHÉP TOÁN VÀ TOÁN TỬ THƯỜNG DÙNG
/* Các toán tử thường dùng */
= toán tử bằng
+ toán tử cộng
- toán tử trừ
* toán tử nhân
/ toán tử chia lấy phần nguyên
% toán tử chia lấy phần dư
== phép so sánh bằng
!= phép so sánh không không bằng (khác)
< phép so sánh nhỏ hơn
> phép so sánh lớn hơn
<= phép so sánh nhỏ hơn hoặc bằng
>= phép so sánh lớn hơn hoặc bằng
&& phép toán logic (AND)
|| phép toán logic (OR)
! phép toán logic (NOT)

/* Các toán tử hợp nhất */
++ tăng 1 đơn vị
-- giảm 1 đơn vị
+= phép toán cộng và gán giá trị
ex: x = 5; x+= 1; //x = 6
-= phép toán trừ và gán giá trị
ex: x = 5; x-= 1; //x = 4
*= phép toán nhân và gán giá trị
ex: x = 5; x*= 3; //x = 15
/= phép toán chia lấy phần nguyên và gán giá trị
ex: x = 6; x/= 2; //x = 3
&= phép toán logic AND và gán giá trị
ex: x = 0b1010; x&= 0110; //x = 0b0010
|= phép toán logic OR và gán giá trị
ex: x = 0b1010; x|= 0110; //x = 0b1110

/* Các toán tử trên bit */
& and ^ xor
<< dịch trái >> dịch phải
| or ~ not

/* THỰC THI VỚI CON TRỎ */
&reference: // lấy địa chỉ của biến mà con trỏ trỏ tới
*dereference:// lấy giá trị của biến mà con trỏ trỏ tới
/* khai báo biến con trỏ kiểu int trỏ tới địa chỉ của biến a */
int a = 5; int *pointer = &a;

/* CÁC KÍ TỰ ĐIỀU KHIỂN VÀ KÍ TỰ ĐẶC BIỆT */
\n Nhảy xuống dòng kế tiếp canh về cột đầu tiên
\t Canh cột tab ngang.
\r Nhảy về đầu hàng, không xuống hàng.
\a Tiếng kêu bip.
\\ In ra dấu \
\" In ra dấu "
\' In ra dấu '
%%: In ra dấu %
\b ~ backspace (xóa 1 ký tự ngay trước)

/* HÀM VÀ CÁC VẤN ĐỀ LIÊN QUAN */
/* Khai báo prototype của hàm max, có 2 đối số đầu vào là a và b thuộc kiểu số nguyên, kết quả trả về của hàm kiểu số nguyên */
int max(int a, int b);

```

```

/* Khai báo biến c là giá trị trả về của hàm max */
int c = max(5,4);
/* Khai báo prototype của hàm không có đối số và không
có kiểu trả về (void) */
void none();

/* TYPEDEF- Định nghĩa kiểu dữ liệu */
/* Định nghĩa kiểu unsigned char là BYTE, khai báo các
biến a, b thuộc kiểu BYTE */
typedef unsigned char BYTE;  BYTE  a, b;

/* KIỂU LIỆT KÊ - ENUMERATION (enum) */
/* khai báo kiểu dữ liệu enum là các ngày trong tuần */
enum daysOfWeek { sunday, monday, tuesday, wednesday };
/* Tạo biến toDay thuộc daysOfWeek và gán giá trị */
daysOfWeek toDay = wednesday;

/* STRUCT - KIỂU DỮ LIỆU DO NGƯỜI DÙNG ĐỊNH NGHĨA */
/* Khai báo struct sinhVien */
struct sinhVien{
    char tenSinhVien;
    char MSSinhVien;
    int tuoiSinhVien;
};
/* Truy xuất đến thành phần MSSinhVien trong struct
sinhVien */
sinhVien.MSSinhVien;
/* Đổi tên struct sinhVien thành 1 biến duy nhất là

```

```

SINHVIEN */
typedef struct sinhVien SINHVIEN;
/* Khai báo biến sinhVienA thuộc struct SINHVIEN */
SINHVIEN sinhVienA;

/* CÁC LỆNH XỬ LÝ TẬP TIN (#include <stdio.h>) */
/* Khai báo 1 biến con trỏ là đường dẫn của 1 file */
const char *filePath = "Đường/dẫn/file/document.txt";
/* Tạo 1 biến con trỏ thuộc kiểu FILE */
FILE *file;
/* Mở 1 file ở đường dẫn filePath và đọc dữ liệu */
file = fopen(filePath, "r");// Trả về NULL nếu thất bại
/* Đóng 1 file đã mở, trả về 0 nếu thành công , ngược
lại trả về EOF */
fclose(file);
/* Viết kí tự c lên file đang mở, trả về EOF nếu ghi
thất bại, trả về mã ASCII của c nếu thành công */
int fputc(int c, FILE *f);
/* Viết chuỗi "hello" lên file đang mở */
int c = fputs("hello", file);
/* Đọc (255-1) kí tự từ file đang mở, kết quả đọc
được
sẽ lưu vào mảng str, việc đọc bị dừng nếu gặp kí tự
'\n' hoặc EOL */
fgets(str, 255, file);
/* Thay đổi vị trí trỏ đến trong file của con trỏ
internal file position indicator về lại đầu file */
int fseek(file, 0, SEEK_SET);
/* Trả về kích thước của nội dung có trong file */
ftell(file);

```

www.cheatography.com/ashlyn-black/cheat-sheets/c-reference/

Lời kết

Các thành viên tham gia đóng góp.

Để hoàn thiện nội dung của sách còn có sự đóng góp của các thành viên sau:

- 1. **Phạm Minh Tuấn (TuanPM)** - Chủ biên
- 2. **Lâm Nhật Quân** - Kỹ sư làm việc tại IoT Maker Việt Nam.
- 3. **Trịnh Hoàng Đức** - Kỹ sư làm việc tại IoT Maker Việt Nam.
- 4. **Lê Phương Trình** - Thực tập sinh tại IoT Maker Việt Nam - Sinh viên Đại Học Bách Khoa, chuyên ngành điện tử viễn thông, khóa học 2014.
- 5. **Trần Phúc Vinh** - Thực tập sinh tại IoT Maker Việt Nam - Sinh viên Đại Học Bách Khoa, chuyên ngành kỹ thuật điện, khóa học 2014.

Và sự đóng góp của cộng đồng tại arduino.esp8266.vn

Lời kết.

Thật vui khi bạn đã đồng hành cùng chúng tôi đi đến hết cuốn sách này. Mục đích của cuốn sách là giúp những người mới bắt đầu tìm hiểu về **Internet Of Things (IoT)** có kiến thức cơ bản và hướng đi chính xác để nghiên cứu về IoT một cách nhanh chóng hơn. Hi vọng cuốn sách sẽ đến tay thật nhiều bạn đam mê lĩnh vực công nghệ còn mới mẻ nhưng rất tiềm năng này. Chúc các bạn thành công trên con đường mà mình đã chọn.

Mặc dù đã cố gắng để hoàn thành tốt nhất nội dung cho cuốn sách, tuy nhiên vẫn không tránh khỏi những thiếu sót. Mọi ý kiến đóng góp xin gửi mail về địa chỉ support@iotmaker.vn

Ngoài ra các bạn có thể tương tác với chúng tôi qua 1 số địa chỉ:

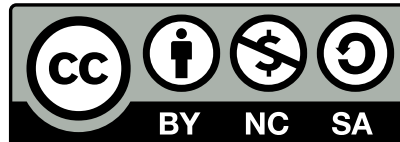
- 1. Webstore : iotmaker.vn/
- 2. Facebook : www.facebook.com/iotmaker.vn/

Một số trang web đóng góp cho cộng đồng (mong nhận được sự chia sẻ và đóng góp của các bạn để cộng đồng IoT Việt Nam nói riêng cũng như cộng đồng kỹ thuật Việt Nam nói chung ngày một phát triển hơn):

- 1. Về ESP8266 với Arduino : arduino.esp8266.vn/

- 2. Về ESP32 : esp32.vn
- 3. Facebook Group: www.facebook.com/groups/iotmaker/

Giấy phép sử dụng tài liệu.



- Tài liệu tuân theo giấy phép CC-BY-NC-SA (creativecommons.org/licenses/by-nc-sa/4.0/legalcode)
- Bản quyền toàn bộ tài liệu này thuộc về [IoT Maker Việt Nam](#), bạn được miễn phí sử dụng cho mục đích cá nhân, học tập và sử dụng trong các dự án của mình, không được sử dụng cho mục đích thương mại. Nếu bạn muốn sửa chữa, phân phối lại, bạn bắt buộc phải giữ nguyên giấy phép và cần có sự đồng ý của [IoT Maker Việt Nam](#).
- Chỉ duy nhất các cá nhân, tổ chức được liệt kê tại iota.edu.vn là được phép sử dụng tài liệu cho mục đích thương mại.