

4ID3 - IoT Devices and Networks

Lab 1

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 |
| Setting up the Workspace | 6 |
| Wiring Diagram | 12 |
| Reading Sensor Data | 14 |
| Publishing to an MQTT Broker | 23 |
| Public Broker | 23 |
| Hotspotting your Microcontroller | 23 |
| Code Changes | 26 |
| Verifying Connection | 30 |
| Mosquitto Terminal Application | 30 |
| Using a Mobile Application | 32 |
| NodeRED Dashboard | 35 |
| Visualizing NodeRED Data | 40 |
| Saving and Pushing Your Project | 46 |
| Exporting a NodeRED Flow as JSON | 46 |
| Committing and Pushing Changes to GitHub | 48 |

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

Arduino Programming Refresher (https://youtu.be/CbJHL_P5RJ8)

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

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

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

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 – What is the role of each of the **fields below**, when connecting to an MQTT broker?

| | |
|---------------------------|--|
| Host IP Address | |
| Topic Name | |
| Username | |
| Password | |
| 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** in Node-Red user interface as you interact with the sensors. Is the change instantaneous?

(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 **analogue**, **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)

Lab 1 - Communicating Sensor Data over WiFi using MQTT

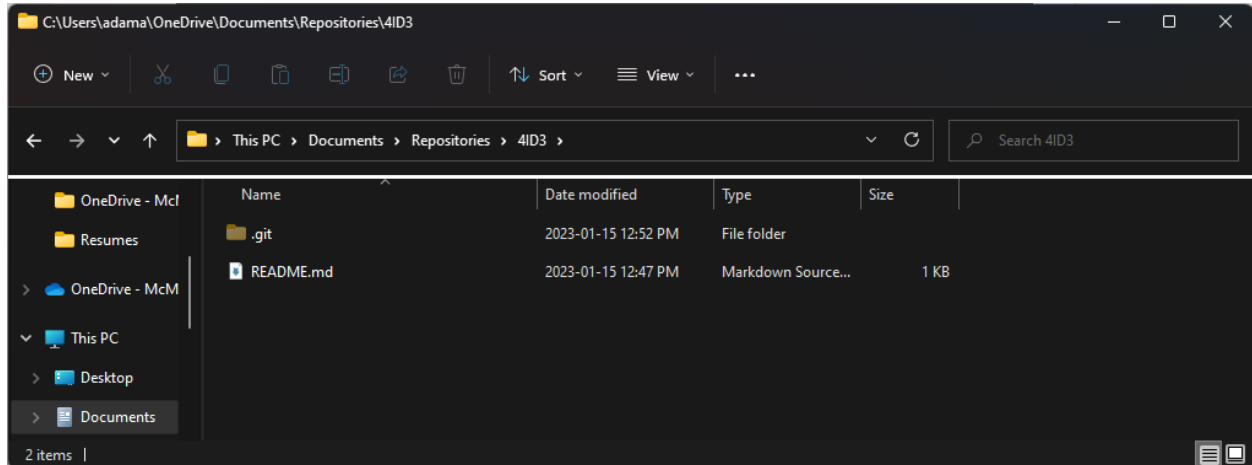
Q6 - Below, write a **LinkedIn post** about **4 key learning takeaways** from this lab.

(Suggested: Paragraph or screenshot, 2 points)

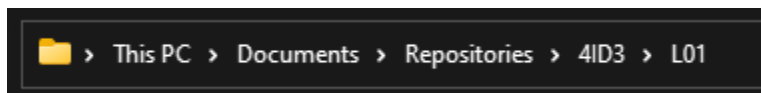
Setting up the Workspace

Each lab, we will be creating a new folder in the local git repository that was created in the provided pre-lab to store and document technologies that you have worked on.

Navigate to your local git repository for this course.

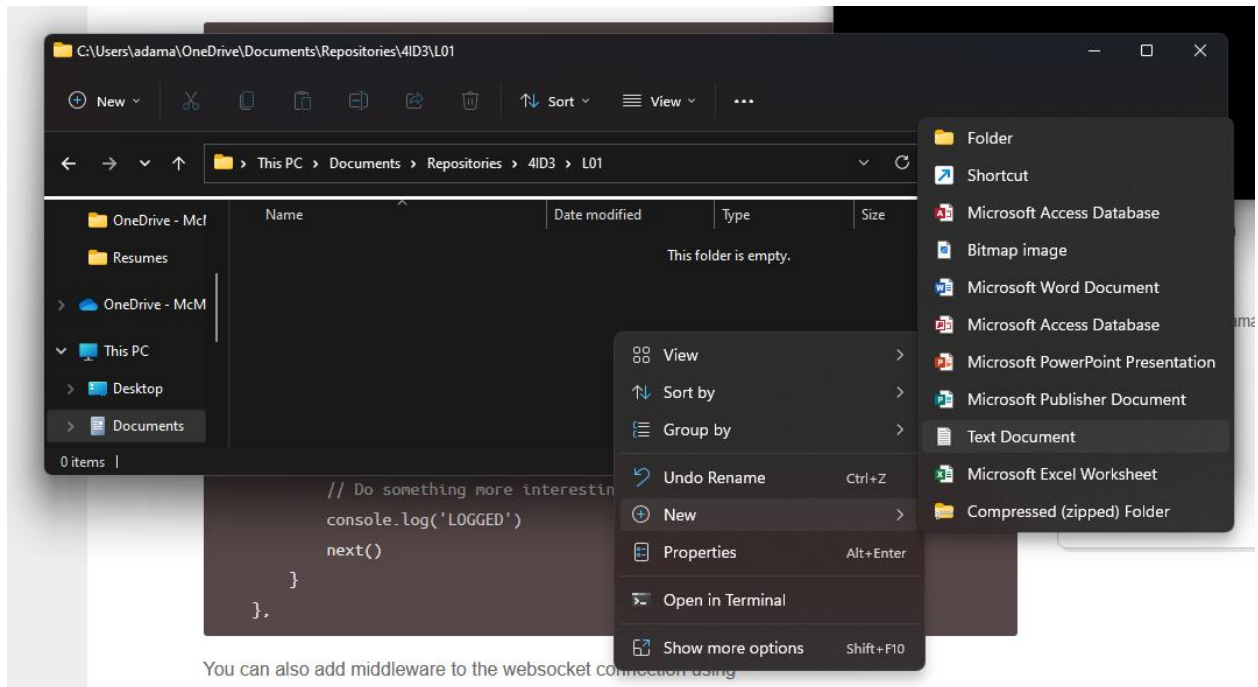


Create a new folder named **L01**. Navigate inside this folder.

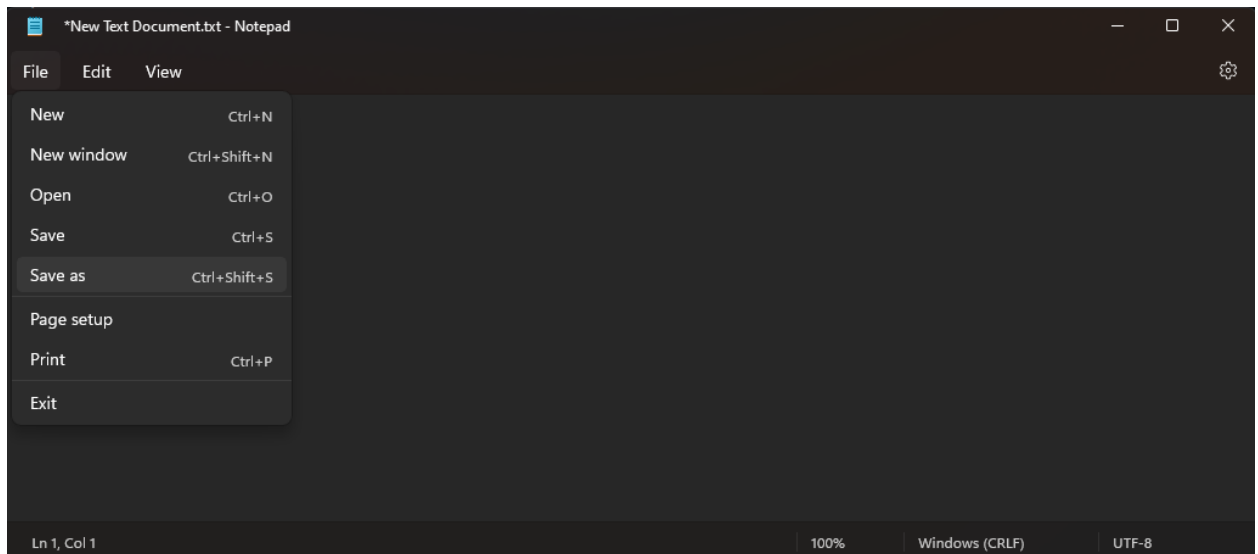


Create a new text file in the folder.

Lab 1 - Communicating Sensor Data over WiFi using MQTT

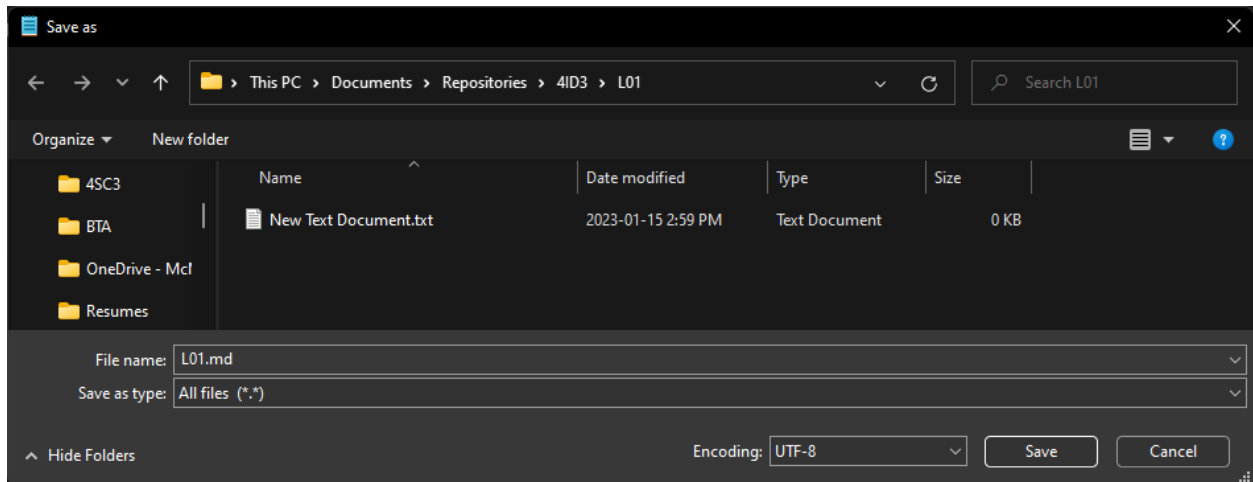


Press **File > Save as**.

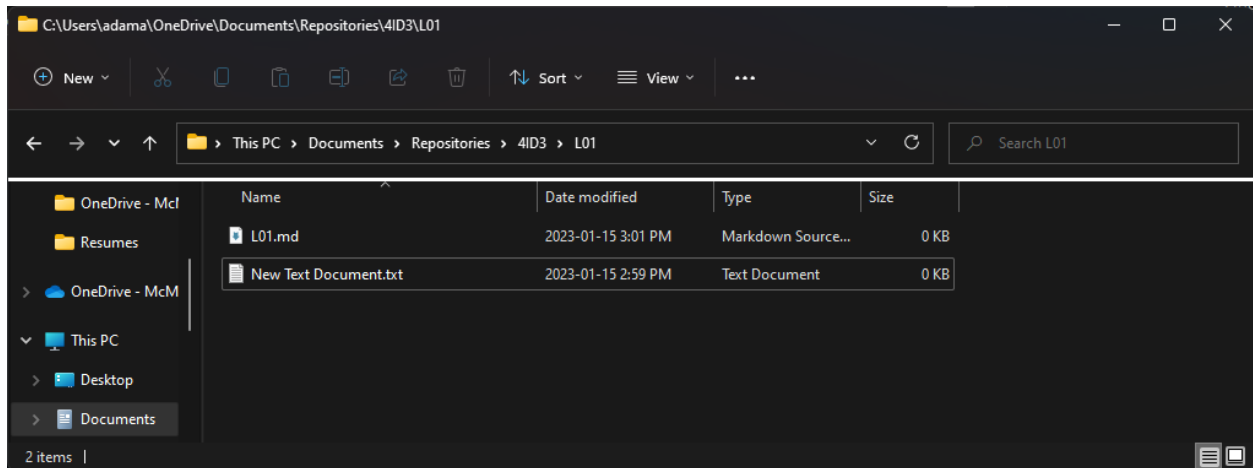


Save it as **L01.md**. Ensure that the **Save as type** is set to **All files (*.*)**.

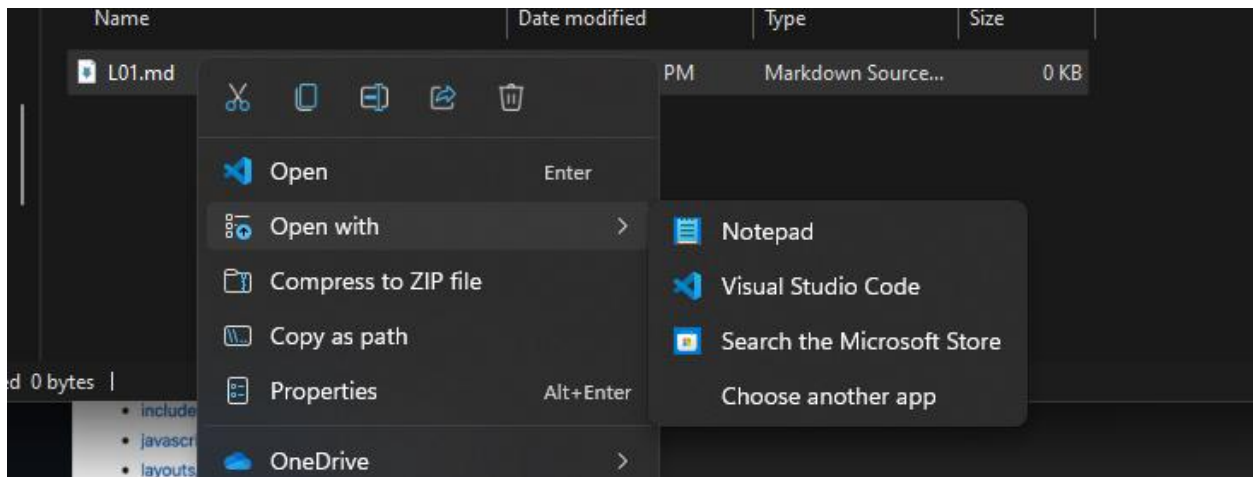
Lab 1 - Communicating Sensor Data over WiFi using MQTT



Now, you should have two files, a **text file** and a **markdown file**. Delete the text file.



To open the markdown file, **right-click** and select **Open with**. Choose **Notepad**.

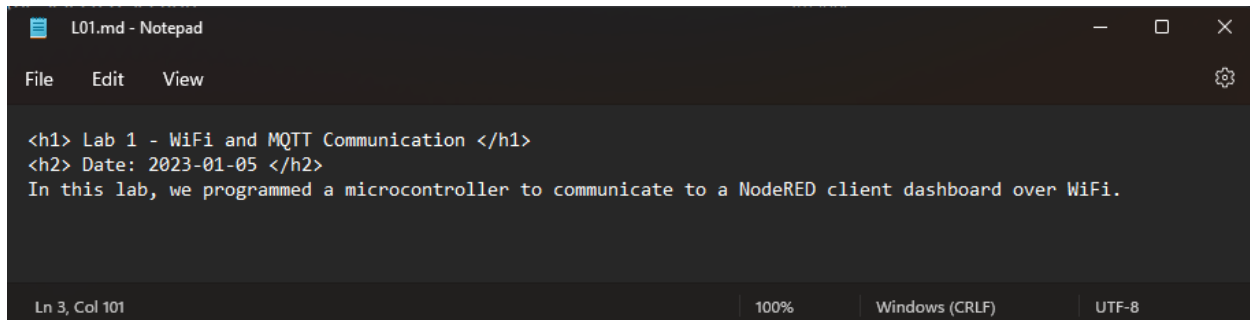


Lab 1 - Communicating Sensor Data over WiFi using MQTT

Writing markdown documents to explain your code is very similar to HTML. A reference guide can be found here:

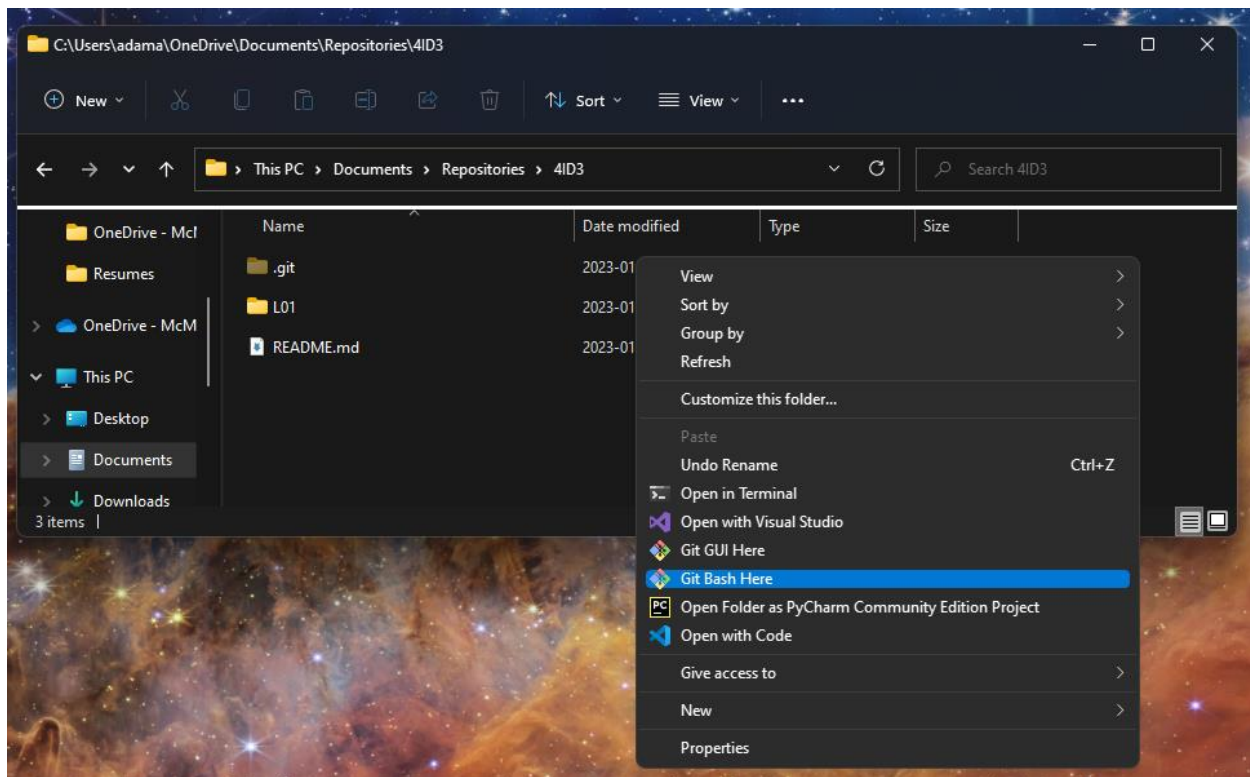
<https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax>

Write the following text in the markdown file and save it.



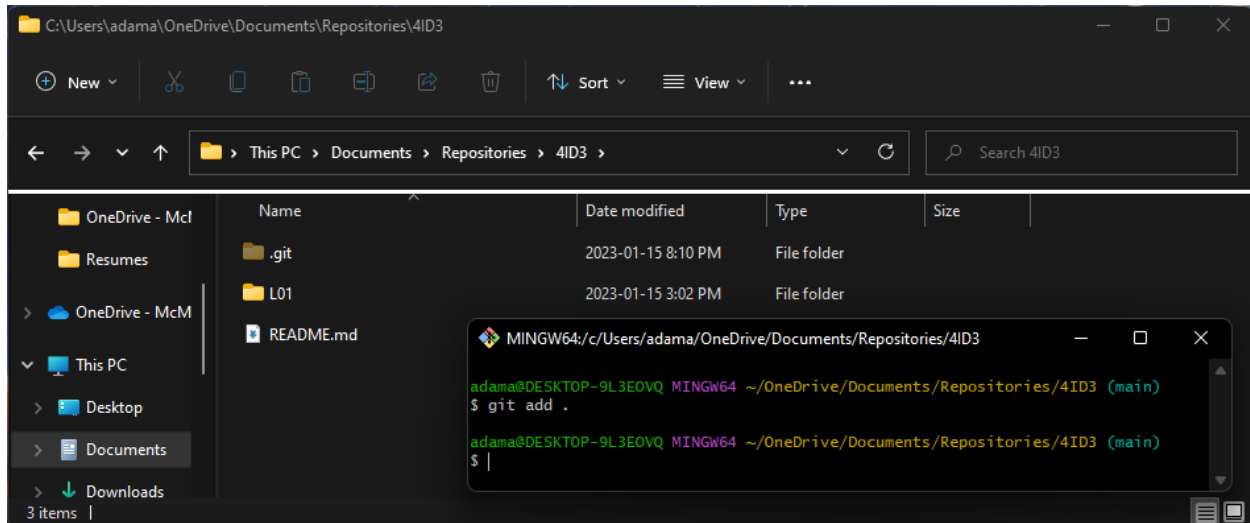
```
<h1> Lab 1 - WiFi and MQTT Communication </h1>
<h2> Date: 2023-01-05 </h2>
In this lab, we programmed a microcontroller to communicate to a NodeRED client dashboard over WiFi.
```

Right-click in the root of your local repository and launch **git bash**.



Lab 1 - Communicating Sensor Data over WiFi using MQTT

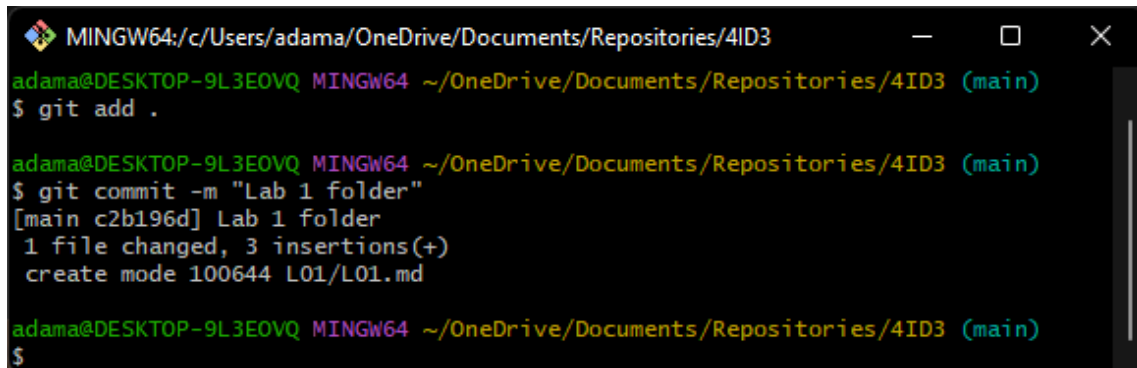
First, we need to add all the changes to the index that will be synced with GitHub. This will be done with the `git add` command.



`git add .`

The period `'.'` is used as a shorthand for selecting all changes.

Next, when we are happy with the changes we chose to upload, we can use the commit command to package them to be synced.



`git commit -m "Lab 1 folder"`

The `'-m'` flag stands for message, and it adds a message that explains what changes were made.

Lastly, to sync your local git repository with GitHub, use the `git push` command.

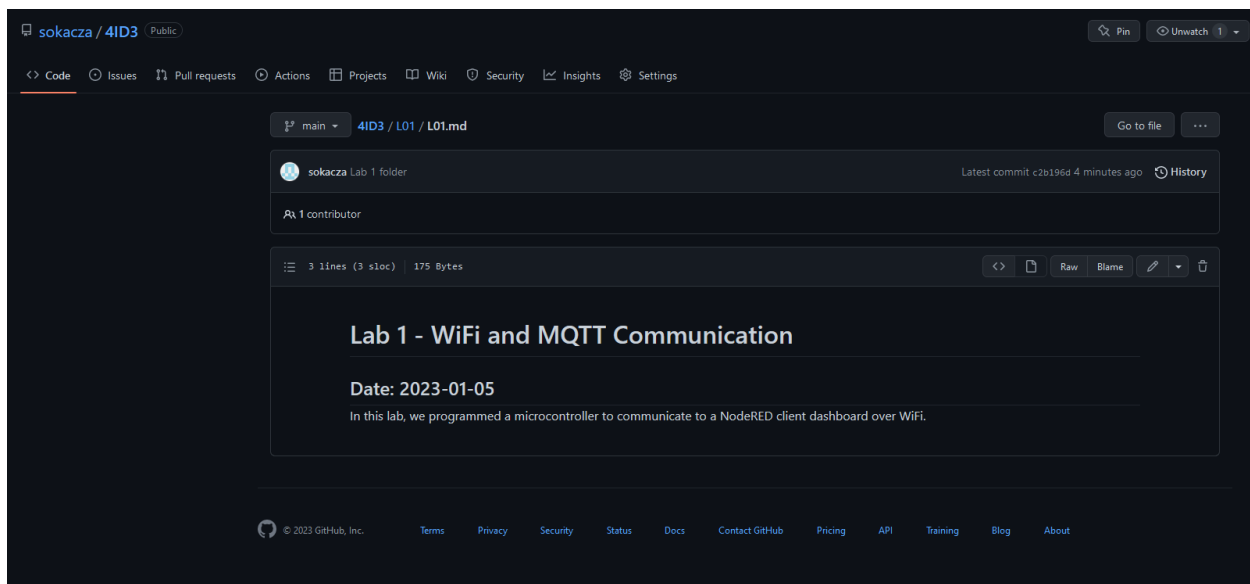
Lab 1 - Communicating Sensor Data over WiFi using MQTT

```
MINGW64:/c/Users/adama/OneDrive/Documents/Repositories/4ID3
adama@DESKTOP-9L3E0VQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 457 bytes | 76.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:sokacza/4ID3.git
   b8c4395..c2b196d  main -> main

adama@DESKTOP-9L3E0VQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)
$ |
```

git push origin main

Now, log into GitHub and verify that the changes have been made.



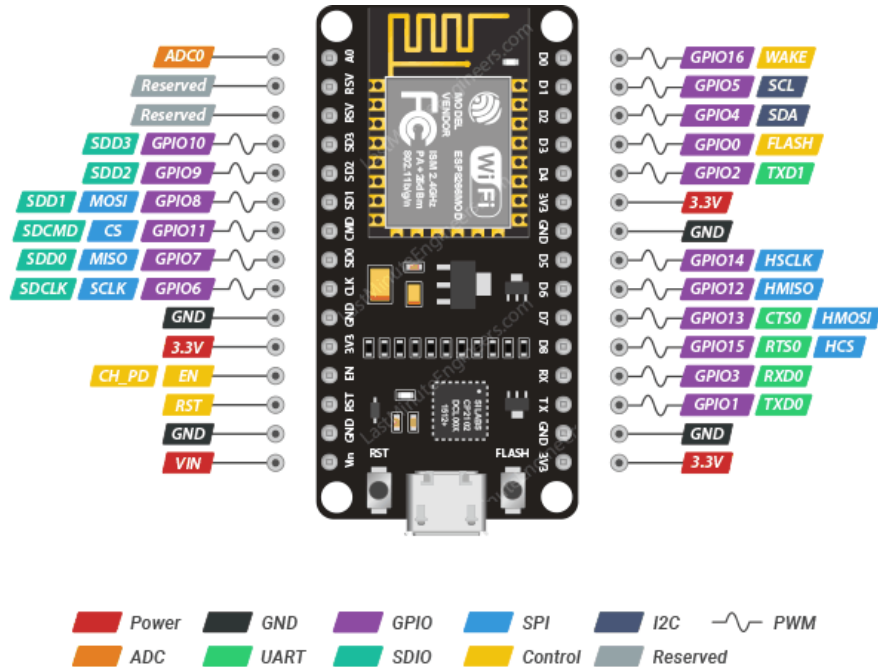
Now, if you are collaborating and wish to sync your local git repo with the remote GitHub repo, use the git pull command. In this case, we see that our local git repo is already up-to-date.

```
MINGW64:/c/Users/adama/OneDrive/Documents/Repositories/4ID3
adama@DESKTOP-9L3E0VQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)
$ git pull origin main
From github.com:sokacza/4ID3
 * branch      main       -> FETCH_HEAD
Already up to date.

adama@DESKTOP-9L3E0VQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)
$ |
```

Wiring Diagram

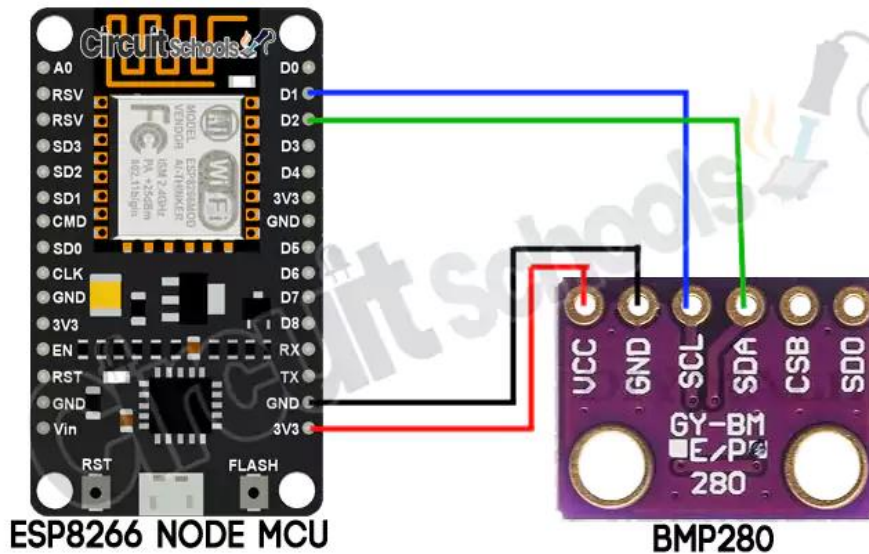
ESP8266 Development Board Pinout:



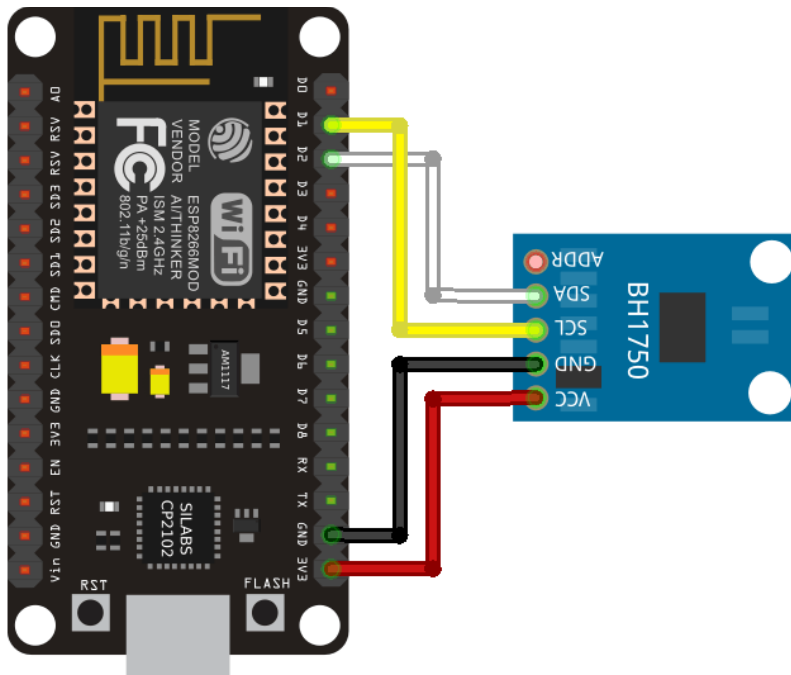
ESP8266 NodeMCU Pinout



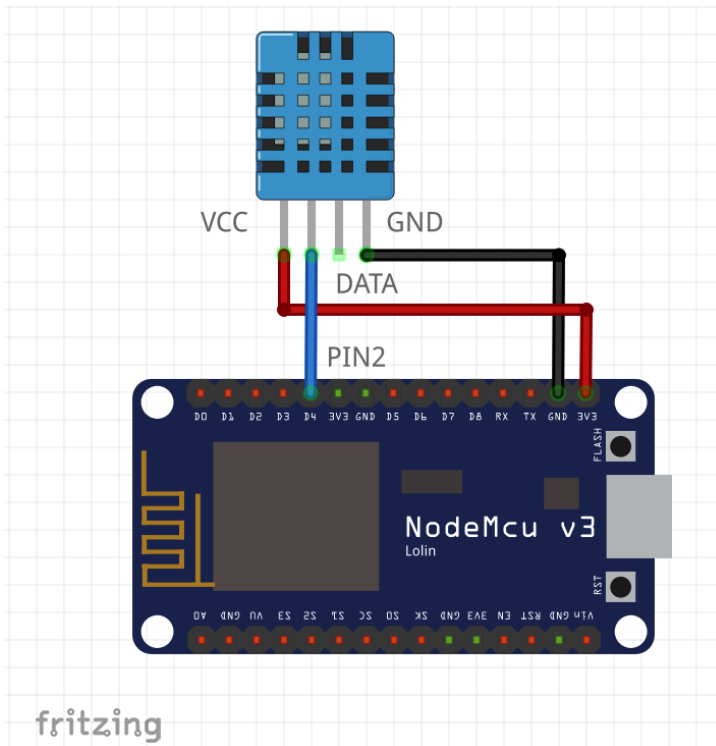
BMP180 Connection:



BH1750 Connection:



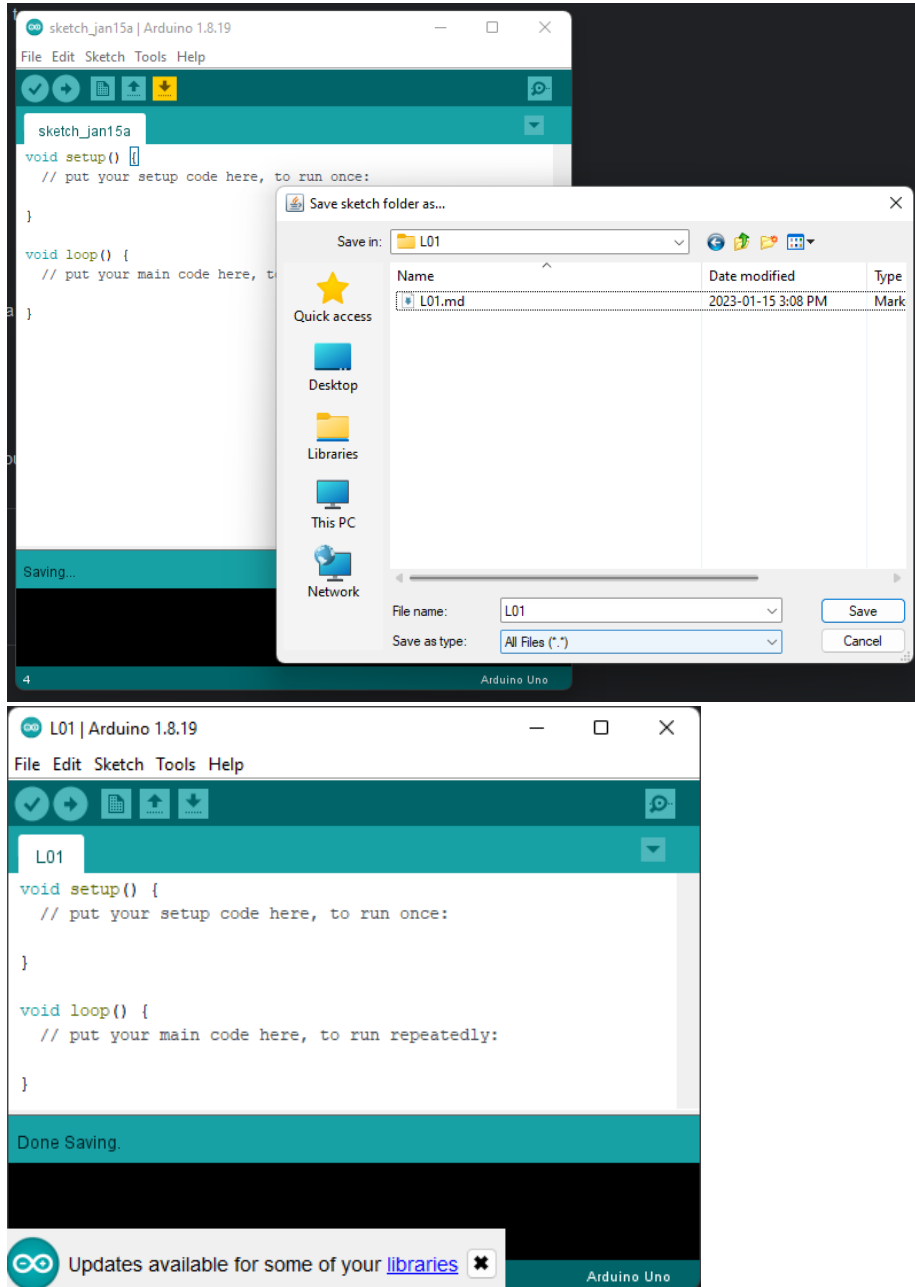
DHT11 Connection (Except connect DHT11 signal pin to D5) :



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:

Lab 1 - Communicating Sensor Data over WiFi using MQTT

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

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

- 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

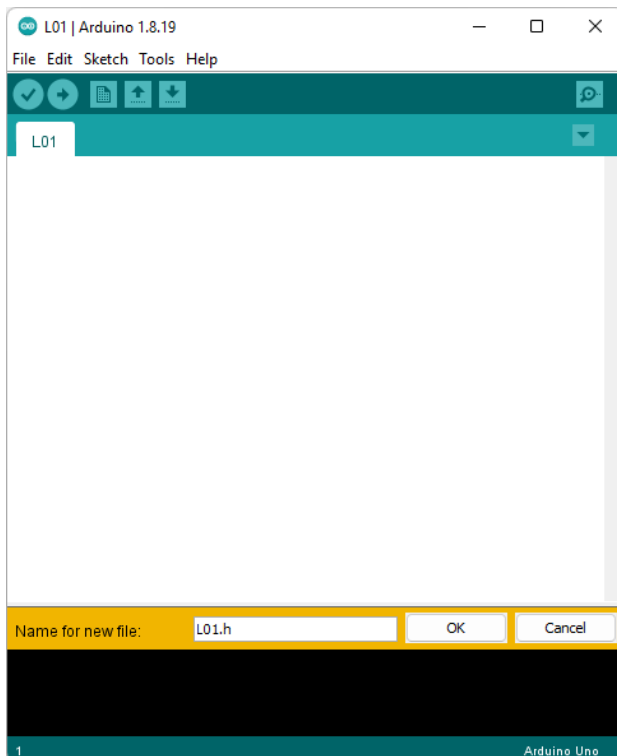
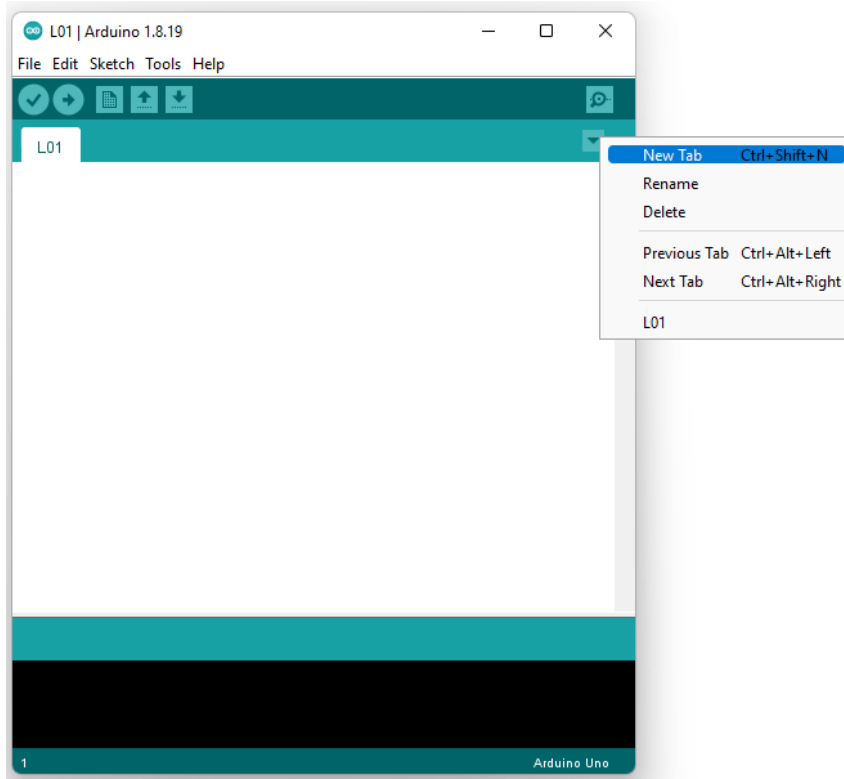
by Nick O'Leary Version 2.8.0 **INSTALLED**

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 **L01.h**.

Lab 1 - 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. Don't worry about including the same one twice, the `#ifndef` directives will prevent the compiler from looking at the same library twice.

```
L01  L01.h
1 //DHT11 Libraries
2 #include <Adafruit_Sensor.h>
3 #include <DHT.h>
4 #include <DHT_U.h>
5
6 //BMP180 Libraries
7 #include <Wire.h>
8 #include <Adafruit_Sensor.h>
9 #include <Adafruit_BMP085_U.h>
10
11 //HP_BH1750 Libraries
12 #include <hp_BH1750.h>
13
```

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.

```
13
14 //Macros
15 #define DHTPIN 14
16 #define DHTTYPE DHT11
17 #define DELAY_BETWEEN_SAMPLES_MS 5000
18
```

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

```
19
20 //Instantiate Sensor Objects
21 DHT_Unified dht(DHTPIN, DHTTYPE);
22 Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);
23 hp_BH1750 BH1750;
24
```

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

Firstly, include our header file.

```
L01 L01.h
1
2 #include "L01.h"
3
4 void setup(){
5
6 }
7
8
9 void loop(){
10
11 }
12
13
```

Next, set up the void setup() function.

```
1
2 #include "L01.h"
3
4 void setup(){
5
6     //Start the serial monitor at 115200 baud
7     Serial.begin(115200);
8
9     //Create a sensor object that is passed into the getSensor method of the dht class
10    //Only the dht sensor requires this
11    sensor_t sensor;
12    dht.temperature().getSensor(&sensor);
13    dht.humidity().getSensor(&sensor);
14
15    //Run the begin() method on each sensor to start communication
16    dht.begin();
17    bmp.begin();
18    BH1750.begin(BH1750_TO_GROUND);
19
20 }
21
```

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.

```
60 void loop(){
61
62     //Polling the DHT and BMP sensor using events
63     sensors_event_t dhtTempEvent, dhtHumEvent, bmpEvent;
64     dht.temperature().getEvent(&dhtTempEvent);
65     dht.humidity().getEvent(&dhtHumEvent);
66     bmp.getEvent(&bmpEvent);
67
68     //Polling the BH sensor
69     BH1750.start();
70     float lux=BH1750.getLux();
71
```

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

```
72 //Printing sensor readings to serial monitor
73 Serial.println("\n-");
74 (!isnan(dhtTempEvent.temperature)) ? Serial.println("Temperature: " + String(dhtTempEvent.temperature) + " degC") : Serial.println("Temperature Sensor Disconnected");
75
76 (!isnan(dhtHumEvent.relative_humidity)) ? Serial.println("Humidity: " + String(dhtHumEvent.relative_humidity) + " %") : Serial.println("Humidity Sensor Disconnected");
77
78 (!isnan(bmpEvent.pressure)) ? Serial.println("Pressure: " + String(bmpEvent.pressure) + " hPa") : Serial.println("Pressure Sensor Disconnected");
79
80 (!isnan(lux)) ? Serial.println("Light Intensity: " + String(lux) + " lux") : Serial.println("Lux Sensor Disconnected");
81
```

This is a shorthand for a one line if-else statement. It is called a ternary statement.

Firstly, we are checking if there is a value stored in each event attribute or if the sensor failed to retrieve a value.

If **isnan()** is **True** then there is **no value** stored. Because we want the opposite condition, we will place a **not operator '!'** in front of the condition.

The **?** operator separates the comparison from the true condition.

The **:** operator separates the true statement from the false statement.

(condition) ? If true : If false

(If we have a temperature value) ? Print the temperature value : Print that the sensor is disconnected

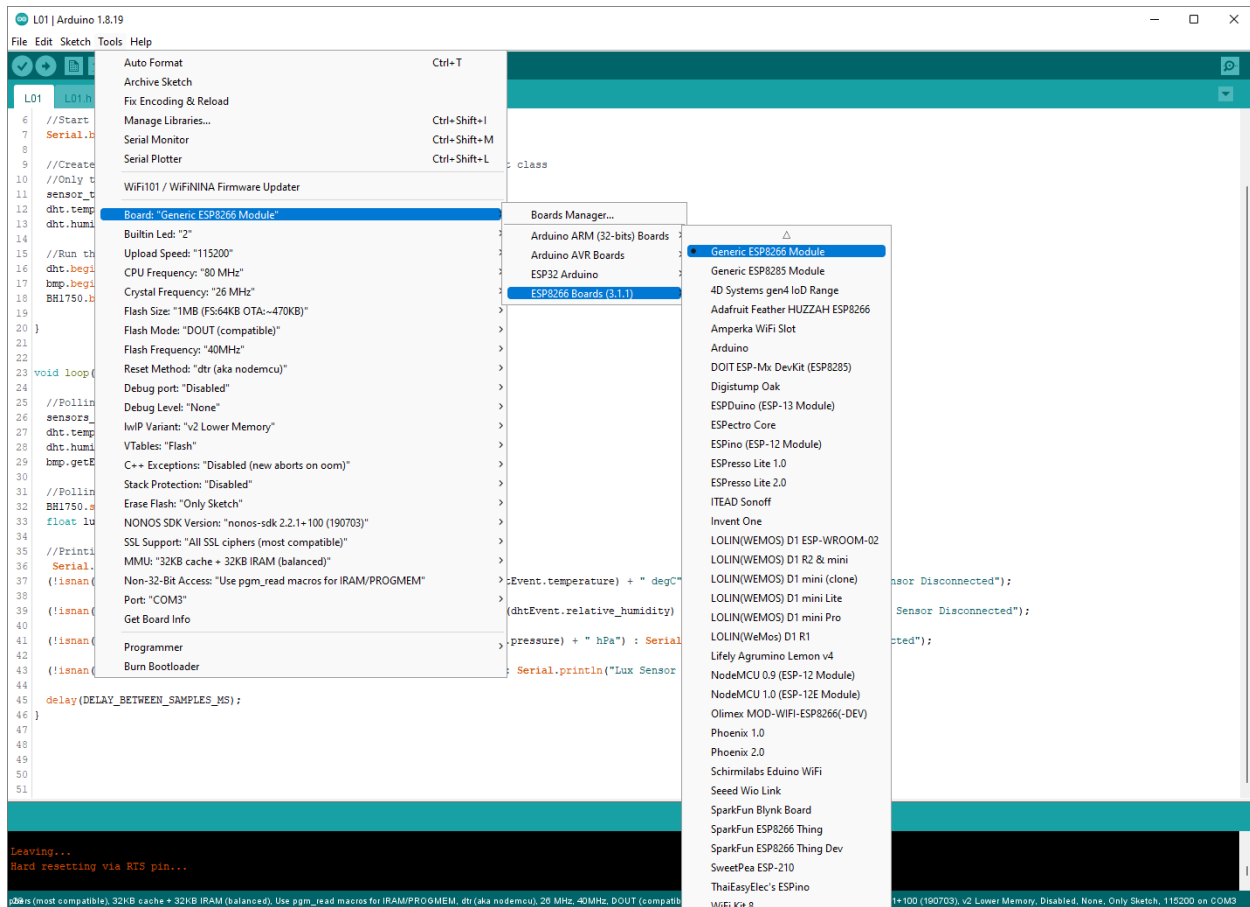
Secondly, to concatenate different datatypes into a print statement, we must **cast them to strings** and **concatenate them** together. An easy way to do this is to use **String(float value)** to cast and **+** operator to concatenate two strings together.

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.

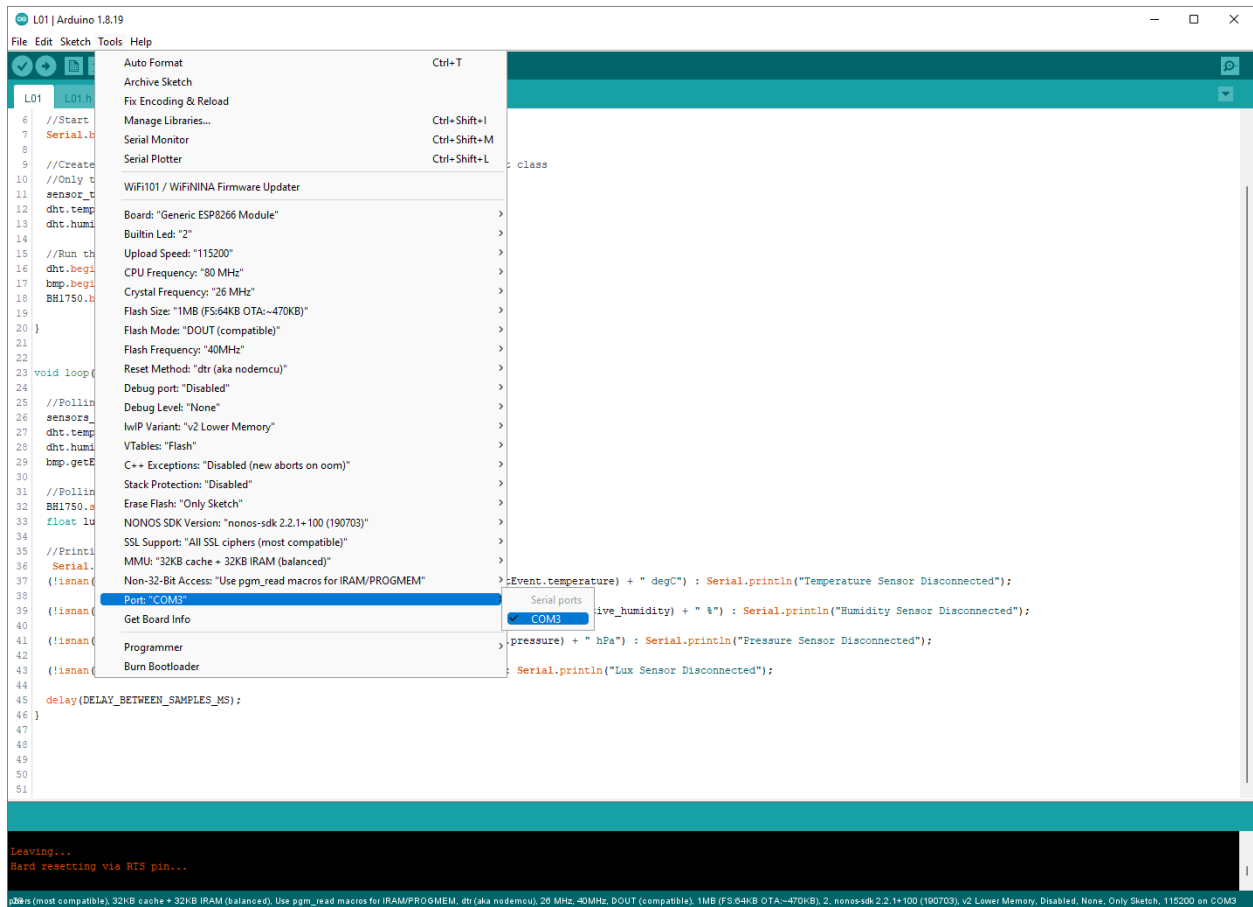
To upload to the board, change the board to Generic ESP8266 Module.

Lab 1 - Communicating Sensor Data over WiFi using MQTT

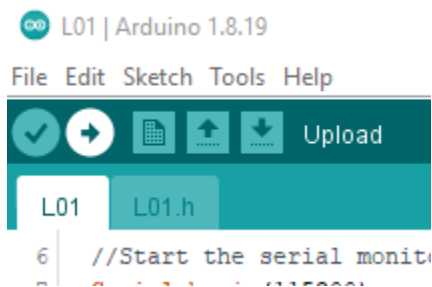


Leave the default communication settings the same, with the exception of the COM port. Select the COM port that your microcontroller is connected to.

Lab 1 - 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.

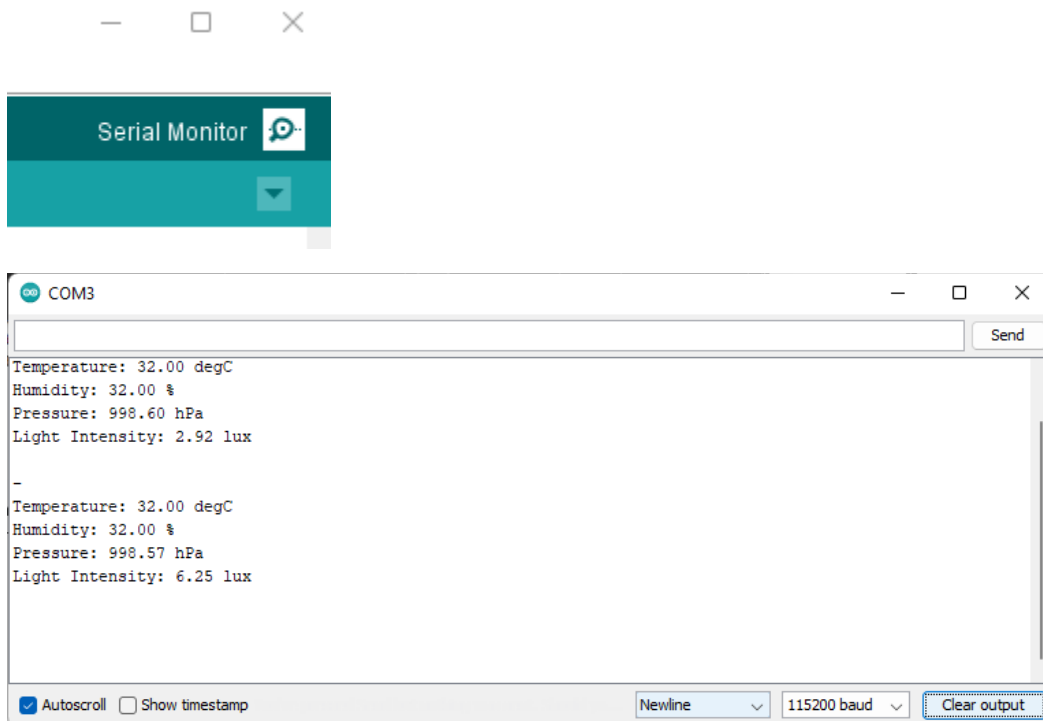


A successful upload should look like this:

```
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...
```

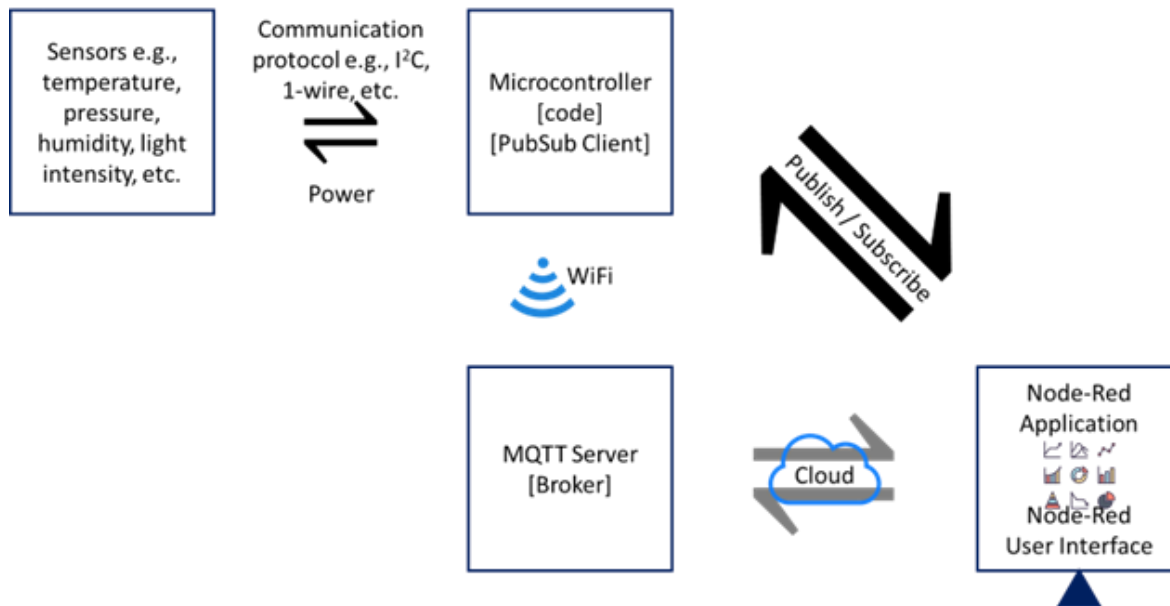
After the upload is complete (NOT DURING), launch the **Serial monitor** to view the microcontroller output.



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 a public (or private) 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

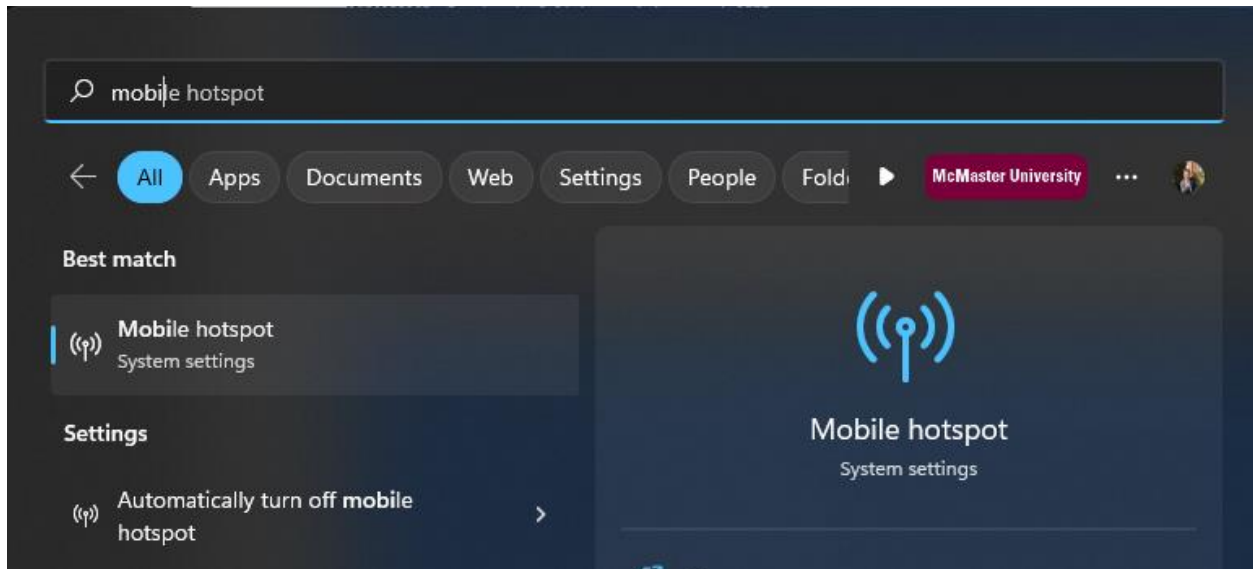
Use the IP address of a known public broker such as Mosquitto's test broker:

test.mosquitto.org

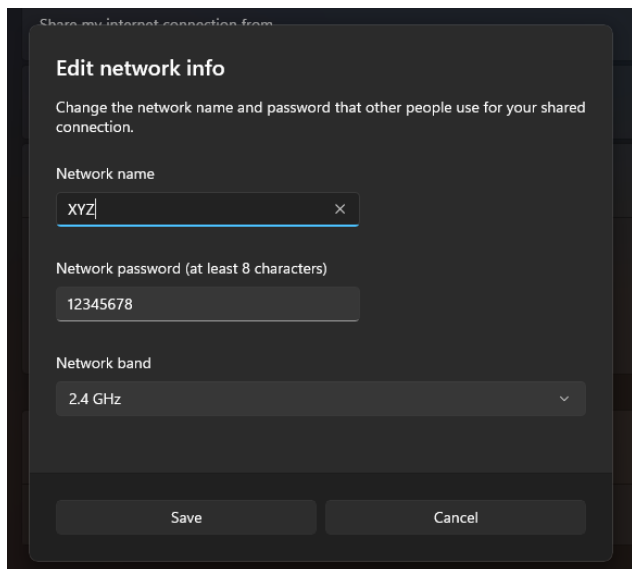
Port is 1883

Hotspotting your Microcontroller

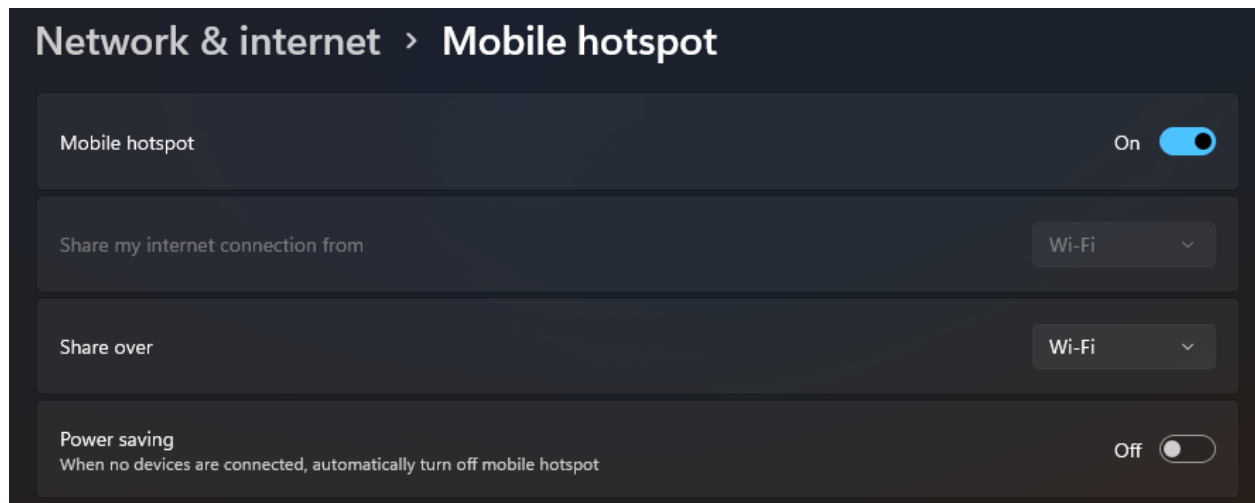
Open **Windows Mobile Hotspot**.



Set your login credentials.



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



Code Changes

```

L01  L01.h$
1 //DHT11 Libraries
2 #include <Adafruit_Sensor.h>
3 #include <DHT.h>
4 #include <DHT_U.h>
5
6 //BMP180 Libraries
7 #include <Wire.h>
8 #include <Adafruit_Sensor.h>
9 #include <Adafruit_BMP085_U.h>
10
11 //HP_BH1750 Libraries
12 #include <hp_BH1750.h>
13
14 //MQTT Libraries
15 #include <ESP8266WiFi.h>
16 #include <PubSubClient.h>
17
18 //Macros
19 #define DHTPIN 14
20 #define DHTTYPE DHT11
21 #define DELAY_BETWEEN_SAMPLES_MS 5000
22
23 //Global Variables
24 char* ssid = "XYZ";
25 char* pass = "12345678";
26 const char* brokerAddress = "192.168.2.13";
27
28 //Instantiate Sensor Objects
29 DHT_Unified dht(DHTPIN, DHTTYPE);
30 Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);
31 hp_BH1750 BH1750;
32
33 //Instantiate MQTT Client
34 WiFiClient espClient;
35 PubSubClient client(espClient);
36

```

In the **setup()** function, we need to initialize the WiFi driver and MQTT client.

```
4 void setup() {
5
6   //Start the serial monitor at 115200 baud
7   Serial.begin(115200);
8
9   //Create a sensor object that is passed into the getSensor method of the dht class
10  //Only the dht sensor requires this
11  sensor_t sensor;
12  dht.temperature().getSensor(&sensor);
13  dht.humidity().getSensor(&sensor);
14
15  //Run the begin() method on each sensor to start communication
16  dht.begin();
17  bmp.begin();
18  BH1750.begin(BH1750_TO_GROUND);
19
20  //Start the WiFi driver and tell it to connect to your local network
21  WiFi.mode(WIFI_STA);
22  WiFi.begin(ssid, pass);
23
24  //While it is connecting, print a '.' to the serial monitor every 500 ms
25  while (WiFi.status() != WL_CONNECTED) {
26    delay(500);
27    Serial.print(".");
28  }
29
30  //Once connected, print the local IP address
31  Serial.println("");
32  Serial.println("WiFi connected");
33  Serial.println("IP address: ");
34  Serial.println(WiFi.localIP());
35
36  //Set the MQTT client to connect to the desired broker
37  client.setServer(brokerAddress, 1883);
38
39 }
40
```

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().

Lab 1 - Communicating Sensor Data over WiFi using MQTT

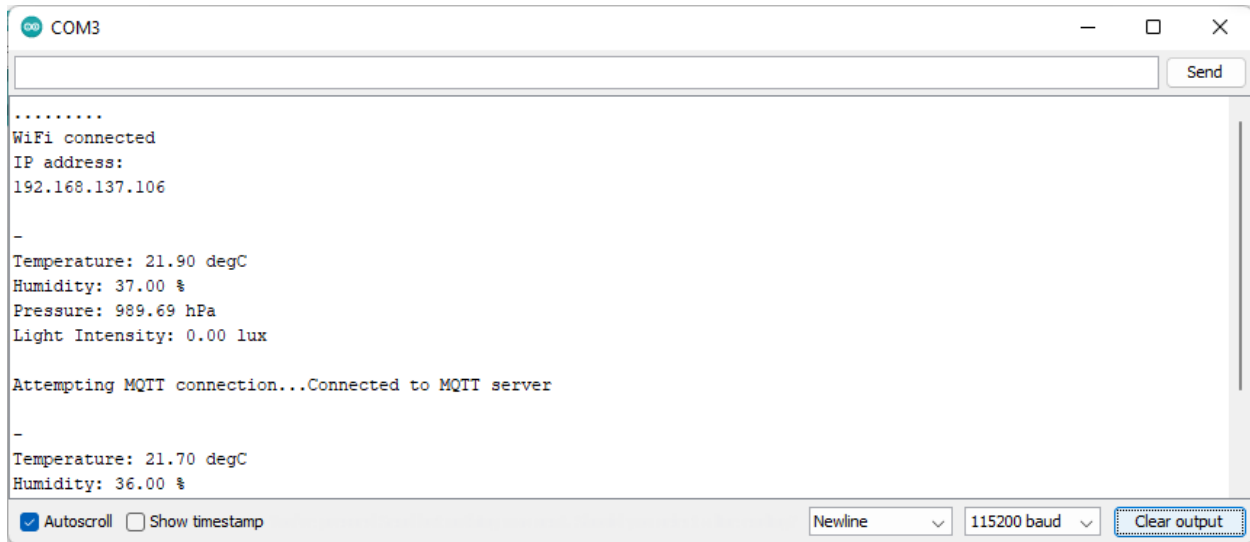
```
40
41 void reconnect() {
42
43     //While the client remains unconnected from the MQTT broker, attempt to reconnect every 2 seconds
44     //Also, print diagnostic information
45     while (!client.connected()) {
46         Serial.print("Attempting MQTT connection...");
47
48         if (client.connect("ESP8266Client")) {
49             Serial.println("Connected to MQTT server");
50             client.subscribe("testTopic");
51         } else {
52             Serial.print("Failed to connect to MQTT server, rc = ");
53             Serial.print(client.state());
54             delay(2000);
55         }
56     }
57 }
58
```

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

```
59
60 void loop(){
61
62     //Polling the DHT and BMP sensor using events
63     sensors_event_t dhtTempEvent, dhtHumEvent, bmpEvent;
64     dht.temperature().getEvent(&dhtTempEvent);
65     dht.humidity().getEvent(&dhtHumEvent);
66     bmp.getEvent(&bmpEvent);
67
68     //Polling the BH sensor
69     BH1750.start();
70     float lux=BH1750.getLux();
71
72     //Printing sensor readings to serial monitor
73     Serial.println("\n\n");
74     (!isnan(dhtTempEvent.temperature)) ? Serial.println("Temperature: " + String(dhtTempEvent.temperature) + " degC") : Serial.println("Temperature Sensor Disconnected");
75
76     (!isnan(dhtHumEvent.relative_humidity)) ? Serial.println("Humidity: " + String(dhtHumEvent.relative_humidity) + " %") : Serial.println("Humidity Sensor Disconnected");
77
78     (!isnan(bmpEvent.pressure)) ? Serial.println("Pressure: " + String(bmpEvent.pressure) + " hPa") : Serial.println("Pressure Sensor Disconnected");
79
80     (!isnan(lux)) ? Serial.println("Light Intensity: " + String(lux) + " lux") : Serial.println("Lux Sensor Disconnected");
81
82     //If the client disconnects from the MQTT broker, attempt to reconnect
83     if (!client.connected()) {
84         reconnect();
85     }
86     if(!client.loop())
87         client.connect("ESP8266Client");
88
89     //Publish the sensor data to the associated topics
90     client.publish("4ID3_G7/temperature", String(dhtTempEvent.temperature).c_str());
91     delay(100);
92     client.publish("4ID3_G7/humidity", String(dhtHumEvent.relative_humidity).c_str());
93     delay(100);
94     client.publish("4ID3_G7/pressure", String(bmpEvent.pressure).c_str());
95     delay(100);
96     client.publish("4ID3_G7/light", String(lux).c_str());
97
98     delay(DELAY_BETWEEN_SAMPLES_MS);
99 }
```

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

Lab 1 - Communicating Sensor Data over WiFi using MQTT



The screenshot shows a serial monitor window with the title 'COM3'. The window contains a text area with the following output:

```
.....  
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 top right of the window is a 'Send' button. At the bottom, there are several controls: a checked 'Autoscroll' checkbox, an unchecked 'Show timestamp' checkbox, a 'Newline' dropdown menu, a '115200 baud' dropdown menu, and a 'Clear output' button.

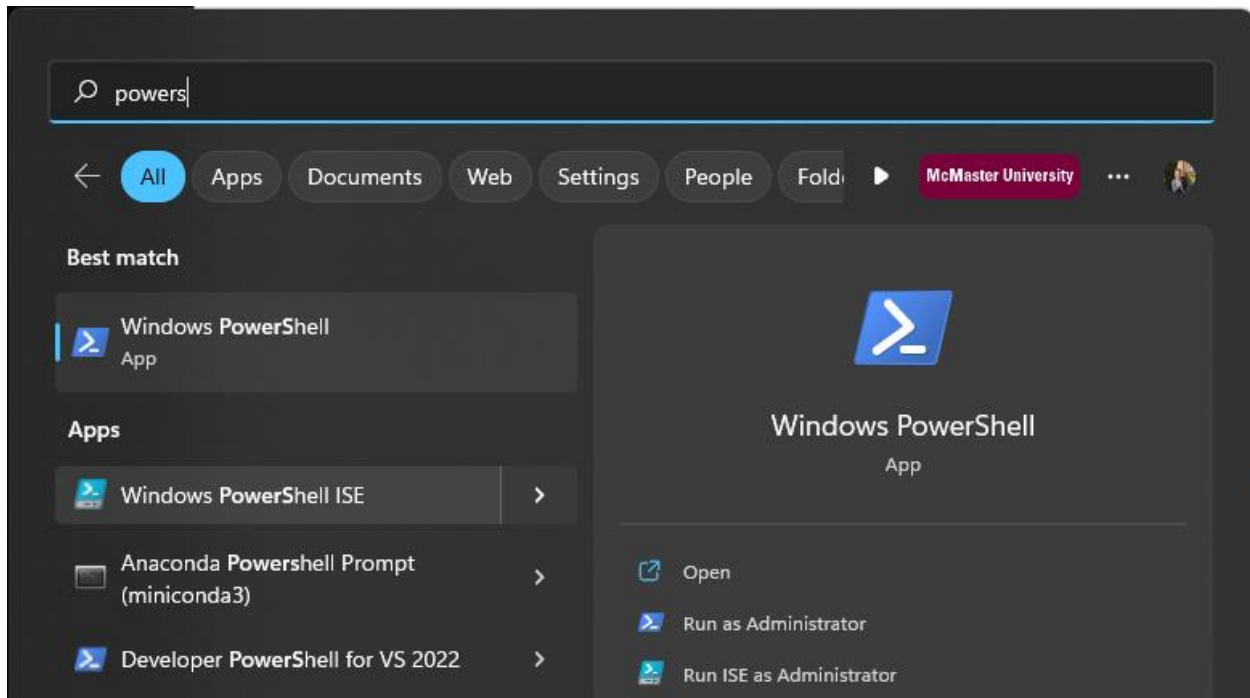
Verifying Connection

We will be using 2 different devices and applications to verify that the data is being published to the corresponding topics.

- a) Using the Mosquitto terminal application
- b) Using a mobile application

Mosquitto Terminal Application

Open Powershell and navigate to the install directory of the Mosquitto MQTT broker, that was installed in the pre-lab.



Lab 1 - Communicating Sensor Data over WiFi using MQTT

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\adama> cd 'C:\Program Files\mosquitto\'
PS C:\Program Files\mosquitto> ls

    Directory: C:\Program Files\mosquitto

Mode                LastWriteTime         Length Name
----                -
d-----          2023-01-15  7:23 PM             devel
-a-----          2022-08-16  9:34 AM              230 aclfile.example
-a-----          2022-08-16  9:34 AM          135368 ChangeLog.txt
-a-----          2022-08-16  9:34 AM           1568 edl-v10
-a-----          2022-08-16  9:34 AM          14197 epl-v20
-a-----          2022-07-05  5:43 PM        3415552 libcrypto-1_1-x64.dll
-a-----          2022-07-05  5:43 PM        685056 libssl-1_1-x64.dll
-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 mosquittopt.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
-a-----          2022-08-16  9:34 AM           1886 NOTICE.md
-a-----          2022-08-16  9:34 AM           355 pwfile.example
-a-----          2022-08-16  9:34 AM           939 README-letsencrypt.md
-a-----          2022-08-16  9:34 AM          2453 README-windows.txt
-a-----          2022-08-16  9:34 AM          3768 README.md
-a-----          2023-01-15  7:23 PM        66085 Uninstall.exe

PS C:\Program Files\mosquitto>
```

Launch the subscribe applications with the following flags:

-h - MQTT broker IP address

-t - the topic that your wish to connect to

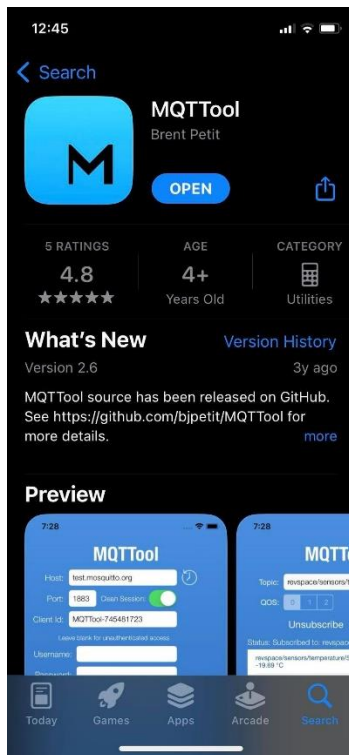
```
Windows PowerShell

PS C:\Program Files\mosquitto> .\mosquitto_sub.exe -h test.mosquitto.org -t "4ID3_67/humidity"
37.00
37.00
87.00
60.00
51.00
46.00
43.00
39.00
39.00
63.00
51.00
45.00
41.00
37.00
38.00
37.00
38.00
```

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

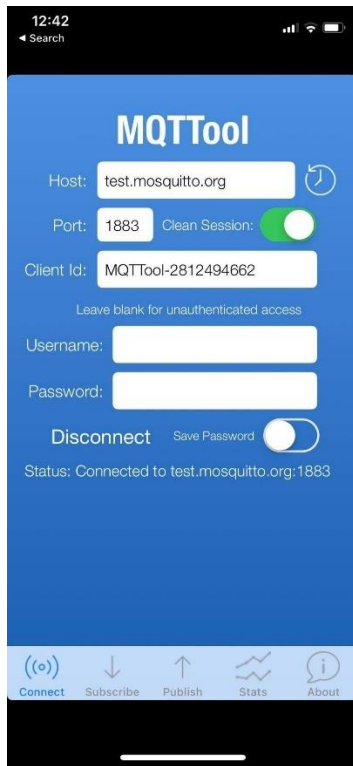
Using a Mobile Application

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

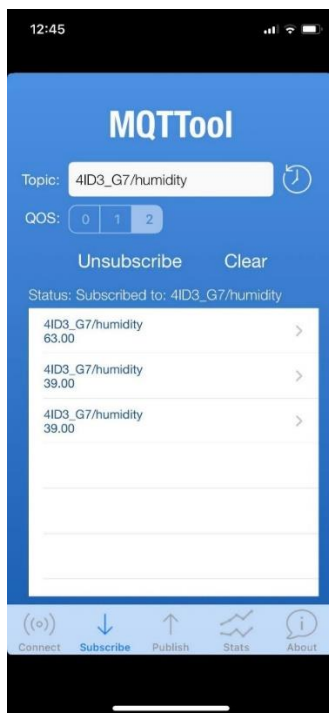


Next, connect to the public mosquito broker:

Lab 1 - Communicating Sensor Data over WiFi using MQTT



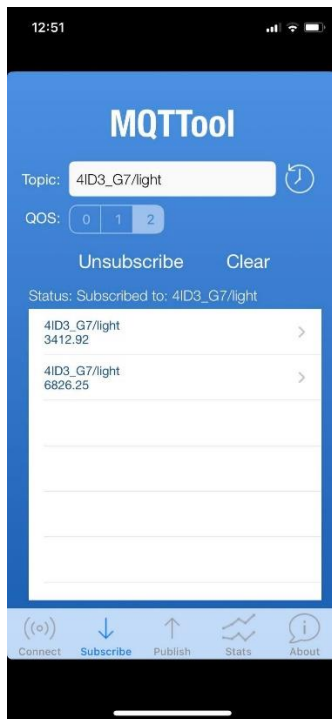
Next, subscribe to the topic of choice.



Lastly, wait for values to populate.

Lab 1 - Communicating Sensor Data over WiFi using MQTT

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

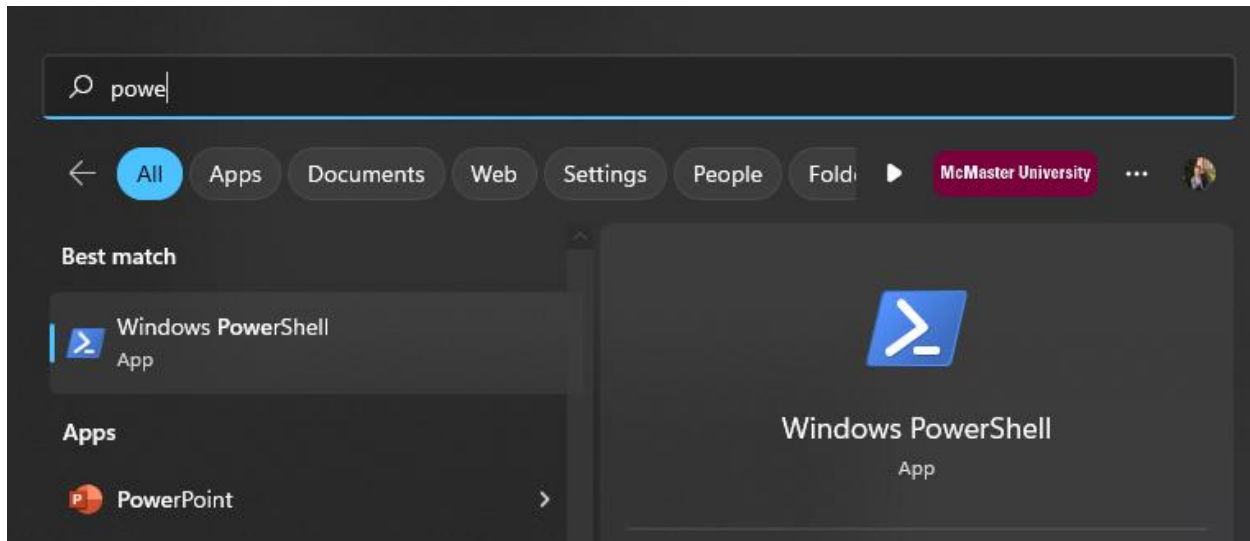


NodeRED Dashboard

Please follow the pre-lab instructions to install NodeRED.

To start NodeRED, open Powershell and type the command:

node-red



```
Select node-red
PS C:\Program Files\mosquitto> node-red
17 Jan 12:58:41 - [info]

Welcome to Node-RED
=====

17 Jan 12:58:41 - [info] Node-RED version: v3.0.2
17 Jan 12:58:41 - [info] Node.js version: v18.13.0
17 Jan 12:58:41 - [info] Windows_NT 10.0.22000 x64 LE
17 Jan 12:58:42 - [info] Loading palette nodes
17 Jan 12:58:44 - [info] Dashboard version 3.3.1 started at /ui
17 Jan 12:58:44 - [info] Settings file : C:\Users\adama\.node-red\settings.js
17 Jan 12:58:44 - [info] Context store : 'default' [module=memory]
17 Jan 12:58:44 - [info] User directory : \Users\adama\.node-red
17 Jan 12:58:44 - [warn] Projects disabled : editorTheme.projects.enabled=false
17 Jan 12:58:44 - [info] Flows file : \Users\adama\.node-red\flows.json
17 Jan 12:58:44 - [info] Creating new flow file
17 Jan 12:58:44 - [warn]

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

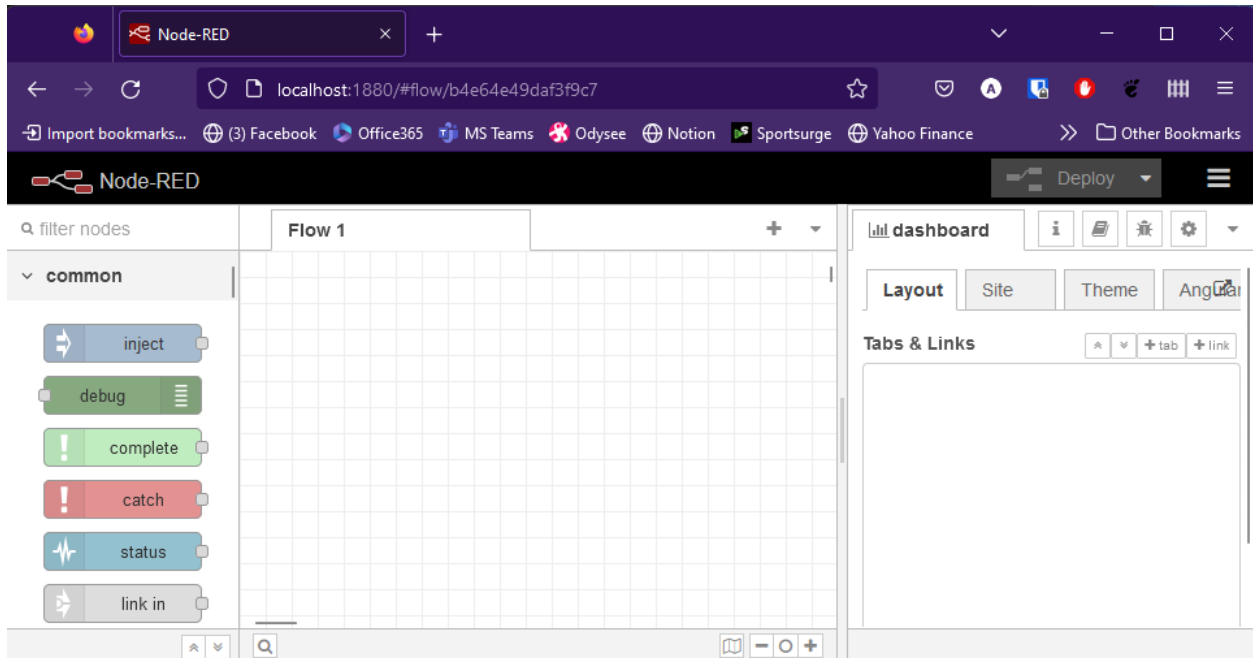
If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

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.
-----

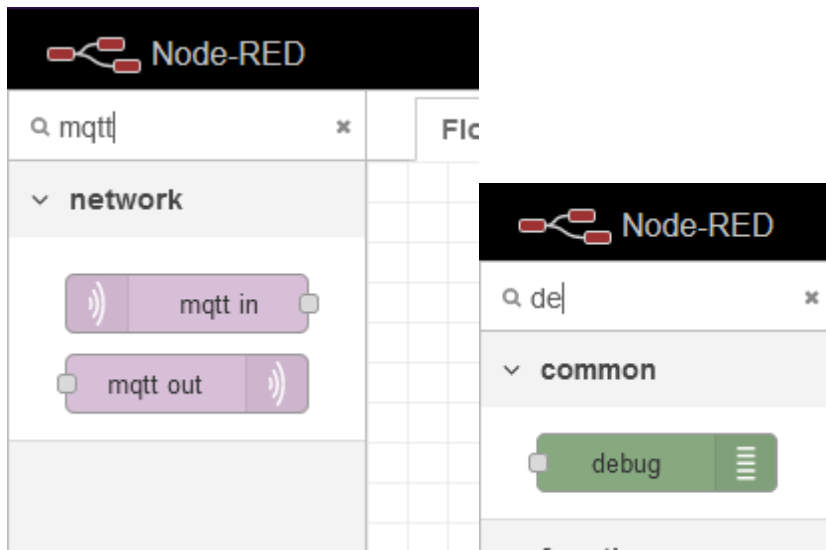
17 Jan 12:58:44 - [warn] Encrypted credentials not found
17 Jan 12:58:44 - [info] Server now running at http://127.0.0.1:1880/
17 Jan 12:58:44 - [info] Starting flows
17 Jan 12:58:44 - [info] Started flows
```

Navigate to the URL presented, in a web browser.

Lab 1 - Communicating Sensor Data over WiFi using MQTT

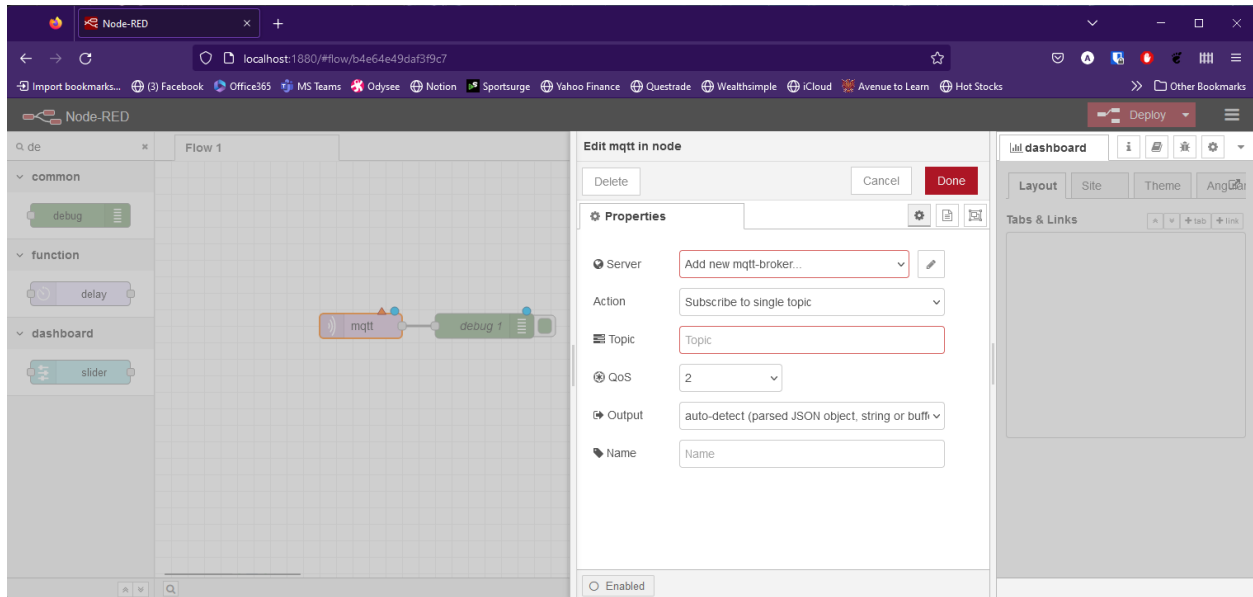


Filter nodes to find the **mqtt in** node and the **debug** node. Drag them into your flow diagram.



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

Lab 1 - 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.

Lab 1 - Communicating Sensor Data over WiFi using MQTT

Edit mqtt in node

Delete Cancel Done

Properties

Server Mosquitto Test

Action Subscribe to single topic

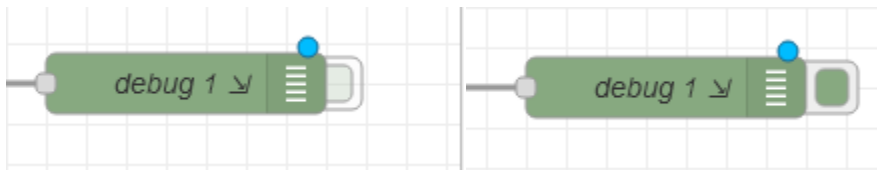
Topic 4ID3_G7/humidity

QoS 0

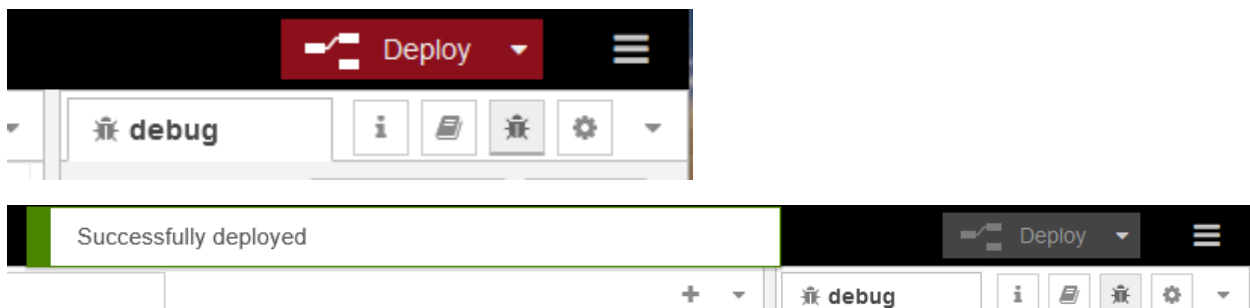
Output auto-detect (parsed JSON object, string or buffer)

Name Name

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



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



Lab 1 - Communicating Sensor Data over WiFi using MQTT

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

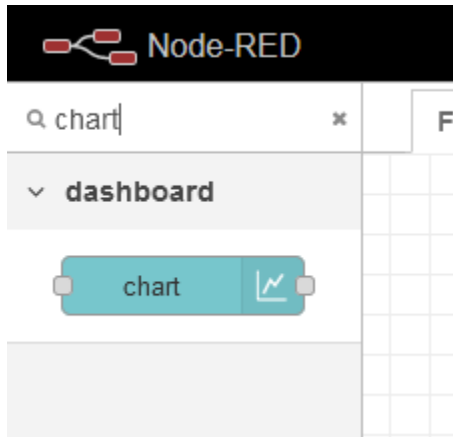
- Left Panel (Palette):** Contains a search bar and two categories of nodes:
 - common:** Includes nodes like `inject`, `debug`, `complete`, `catch`, `status`, `link in`, `link call`, `link out`, and `comment`.
 - function:** Includes a `function` node.
- Center Panel (Canvas):** Displays a flow named "Flow 1". It contains two nodes connected by a wire: an MQTT node labeled `4ID3_G7/humidity` (with a green "connected" status indicator) and a `debug 1` node.
- Right Panel (Debug Console):** Shows a log of messages. The top of the panel has a "debug" header and a "all nodes" filter. The log entries are as follows:

| Timestamp | Node | Message |
|--------------------------|---------------|---|
| 2023-01-17, 1:21:17 p.m. | node: debug 1 | 4ID3_G7/humidity : msg.payload : number |
| 2023-01-17, 1:21:23 p.m. | node: debug 1 | 4ID3_G7/humidity : msg.payload : number |
| 2023-01-17, 1:21:28 p.m. | node: debug 1 | 4ID3_G7/humidity : msg.payload : number |
| 2023-01-17, 1:21:52 p.m. | node: debug 1 | 4ID3_G7/humidity : msg.payload : number |
| 2023-01-17, 1:21:57 p.m. | node: debug 1 | 4ID3_G7/humidity : msg.payload : number |
| 2023-01-17, 1:22:03 p.m. | node: debug 1 | 4ID3_G7/humidity : msg.payload : number |

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 title bar shows the breadcrumb: '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' (value: 'Home'), 'Icon' (value: 'dashboard'), 'State' (toggle: 'Enabled'), and 'Nav. Menu' (toggle: 'Visible'). At the bottom, there is a yellow informational box with text about the 'Icon' field, mentioning Material Design icons, Font Awesome icons, and Weather icons, and providing an example for Google Material icons: '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.

Lab 1 - 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

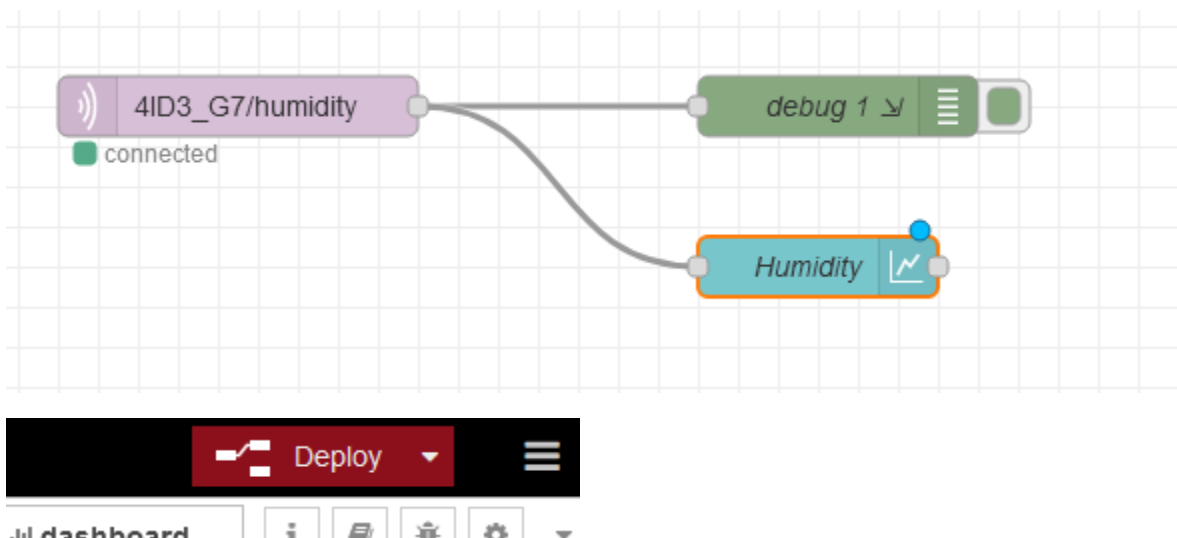
Series Colours

Blank label display this text before valid data arrives

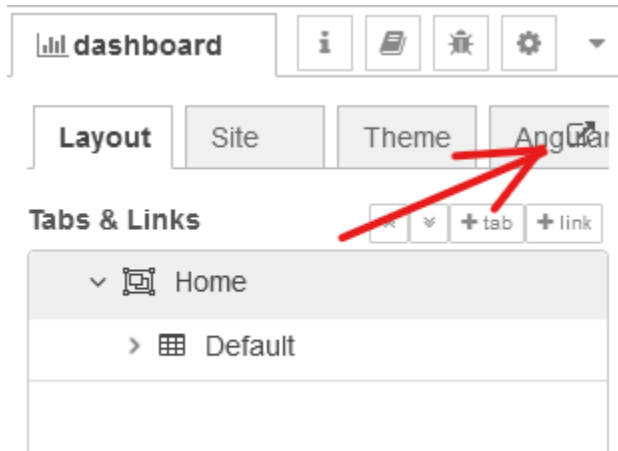
</> Class Optional CSS class name(s) for widget

Name Humidity

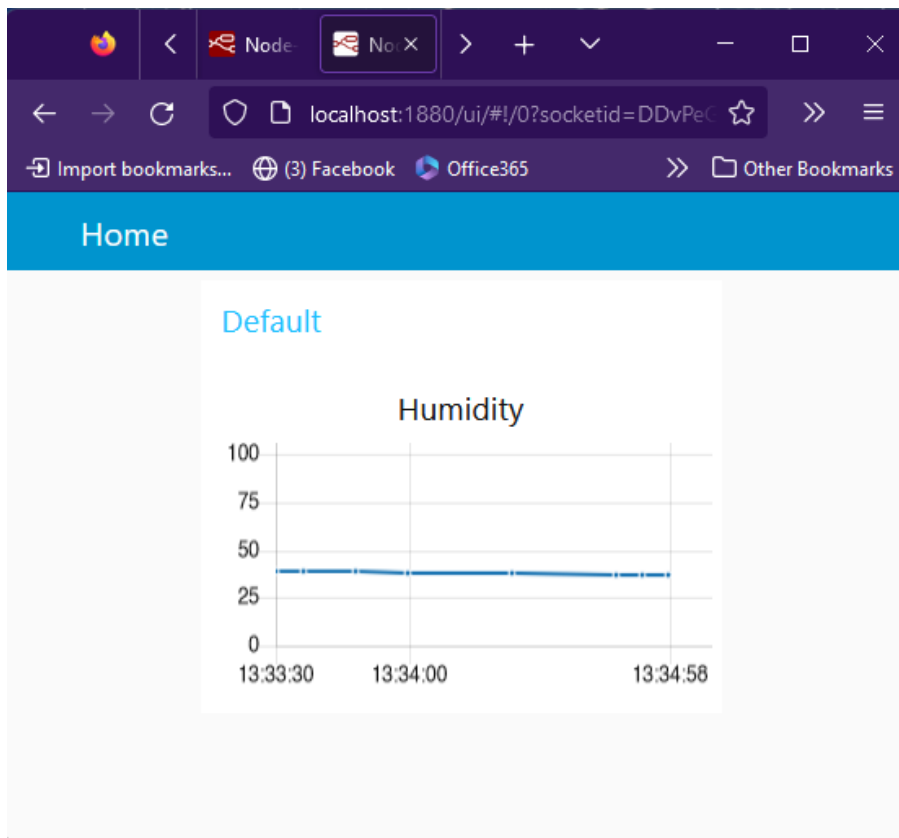
Connect your **mqtt in** node to the input of your **chart** node. Press **Deploy** to save changes.



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.



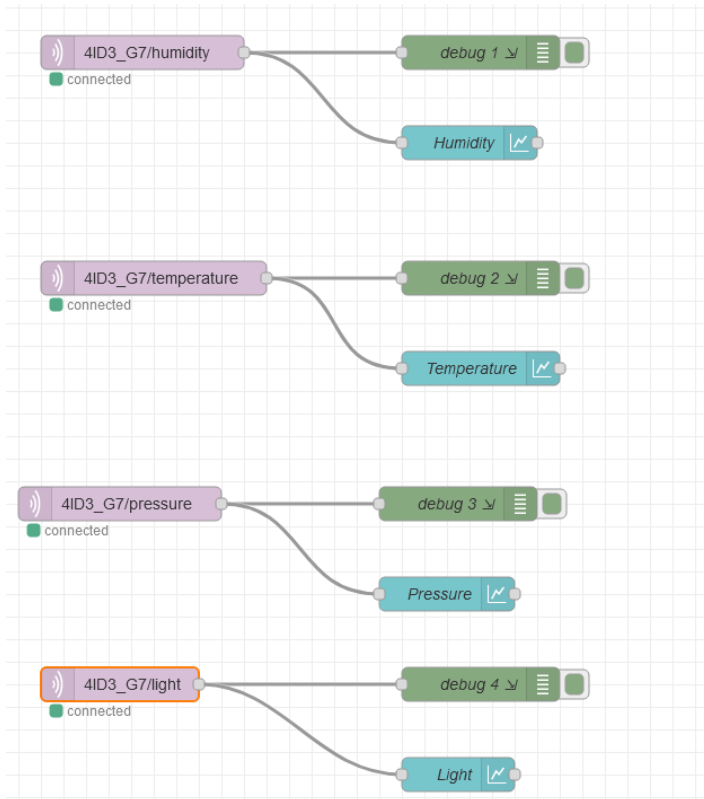
Wait for data to populate.



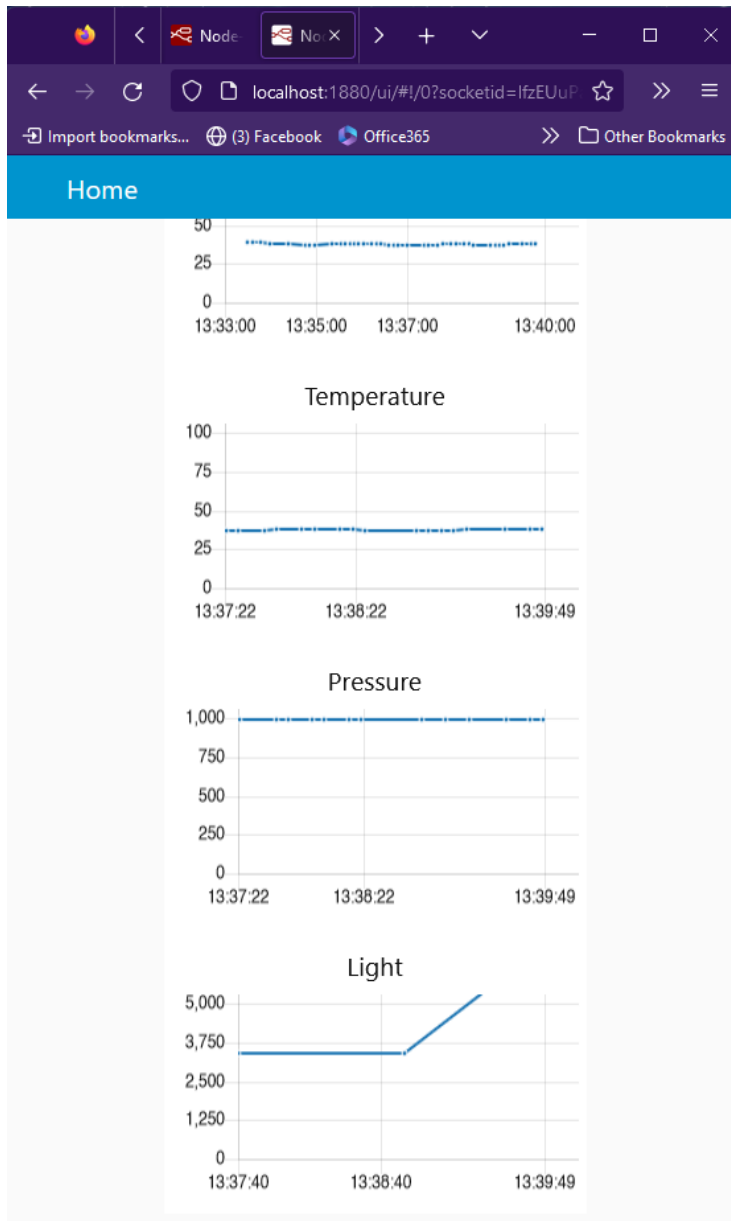
Now that you have seen how to visualize one topic, attempt to visualize the rest.

The resultant flow should look like this:

Lab 1 - Communicating Sensor Data over WiFi using MQTT



Lab 1 - 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

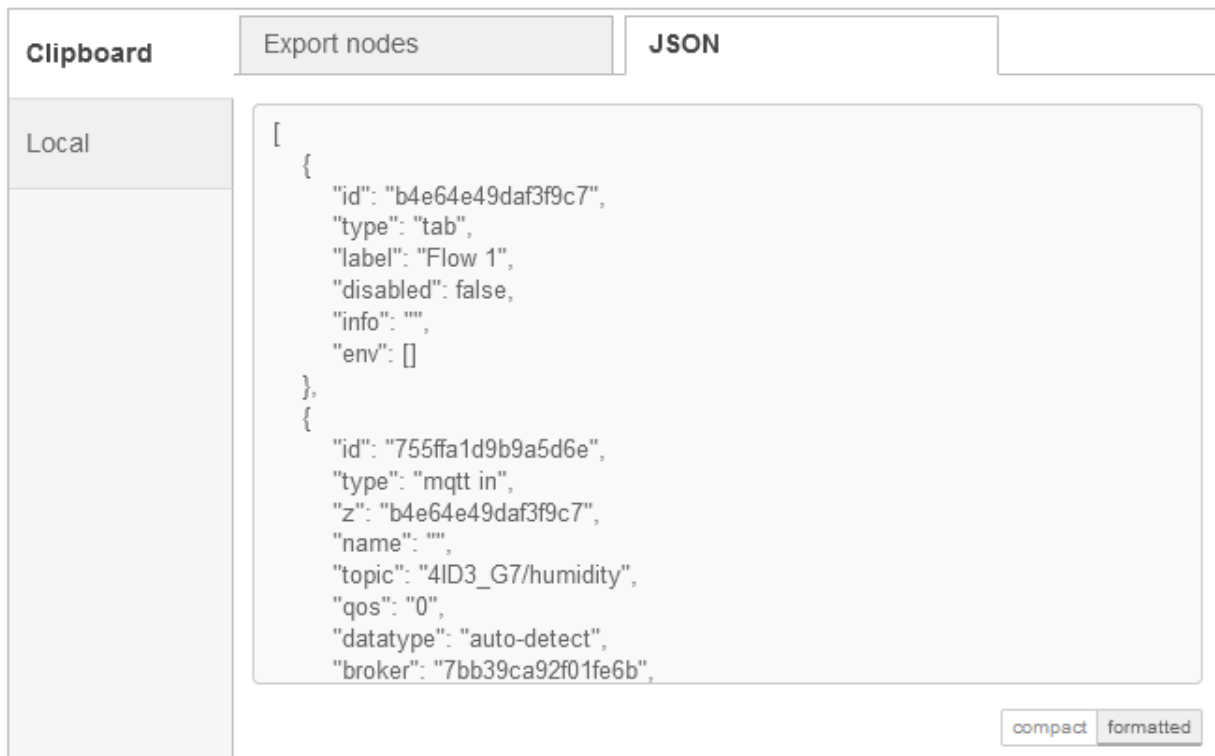
Exporting a NodeRED Flow as JSON

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

Select **current flow**.



Select **JSON**.



Press **Download**.

Export nodes

Export selected nodes current flow all flows

Clipboard

Local

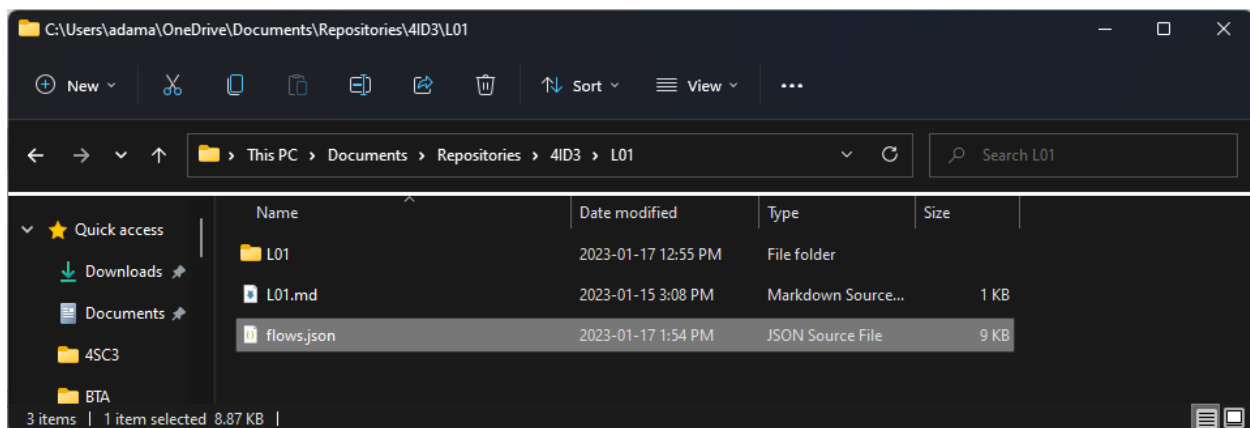
Export nodes

```
[
  {
    "id": "b4e64e49daf3f9c7",
    "type": "tab",
    "label": "Flow 1",
    "disabled": false,
    "info": "",
    "env": []
  },
  {
    "id": "755ffa1d9b9a5d6e",
    "type": "mqtt in",
    "z": "b4e64e49daf3f9c7",
    "name": "",
    "topic": "4ID3_G7/humidity",
    "qos": "0",
    "datatype": "auto-detect",
    "broker": "7bb39ca92f01fe6b",
  }
]
```

compact formatted

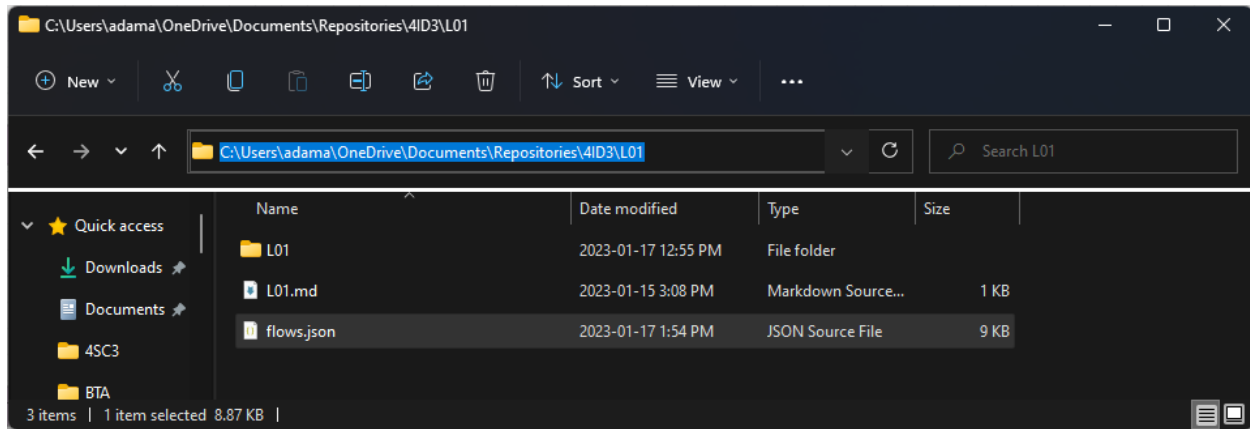
Cancel Download Copy to clipboard

Cut and paste the **flows.json** file from your **Downloads/** folder to the **L01** folder in the **local repo**.



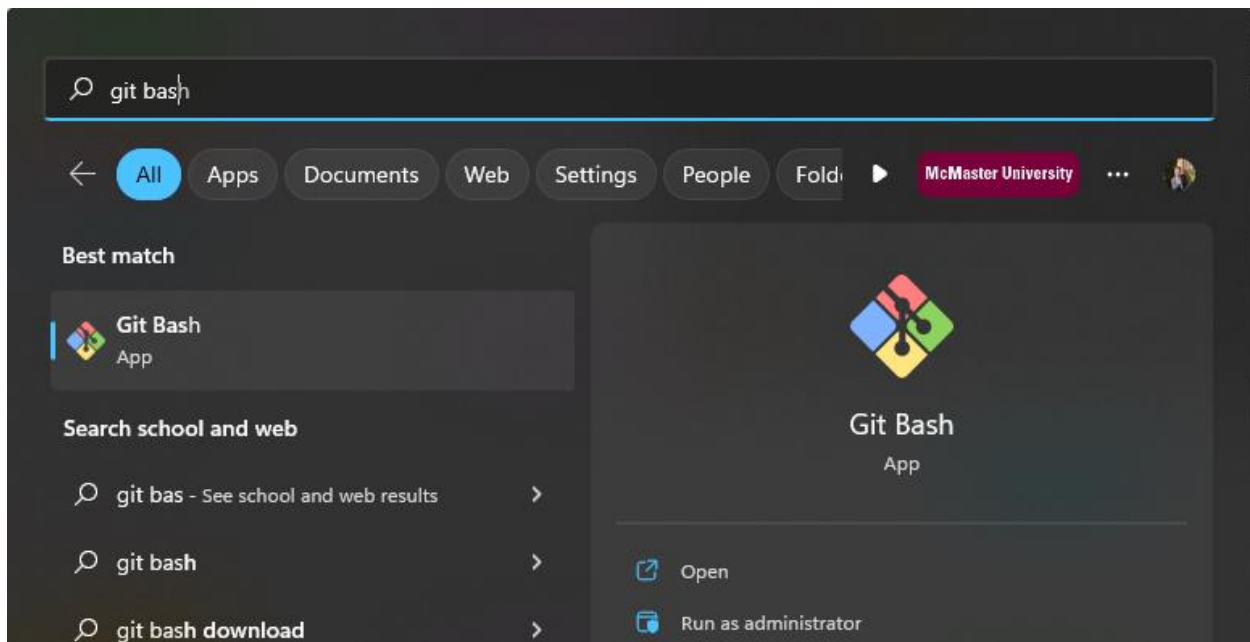
Committing and Pushing Changes to GitHub

Click on the little **folder icon** in the **navigation bar** to view to **path**.



Copy this path to clipboard.

Open **Git Bash**.



Use the **cd** command to change directories into the L01 folder.


```
MINGW64:/c/Users/adama/OneDrive/Documents/Repositories/4ID3/L01

adama@DESKTOP-9L3E0VQ MINGW64 ~
$ cd "C:\Users\adama\OneDrive\Documents\Repositories\4ID3\L01"

adama@DESKTOP-9L3E0VQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3/L01 (main)
$ ls -la
total 13
drwxr-xr-x 1 adama 197609  0 Jan 17 13:55 ./
drwxr-xr-x 1 adama 197609  0 Jan 15 14:58 ../
drwxr-xr-x 1 adama 197609  0 Jan 17 12:55 L01/
-rw-r--r-- 1 adama 197609 177 Jan 15 15:08 L01.md
-rw-r--r-- 1 adama 197609 9092 Jan 17 13:54 flows.json

adama@DESKTOP-9L3E0VQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3/L01 (main)
$
```

Go back one directory using the `cd ..` command. You should be in the **4ID3** folder now.

```
MINGW64:/c/Users/adama/OneDrive/Documents/Repositories/4ID3

drwxr-xr-x 1 adama 197609  0 Jan 17 12:55 L01/
-rw-r--r-- 1 adama 197609 177 Jan 15 15:08 L01.md
-rw-r--r-- 1 adama 197609 9092 Jan 17 13:54 flows.json

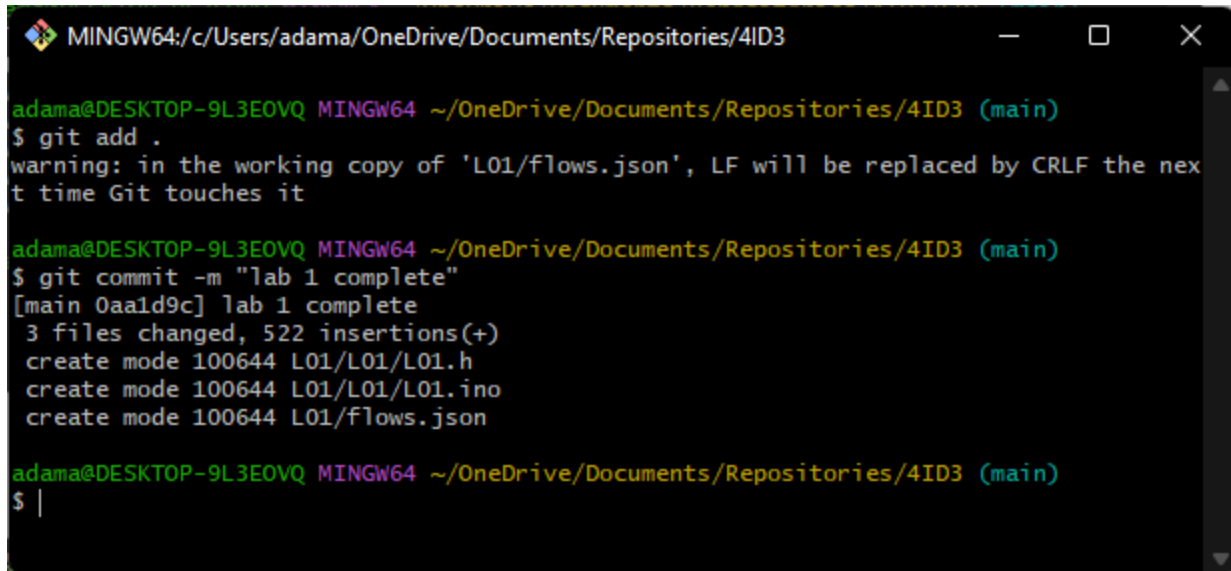
adama@DESKTOP-9L3E0VQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3/L01 (main)
$ cd ..

adama@DESKTOP-9L3E0VQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)
$ ls -la
total 5
drwxr-xr-x 1 adama 197609  0 Jan 15 14:58 ./
drwxr-xr-x 1 adama 197609  0 Jan 15 14:56 ../
drwxr-xr-x 1 adama 197609  0 Jan 15 20:17 .git/
drwxr-xr-x 1 adama 197609  0 Jan 17 13:55 L01/
-rw-r--r-- 1 adama 197609 14 Jan 15 12:47 README.md

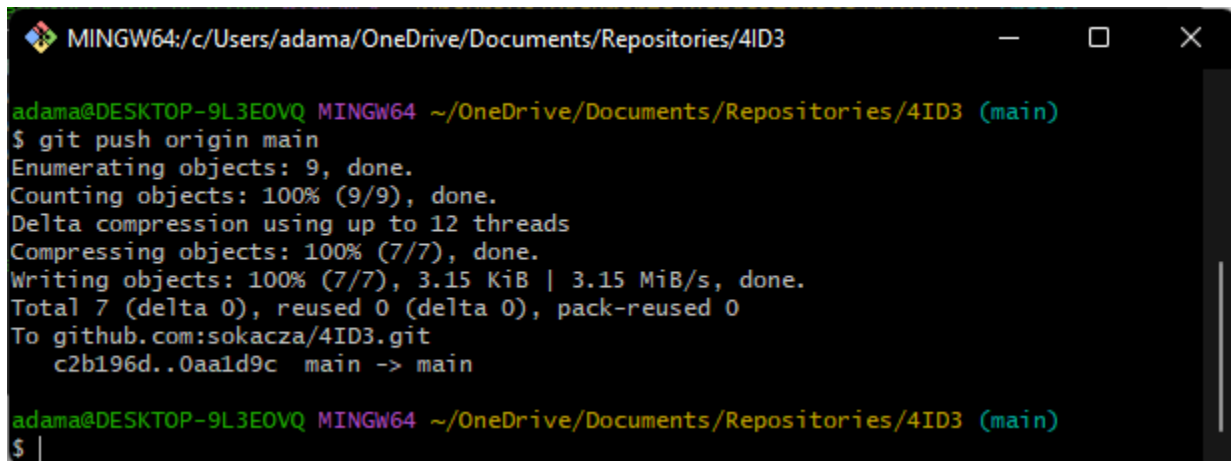
adama@DESKTOP-9L3E0VQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)
$ |
```

Add all the new changes using the `git add .` command.

Commit the changes using the `git commit -m ""` command. Use a message that represents what has been changed since the previous commit.

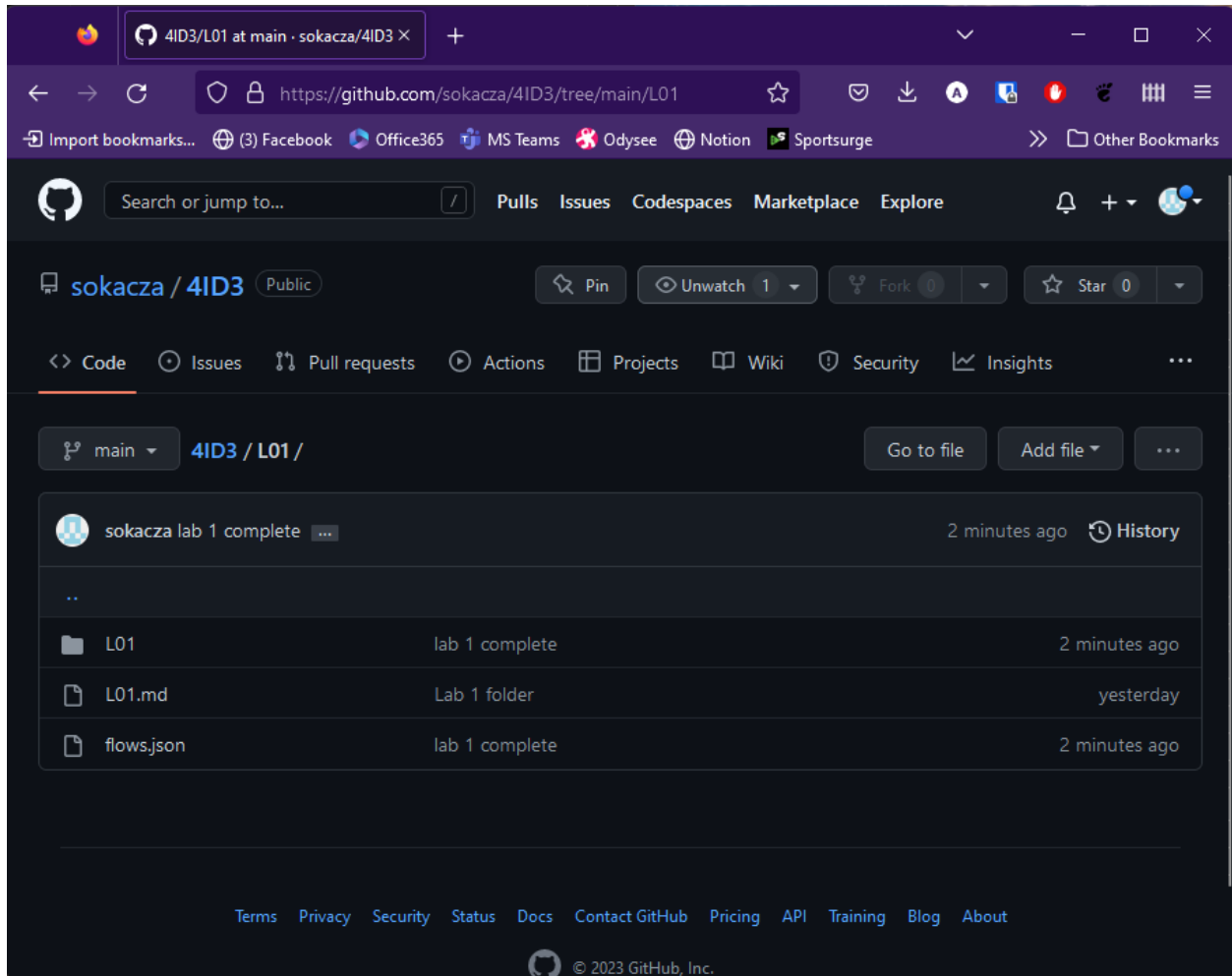
A terminal window titled 'MINGW64:/c/Users/adama/OneDrive/Documents/Repositories/4ID3' with standard window controls. The prompt is 'adama@DESKTOP-9L3E0VQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)'. The user enters '\$ git add .' followed by a warning: 'warning: in the working copy of 'L01/flows.json', LF will be replaced by CRLF the next time Git touches it'. Then the user enters '\$ git commit -m "lab 1 complete"', resulting in '[main 0aa1d9c] lab 1 complete' and a summary: '3 files changed, 522 insertions(+), create mode 100644 L01/L01/L01.h, create mode 100644 L01/L01/L01.ino, create mode 100644 L01/flows.json'. The prompt returns to '\$ |'.

Push the changes in the **main branch** of your **local repo** to the remote **GitHub repo** named **origin**.

A terminal window titled 'MINGW64:/c/Users/adama/OneDrive/Documents/Repositories/4ID3' with standard window controls. The prompt is 'adama@DESKTOP-9L3E0VQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)'. The user enters '\$ git push origin main'. The output shows: 'Enumerating objects: 9, done.', 'Counting objects: 100% (9/9), done.', 'Delta compression using up to 12 threads', 'Compressing objects: 100% (7/7), done.', 'Writing objects: 100% (7/7), 3.15 KiB | 3.15 MiB/s, done.', 'Total 7 (delta 0), reused 0 (delta 0), pack-reused 0', and 'To github.com:sokacza/4ID3.git c2b196d..0aa1d9c main -> main'. The prompt returns to '\$ |'.

Log into your GitHub account to verify that all changes have been uploaded.

Lab 1 - Communicating Sensor Data over WiFi using MQTT



END