

4ID3 - IoT Devices and Networks

Lab 3

Communicating Sensor Data over a LoRa Network

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:

WBOOTH SCHOOL OF ENGINEERING
PRACTICE AND TECHNOLOGY

McMaster-Mohawk Bachelor of Technology Partnership



Objective

In this lab, sensor data will be read, encoded, and transmitted from a LoRa *Field Device* to a LoRa *Gateway Device*. The gateway will then forward that information to a Python *HTTP Server*, being run on a PC on the local WiFi network WiFi. This HTTP server will parse the sent data, publish to a corresponding *MQTT* path, update a *MongoDB database*, and return a response to the gateway device if the data transmission was successful.

Contents

Objective	2
Feedback	3
Additional Resources	3
Pre-Lab Questions.....	4
Post-Lab Questions	4
Exercise A Results:	6
Exercise B Results:.....	6
Exercise C Results:.....	6
Setting up the Workspace.....	7
Installing Python	14
Installing VSCode.....	15
Installing and Connecting to MongoDB	18
ESP32 Libraries.....	24
Ping Pong	25
Device Network.....	34
Field Device	36
Gateway Device	40
HTTP Server.....	43
Repo Setup	44
Python Libraries	48
IoT Server Overview	49
Setting up the IoT Network.....	58
Integrating Everything Together	60
Exercise A	63
Exercise B	64
Exercise C	64

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>)

Python Webservers (<https://youtu.be/DeFST8tvuI>)

MongoDB Databases with PyMongo (https://youtu.be/rE_bJl2GAY8)

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

ESP32 Overview (<https://youtu.be/UuxBfKA3U5M>)

Pre-Lab Questions

Q1 - In your own words, what is the difference between LoRa technology and LoRaWAN? What layer of the OSI model do each reside in?

(Suggested: Short paragraph)

Q2 - What is point-to-point communication? What role does a transceiver play? How does it differ from more sophisticated network topologies?

(Suggested: Short paragraph)

Q3 - A server URL is made up of the following components:

http://127.0.0.1:3000/data -> PROTOCOL://IP_ADDRESS:PORT/PATH

What is the role of each URL component?

(Suggested: List)

Q4 - What is an API request in web development? What is the difference between a GET and POST request to an API?

(Suggested: Short paragraph)

Post-Lab Questions

Q1 - Draw a diagram of part B of this lab to identify each component of this IoT network and describe the information being exchanged between the components.

(Suggested: Sketch)

Q2 - If a LoRa device is configured for use in North America, can it be legally used in Europe? Briefly describe how LoRa frequency bands are regulated differently in these two locations.

(Suggested: A few sentences)

Q3 - How does the LoRa Spreading Factor and Transceiver Power affect transmission range? What spreading factor would you suggest when transmitting over vast distances?

(Suggested: Short paragraph)

Q4 - What is Chirp Spread Spectrum and how does it differ from other modulation techniques? Provide a sketch in your answer.

(Suggested: 3 sketches + description)

Q5 - When communicating to a transceiver over serial communication, a TX pin from the microcontroller must be wired to the transceiver RX, and vice versa. Why is this the case? If the TX pin on the microcontroller was faulty, would it still be able to receive data from the transceiver?

(Suggested: A few sentences)

Q6 - In part B, data is transmitted from the gateway device to a webserver on the same network using a POST request. The IP address, port, and path are all critical to ensure the data arrives to its expected

Lab 3 - Communicating Sensor Data over a LoRa Network

destination. What is a **webserver route** in web development? How does a webserver route relate to a URL path?

(Suggested: Short paragraph)

Q7 - In this lab, the HTTP server is used to:

- a) Reroute data to an MQTT broker
- b) Insert data as documents in a MongoDB database
- c) Display data to the user as an HTML web page

Describe how webservers might be advantageous to use in larger IoT networks and compare the processing power and software capabilities of a PC running a server to a resource restricted microcontroller.

(Suggested: Short paragraph)

Q8 - Write a brief LinkedIn post about key learning takeaways from this lab.

(Suggested: Short paragraph)

Exercise A Results:

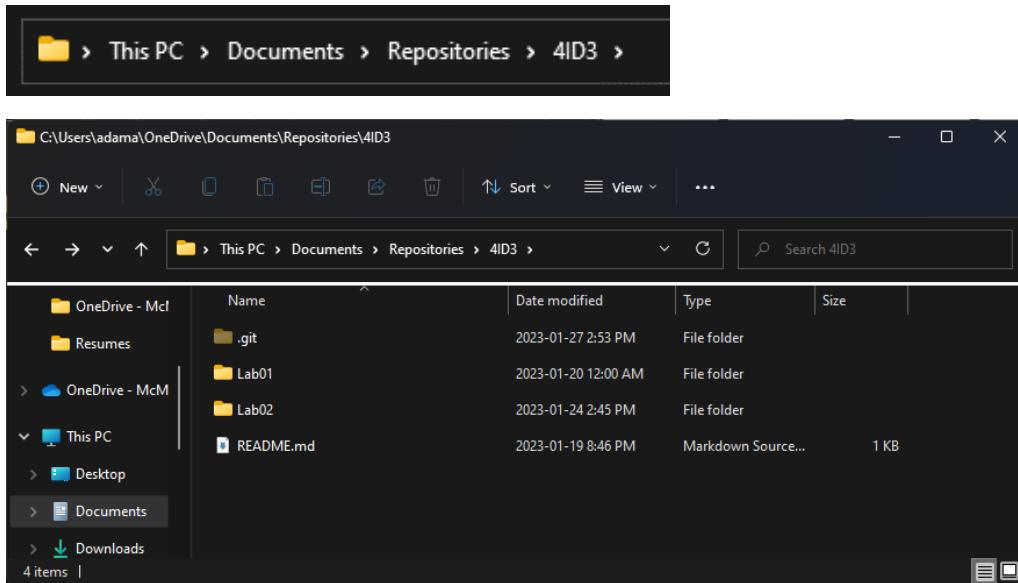
Exercise B Results:

Exercise C Results:

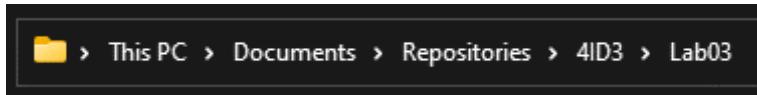
Setting up the Workspace

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.

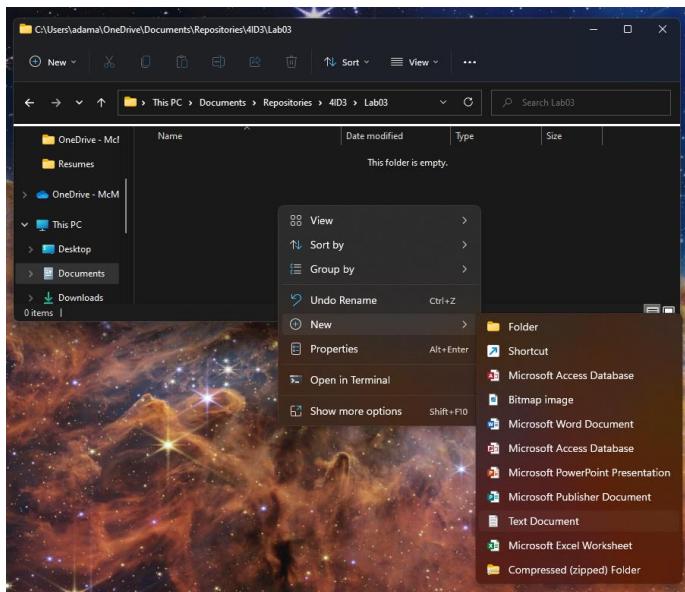
Navigate to your local git repository for this course.



Create a new folder named **Lab02**. Navigate inside this folder.

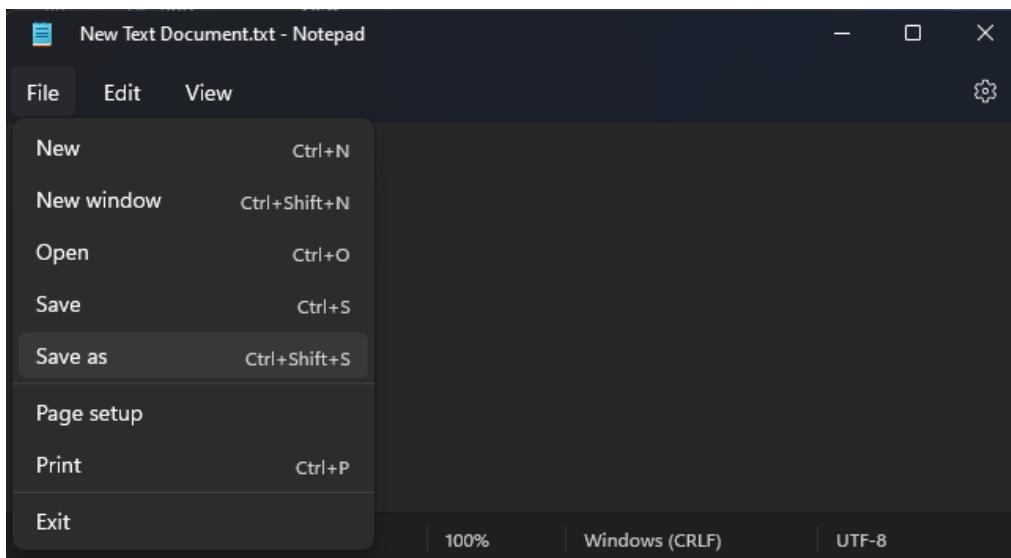


Create a new text file in the folder.

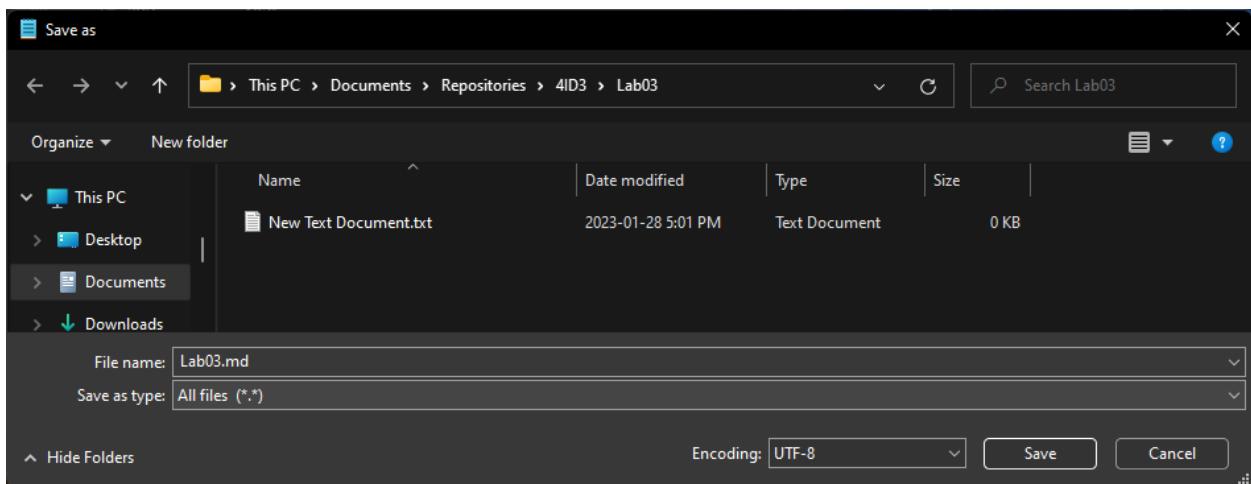


Lab 3 - Communicating Sensor Data over a LoRa Network

Press **File > Save as.**

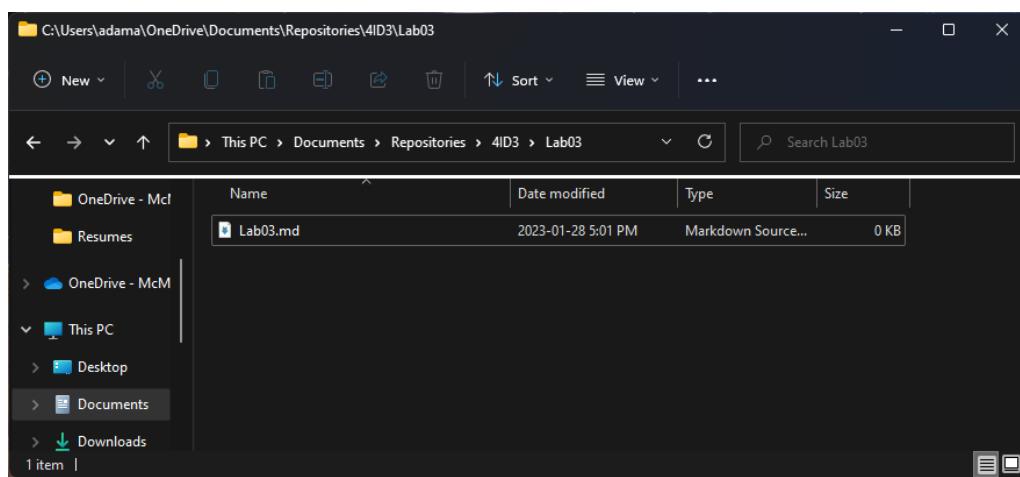
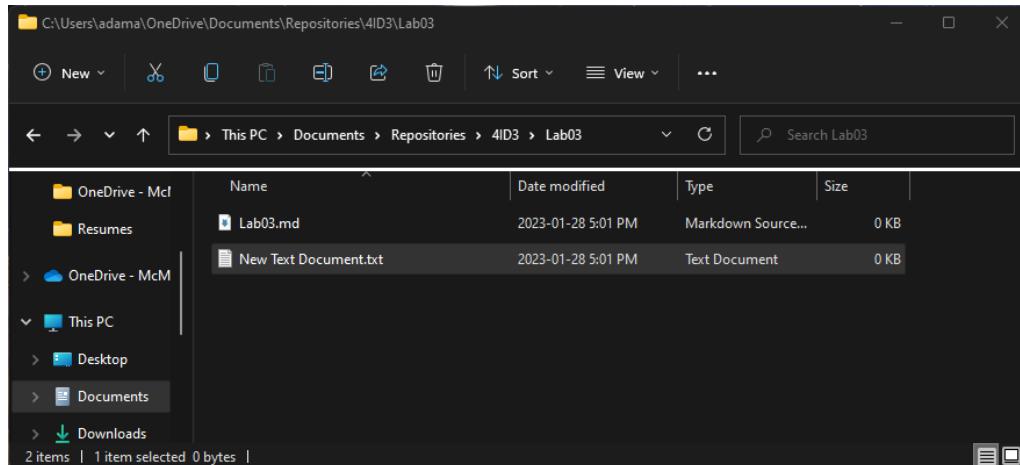


Save it as **Lab02.md**. Ensure that the **Save as type** is set to **All files (*.*)**.

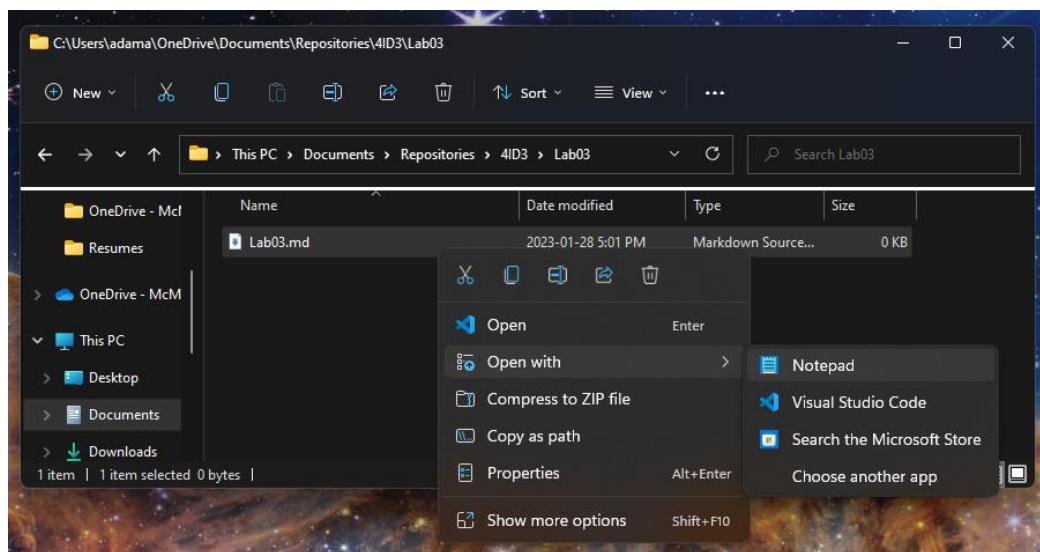


Now, you should have two files, a **text file** and a **markdown file**. Delete the text file.

Lab 3 - Communicating Sensor Data over a LoRa Network



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

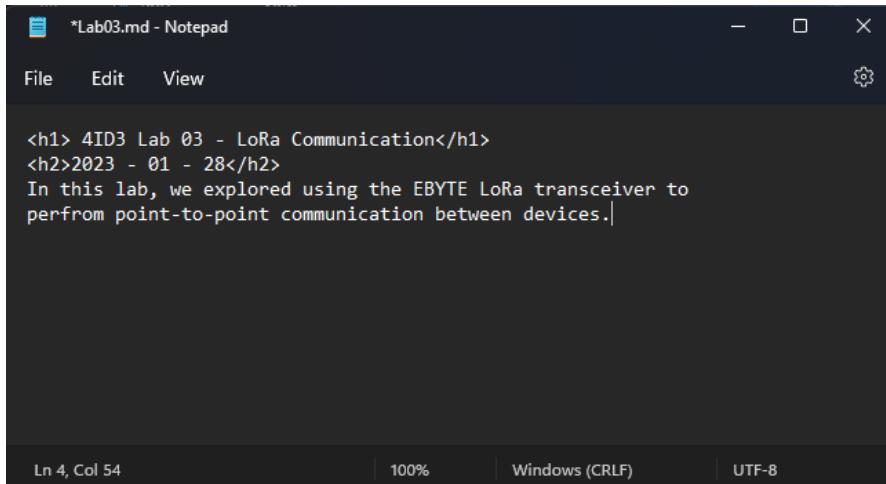


Lab 3 - Communicating Sensor Data over a LoRa Network

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.



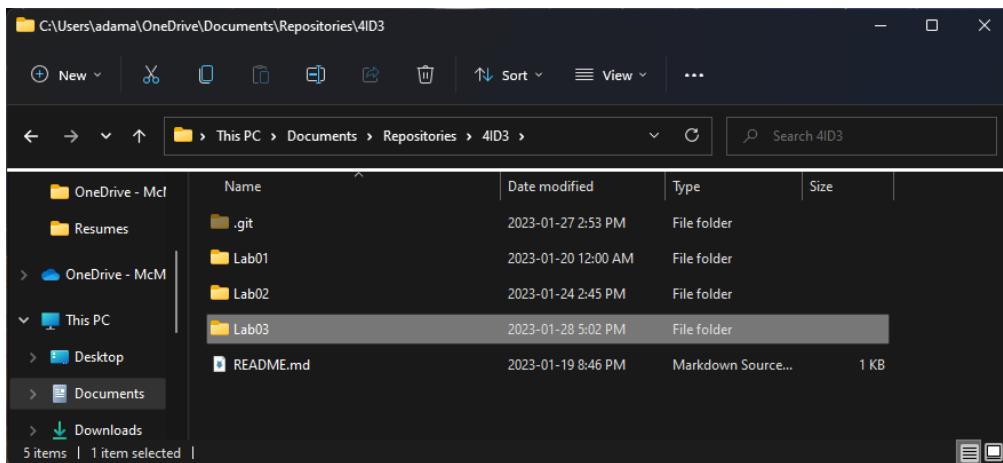
```
*Lab03.md - Notepad
File Edit View
<h1> 4ID3 Lab 03 - LoRa Communication</h1>
<h2>2023 - 01 - 28</h2>
In this lab, we explored using the EBYTE LoRa transceiver to
perform point-to-point communication between devices.|

Ln 4, Col 54 | 100% | Windows (CRLF) | UTF-8
```

```
<h1> 4ID3 Lab 03 - LoRa Communication</h1>
<h2>2023 - 01 - 28</h2>
In this lab, we explored using the EBYTE LoRa transceiver to
perform point-to-point communication between devices.
```

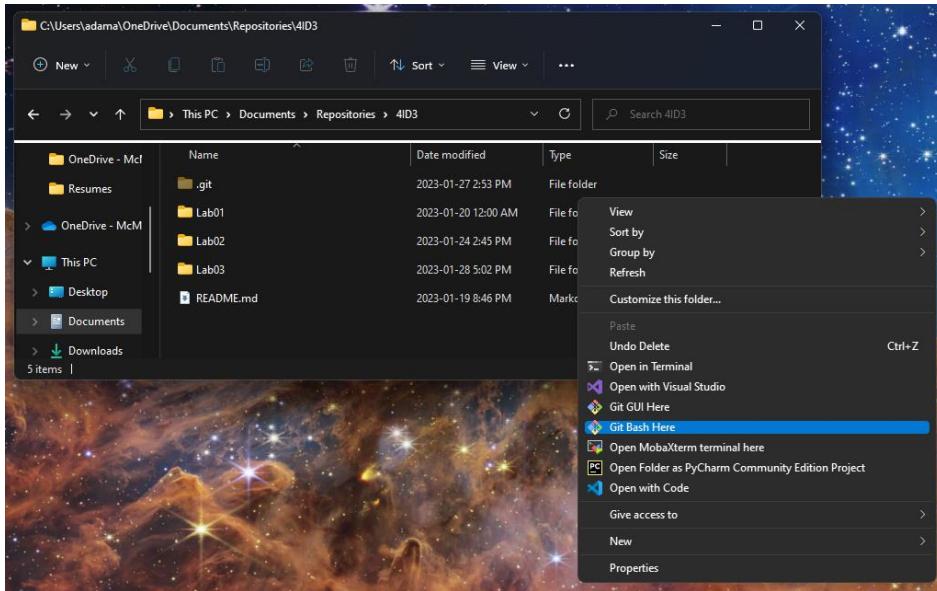
Save the file.

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

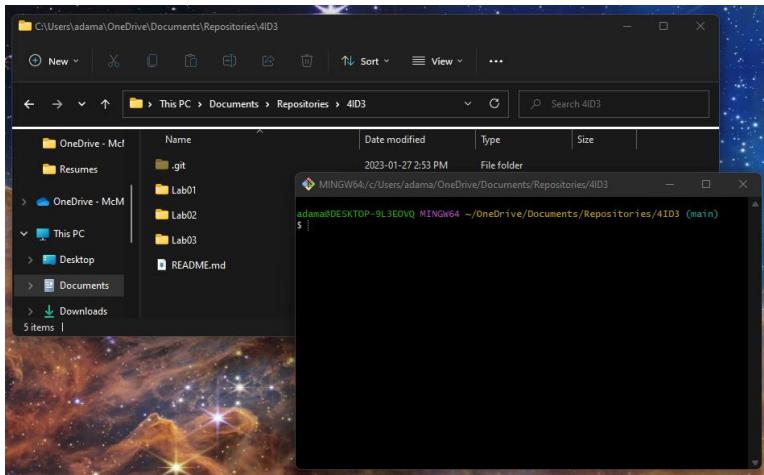


Lab 3 - Communicating Sensor Data over a LoRa Network

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



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.

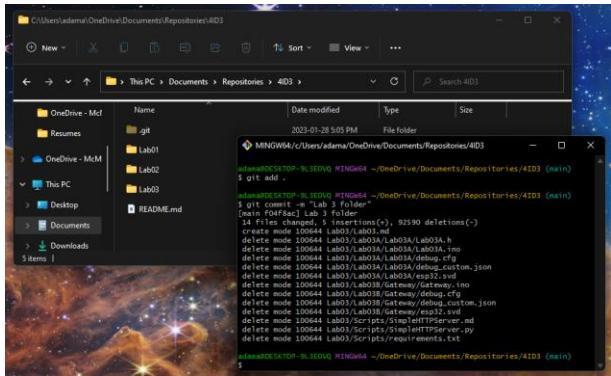


`git add .`

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.

Lab 3 - Communicating Sensor Data over a LoRa Network



```
git commit -m "Lab 3 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 screenshot of a terminal window titled 'MINGW64:c/Users/adama/OneDrive/Documents/Repositories/4ID3'. The user runs the command '\$ git push origin main'. The terminal shows the progress of the push operation: 'Enumerating objects: 12, done.', 'Counting objects: 100% (12/12), done.', 'Delta compression using up to 12 threads.', 'Compressing objects: 100% (6/6), done.', 'Writing objects: 100% (7/7), 725 bytes | 725.00 KiB/s, done.', 'Total 7 (delta 3), reused 0 (delta 0), pack-reused 0.', 'remote: Resolving deltas: 100% (3/3), completed with 3 local objects.', 'To github.com:sokacza/4ID3.git c5a0d81..f04f8ac main -> main'. The user then types '\$' at the end of the command.

```
git push origin main
```

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

4ID3 / Lab03 / Lab03.md

sokacza Lab 3 folder

Latest commit f04f8ac 1 minute ago History

1 contributor

4 lines (4 sloc) | 183 Bytes

4ID3 Lab 03 - LoRa Communication

2023 - 01 - 28

In this lab, we explored using the EBYTE LoRa transceiver to perform point-to-point communication between devices.

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

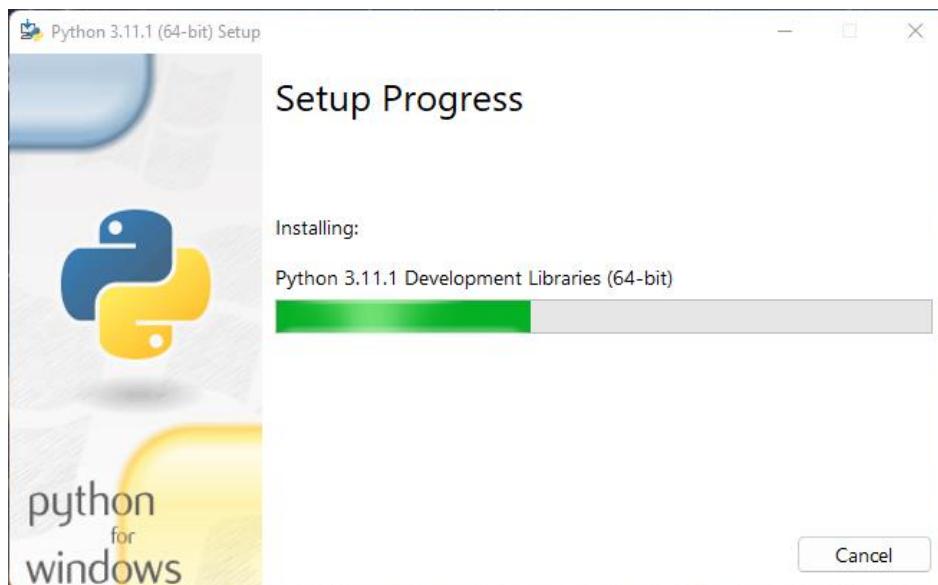
Installing Python

Navigate to the following website and install the latest version of Python 3.

<https://www.python.org/downloads/>

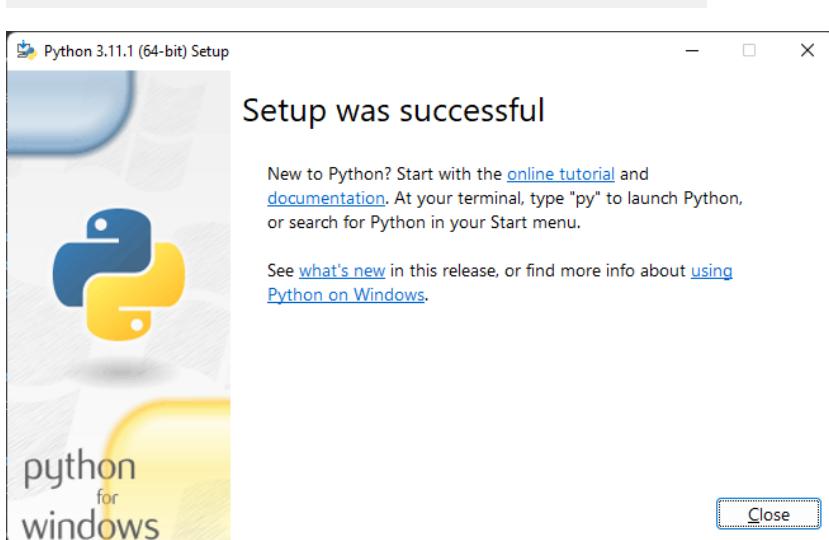
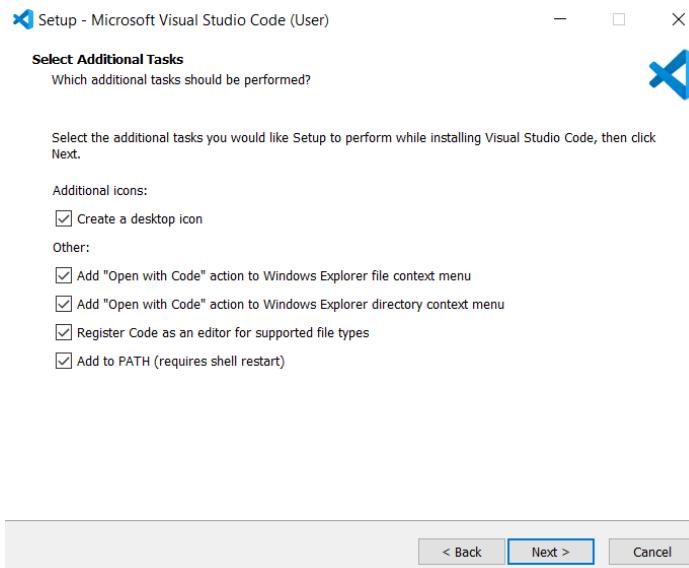


Install using the default options. If there is a checkbox to **Add to Path**, please **check this checkbox**.



MAKE SURE THAT YOU CHECK OFF 'ADD TO PATH' AND 'DISABLE PATH LENGTH LIMIT'

Lab 3 - Communicating Sensor Data over a LoRa Network



Once Python is installed, please **restart** your computer.

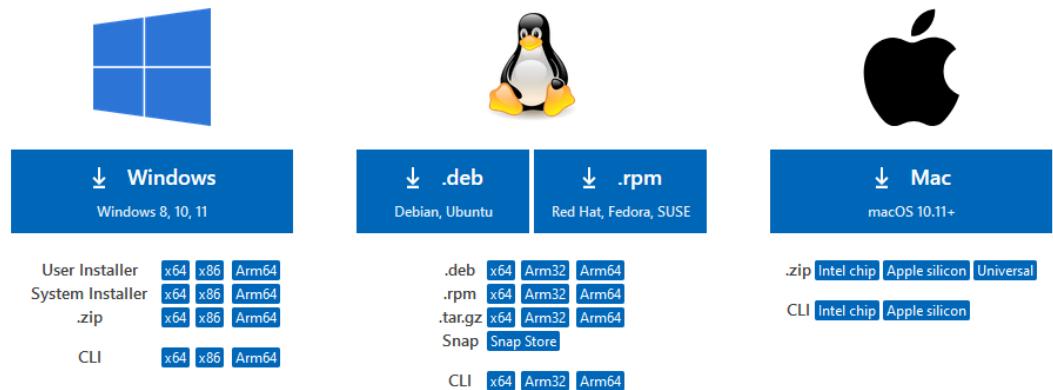
Installing VSCode

Navigate to the following URL:

<https://code.visualstudio.com/download>

Download Visual Studio Code

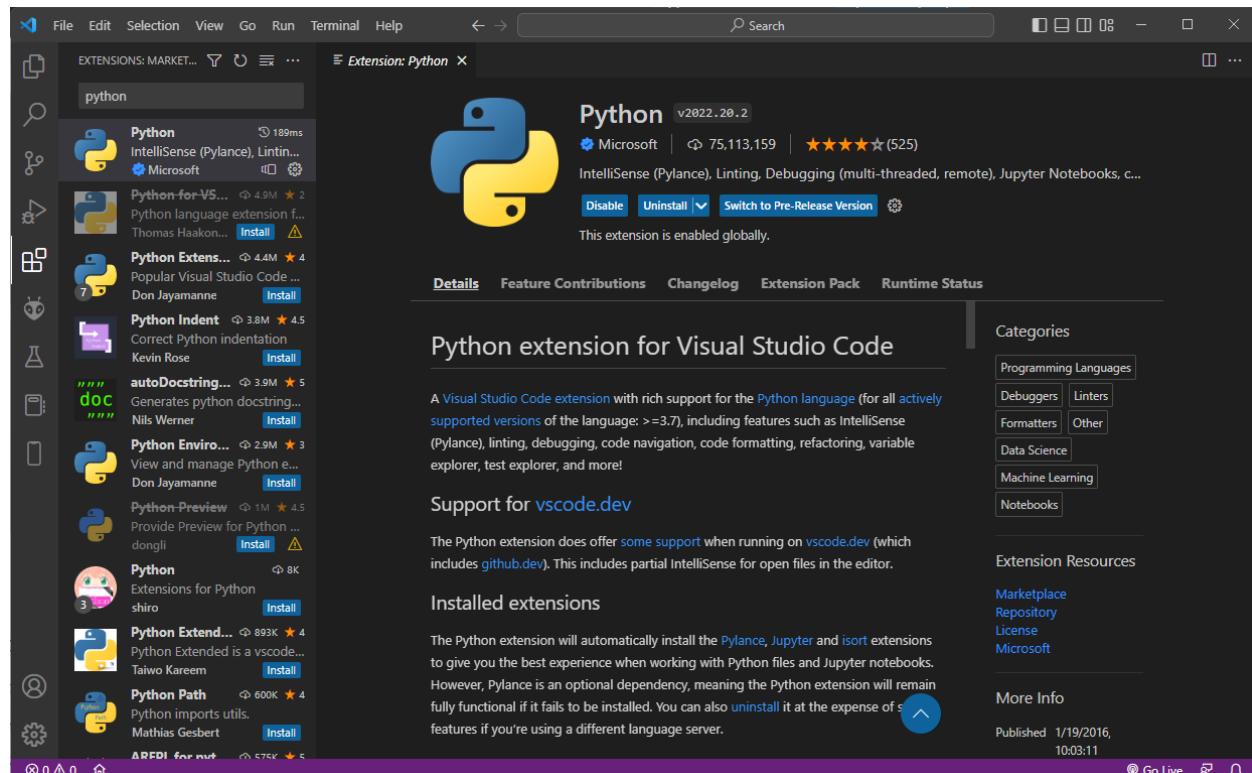
Free and built on open source. Integrated Git, debugging and extensions.



Run the installer using the **default install options**.

Once installed launch **VSCode**.

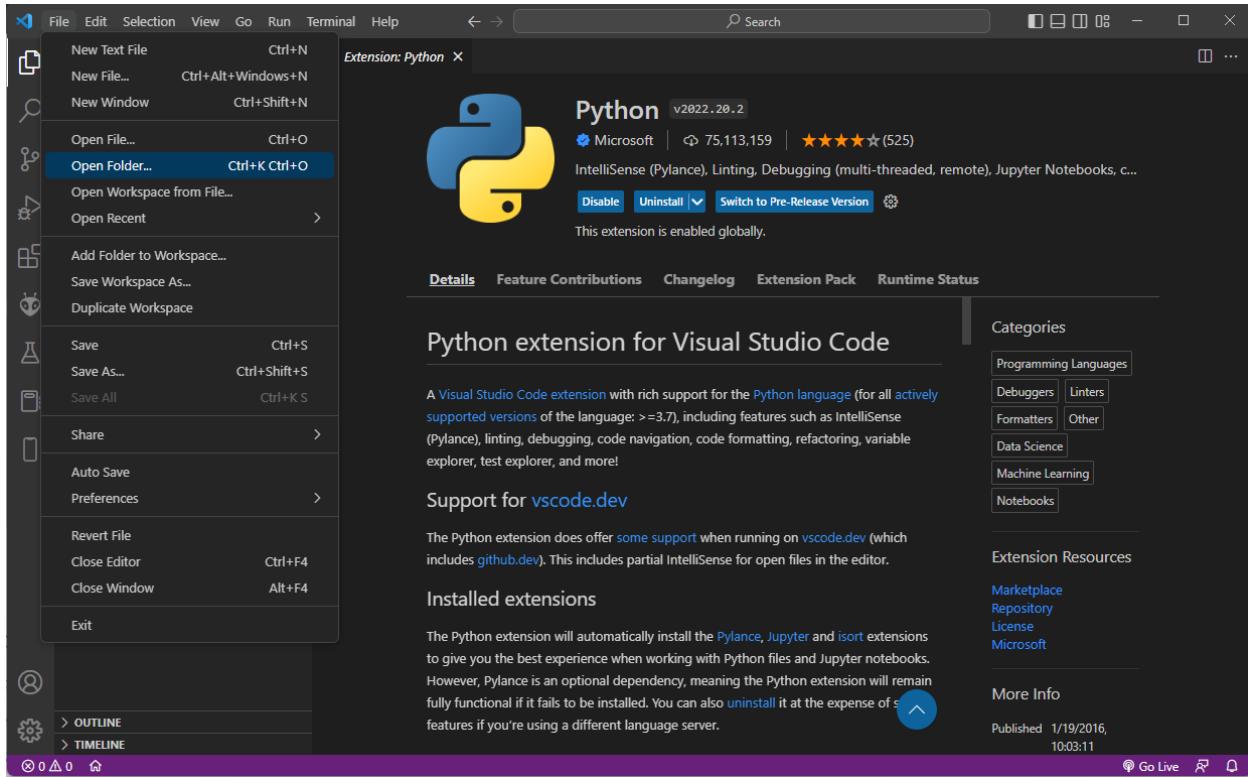
Open the **Extensions** tab using the ribbon on the left-side.



Lab 3 - Communicating Sensor Data over a LoRa Network

Search for **Python** and **Install** the one provided by **Microsoft**.

Within VSCode, do **File > Open Folder**.



Open your local repository.

Installing and Connecting to MongoDB

Navigate to the following URL and download **MongoDB Community**.

<https://www.mongodb.com/try/download/community-kubernetes-operator>

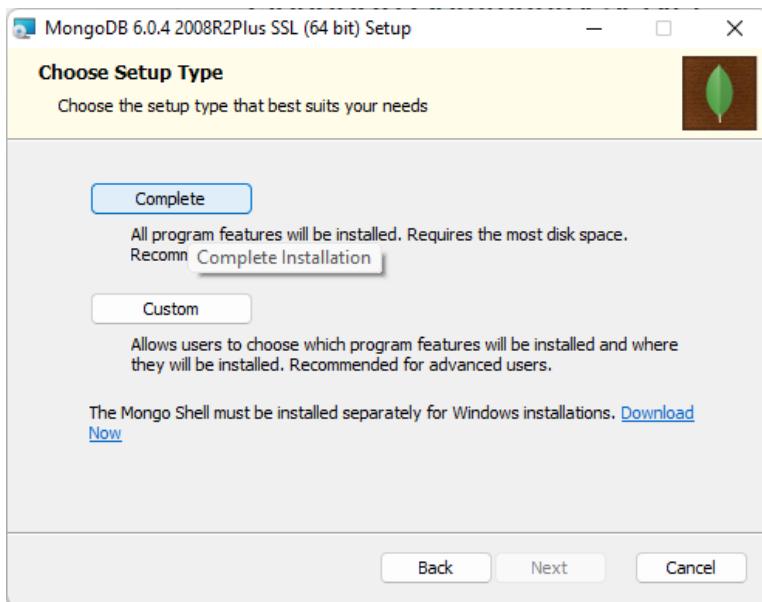


Run the installer using default settings. It will say **This may contain malicious software**. Ignore this warning, this software is very reputable, the packaging format is just out-of-date.

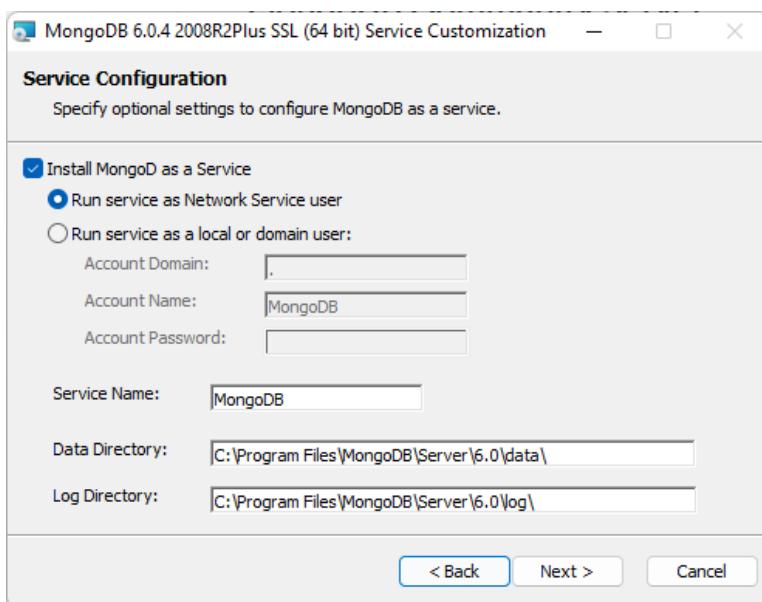


Perform a **Complete** installation.

Lab 3 - Communicating Sensor Data over a LoRa Network

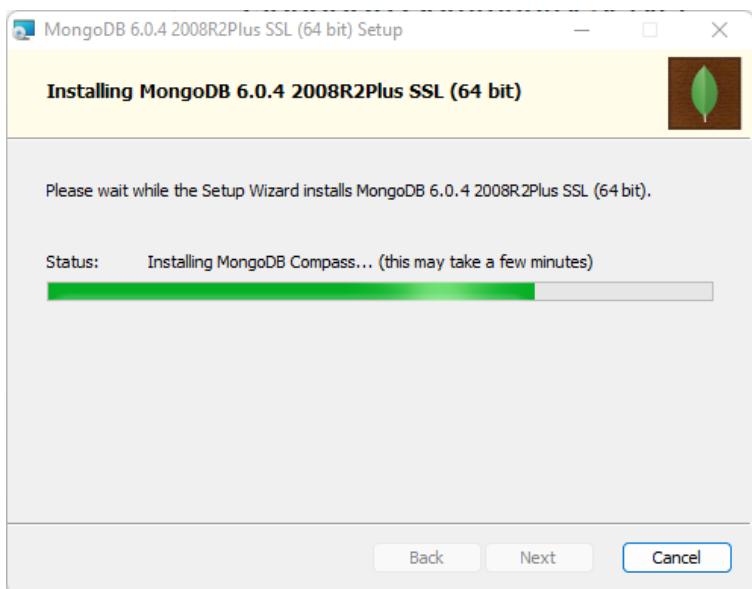
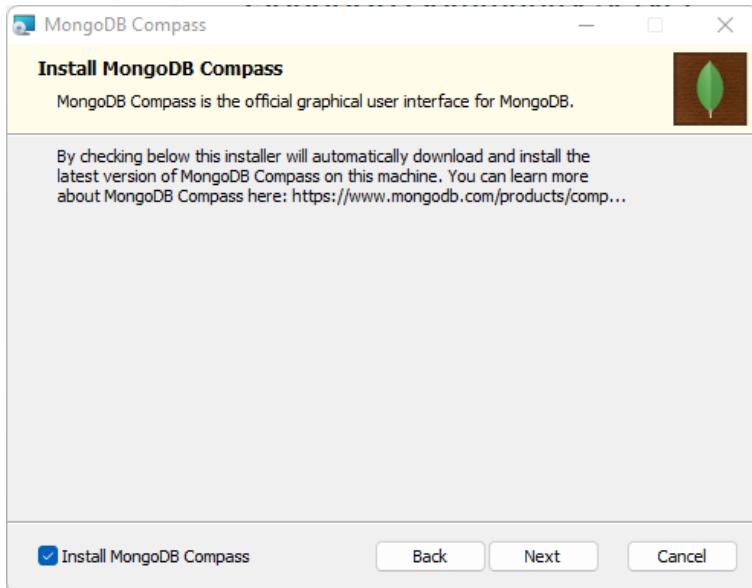


Leave **Service Configuration** as default.



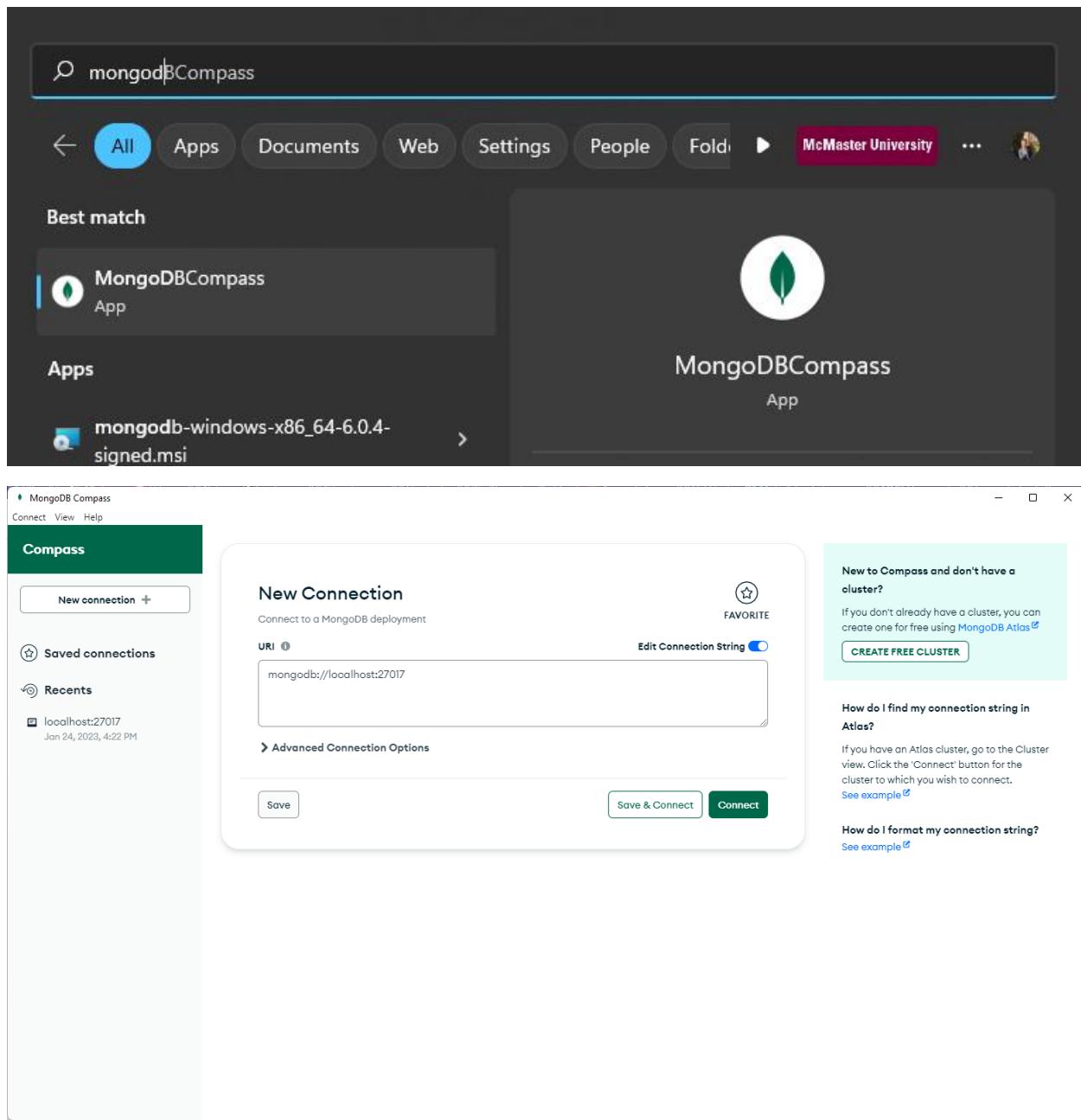
Ensure that **Install MongoDB Compass** is checked.

Lab 3 - Communicating Sensor Data over a LoRa Network



Launch **MongoDB Compass**.

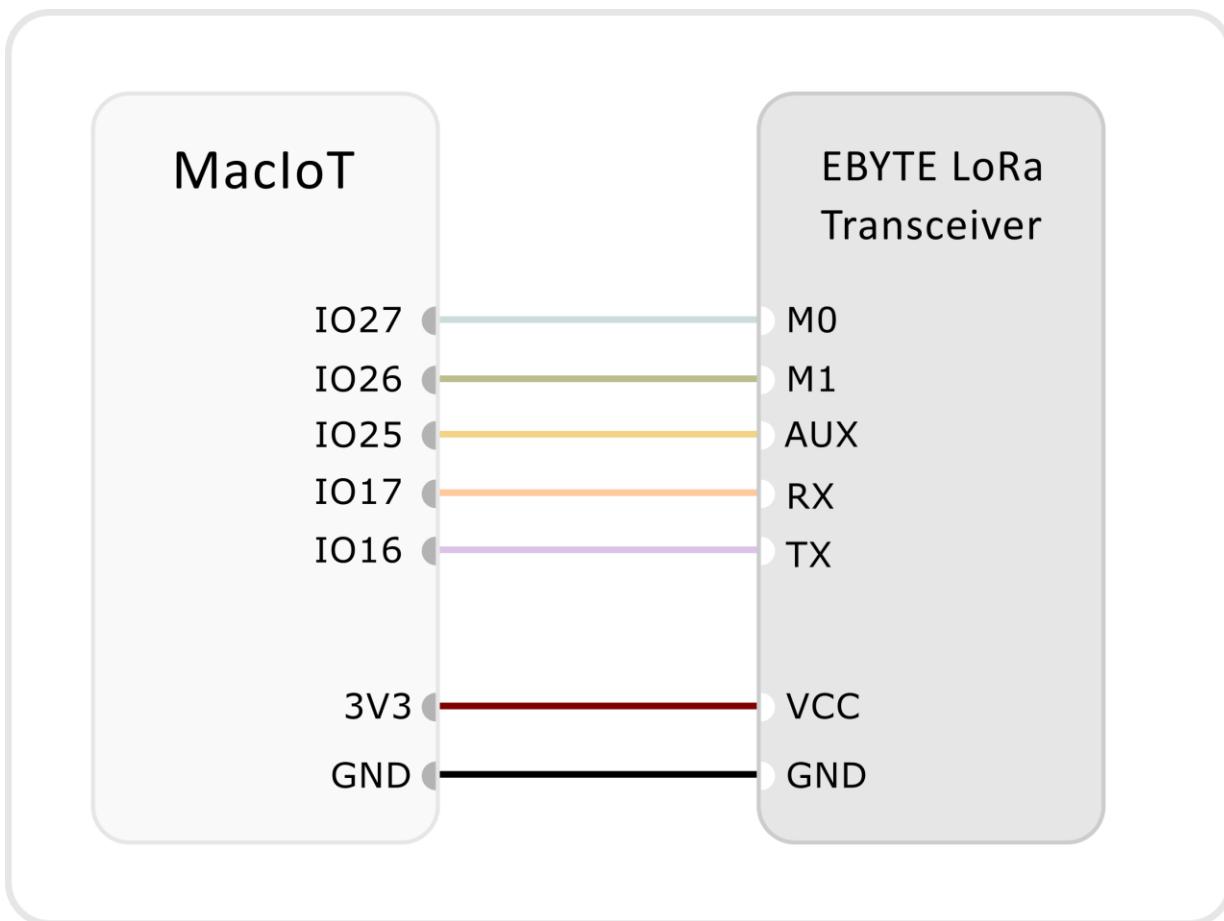
Lab 3 - Communicating Sensor Data over a LoRa Network



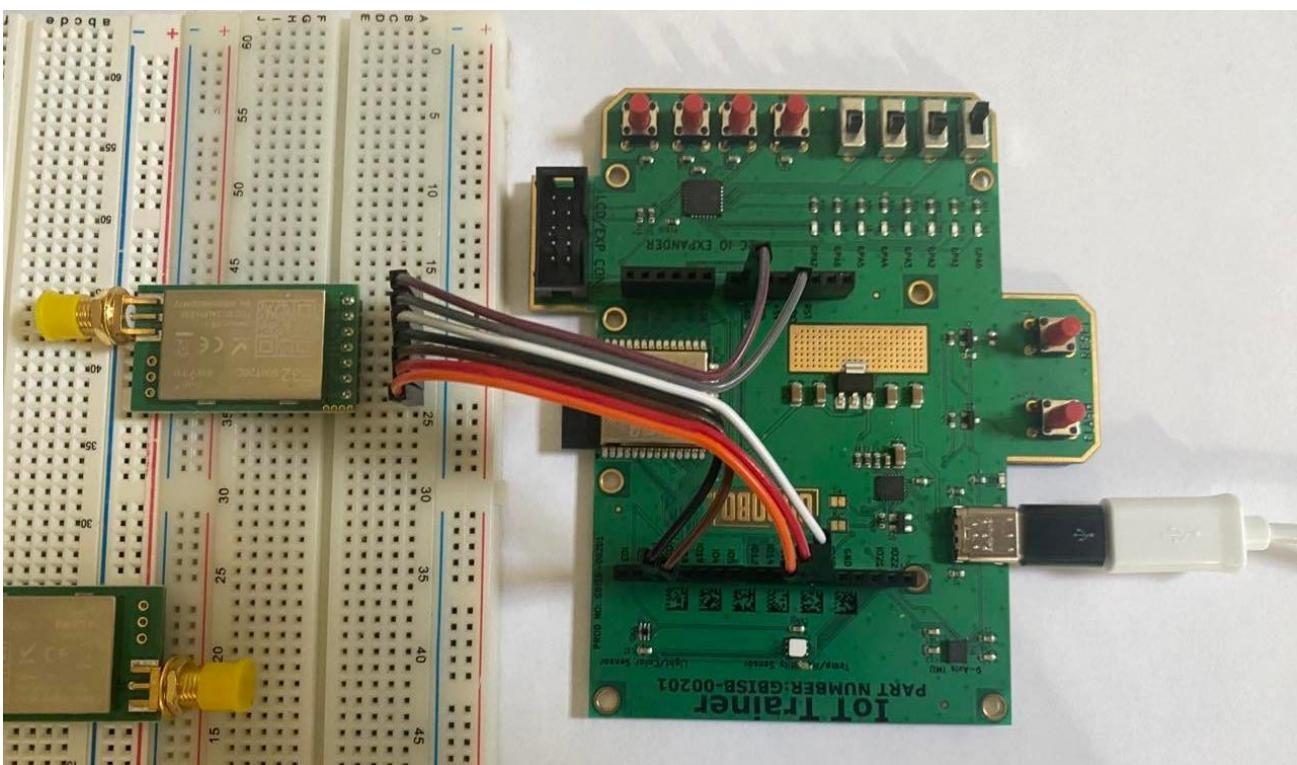
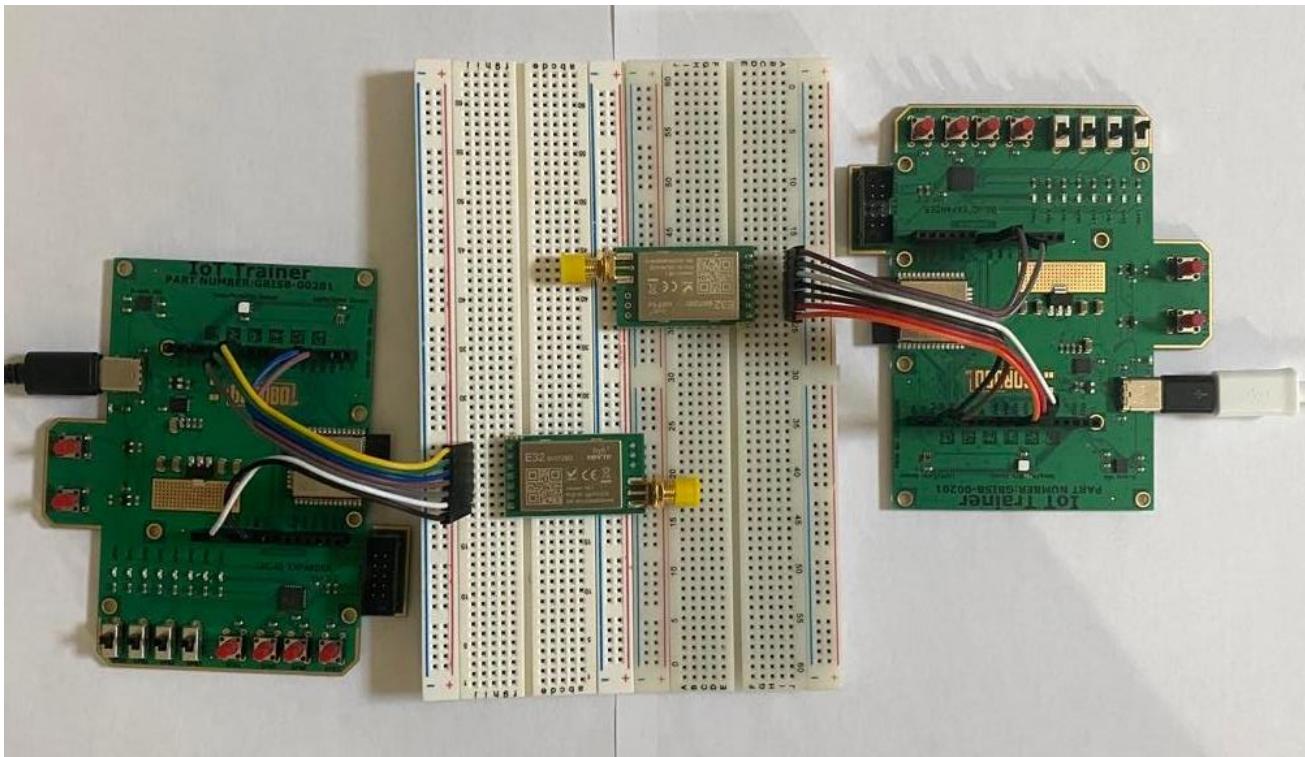
Press **Connect**.

Save & Connect **Connect**

Wiring Diagram



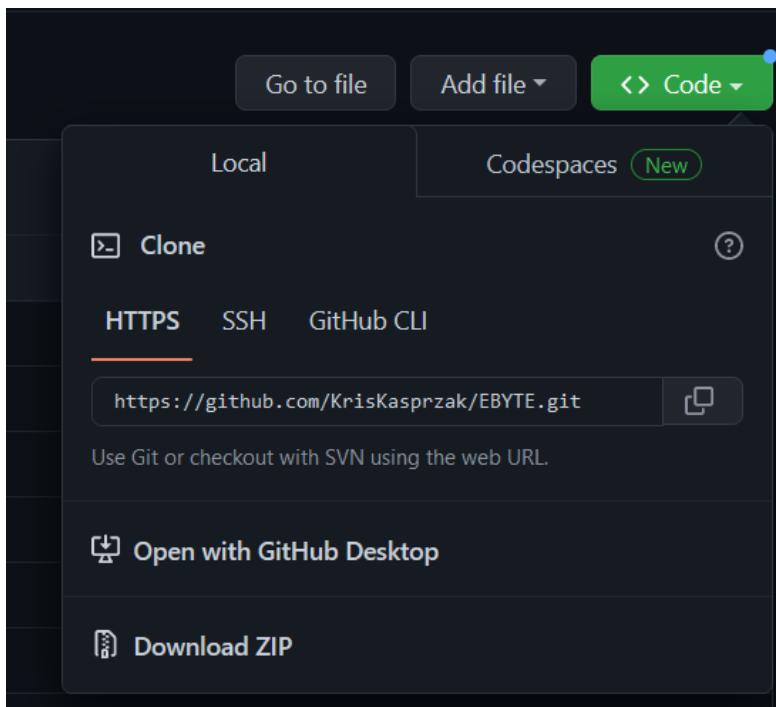
Lab 3 - Communicating Sensor Data over a LoRa Network



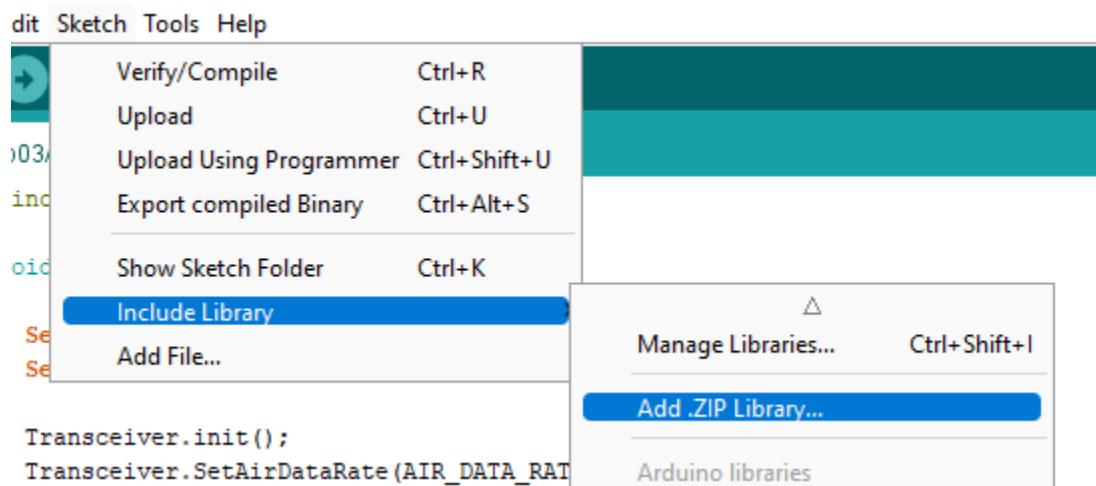
ESP32 Libraries

Navigate to the following repo and download the library as a **zip** file.

<https://github.com/KrisKasprzak/EBYTE>

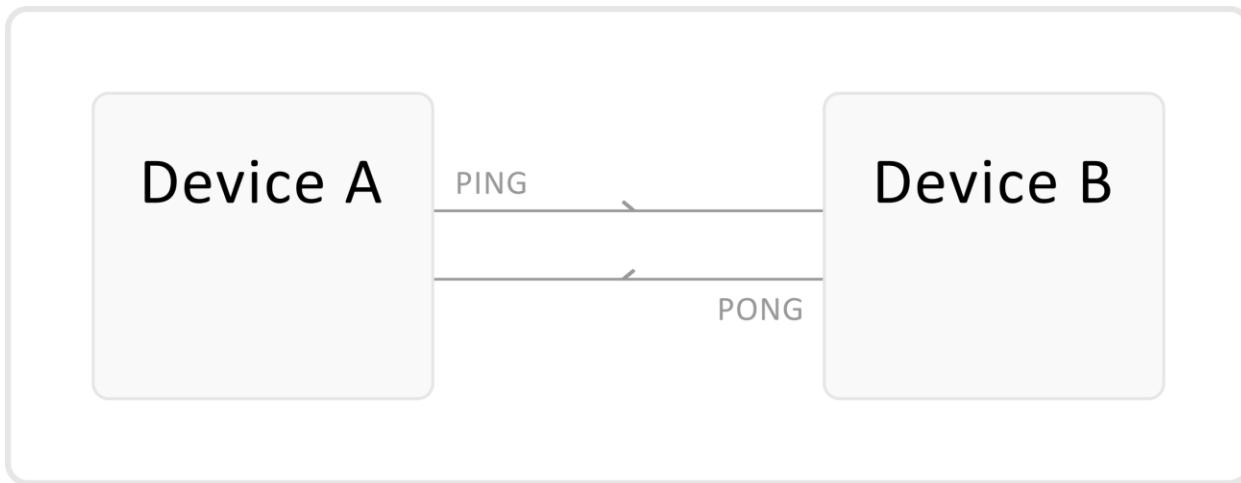


Install it in the Arduino IDE by navigating to **Sketch > Include Library > Add .ZIP Library**.



Ping Pong

The goal of this experiment is to observe point-to-point communication between two LoRa transceivers.



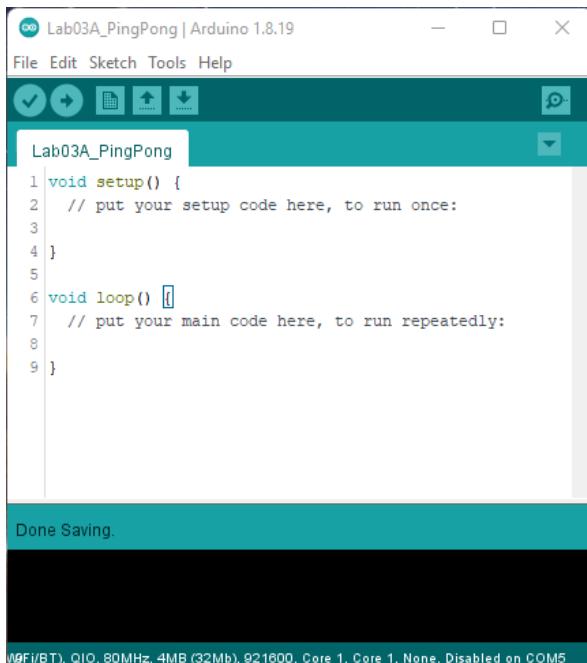
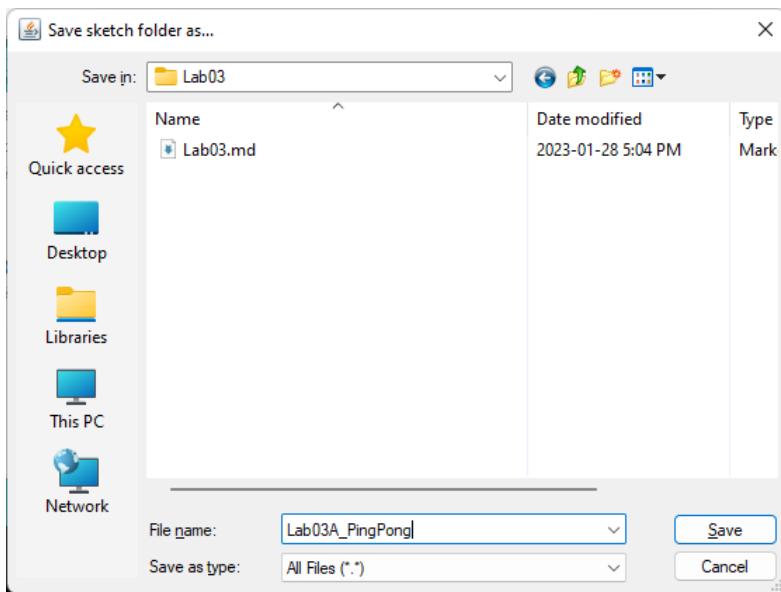
Open the Arduino IDE and create a new project. Name it **Lab03A_PingPong**.

The screenshot shows the Arduino IDE interface with a single open sketch named "sketch_jan28a". The code is as follows:

```
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 }
```

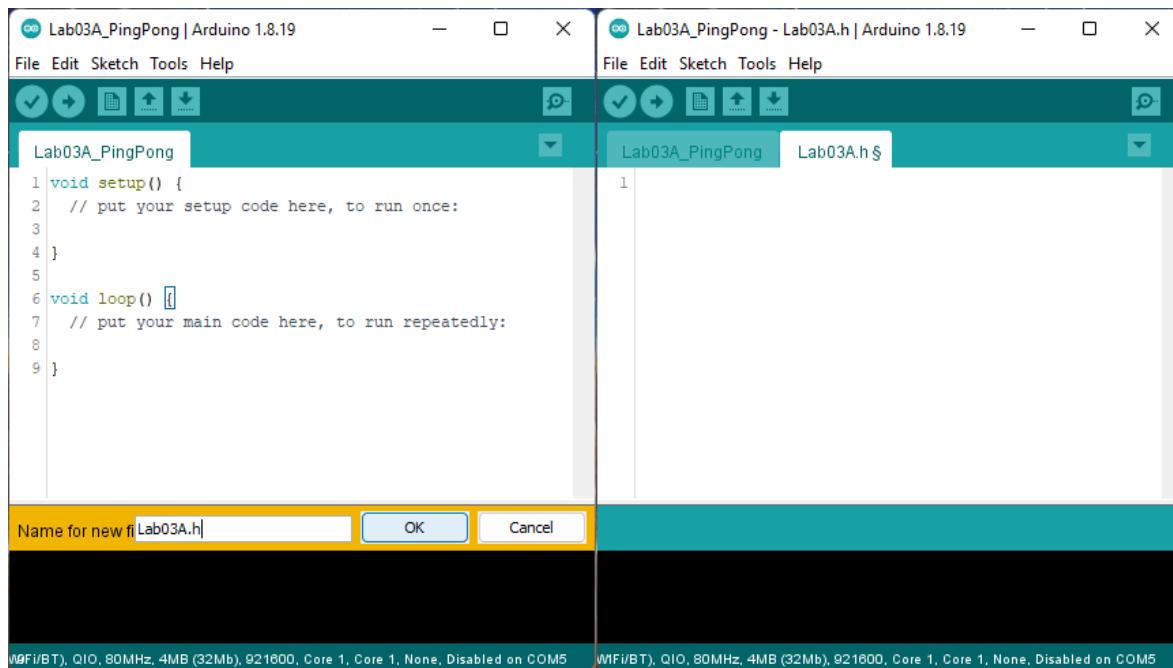
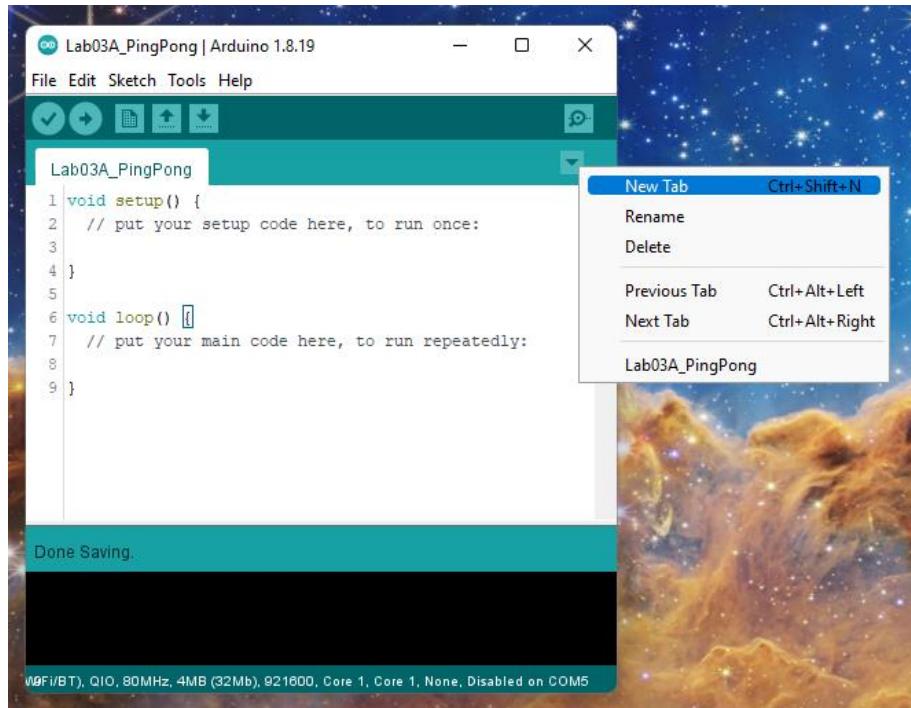
The status bar at the bottom indicates the board is an "ed, Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (32Mb), 921600, Core 1, Core 1, None, Disabled on COM5".

Lab 3 - Communicating Sensor Data over a LoRa Network



Create a header file to define and instantiate globally accessible data.

Lab 3 - Communicating Sensor Data over a LoRa Network



Include the following libraries and pin definitions.

Lab03A_PingPong	Lab03A.h
-----------------	----------

```
1 //Libraries
2 #include "EBYTE.h"
3 #include <HardwareSerial.h>
4
5 //Pin defines
6 #define PIN_RX 16    // Serial2 RX -> EBYTE TX
7 #define PIN_TX 17    // Serial2 TX pin -> EBYTE RX
8 #define PIN_M0 27
9 #define PIN_M1 26
10 #define PIN_AX 25
11
12 #define TIME_PER_SEND_MS 3000
```

Create the following to tune the frequency band of your transceiver.

Lab03A_PingPong	Lab03A.h §
-----------------	------------

```
14 //Transceiver setup
15 #define TRANSCEIVER_CHANNEL 42
16 EBYTE Transceiver(&Serial2, PIN_M0, PIN_M1, PIN_AX);
17
```

Set the device details to be unique for your group.

Lab03A_PingPong	Lab03A.h §
-----------------	------------

```
17
18 //Device details
19 String GROUP_NAME = "GroupA";
20 String DEVICE_ID = "Device5";
21
```

Navigate back to your implementation file.

Include your newly created header file.

Lab03A_PingPong	Lab03A.h
-----------------	----------

```
1 #include "Lab03A.h"
2
```

In the **setup()** file, initialize the serial monitor connected to your PC through UART, initialize the hardware serial 2 built-in to the ESP32 microcontroller. These must be set to 9600 baud for compatibility with the transceiver.

Using the Transceiver EBYTE transceiver object you are able to configure the settings of the transceiver through the 3 additional configuration pins. The library simplifies the process so you don't need to submit AT configuration commands.

```
Lab03A_PingPong Lab03A.h $  
1 #include "Lab03A.h"  
2  
3 void setup() {  
4  
5   Serial.begin(9600);  
6   Serial2.begin(9600);  
7  
8   Transceiver.init();  
9   Transceiver.SetAirDataRate(AIR_DATA_RATE);  
10  Transceiver.SetChannel[TRANSCEIVER_CHANNEL];  
11  //Transceiver.SetMode(MODE_NORMAL);  
12  //Transceiver.SetTransmitPower(OPT_TP20);  
13  //Transceiver.SaveParameters(PERMANENT);  
14  Transceiver.PrintParameters();  
15 }  
16
```

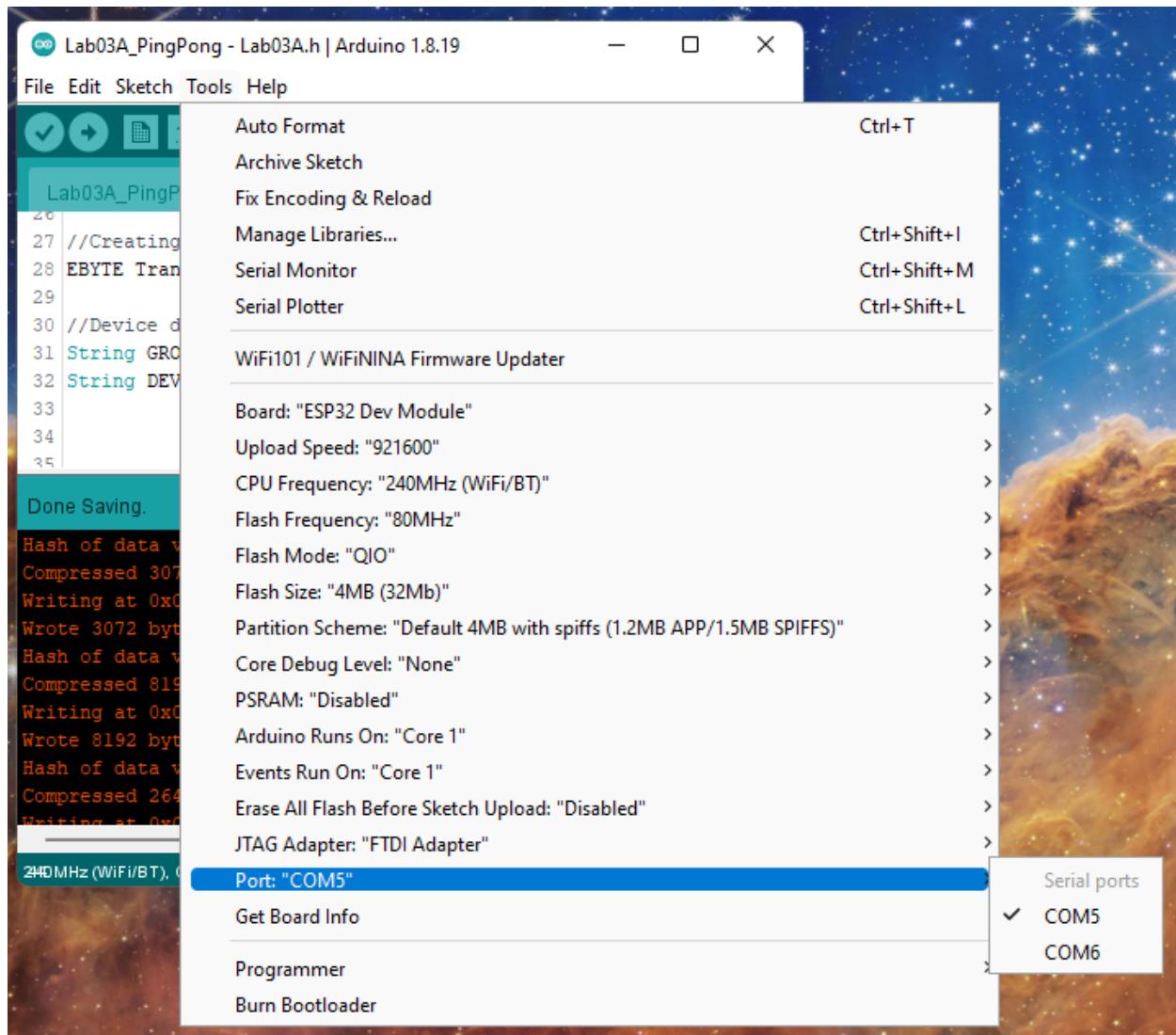
In the **loop()** function, there are two things that happen. First, the serial buffer is checked. If there is data being sent from the transceiver to the ESP32, it will read that string and print that to the serial monitor with **PING:** in front of the data. If a specific amount of time elapses, it will periodically transmit data.

```
Lab03A_PingPong Lab03A.h  
10  
17 unsigned long startTime = millis();  
18  
19 void loop() {  
20  
21   if (Serial2.available() > 1) {  
22     Serial.println("Receiving Data...");  
23     String incomingData = Serial2.readString();  
24     Serial.println("PING: " + incomingData);  
25   }  
26  
27   if(millis() - startTime > TIME_PER_SEND_MS){  
28     Serial2.println(GROUP_NAME + " - " + DEVICE_ID);  
29     Serial.println("PONG SENT");  
30     startTime = millis();  
31   }  
32 }  
33  
34 }
```

Lab 3 - Communicating Sensor Data over a LoRa Network

Make the device ID unique to your 1st microcontroller, select the COM port, and upload the sketch.

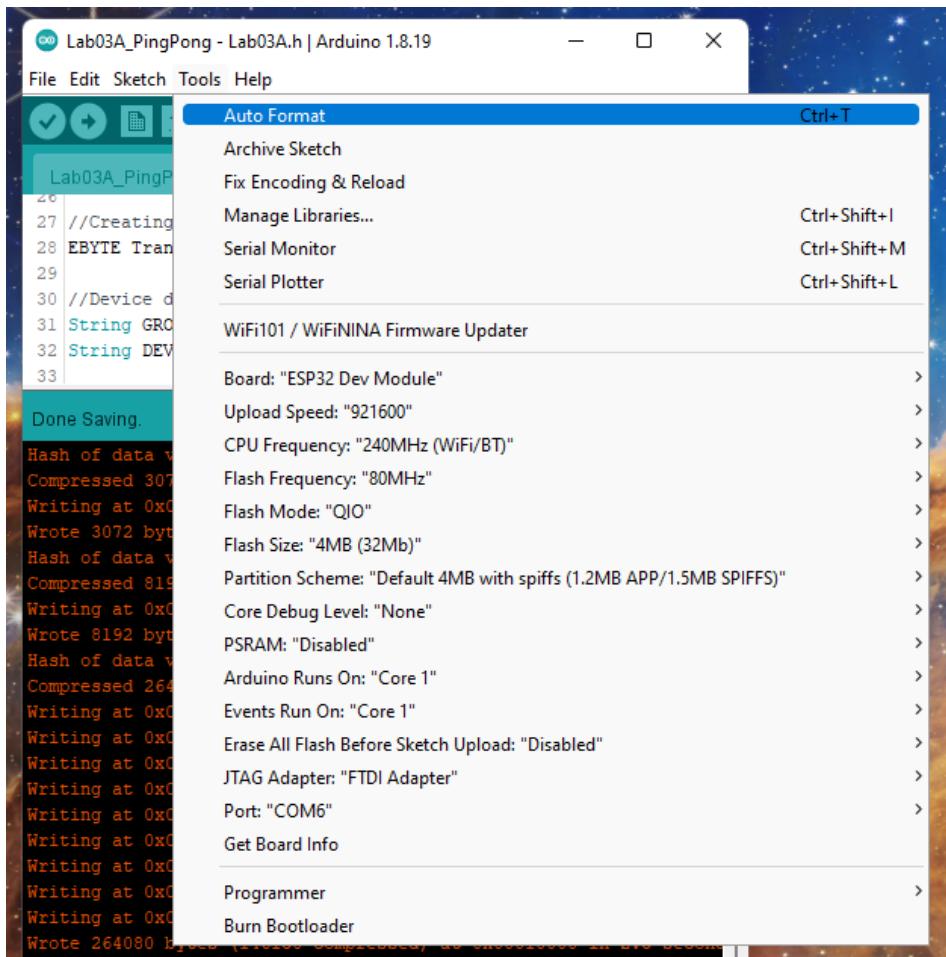
```
30 //Device details
31 String GROUP_NAME = "GroupA";
32 String DEVICE_ID = "Device5";
33
```



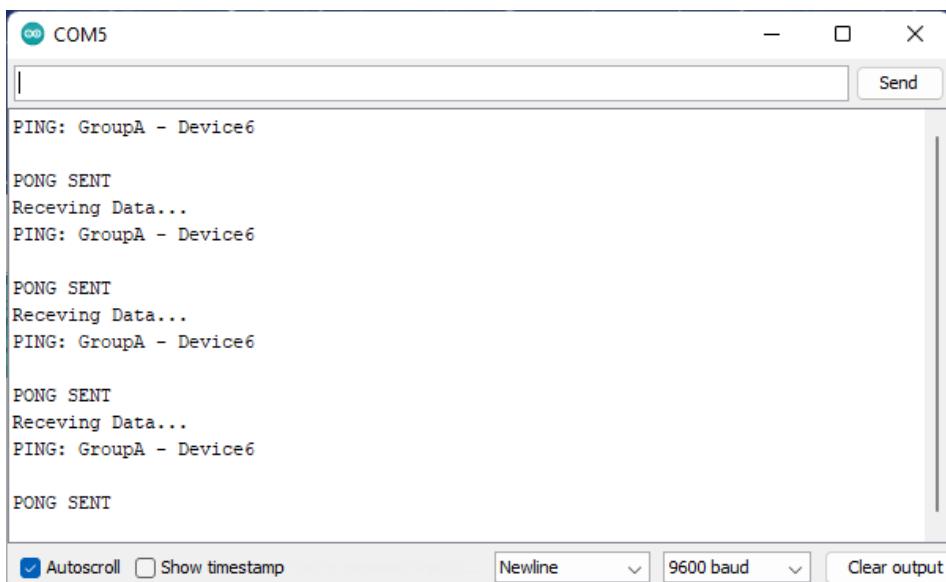
```
Done Saving.  
Hash of data verified.  
Compressed 3072 bytes to 146...  
Writing at 0x00008000... (100 %)  
Wrote 3072 bytes (146 compressed) at 0x00008000 in 0.1 seconds (ef  
Hash of data verified.  
Compressed 8192 bytes to 47...  
Writing at 0x0000e000... (100 %)  
Wrote 8192 bytes (47 compressed) at 0x0000e000 in 0.1 seconds (ef  
Hash of data verified.  
Compressed 264080 bytes to 146150...  
Writing at 0x00010000... (11 %)  
Writing at 0x0001c5a3... (22 %)  
Writing at 0x0002527d... (33 %)  
Writing at 0x0002a392... (44 %)  
Writing at 0x0002f8f9... (55 %)  
Writing at 0x00034fce... (66 %)  
Writing at 0x0003e958... (77 %)  
Writing at 0x00045f4f... (88 %)  
Writing at 0x0004b44c... (100 %)  
Wrote 264080 bytes (146150 compressed) at 0x00010000 in 2.5 seconds  
Hash of data verified.  
  
Leaving...  
Hard resetting via RTS pin...
```

Modify the device ID to be unique to your 2nd microcontroller and upload the sketch to your 2nd ESP32. Also, change the COM port.

Lab 3 - Communicating Sensor Data over a LoRa Network



Open the 1st devices COM port and ensure that you are receiving the data that the 2nd device is transmitting.



Lab 3 - Communicating Sensor Data over a LoRa Network

Open the 2nd devices COM port and ensure that you are receiving data being transmitted from the 1st devices transceiver.

```
COM6
OptionPower (HEX/DEC/BIN) : 0/0/0
-----
PONG SENT
Receiving Data...
PING: GroupA - Device5

PONG SENT
Receiving Data...
PING: GroupA - Device5

PONG SENT
Receiving Data...
PING: GroupA - Device5

PONG SENT
```

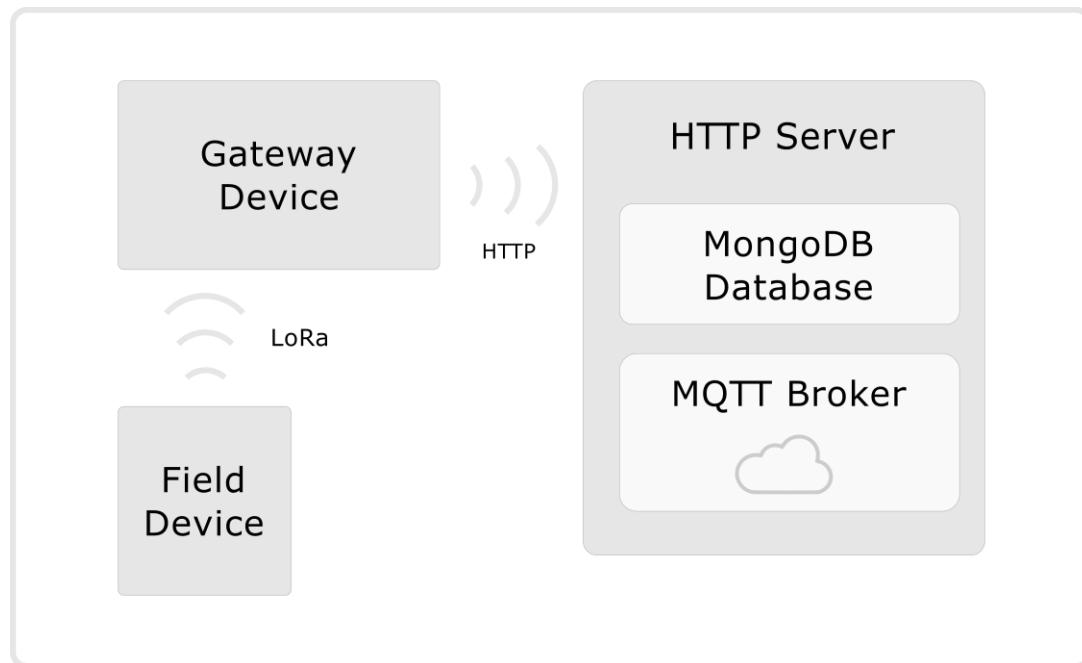
Autoscroll Show timestamp Newline 9600 baud Clear output

Device Network

In this lab section, we will be developing a more sophisticated IoT network that involves 5 key components:

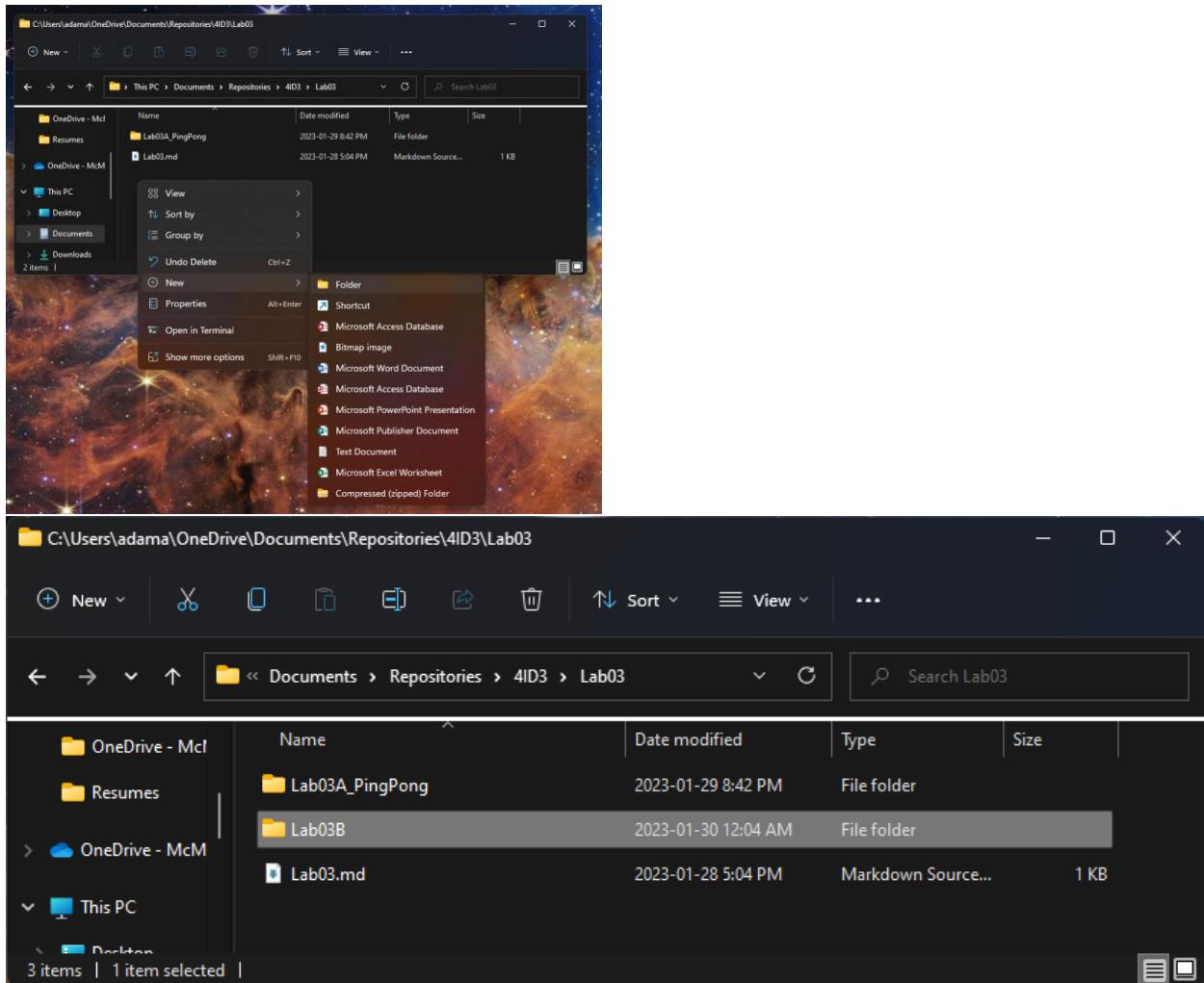
- a) Field devices (*Sensors -> LoRa*)
- b) Gateway devices (*LoRa -> WiFi*)
- c) Server Back-end (*HTTP request routing*)
- d) Dashboard Front-end (*Remote client access*)
- e) Database connection (*Data collection and processing*)

The goal of building this IoT network is to demonstrate how different technologies and systems work together to build more robust and plausible networks.



Navigate to your **4ID3/Lab03** repository and create a folder called **Lab03B**.

Lab 3 - Communicating Sensor Data over a LoRa Network

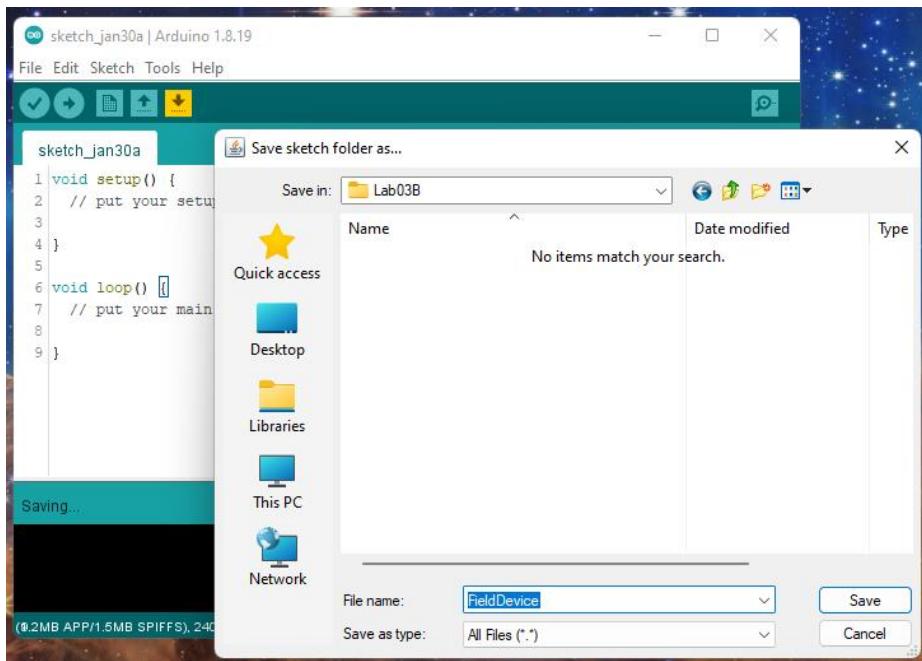


Lab 3 - Communicating Sensor Data over a LoRa Network

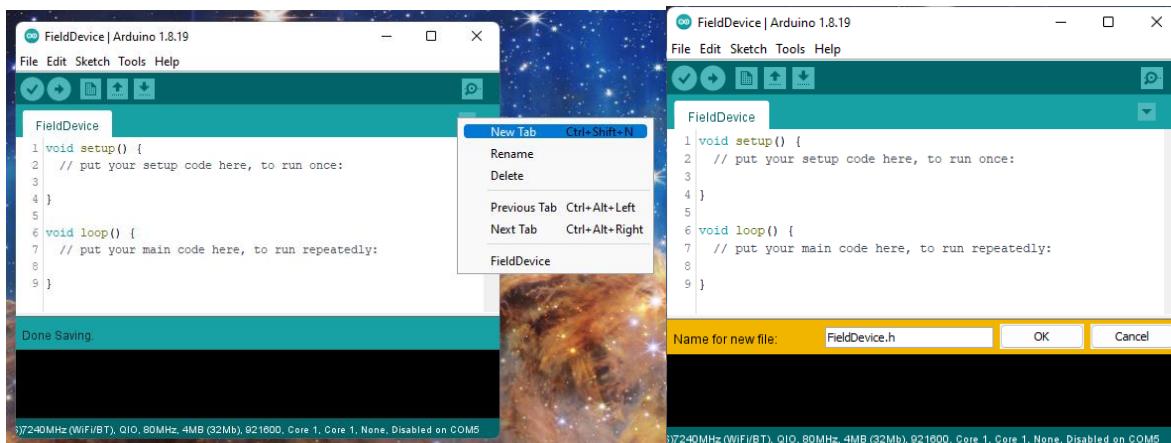
Field Device

The role of the device is to collect data from sensors and transmit that data over longer distances to a central device that has many field devices connected to it. The field device will be built similar to the previous lab section.

Open the Arduino IDE and create a new sketch. Name it **FieldDevice** and save it in the **Lab03/Lab03B** folder.



Next, create a header file to store constants, singleton classes, and configuration variables.



Firstly, in the header file, include the following libraries:

Lab 3 - Communicating Sensor Data over a LoRa Network

```
FieldDevice FieldDevice.h
1 //Libraries
2 #include <Arduino.h>
3 #include <Wire.h>
4 #include <AsyncAPDS9306.h>
5 #include "EBYTE.h"
6 #include <HardwareSerial.h>
7
```

Next, define your pin data.

```
FieldDevice FieldDevice.h
7
8 //Pin definitions
9 #define PIN_RX 16
10 #define PIN_TX 17
11 #define PIN_M0 27
12 #define PIN_M1 26
13 #define PIN_AX 25
14
15 //Sample frequency
16 #define DELAY_BETWEEN_SAMPLES_MS 5000
17
```

Next, configure your LoRa object.

```
FieldDevice FieldDevice.h §
18 //Transceiver setup
19 #define TRANSCEIVER_CHANNEL 42
20 EBYTE Transceiver(&Serial2, PIN_M0, PIN_M1, PIN_AX);
21
```

Set up your device information.

```
30
31 //Device information
32 String groupName = "GroupA";
33 String deviceName = "DeviceA";
34
```

Set up your sensors.

```
34
35 //Sensor IIC addresses
36 #define ADDR (byte) (0x40)
37 #define TMP_CMD (byte) (0xF3)
38
39 //Instantiating sensor object and configuration
40 AsyncAPDS9306 lightSensor;
41 const APDS9306_ALS_GAIN_t aGain = APDS9306_ALS_GAIN_1;
42 const APDS9306_ALS_MEAS_RES_t aTime = APDS9306_ALS_MEAS_RES_16BIT_25MS;
43
```

Next, include this header file in your implementation file.

FieldDevice FieldDevice.h

```
1 #include "FieldDevices.h"
2
3 void setup() {
4     // put your setup code here, to run once:
5
6 }
7
```

In the **setup()** function, initialize your transceiver.

```
FieldDevice FieldDevice.h §
1 #include "FieldDevice.h"
2
3 void setup() {
4     Serial.begin(9600);
5     Serial.print("\n\n-----\n"
6     + groupName + " : " + deviceName + "\n-----\n\n");
7
8     Wire.begin();
9     Wire.beginTransmission(ADDR);
10    Wire.endTransmission();
11    delay(300);
12
13    lightSensor.begin(aGain, aTime);
14
15    Serial.println("Ready for LoRa connection!");
16
17    Serial2.begin(9600);
18
19    Transceiver.init();
20    Transceiver.SetChannel(TRANSCEIVER_CHANNEL);
21    Transceiver.PrintParameters();
22
23 }
```

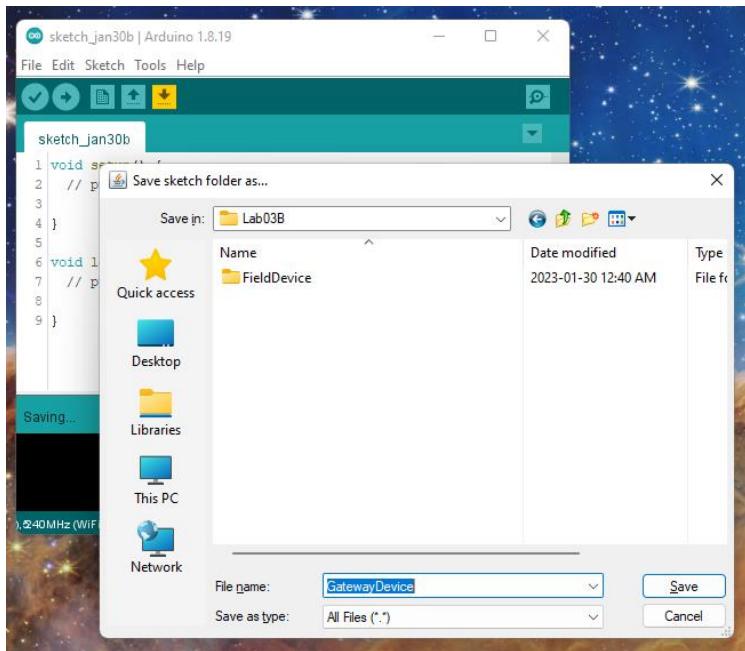
In the **loop()** function, sample the sensors periodically, aggregate the data into a JSON message, and transmit that message over LoRa.

```
FieldDevice  FieldDevice.h
26
27 void loop() {
28
29     if(millis() - startTime > DELAY_BETWEEN_SAMPLES_MS) {
30
31         //Temp sensor
32         Wire.beginTransmission(ADDR);
33         Wire.write(TMP_CMD);
34         Wire.endTransmission();
35         delay(100);
36
37         Wire.requestFrom(ADDR, 2);
38
39         char data[2];
40         if(Wire.available() == 2){
41             data[0] = Wire.read();
42             data[1] = Wire.read();
43         }
44
45         float temp = ((data[0] * 256.0) + data[1]);
46         float tempC = ((175.72 * temp) / 65536.0) - 46.85;
47         Serial.println("Temperature: " + String(tempC) + " degC");
48
49         //Sample light sensor
50         AsyncAPDS9306Data lightData = lightSensor.syncLuminosityMeasurement();
51
52         //Calculate luminosity
53         float lux = lightData.calculateLux();
54         Serial.println("Luminosity: " + String(lux) + " Lux");
55
56
57         //Format data as a JSON string
58         String sendData = "{ \"": + groupName + "\": { \"": + deviceName + "\": { \"": + "Temp\": \""
59             + String(tempC) + "\", \"": + "Luminosity\": \"" + String(lux) + "\" } } }" + '\n';
60
61         Serial.println("Prepared LoRa message: " + sendData);
62
63         Serial2.println(sendData);
64         Serial.println("LoRa sent!");
65
66     }
67
68 }
69 }
```

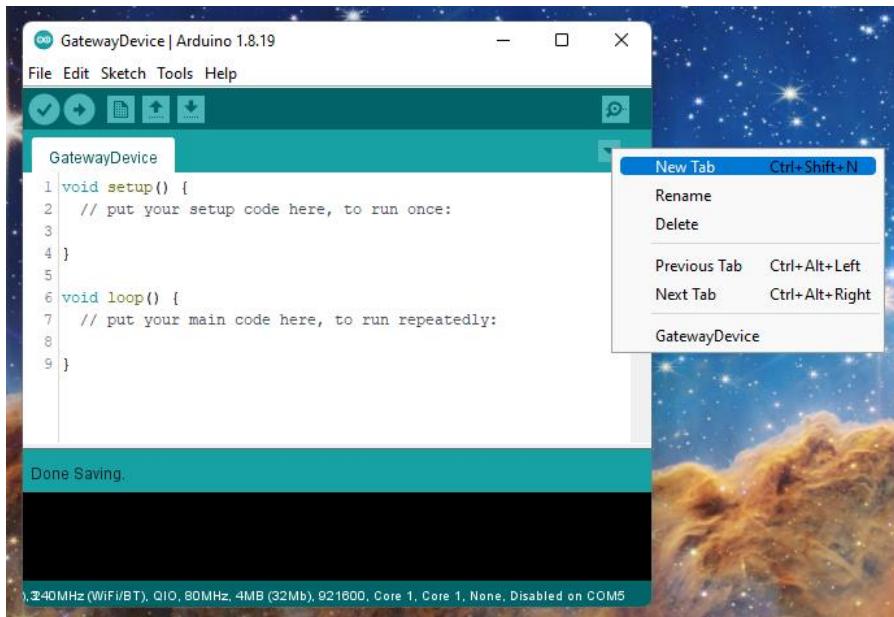
Gateway Device

The role of the gateway device is to receive data from multiple field devices and transmit them to a server that can further process the data.

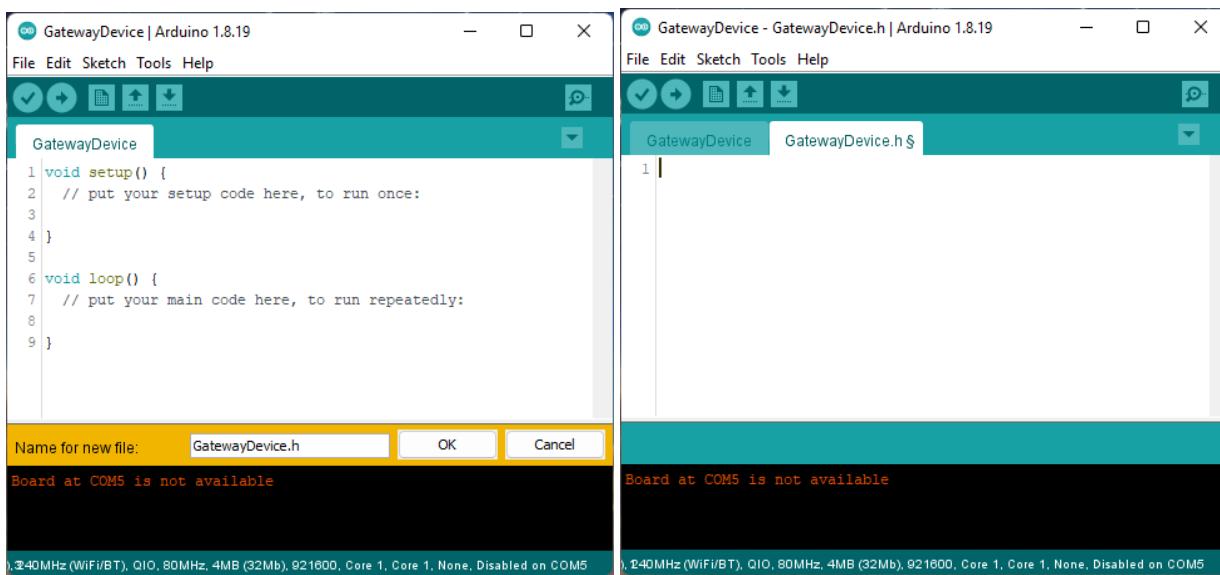
Start off by creating a new sketch and naming it **GatewayDevice**. Save it in the **4ID3/Lab03** folder.



Next, create a header file to include any global variables or objects.



Lab 3 - Communicating Sensor Data over a LoRa Network



Set up the header file like this:

The image shows a single instance of the Arduino IDE with the header file 'GatewayDevice.h' open. The tab bar at the top shows 'GatewayDevice' and 'GatewayDevice.h §'. The code in the editor is as follows:

```
1 //Libraries  
2 #include "EBYTE.h"  
3 #include <WiFi.h>  
4 #include <HTTPClient.h>  
5 #include <HardwareSerial.h>  
6  
7 //Pin definitions  
8 #define PIN_RX 16  
9 #define PIN_TX 17  
10 #define PIN_M0 27  
11 #define PIN_M1 26  
12 #define PIN_AX 25  
13  
14 //Transceiver setup  
15 #define TRANSCEIVER_CHANNEL 42  
16 EBYTE Transceiver(&Serial2, PIN_M0, PIN_M1, PIN_AX);  
17  
18 //WiFi login credentials  
19 const char* ssid = "GroupA";  
20 const char* password = "12345678";  
21  
22 //HTTP server URL  
23 const char* serverName = "http://192.168.137.38:3000/mqtt";  
24 //const char* serverName = "http://192.168.137.38:3000/database";  
25
```

Lab 3 - Communicating Sensor Data over a LoRa Network

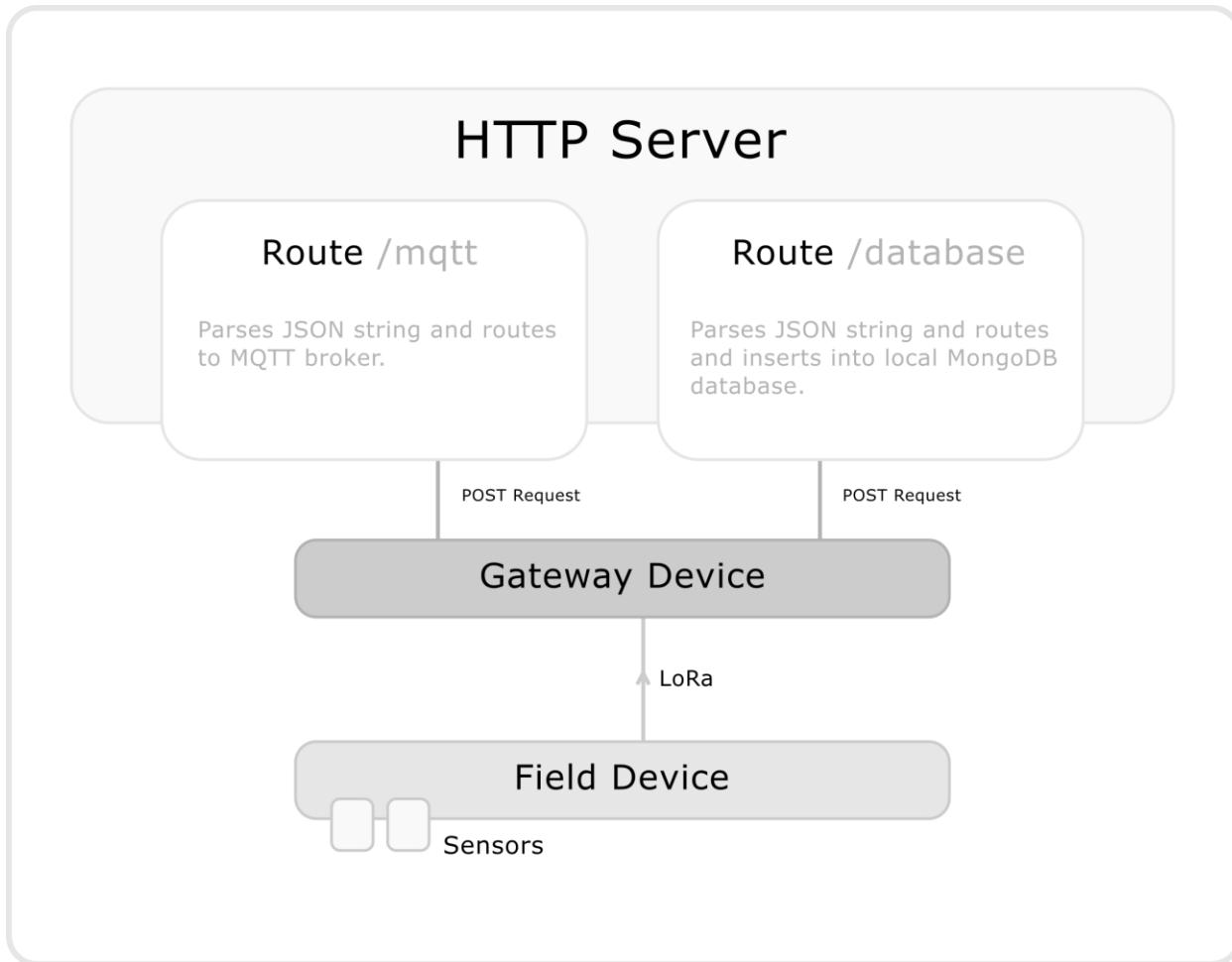
Next, in the implementation file, we want to poll if the LoRa serial receive buffer has data in it. If so, read that data and make a POST request to the server.

```
GatewayDevice  GatewayDevice.h §
1 #include "GatewayDevice.h"
2
3 void setup() {
4     Serial.begin(9600);
5     Serial2.begin(9600);
6     Transceiver.init();
7     Transceiver.SetChannel(TRANSCEIVER_CHANNEL);
8     Transceiver.PrintParameters();
9
10    WiFi.begin(ssid, password);
11    Serial.println("Connecting");
12    while(WiFi.status() != WL_CONNECTED) {
13        delay(500);
14        Serial.print(".");
15    }
16    Serial.println("");
17    Serial.print("Connected to WiFi network with IP Address: ");
18    Serial.println(WiFi.localIP());
19
20 }
21
```

```
22 void loop() {  
23  
24     if (Serial2.available() > 1) {  
25         Serial.println("Receiving Data...");  
26         String incomingData = Serial2.readString();  
27         Serial.println("PING: " + incomingData);  
28  
29         if(WiFi.status()== WL_CONNECTED){  
30             WiFiClient client;  
31             HTTPClient http;  
32  
33             http.begin(client, serverName);  
34  
35             http.addHeader("Content-Type", "text/plain");  
36  
37             int httpResponseCode = http.POST(incomingData);  
38  
39             Serial.print("HTTP Response code: ");  
40             Serial.println(httpResponseCode);  
41  
42             http.end();  
43  
44     }  
45  
46 }  
47 }
```

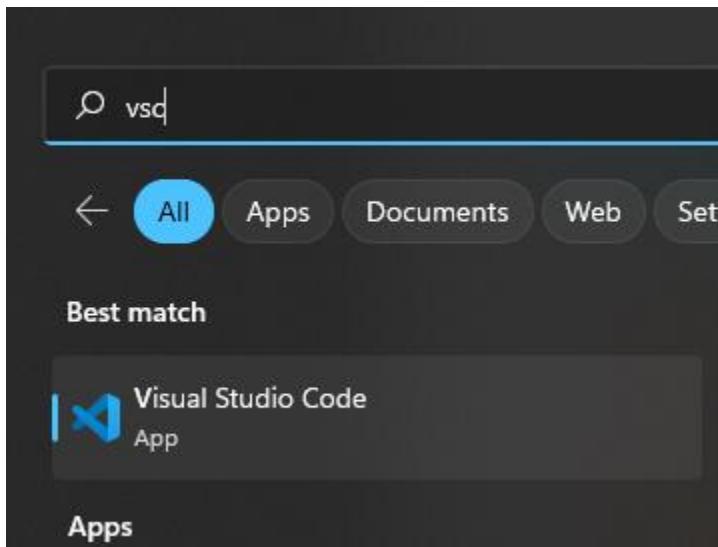
HTTP Server

The role of the HTTP server is to route the data that is being sent by the gateway to a set of pre-defined callback functions called **routes**.

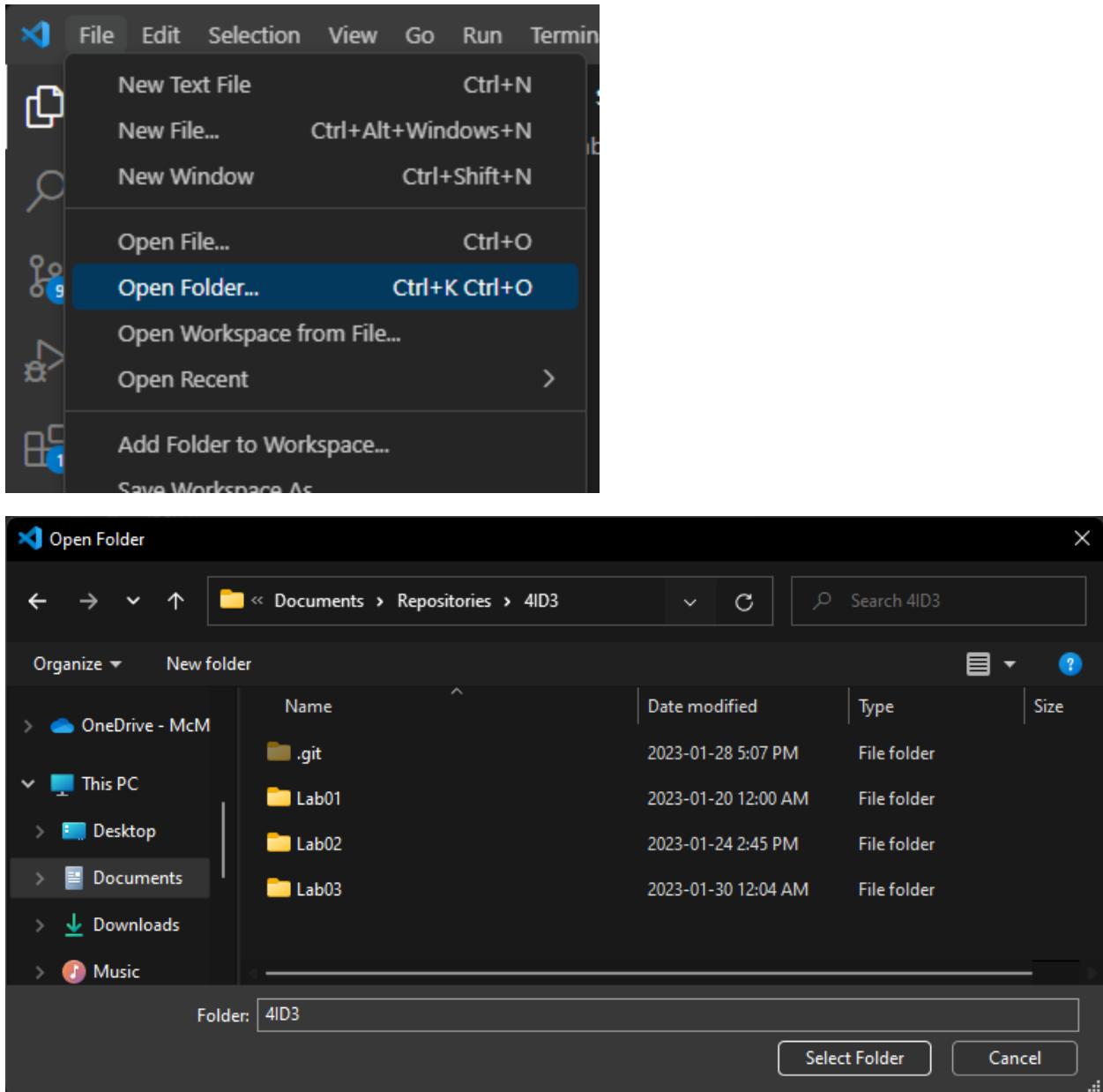


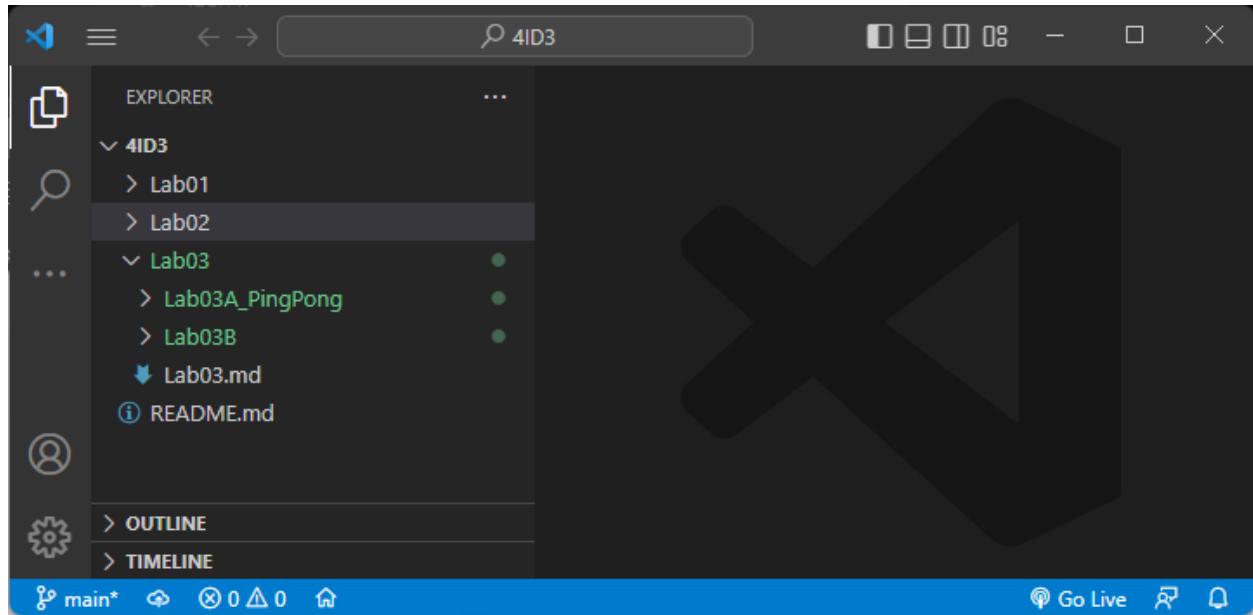
Repo Setup

Open **VSCode** and open your local repository in the project viewer.

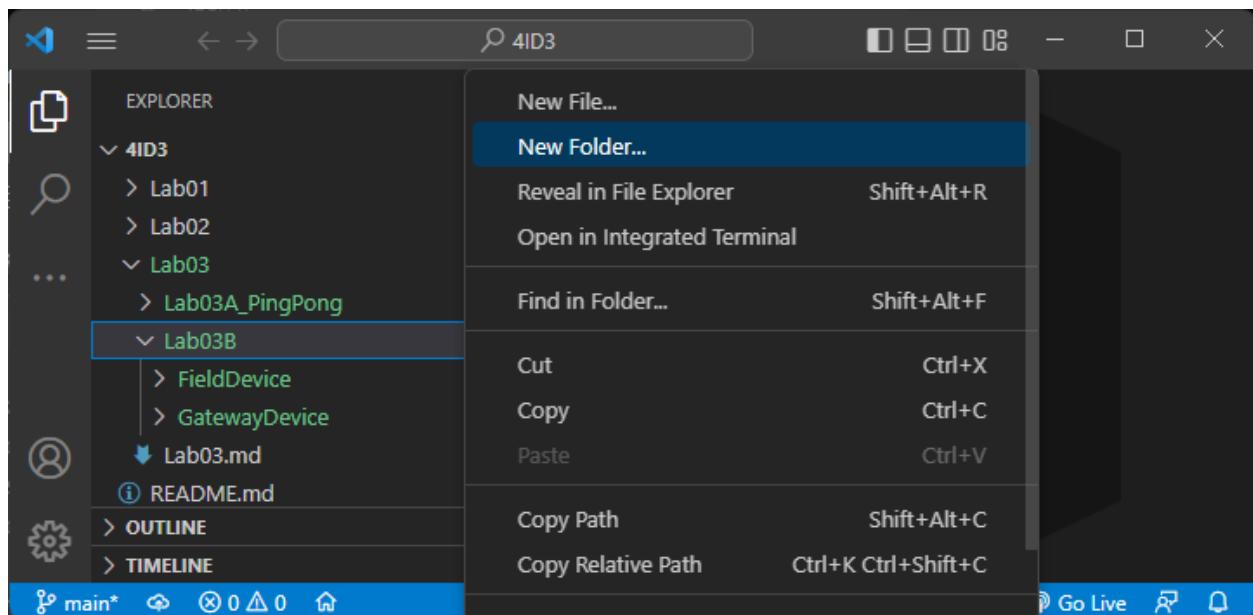


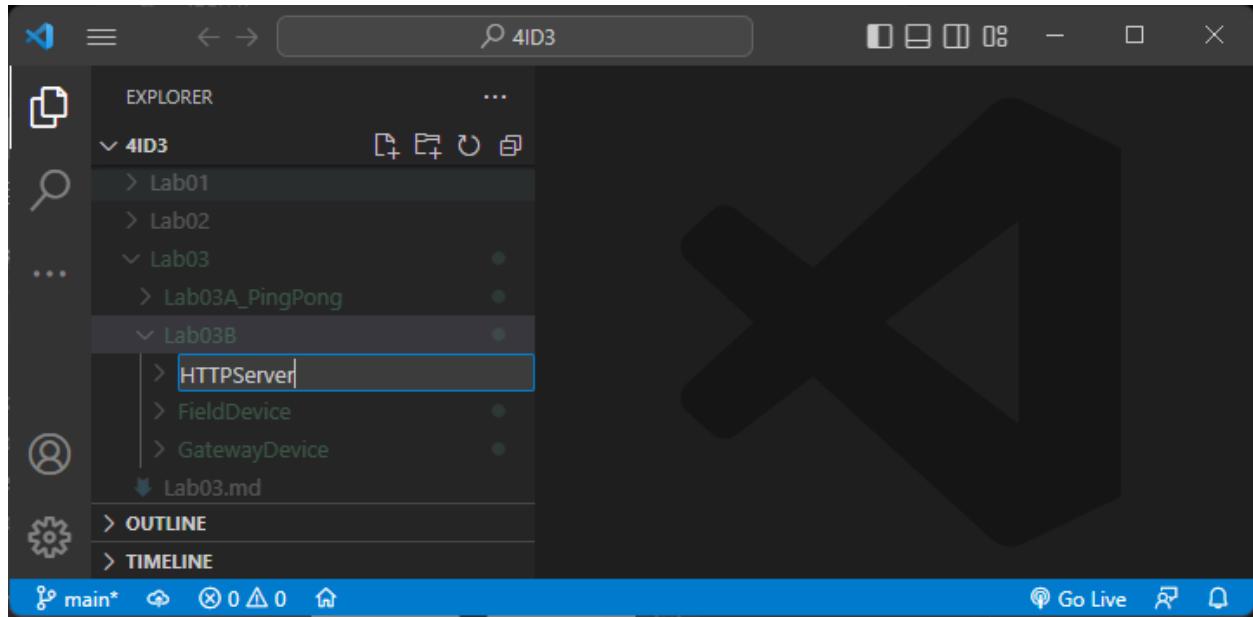
Lab 3 - Communicating Sensor Data over a LoRa Network



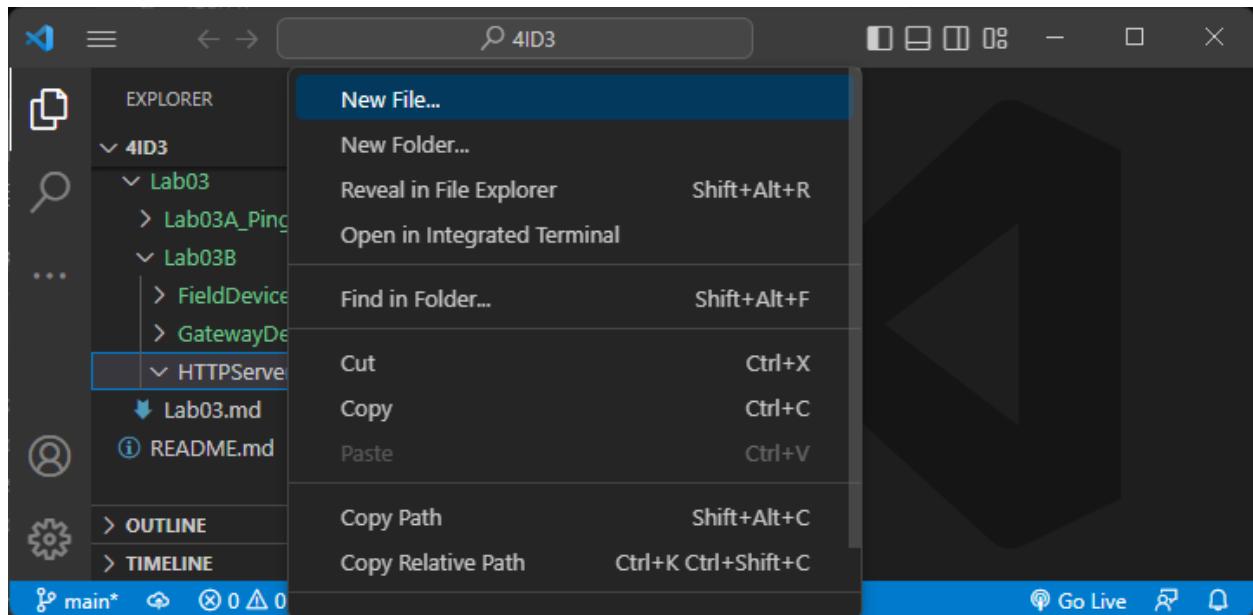


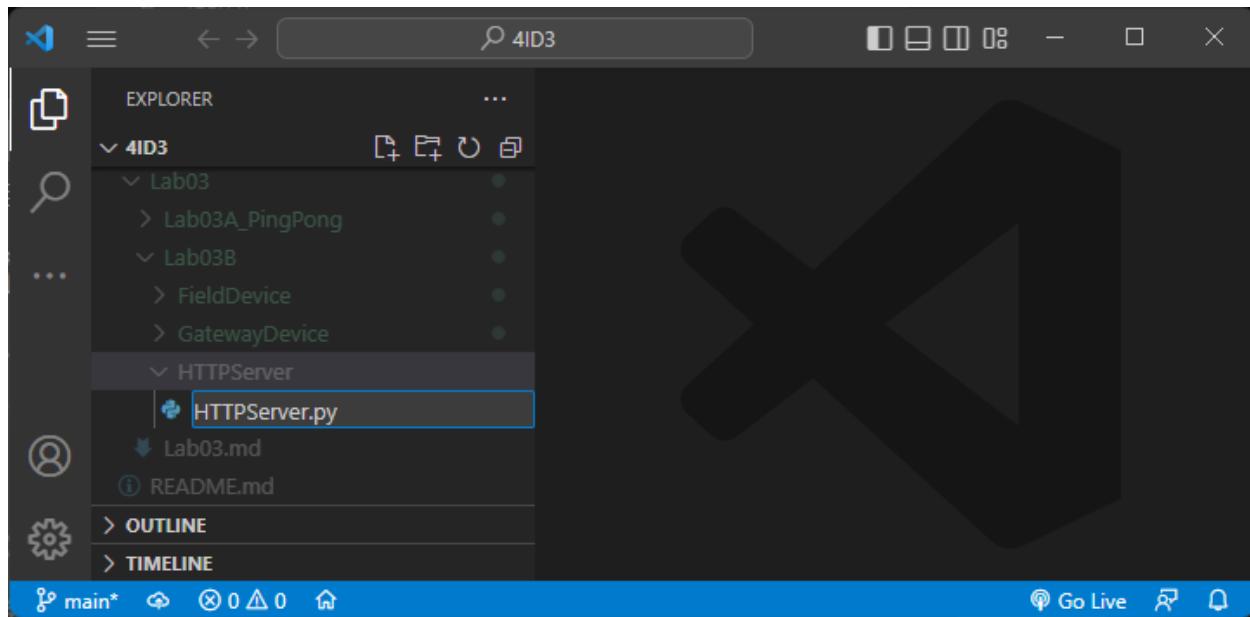
Create a new folder in the **4ID3/Lab03/Lab03B** directory called **HTTPServer.py**.





Create a new **python script** in this directory called **HTTPServer.py**.





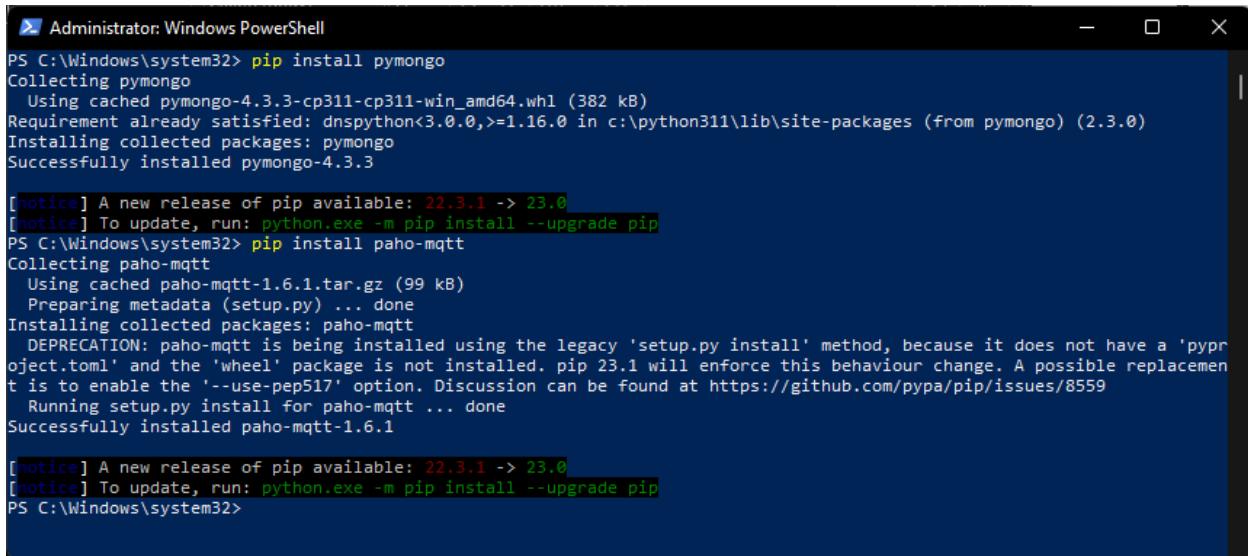
Python Libraries

Install the following libraries using the **pip** package manager using **PowerShell**, running as **Administrator**.

```
pip install pymongo
```

```
pip install paho-mqtt
```





```
Administrator: Windows PowerShell
PS C:\Windows\system32> pip install pymongo
Collecting pymongo
  Using cached pymongo-4.3.3-cp311-cp311-win_amd64.whl (382 kB)
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in c:\python311\lib\site-packages (from pymongo) (2.3.0)
Installing collected packages: pymongo
Successfully installed pymongo-4.3.3

[notice] A new release of pip available: 22.3.1 -> 23.0
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\Windows\system32> pip install paho-mqtt
Collecting paho-mqtt
  Using cached paho-mqtt-1.6.1.tar.gz (99 kB)
  Preparing metadata (setup.py) ... done
Installing collected packages: paho-mqtt
  DEPRECATION: paho-mqtt is being installed using the legacy 'setup.py install' method, because it does not have a 'pyproject.toml' and the 'wheel' package is not installed. pip 23.1 will enforce this behaviour change. A possible replacement is to enable the '--use-pep517' option. Discussion can be found at https://github.com/pypa/pip/issues/8559
    Running setup.py install for paho-mqtt ... done
Successfully installed paho-mqtt-1.6.1

[notice] A new release of pip available: 22.3.1 -> 23.0
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\Windows\system32>
```

IoT Server Overview

Return to the empty python script and begin by importing libraries and creating variables to store constant information.

```
HTTPServer.py U X
Lab03 > Lab03B > HTTPServer > HTTPServer.py > ...
1  #
2  #      Simple IoT Server
3  #      Adam Sokacz
4  #      2023 - 01 - 26
5  #
6
7  #      Libraries
8  from http.server import HTTPServer, BaseHTTPRequestHandler
9  import time
10 import json
11 import pymongo
12 import paho.mqtt.client as mqtt
13
14
15 #      HTTP Server
16 HTTP_IP = '0.0.0.0'
17 HTTP_PORT = 3000
18
19 #      MQTT Connection
20 MQTT_IP = 'test.mosquitto.org'
21 MQTT_PORT = 1883
22 MQTT_ROUTE = '/'
23
24 #      Database Connection
25 MONGODB_IP = "mongodb://localhost"
26 MONGODB_PORT = 27017
27 MONGODB_ROUTE = '/'
```

Set up your main function as such:

```
135
136 def main():
137     #Tuple that stores the HTTP server data
138     serverAddress = (HTTP_IP, HTTP_PORT)
139     #Instantiate the server object
140     server = HTTPServer(serverAddress, requestHandler)
141     #Print useful data to the terminal
142     print('\n\n-----\
143         \nSimple HTTP IoT Server\n---\
144         -----\\n\\n')
145     print(f'HTTP server running on {HTTP_IP} port {HTTP_PORT}')
146     time.sleep(1.56)
147     print('Routes active: \\n    -> \\\\mqtt \\n    -> \\\\database')
148     print(f"Connecting to MQTT -> {MQTT_IP}:{MQTT_PORT}")
149     time.sleep(1.12)
150     print(f"Connecting to Database -> {MONGODB_IP}:{MONGODB_PORT}{MONGODB_ROUTE}")
151     time.sleep(1.12)
152     print("\n\\nServer Ready\\n")
153     #Serve the page until the thread exits
154     server.serve_forever()
155
156 if __name__ == '__main__':
157     main()
158
```

Notice that the HTTP server requires a callback function to be passed into it on object creation. This function handles what to do when a device or computer connects to the server. It should be implemented above.

First, implement the GET requests. There are only a couple here for testing and to view the most recent data.

```
56 # Handles when a device makes a POST or GET request to HTTP server
57 class requestHandler(BaseHTTPRequestHandler):
58
59     def do_GET(self):
60         global uiDict
61         if self.path.endswith('/ui'):
62             self.send_response(200)
63             self.end_headers()
64             out = ''
65             groupName = list(uiDict.keys())[0]
66             deviceId = list(uiDict[groupName].keys())[0]
67             for key, val in uiDict[groupName][deviceId].items():
68                 out += f'<li>{key} - {val}</li>'
69
70             self.wfile.write(f'<html><body><h1>{groupName}</h1>\n
71             <h2>{deviceId}</h2><ul>{out}</ul></body></html>\n'.encode())
72
73         if self.path.endswith('/light'):
74             self.send_response(200)
75             self.end_headers()
76             self.wfile.write(f'{LIGHT}\n'.encode())
77
78         if self.path.endswith('/test'):
79             self.send_response(200)
80             self.send_header('content-type', "text/html")
81             self.end_headers()
82             self.wfile.write(f'<html><body><h1>OK\n</h1></body></html>'.encode())
83
84
85     def do_POST(self):
```

Next, set up the **/database route**, which looks at the payload, restructures it, and inserts it into the local database on your system.

```
84
85    def do_POST(self):
86        global uiDict
87        if self.path.endswith('/database'):
88            #Reading the data sent from microcontroller and formatting it
89            content_length = int(self.headers['Content-Length'])
90            data = self.rfile.read(content_length)
91            self.send_response(200)
92            self.end_headers()
93            data = str(data)
94            firstSplitIndex = data.find('{')
95            secondSplitIndex = data.rfind('}')
96            data = data[firstSplitIndex: secondSplitIndex+1]
97
98            #Interpreting microcontroller data as JSON
99            jDict = json.loads(data)
100           uiDict = jDict
101           groupName = list(jDict.keys())[0]
102           deviceId = list(jDict[groupName])[0]
103           mydb = myclient[groupName]
104           mycollection = mydb[deviceId]
105
106           #Inserting into database
107           ret = mycollection.insert_one(jDict[groupName][deviceId])
108
109           #Returning data + OK
110           self.wfile.write(f'{data}\nOK\n'.encode())
111
```

Next, set up the **/mqtt route**, which looks at the payload, restructures it, then publishes it to its corresponding MQTT path.

```
111
112     if self.path.endswith('/mqtt'):
113         #Reading the data sent from microcontroller and formatting it
114         content_length = int(self.headers['Content-Length'])
115         data = self.rfile.read(content_length)
116         self.send_response(200)
117         self.end_headers()
118         data = str(data)
119         firstSplitIndex = data.find('{')
120         secondSplitIndex = data.rfind('}')
121         data = data[firstSplitIndex: secondSplitIndex+1]
122
123         #Interpreting microcontroller data as JSON
124         jDict = json.loads(data)
125         uiDict = jDict
126         groupName = list(jDict.keys())[0]
127         deviceId = list(jDict[groupName])[0]
128
129         #Publishing to MQTT
130         for key, val in jDict[groupName][deviceId].items():
131             client.publish(f'{groupName}/{deviceId}/{key}', val.encode("UTF-8"))
132
133         #Returning data + OK
134         self.wfile.write(f'{data}\n'.encode())
135         self.wfile.write(b'OK\n')
136
```

Lastly, set up the MQTT and MongoDB objects earlier in the script.

```

29 LIGHT = 'OFF'
30
31 # Instantiating MQTT and callback functions
32 def on_connect(client, userdata, flags, rc):
33     print("Connected to " +str(rc))
34
35 def on_message(client, userdata, msg):
36     print(msg.topic+" "+str(msg.payload))
37     data = str(msg.payload.decode('utf-8'))
38
39     if data.strip() == "{\"Light\":\"ON\"}":
40         LIGHT = "ON"
41     elif data.strip() == "{\"Light\":\"OFF\"}":
42         LIGHT = "OFF"
43
44
45 client = mqtt.Client()
46 client.on_connect = on_connect
47 client.on_message = on_message
48 client.connect(MQTT_IP, MQTT_PORT, 60)
49 client.subscribe('Light', 2)
50
51 # Instantiating database connection
52 myclient = pymongo.MongoClient(f"{MONGODB_IP}:{MONGODB_PORT}{MONGODB_ROUTE}")
53
54 uiDict = dict({"uninit": {"uninit": {"uninit": "data"}}})
55
56 # Handles when a device makes a POST or GET request to HTTP server

```

Full code:

```

#
# Simple IoT Server
# Adam Sokacz
# 2023 - 01 - 26
#
# Libraries
from http.server import HTTPServer, BaseHTTPRequestHandler
import time
import json
import pymongo
import paho.mqtt.client as mqtt

# HTTP Server
HTTP_IP = '0.0.0.0'
HTTP_PORT = 3000

# MQTT Connection
MQTT_IP = 'test.mosquitto.org'

```

Lab 3 - Communicating Sensor Data over a LoRa Network

```
MQTT_PORT = 1883
MQTT_ROUTE = '/'

# Database Connection
MONGODB_IP = "mongodb://localhost"
MONGODB_PORT = 27017
MONGODB_ROUTE = '/'

LIGHT = 'OFF'

# Instantiating MQTT and callback functions
def on_connect(client, userdata, flags, rc):
    print("Connected to " + str(rc))

def on_message(client, userdata, msg):
    print(msg.topic + " " + str(msg.payload))
    data = str(msg.payload.decode('utf-8'))

    if data.strip() == "{\"Light\":\"ON\"}":
        LIGHT = "ON"
    elif data.strip() == "{\"Light\":\"OFF\"}":
        LIGHT = "OFF"

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect(MQTT_IP, MQTT_PORT, 60)
client.subscribe('Light', 2)

# Instantiating database connection
myclient = pymongo.MongoClient(f'{MONGODB_IP}:{MONGODB_PORT}{MONGODB_ROUTE}')

uiDict = dict({"uninit": {"uninit": {"uninit": "data"}}})

# Handles when a device makes a POST or GET request to HTTP server
class requestHandler(BaseHTTPRequestHandler):

    def do_GET(self):
        global uiDict
        if self.path.endswith('/ui'):
            self.send_response(200)
            self.end_headers()
            out = ""
            groupName = list(uiDict.keys())[0]
            deviceId = list(uiDict[groupName].keys())[0]
            for key, val in uiDict[groupName][deviceId].items():
                out += f'<li>{key} - {val}</li>'

            self.wfile.write(f'<html><body><h1>{groupName}</h1>
<h2>{deviceId}</h2><ul>{out}</ul></body></html>\n'.encode())

        if self.path.endswith('/light'):
            self.send_response(200)
            self.end_headers()
            self.wfile.write(f'{LIGHT}\n'.encode())

        if self.path.endswith('/test'):
            self.send_response(200)
            self.send_header('content-type', 'text/html')
            self.end_headers()
```

Lab 3 - Communicating Sensor Data over a LoRa Network

```
self.wfile.write(f'<html><body><h1>OK\n</h1></body></html>'.encode())

def do_POST(self):
    global uiDict
    if self.path.endswith('/database'):
        #Reading the data sent from microcontroller and formatting it
        content_length = int(self.headers['Content-Length'])
        data = self.rfile.read(content_length)
        self.send_response(200)
        self.end_headers()
        data = str(data)
        firstSplitIndex = data.find('{')
        secondSplitIndex = data.rfind('}')
        data = data[firstSplitIndex: secondSplitIndex+1]

        #Interpreting microcontroller data as JSON
        jDict = json.loads(data)
        uiDict = jDict
        groupName = list(jDict.keys())[0]
        deviceId = list(jDict[groupName])[0]
        mydb = myclient[groupName]
        mycollection = mydb[deviceId]

        #Inserting into database
        ret = mycollection.insert_one(jDict[groupName][deviceId])

        #Returning data + OK
        self.wfile.write(f'{data}\nOK\n'.encode())

    if self.path.endswith('/mqtt'):
        #Reading the data sent from microcontroller and formatting it
        content_length = int(self.headers['Content-Length'])
        data = self.rfile.read(content_length)
        self.send_response(200)
        self.end_headers()
        data = str(data)
        firstSplitIndex = data.find('{')
        secondSplitIndex = data.rfind('}')
        data = data[firstSplitIndex: secondSplitIndex+1]

        #Interpreting microcontroller data as JSON
        jDict = json.loads(data)
        uiDict = jDict
        groupName = list(jDict.keys())[0]
        deviceId = list(jDict[groupName])[0]

        #Publishing to MQTT
        for key, val in jDict[groupName][deviceId].items():
            client.publish(f'{groupName}/{deviceId}/{key}', val.encode("UTF-8"))

        #Returning data + OK
        self.wfile.write(f'{data}\n'.encode())
        self.wfile.write(b'OK\n')

def main():
    #Tuple that stores the HTTP server data
    serverAddress = (HTTP_IP, HTTP_PORT)
    #Instantiate the server object
    server = HTTPServer(serverAddress, requestHandler)
```

Lab 3 - Communicating Sensor Data over a LoRa Network

```
#Print useful data to the terminal
print('\n\n-----\
\nSimple HTTP IoT Server\n---\
-----\n\n')
print(f'HTTP server running on {HTTP_IP} port {HTTP_PORT}')
time.sleep(1.56)
print('Routes active: \n -> \\mqtt \n -> \\database')
print(f'Connecting to MQTT->{MQTT_IP}:{MQTT_PORT}')
time.sleep(1.12)
print(f'Connecting to Database -> {MONGODB_IP}:{MONGODB_PORT}/{MONGODB_ROUTE}')
time.sleep(1.12)
print("\n\nServer Ready\n")
#Serve the page until the thread exits
server.serve_forever()

if __name__ == '__main__':
    main()
```

Setting up the IoT Network

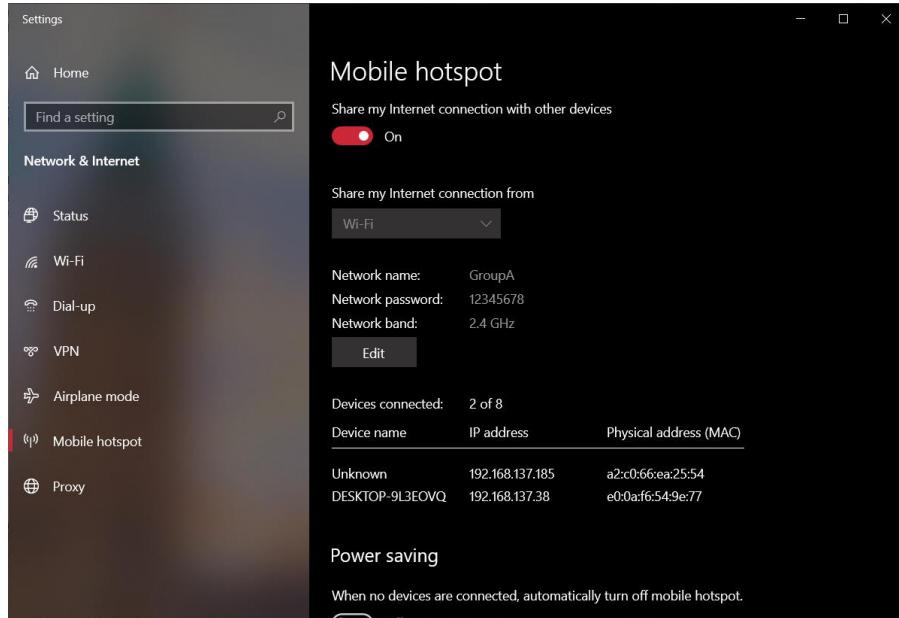
Have a **groupmate** use their computer to start a **Windows Mobile Hotspot**.

Add the login credentials to this hotspot into the **GatewayDevice** configuration and **upload** that to the MacIoT board.

Upload the **FieldDevice** sketch to the other MacIoT board.

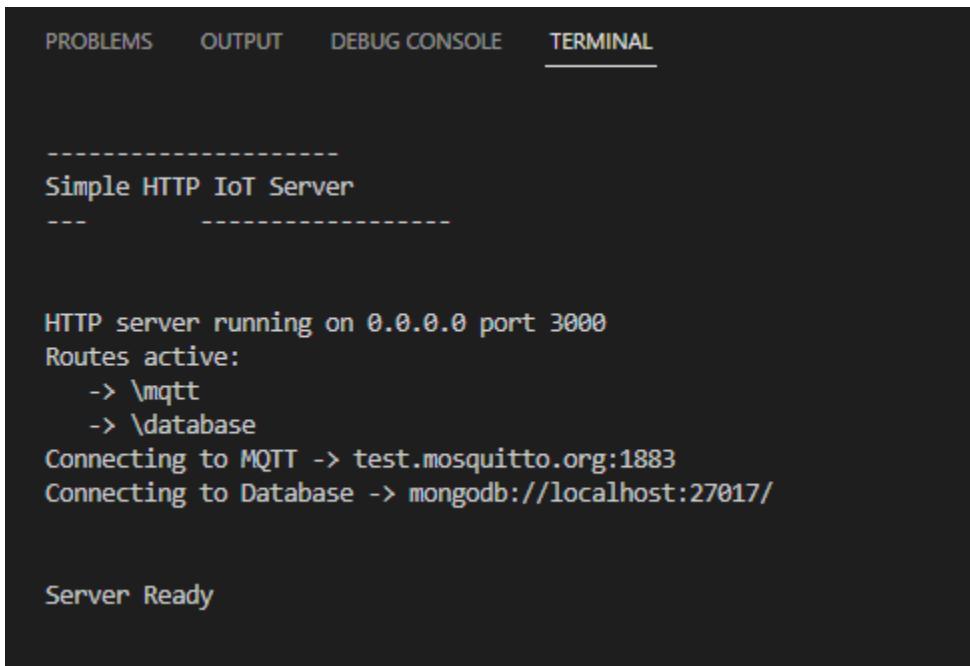
Connect your PC (*The PC running the HTTP server*) to the **same Mobile Hotspot** network.

In my case, the **IP address** for my **server PC** is **192.168.137.38**.



Next, launch the python script.

`python HTTPServer.py`



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

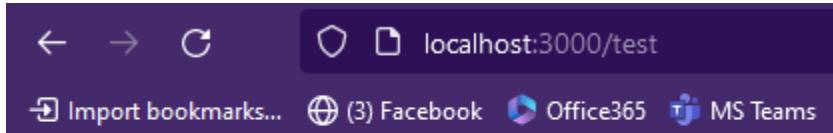
-----
Simple HTTP IoT Server
-----

HTTP server running on 0.0.0.0 port 3000
Routes active:
-> \mqtt
-> \database
Connecting to MQTT -> test.mosquitto.org:1883
Connecting to Database -> mongodb://localhost:27017/

Server Ready
```

In a web browser on your **server PC**, search the following URL:

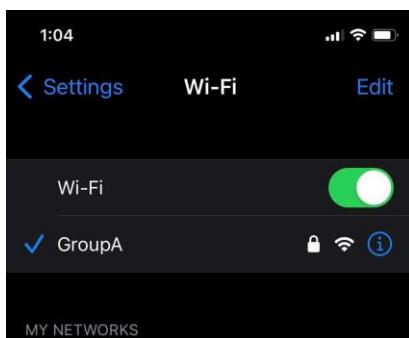
`localhost:3000/test`



OK

You should receive **OK** without any hassles.

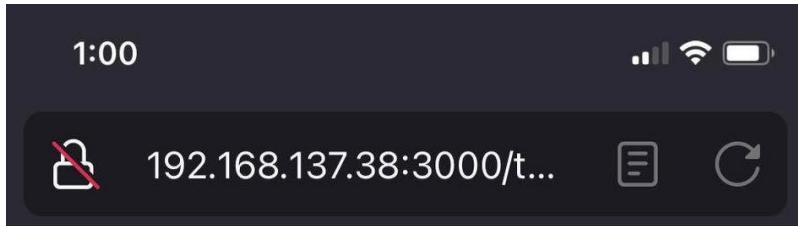
Using either a **mobile phone** or third PC, connect to your group's mobile hotspot.



Lab 3 - Communicating Sensor Data over a LoRa Network

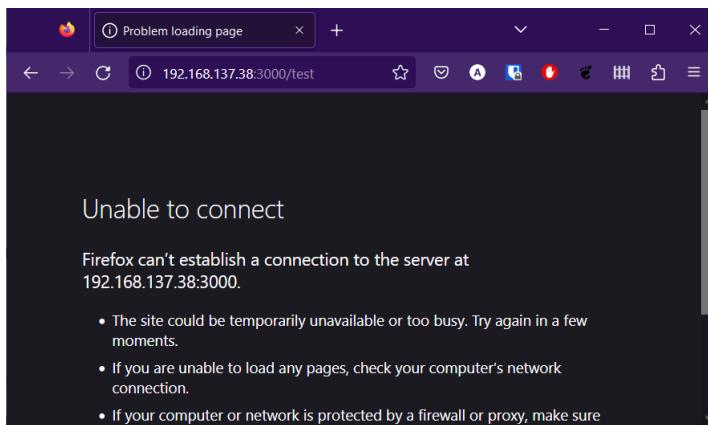
Type the IP address and path of your server URL and test route into a web browser on your mobile phone.

In my case, this would be: `192.168.137.38:3000/test`.



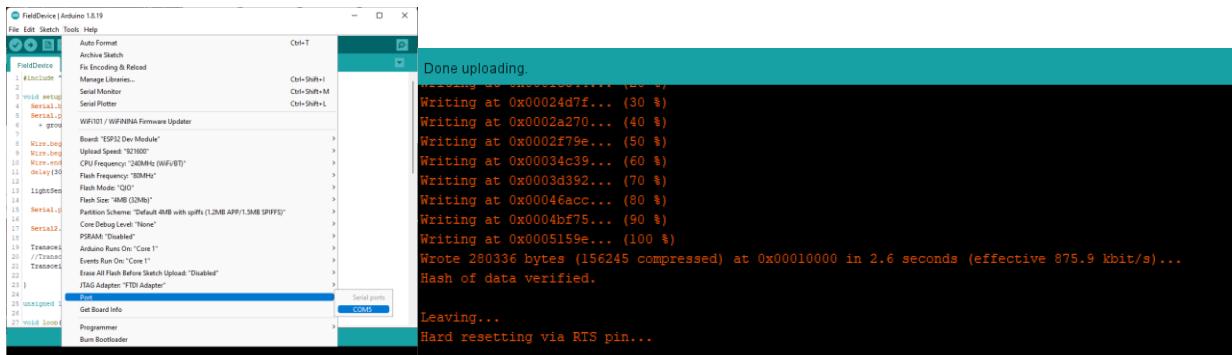
OK

The reason why we require a third device, is that the PC running the mobile hotspot cannot connect to any servers on its network for security reasons.

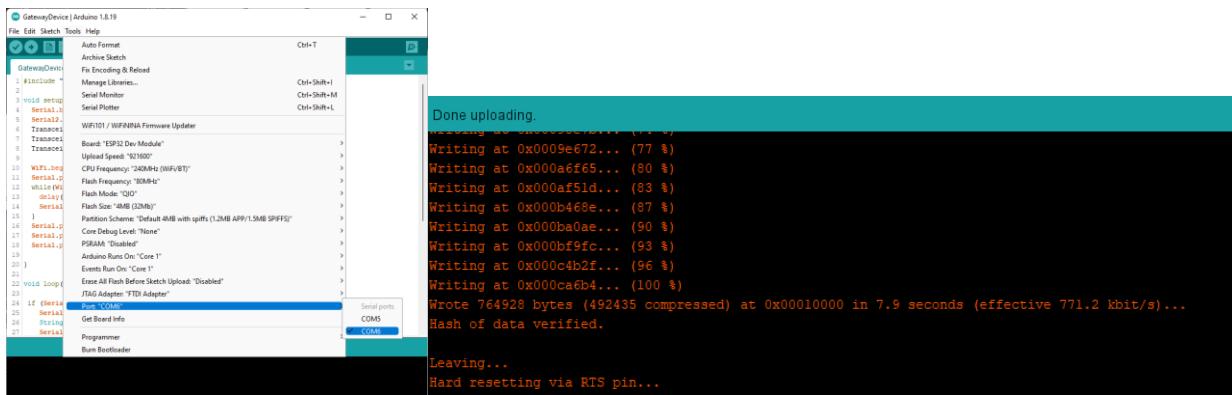


Integrating Everything Together

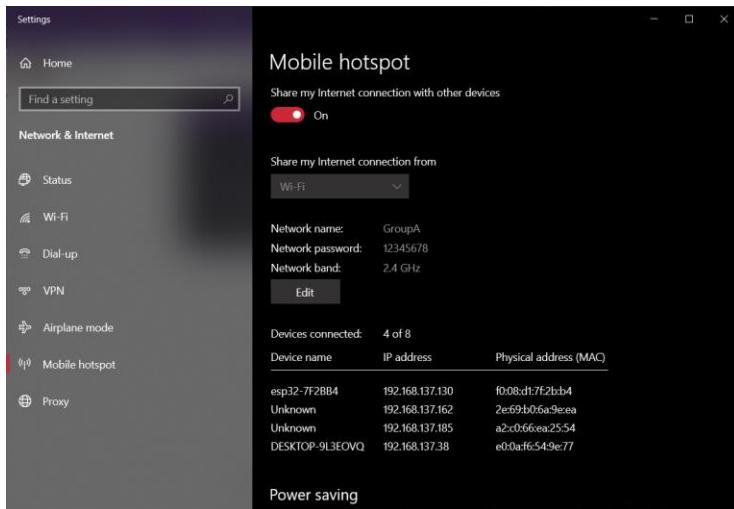
Now that each component of this IoT network is created, **upload** your **FieldDevice** and **GatewayDevice** **sketches** to two different MacIoT boards.



Lab 3 - Communicating Sensor Data over a LoRa Network



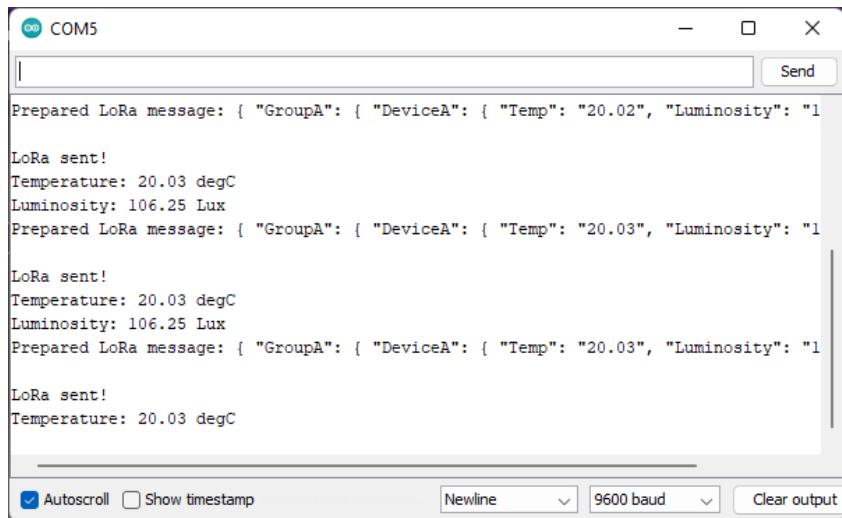
Watch the mobile hotspot PC and confirm that the **GatewayDevice** has connected successfully.



View each COM port and ensure that they are communicating through LoRa.

Field Device:

Lab 3 - Communicating Sensor Data over a LoRa Network

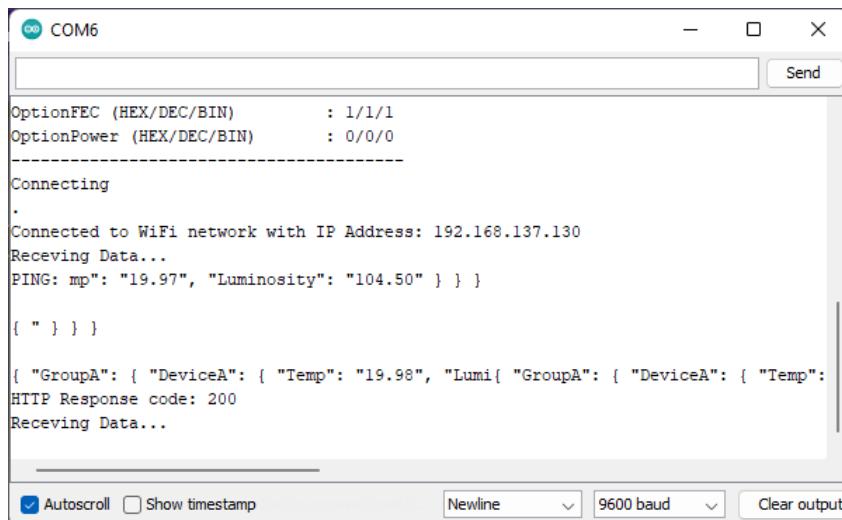


The screenshot shows a terminal window titled "COM5". The window has a "Send" button at the top right and a text area below it. The text area displays the following messages:

```
Prepared LoRa message: { "GroupA": { "DeviceA": { "Temp": "20.02", "Luminosity": "106.25" } } }
LoRa sent!
Temperature: 20.03 degC
Luminosity: 106.25 Lux
Prepared LoRa message: { "GroupA": { "DeviceA": { "Temp": "20.03", "Luminosity": "106.25" } } }
LoRa sent!
Temperature: 20.03 degC
Luminosity: 106.25 Lux
Prepared LoRa message: { "GroupA": { "DeviceA": { "Temp": "20.03", "Luminosity": "106.25" } } }
LoRa sent!
Temperature: 20.03 degC
```

At the bottom of the window, there are several configuration options: "Autoscroll" (checked), "Show timestamp" (unchecked), "Newline" dropdown, "9600 baud" dropdown, and a "Clear output" button.

Gateway Device:



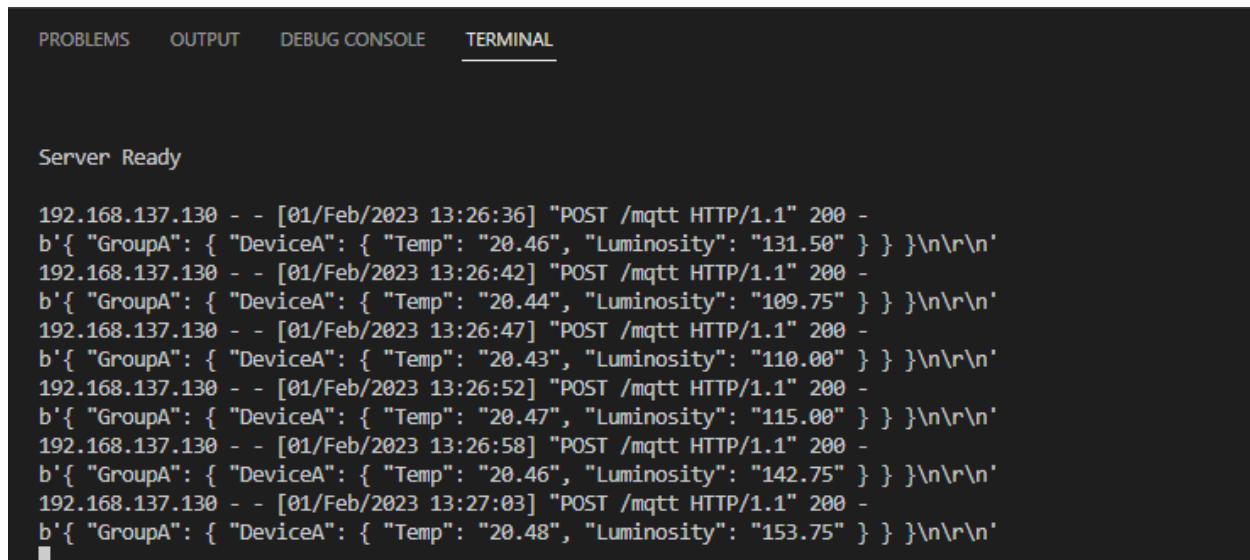
The screenshot shows a terminal window titled "COM6". The window has a "Send" button at the top right and a text area below it. The text area displays the following logs:

```
OptionFEC (HEX/DEC/BIN)      : 1/1/1
OptionPower (HEX/DEC/BIN)     : 0/0/0
-----
Connecting
.
Connected to WiFi network with IP Address: 192.168.137.130
Receiving Data...
PING: mp": "19.97", "Luminosity": "104.50" } } ]
{ " } } }

{ "GroupA": { "DeviceA": { "Temp": "19.98", "Lumi{ "GroupA": { "DeviceA": { "Temp": "19.97", "Luminosity": "104.50" } } }
HTTP Response code: 200
Receiving Data...
```

At the bottom of the window, there are several configuration options: "Autoscroll" (checked), "Show timestamp" (unchecked), "Newline" dropdown, "9600 baud" dropdown, and a "Clear output" button.

In the **server** script, you should see data printing.



The screenshot shows a terminal window with tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab is active, displaying the following text:

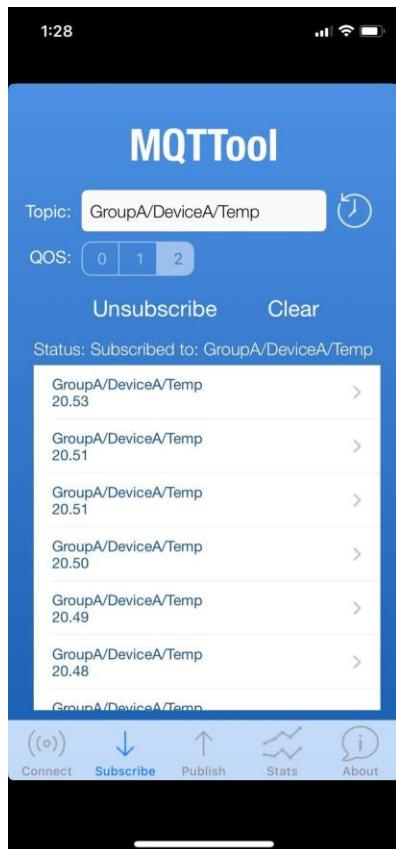
```
Server Ready

192.168.137.130 - - [01/Feb/2023 13:26:36] "POST /mqtt HTTP/1.1" 200 -
b'{ "GroupA": { "DeviceA": { "Temp": "20.46", "Luminosity": "131.50" } } }\n\r\n'
192.168.137.130 - - [01/Feb/2023 13:26:42] "POST /mqtt HTTP/1.1" 200 -
b'{ "GroupA": { "DeviceA": { "Temp": "20.44", "Luminosity": "109.75" } } }\n\r\n'
192.168.137.130 - - [01/Feb/2023 13:26:47] "POST /mqtt HTTP/1.1" 200 -
b'{ "GroupA": { "DeviceA": { "Temp": "20.43", "Luminosity": "110.00" } } }\n\r\n'
192.168.137.130 - - [01/Feb/2023 13:26:52] "POST /mqtt HTTP/1.1" 200 -
b'{ "GroupA": { "DeviceA": { "Temp": "20.47", "Luminosity": "115.00" } } }\n\r\n'
192.168.137.130 - - [01/Feb/2023 13:26:58] "POST /mqtt HTTP/1.1" 200 -
b'{ "GroupA": { "DeviceA": { "Temp": "20.46", "Luminosity": "142.75" } } }\n\r\n'
192.168.137.130 - - [01/Feb/2023 13:27:03] "POST /mqtt HTTP/1.1" 200 -
b'{ "GroupA": { "DeviceA": { "Temp": "20.48", "Luminosity": "153.75" } } }\n\r\n'
```

In your browser, navigate to: **ip_address:port/ui** to see the most recent parsed data.

Exercise A

Using a mobile phone, connect and view the data being published. Submit a screenshot with your report.



Exercise B

Create a NodeRED flow to visualize the data being published to **test.mosquitto.org**. Submit a screenshot with your report.

Exercise C

Modify the **URL** on the **Gateway Device** to make HTTP requests to the following URL instead of the MQTT route.

<server_ip>:3000/database

The setting can be found here:

```
26
27 //WiFi login credentials
28 const char* ssid = "GroupA";
29 const char* password = "12345678";
30
31 //HTTP server URL
32 const char* serverName = "http://192.168.137.38:3000/mqtt";
33 //Database Server at http://192.168.137.38:3000/database
```



View the data being added to your local database. Save a screenshot of the MongoDB Compass database viewer.

_id	Temp	Luminosity
ObjectId('63dab0c2add800b9c61259c6')	"20.81"	"83.25"
ObjectId('63dab0c2add800b9c61259c7')	"20.80"	"84.50"
ObjectId('63dab0c8add800b9c61259c8')	"20.81"	"85.75"

Lab 3 - Communicating Sensor Data over a LoRa Network

If you have time, plot this data in excel.

END