

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG  
CƠ SỞ TẠI THÀNH PHỐ HỒ CHÍ MINH**

**KHOA KĨ THUẬT ĐIỆN TỬ 2**



**LẬP TRÌNH NHÚNG CƠ BẢN**

Sinh viên thực hiện: Lê Hữu Thanh

MSSV: N18DCDT051

Lớp: D18CQKD01-N

Giáo viên hướng dẫn: TS. Chung Tấn Lâm

TP.HCM – tháng 9 năm 2022

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG  
CƠ SỞ TẠI THÀNH PHỐ HỒ CHÍ MINH  
KHOA KĨ THUẬT ĐIỆN TỬ 2**

---



**LẬP TRÌNH NHÚNG CƠ BẢN**

Sinh viên thực hiện: Lê Hữu Thanh  
MSSV: N18DCDT051  
Lớp: D18CQKD01-N  
Giáo viên hướng dẫn: TS. Chung Tấn Lâm

TP.HCM – tháng 9 năm 2022

## LỜI MỞ ĐẦU

Trong những năm gần đây, IoT (Internet of Thing) đã xuất hiện và được ứng dụng rộng rãi trong cuộc sống. IoT giúp mọi người sống và làm việc thông minh hơn, có thể kiểm soát được thời gian của họ một cách tốt nhất. IoT cung cấp cho các doanh nghiệp cái nhìn về thời gian mà hệ thống của họ thực sự hoạt động, cung cấp thông tin chi tiết về mọi thứ từ hiệu suất của máy móc đến chuỗi cung ứng và hoạt động hậu cần. IoT giúp công ty tự động hóa các quy trình và giảm chi phí lao động. Giúp giảm chất thải và cải thiện dịch vụ, làm cho việc sản xuất và giao hàng ít tốn kém hơn, cũng như mang lại sự minh bạch trong các giao dịch của khách hàng. Theo xu hướng phát triển đó, quyển sách này được biên soạn nhằm giúp các bạn có những khái niệm cơ bản về IoT nói riêng hay lập trình nhúng nói chung: “LẬP TRÌNH NHÚNG CƠ BẢN”.

Quyển sách này có gồm có bốn chương:

- Chương I: Môi trường làm việc
- Chương II: Thư viện ESP32
- Chương III: Node-red
- Chương IV: Database
- Chương V: Project SmartHome
- Chương VI: Tổng kết

Để các bạn có thể tiếp cận nhanh hơn thì ngoài việc cung cấp cho các bạn những kiến thức cơ bản thì ở quyển sách này sẽ đồng thời cùng bạn xây dựng một dự án “nhà thông minh” mini.

Chương I mình sẽ hướng dẫn các bạn tạo một môi trường code thuận tiện nhất. Ở chương II mình sẽ cung cấp các lý thuyết cơ bản và cùng bạn bước đầu tạo ra dự án cho bản thân. Ở Chương III và chương IV mình cũng sẽ cung cấp lý thuyết cơ bản và đồng thời cùng bạn tạo một server để quản lý thông tin và lưu trữ dữ liệu của các thiết bị. Chương V sẽ tổng kết, nêu các ưu nhược điểm và hướng phát triển tiếp theo.

Giờ thì cùng nhau triển nào!

## MỤC LỤC

LỜI MỞ ĐẦU .....	3
MỤC LỤC .....	4
MỤC LỤC HÌNH .....	7
CHƯƠNG I: MÔI TRƯỜNG LÀM VIỆC .....	8
1.1    Tổng quan: .....	8
1.2    Môi trường lập trình: .....	8
1.2.1    Ưu nhược điểm: .....	8
1.2.2    Cài đặt Visual Studio Code:.....	9
1.2.3    Cài đặt Arduino:.....	13
1.3    Môi trường quản lý code (mã nguồn) online:.....	21
1.4    Tổng kết chương:.....	28
CHƯƠNG II: THƯ VIỆN ESP 32.....	29
2.1    Tổng quan: .....	29
2.2    Multi WiFi: .....	30
2.2.1    WiFi: .....	31
2.2.2    Cá nhân hóa thư viện Multi Wifi: .....	32
a)    Hướng đi code: .....	34
b)    Giải thích code: .....	34
c)    Full Code library my_wifi_multi.h và chạy thử thư viện.....	40
2.3    OTA: .....	44
2.3.1    OTA là gì?.....	44
2.3.2    Cá nhân hóa thư viện OTA .....	45
a)    Hướng đi code: .....	46
b)    Giải thích code: .....	46
c)    Full Code library my_ota.h và chạy thử thư viện.....	50
2.4    Interrup: .....	53
2.4.1    Interrup là gì? .....	53
2.4.2    Cá nhân hóa thư viện Interrup.....	53
a)    Hướng đi code: .....	54
b)    Giải thích code: .....	54
c)    Full Code library my_interrupt.h và chạy thử thư viện .....	57

2.5 MQTT: .....	58
2.5.1 MQTT là gì?.....	58
2.5.2 Cá nhân hóa thư viện MQTT .....	59
d) Hướng đi code: .....	60
e) Giải thích code:.....	60
f) Full Code library my_interrupt.h và chạy thử thư viện .....	66
2.6 Quy chuẩn Project:.....	68
2.6.1 Gộp thư viện:.....	68
2.6.2 Quy chuẩn project: .....	70
CHƯƠNG III: NODE-RED.....	72
3.1 Tổng quan: .....	72
3.2 Cài đặt Node-red:.....	72
3.3 Hướng dẫn sử dụng cơ bản node-red.....	78
3.3.1 Sử dụng cơ bản: .....	78
a) Palete: Đây là nơi chứa các node mà bạn đã install package. .....	78
b) Màn hình làm việc:.....	78
c) Thanh công cụ: .....	79
d) Thêm các package vào node-red (cái này mình hay gọi là library trên Node-red) 82	
3.3.2 Chạy ví dụ: .....	84
CHƯƠNG IV: MY SQL.....	101
4.1 Tổng quan: .....	101
4.2 Cài đặt MySQL:.....	101
4.3 Hướng dẫn sử dụng cơ bản MySQL.....	104
4.3.1 Sử dụng cơ bản: .....	104
a) Tạo một Database: .....	104
b) Tạo một Table trong Database : .....	105
CHƯƠNG V : PROJECT ESP32_SMARTHOME.....	108
5.1 Thiết kế hệ thống: .....	108
5.1.1 Thiết kế sơ đồ khái niệm :.....	108
5.1.2 Thiết kế thuật toán hệ thống: .....	110
a) Phân tích thiết kế hệ thống: .....	110

b) Phân tích thiết kế logic code: .....	110
5.1.3 Hoàn thiện firmware: .....	115
a) configs.h: .....	115
b) app.h: .....	117
c) ESP32_SmartHome.ino:.....	125
5.1.4 Hoàn thiện Node-red:.....	128
a) Thêm thư viện UI Daskboard .....	132
b) Thêm UI vào dashboard .....	133
5.1.5 Hoàn thiện lưu trữ dữ liệu xuống database: .....	142
a) Thêm thư viện MySQL.....	142
b) Truy xuất dữ liệu từ node MySQL.....	143
c) Thêm dữ liệu vào database từ node MySQL.....	147
d) Hiển thị và lấy dữ liệu từ database .....	150
5.2 Tổng kết chương: .....	151
CHƯƠNG VI : TỔNG KẾT .....	152
1. Kết luận và nhận xét: .....	152
2. Định hướng phát triển tương lai: .....	152
3. Tổng kết: .....	152

## MỤC LỤC HÌNH

Hình 1.1 ESP-WROOM-32 DEVBOARD	29
Hình 1.2 Sơ đồ PINOUT của ESP32-WROOM	<b>Error! Bookmark not defined.</b>
Hình 1.3 Cấu hình CPU của ESP32	<b>Error! Bookmark not defined.</b>
Hình 1.4 Led đơn	<b>Error! Bookmark not defined.</b>
Hình 1.5 Sơ đồ chi tiết led đơn	<b>Error! Bookmark not defined.</b>
Hình 1.6 Nút nhấn	<b>Error! Bookmark not defined.</b>
Hình 1.7 điện trở	<b>Error! Bookmark not defined.</b>
Hình 1.8 Ký hiệu điện trở thường	<b>Error! Bookmark not defined.</b>
Hình 1.9 cảm biến nhiệt độ độ ẩm DHT11	<b>Error! Bookmark not defined.</b>
Hình 1.10 Màn hình oled ssd1306	<b>Error! Bookmark not defined.</b>
Hình 1.11 Sơ đồ khái cách thức hoạt động của giao thức MQTT	<b>Error! Bookmark not defined.</b>
Hình 1.12 Cách thử truyền nhận dữ liệu của giao thức MQTT	<b>Error! Bookmark not defined.</b>
Hình 1.13 Các phím tắt để sử dụng nhanh các tác vụ trên Visual Code	<b>Error! Bookmark not defined.</b>
Hình 1.14 các biểu tượng lệnh sử dụng nhanh	<b>Error! Bookmark not defined.</b>
Hình 1.15 Tạo project mới	<b>Error! Bookmark not defined.</b>
Hình 1.16 Chính sửa các cấu hình	<b>Error! Bookmark not defined.</b>
Hình 1.17 Biểu tượng Node-Red	72
Hình 1.18 Danh sách các thư viện về IOT	<b>Error! Bookmark not defined.</b>
Hình 1.19 Bốn phần chính trên Node-Red	<b>Error! Bookmark not defined.</b>
Hình 1.20 giao diện Node-Red	<b>Error! Bookmark not defined.</b>
Hình 1.21 Node Inject	<b>Error! Bookmark not defined.</b>
Hình 1.22 Node Debug	<b>Error! Bookmark not defined.</b>
Hình 1.23 Các dạng inject trên node inject	<b>Error! Bookmark not defined.</b>
Hình 1.24 Các loại debug trong node debug	<b>Error! Bookmark not defined.</b>
Hình 1.25 Hoàn thành bằng cách nối chúng chung một flow	<b>Error! Bookmark not defined.</b>
Hình 1.26 Biểu tượng của MySQL	<b>Error! Bookmark not defined.</b>
Hình 1.27 Cách thức hoạt động MySQL	<b>Error! Bookmark not defined.</b>

## CHƯƠNG I: MÔI TRƯỜNG LÀM VIỆC

### 1.1 Tổng quan:

Chắc các bạn cũng đã từng nghe về Visual Studio Code hay Arduino IDE - hai môi trường để lập trình cho các vi xử lý. Nếu bạn chưa nghe về hai môi trường này thì nói nôm na hai môi trường này là nơi để các bạn lập trình các logic và nhúng chúng vào vi xử lý (hai môi trường này giống như CodeVisionAVR để lập trình cho ATmega).

Và để lập trình cho dự án “nhà thông minh” này một cách thuận tiện thì mình sẽ sử dụng cả 2 môi trường lập trình này (lý do và cách cài đặt mình sẽ đề cập ở phần dưới).

Ngoài 2 môi trường để lập trình thì mình còn sử dụng GitHub để lưu trữ mã nguồn. Nếu bạn chưa hình dung GitHub làm gì ở đây, thì nói nôm na GitHub là nơi lưu trữ những dự án và quản lý chúng một cách dễ dàng trên internet.

### 1.2 Môi trường lập trình:

#### 1.2.1 Ưu nhược điểm:

Về 2 môi trường này sẽ có những nhược điểm của cái này và cái kia sẽ bù vào cái nhược điểm đó.

Visual Studio Code		Arduino IDE	
Ưu điểm	Nhược điểm	Ưu điểm	Nhược điểm
Nhắc các syntax (giúp ta code nhanh hơn và đỡ bị sai cú pháp)			Không nhắc các syntax (cú pháp câu lệnh)
	Phải thêm từng thư viện cho từng project (dự án)	Thêm thư viện một lần và sử dụng cho tất cả các project (dự án)	
Có thể tìm nhanh các hàm hay các biến (chỉ cần click vào hàm thì sẽ nhảy đến hàm đó)			Chỉ có thể tìm các hàm và biến trong cùng một folder. (ta sẽ gặp bất lợi khi tìm các hàm ở thư viện)
	Sửa code trên Visual Studio Code (VSC) thì phải tắt	Sửa code trên Arduino thì trên Visual Studio	

## Chương 1: Cơ sở lý thuyết

	Arduino và mở lại thì code mới được cập nhật ở Arduino	Code(VSC) thì sẽ tự động cập nhật.	
--	--	---------------------------------------	--

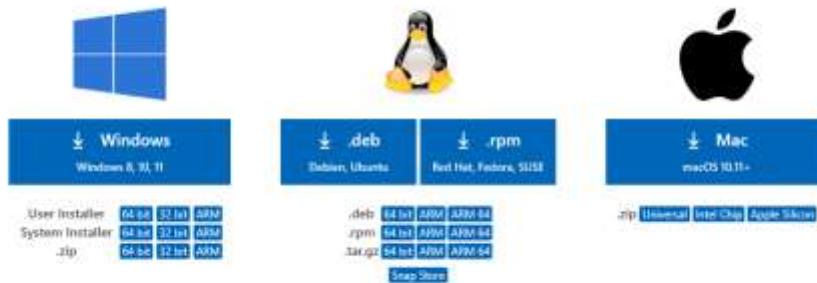
Chính vì có những ưu và nhược điểm trên nên mình sẽ code trên cả 2 môi trường. Nhưng môi trường mình code chủ yếu là Arduino. Còn VSC chỉ là công cụ hỗ trợ cẩn chỉnh lè code (format code) và tìm hàm nhanh chóng.

Lời khuyên: Nếu bạn là người mới code thì hãy code trên Arduino IDE để tạo cho bản thân một thói quen tốt khi code. code trên Arduino sẽ giúp bạn nhớ được cấu trúc lệnh, tựu căn chỉnh code cho thẳng hàng và logic hơn.

### 1.2.2 Cài đặt Visual Studio Code:

Các bạn tải VSC theo link bên dưới

[Download Visual Studio Code - Mac, Linux, Windows](#)



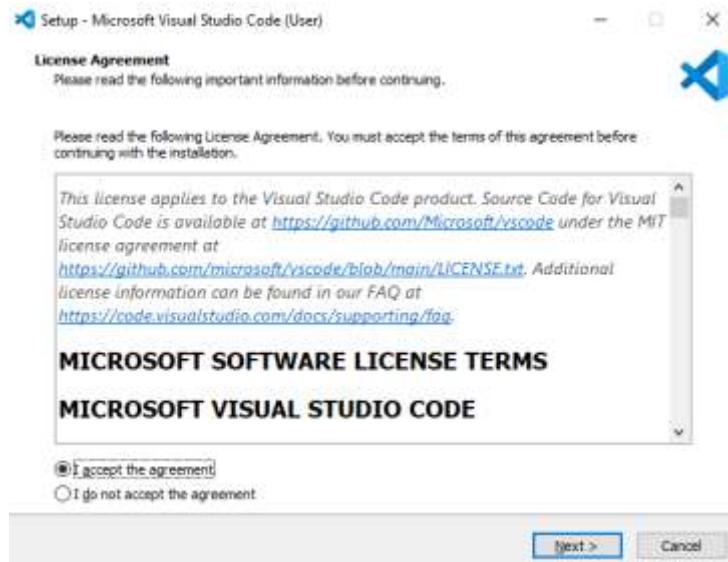
Hình 1.1 giao diện tải VSC

Chọn hệ điều hành bạn sử dụng rồi ấn tải xuống ở đây máy mình sử dụng là Windows.

Bước 1: mở file setup VSC vừa tải xuống.

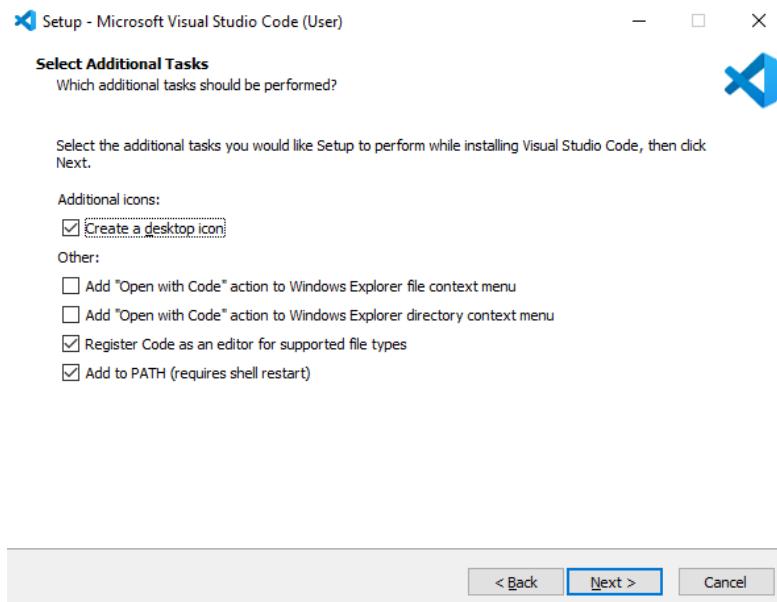
Bước 2: chọn “I accept the agreement” và ấn NEXT

## Chương 1: Cơ sở lý thuyết



Hình 1.2 cài đặt VSC

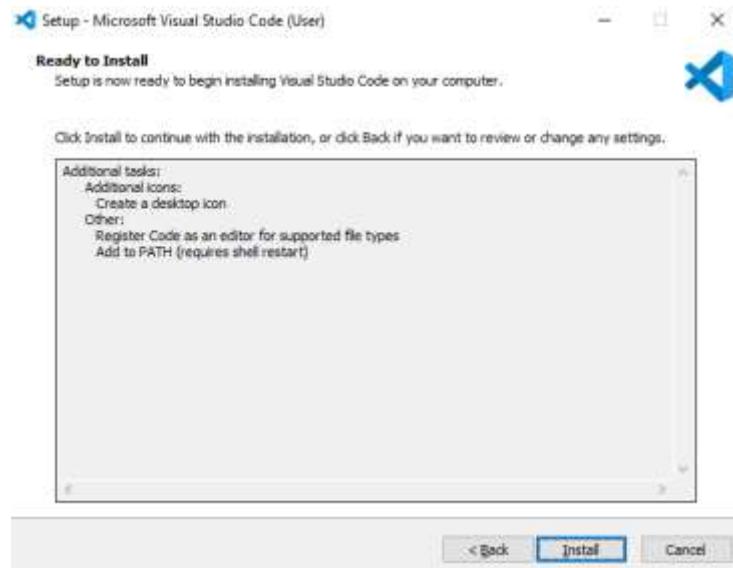
Bước 3: Chọn các tích nhu hình bên dưới rồi ấn NEXT



Hình 1.3 cài đặt VSC

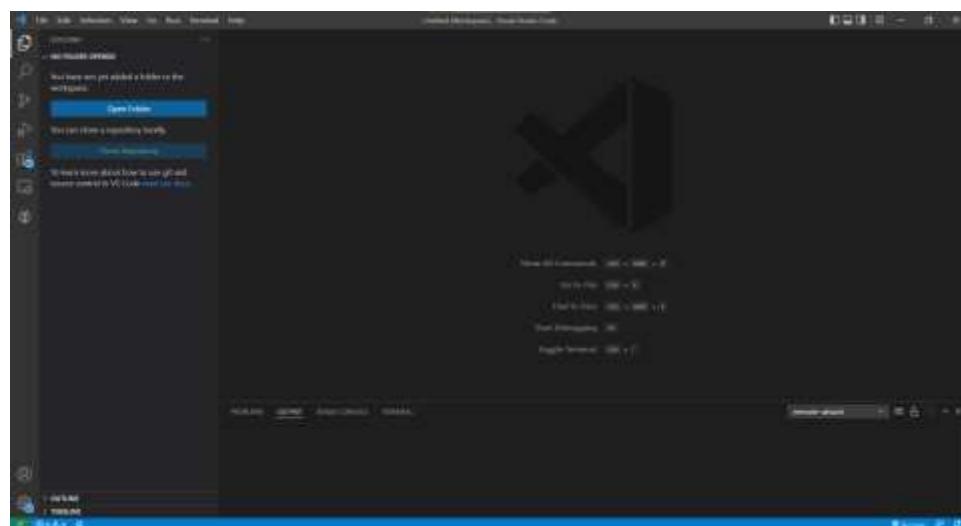
Bước 4: ấn INSTALL để cài đặt và sau đó đợi

## Chương 1: Cơ sở lý thuyết



Hình 1.4 cài đặt VSC

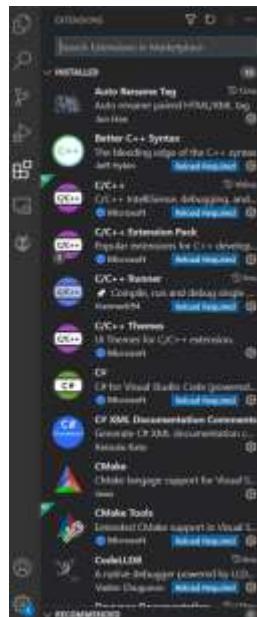
### Bước 5: Mở VCS



Hình 1.5 giao diện VSC

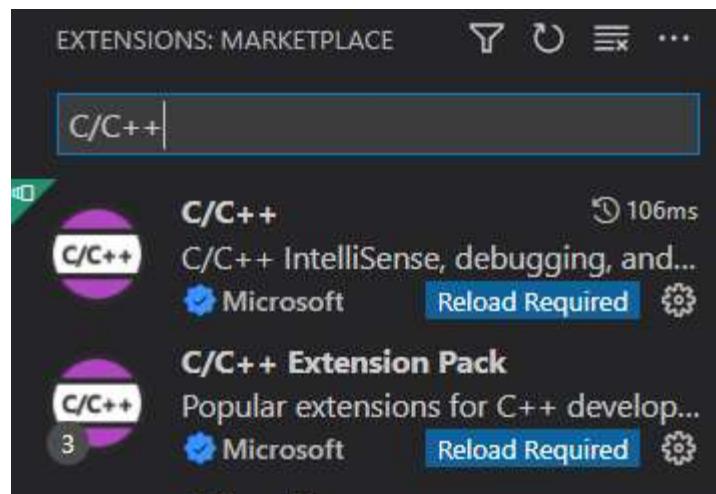
### Bước 6: nhấn tổ hợp phím Ctrl + Shift + X để mở Extension

## Chương 1: Cơ sở lý thuyết



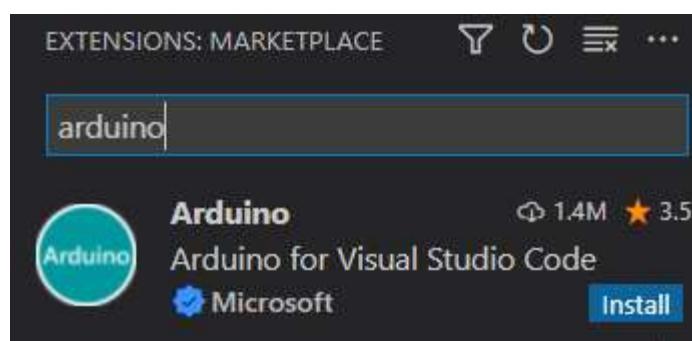
Hình 1.6 cài đặt extension

Bước 7 : tìm “C/C++” và install **C/C++** và **C/C++ Extension Pack**



Hình 1.7 cài đặt extension

Bước 8 : tìm “Arduino” và install **Arduino**



# Chương 1: Cơ sở lý thuyết

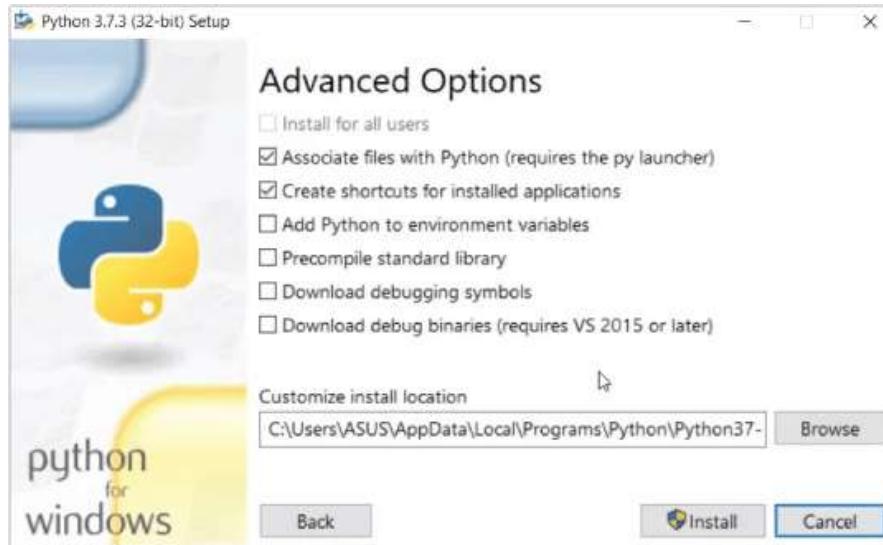
Hình 1.8 cài đặt extension

## 1.2.3 Cài đặt Arduino:

Để tránh bị lỗi khi nạp code thì trước khi cài Arduino các bạn nên cài đặt python trước.

- Link Python : [Download Python | Python.org](https://www.python.org/)

Bước 1: tải **Python** về máy và cài đặt (nhớ chọn **Add Python to environment variables** )



Hình 1.9 cài đặt Python

Bước 2: tải **Arduino IDE** theo hệ điều hành phù hợp với máy tính của các bạn

- Link Arduino: [Software | Arduino](https://www.arduino.cc/en/software)



Hình 1.10 chọn download Arduino phù hợp

Bước 3: chọn **JUST DOWNLOAD**

## Chương 1: Cơ sở lý thuyết



Hình 1.11 tải Arduino

Bước 4: Mở phần mềm Arduino IDE vừa mới tải về lên.



Hình 1.12 cài đặt Arduino

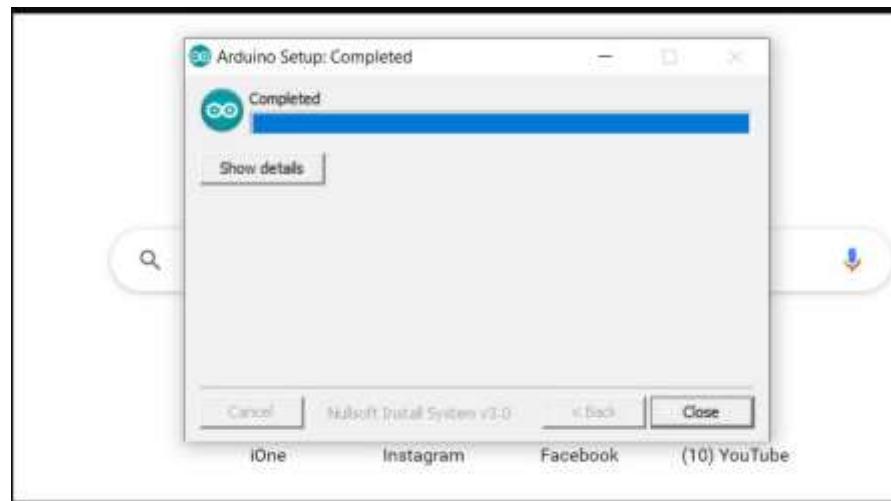
Bước 5: Phần mềm sẽ hiện ra hộp thoại License Agreement. Bạn lần lượt chọn I agree > chọn Next > nhấn Install.

## Chương 1: Cơ sở lý thuyết



Hình 1.13 cài đặt Arduino

Bước 6: Hoàn thành cài đặt Arduino.

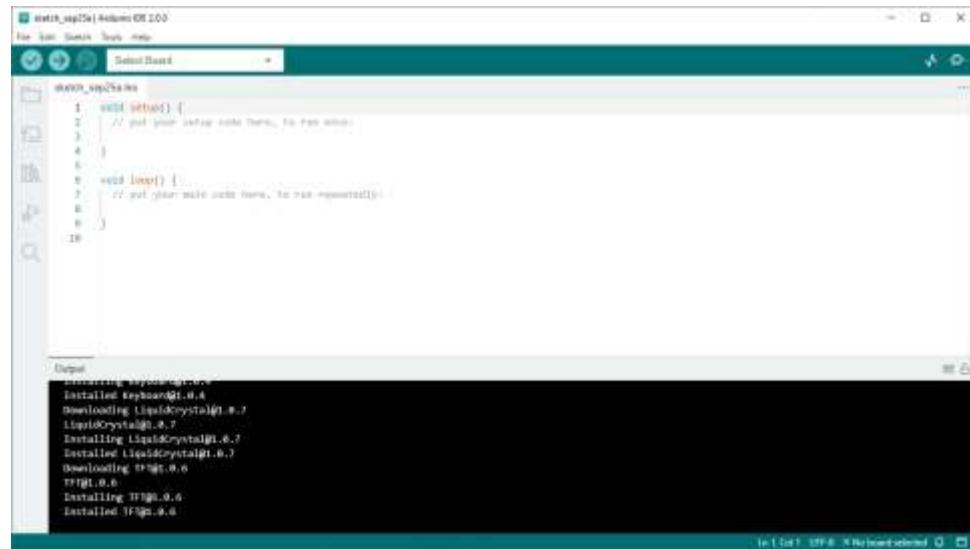


Hình 1.14 cài đặt Arduino

Bước 7: Sau đó mở Arduino lên thì ứng dụng sẽ yêu cầu cần cài đặt thêm các drive liên quan đến việc nạp code thì các bạn cứ ấn Install hết là được.

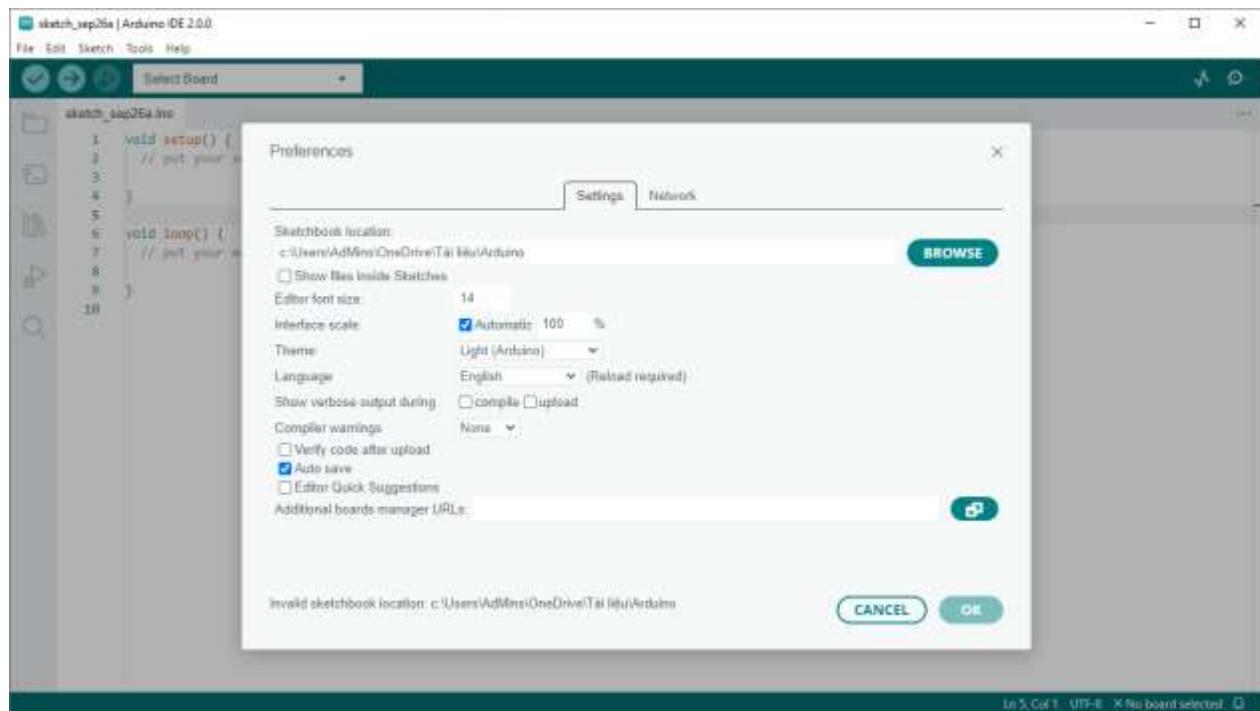
Bước 8: Mở Arduino IDE ta được giao diện như bên dưới.

## Chương 1: Cơ sở lý thuyết



Hình 1.15 giao diện Arduino

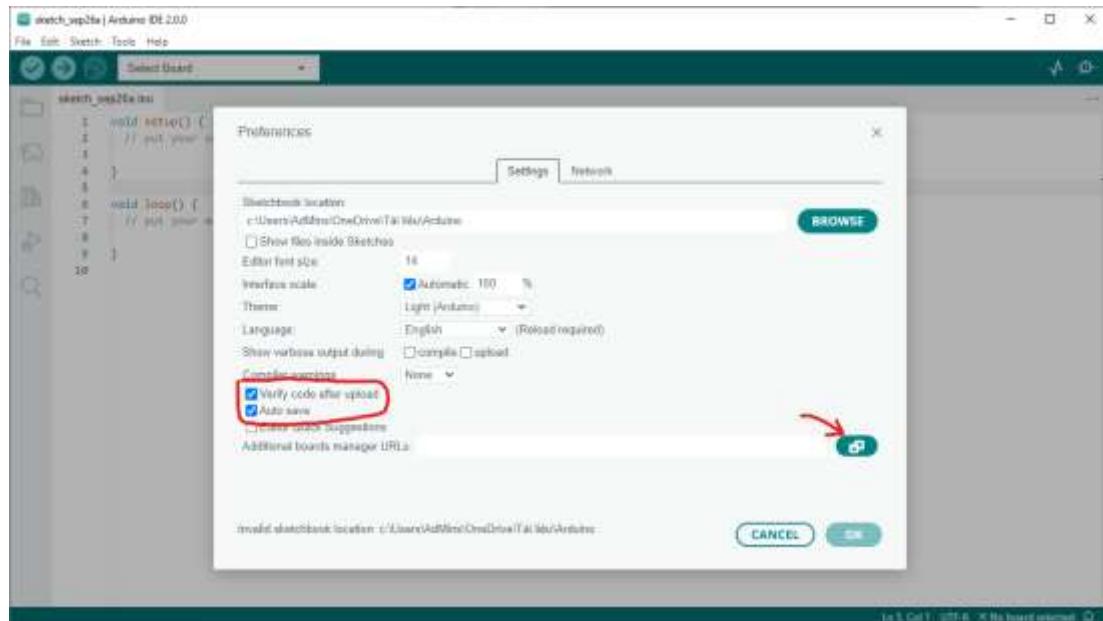
Bước 9: chọn **File > Preferences...** ta được màn hình như bên dưới



Hình 1.16 thêm thư viện vi sử lý.

Bước 10: tích chọn **Verify code after upload** và **Auto Save** sau đó nhấn vào cửa sổ nhỏ bên cạnh **Additional Boards Manager URLs**

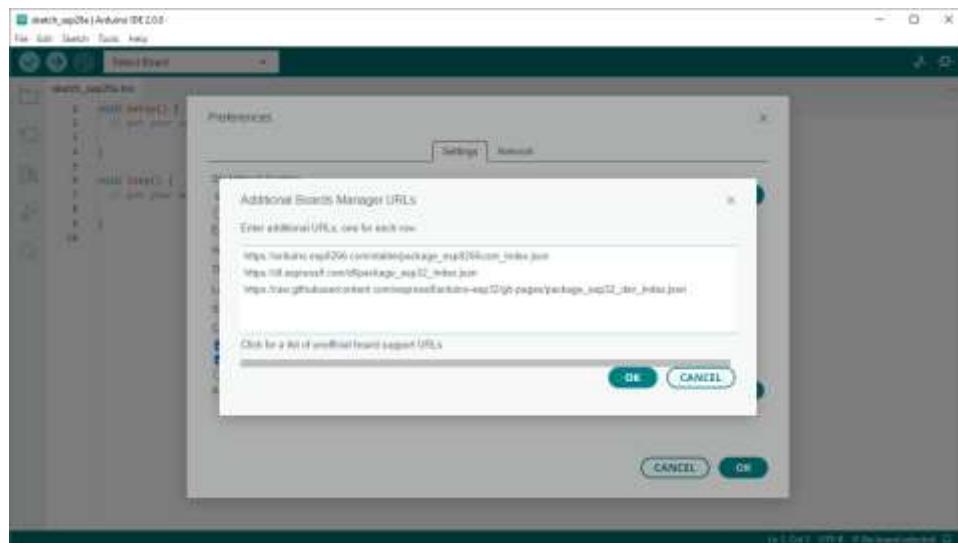
## Chương 1: Cơ sở lý thuyết



Hình 1.17 thêm thư viện vi sử lý.

Bước 11: copy 3 link bên dưới vào **Additional Boards Manager URLs** rồi sau đó nhấn ok

```
https://arduino.esp8266.com/stable/package_esp8266com_index.json  
https://dl.espressif.com/dl/package_esp32_index.json  
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_dev_index.json
```

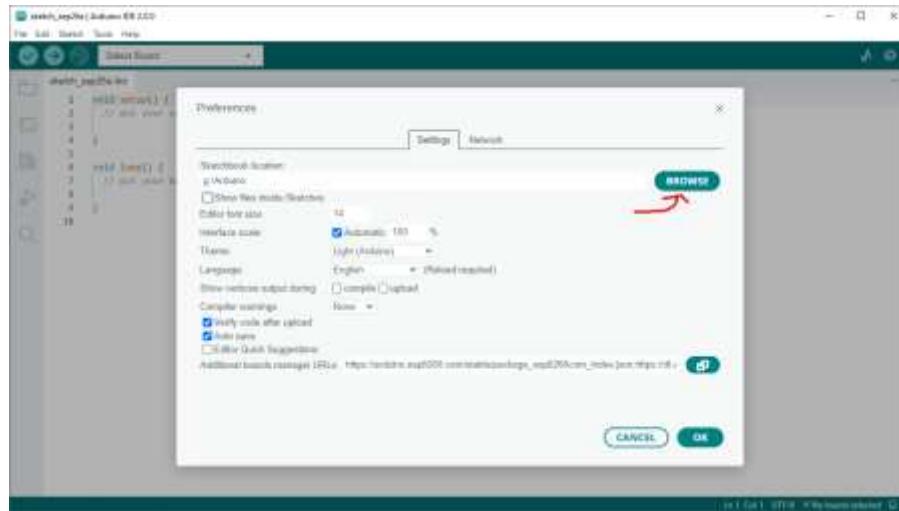


Hình 1.18 thêm thư viện vi sử lý.

## Chương 1: Cơ sở lý thuyết

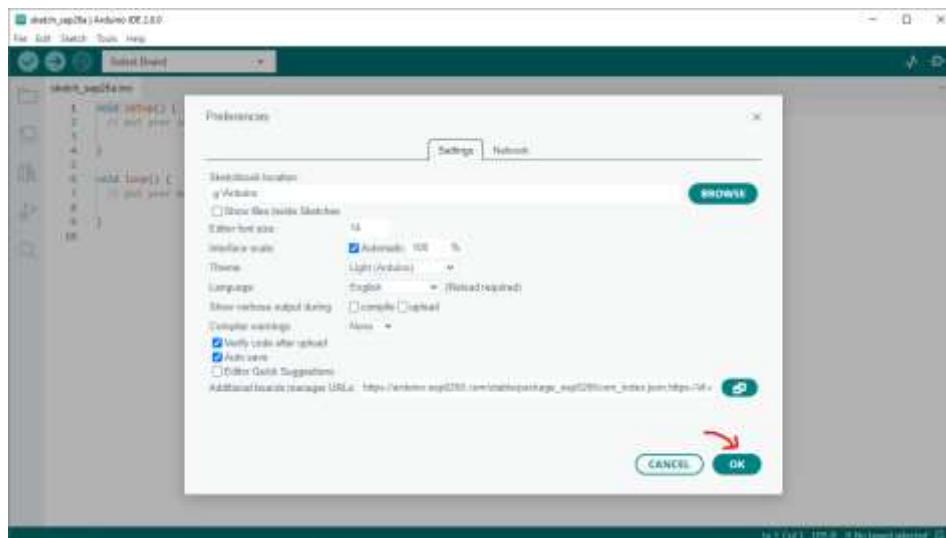
Bước 12: Tạo một Folder để chứa tất cả các project (dự án) rồi dẫn đường dẫn thư mục đó tại **Sketchbook location**

Như mình thì mình tạo 1 Foder tại ổ đĩa G với tên là Arduino (Folder này sẽ là nơi chứa tất cả các project và tất cả các library)



Hình 1.19 thêm thư viện vi sử lý.

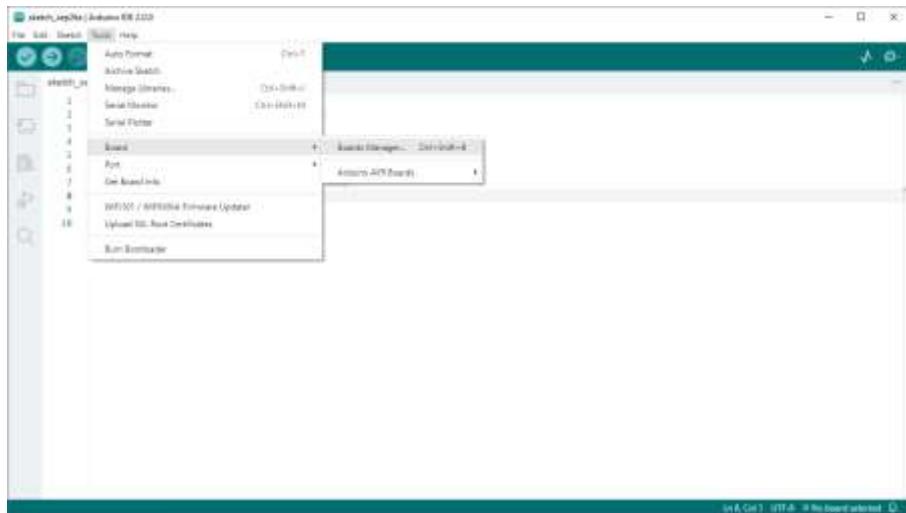
Bước 13: nhấn oke để hoàn thành việc thêm thư viện vi sử lý và Folder chứa các project



Hình 1.20 thêm thư viện vi sử lý.

Bước 14: chọn **Tool > Board > Boards Manager...**

## Chương 1: Cơ sở lý thuyết



Hình 1.21 thêm thư viện vi sử lý.

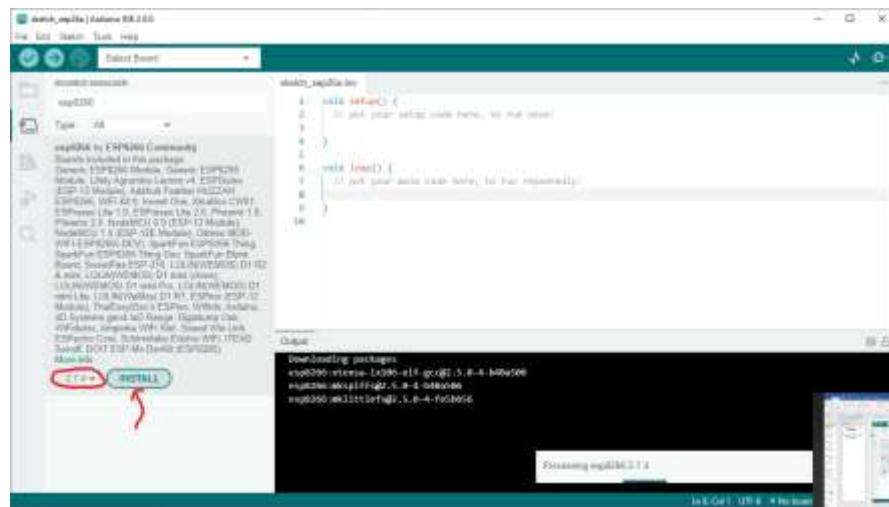
Bước 15: tìm “**esp32**” và chọn version **2.0.3** và **install**. (chờ install hơi lâu nha)



Hình 1.22 thêm thư viện vi sử lý.

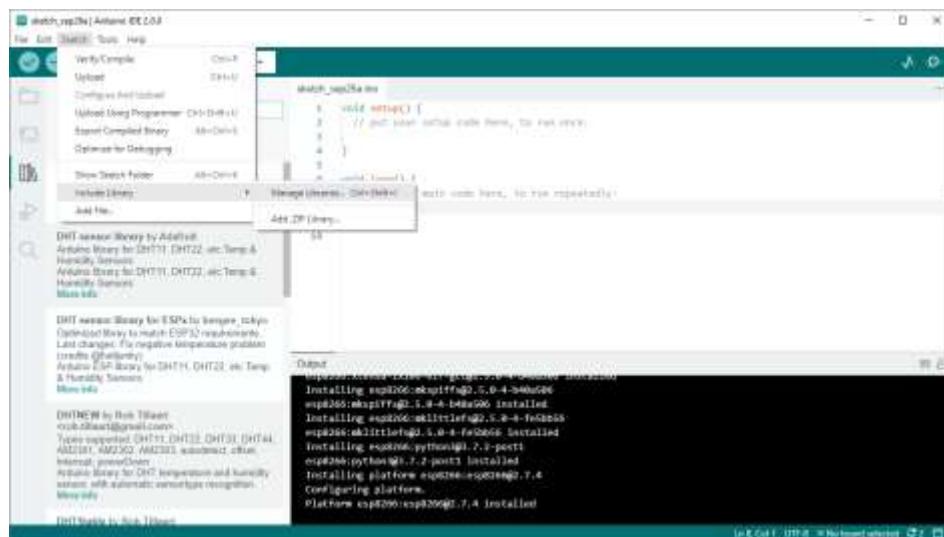
Bước 16: tương tự tìm “**esp8266**” và chọn version **2.7.4** và **install**.

## Chương 1: Cơ sở lý thuyết



Hình 1.23 thêm thư viện vi xử lý.

Bước 17: thêm thư viện vào các Project. Chọn **Sketch > Include Library > Manager libraries...**

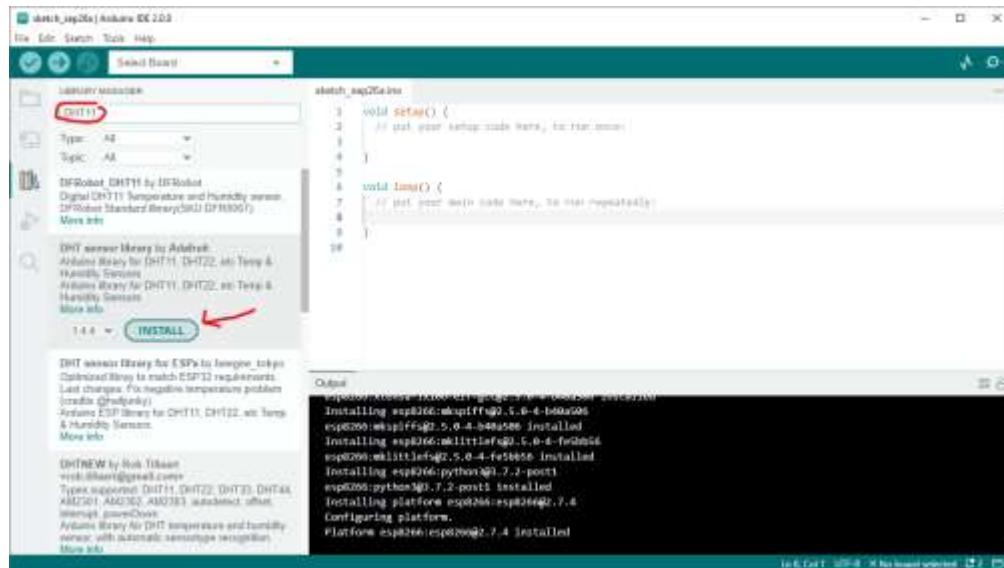


Hình 1.24 thêm thư viện cho project.

Bước 18: Tìm thư viện muốn thêm vào và **install** chúng. Như vậy là ta đã thêm được thư viện vào tất cả các project.

Như ở đây mình cài thư viện DHT (DHT Sensor library)

## Chương 1: Cơ sở lý thuyết



Hình 1.24 thêm thư viện cho project.

### 1.3 Môi trường quản lý code (mã nguồn) online:

Có khá nhiều môi trường để các bạn có thể lựa chọn để lưu trữ và quản lý code của mình như GitLab, BitBucket, CodeBase, GitHub,... Nhưng ở đây mình sẽ lựa chọn **GitHub** vì nó khá gần gũi với cộng đồng code hiện tại. Và GitHub cũng có vài tính năng hay ho khác.

#### Tạo tài khoản và các thao tác cơ bản với GitHub:

- Link GitHub : [GitHub: Where the world builds software · GitHub](https://github.com/)

Bước 1: đăng ký tài khoản nhấn vào **Sign up**



Hình 1.25 Đăng ký tài khoản GitHub.

Bước 2: nhập gmail rồi ấn **continue**

## Chương 1: Cơ sở lý thuyết



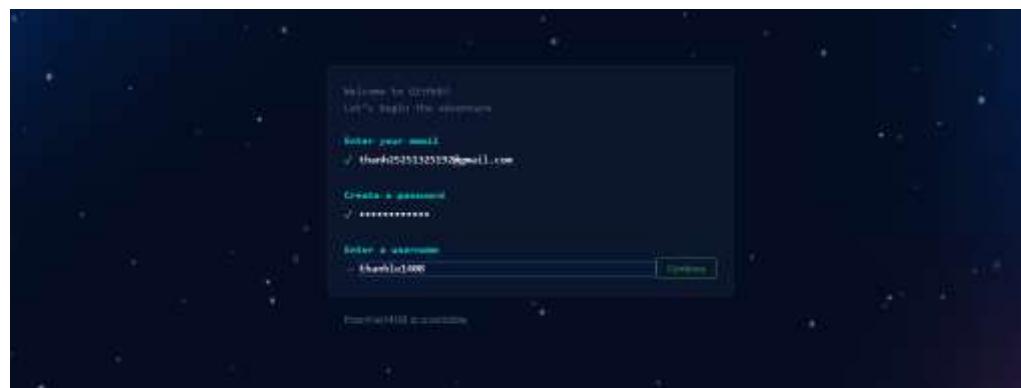
Hình 1.26 Đăng ký tài khoản GitHub.

Bước 3: nhập pass rồi ấn **continue**



Hình 1.27 Đăng ký tài khoản GitHub.

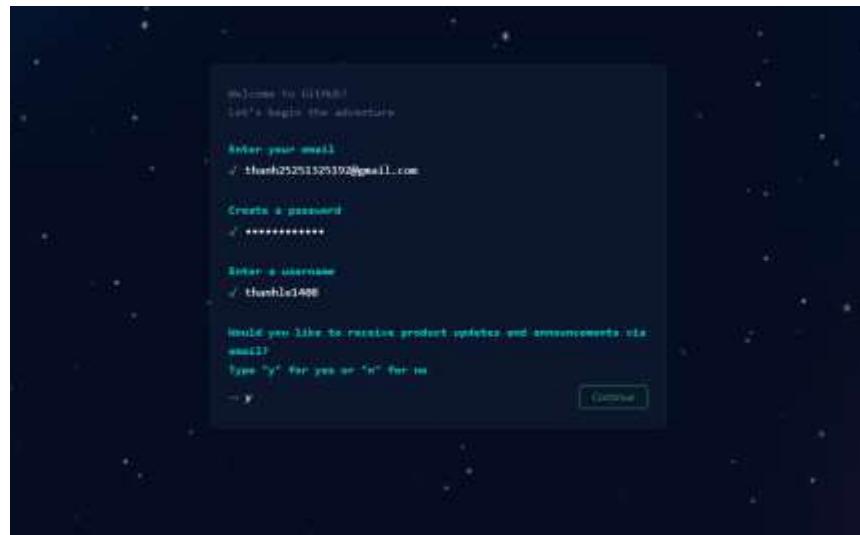
Bước 4: nhập tên rồi ấn **continue**



Hình 1.28 Đăng ký tài khoản GitHub.

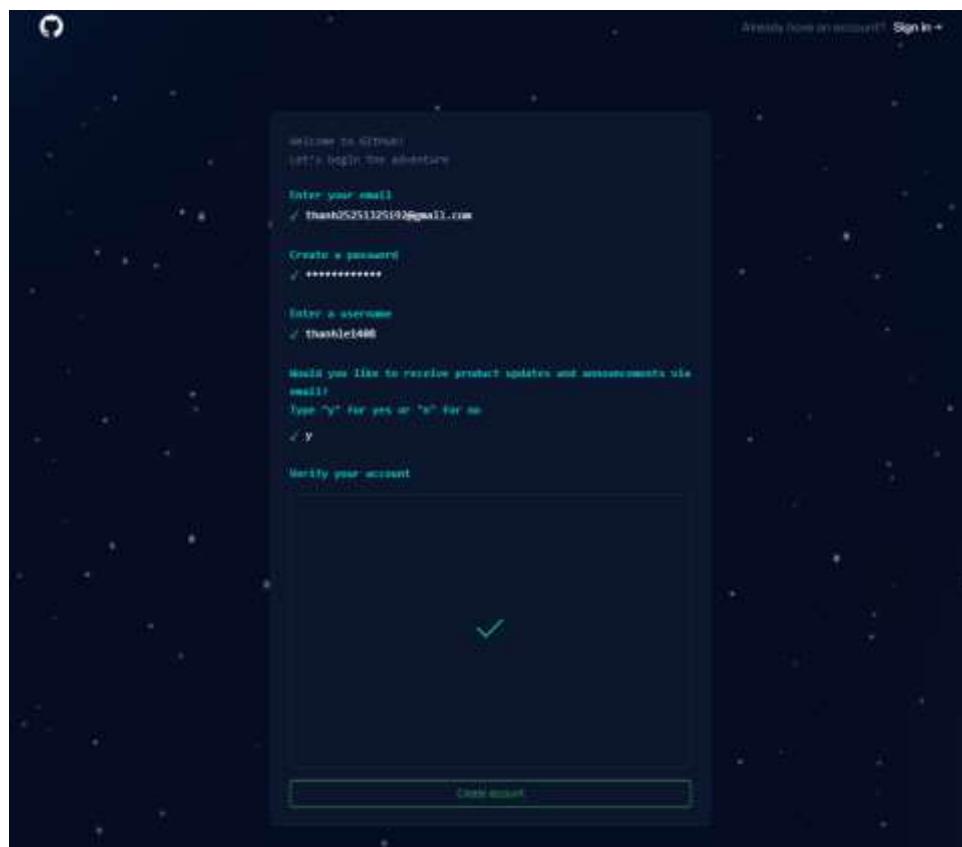
Bước 5: nhập y rồi ấn **continue**

## Chương 1: Cơ sở lý thuyết



Hình 1.29 Đăng ký tài khoản GitHub.

Bước 6: sau đó **verify your account** và nhấn **Create account**

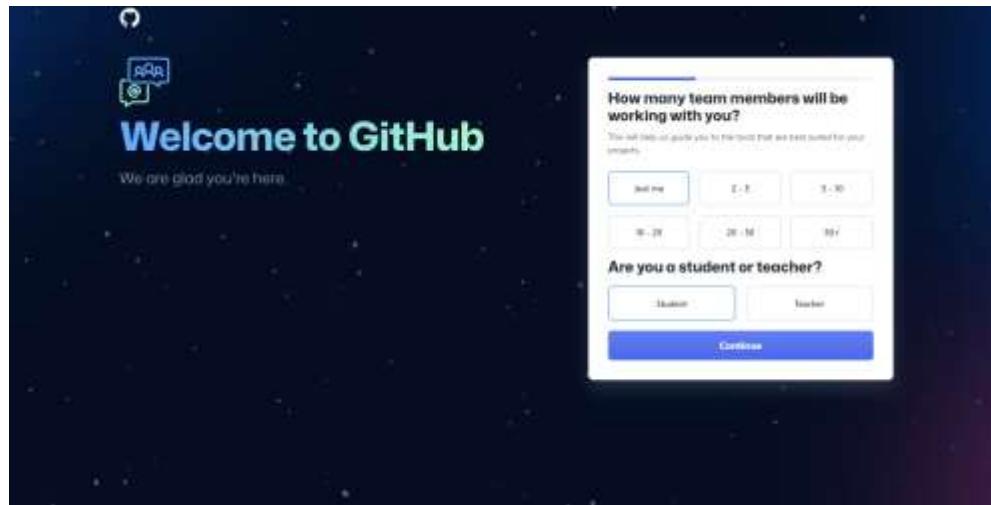


Hình 1.30 Đăng ký tài khoản GitHub.

Bước 7: vào gmail vừa đăng ký lấy 8 ký tự để xác nhận tài khoản

Bước 8: sau đó **chọn “Just me” và “Student”** và nhấn **Continue**

## Chương 1: Cơ sở lý thuyết



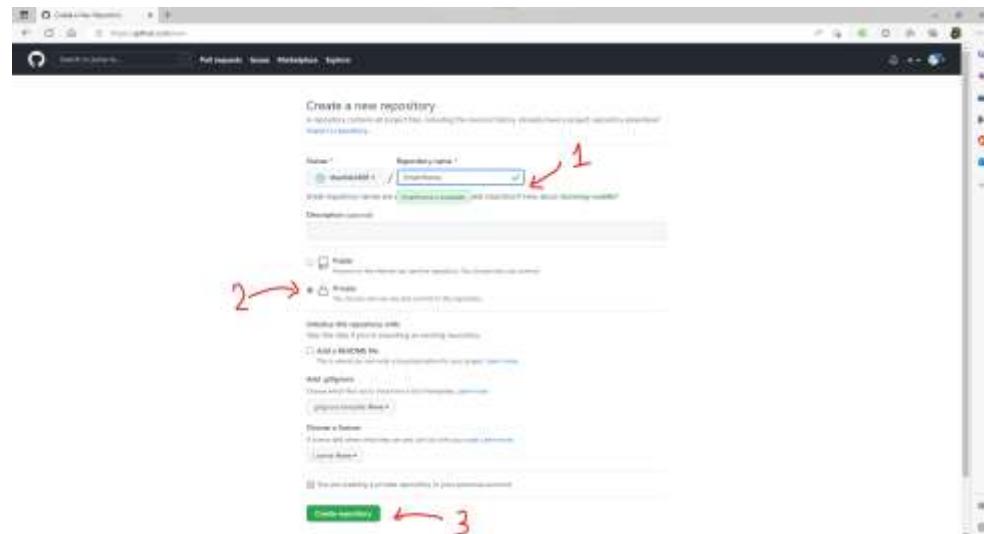
Hình 1.31 Đăng ký tài khoản GitHub.

Bước 9: nhấn **Continue**

Bước 10: nhấn **Continue for free**

Bước 11: tạo 1 repository mới. Nhấn vào dấu "+" > **New repository**

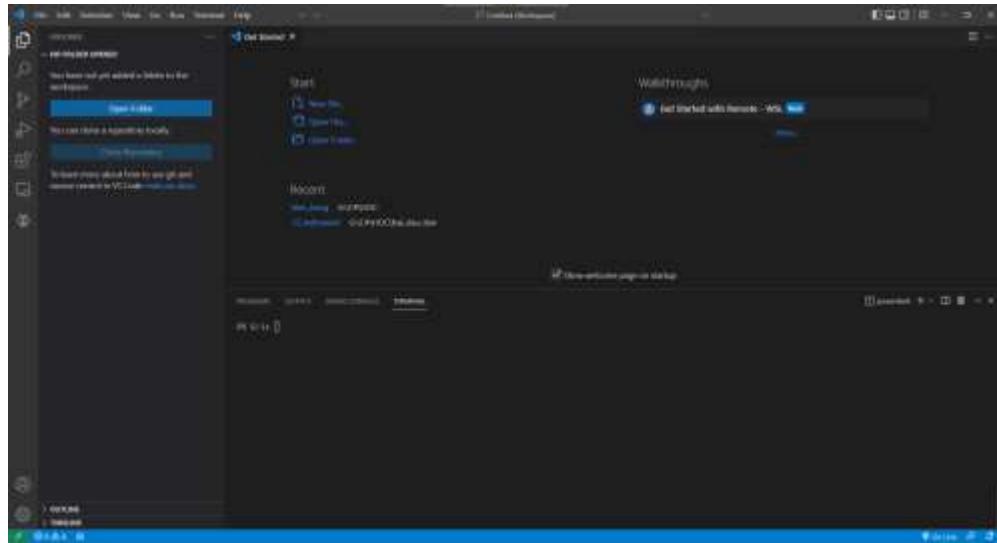
Bước 12: nhập tên project tại **repository new** sau đó chọn **Private** rồi nhấn vào **Create repository**



Hình 1.32 Tạo repository

Bước 13: Vào Visual Studio Code nhấn tổ hợp phím **Ctrl + `** để mở Terminal

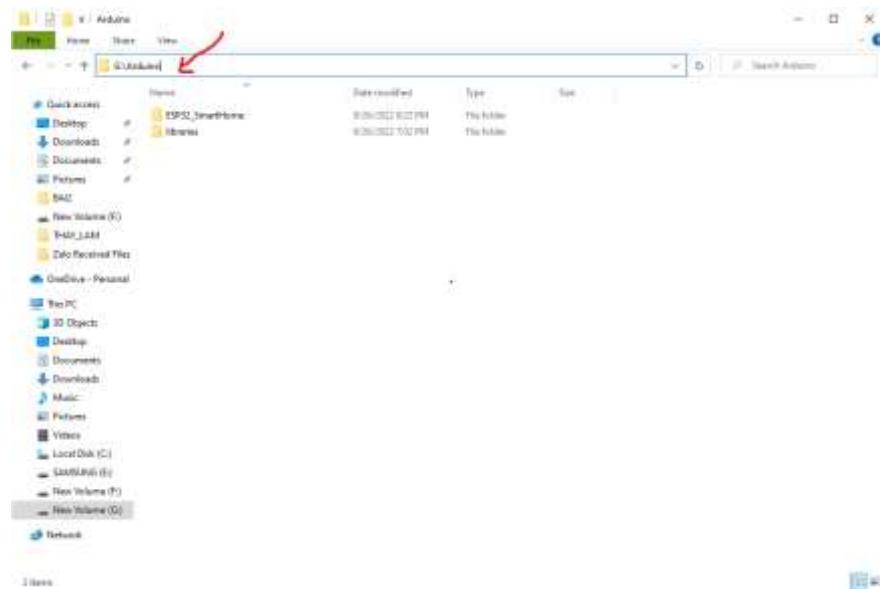
## Chương 1: Cơ sở lý thuyết



Hình 1.33 Các thao tác cơ bản với GitHub

Bước 14: Vào Folder chứa project và library mà bạn đã tạo ở bước cài đặt Arduino. Sau đó click chuột vào đường dẫn rồi sau đó copy đường dẫn thư mục

Như mình đã tạo Folder Arduino ở ô G thì đường dẫn sẽ là **G:\Arduino**



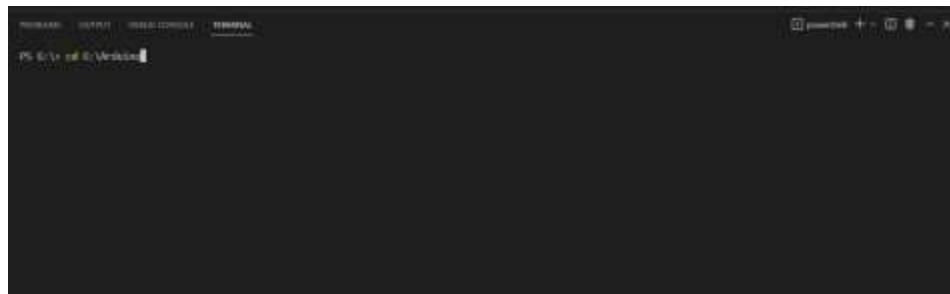
Hình 1.34 Các thao tác cơ bản với GitHub

Bước 15: Vào lại Visual Studio Code và gõ lệnh

**cd G:\Arduino**

trong đó **G:\Arduino** là đường dẫn vào thư mục

## Chương 1: Cơ sở lý thuyết



Hình 1.34 Các thao tác cơ bản với GitHub

### Bước 16: gõ lệnh Git init

```
PS C:\Users\Huy\Documents\GitHub\Arduinos\git-scm.com> git init
warning: 'git' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if it exists, try again.
At line:1 char:1
  + git init
  + CategoryInfo          : ObjectNotFound: (git:cmdlet) [CommandNotFoundException]
  + FullyQualifiedErrorId : CommandNotFoundException
Try the one cross-platform PowerShell: https://aka.ms/powershell
PS C:\Users\Huy\Documents\GitHub\Arduinos\git-scm.com> git init
Initialized empty Git repository in C:/Users/Huy/Arduinos/.git/
PS C:\Users\Huy\Documents\GitHub\Arduinos\git-scm.com>
```

Hình 1.35 Các thao tác cơ bản với GitHub

Note: Nếu bạn bị lỗi đỏ như mình khi gõ **git init**. Bạn hãy cài **Git** ở link bên cạnh ([Git - Downloading Package \(git-scm.com\)](#)) chọn bản Standalone Installer 64bit hay 32bit tùy thuộc vào máy của bạn. sau đó mở file vừa tải xong bạn Next hết là được. sau đó vào lại VSC gõ lại **git init**

### Bước 17: gõ lệnh git add .

(lệnh này là thêm tất cả các file mới nhất lên commit)

```
PS D:\> cd C:\Arduinos
PS C:\Arduinos> git init
Initialized empty Git repository in C:/Arduinos/.git/
PS C:\Arduinos> git add .
warning: In the working copy of 'USM3_SmartHome\USM3_SmartHome.iss', LF will be replaced by CRLF the next time GIT touches it.
warning: In the working copy of 'Library\OMT_sensor_Library\CONTROLING.iss', LF will be replaced by CRLF the next time GIT touches it.
warning: In the working copy of 'Library\OMT_sensor_Library\OMT.cpp', LF will be replaced by CRLF the next time GIT touches it.
warning: In the working copy of 'Library\OMT_sensor_Library\OMT.h', LF will be replaced by CRLF the next time GIT touches it.
warning: In the working copy of 'Library\OMT_sensor_Library\OMT_ADE.iss', LF will be replaced by CRLF the next time GIT touches it.
warning: In the working copy of 'Library\OMT_sensor_Library\code_of_contact.iss', LF will be replaced by CRLF the next time GIT touches it.
warning: In the working copy of 'Library\OMT_sensor_Library\complexOMT_initial_Sensor\OMT_Initial_Sensor.iss', LF will be replaced by CRLF the next time GIT touches it.
warning: In the working copy of 'Library\OMT_sensor_Library\complexOMT_initial_Sensor\OMT_initial_Sensor.iss', LF will be replaced by CRLF the next time GIT touches it.
warning: In the working copy of 'Library\OMT_sensor_Library\Properties\OMTinitialSensor.properties', LF will be replaced by CRLF the next time GIT touches it.
warning: In the working copy of 'Library\OMT_sensor_Library\License.txt', LF will be replaced by CRLF the next time GIT touches it.
PS C:\Arduinos>
```

Hình 1.36 Các thao tác cơ bản với GitHub

### Bước 18: gõ lệnh git commit -m “first commit”

## Chương 1: Cơ sở lý thuyết

( lệnh này là đưa các file vừa add lên. trong đó những từ trong nháy kép **first commit** bạn có thể thay thế bất kể nội dung gì cũng dc)

```
PS G:\Arduino> git commit -m "first commit"
[master (root-commit) 071a664] first commit
 13 files changed, 1257 insertions(+)
 create mode 100644 ESP32_SmartHome/ESP32_SmartHome.ino
 create mode 100644 libraries/DHT_sensor_library/CONTRIBUTING.md
 create mode 100644 libraries/DHT_sensor_library/DHT.cpp
 create mode 100644 libraries/DHT_sensor_library/DHT.h
 create mode 100644 libraries/DHT_sensor_library/DHT_U.h
 create mode 100644 libraries/DHT_sensor_library/README.md
 create mode 100644 libraries/DHT_sensor_library/code-of-conduct.md
 create mode 100644 libraries/DHT_sensor_library/examples/DHT_Unified_Sensor/DHT_Unified_Sensor.ino
 create mode 100644 libraries/DHT_sensor_library/examples/DHTtester/DHTtester.ino
 create mode 100644 libraries/DHT_sensor_library/keywords.txt
 create mode 100644 libraries/DHT_sensor_library/library.properties
 create mode 100644 libraries/DHT_sensor_library/license.txt
```

Hình 1.37 Các thao tác cơ bản với GitHub

Note: nếu các bạn gặp lỗi như hình bên dưới như mình thì gõ lệnh

**git config --global user.email “<https://github.com/thanhle1408/SmartHome.git>”**

(đường link kia là bạn lấy trên GitHub ở phần HTTPS)



```
PS G:\Arduino> git commit -m "first commit"
Author identity unknown

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

fatal: unable to auto-detect email address (got 'AdMins@DESKTOP-DLGRKJK.(none)')
```

Hình 1.38 Lỗi

Bước 19: gõ lệnh **git push**

( lệnh này là đưa các file vừa commit lên internet)

## Chương 1: Cơ sở lý thuyết

```
PS G:\Arduino> git push --set-upstream origin main
info: please complete authentication in your browser...
Enumerating objects: 21, done.
Counting objects: 100% (21/21), done.
Delta compression using up to 6 threads
Compressing objects: 100% (18/18), done.
Writing objects: 100% (21/21), 16.25 KiB | 1.81 MiB/s, done.
Total 21 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/thanhle1408/SmartHome.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
PS G:\Arduino>
```

Hình 1.39 Các thao tác cơ bản với GitHub

Note: nếu các bạn gặp lỗi như hình bên dưới như mình thì gõ lệnh

**git push --set-upstream origin main**

nếu dẫn đến 1 cửa sổ thì đăng nhập GitHub rồi chọn Auth... gì gì đó (mình quên gõ)

```
PS G:\Arduino> git push
fatal: The current branch main has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin main

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.
```

Hình 1.40 Lỗi

Như vậy ta đã xong việc setup môi trường làm việc rồi. Giờ thì chuẩn bị triển khai

### 1.4 Tổng kết chương:

Mình đã hướng dẫn khá chi tiết các bước để cài đặt VSC, Arduino và đăng ký GitHub và dùng các lệnh cơ bản trên Terminal

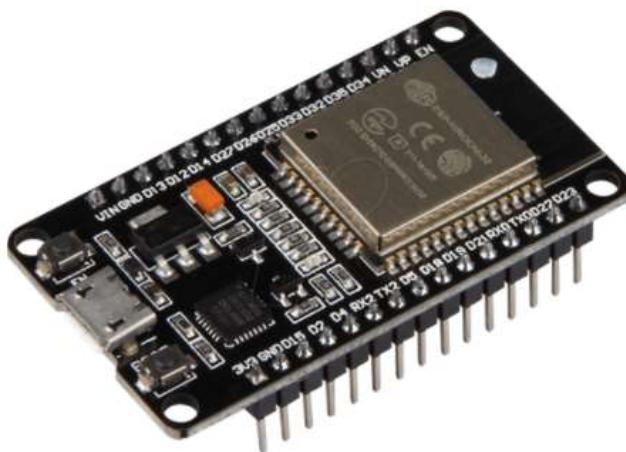
Như vậy:

- Mình sẽ sử dụng VSC khi phải tìm hàm hoặc biến, tạo các file thư viện “.h” trong Folder một cách dễ dàng. ngoài ra mình sử dụng Terminal ở VSC để quản lý source code trên GitHub. Về sau, mình sẽ cài đặt database hay node-red bằng Terminal nữa.
- Mình sẽ sử dụng Arduino để viết code, nhúng code vào ESP32.
- GitHub là nơi mình sẽ lưu lại source code sau mỗi phần để đảm bảo dù máy tính hiện tại có bị sao đi chăng nữa thì mình vẫn có thể tải code về máy khác làm được.

## CHƯƠNG II: THƯ VIỆN ESP 32

### 2.1 Tổng quan:

ESP32 là một series các vi điều khiển trên một vi mạch giá rẻ (khoảng 100k), tiêu tốn năng lượng thấp, có hỗ trợ WiFi và dual-mode Bluetooth. Ta có thể chạy dual core song song (MultiTask), RAM(520KB) nhiều hơn so với các vi xử lý khác, Nhiều cổng giao tiếp và có rất nhiều tính năng hay ho khác. Ngoài ra các bạn cũng có thể sử dụng ESP8266 – một vi mạch tiền nhiệm của ESP32 (cấu hình thấp hơn một chút, và không có Bluetooth). Để biết chi tiết hơn các thông số kỹ thuật khác các bạn có thể truy cập vào link trang chủ của ESPRESSIF ([Development Boards | Espressif Systems](#))



Hình 2.1 ESP-WROOM-32 DEVBOARD

Ở chương này mình sẽ hướng dẫn các bạn một số tính năng trên ESP32. Và một số thiết bị khác để phụ trợ cho các tính năng này (các sensor, nút nhấn, led,...).

Các tính năng mình sẽ chú tâm ở chương này:

- Multi WiFi – Có thể thay đổi mạng khác khi mạng hiện tại bị mất kết nối.
- OTA (Over The Air) – Nạp code thông qua Wifi
- Interrupt - Ngắt ngoài (thực hiện lệnh ngắt khi có sự kiện xảy ra)
- MQTT (Message Queuing Telemetry Transport) - là giao thức truyền thông điệp (message) theo mô hình publish/subscribe (cung cấp / thuê bao)

**Để bắt đầu cho Project SmartHome này thì các bạn tạo một file **ESP32\_SmartHome** trong Folder cài đặt. Folder mà các bạn đã tạo ở phần cài đặt Arduino (đường dẫn folder của mình là G:\Arduino)**

Bước 1: Mở Arduino IDE chọn File > Save.

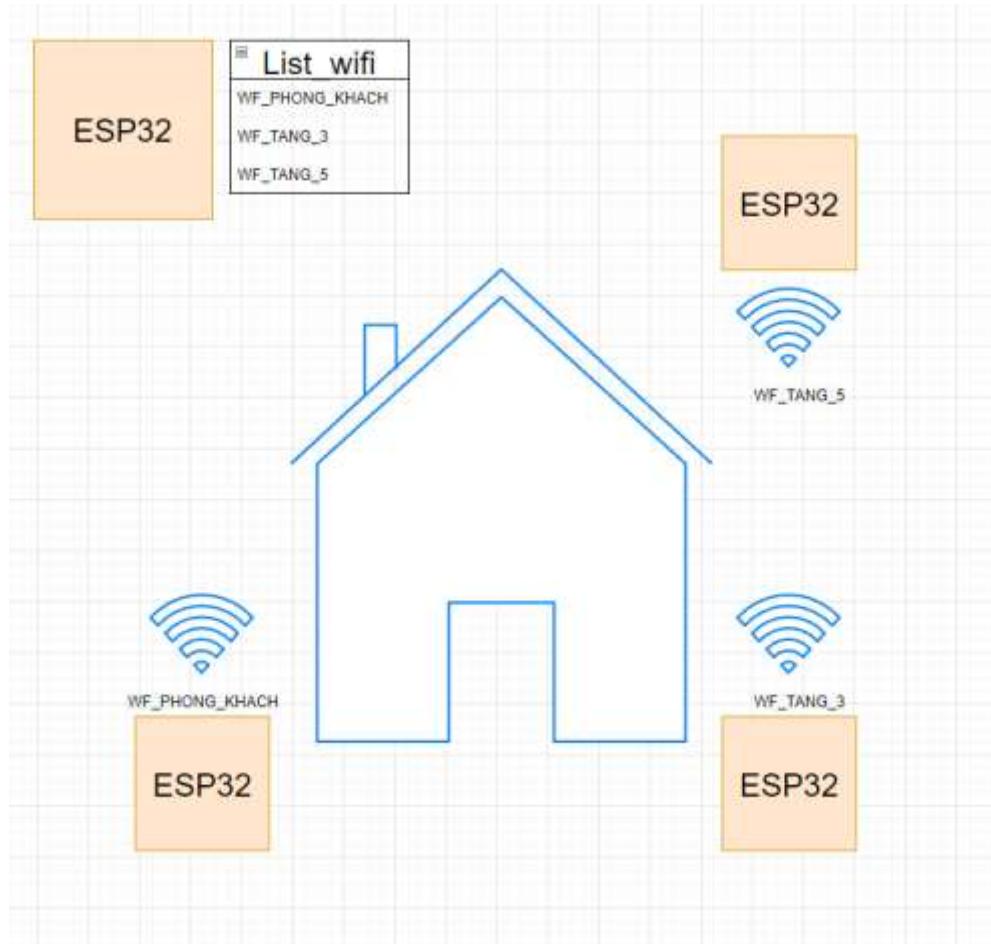
## Chương 1: Cơ sở lý thuyết

Bước 2: chọn Save ở Folder đã tạo ở bước cài đặt Arduino (của mình là G:\Arduino)

Bước 3: các bạn tắt đi rồi vào đúng folder để kiểm tra đã có file đã tạo chưa nhé?.

### 2.2 Multi WiFi:

Multi WiFi nói nôm na là các bạn sẽ cho ESP32 biết trước được một danh sách các wifi (gồm SSID – tên mạng, PASS – mật khẩu) để rồi ESP32 sẽ tự kết nối với danh sách ấy. Nếu truy cập hiện tại tự nhiên bị ngắt kết nối thì ESP32 sẽ tự tìm điểm truy cập mạng khác trong danh sách bạn đã cho trước mà tự động kết nối vào.



Hình 2.2 Multi WiFi

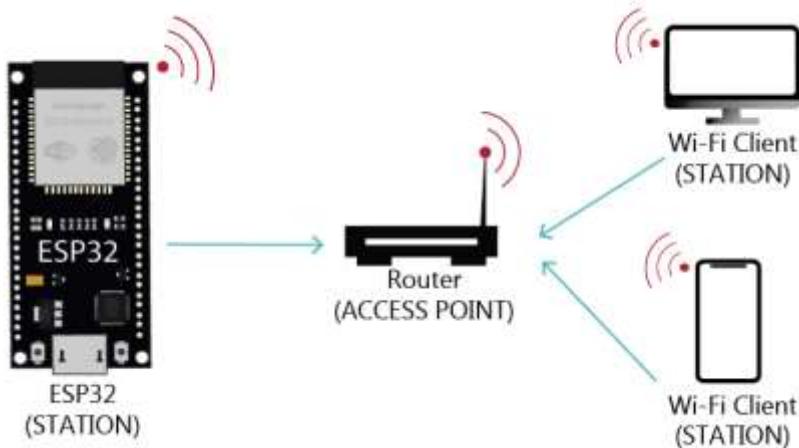
Giả sử nhà bạn có 3 Modem wifi được đặt tên (WF\_PHONG\_KHACH, WF\_TANG\_3, WF\_TANG\_5) bạn cung cấp cho ESP32 biết cả ba wifi đó rồi các bạn có thể mang con ESP này đi khắp căn nhà của bạn mà không lo rằng ESP32 sẽ bị mất kết nối mạng. Chức năng này khá giống với chức năng tự động kết nối mạng của điện thoại di động đây.

Về nguyên lý hoạt động của Multi WiFi là vậy giờ ta cùng bắt đầu lập trình nào.

### 2.2.1 WiFi:

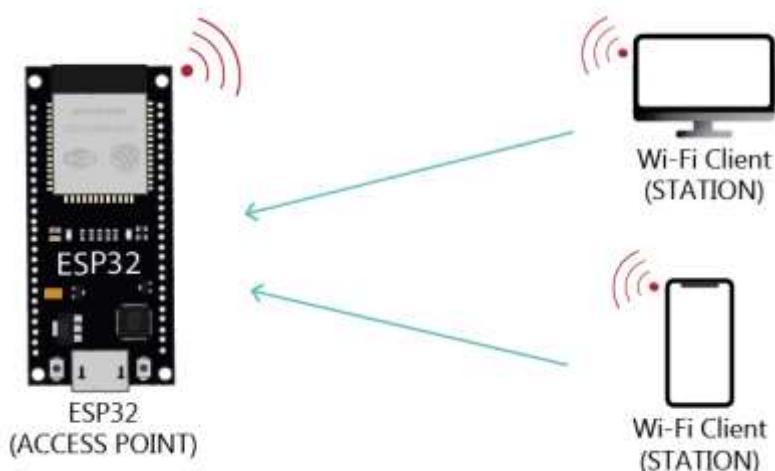
Wifi thì chắc các bạn đã biết nhưng ở ESP32 WiFi sẽ có ba mode:

- WIFI\_STA (Station) – kết nối đến một Access Point (điểm truy cập mạng).



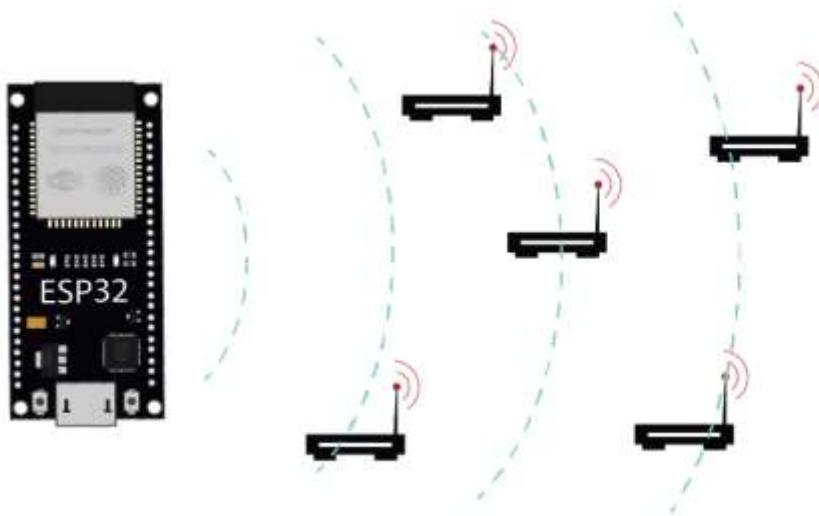
Hình 2.3 STA (Station)

- WIFI\_AP (Access Point) tự làm một điểm truy cập mạng



Hình 2.4 AP (Access Point)

- WIFI\_STA\_AP – vừa làm điểm truy cập, vừa truy cập một điểm khác.



Hình 2.5 STA\_AP (Stasion – Access Point)

Ta có bảng công dụng sau:

STA	AP	STA_AP
- Được sử dụng nhiều nhất. Và đây cũng là chế độ mặc định của ESP	- ESP32 thường dùng chế độ này khi làm server để lưu trữ dữ liệu, hay lấy dữ liệu của các thiết bị khác.	- ESP32 thường dùng chế độ này khi muốn tìm các mạng xung quanh. Rồi sau đó kết nối với mạng vừa tìm kiếm.

Các thành phần khác liên quan đến WiFi như: Wifi connection Status, Wifi Event hay Wifi connection Strength,... . Thì các bạn lên mạng tìm hiểu thêm nhé.

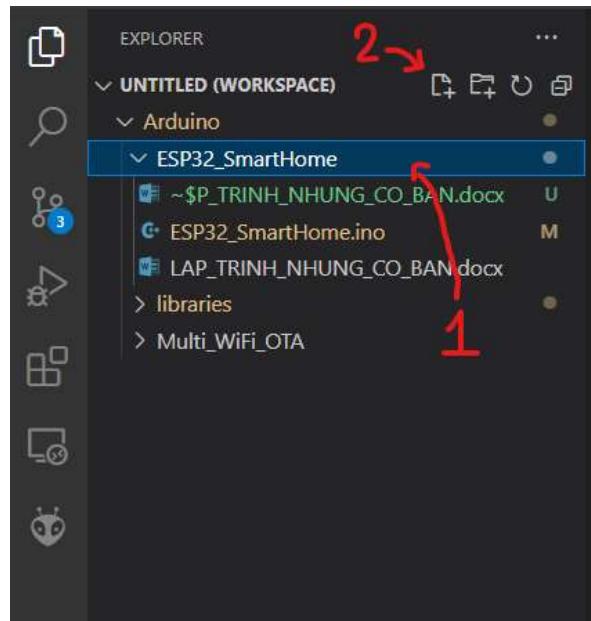
### 2.2.2 Cá nhân hóa thư viện Multi Wifi:

Về cơ bản thì Arduino IDE đã có thư viện MultiWiFi cho ESP32 nhưng việc sử dụng hơi phức tạp và chưa được cá nhân hóa nên việc sử dụng khá rườm rà. Chính vì vậy phải cá nhân hóa thư viện để tiện sử dụng sau này. Chúng ta sẽ tạo một thư mục, nơi này sẽ chứa các thư viện mà ta đã cá nhân hóa (nhưng đó là sau này ☺). Còn giờ thì để dễ dàng cho việc fix bug thì các thư viện cá nhân hóa này mình sẽ để chung với project Smart Home luôn.

Để tạo thư viện mới thì các bạn thực hiện các bước sau:

Bước 1: Mở VSC chọn File > Add Folder to Workspace rồi chọn Folder bừa bạn đã tạo ở bước Cài đặt Arduino IDE (Như mình là **G:\Arduino**)

Bước 2: Sau khi đã mở được Folder các bạn nhấn vào Folder **ESP\_SmartHome** rồi chọn biểu tượng thêm file. Sau đó các bạn đặt tên **<tên thư viện>.h** (của mình đặt tên là **my\_multi\_wifi.h**)



Hình 2.6 tạo file thư viện

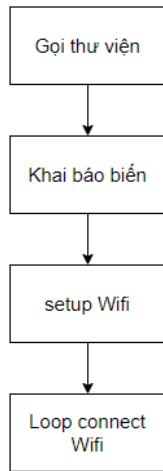
Như vậy ta đã tạo được file thư viện

Trước khi vào code thì mình có vài lời khuyên cho các bạn. Và đây là kinh nghiệm bản thân nên có thể nó vẫn chưa hoàn thiện. Nên bạn có thể áp dụng chúng nếu thấy hay. Hihi 😊

Để giúp bạn tránh tấu hỏa nhập ma khi đọc lại code của chính mình thì hãy cố gắng lưu ý vài điểm này nhé:

- Hạn chế tối đa việc sử dụng for lồng for hay if lồng if.
- Hãy bỏ dần if else mà thay vì thế hãy dùng if return.
- Một hàm thì chỉ nên làm một nhiệm vụ duy nhất.
- Đặt tên hàm hay tên biến thì hãy đặt đúng với nhiệm vụ của chúng (đừng đặt biến tên chung chung như a,b,c,m,n,...).
- À mà i và j trong hàm for là một ngoại lệ nhé.

a) Hướng đi code:



Hình 2.8 Các bước thực hiện

- Gọi thư viện: Thêm các thư viện mà ESP đã hỗ trợ.
- Khai báo biến: Khai báo các biến cục bộ và cả biến global (bao gồm cả các hàm)
- Setup wifi: init mode wifi (khởi tạo chế độ wifi cho esp)
- Loop connect wifi: luôn kiểm tra tình trạng kết nối mạng.

b) Giải thích code:

**Bước 1:** Để viết cá nhân hóa thư viện hay bắt kể tự viết một thư viện nào khác thì ta phải khai báo tag của file (tên thư viện) hay nói cách khác là định nghĩa tên thư viện. (Như mình ở đây, tag thư viện của là \_\_MY\_MULTI\_WIFI\_H\_\_) đặt tag giống như tên thư viện để tranh nhầm lẫn các thư viện khác khi khai báo.

```
#ifndef __MY_MULTI_WIFI_H__  
#define __MY_MULTI_WIFI_H__  
  
#endif
```

Hình 2.9 Giải thích code

Hàm điều kiện **#ifndef (if not define) #endif** – hiểu nôm na rằng là nếu như trước đó thư viện **my\_multi\_wifi.h** chưa được định nghĩa thì mới được định nghĩa **#define**. Như vậy sau khi bạn đã định nghĩa tên thư viện như trên hình thì bạn có thể thêm thư viện **#include “my\_multi\_wifi.h”** ở mọi file mà không sợ bị báo lỗi trùng lặp thư viện.

**Bước 2:** Khai báo các thư viện cần sử dụng để có thể setup Multi Wifi. Ở đây mình khai báo hai thư viện (**<WiFi.h>** và **<WiFiMulti.h>**)

## Chương 1: Cơ sở lý thuyết

```
// ##### Include Librarys #####
#include <WiFi.h>
#include <WiFiMulti.h>
// #####
```

Hình 2.10 Khai báo thư viện

### Bước 3: Khai báo biến sử dụng trong thư viện

```
// ##### Define Variables #####
#ifndef PIN_STATUS_LED
#define PIN_STATUS_LED 2
#endif

#ifndef NUM_WIFI
#define NUM_WIFI 2
#endif

#ifndef DELAY_TIME_LED_STT
#define DELAY_TIME_LED_STT 2000
#endif

const char *wifi_ssid[]{"4G-UFI-E701", "Sophie"};
const char *wifi_pw[] = {"aaaaaaaaaa", "aaaaaaaaaa"};

unsigned long current_time_in_wifi = 0;
unsigned long old_time_in_wifi = 0;
// #####
```

Hình 2.10 Khai báo biến

Như các bạn có thể thấy ở trên hình thì có 2 kiểu khai báo biến. Nói chính xác hơn thì một cái là định nghĩa tag và một cái là khai báo biến. Tag này ta cũng có thể sử dụng như một biến thông thường.

```
#ifndef NUM_WIFI
#define NUM_WIFI 2
#endif
```

```
unsigned long current_time_in_wifi = 0;
unsigned long old_time_in_wifi = 0;
```

Hình 2.11 bên phải là khai báo biến, bên trái là định nghĩa tag

Hai cách khai báo này đều có thể dùng global nhưng để phân biệt rõ ràng và tránh nhầm lẫn lung tung thì **tag** các bạn sẽ sử dụng cho global (sau này sẽ được định nghĩa ở file **configs.h**) còn biến thì sử dụng trong cục bộ.

## Chương 1: Cơ sở lý thuyết

Ưu điểm của tag là các bạn có thể #define nhiều lần mà không bị ảnh hưởng gì đến code. ESP sẽ sử dụng cái tag đầu tiên mà bạn đã định nghĩa, nếu như không có định nghĩa nào về cái tag đó thì ESP sẽ lấy giá trị mặc định mà bạn đã **#ifndef**.

Ở đây mình định nghĩa như sau:

- Định nghĩa chân GPIO 2 làm chân đèn tín hiệu (cho tag **PIN\_STATUS\_LED**). Để thông báo đã kết nối WiFi.
- Định nghĩa số lượng mạng (cho tag **NUM\_WIFI**)
- Định nghĩa thời gian đèn nháy (cho tag **LED\_STT\_DELAY\_TIME**)
- Khai báo mảng tên wifi và mảng mật khẩu tương ứng với tên wifi ấy. (**wifi\_ssid** và **wifi\_pw**)
- Khai báo biến thời gian hiện tại (**current\_time\_in\_wifi**)
- Khai báo biến thời gian hiện tại (**old\_time\_in\_wifi**)

**Bước 4:** Tạo các hàm với từng chức năng riêng (**một hàm chỉ nên làm một nhiệm vụ**) trong thư viện.

- **Hàm setup led trạng thái wifi:**

```
// ##### setup Status Led #####
void setupStatusLed()
{
    pinMode(PIN_STATUS_LED,OUTPUT);
    digitalWrite(PIN_STATUS_LED, LOW);
}
// #####
```

Hình 2.12 Khai báo hàm trạng thái wifi

Khai báo PIN\_STATUS\_LED là chân ngõ ra. Và tắt led.

- **Hàm hiển thị thông tin Wifi**

```
void showInforWifi()
{
    Serial.println("_____WIFI_____");
    Serial.println("WiFi connected");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
    Serial.print("SSID: ");
    Serial.println(WiFi.SSID());
    Serial.print("RSSI: ");
    Serial.println(WiFi.RSSI());
    Serial.println("_____");
}
```

## Chương 1: Cơ sở lý thuyết

Hình 2.13 Khai báo hàm thông tin wifi

In các thông tin wifi mà ESP32 đã kết nối thành công lên **Serial**

### - Hàm kết nối wifi

```
void connectWiFiMulti()
{
    digitalWrite(PIN_STATUS_LED, HIGH);
    Serial.println("Connecting Wifi...");
    if (wifiMulti.run() == WL_CONNECTED)
    {
        showInfoForWifi();
    }
    else
    {
        Serial.println("WiFi not connected!");
        digitalWrite(PIN_STATUS_LED, LOW);
    }
}
```

Hình 2.14 Khai báo kết nối wifi

Khi bắt đầu kết nối wifi, led trạng thái sẽ bắt đầu sáng. Nếu kết nối thành công thì hiển thị thông tin mà ESP đã kết nối được. Không kết nối được thì hiển thị “**Wifi not connected**”.

### - Hàm setup multi wifi

```
void setupWiFiMulti()
{
    // setup status led (ESP32 GPIO2)
    setupStatusLed();

    // setup mode WiFi
    WiFi.mode(WIFI_STA);

    // Disable WiFi
    if (NUM_WIFI <= 0)
        return;
    if (wifi_ssid.length() != NUM_WIFI || wifi_pw.length() != NUM_WIFI)
    {
        Serial.println("Invalid declaration. Check list wifi again!!!");
        return;
    }

    // add list wifi
    for (byte i = 0; i < NUM_WIFI; i++)
        wifiMulti.addAP(wifi_ssid[i], wifi_pw[i]);

    connectWiFiMulti();
}
```

Hình 2.15 Khai báo hàm setup multi wifi

## Chương 1: Cơ sở lý thuyết

Gọi hàm **setupStatusLed()**; để thiết lập led.

Sử dụng **Wifi.mode(<tên mode>)**; để khai báo mode wifi mà ESP sẽ sử dụng.

Sau đó kiểm tra số lượng kết nối, mảng tên wifi và mảng pass có đủ không? Nếu không thì không kết nối wifi và hiển thị lỗi khai báo số lượng lỗi (hình 2.16)

```
// Disable WiFi
if (NUM_WIFI <= 0)
    return;
if (wifi_ssid.length() != NUM_WIFI || wifi_ssid.length() != NUM_WIFI )
{
    Serial.println("Invalid declaration. Check list wifi again!!!");
    return;
}
```

Hình 2.16 Kiểm tra điều kiện

Nếu không lỗi thì thêm wifi vào danh sách kết nối bằng **wifiMulti.addAP(<tên mạng>,<mật khẩu>);**

Sau khi thêm danh sách thì gọi hàm **connectWiFiMulti();** để bắt đầu kết nối wifi (hình 2.17)

```
// add list wifi
for (byte i = 0; i < NUM_WIFI; i++)
    wifiMulti.addAP(wifi_ssid[i], wifi_pw[i]);

connectWiFiMulti();
```

Hình 2.17 thêm wifi vào danh sách kết nối.

- **Hàm loopWiFiMulti()**

## Chương 1: Cơ sở lý thuyết

```
void loopWiFiMulti()
{
    if (NUM_WIFI <= 0)
        return;

    // check connect. 1m/1time
    current_time_in_wifi = millis();
    if (current_time_in_wifi - old_time_in_wifi < LED_STT_DELAY_TIME * 1000)
        return;

    old_time_in_wifi = millis();
    if (wifiMulti.run() == WL_CONNECTED)
    {
        int stt = digitalRead(PIN_STATUS_LED);
        digitalWrite(PIN_STATUS_LED, !stt); // on ledstatus
        return;
    }

    // if Wifi disconnet, try to connect other
    Serial.println("Not connected to WiFi. Re-connecting now...");
    connectWiFiMulti();
}
```

Hình 2.18 luôn luôn kiểm tra trạng thái kết nối

Đầu tiên vẫn kiểm tra có danh sách wifi không?

Sau đó cứ 1p kiểm tra kết nối wifi 1 lần.

Và thấy đèn nhấp nháy tức là wifi còn kết nối. nếu không thì ESP sẽ tự kết nối lại wifi.

\*\* Kết quả:

```
13:36:52.942 -> Connecting Wifi...
13:36:59.158 -> _____WIFI_____
13:36:59.158 -> WiFi connected
13:36:59.158 -> IP address: 192.168.1.13
13:36:59.158 -> SSID: THENAM
13:36:59.158 -> RSSI: -74
13:36:59.158 -> _____
```

Hình 2.25 Kết quả nếu kết nối thành công

```
13:31:21.331 -> Connecting Wifi...
13:31:27.022 -> WiFi not connected!
13:32:24.416 -> Not connected to WiFi. Re-connecting now...
13:32:24.416 -> Connecting Wifi...
13:32:30.316 -> WiFi not connected!
```

Hình 2.26 Kết quả nếu kết nối thất bại

c) Full Code library my\_wifi\_multi.h và chạy thử thư viện

```
#ifndef __MY_MULTI_WIFI_H__
#define __MY_MULTI_WIFI_H__

// ##### Include Librarys #####
#include <WiFi.h>
#include <WiFiMulti.h>
// #####
// #####
// ##### Define Librarys #####
WiFiMulti wifiMulti;
// #####
```

**Note: để lấy full code nhấn vào ô chứa code rồi Ctrl + C. Vào file my\_multi\_wifi.h trong Arduino rồi Ctrl + V.**

- Test thư viện có hoạt động hay không thì các bạn mở file <tên project>.ino (của mình là **ESP32\_SmartHome.ino**)

```
#include "my_multi_wifi.h"

void setup() {
    Serial.begin(115200);
    setupWiFiMulti();
}

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

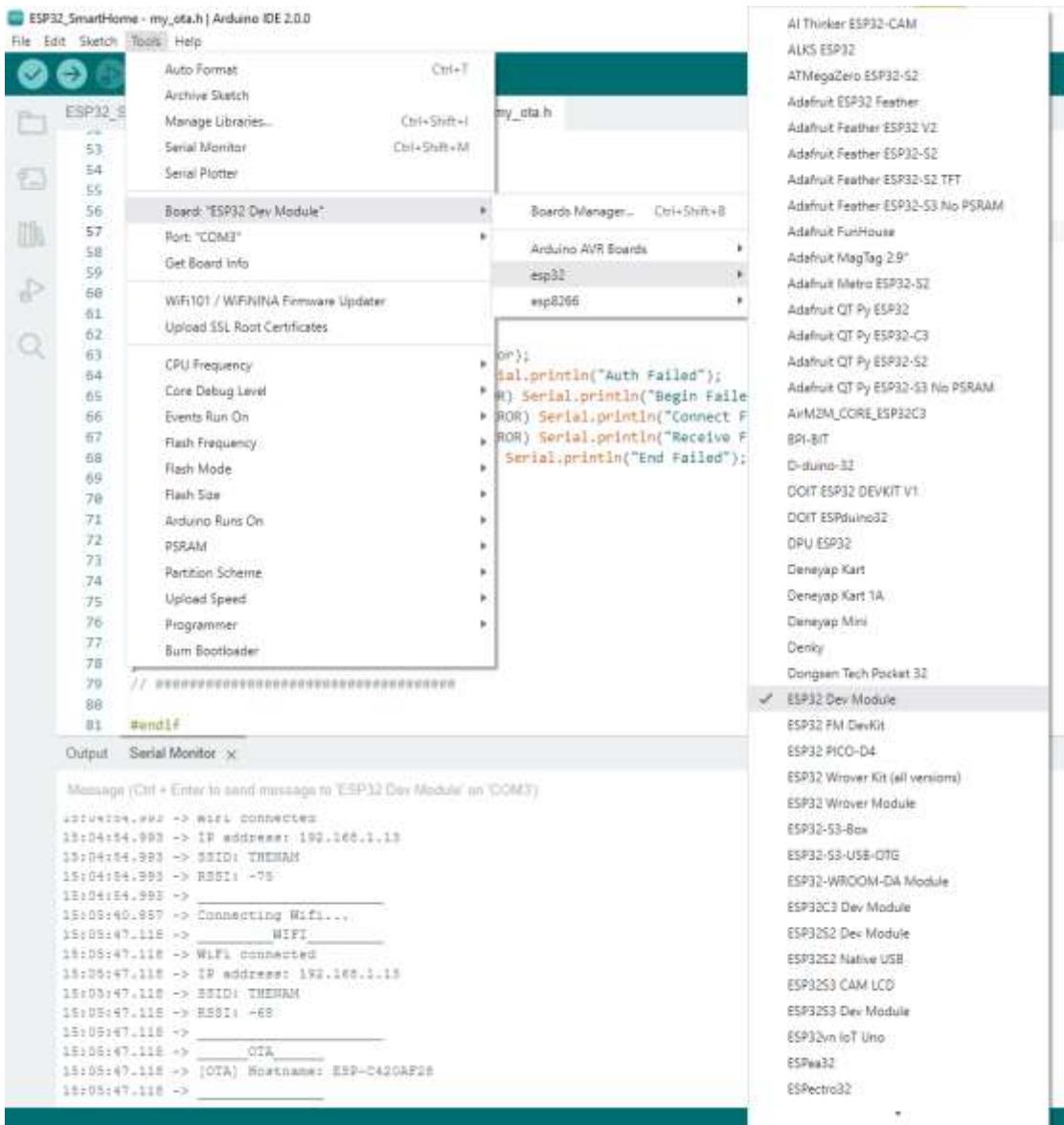
Hình 2.19 file **ESP32\_SmartHome.ino**

Bước 1: Khai báo thư viện my\_multi\_wifi.h bằng lệnh #include “my\_multi\_wifi.h”

Bước 2: Sau đó gọi hàm **setupWiFiMulti();** vào trong hàm **setup()** tổng. Và gọi hàm **loopWiFiMulti();** vào hàm **loop()** tổng

Bước 3: Chọn **Tools > Board > esp32 > ESP32 Dev Module**

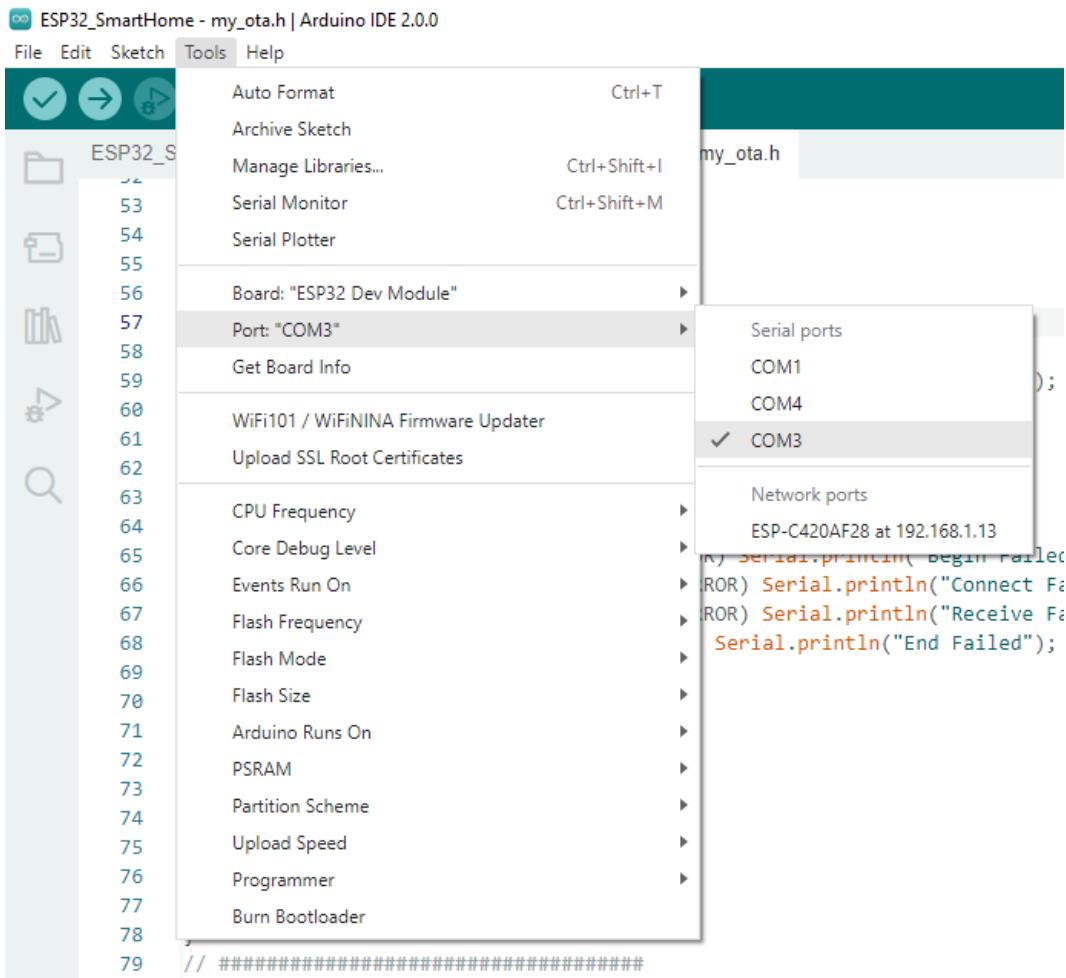
## Chương 1: Cơ sở lý thuyết



Hình 2.19 Chọn board để nạp code

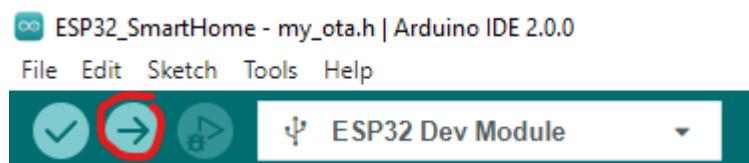
Bước 4: Chọn **Tools > Port > COM ?** (ở dây chip mình kết nối COM3)

## Chương 1: Cơ sở lý thuyết



Hình 2.20 Chọn port để nạp code

Bước 5: Nhấn vào biểu tượng mũi tên để nạp code



Hình 2.21 nạp code

Bước 6: Chờ đến khi nào Verify xuất hiện hình bên dưới thì ấn nút BOOT của chíp

The screenshot shows the Serial Monitor window with the output tab selected. It displays the following text:  
Sketch uses 696129 bytes (53%) of program storage space. Maximum is 1310720 bytes.  
Global variables use 41424 bytes (12%) of dynamic memory, leaving 286256 bytes for local variables. Maximum is 327680 bytes.  
esptool.py v3.3  
Serial port COM3  
Connecting.....

Hình 2.22 Hãy nhấn nút BOOT khi thấy dòng connecting.....

## Chương 1: Cơ sở lý thuyết

Bước 7: Nếu như (hình 2.23) tức là các bạn đã nạp xong code rồi đó

```
Output  Serial Monitor
Writing at 0x00093984... (0%)
Writing at 0x0007ecd2... (67%)
Writing at 0x0008423d... (71%)
Writing at 0x00089b8d... (75%)
Writing at 0x0008f9d6... (78%)
Writing at 0x0009522e... (82%)
Writing at 0x0009d6a7... (85%)
Writing at 0x000a5f20... (89%)
Writing at 0x000aaf53... (92%)
Writing at 0x000b0c7c... (96%)
Writing at 0x000b601c... (100%)
Wrote 701904 bytes (457507 compressed) at 0x00010000 in 6.9 seconds (effective 811.6 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

Hình 2.23 Hoàn thành nạp code

Bước 8: Nhấn vào biểu tượng kính lúp để mở màn hình Serial để xem kết quả



Hình 2.24 Mở Serial

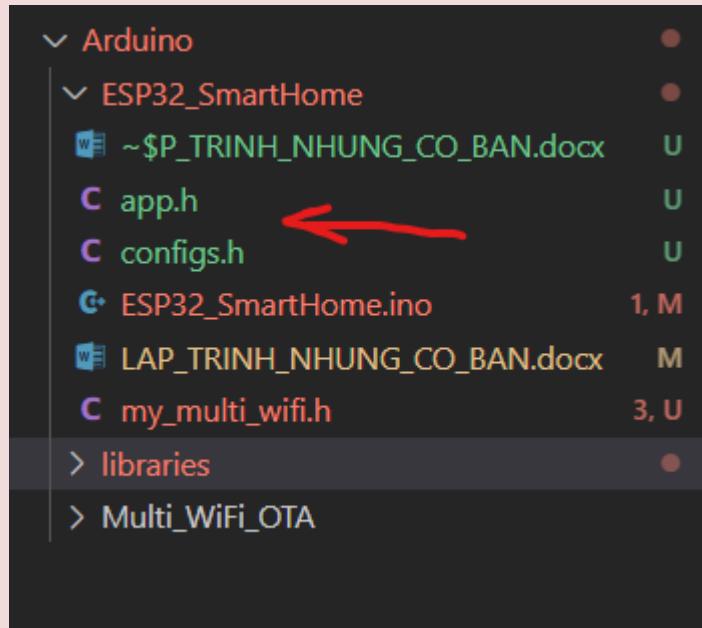
Note : nếu mở Serial nhưng không thấy gì hãy nhấn reset chip bằng cách ấn vào nút EN trên chip

## Chương 1: Cơ sở lý thuyết

Và trước khi vào phần mới thì các bạn tạo thêm cho mình 2 file vào trong folder của project (tên là app.h và configs.h)

Một project chuẩn thì cần có 3 file:

- 1 file **configs.h** (file định nghĩa các tag)
- 1 file **app.h** (file chứa các hàm ứng dụng cho project)
- 1 file **<Tên project>.ino** (file để Arduino có thể biên dịch code – file này khi save thì arduino đã tự tạo cho các bạn rồi)



### 2.3 OTA:

#### 2.3.1 OTA là gì?

OTA (Over The Air) cho phép bạn cập nhật một firmware vào ESP bằng Wifi thay vì phải cắm USB để cập nhật firmware

Ở ESP sẽ có 2 loại OTA:

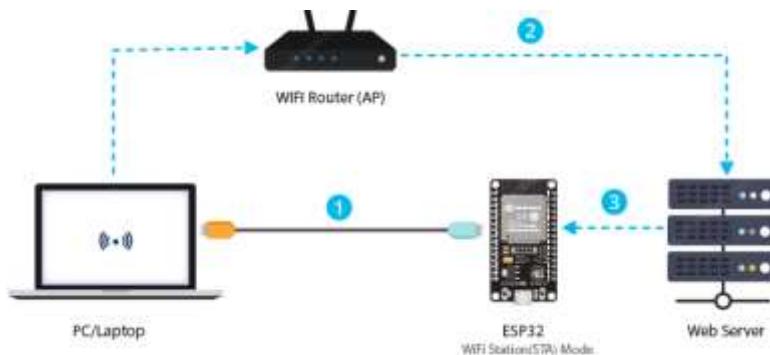
- Basic OTA: cập nhật firmware cho một thiết bị thông qua Arduino IDE.



Hình 2.27 Basic OTA

Basic OTA: là ESP và máy tính sẽ cùng kết nối với một wifi. Sau đó ta sẽ nạp bản cập nhật mới nhất vào ESP thông qua Arduino IDE.

- Server OTA: các thiết bị ESP sẽ tự động kết nối với Server, kiểm tra và tải phiên bản firmware mới nhất về chip.



Hình 2.28 Server OTA

Server OTA: là ta sẽ Export compiled binary (có đuôi .bin) file code rồi sau đó đẩy lên server bằng wifi. Server sẽ tự động đẩy bản cập nhật mới nhất xuống ESP hoặc có thể ESP sẽ tự động truy cập và server để lấy bản cập nhật mới nhất về

**Và ở phần OTA này mình chỉ hướng dẫn các bạn Basic OTA thôi. Còn Server OTA thì các bạn tự tìm hiểu thêm nhé.**

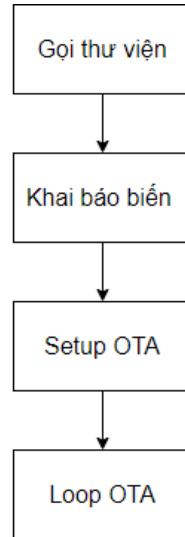
**Giờ thì bắt đầu code thôi. !!!**

### 2.3.2 Cá nhân hóa thư viện OTA

Cách tạo thư viện thì mình đã hướng dẫn ở khá chi tiết ở phần **Cá nhân hóa thư viện Multi Wifi** rồi nên các bạn chưa rõ có thể mwor lịa để xem nhé.

**Ở thư viện này mình đặt tên là my\_ota.h**

a) Hướng đi code:



Hình 2.28 Server OTA

- Gọi thư viện: Thêm các thư viện mà ESP và Arduino đã hỗ trợ cho OTA.
- Khai báo biến: Khai báo các biến cục bộ và cả biến global (bao gồm cả các hàm)
- Setup OTA: init mode wifi (khởi tạo chế độ wifi cho esp)
- Loop OTA: luôn giữ OTA hoạt động

b) Giải thích code:

**Bước 1:** Vẫn như thế để viết cá nhân hóa thư viện ta phải là định nghĩa tên thư viện. (Như mình ở đây, tag thư viện của là `_MY_OTA_H_`) đặt tag giống như tên thư viện để tránh nhầm lẫn các thư viện khác khi khai báo.

```
1  #ifndef _MY_OTA_H_
2  #define _MY_OTA_H_
3
4  #endif
```

Hình 2.29 Khai báo tag thư viện

Mục đích để không sợ bị báo lỗi trùng lặp thư viện.

**Bước 2:** Khai báo các thư viện cần sử dụng để có thể setup Basic OTA. Ở đây mình khai báo hai thư viện (`<ArduinoOTA.h>` và `<WiFiUdp.h>`)

```
// ##### Include Librarys #####
#include <ArduinoOTA.h>
#include <WiFiUdp.h>
// #####
```

Hình 2.30 Khai báo thư viện

**Bước 3:** Khai báo biến sử dụng trong thư viện

```
// ##### Define Variables #####
#ifndef OTA_HOSTNAME
| | #define OTA_HOSTNAME "ESP-%06X"
#endif
// #####
```

Hình 2.31 Khai báo biến

Ở đây mình định nghĩa như sau:

- Định nghĩa tên chip để có thể phân biệt được khi nạp OTA qua wifi (cho tag **OTA\_HOSTNAME**)

**Bước 4:** Tạo các hàm với từng chức năng riêng (**một hàm chỉ nên làm một nhiệm vụ**) trong thư viện.

- **Hàm lấy ID của chip:**

```
uint64_t getChipId()
{
    uint64_t chipid = ESP.getEfuseMac();
    uint32_t discard_two_bytes = (uint32_t)(chipid >> 16);
    return discard_two_bytes;
}
```

Hình 2.32 lấy 6 mã cuối để làm ID

Vì mỗi chip có 1 ID riêng biệt nên ta sẽ lấy 6 mã cuối để làm tên hostname cho chip. Hàm `ESP.getEfuseMac()`; là hàm để lấy địa MAC của thiết bị.

Còn biến `discard_two_bytes` để lấy 6 ký tự cuối cùng của địa chỉ MAC.

- **Hàm hiển thị thông tin OTA**

## Chương 1: Cơ sở lý thuyết

```
void showInforOTA(){
    // Port defaults to 3232
    // ArduinoOTA.setPort(3232);

    // Hostname defaults is: esp3232-[MAC]
    char hostname[24];
    sprintf(hostname, OTA_HOSTNAME, getChipId());

    ArduinoOTA.setHostname(hostname);
    Serial.println("____OTA____");
    Serial.print("[OTA] Hostname: ");
    Serial.println(hostname);
    Serial.println("_____");
}
```

Hình 2.33 Khai báo hàm hiển thị thông tin OTA

### In các thông tin OTA

#### - Hàm setup OTA

```
void setupOTA(){  
  
    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);
        })  
  
        .onEnd([]()){
            Serial.println("\nEnd");
        })  
  
        .onProgress([](unsigned int progress, unsigned int total){
            Serial.printf("Progress: %u%\r", (progress / (total / 100)));
        })  
  
        .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");
        });
    }  
  
    //show information OTA and init OTA
    showInforOTA();
    ArduinoOTA.begin();
}
```

Hình 2.34 Khai báo hàm setup OTA

## Chương 1: Cơ sở lý thuyết

Basic OTA là cập nhật firmware về ESP bằng Arduino IDE nên ta sử dụng ArduinoOTA trong thư viện để khai báo các trạng thái khi kết nối OTA.

Khởi động OTA thì Arduino sẽ thực hiện các lệnh trong **.onStart** (hình 2.35)

```
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);
}
```

Hình 2.35 .onStart

Kết thúc OTA thì Arduino sẽ thực hiện các lệnh trong **.onEnd** (hình 2.36)

```
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);
}
```

Hình 2.36 .onEnd

Tải firmware về chip thì Arduino sẽ thực hiện các lệnh trong **.onProgress** (hình 2.37)

```
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);
}
```

Hình 2.37 .onProgress

OTA bị lỗi thì Arduino sẽ thực hiện các lệnh trong **.onError** (hình 2.38)

## Chương 1: Cơ sở lý thuyết

```
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);
  })
```

Hình 2.38 .onError

Sau đó gọi hàm **showInforOTA()**; để hiển thị và khởi động OTA bằng **ArduinoOTA.begin();**

```
//show information OTA and init OTA
showInforOTA();
ArduinoOTA.begin();
```

Hình 2.39 hiển thị thông tin và khởi tạo OTA

### - Hàm loopOTA

```
void loopOTA(){
  ArduinoOTA.handle();
}
```

Hình 2.40 Khai báo hàm loop OTA

Chỉ cần sử dụng hàm **ArduinoOTA.handle();** để giữ trạng thái có thể cập nhật firmware bằng OTA bất cứ lúc nào.

### c) Full Code library my\_ota.h và chạy thử thư viện

```
#ifndef __MY_OTA_H__
#define __MY_OTA_H__

// ##### Include Librarys #####
#include <WiFi.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
// ###### Define Variables #####
#ifndef OTA_HOSTNAME
```

**Note: để lấy full code nhấn vào ô chứa code rồi Ctrl + C. Vào file my\_ota.h trong Arduino rồi Ctrl + V.**

- Test thư viện có hoạt động hay không thì các bạn mở file <tên project>.ino (của mình là **ESP32\_SmartHome.ino**)

```
#include "my_multi_wifi.h"
#include "my_ota.h"

void setup() {
    Serial.begin(115200);
    setupWiFiMulti();
    delay(50);
    setupOTA();
}

void loop() {
    loopWiFiMulti();
    loopOTA();
}
```

Hình 2.41 file **ESP32\_SmartHome.ino**

Bước 1: Khai báo thêm thư viện my\_ota.h bằng lệnh #include “my\_ota.h”

Bước 2: Sau đó gọi thêm hàm **setupOTA()**; đặt sau hàm **setupWiFiMulti()** và cả hai cùng trong hàm **setup()** tổng. Và gọi thêm hàm **loopOTA()**; vào hàm **loop()** tổng

Bước 3: Nạp code (Mình đã hướng dẫn khá chi tiết ở phần c) Full code library my\_multi\_wifi.h rồi nên bạn có thể lên lại để xem nhé)

**Note : nếu mở Serial nhưng không thấy gì hãy nhấn reset chip bằng cách ấn vào nút EN trên chip**

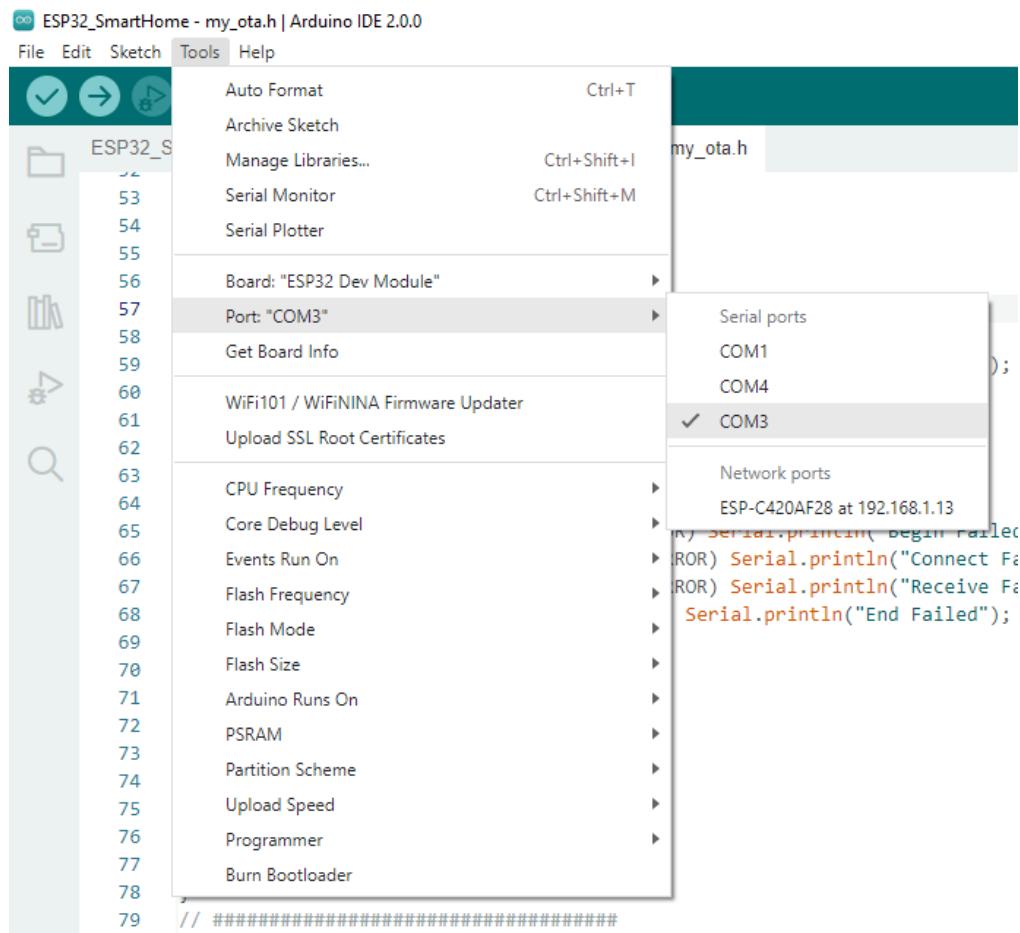
\*\* Kết quả:

## Chương 1: Cơ sở lý thuyết

```
16:43:21.790 -> Connecting Wifi...
16:43:28.064 -> _____ WIFI _____
16:43:28.064 -> WiFi connected
16:43:28.108 -> IP address: 192.168.1.13
16:43:28.108 -> SSID: THENAM
16:43:28.108 -> RSSI: -72
16:43:28.108 ->
16:43:28.155 -> _____ OTA _____
16:43:28.155 -> [OTA] Hostname: ESP-C420AF28
16:43:28.155 ->
```

Hình 2.42 Kết quả

Bước 4: kiểm tra nạp OTA. Trước tiên hãy chờ khoảng 30s để ESP setupOTA rồi Chọn Tools > port > **ESP-<id chip> at <ip chip>** (của mình là **ESP-C420AF28 at 192.168.1.13**)



Hình 2.43 Network ports

Bước 5: ấn mũi tên nạp code. Rồi đợi đến khi nào load xong là dc.

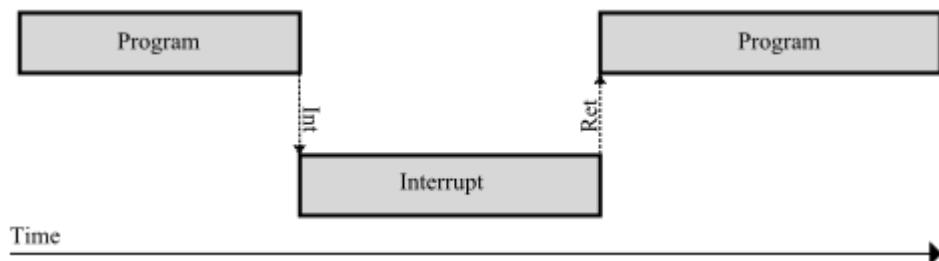
### Note: Nạp code bằng OTA

- Nhược điểm là kết nối mạng của bạn chập chờn hay yếu thì việc nạp code bằng OTA này rất dễ hỏng giữa chừng. Mà nạp bằng OTA giữa chừng mà bị thất bại thì bắt buộc bạn phải nạp lại code bằng USB.

#### 2.4 Interrup:

##### 2.4.1 Interrup là gì?

Interrup hiểu nôm na là các sự kiện ngẫu nhiên làm gián đoạn quá trình của một sự kiện đang xảy ra. Để có thể dễ hiểu khái niệm mới này ta cùng đưa ra một ví dụ trong thực tế như sau: Bạn đang làm bài thì mẹ bạn gọi bạn xuống bếp cắm nồi cơm. Như vậy sự kiện ngẫu nhiên ở đây là **mẹ bạn gọi**, sự kiện đang xảy ra ở đây là **Bạn đang làm bài**.



Hình 2.44 Ngắt

Mỗi ngắt đều có cấu hình độ ưu tiên nhất định. Có thể chia làm 2 loại ngắt:

- Hardware Interrupts: (Ngắt cứng) – Được kích hoạt từ các sự kiện bên ngoài, ví dụ: GPIO (khi nhấn phím), Touch (khi phát hiện chạm).
- Software Interrupts: (Ngắt mềm) – Được kích hoạt từ các sự kiện bên trong, ví dụ: Timer, Watchdog(khi tràn bộ đếm thời gian).

Ở đây mình sẽ hướng dẫn các bạn làm về ngắt cứng (ngắt ngoài)

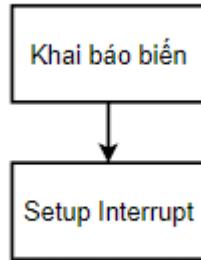
Giờ thì vào code nào. ☺

##### 2.4.2 Cá nhân hóa thư viện Interrup

Cách tạo thư viện thì mình đã hướng dẫn ở khá chi tiết ở phần **Cá nhân hóa thư viện Multi Wifi** rồi nên các bạn chưa rõ có thể mở lại để xem nhé.

Ở thư viện này mình đặt tên là **my\_interrupt.h**

a) Hướng đi code:



Hình 2.45 hướng đi code

- Khai báo biến: Khai báo các biến cục bộ và cả biến global (bao gồm cả các hàm)
- Setup Interrupt: thiết lập các chân ngắt ngoài.

b) Giải thích code:

**Bước 1:** Vẫn như thế để viết cá nhân hóa thư viện ta phải là định nghĩa tên thư viện. (Như mình ở đây, tag thư viện của là `__MY_INTERRUPT_H__`) đặt tag giống như tên thư viện để tránh nhầm lẫn các thư viện khác khi khai báo.

```
#ifndef __MY_INTERRUPT_H__
#define __MY_INTERRUPT_H__

#endif
```

Hình 2.46 Khai báo tag thư viện

Mục đích để không sợ bị báo lỗi trùng lặp thư viện.

**Bước 2:** Khai báo biến sử dụng trong thư viện

```
#ifndef NUM_PIN_INTERRUPT
| | #define NUM_PIN_INTERRUPT    1
#endif

#ifndef EN_INTERRUPT
| | #define EN_INTERRUPT        false
#endif

#ifndef MODE_INTERRUPT
| | #define MODE_INTERRUPT      RISING
#endif

//function will be write in app.h
void IRAM_ATTR callbackInterrupt();
```

Hình 2.47 Khai báo biến

Ở đây mình định nghĩa như sau:

- Định nghĩa số lượng chân ngắt ngoài (cho tag **NUM\_PIN\_INTERRUPT**)
- Định nghĩa có sử dụng Interrupt không (cho tag **EN\_INTERRUPT**)
- Định nghĩa mode event (cho tag **MODE\_INTERRUPT**)
- Định nghĩa hàm callback khi có sự kiện ngắt xuất hiện (cho hàm **callbackInterrupt**). Ta phải có thêm một hành động là gắn cờ **IRAM\_ATTR** vào trước hàm ngắt theo quy định của espresif.

Note: ta chỉ khai báo tên hàm thôi còn code thì ta sẽ viết ở **app.h** để có thể tùy chỉnh các nhiệm vụ khác nhau tùy thuộc vào project của ta là gì.

**Bước 3:** Tạo các hàm với từng chức năng riêng (**một hàm chỉ nên làm một nhiệm vụ**) trong thư viện.

- **Hàm khai báo các chân interrupt:**

```
void defineIOInterrupt()
{
    for (int i = 0; i < NUM_PIN_INTERRUPT; i++)
    {
        pinMode(list_io_interrupt[i], INPUT);
        attachInterrupt(digitalPinToInterrupt(list_io_interrupt[i]), callbackInterrupt, MODE_INTERRUPT);
        Serial.println("IO" + String(list_io_interrupt[i]) + " has become interruptIO");
        delay(10);
    }
}
```

Hình 2.48 khai báo hàm

**pinMode(<GPIO>, <INPUT>);** là hàm để định nghĩa rằng <GPIO> là chân ngõ vào hay là chân ngõ ra

**attachInterrupt(digitalPinToInterrupt(<GPIO>), <callback>, <mode>);** là hàm khai báo <pin> là chân ngắt với chế độ phát hiện ngắt là <mode> và sau đó vào hàm callback thực hiện lệnh.

<mode> có năm chế độ phát hiện ngắt:

- + **LOW:** kích hoạt ngắt bất cứ lúc nào khi chân khai báo ở mức 0 (low)
- + **HIGH:** kích hoạt ngắt bất cứ lúc nào khi chân khai báo ở mức 1 (high)
- + **CHANGE:** kích hoạt ngắt bất cứ lúc nào khi chân khai báo ở mức 0 (low) lên mức 1 (high) hay ở mức 1 (high) xuống mức 0 (low). (Nếu chọn ở chế độ này thì sẽ gặp trường hợp 2 lần vào ngắt khi mà bạn sử dụng nút nhấn)
- + **RISING:** kích hoạt ngắt bất cứ lúc nào khi chân khai báo ở mức 0 (low) lên mức 1 (high)
- + **FALLING:** kích hoạt ngắt bất cứ lúc nào khi chân khai báo ở mức 1 (high) xuống mức 0 (low)



Hình 2.49 các sự kiện ngắt

- **Hàm setup Interrupt**

```
void setupInterrupt()
{
    if(!EN_INTERRUPT || NUM_PIN_INTERRUPT <= 0)
        return;

    Serial.println("=====INTERRUPT=====");
    defineIOInterrupt();
    Serial.println("Setup interrupt done !");
    Serial.println("=====-----");
}
```

Hình 2.50 Khai báo hàm setup interrupt

Về cơ bản hàm setup này chỉ để kiểm tra các điều kiện cho đúng.

Kiểm tra xem interrupt có đc kích hoạt hay không và kiểm tra số lượng chân khai báo có ít hơn 1 hay không (hình 2.51)

```
if(!EN_INTERRUPT || NUM_PIN_INTERRUPT <= 0)
    return;
```

Hình 2.51 kiều tra interrupt có được kích hoạt và kiểm tra lỗi danh sách

Nếu thỏa điều kiện thì gọi hàm defineIOInterrupt();

### c) Full Code library my\_interrupt.h và chạy thử thư viện

\*\* full code **my\_interrupt.h**

```
#ifndef __MY_INTERRUPT_H__
#define __MY_INTERRUPT_H__

//##### define tag #####
#ifndef NUM_PIN_INTERRUPT
#define NUM_PIN_INTERRUPT    1
#endif

#ifndef EN_INTERRUPT
#define EN_INTERRUPT         false
#endif
```

**Note: để lấy full code nhấn vào ô chứa code rồi Ctrl + C. Vào file my\_ota.h trong Arduino rồi Ctrl + V.**

- Test thư viện có hoạt động hay không thì các bạn mở file <tên project>.ino (của mình là **ESP32\_SmartHome.ino**)

\*\* full code **ESP32\_SmartHome.ino**

```
#define EN_INTERRUPT          true      // true is enable interrupt
#define MODE_INTERRUPT        RISING
#define NUM_PIN_INTERRUPT     1         // NUM_PIN_INTERRUPT ==
int list_io_interrupt.length()           // change pin if chose multi
```

```
#define EN_INTERRUPT          true      // true is enable interrupt
#define MODE_INTERRUPT        RISING
#define NUM_PIN_INTERRUPT     1         // NUM_PIN_INTERRUPT == list_io_interrupt.length()
int list_io_interrupt[] = {21};           // change pin if chose multi

#include "my_interrupt.h"

void IRAM_ATTR callbackInterrupt()
{
    | Serial.println("set up interrupt successfully");
}

void setup()
{
    | Serial.begin(115200);
    | setupInterrupt();
}

void loop()
{
    delay(50);
}
```

Hình 2.52 file **ESP32\_SmartHome.ino**

## Chương 1: Cơ sở lý thuyết

Bước 1: định nghĩa lại các tag. Tag **EN\_INTERRUPT** là **true** để kích hoạt ngắt. Tag **NUM\_PIN\_INTERRUPT** phải bằng số phần tử trong mảng **list\_io\_interrupt** nhé.(trong mảng này các bạn có thể thay đổi IO mà các bạn muốn khai báo ngắt là dc. Như mình là **IO 21**)

Bước 2: Khai báo thêm thư viện my\_interrupt.h bằng lệnh #include “my\_interrupt.h”

Bước 3: Sau đó viết thêm hàm **callbackInterrupt()**; để khi ESP bắt đc sự kiện ngắt thì vào trong hàm này để thực hiện. (**lưu ý trong hàm này không được để trống.**)

Bước 4: gọi hàm setupInterrupt(); vào trong hàm setup(); . Còn ở hàm loop() các bạn để **thêm hàm delay() để tránh bị reset chip.**

Bước 5: Nạp code (Mình đã hướng dẫn khá chi tiết ở phần c) Full code library my\_multi\_wifi.h rồi nên bạn có thể lên lại để xem nhé)

**Note :** nếu mở Serial nhưng không thấy gì hãy nhấn reset chip bằng cách ấn vào nút EN trên chip

\*\* ta được kết quả khi nhấn vào nút vật lý:

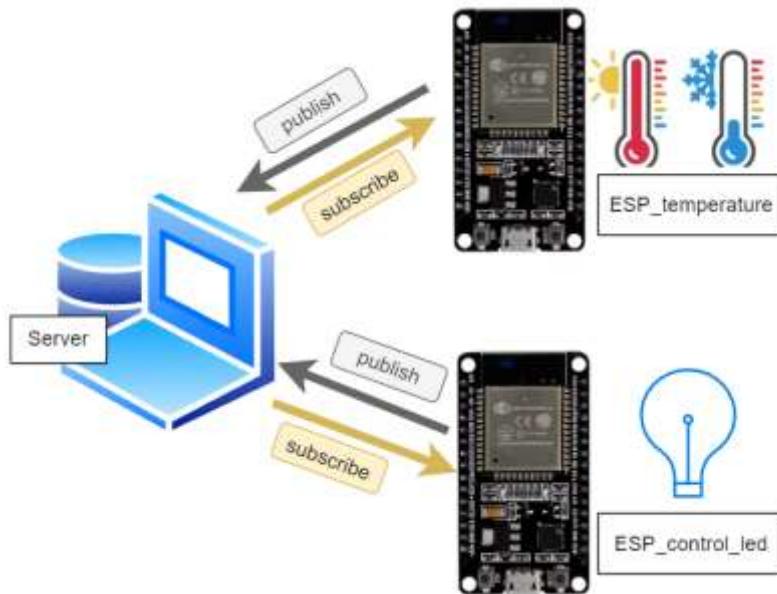
```
20:33:25.701 -> =====INTERRUPT=====  
20:33:25.701 -> IO21 has become interruptIO  
20:33:25.701 -> Setup interrupt done !  
20:33:25.745 -> ======  
20:34:07.690 -> set up interrupt successfully
```

Hình 2.53 kết quả

## 2.5 MQTT:

### 2.5.1 MQTT là gì?

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



Hình 2.54 MQTT

Để dễ hình dung cách thức hoạt động của MQTT thì mình sẽ lấy ví dụ thực tế và so sánh với (Hình 2.54) cho các bạn dễ hiểu nhé.

Bạn hãy tưởng tượng rằng Server (**MQTT Broker**) là một kênh Youtube nào đó. Và ta có các khách hàng ở đây là ESP\_temperature và ESP\_control\_led (**MQTT Client**). Các khách hàng này có thể **Subscribe** kênh để khi nào mà Server có video mới thì sẽ gửi cho khách hàng xem với nội dung là **gửi video (Topic)**. Và Khách hàng có thể đăng ký gói hội viên ở kênh này (**Publish**) và phải gửi tiền mỗi tháng là 10\$ với nội dung là **trả tiền tháng (Topic)**. (Khách hàng có thể không **Subscribe** nhưng vẫn có thể trả hội viên hàng tháng **Publish**. Và có thể chọn ngược lại)

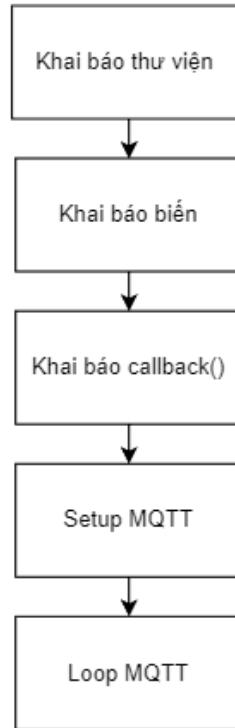
Giờ thì vào code nào. 😊

### 2.5.2 Cá nhân hóa thư viện MQTT

Cách tạo thư viện thì mình đã hướng dẫn ở khá chi tiết ở phần **Cá nhân hóa thư viện Multi Wifi** rồi nên các bạn chưa rõ có thể mở lại để xem nhé.

Ở thư viện này mình đặt tên là **my\_mqtt.h**

d) Hướng đi code:



Hình 2.55 hướng đi code

- Khai báo thư viện: Thêm các thư viện mà ESP và Arduino đã hỗ trợ cho MQTT.
- Khai báo biến: Khai báo các biến cục bộ và cả biến global (bao gồm cả các hàm)
- Khai báo Callback(): là hàm sẽ thực hiện lệnh từ Server gửi xuống.
- Setup MQTT: kết nối với Server
- Loop MQTT: luôn giữ MQTT ở trạng thái lắng nghe lệnh từ Server.

e) Giải thích code:

**Bước 1:** Vẫn như thế để viết cá nhân hóa thư viện ta phải là định nghĩa tên thư viện. (Như mình ở đây, tag thư viện của là `__MY_MQTT_H__`) đặt tag giống như tên thư viện để tránh nhầm lẫn các thư viện khác khi khai báo.

```
#ifndef __MQTT_SETUP_H__
#define __MQTT_SETUP_H__

#endif
```

Hình 2.56 Khai báo tag thư viện

Mục đích để không sợ bị báo lỗi trùng lặp thư viện.

## Chương 1: Cơ sở lý thuyết

**Bước 2:** Thêm các thư viện cần dùng khi setup MQTT (hình 2.57)

```
#include <SPI.h>
#include <PubSubClient.h>
#include <WiFiClient.h>
```

Hình 2.57 Khai báo tag thư viện

Thư viện SPI.h và WiFiClient.h thì trê Arduino đã hỗ trợ rồi. còn thư viện **PubSubClient.h** thì các bạn hãy vào **Tools>Manager Libraries...** rồi tìm **PubSubClient** của tác giả **Nick O'Leary** rồi install vào là được.

**Bước 3:** Khai báo client

```
WiFiClient client;
PubSubClient mqtt_client(client);
```

Hình 2.58 Khai báo client

Đây là 2 dòng để khai báo là ESP là MQTT Client

**Bước 4:** Khai báo biến sử dụng trong thư viện

## Chương 1: Cơ sở lý thuyết

```
#ifndef DELAY_TIME_MQTT
#define DELAY_TIME_MQTT          1*60           //1m
#endif

#ifndef MQTT_SERVER
#define MQTT_SERVER                "localhost"
#endif

#ifndef MQTT_PORT
#define MQTT_PORT                  1883
#endif

#ifndef MQTT_ID
#define MQTT_ID                     "ESP"
#endif

#ifndef TOPPIC_SUB
#define TOPPIC_SUB                 "S-ESP"
#endif

#ifndef TOPPIC_PUB
#define TOPPIC_PUB                 "P-ESP"
#endif

unsigned long old_time_mqtt = 0;
String buffer_data_from_server = "";
const char *mqttserver = MQTT_SERVER;
const int mqttport = MQTT_PORT;
const char *mqttid = MQTT_ID;
const char *toppicsub = TOPPIC_SUB;
const char *topicpub = TOPPIC_PUB;
```

Hình 2.59 Khai báo biến

Ở đây mình định nghĩa như sau:

- Định nghĩa thời gian định kì mà ESP gửi data cho Server (tag **DELAY\_TIM\_MQTT**)
- Định nghĩa địa chỉ để truy cập vào server (tag **MQTT\_SERVER**)
- Định nghĩa port để vào được địa chỉ server (tag **MQTT\_PORT**)
- Định nghĩa tên cho ESP để không bị trùng với các ESP khác (tag **MQTT\_ID**)
- Định nghĩa topic subscribe mà ESP đăng ký với Server (tag **TOPPIC\_SUB**)
- Định nghĩa topic publish mà ESP đăng ký với Server (tag **TOPPIC\_PUB**)
- Khai báo biến **old\_time\_mqtt** là thời gian gần nhất kể từ lúc data được gửi đi cho Server
- Khai báo biến **buffer\_data\_from\_server** là bộ đệm tạm thời để sử dụng trong hàm xử lý lệnh
- Các khai báo còn lại là khai báo cho đúng kiểu dữ liệu thôi.

Note: ta chỉ khai báo tên hàm thôi còn code thì ta sẽ viết ở **app.h** để có thể tùy chỉnh các nhiệm vụ khác nhau tùy thuộc vào project của ta là gì.

## Chương 1: Cơ sở lý thuyết

**Bước 3:** Tạo các hàm với từng chức năng riêng (**một hàm chỉ nên làm một nhiệm vụ**) trong thư viện.

- **Hàm xử lý lệnh từ ESP gửi xuống (chỉ khai báo một châm):**

```
void executeMqttCommand(String command);
```

Hình 2.60 khai báo hàm

Note: ta chỉ khai báo tên hàm thôi còn code thì ta sẽ viết ở **app.h** để có thể tùy chỉnh các nhiệm vụ khác nhau tùy thuộc vào project của ta là gì.

- **Hàm gửi dữ liệu cho Server :**

```
void mqttSendServer(String data)
{
    if (!mqtt_client.connected())
    {
        statusMQTT();
        return;
    }

    if ( millis() - old_time_mqtt > DELAY_TIME_MQTT * 1000 )
    {
        mqtt_client.publish(topicpub, data.c_str());
        old_time_mqtt = millis();
        return;
    }
}
```

Hình 2.61 Khai báo hàm

ở hàm này sẽ kiểm tra kết nối MQTT. Nếu k bị lỗi thì một phút thi gửi dữ liệu lên cho Server một lần.

- **Hàm thông báo lỗi kết nối MQTT :**

## Chương 1: Cơ sở lý thuyết

```
void statusMQTT()
{
    int state = mqtt_client.state();
    switch (state)
    {
        case -4:
            Serial.println("MQTT_CONNECTION_TIMEOUT");
            break;
        case -3:
            Serial.println("MQTT_CONNECTION_LOST");
            break;
        case -2:
            Serial.println("MQTT_CONNECT_FAILED");
            break;
        case -1:
            Serial.println("MQTT_DISCONNECTED");
            break;
        case 0:
            Serial.println("MQTT_CONNECTED");
            break;
        case 1:
            Serial.println("MQTT_CONNECT_BAD_PROTOCOL");
            break;
        case 2:
            Serial.println("MQTT_CONNECT_BAD_CLIENT_ID");
            break;
        case 3:
            Serial.println("MQTT_CONNECT_UNAVAILABLE");
            break;
        case 4:
            Serial.println("MQTT_CONNECT_BAD_CREDENTIALS");
            break;
        case 5:
            Serial.println("MQTT_CONNECT_UNAUTHORIZED");
            break;

        default:
            Serial.println("unknown!!!!");
            break;
    }
}
```

Hình 2.62 Khai báo hàm

ở hàm này sẽ kiểm tra kết nối MQTT và in ra trạng thái hiện tại của MQTT

- **Hàm in ra thông tin kết nối MQTT :**

## Chương 1: Cơ sở lý thuyết

```
void showInforMQTT(){
    Serial.println("_____MQTT_____");
    Serial.println("Connected MQTT");
    Serial.print("MQTT SERVER:");
    Serial.print(String(mqttserver));
    Serial.println(": " + String(mqttport));
    Serial.print("MQTT ID:");
    Serial.println(mqttid);
    Serial.print("TOPIC SUB:");
    Serial.println(toppicsub);
    Serial.print("TOPIC PUB:");
    Serial.println(toppicpub);
    Serial.println("_____");
}
```

Hình 2.63 Khai báo hàm

Hàm này sẽ in ra các thông tin liên quan đến MQTT

- **Hàm callback MQTT (hàm này là bắt buộc phải có):**

```
void callbackMqtt(char *topic, byte *payload, unsigned int length)
{
    Serial.print("Received from: ");
    Serial.println(topic);
    for (int i = 0; i < length; i++)
    {
        char temp_data = (char)payload[i];
        buffer_data_from_sever += temp_data;
    }
    Serial.println(buffer_data_from_sever);

    executeMqttCommand(buffer_data_from_sever);

    // reset buffer_data_from_sever
    buffer_data_from_sever = "";
}
```

Hình 2.64 Khai báo hàm

Hàm này sẽ đọc từng ký tự mà Server gửi đến ESP. Sau khi đọc xong thì vào hàm xử lý lệnh. Xử lý lệnh xong thì xóa tin nhắn vừa đọc được (việc này để tránh dữ liệu trùng lặp nhau)

- **Hàm setup Interrupt**

## Chương 1: Cơ sở lý thuyết

```
void setupMQTT()
{
    Serial.println("Connecting MQTT ...");

    mqtt_client.setServer(mqttserver, mqttport);
    mqtt_client.setCallback(callbackMqtt); // Cần 1 hàm callback(topic, payload (nội dung cần truyền), tổng số ký tự có trong payload)

    while (!mqtt_client.connect(mqttid))
    {
        delay(1000);
        statusMQTT();
        Serial.print(".");
    }

    mqtt_client.publish(topicpub, "Connected !!! Waiting command");
    mqtt_client.subscribe(topicsub);

    showInfoMQTT();
}
```

Hình 2.65 Khai báo hàm setup MQTT

Hàm setup này sẽ set các thông số để init MQTT

**mqtt\_client.setServer(mqttserver, mqttport);** là hàm khai báo Server

**mqtt\_client.setCallback(callbackMqtt);** là hàm khai báo hàm callback

sau đó chờ kết nối MQTT. Không thành công thì thông báo lý do bằng hàm **statusMQTT();**. Còn nếu thành công thì đăng ký topic với Server

### - Hàm loopMQTT

```
void loopMQTT()
{
    mqtt_client.loop();
}
```

Hình 2.66 Khai báo hàm loop MQTT

Chỉ cần sử dụng hàm **mqtt\_client.loop();** để giữ trạng thái luôn luôn lắng nghe lệnh từ server. (Nếu k có hàm này thì khi server gửi tin ESP sẽ k biết đế mà xử lý lệnh)

### f) Full Code library my\_interrupt.h và chạy thử thư viện

\*\* full code **my\_mqtt.h**

```
#ifndef __MQTT_SETUP_H__
#define __MQTT_SETUP_H__

#include <SPI.h>
#include <PubSubClient.h>
#include <WiFiClient.h>

WiFiClient client;
PubSubClient mqtt_client(client);

#ifndef DELAY_TIME_MQTT
```

**Note: để lấy full code nhấn vào ô chứa code rồi Ctrl + C. Vào file my\_ota.h trong Arduino rồi Ctrl + V.**

## Chương 1: Cơ sở lý thuyết

- Vì giao thức MQTT cần có Server thì mới có thể test được nên phần này sẽ được lùi lại vào chương 3. Sau khi mà ta đã cài đặt và làm về Node-red nhé. Nhưng các bạn có thể check xem code đã viết đúng chưa bằng cách gọi thư viện vào trong file <tên project>.ino (của mình là **ESP32\_SmartHome.ino**)

\*\* full code **ESP32\_SmartHome.ino**

```
#include "my_mqtt.h"
void setup()
{
    Serial.begin(115200);
}
```

Nếu không báo lỗi gì thì có nghĩa là tạm thời code của ta chưa lỗi.

Oke vậy mọi thư viện đã được setup xong rồi. Giờ thì các bạn hãy dùng git để push code lên GitHub lưu code lên Internet nào.

## Chương 1: Cơ sở lý thuyết

Mở VSC rồi nhấn tổ hợp phím **Ctrl + `** để mở Terminal

Nhập **cd <đường dẫn vào thư mục mà bạn đã cài ở phần setup GitHub>** (của mình là **cd G:\Arduino**)

```
PS G:\Arduino>
```

Nếu vào đường dẫn thì sẽ hiện như hình trên thẻ này.

Sau đó nhập **git add .** (thêm tất cả các file mới)

```
PS G:\Arduino> git add .
warning: in the working copy of 'ESP32_Smarthome/ESP32_Smarthome.ino', LF will be replaced by CRLF the next time Git touches it
PS G:\Arduino>
```

Nhập tiếp **git commit -m "add my library"** (commit code lên với nội dung là add my library)

```
PS G:\Arduino> git commit -m "add my library"
[main 1af3f3a] add my library
 2 files changed, 1 deletion(-)
PS G:\Arduino>
```

Nhập tiếp **git push**

```
PS G:\Arduino> git push
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 6 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 177.33 KiB | 1.08 MiB/s, done.
Total 5 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/thanhle1408/SmartHome.git
  99e3fb0..1af3f3a main -> main
PS G:\Arduino>
```

Hãy push code lên GitHub thường xuyên nhé

### 2.6 Quy chuẩn Project:

Trước khi qua Chương mới thì mình muốn quy chuẩn một project thì cần có những gì nhé. Để sau này ta có thể dễ quản lý và dễ dàng truy suất file

**Bước đầu tiên là gộp tất cả các thư viện mà ta đã làm ở 2.2 Multi Wifi , 2.3 OTA, 2.4 Interrupt, 2.5 MQTT lại thành 1 file my libraris.**

**Sau đó là các file cần có cho một Project**

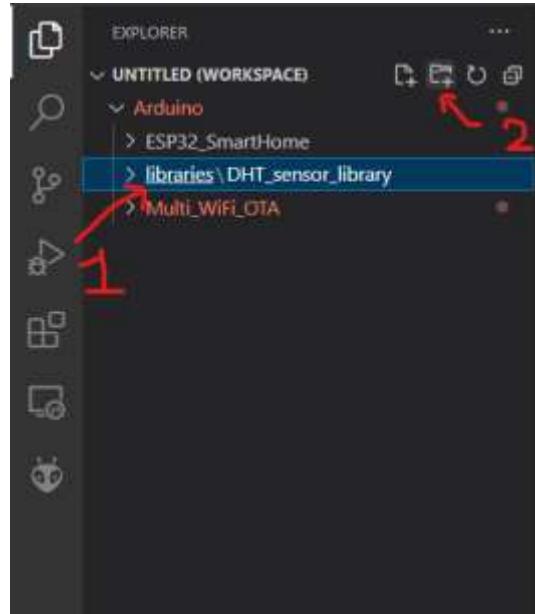
#### 2.6.1 Gộp thư viện:

Trước tiên các bạn cần tạo một thư viện cá nhân trước đã.

## Chương 1: Cơ sở lý thuyết

Bước 1: Mở VSC chọn File > Add Folder to Workspace rồi chọn Folder bừa bạn đã tạo ở bước Cài đặt Arduino IDE (Như mình là **G:\Arduino**)

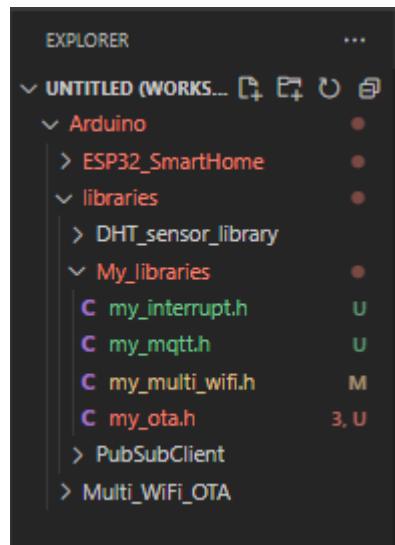
Bước 2: nhấn vào chữ library rồi ấn vào biểu tượng tạo Folder. Sau đó đặt tên cho folder mới (Như mình thì mình đặt tên là **My\_libraries**)



Hình 2.67 Tạo Folder My\_libraries

Note: Nếu bạn không có folder libraries như mình thì bạn đã bỏ qua bước thêm thư viện vào Arduino rồi đây. Hãy quay lại bước hướng dẫn cài đặt Arduino IDE ở Chương I mình cũng đã hướng dẫn thêm thư viện DHT rồi đó.

Bước 3: Cắt (Ctrl + X) tất cả các file có dạng **my\_<tên thư viện>.h** ở Folder **ESP32\_SmartHome** và dán (Ctrl + V) tất cả các file đó vào folder **My\_libraries**

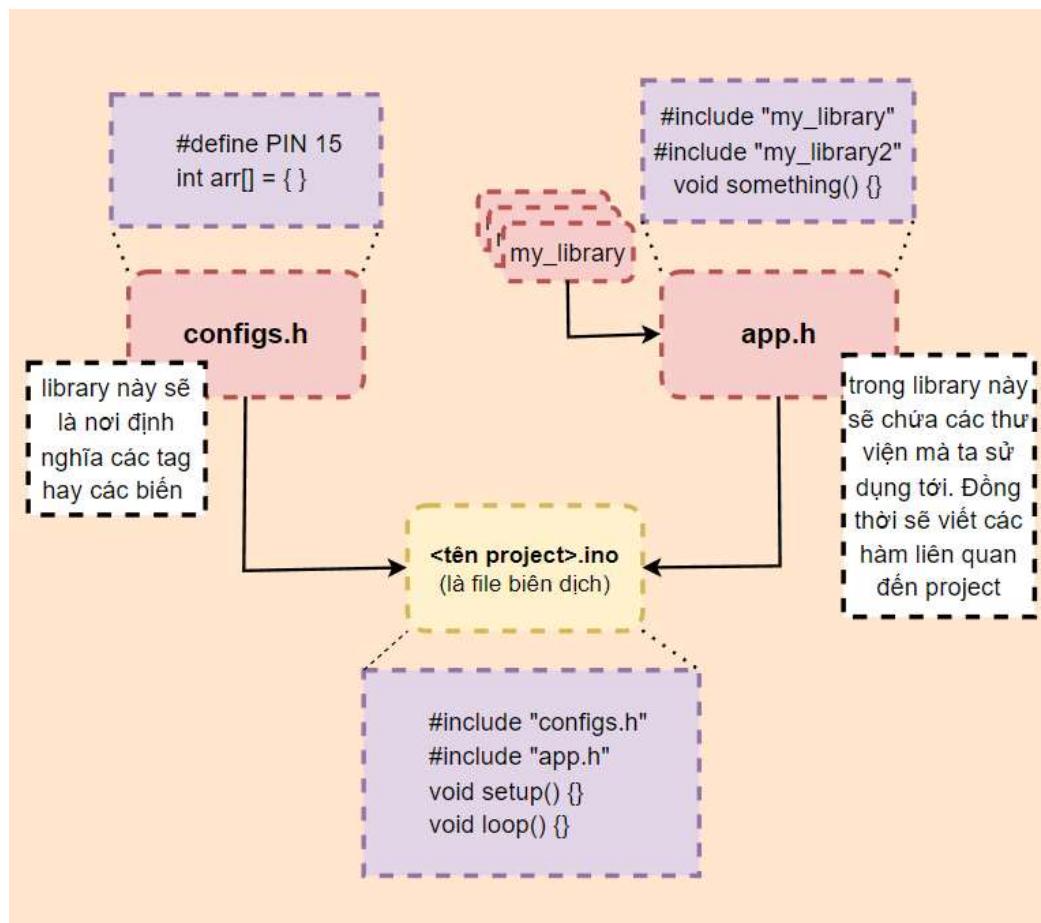


Hình 2.68 Dán các file thư viện đã cá nhân hóa vào folder My\_libraries

**NOTE: nếu mở bằng VSC mà thấy báo lỗi màu đỏ thì các bạn cứ kệ nó đi vì VSC mình chúng ta chưa cài Extension hỗ trợ check syntax trên Arduino nên nó báo lỗi thôi. Mục đích chúng ta sài VSC chỉ để check truy suất file một cách nhanh chóng thôi.**

### 2.6.2 Quy chuẩn project:

Qua 4 phần cá nhân hóa thư viện ở mục Multi Wifi, OTA, Interrupt, MQTT thì các bạn cũng đã dần quen với việc viết thư viện. Và giờ ta hãy quy chuẩn đôi chút về một project nhé.



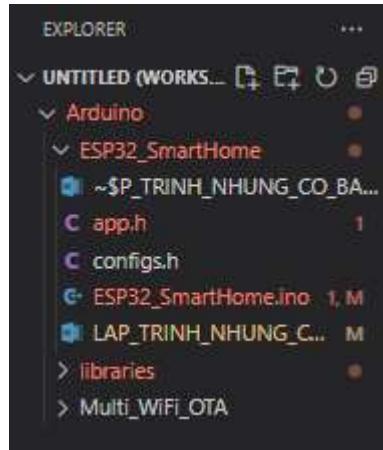
Hình 2.69 tiêu chuẩn các file của một project

Một project sẽ phải có ba file cố định:

- `<project>.ino` : đây là file để Arduino IDE biên dịch được code.
- `app.h` : Nơi bạn khai báo tất cả các thư viện mà bạn sẽ sử dụng ở project. Và cũng là nơi viết các hàm áp dụng trong project.
- `configs.h` : Nơi bạn khai báo biến global hay định nghĩa lại các tag ở trong các thư viện để phù hợp với Project hơn

## Chương 1: Cơ sở lý thuyết

Như mình thì mình để ở hình bên dưới (hình 2.70)



Hình 2.70 ba file tiêu chuẩn của một project

**NOTE: Các bạn đừng bận tâm tới 2 cái file word  
LAP\_TRINH\_NHUNG\_CO\_BAN nhé. Mình để vào file để git push luôn một thê ấy  
mà.**

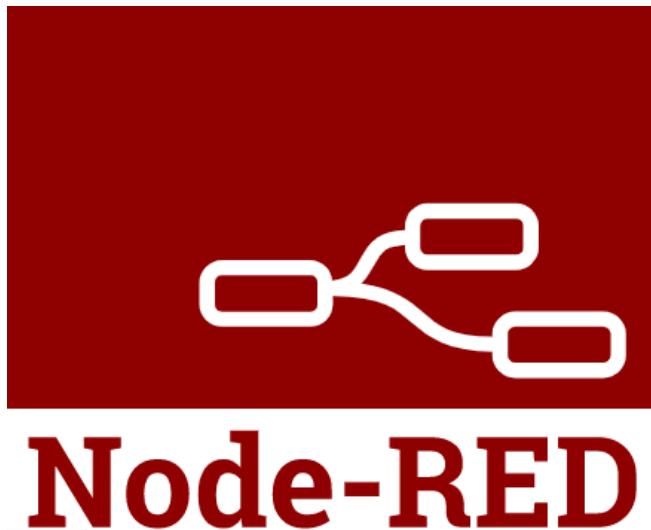
À làm xong về Phần quy chuẩn này thì các bạn hãy Git push code lên GitHub luôn  
nhé.

```
PS G:\Arduino> git commit -m "configs project"

The most similar command is
    commit
PS G:\Arduino> git commit -m "configs project"
[main 8b9eea2] configs project
 7 files changed, 15 insertions(+), 135 deletions(-)
 delete mode 100644 ESP32_SmartHome/my_multi_wifi.h
 rename {ESP32_SmartHome => libraries/My_libraries}/my_interrupt.h (100%)
 rename {ESP32_SmartHome => libraries/My_libraries}/my_mqtt.h (100%)
 rename {ESP32_SmartHome => libraries/My_libraries}/my_ota.h (100%)
PS G:\Arduino> git push
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 6 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 221.83 KiB | 1.47 MiB/s, done.
Total 7 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/thanhle1408/SmartHome.git
 1af3f3a..8b9eea2 main -> main
PS G:\Arduino>
```

## CHƯƠNG III: NODE-RED

### 3.1 Tổng quan:



Hình 3.1 Biểu tượng Node-Red

Nói nôm na Node-RED là một công cụ **lập trình kéo-thả** (FBP) để kết nối các thiết bị phần cứng, API và online services với nhau. Nó cung cấp một trình soạn thảo dựa trên trình duyệt giúp dễ dàng kết nối các luồng với nhau bằng cách sử dụng một loạt các Node trong bảng màu (palette) có thể được triển khai chỉ bằng một cú nhấp chuột.

Nay giờ các bạn đã được nghe mình nhắc đến khá nhiều về Server rồi áy. Để làm quen với server và không tốn kinh phí thì **máy tính cá nhân** là lựa chọn hợp lý nhất. Ở đây mình sẽ sử dụng PC chạy hệ điều hành **windows** để làm server nhé.

Và để làm quen và đơn giản hóa server thì mình sẽ giảm thiểu tối đa sử dụng command trên Terminal và đồng thời sử dụng Node-red – lập trình kéo thả trên server. Mặc dù là có lập trình bằng JavaScript nhưng các bạn không cần phải biết nhiều về JavaScript. Chỉ cần kéo thả là được. Và đây là cơ bản nên mình sẽ hướng dẫn cơ bản nhất có thể, để các bạn có một sản phẩm thực tế, ngon nghẻ nhưng vẫn đơn giản.

### 3.2 Cài đặt Node-red:

Vì Node-red chạy trên nền tảng NodeJS nên trước khi cài Node-red thì ta phải cài NodeJS trước. Các bạn làm theo mình nhé.

**Bước 1:** tải NodeJS phù hợp với máy tính của bạn nhé (32bit hay 64bit)

Link tải: [Download | Node.js \(nodejs.org\)](https://nodejs.org)

## Chương 1: Cơ sở lý thuyết

### Downloads

Latest LTS Version: 16.17.1 (includes npm 8.15.0)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

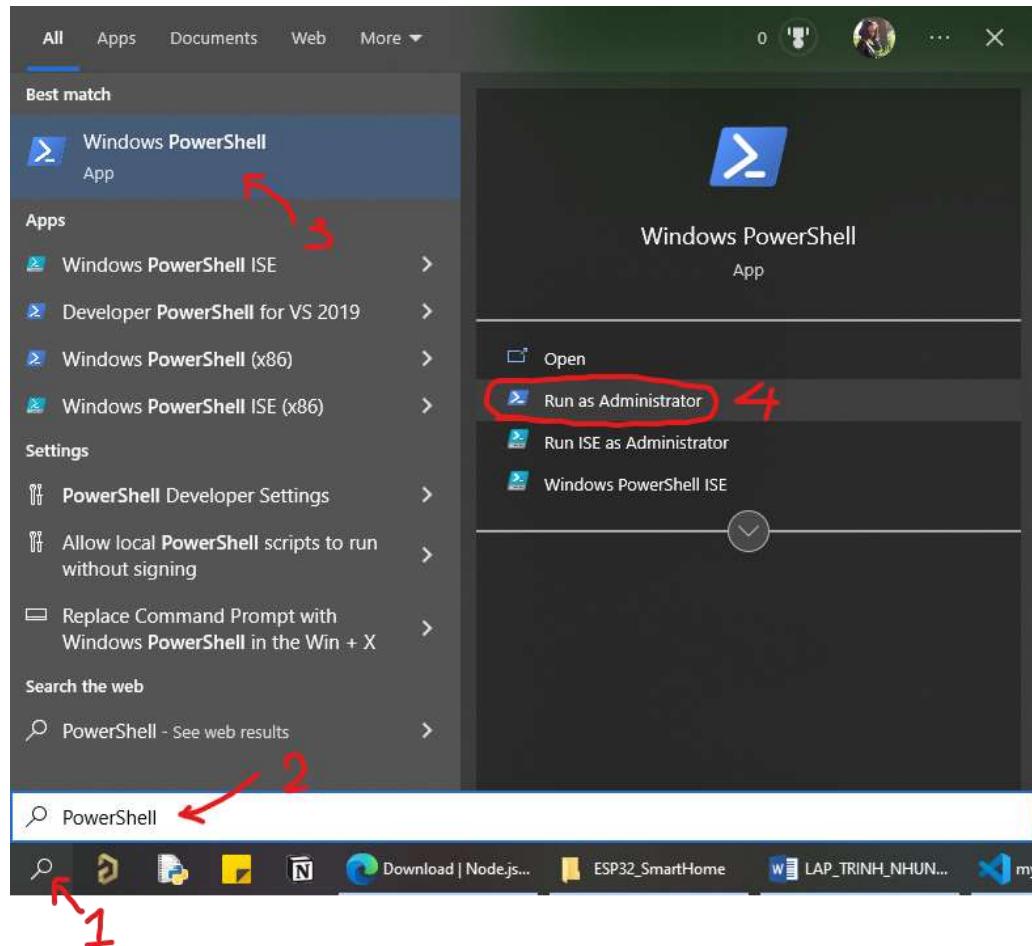


Hình 3.2 Down bản phù hợp với máy tính của bạn

**Bước 2:** Các bạn cứ ấn Next rồi Install (nếu có I accept thì tích vào ô rồi Next tiếp)

**Bước 3:** Để kiểm tra chắc chắn là đã cài bản mới nhất rồi thì các bạn mở **PowerShell** dưới quyền admin. Hay **cmd** dưới quyền admin cũng được (nhưng để chắc chắn thì mình cài đặt trên **powershell** còn run server thì mình chạy trên **cmd**)

## Chương 1: Cơ sở lý thuyết



Hình 3.3 Mở PowerShell

**Bước 4:** nhập **node -v** (kiểm tra phiên bản nodejs) **npm -v** (kiểm tra phiên bản npm – quản lý packet) kết quả như (hình 3.4) là cài nodejs xong rồi nhé.

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> node -v
v16.17.1
PS C:\WINDOWS\system32> npm -v
8.15.0
PS C:\WINDOWS\system32>
```

Hình 3.4 PowerShell

**Bước 5:** cài đặt Node-red trên PowerShell bằng cách nhập câu lệnh  
**npm install -g --unsafe-perm node-red.**

Ngoài ra các bạn có thể đọc hướng dẫn hay cách fix lỗi trên trang chủ của Node-red bằng link ở dưới nhé

(Link tải: [Running Node-RED locally : Node-RED \(nodered.org\)](https://nodered.org/))

## Chương 1: Cơ sở lý thuyết

**Note:** Nếu các bạn đến đây mà bị lỗi như mình (hình 3.5) thì gõ thêm dòng lệnh **npm audit fix --force**

```
PS C:\WINDOWS\system32> npm install -g --unsafe-perm node-red
added 292 packages, and audited 293 packages in 15s

38 packages are looking for funding
  run `npm fund` for details

5 vulnerabilities (4 low, 1 moderate)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
npm notice New minor version of npm available! 8.15.0 -> 8.19.2
npm notice Changelog: https://github.com/npm/cli/releases/tag/v8.19.2
npm notice Run npm install -g npm@8.19.2 to update!
npm notice
PS C:\WINDOWS\system32>
```

Hình 3.5 Lỗi

Màn hình như (hình 3.6) thì các bạn đã cài node red xong rồi áy nhé

```
PS C:\WINDOWS\system32> npm audit fix --force
npm WARN                                         Recommended protections disabled.

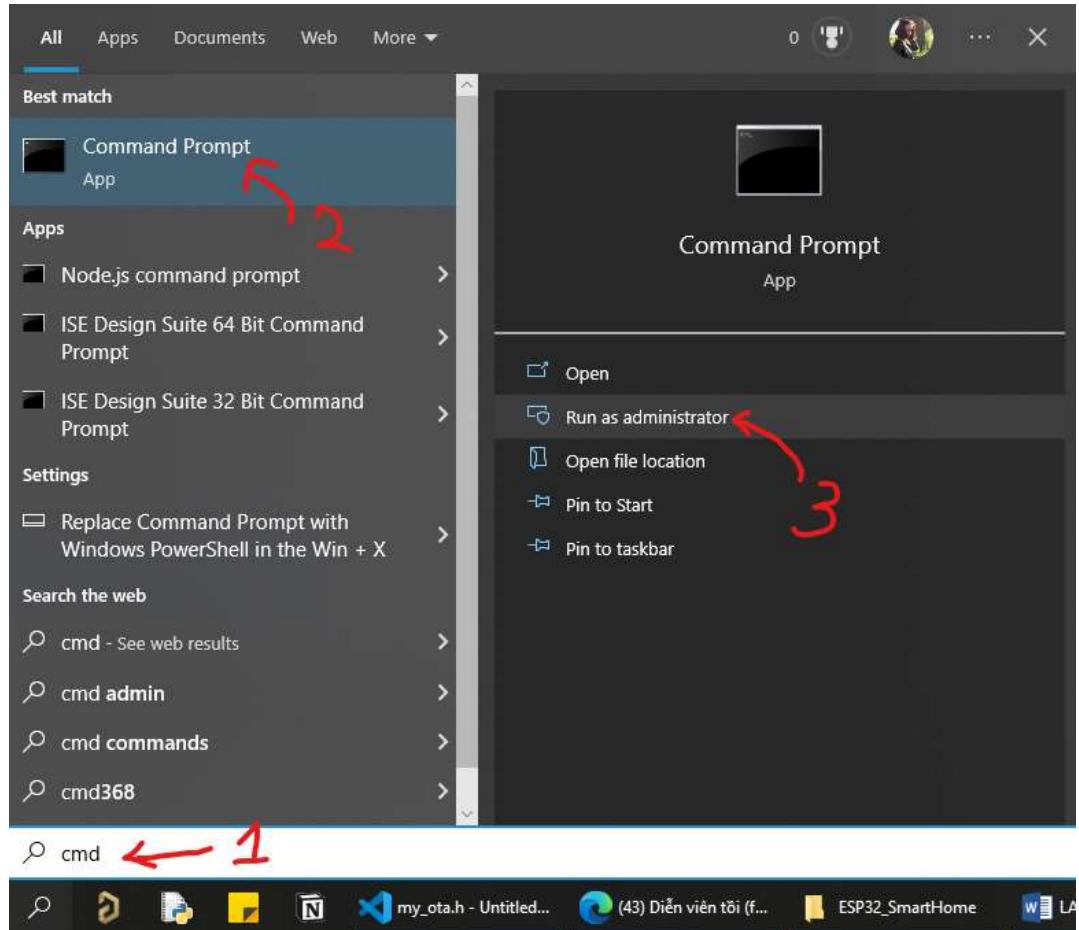
up to date, audited 1 package in 143ms

found 0 vulnerabilities
PS C:\WINDOWS\system32>
```

Hình 3.6 fix Lỗi

**Bước 6:** mở cmd:

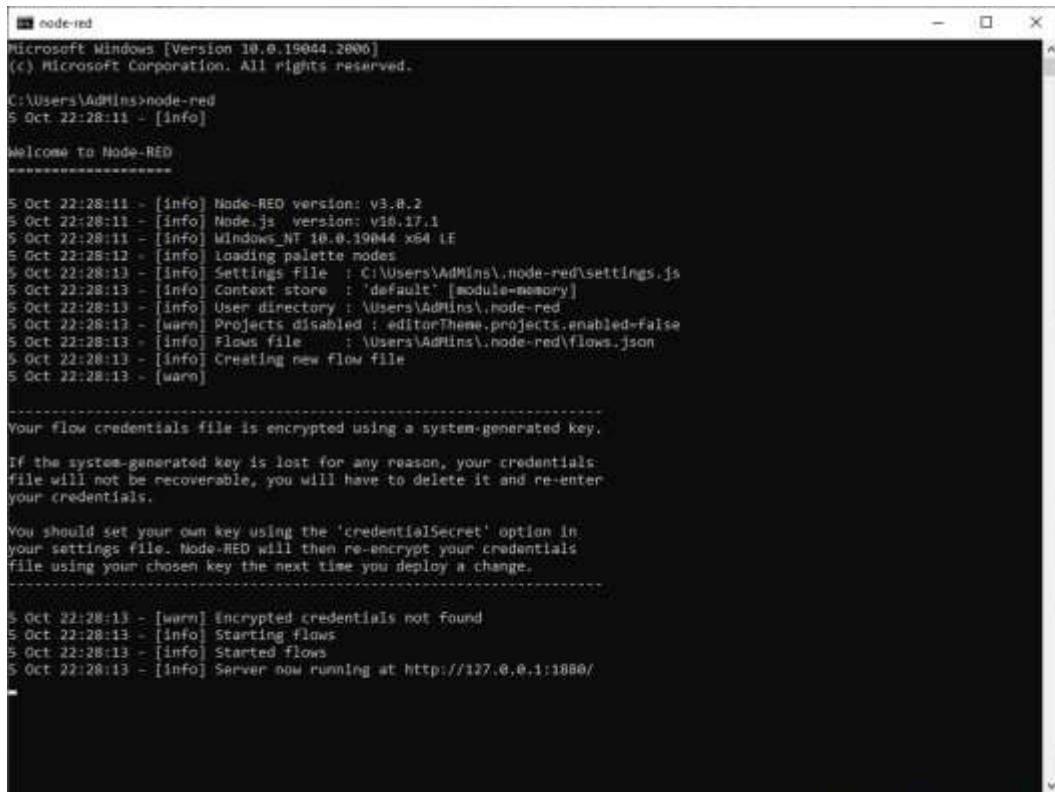
## Chương 1: Cơ sở lý thuyết



Hình 3.7 mở cmd

Bước 7: nhập lệnh **node-red**

## Chương 1: Cơ sở lý thuyết



```
code-red
Microsoft Windows [Version 10.0.19044.2886]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Admins>node-red
5 Oct 22:28:11 - [info]

Welcome to Node-RED

5 Oct 22:28:11 - [info] Node-RED version: v3.8.2
5 Oct 22:28:11 - [info] Node.js version: v18.17.1
5 Oct 22:28:11 - [info] Windows_NT 10.0.19044 x64 IE
5 Oct 22:28:12 - [info] loading palette nodes
5 Oct 22:28:13 - [info] Settings file : C:\Users\Admins\.node-red\settings.json
5 Oct 22:28:13 - [info] Context store : 'default' [module-memory]
5 Oct 22:28:13 - [info] User directory : \Users\Admins\.node-red
5 Oct 22:28:13 - [warn] Projects disabled : editorTheme.projects.enabled=false
5 Oct 22:28:13 - [info] Flows file : \Users\Admins\.node-red\flows.json
5 Oct 22:28:13 - [info] Creating new flow file
5 Oct 22:28:13 - [warn]

Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.

5 Oct 22:28:13 - [warn] Encrypted credentials not found
5 Oct 22:28:13 - [info] Starting flows
5 Oct 22:28:13 - [info] Started flows
5 Oct 22:28:13 - [info] Server now running at http://127.0.0.1:1880/
```

Hình 3.8 mở server node-red

Bước 8: vào trình duyệt với đường dẫn <http://127.0.0.1:1880/> (localhost:1880)



Hình 3.9 mở server node-red

Nếu đã ra được giao diện như trên thì các bạn đã cài thành công rồi nhé.

**NOTE: cmd luôn luôn phải chạy để duy trì server. Vì vậy node-red sẽ không chạy nếu các bạn tắt cửa sổ cmd nhé.**

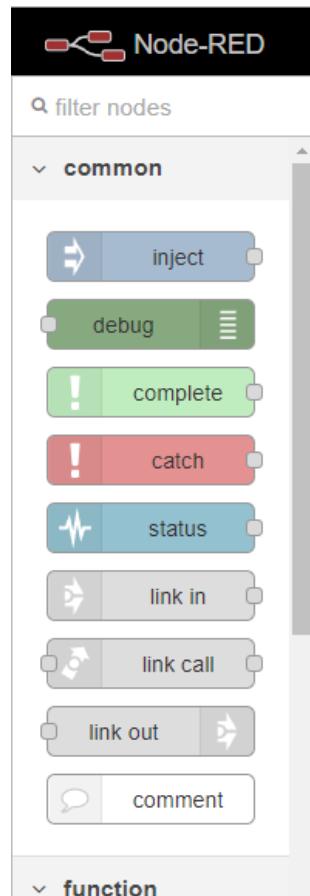
### 3.3 Hướng dẫn sử dụng cơ bản node-red

Để tránh mất thời gian thì ở phần này mình chỉ giới thiệu cơ bản thôi. Còn chi tiết các node sử dụng trong project ESP32\_SmartHome thì mình sẽ hướng dẫn ở phần sau nhé.

#### 3.3.1 Sử dụng cơ bản:

Mình sẽ đi từ trái sang phải nhé.

a) Palete: Đây là nơi chứa các node mà bạn đã install package.

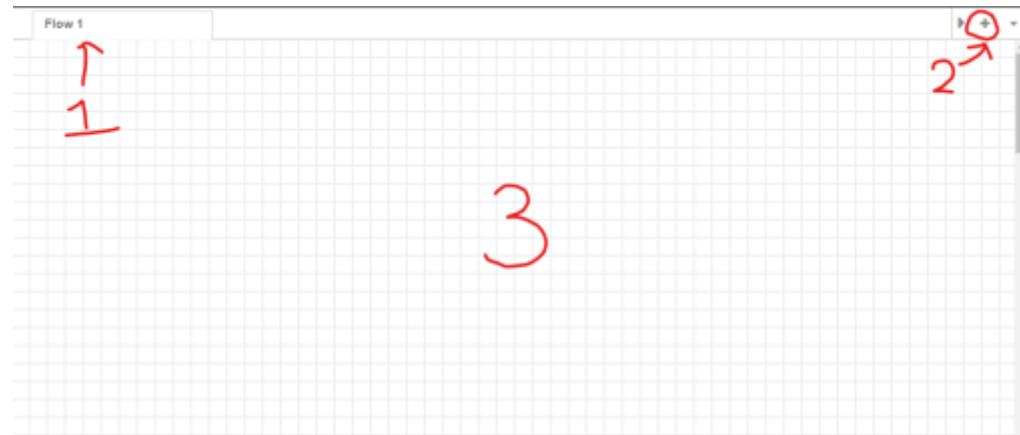


Hình 3.10 Palete

#### b) Màn hình làm việc:

- Vị trí 1 (Flow1): là nơi làm việc hiện tại. Ở Node-red các Flow sẽ hoạt động đồng thời. Mục đích chủ yếu của các Flow này là để chúng ta phân chia mỗi flow là một chung năng. Như vậy ta có thể dễ dàng tìm kiếm và quản lý. (cái này khá giống với sheet trong excel vậy)
- Vị trí 2 (+): là nơi các bạn thêm 1 flow mới
- Vị trí 3: là nơi các bạn sẽ chứa các node

## Chương 1: Cơ sở lý thuyết



Hình 3.11 nơi làm việc

### c) Thanh công cụ:

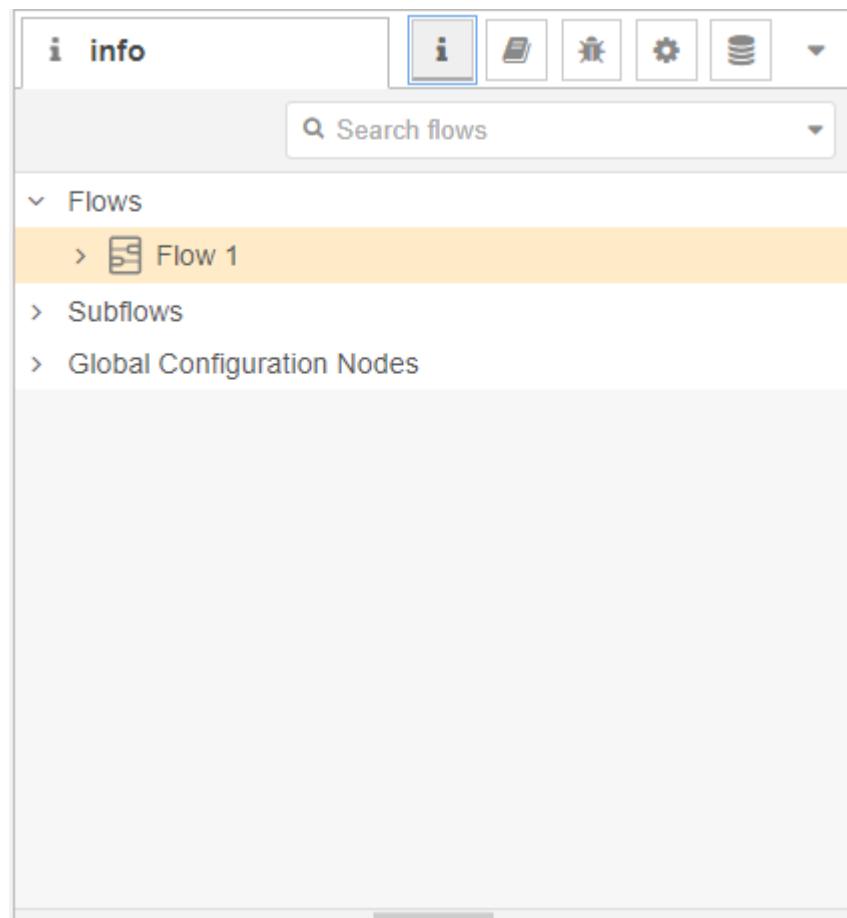
- Vị trí 1 (Deploy): là nút nạp áp dụng chỉnh sửa mới. đồng thời là nút reset lại các flow của Node-red
- Vị trí 2: là nơi hiển thị các thông tin chi tiết
- Vị trí 3 thanh công cụ



Hình 3.12 thanh công cụ

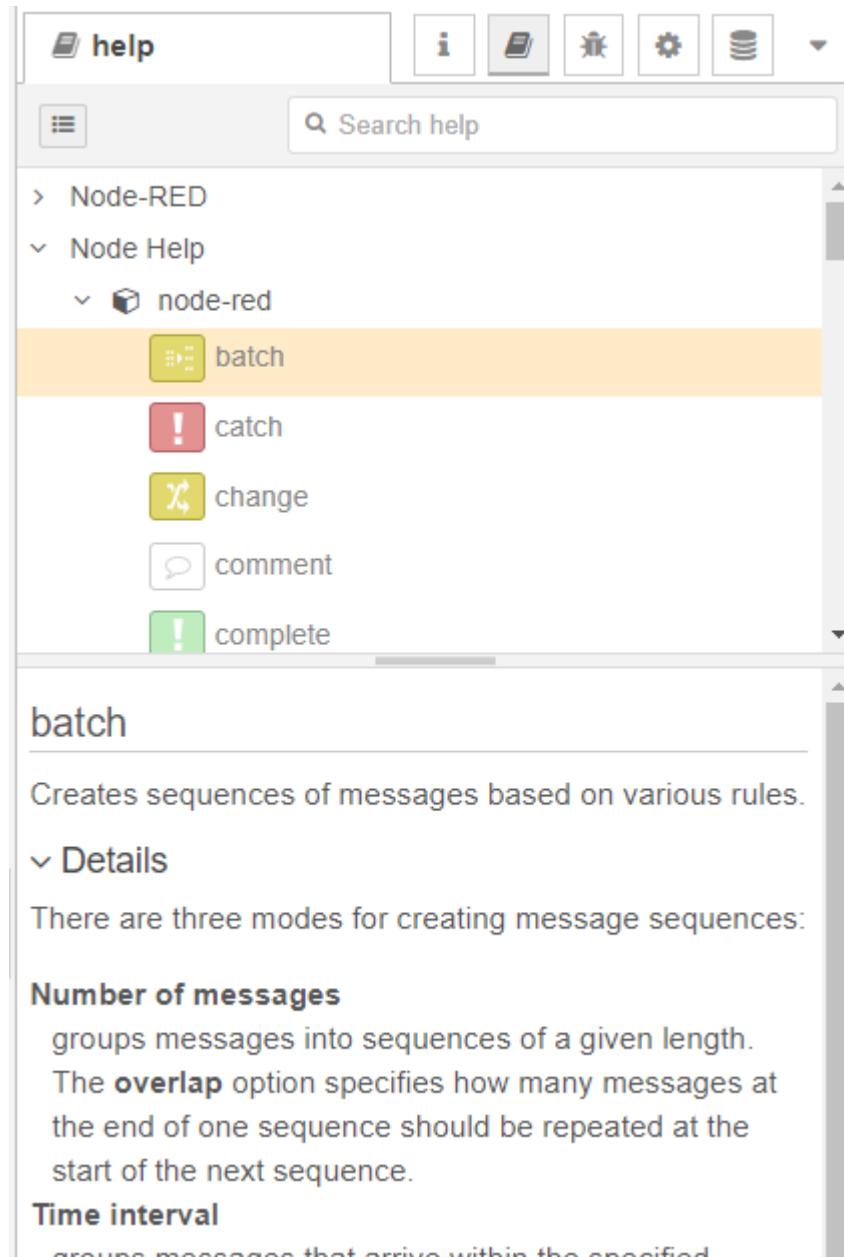
## Chương 1: Cơ sở lý thuyết

Hình 3.13 : là nơi thống kê các node sử dụng ở từng flow. Giúp bạn quản lý các flow.



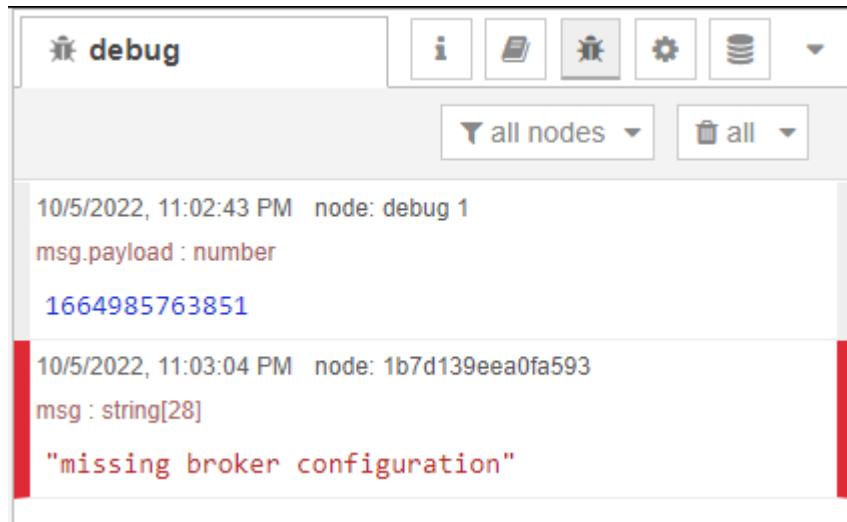
Hình 3.13 information

Hình 3.14 : là nơi hướng dẫn các bạn sử dụng node



Hình 3.14 Help

Hình 3.15 : là nơi để thông báo lỗi, hiển thị các thông tin mà node Debug nhận được



Hình 3.15 Debug

còn lại config và data thì phần này k sử dụng nhiều nên chúng ta next nhé. ☺

**d) Thêm các package vào node-red (cái này mình hay gọi là library trên Node-red)**

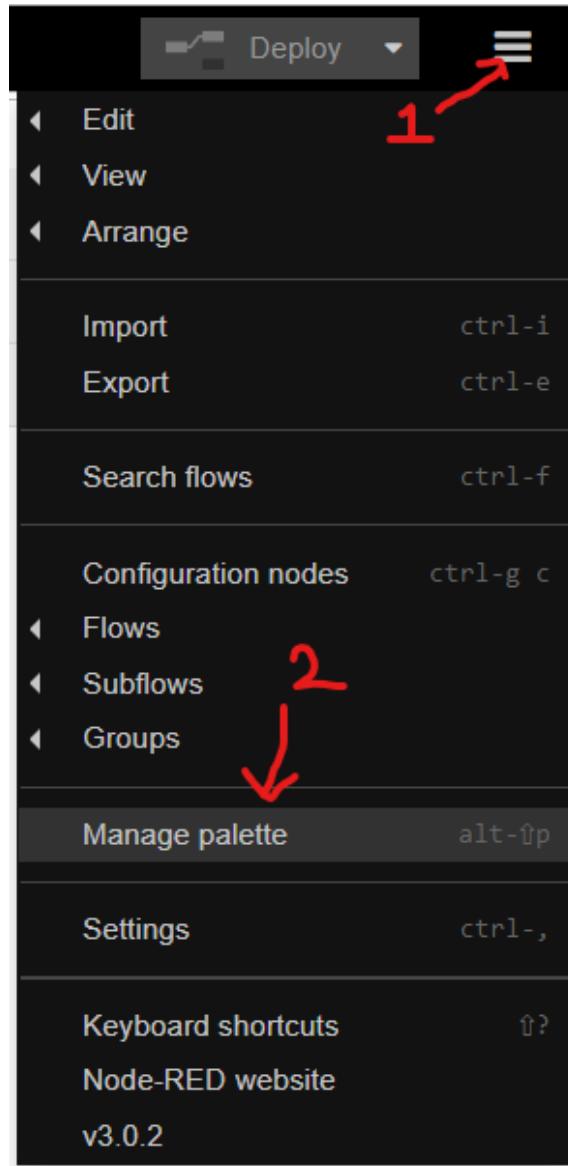
Ở trên trang chủ của Node-red có rất nhiều các package để hỗ trợ chúng ta. Các bạn có thể tham khảo link bên dưới để biết thêm các thông tin chi tiết về các thư viện.

([Library - Node-RED \(nodered.org\)](#))

Mình sẽ hướng dẫn các bạn thêm thư viện **aedes** thư viện giả lập MQTT Broker (serverMQTT):

**Bước 1:** chọn ô 3 gạch ở bên phải trình duyệt rồi chọn **Manage palette**

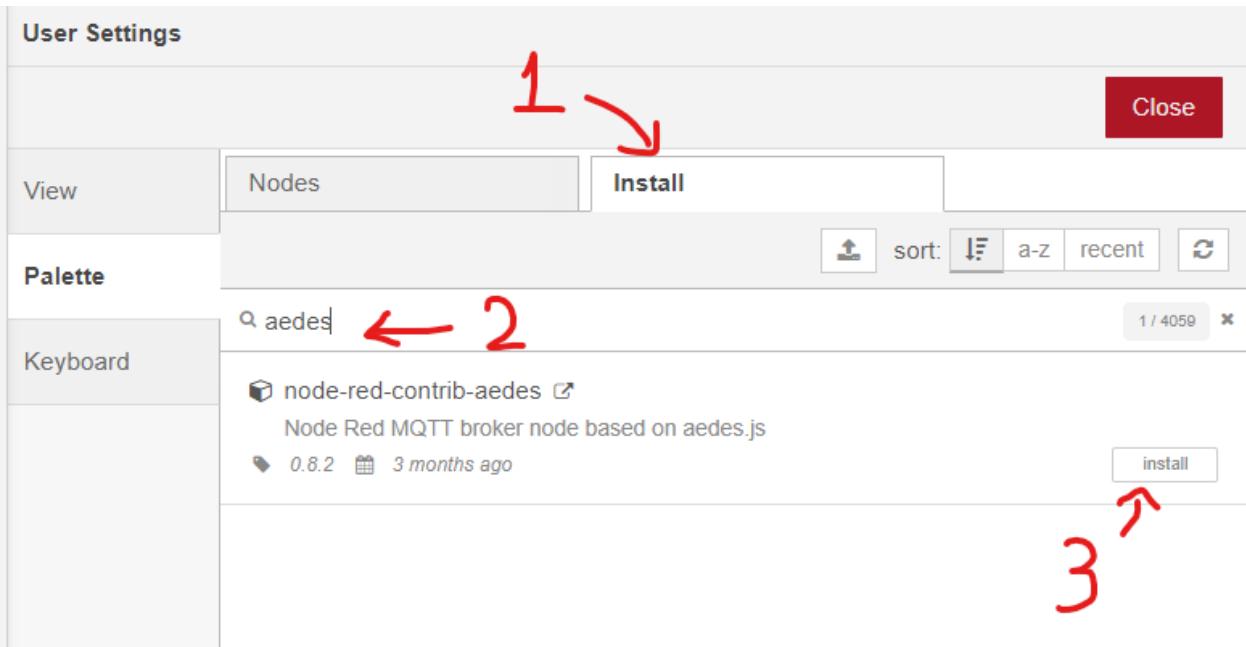
## Chương 1: Cơ sở lý thuyết



Hình 3.16 mở quản lý thư viện

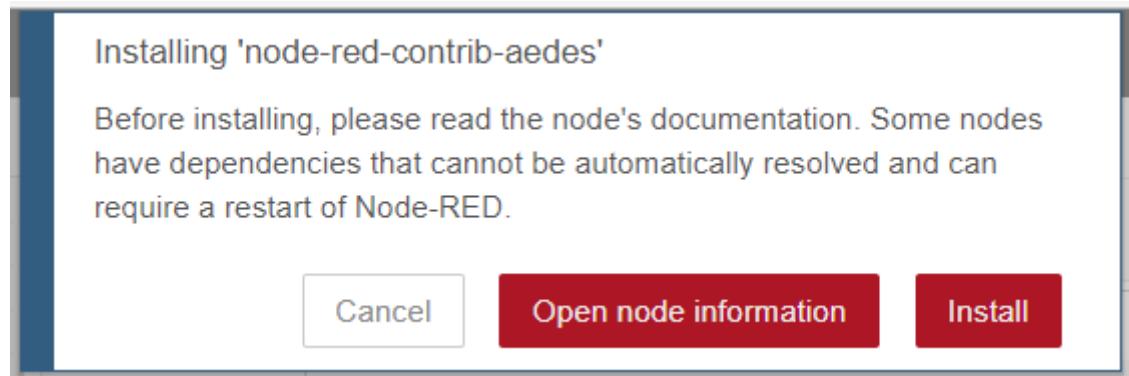
Bước 2: chọn **Install** gõ “**aedes**” rồi ấn nút **install** để cài đặt thư viện

## Chương 1: Cơ sở lý thuyết



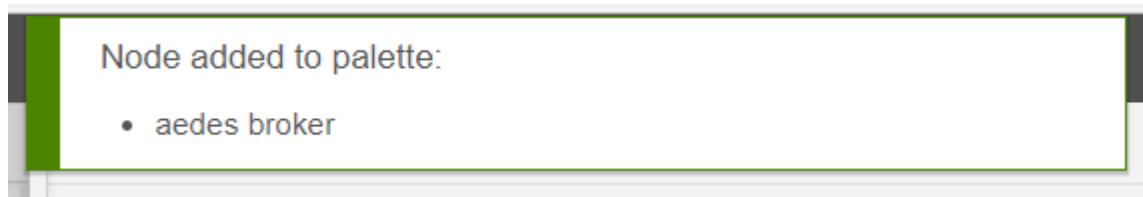
Hình 3.17 cài đặt thư viện

Bước 3: ấn Install một lần nữa để cài đặt



Hình 3.18 cài đặt thư viện

Hiển thị như (hình 3.19) là đã cài đặt thư viện thành công rồi.



Hình 3.19 cài đặt thành công

### 3.3.2 Chạy ví dụ:

Để dễ hiểu hơn thì mình sẽ làm một ví dụ đồng thời sẽ hướng dẫn chi tiết và giải thích các bước nhé.

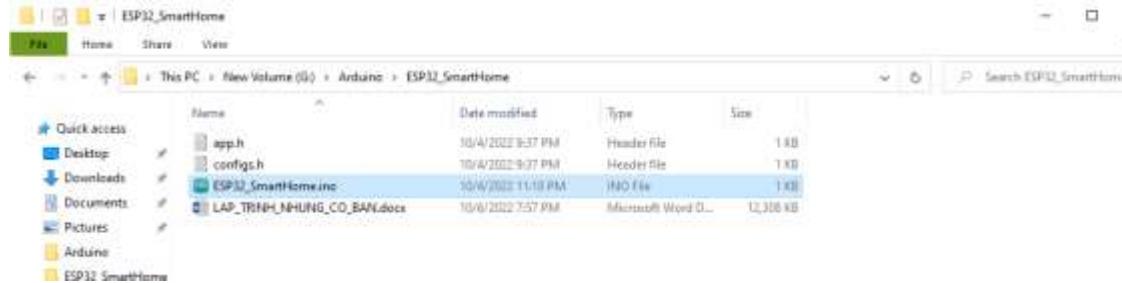
## Chương 1: Cơ sở lý thuyết

Ví dụ này là **kết nối MQTT**, đồng thời test thư viện hôm trước chúng ta đã viết ở chương 2, phần MQTT.

Giờ thì bắt đầu thôi.

**Bước 1:** Mở file **ESP32\_SmartHome.ino** (nếu bạn đặt khác mình thì hãy vào file .ino của bạn đã đặt trước đó nhé.) đường dẫn của mình

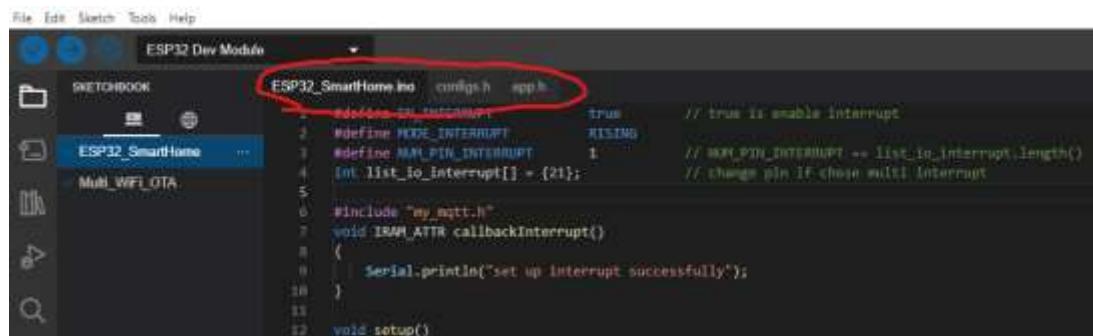
G:\Arduino\ESP32\_SmartHome\ ESP32\_SmartHome.ino



Hình 3.20 mở file .ino

Bạn mở folder project phải đủ ba file nhé (**app.h, configs.h và file .ino**). Và các file **my\_<tên thư viện>.h** đã được chuyển hết vào folder **My\_libraries** rồi nhé.

Mở file .ino phải đủ ba file như hình 3.21 này nhé.



Hình 3.21 file tiêu chuẩn

**Bước 2:** Code ở file **ESP32\_SmartHome.ino** ta chỉ gọi hàm ở thư viện **app.h**

Code: **ESP32\_SmartHome.ino**

```
#include "configs.h"
#include "app.h"

void setup() {
    setupApp();
}
```

**Note:** để lấy full code nhấn vào ô chứa code rồi Ctrl + C. Vào file my\_ota.h trong Arduino rồi Ctrl + V.

## Chương 1: Cơ sở lý thuyết

**Bước 3:** Code ở file **app.h** sẽ là các **hàm mới liên quan đến project và gọi các hàm** mà ta đã viết ở thư viện mà ta đã cá nhân hóa trước đó.

Code file **app.h**:

```
#ifndef __APP_H__  
#define __APP_H__  
  
#include "my_multi_wifi.h"  
#include "my_mqtt.h"
```

**Note: để lấy full code nhấn vào ô chứa code rồi Ctrl + C. Vào file my\_ota.h trong Arduino rồi Ctrl + V.**

**Bước 4:** Code ở file **Configs.h** sẽ là **thay đổi giá trị của tag để phù hợp với project**

Code file **Configs.h**:

```
#ifndef __CONFIGS_H__  
#define __CONFIGS_H__  
  
#define DELAY_TIME_MQTT      1*60          //1m  
#define MQTT_SERVER           "192.168.1.9"  
#define MQTT_PORT              1883  
#define MQTT_ID                "ESP"
```

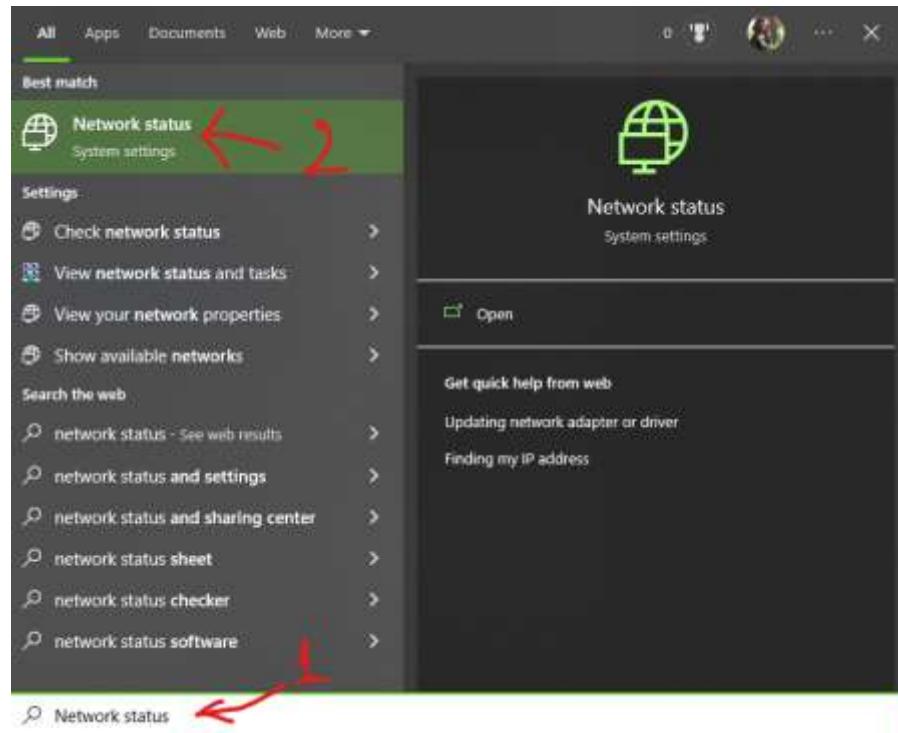
**Note: để lấy full code nhấn vào ô chứa code rồi Ctrl + C. Vào file my\_ota.h trong Arduino rồi Ctrl + V.**

**Lưu ý:** chõ tag **MQTT\_SERVER** các bạn hãy đổi thành địa chỉ mạng máy tính của bạn (như địa chỉ của mình là 192.168.1.9).

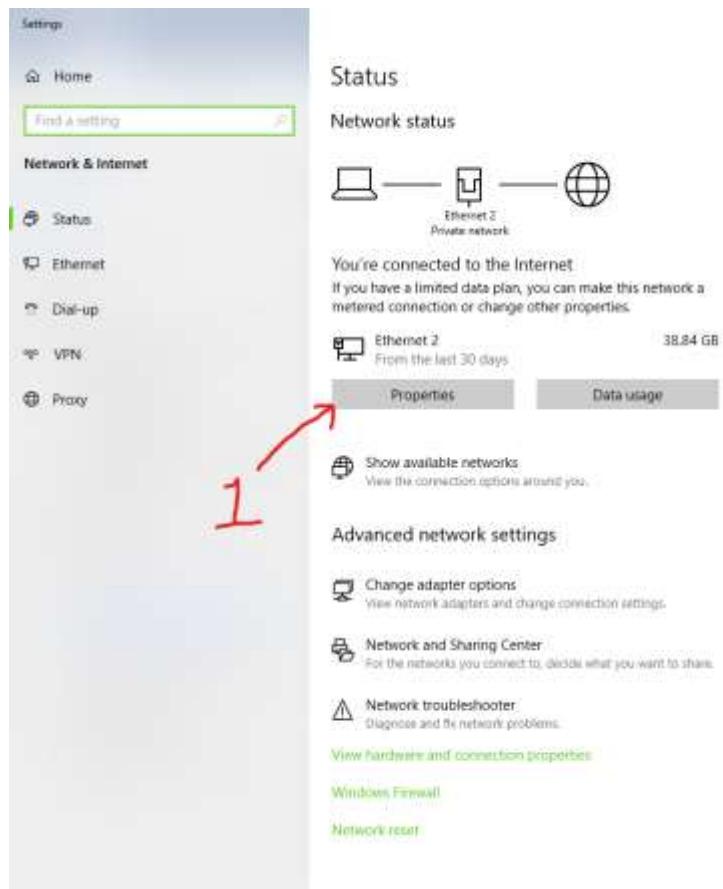
Hình 3.22 đổi địa chỉ server MQTT

Nếu các bạn k biết lấy địa chỉ mạng của máy bạn ở đâu thì làm theo các bước sau.

## Chương 1: Cơ sở lý thuyết



Hình 3.23 Tìm Network status ở thanh tìm kiếm rồi ấn vào



Hình 3.24 nhấp vào Properties

## Chương 1: Cơ sở lý thuyết

### Properties

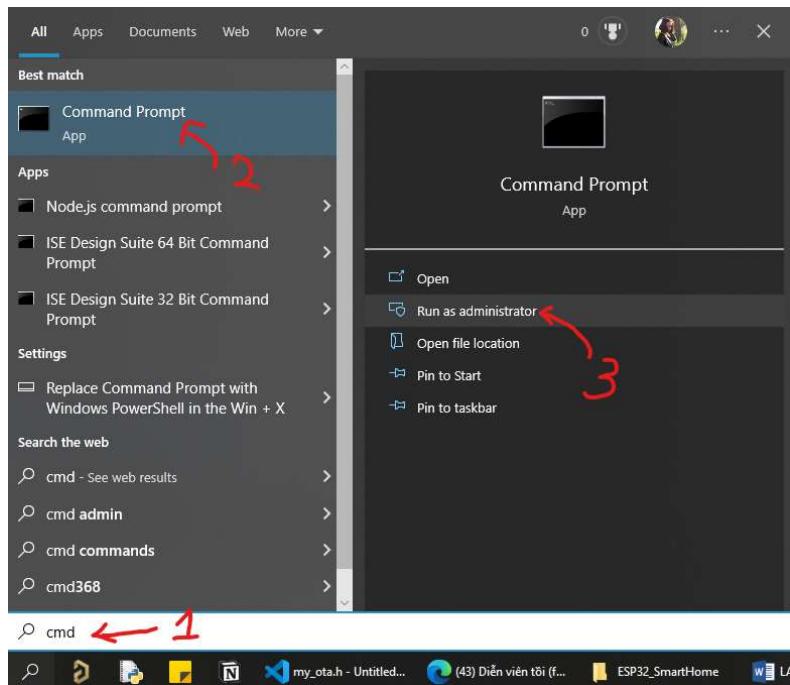
Link speed (Receive/Transmit):	1000/1000 (Mbps)
IPv6 address:	2405:4800:5717:c465:d4ac:b19e:e40b:e24
Link-local IPv6 address:	fe80::d4ac:b19e:e40b:e24%14
IPv4 address:	192.168.1.9
IPv4 DNS servers:	192.168.1.1
Manufacturer:	Realtek
Description:	Realtek Gaming GbE Family Controller
Driver version:	10.36.701.2019
Physical address (MAC):	E0-D5-5E-A6-EA-58

**Copy**

Hình 3.25 kéo xuống **Properties** thì bạn sẽ thấy địa chỉ máy bạn tại đây

Như vậy các bạn setup xong ESP ở thì đến phần Node-red nào.

**Bước 5:** mở cmd và gõ lệnh **node-red** để mở server



Hình 3.26 mở cmd

## Chương 1: Cơ sở lý thuyết

```
C:\WINDOWS\system32>node-red
6 Oct 21:58:37 - [info] Welcome to Node-RED
=====
6 Oct 21:58:37 - [info] Node-RED version: v3.0.2
6 Oct 21:58:37 - [info] Node.js version: v16.17.1
6 Oct 21:58:37 - [info] Windows_NT 10.0.19044 x64 LE
6 Oct 21:58:38 - [info] Loading palette nodes
6 Oct 21:58:39 - [info] Settings file : C:\Users\Admins\.node-red\settings.js
6 Oct 21:58:39 - [info] Context store : 'default' (module:memory)
6 Oct 21:58:39 - [info] User directory : \Users\Admins\.node-red
6 Oct 21:58:39 - [warn] Projects disabled : editorTheme.projects.enabled=false
6 Oct 21:58:39 - [info] Flows file : \Users\Admins\.node-red\flows.json
6 Oct 21:58:39 - [info] Server now running at http://127.0.0.1:1880/
6 Oct 21:58:39 - [warn]

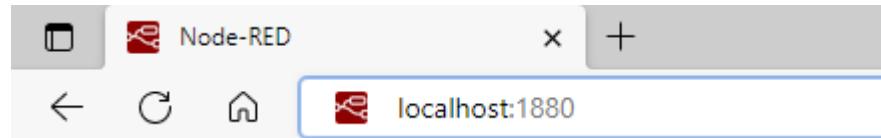
-----
Your flow credentials file is encrypted using a system-generated key.
If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the "credentialSecret" option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.

-----
6 Oct 21:58:39 - [info] Starting Flows
6 Oct 21:58:39 - [info] Started flows
6 Oct 21:58:39 - [info] [aedes broker:458494f5702e8a93] Binding aedes mqtt server on port: 1883
6 Oct 21:58:39 - [info] [mqtt-broker:48eb4024f54530e0] Connected to broker: mqtt://192.168.1.9:1883
```

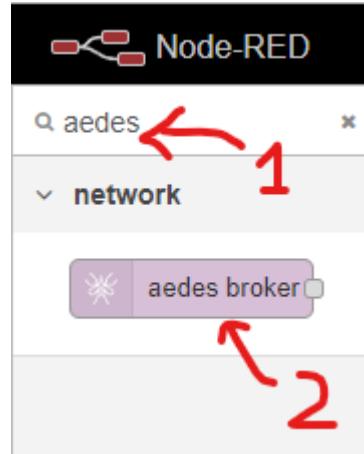
Hình 3.27 sau khi đã nhập lệnh node-red

Bước 6: mở trình duyệt rồi gõ **localhost:1880**



Hình 3.28 mở localhost:1880

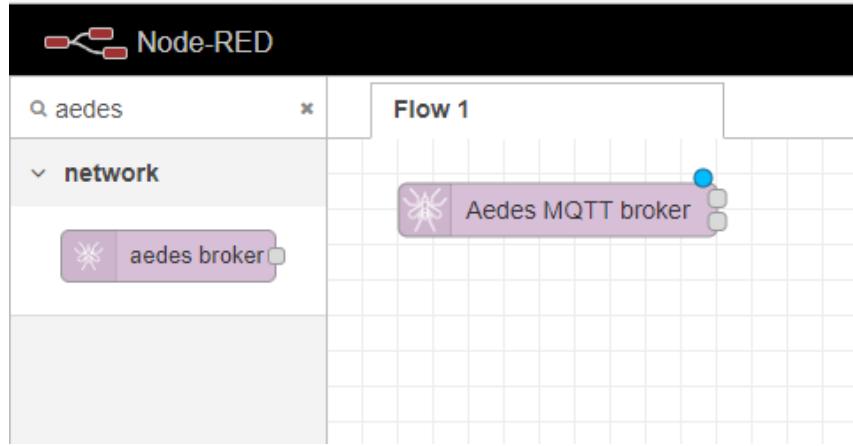
Bước 7: tìm aedes (máy chủ MQTT ảo) ở thanh tìm kiếm sau đó kéo thả vào vùng làm việc.



Hình 3.28 tìm aedes

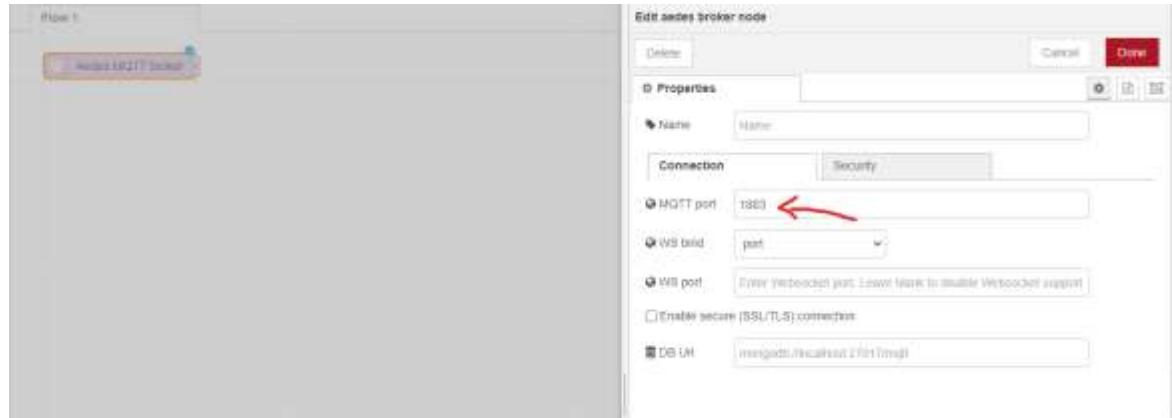
## Chương 1: Cơ sở lý thuyết

Mà nếu đến đây các bạn không thấy cái node nào có tên như trên thì có nghĩa là bạn chưa install thư viện hay là install thư viện bị lỗi rồi nhé. Hãy quay lại phần 3.3.1 d) Thêm các package vào node-red (ở đây mình cũng đã hướng dẫn cách thêm thư viện nhé)



Hình 3.29 kéo node aedes ra vùng làm việc

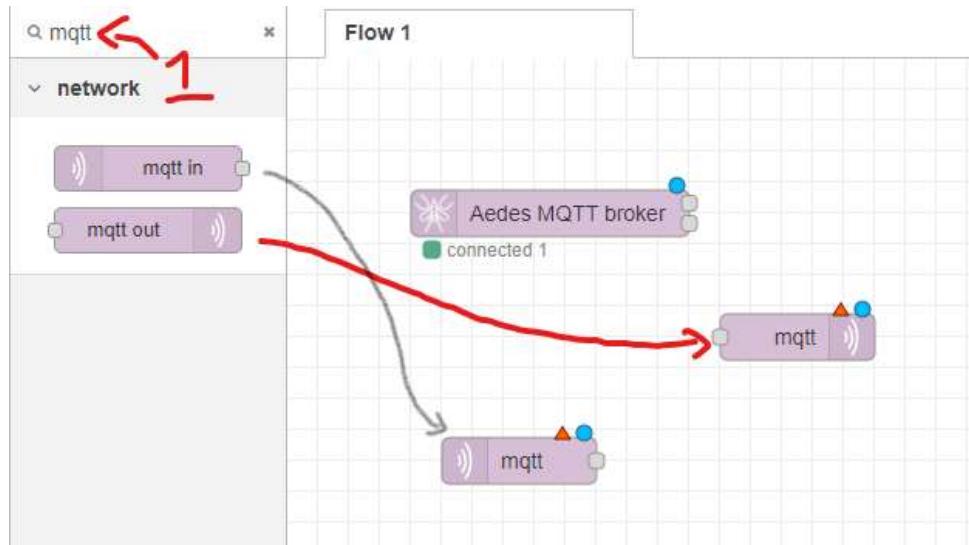
### Bước 8: Nhấn đúp vào node aedes và sửa port MQTT thành 1883



Hình 3.30 sửa MQTT port

### Bước 9: tương tự bước 7 các bạn tìm node mqtt in và node mqtt out rồi sau đó kéo 2 node đó ra ngoài

## Chương 1: Cơ sở lý thuyết



Hình 3.31 kéo mqtt-in và mqtt-out

Bước 10: nhấn đúp vào mqtt out và configs như **hình 3.32 và 3.33**

### (Hình 3.32)

- (1) nhấn đúp vào **mqtt out**
- (2) sửa **Topic** thành **S-ESP** (topic này là tag **TOPIC\_SUB** ở dưới **ESP** bạn đã đăng ký)
- (3) nhấn vào biểu tượng cây bút để thêm địa chỉ server

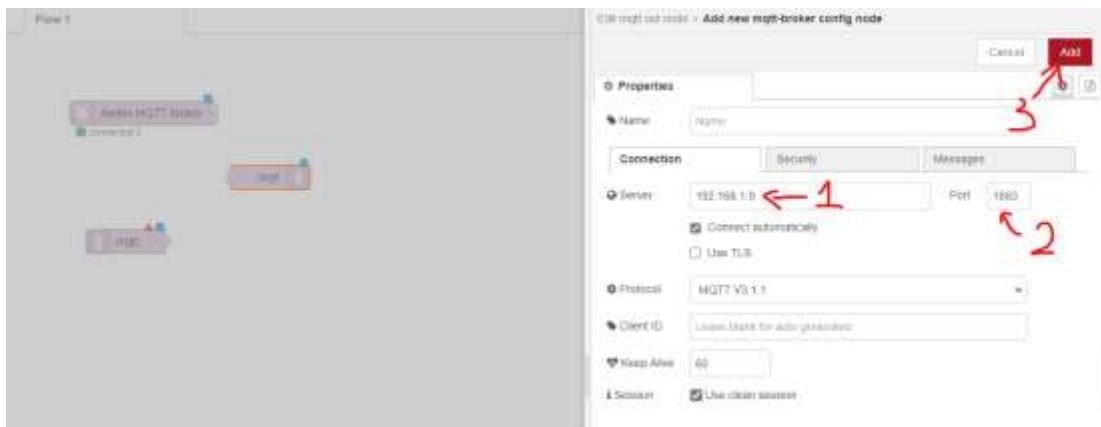


Hình 3.32 kéo thêm topic

### (Hình 3.33)

- (1) sửa **Server** thành <**địa chỉ mạng** của **máy bạn**> (của mình là 192.168.1.9)
- (2) sửa **Port** thành **1883** (port này là tag **MQTT\_PORT** ở dưới **ESP** bạn đã đăng ký)
- (3) nhấn chữ **Add** để lưu server

## Chương 1: Cơ sở lý thuyết



Hình 3.33 thêm địa chỉ server

Sau đó ấn **Done** để hoàn thành

**Bước 11:** nhấn đúp vào **mqtt in** và config như **hình 3.34**

(**Hình 3.32**)

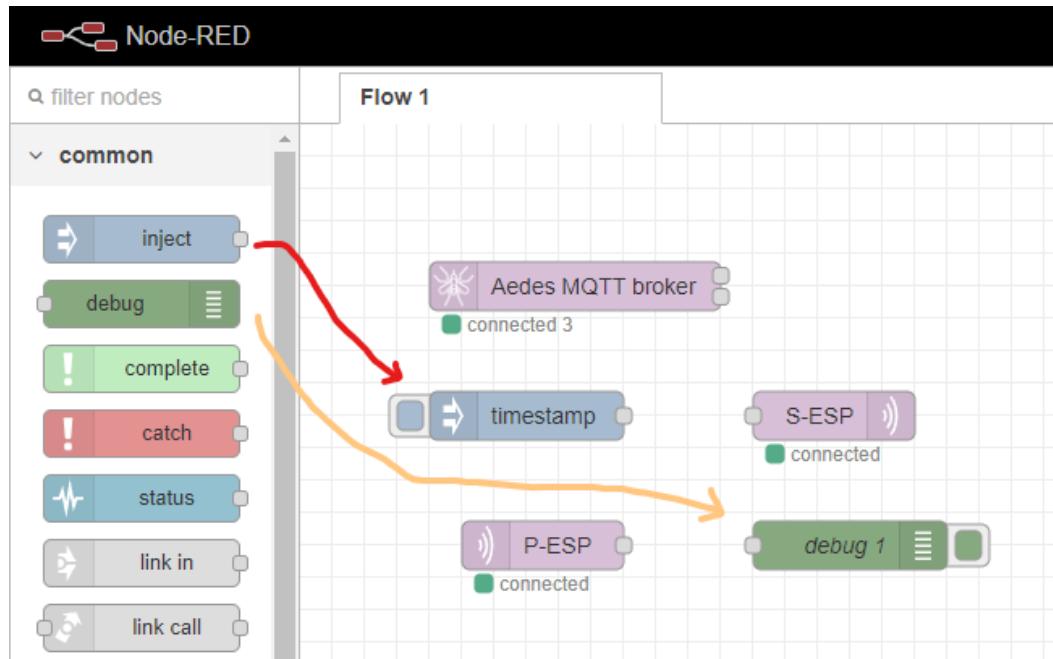
- (0) nhấn đúp vào **mqtt out**
- (1) sửa **Topic** thành **P-ESP** (topic này là tag **TOPIC\_PUB** ở dưới **ESP** bạn đã đăng ký)
- (2) nhán vào **Done** để hoàn thành



Hình 3.34 config mqtt out

**Bước 12:** tìm và kéo node **inject** và node **debug**

## Chương 1: Cơ sở lý thuyết



Hình 3.35 kéo node **inject** và node **debug**

Bước 13: config node **inject**

(hình 3.36)

- (0) nhấn đúp vào node có tên **timestamp**
- (1) nhấn vào mũi tên chỉ xuống
- (2) chọn **String** (bước này để chọn kiểu dữ liệu sẽ gửi về cho esp)

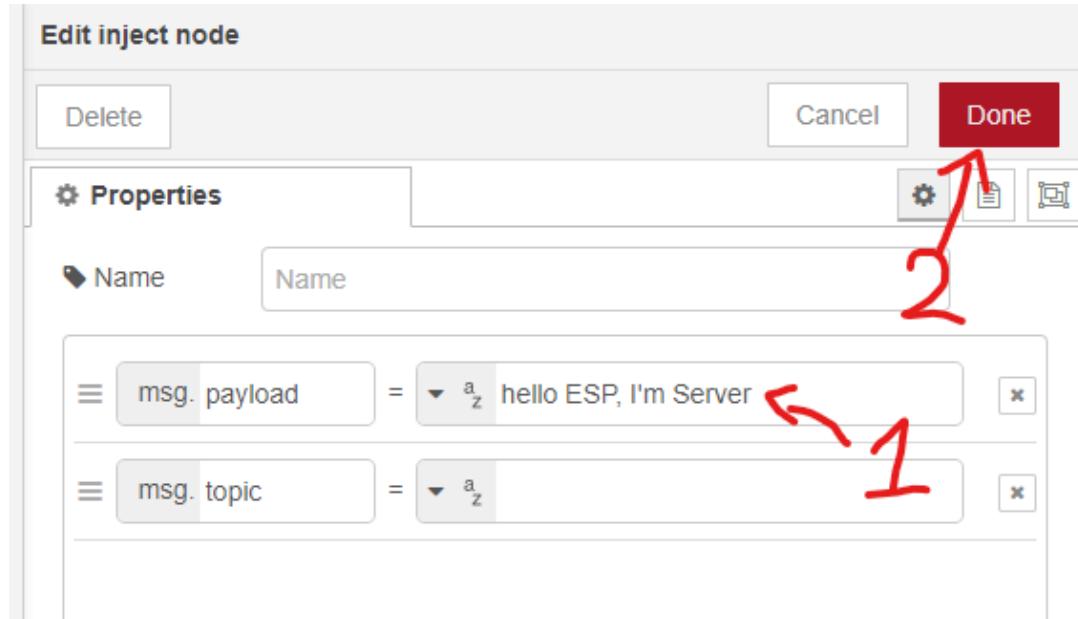


Hình 3.36 config **inject**

(hình 3.37)

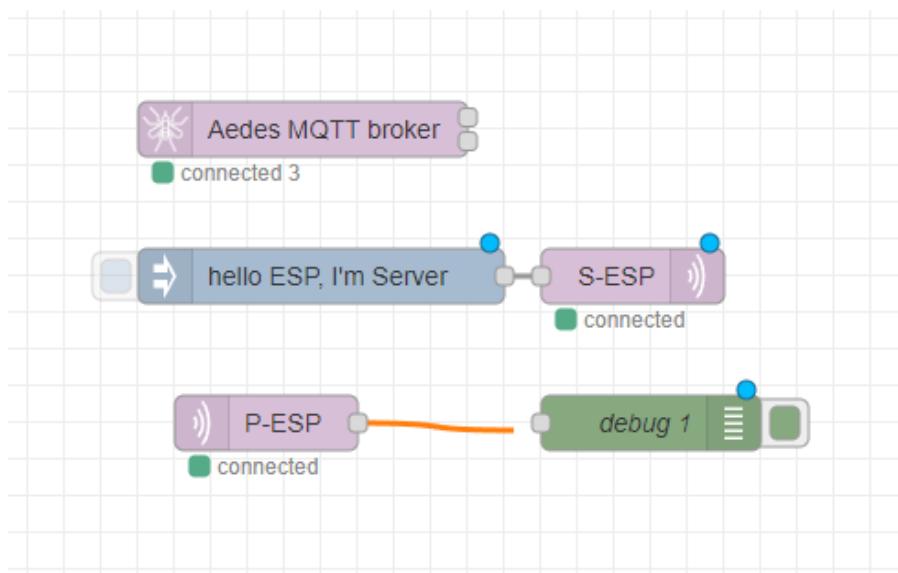
- (1) sửa nội dung tùy thích
- (2) nhấn Done để hoàn thành

## Chương 1: Cơ sở lý thuyết



Hình 3.37 sửa nội dung inject

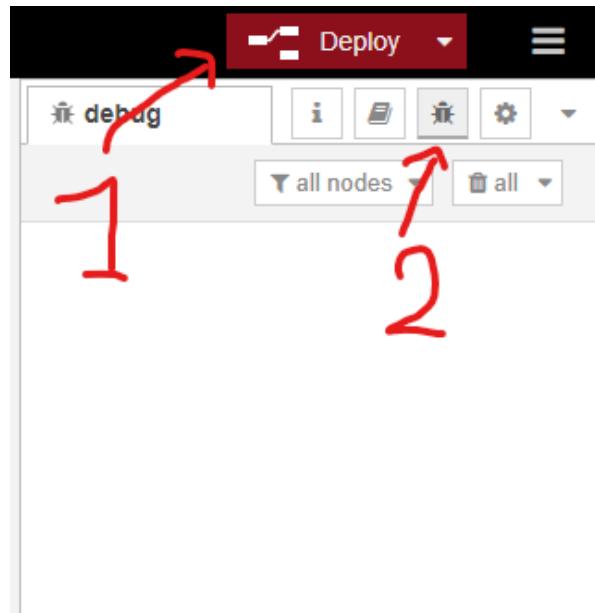
Bước 14: Nối node **inject** với node **mqtt out** và nối node **mqtt in** với node **debug**



Hình 3.38 nối các node lại với nhau

Bước 15: Nhấn nút **deploy** để hoàn tất, sau đó nhấn vào **hình con bọ** để mở bảng hiển thị thông tin

## Chương 1: Cơ sở lý thuyết



Hình 3.39 deploy

Bước 16: Mở lại Arduino rồi sau đó nạp code vào ESP32 để xem kết quả

```
22:53:24.594 -> entry 0x400805f0
22:53:24.969 -> Connecting Wifi...
22:53:30.782 -> _____ WIFI _____
22:53:30.782 -> WiFi connected
22:53:30.782 -> IP address: 192.168.1.10
22:53:30.782 -> SSID: NTGD
22:53:30.782 -> RSSI: -81
22:53:30.782 -> _____
22:53:30.814 -> Connectting MQTT ...
22:53:30.860 -> _____ MQTT _____
22:53:30.860 -> Connected MQTT
22:53:30.860 -> MQTT SERVER:192.168.1.9:1883
22:53:30.860 -> MQTT ID:ESP
22:53:30.860 -> TOPIC SUB:S-ESP
22:53:30.860 -> TOPIC PUB:P-ESP
22:53:30.860 -> _____
```

Hình 3.40 connect thành công MQTT

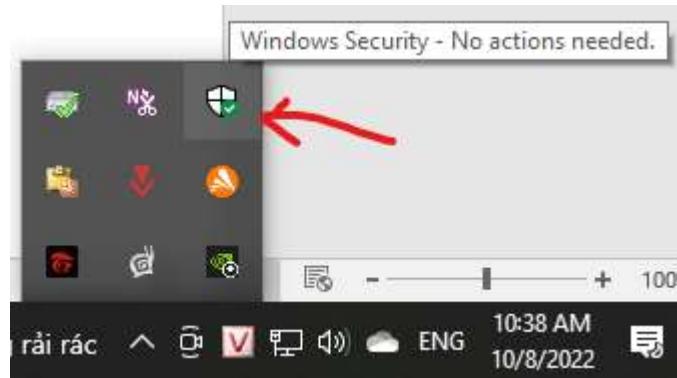
Bước 17 Mở lại node-red ở web để xem kết quả.

Nếu ở Serial báo lỗi thì.

- các bạn kiểm tra lại **wifi\_ssid** và **wifi\_pw** và **NUM\_WIFI** ở file **my\_multi\_wifi.h** có đúng với wifi nhà bạn không? **NUM\_WIFI ==** tổng phần tử trong mảng **wifi\_ssid ==** tổng phần tử trong mảng **wifi\_pw** không?

## Chương 1: Cơ sở lý thuyết

Nếu đã đúng hết và nạp lại code. Nhưng các bạn vẫn bị lỗi connect MQTT thì các bạn hãy **tắt hết tường lửa** trên máy tính của các bạn



### Domain network

Firewall is on.

### Private network (active)

Firewall is on.

### Public network

Firewall is on.

**Ấn vào và off hết cả 3 phần trên.**

Rồi sau đó bạn khởi động lại **node-red** và nạp lại code cho **ESP**. Sau đó mở trình duyệt vào localhost:1880 Các bạn kiểm tra:

- Dưới **node aedes** nếu như **connected 3** thì ESP mới kết nối thành công MQTT. **Connected 2** thì ESP chưa kết nối được với server. Các bạn ấn reset lại ESP (nút EN)

Chờ khoảng 1 phút ta sẽ thấy tin nhắn mà ESP32 gửi lên mà ta đã viết ở hàm **mqttSendServer** ở phần **loopApp()** trong file **App.h**

## Chương 1: Cơ sở lý thuyết

```
void loopApp()
{
    loopWiFiMulti();

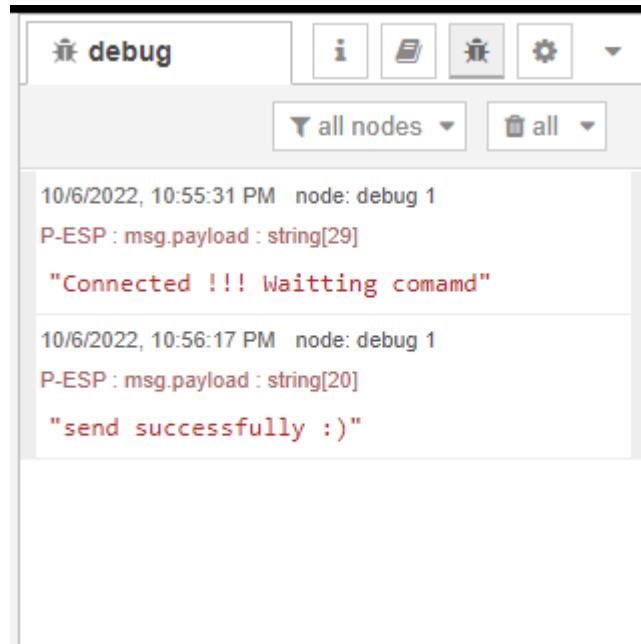
    delay(50);

    loopMQTT();

    String data_esp_send_server = "send successfully :)";
    mqttSendServer(data_esp_send_server);

}
```

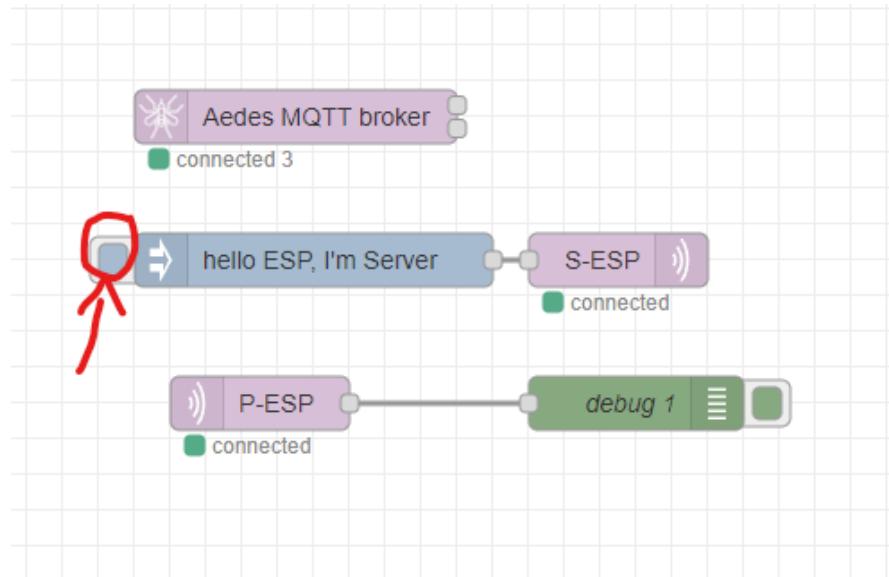
Hình 3.41 hàm gửi thông tin đi cho server



Hình 3.42 thông tin mà node-red đọc được

**Bước 18** Nhấn vào nút inject của node inject để Server gửi tin nhắn xuống ESP.

## Chương 1: Cơ sở lý thuyết



Hình 3.43 nhấn vào nút inject

Bước 19 quay lại Serial monitor ở Arduino để xem kết quả mà Server gửi về

The Serial Monitor window displays the following log output:

```
Output Serial Monitor X

Message (Ctrl + Enter to send message to 'ESP32 Dev Module' on 'COM3')

22:55:31.470 -> WiFi connected
22:55:31.470 -> IP address: 192.168.1.10
22:55:31.470 -> SSID: NTGD
22:55:31.470 -> RSSI: -83
22:55:31.470 -> _____
22:55:31.516 -> Connectting MQTT ...
22:55:31.563 -> _____MQTT_____
22:55:31.563 -> Connected MQTT
22:55:31.563 -> MQTT SERVER:192.168.1.9:1883
22:55:31.563 -> MQTT ID:ESP
22:55:31.563 -> TOPIC SUB:S-ESP
22:55:31.563 -> TOPIC PUB:P-ESP
22:55:31.563 -> _____
23:07:25.572 -> Recived from: S-ESP
23:07:25.572 -> hello ESP, I'm Server
23:07:25.572 -> command from Server: hello ESP, I'm Server
```

Hình 3.44 kết quả sau khi Server gửi xuống MQTT

Hình 3.35 là hàm mà sau khi ESP đọc xong lệnh thì sẽ in ra chữ “**command from Server: Hello ESP, I'm Server**”

## Chương 1: Cơ sở lý thuyết

```
// handle Command from Server
void executeMqttCommand(String command)
{
    Serial.println("command from Server: " + command);
}
```

Hình 3.45 hàm xử lý lệnh

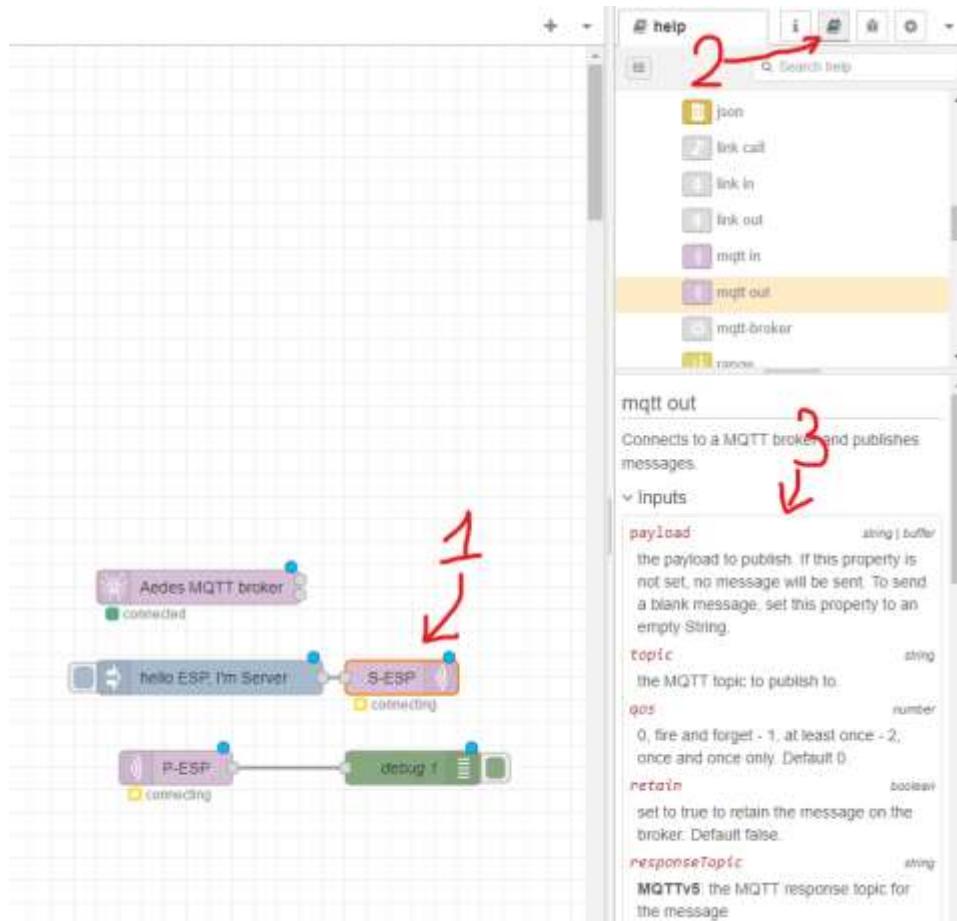
NOTE: Vì ở Node-red có rất nhiều node nên mình khó có thể liệt kê chúng và hướng dẫn chi tiết từng node được. Nên các bạn có thể làm các bước sau để có thể đọc được thông số đầu vào cần thiết và đầu ra của node đấy.

Bước 1: các bạn khởi động Server (mở cmd nhập lệnh **node-red**)

Bước 2: mở trình duyệt **localhost:1880**

Bước 3: (**hình 3.46**)

- (1) Chọn node cần tìm thông tin
- (2) Nhấn vào quyền sách để xem thông tin
- (3) Thông tin chi tiết về các thuộc tính có trong node



## Chương 1: Cơ sở lý thuyết

Hình 3.46 xem thông tin props

Như vậy ta đã biết được cách thức hoạt động của MQTT và đồng thời cũng đã làm quen với các thao tác trên node-red. Và Ở chương tiếp theo mình sẽ hướng dẫn về MySQL

## CHƯƠNG IV: MY SQL

### 4.1 Tổng quan:



Hình 4.1 Biểu tượng MySQL

Nói nôm na MySQL là hệ quản trị cơ sở dữ liệu tự do. MySQL là hệ quản trị phổ biến nhất thế giới và được các nhà phát triển rất ưa chuộng trong quá trình phát triển ứng dụng.

Và MySQL cũng cần một nơi để có thể lưu trữ các dữ liệu ấy. Và máy tính cá nhân của bạn cũng có thể làm được điều ấy.

Và để làm quen và đơn giản hóa MySQL thì mình sẽ giảm thiểu tối đa sử dụng lệnh query trên MariaDB mà mình sẽ sử dụng MySQL Workbench để dễ quản lý bằng những click chuột. Và đây là cơ bản nên mình sẽ hướng dẫn cơ bản nhất có thể.

### 4.2 Cài đặt MySQL:

Vì Node-red chạy trên nền tảng NodeJS nên trước khi cài Node-red thì ta phải cài NodeJS trước. Các bạn làm theo mình nhé.

**Bước 1:** tải MySQL Installer (mysql-installer-community-8.0.30.0.msi)

Link tải: [MySQL :: Download MySQL Installer](#)

## Chương 1: Cơ sở lý thuyết



Hình 4.2 Down bản mysql-installer-community-8.0.30.0.msi

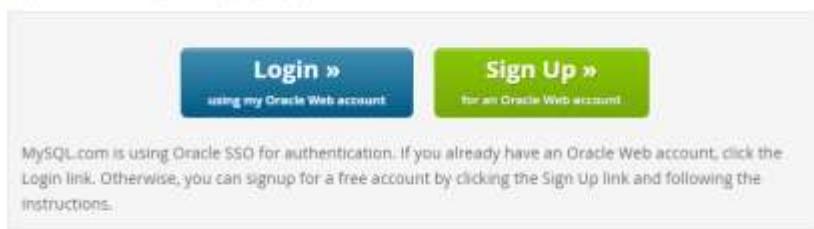
**Bước 2:** Các bạn nhấn **No thanks, just start my download.**

### ④ MySQL Community Downloads

Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

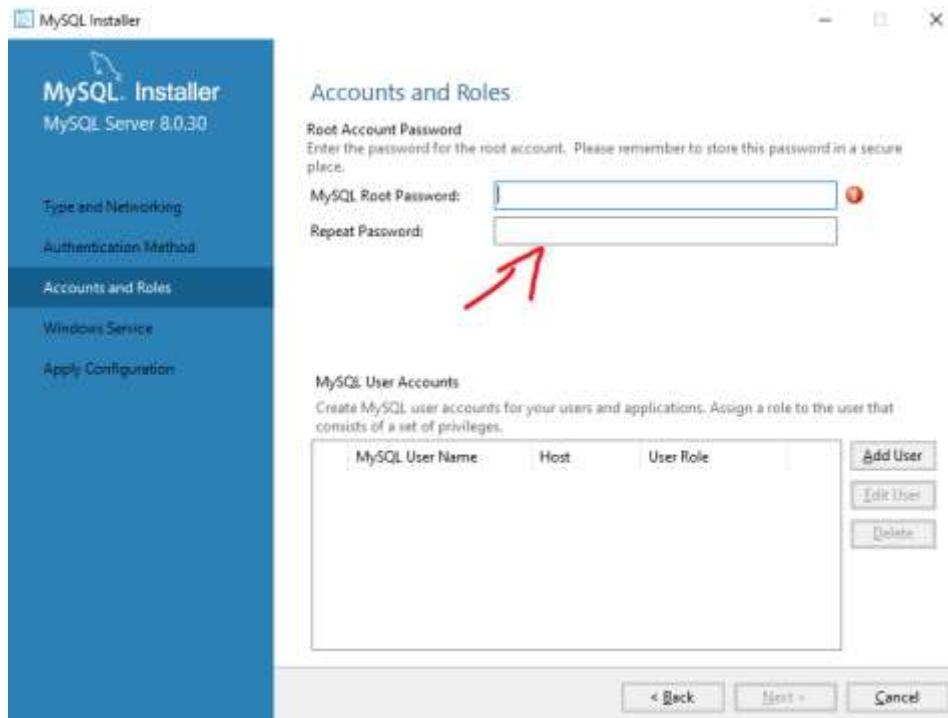
- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system



Hình 4.3 Chọn just start

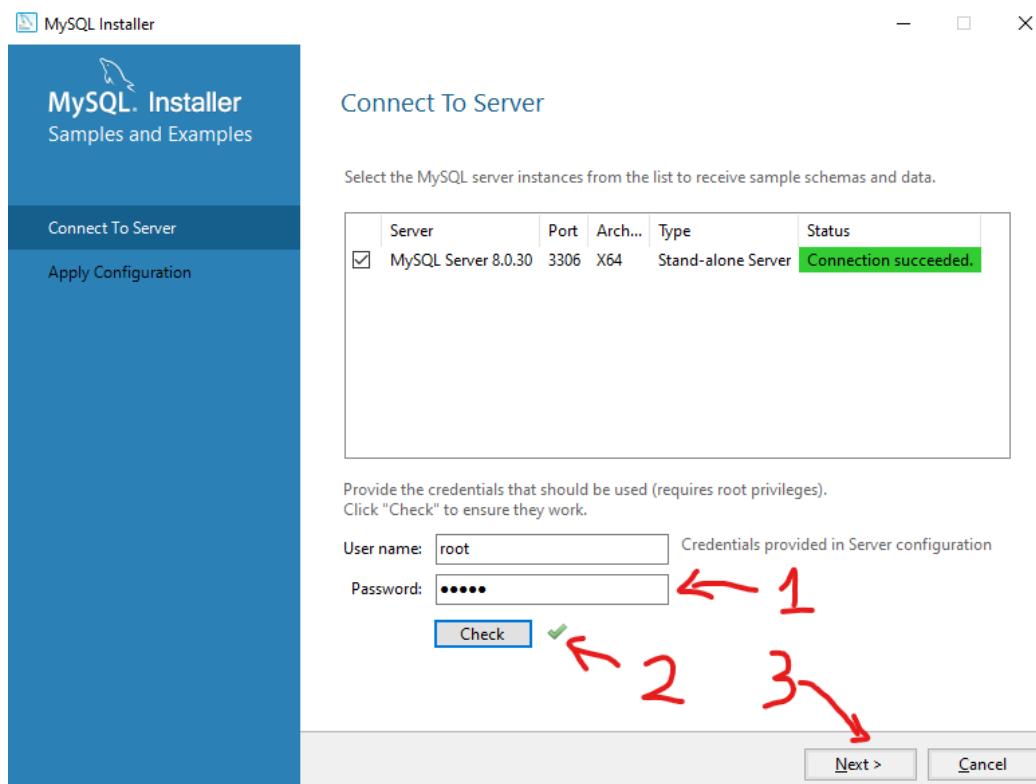
**Bước 3:** nhấn vào file setup vừa tải xuống rồi ấn **Next -> Execute -> Next -> Next...** đến khi nào hiển thị nhận mật khẩu root thì các bạn nhập mật khẩu của bạn vào (nhớ đặt dễ nhớ thôi nhe.)

## Chương 1: Cơ sở lý thuyết



Hình 4.4 Nhập mật khẩu

**Bước 4:** rồi nhấn **Next**, **Execute** và **Finish** rồi **Next** tiếp đến đây thì nhập mật khẩu bạn vừa nhập trước đó. Rồi **Next**, **Execute** và **Finish**

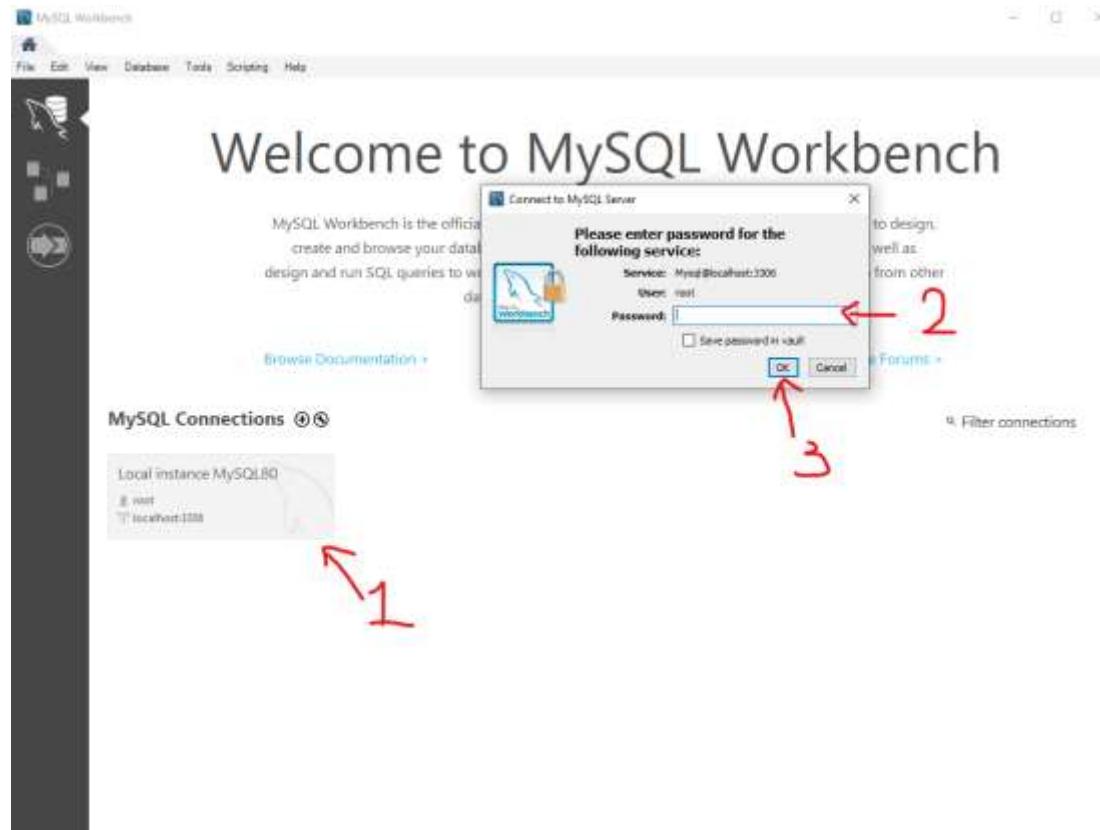


Hình 4.5 Connect server

## Chương 1: Cơ sở lý thuyết

### Bước 5: mở MySQL Workbench ra

- (1): nhấn vào server.
- (2): nhập mật khẩu bạn vừa tạo lúc cài đặt
- (3): nhấn oke



Hình 4.6 mở Database Server

Như vậy ta đã hoàn thành bước cài đặt MySQL

### 4.3 Hướng dẫn sử dụng cơ bản MySQL

Để tránh mất thời gian thì ở phần này mình chỉ giới thiệu cơ bản thôi. Còn chi tiết các sử dụng thì trong project ESP32\_SmartHome mình sẽ hướng dẫn kỹ hơn nhé.

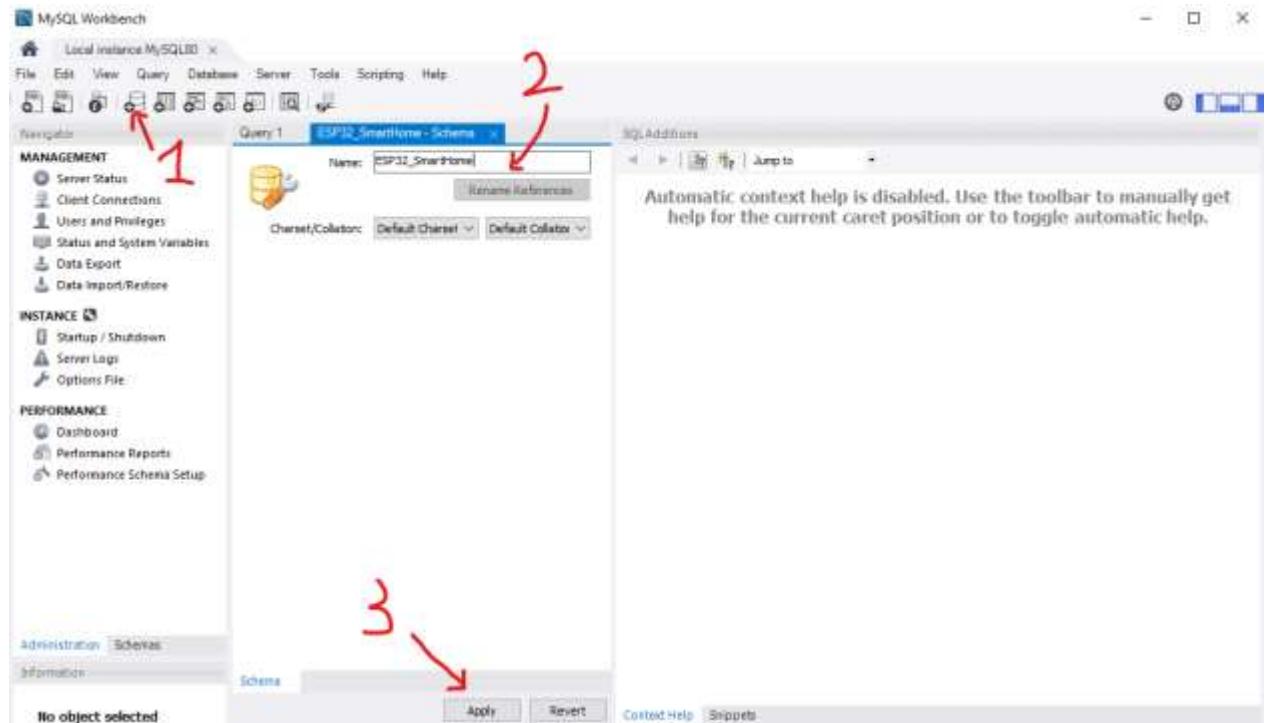
#### 4.3.1 Sử dụng cơ bản:

##### a) Tạo một Database:

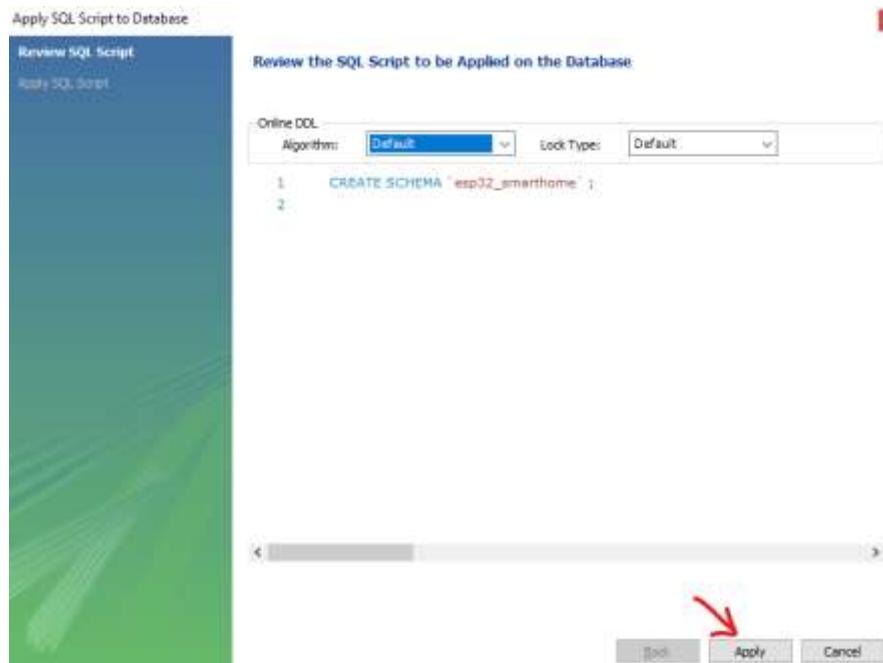
(Hình 4.7)

- (1): chọn biểu tượng tạo database
- (2): Đặt tên database (mình đặt trùng với tên Project luôn để dễ quản lý)
- (3): nhấn Apply để áp dụng

## Chương 1: Cơ sở lý thuyết



Hình 4.7 tạo database



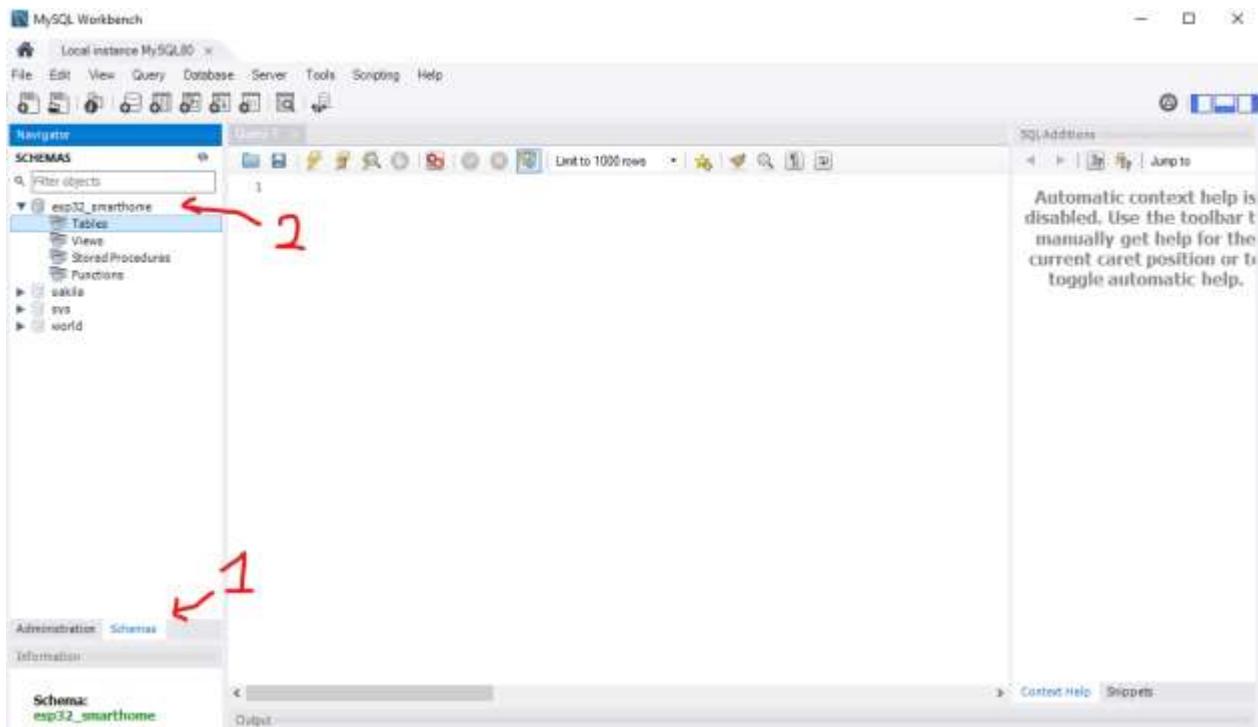
Hình 4.8 tạo database

### b) Tạo một Table trong Database :

(Hình 4.7)

- (1): chọn Schemas
- (2): chọn Database bằng cách nhấp double click vào tên Database

## Chương 1: Cơ sở lý thuyết

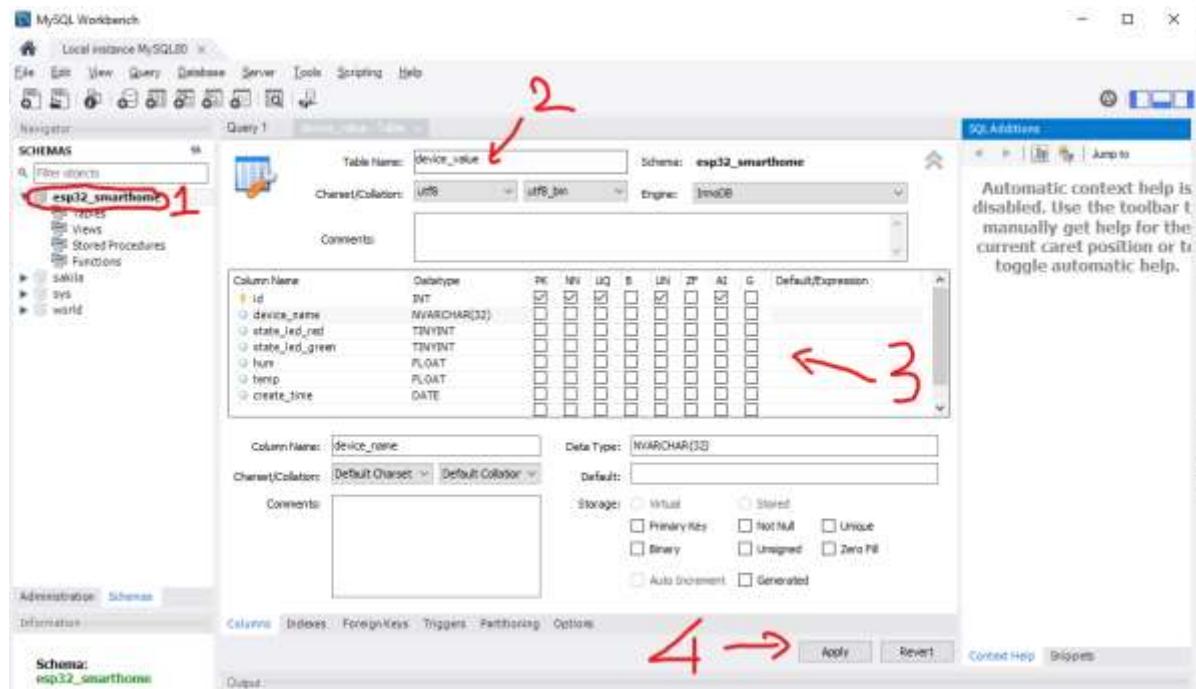


Hình 4.9 tạo table

### (Hình 4.7)

- (1): chọn tên Database (chữ phải có màu đậm nhé)
- (2): đặt tên cho bảng (mình đặt là device\_value)
- (3): tạo các cột thông số như mình nhé
  - o **id (INT) chọn PK, NN, UQ, UN, AI**
  - o **device\_name (NVARCHAR)**
  - o **state\_led\_red (TINYINT)**
  - o **state\_led\_green (TINYINT)**
  - o **hum (FLOAT)**
  - o **temp (FLOAT)**
  - o **create\_time (DATE)**
- (4): ấn Apply say đó nhấn Apply tiếp để confirm.

## Chương 1: Cơ sở lý thuyết



Hình 4.10 tạo table

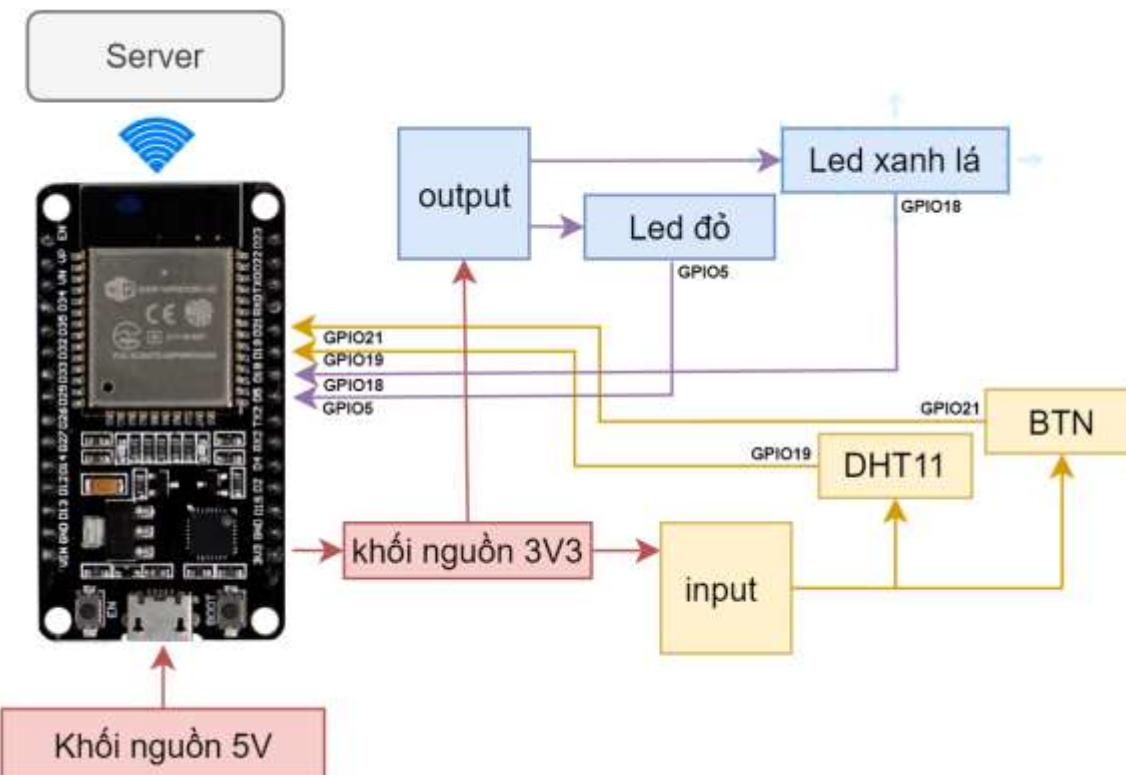
Như vậy ta đã xong các bước để tạo 1 database và 1 bảng cho project của chúng ta . Các bạn có thể tự tìm hiểu các lệnh query data thì các bạn có thể lên gg search để tìm nhé.Còn các lệnh mình sử dụng ở phần sau thì vào Chương 5 mình sẽ nói rõ hơn hé.

## CHƯƠNG V : PROJECT ESP32\_SMARTHOME

### 5.1 Thiết kế hệ thống:

Ở Project này sẽ có 2 led (một led đỏ, một led xanh lá), một nút nhấn và một cảm biến độ ẩm (DHT11) được mô phỏng như một căn phòng thông minh. Căn phòng này được **điều khiển** trên website (Node-red dashboard UI). **Quản lý** thông tin phần cứng trên Node-red và được lưu trữ dữ liệu ở MySQL.

#### 5.1.1 Thiết kế sơ đồ khái niệm phần cứng :

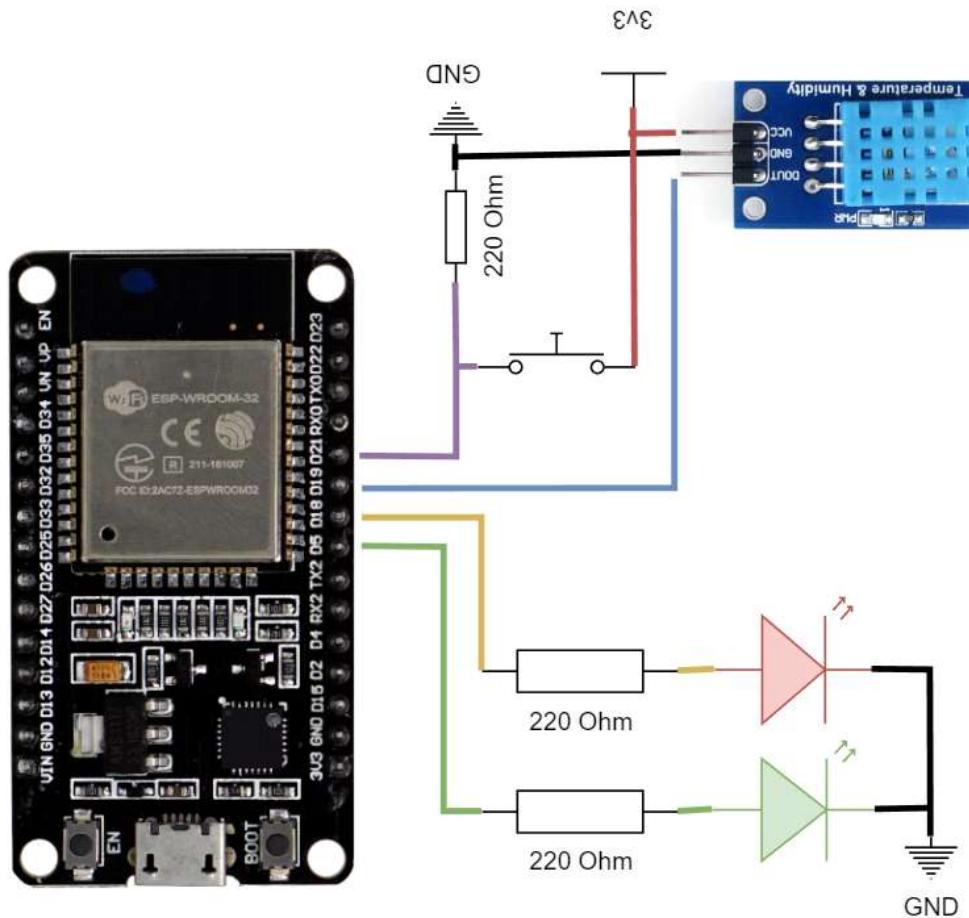


Hình 5.1 Sơ đồ khái niệm của toàn hệ thống

- **Khối nguồn (5V)** cấp điện áp cho ESP32 và **khối nguồn (3V3)** cấp điện áp cho các khái niệm khác.
- **Khái niệm input (nút nhấn và dht11):** là khái niệm input của ESP, khái niệm này chỉ nhận tín hiệu (ở đây DHT11 mìn hìngh nối chân **GPIO19** và **BTN** mìn hìngh nối chân **GPIO21**)
- **Khái niệm output (2 led đơn):** là khái niệm output của ESP, khái niệm này sẽ xuất tín hiệu. (ở đây **Led đỏ** mìn hìngh nối chân **GPIO18** và **Led xanh lá** mìn hìngh nối chân **GPIO5**)
- **Khái niệm Server** ở đây là nơi mìn hìngh sẽ xử lý dữ liệu để hiển thị ra màn hình đồng thời lưu dữ liệu vào database

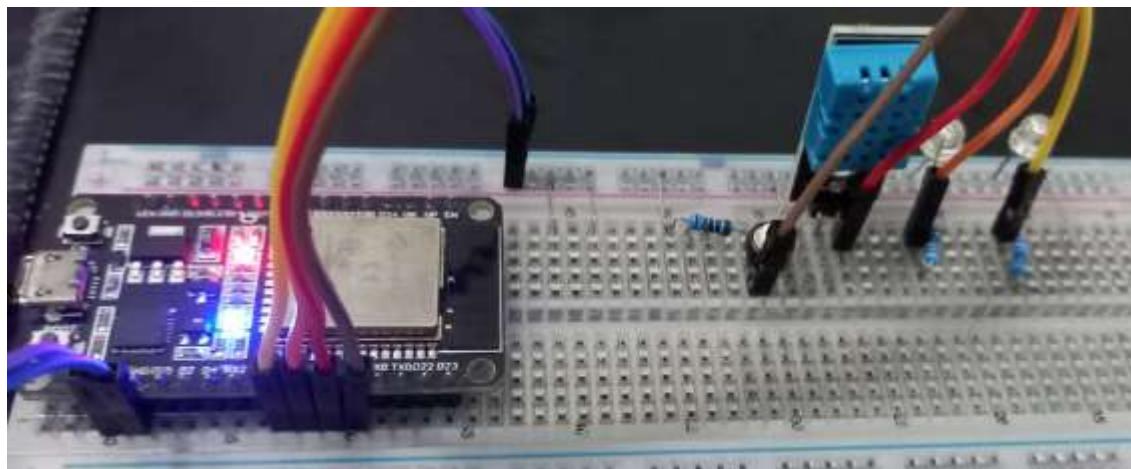
## Chuong 1: Cơ sở lý thuyết

### Schematic ESP32\_SmartHome



Hình 5.2 Schematic ESP32\_SmartHome

### Mạch thực tế project ESP32\_SmartHome

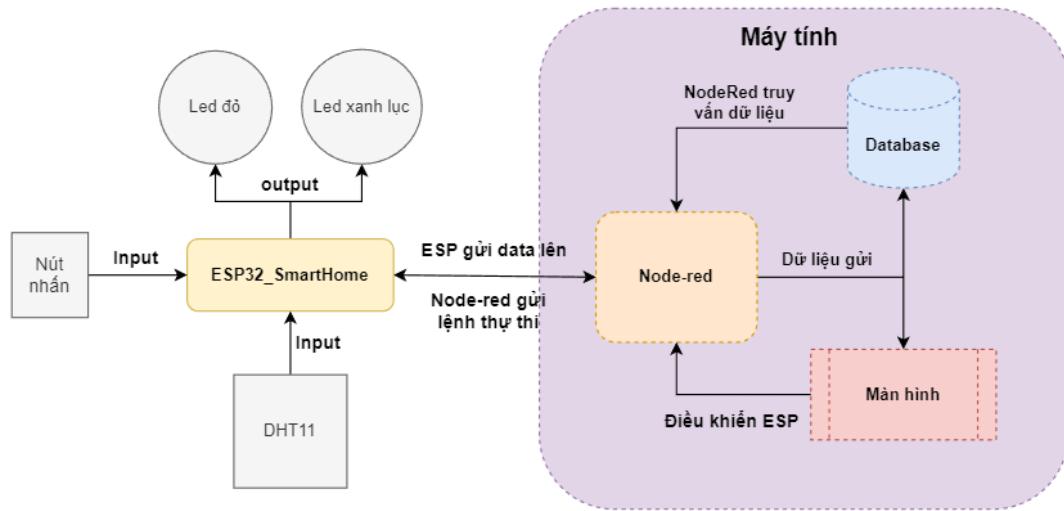


Hình 5.3 mạch thực tế project ESP32\_SmartHome

### 5.1.2 Thiết kế thuật toán hệ thống:

#### a) Phân tích thiết kế hệ thống:

Ta có sơ đồ khối như sau:



Hình 5.4 Sơ đồ phân tích chi tiết hệ thống

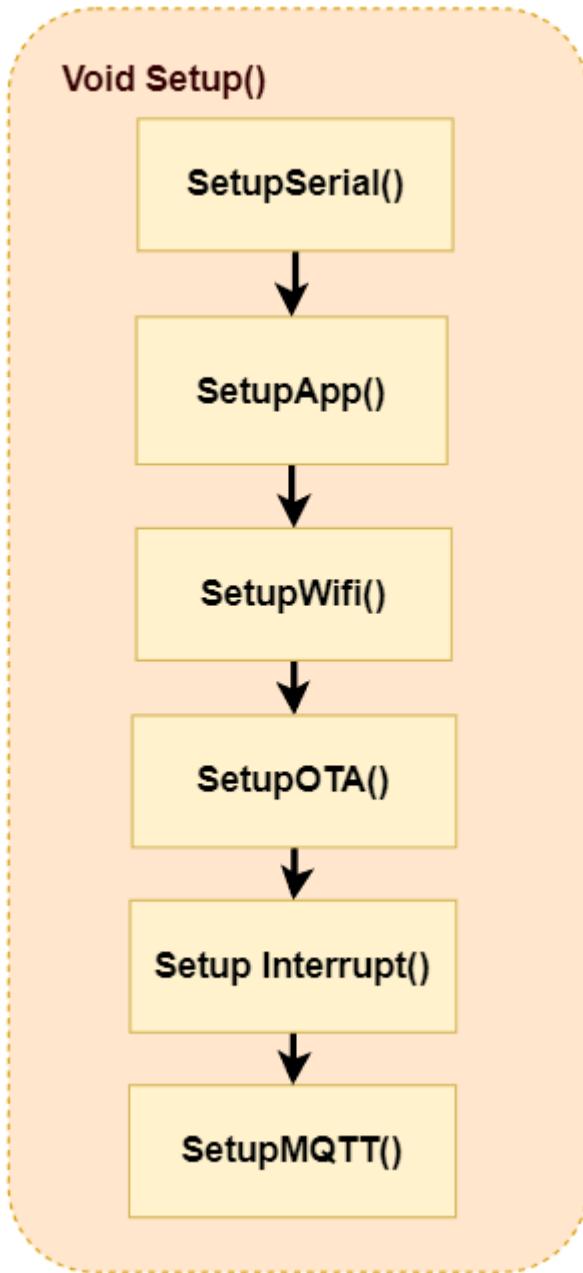
Ta có 1 vi xử lý ESP 32, một database, một MQTT Broker (dùng Node-red để kết nối Database, Màn hình và ESP)

- **ESP32\_SmartHome:** điều khiển và nhận lệnh điều khiển từ Node-red thông qua MQTT Broker – MQTT Server (sử dụng Node-red để mô phỏng một Broker ảo).
- **Node-red:** là nơi kết nối phần cứng ESP và Server trên máy tính.
- **Database:** là nơi nhập và truy xuất dữ liệu.
- **Màn hình:** là nơi hiển thị cho người dùng dễ quan sát và quản lý.

#### b) Phân tích thiết kế logic code:

Vì ta đã cá nhân hóa các thư viện ở chương 2 trước đó rồi nên giờ thì chỉ cần gọi ra và áp dụng vào bài thôi. Ở project này cũng như các project khác đều có 3 file (<project\_name>.ino, app.h, configs.h)

- **File *ESP32\_SmartHome.ino*:**
  - + *Void Setup()*:



Hình 5.5 Sơ đồ logic code void Setup

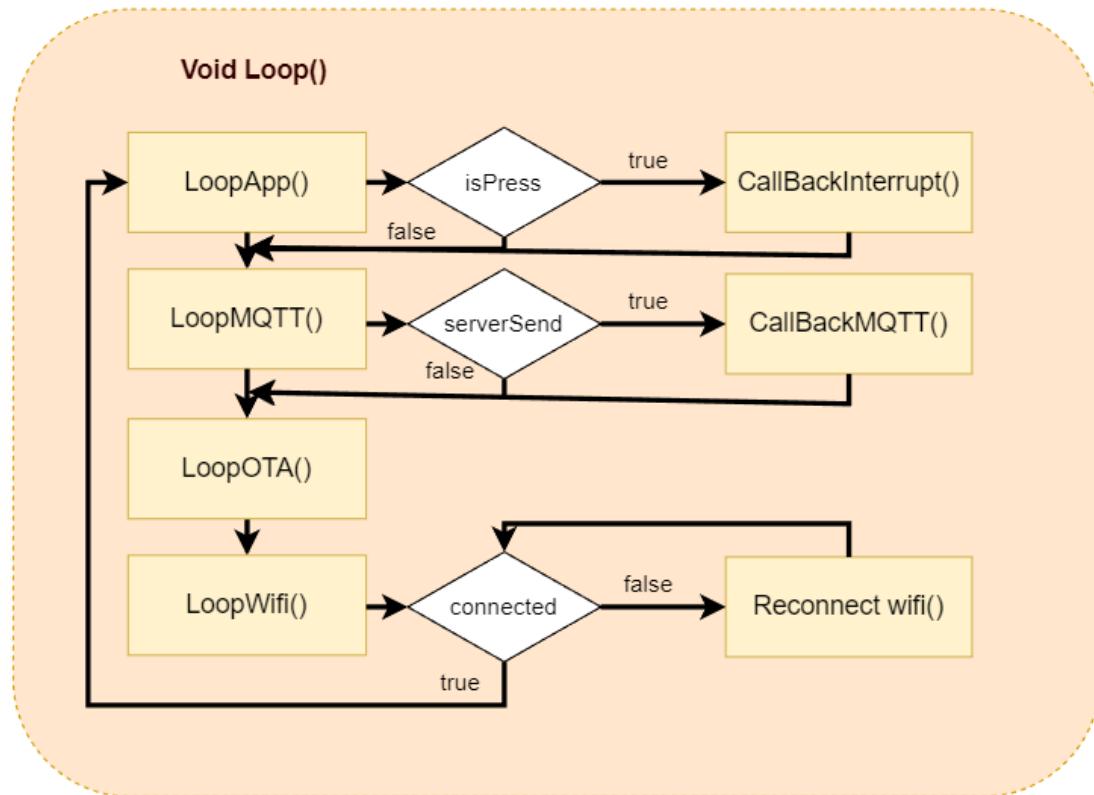
Ở phần Void Setup() sẽ là hàm cài đặt các thông số cần thiết để chạy được project SmartHome

Hàm này chỉ đi qua một lần và ta sẽ thực hiện một cách tuần tự từ trên xuống dưới:

- Đầu tiên **SetupSerial()** là khai báo serial và **OI led thông báo** để ta có thể debug sẽ dễ dàng hơn.
- **SetupApp()** là khai báo thiết bị có sử dụng trong project

## Chương 1: Cơ sở lý thuyết

- Sau đó là **SetupWifi()** như tên gọi của hàm thì ở phần này ta đặt các thông số về wifi
- **SetupOTA()** giúp ta có thể upload code thông qua wifi mà không cần dùng đến dây USB để upload
- **Setup Interrupt()** khai báo chân có sự kiện ngắn (ở project này là khai báo chân 21 là chân nút nhấn)
- Cuối cùng ta **setupMQTT()** để ESP có thể sẵn sàng giao tiếp với sever thông qua giao thức MQTT.  
+ **void Loop():**



Hình 5.6 Sơ đồ logic code void Loop

Ở phần Void Loop() sẽ là hàm mà ESP sẽ luôn luôn chạy. Nên ở sơ đồ trên ta thấy ở hàm cuối cùng có mũi tên vòng lại hàm đầu tiên. Và ta có 2 hàm callback, một cái là của Interrupt và một cái của Server gửi data xuống.

- **LoopApp():** là nơi duy trì các lệnh thực hiện trong app.h trong đó bao gồm cả lệnh chờ callback từ interrupt
- **LoopMQTT():** ngoài việc duy trì kết nối với server thì còn luôn luôn kiểm tra xem Server có gửi lệnh không để thực hiện hàm callback.
- **LoopOTA():** duy trì trạng thái luôn luôn có thể cập nhật bằng wifi.

## Chương 1: Cơ sở lý thuyết

- **LoopWifi()**: kiểm tra kết nối mạng. Nếu bị mất kết nối mạng thì sẽ kết nối lại đến khi nào được thì khởi động lại chip
- **File Configs.h:**

Ở file này là nơi ta sẽ khai báo các tag cục bộ cũng như một vài biến khác sử dụng trong thư viện. Ở chương trước mình đã hướng dẫn các bạn, cho tất cả các file cá nhân hóa thư viện vào mục libraries nên khi mở file .ino của project ESP32\_SmartHome lên thì ta không thể thấy được các file thư viện để mà chỉnh sửa. Chính vì vậy ta sẽ sử dụng tag để thay đổi các giá trị trong thư viện.

```
//Wifi
const char *wifi_ssid[]{"THENAM", "NTGD"};
const char *wifi_pw[] = {"0964941600", "112233445566"};
#define NUM_WIFI          2           // NUM_WIFI == wifi_pw.length() == wifi_ssid.length()
#define LED_STT_DELAY_TIME 60         // 60s

//ota
#define OTA_HOSTNAME      "ESP-%06X"    // can change 'ESP', can't change '-%06X'

//interrupt
int list_io_interrupt[] = {21};           // list_io_interrupt.length() == NUM_PIN_INTERRUPT
#define NUM_PIN_INTERRUPT   1
#define EN_INTERRUPT        true        // if use interrupt, will be change false to true
#define MODE_INTERRUPT       RISING     // 5 mode: RISING, FALING, LOW, HIGH, CHANGE

//MQTT
#define DELAY_TIME_MQTT    1*60        //1m
#define MQTT_SERVER         "192.168.1.3" //your IP
#define MQTT_PORT           1883
#define MQTT_ID             "ESP"       // can change
#define TOPPIC_SUB          "S-ESP"     // can change 'ESP', can't change 'S-'
#define TOPPIC_PUB          "P-ESP"     // can change 'ESP', can't change 'P-'

//Application
#define RED_LED_PIN         18
#define GREEN_LED_PIN        5

#define DHT_PIN              19
#define DHTTYPE            DHT11|
```

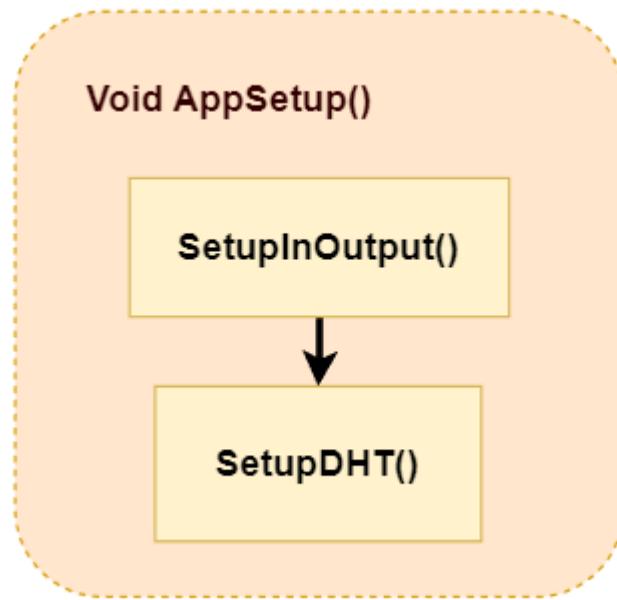
Hình 5.7 các tag để có thể phù hợp với project hơn

Ta chỉ cần thay đổi các tag mà không cần vào thư viện để chỉnh sửa chúng.

- **File app.h :**

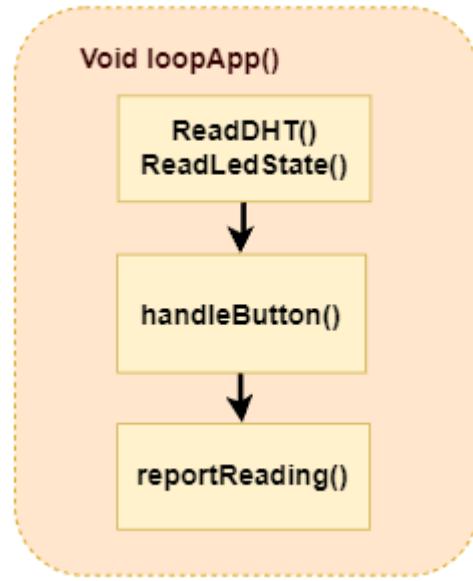
Ở file này ta sẽ code những ứng dụng liên qua đến project. Như ở project SmartHome này thì ta sẽ

+ **Void setupApp():**



Hình 5.7 Sơ đồ logic code void setupApp()

- SetupInOutput() là hàm khai báo chân IO
  - SetupDHT() nơi mà để khai báo sử dụng thiết bị DHT nào?
- + *Void loopApp():*



Hình 5.7 Sơ đồ logic code void setupApp()

- **readDHT()** và **ReadLedState()** là hai hàm để thu thập dữ liệu của DHT đồng thời lấy giá trị trạng thái của led hiện tại

## Chương 1: Cơ sở lý thuyết

- **handleButton()** hàm này để xử lý nút. Vì mình có 1 nút mà có 2 đèn nên ở đây mình sẽ xử lý các sự kiện về nút nhấn.
- và cuối cùng là **reportReading()**. Hàm gửi dữ liệu đi cho Server với chu kỳ 1 phút 1 lần.

### 5.1.3 Hoàn thiện firmware:

#### Giải thích và full code:

Vì ta có 3 file (.ino, app.h, configs.h) nên mình sẽ đi tuần tự từng file theo đúng như máy compile.

##### a) configs.h:

Ở file này như mình đã nói ở phần 5.1.2 b) thì là nơi ta sẽ khai báo các tag ở trong thư viện và các tag ta sẽ dùng ở trong project

```
//Wifi
const char *wifi_ssid[]{"THENAM", "NTGD"};
const char *wifi_pw[] = {"0964941600", "112233445566"};
#define NUM_WIFI          2           // NUM_WIFI == wifi_pw.length() == wifi_ssid.length()
#define LED_STT_DELAY_TIME 60         // 60s
```

Hình 5.8 Các tag liên quan đến wifi

(hình 5.8) Ở phần này các bạn cần lưu ý **NUM\_WIFI** phải bằng với tổng các phần tử trong mảng **wifi\_ssid** và đồng thời bằng tổng các phần tử ở mảng **wifi\_pw**. Còn các tag **LED\_STT\_DELAY\_TIME** là thời gian mà ESP sẽ kiểm tra tình trạng wifi.

```
//ota
#define OTA_HOSTNAME      "ESP-%06X"      // can change 'ESP', can't change '-%06X'
```

Hình 5.9 Các tag liên quan đến ota

(hình 5.9) Ở phần này các bạn cần lưu ý **OTA\_HOSTNAME** các bạn có thể sửa chữ ESP nhưng đừng sửa -%06X nhé vì đây là kiểu định dạng string (các bạn có thể tìm hiểu thêm thì có thể lên Google tìm printf() trong C)

```
//interrupt
int list_io_interrupt[] = {21};           // list_io_interrupt.length() == NUM_PIN_INTERRUPT
#define NUM_PIN_INTERRUPT    1
#define EN_INTERRUPT        true        // if use interrupt, will be change false to true
#define MODE_INTERRUPT       RISING     // 5 mode: RISING, FALING, LOW, HIGH, CHANGE
```

Hình 5.10 Các tag liên quan đến interrupt

(hình 5.10) Ở phần này các bạn cần lưu ý **NUM\_PIN\_INTERRUPT** phải bằng với tổng các phần tử trong mảng **list\_io\_interrupt**. **EN\_INTERRUPT** tag này khi nào các bạn điền là true thì mới có thể sử dụng interrupt được, còn nếu là false thì dù ở trên các bạn có thay đổi như thế nào thì hàm interrupt sẽ không hoạt động đâu.

## Chương 1: Cơ sở lý thuyết

**MODE\_INTERRUPT** là sự kiện mà ESP sẽ lấy làm điều kiện để vào hàm callback làm nhiệm vụ.

```
//MQTT
#define DELAY_TIME_MQTT      1*60          //1m
#define MQTT_SERVER           "192.168.1.3"  //your IP
#define MQTT_PORT              1883
#define MQTT_ID                "ESP"         // can change
#define TOPPIC_SUB             "S-ESP"       // can change 'ESP', can't change 'S-'
#define TOPPIC_PUB              "P-ESP"       // can change 'ESP', can't change 'P-'
```

Hình 5.11 Các tag liên quan đến mqtt

(hình 5.11) các tag khác các bạn có thể để nguyên như thế nhưng hãy lưu ý đến tag **MQTT\_SERVER** - Đây là tag địa chỉ mạng server (IP máy tính của bạn), hãy đảm bảo rằng bạn đã mở node-red trên máy tính và đã điền đúng địa chỉ IP máy tính ở phần này nhé vì ESP sẽ không chạy nếu chưa được kết nối MQTT đâu. **DELAY\_TIME\_MQTT** thì các bạn có thể thay đổi để cập nhật trạng thái thường xuyên lên Server nhé.

```
//Application
#define RED_LED_PIN           18
#define GREEN_LED_PIN          5

#define DHT_PIN                 19
#define DHTTYPE                 DHT11
```

Hình 5.12 Các tag liên quan đến project

(hình 5.11) Khu vực này để khai báo các chân liên quan đến project như hai chân led và chân DHT. Còn chân button thì mình đã khai báo ở trên phần interrupt rồi nhé.

\*\* full code **configs.h**

```
#ifndef __CONFIGS_H__
#define __CONFIGS_H__

//Wifi
const char *wifi_ssid[]{"THENAM", "NTGD"};           // your wifi name
const char *wifi_pw[] = {"0964941600", "112233445566"}; //your wifi password
#define NUM_WIFI               2                      // NUM_WIFI ==
wifi_pw.length() == wifi_ssid.length()
#define LED_STT_DELAY_TIME     60                     // 60s

//ota
```

**Note: để lấy full code nhấn vào ô chứa code rồi Ctrl + C. Vào file configs.h trong Arduino rồi Ctrl + V.**

Hãy đẩy code lên GitHub để lưu nhé

**git add .**

**git commit -m "add configs.h"**

**git push**

**b) app.h:**

Ở file này mình sẽ chỉ code những thứ liên quan đến project SmartHome thôi nhé. Như vậy ở đây mình sẽ code cho nút nhấn và DHT, ngoài ra còn phải thu thập data và gửi data lên Server.

```
//library DHT
#include <Adafruit_Sensor.h>
#include <DHT.h>
```

Hình 5.13 khai báo thư viện liên quan đến DHT

(Hình 5.13) Ở phần này nếu có báo lỗi DHT chưa được khai báo thì có nghĩa là các bạn chưa cài thư viện DHT nhé (link thư viện: [DHT sensor library - Arduino Reference](#)) nếu muốn hướng dẫn thêm thư viện thì các bạn hãy quay lại 1.2.3 Cài đặt Arduino mình đã hướng dẫn chi tiết tại đó.

```
===== Var =====
float hum = 0;
float temp = 0;
float buffer_hum = 0;
float buffer_temp = 0;
bool buffer_red_led_status = false;
bool buffer_green_led_status = false;

bool is_press = false;
int count_press = 0;
unsigned long long old_time_press = 0;
unsigned long long delay_time_press = 200;

unsigned long long old_time_report = 0;
```

Hình 5.14 khai báo các biến dùng trong app.h

(Hình 5.14) Ở đây mình khai báo

- **hum, temp** là giá trị để gán nhiệt độ khi DHT đo được.

## Chương 1: Cơ sở lý thuyết

- Các biến có **buffer** ở đầu là các biến dùng để lưu tạm giá trị để khi dùng hàm reportReading() thì ta sẽ lấy giá trị ở biến buffer này mà không cần chạy một hàm khác nữa. Việc khai báo này là để tránh bị delay khi lấy giá trị.
- **Is\_press** là biến thông báo sự kiện nút nhấn đã được nhấn.
- **Count\_press** là biến để đếm số lần nhấn (vì mình chỉ có 1 nút nhấn nên phải sử dụng cách đếm số lần nhấn để xử lý sự kiện bằng 1 nút)
- **Old\_time\_press** là biến dùng để lưu lại thời gian lúc vừa nhấn nút.
- **delay\_time\_press** là thời gian trễ để xử lý nút nhấn.
- **old\_time\_report** là biến dùng để lưu lại thời gian lúc vừa thực hiện xong việc gửi dữ liệu lên server

```
DHT dht(DHT_PIN, DHTTYPE);
```

Hình 5.15 khai báo sử dụng DHT loại nào

(Hình 5.15) là nơi ta sẽ khai báo chân đọc tín hiệu và loại DHT sẽ sử dụng trong project (đã khai báo ở file configs.h)

## Chương 1: Cơ sở lý thuyết

```
//===== report reading ======
void reportReading()
{
    char chip_name[24];
    sprintf(chip_name, OTA_HOSTNAME, getChipId());

    String json_string = "";

    json_string += "{";

    json_string += "\"chipID\":=\"";
    json_string += String(chip_name);
    json_string += "\",";

    json_string += "\"red_led\":\"";
    json_string += String(buffer_red_led_status);
    json_string += ",";

    json_string += "\"green_led\":\"";
    json_string += String(buffer_green_led_status);
    json_string += ",";

    json_string += "\"hum\":\"";
    json_string += String(buffer_hum);
    json_string += ",";

    json_string += "\"temp\":\"";
    json_string += String(buffer_temp);
    json_string += "\",";

    json_string += "\"now\":\"";
    json_string += String(millis());
    json_string += "\"";

    json_string += "}";

    Serial.println( String(json_string) );

    if(is_press == true)
    {
        mqttSendServerNow(json_string);
        json_string = "";
        return;
    }

    mqttSendServer(json_string);
    json_string = "";
}
```

Hình 5.16 hàm reportReading()

(Hình 5.16) Hàm này sẽ là hàm lấy các dữ liệu được lưu ở các biến buffer rồi ghép thành một chuỗi JSON (nếu bạn chưa biết JSON là gì thì nó nôm na là một kiểu quản lý object và cũng dễ dàng lấy dữ liệu trên Javascript)

## Chương 1: Cơ sở lý thuyết

Ở cuối mình có gọi thêm 2 hàm trong thư viện my\_mqtt.h là mqttSendServerNow()- dùng để gửi luôn dữ liệu lên cho Server nếu có sự kiện nút nhấn được gọi và hàm mqttSendServer()- dùng để gửi dữ liệu lên server theo chu kì một phút một lần.

```
void setupDHT()
{
    dht.begin();
    Serial.println("setup DHT done !!!");
}
```

Hình 5.18 hàm setupDHT()

(Hình 5.18) Hàm này sẽ là hàm khởi tạo thiết bị DHT và bắt đầu cho ESP đọc dữ liệu từ DHT gửi về.

```
void readDHT()
{
    hum = dht.readHumidity();
    temp = dht.readTemperature();
    if (isnan(hum) || isnan(temp)) {
        Serial.println(F("Failed to read from DHT sensor!"));
        buffer_hum = -1;
        buffer_temp = -1;
        return;
    }
    buffer_hum = hum ;
    buffer_temp = temp ;
}
```

Hình 5.19 hàm readDHT()

(Hình 5.19) Hàm này sẽ là hàm đọc dữ liệu từ DHT gửi về. và sau đó gán giá trị vào biến buffer.

```
void readLedState()
{
    buffer_red_led_status = digitalRead(RED_LED_PIN);
    buffer_green_led_status = digitalRead(GREEN_LED_PIN);
}
```

Hình 5.20 hàm readLedState()

(Hình 5.20) Hàm này sẽ là hàm đọc trạng thái đèn và sau đó gán giá trị vào biến buffer.

## Chương 1: Cơ sở lý thuyết

```
void setupInOutput()
{
    pinMode(RED_LED_PIN, OUTPUT);
    pinMode(GREEN_LED_PIN, OUTPUT);
    pinMode(DHT_PIN, INPUT);
    digitalWrite(RED_LED_PIN, LOW);
    digitalWrite(GREEN_LED_PIN, LOW);
}
```

Hình 5.21 hàm setupInOutput()

(Hình 5.21) Hàm này sẽ là hàm khai báo chức năng cho chân IO.

```
void __attribute__((__IRAM_ATTR)) callbackInterrupt()
{
    is_press = true;
    count_press++;
    old_time_press = millis();
}
```

Hình 5.22 hàm callbackInterrupt()

(Hình 5.22) Hàm này sẽ là hàm phát cờ để thông báo cho hàm loop rằng nút nhấn đã được nhấn (về cơ bản có thể thực hiện luôn lệnh sáng led ở trong hàm này luôn nếu tác vụ đơn giản. Còn mình dùng một nút nên mình cần nhiều tác vụ khác cần xử lý trước khi sáng led).

## Chương 1: Cơ sở lý thuyết

```
void handleButton()
{
    if(!is_press)
        return;

    if(millis() - old_time_press > delay_time_press)
    {
        if(count_press == 1)
        {
            buffer_red_led_status = digitalRead(RED_LED_PIN);
            digitalWrite(RED_LED_PIN, !buffer_red_led_status);
            buffer_red_led_status = digitalRead(RED_LED_PIN);
            Serial.println("red led: " + String(buffer_red_led_status ? "on" : "off"));
            reportReading();
            is_press = false;
            count_press = 0;
            return;
        }
        if(count_press == 2)
        {
            buffer_green_led_status = digitalRead(GREEN_LED_PIN);
            digitalWrite(GREEN_LED_PIN, !buffer_green_led_status);
            buffer_green_led_status = digitalRead(GREEN_LED_PIN);
            Serial.println("green led: " + String(buffer_green_led_status ? "on" : "off"));
            reportReading();
            is_press = false;
            count_press = 0;
            return;
        }
        if(count_press > 2)
        {
            Serial.println("count_press > 2");
            is_press = false;
            count_press = 0;
            return;
        }
    }
}
```

Hình 5.23 hàm handleButton()

(Hình 5.23) Vì mình có một nút nên mình sẽ phải xử lý để mà điều khiển 2 led. Ở hàm này nếu nhấn nút 1 lần thì sẽ điều khiển led đỏ. Còn nhấn 2 lần thì sẽ điều khiển led xanh. Còn nếu nhấn nhiều hơn 2 lần thì không có thao tác nào được hiện mà chỉ có thông báo thôi.

Các bạn hãy thử thêm chức năng có 1 nút này nhé như là nhấn 3 lần thì 2 led cùng tắt. hãy thử đi nhé.

## Chương 1: Cơ sở lý thuyết

```
void setupApp()
{
    setupInOutput();
    delay(50);

    //DHT
    setupDHT();
    delay(50);
}

void loopApp()
{
    //DHT
    readDHT();

    //read led state
    readLedState();

    //handle pressing button
    handleButton();

    //10s show on Serial
    //1m send data to Server
    if(millis() - old_time_report > 10000)
    {
        old_time_report = millis();
        reportReadding();
    }
}
```

Hình 5.24 hàm setupApp() và loopApp()

(Hình 5.24) ở 2 hàm này thì sẽ gọi tất cả các hàm mình đã viết ở trên vào thôi. Hàm setupApp thì dễ rồi còn hàm loopApp thì hàm này ta sẽ luôn luôn đọc các giá trị sử lý trạng thái nút nhấn. và 10s in ra serial để ta quan sát dễ debug hơn.

## Chương 1: Cơ sở lý thuyết

```
//===== application in .ino =====
void offAllLed()
{
    digitalWrite(RED_LED_PIN,HIGH);
    digitalWrite(GREEN_LED_PIN,HIGH);
    buffer_red_led_status = true;
    buffer_green_led_status = true;
}
void onAllLed()
{
    digitalWrite(RED_LED_PIN,LOW);
    digitalWrite(GREEN_LED_PIN,LOW);
    buffer_red_led_status = false;
    buffer_green_led_status = false;
}

void onled(int pinIO)
{
    digitalWrite(pinIO,HIGH);
    pinIO == RED_LED_PIN ? buffer_red_led_status = true : buffer_green_led_status = true;
}

void offLed(int pinIO)
{
    digitalWrite(pinIO,LOW);
    pinIO == RED_LED_PIN ? buffer_red_led_status = false : buffer_green_led_status = false;
}

void toggleLed(int pinIO)
{
    int value_tmp = digitalRead(pinIO);
    digitalWrite(pinIO,!value_tmp);
    if(pinIO == RED_LED_PIN)
    {
        buffer_red_led_status = !value_tmp;
        return;
    }
    buffer_green_led_status = !value_tmp;
}
```

Hình 5.25 các hàm liên quan đến file .ino

(Hình 5.25) các hàm này sẽ thực hiện các chức năng mà ở file .imo sử dụng. Vì ở file .ino phải thỏa điều kiện rằng chỉ nên gọi hàm. Hạn chế mức tối đa gọi biến từ các file khác.

## Chương 1: Cơ sở lý thuyết

\*\* full code **app.h**

```
#ifndef __APP_H__
#define __APP_H__

//library DHT
#include <Adafruit_Sensor.h>
#include <DHT.h>

//===== Var =====
float hum = 0;
float temp = 0;
float buffer_hum = 0;
```

**Note: để lấy full code nhấn vào ô chứa code rồi Ctrl + C. Vào file app.h trong Arduino rồi Ctrl + V.**

Hãy dán code lên GitHub để lưu nhé

**git add .**

**git commit -m “add app.h”**

**git push**

c) **ESP32\_SmartHome.ino:**

Đầu tiên thì ta cũng sẽ phải khai báo thư viện thôi

```
#include "configs.h"

##### include your libraries inthere #####
#include "my_multi_wifi.h"
#include "my_ota.h"
#include "my_interrupt.h"
#include "my_mqtt.h"
#####
#include "app.h"
```

Hình 5.26 thêm thư viện vào

(Hình 5.26) hãy thêm tất cả các thư viện mà ta đã cá nhân hóa vào project chúng ta đã viết ở trên vào thôi. Hãy để đúng thứ tự thư viện nhé không thì sẽ bị lỗi khao báo đấy nhé. Trong phần “`include your libraries inthere`” thì bạn có thể thêm nhiều thư viện khác hay đặt lôn xộn thế nào cũng dc. Nhưng file configs.h phải là đầu tiên để ESP lấy tag ta khai báo trong đây trước khong thì ESP sẽ lấy tag mặc định trong thư viện. Còn file

## Chương 1: Cơ sở lý thuyết

app.h thì là file có thể dùng các hàm trong thư viện nên phải đặt sau các thư viện mới dùng được nhé.

```
// handle Command from Server
void executeMqttCommand(String command)
{
    Serial.println("command from Server: " + command);
    if(command.length() < 0)
        return;

    int cmd_index = -2;
    cmd_index = command.indexOf(";");
    //handle without ;
    //eg: on
    if(cmd_index < 0)
    {
        if(command == "on")
        {
            onAllLed();
            return;
        }
        if(command == "off")
        {
            offAllLed();
            return;
        }
    }

    //handle with ;
    int io = command.substring(0,cmd_index).toInt();
    String sub_cmd = command.substring(cmd_index + 1);

    if(sub_cmd == "on")
    {
        onled(io);
        return;
    }

    if(sub_cmd == "off")
    {
        offLed(io);
        return;
    }

    if(sub_cmd == "toggle")
    {
        toggleLed(io);
        return;
    }
}
```

Hình 5.27 hàm xử lý lệnh từ server gửi xuống

## Chương 1: Cơ sở lý thuyết

(Hình 5.27) Ở hàm này mình sẽ sử lý với 2 loại lệnh. Lên có ; và lệnh không có ;. Nếu không có ; thì mình sẽ tự hiểu rằng sẽ áp dụng cho tất cả còn nếu có ; thì sẽ điều khiển cho một thiết bị cụ thể nào đó. Hàm này chỉ đơn giản là kiểm tra tách chuỗi và sử lý thôi.

Ví dụ như **on** đơn thuần thế này mình sẽ hiểu rằng đang ra lệnh mở tất cả các đèn và ngược lại **off** tắt tất cả các đèn. Còn nếu **18;on** như này thì mình sẽ hiểu rằng đang lệnh cho chân 18 mở đèn (HIGH)

```
void setup()
{
    Serial.begin(112500);
    pinMode(2, OUTPUT); //this's LED of board ESP. use it for notification. So don't change

    setupApp();
    delay(50);

    setupWiFiMulti();
    delay(50);

    setupOTA();
    delay(50);

    setupInterrupt();
    delay(50);

    setupMQTT();
    delay(50);
}
```

Hình 5.28 hàm setup()

(Hình 5.28) hàm setup này mình sẽ gọi các hàm mình đã viết theo sơ đồ thiết logic code trước đó thôi (bạn có thể xem lại mục **5.1.2 b) phân tích thiết kế logic code**)

```
void setup()
{
    Serial.begin(112500);
    pinMode(2, OUTPUT); //this's LED of board ESP. use it for notification. So don't change

    setupApp();
    delay(50);

    setupWiFiMulti();
    delay(50);

    setupOTA();
    delay(50);

    setupInterrupt();
    delay(50);

    setupMQTT();
    delay(50);
}
```

## Chương 1: Cơ sở lý thuyết

Hình 5.29 hàm loop()

(Hình 5.29) hàm loop này mình sẽ gọi các hàm mình đã viết theo sơ đồ thiết logic code trước đó thôi (bạn có thể xem lại mục **5.1.2 b) phân tích thiết kế logic code**)

\*\* full code **ESP32\_SmartHome.ino**

```
#include "configs.h"

//##### include your libraries in there #####
#include "my_multi_wifi.h"
#include "my_ota.h"
#include "my_interrupt.h"
#include "my_mqtt.h"
//#####
#include "app.h"
```

**Note: để lấy full code nhấn vào ô chứa code rồi Ctrl + C. Vào file ESP32\_SmartHome.ino trong Arduino rồi Ctrl + V.**

Hãy đẩy code lên gitHub để lưu nhé

**git add .**

**git commit -m “add app.h”**

**git push**

### 5.1.4 Hoàn thiện Node-red:

Giờ thì hãy khởi động server và mở node-red trc đâ.

**Mở cmd rồi gõ node-red’**

## Chương 1: Cơ sở lý thuyết

```
15 Oct 08:21:26 - [info]
Welcome to Node-RED
=====

15 Oct 08:21:26 - [info] Node-RED version: v3.0.2
15 Oct 08:21:26 - [info] Node.js version: v16.17.1
15 Oct 08:21:26 - [info] Windows_NT 10.0.19044 x64 LE
15 Oct 08:21:27 - [info] Loading palette nodes
15 Oct 08:21:28 - [info] Settings file : C:\Users\AdMins\.node-red\settings.js
15 Oct 08:21:28 - [info] Context store : 'default' [module=memory]
15 Oct 08:21:28 - [info] User directory : \Users\AdMins\.node-red
15 Oct 08:21:28 - [warn] Projects disabled : editorTheme.projects.enabled=false
15 Oct 08:21:28 - [info] Flows file : \Users\AdMins\.node-red\flows.json
15 Oct 08:21:28 - [warn]

-----
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

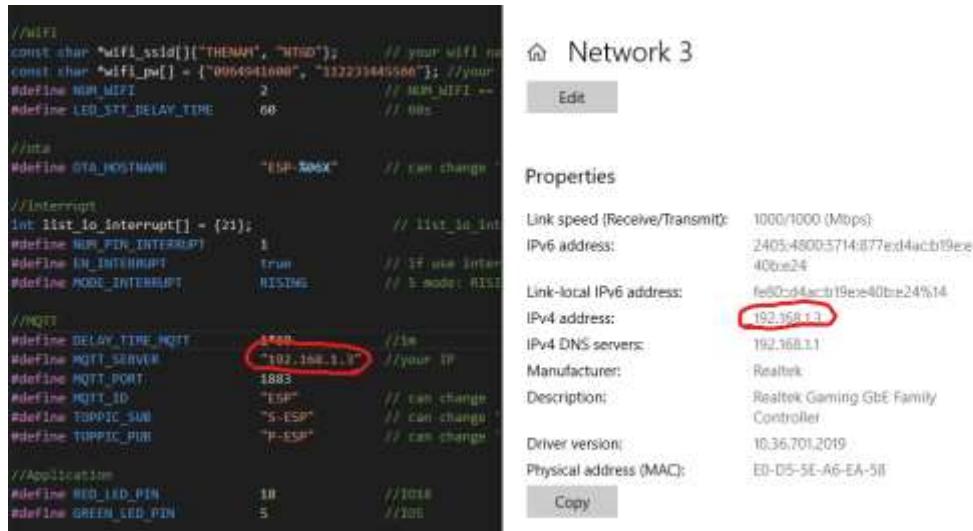
You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
-----

15 Oct 08:21:28 - [info] Server now running at http://127.0.0.1:1880/
15 Oct 08:21:28 - [info] Starting flows
15 Oct 08:21:28 - [info] Started flows
15 Oct 08:21:28 - [info] [aedes broker:458494f5702e8a93] Binding aedes mqtt server on port: 1883
15 Oct 08:21:28 - [info] [mqtt-broker:d29e84c9f6f3e7c2] Connected to broker: mqtt://192.168.1.3:1883
```

Hình 5.30 khởi tạo node-red thành công

Giờ thì mở node-red trên trình duyệt nào. Vào trình duyệt rồi gõ **localhost:1880**

**Nạp code cho ESP rồi chờ ESP kết nối vào server** (nhớ hãy kiểm tra ip mạng máy tính rồi sửa tag **MQTT\_SERVER** trong file config.h cho giống nhau rồi hẵng nạp code nhé hình Hình 5.31)



Hình 5.31 ip mạng server

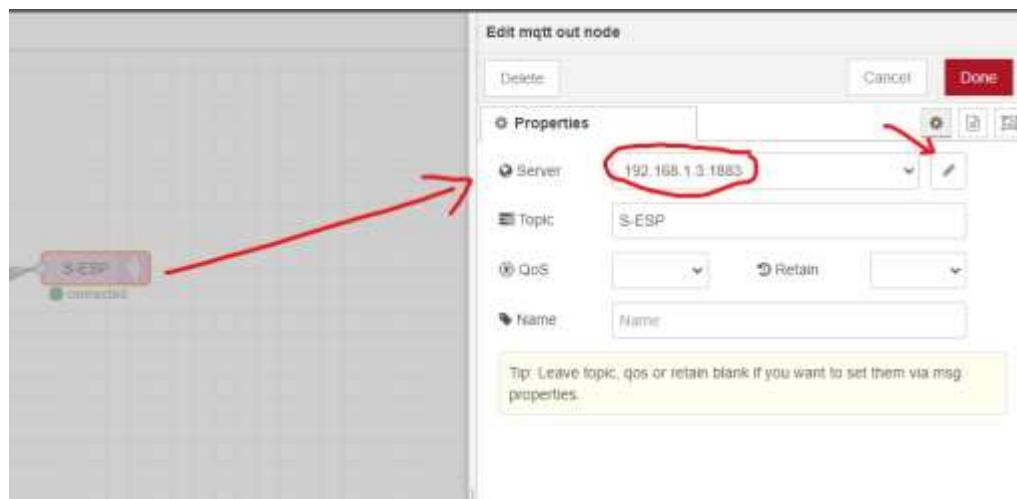
## Chương 1: Cơ sở lý thuyết

Nếu như bạn không có các node như hình 5.33 thế này thì bạn hãy quay lại phần **3.3 Hướng dẫn cơ bản node-red** mình đã hướng dẫn các bước thêm các node này rồi nhé.



Hình 5.32 giao diện Node-red

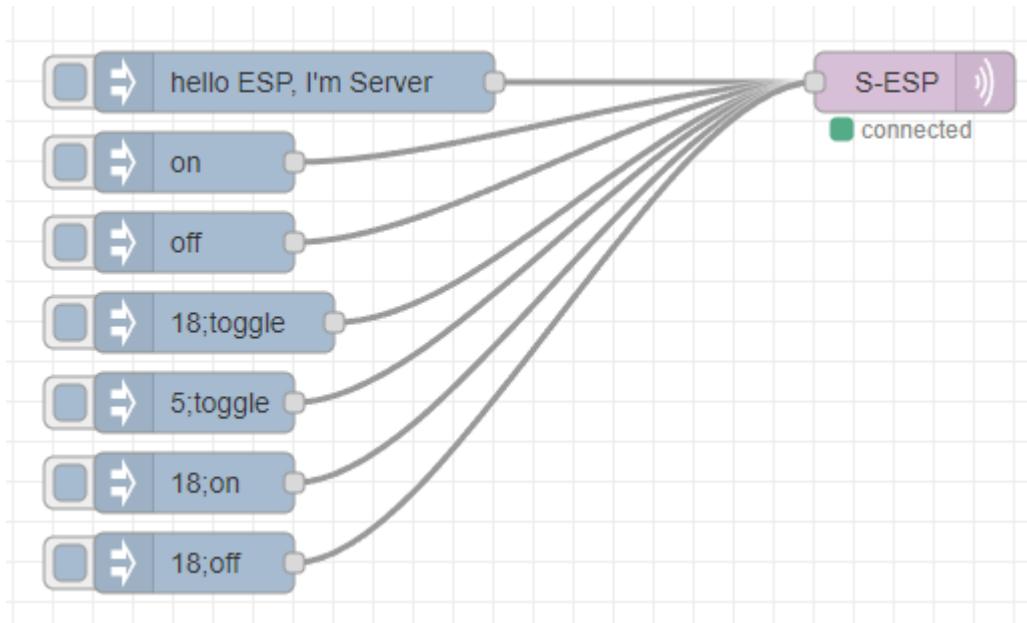
Thêm một lưu ý nữa hãy kiểm tra xem node(S-ESP) và node(P-ESP) đã kết nối đúng với tag **MQTT\_SERVER** chưa? Nếu chưa thì sửa lại sau đó ấn **deploy** bên góc phải màn hình và **ấn nút EN** trên ESP để reset chip luôn nhé.



Hình 5.33 sửa ip

Nếu bạn đã kết nối được thì giờ hãy thêm các node inject vào để ta có thể test thử lệnh mqtt nhé.(hình5.34)

## Chương 1: Cơ sở lý thuyết



Hình 5.34 thêm node inject

(Hình 5.35) chỉnh các thuộc tính các inject như sau:

- (1) chọn node inject
- (2) sửa timestamp thành kiểu string
- (3) ghi lệnh ta muốn gửi xuống dưới ESP

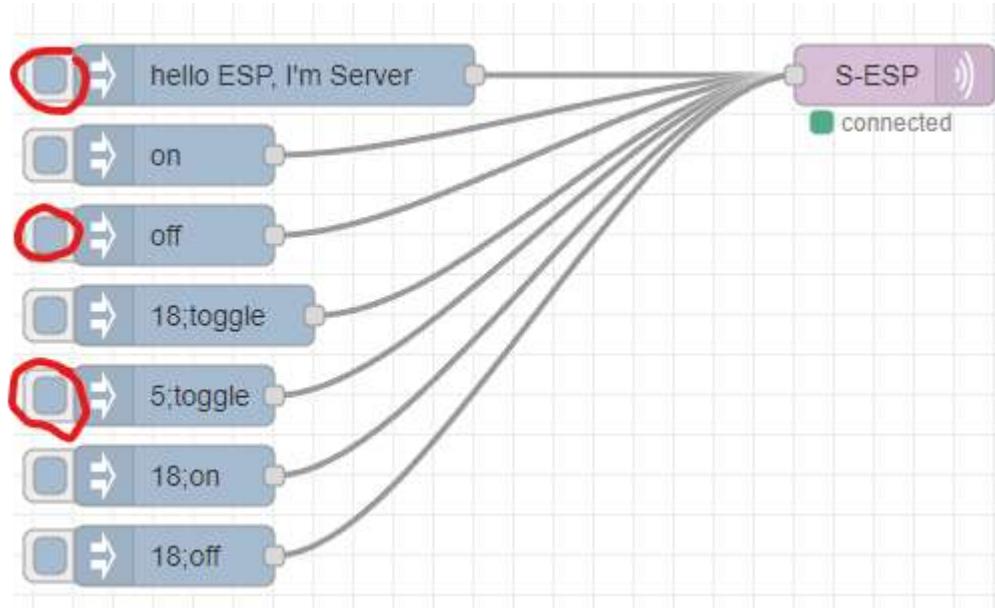
Vì ở dưới ESP mình đã quy định rằng **on** là lệnh áp dụng cho tất cả và **18;on** là lệnh chỉ áp dụng cho một thiết bị thì ở trên node-red cũng sẽ dùng như thế nhé. Các bạn ghi khác thì ESP sẽ không hiểu đâu.



Hình 5.35 Sửa các thuộc tính node inject

## Chương 1: Cơ sở lý thuyết

(Hình 5.36) Giờ thì hãy ấn vào ô vuông bên cạnh node inject rồi quan sát ở dưới ESP có thực hiện không nhé.



Hình 5.36 gửi lệnh xuống ESP

Giờ nhìn cái giao diện thế này nhìn có vẻ không được trực quan lắm nên là các bạn hãy cùng mình tạo một UI để nhìn cho nó dễ quan sát hơn nhé.

### a) Thêm thư viện UI Dashboard

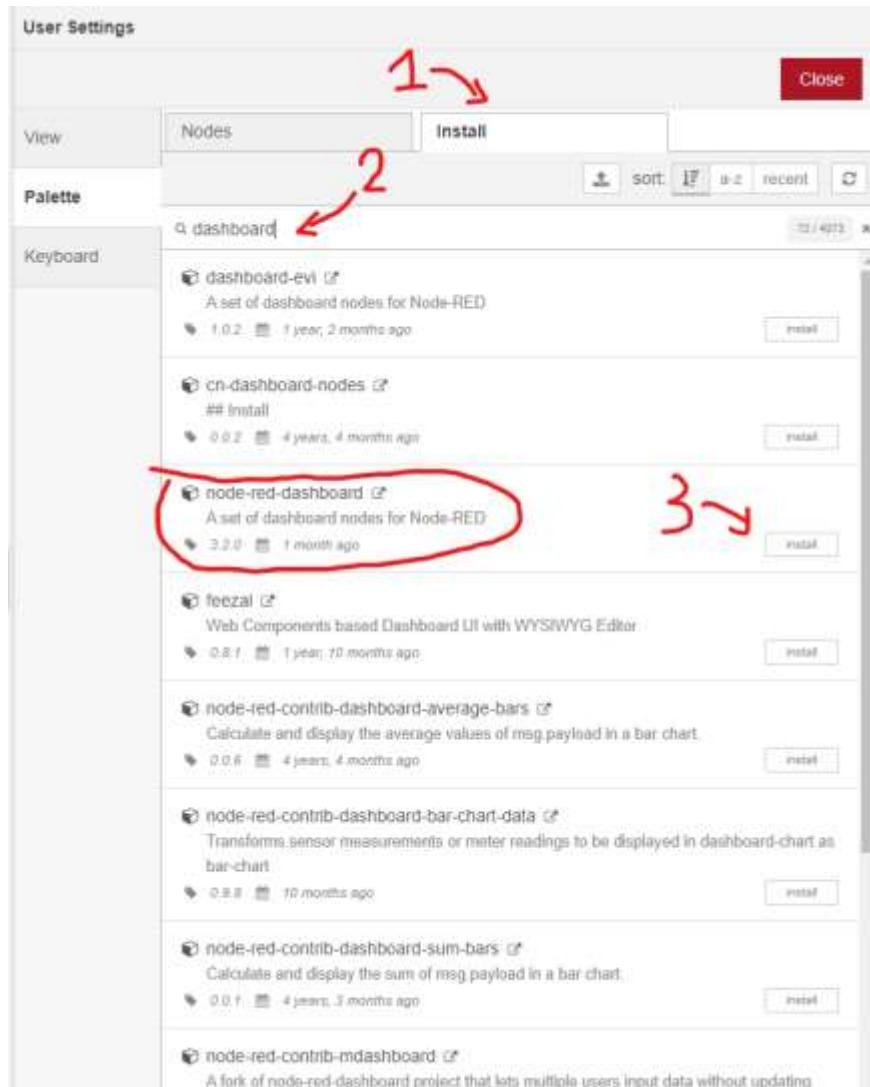
Giờ thì hãy thêm packet node-red-dashboard. Như mình đã hướng dẫn chi tiết thêm packet mục **3.3 Hướng dẫn sử dụng cơ bản Node-red** rồi nên các bạn quên có thể quay lại đó xem lại nhé.

Thư viện này giống như ta tạo nên một trang web để ta dễ nhìn hơn và thao tác đơn giản hơn.

(Hình 5.37) cài packet dashboard

- (1) nhấn vào install
- (2) tìm từ khóa “dashboard”
- (3) install packet “node-red-dashboard”

## Chương 1: Cơ sở lý thuyết



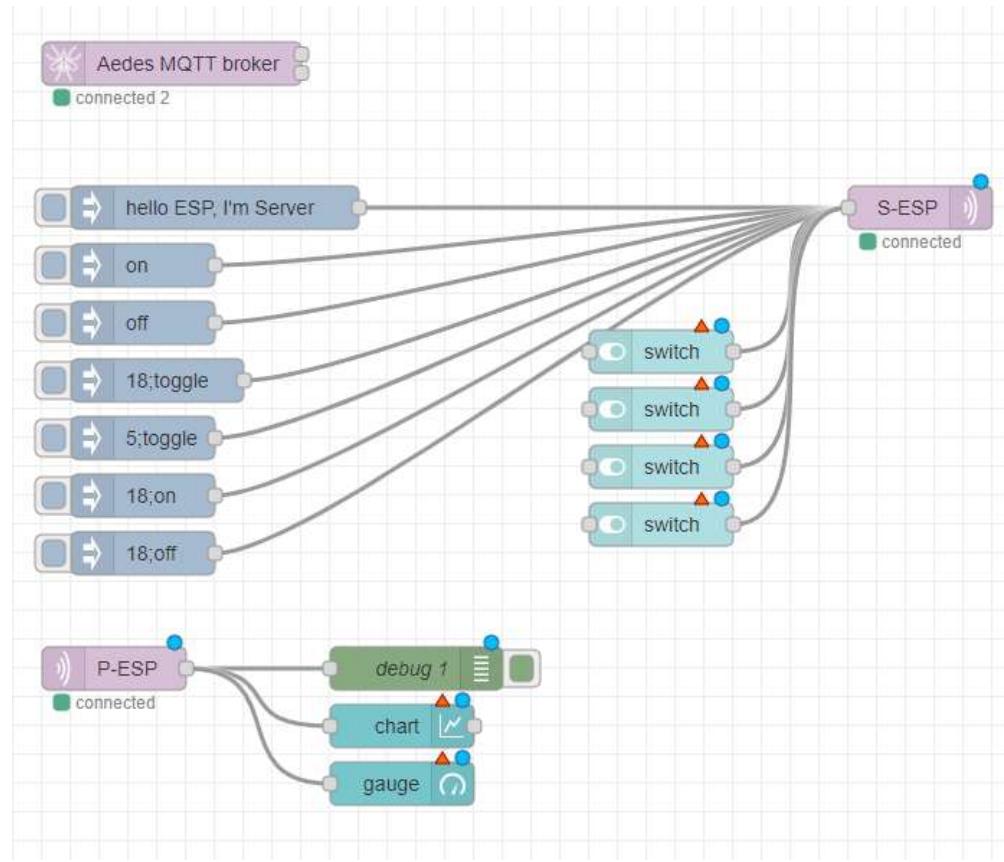
Hình 5.37 install packet dashboard

### b) Thêm UI vào dashboard

Ở đây mình chỉ hướng dẫn cơ bản nên mình chỉ dùng mấy node dashboard cơ bản thôi. Còn bạn có thể đọc thêm và tìm hiểu thêm ở hình quen sách hay lên document của Node-red để đọc nhé.

Kéo node Switch, node Chart, node Gauge ra và nối chúng như hình Hình(5.38)

## Chương 1: Cơ sở lý thuyết

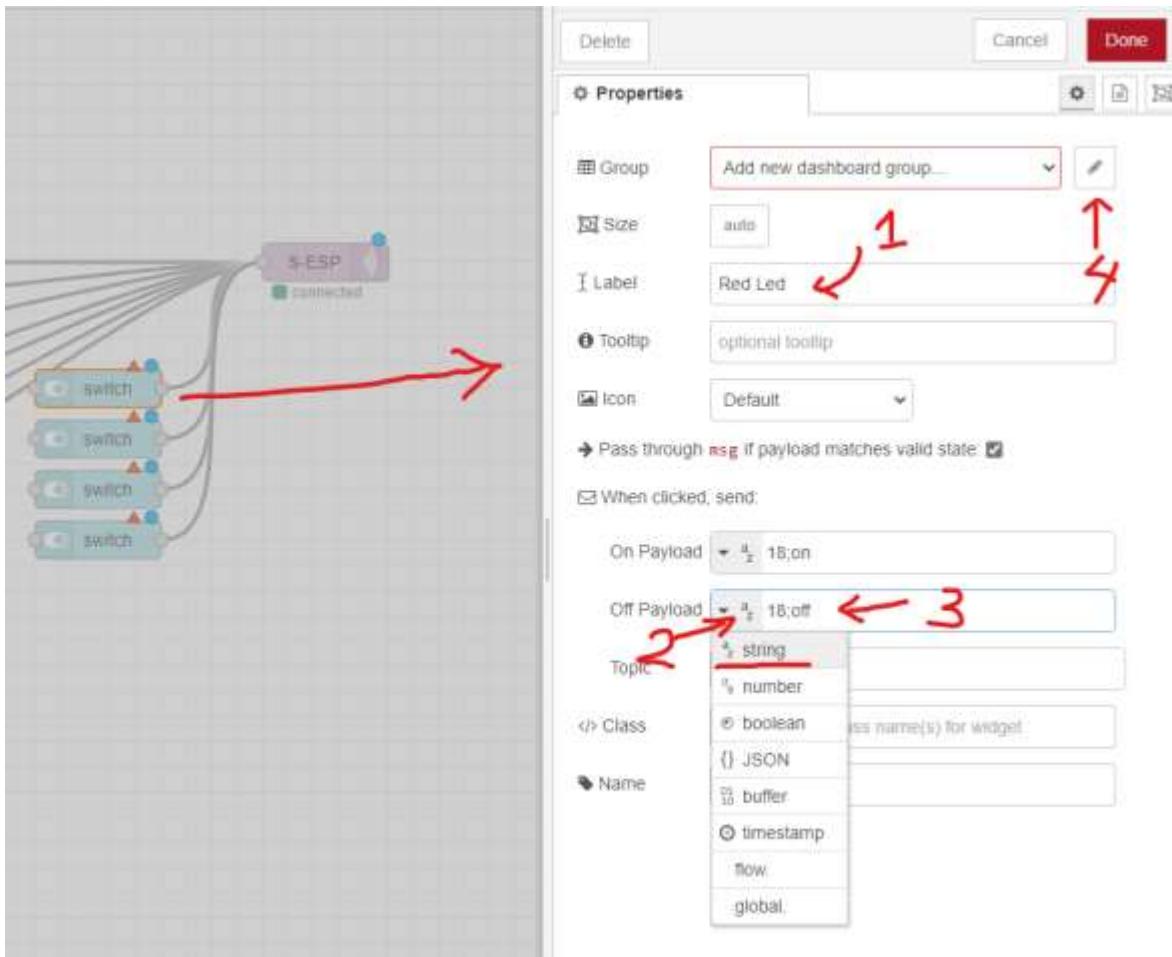


Hình 5.38 kéo các node dashboard ra ngoài

Sau đó setting thuộc tính cho từng node như sau:

- Node switch (hình5.39):
  - (1) Sửa tên nút switch ở mục Lable
  - (2) Sửa kiểu dữ liệu mà node gửi đi thành String
  - (3) ghi các lệnh tương ứng khi gạt cần sang on và off
  - (4) thêm node vào group

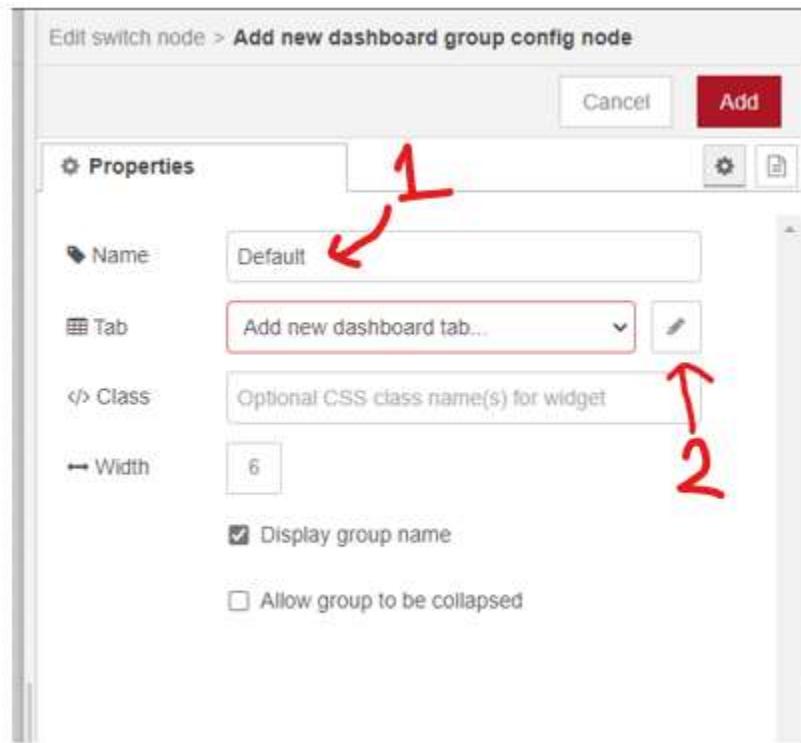
## Chương 1: Cơ sở lý thuyết



Hình 5.39 chỉnh sửa thuộc tính node switch

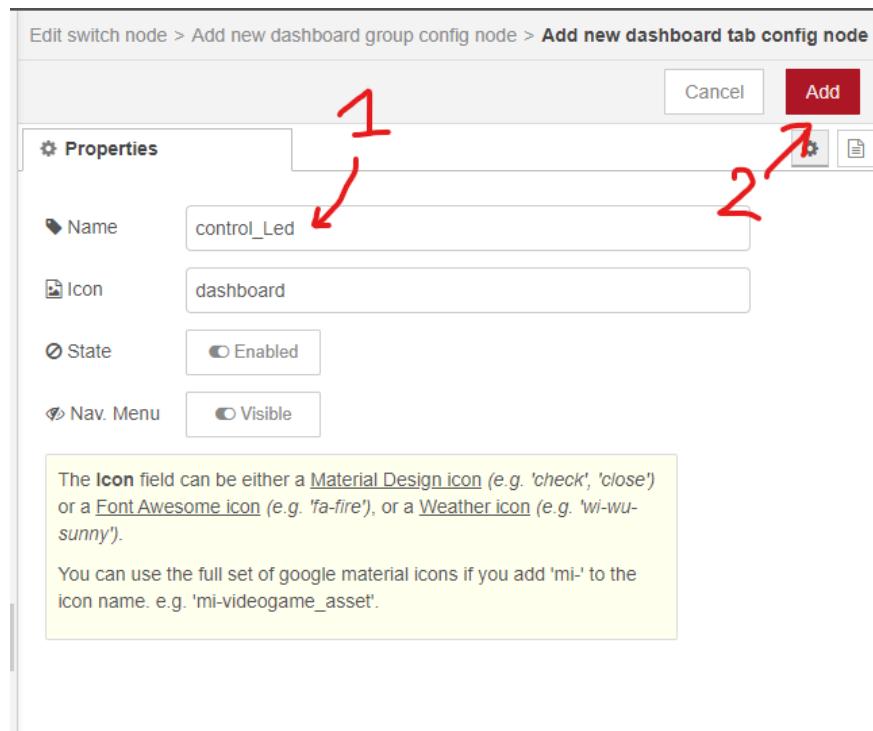
(hình 5.40) thêm sửa tên Gorup nhân vào cây bút để sửa tab

## Chương 1: Cơ sở lý thuyết



Hình 5.40 sửa group

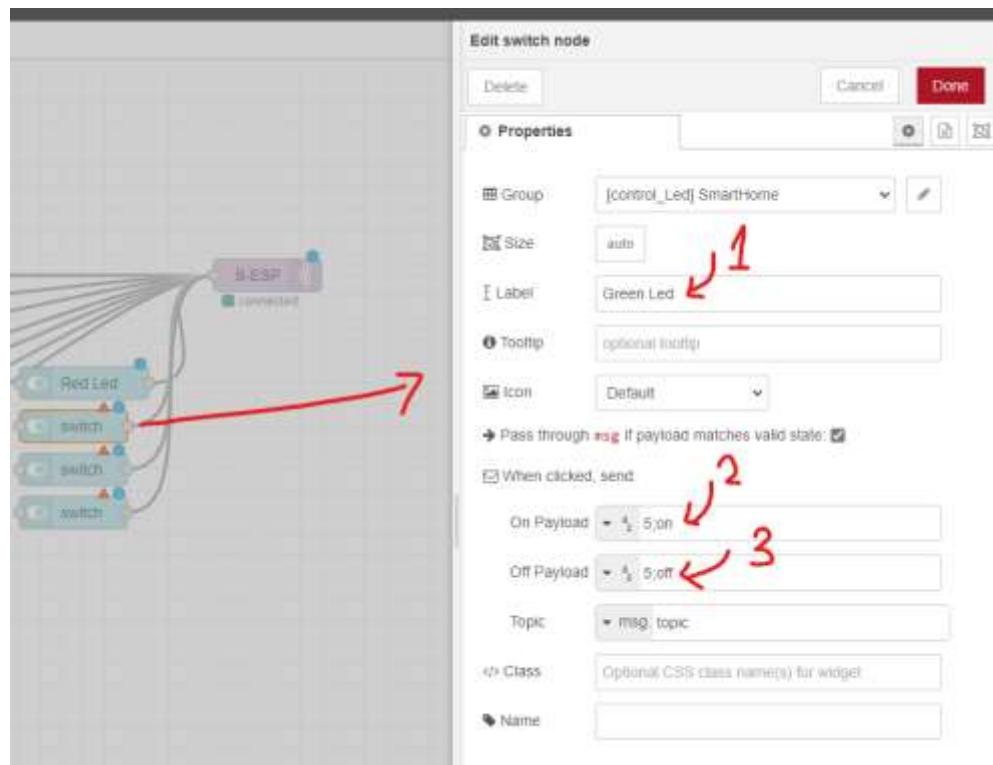
(hình 5.41) đổi tên tab vào ấn nút add để thêm tab này vào group



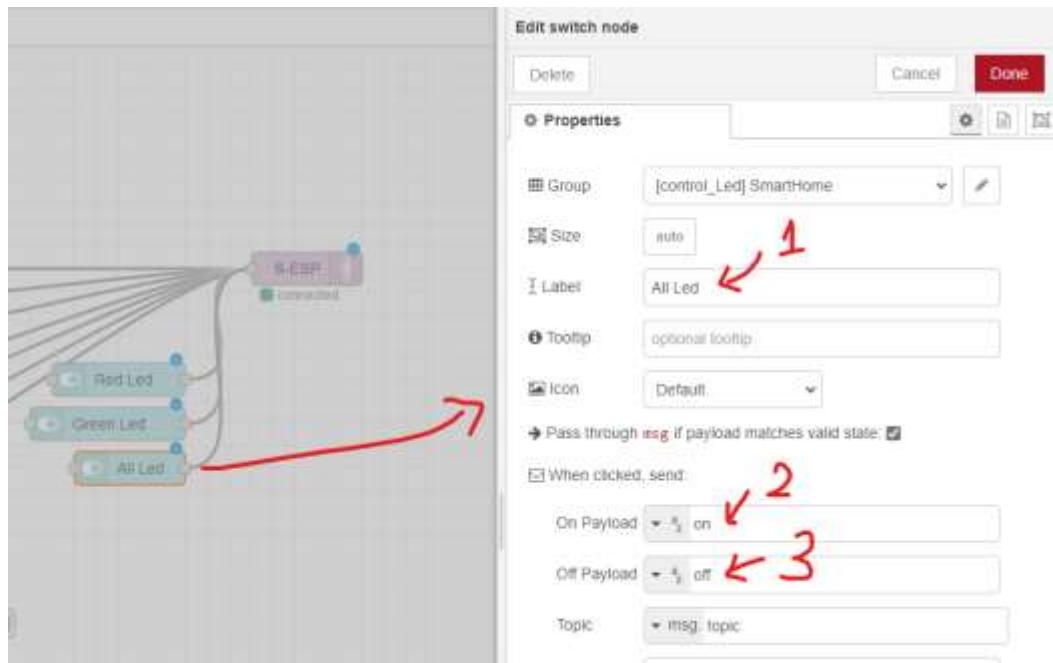
Hình 5.41 sửa tab

## Chương 1: Cơ sở lý thuyết

Tương tự như thế các bạn chỉnh sửa thuộc tính còn lại cho các node Switch, node Gouge khác như hình dưới là được.

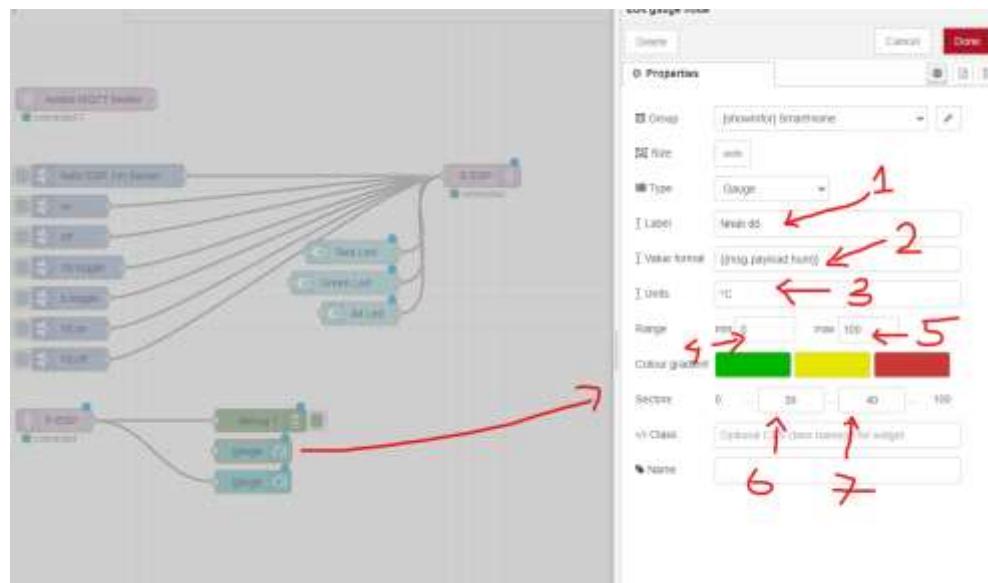


Hình 5.42 thuộc tính green led

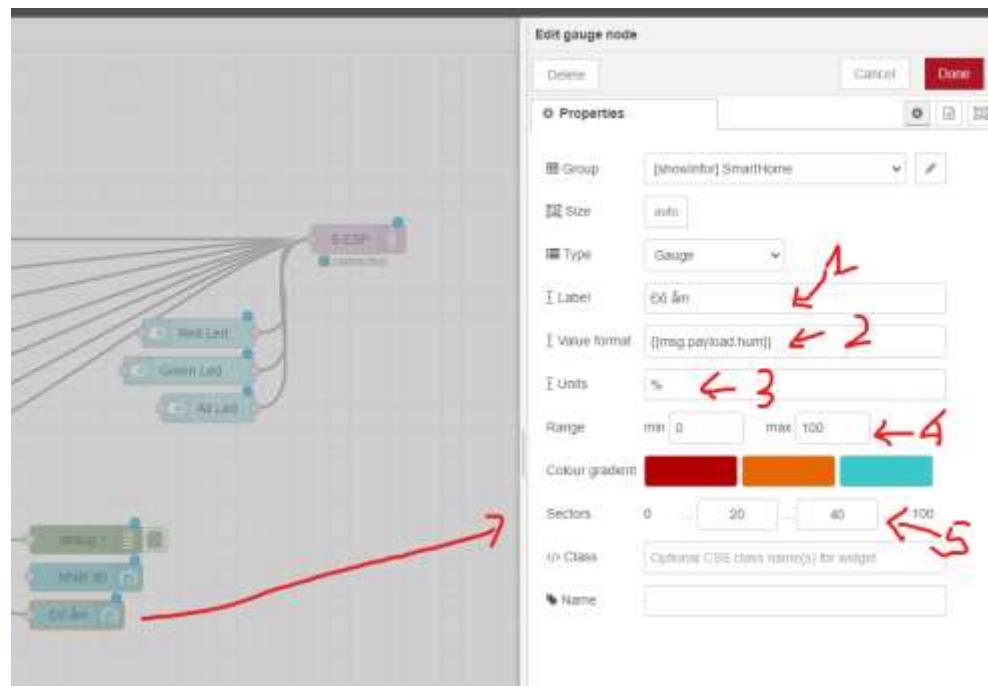


Hình 5.43 thuộc tính all led

## Chương 1: Cơ sở lý thuyết



Hình 5.43 thuộc tính nhiệt độ

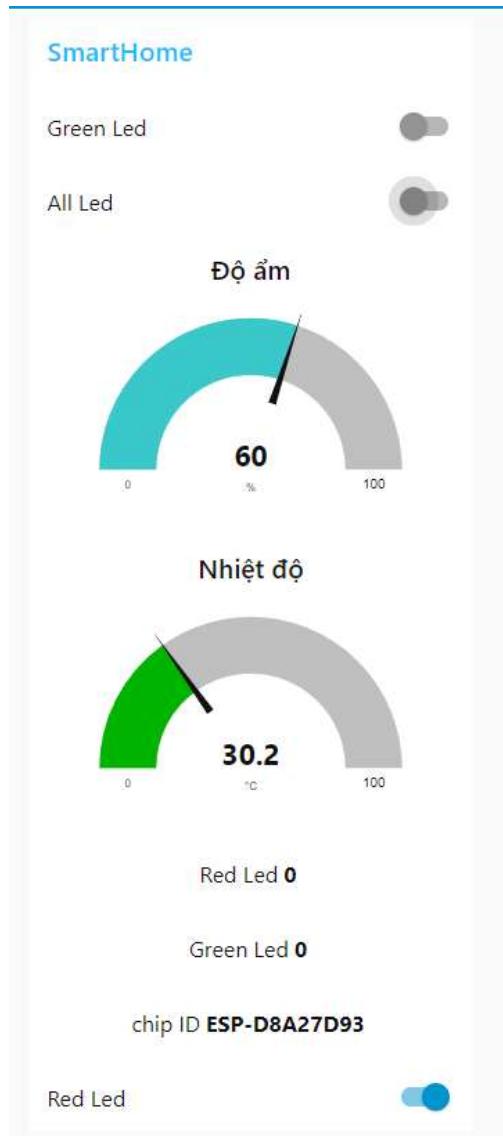


Hình 5.43 thuộc tính độ ẩm

Sau khi thay đổi các thuộc tính như trên thì nhấn Deploy màu đỏ bên phải. Giờ hưởng thụ thành quả thôi nào.

Mở tab google mới rồi tìm **localhost:1880/ui** để nhìn UI dashboard trên Node-red

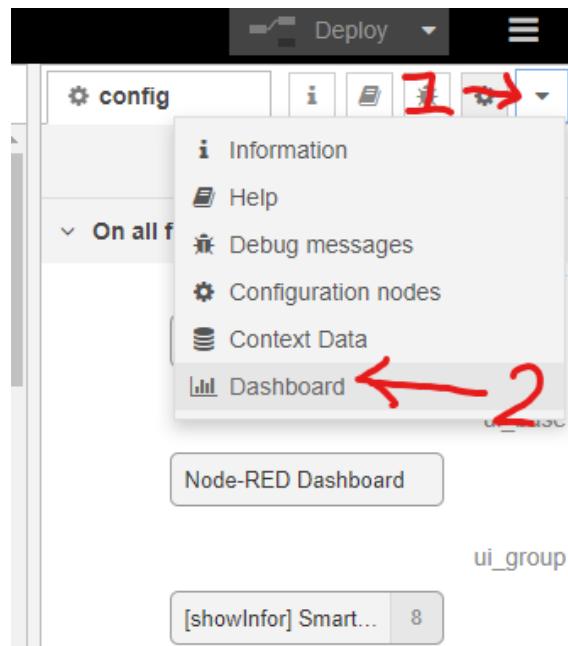
## Chương 1: Cơ sở lý thuyết



Hình 5.44 dashboard

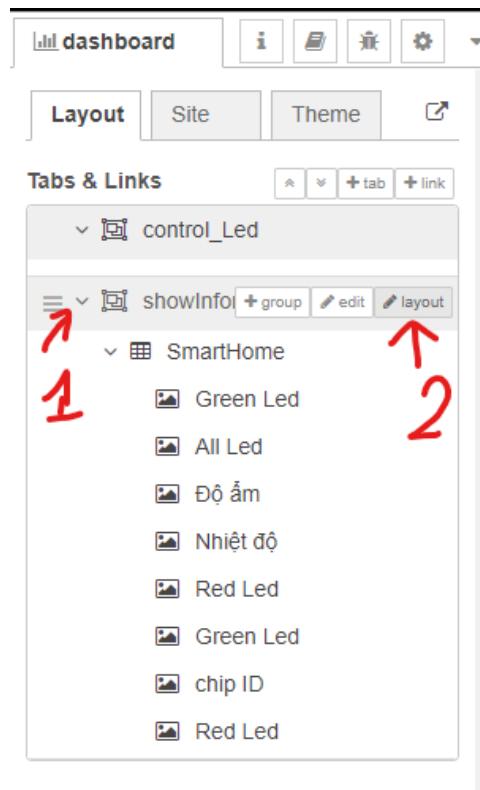
Và nhìn dashboard này hơi lộn xộn nên các bạn có thể chỉnh vị trí các dashboard này theo các bước dưới nhé.

## Chương 1: Cơ sở lý thuyết



Hình 5.45 chỉnh dashboard

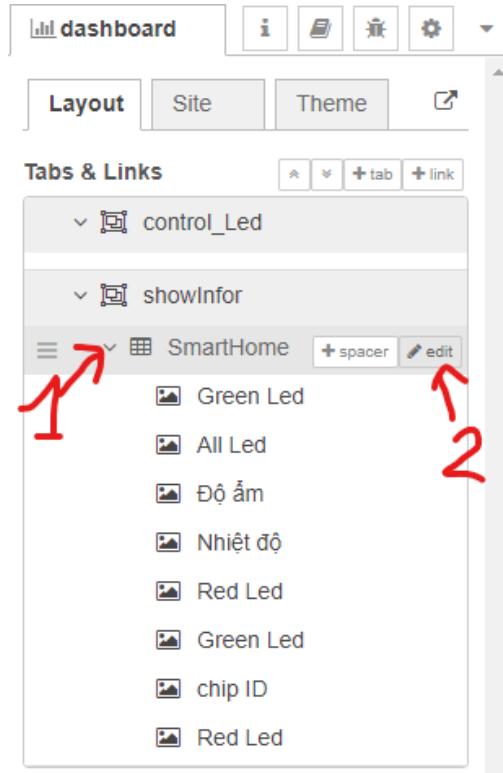
(Hình 5.46) MỞ tab showInfor rồi ấn vào nút layout để chỉnh sửa



Hình 5.46 chỉnh dashboard

(Hình 5.47) kéo thả để chỉnh sửa

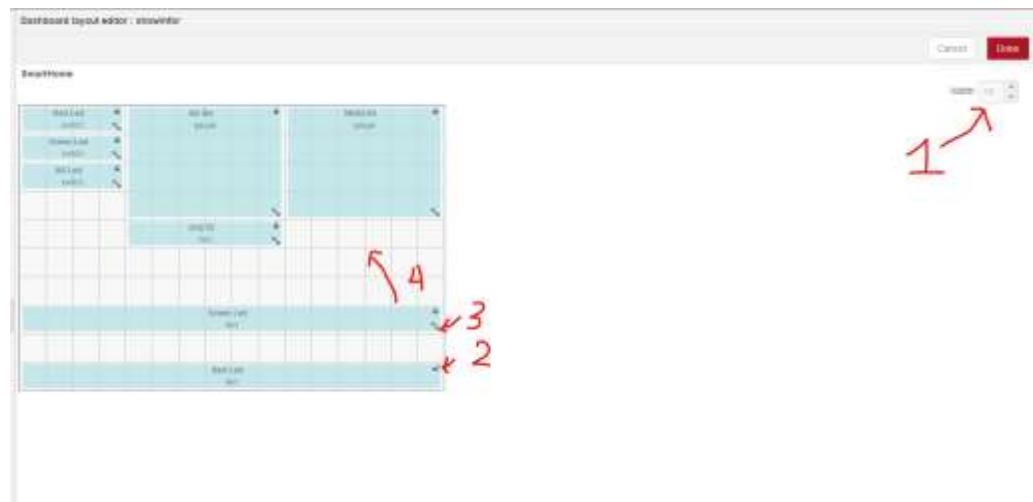
## Chương 1: Cơ sở lý thuyết



Hình 5.47 chỉnh dashboard

### (Hình 5.48) chỉnh sửa giống với dashboard

- (1) chỉnh kích thước phù hợp với màn hình
- (2) mở khóa chỉnh sửa
- (3) nhấn để chỉnh sửa kích thước
- (4) Kéo đến vị trí muốn sắp xếp



Hình 5.48 chỉnh dashboard

## Chương 1: Cơ sở lý thuyết

Giờ thì ấn **Deploy** rồi qua bên tab **localhost:1880/ui** để xem lại thành quả nhé.



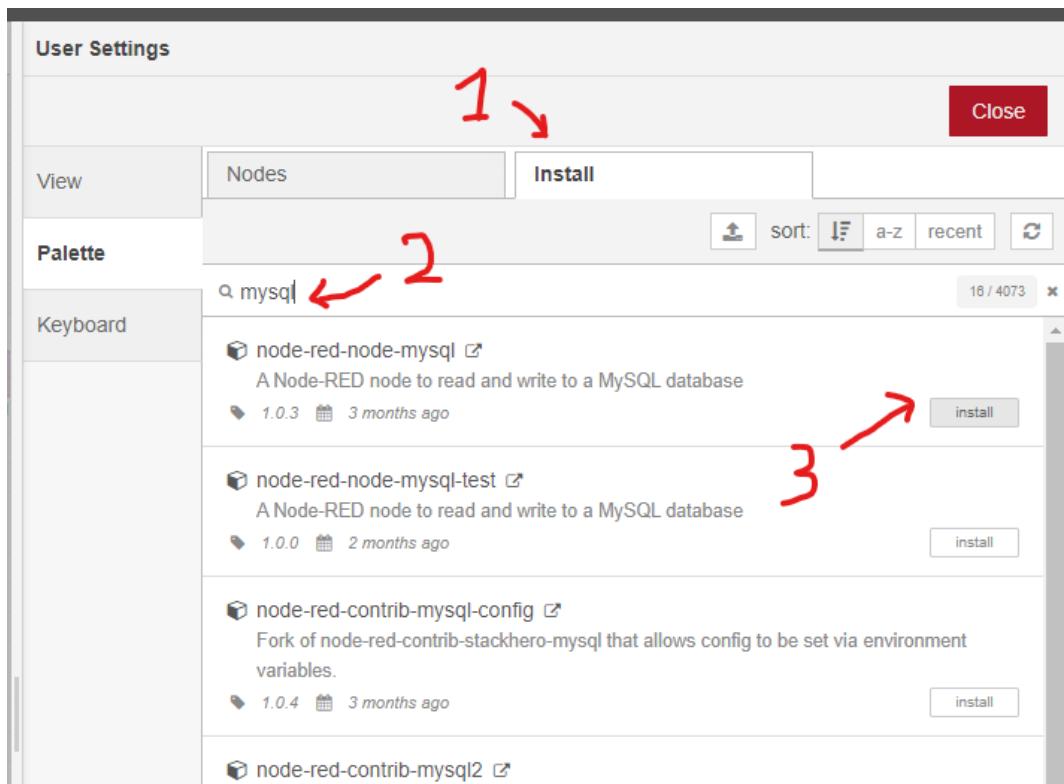
Hình 5.49 dashboard sau khi đã hiển thị

### 5.1.5 Hoàn thiện lưu trữ dữ liệu xuống database:

#### a) Thêm thư viện MySQL

Hãy khởi động node-red trước khi vào **localhost:1880** nhé.

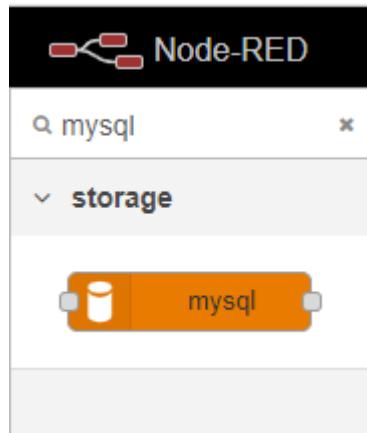
Sau đó vào **Manager Palete** rồi nhân vào **install(1)** tìm từ khóa “**mysql**” rồi **install(3)** cái packet đầu tiên (**node-red-node-mysql**)



Hình 5.50 cài đặt packet MySQL

## Chương 1: Cơ sở lý thuyết

Bạn qua thanh tìm kiếm node với từ khóa “mysql”

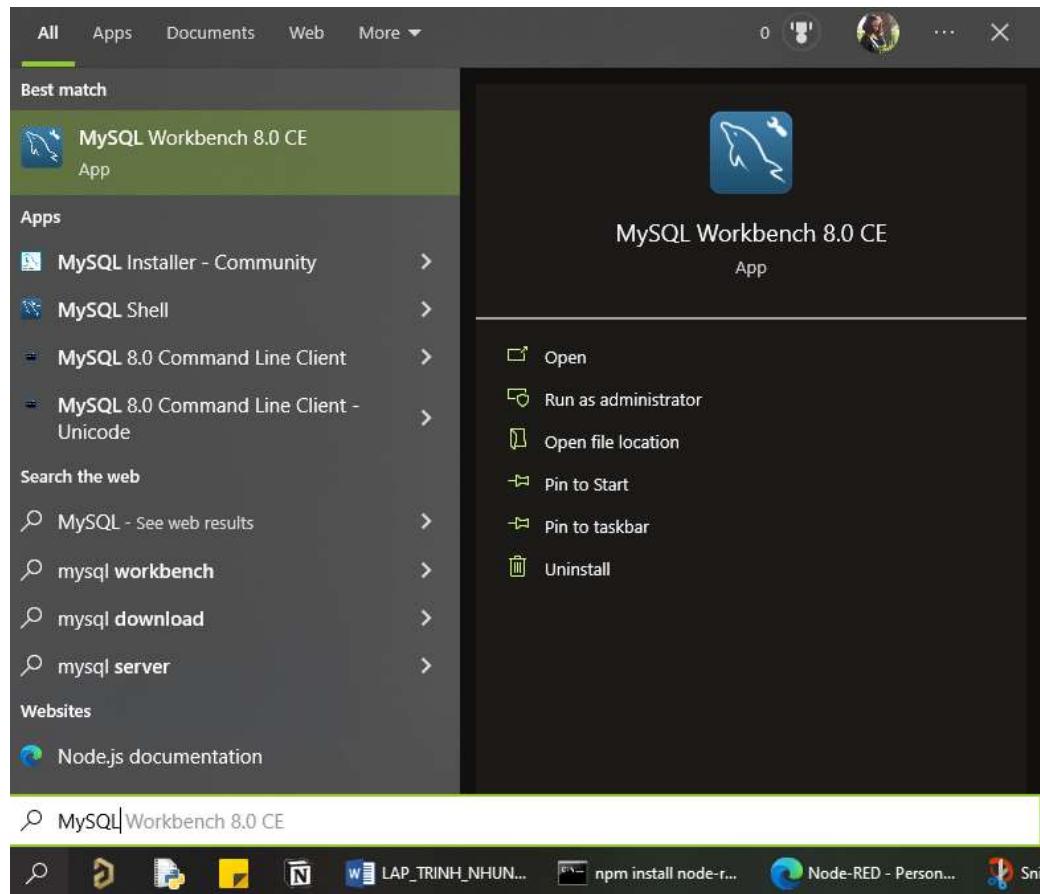


Hình 5.51 node mysql

### b) Truy xuất dữ liệu từ node MySQL

Trước khi sử dụng node mysql thì các bạn phải khởi động database đã nhé.

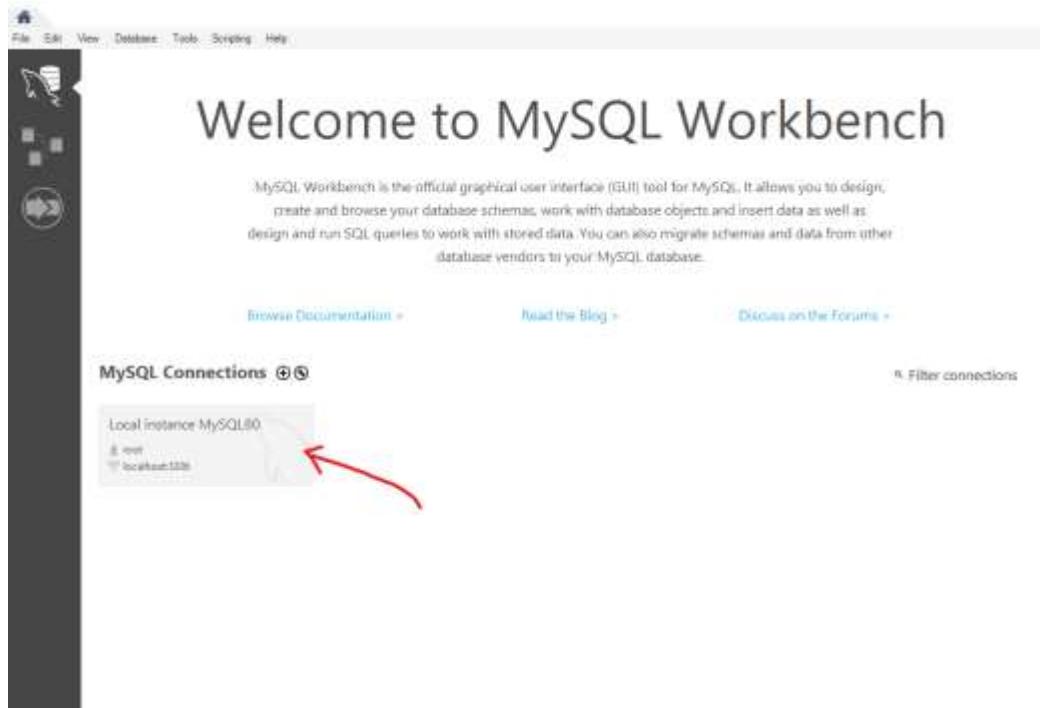
**Bước 1:** mở MySQL Workbench lên nhá.



Hình 5.52 mở MySQL Workbench

## Chương 1: Cơ sở lý thuyết

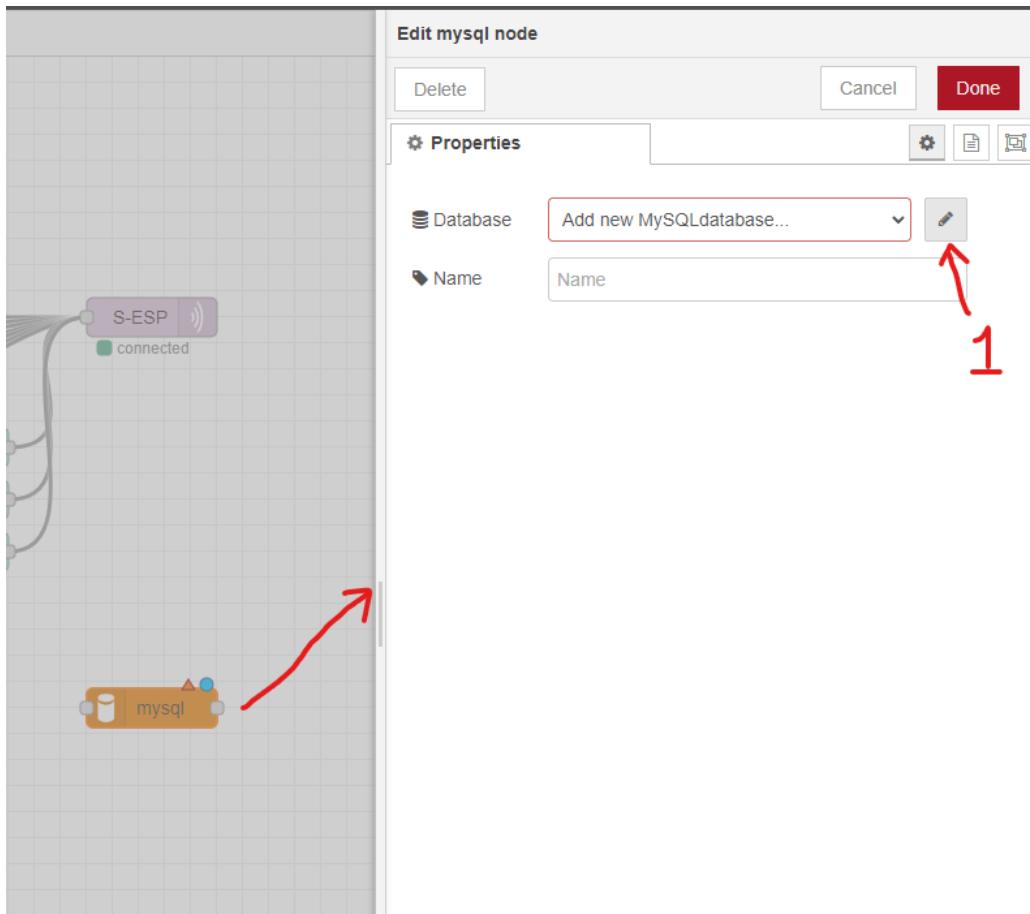
### Bước 2: chọn user



Hình 5.53 chọn user MySQL Workbench

**Bước 3:** Như vậy là ta đã khởi động xong database giờ thì mở lại node-red (localhost:1880) và kéo node mysql ra ngoài. Rồi nhấn đúp vào và chọn **Add new MySQLdatabase...** (hình 5.54)

## Chương 1: Cơ sở lý thuyết

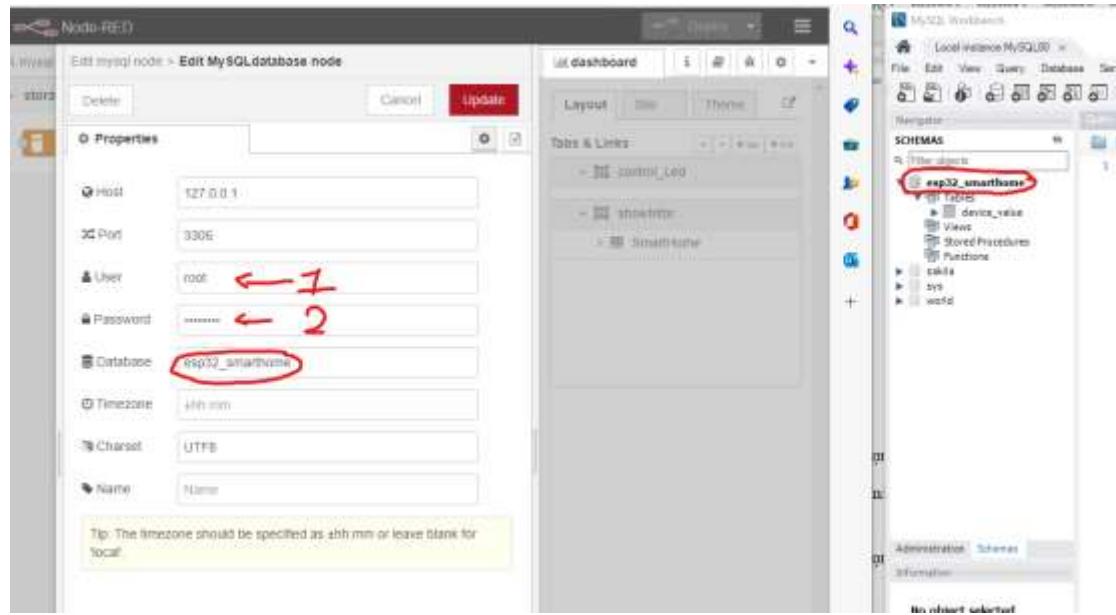


Hình 5.54 chọn user MySQL Workbench

### Bước 4: Chính các thuộc tính như mình nhé

- (1) ghi tên bạn vừa chọn ở bước 2
- (2) mật khẩu vào tài khoản ấy
- (3) ở phần database thì các bạn ghi giống với database bùa bạn đặt tên như câu mình là **esp32\_smarthome**

## Chương 1: Cơ sở lý thuyết



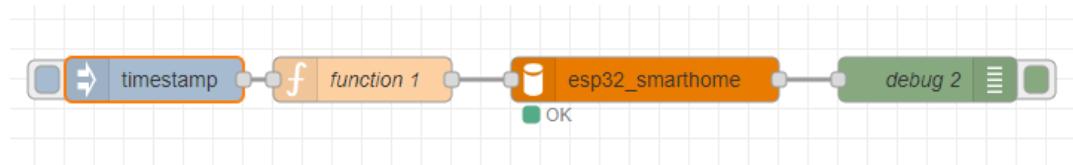
Hình 5.55 chọn user MySQL Workbench

Nếu node hiển thị là connected có nghĩa rằng bạn đã kết nối với database thành công rồi nhé.



Hình 5.56 connected database

**Bước 5:** các bạn kéo node inject, node function, node debug ra và nối như hình 5.56 nhé



Hình 5.56 flow truy xuất data từ database

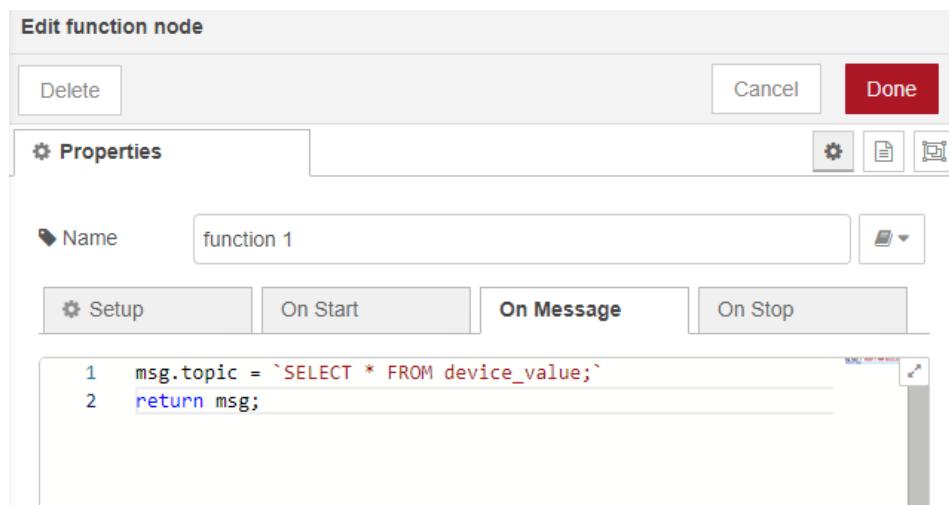
- Ở node inject các bạn không cần chỉnh sửa gì hết
- Node function 1 các bạn nhấn đúp lên và viết code theo mình nếu các bạn không biết gì về truy vấn dữ liệu. Còn bạn biết về truy vấn dữ liệu thì có thể nhập bất kỳ câu truy vấn nào và gán vào msg.topic thì các bạn sẽ lấy được kết quả nhé.

## Chương 1: Cơ sở lý thuyết

```
msg.topic = `SELECT * FROM device_value;`  
return msg;
```

Nội dung câu truy vấn này là Chọn tất cả từ bảng device\_value

Cấu trúc câu là : `SELECT <column_name> FROM <table_name>;`

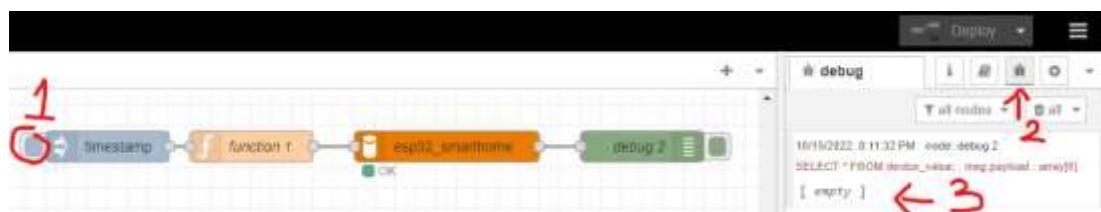


Hình 5.57 nội dung trong node function

- Node mysql thì mình đã hướng dẫn ở bước trước.
- Node debug thì cũng không cần chỉnh sửa gì.

(Hình 5.58) kiểm tra xem ta viết truy vấn có đúng k nhé?

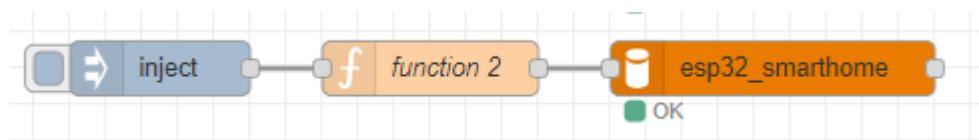
- o (1) Các bạn nhấn vào nút bên cạnh để bắt đầu truy vấn
- o (2)ấn vào con bọ để xem debug
- o (3) nếu mà hiện chữ empty này là đúng nha (vì ta chưa thêm gì vào bảng thì làm sao có gì để mà truy xuất được dữ liệu)



Hình 5.58 kiểm tra có đúng cấu trúc truy vấn dữ liệu không

### c) Thêm dữ liệu vào database từ node MySQL

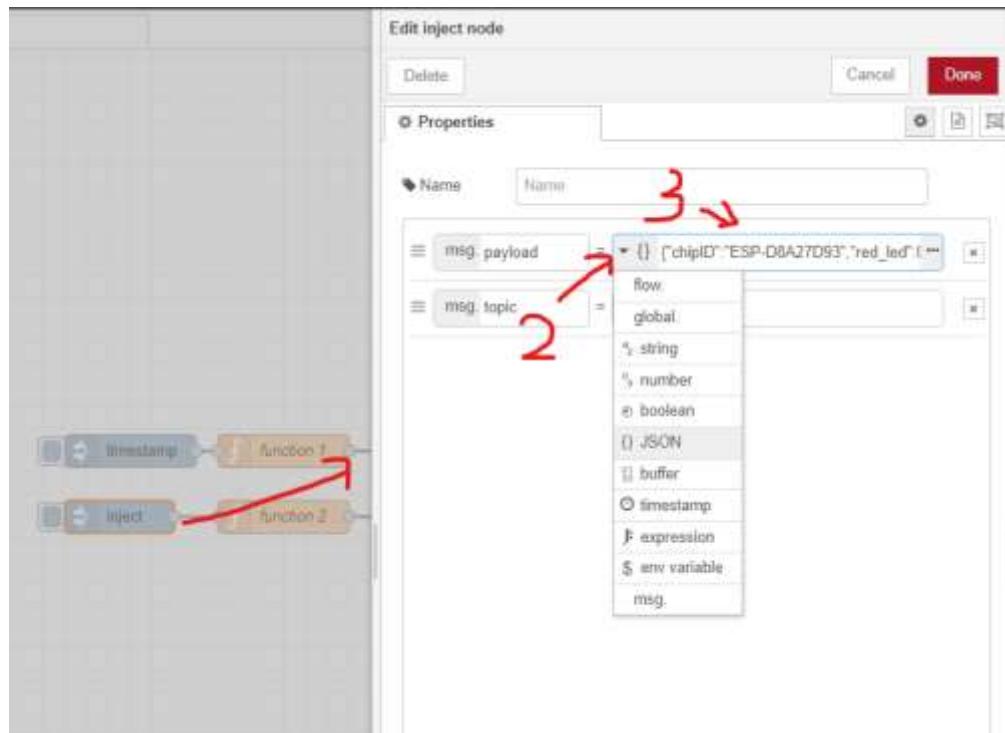
Các bạn kéo node inject, node function ra và nối như hình 5.59 nhé



## Chương 1: Cơ sở lý thuyết

Hình 5.59 flow thêm data vào database

- Node inject các bạn thay đổi các thuộc tính như (hình 5.60)
  - o (1) nhấn đúp vào node inject
  - o (2) thay đổi timestamp thành JSON
  - o (3) copy dữ liệu ESP gửi lên vào ({ "chipID": "ESP-D8A27D93", "red\_led": 0, "green\_led": 0, "hum": 65, "temp": 29.5, "now": "952496" })



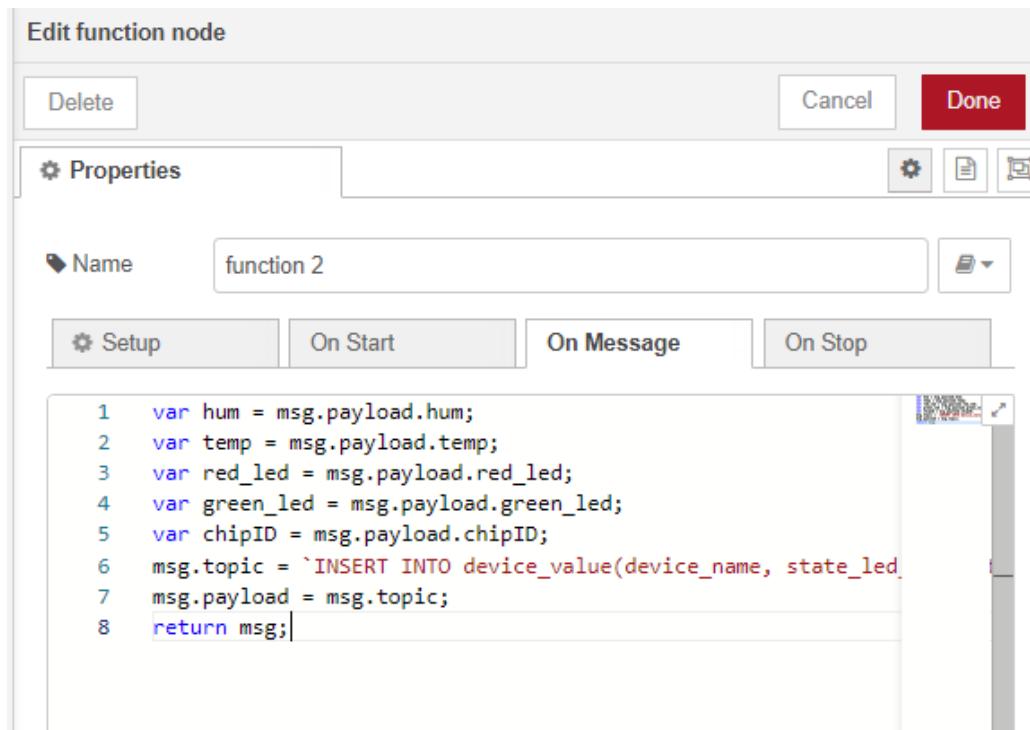
Hình 5.60 sửa thuộc tính node inject

- Node function các bạn thay đổi nội dung dưới như mình nhé (hình 5.61)  
(

```
var hum = msg.payload.hum;
var temp = msg.payload.temp;
var red_led = msg.payload.red_led;
var green_led = msg.payload.green_led;
var chipID = msg.payload.chipID;
msg.topic = `INSERT INTO device_value(device_name, state_led_red,
state_led_green, hum, temp, create_time) VALUES("${chipID}",
${red_led}, ${green_led}, ${hum}, ${temp}, now());`;
msg.payload = msg.topic;
return msg;
```

)

## Chương 1: Cơ sở lý thuyết



Hình 5.61 thêm code vào node function

ở câu truy vấn này thì có nghĩa rằng thêm vào bảng các giá trị sau.

Cấu trúc của câu là: **INSERT INTO <table\_name> (<column1\_name>,<column2\_name>,<column3\_name>,...) VALUES (<value1>,<value2>,<value3>,...)**

Các bạn có thể tìm hiểu thêm bằng cách lên mạng nhé. ☺

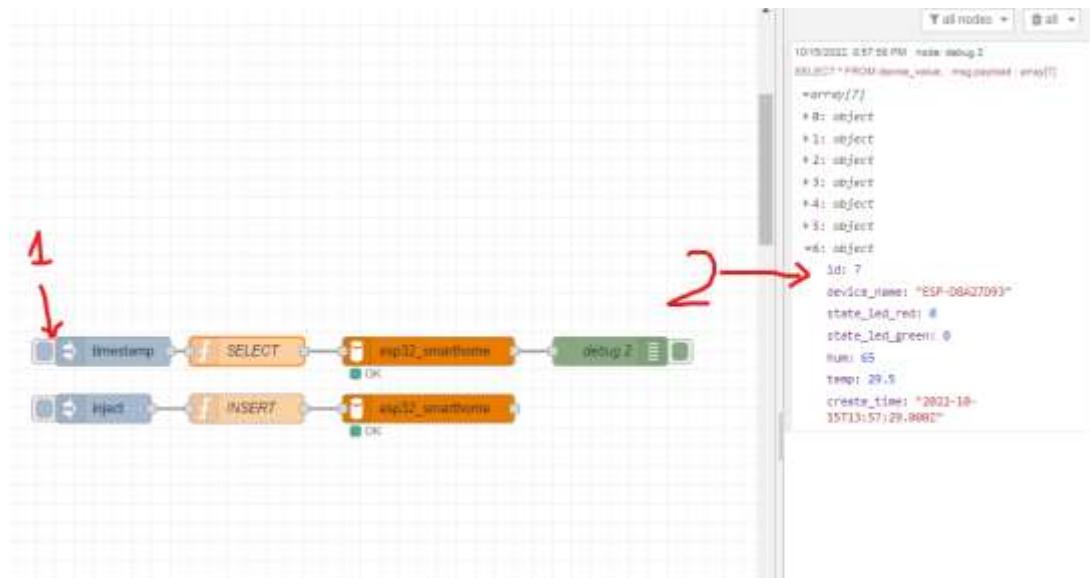
Sau khi xong thì các bạn án nút inject nếu dưới database ghi chữ OKE thì đã thêm thành công



Hình 5.62 kiểm tra có đúng cấu trúc thêm dữ liệu không

**NOTE: nãy ta truy vấn không có dữ liệu giờ thì các bạn hãy án nút inject ở phần 5.1.5 b) để xem ta có dữ liệu chưa nhé (hình 5.63)**

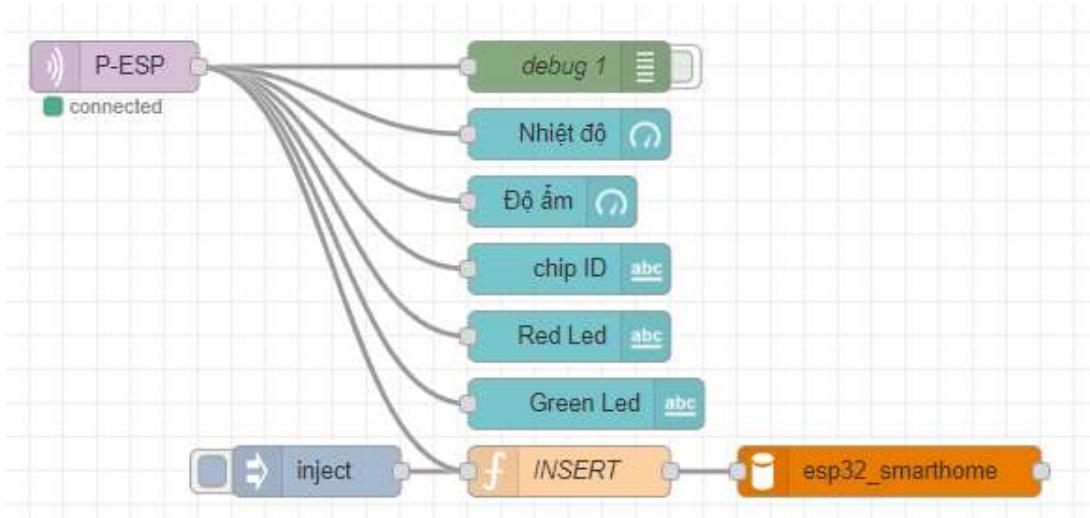
## Chương 1: Cơ sở lý thuyết



Hình 5.63 truy vấn dữ liệu

### d) Hiển thị và lấy dữ liệu từ database

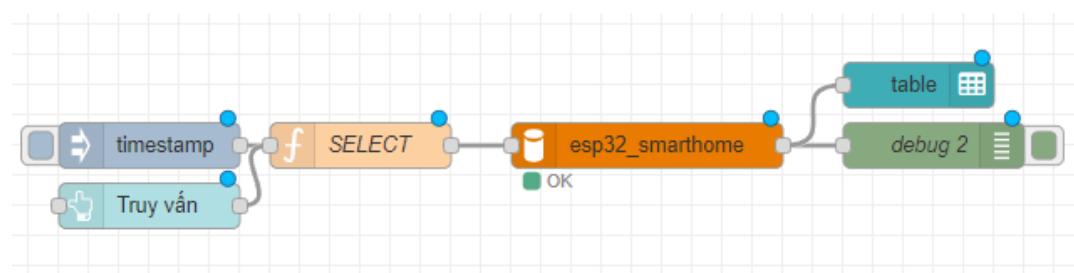
Giờ thì nối node(P-ESP) vào node INSERT để lưu dữ liệu vào database nhé



Hình 5.64 flow hoàn chỉnh

Để có thể xem dữ liệu trên **localhost:1880/ui** thì các bạn tải thêm **packet (node-red-node-ui-table)** này nữa nhé. Và nối các node như hình bên dưới (hình 5.65). Node table và node button thì các bạn không cần chỉnh gì hết chỉ cần nối vào là được.

## Chương 1: Cơ sở lý thuyết



Hình 5.65 flow hoàn chỉnh để truy vấn database

Và tada ta có thể truy xuất dữ liệu khi được nhấn vào nút “truy vấn”



Hình 5.65 hiển thị bảng sau khi truy vấn

### 5.2 Tổng kết chương:

Như vậy mình đã hướng dẫn xong project smart home rồi nhé. Vì đây là hướng dẫn cơ bản nên code ở firmware còn sơ sài, node-red còn khá đơn giản, database thì đơn thuần. Nhưng ở firmware thì mình đã hướng dẫn các bạn form của một project thì cần những file gì rồi (your\_project.ino, app.h, configs.h). Node-red thì mình cũng đã hướng dẫn các bạn một vài node cơ bản rồi, các node khác các bạn có thể ấn vào hình quyền sách bên cạnh con bọ (debug) để xem hướng dẫn nhé. MySQL mình cũng đã hướng dẫn các bạn tạo ra database, bảng trên MySQL Workbench và cũng đã hướng dẫn 2 lệnh cơ bản được dùng khá phổ biến khi truy vấn dữ liệu (INSERT INTO, SELECT). Và các bạn cũng có thể lên google để tìm hiểu thêm về ngôn ngữ truy vấn này nhé.

## CHƯƠNG VI : TỔNG KẾT

### 1. Kết luận và nhận xét:

#### - **Ưu điểm:**

- + Firmware ở ESP rất dễ quản lý, với form có thể tái sử dụng nhiều lần và không cần phải code đi code lại nhiều lần.
- + Node-red là cầu nối có thể liên kết được ESP(firmware), database(MySQL) và giao diện người dùng (Node-red dashboard). Và Node-red hoạt động ổn định.
- + MQTT sever hoạt động ổn định, truyền dữ liệu bằng phương thức MQTT rất nhanh.
- + Có giao diện để người dùng có thể thao tác trực tiếp lên.

#### - **Nhược điểm:**

- + Firmware sẽ hơi phức tạp với những người mới tiếp xúc.
- + ESP sẽ không chạy được tiếp nếu kết nối MQTT thất bại.
- + Node-red và database vẫn còn phải khởi động bằng tay.
- + Giao diện web còn sơ sài và chưa được đẹp.

### 2. Định hướng phát triển tương lai:

Về Firmware: sẽ sử dụng dual-core một core chạy các kết nối và một core để chạy các ứng dụng. Vì vậy có thể khắc phục được tình trạng không kết nối thì sẽ không chạy tiếp được.

Về Node-red: Sẽ chạy node-red bằng service thông qua pm2. Vì vậy có thể khắc phục được tình trạng mỗi lần mở lại máy là phải khởi động lại node-red bằng tay.

Về MySQL: Cũng sẽ chạy database bằng service thông qua pm2. Vì vậy có thể khắc phục được tình trạng mỗi lần mở lại máy là phải khởi động lại database bằng tay.

Về giao diện: Sẽ viết thêm các node template để CSS lại giao diện cho đẹp hơn, chỉnh sửa Theme cho dashboard.

### 3. Tổng kết:

Về quyển sách này còn khá nhiều lỗi chính tả hay một vài chỗ còn lủng củng rất mong mọi người thông cảm. Vì là vừa thực hiện vừa ghi chép lại và vừa chỉnh sửa cho phù hợp nên có thể code sẽ không giống với lúc giải thích đôi chút. Nhưng về cơ bản thì không thay đổi nhiều. Đặc biệt đây là quyển sách hướng dẫn các bạn theo kinh nghiệm cá nhân nên có thể đúng hoặc có thể còn thiếu sót và cũng không thể tránh khỏi những sai sót

Cuối cùng mình xin cảm ơn thầy Chung Tân Lâm đã tạo điều kiện tốt nhất để mình có thể chia sẻ những kinh nghiệm cá nhân mình đã học được trong đợt thực tập lần này.

## **TÀI LIỆU THAM KHẢO**

Danh mục các website tham khảo:

[1] Nghiên cứu, tham khảo tổng quan Module ESP32

<https://vi.wikipedia.org/wiki/ESP32>

<https://www.espressif.com/en/products/socs/esp32/>

<https://deviot.vn/tutorials/esp32/>

[2] Nghiên cứu, tham khảo tổng quan về giao thức MQTT

<https://mqtt.org/getting-started/>

<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/>

[3] Nghiên cứu, tham khảo tổng quan về MySQL

<https://dev.mysql.com/doc/>

<https://www.w3schools.com/sql/>

[4] Nghiên cứu, tham khảo tổng quan về Node-red

<https://nodered.org/docs/>