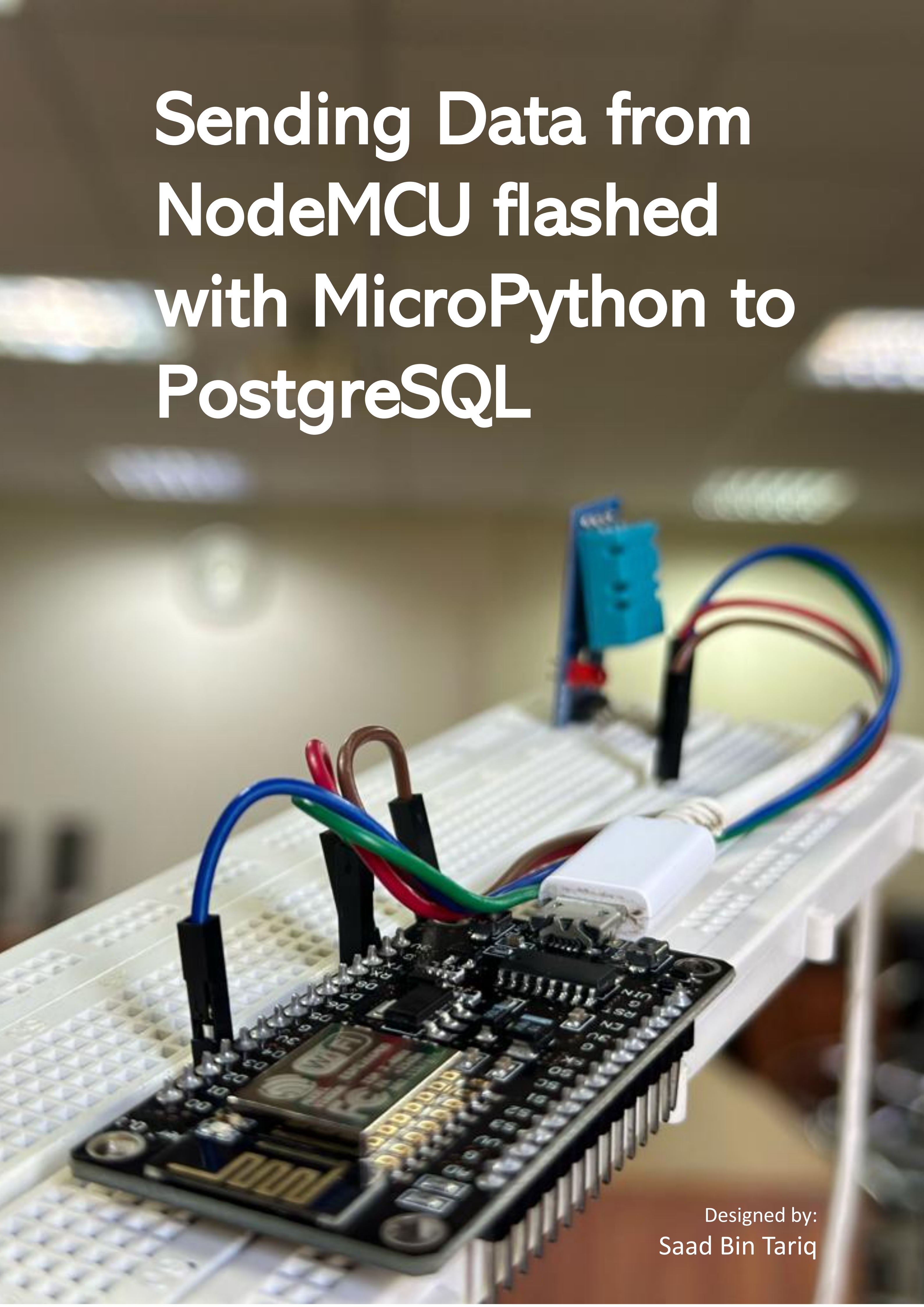
# Sending Data from NodeMCU flashed with MicroPython to PostgreSQL



Designed by:
Saad Bin Tariq

## Introduction

In this manual, you will embark on a journey to use a NodeMCU, flashed with MicroPython, with a DHT11 sensor and a PostgreSQL database. This project aims to provide you with a hands-on experience in the world of IoT (Internet of Things) and backend development, while also incorporating modern web technologies for data visualization.

## Aim

Our primary objective is to collect temperature and humidity data from a DHT11 sensor using a NodeMCU (ESP8266), and then securely store it in a PostgreSQL database through Node.js backend APIs. Once the data is successfully stored, we will create a React frontend application to display this information in a user-friendly interface.

## Components Required

- NodeMCU (ESP8266): The NodeMCU is a microcontroller board based on the ESP8266 WiFi module. It will serve as the IoT device to collect sensor data.

- DHT11/22 Sensor: The DHT11 is a low-cost digital temperature and humidity sensor. It will be used to capture environmental data.

- USB to Micro USB Cable: Required to connect the NodeMCU to your computer for programming and power.

- Breadboard and Jumper wires: Required for easy connection of the components.

## What is MicroPython?

MicroPython is a lightweight implementation of Python 3 designed for microcontrollers and embedded systems with limited resources. It supports Python syntax, offers an interactive environment for real-time testing, and has extensive hardware support. It's open source and ideal for IoT projects.

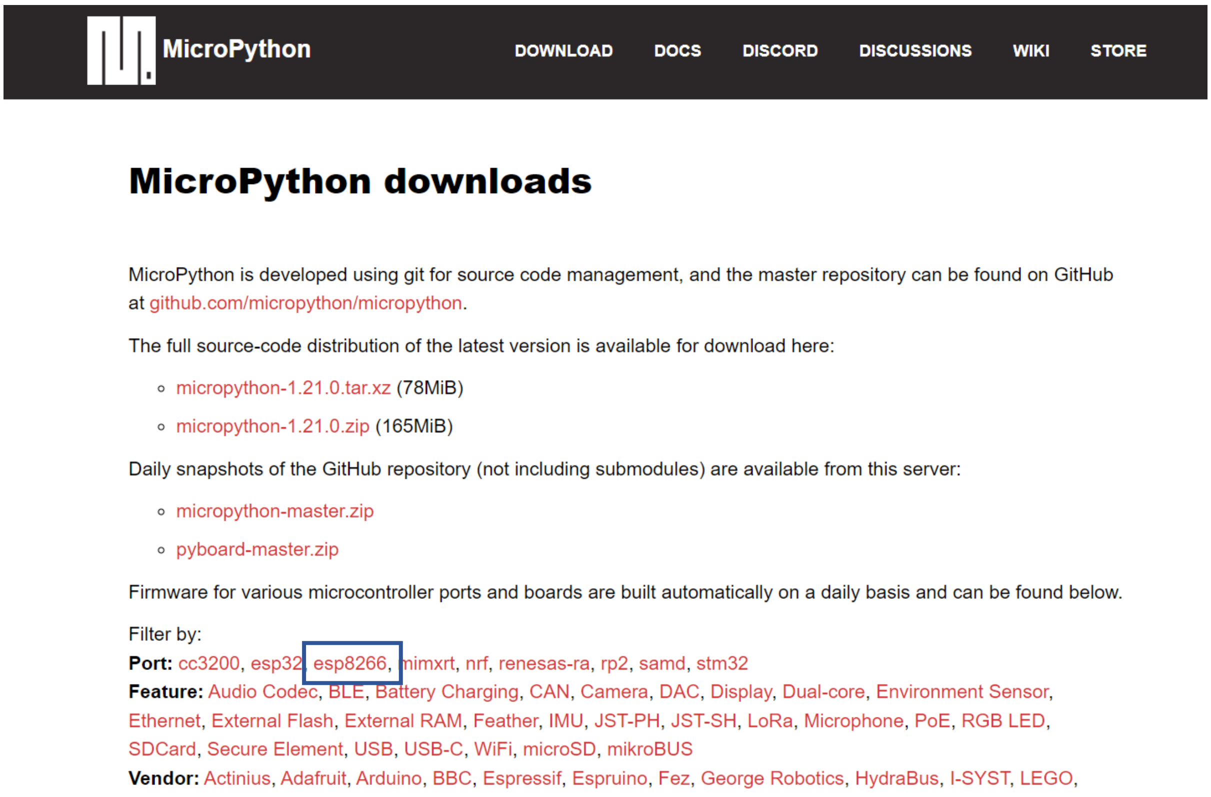
For more information, visit the official MicroPython website: https://micropython.org/

## Exercises

### 1. Flashing NodeMCU with MicroPython

Flashing an ESP8266 with MicroPython involves installing the MicroPython firmware onto the ESP8266 module. Here are the steps to flash an ESP8266 with MicroPython:

Download the MicroPython firmware for ESP8266 from the official MicroPython website. Ensure you select the appropriate firmware for your specific ESP8266 board model. I have used esp8266, hence, will download the firmware accordingly. To download the firmware, visit https://micropython.org/download/ and click on esp8266 from the port options.


MicroPython DOWNLOAD DOCS DISCORD DISCUSSIONS WIKI STORE

MicroPython downloads

MicroPython is developed using git for source code management, and the master repository can be found on GitHub at github.com/micropython/micropython.

The full source-code distribution of the latest version is available for download here:

- micropython-1.21.0.tar.xz (78MiB)
- micropython-1.21.0.zip (165MiB)

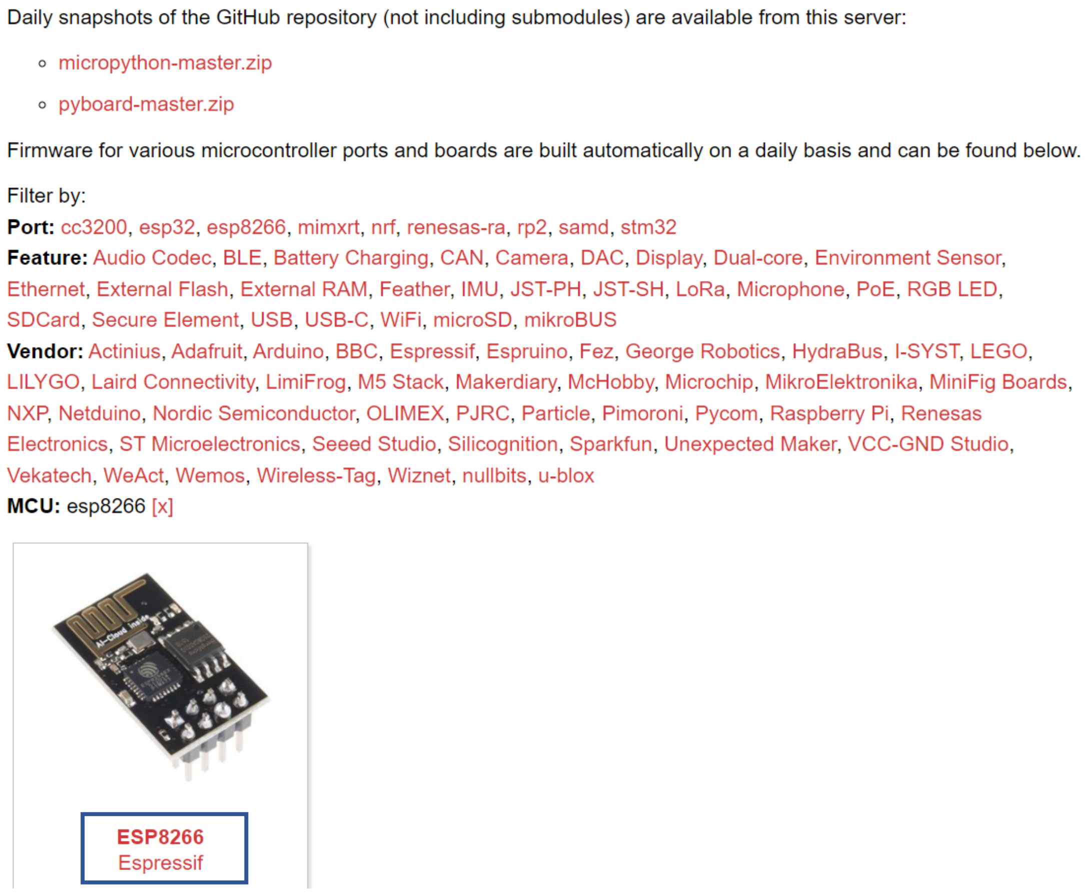
Daily snapshots of the GitHub repository (not including submodules) are available from this server:

- micropython-master.zip
- pyboard-master.zip

Firmware for various microcontroller ports and boards are built automatically on a daily basis and can be found below.

Filter by:
**Port:** cc3200, esp32, esp8266, mimxrt, nrf, renesas-ra, rp2, samd, stm32
**Feature:** Audio Codec, BLE, Battery Charging, CAN, Camera, DAC, Display, Dual-core, Environment Sensor, Ethernet, External Flash, External RAM, Feather, IMU, JST-PH, JST-SH, LoRa, Microphone, PoE, RGB LED, SDCard, Secure Element, USB, USB-C, WiFi, microSD, mikroBUS
**Vendor:** Actinius, Adafruit, Arduino, BBC, Espressif, Espruino, Fez, George Robotics, HydraBus, I-SYST, LEGO,

Scroll down to the *espressif* hyperlink and click it.


Daily snapshots of the GitHub repository (not including submodules) are available from this server:

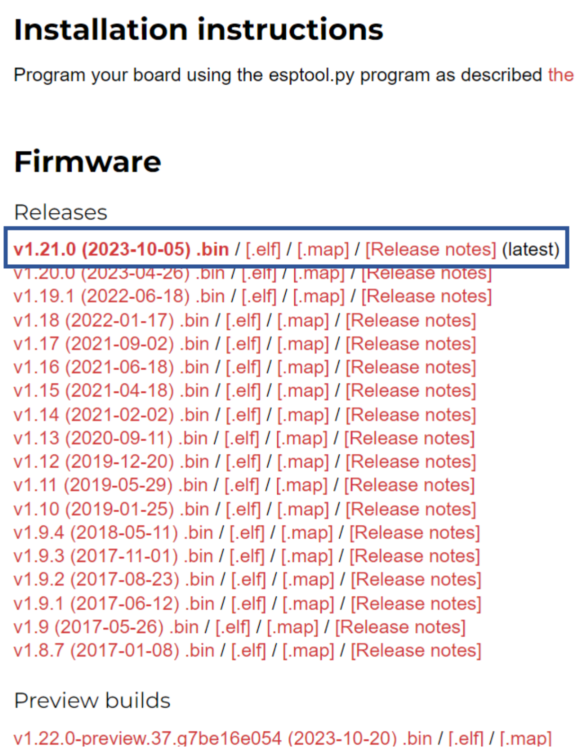
- micropython-master.zip
- pyboard-master.zip

Firmware for various microcontroller ports and boards are built automatically on a daily basis and can be found below.

Filter by:
**Port:** cc3200, esp32, esp8266, mimxrt, nrf, renesas-ra, rp2, samd, stm32
**Feature:** Audio Codec, BLE, Battery Charging, CAN, Camera, DAC, Display, Dual-core, Environment Sensor, Ethernet, External Flash, External RAM, Feather, IMU, JST-PH, JST-SH, LoRa, Microphone, PoE, RGB LED, SDCard, Secure Element, USB, USB-C, WiFi, microSD, mikroBUS
**Vendor:** Actinius, Adafruit, Arduino, BBC, Espressif, Espruino, Fez, George Robotics, HydraBus, I-SYST, LEGO, LILYGO, Laird Connectivity, LimiFrog, M5 Stack, Makerdiary, McHobby, Microchip, MikroElektronika, MiniFig Boards, NXP, Netduino, Nordic Semiconductor, OLIMEX, PJRC, Particle, Pimoroni, Pycom, Raspberry Pi, Renesas Electronics, ST Microelectronics, Seeed Studio, Silicognition, Sparkfun, Unexpected Maker, VCC-GND Studio, Vekatech, WeAct, Wemos, Wireless-Tag, Wiznet, nullbits, u-blox
**MCU:** esp8266 [x]

**ESP8266**
Espressif

Download the firmware in a known directory according to your NodeMCU version. I am using the latest version *v1.21.0*



## Installation instructions

Program your board using the esptool.py program as described the tutorial.

## Firmware

Releases

**v1.21.0 (2023-10-05) .bin** / [.elf] / [.map] / [Release notes] (latest)
v1.20.0 (2023-04-26) .bin / [.elf] / [.map] / [Release notes]
v1.19.1 (2022-06-18) .bin / [.elf] / [.map] / [Release notes]
v1.18 (2022-01-17) .bin / [.elf] / [.map] / [Release notes]
v1.17 (2021-09-02) .bin / [.elf] / [.map] / [Release notes]
v1.16 (2021-06-18) .bin / [.elf] / [.map] / [Release notes]
v1.15 (2021-04-18) .bin / [.elf] / [.map] / [Release notes]
v1.14 (2021-02-02) .bin / [.elf] / [.map] / [Release notes]
v1.13 (2020-09-11) .bin / [.elf] / [.map] / [Release notes]
v1.12 (2019-12-20) .bin / [.elf] / [.map] / [Release notes]
v1.11 (2019-05-29) .bin / [.elf] / [.map] / [Release notes]
v1.10 (2019-01-25) .bin / [.elf] / [.map] / [Release notes]
v1.9.4 (2018-05-11) .bin / [.elf] / [.map] / [Release notes]
v1.9.3 (2017-11-01) .bin / [.elf] / [.map] / [Release notes]
v1.9.2 (2017-08-23) .bin / [.elf] / [.map] / [Release notes]
v1.9.1 (2017-06-12) .bin / [.elf] / [.map] / [Release notes]
v1.9 (2017-05-26) .bin / [.elf] / [.map] / [Release notes]
v1.8.7 (2017-01-08) .bin / [.elf] / [.map] / [Release notes]

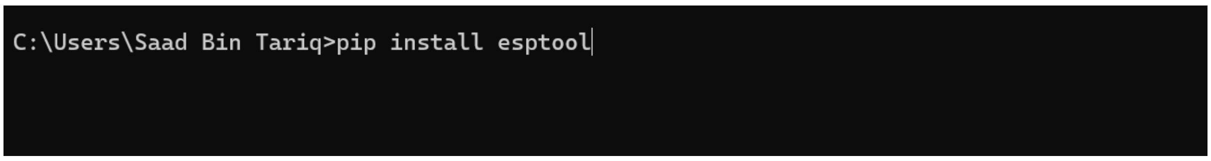
Preview builds

v1.22.0-preview.37.g7be16e054 (2023-10-20) .bin / [.elf] / [.map]

Open your command prompt and install esptool, this is a command-line utility for flashing.

For Windows

*pip install esptool*

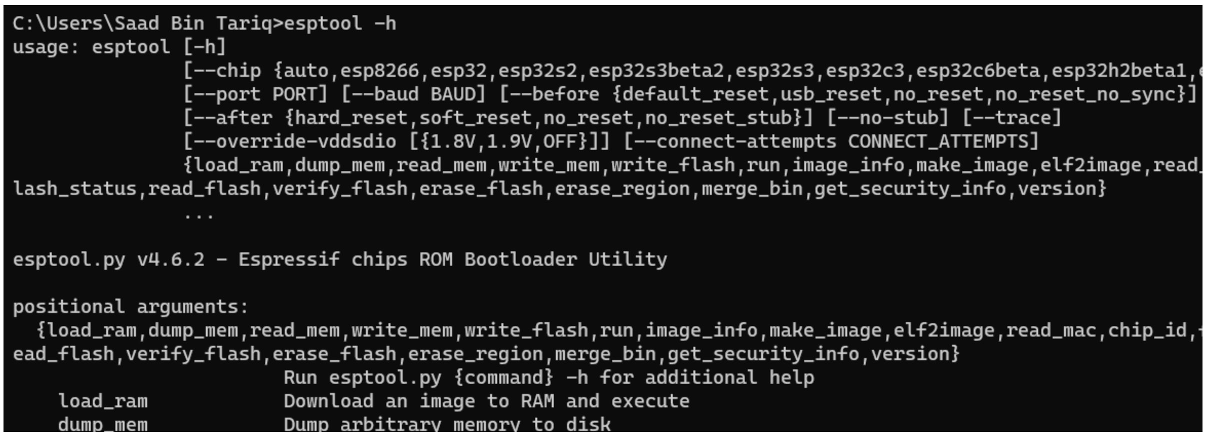


```
C:\Users\Saad Bin Tariq>pip install esptool
```

Run the following command, to check if esptool is properly installed.

*esptool -h*

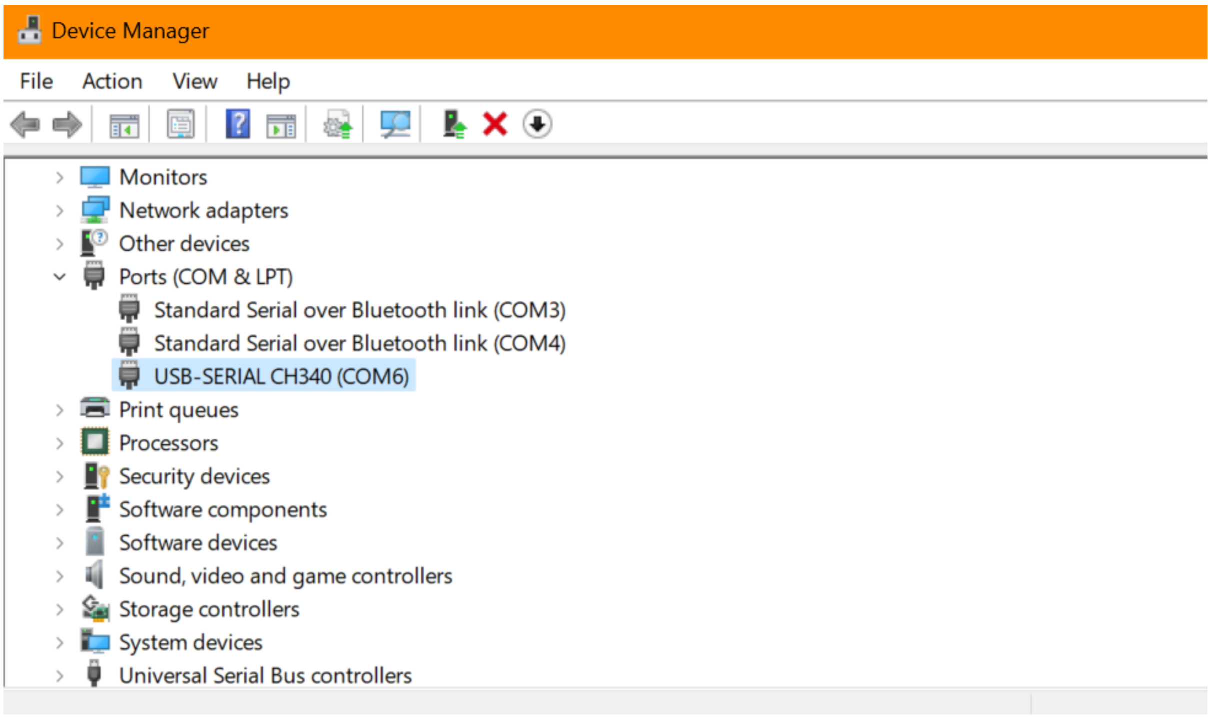
If you are getting the following output, it means that you have installed it correctly.



```
C:\Users\Saad Bin Tariq>esptool -h
usage: esptool [-h]
               [--chip {auto,esp8266,esp32,esp32s2,esp32s3beta2,esp32s3,esp32c3,esp32c6beta,esp32h2beta1,
               [--port PORT] [--baud BAUD] [--before {default_reset,usb_reset,no_reset,no_reset_no_sync}]
               [--after {hard_reset,soft_reset,no_reset,no_reset_stub}] [--no-stub] [--trace]
               [--override-vddsdio [{1.8V,1.9V,OFF}]] [--connect-attempts CONNECT_ATTEMPTS]
               {load_ram,dump_mem,read_mem,write_mem,write_flash,run,image_info,make_image,elf2image,read
lash_status,read_flash,verify_flash,erase_flash,erase_region,merge_bin,get_security_info,version}
               ...

esptool.py v4.6.2 - Espressif chips ROM Bootloader Utility

positional arguments:
  {load_ram,dump_mem,read_mem,write_mem,write_flash,run,image_info,make_image,elf2image,read_mac,chip_id,
ead_flash,verify_flash,erase_flash,erase_region,merge_bin,get_security_info,version}
                        Run esptool.py {command} -h for additional help
    load_ram            Download an image to RAM and execute
    dump_mem            Dump arbitrary memory to disk
```

Connect the ESP8266 to your computer using a USB cable. Ensure the ESP8266 is in programming mode. Determine the COM or serial port to which your ESP8266 is connected, this information is crucial for flashing.

Go to your device manager and click on Ports, to see details about your COM Port.



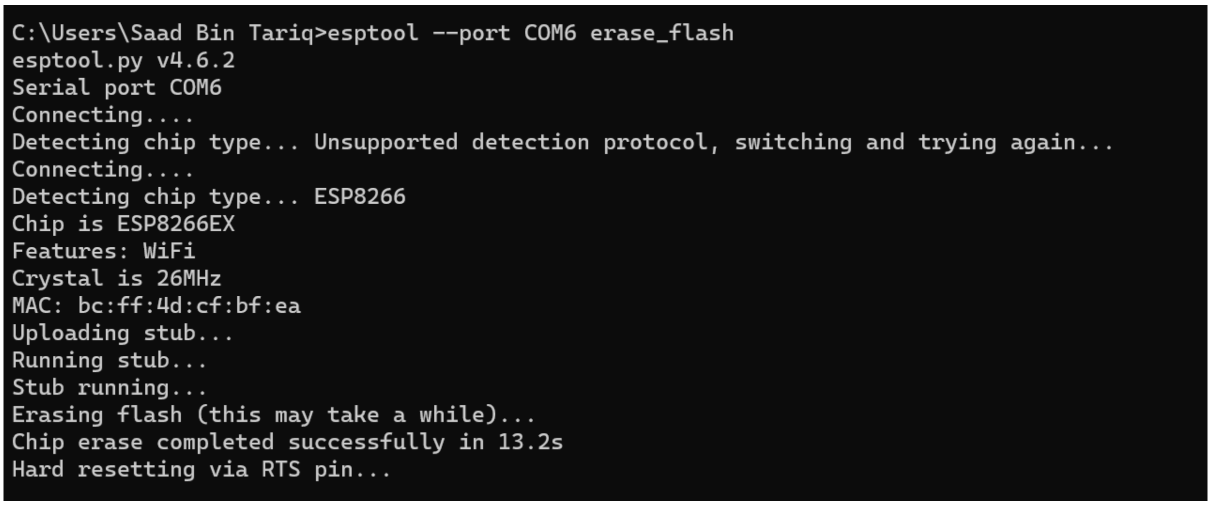
In order to flash MicroPython we first need to enter the bootloader mode and erase everything on the board and then upload the firmware again.

To activate the bootloader mode, press and hold the FLASH button on ESP8266 and press once the Reset (RST) button and then release the FLASH button. A single blink of Blue LED on the board will identify that bootloader mode is active.

To flash the esp8266 with new firmware, we first need to erase flash, using the following command:

*esptool --port COM6 erase_flash*

Change the 'COM6' with your correct COM Port



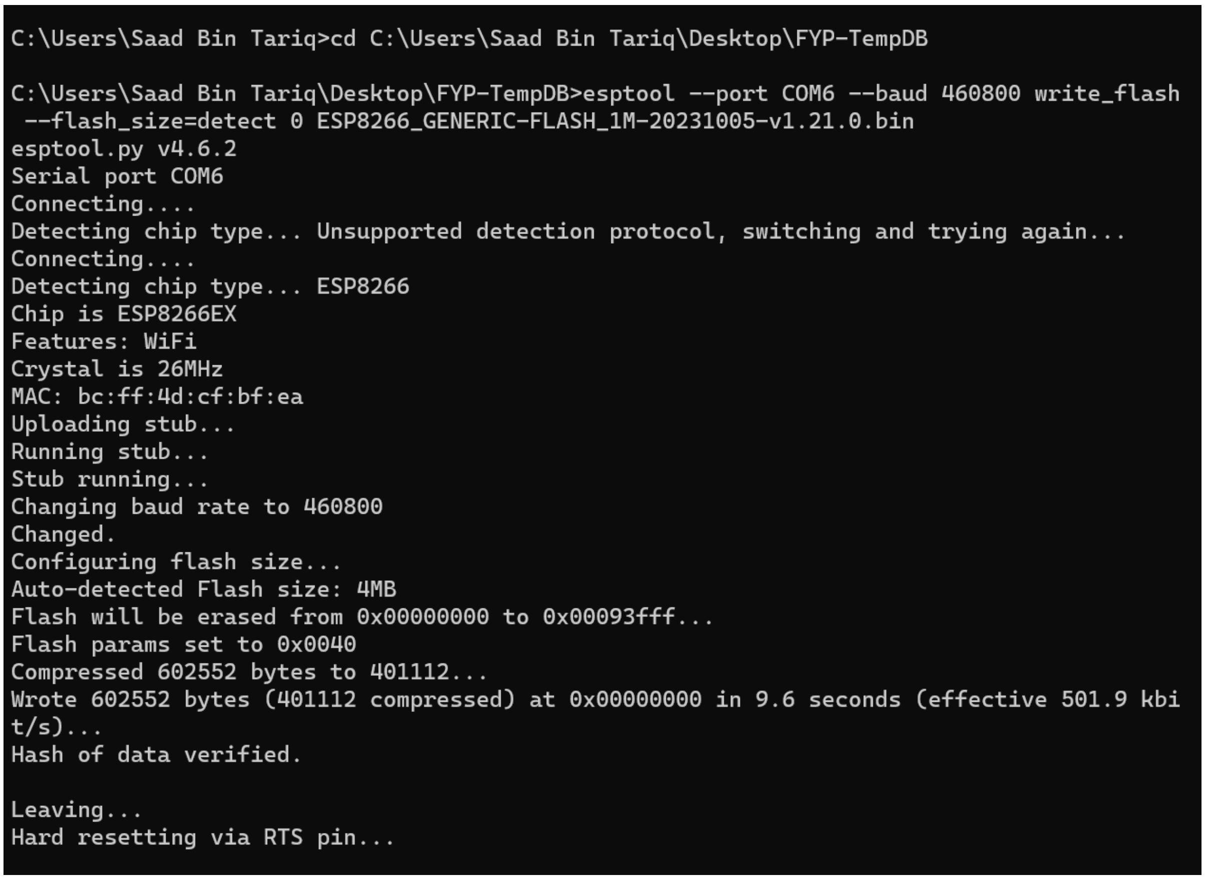
```
C:\Users\Saad Bin Tariq>esptool --port COM6 erase_flash
esptool.py v4.6.2
Serial port COM6
Connecting....
Detecting chip type... Unsupported detection protocol, switching and trying again...
Connecting....
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: bc:ff:4d:cf:bf:ea
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 13.2s
Hard resetting via RTS pin...
```

Note: Sometimes you might get errors while connecting the device, try reinstalling the driver *CH340* . You can download the driver from [https://oemdrivers.com/usb-ch340](https://oemdrivers.com/usb-ch340)

Now to flash the board with MicroPython firmware, open Windows Command Prompt and navigate to the folder where you saved the bin file previously:

> *cd C:\Users\Saad Bin Tariq\Desktop\FYP-TempDB*
>
> *esptool --port COM6 --baud 460800 write_flash --flash_size=detect 0 ESP8266_GENERIC-FLASH_1M-20231005-v1.21.0.bin*

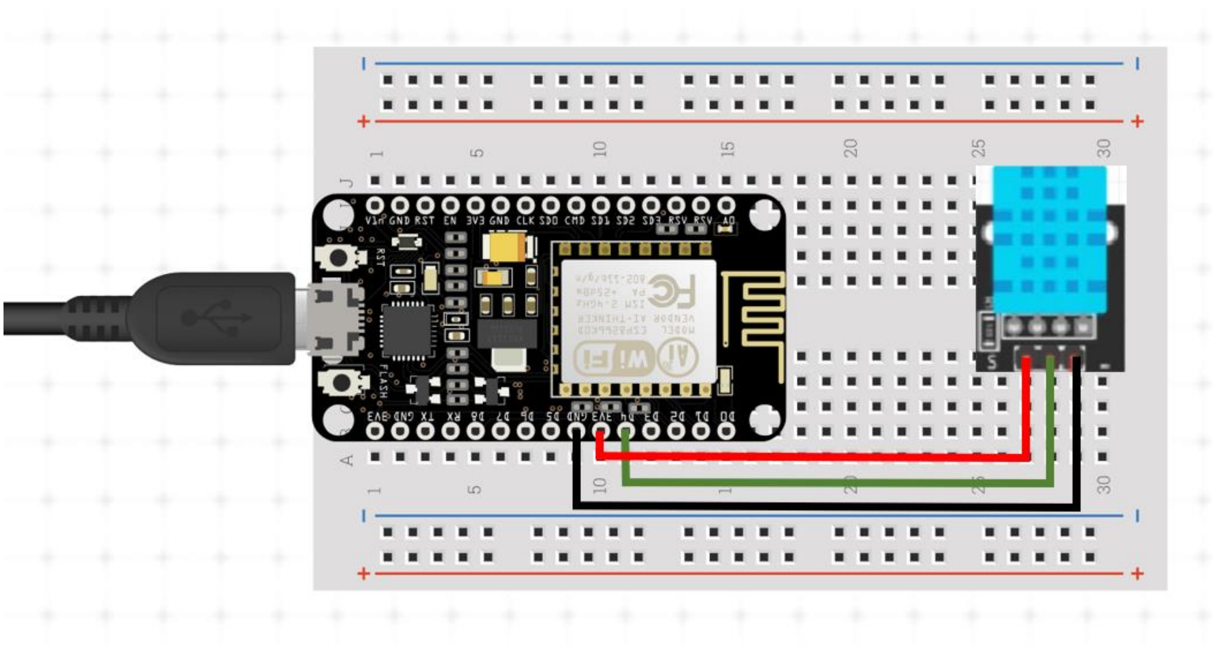
Note: Replace your COM Port with 'COM6' and the name of the (MicroPython) firmware bin file.



```
C:\Users\Saad Bin Tariq>cd C:\Users\Saad Bin Tariq\Desktop\FYP-TempDB

C:\Users\Saad Bin Tariq\Desktop\FYP-TempDB>esptool --port COM6 --baud 460800 write_flash
 --flash_size=detect 0 ESP8266_GENERIC-FLASH_1M-20231005-v1.21.0.bin
esptool.py v4.6.2
Serial port COM6
Connecting....
Detecting chip type... Unsupported detection protocol, switching and trying again...
Connecting....
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: bc:ff:4d:cf:bf:ea
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Flash will be erased from 0x00000000 to 0x00093fff...
Flash params set to 0x0040
Compressed 602552 bytes to 401112...
Wrote 602552 bytes (401112 compressed) at 0x00000000 in 9.6 seconds (effective 501.9 kbi
t/s)...
Hash of data verified.

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

## 2. Connecting Components

The wire diagram below shows how to connect your DHT11 with NodeMCU. In summary, As shown in the diagram above, connect the VCC on your DHT11 to 3V on the ESP8266, GND on DHT11 must be connected to a Ground pin on ESP8266 labelled as G. Lastly, connect the DATA pin on DHT 11 to a General-Purpose Input/Output pin (GPI/O) on ESP8266, as shown above, I had connected the DATA cable to D4 on ESP8266.

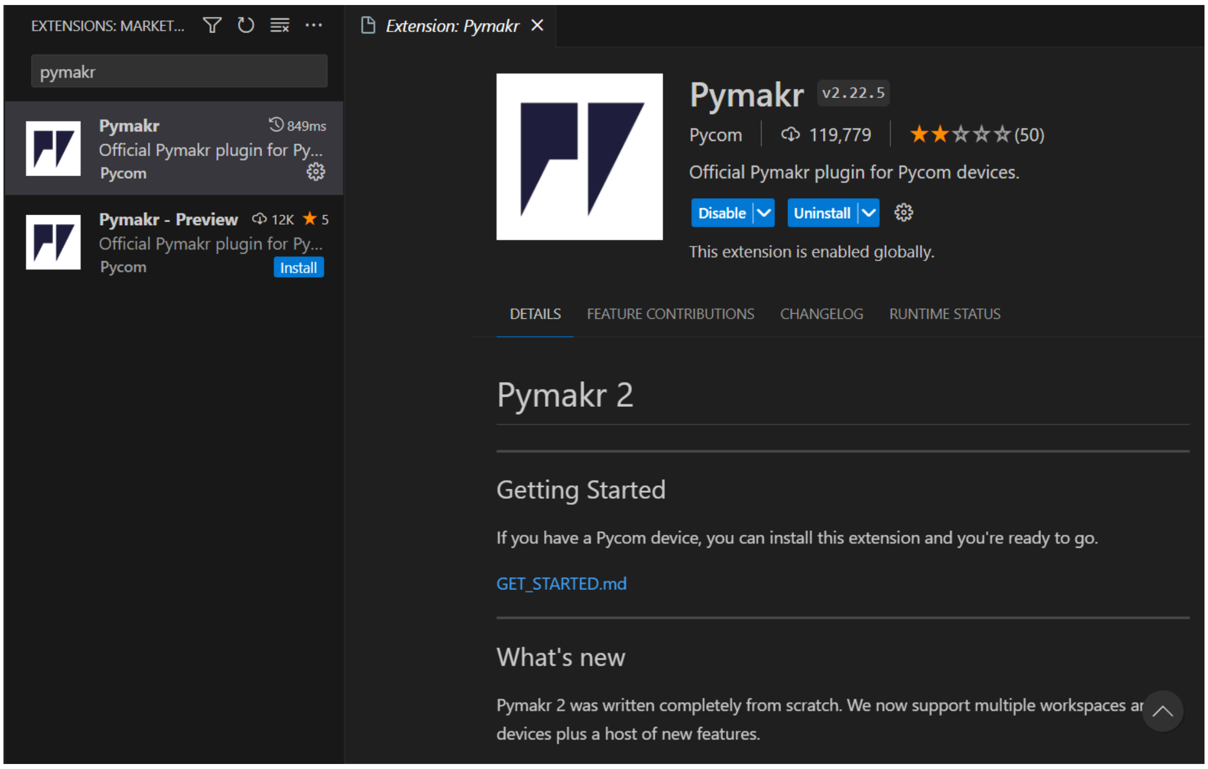


🟥 VCC
🟩 DATA
⬛ GND

### 3. Uploading Python code on ESP8266

We will be using PyMakr, VS Code with PyMakr is a popular development environment for writing and uploading Python code to ESP8266 and ESP32 microcontrollers.

Install the extension in VS Code:



Official Pymakr plugin for Pycom devices.

This extension is enabled globally.

Pymakr 2

Getting Started

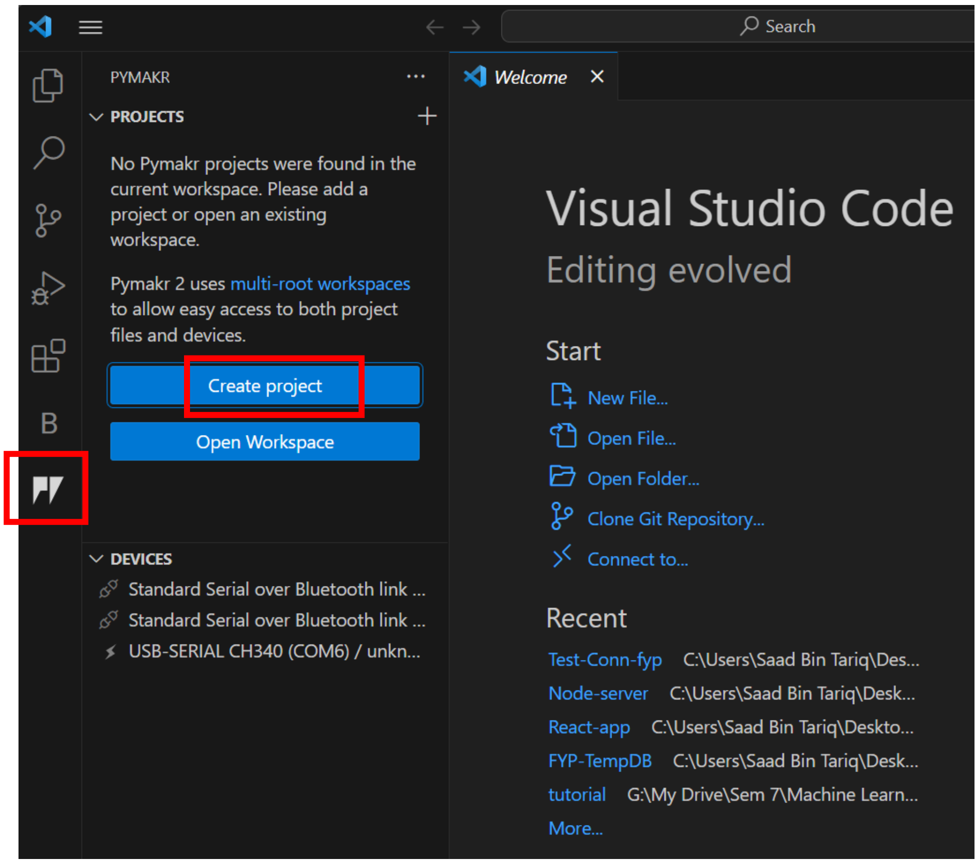
If you have a Pycom device, you can install this extension and you're ready to go.

GET_STARTED.md

What's new

Pymakr 2 was written completely from scratch. We now support multiple workspaces ar devices plus a host of new features.

Click on the create project button, and give the directory of your project folder:

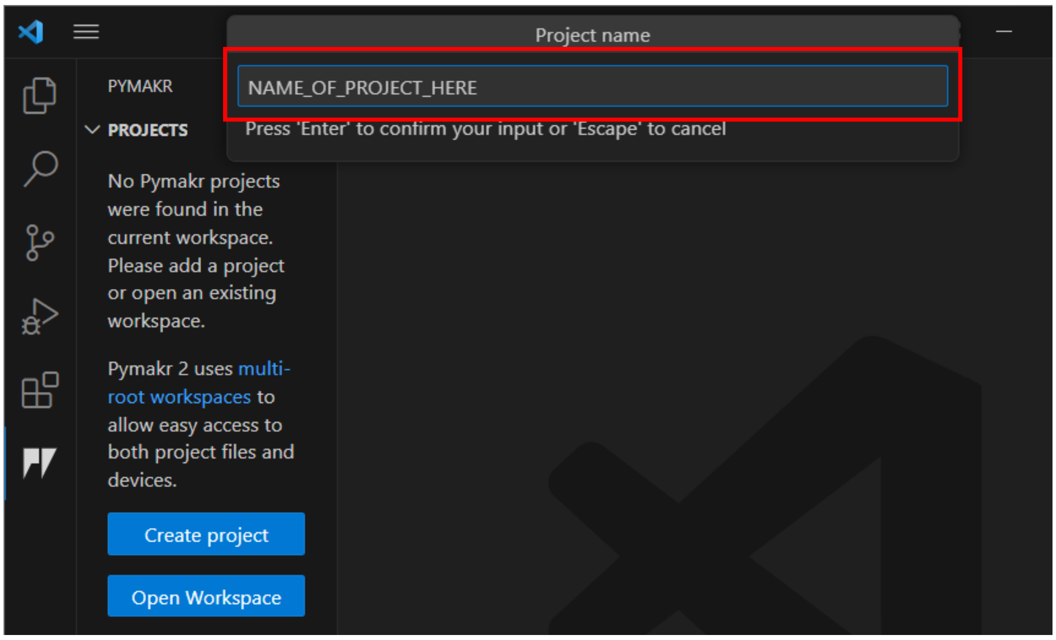


No Pymakr projects were found in the current workspace. Please add a project or open an existing workspace.

Pymakr 2 uses multi-root workspaces to allow easy access to both project files and devices.

Standard Serial over Bluetooth link ...
Standard Serial over Bluetooth link ...
USB-SERIAL CH340 (COM6) / unkn...

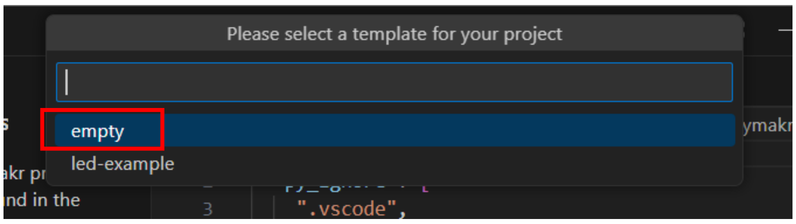
| | |
|---|---|
| Test-Conn-fyp | C:\Users\Saad Bin Tariq\Des... |
| Node-server | C:\Users\Saad Bin Tariq\Desk... |
| React-app | C:\Users\Saad Bin Tariq\Deskto... |
| FYP-TempDB | C:\Users\Saad Bin Tariq\Desk... |
| tutorial | G:\My Drive\Sem 7\Machine Learn... |



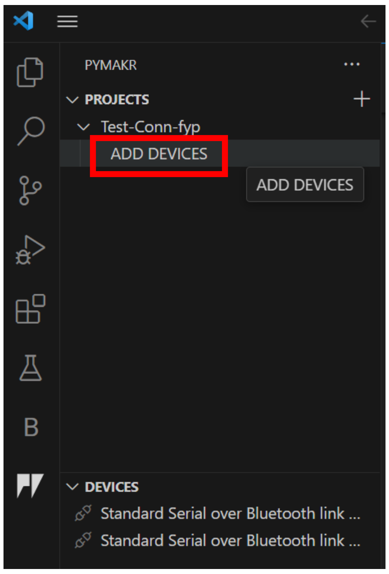
NAME_OF_PROJECT_HERE

Press 'Enter' to confirm your input or 'Escape' to cancel

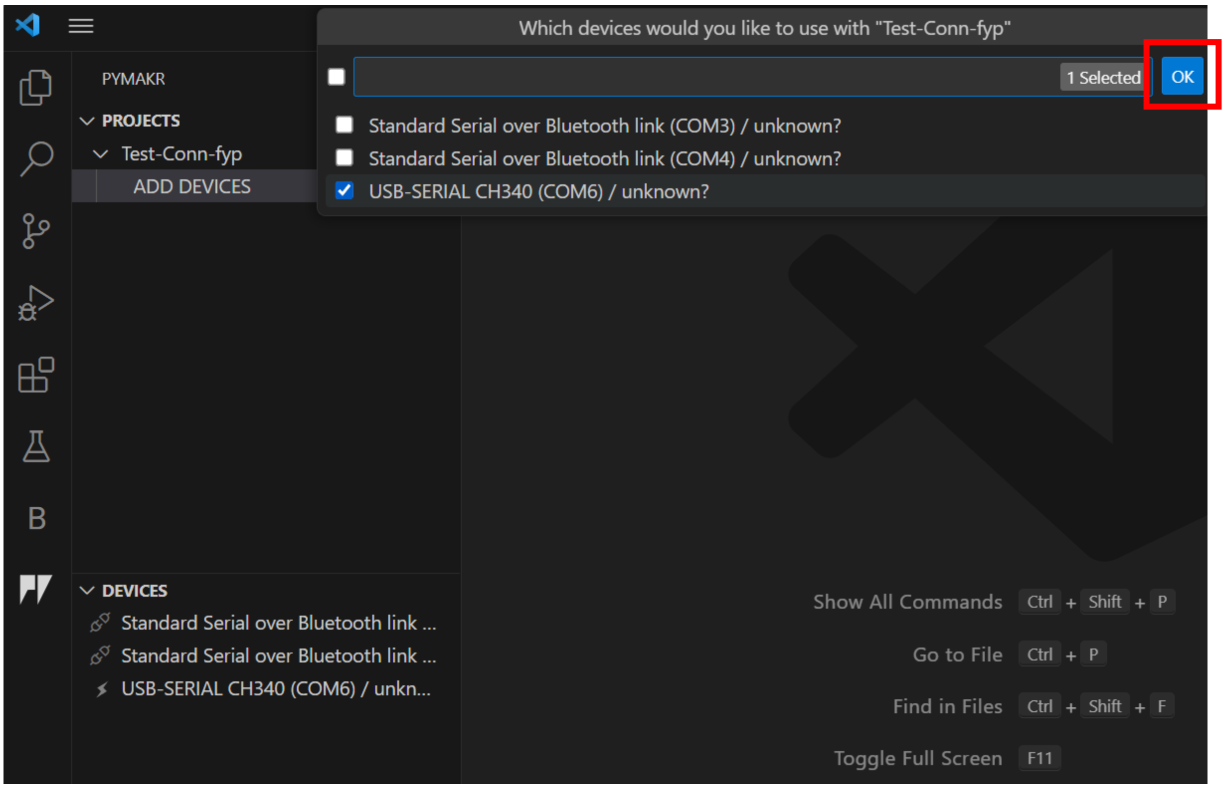
Click on empty



Add your NodeMCU device.


PYMAKR
PROJECTS
Test-Conn-fyp
ADD DEVICES
ADD DEVICES
DEVICES
Standard Serial over Bluetooth link ...
Standard Serial over Bluetooth link ...

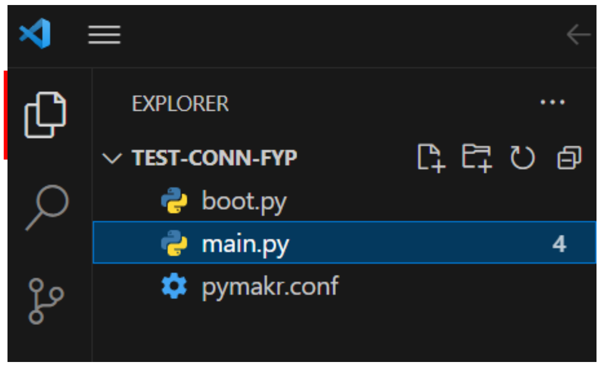
Select your device and click OK.


Which devices would you like to use with "Test-Conn-fyp"

- [ ] Standard Serial over Bluetooth link (COM3) / unknown?
- [ ] Standard Serial over Bluetooth link (COM4) / unknown?
- [x] USB-SERIAL CH340 (COM6) / unknown?

1 Selected
OK
PYMAKR
PROJECTS
Test-Conn-fyp
ADD DEVICES
DEVICES
Standard Serial over Bluetooth link ...
Standard Serial over Bluetooth link ...
USB-SERIAL CH340 (COM6) / unkn...
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Toggle Full Screen F11

Now in the extensions tab, you will find the main.py file. This will hold the main python code.



Paste the following code in main.py file, this code reads the temperature & humidity and sends it to the Node API:

```python
import machine
import dht
import time
import urequests

# Define DHT11 sensor pin
dht_pin = 2  # D4

# Define Wi-Fi credentials
wifi_ssid = "Ta**o" #Add your wifi name
wifi_password = "za***05" #Add your wifi password

# Define the HTTP endpoint to send data
http_endpoint = "https://cool-webs-share.loca.lt/sensor-data"

# Set GPIO pin for DHT11
dht_sensor = dht.DHT11(machine.Pin(dht_pin))

# Connect to Wi-Fi
def connect_to_wifi():
    import network
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print("Connecting to WiFi...")
        wlan.connect(wifi_ssid, wifi_password)
        while not wlan.isconnected():
            pass
    print("Connected to WiFi:", wlan.ifconfig())

# Main loop
try:
    connect_to_wifi()

    while True:
        # Perform a measurement
        dht_sensor.measure()

        # Read temperature and humidity
        temperature_celsius = dht_sensor.temperature()
        humidity_percentage = dht_sensor.humidity()

        # Print the results
        print("Temperature: {:.2f} °C, Humidity: {:.2f}
%".format(temperature_celsius, humidity_percentage))

        # Send data to the server using HTTP POST
        data = {"temperature": temperature_celsius, "humidity":
humidity_percentage}
        response = urequests.post(http_endpoint, json=data)

        # Check the response
        if response.status_code == 200:
            print("Data sent successfully")
        else:
            print("Failed to send data. Status code:", response.status_code)

        # Close the response
        response.close()
        # Wait for 10 seconds before the next measurement
        time.sleep(30)

except KeyboardInterrupt:
    print("Measurement stopped by user")
```

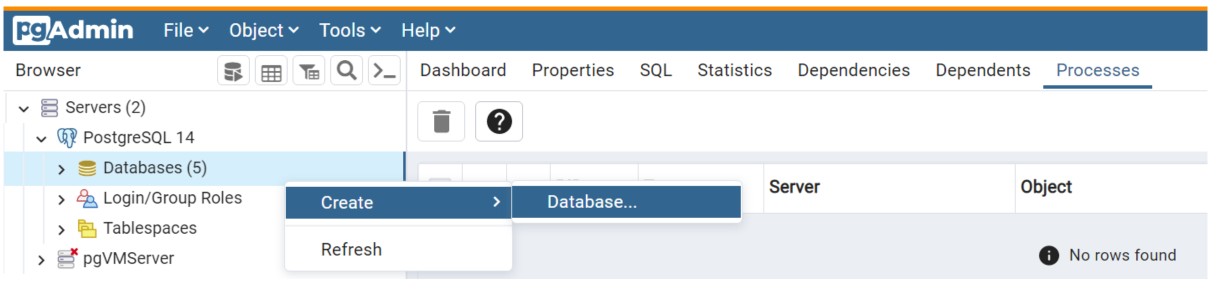
Note: In this this file you will have to make few changes, change your WIFI SSID and password and the endpoint url, about which I will talk in detail later. Endpoint url will be our hosted API to send data to postgres database.

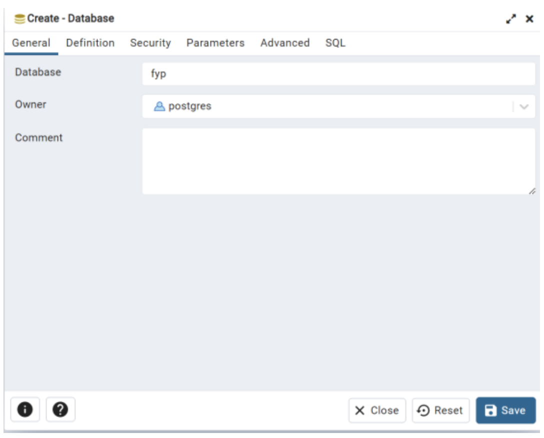
## 4. Creating the PostgreSQL Database

Now, we need to create a database where we will be storing our temperature and humidity data as well as time stamps. As per this exercise, we are using PostgreSQL.

Create a new database and then a new schema table that includes at least two columns, named temperature and humidity, to store the float values collected from the DHT11 sensor.

Create a new Database, enters name, and then click Save.

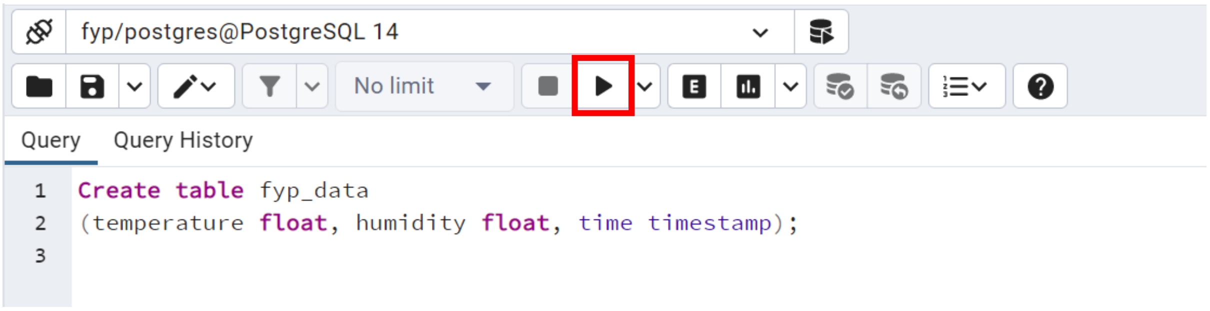




Database: fyp
Owner: postgres
Comment:

In the Query tool of your desired database, you may run the following SQL statements to create a table, and click the run button:

*Create table fyp_data*
*(temperature float, humidity float, time timestamp);*

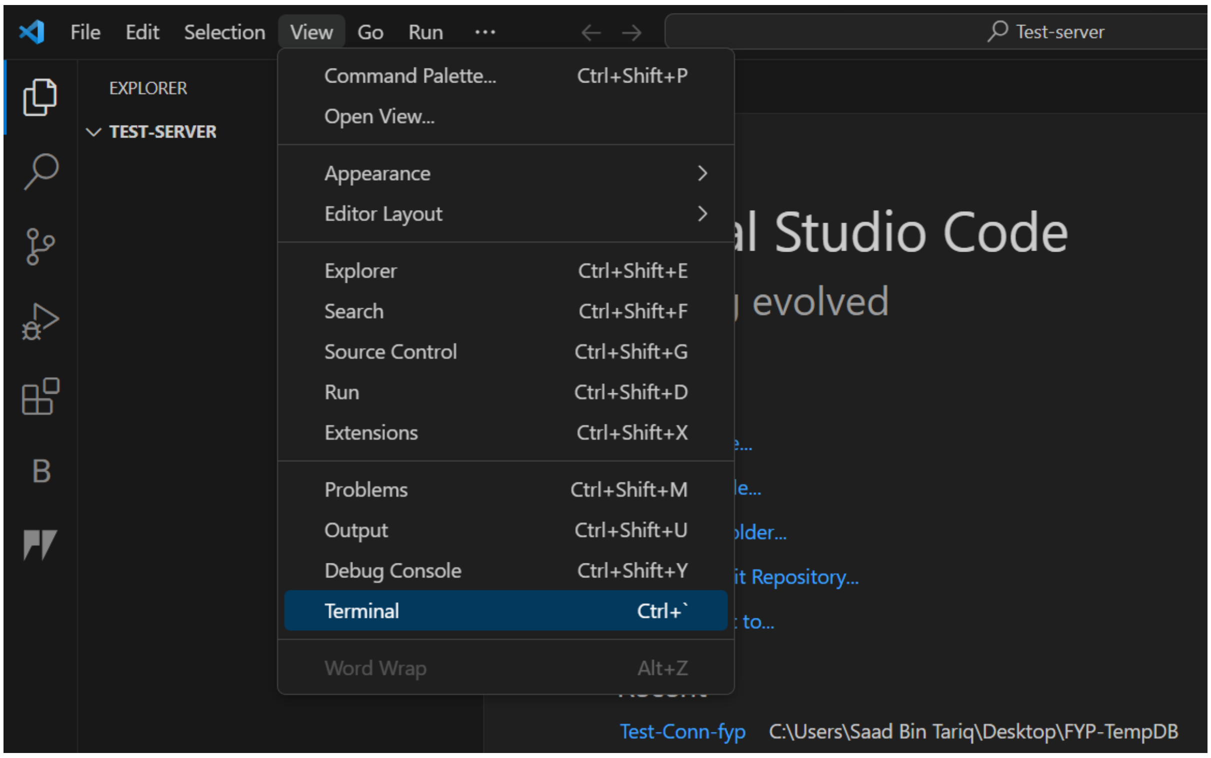


```
Create table fyp_data
(temperature float, humidity float, time timestamp);
```

## 5. Making Node Backend APIs to store and fetch data

Node.js is a server-side JavaScript runtime that allows you to create and run a web server. It's commonly used for receiving data from devices like the ESP8266 and processing that data via APIs. Node.js enables you to create a server that listens for incoming data from devices, processes it, and can then send that data to an API or store it in a database. This makes it a powerful tool for IoT and data integration, where the ESP8266 can send data to a Node.js server, which, in turn, can forward or process that data for various purposes.

Open the folder in VS Code where you want to save the node code, click view to make terminal visible.



Note: Make sure you have previously installed node or visit https://nodejs.org/en/download to download.

Now in the terminal, we must Initialize a Node.js project. To manage dependencies and configuration for your Node.js project, you should initialize it using the following command:

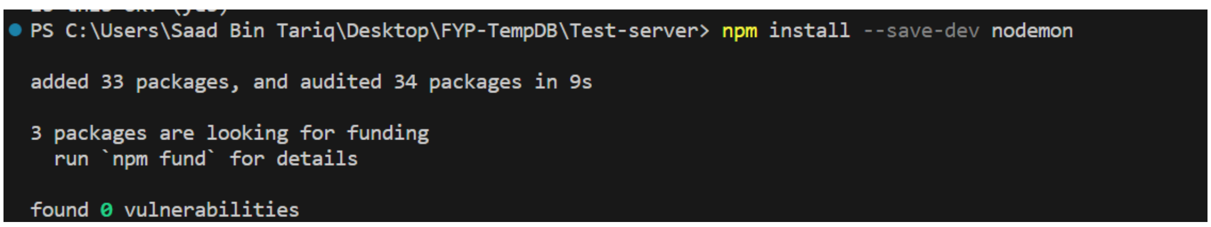
*npm init*

Press enter, enter… and then y to initialize the project, this will create a package.json file where you can see all your dependencies.

We will be using Nodemon to start our Node app, it is a tool for Node.js that automatically restarts your server when code changes occur, streamlining the development process by eliminating the need for manual server restarts. It greatly enhances the efficiency of Node.js development.

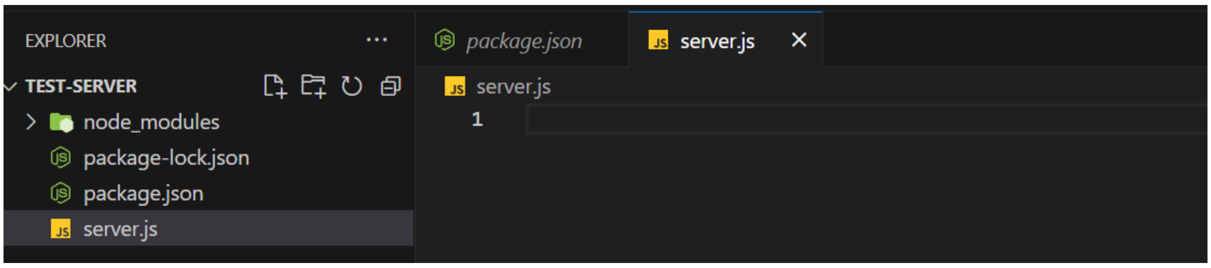
Run the following command in the PowerShell:

*npm install --save-dev nodemon*



```
PS C:\Users\Saad Bin Tariq\Desktop\FYP-TempDB\Test-server> npm install --save-dev nodemon

  added 33 packages, and audited 34 packages in 9s

  3 packages are looking for funding
    run `npm fund` for details

  found 0 vulnerabilities
```

Make a new file and name it server.js



Paste the following code in server.js file:

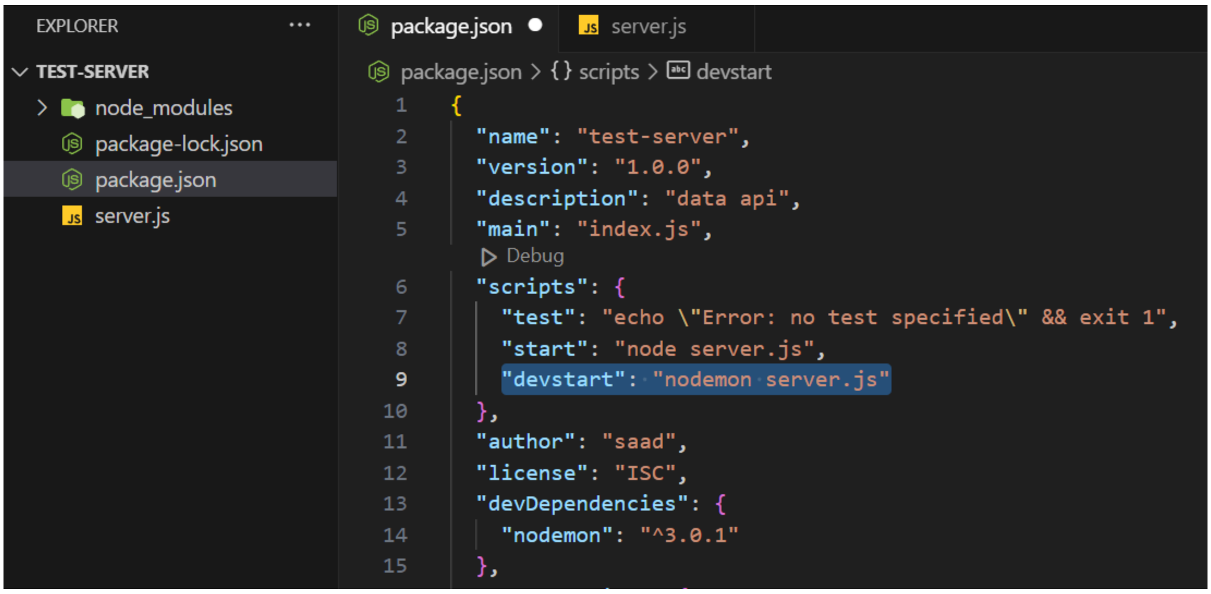
Note: Change the Postgres configuration details and the table name (if yours is different).

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const { Client } = require('pg');

const app = express();
const port = 3000;

// PostgreSQL database configuration
const dbConfig = {
  user: 'postgres',
  host: 'localhost',
  database: 'fyp',
  password: 'postgres',
  port: 5432,
};

const client = new Client(dbConfig);
```

```
// Middleware to parse JSON in HTTP requests
app.use(cors());
app.use(bodyParser.json());

// Connect to the PostgreSQL database
client.connect()
  .then(() => {
    console.log('Connected to PostgreSQL database');
  })
  .catch((err) => {
    console.error('Error connecting to PostgreSQL database', err);
  });

// Endpoint to receive temperature and humidity data in a POST request
app.post('/sensor-data', async (req, res) => {
  try {
    // Extract temperature and humidity from the request body
    const temperature = req.body.temperature;
    const humidity = req.body.humidity;

    // Validate and process the data as needed
    console.log(`Received Temperature: ${temperature}°C, Humidity:
${humidity}%`);

    // Insert data into the database
    //change your table name
    const query = 'INSERT INTO fyp_data (temperature, humidity) VALUES ($1,
$2)';
    const values = [temperature, humidity];

    await client.query(query, values);

    // Respond to the client
    res.status(200).json({ message: 'Data received and inserted successfully'
});
  } catch (error) {
    console.error('Error processing data:', error);
    res.status(500).json({ error: 'Internal Server Error' });
  }
});

// Endpoint to retrieve all data from the table
app.get('/all-data', async (req, res) => {
  try {
    const query = 'SELECT * FROM fyp_data ORDER BY timestamp DESC';
    const result = await client.query(query);
    res.status(200).json(result.rows);
  } catch (error) {
    console.error('Error retrieving data:', error);
    res.status(500).json({ error: 'Internal Server Error' });
  }
});

// Start the Express server
app.listen(port, () => {
  console.log(`Server listening on port ${port}`);
});

// Handle server termination
process.on('SIGINT', () => {
  console.log('Closing the server and PostgreSQL connection');
  client.end()
    .then(() => {
      console.log('PostgreSQL connection closed');
      process.exit(0);
    })
    .catch((err) => {
      console.error('Error closing PostgreSQL connection', err);
      process.exit(1);
    });
});
```

Now open the package.json file and add the following line under Scripts:

*"devstart": "nodemon server.js"*

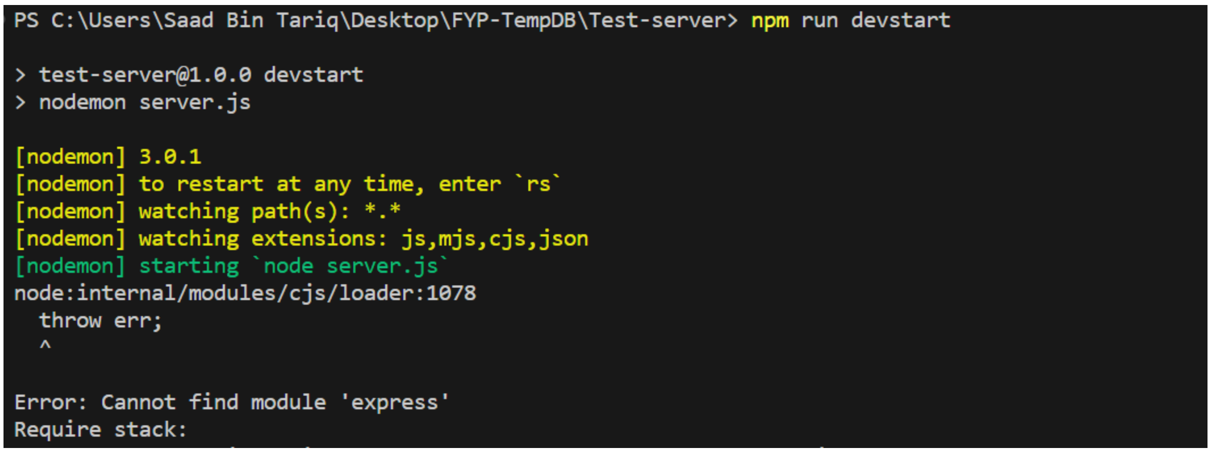


```
{
  "name": "test-server",
  "version": "1.0.0",
  "description": "data api",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js",
    "devstart": "nodemon server.js"
  },
  "author": "saad",
  "license": "ISC",
  "devDependencies": {
    "nodemon": "^3.0.1"
  },
```

To start the Node App, run the following command:

*npm run devstart*

This will produce an error that express is not installed. Press ‘Ctrl + C’ to stop the Server.

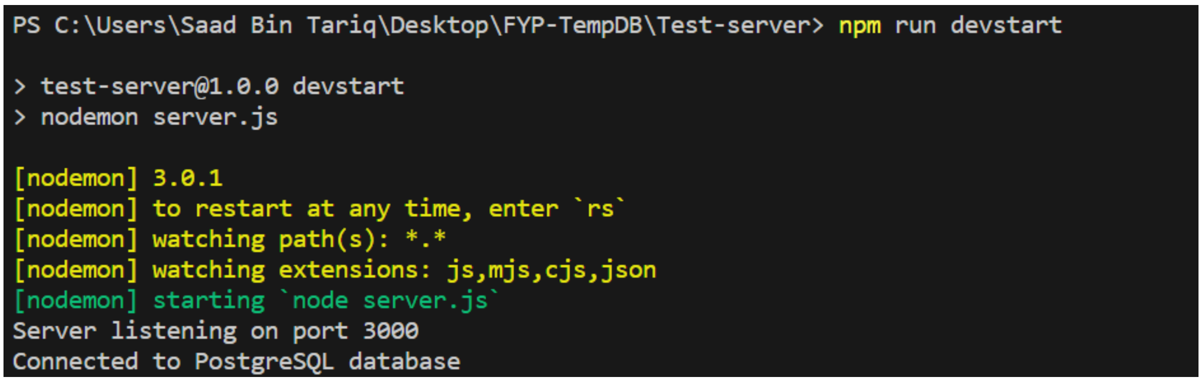


```
PS C:\Users\Saad Bin Tariq\Desktop\FYP-TempDB\Test-server> npm run devstart

> test-server@1.0.0 devstart
> nodemon server.js

[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
node:internal/modules/cjs/loader:1078
  throw err;
  ^

Error: Cannot find module 'express'
Require stack:
```

We need to install all the modules required, run the following command, or install one by one and restart the server:

*npm install express, body-parser, cors, pg*

*npm install -g localtunnel*



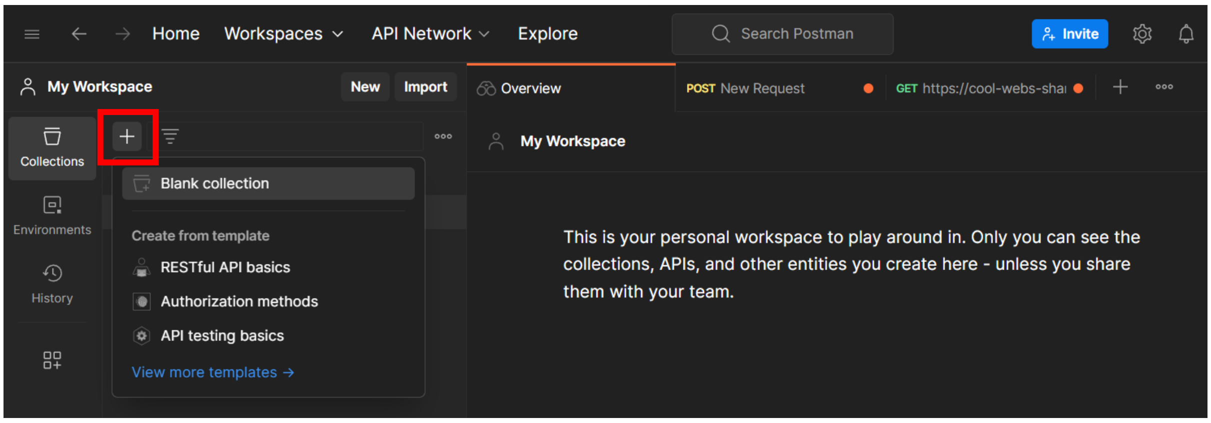
```
PS C:\Users\Saad Bin Tariq\Desktop\FYP-TempDB\Test-server> npm run devstart

> test-server@1.0.0 devstart
> nodemon server.js

[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Server listening on port 3000
Connected to PostgreSQL database
```

## 6. Testing the APIs with Postman (Optional)

Postman is a powerful and user-friendly API testing and development tool that simplifies the process of working with APIs. It allows developers to send HTTP requests to APIs, examine responses, and test various aspects of API functionality. With a user-friendly graphical interface, Postman provides features for creating, organizing, and running tests, making it an indispensable tool for API development and testing. Its capabilities include handling various request types, managing environments, scripting tests, and generating detailed documentation, making it a comprehensive solution for API testing and development.

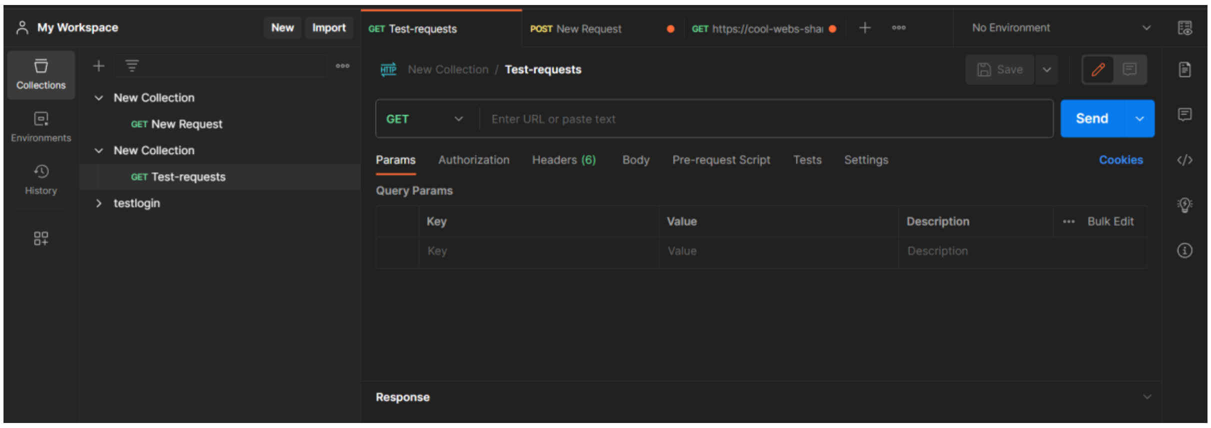
To download: [https://www.postman.com/downloads/](https://www.postman.com/downloads/) or continue on the online app.

Click on '+' to add a new blank collection, and add a request



This is your personal workspace to play around in. Only you can see the collections, APIs, and other entities you create here - unless you share them with your team.

You will see the following GUI.



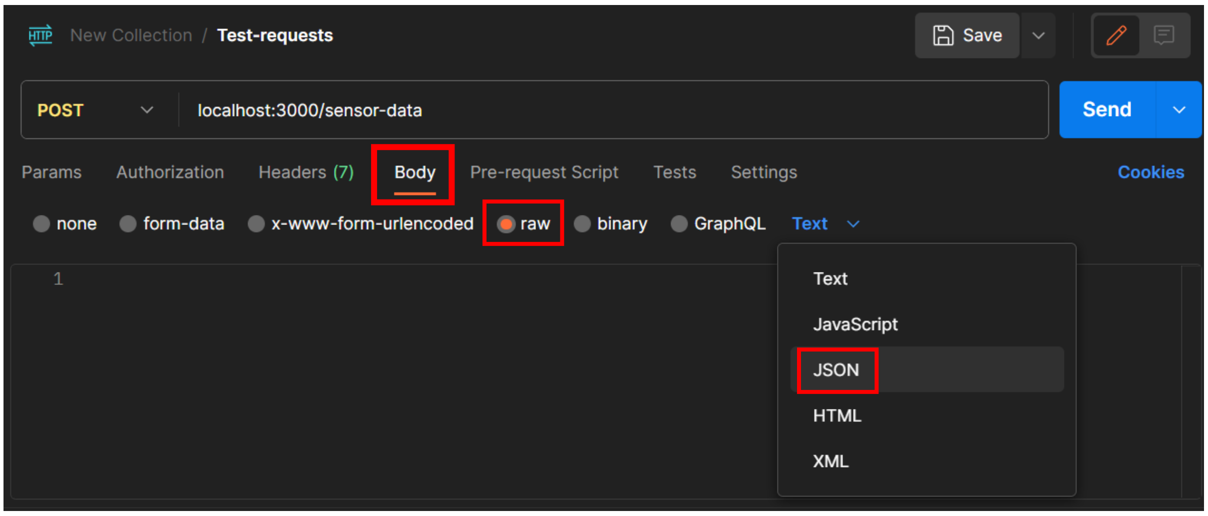
To the create data API change the method to POST and add the url:

*http://localhost:3000/sensor-data*

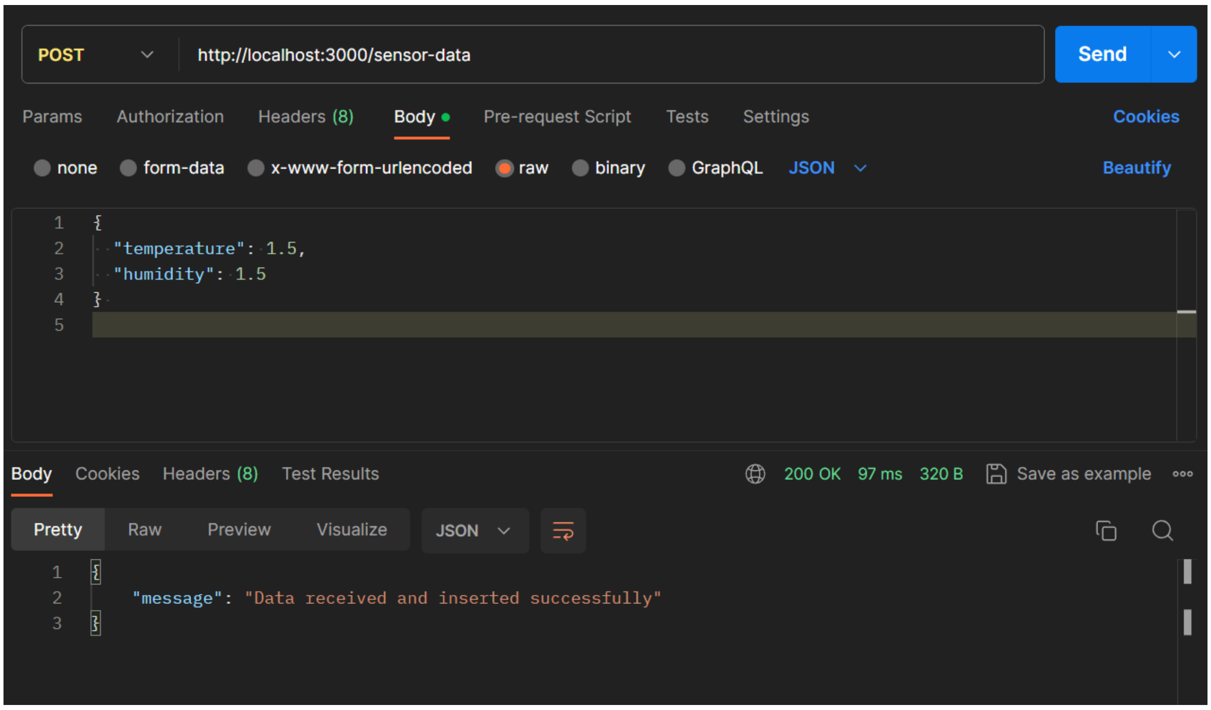
Then click on 'Body', then 'raw' and then JSON. Past the following as a dummy data, click Send:

```
{
  "temperature": 1.5,
  "humidity": 1.5
}
```

Note: Make sure that the Node Server is running



The following should be your output:


POST http://localhost:3000/sensor-data

```
{
  "temperature": 1.5,
  "humidity": 1.5
}
```

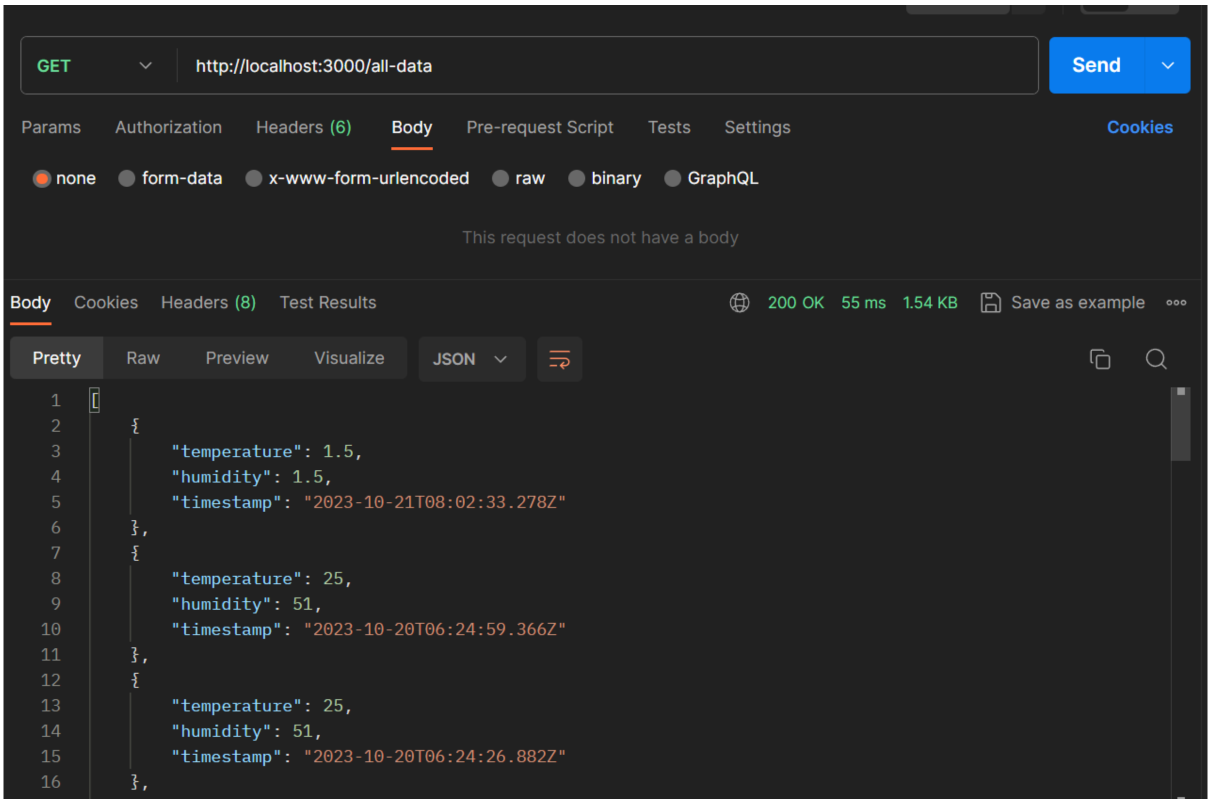
200 OK 97 ms 320 B

```
{
    "message": "Data received and inserted successfully"
}
```

You can repeat the steps for the GET request as well, to see all the data in the database.

Change the method to GET, set Body to ‘None’ and the URL.

*http://localhost:3000/all-data*


GET http://localhost:3000/all-data

Body: none

This request does not have a body

200 OK 55 ms 1.54 KB

```
[
    {
        "temperature": 1.5,
        "humidity": 1.5,
        "timestamp": "2023-10-21T08:02:33.278Z"
    },
    {
        "temperature": 25,
        "humidity": 51,
        "timestamp": "2023-10-20T06:24:59.366Z"
    },
    {
        "temperature": 25,
        "humidity": 51,
        "timestamp": "2023-10-20T06:24:26.882Z"
    },
```

## 7. Making the Localhost globally available on the internet

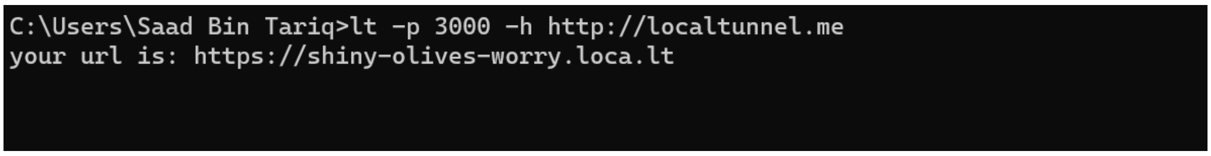
We will be using Localtunnel which allows you to easily share a web service on your local development machine without messing with DNS and firewall settings. Localtunnel will assign you a unique publicly accessible url that will proxy all requests to your locally running webserver.

You have already it installed this globally on machine when performing the Node exercise. For more information visit: https://theboroer.github.io/localtunnel-www/

Make sure that the Node server is running. Open Windows Command Prompt and run the following command:

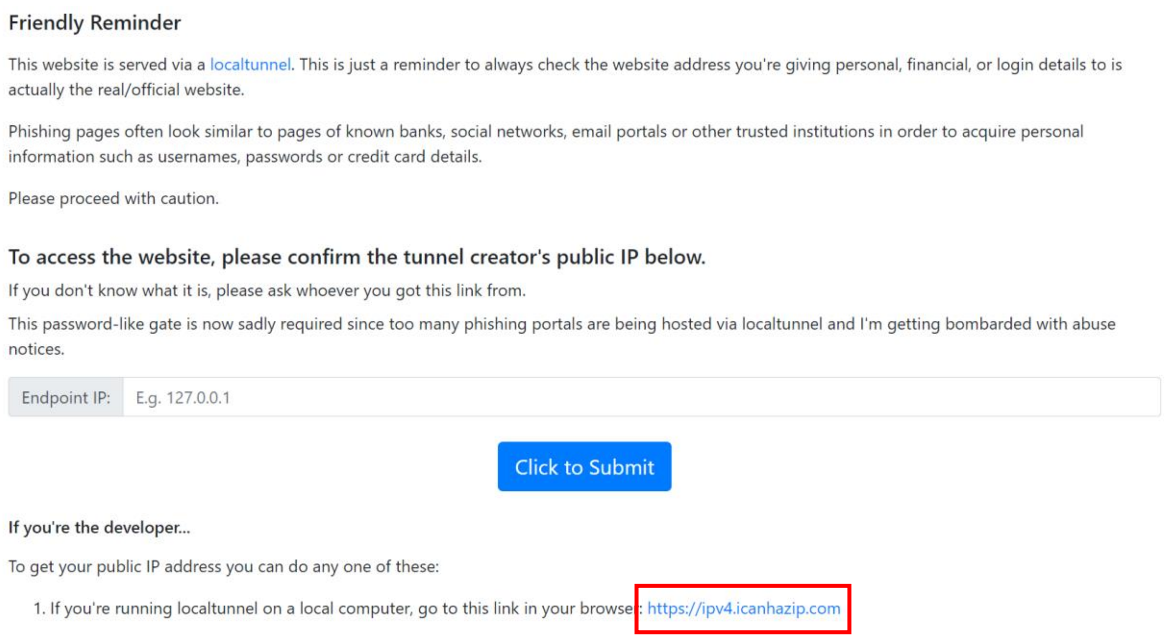
*lt -p 3000 -h http://localtunnel.me*

You will receive a url, for example https://flkajsfljas.loca.lt, that you can share with anyone for as long as your local instance of lt remains active. Any requests will be routed to your local service at the specified port.



```
C:\Users\Saad Bin Tariq>lt -p 3000 -h http://localtunnel.me
your url is: https://shiny-olives-worry.loca.lt
```

Clicking on the URL will open the following webpage, click on the link, copy the displayed IP address and then, click on the submit button.



Friendly Reminder

This website is served via a localtunnel. This is just a reminder to always check the website address you're giving personal, financial, or login details to is actually the real/official website.

Phishing pages often look similar to pages of known banks, social networks, email portals or other trusted institutions in order to acquire personal information such as usernames, passwords or credit card details.

Please proceed with caution.

To access the website, please confirm the tunnel creator's public IP below.

If you don't know what it is, please ask whoever you got this link from.

This password-like gate is now sadly required since too many phishing portals are being hosted via localtunnel and I'm getting bombarded with abuse notices.

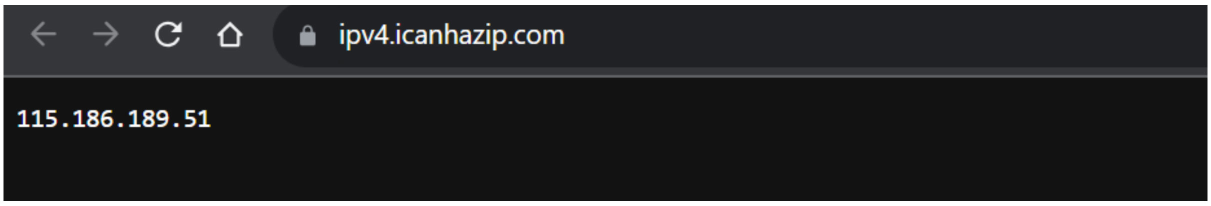
Endpoint IP: E.g. 127.0.0.1

If you're the developer...

To get your public IP address you can do any one of these:

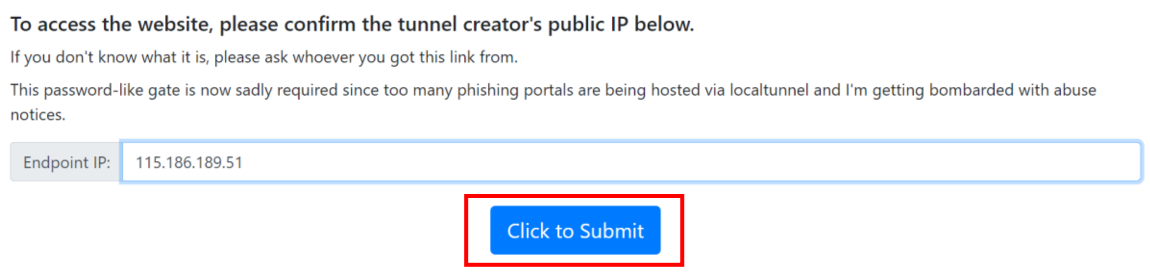
1. If you're running localtunnel on a local computer, go to this link in your browser: https://ipv4.icanhazip.com

Copy this IP address:



```
115.186.189.51
```

Paste and submit.

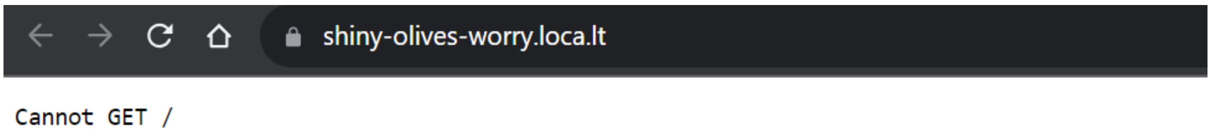

To access the website, please confirm the tunnel creator's public IP below.

If you don't know what it is, please ask whoever you got this link from.

This password-like gate is now sadly required since too many phishing portals are being hosted via localtunnel and I'm getting bombarded with abuse notices.

Endpoint IP: 115.186.189.51

If everything is working perfectly, you should get the following screen with the new tunnel URL:

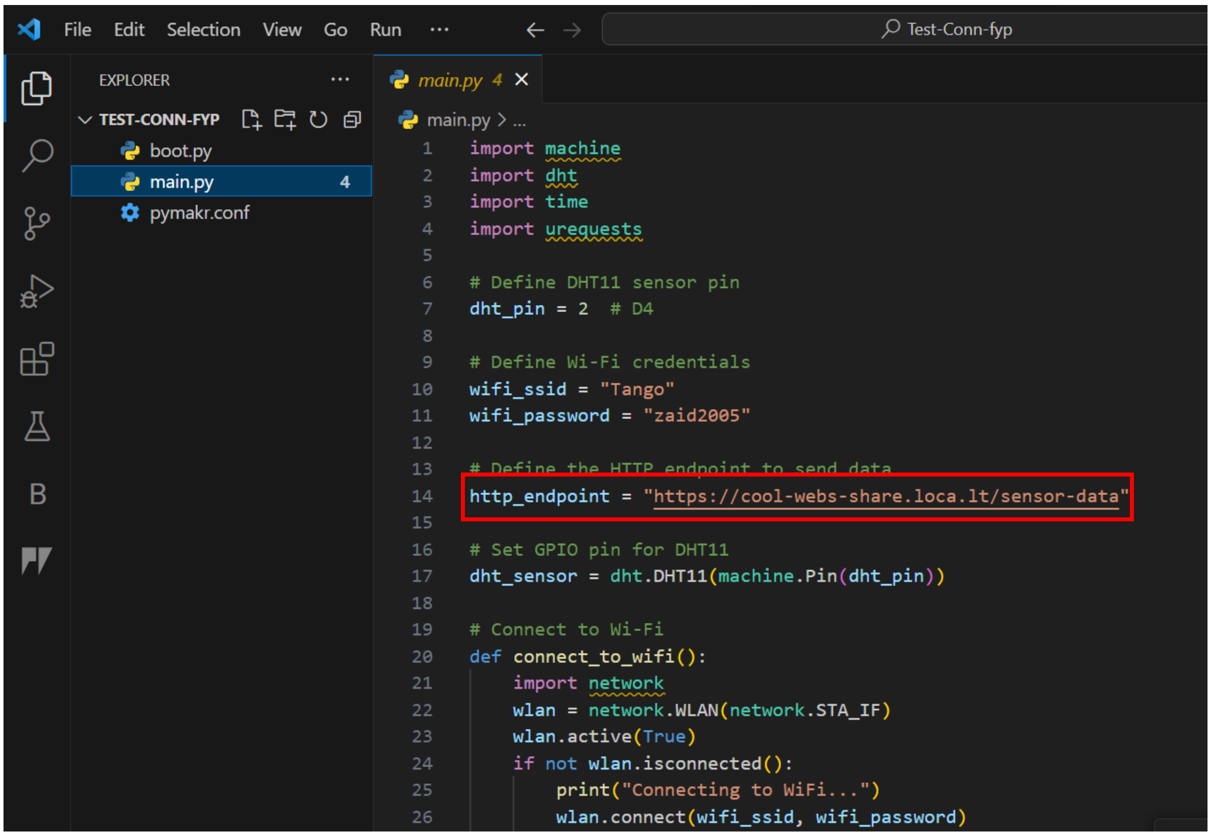


```
Cannot GET /
```

Note: You may have to repeat this step till you successfully get the following screen. From now our Node server is hosted on https://shiny-olives-worry.loca.lt/ and we will use this URL to make POST and GET requests. If on any point you get an error message that data can not be send, most of time you just have to regenerate this URL. A permanent solution is to host your Node Server on the internet by using hosting sites like Render etc.

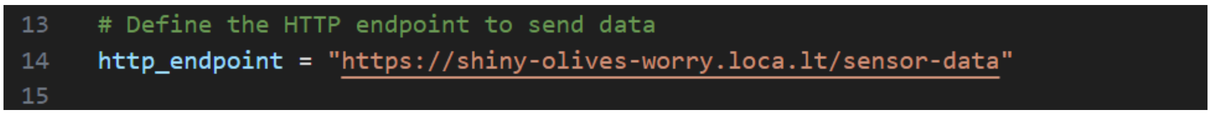
### 8. Replace the Hosted URL in Main.py file

Open that main.py file (PyMakr) and replace the URL.



```python
import machine
import dht
import time
import urequests

# Define DHT11 sensor pin
dht_pin = 2  # D4

# Define Wi-Fi credentials
wifi_ssid = "Tango"
wifi_password = "zaid2005"

# Define the HTTP endpoint to send data
http_endpoint = "https://cool-webs-share.loca.lt/sensor-data"

# Set GPIO pin for DHT11
dht_sensor = dht.DHT11(machine.Pin(dht_pin))

# Connect to Wi-Fi
def connect_to_wifi():
    import network
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print("Connecting to WiFi...")
        wlan.connect(wifi_ssid, wifi_password)
```

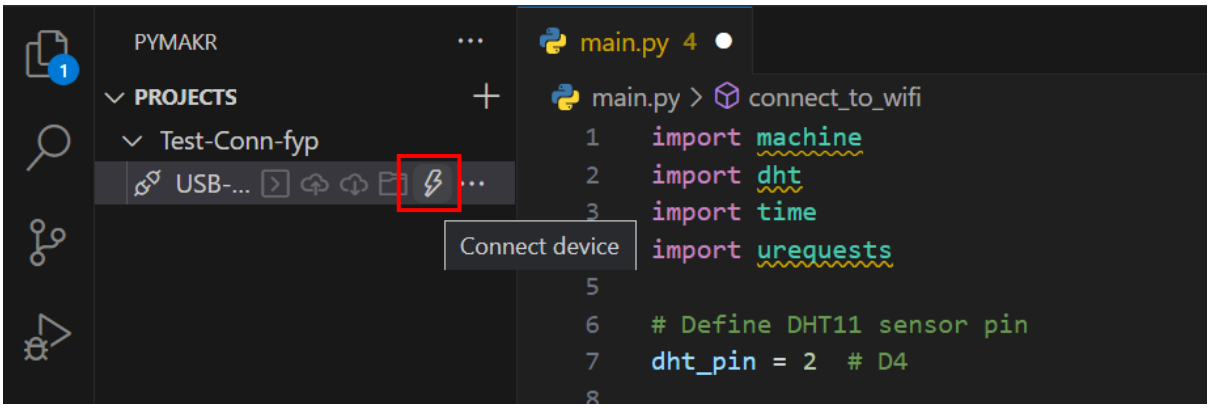
Make sure you keep the '/sensor-data' in the URL



```
# Define the HTTP endpoint to send data
http_endpoint = "https://shiny-olives-worry.loca.lt/sensor-data"
```

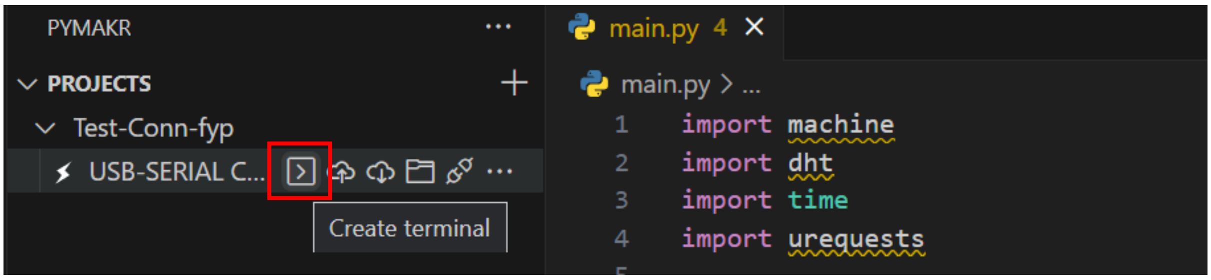
## 9. Uploading the Python code on ESP8266

Select the PyMakr tab in VS Code and connect the device (hover on it to get options).



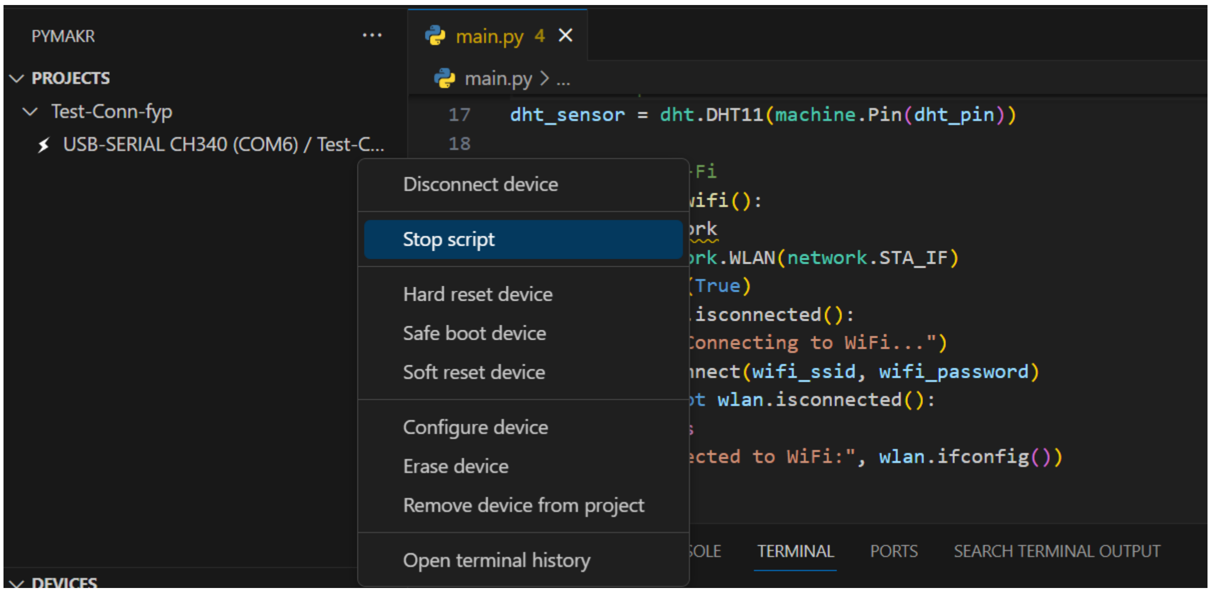
```
import machine
import dht
import time
import urequests

# Define DHT11 sensor pin
dht_pin = 2  # D4
```

Create the terminal to see the logs.



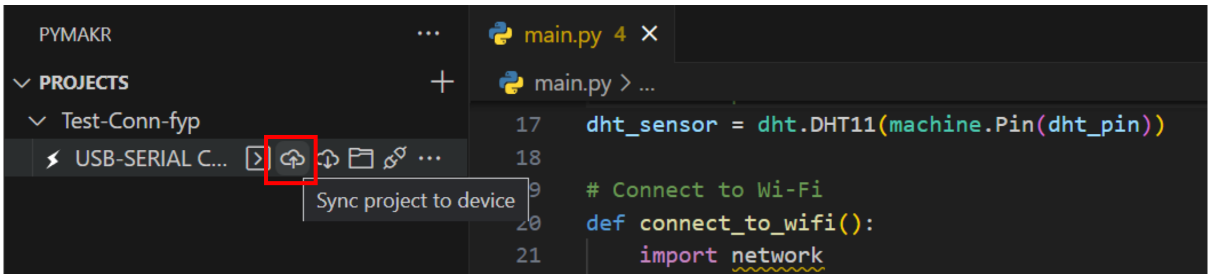
```
import machine
import dht
import time
import urequests
```

To upload the code, first the previously running scripts by clicking on the three dots.



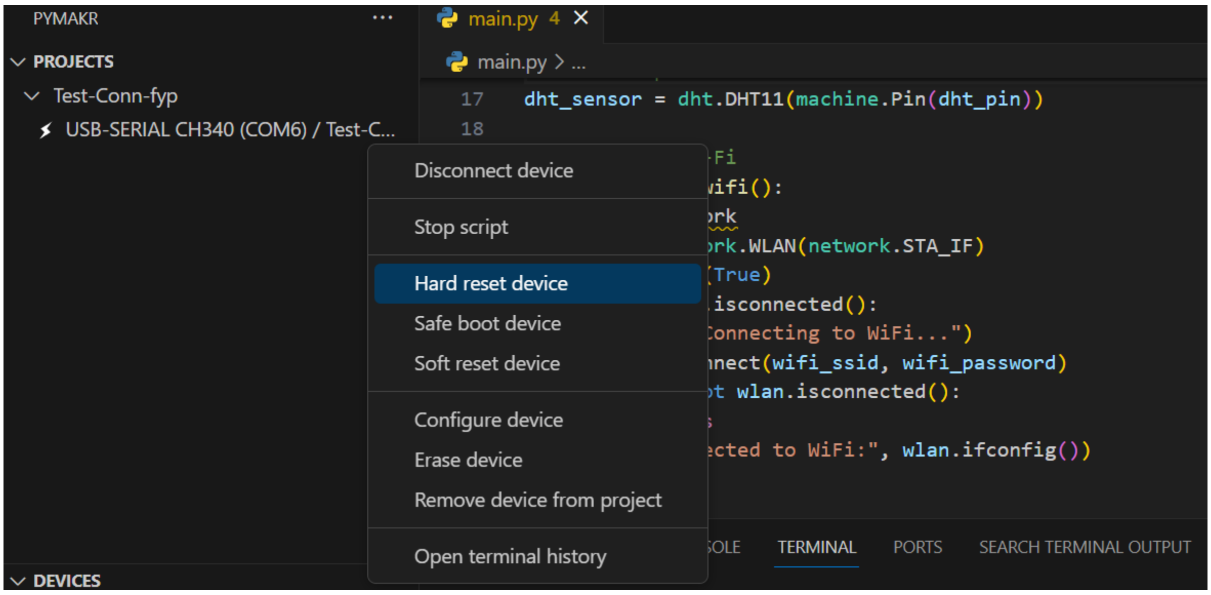
```
dht_sensor = dht.DHT11(machine.Pin(dht_pin))
```

Next, click on the sync button and the code will be uploaded. You will see the progress of the upload as a notification window.

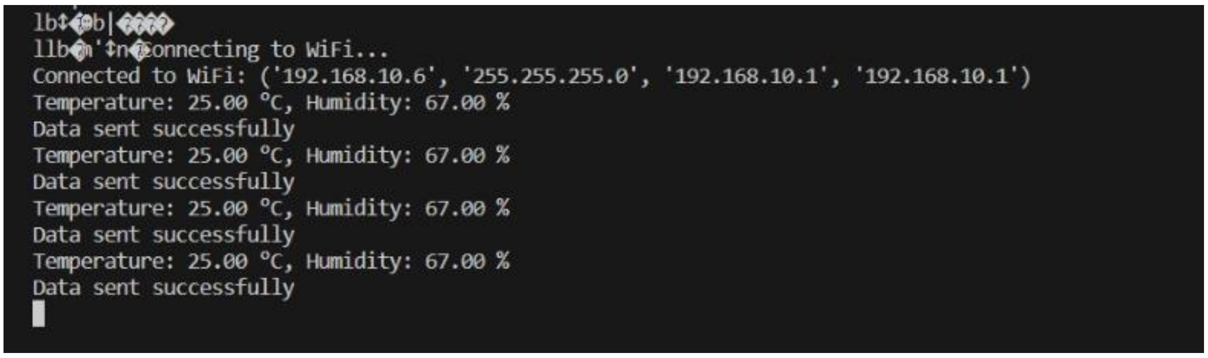


```
dht_sensor = dht.DHT11(machine.Pin(dht_pin))

# Connect to Wi-Fi
def connect_to_wifi():
    import network
```

The last step is to click on the three dots and hard reset the device.

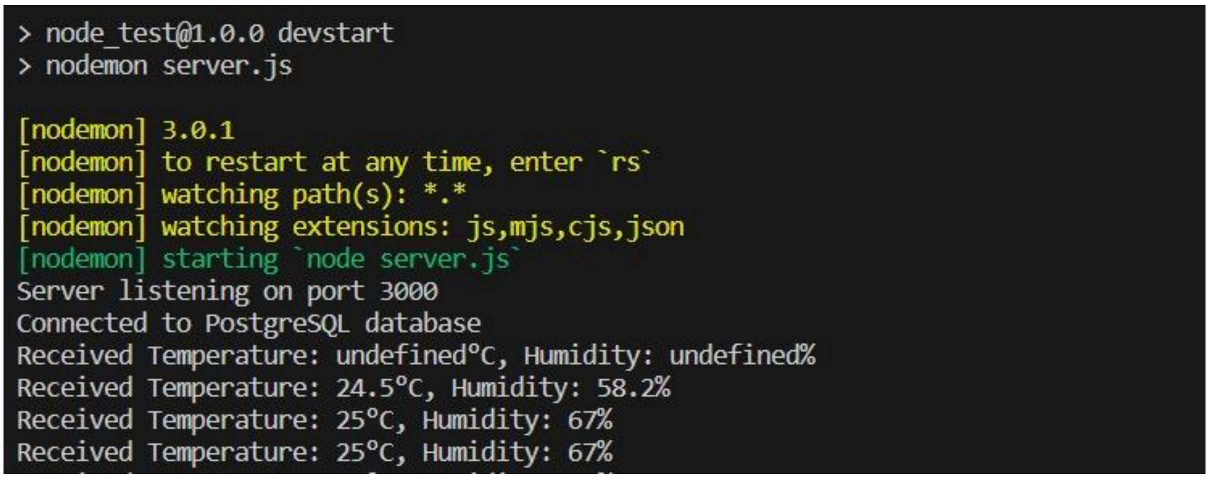


Now you will see in the terminal the results.



```
lb‡�b|����
llb�n'‡n�onnecting to WiFi...
Connected to WiFi: ('192.168.10.6', '255.255.255.0', '192.168.10.1', '192.168.10.1')
Temperature: 25.00 °C, Humidity: 67.00 %
Data sent successfully
Temperature: 25.00 °C, Humidity: 67.00 %
Data sent successfully
Temperature: 25.00 °C, Humidity: 67.00 %
Data sent successfully
Temperature: 25.00 °C, Humidity: 67.00 %
Data sent successfully
```

Simultaneously, if you want to see that data is successfully stored in the Database you can either Display rows in PgAdmin or see the console on the Node Server.



```
> node_test@1.0.0 devstart
> nodemon server.js

[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Server listening on port 3000
Connected to PostgreSQL database
Received Temperature: undefined°C, Humidity: undefined%
Received Temperature: 24.5°C, Humidity: 58.2%
Received Temperature: 25°C, Humidity: 67%
Received Temperature: 25°C, Humidity: 67%
```

## 10. Creating a React Front-end GUI to display the Data using Vite

React.js is an open-source JavaScript library for building user interfaces. It's maintained by Facebook and a community of developers. React simplifies UI development by allowing you to create reusable, component-based views that efficiently update in response to data changes. It's known for its declarative approach, virtual DOM for performance optimization, and strong ecosystem of tools and libraries. React is widely used in web and mobile app development and is a key technology in the creation of interactive and dynamic user interfaces. For more information visit: https://react.dev/

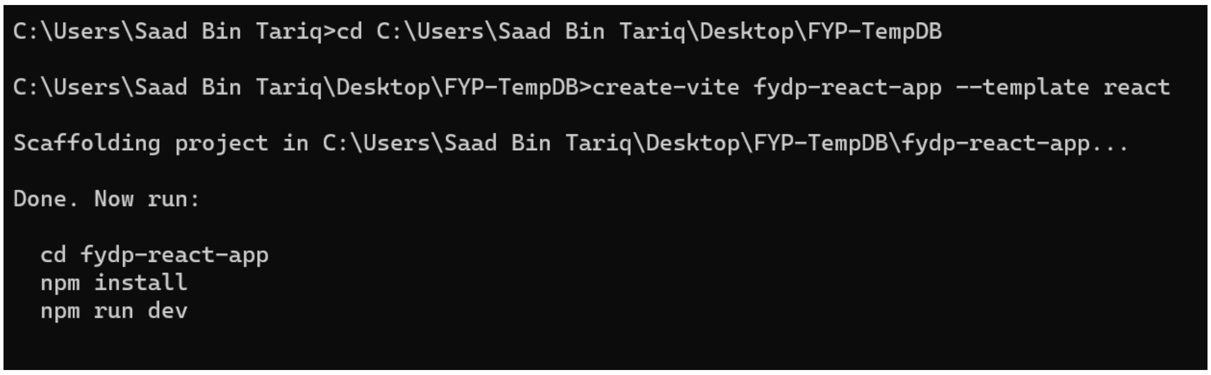
Creating a React front-end GUI using Vite involves setting up a modern development environment for building efficient and fast web applications. Vite is a build tool that offers near-instantaneous development server start and optimized production builds.

Open a new folder in VS Code where you want to save your React project. And in the PowerShell run:

*npm install -g create-vite*

Now open your Windows Command Prompt and navigate your folder, create the React project using the command:

*create-vite fydp-react-app --template react*



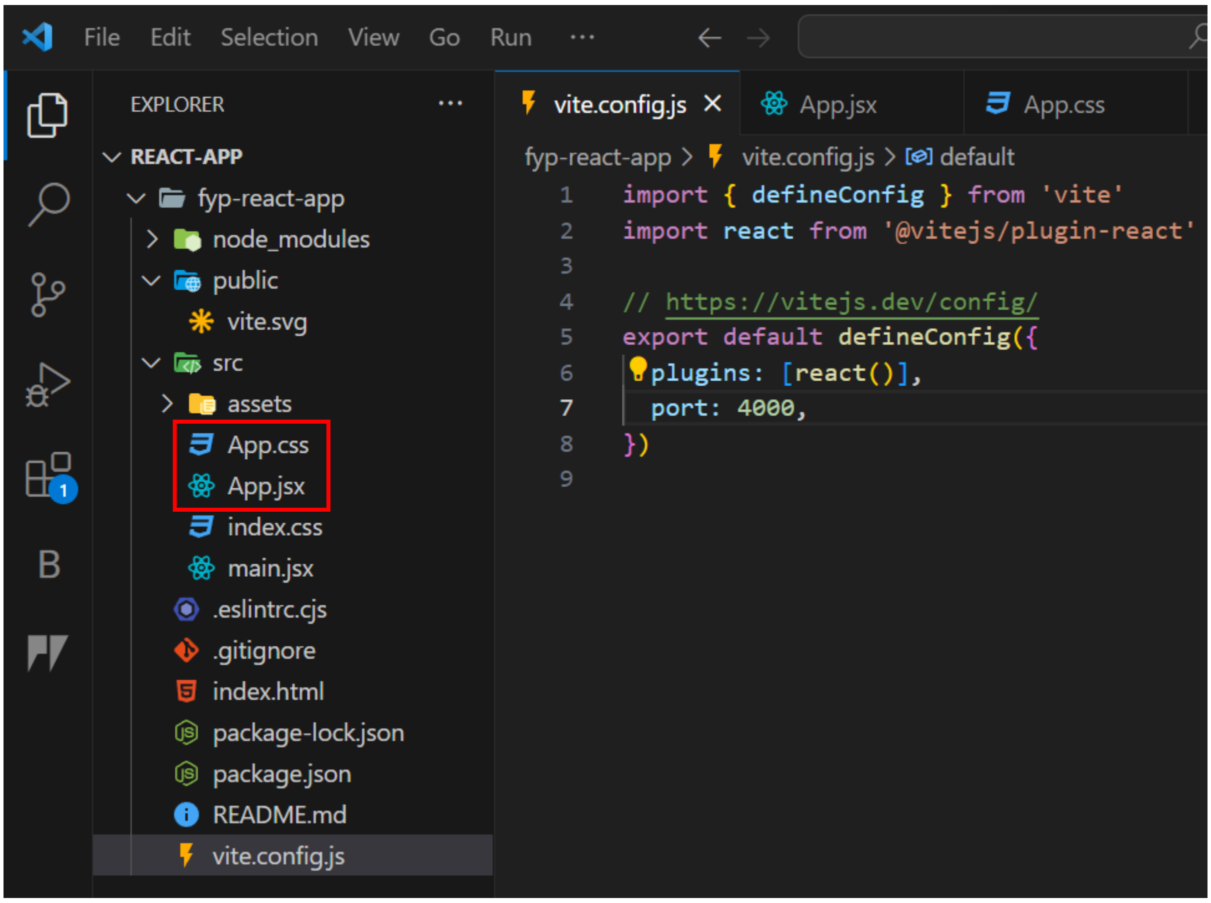
```
C:\Users\Saad Bin Tariq>cd C:\Users\Saad Bin Tariq\Desktop\FYP-TempDB

C:\Users\Saad Bin Tariq\Desktop\FYP-TempDB>create-vite fydp-react-app --template react

Scaffolding project in C:\Users\Saad Bin Tariq\Desktop\FYP-TempDB\fydp-react-app...

Done. Now run:

  cd fydp-react-app
  npm install
  npm run dev
```

Note: You can use any project name instead of ‘fydp-react-app’

Run the following commands one by one:

*cd fydp-react-app*
*npm install*

This will make you all the folders required to start the React app. You can further explores these folders.

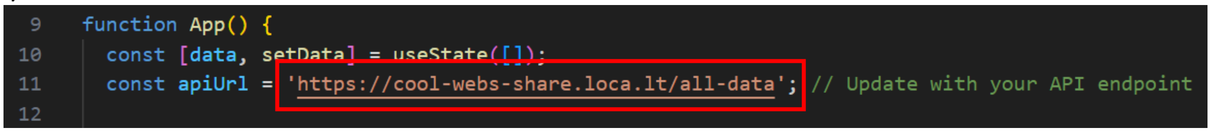


```javascript
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react()],
  port: 4000,
})
```

Open the App.jx file and paste the following code:

```javascript
import React, { useState, useEffect  } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'




function App() {
  const [data, setData] = useState([]);
  const apiUrl = 'https://cool-webs-share.loca.lt/all-data'; // Update with
your API endpoint

  useEffect(() => {
    // Function to fetch data from the API
    const fetchData = async () => {
      try {
        const response = await fetch(apiUrl);
        if (response.ok) {
          const result = await response.json();
          setData(result);
        }
      } catch (error) {
        console.error('Error fetching data:', error);
      }
    };
    // Call fetchData initially and every 10 seconds
    fetchData();
    const interval = setInterval(fetchData, 10000);

    // Cleanup the interval when the component unmounts
    return () => clearInterval(interval);
  }, []);

  return (
    <div className="container">
      <h1 className="heading">FYP Sensor Data</h1>
      <table className="data-table">
        <thead>
          <tr>
            <th>Temperature (°C)</th>
            <th>Humidity (%)</th>
            <th>Timestamp</th>
          </tr>
        </thead>
        <tbody>
          {data.map((item, index) => (
            <tr key={index}>
              <td>{item.temperature}°C</td>
              <td>{item.humidity}%</td>
              <td>{item.timestamp}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
}
export default App
```

Now replace the api_url with your tunnel URL you previously generated and keep the string '/all-data' at the end



```
function App() {
  const [data, setData] = useState([]);
  const apiUrl = 'https://cool-webs-share.loca.lt/all-data'; // Update with your API endpoint
```

Open App.css and paste the following code:

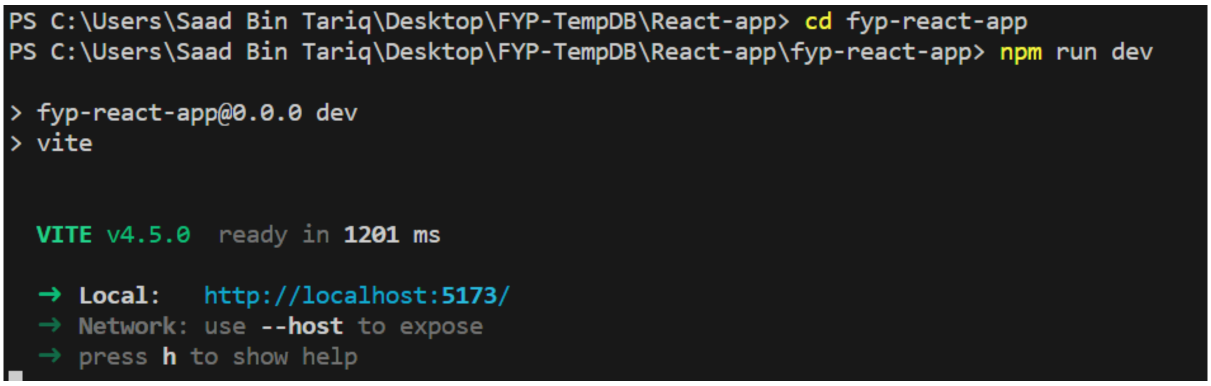
```
#root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  text-align: center;
}

.logo {
  height: 6em;
  padding: 1.5em;
  will-change: filter;
```

```css
  transition: filter 300ms;
}
.logo:hover {
  filter: drop-shadow(0 0 2em #646cffaa);
}
.logo.react:hover {
  filter: drop-shadow(0 0 2em #61dafbaa);
}

@keyframes logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

@media (prefers-reduced-motion: no-preference) {
  a:nth-of-type(2) .logo {
    animation: logo-spin infinite 20s linear;
  }
}

.card {
  padding: 2em;
}

.read-the-docs {
  color: #888;
}

/* App.css */
body {
  font-family: Arial, sans-serif;
  background-color: #f0f0f0;
  margin: 0;
  padding: 0;
}

.container {
  max-width: 800px;
  margin: 0 auto;
  padding: 20px;
  background-color: #fff;
  border-radius: 5px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.heading {
  font-size: 24px;
  font-weight: bold;
  margin-bottom: 20px;
  color: black; /* Text color */
  background-color: #f2f2f2; /* Background color */
  padding: 10px;
  border-radius: 5px;
}

.data-table {
  width: 100%;
  border-collapse: collapse;
  border: 1px solid #ddd;
  margin-top: 20px;
}

.data-table th,
.data-table td {
  border: 1px solid #ddd;
  padding: 10px;
  text-align: center;
  color: black; /* Text color */
  background-color: #f2f2f2; /* Background color */
}

.data-table th {
  font-weight: bold;
}

.data-table tr:nth-child(even) {
  background-color: #f2f2f2;
}

.data-table tr:hover {
  background-color: #ddd;
}
```

To start the application run the following command in the terminal

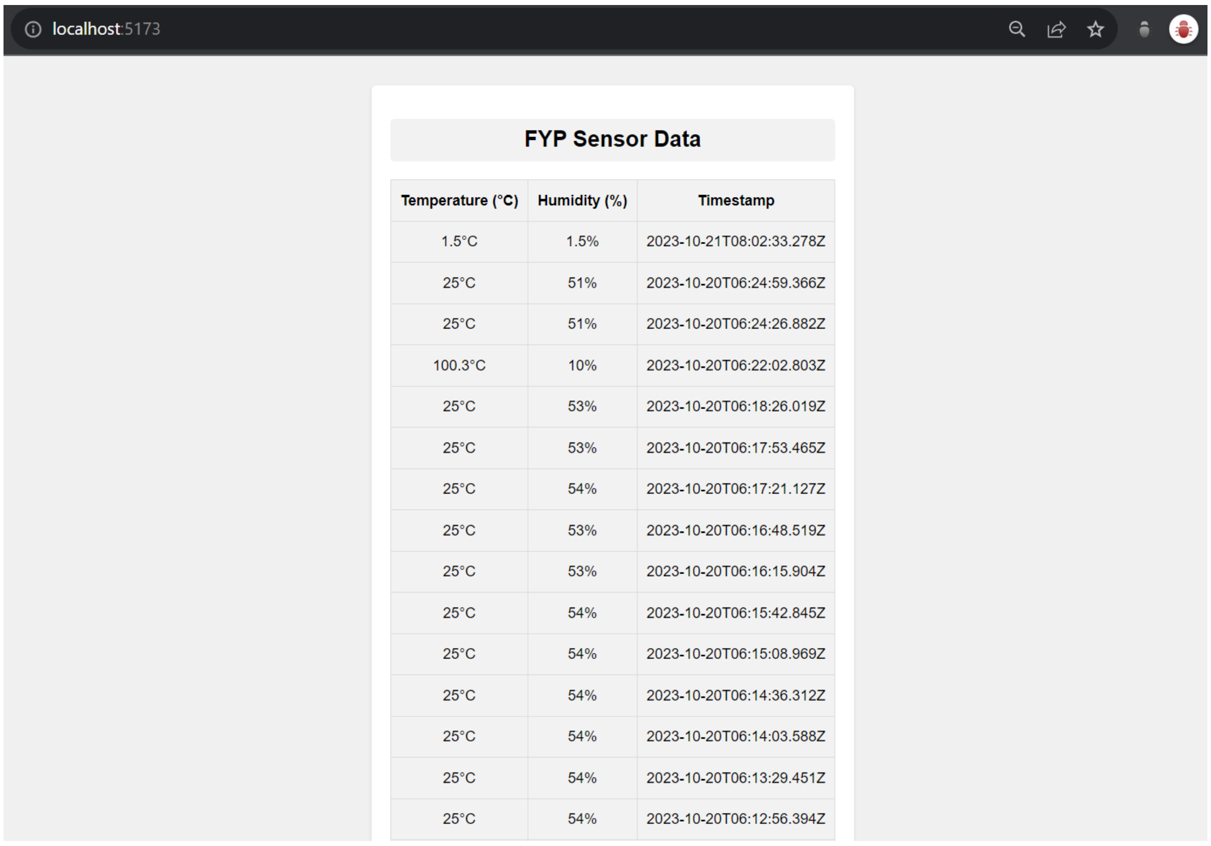
*cd your-project-name*

npm run dev



```
PS C:\Users\Saad Bin Tariq\Desktop\FYP-TempDB\React-app> cd fyp-react-app
PS C:\Users\Saad Bin Tariq\Desktop\FYP-TempDB\React-app\fyp-react-app> npm run dev

> fyp-react-app@0.0.0 dev
> vite


  VITE v4.5.0  ready in 1201 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h to show help
```

Paste http://localhost:5173/ in google chrome.



FYP Sensor Data

| Temperature (°C) | Humidity (%) | Timestamp |
|---|---|---|
| 1.5°C | 1.5% | 2023-10-21T08:02:33.278Z |
| 25°C | 51% | 2023-10-20T06:24:59.366Z |
| 25°C | 51% | 2023-10-20T06:24:26.882Z |
| 100.3°C | 10% | 2023-10-20T06:22:02.803Z |
| 25°C | 53% | 2023-10-20T06:18:26.019Z |
| 25°C | 53% | 2023-10-20T06:17:53.465Z |
| 25°C | 54% | 2023-10-20T06:17:21.127Z |
| 25°C | 53% | 2023-10-20T06:16:48.519Z |
| 25°C | 53% | 2023-10-20T06:16:15.904Z |
| 25°C | 54% | 2023-10-20T06:15:42.845Z |
| 25°C | 54% | 2023-10-20T06:15:08.969Z |
| 25°C | 54% | 2023-10-20T06:14:36.312Z |
| 25°C | 54% | 2023-10-20T06:14:03.588Z |
| 25°C | 54% | 2023-10-20T06:13:29.451Z |
| 25°C | 54% | 2023-10-20T06:12:56.394Z |

This is the final output and the data on this webpage in updated in real time. You always can add more components in the React App to make it more appealing and attractive.

In conclusion, the project to connect an ESP8266 flashed with MicroPython to a PostgreSQL database and visualize temperature and humidity data in a React frontend. By seamlessly integrating IoT (Internet of Things) capabilities with a robust database system, this project can be utilized for real-time environmental monitoring. It finds applications in weather stations, smart agriculture, and indoor climate control systems, providing valuable data for decision-making and automation. Additionally, it can serve as a foundation for developing custom IoT solutions, offering endless possibilities for data-driven innovations in various domains, from home automation to industrial processes. Overall, this project demonstrates the power of combining hardware, software, and data visualization to create functional and impactful solutions.