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:

W BOOTH SCHOOL OF ENGINEERING PRACTICE AND TECHNOLOGY

McMaster-Mohawk Bachelor of Technology Partnership

McMaster University ENGINEERING

MOHAWK

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

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/DeFST8tvtuI)

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? How does it differ from more sophisticated networks?

(Suggested: A few sentences)

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 an HTTP server 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 target. 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:

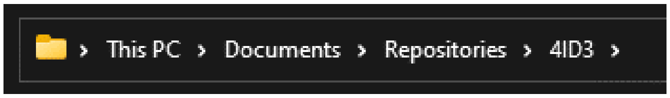
Exercise B Results:

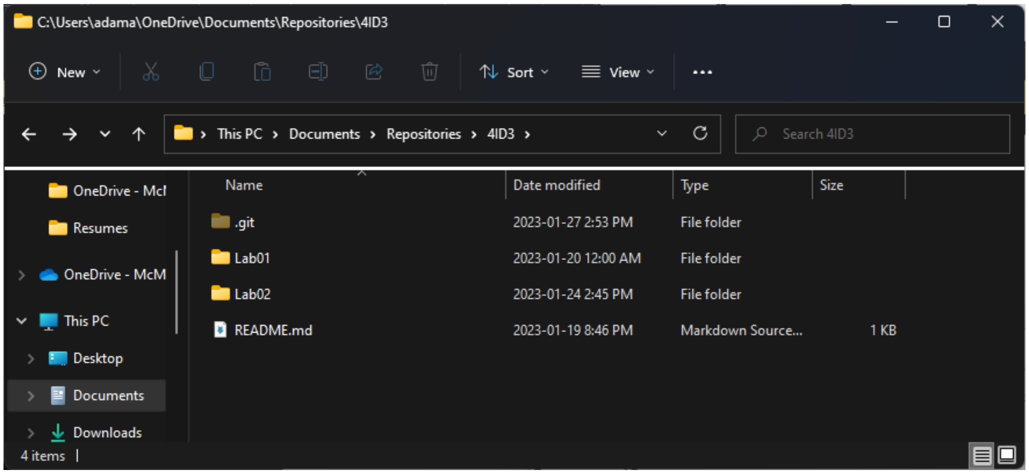
Exercise C Results:

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.

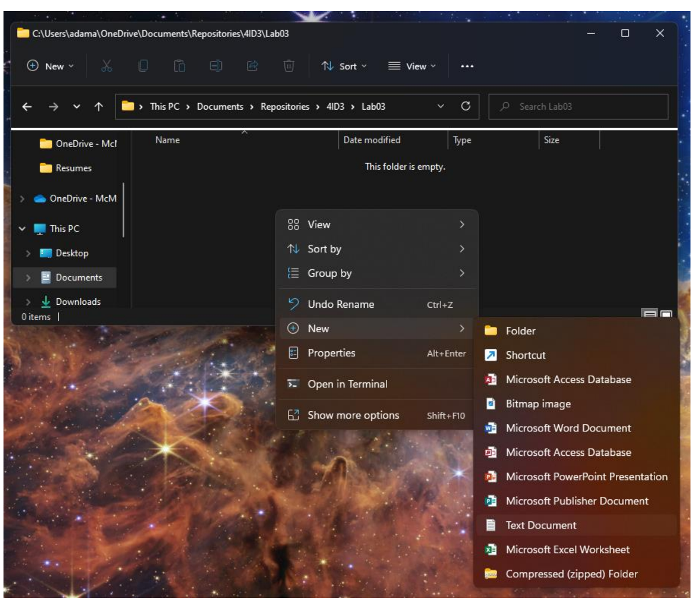
Navigate to your local git repository for this course.



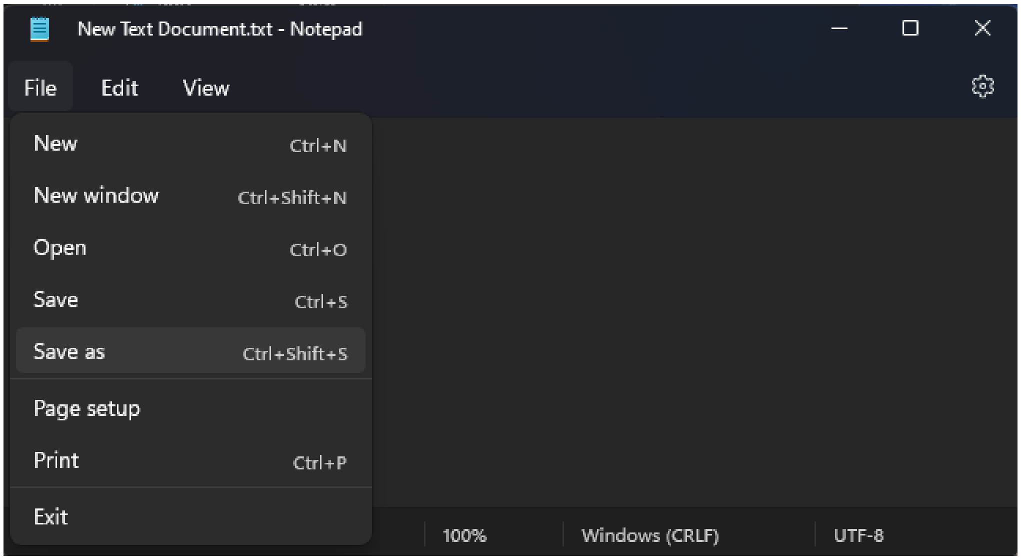


Create a new folder named **Lab02**. Navigate inside this folder.

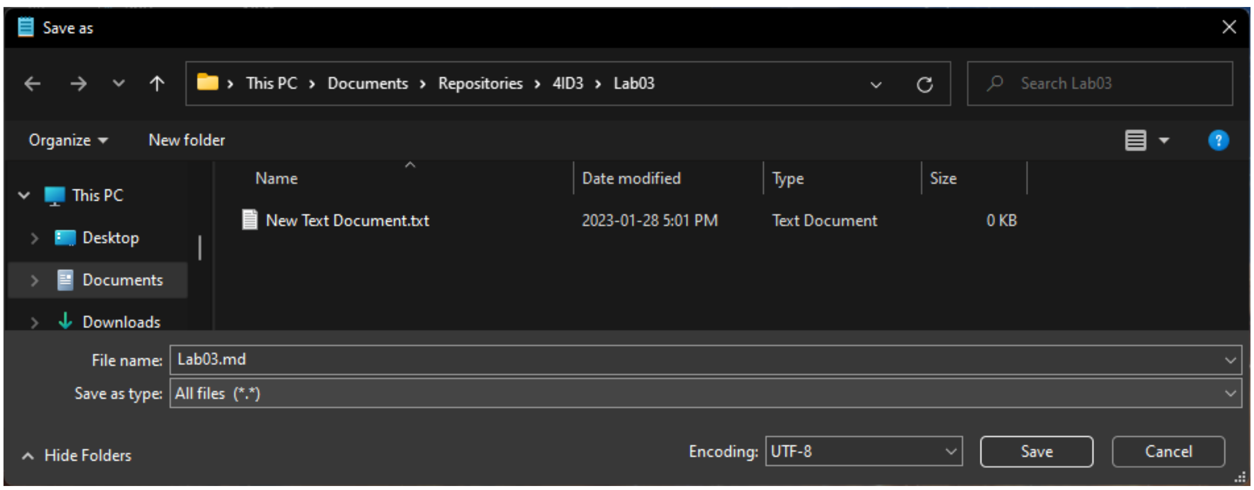
Create a new text file in the folder.



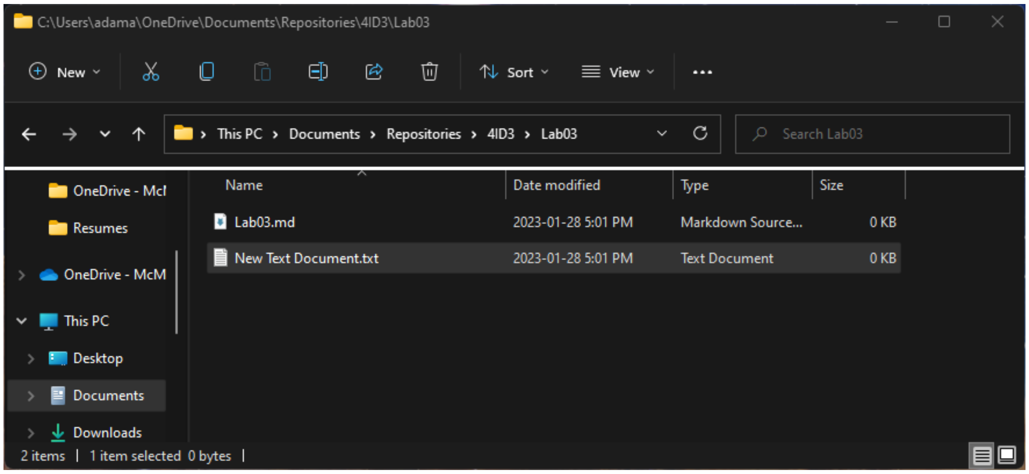
Press **File > Save as.**

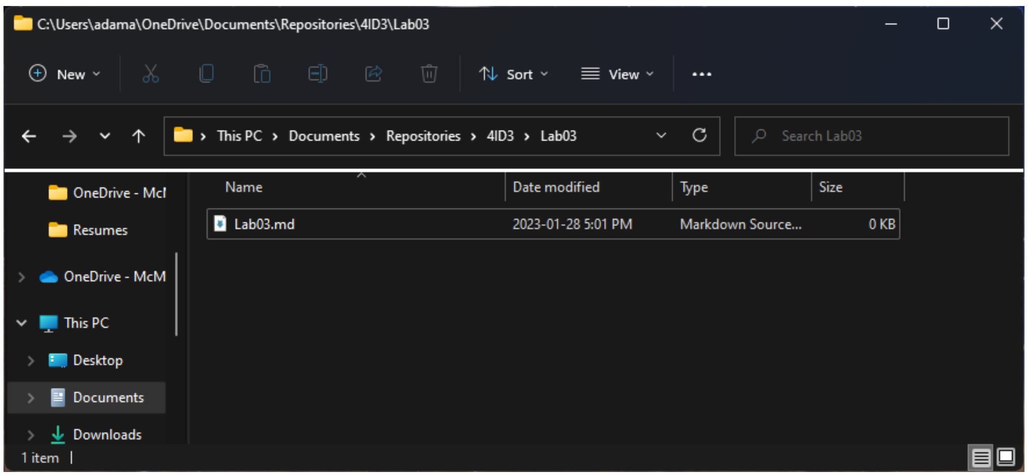


Save it as **Lab02.md**. Ensure that the **Save as type** is set to **All files (*.*)**.

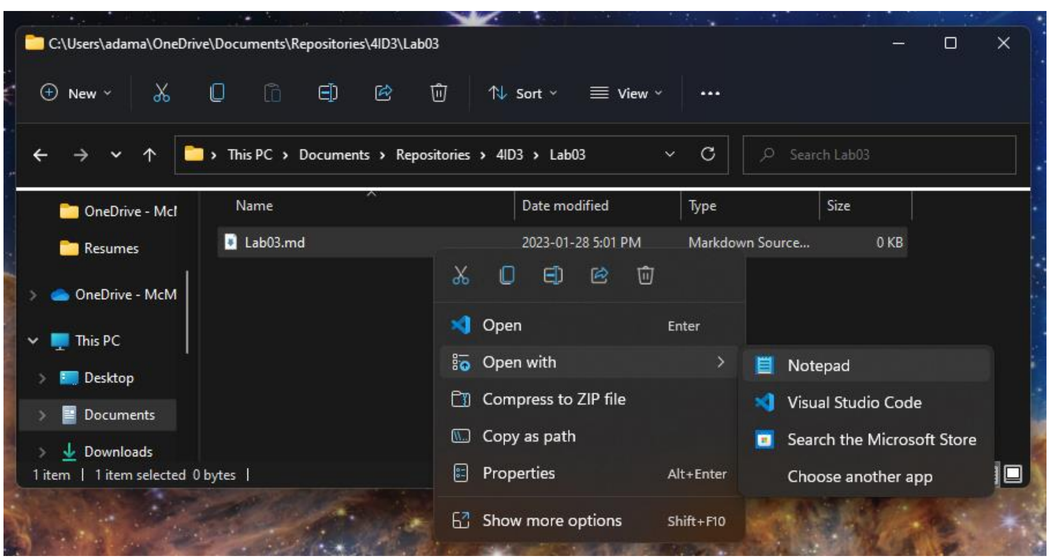


Now, you should have two files, a **text file** and a **markdown file**. Delete the text file.





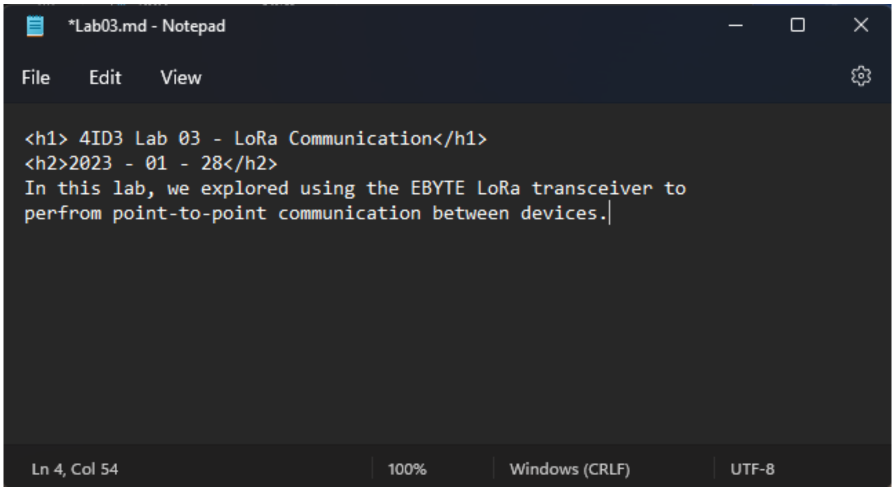
To open the markdown file, **right-click** and select **Open with**. Choose **Notepad**.



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

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

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



<h1> 4ID3 Lab 03 - LoRa Communication</h1>

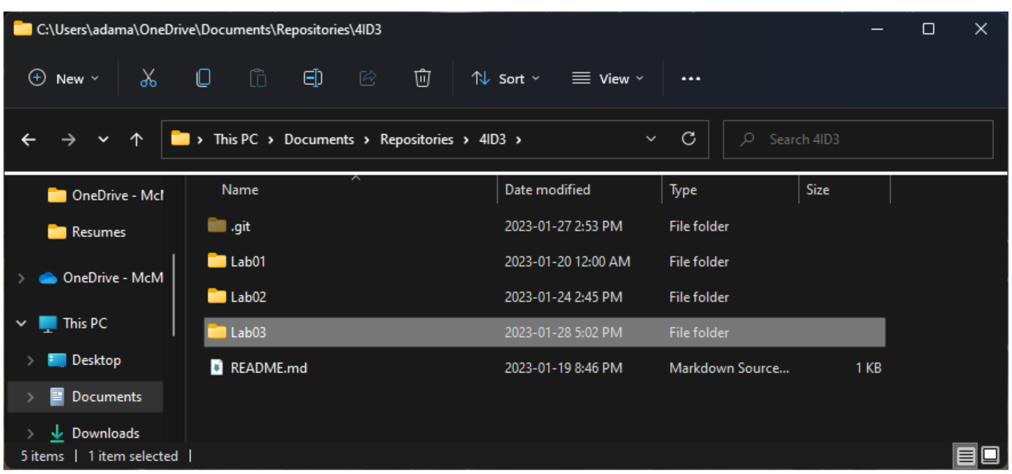
<h2>2023 - 01 - 28</h2>

In this lab, we explored using the EBYTE LoRa transceiver to

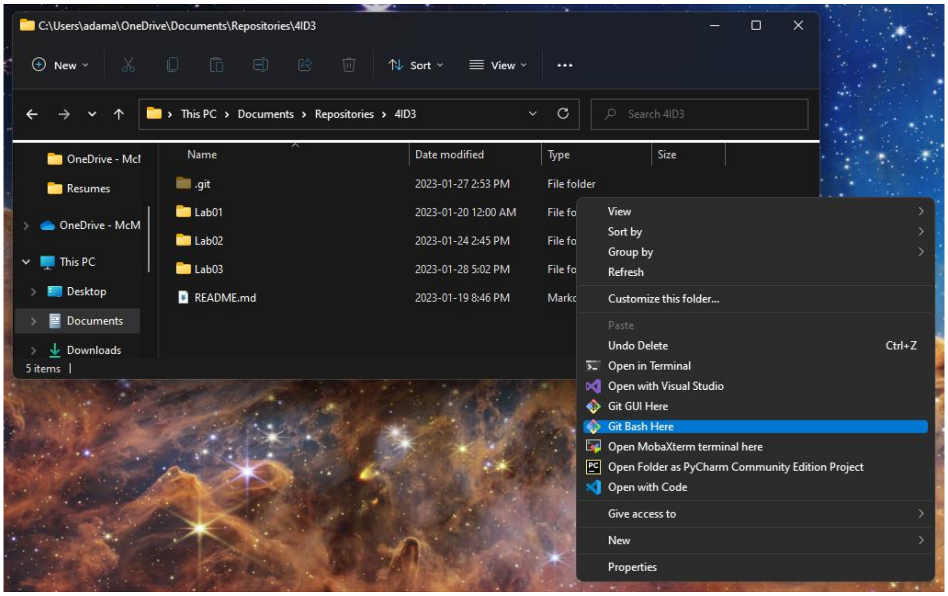
perform point-to-point communication between devices.

Save the file.

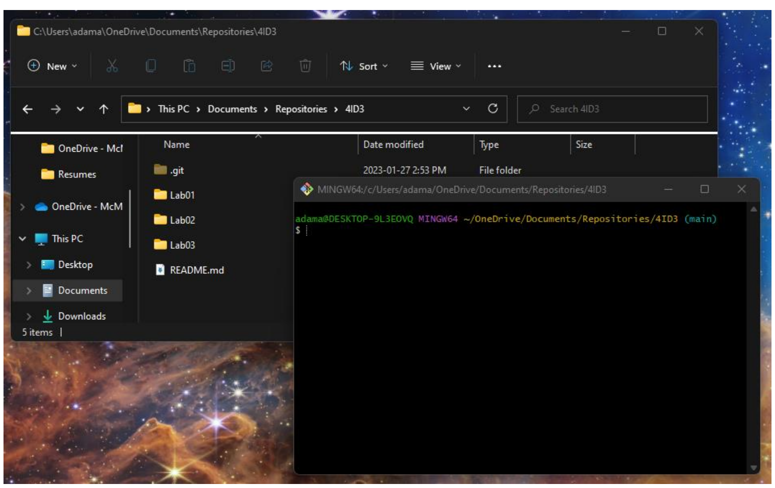
Navigate to the **root folder** (*4ID3/, root main highest-level folder*) of your local repository.



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



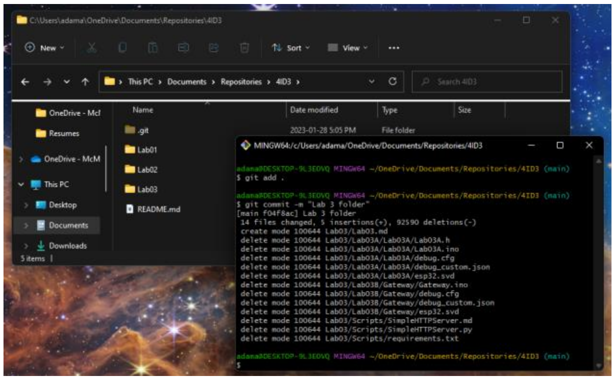
First, we need to add all the changes to the index that will be synced with GitHub. This will be done with the git add command.



git add .

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

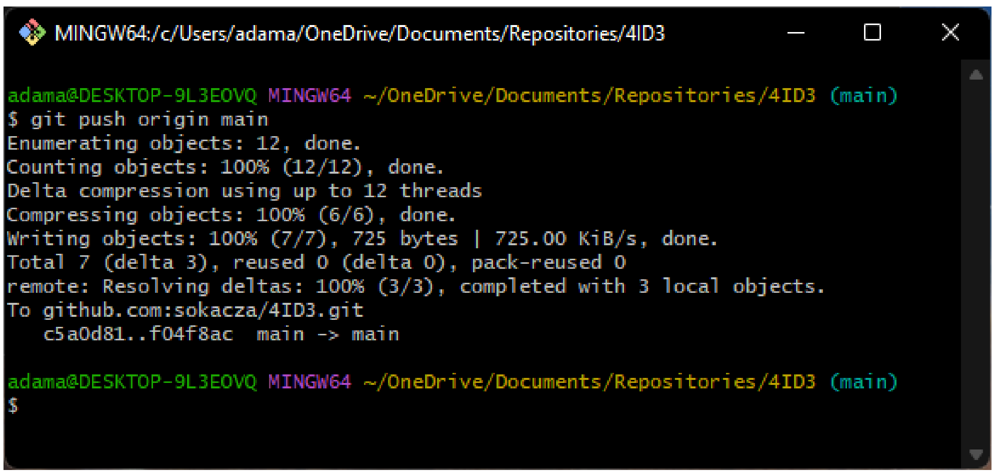
Next, when we are happy with the changes we chose to upload, we can use the commit command to package them to be synced.



git commit -m "Lab 3 folder"

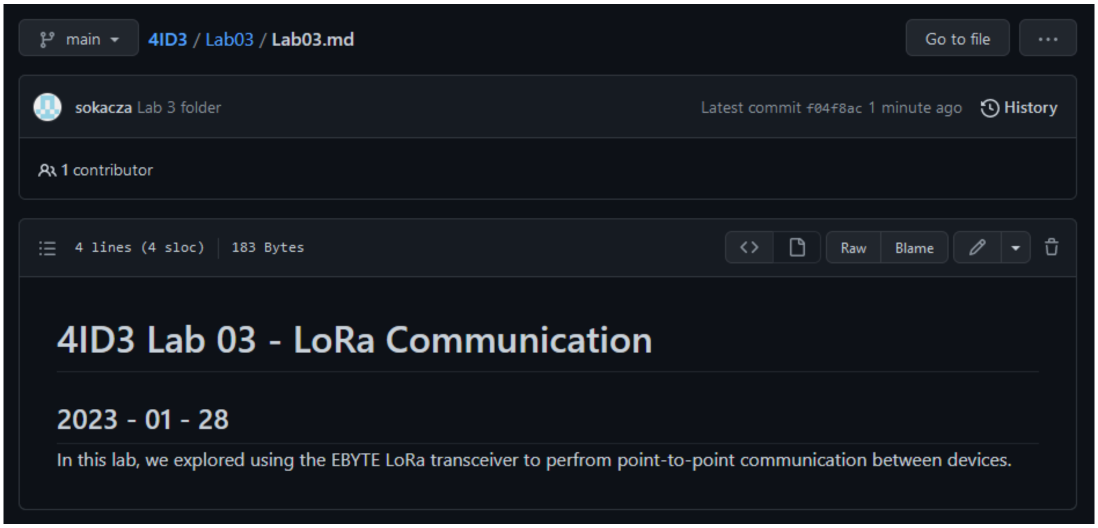
The '-m' flag stands for message, and it adds a message that explains what changes were made.

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



git push origin main

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



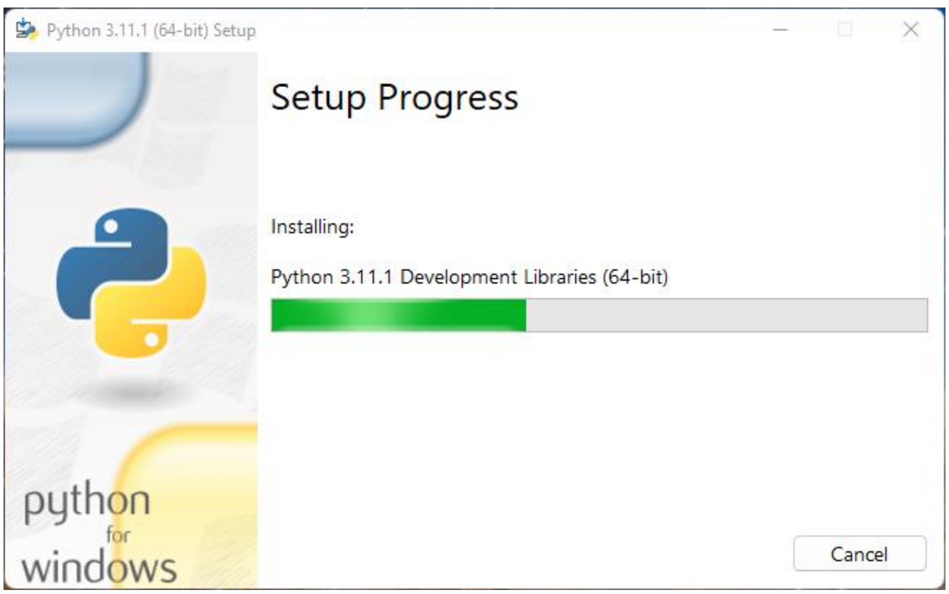
Now, if you are collaborating and wish to sync your local git repo with the remote GitHub repo, use the git pull command. In this case, we see that our local git repo is already up-to-date.

Installing Python

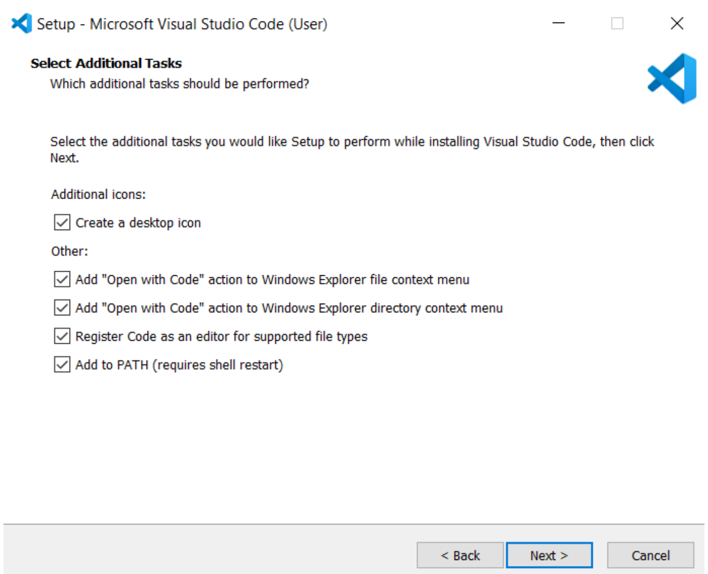
Navigate to the following website and install the latest version of Python 3.

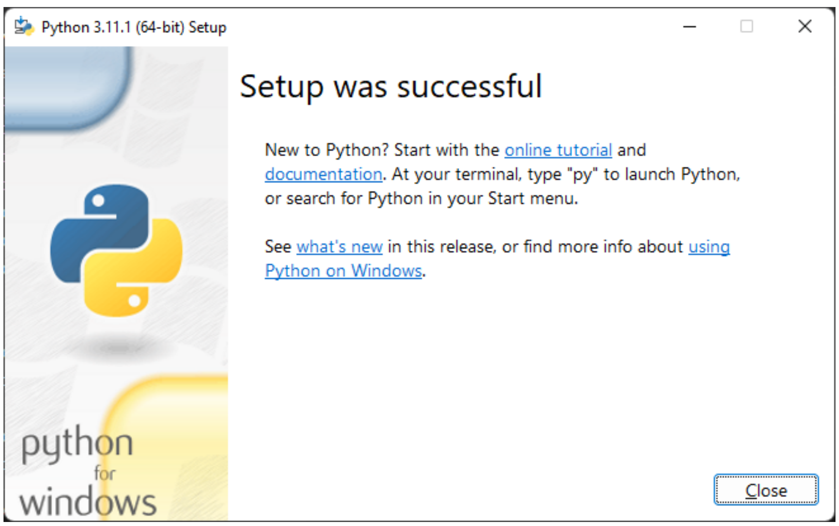
https://www.python.org/downloads/

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'



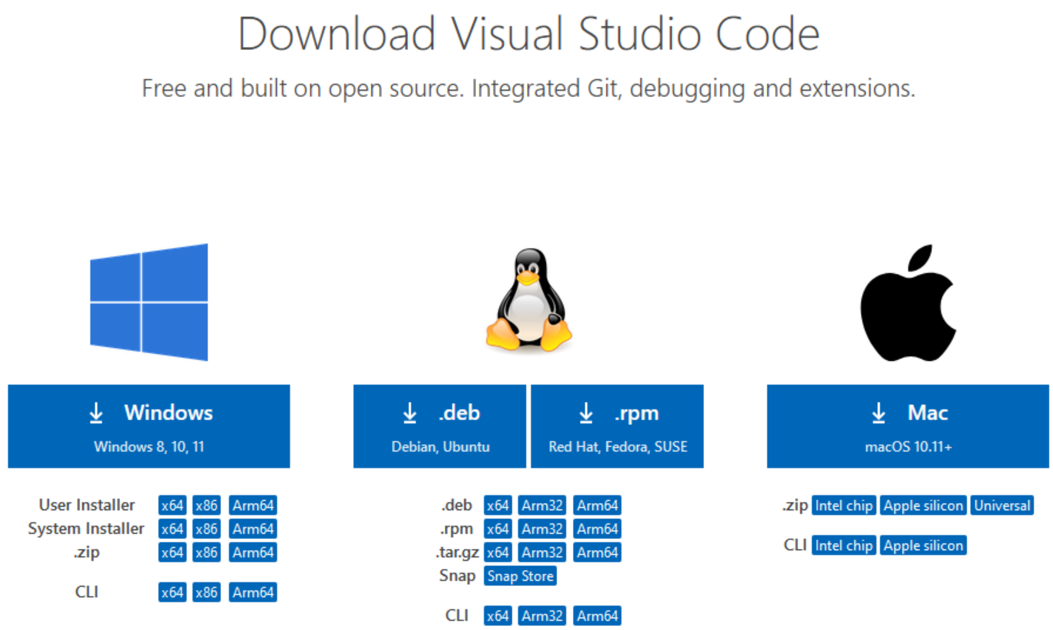


Once Python is installed, please **restart** your computer.

Installing VSCode

Navigate to the following URL:

https://code.visualstudio.com/download



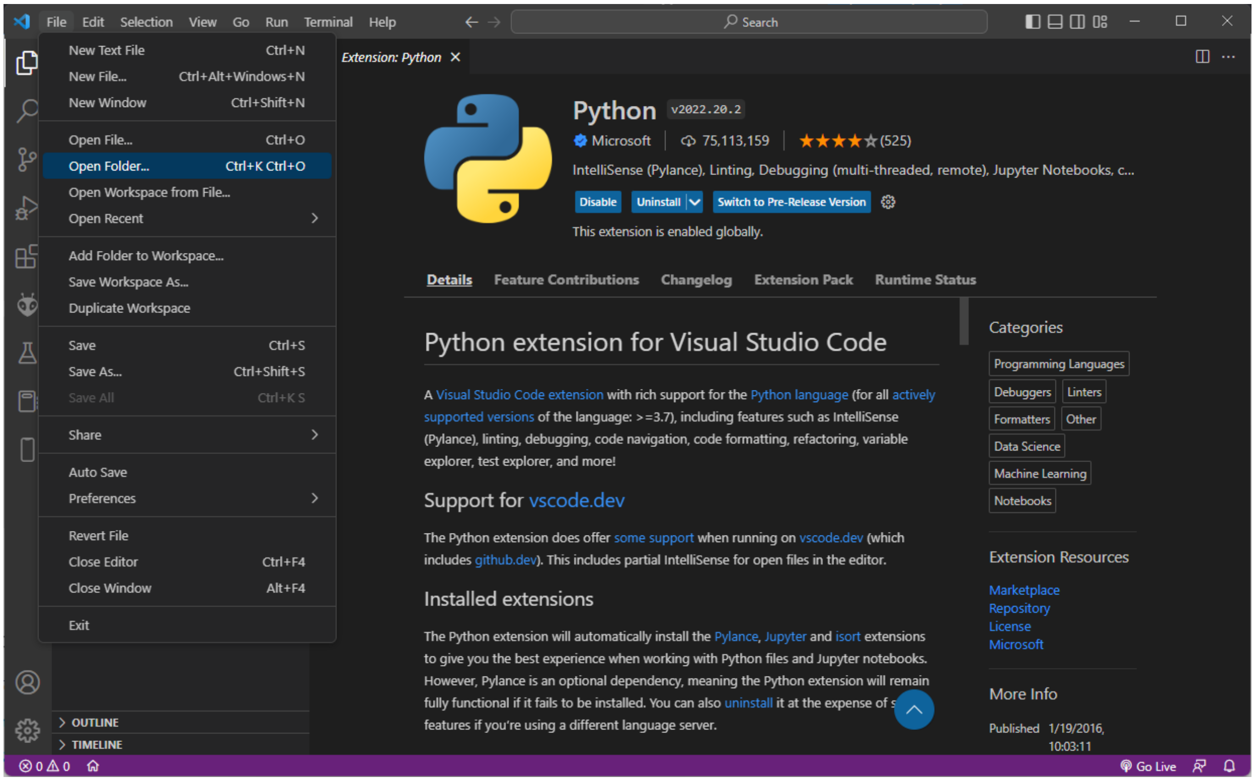
Run the installer using the **default install options**.

Once installed launch **VSCode**.

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

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

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

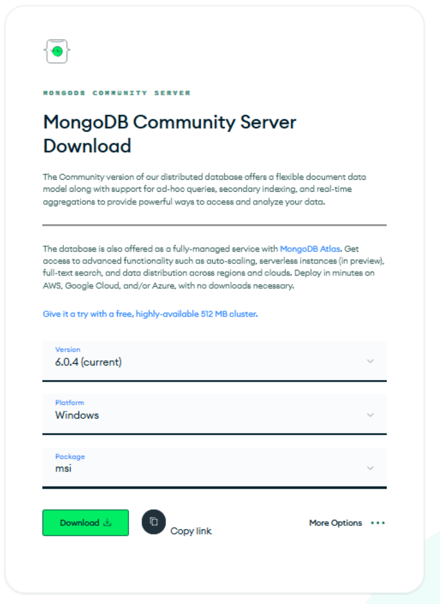


Open your local repository.

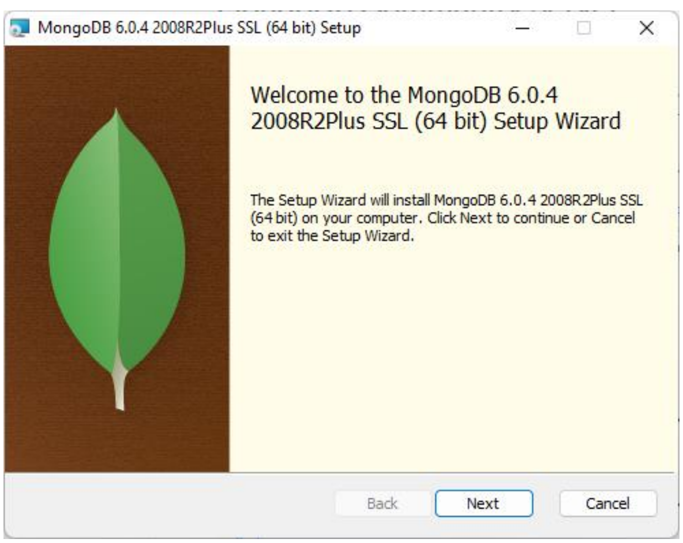
Installing and Connecting to MongoDB

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

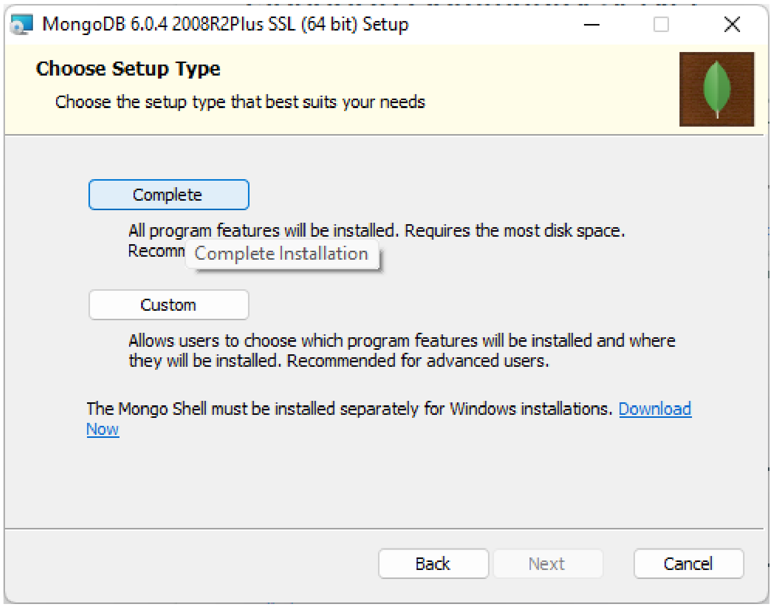
https://www.mongodb.com/try/download/community-kubernetes-operator



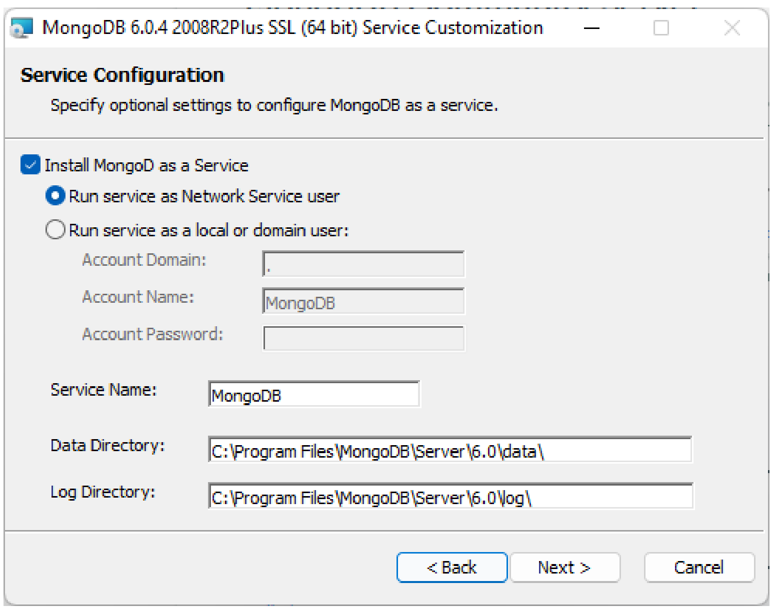
Run the installer using default settings. It will say **This may contain malicious software**. Ignore this warning, this software is very reputable, the packaging format is just out-of-date.



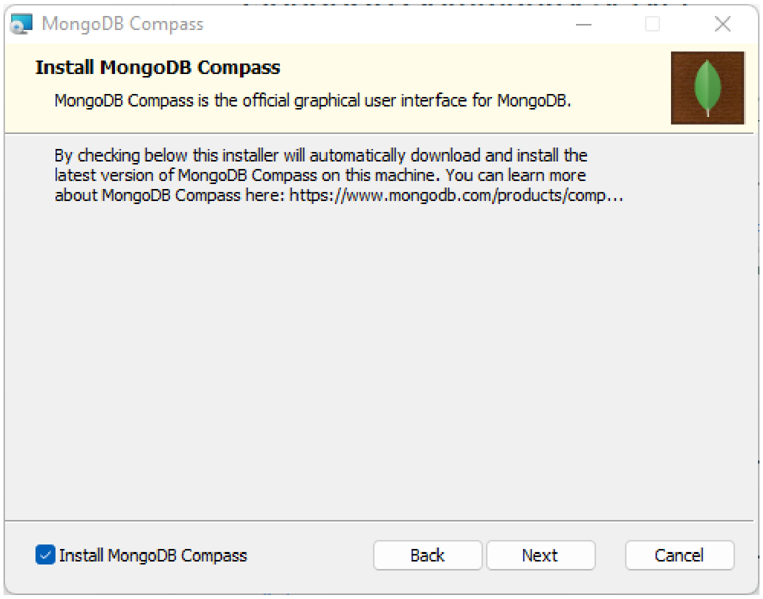
Perform a **Complete** installation.

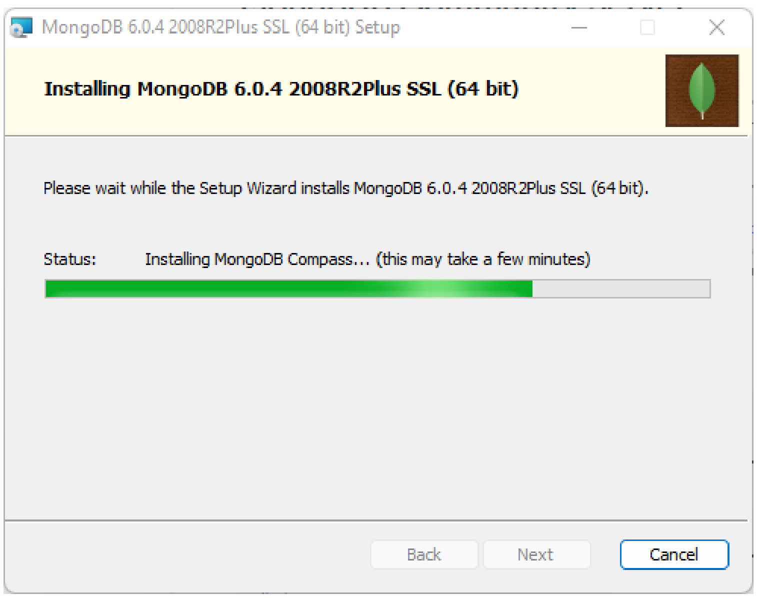


Leave **Service Configuration** as default.

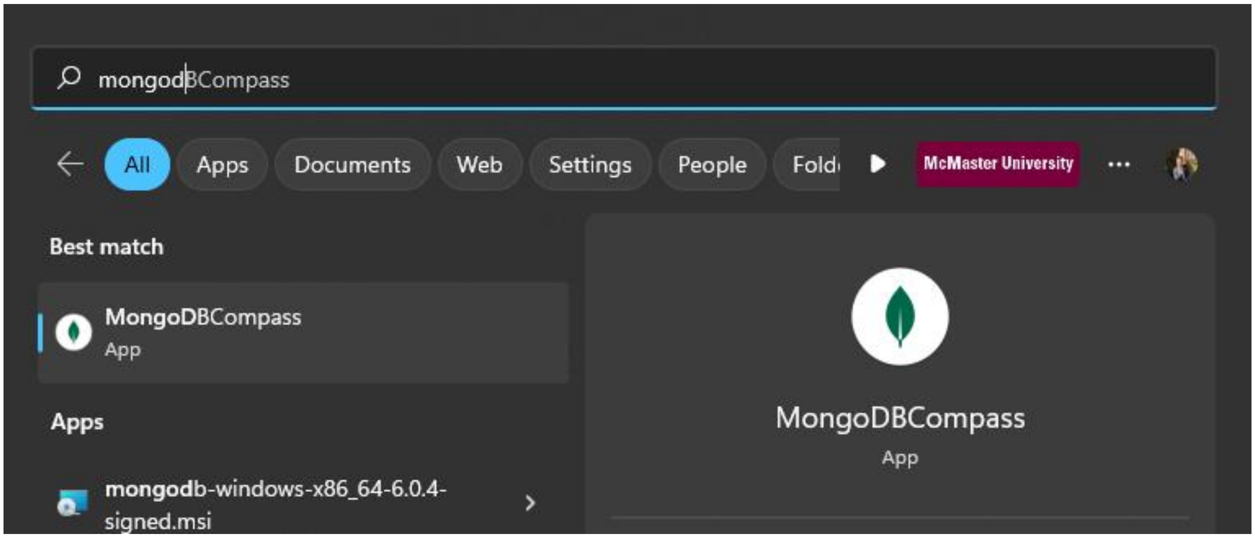


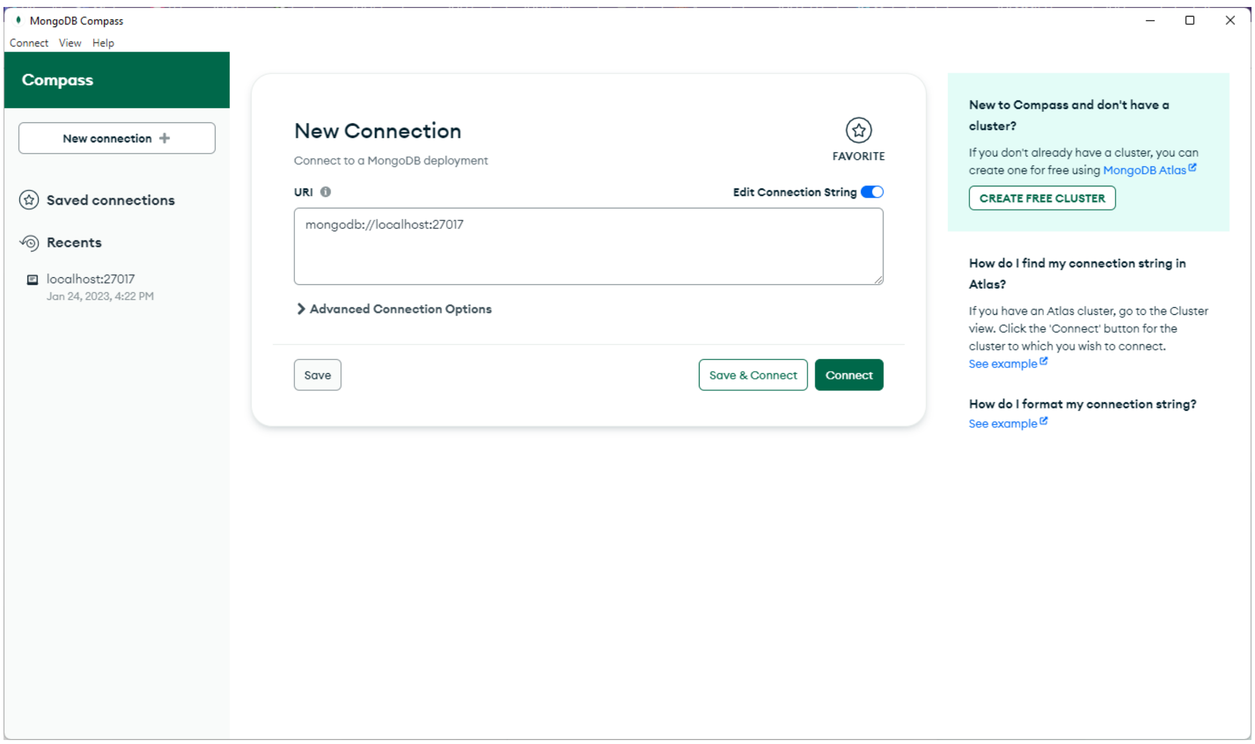
Ensure that **Install MongoDB Compass** is **checked**.





Launch **MongoDB Compass**.





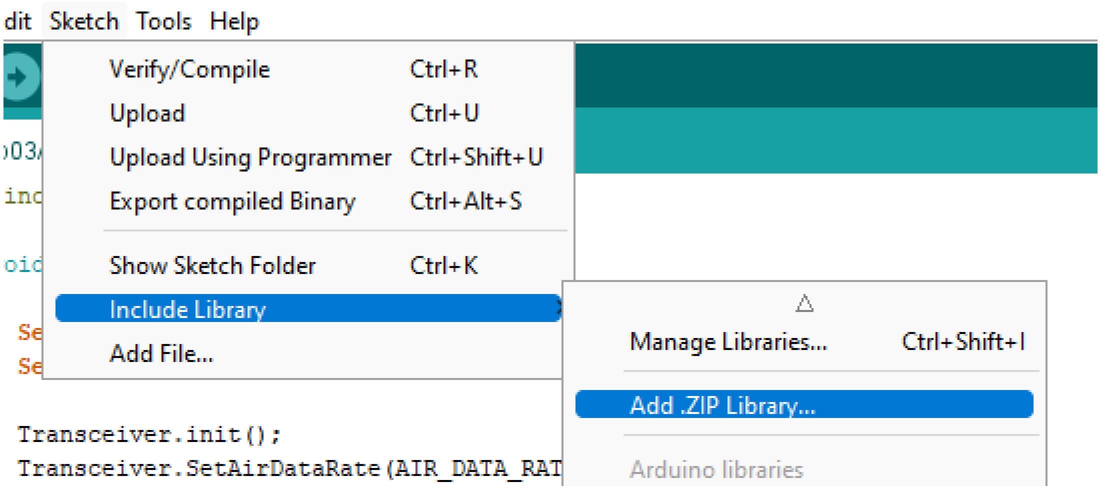
Press **Connect**.

ESP32 Libraries

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

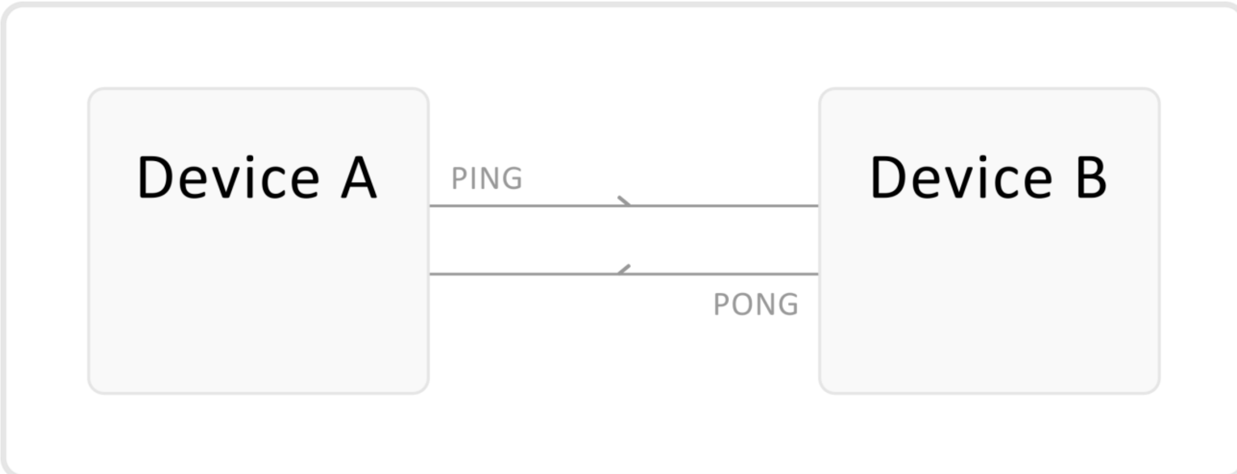
https://github.com/xreef/EByte_LoRa_E22_Series_Library

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

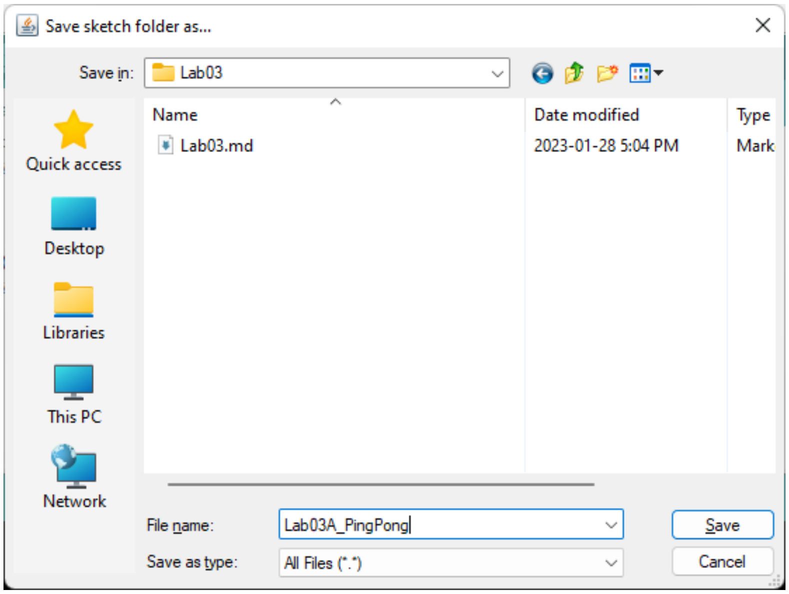


Ping Pong

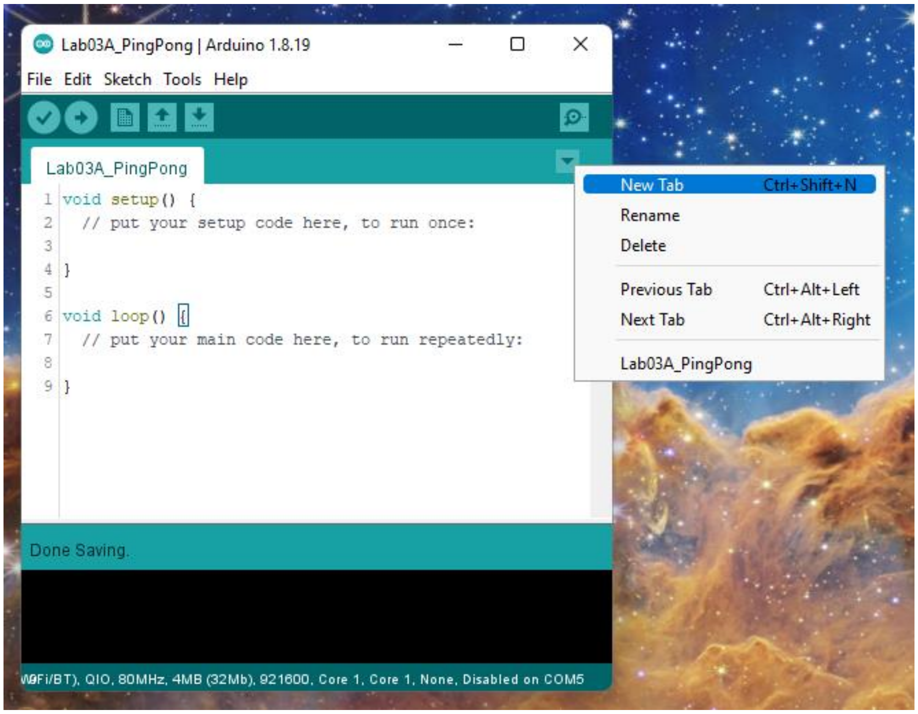
The goal of this experiment is to observer point-to-point communication between two LoRa transceivers.

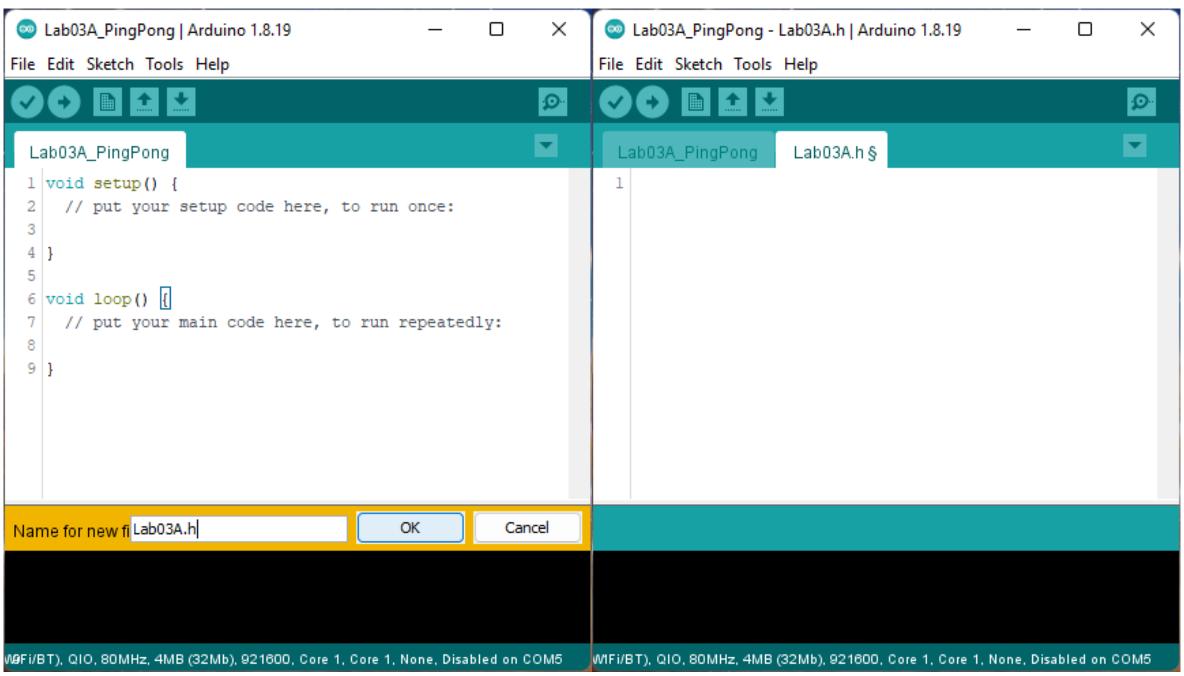


Open the Arduino IDE and create a new project. Name it **Lab03A_PingPong**.



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





Include the following libraries and pin definitions.

Lab03A_PingPong | Lab03A.h

```
//Libraries
#include "EBYTE.h"
#include <HardwareSerial.h>

//Pin defines
#define PIN_RX 16   // Serial2 RX -> EBYTE TX
#define PIN_TX 17   // Serial2 TX pin -> EBYTE RX
#define PIN_M0 27
#define PIN_M1 26
#define PIN_AX 25

#define TIME_PER_SEND_MS 3000
```

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

Lab03A_PingPong | Lab03A.h §

```
//Data rate enumerations
enum eAirDataRate {
  BAUD300 = 0b000,
  BAUD1200 = 0b001,
  BAUD2400 = 0b010,
  BAUD4800 = 0b011,
  BAUD9600 = 0b100,
  BAUD19200 = 0b101
};

//Transceiver frequency
const eAirDataRate AIR_DATA_RATE = BAUD300;

//Creating a transceiver object
EBYTE Transceiver(&Serial2, PIN_M0, PIN_M1, PIN_AX);


```

Set the device details to be unique for your group.

Lab03A_PingPong | Lab03A.h §

```
//Creating a transceiver object
EBYTE Transceiver(&Serial2, PIN_M0, PIN_M1, PIN_AX);

//Device details
String GROUP_NAME = "GroupA";
String DEVICE_ID = "Device6";

```

Navigate back to your implementation file.

Include your newly created header file.

Lab03A_PingPong | Lab03A.h

```
#include "Lab03A.h"

```

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

```
#include "Lab03A.h"

void setup() {

  Serial.begin(9600);
  Serial2.begin(9600);

  Transceiver.init();
  Transceiver.SetAirDataRate(AIR_DATA_RATE);
  //Transceiver.SetMode(MODE_NORMAL);
  //Transceiver.SetTransmitPower(OPT_TP20);
  //Transceiver.SaveParameters(PERMANENT);
  Transceiver.PrintParameters();
}

```

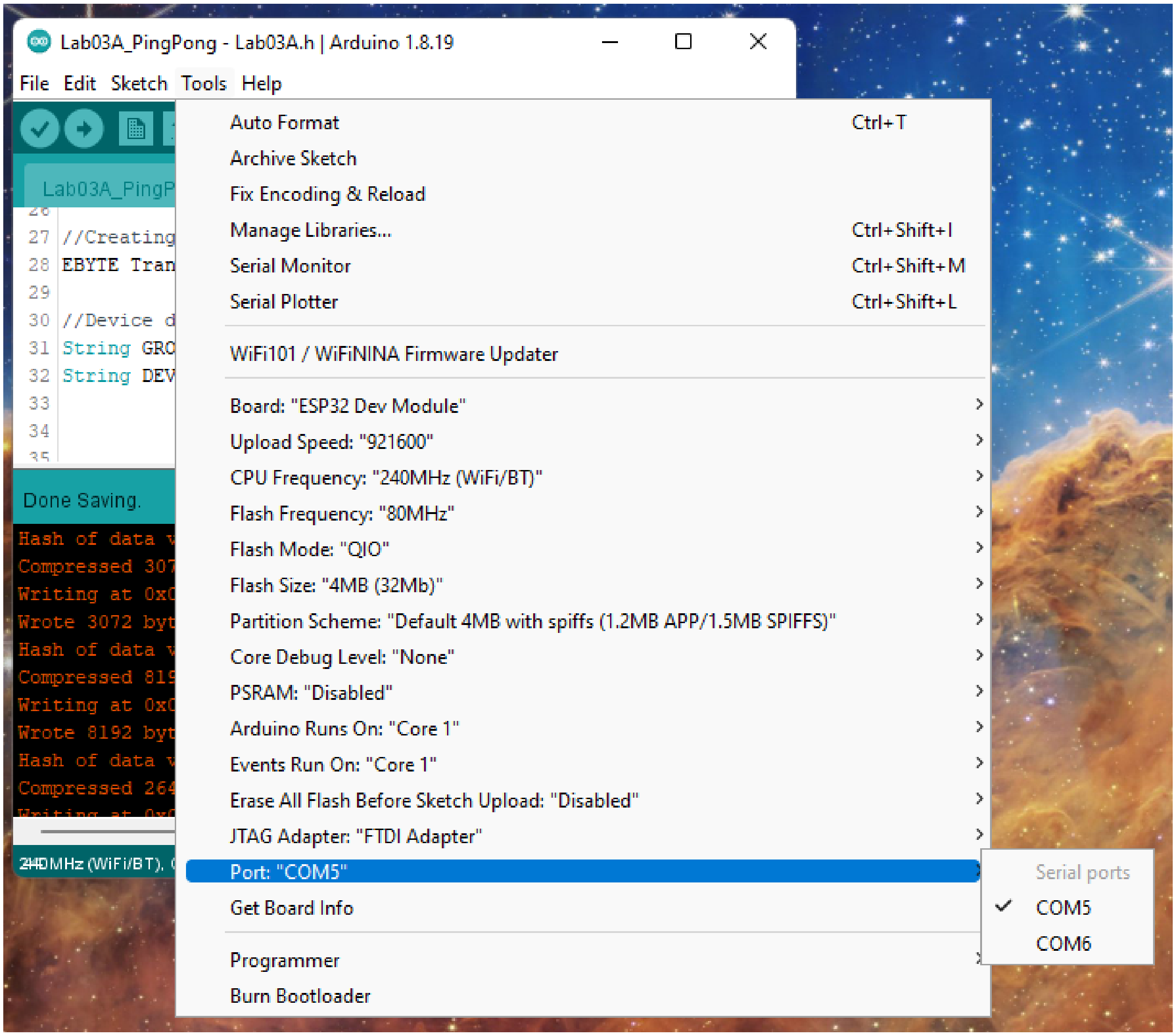
In the **loop()** function, there are two things that happen. First, the serial buffer is checked. If there is data being sent from the transceiver to the ESP32, it will read that string and print that to the serial monitor with **PING:** in front of the data. If a specific amount of time elapses, it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

Lab03A_PingPong Lab03A.h

```
unsigned long startTime = millis();

void loop() {

  if (Serial2.available() > 1) {
    Serial.println("Receving Data...");
    String incomingData = Serial2.readString();
    Serial.println("PING: " + incomingData);

  }

  if(millis() - startTime > TIME_PER_SEND_MS){
    Serial2.println(GROUP_NAME + " - " + DEVICE_ID);
    Serial.println("PONG SENT");
    startTime = millis();
  }

}

```

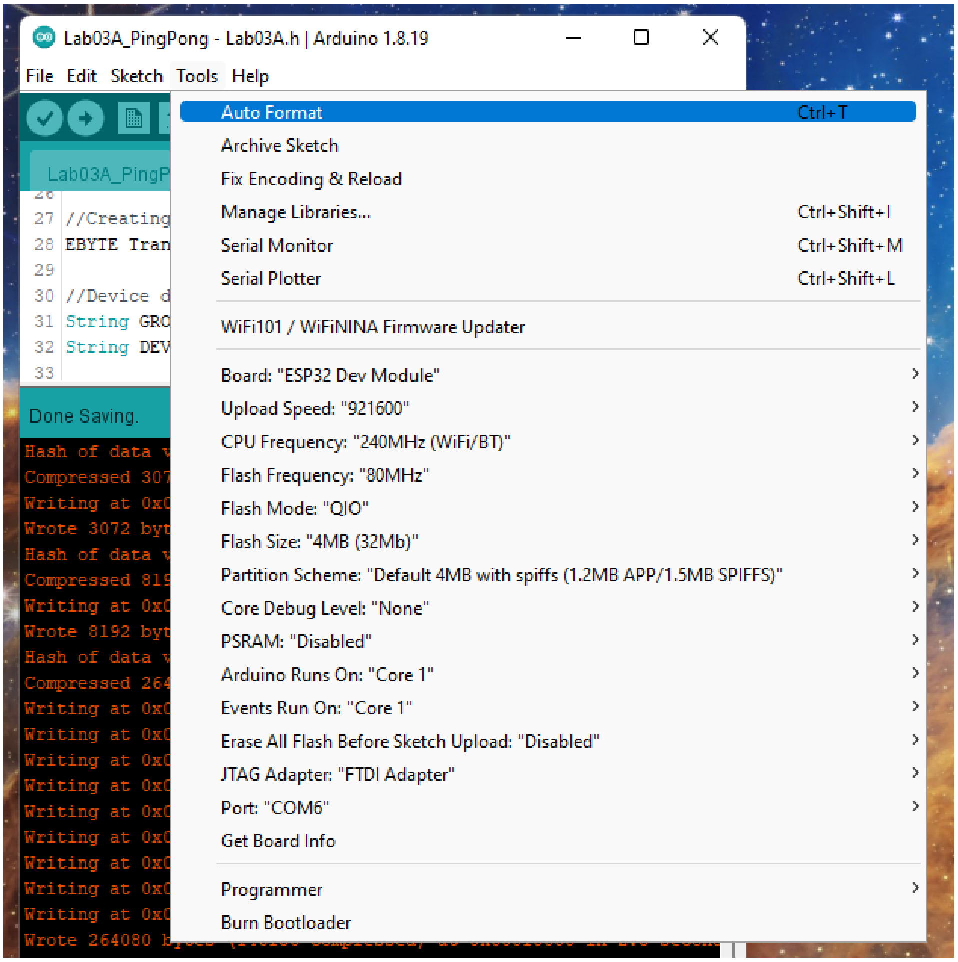
Make the device ID unique to your 1st microcontroller, select the COM port, and upload the sketch.

```
//Device details
String GROUP_NAME = "GroupA";
String DEVICE_ID = "Device5";
```

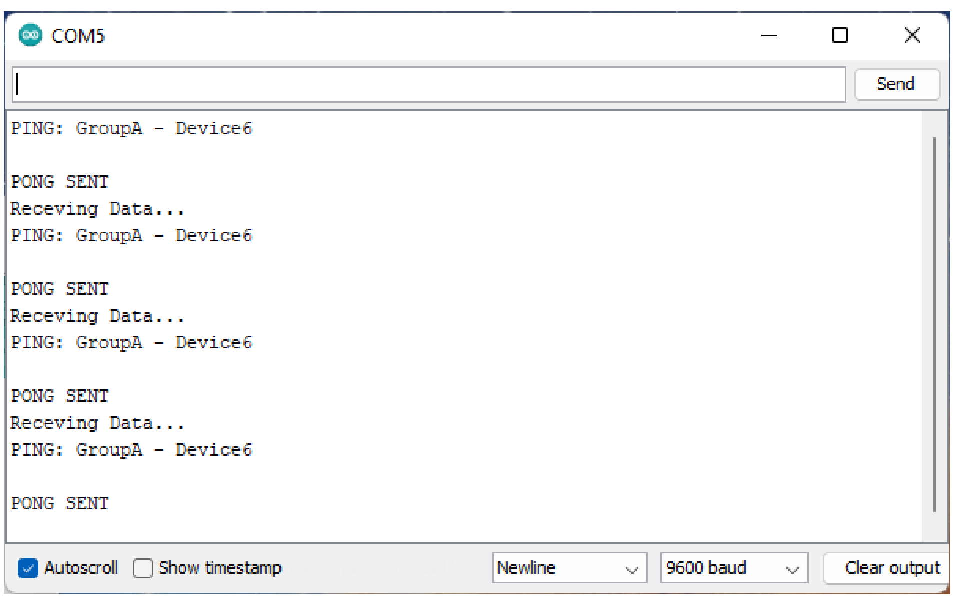

Lab03A_PingPong - Lab03A.h | Arduino 1.8.19
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "ESP32 Dev Module"
Upload Speed: "921600"
CPU Frequency: "240MHz (WiFi/BT)"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Flash Size: "4MB (32Mb)"
Partition Scheme: "Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS)"
Core Debug Level: "None"
PSRAM: "Disabled"
Arduino Runs On: "Core 1"
Events Run On: "Core 1"
Erase All Flash Before Sketch Upload: "Disabled"
JTAG Adapter: "FTDI Adapter"
Port: "COM5"
Get Board Info
Programmer
Burn Bootloader
Serial ports
COM5
COM6
Done Saving.

```
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 (e
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 secon

Hash of data verified.

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

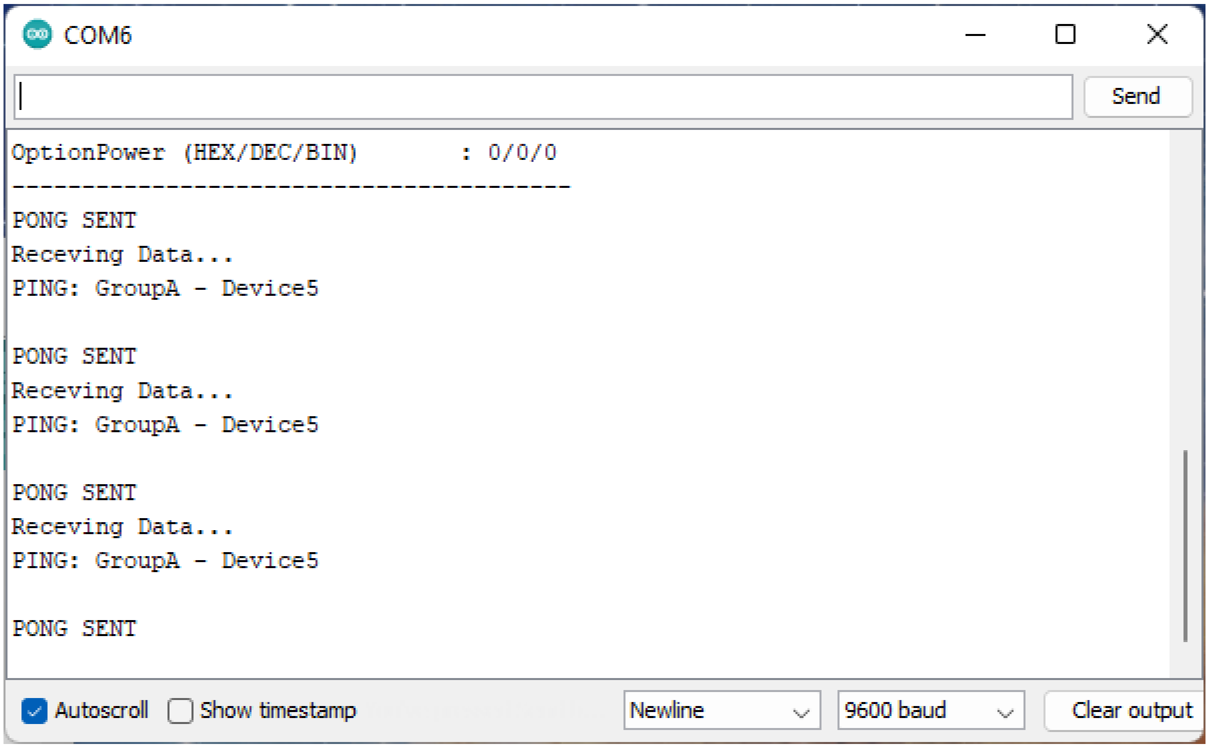
Modify the device ID to be unique to your 2nd microcontroller and upload the sketch to your 2nd ESP32. Also, change the COM port.



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



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

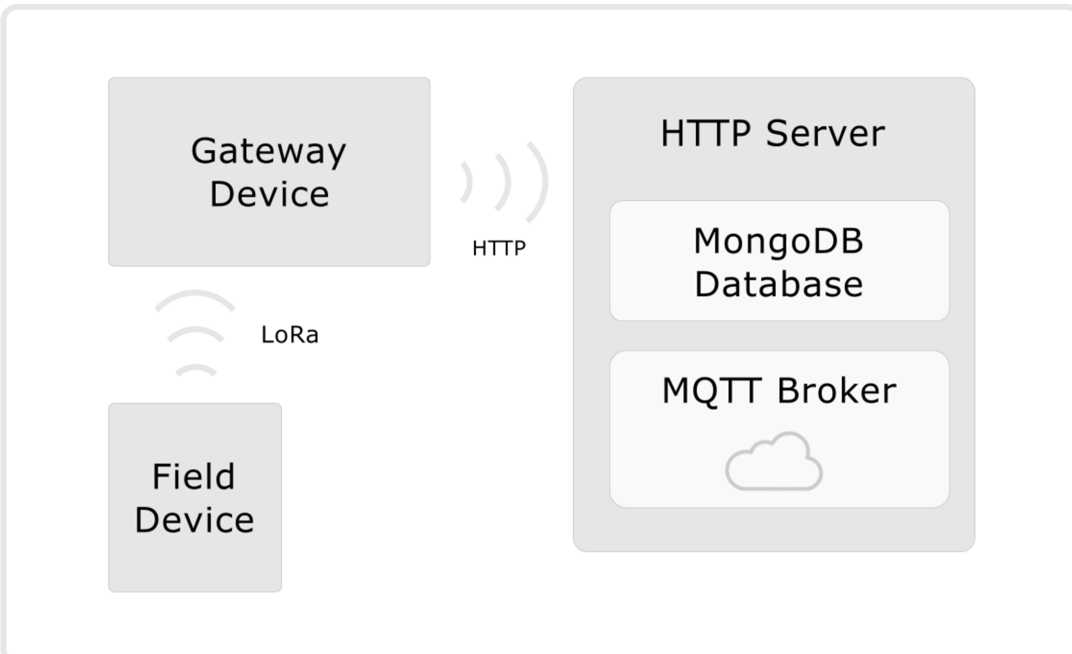


Device Network

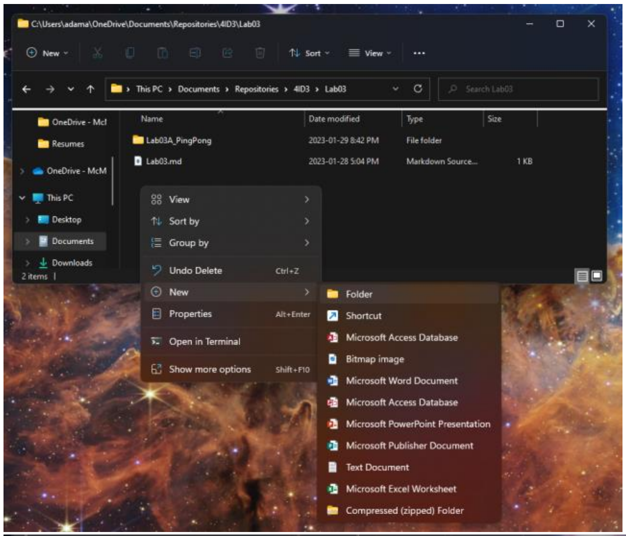
In this lab section, we will be developing a more sophisticated IoT network that involves 5 key components:

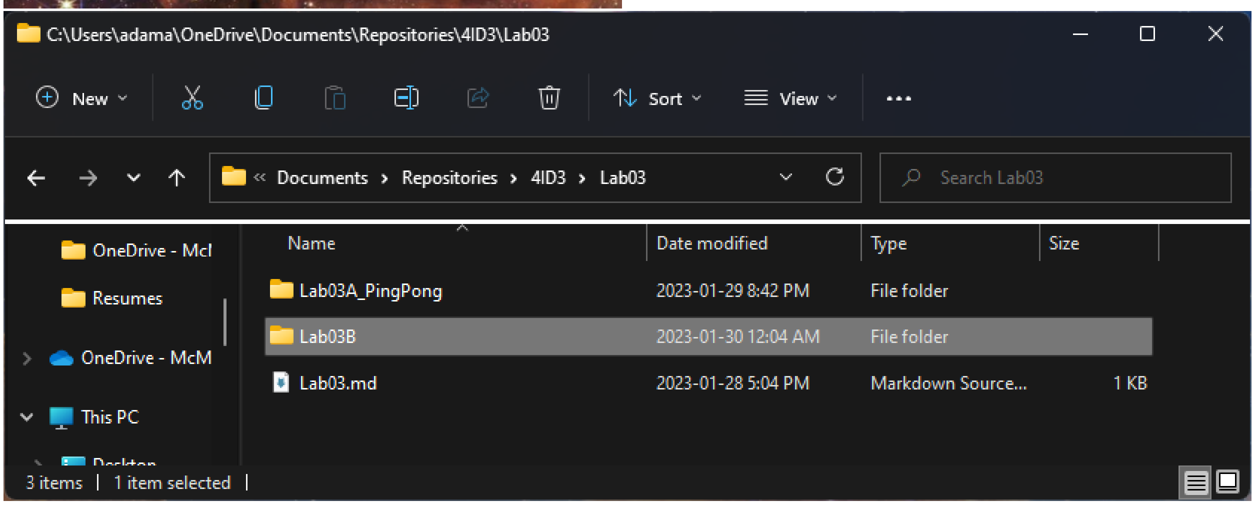
a) Field devices *(Sensors -> LoRa)*
b) Gateway devices *(LoRa -> WiFi)*
c) Server Back-end *(HTTP request routing)*
d) Dashboard Front-end *(Remote client access)*
e) Database connection *(Data collection and processing)*

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**.

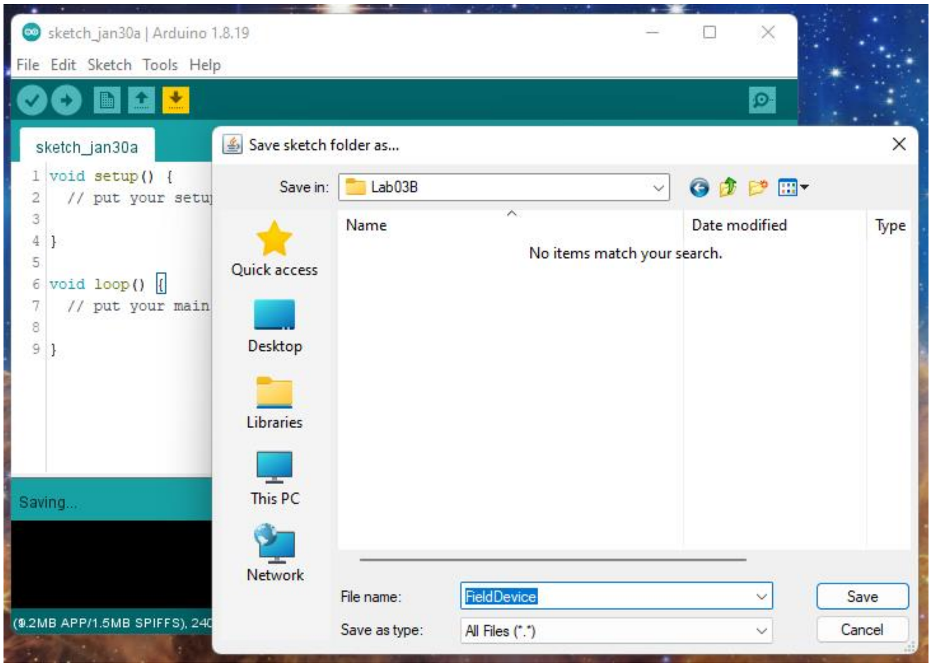

View
Sort by
Group by
Undo Delete
New
Properties
Open in Terminal
Show more options
Folder
Shortcut
Microsoft Access Database
Bitmap image
Microsoft Word Document
Microsoft Access Database
Microsoft PowerPoint Presentation
Microsoft Publisher Document
Text Document
Microsoft Excel Worksheet
Compressed (zipped) Folder


C:\Users\adama\OneDrive\Documents\Repositories\4ID3\Lab03
New
Sort
View
Documents › Repositories › 4ID3 › Lab03
Search Lab03
Name
Date modified
Type
Size
Lab03A_PingPong
2023-01-29 8:42 PM
File folder
Lab03B
2023-01-30 12:04 AM
File folder
Lab03.md
2023-01-28 5:04 PM
Markdown Source...
1 KB
OneDrive - McI
Resumes
OneDrive - McM
This PC
3 items | 1 item selected |

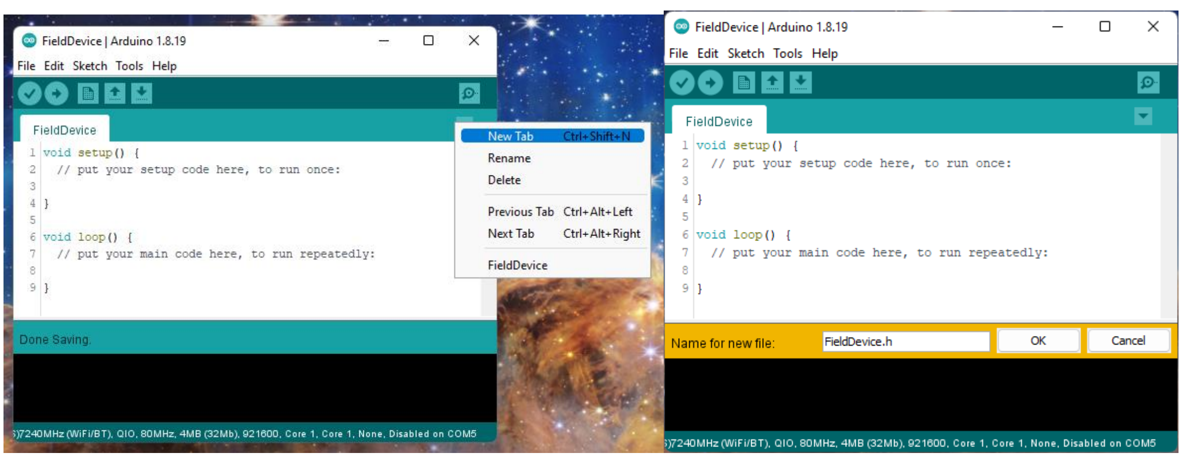
Field Device

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:

FieldDevice | FieldDevice.h

```
//Libraries
#include <Arduino.h>
#include <Wire.h>
#include <AsyncAPDS9306.h>
#include "EBYTE.h"
#include <HardwareSerial.h>

```

Next, define your pin data.

FieldDevice | FieldDevice.h

```

//Pin definitions
#define PIN_RX 16
#define PIN_TX 17
#define PIN_M0 27
#define PIN_M1 26
#define PIN_AX 25

//Sample frequency
#define DELAY_BETWEEN_SAMPLES_MS 5000

```

Next, configure your LoRa object.

FieldDevice | FieldDevice.h

```

//Datarate enumeration
enum eAirDataRate {
  BAUD300 = 0b000,
  BAUD1200 = 0b001,
  BAUD2400 = 0b010,
  BAUD4800 = 0b011,
  BAUD9600 = 0b100,
  BAUD19200 = 0b101
};

const eAirDataRate AIR_DATA_RATE = BAUD300;
EBYTE Transceiver(&Serial2, PIN_M0, PIN_M1, PIN_AX);

```

Set up your device information.

```

//Device information
String groupName = "GroupA";
String deviceName = "DeviceA";

```

Set up your sensors.

```

//Sensor IIC addresses
#define ADDR (byte)(0x40)
#define TMP_CMD (byte)(0xF3)

//Instantiating sensor object and configuration
AsyncAPDS9306 lightSensor;
const APDS9306_ALS_GAIN_t aGain = APDS9306_ALS_GAIN_1;
const APDS9306_ALS_MEAS_RES_t aTime = APDS9306_ALS_MEAS_RES_16BIT_25MS;

```

Next, include this header file in your implementation file.

FieldDevice | FieldDevice.h

```
#include "FieldDevices.h"

void setup() {
  // put your setup code here, to run once:

}

```

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

FieldDevice | FieldDevice.h

```
#include "FieldDevices.h"

void setup() {
  Serial.begin(9600);
  Serial.print("\n\n-------------------------\n"
    + groupName + " : " + deviceName + "\n-------------------------\n\n");

  Wire.begin();
  Wire.beginTransmission(ADDR);
  Wire.endTransmission();
  delay(300);

  lightSensor.begin(aGain, aTime);

  Serial.println("Ready for LoRa connection!");

  Serial2.begin(9600);

  Transceiver.init();
  //Transceiver.SetAirDataRate(AIR_DATA_RATE);
  Transceiver.PrintParameters();

}

```

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

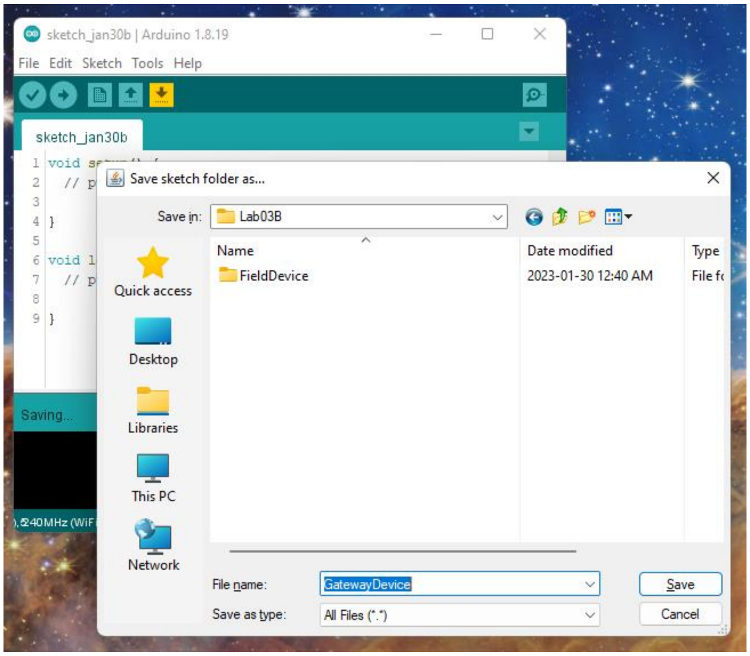
FieldDevice | FieldDevice.h

```
void loop() {

  if(millis() - startTime > DELAY_BETWEEN_SAMPLES_MS){

    //Temp sensor
    Wire.beginTransmission(ADDR);
    Wire.write(TMP_CMD);
    Wire.endTransmission();
    delay(100);

    Wire.requestFrom(ADDR, 2);

    char data[2];
    if(Wire.available() == 2){
      data[0] = Wire.read();
      data[1] = Wire.read();
    }

    float temp = ((data[0] * 256.0) + data[1]);
    float tempC = ((175.72 * temp) / 65536.0) - 46.85;
    Serial.println("Temperature: " + String(tempC) + " degC");

    //Sample light sensor
    AsyncAPDS9306Data lightData = lightSensor.syncLuminosityMeasurement();

    //Calculate luminosity
    float lux = lightData.calculateLux();
    Serial.println("Luminosity: " + String(lux) + " Lux");


    //Format data as a JSON string
    String sendData = "{ \"" + groupName + "\": { \"" + deviceName + "\": { \"Temp\": \""
        + String(tempC) + "\", \"Luminosity\": \"" + String(lux) + "\" } } }" + '\n';

    Serial.println("Prepared LoRa message: " + sendData);

    Serial2.println(sendData);
    Serial.println("LoRa sent!");

  }

}
```

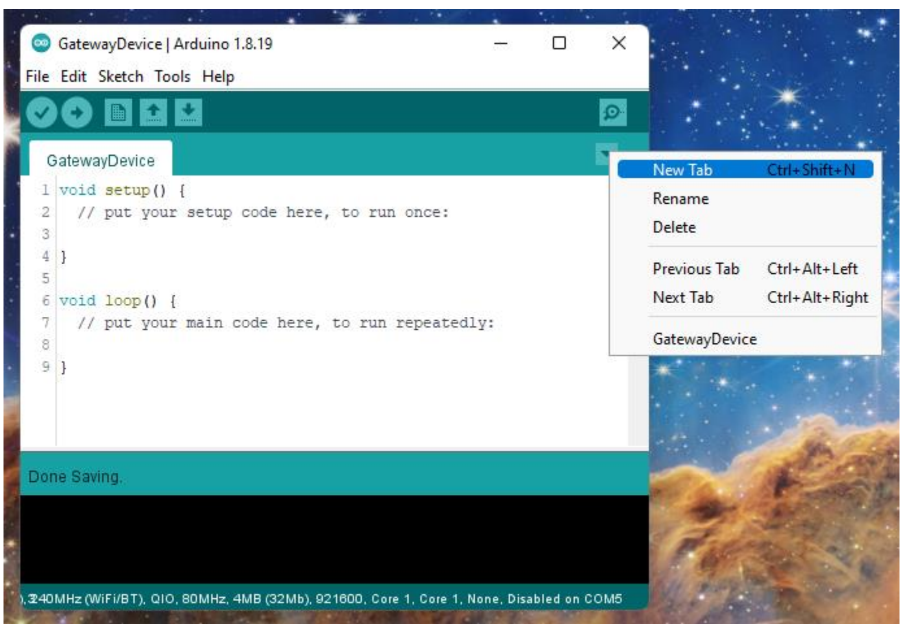
Gateway Device

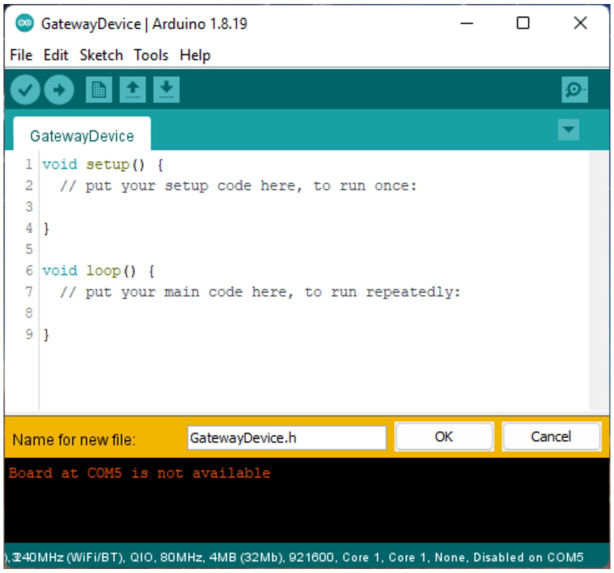
The role of the gateway device is to receive data from multiple field devices and transmit them to a server that can further process the data.

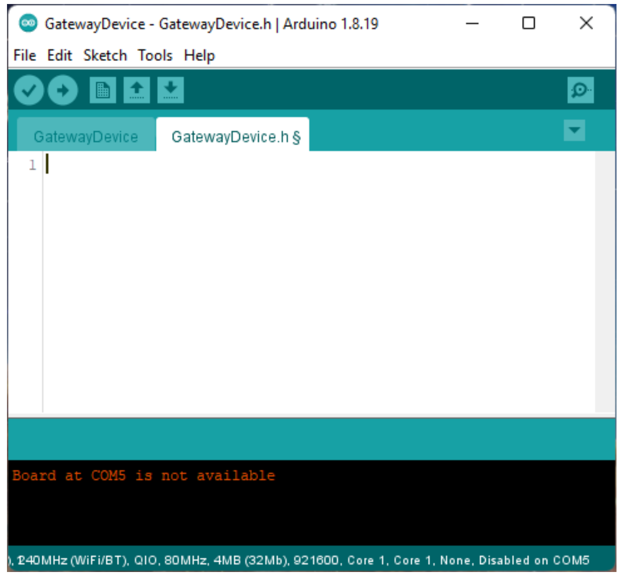
Start off by creating a new sketch and naming it **GatewayDevice**. Save it in the **4ID3/Lab03** folder.



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







Set up the header file like this:

GatewayDevice | GatewayDevice.h

```
//Libraries
#include "EBYTE.h"
#include <WiFi.h>
#include <HTTPClient.h>
#include <HardwareSerial.h>

//Pin definitions
#define PIN_RX 16
#define PIN_TX 17
#define PIN_M0 27
#define PIN_M1 26
#define PIN_AX 25

//Transceiver setup
enum eAirDataRate {
  BAUD300 = 0b000,
  BAUD1200 = 0b001,
  BAUD2400 = 0b010,
  BAUD4800 = 0b011,
  BAUD9600 = 0b100,
  BAUD19200 = 0b101
};

const eAirDataRate AIR_DATA_RATE = BAUD300;
EBYTE Transceiver(&Serial2, PIN_M0, PIN_M1, PIN_AX);

//WiFi login credentials
const char* ssid = "GroupA";
const char* password = "12345678";

//HTTP server URL
const char* serverName = "http://192.168.137.38:3000/mqtt";
//Database Server at http://192.168.137.38:3000/database

```

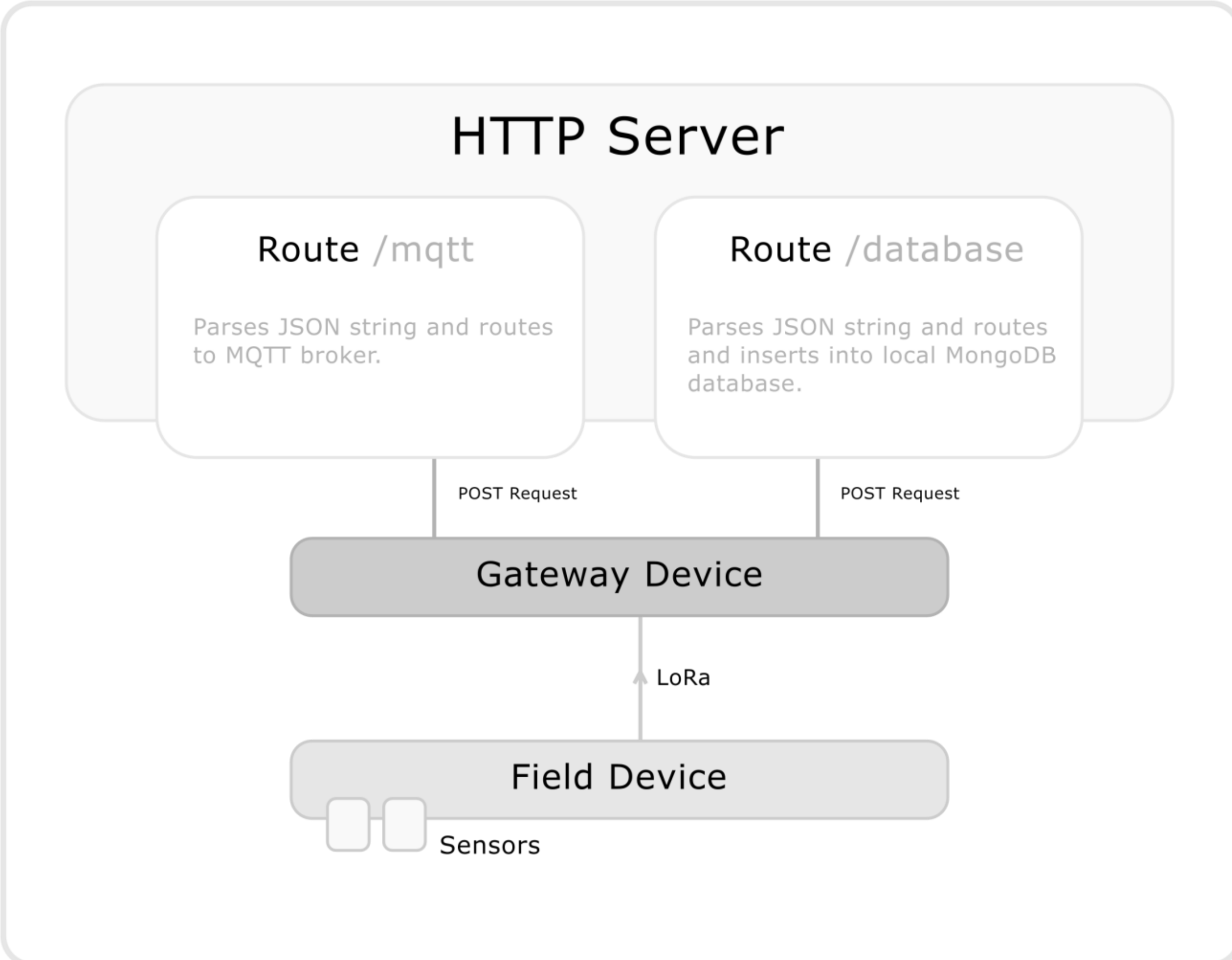
Next, in the implementation file, we want to poll if the LoRa serial receive buffer has data in it. If so, read that data and make a POST request to the server.

GatewayDevice | GatewayDevice.h

```
#include "GatewayDevice.h"

void setup() {
  Serial.begin(9600);
  Serial2.begin(9600);
  Transceiver.init();
  Transceiver.SetAirDataRate(AIR_DATA_RATE);
  Transceiver.PrintParameters();

  WiFi.begin(ssid, password);
  Serial.println("Connecting");
  while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to WiFi network with IP Address: ");
  Serial.println(WiFi.localIP());

}

void loop() {

 if (Serial2.available() > 1) {
    Serial.println("Receving Data...");
    String incomingData = Serial2.readString();
    Serial.println("PING: " + incomingData);

    if(WiFi.status()== WL_CONNECTED){
      WiFiClient client;
      HTTPClient http;

      http.begin(client, serverName);

      http.addHeader("Content-Type", "text/plain");

      int httpResponseCode = http.POST(incomingData);

      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);

      http.end();

    }

  }
}
```

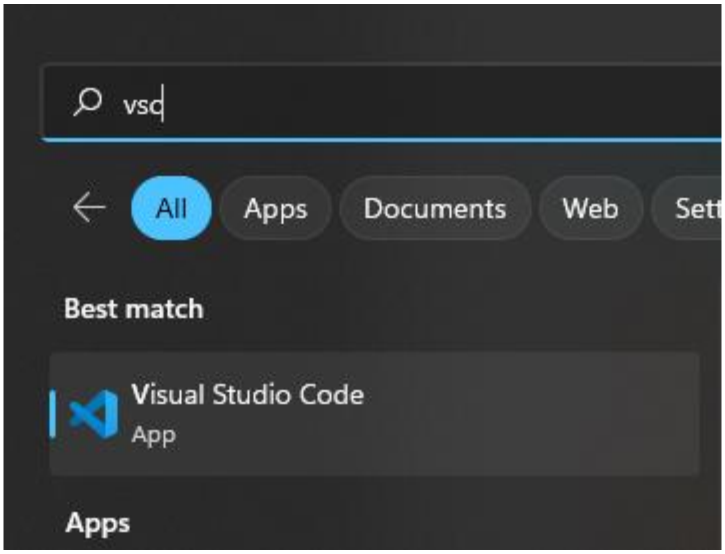
HTTP Server

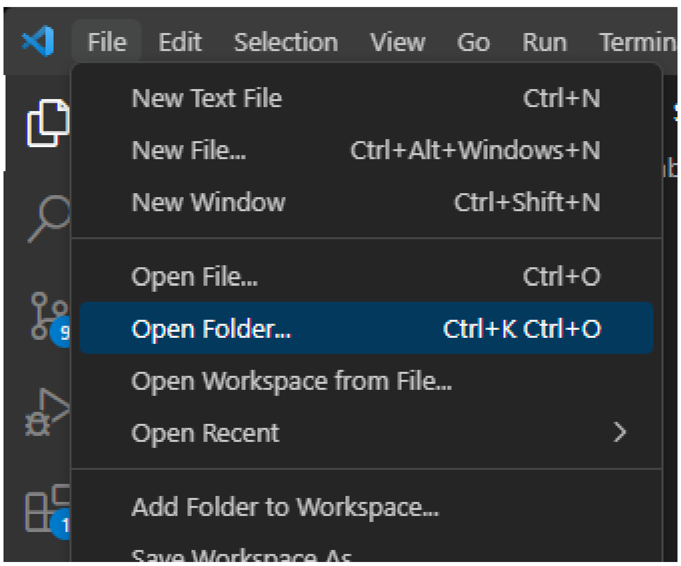
The role of the HTTP server is to route the data that is being sent by the gateway to a set a pre-defined callback functions called **routes**.

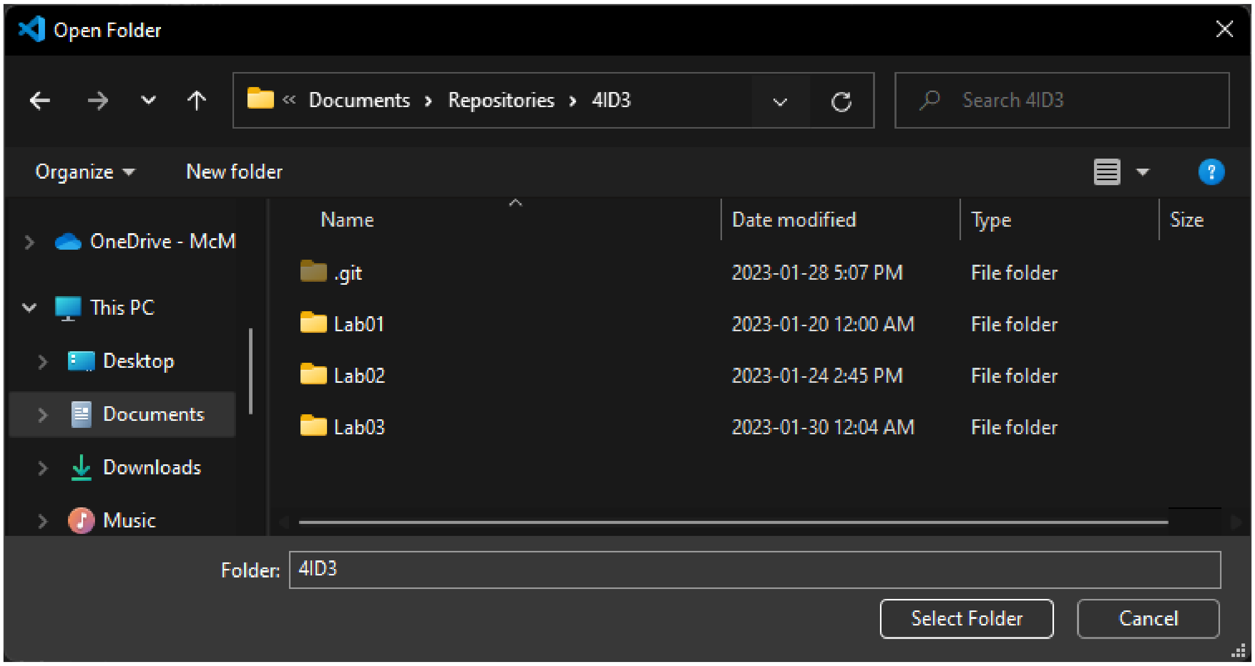


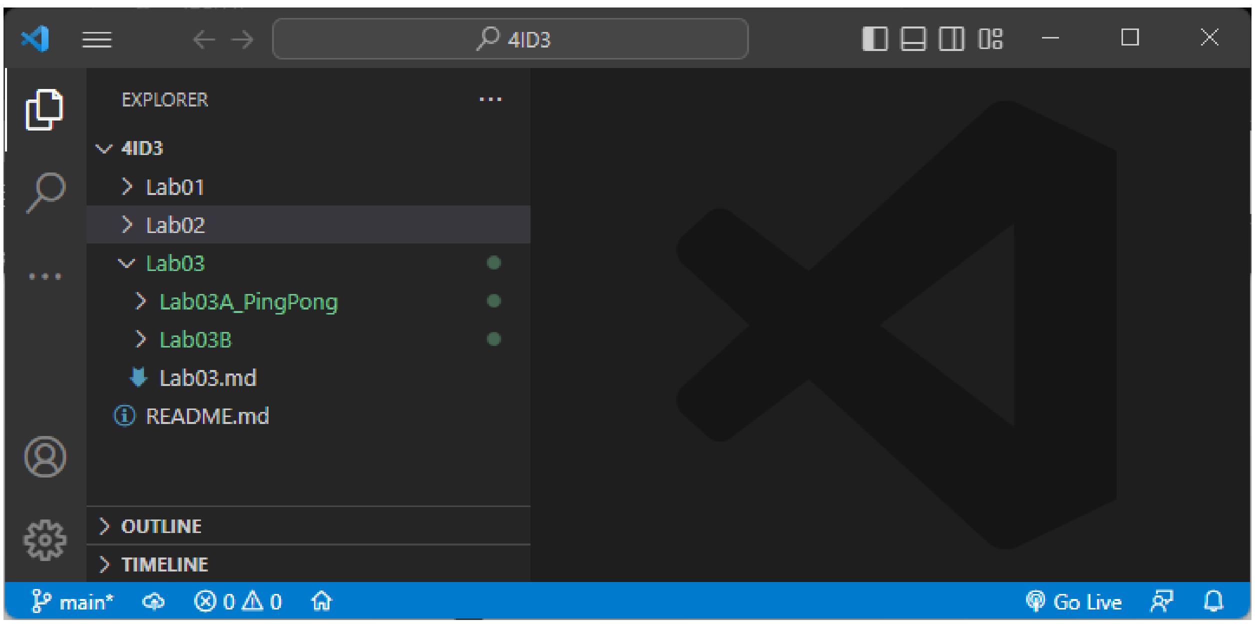
Repo Setup

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

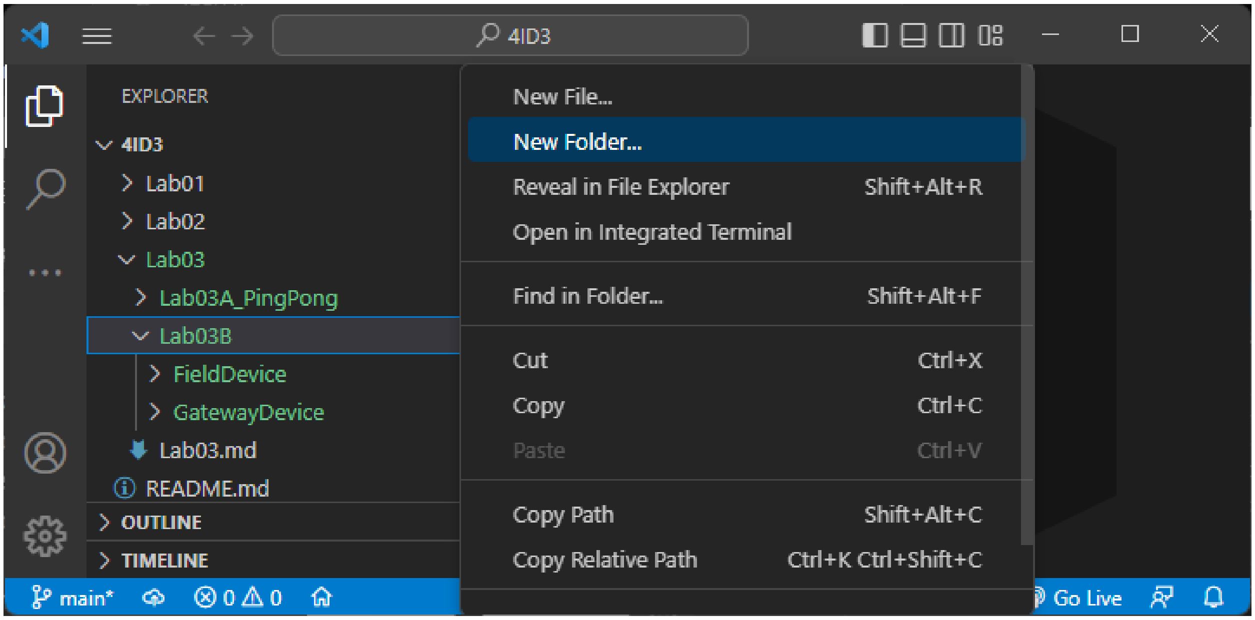

vsc
All
Apps
Documents
Web
Best match
Visual Studio Code
App
Apps

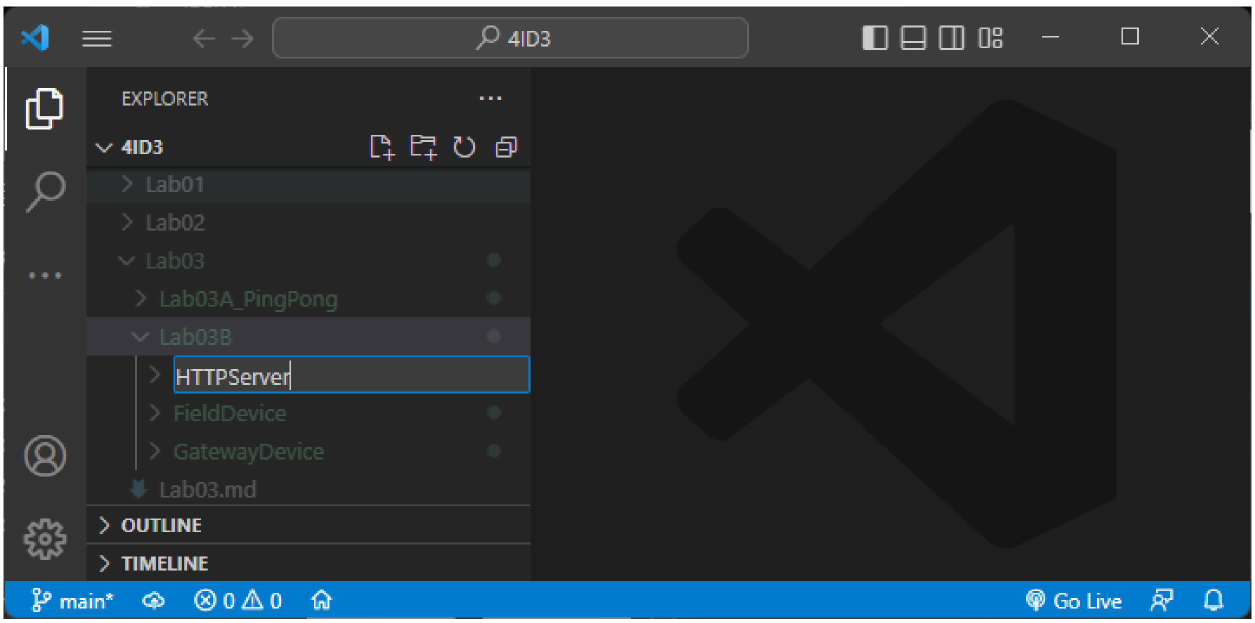

File
Edit
Selection
View
Go
Run
New Text File Ctrl+N
New File... Ctrl+Alt+Windows+N
New Window Ctrl+Shift+N
Open File... Ctrl+O
Open Folder... Ctrl+K Ctrl+O
Open Workspace from File...
Open Recent
Add Folder to Workspace...



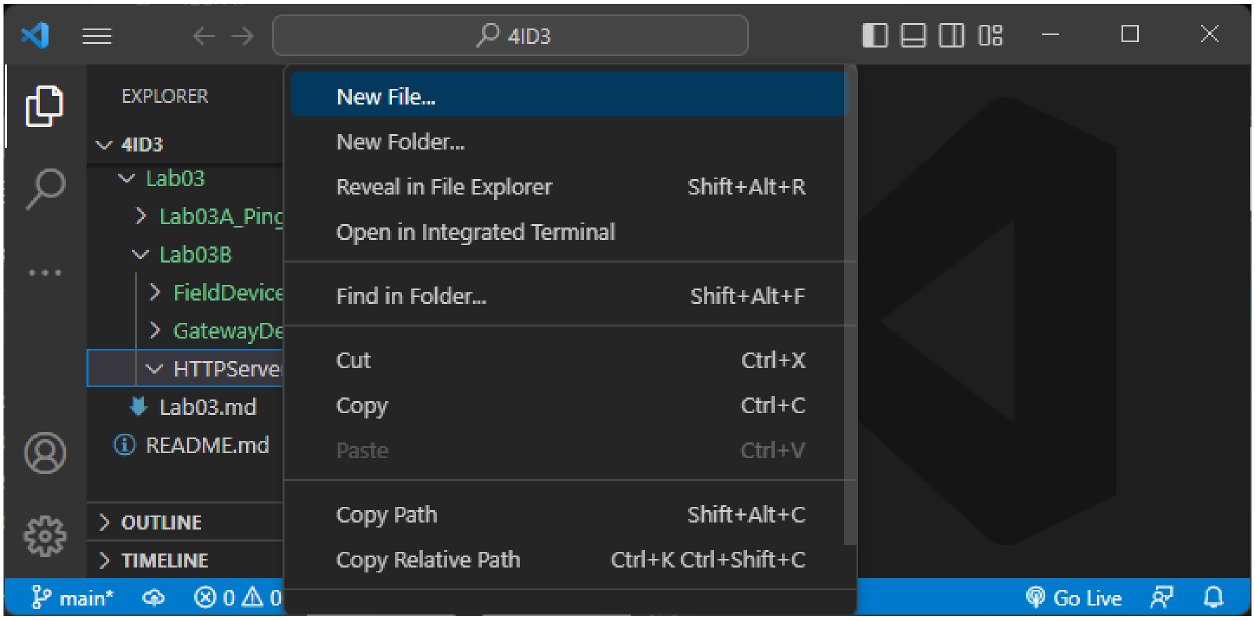


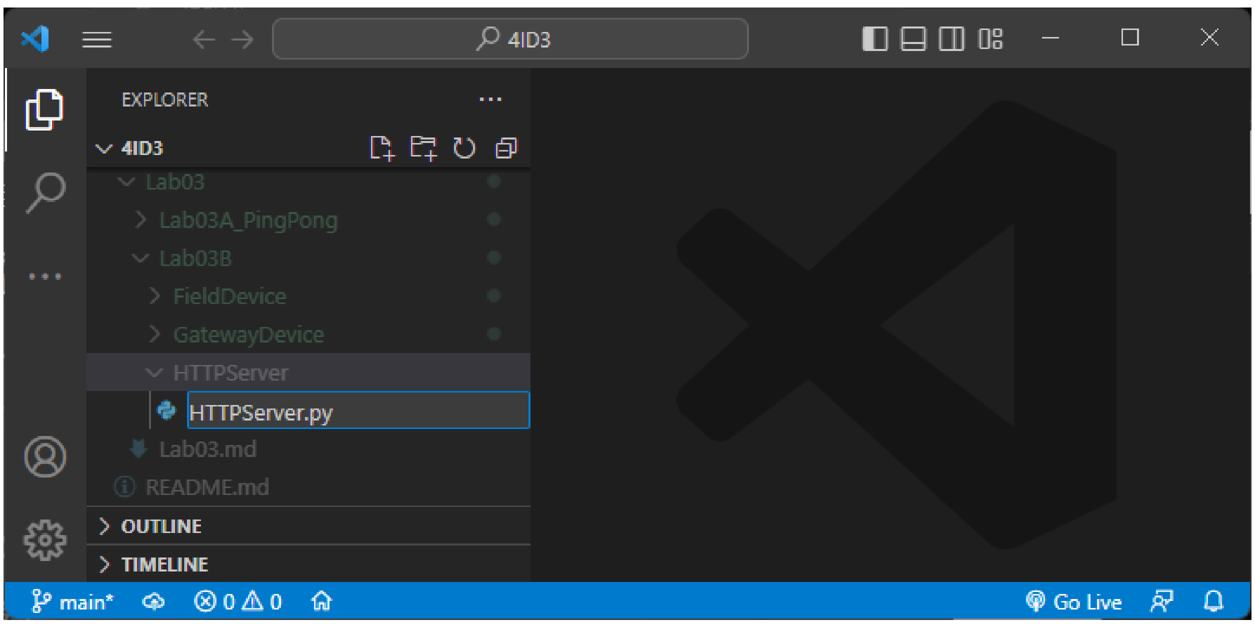
Create a new folder in the **4ID3/Lab03/Lab03B** directory called **HTTPServer.py**.





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



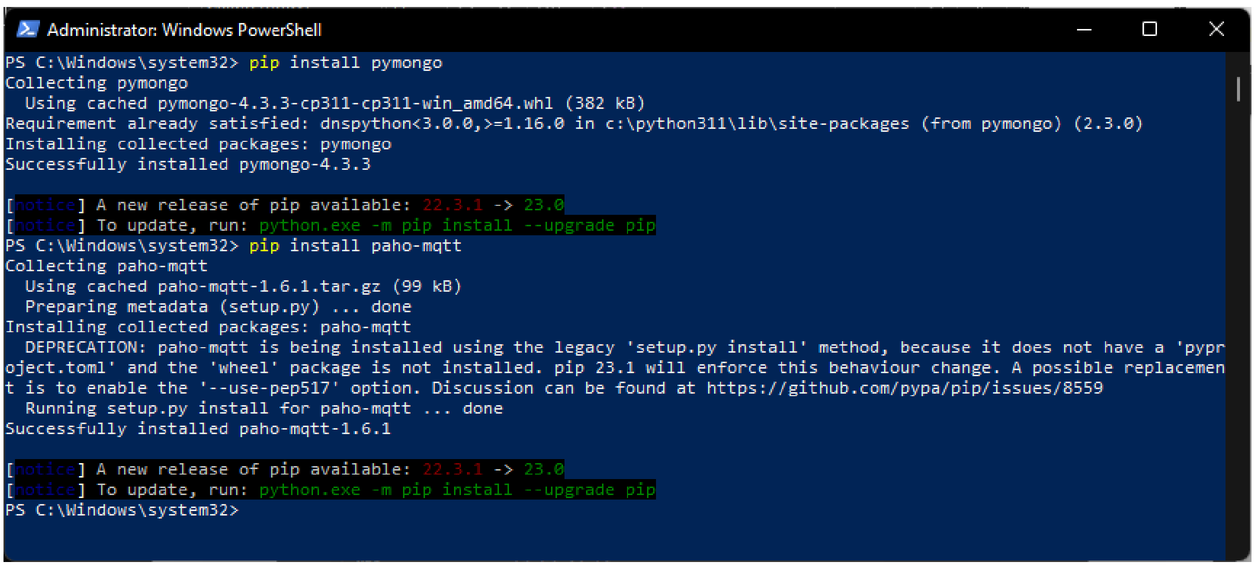


Python Libraries

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

pip install pymongo

pip install paho-mqtt



IoT Server Overview

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

HTTPServer.py U ×

Lab03 > Lab03B > HTTPServer > HTTPServer.py > ...

```python
#
#   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'
MQTT_PORT = 1883
MQTT_ROUTE = '/'

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

```

Set up your main function as such:

```python

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

```

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.

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


    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.

```python

    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())

```

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

```python

        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')

```

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

```python
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
```

Full code:

```python
#
#  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'
```

```
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()
```

```
            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)
```

```
    #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

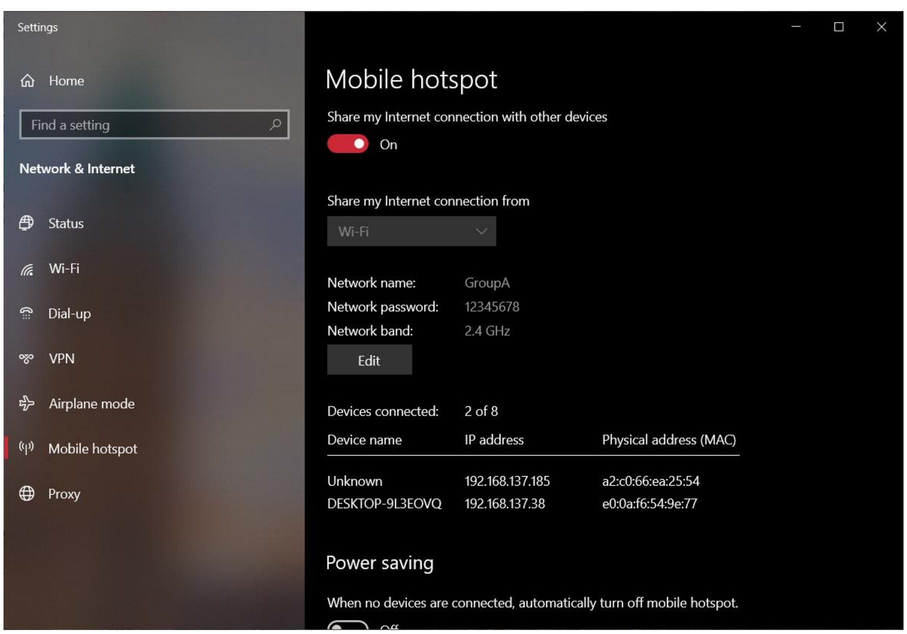
Have a **groupmate** use their computer to start a **Windows Mobile Hotspot**.

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.

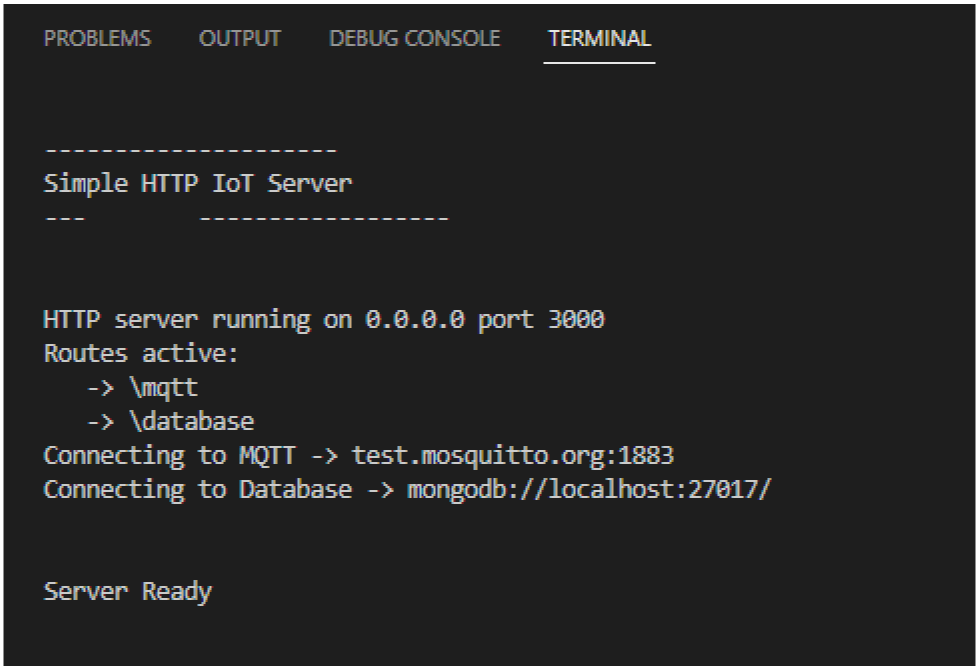
Connect your PC (*The PC running the HTTP server*) to the **same Mobile Hotspot** network.

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



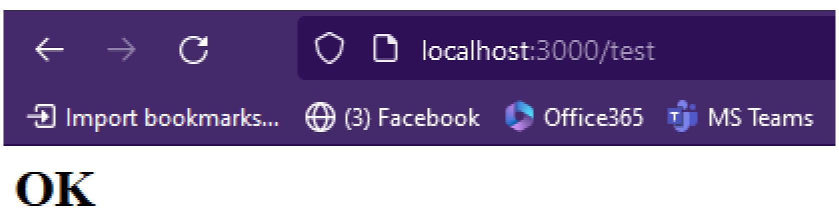
Next, launch the python script.

python HTTPServer.py



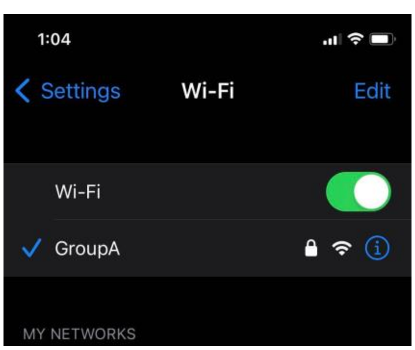
In a web browser on your **server PC**, search the following URL:

localhost:3000/test



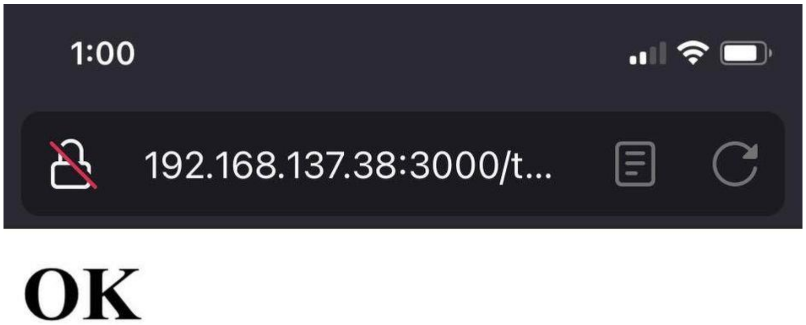
You should receive **OK** without any hassles.

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

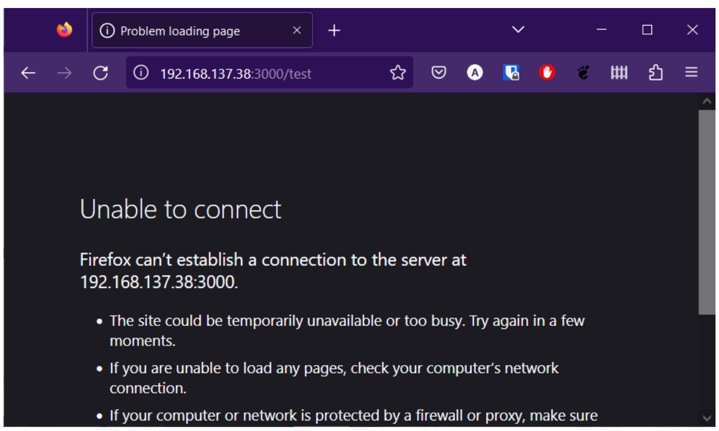


Type the IP address and path of your server URL and test route into a web browser on your mobile phone.

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

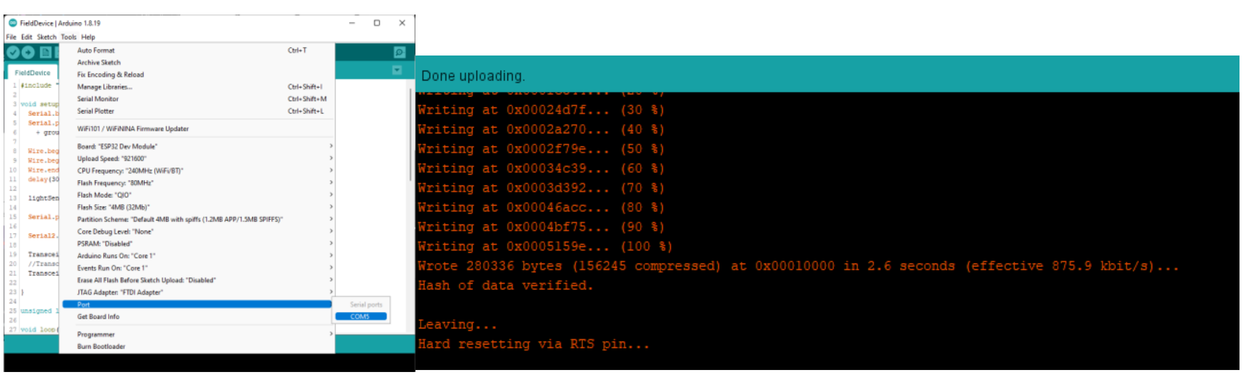


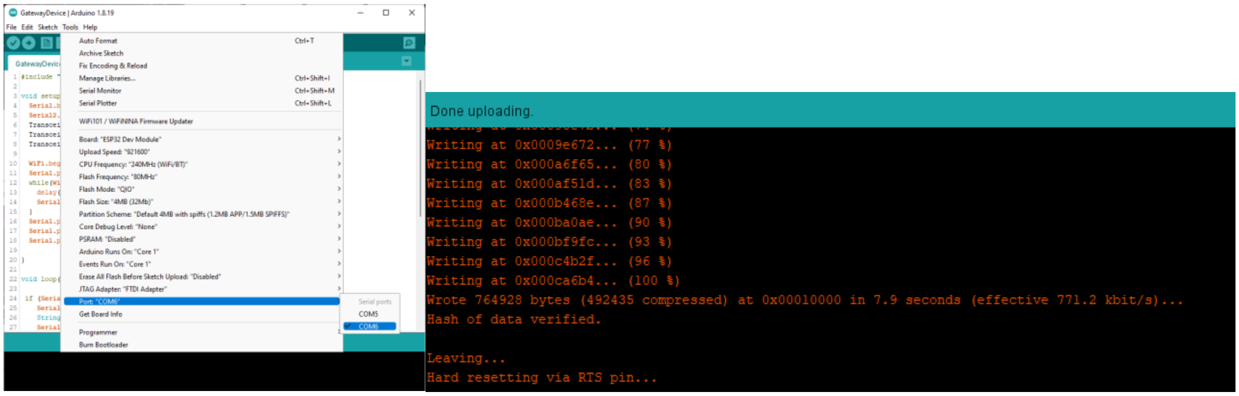
The reason why we require a third device, is that the PC running the mobile hotspot cannot connect to any servers on its network for security reasons.



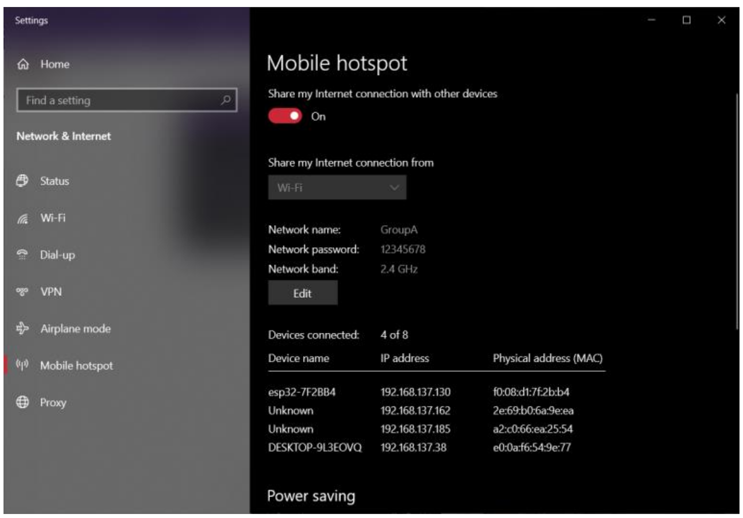
Integrating Everything Together

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



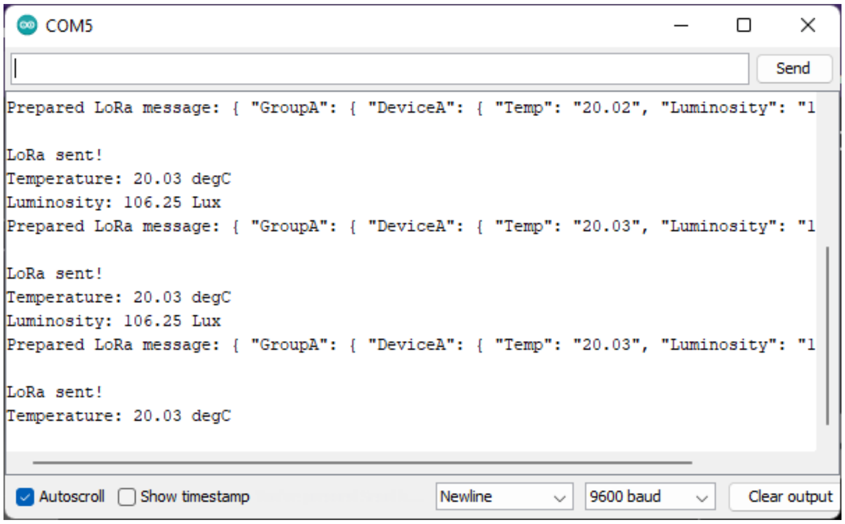


Watch the mobile hotspot PC and confirm that the **GatewayDevice** has connected successfully.

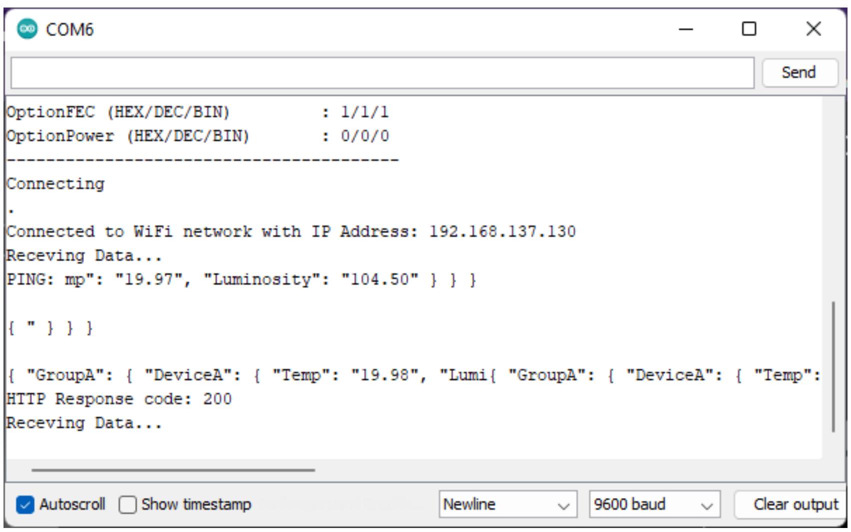


View each COM port and ensure that they are communicating through LoRa.

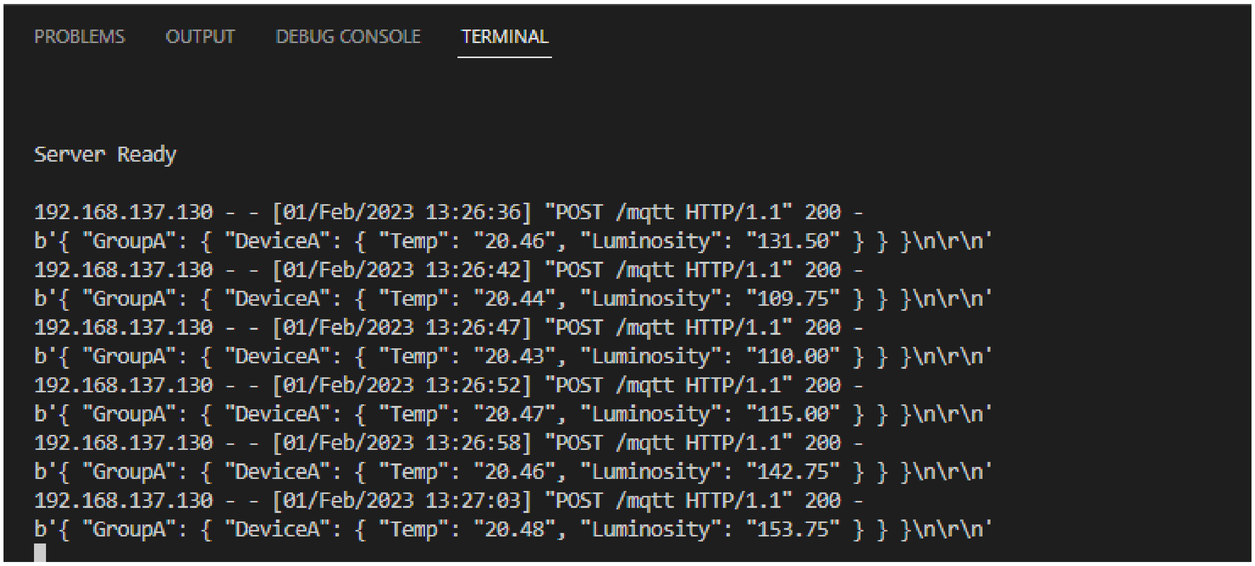
Field Device:



Gateway Device:



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

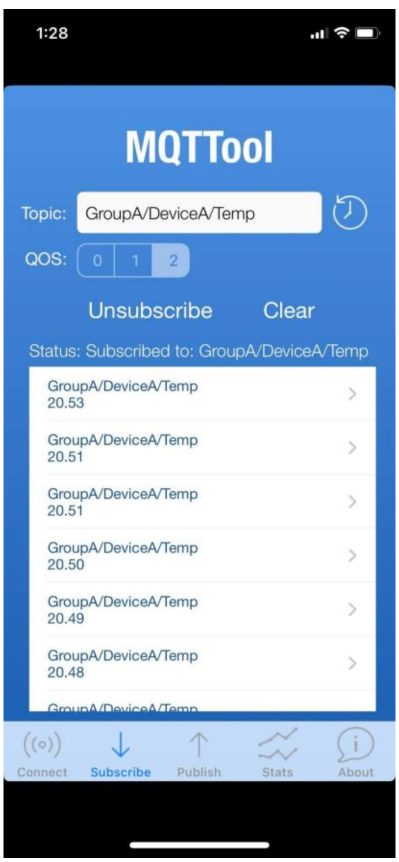


```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

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

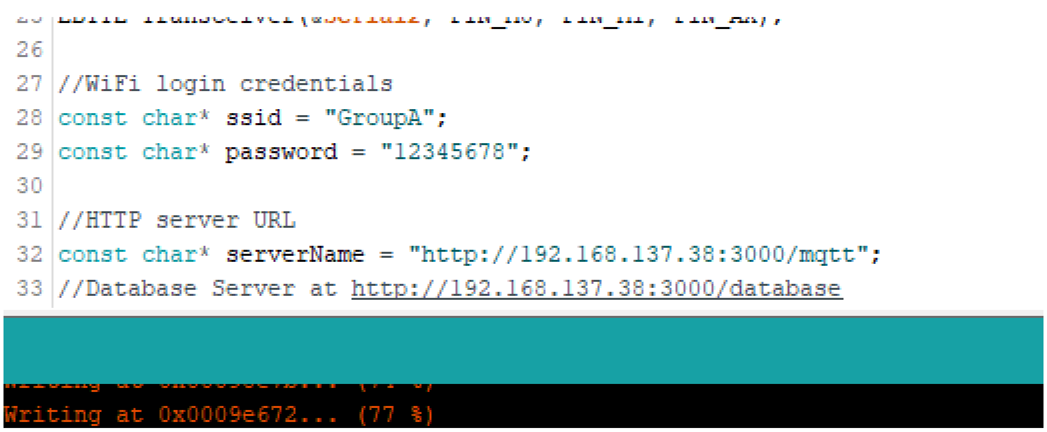
Create a NodeRED flow to visualize the data being published to **test.mosquitto.org**. Submit a screenshot with your report.

Exercise C

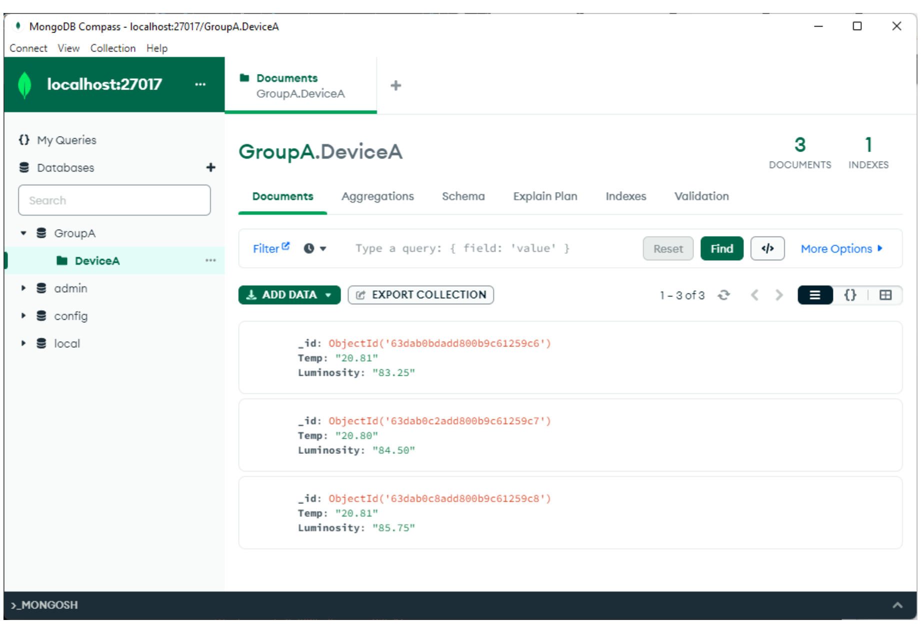
Modify the **URL** on the **Gateway Device** to make HTTP requests to the following URL instead of the MQTT route.

<server_ip>:3000/database

The setting can be found here:



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



If you have time, plot this data in excel.

END