

4ID3 - IoT Devices and Networks

Lab 1

Communicating Sensor Data over WiFi using MQTT

Adam Sokacz

Salman Bawa

Group Number:

Name, Student Number:

Name, Student Number:

Name, Student Number:

Submission Date:



Objective

A practice-based understanding on the “things” of the Internet of Things, and the “Internet” of the Internet of Things, including temperature, pressure, humidity, and light intensity sensors, connected to ESP 8266 microcontroller (with Publication-Subscription client), communicating over 802.11ah WiFi to an MQTT server on cloud, corresponding with Node-Red application, showcasing outcome in a Node-Red user interface.

Contents

Objective	2
Feedback	3
Additional Resources	3
Pre-Lab Questions	4
Post-Lab Questions	4
Setting up the Workspace	5
Wiring Diagram	12
Reading Sensor Data	14
Publishing to an MQTT Broker	23
Connecting to the Mosquitto Public Test Broker	23
Hotspotting your Microcontroller	23
Implementing MQTT	25
Verifying Connection	31
Mosquitto Terminal Application	31
Using a Mobile Application	33
NodeRED Dashboard	36
Visualizing NodeRED Data	41
Saving and Pushing Your Project	47
Exporting a NodeRED Flow as JSON	47
Committing and Pushing Changes to GitHub	49

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

Lab GitHub Repo (<https://github.com/sokacza/4ID3>)

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? What is a topic?

Q2 - In your own words, what does QoS mean for MQTT transmissions?

Q3 - Define LAN and the role of IP addresses in communicating between computers over a LAN.

Post-Lab Questions

Q1 - Explain your observations in Node-Red user interface as you interact with the sensors. Is the change instantaneous?

Q2 - Turn the microcontroller power off; is the Node-Red application still connected to the MQTT server? Why is sensor reading not being registered in the Node-Red user interface?

Q3 - Turn the microcontroller power on, and turn the Node-Red application off. Is the Node-Red application still connected to the MQTT server? The sensor reading is not being registered in the Node-Red user interface, but is the sensor still communicating with the MQTT server?

Q4 - Would it be reasonable to implement a WiFi sensor network across a forest? Across a city? Describe how geographical factors, environmental factors, etc. influence your choice of access technology.

Q5 - List 3 practical applications of WiFi sensor networks that would benefit your school or neighbourhood community.

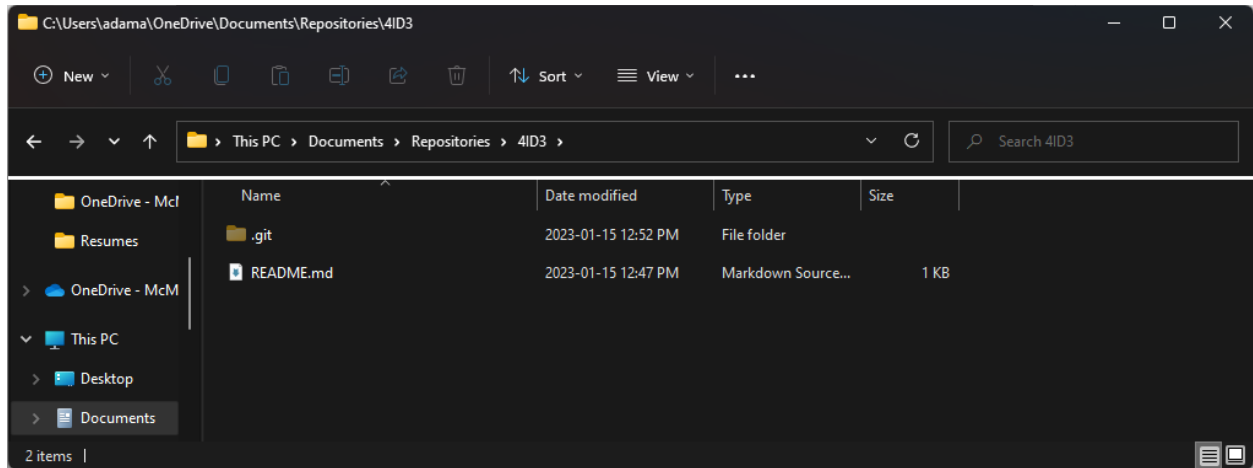
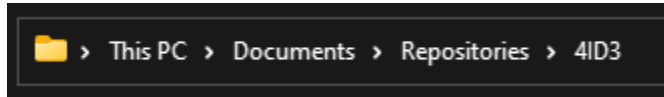
Q6 - List 3 security concerns of this network. How might these concerns be mitigated?

Q7 - Below, write a 3 sentence LinkedIn post about key learning takeaways from this lab.

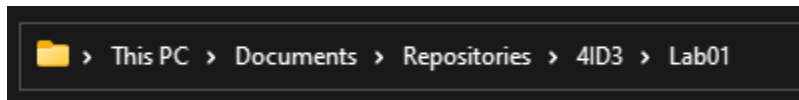
Setting up the Workspace

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.

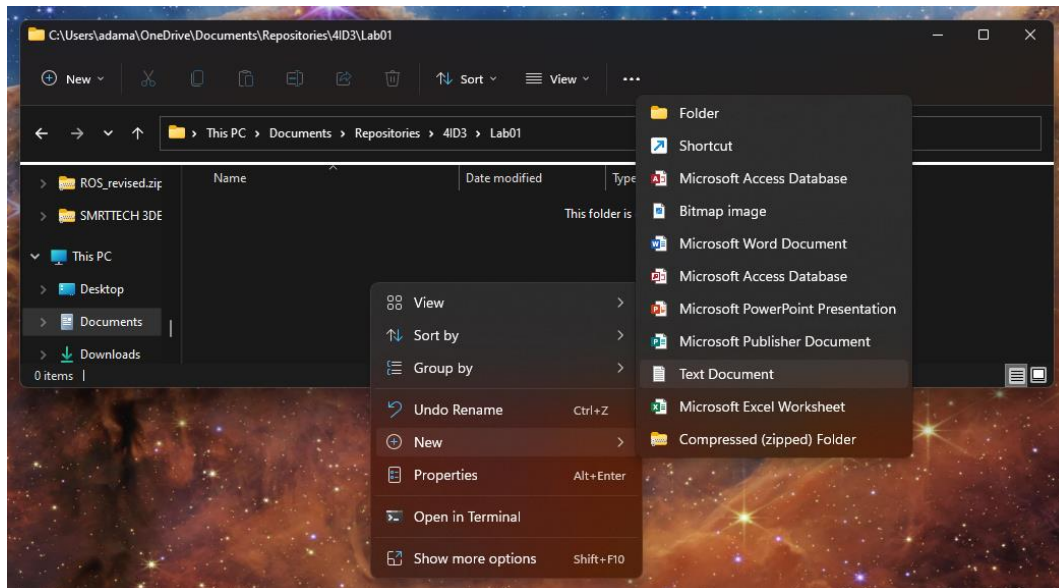
Navigate to your local git repository for this course.



Create a new folder named **Lab01**. Navigate inside this folder.

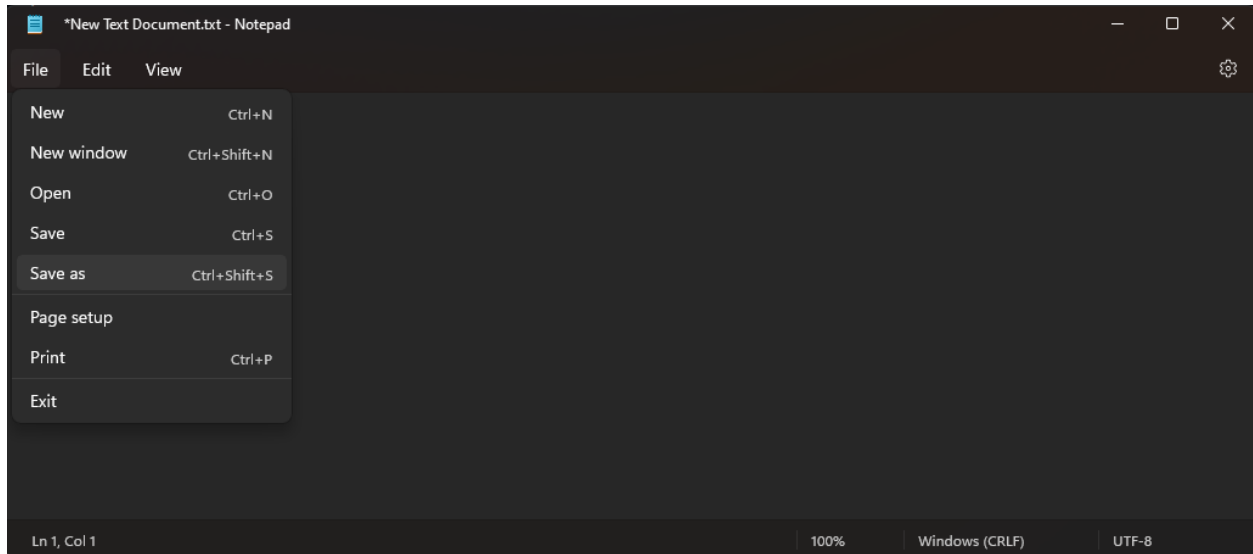


Create a new text file in the folder.

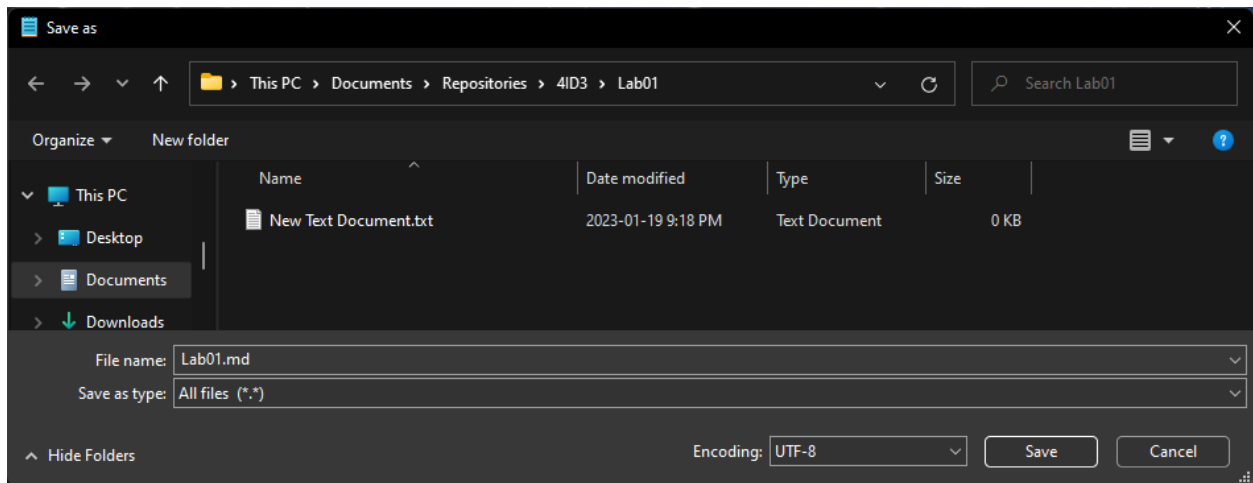


Lab 1 - Communicating Sensor Data over WiFi using MQTT

Press **File > Save as**.

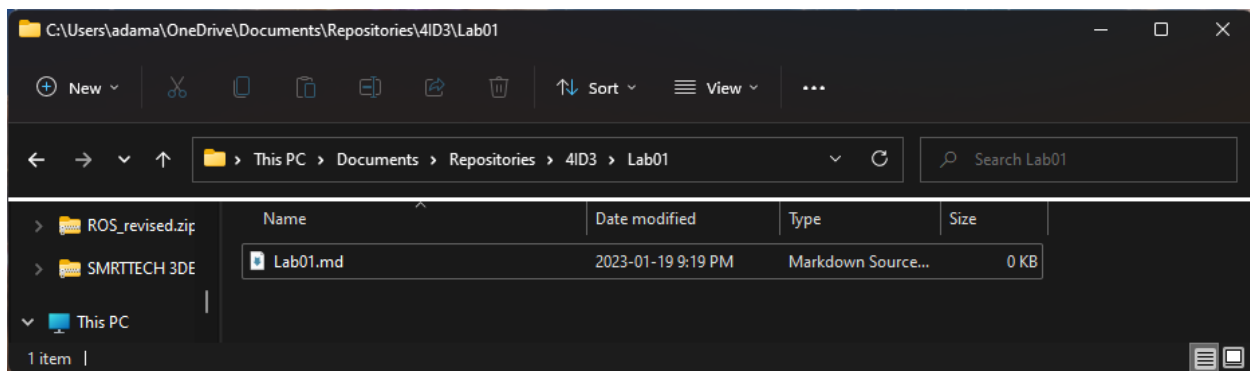
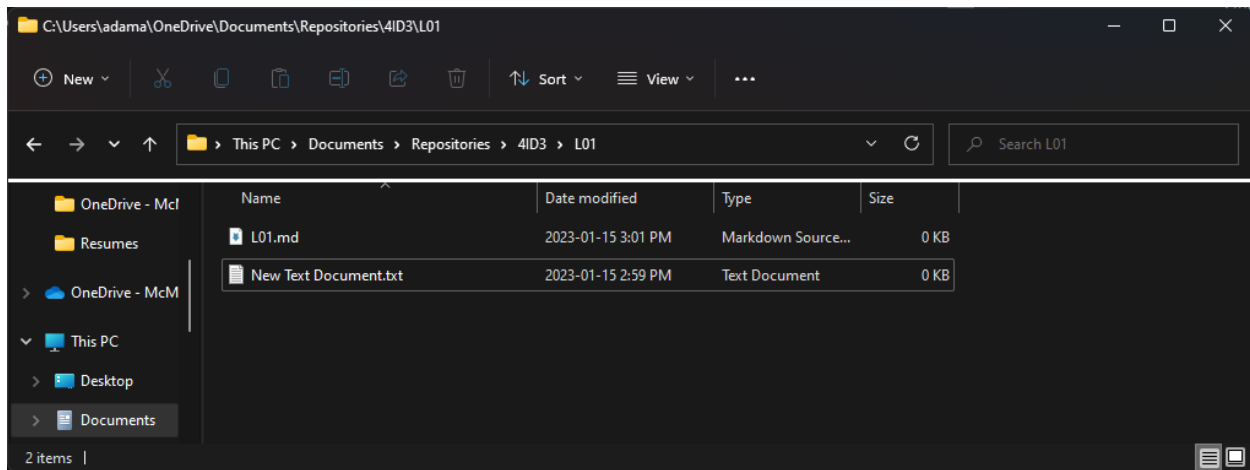


Save it as **Lab01.md**. Ensure that the **Save as type** is set to **All files (*.*)**.

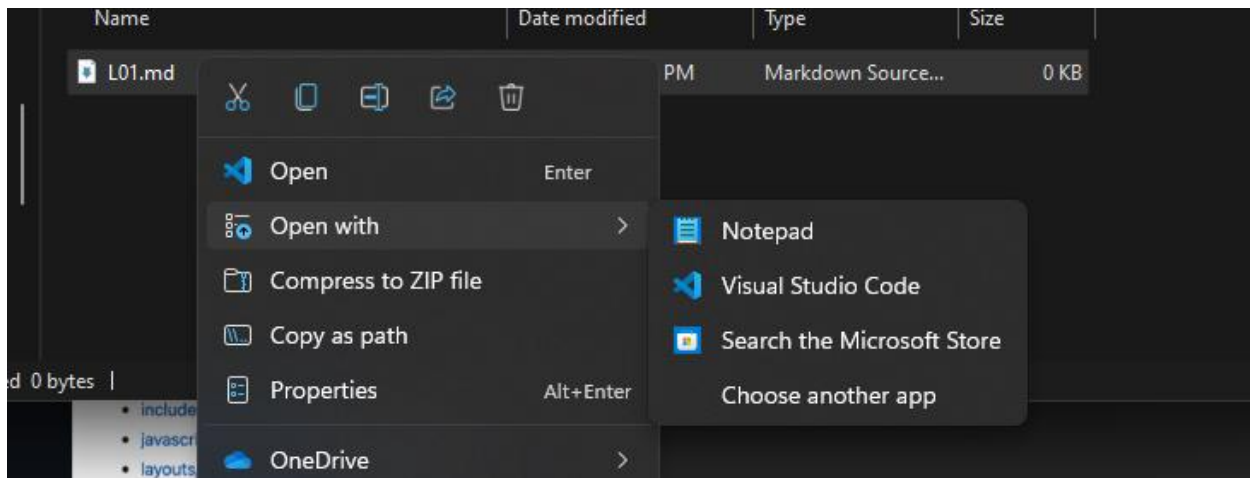


Now, you should have two files, a **text file** and a **markdown file**. Delete the text file.

Lab 1 - Communicating Sensor Data over WiFi using MQTT



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

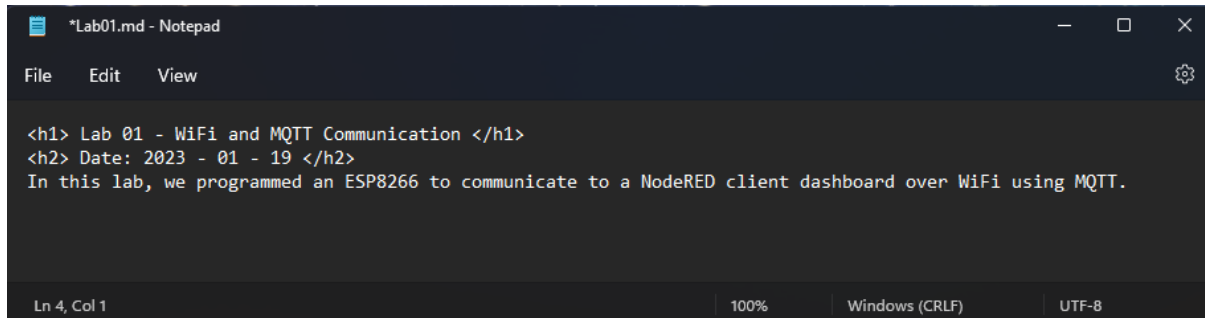


Writing markdown documents to explain your code is very similar to HTML. A reference guide can be found [here](#):

Lab 1 - Communicating Sensor Data over WiFi using MQTT

<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.



```
*Lab01.md - Notepad
File Edit View
<h1> Lab 01 - WiFi and MQTT Communication </h1>
<h2> Date: 2023 - 01 - 19 </h2>
In this lab, we programmed an ESP8266 to communicate to a NodeRED client dashboard over WiFi using MQTT.
Ln 4, Col 1 100% Windows (CRLF) UTF-8
```

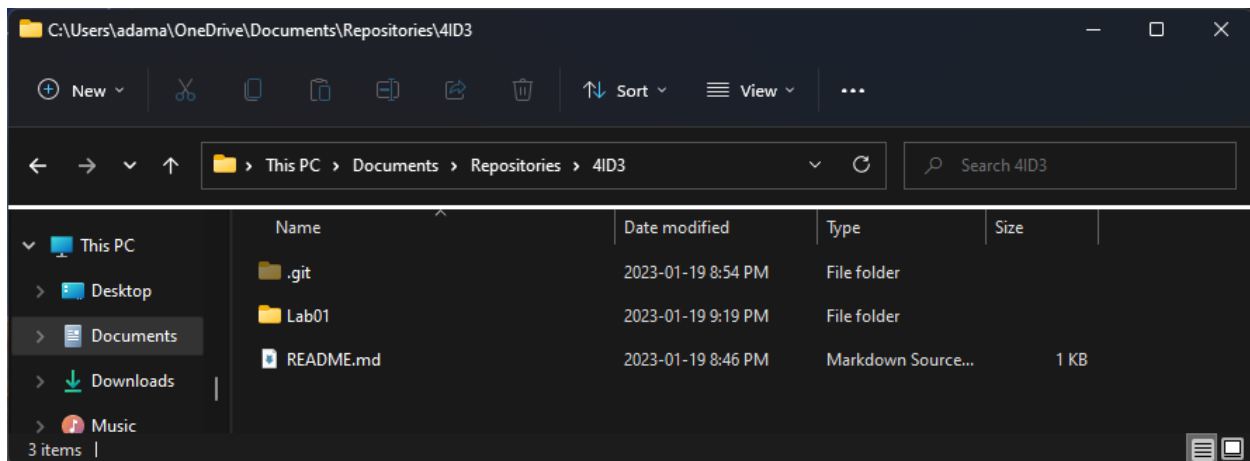
<h1> Lab 01 - WiFi and MQTT Communication </h1>

<h2> Date: 2023 - 01 - 19 </h2>

In this lab, we programmed an ESP8266 to communicate to a NodeRED client dashboard over WiFi using MQTT.

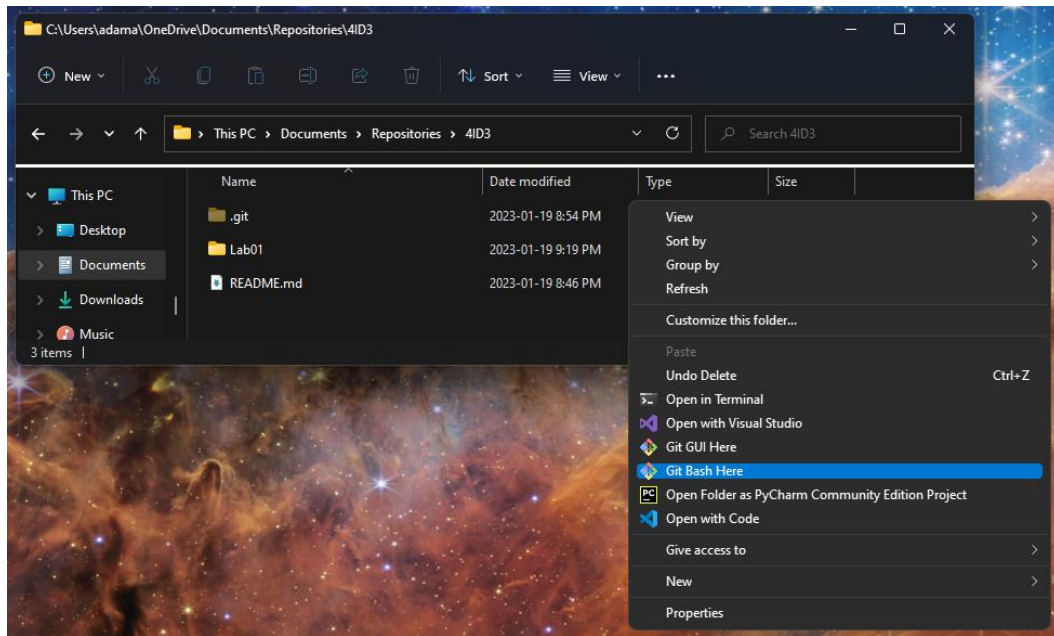
Save the file.

Navigate to the **root folder** (4ID3/, root main highest-level folder) of your local repository.

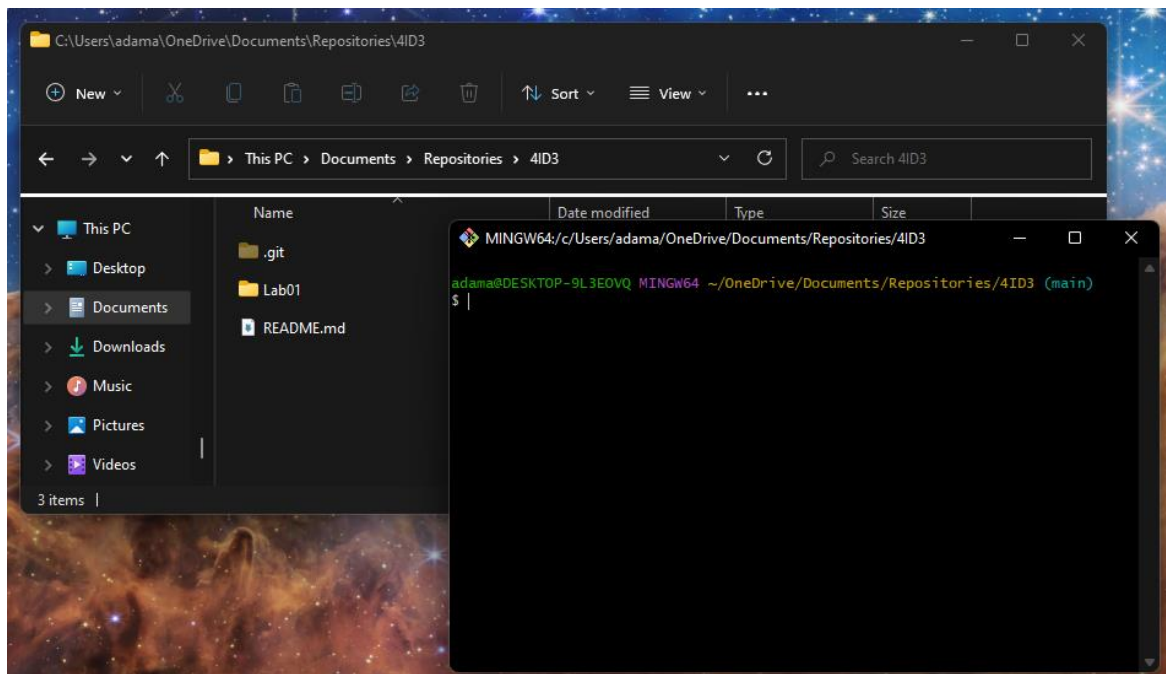


Right-click in the root folder of your local repository and launch **git bash**.

Lab 1 - Communicating Sensor Data over WiFi using MQTT



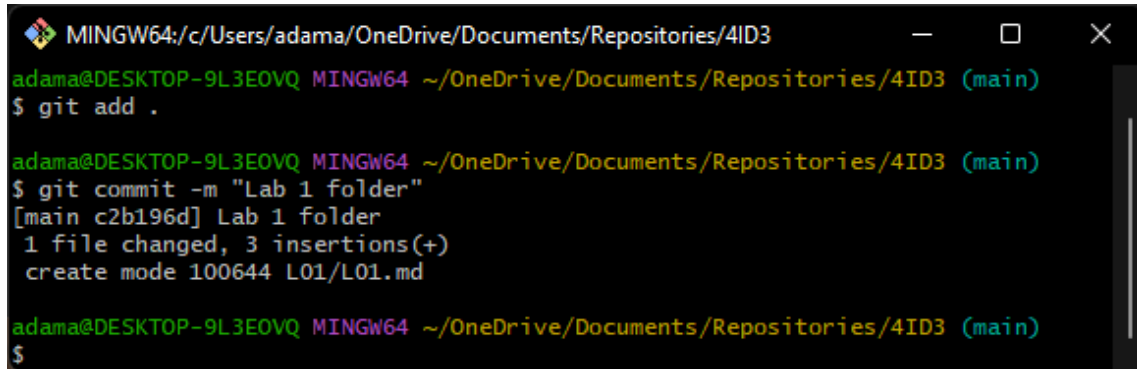
First, we need to add all the changes to the index that will be synced with GitHub. This will be done with the git add command.



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.

A terminal window titled 'MINGW64:/c/Users/adama/OneDrive/Documents/Repositories/4ID3' showing the execution of git commands. The user runs 'git add .' and then 'git commit -m "Lab 1 folder"'. The output shows the commit message '[main c2b196d] Lab 1 folder' and details that 1 file changed with 3 insertions, creating a new file 'L01/L01.md' with mode 100644.

```
MINGW64:/c/Users/adama/OneDrive/Documents/Repositories/4ID3
adama@DESKTOP-9L3EOVQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)
$ git add .

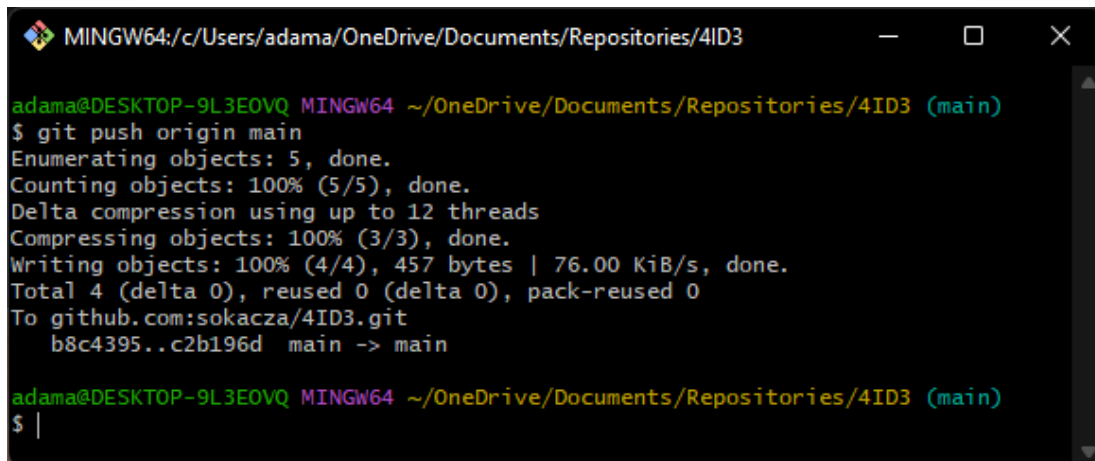
adama@DESKTOP-9L3EOVQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)
$ git commit -m "Lab 1 folder"
[main c2b196d] Lab 1 folder
1 file changed, 3 insertions(+)
create mode 100644 L01/L01.md

adama@DESKTOP-9L3EOVQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)
$
```

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.

A terminal window titled 'MINGW64:/c/Users/adama/OneDrive/Documents/Repositories/4ID3' showing the execution of 'git push origin main'. The output shows the process of enumerating objects, counting objects, compressing objects, and writing objects to the remote repository. It indicates that 4 objects were pushed, with 0 reused, and the commit 'b8c4395..c2b196d' was pushed to the 'main' branch on 'github.com:sokacza/4ID3.git'.

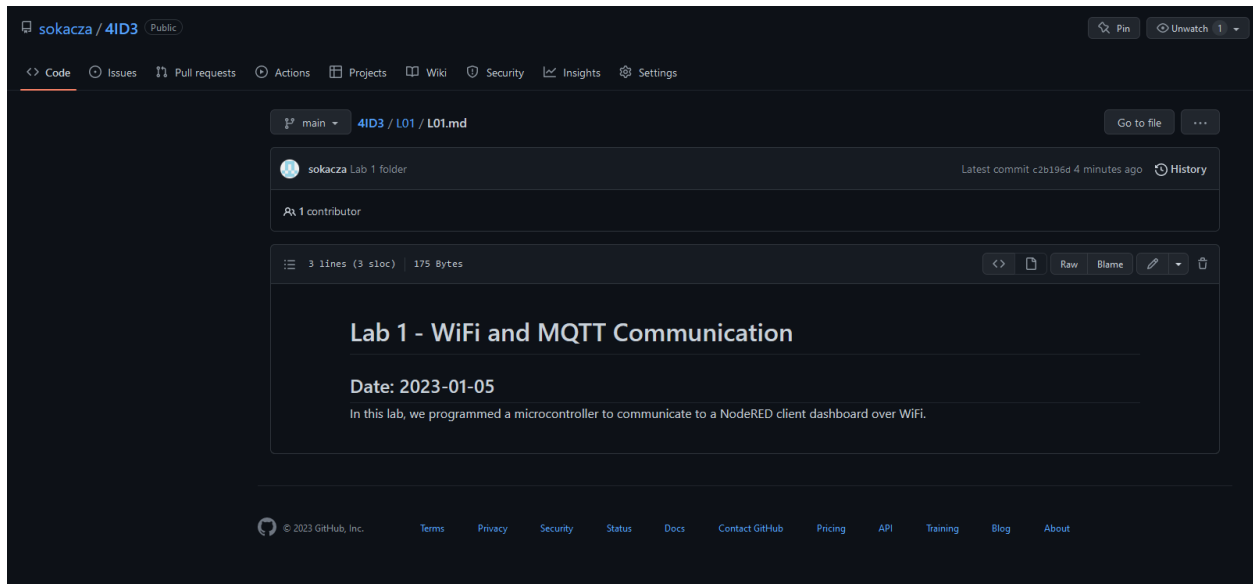
```
MINGW64:/c/Users/adama/OneDrive/Documents/Repositories/4ID3
adama@DESKTOP-9L3EOVQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), 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-9L3EOVQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)
$ |
```

git push origin main

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

Lab 1 - Communicating Sensor Data over WiFi using MQTT



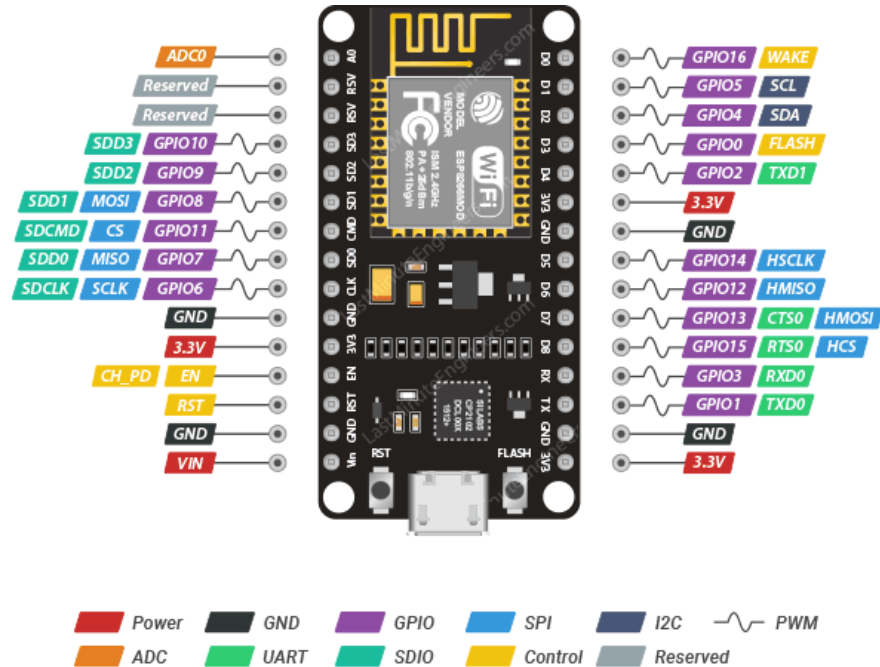
Now, if you are collaborating and wish to sync your local git repo with the remote GitHub repo, use the git pull command. In this case, we see that our local git repo is already up-to-date.

```
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

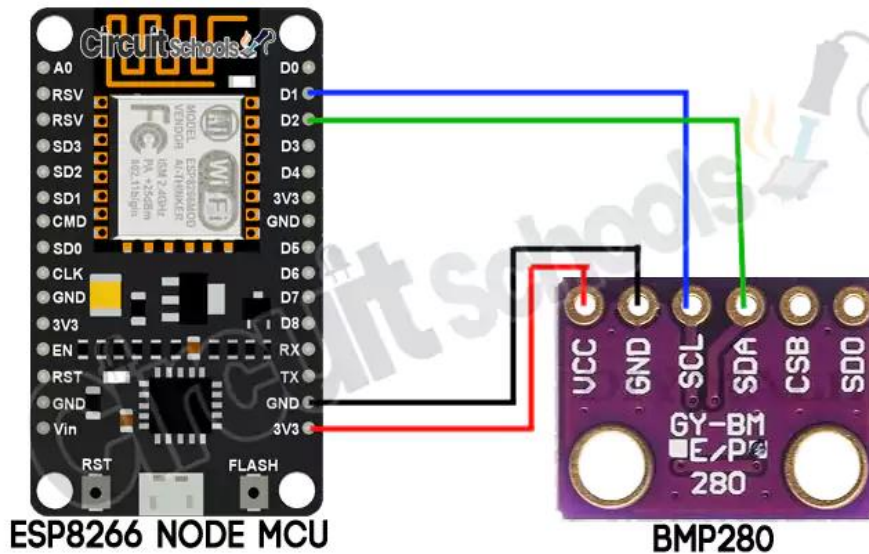
ESP8266 Development Board Pinout:



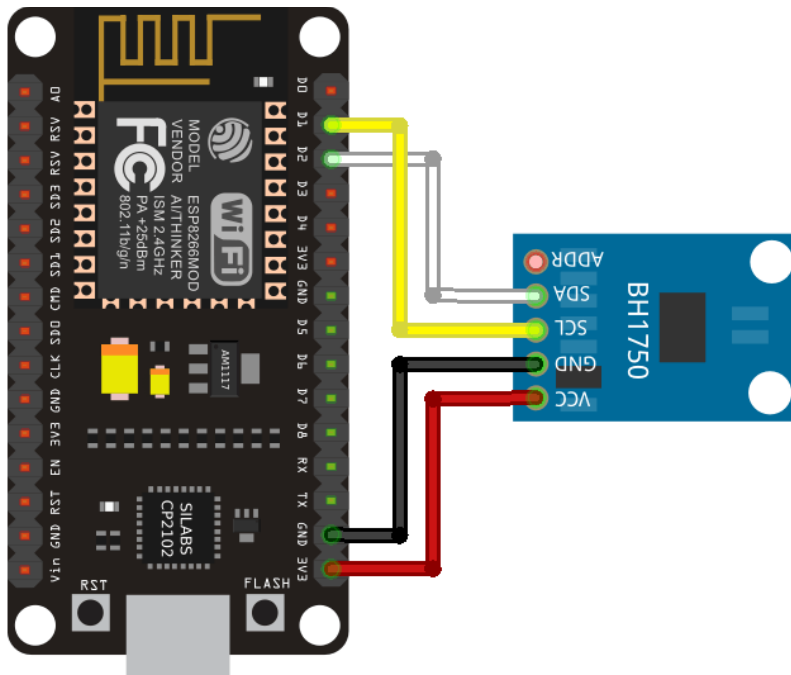
ESP8266 NodeMCU Pinout



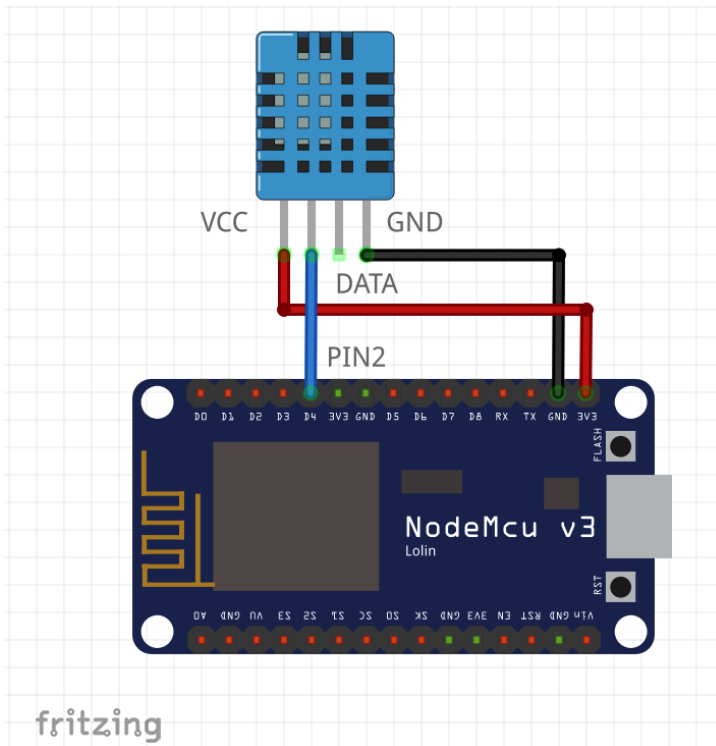
BMP180 Connection:



BH1750 Connection:



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

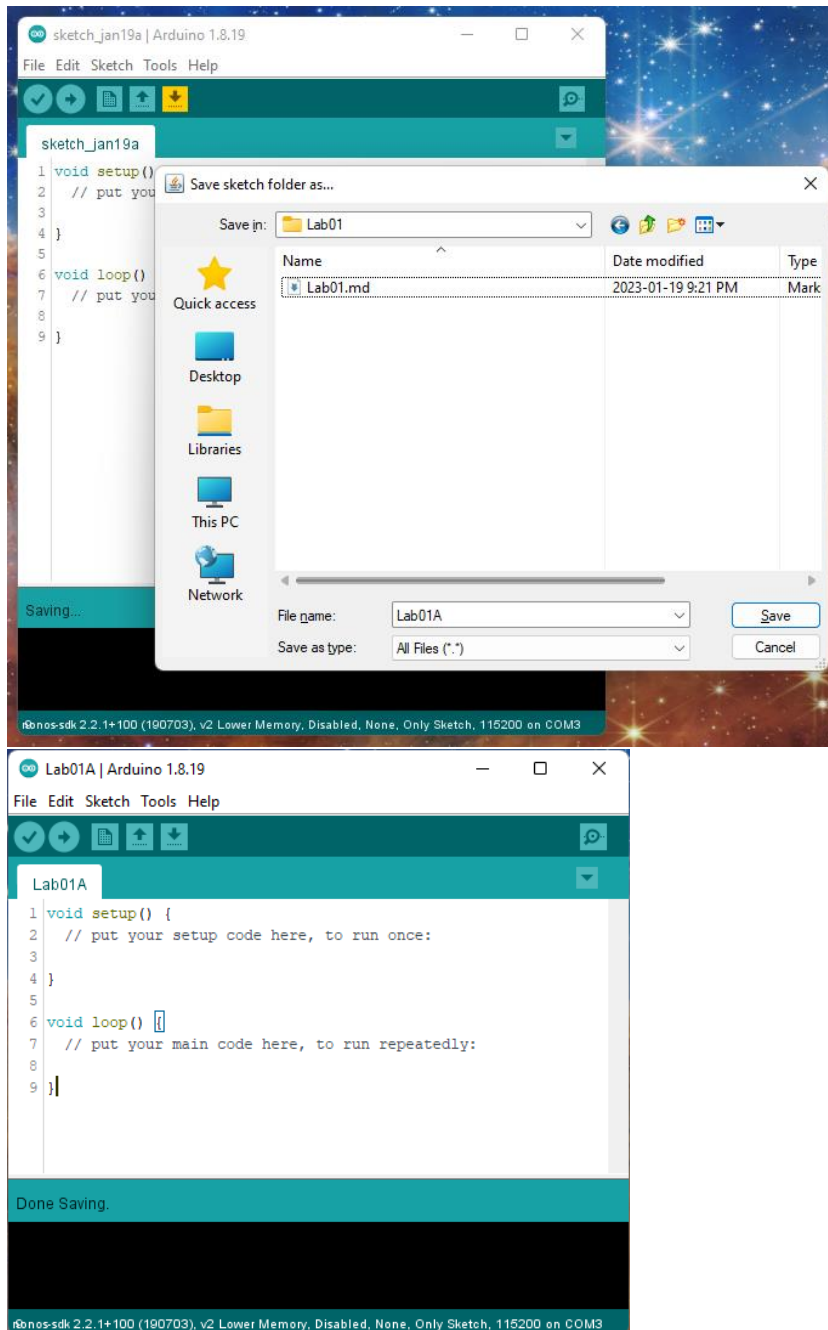


Reading Sensor Data

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.

The code for this step can be found here:

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**:

- 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

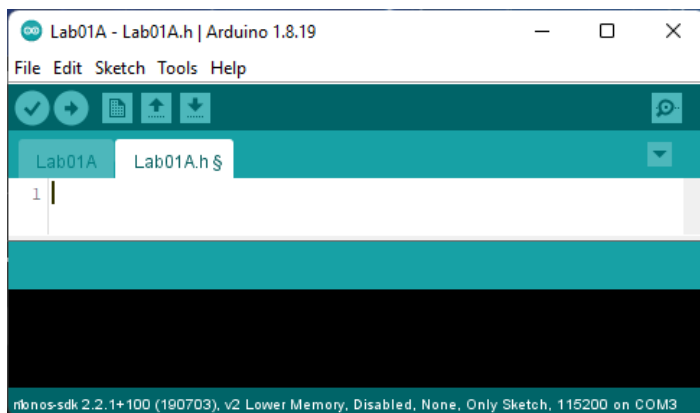
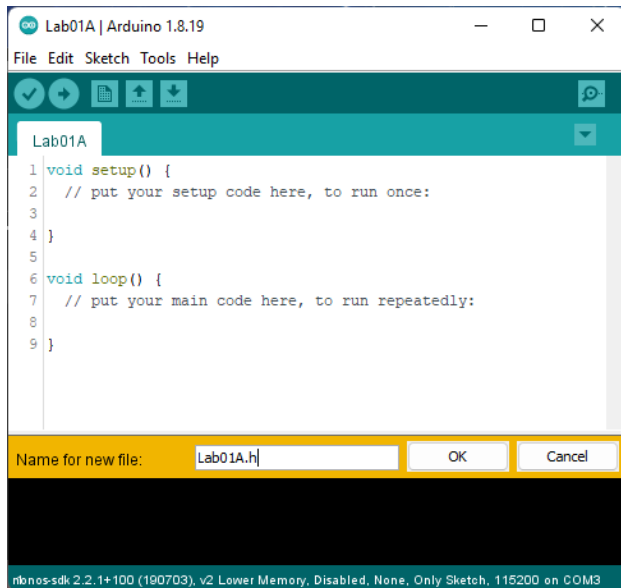
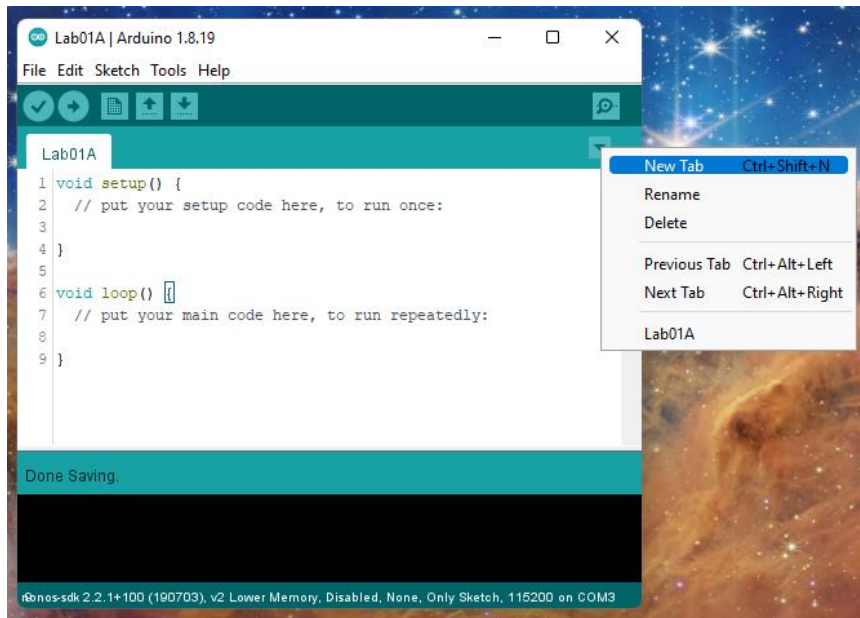
by Nick O’Leary Version 2.8.0 **INSTALLED**

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.

```
Lab01A Lab01A.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.

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

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

Firstly, include our header file.

```
Lab01A Lab01A.h
1 void setup() {
2   // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8
9 }
```

Next, set up the void setup() function.

```
Lab01A Lab01A.h
1 #include "Lab01A"
2
3 void setup() {
4   //Start the serial monitor at 115200 baud
5   Serial.begin(115200);
6
7   //Create a sensor object that is passed into the getSensor method of the dht class
8   //Only the dht sensor requires this
9   sensor_t sensor;
10  dht.temperature().getSensor(&sensor);
11  dht.humidity().getSensor(&sensor);
12
13  //Run the begin() method on each sensor to start communication
14  dht.begin();
15  bmp.begin();
16  BH1750.begin(BH1750_TO_GROUND);
17
18 }
19
```

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.

```
18 void loop() {
19
20     //Polling the DHT and BMP sensor using events
21     sensors_event_t dhtTempEvent, dhtHumEvent, bmpEvent;
22     dht.temperature().getEvent(&dhtTempEvent);
23     dht.humidity().getEvent(&dhtHumEvent);
24     bmp.getEvent(&bmpEvent);
25
26     //Polling the BH sensor
27     BH1750.start();
28     float lux=BH1750.getLux();
29 }
```

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

```
30 //Printing sensor readings to serial monitor
31 Serial.println("\n~");
32 (!isnan(dhtTempEvent.temperature)) ? Serial.println("Temperature: " + String(dhtTempEvent.temperature) + " degC") : Serial.println("Temperature Sensor Disconnected");
33
34 (!isnan(dhtHumEvent.relative_humidity)) ? Serial.println("Humidity: " + String(dhtHumEvent.relative_humidity) + " %") : Serial.println("Humidity Sensor Disconnected");
35
36 (!isnan(bmpEvent.pressure)) ? Serial.println("Pressure: " + String(bmpEvent.pressure) + " hPa") : Serial.println("Pressure Sensor Disconnected");
37
38 (!isnan(lux)) ? Serial.println("Light Intensity: " + String(lux) + " lux") : Serial.println("Lux Sensor Disconnected");
39
```

|

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.

Lab 1 - Communicating Sensor Data over WiFi using MQTT

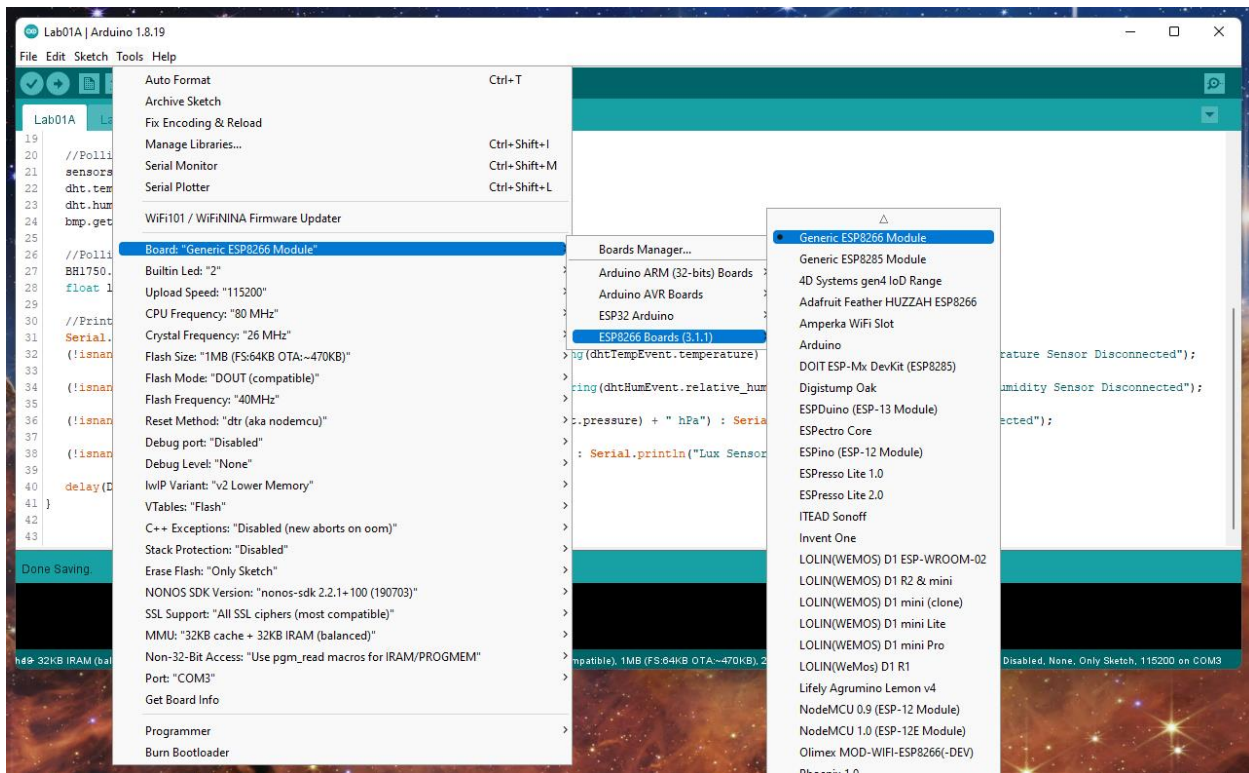
```
39  
40     delay(DELAY_BETWEEN_SAMPLES_MS);  
41 }  
42
```

It should look like the following:



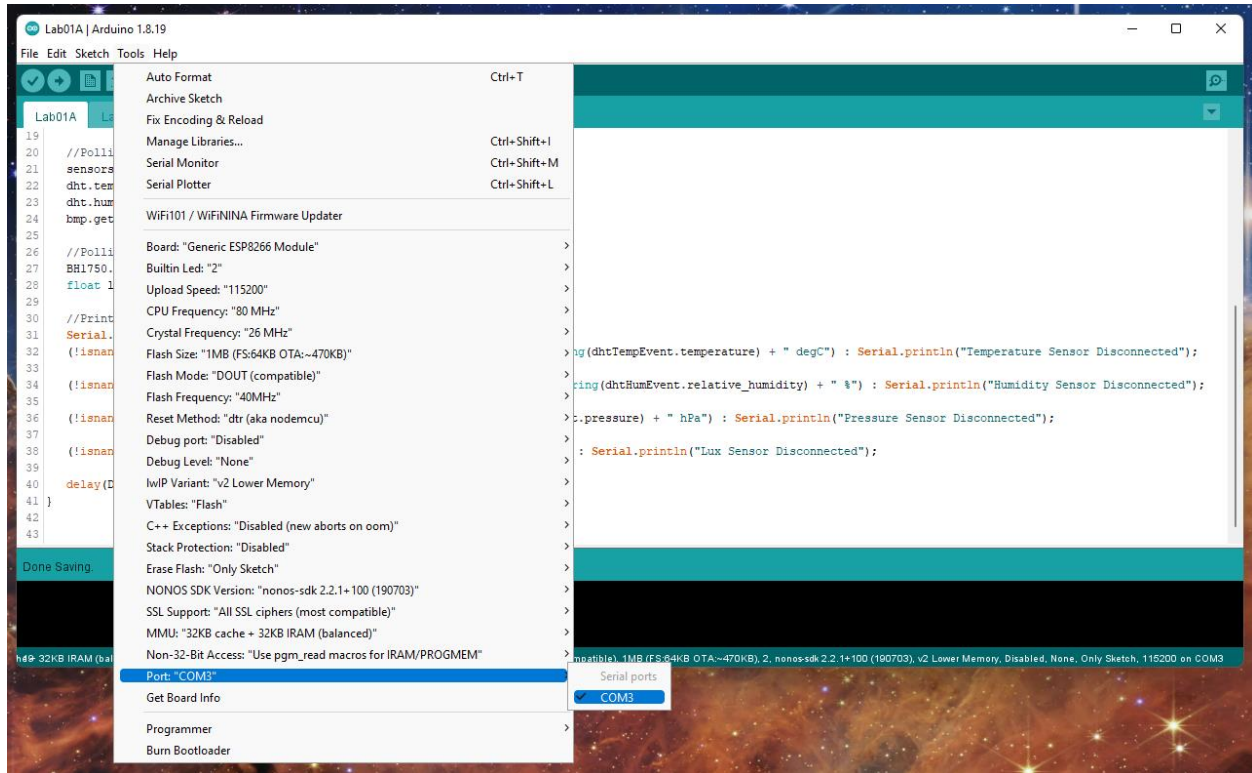
```
Lab01A$ Lab01A.h  
17  
18 void loop() {  
19     //Polling the DHT and BMP sensor using events  
20     sensors_event_t dhtTempEvent, dhtHumEvent, bmpEvent;  
21     dht.temperature().getEvent(&dhtTempEvent);  
22     dht.humidity().getEvent(&dhtHumEvent);  
23     bmp.getEvent(&bmpEvent);  
24  
25     //Polling the BH sensor  
26     BH1750.start();  
27     float lux=BH1750.getLux();  
28  
29     //Printing sensor readings to serial monitor  
30     Serial.println("\n");  
31     (!isnan(dhtTempEvent.temperature)) ? Serial.println("Temperature: " + String(dhtTempEvent.temperature) + " degC") : Serial.println("Temperature Sensor Disconnected");  
32  
33     (!isnan(dhtHumEvent.relative_humidity)) ? Serial.println("Humidity: " + String(dhtHumEvent.relative_humidity) + " %") : Serial.println("Humidity Sensor Disconnected");  
34  
35     (!isnan(bmpEvent.pressure)) ? Serial.println("Pressure: " + String(bmpEvent.pressure) + " hPa") : Serial.println("Pressure Sensor Disconnected");  
36  
37     (!isnan(lux)) ? Serial.println("Light Intensity: " + String(lux) + " lux") : Serial.println("Lux Sensor Disconnected");  
38  
39     delay(DELAY_BETWEEN_SAMPLES_MS);  
40 }  
41
```

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

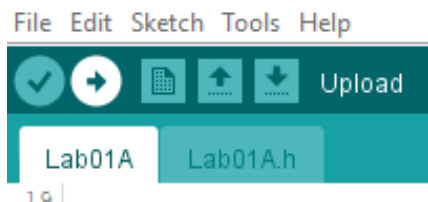


Lab 1 - Communicating Sensor Data over WiFi using MQTT

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



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

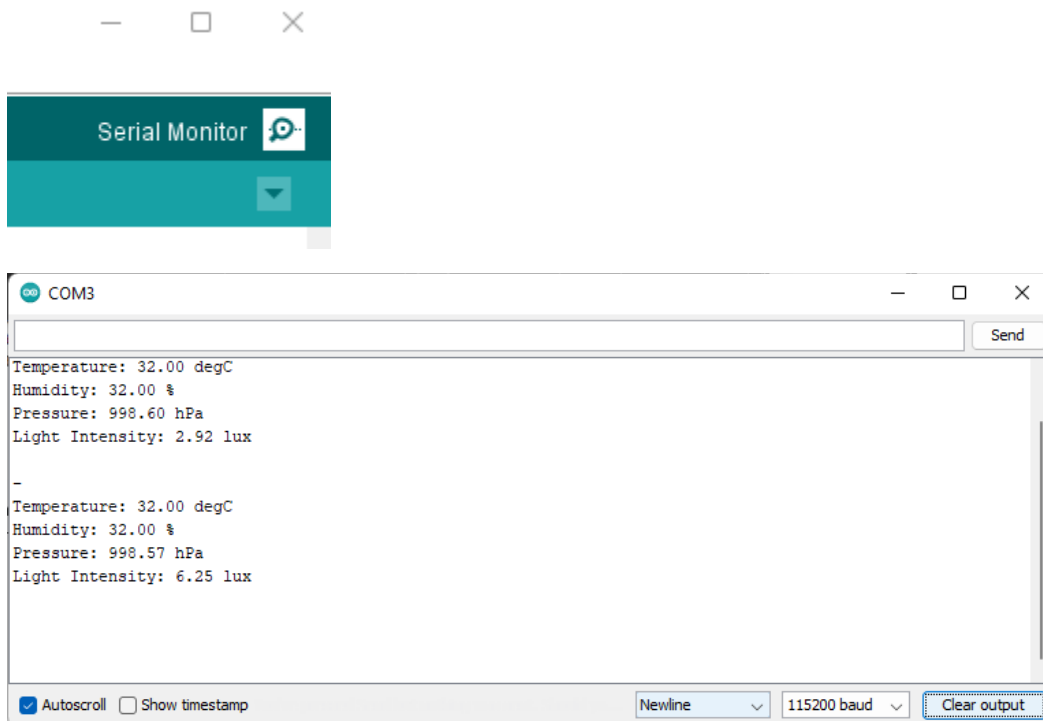


A successful upload should look like this:

```
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...
```

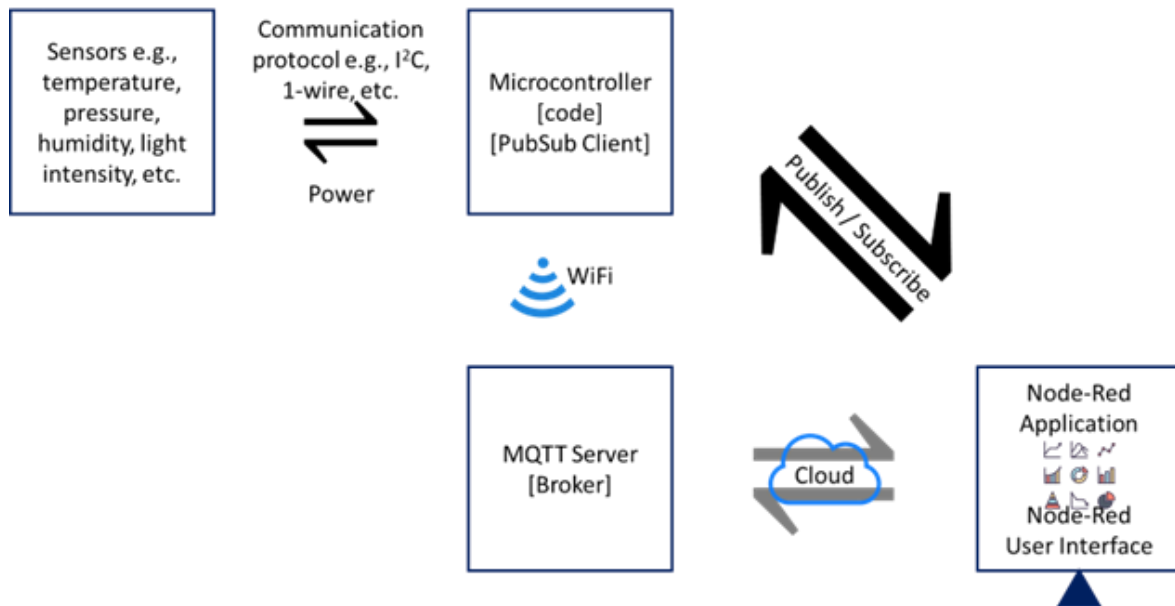
After the upload is complete (NOT DURING), launch the **Serial monitor** to view the microcontroller output.



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

Connecting to the Mosquitto Public Test Broker

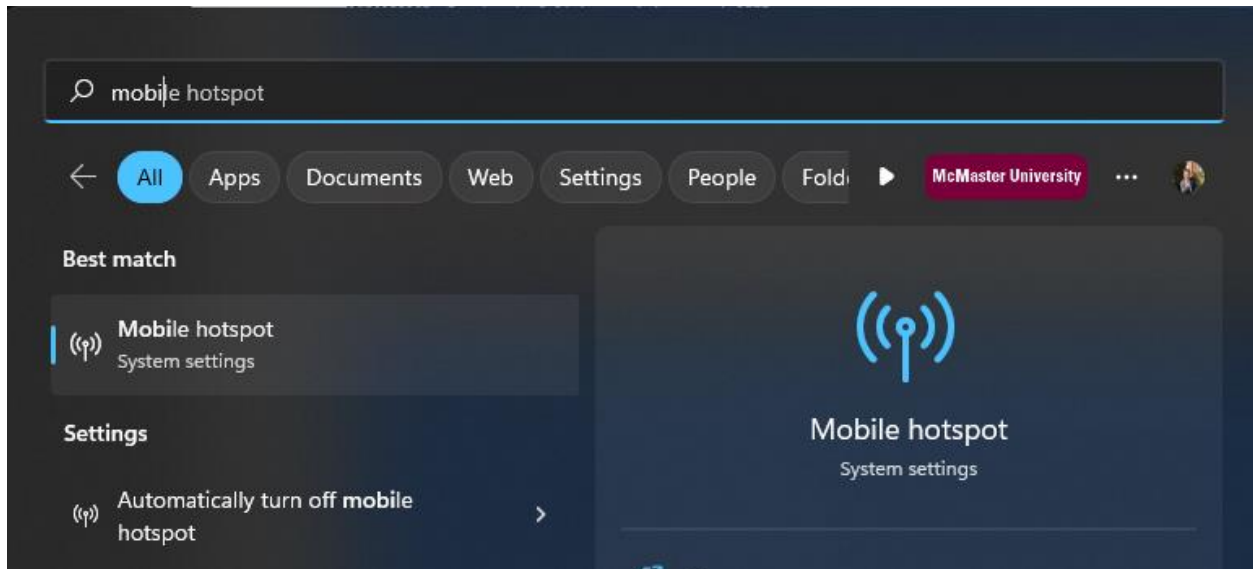
Use the IP address of a known public broker such as Mosquitto's test broker:

test.mosquitto.org

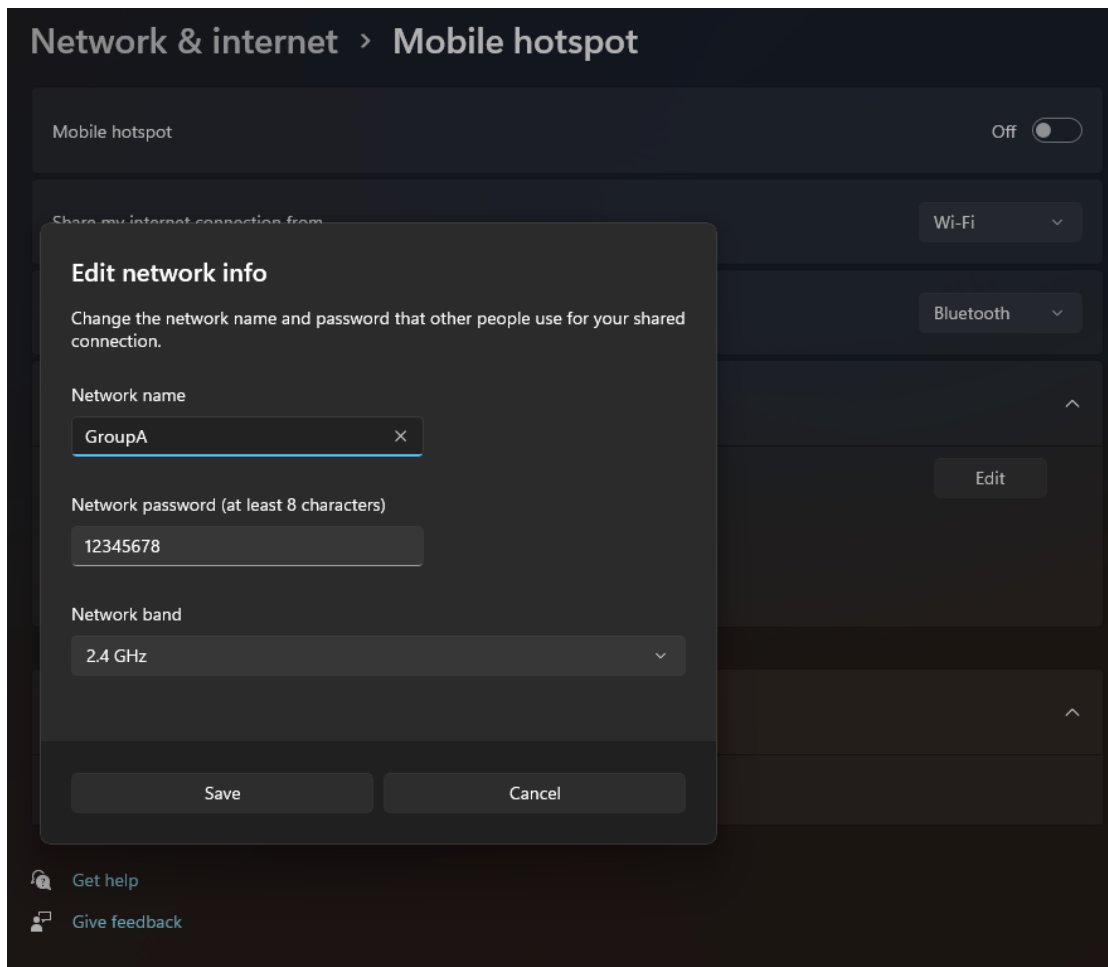
Port is 1883

Hotspotting your Microcontroller

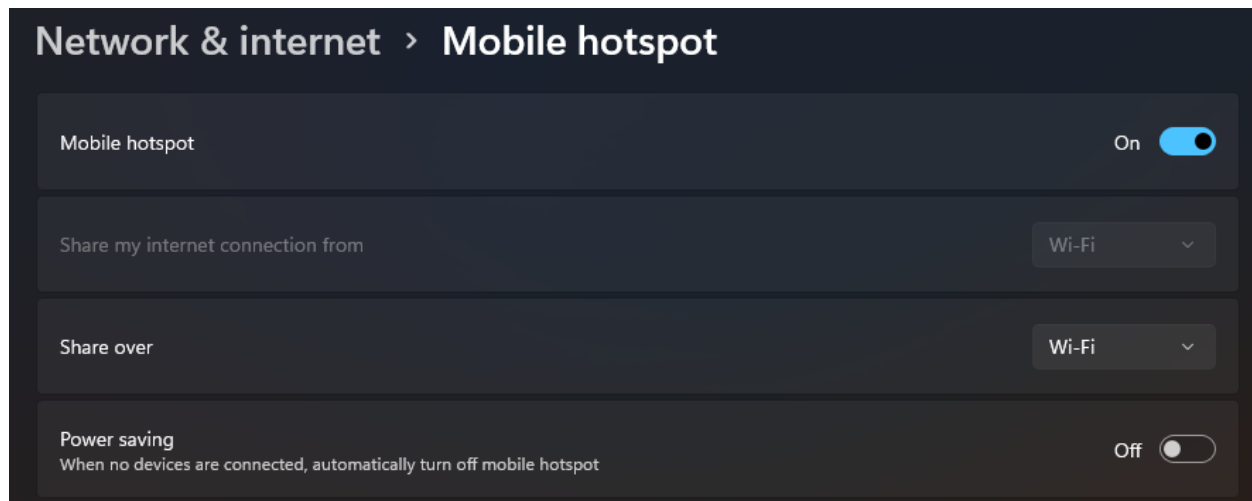
Open **Windows Mobile Hotspot**.



Set your login credentials.

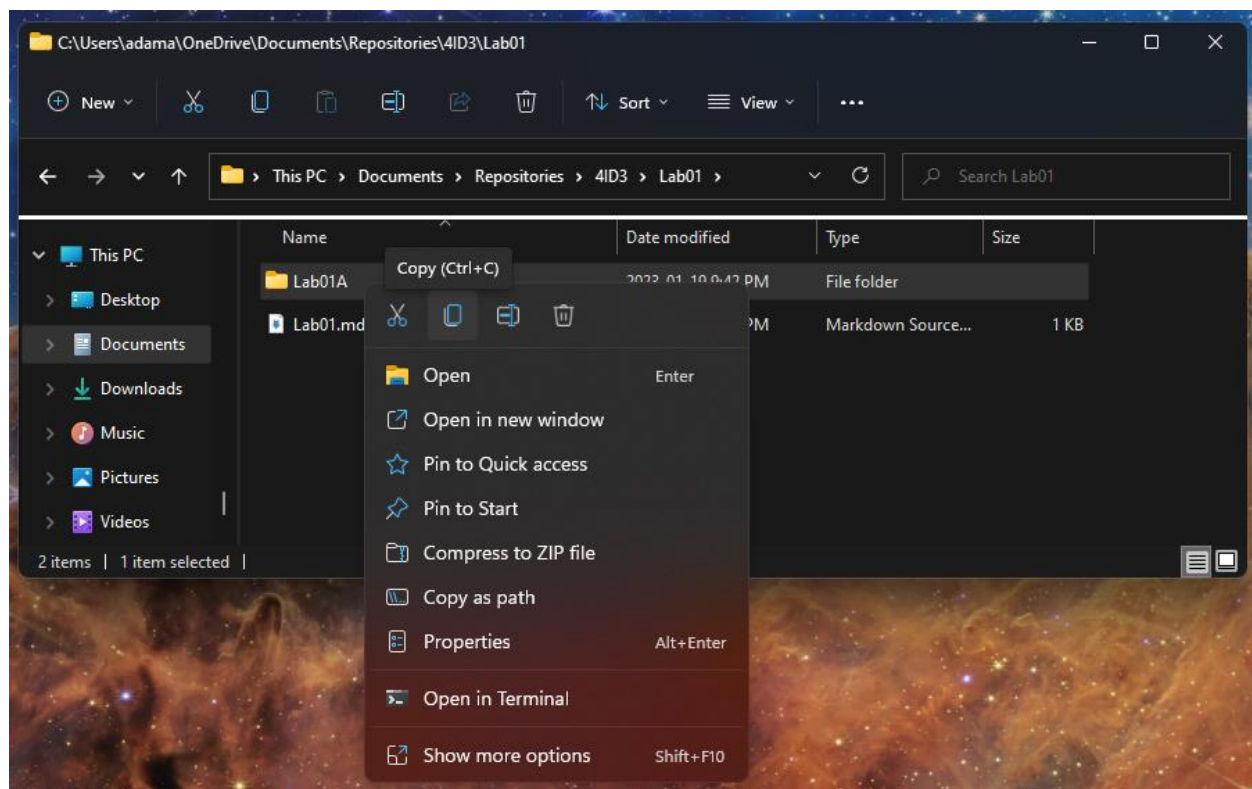


Toggle it **on** and **disable power saving**.

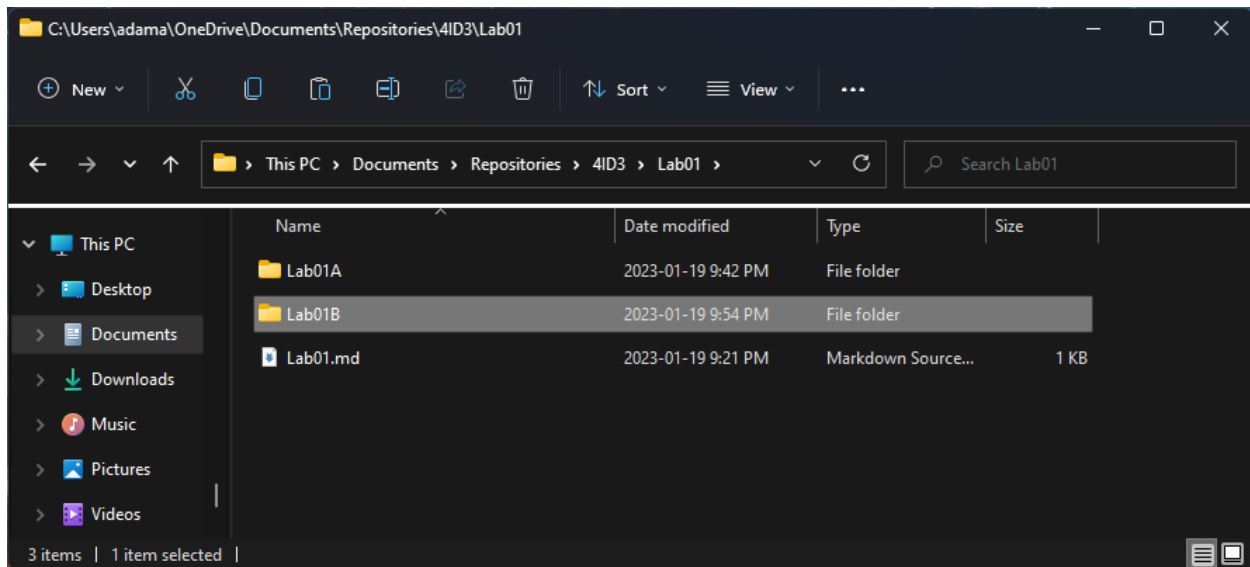


Implementing MQTT

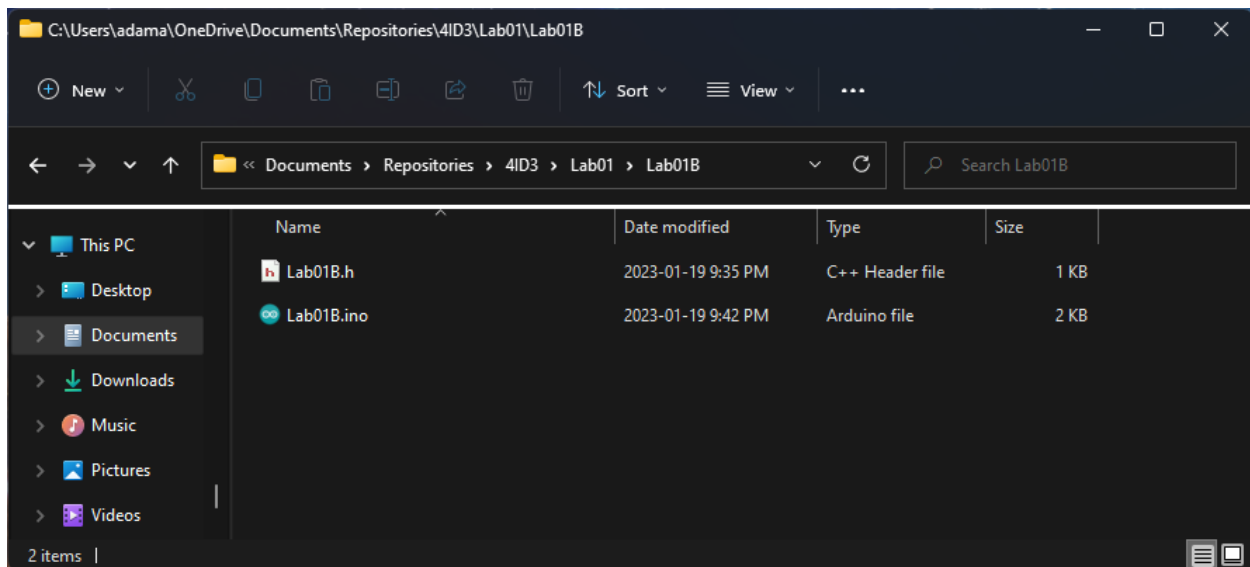
Duplicate your project by **Copying Lab01A/** and **renaming** it to **Lab01B/**.



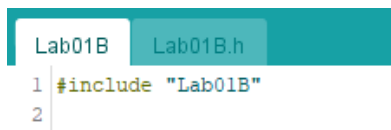
Lab 1 - Communicating Sensor Data over WiFi using MQTT



Rename the Arduino files to reflect the new project name **Lab01B**.



Change the include to **Lab01B**.



Lab01B	Lab01B.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 = "GroupA";
25 char* pass = "12345678";
26 const char* brokerAddress = "test.mosquitto.org";
27 uint16_t addressPort = 1883;
28
29 //Instantiate Sensor Objects
30 DHT_Unified dht(DHTPIN, DHTTYPE);
31 Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);
32 hp_BH1750 BH1750;
33
34 //Instantiate MQTT Client
35 WiFiClient espClient;
36 PubSubClient client(espClient);
37

```

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

```
Lab01B Lab01B.h
1 #include "Lab01B.h"
2
3 void setup() {
4     //Start the serial monitor at 115200 baud
5     Serial.begin(115200);
6
7     //Create a sensor object that is passed into the getSensor method of the dht class
8     //Only the dht sensor requires this
9     sensor_t sensor;
10    dht.temperature().getSensor(&sensor);
11    dht.humidity().getSensor(&sensor);
12
13    //Run the begin() method on each sensor to start communication
14    dht.begin();
15    bmp.begin();
16    BH1750.begin(BH1750_TO_GROUND);
17
18    //Start the WiFi driver and tell it to connect to your local network
19    //WiFi.mode(WIFI_STA);
20    WiFi.begin(ssid, pass);
21
22    //While it is connecting, print a '.' to the serial monitor every 500 ms
23    while (WiFi.status() != WL_CONNECTED) {
24        delay(500);
25        Serial.print(".");
26    }
27
28    //Once connected, print the local IP address
29    Serial.println("");
30    Serial.println("WiFi connected");
31    Serial.println("IP address: ");
32    Serial.println(WiFi.localIP());
33
34    //Set the MQTT client to connect to the desired broker
35    client.setServer(brokerAddress, addressPort);
36
37 }
38
```

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

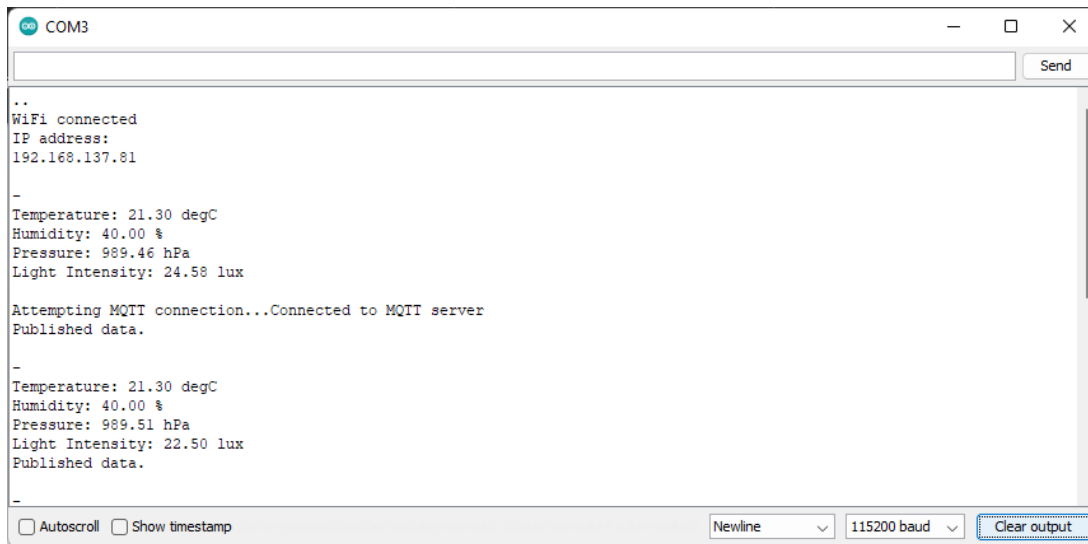
```
40
41 void reconnect() {
42
43     //While the client remains unconnected from the MQTT broker, attempt to reconnect every 2 seconds
44     //Also, print diagnostic information
45     while (!client.connected()) {
46         Serial.print("Attempting MQTT connection...");
47
48         if (client.connect("ESP8266Client")) {
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.

```
Lab01B$ Lab01B.h
56
57 void loop() {
58
59     //Polling the DHT and BMP sensor using events
60     sensors_event_t dhtTempEvent, dhtHumEvent, bmpEvent;
61     dht.temperature().getEvent(&dhtTempEvent);
62     dht.humidity().getEvent(&dhtHumEvent);
63     bmp.getEvent(&bmpEvent);
64
65     //Polling the BH sensor
66     BH1750.start();
67     float lux=BH1750.getLux();
68
69     //Printing sensor readings to serial monitor
70     Serial.println("\n-");
71     (!isnan(dhtTempEvent.temperature)) ? Serial.println("Temperature: " + String(dhtTempEvent.temperature) + " degC") : Serial.println("Temperature Sensor Disconnected");
72
73     (!isnan(dhtHumEvent.relative_humidity)) ? Serial.println("Humidity: " + String(dhtHumEvent.relative_humidity) + " %") : Serial.println("Humidity Sensor Disconnected");
74
75     (!isnan(bmpEvent.pressure)) ? Serial.println("Pressure: " + String(bmpEvent.pressure) + " hPa") : Serial.println("Pressure Sensor Disconnected");
76
77     (!isnan(lux)) ? Serial.println("Light Intensity: " + String(lux) + " lux") : Serial.println("Lux Sensor Disconnected");
78
79     //If the client disconnects from the MQTT broker, attempt to reconnect
80     if (!client.connected()) {
81         reconnect();
82     }
83     if(!client.loop())
84         client.connect("ESP8266Client");
85
86     //Publish the sensor data to the associated topics
87     client.publish("4ID3_GroupA/temperature", String(dhtTempEvent.temperature).c_str());
88     delay(100);
89     client.publish("4ID3_GroupA/humidity", String(dhtHumEvent.relative_humidity).c_str());
90     delay(100);
91     client.publish("4ID3_GroupA/pressure", String(bmpEvent.pressure).c_str());
92     delay(100);
93     client.publish("4ID3_GroupA/light", String(lux).c_str());
94     Serial.println("Published data.");
95
96     delay(DELAY_BETWEEN_SAMPLES_MS);
97 }
98
```

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 terminal window titled "COM3". The window has a text input field at the top with a "Send" button. The main area displays the following text:

```
..
WiFi connected
IP address:
192.168.137.81

-
Temperature: 21.30 degC
Humidity: 40.00 %
Pressure: 989.46 hPa
Light Intensity: 24.58 lux

Attempting MQTT connection...Connected to MQTT server
Published data.

-
Temperature: 21.30 degC
Humidity: 40.00 %
Pressure: 989.51 hPa
Light Intensity: 22.50 lux
Published data.

-
```

At the bottom of the window, there are checkboxes for "Autoscroll" and "Show timestamp", a "Newline" dropdown menu, a "115200 baud" dropdown menu, and a "Clear output" button.

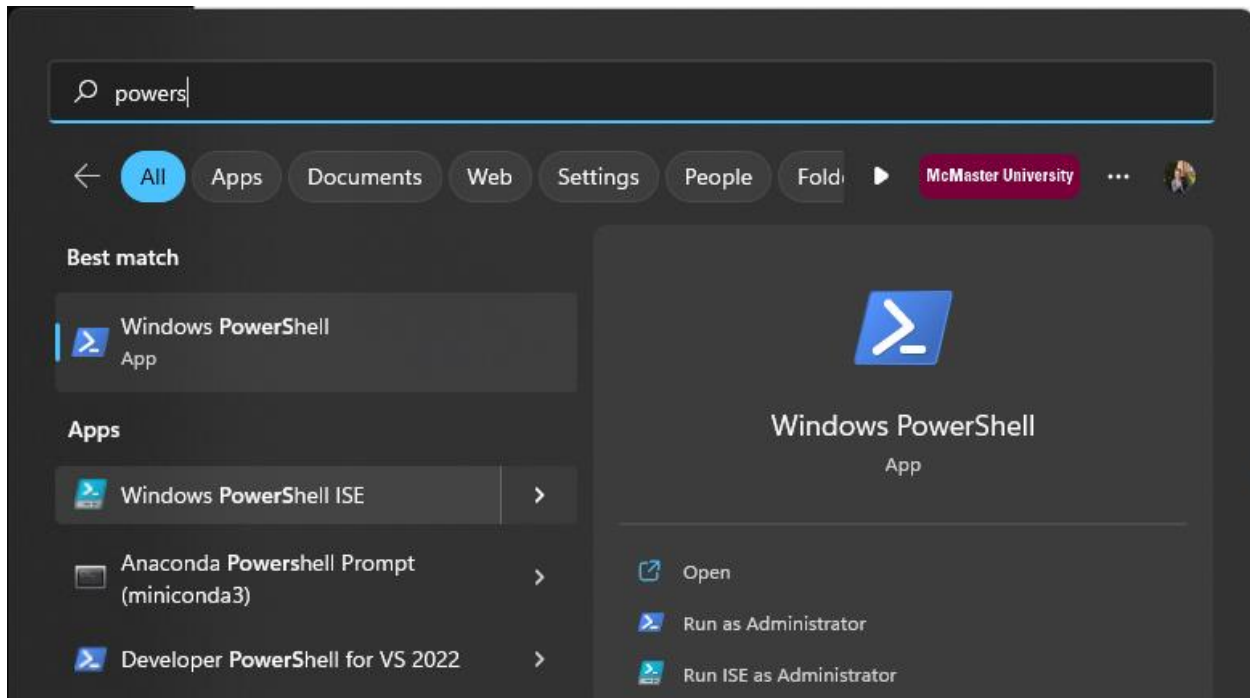
Verifying Connection

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

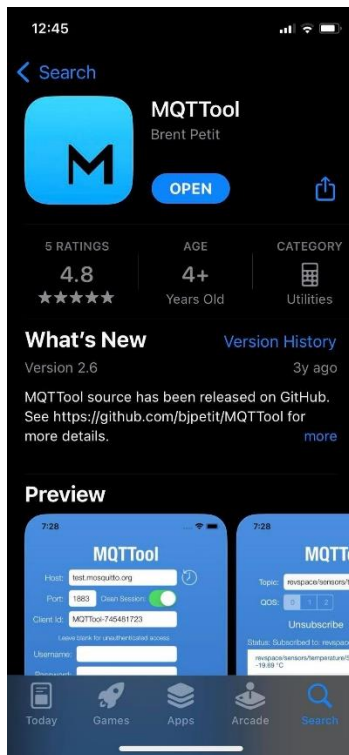
```
Windows PowerShell

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.

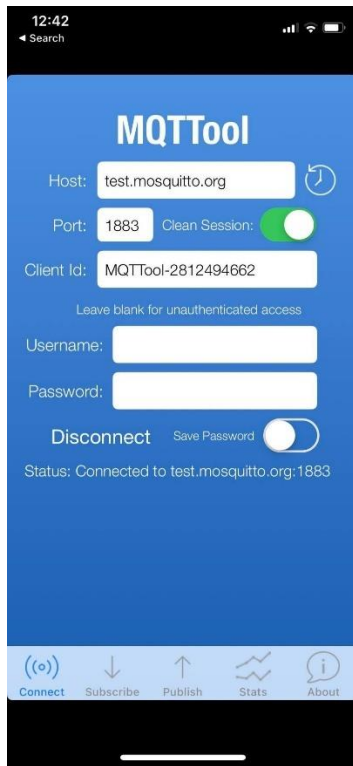
Using a Mobile Application

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

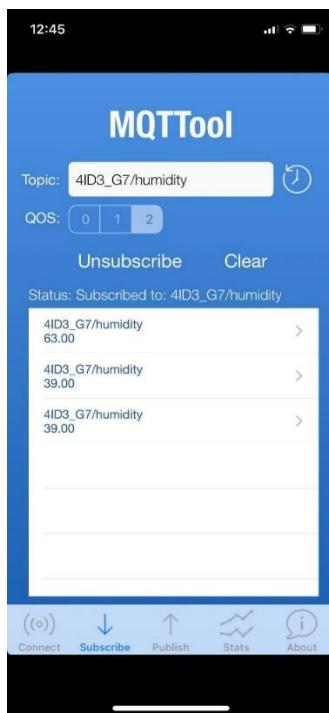


Next, connect to the public mosquitto broker:

Lab 1 - Communicating Sensor Data over WiFi using MQTT



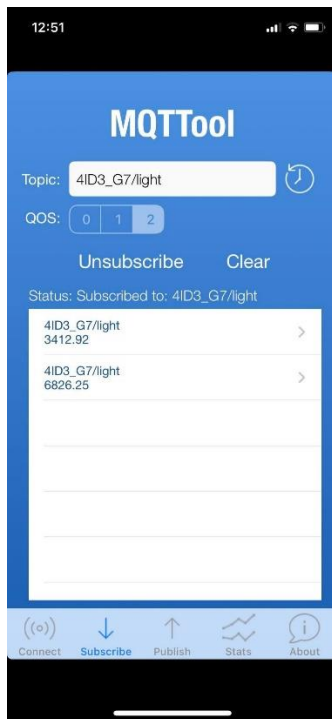
Next, subscribe to the topic of choice.



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.

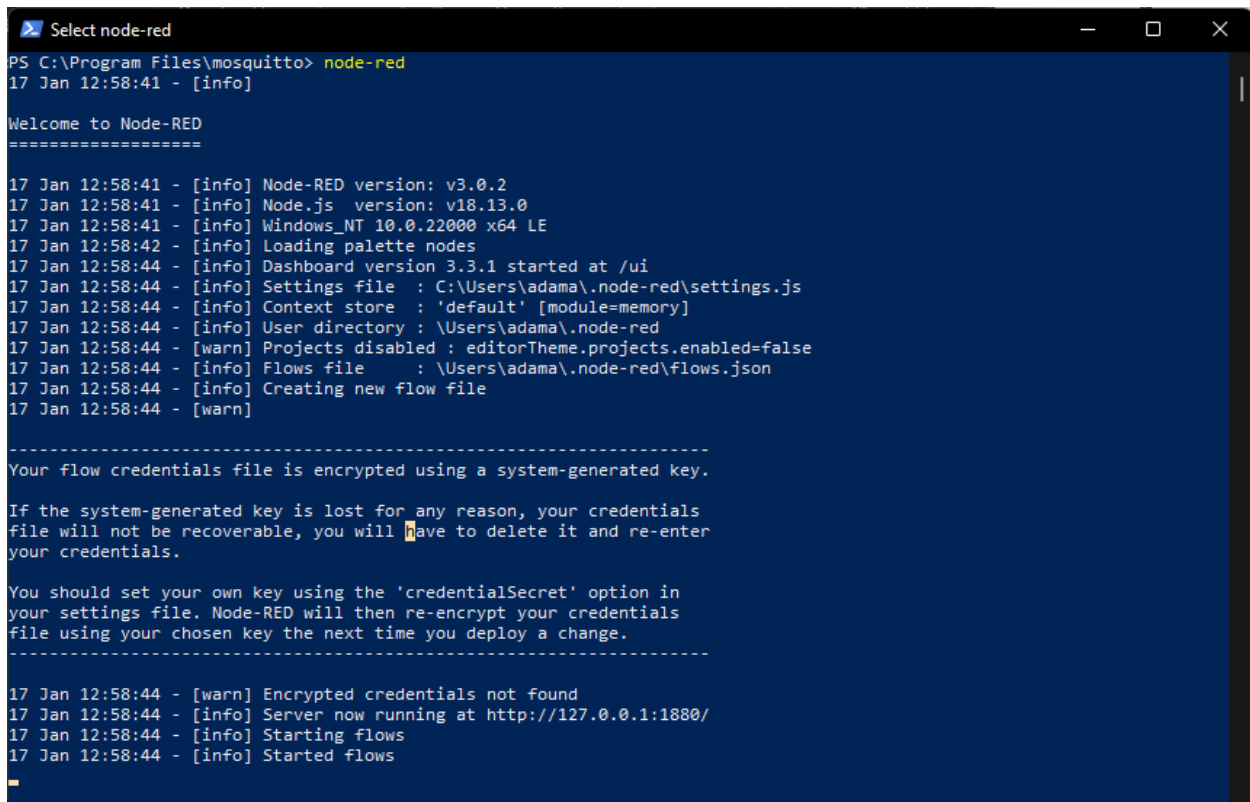
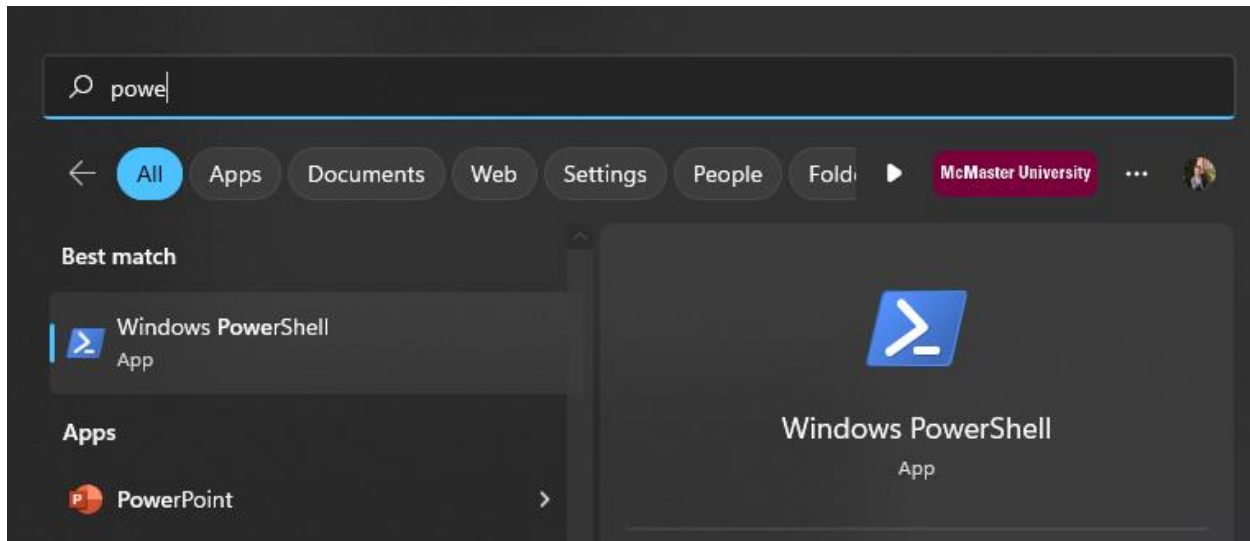


NodeRED Dashboard

Please follow the pre-lab instructions to install NodeRED.

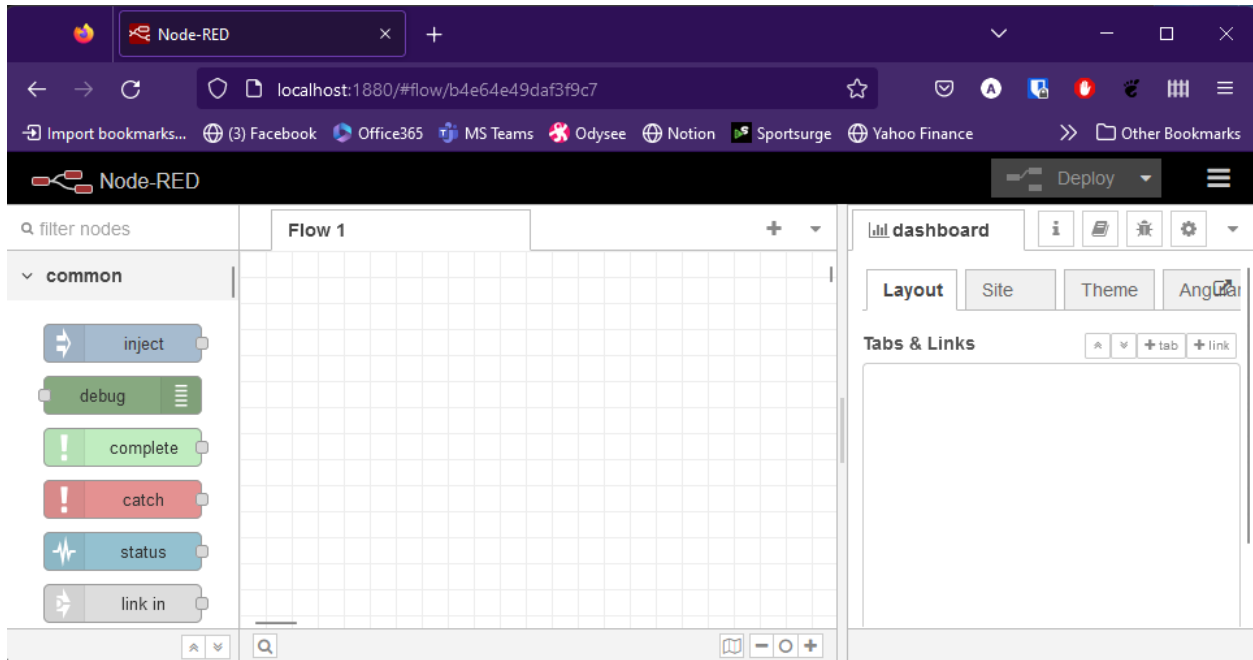
To start NodeRED, open Powershell and type the command:

`node-red`

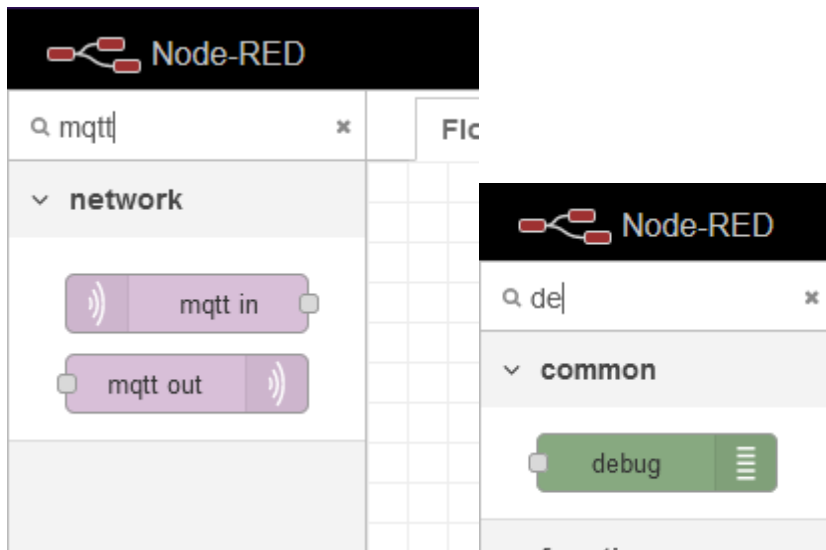


Navigate to the URL presented, in a web browser.

Lab 1 - Communicating Sensor Data over WiFi using MQTT

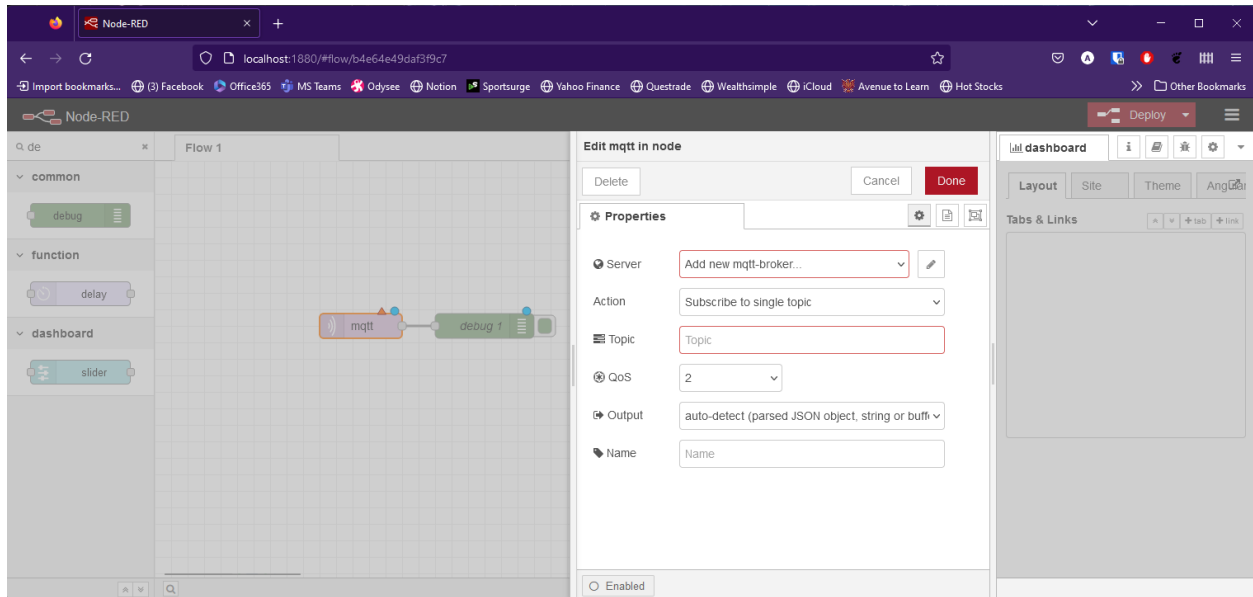


Filter nodes to find the **mqtt in** node and the **debug** node. Drag them into your flow diagram.



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

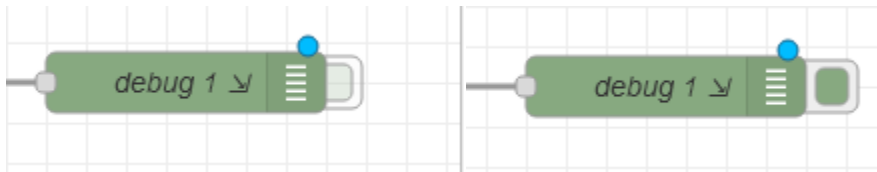
Topic 4ID3_GroupA/humidity

QoS 0

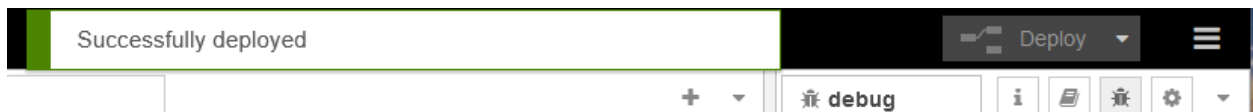
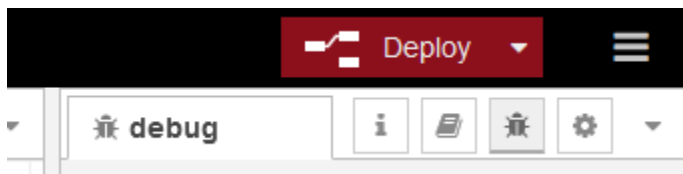
Output auto-detect (parsed JSON object, string or buffer)

Name Name

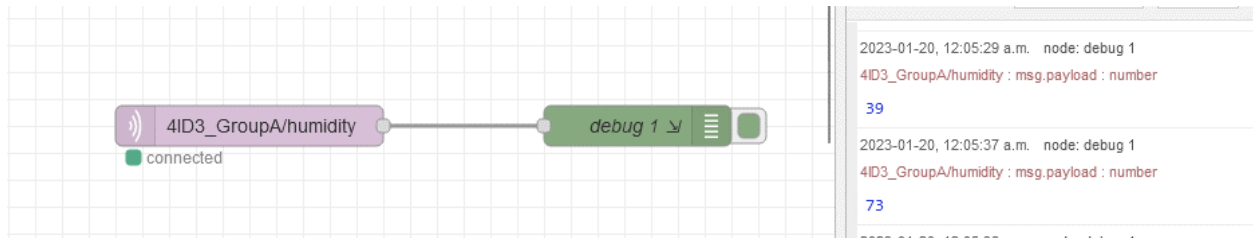
Enable the **debug** node by pressing the **green box**.



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



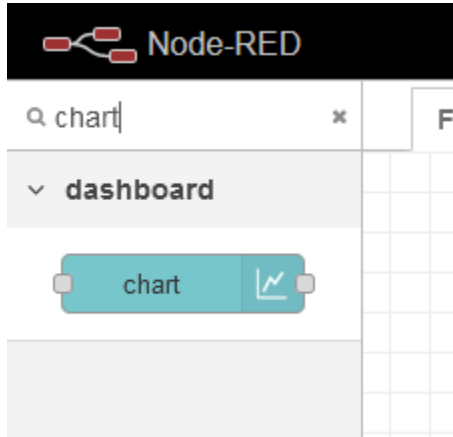
Lab 1 - Communicating Sensor Data over WiFi using MQTT



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 information box at the bottom explains that the 'Icon' field 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.

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

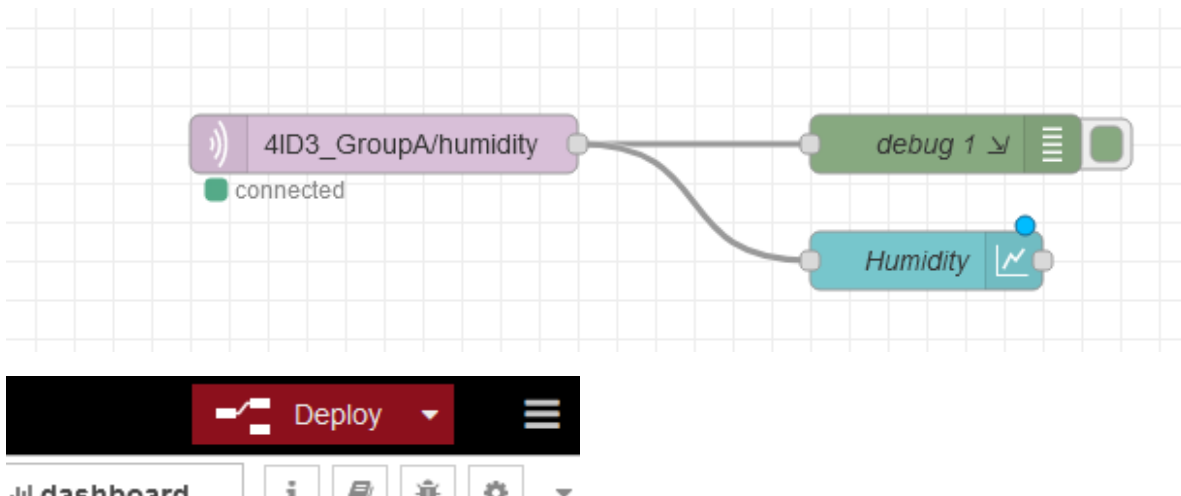
Class

Optional CSS class name(s) for widget

Name

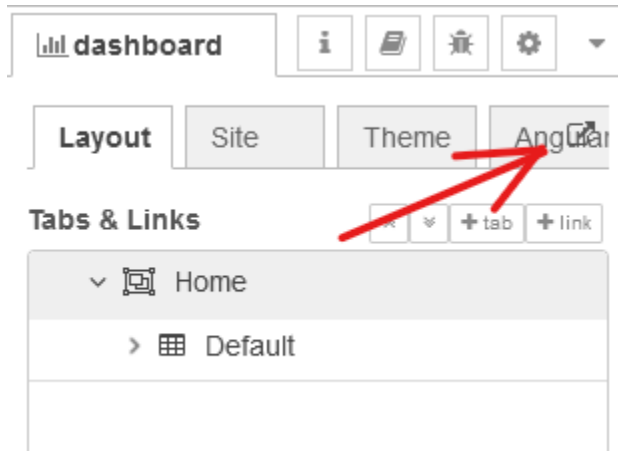
Humidity

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

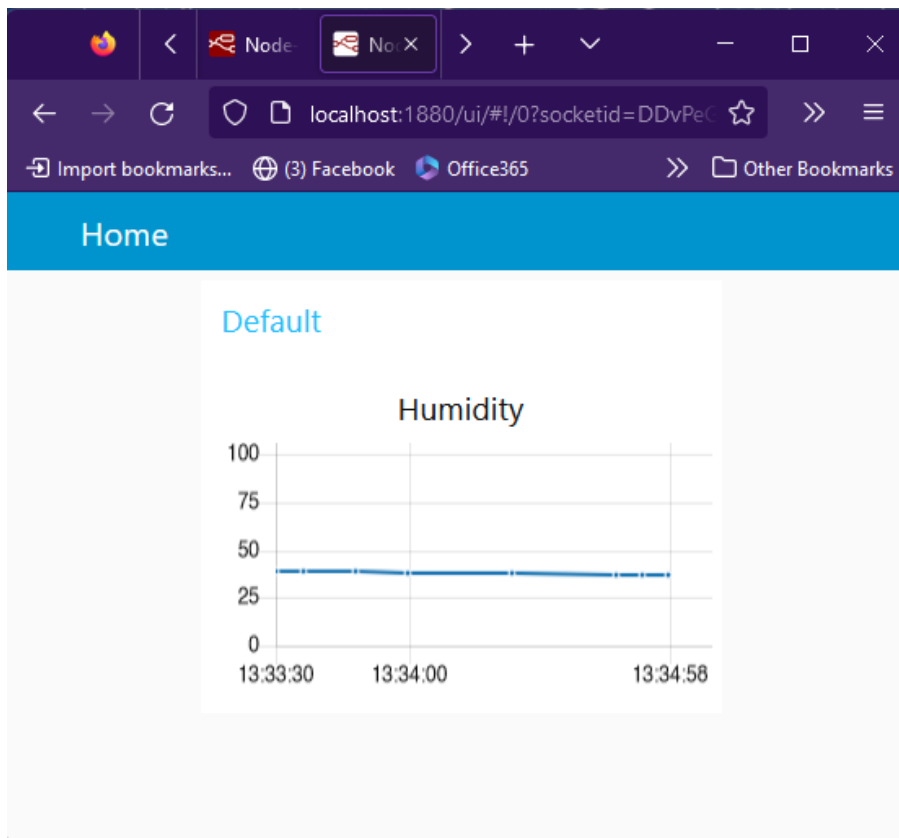


To view the dashboard, either append **ui/** to your url (<http://localhost:1880/ui>) or press the **open dashboard** icon in the top right corner of the dashboard panel.

Lab 1 - Communicating Sensor Data over WiFi using MQTT



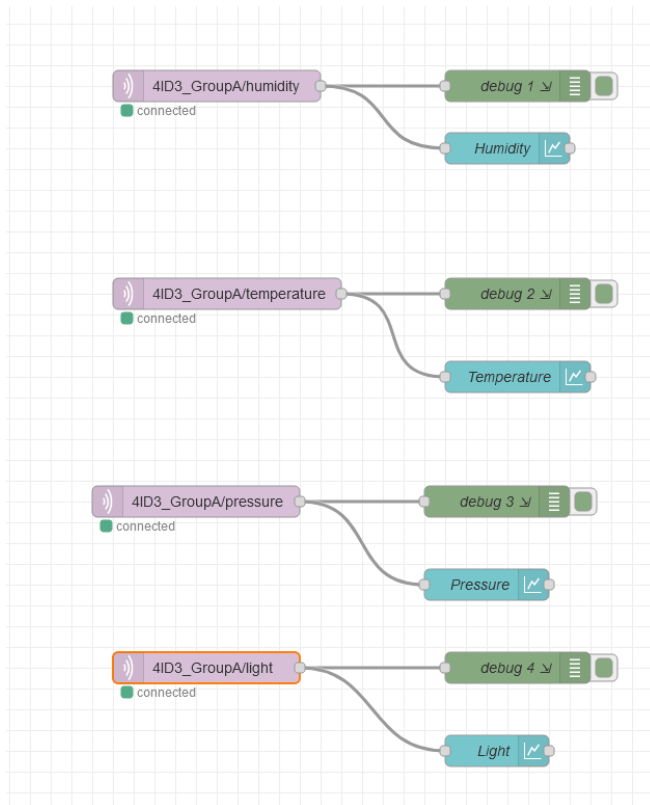
Wait for data to populate.



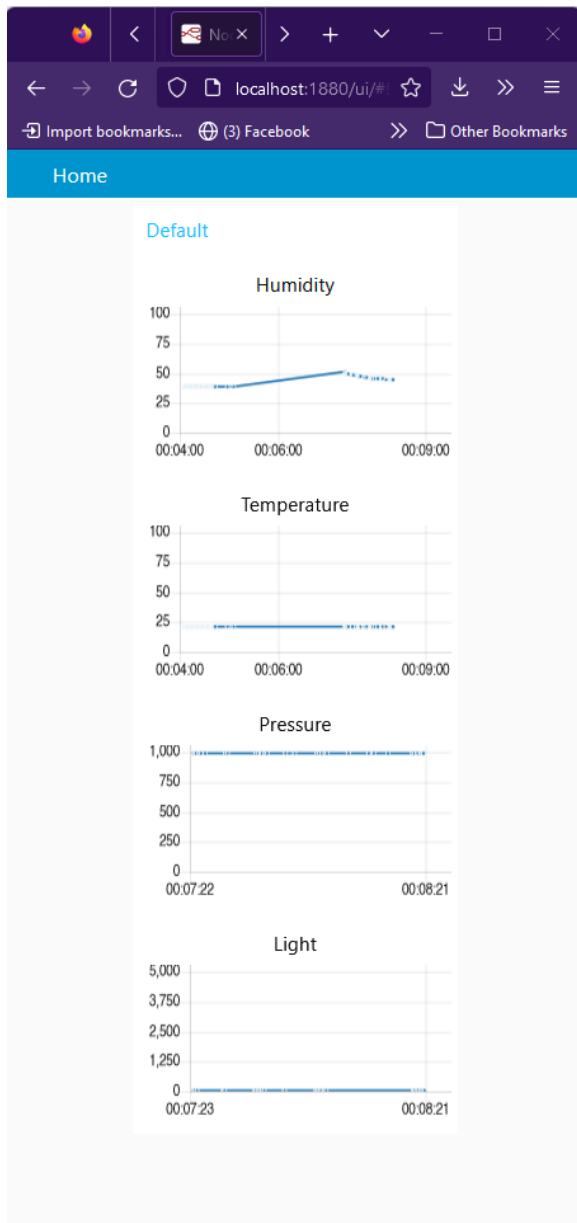
Now that you have seen how to visualize one topic, attempt to visualize the rest.

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

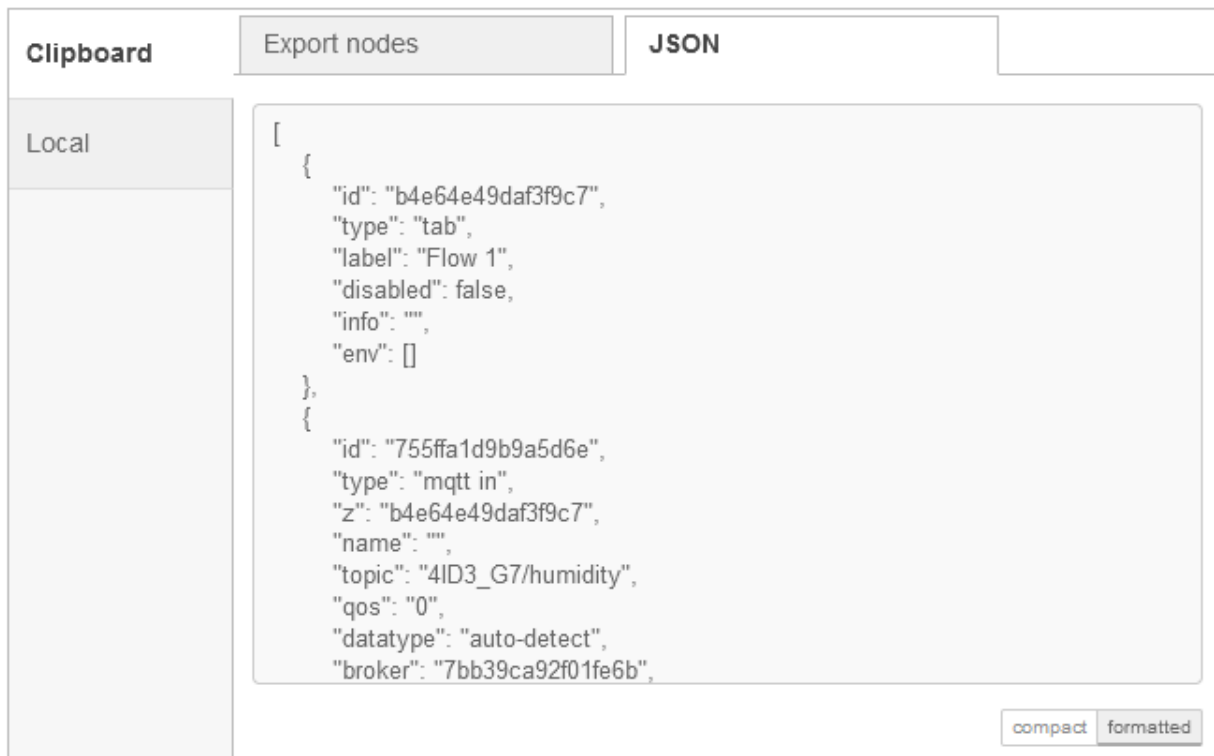
Exporting a NodeRED Flow as JSON

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

Export nodes

JSON

Local

```
[
  {
    "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

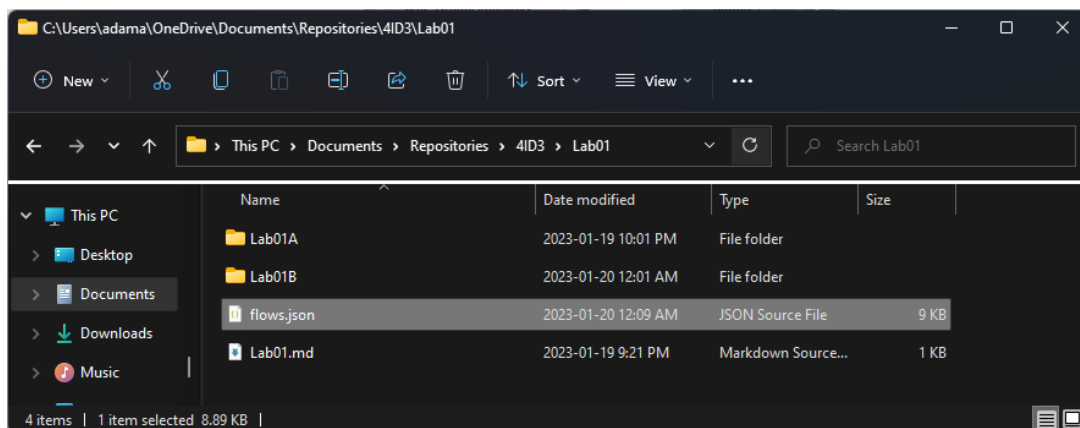
formatted

Cancel

Download

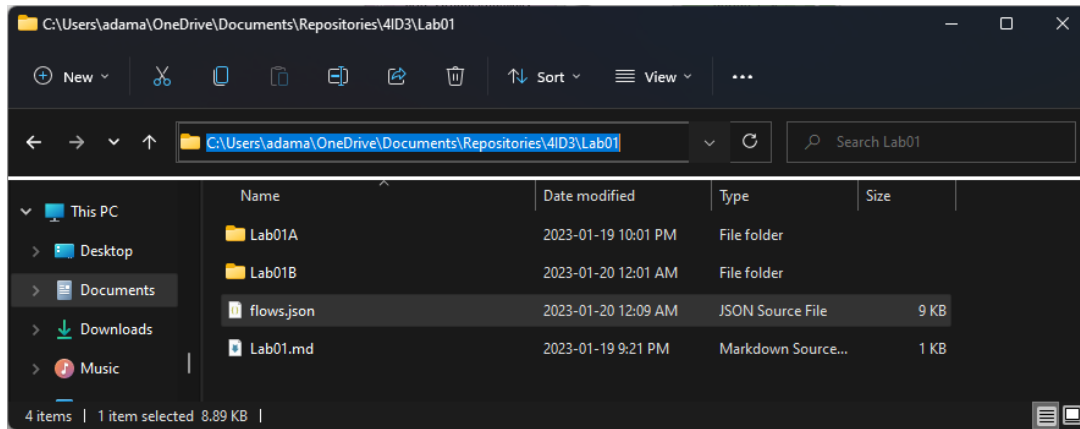
Copy to clipboard

Cut and paste the **flows.json** file from your **Downloads/** folder to the **Lab01** folder in the **local repo**.



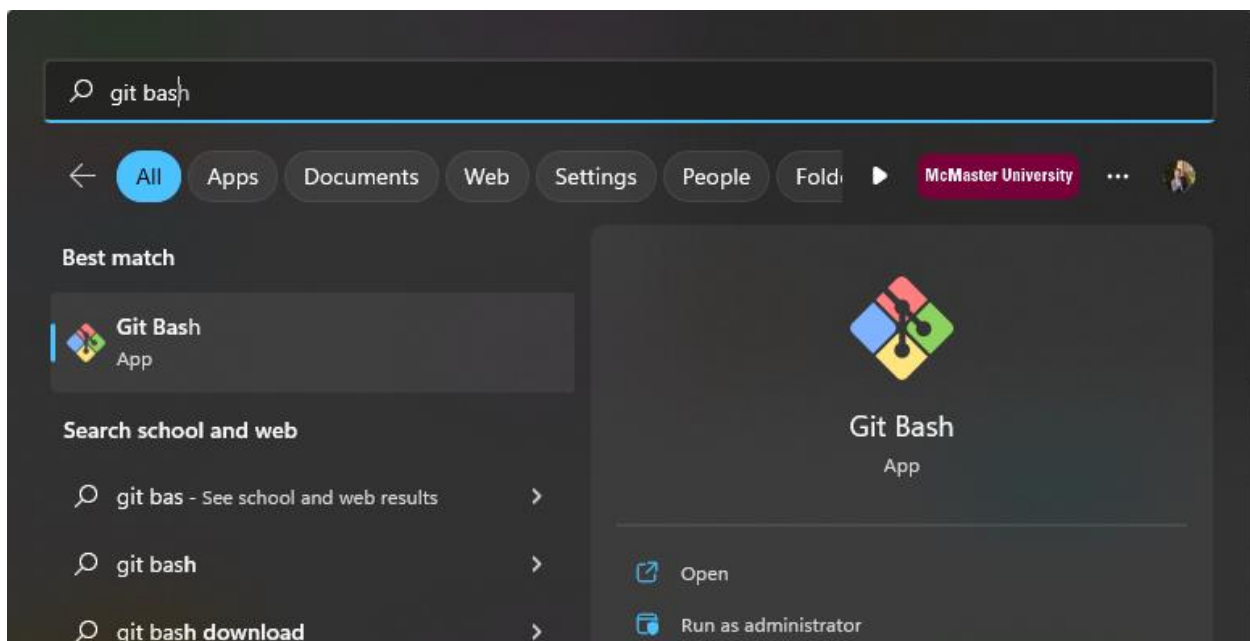
Committing and Pushing Changes to GitHub

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/Lab01
adama@DESKTOP-9L3EOVQ MINGW64 ~
$ cd "C:\Users\adama\OneDrive\Documents\Repositories\4ID3\Lab01"

adama@DESKTOP-9L3EOVQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3/Lab01 (main)
$ ls -la
total 13
drwxr-xr-x 1 adama 197609  0 Jan 20 00:00 ./
drwxr-xr-x 1 adama 197609  0 Jan 19 21:16 ../
-rw-r--r-- 1 adama 197609 188 Jan 19 21:21 Lab01.md
drwxr-xr-x 1 adama 197609  0 Jan 19 22:01 Lab01A/
drwxr-xr-x 1 adama 197609  0 Jan 20 00:01 Lab01B/
-rw-r--r-- 1 adama 197609 9108 Jan 20 00:09 flows.json

adama@DESKTOP-9L3EOVQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3/Lab01 (main)
$ A|
```

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

```
MINGW64:/c/Users/adama/OneDrive/Documents/Repositories/4ID3
adama@DESKTOP-9L3EOVQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3/Lab01 (main)
$ cd ..

adama@DESKTOP-9L3EOVQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)
$ ls -la
total 5
drwxr-xr-x 1 adama 197609 0 Jan 19 21:16 ./
drwxr-xr-x 1 adama 197609 0 Jan 15 14:56 ../
drwxr-xr-x 1 adama 197609 0 Jan 19 20:54 .git/
drwxr-xr-x 1 adama 197609 0 Jan 20 00:00 Lab01/
-rw-r--r-- 1 adama 197609 7 Jan 19 20:46 README.md

adama@DESKTOP-9L3EOVQ 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.

```
MINGW64:/c/Users/adama/OneDrive/Documents/Repositories/4ID3
adama@DESKTOP-9L3E0VQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)
$ git add .
warning: in the working copy of 'Lab01/flows.json', LF will be replaced by CRLF the next time
Git touches it

adama@DESKTOP-9L3E0VQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)
$ git commit -m "Completed Lab01"
[main 0e08199] Completed Lab01
6 files changed, 589 insertions(+)
create mode 100644 Lab01/Lab01.md
create mode 100644 Lab01/Lab01A/Lab01A.h
create mode 100644 Lab01/Lab01A/Lab01A.ino
create mode 100644 Lab01/Lab01B/Lab01B.h
create mode 100644 Lab01/Lab01B/Lab01B.ino
create mode 100644 Lab01/flows.json

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

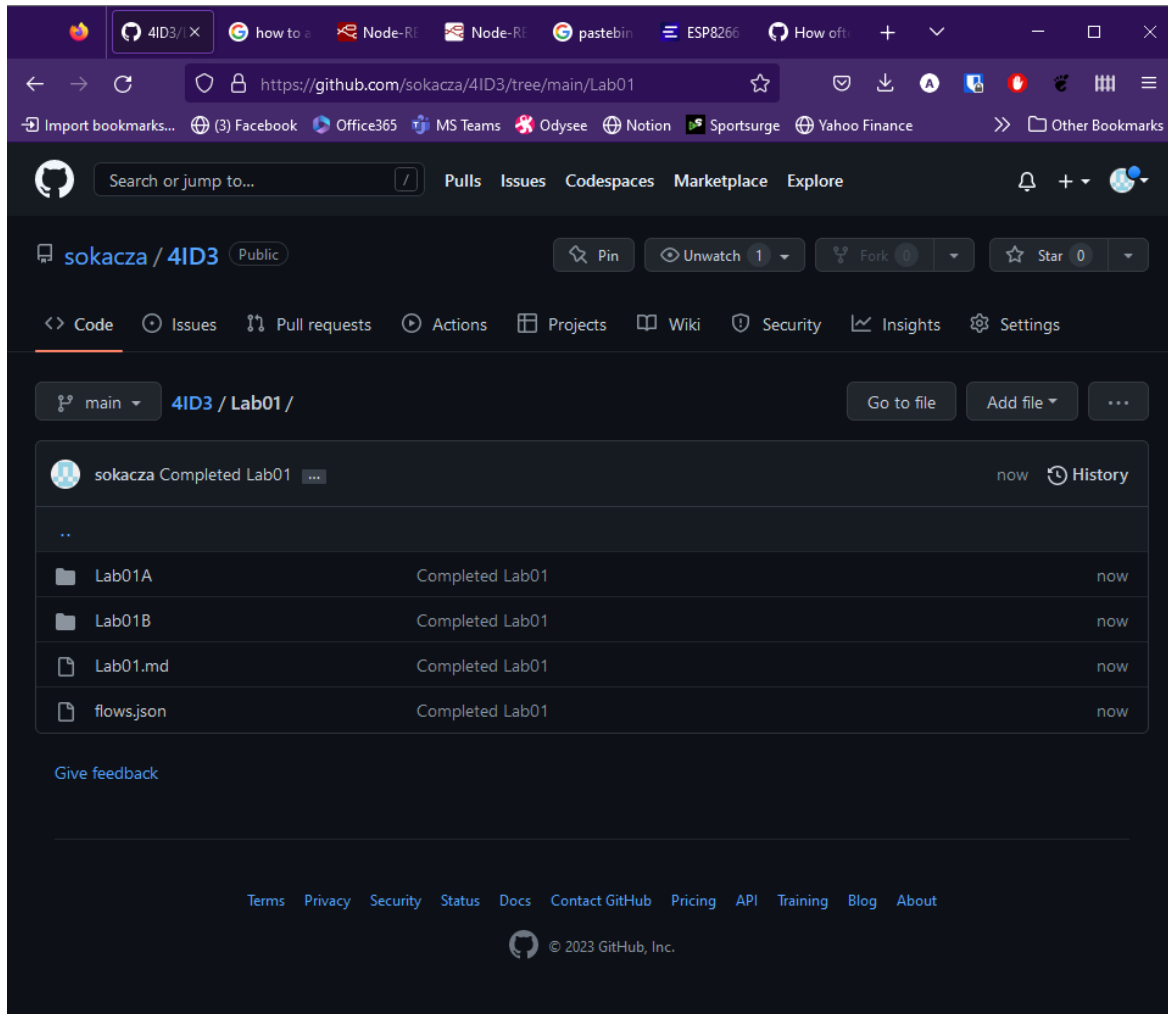
Push the changes in the **main** branch of your **local** repo to the remote **GitHub** repo named **origin**.

```
MINGW64:/c/Users/adama/OneDrive/Documents/Repositories/4ID3
adama@DESKTOP-9L3E0VQ MINGW64 ~/OneDrive/Documents/Repositories/4ID3 (main)
$ git push origin main
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 12 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (11/11), 3.57 KiB | 3.57 MiB/s, done.
Total 11 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
To github.com:sokacza/4ID3.git
2fbfd03..0e08199 main -> main

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

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

Lab 1 - Communicating Sensor Data over WiFi using MQTT



END