

4ID3 - IoT Devices and Networks

Communicating Sensor Data over WiFi using MQTT

Adam Sokacz, Ishwar Singh, and Salman Bawa

Sponsored by *Future Skills Center, Canada* and *McMaster W Booth SEPT*

Group Number:

Name, Student Number:

Name, Student Number:

Name, Student Number:

Submission Date:



Objective

In this lab, we will be constructing a basic IoT network with a single node that collects data from connected sensors and communicates that data to a remote MQTT broker using a local WiFi connection.

Contents

Objective	2
Feedback	3
Additional Resources	3
Pre-Lab Questions	4
Post-Lab Questions	4
Pulling from GitHub	6
Wiring Diagram	7
Reading Sensor Data	8
Publishing to an MQTT Broker	18
Public Broker	18
Hotspotting your Microcontroller.....	18
Code Changes.....	21
Verifying Connection.....	27
Using a Desktop Terminal Application	27
Using a Mobile Client Application.....	29
Using a Desktop Client Application.....	31
NodeRED Dashboard.....	33
Visualizing NodeRED Data.....	38
Saving and Pushing Your Project.....	44
Exporting a NodeRED Flow as JSON.....	44
Pushing Changes to GitHub.....	46

Feedback

Q1 - What would you rate the difficulty of this lab?

(1 = easy, 5 = difficult)

1

2

3

4

5

Comments about the difficulty of the lab:

Q2 - Did you have enough time to complete the lab within the designated lab time?

YES

NO

Q3 - How easy were the lab instructions to understand?

(1 = easy, 5 = unclear)

1

2

3

4

5

List any unclear steps:

Q4 - Could you see yourself using the skills learned in this lab to tackle future engineering challenges?

(1 = no, 5 = yes)

1

2

3

4

5

Additional Resources

Mosquitto MQTT Broker Tutorial (<https://youtu.be/DH-VSAACtBk>)

NodeRED Fundamentals Tutorial (<https://youtu.be/3AR432bguOY>)

ESP8266 Overview (<https://youtu.be/dGrJi-ebZgl>)

Lab GitHub Repo (https://github.com/sokacza/4ID3_Labs)

Pre-Lab Questions

Q1 - In your own words, describe the **publish-subscribe messaging pattern**. What role does the broker play? What role do the clients play?

(Suggested: 2 sentences)

Q2 - In your own words, what does **QoS** mean for MQTT transmissions? What **levels** of QoS exist?

(Suggested: 3 sentences)

Q3 – In your own words, what is the role of each of the **fields below**, when connecting to an MQTT broker?

Host IP Address	
Topic Name	
Protocol (tcp/ws/wss/tls)	

Post-Lab Questions

Q1 – Using software like **PowerPoint** or **Draw.IO**, create a diagram to represent the nodes in this IoT network and **label the data** being exchanged between each node.

(Suggested: Diagram, 5 points)

Q2 - Explain your **observations** of the Node-Red user interface as you interact with the sensors. Is the change instantaneous? What changes?

(Suggested: 2 sentences, 2 points)

Q3 – If the microcontroller loses battery, will the **NodeRED dashboard** still be connected to the MQTT server? Explain your answer.

(Suggested: 3 sentences, 2 points)

Q3 – Draw a diagram or in a detailed paragraph, explain the differences in connecting an **analog**, **IIC**, and **SPI** sensor to an ESP-based development board. The submission must include **pin connections** on the development board and where they correspond to on each sensor.

(Suggested: Sketch, 5 points)

Q4 – Using the official Paho-MQTT examples, write a short **Python script** that subscribes to the same topic and MQTT broker used in this lab, and **prints** the published payload to the terminal window.

<https://github.com/eclipse/paho.mqtt.python/tree/master/examples>

(Suggested: Python script, 5 points)

Q5 – Using a remote MQTT broker on the cloud allows client Node-RED dashboards to visualize sensor data without needing to be in the same **geographical location** as the network. Explain **2 security** vulnerabilities with the network established in this lab and **2 ways** these issues could be **mitigated**.

(Suggested: 4 sentences, 2 points)

Q6 - Below, write a **LinkedIn post** about **4 key learning takeaways** from this lab.

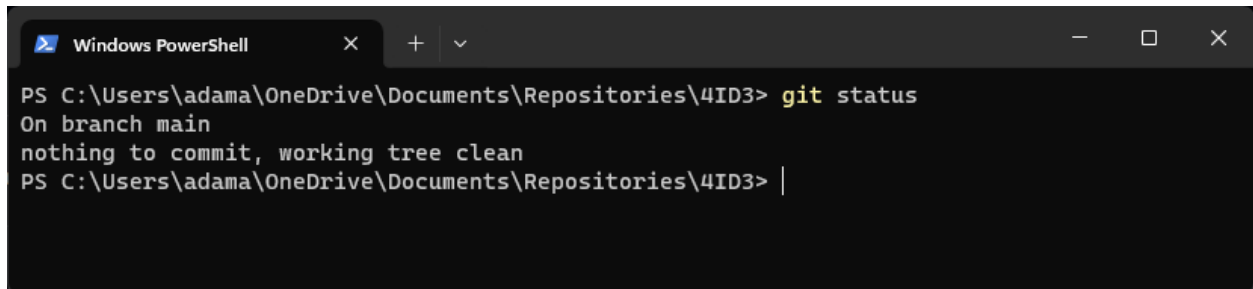
Communicating Sensor Data over WiFi using MQTT

(Suggested: Paragraph or screenshot, 2 points)

Pulling from GitHub

View the status of your local repository to verify that there aren't any uncommitted changes.

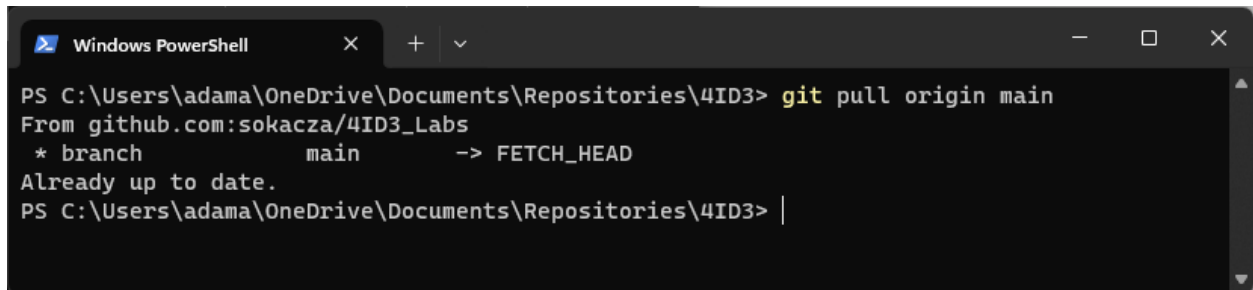
git status

A screenshot of a Windows PowerShell terminal window. The title bar shows 'Windows PowerShell' with standard window controls. The terminal text shows the command 'git status' being executed in the directory 'C:\Users\adama\OneDrive\Documents\Repositories\4ID3'. The output indicates the repository is on the 'main' branch and the working tree is clean.

```
PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3> git status
On branch main
nothing to commit, working tree clean
PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3> |
```

Pull changes from GitHub to ensure that your local repository is up-to-date.

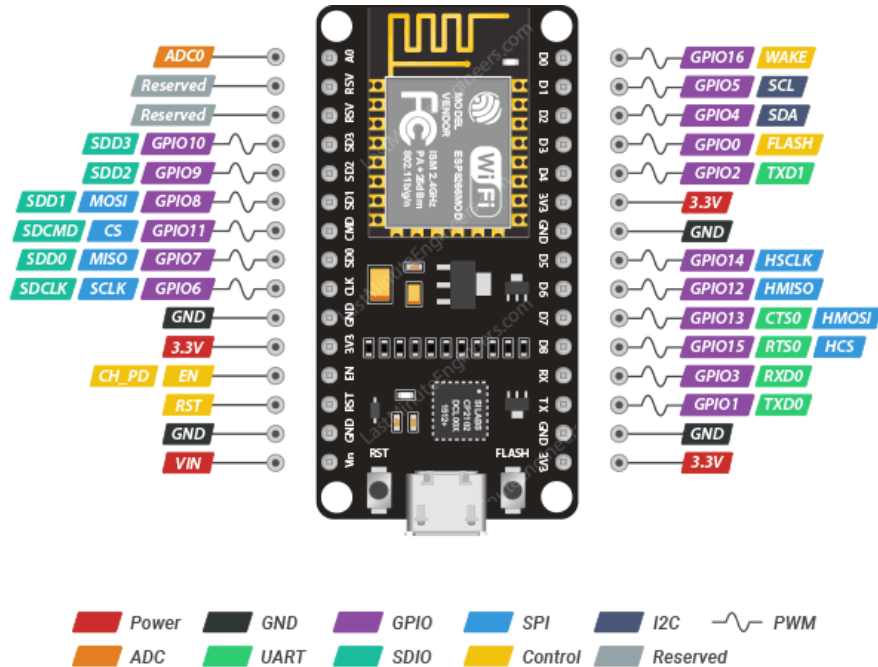
git pull origin main

A screenshot of a Windows PowerShell terminal window. The title bar shows 'Windows PowerShell' with standard window controls. The terminal text shows the command 'git pull origin main' being executed in the directory 'C:\Users\adama\OneDrive\Documents\Repositories\4ID3'. The output shows the source as 'github.com:sokacza/4ID3_Labs', the branch mapping, and a confirmation that the repository is already up to date.

```
PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3> git pull origin main
From github.com:sokacza/4ID3_Labs
 * branch          main      -> FETCH_HEAD
Already up to date.
PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3> |
```

Wiring Diagram

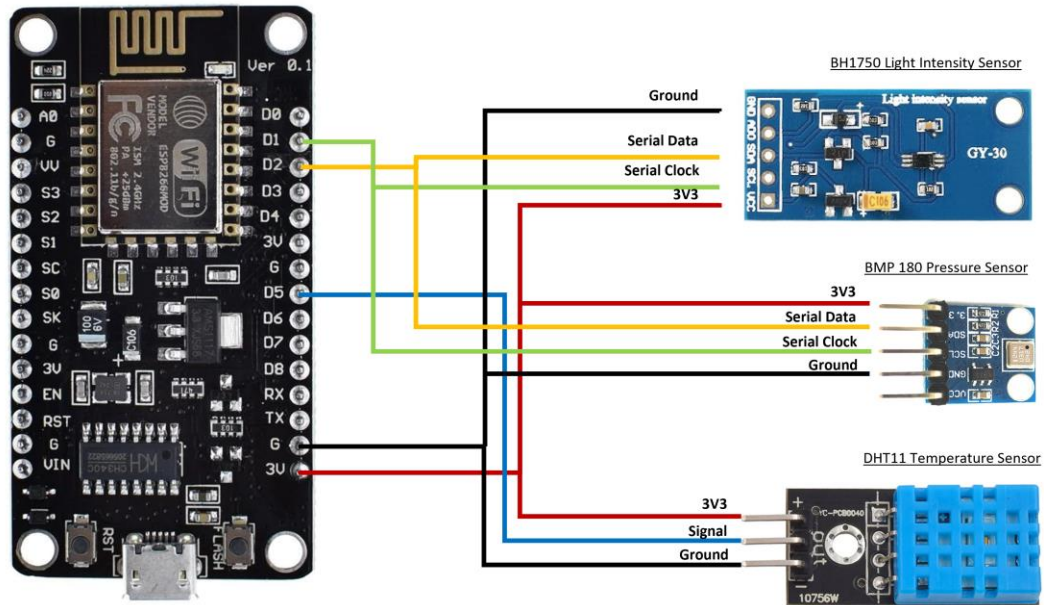
ESP8266 Development Board Pinout:



ESP8266 NodeMCU Pinout



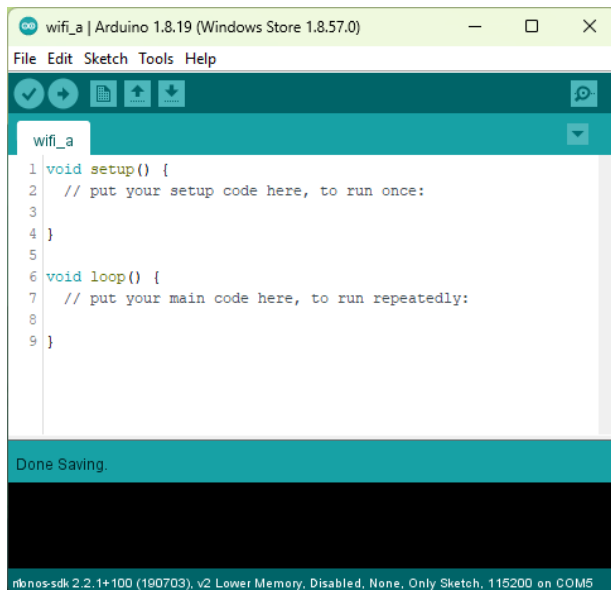
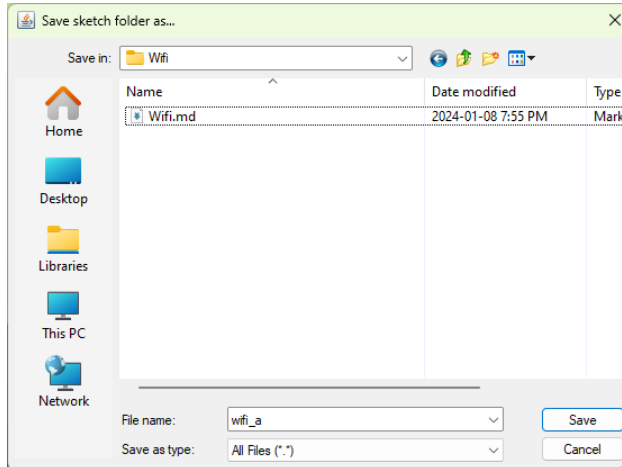
ESP8266 Development Board



Reading Sensor Data

The goal of this section of the lab is to read data from 3 sensors and print them to the serial monitor, before we communicate to a server.

Launch the Arduino IDE and **Save as** into your lab 1 folder.



The 3 sensors we will be using are as follows:

- DHT11 for Temperature and Humidity
- BMP180 for Pressure
- BH1750 for Light Intensity

Ensure that the following libraries are installed from the **Arduino Library Manager**:

Communicating Sensor Data over WiFi using MQTT

- Adafruit Unified Sensor by Adafruit
- Adafruit BMP085 Unified by Adafruit
- Hp_BH1750 by Stefan Armbrorst
- PubSubClient by Nick O'Leary
- The ESP8266 driver as described in the pre-lab

Adafruit BMP085 Unified

by Adafruit Version 1.1.1 **INSTALLED**

Unified sensor driver for Adafruit's BMP085 & BMP180 breakouts Unified sensor driver for Adafruit's BMP085 & BMP180 breakouts
[More info](#)

Adafruit Unified Sensor

by Adafruit Version 1.1.7 **INSTALLED**

Required for all Adafruit Unified Sensor based libraries. A unified sensor abstraction layer used by many Adafruit sensor libraries.
[More info](#)

hp_BH1750

by Stefan Armbrorst Version 1.0.2 **INSTALLED**

Digital light sensor breakout boards containing the BH1750FVI IC high performance non-blocking BH1750 library
[More info](#)

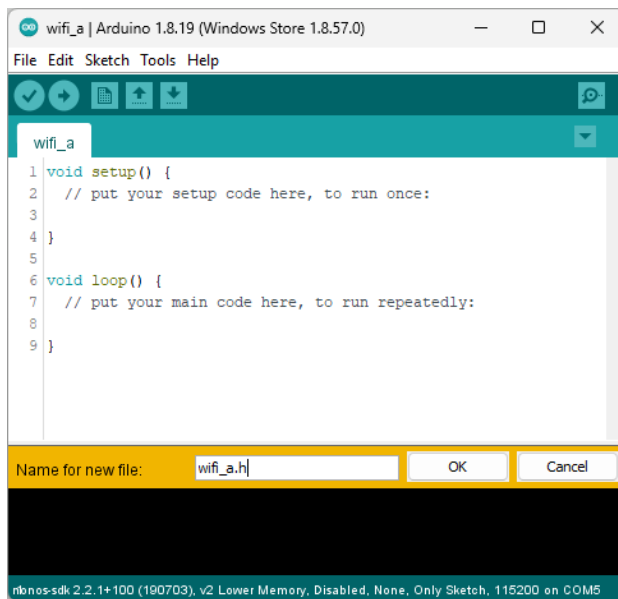
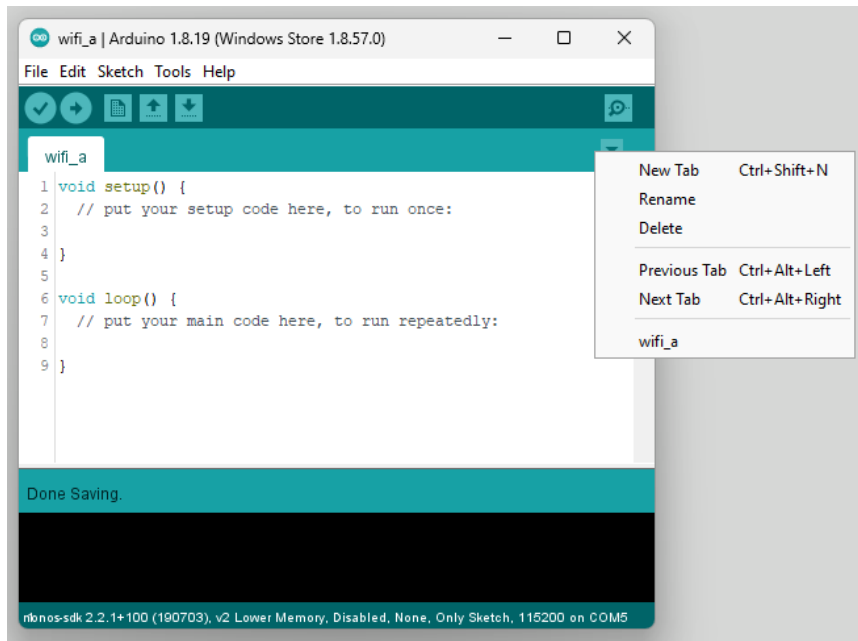
PubSubClient

by Nick O'Leary Version 2.8.0 **INSTALLED**

A client library for MQTT messaging. MQTT is a lightweight messaging protocol ideal for small devices. This library allows you to send and receive MQTT messages. It supports the latest MQTT 3.1.1 protocol and can be configured to use the older MQTT 3.1 if needed. It supports all Arduino Ethernet Client compatible hardware, including the Intel Galileo/Edison, ESP8266 and TI CC3000.
[More info](#)

Create a new header file. Name this file **wifi_a.h**.

Communicating Sensor Data over WiFi using MQTT



In this file, we will keep our dependencies, preprocessor macros, global variables, and global objects.

Include the libraries for each sensor.

File: wifi_a.h

```
//DHT11 Libraries
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>

//BMP180 Libraries
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP085_U.h>

//HP_BH1750 Libraries
#include <hp_BH1750.h>
```

Below, include any macros that you need. Macros act as a find-and-replace. The C++ preprocessor will copy '14' everywhere in the code where 'DHTPIN' is found at compile time.

File: wifi_a.h

```
//HP_BH1750 Libraries
#include <hp_BH1750.h>

//Macros
#define DHTPIN 14
#define DHTTYPE DHT11
#define DELAY_BETWEEN_SAMPLES_MS 5000
```

Lastly, lets instantiate some global objects for classes that we will be using throughout the code.

File: wifi_a.h

```
#define DELAY_BETWEEN_SAMPLES_MS 5000

//Instantiate Sensor Objects
DHT_Unified dht(DHTPIN, DHTTYPE);
Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);
hp_BH1750 BH1750;
```

This concludes the header file. Let's move back to the implementation file.

Firstly, include our header file.

File: wifi_a.ino

```
#include "wifi_a.h"
```

```
void setup(){
```

```
}
```

```
void loop(){
```

```
}
```

Next, set up the void setup() function.

File: wifi_a.ino

```
void setup() {
```

```
  //Start the serial monitor at 115200 baud
```

```
  Serial.begin(115200);
```

```
  //Create a sensor object that is passed into the getSensor method of the dht class
```

```
  //Only the dht sensor requires this
```

```
  sensor_t sensor;
```

```
  dht.temperature().getSensor(&sensor);
```

```
  dht.humidity().getSensor(&sensor);
```

```
  //Run the begin()method on each sensor to start communication
```

```
  dht.begin();
```

```
  bmp.begin();
```

```
  BH1750.begin(BH1750_TO_GROUND);
```

```
}
```

Moving onto the void loop() function, it should be noticed that both the DHT sensor and BMP sensor are part of the same Adafruit unified library, so they will be polled differently than the BH sensor.

File: wifi_a.ino

```
void loop() {  
  
    //Polling the DHT and BMP sensor using events  
    sensors_event_t dhtTempEvent, dhtHumEvent, bmpEvent;  
    dht.temperature().getEvent(&dhtTempEvent);  
    dht.humidity().getEvent(&dhtHumEvent);  
    bmp.getEvent(&bmpEvent);  
  
    //Polling the BH sensor  
    BH1750.start();  
    float lux=BH1750.getLux();  
}
```

Next, we want to print the sensor readings to the serial monitor.

File: wifi_a.ino

```
void loop() {

    [...]

    //Printing sensor readings to serial monitor
    Serial.println("\n-");

    if(!isnan(dhtTempEvent.temperature)){
        Serial.println("Temperature: " + String(dhtTempEvent.temperature) + " degC");
    }
    else{
        Serial.println("Temperature Sensor Disconnected");
    }
    if(!isnan(dhtHumEvent.relative_humidity)){
        Serial.println("Humidity: " + String(dhtHumEvent.relative_humidity) + " %");
    }
    else{
        Serial.println("Humidity Sensor Disconnected");
    }
    if(!isnan(bmpEvent.pressure)){
        Serial.println("Pressure: " + String(bmpEvent.pressure) + " hPa");
    }
    else{
        Serial.println("Pressure Sensor Disconnected");
    }
    if(!isnan(lux)){
        Serial.println("Light Intensity: " + String(lux) + " lux");
    }
    else{
        Serial.println("Lux Sensor Disconnected");
    }

    delay(DELAY_BETWEEN_SAMPLES_MS);

}
```

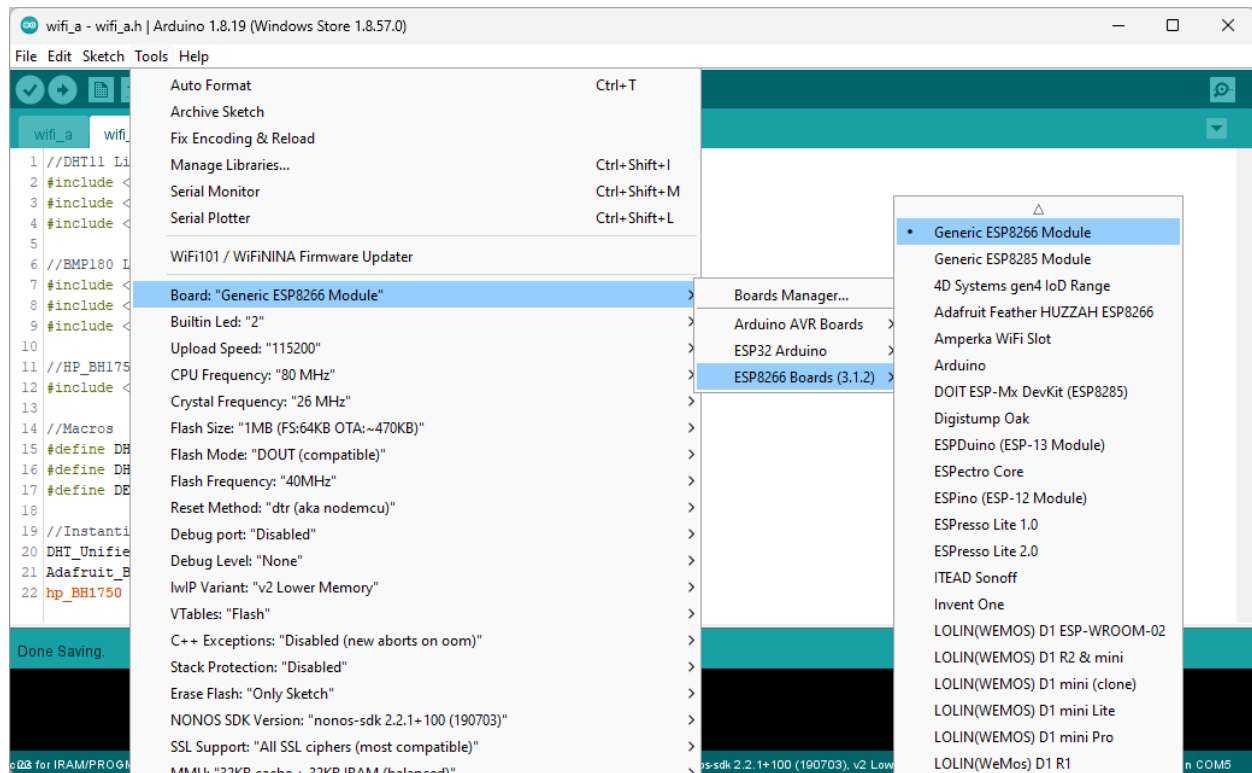
To concatenate different datatypes into a print statement, we must **cast them to strings** and **concatenate them** together. An easy way to do this is to use **String(float value)** to cast and **+** operator to concatenate two strings together.

Thirdly, when printing to the serial monitor, the **print()** command does not include a newline character. In order to print on a new line, we could add it in **print(string + '\n')** or use the **println()** function.

Lastly, we want to let time pass between polls. This can be done simply by adding a delay in the loop.

To upload to the board, change the board to Generic ESP8266 Module.

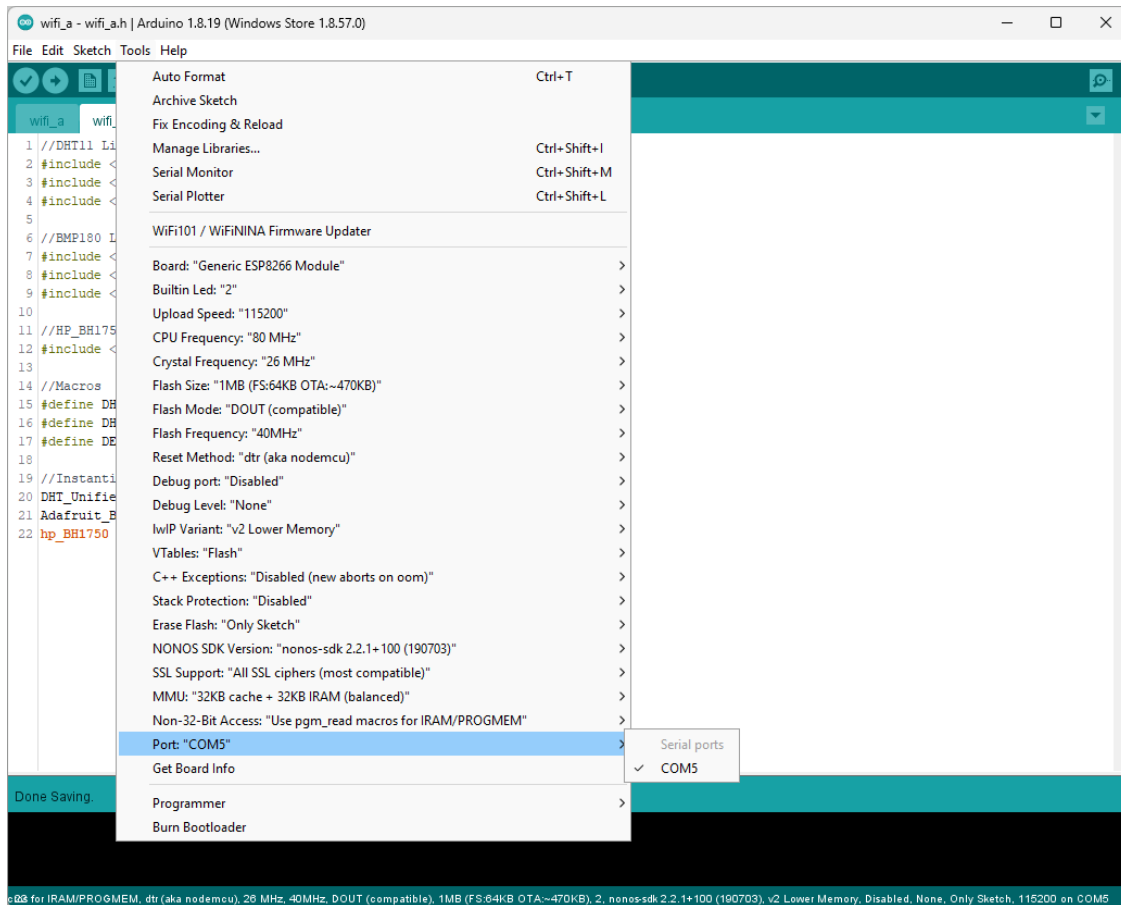
Tools > Board > ESP8266 Boards > Generic ESP8266 Module



Leave the default communication settings the same, with the exception of the COM port. Select the COM port that your microcontroller is connected to.

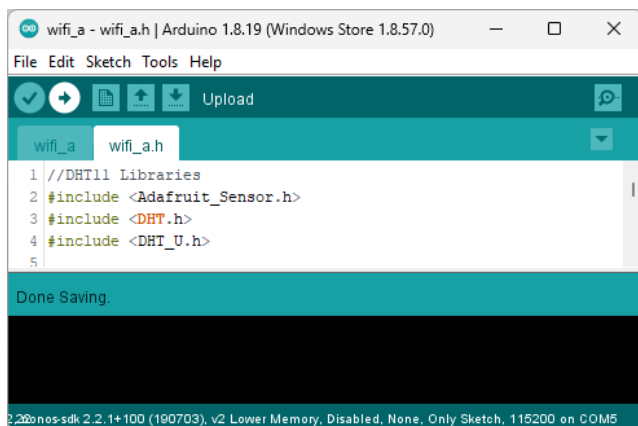
Tools > Port > COM<#>

Communicating Sensor Data over WiFi using MQTT



Press the **upload** button to compile and upload the code. Keep an eye on the terminal window for any errors that arise.

Sketch > Upload



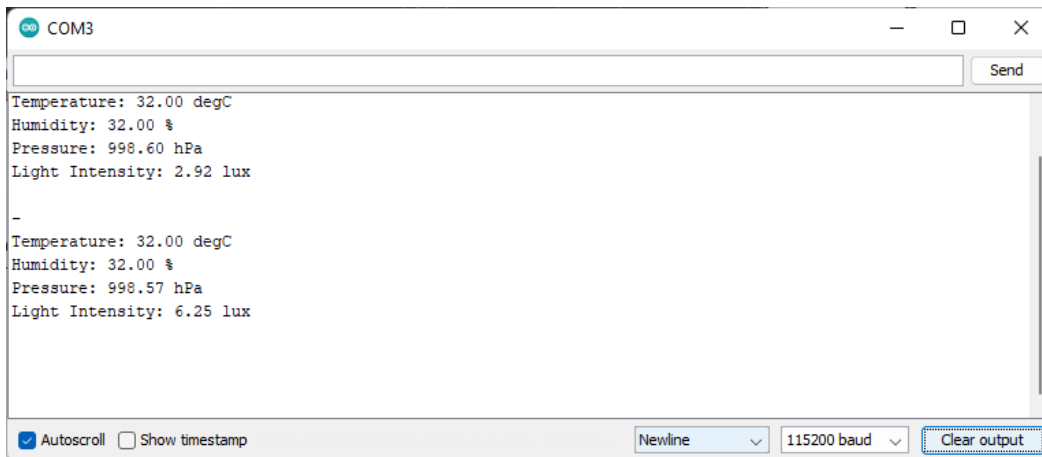
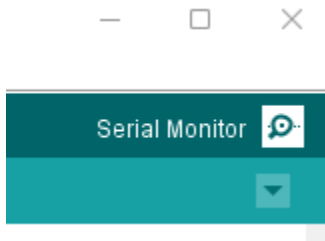
A successful upload should look like this:


```
Crystal is 26MHz
MAC: 48:3f:da:aa:57:09
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0340
Compressed 280256 bytes to 205447...
Writing at 0x00000000... (7 %)
Writing at 0x00004000... (15 %)
Writing at 0x00008000... (23 %)
Writing at 0x0000c000... (30 %)
Writing at 0x00010000... (38 %)
Writing at 0x00014000... (46 %)
Writing at 0x00018000... (53 %)
Writing at 0x0001c000... (61 %)
Writing at 0x00020000... (69 %)
Writing at 0x00024000... (76 %)
Writing at 0x00028000... (84 %)
Writing at 0x0002c000... (92 %)
Writing at 0x00030000... (100 %)
Wrote 280256 bytes (205447 compressed) at 0x00000000 in 18.2 seconds (effective 122.9 kbit/s)...
Hash of data verified.

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

After the upload is complete (NOT DURING), launch the **Serial monitor** to view the microcontroller output.

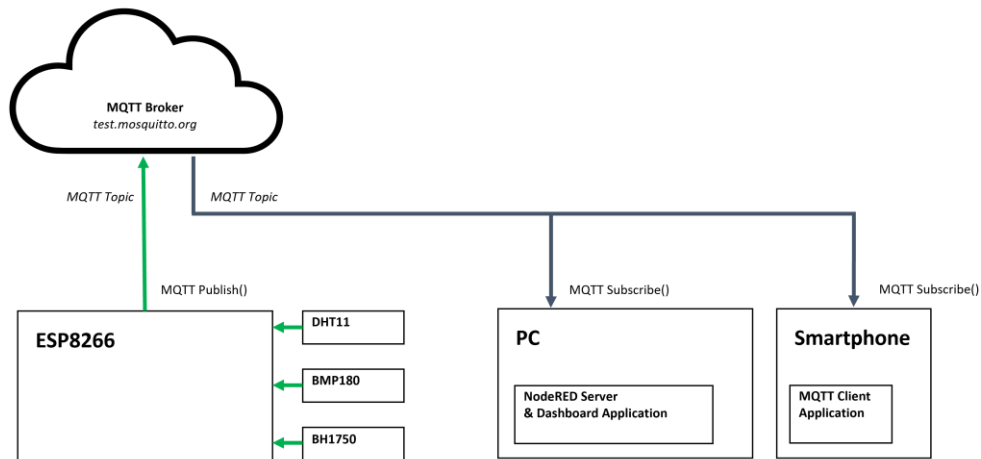
Tools > Serial Monitor



Publishing to an MQTT Broker

The ESP8266 has a built-in WiFi transceiver, which allows it to connect to the internet easily. We will be using the PubSubClient library to:

- Connect to a local WiFi network
- Connect to an MQTT broker
- Connect to a NodeRED interface



Modify your header file to include some **global variables** for the WiFi driver, the **MQTT libraries**, and instantiate **the MQTT Client** object. Also change the delay to 20 seconds (20,000 ms).

When adding the broker ip address, you have two options:

- Use an online public broker
- Use a local broker on the same network

Public Broker

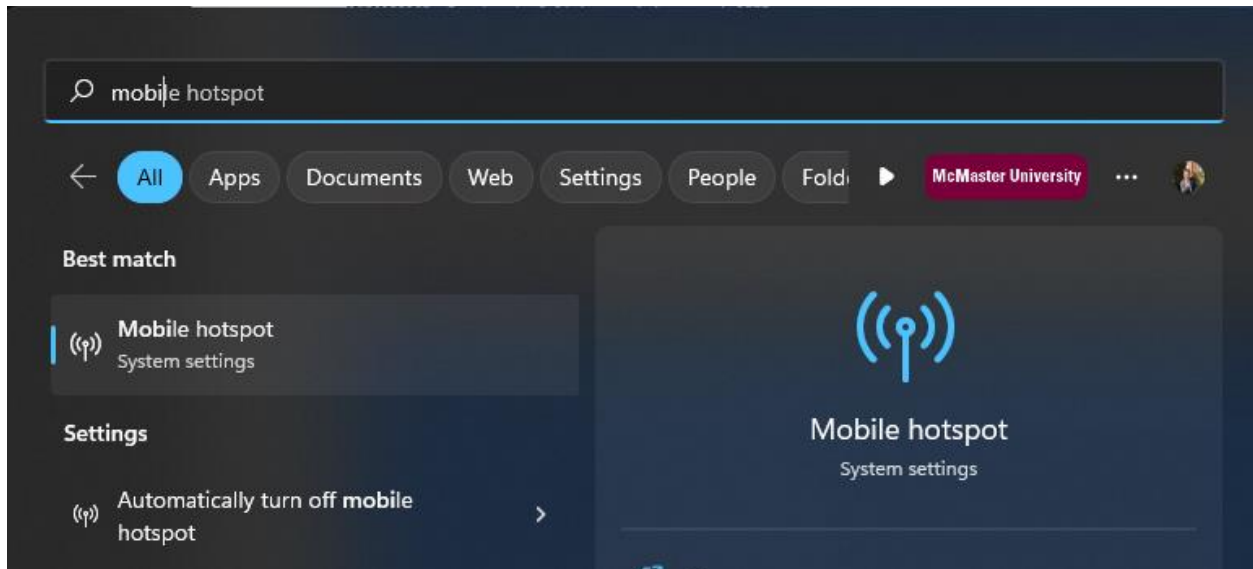
Use the IP address of a known public broker such as Mosquitto's test broker:

test.mosquitto.org

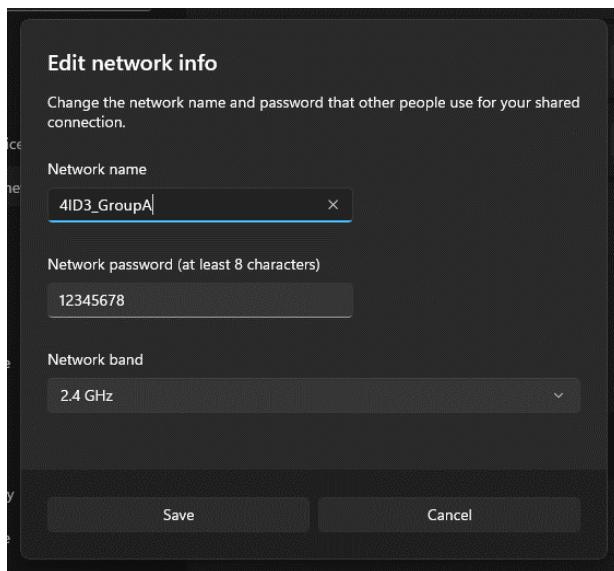
Port is 1883

Hotspotting your Microcontroller

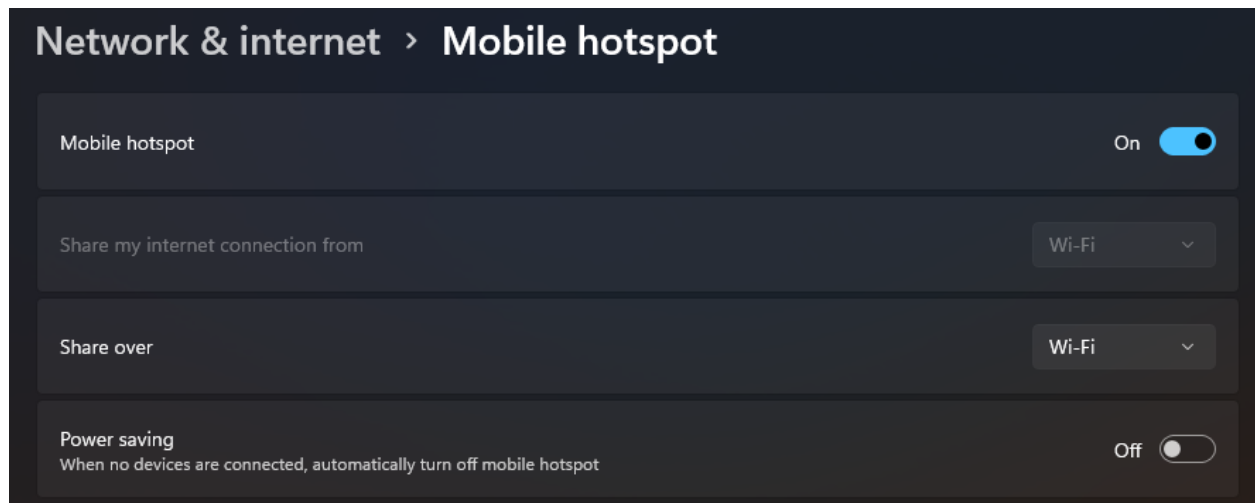
Open **Windows Mobile Hotspot**.



Set your login credentials.

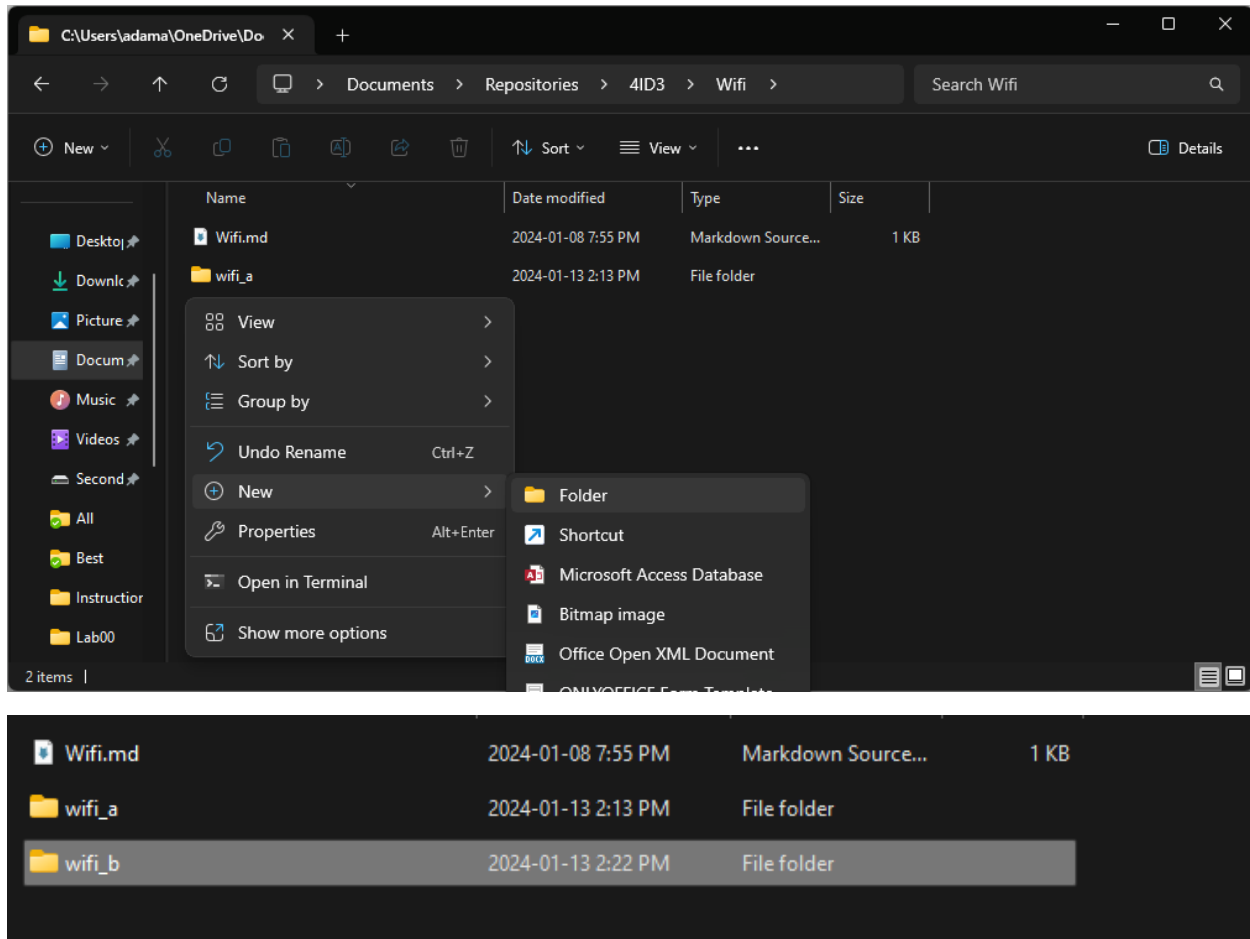


Toggle it **on** and **disable power saving**.



Code Changes

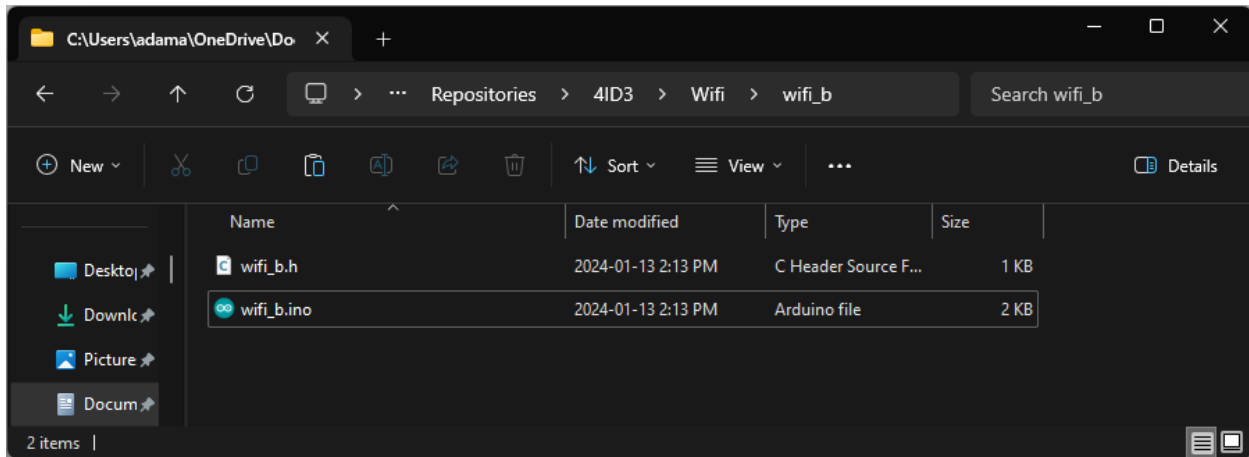
Create a new folder in the lab folder in your course repository called **wifi_b/** .



SaveAs (copy) the existing project as **wifi_b**.

wifi_a.h -> (SaveAs) -> *wifi_b.h*

wifi_a.ino -> (SaveAs) -> *wifi_b.ino*



Open the files in a new Arduino IDE tab.

Select the header file tab.

Above the existing code, include the following communication libraries:

File: *wifi_b.h*

```
//MQTT Libraries
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
```

Below the existing code, include the following global variables:

File: *wifi_b.h*

```
//Instantiate Sensor Objects
DHT_Unified dht(DHTPIN, DHTTYPE);
Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);
hp_BH1750 BH1750;

//Global Variables
char* ssid = "GroupA";
char* pass = "12345678";
const char* brokerAddress = "test.mosquitto.org";
uint16_t addressPort = 1883;
```

Next, instantiate the WifiClient object and PubSubClient object below.

File: wifi_b.h

```
//Instantiate MQTT Client
WiFiClient espClient;
PubSubClient client(espClient);
```

In the implementation file, modify the include statement to include the correct header file.

File: wifi_b.ino

```
#include "wifi_b.h"
```

Modify the **setup()** function, to initialize the WiFi driver and MQTT client.

File: wifi_b.ino

```
#include "wifi_b.h"

void setup() {

    [...]

    //Start the WiFi driver and tell it to connect to your local network
    //WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, pass);

    //While it is connecting, print a '.' to the serial monitor every 500 ms
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    //Once connected, print the local IP address
    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());

    //Set the MQTT client to connect to the desired broker
    client.setServer(brokerAddress, addressPort);
}
```

If the microcontroller loses connection to the MQTT server, we want to have a callback function that would attempt to reconnect.

Below the setup() function, create a reconnect function. This function will be called from void loop().

File: wifi_b.ino

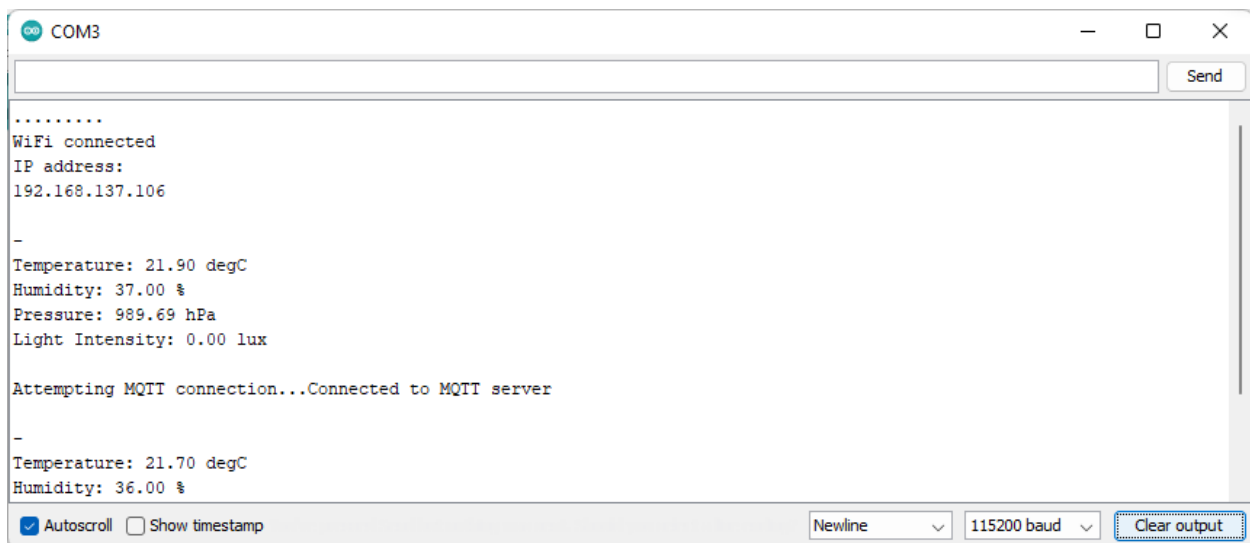
```
void reconnect() {  
  
    //While the client remains unconnected from the MQTT broker, attempt to reconnect every 2  
seconds  
    //Also, print diagnostic information  
    while (!client.connected()) {  
        Serial.print("\nAttempting MQTT connection...");  
  
        if (client.connect("ESP8266Client")) {  
            Serial.println("Connected to MQTT server");  
            client.subscribe("testTopic");  
        } else {  
            Serial.print("\nFailed to connect to MQTT server, rc = ");  
            Serial.print(client.state());  
            delay(2000);  
        }  
    }  
}
```

In void loop(), if connection is lost then call the reconnect function. Once you are connected, publish each sensor data to its associated topic.

File: wifi_b.ino

```
void loop() {  
  
    [...]  
  
    if(!client.loop())  
        client.connect("ESP8266Client");  
  
    //Publish the sensor data to the associated topics  
    client.publish("4ID3_GroupA/temperature", String(dhtTempEvent.temperature).c_str());  
    delay(100);  
    client.publish("4ID3_GroupA/humidity", String(dhtHumEvent.relative_humidity).c_str());  
    delay(100);  
    client.publish("4ID3_GroupA/pressure", String(bmpEvent.pressure).c_str());  
    delay(100);  
    client.publish("4ID3_GroupA/light", String(lux).c_str());  
    Serial.println("Published data.");  
  
    delay(DELAY_BETWEEN_SAMPLES_MS);  
}
```

Ensure the code compiles and reupload the modified code to the microcontroller.



The screenshot shows the Arduino IDE serial monitor window for COM3. The output text is as follows:

```
.....  
WiFi connected  
IP address:  
192.168.137.106  
  
-  
Temperature: 21.90 degC  
Humidity: 37.00 %  
Pressure: 989.69 hPa  
Light Intensity: 0.00 lux  
  
Attempting MQTT connection...Connected to MQTT server  
  
-  
Temperature: 21.70 degC  
Humidity: 36.00 %
```

At the bottom of the window, there are controls for 'Autoscroll' (checked), 'Show timestamp' (unchecked), a 'Newline' dropdown menu, a '115200 baud' dropdown menu, and a 'Clear output' button.

Verifying Connection

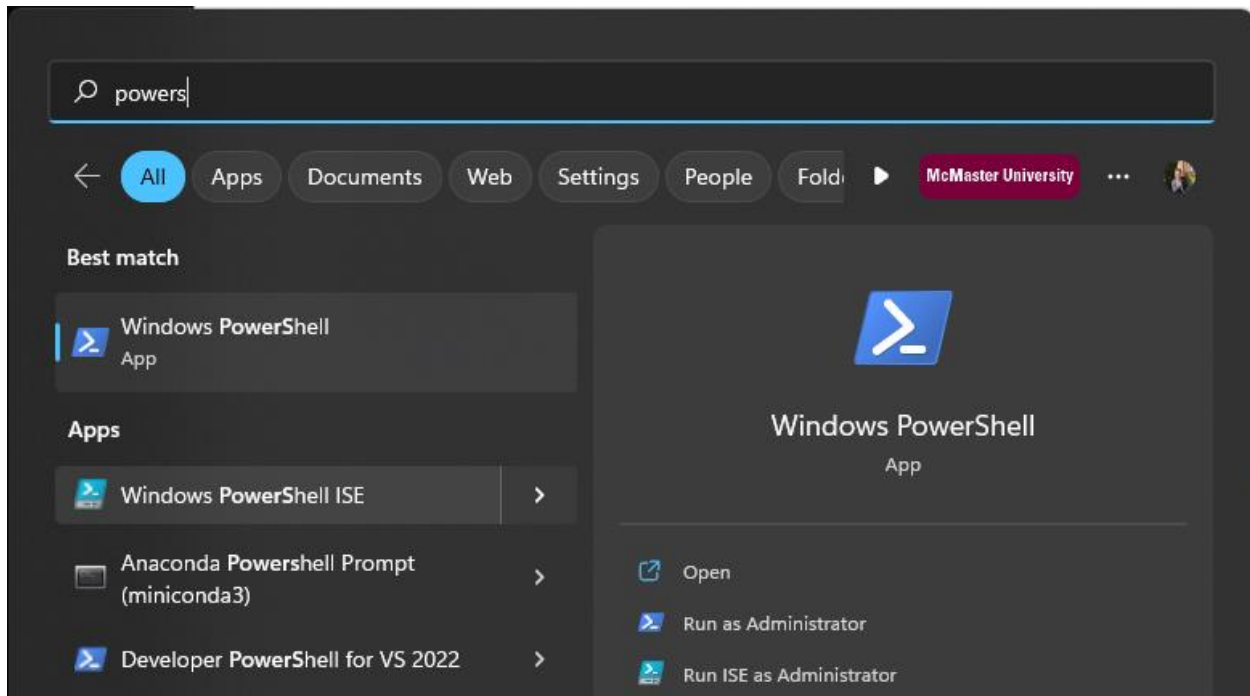
To verify and demonstrate the communication of sensor data over the internet, we will be connecting to the public mqtt broker from a variety of client applications.

Please try each method on a different device.

Using a Desktop Terminal Application

Open a Powershell terminal.

Windows Terminal > Powershell or Windows Powershell



Navigate to the installation directory of the mosquitto mqtt client.

```
cd 'C:\Program Files\mosquitto\'
```

```
dir | findstr("mosquitto")
```

```
Windows PowerShell
PS C:\Users\adama> cd 'C:\Program Files\mosquitto\'
PS C:\Program Files\mosquitto> dir | findstr("mosquitto")
Directory: C:\Program Files\mosquitto
-a----      2022-08-16   9:34 AM           40449 mosquitto.conf
-a----      2022-08-16   9:35 AM           87040 mosquitto.dll
-a----      2022-08-16   9:41 AM          382464 mosquitto.exe
-a----      2022-08-16   9:35 AM           18432 mosquitto_topp.dll
-a----      2022-08-16   9:35 AM           76288 mosquitto_ctrl.exe
-a----      2022-08-16   9:37 AM          122880 mosquitto_dynamic_security.dll
-a----      2022-08-16   9:34 AM           22528 mosquitto_passwd.exe
-a----      2022-08-16   9:35 AM           51712 mosquitto_pub.exe
-a----      2022-08-16   9:35 AM           79872 mosquitto_rr.exe
-a----      2022-08-16   9:35 AM           81920 mosquitto_sub.exe
PS C:\Program Files\mosquitto> |
```

Launch the subscribe applications with the following flags:

```
.\mosquitto_sub.exe -h test.mosquitto.org -t "4ID3_GroupA/humidity"
```

-h - MQTT broker IP address

-t - the topic that your wish to connect to

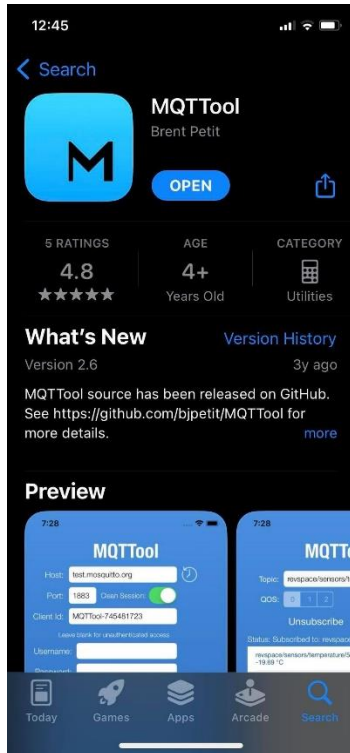
```
Windows PowerShell
PS C:\Program Files\mosquitto> .\mosquitto_sub.exe -h test.mosquitto.org -t "4ID3_GroupA/humidity"
37.00
37.00
38.20
38.30
|
```

After waiting a period of time, you should see the values being printed to the terminal window every 20 seconds.

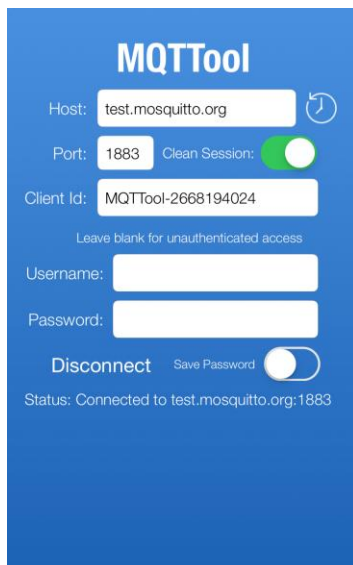
Using a Mobile Client Application

The phone application will likely be delayed from when the terminal application receives the message. If you are using iOS, install the following app:

<https://apps.apple.com/ca/app/mqtttool/id1085976398>



Next, connect to the public mosquitto broker:



Next, subscribe to the topic of choice.

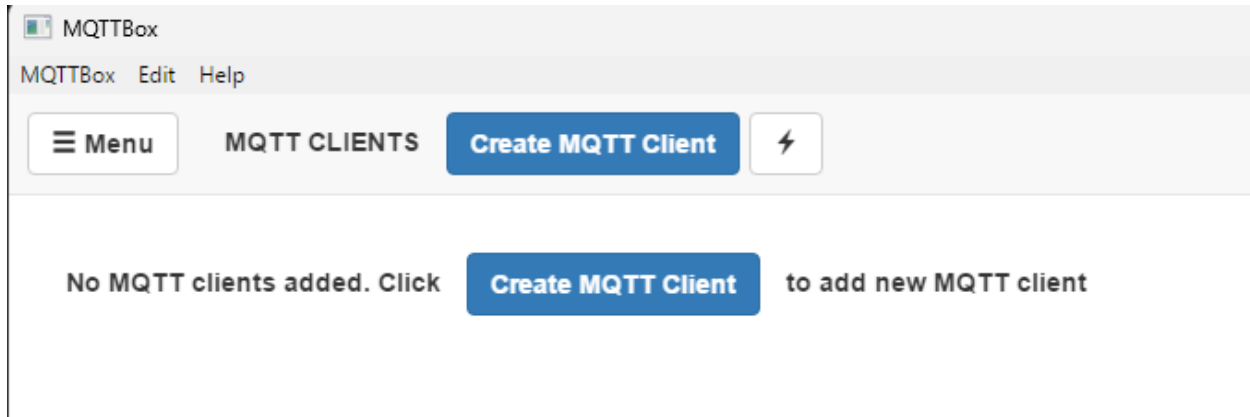
Lastly, wait for values to populate.

You can also try it with other sensor readings to ensure all of them are working correctly.



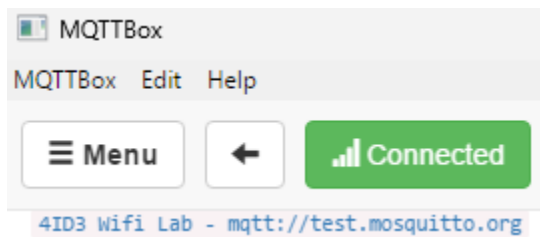
Using a Desktop Client Application

Open **MQTTBox** and select **Create New MQTT Client**.




Configure the following connection details:

Press **Save** and ensure that the toolbar says **Connected**.



Subscribe to your desired topic and watch as messages populate.

 4ID3_GroupA/humidity

38.30

qos : 0, **retain** : false, **cmd** : publish, **dup** : false, **topic** : 4ID3_GroupA/humidity, **messageid** : , **length** : 27, **Raw payload** : 5156465148

38.20

qos : 0, **retain** : false, **cmd** : publish, **dup** : false, **topic** : 4ID3_GroupA/humidity, **messageid** : , **length** : 27, **Raw payload** : 5156465048

37.00

qos : 0, **retain** : false, **cmd** : publish, **dup** : false, **topic** : 4ID3_GroupA/humidity, **messageid** : , **length** : 27, **Raw payload** : 5155464848

37.00

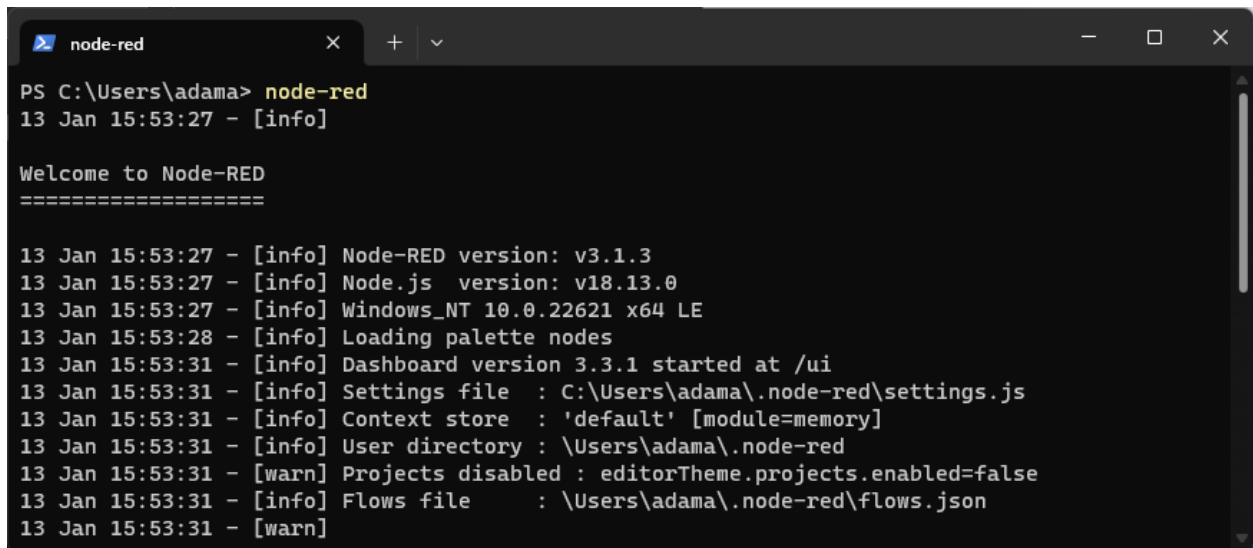
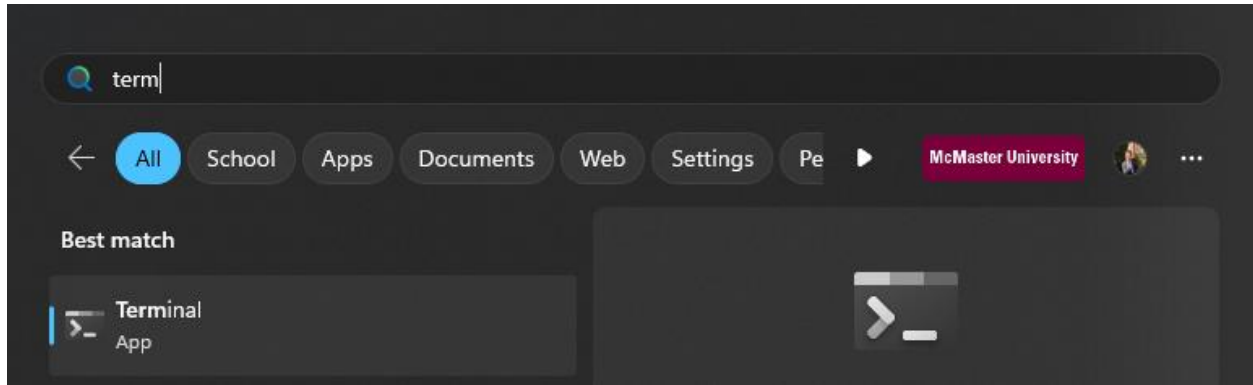
qos : 0, **retain** : false, **cmd** : publish, **dup** : false, **topic** : 4ID3_GroupA/humidity, **messageid** : , **length** : 27, **Raw payload** : 5155464848

NodeRED Dashboard

Please follow the pre-lab instructions to install NodeRED.

To start NodeRED, open Windows Terminal > Powershell and type the command:

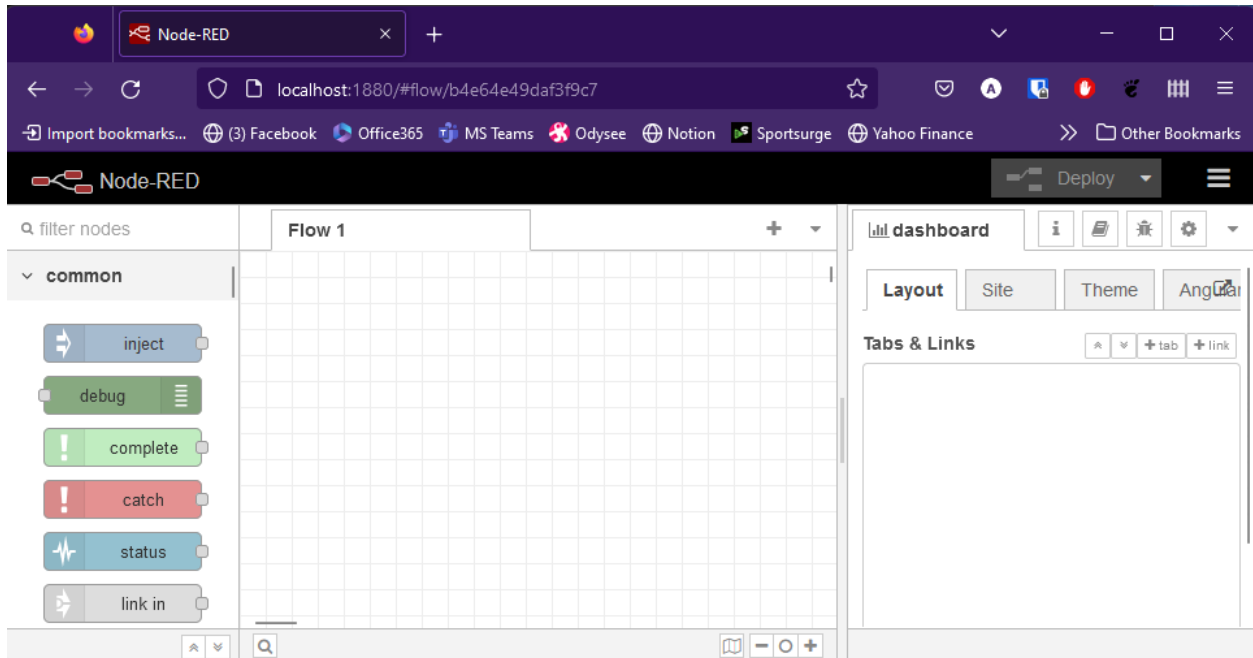
node-red



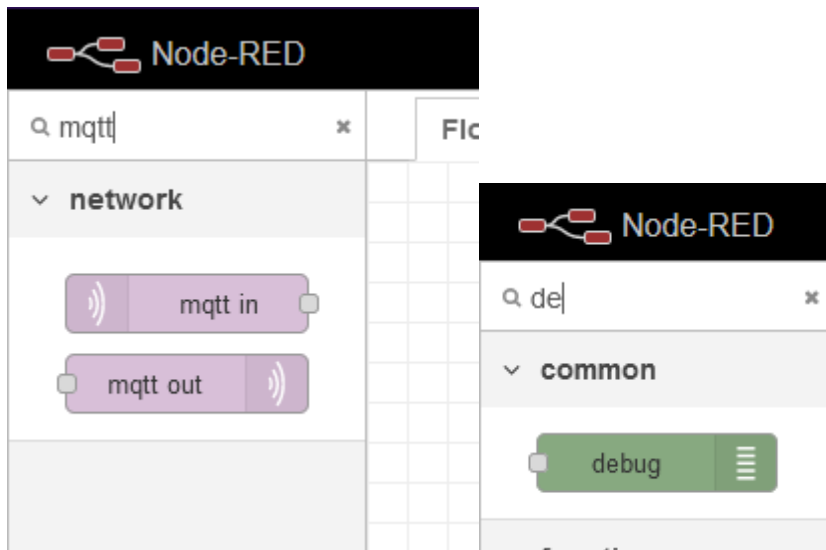
Navigate to the URL presented, in a web browser:

Web Browser > *127.0.0.1:1880*

Communicating Sensor Data over WiFi using MQTT

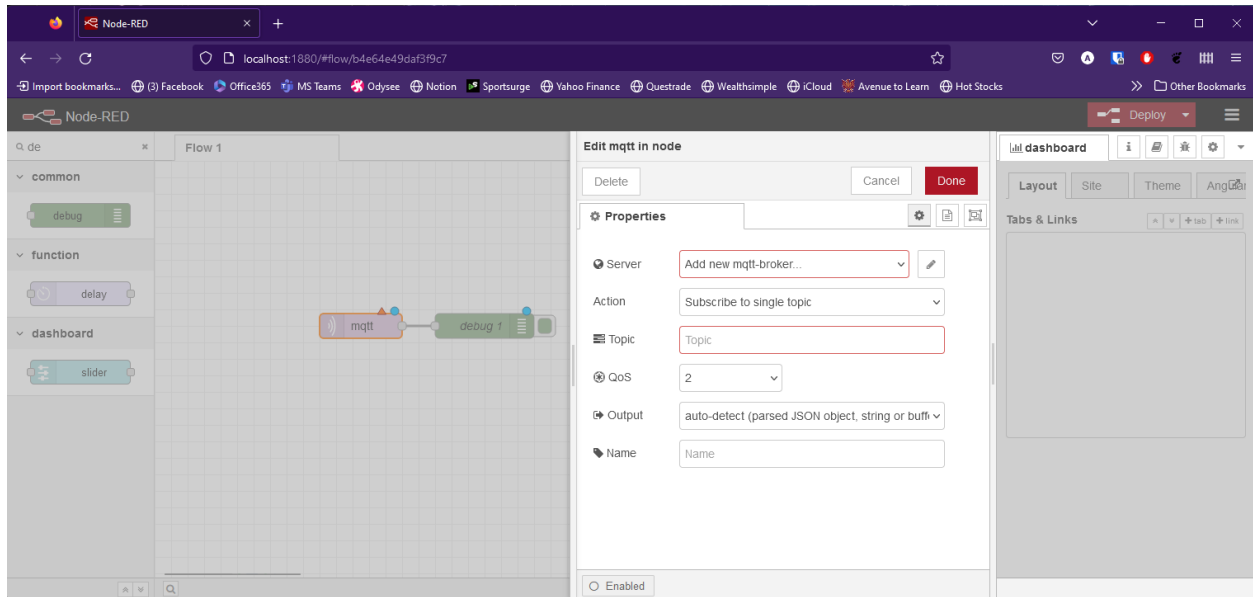


Filter nodes to find the **mqtt in** node and the **debug** node. Drag them into your flow diagram.



Click on the **mqtt in** node to begin editing its configuration.

Communicating Sensor Data over WiFi using MQTT



Fill in the information for the public broker.

Edit mqtt in node > **Edit mqtt-broker node**

Delete Cancel **Update**

Properties

Name Mosquitto Test

Connection Security Messages

Server test.mosquitto.org Port 1883

☒ Connect automatically
☐ Use TLS

Protocol MQTT V3.1.1

Client ID Leave blank for auto generated

Keep Alive 60

Session ☒ Use clean session

Press **Update**.

Under **Properties** fill in the **Topic** field.

Edit mqtt in node

Delete Cancel Done

Properties

Server Mosquitto Test

Action Subscribe to single topic

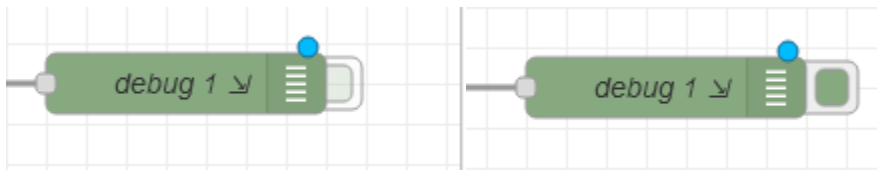
Topic 4ID3_G7/humidity

QoS 0

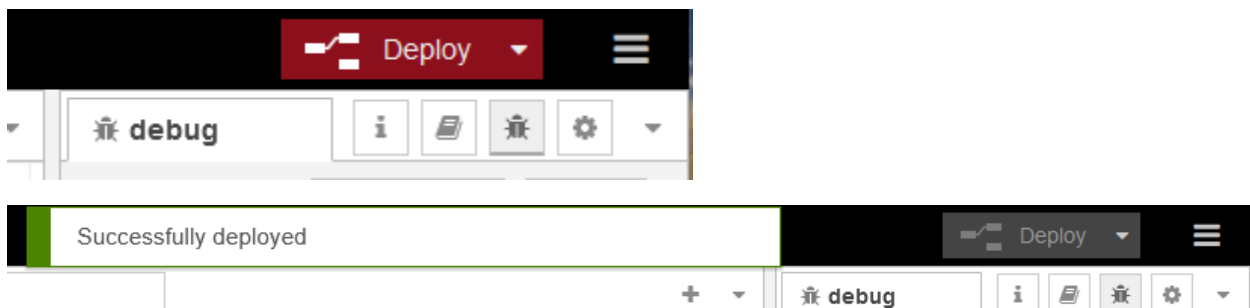
Output auto-detect (parsed JSON object, string or buffer)

Name Name

Enable the **debug** node by pressing the **green box**.



Lastly, press **Deploy** and watch the **Debug Panel** populate with sensor values.



Communicating Sensor Data over WiFi using MQTT

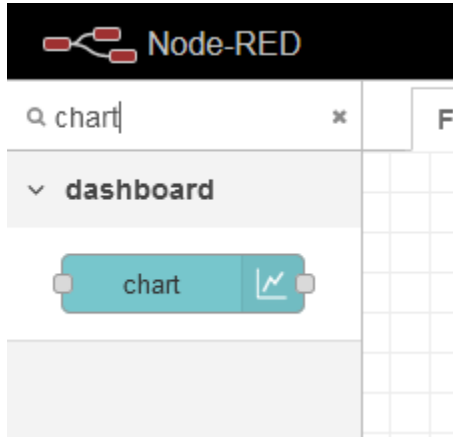
The screenshot displays the Node-RED web interface in a browser window. The address bar shows the URL `localhost:1880/#flow/b4e64e49daf3f9c7`. The interface is divided into three main sections:

- Left Panel (Node Palette):** Contains two categories of nodes:
 - common:** Includes nodes like `inject`, `debug`, `complete`, `catch`, `status`, `link in`, `link call`, `link out`, and `comment`.
 - function:** Includes a `function` node.
- Center Canvas (Flow 1):** Shows a single flow with two nodes connected by a wire:
 - A purple MQTT node labeled `4ID3_G7/humidity` with a green status indicator and the text "connected".
 - A green `debug 1` node.
- Right Panel (Debug Console):** Displays a log of messages. The filter is set to "all nodes". The log shows five entries, each with a timestamp, node name, and message content:
 - 2023-01-17, 1:21:17 p.m. node: debug 1
 - 4ID3_G7/humidity : msg.payload : number
 - 40
 - 2023-01-17, 1:21:23 p.m. node: debug 1
 - 4ID3_G7/humidity : msg.payload : number
 - 40
 - 2023-01-17, 1:21:28 p.m. node: debug 1
 - 4ID3_G7/humidity : msg.payload : number
 - 40
 - 2023-01-17, 1:21:52 p.m. node: debug 1
 - 4ID3_G7/humidity : msg.payload : number
 - 40
 - 2023-01-17, 1:21:57 p.m. node: debug 1
 - 4ID3_G7/humidity : msg.payload : number
 - 40
 - 2023-01-17, 1:22:03 p.m. node: debug 1
 - 4ID3_G7/humidity : msg.payload : number
 - 40

Visualizing NodeRED Data

In the pre-lab, the node-red-dashboard add-on was installed which enables us to create a dashboard of graphs, charts, and toggles to visualize and control data.

In the **filter nodes** field, search for **chart** and drag it into your flow diagram.



When you click on the node, you will need to create a new **Dashboard Tab**. Use the default name and press **Add**.

A screenshot of the 'Add new dashboard tab config node' dialog box in Node-RED. The breadcrumb path at the top reads 'Edit chart node > Add new dashboard group config node > Add new dashboard tab config node'. There are 'Cancel' and 'Add' buttons. Below is a 'Properties' section with four fields: 'Name' (set to 'Home'), 'Icon' (set to 'dashboard'), 'State' (a toggle switch set to 'Enabled'), and 'Nav. Menu' (a toggle switch set to 'Visible'). A yellow tooltip box at the bottom explains the 'Icon' field, stating it can be a Material Design icon, a Font Awesome icon, or a Weather icon, and provides examples like 'mi-videogame_asset'.

Add the dashboard group to that new dashboard by pressing **Add**.

Edit chart node > **Add new dashboard group config node**

Cancel

Add

Properties

Name

Default

Tab

Home

▼

Class

Optional CSS class name(s) for widget

Width

6

☒ Display group name

☐ Allow group to be collapsed

Lastly, edit the chart node to visualize your data nicely. Press **Done** when complete.

Communicating Sensor Data over WiFi using MQTT

Edit chart node

Delete

Cancel

Done

Properties

Group

[Home] Default

Size

auto

Label

Humidity

Type

Line chart

☒ enlarge points

X-axis

last 1 hours

OR

1000 points

X-axis Label

HH:mm:ss

☐ as UTC

Y-axis

min 0

max 100

Legend

None

Interpolate linear

Series Colours

Blank label

display this text before valid data arrives

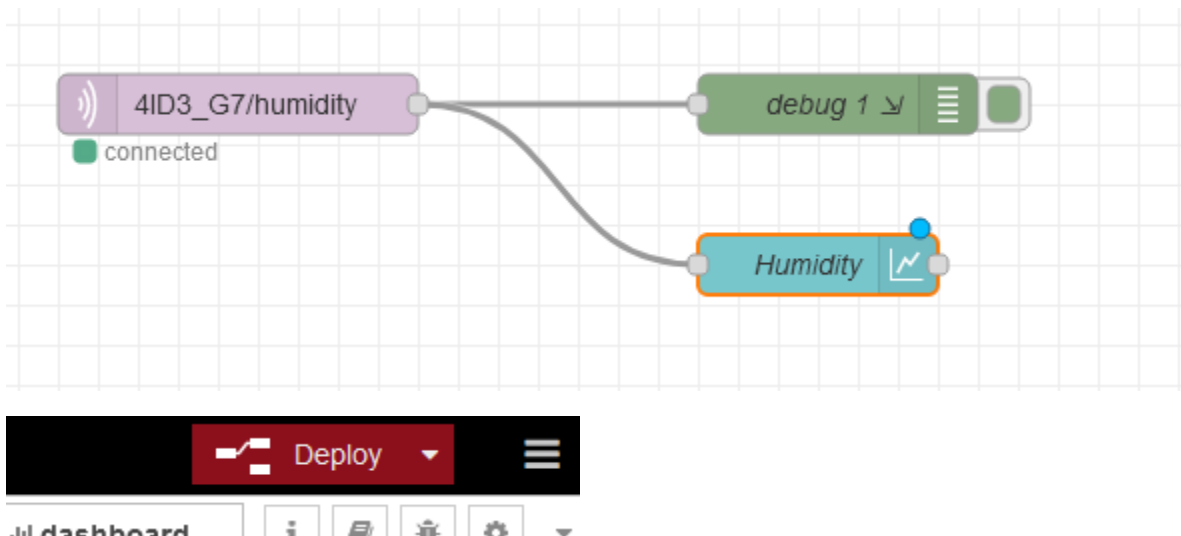
</> Class

Optional CSS class name(s) for widget

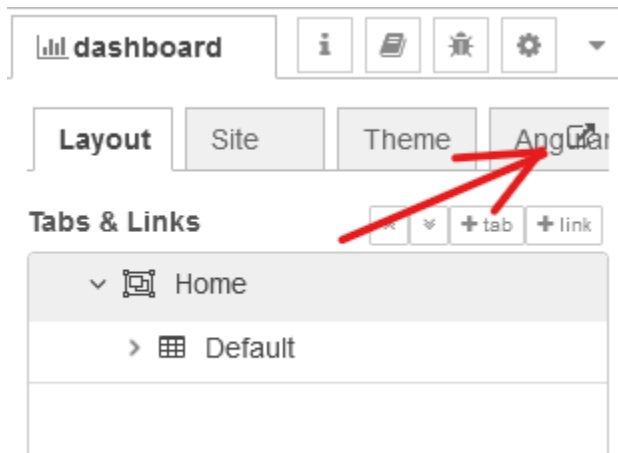
Name

Humidity

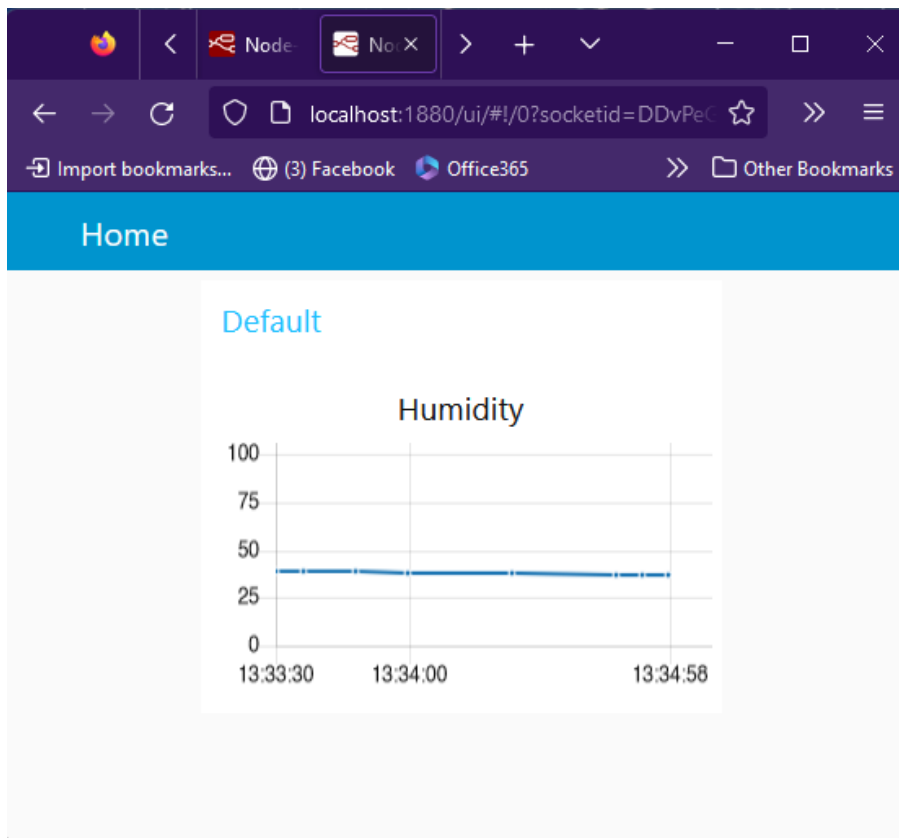
Connect your **mqtt in** node to the input of your **chart** node. Press **Deploy** to save changes.



To view the dashboard, either append **ui/** to your url (<http://localhost:1880/ui>) or press the **open dashboard** icon in the top right corner of the dashboard panel.



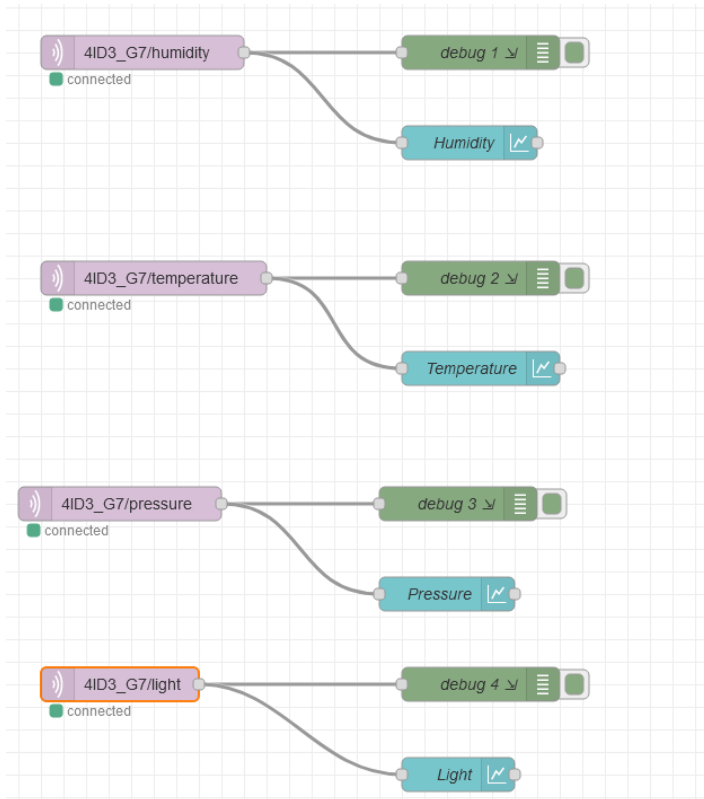
Wait for data to populate.

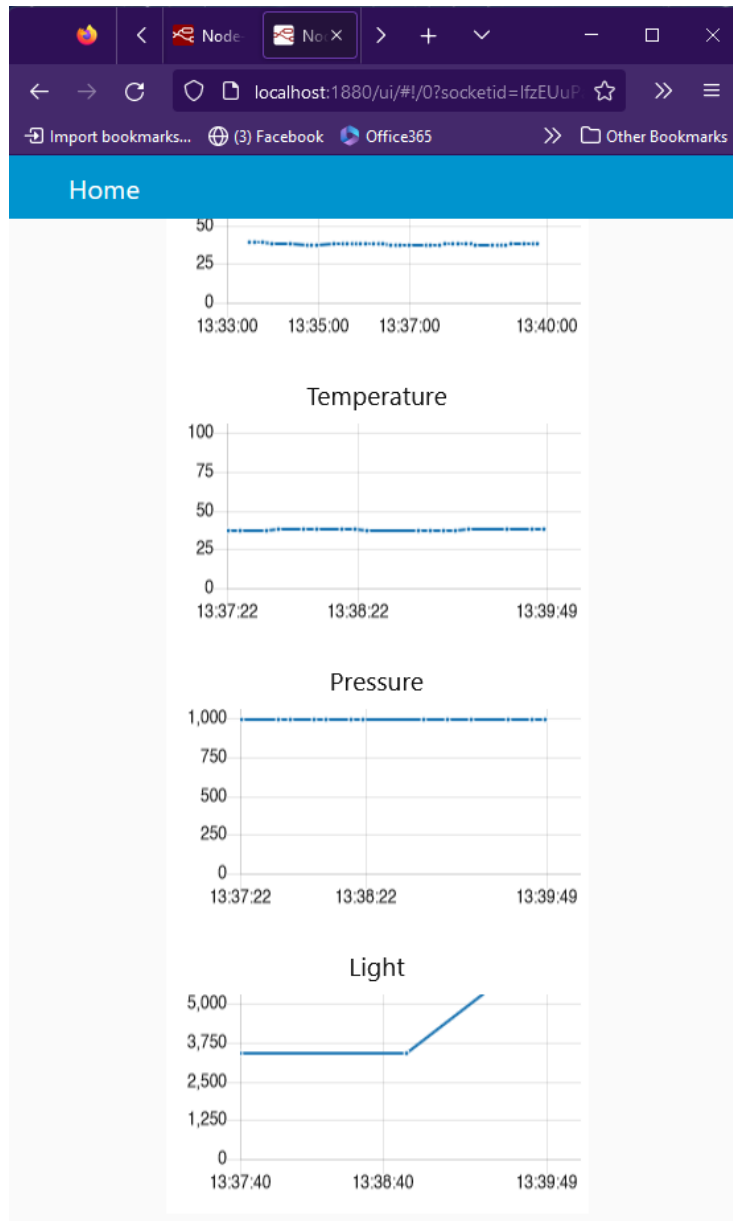


Now that you have seen how to visualize one topic, attempt to visualize the rest.

The resultant flow should look like this:

Communicating Sensor Data over WiFi using MQTT





If values are not showing up, ensure that your y-axis scale is correct.

Saving and Pushing Your Project

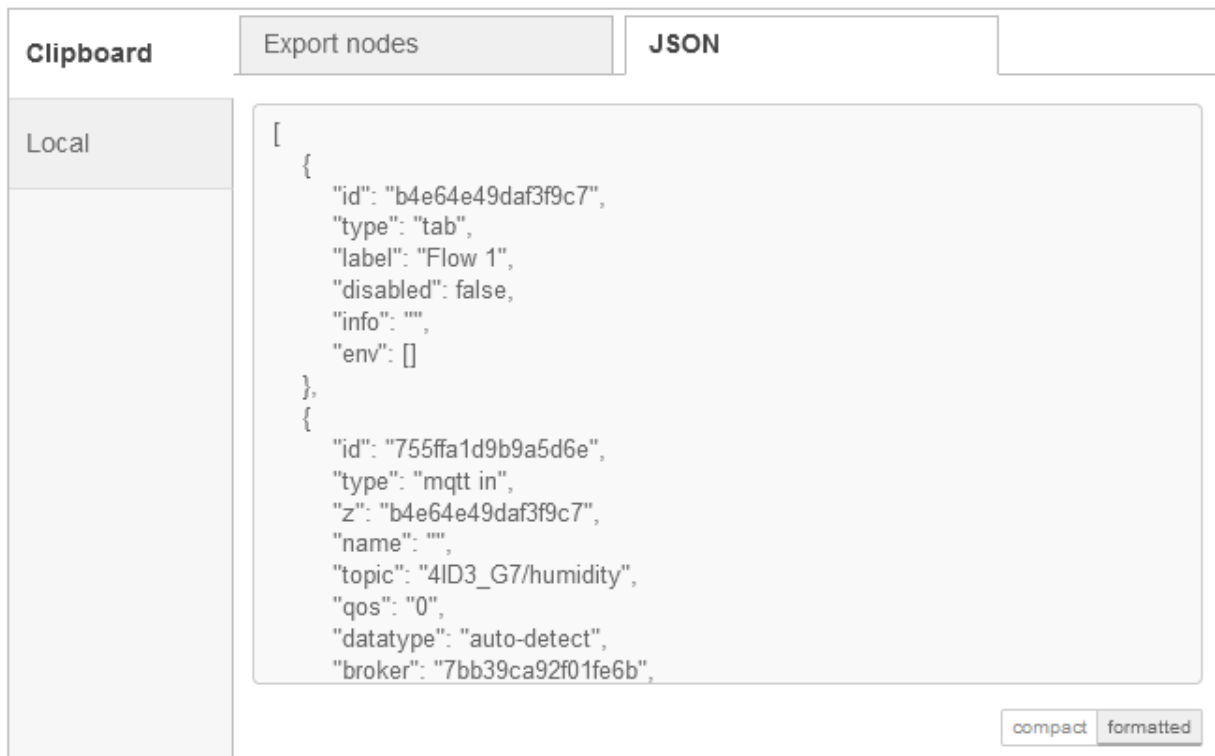
Exporting a NodeRED Flow as JSON

Click on the **hamburger menu** and select **export**.

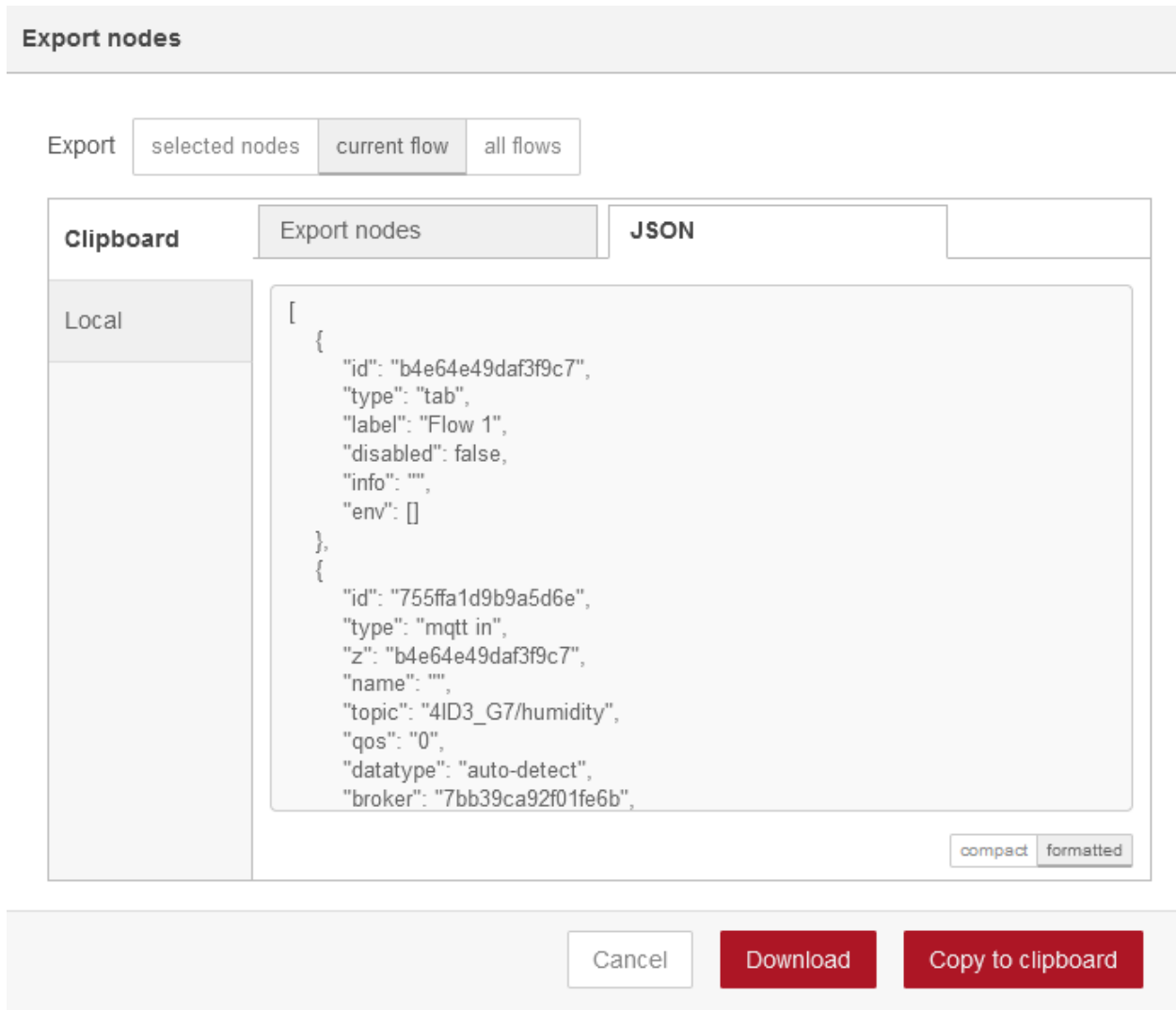
Select **current flow**.



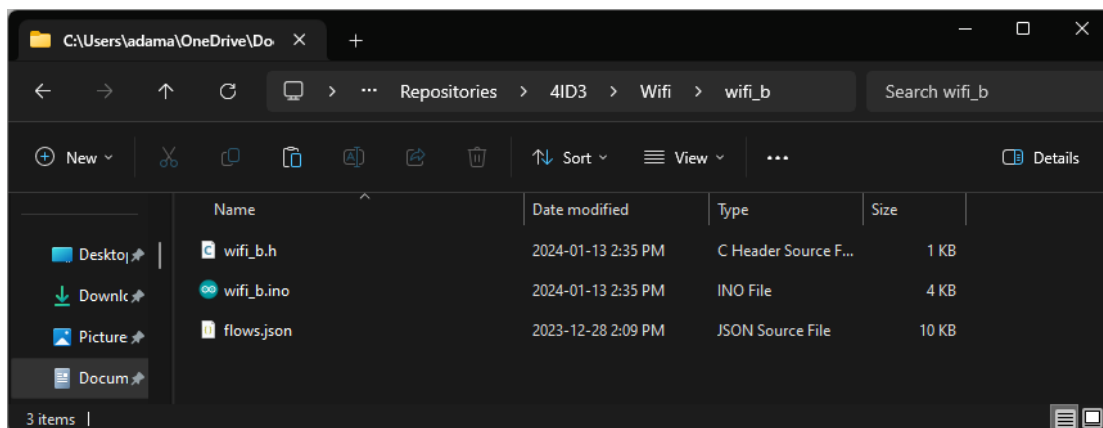
Select **JSON**.



Press **Download**.

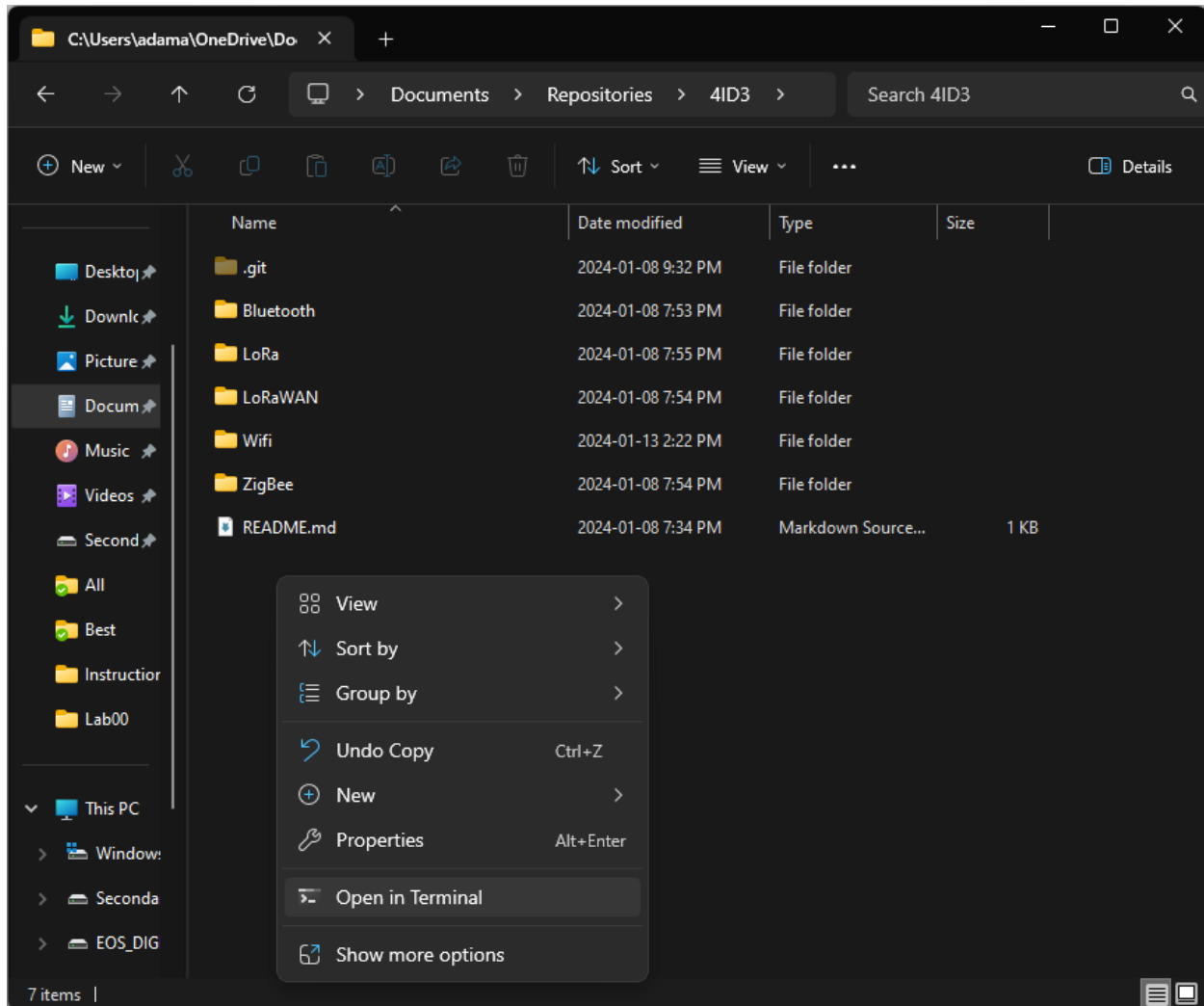


Cut and paste the **flows.json** file from your **Downloads/** folder to the **Wifi/** folder in the **local repo**.



Pushing Changes to GitHub

Right click on the root directory, *Repositories/4ID3/*, of your lab repository and select **Open with Terminal** from the context menu.



Use the diff command to compare the changes in your repo to the last commit in your **main** branch.

```
git diff main
```

```
:q
```

```
Windows PowerShell
PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3> git diff main
diff --git a/Wifi/wifi_a/wifi_a.h b/Wifi/wifi_a/wifi_a.h
deleted file mode 100644
index 2f31b31..0000000
--- a/Wifi/wifi_a/wifi_a.h
+++ /dev/null
@@ -1,22 +0,0 @@
-//DHT11 Libraries
-#include <Adafruit_Sensor.h>
-#include <DHT.h>
-#include <DHT_U.h>
-
-//BMP180 Libraries
-#include <Wire.h>
-#include <Adafruit_Sensor.h>
-#include <Adafruit_BMP085_U.h>
-
-//HP_BH1750 Libraries
-#include <hp_BH1750.h>
-
-//Macros
-#define DHTPIN 14
-#define DHTTYPE DHT11
-#define DELAY_BETWEEN_SAMPLES_MS 5000
-
-//Instantiate Sensor Objects
-DHT_Unified dht(DHTPIN, DHTTYPE);
-Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);
-hp_BH1750 BH1750;
diff --git a/Wifi/wifi_a/wifi_a.ino b/Wifi/wifi_a/wifi_a.ino
```

Add your changes to the staging area.

git add .

```
Windows PowerShell
PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3> git add .
PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3> |
```

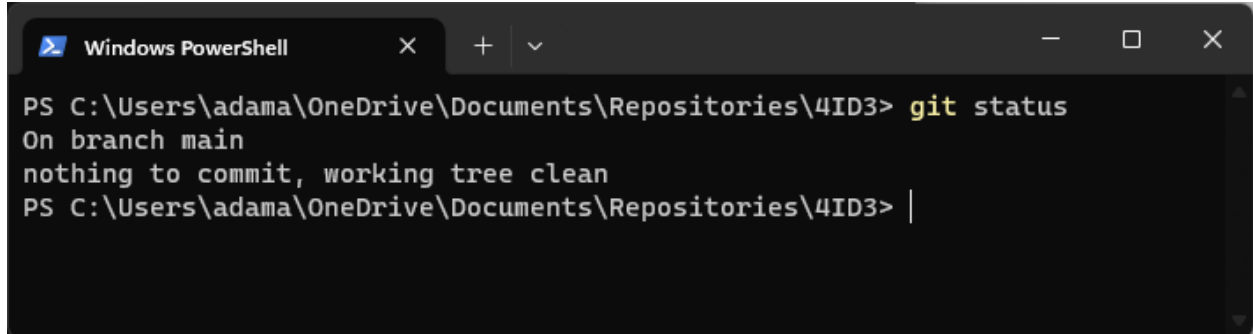
Commit these changes.

git commit -m "wifi lab complete"

```
Windows PowerShell
PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3> git commit -m "wifi lab complete"
[main 8019f43] wifi lab complete
5 files changed, 613 insertions(+)
create mode 100644 Wifi/wifi_a/wifi_a.h
create mode 100644 Wifi/wifi_a/wifi_a.ino
create mode 100644 Wifi/wifi_b/flows.json
create mode 100644 Wifi/wifi_b/wifi_b.h
create mode 100644 Wifi/wifi_b/wifi_b.ino
PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3> |
```

Verify that all your changes have been committed.

git status

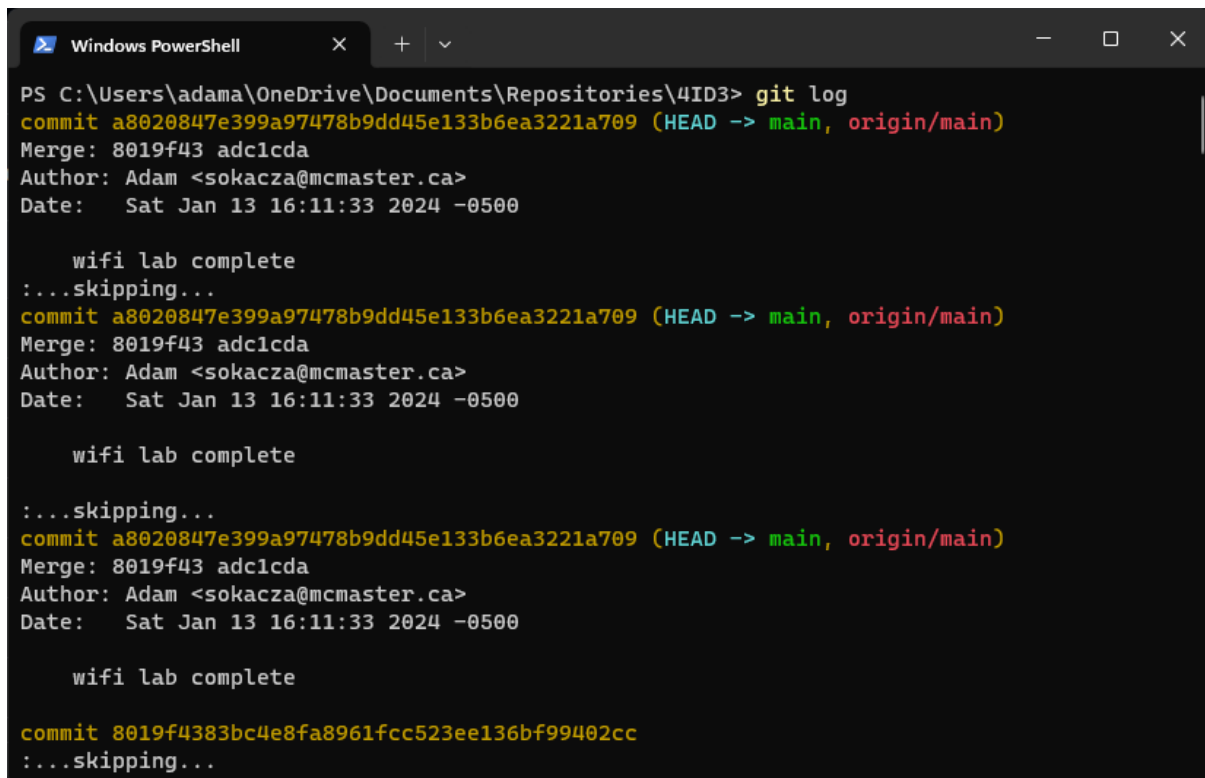


```
Windows PowerShell
PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3> git status
On branch main
nothing to commit, working tree clean
PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3> |
```

View a list of all your commits.

git log

:q



```
Windows PowerShell
PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3> git log
commit a8020847e399a97478b9dd45e133b6ea3221a709 (HEAD -> main, origin/main)
Merge: 8019f43 adclcd
Author: Adam <sokacza@mcmaster.ca>
Date: Sat Jan 13 16:11:33 2024 -0500

    wifi lab complete
...skipping...
commit a8020847e399a97478b9dd45e133b6ea3221a709 (HEAD -> main, origin/main)
Merge: 8019f43 adclcd
Author: Adam <sokacza@mcmaster.ca>
Date: Sat Jan 13 16:11:33 2024 -0500

    wifi lab complete
...skipping...
commit a8020847e399a97478b9dd45e133b6ea3221a709 (HEAD -> main, origin/main)
Merge: 8019f43 adclcd
Author: Adam <sokacza@mcmaster.ca>
Date: Sat Jan 13 16:11:33 2024 -0500

    wifi lab complete
...skipping...
commit 8019f4383bc4e8fa8961fcc523ee136bf99402cc
...skipping...
```

Push your changes to GitHub.

git push origin main

```
Windows PowerShell
PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3> git push origin main
Enumerating objects: 29, done.
Counting objects: 100% (29/29), done.
Delta compression using up to 12 threads
Compressing objects: 100% (19/19), done.
Writing objects: 100% (26/26), 4.54 KiB | 1.13 MiB/s, done.
Total 26 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), completed with 1 local object.
To github.com:sokacza/4ID3_Labs.git
   adc1cda..a802084  main -> main
PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3> |
```

Use a web browser to confirm that GitHub reflects these changes.

4ID3_Labs / Wifi / Add file ...

 sokacza

wifi lab complete

8019f43 · 15 minutes ago History

Name	Last commit message	Last commit date
..		
wifi_a	wifi lab complete	15 minutes ago
wifi_b	wifi lab complete	15 minutes ago
Wifi.md	lab folders	5 days ago

END