Introduction

The Raspberry Pi is a versatile and powerful single-board computer that can be used for a wide range of projects, including building a weather station. One of the key components of a weather station is the sensors used to measure weather parameters such as temperature, humidity, and air pressure.

In this project, we will be using two sensors, the DHT11 and BMP180, to build a weather station using a Raspberry Pi. The DHT11 sensor is a basic digital temperature and humidity sensor that can be easily connected to the Raspberry Pi and used to measure the temperature and humidity of the surrounding environment. The BMP180 sensor is a high-precision barometric pressure sensor that can be used to measure the air pressure, which can then be used to calculate the altitude and weather trends.

By combining these sensors with a Raspberry Pi, we can collect accurate and reliable weather data that can be analyzed and displayed in a variety of ways. We will explore how to connect the sensors to the Raspberry Pi, how to program the Raspberry Pi to collect and log weather data, and how to display the data using various visualization tools.

Through this project, we will gain a better understanding of how to build a weather station using Raspberry Pi, as well as how to use the Python programming language to collect, analyze, and visualize weather data.

Chapters

Chapter 1: Understanding the DHT11 Sensor (ALIYU)
Digital temperature and humidity sensors like the DHT11 are frequently utilized in do-it-yourself applications. It is a reasonably accurate, low-cost sensor that can measure temperature and humidity. We will go through the DHT11 sensor's operation and features in this chapter.

Chapter 2: Understanding the Barometric Sensor (**ALIYU)**
A pressure sensor that can gauge atmospheric pressure is the barometer sensor. It is frequently employed in weather stations and other applications that monitor the environment. We will go through the barometer sensor's operation and features in this chapter.

Chapter 3: Setting up the Raspberry Pi (EMMANUEL)
We will go over how to configure the Raspberry Pi for this project in this chapter. We will go over the necessary hardware and software requirements and give detailed setup instructions for the Raspberry Pi.

Chapter 4: Connecting the Sensors (EMMANUEL)
We will go over how to connect the DHT11 and barometer sensors to the Raspberry Pi in this chapter. The pin connections and wiring needed for each sensor will be described in great detail.

Chapter 5: Raspberry Pi programming(IMMANUEL)
We will go over programming the Raspberry Pi to read data from the sensors in this chapter. To read the sensor data and save it in a database, we will develop a script using the Python programming language.

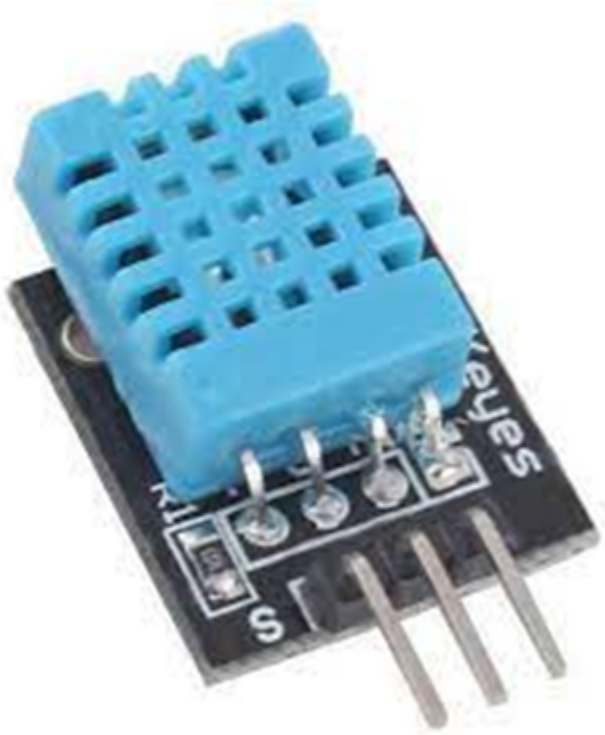
Chapter 6: Storing the Sensor Data(IMMANUEL)
We will go over how to save the sensor data in a database in this chapter. The sensor data will be stored using Firebase.

Chapter 7: Visualizing the Data(IMMANUEL)
In this chapter, we'll talk about using a web-based interface to visualize sensor data. We will develop a web application that shows the sensor data in real-time using Firebase.

Chapter 8: Final Verdict(JEFFREY)
We will provide a summary of the project and its results in this chapter.

Chapter 1: Understanding the DHT11 Sensor

DHT11

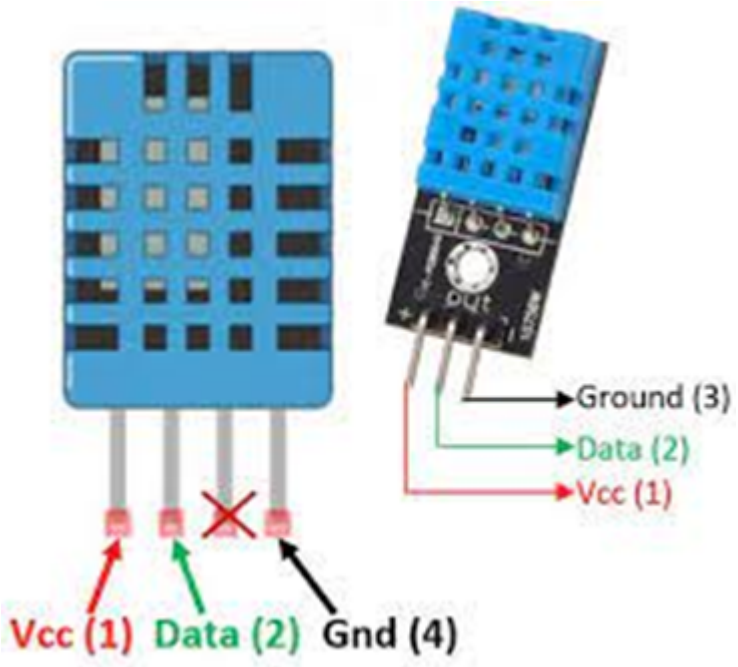


DHT11 is a commonly used low-cost digital sensor used for sensing temperature and humidity, it is integrated with a dedicated NTC chip to measure temperature and an 8-bit microcontroller to show the results of the humidity and temperature as serialized data.

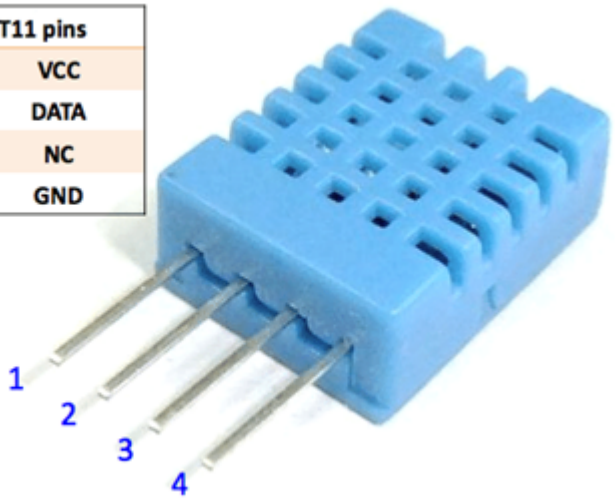
This sensor can be interfaced easily with any micro-controller such as Raspberry and used in applications like weather stations and smart homes.

It comes with a thermistor and a capacitive humidity sensing element for sensing temperature, which can measure temperature in the range of 0-80°C and humidity in the range of 20-70% RH with high accuracy.

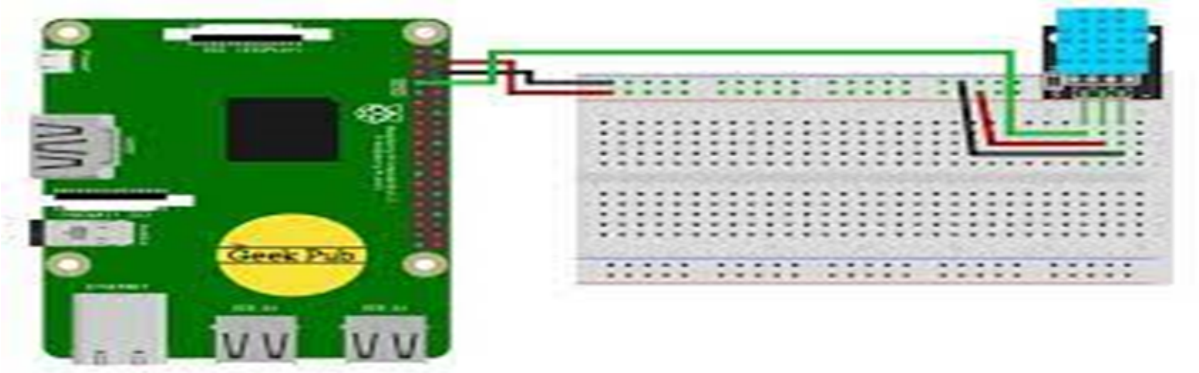
The four pins which the DHT11 sensor has are: VCC, Data, NC (no connection) and Ground, the VCC pin is connected to a 3.5V to 5.5V power supply, and the GND pin is connected to the ground of a circuit. Communication with the microcontroller is made using a single wire



DHT11 pins	
1	VCC
2	DATA
3	NC
4	GND



The signal for data is a pulse width modulated (PWM) signal, where a pulse between 26 to 28us represents a logic one, and a high pulse of 70us represents a logic zero. The sensor is also equipped with a pull-up resistor between the VCC and data pins.



To link the DTH11 to an interface, the sensor's VCC pin was connected to a 5.5V power supply, its GND pin to the ground, and its data pin to any digital pin on the microcontroller to interface the DHT11 sensor with it. The input/output (I/O) pin of the microcontroller was linked to the data pin of the sensor. The microcontroller then sends a start signal, which receives the data, and checks the checksum to be able to read the data from the DHT11 sensor.

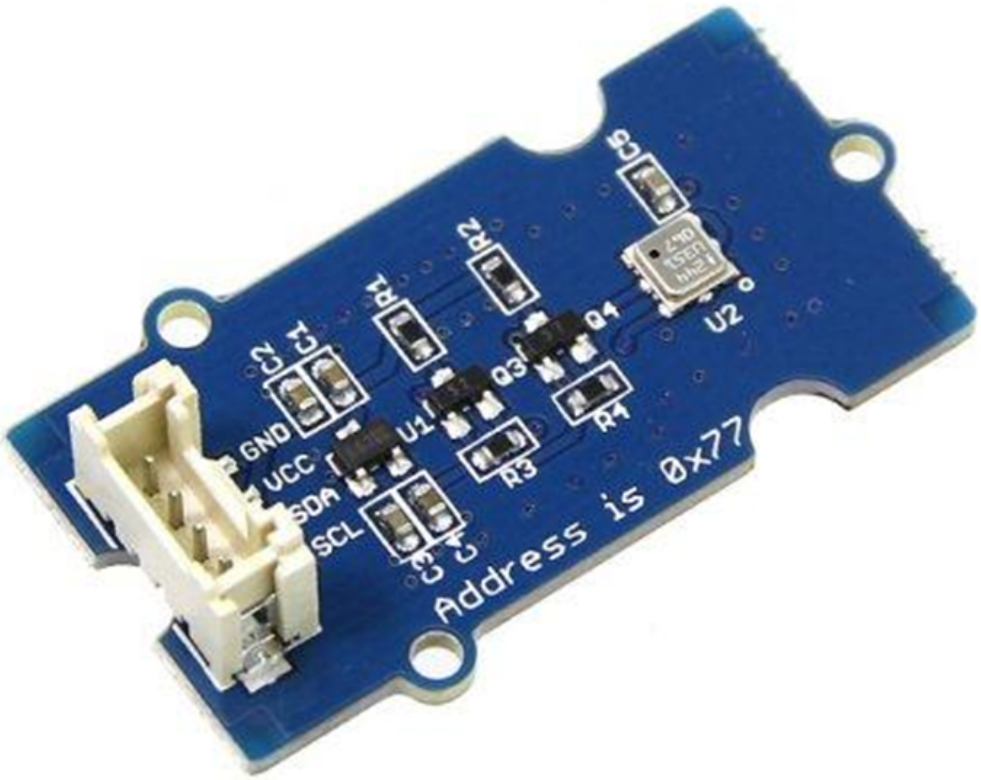
Features of DTH11:

- Cheap price.
- 3.5 to 5.5 volts power.
- 2.5mA max current use during conversion (while requesting data).
- Suitable for readings of 20-80% humidity with 5% accuracy.
- Suitable for readings of 0-50°C temperature ±2°C accuracy.
- No more than 1 Hz sampling rate (once per second).
- Body size is 15.5mm x 12mm x 5.5mm.
- Has 4 pins with about 0.1" spacing.

The DHT11 sensor can be affected by various environmental factors such as temperature, humidity, and electromagnetic interference. The sensor may also produce wrong data due to wrong wiring or timing problems. To troubleshoot the sensor, check the wiring connections, verify the timing, and ensure that the data is properly formatted and used.

Chapter 2: Understanding the Barometric Sensor

BMP180



The BMP180 sensor is a digital barometer with excellent accuracy and minimal power that can gauge temperature and atmospheric pressure. This sensor uses piezo-resistive technology, which enables it to precisely measure variations in atmospheric pressure. With an accuracy of up to 0.03 hPa, its measuring range is 300 to 1100 hPa. The BMP180 sensor is commonly used in indoor navigation systems, height measuring, and weather forecasting. It is frequently used to establish altitude and guarantee stable flight in drones and other remote-controlled vehicles.

Listed below are a few characteristics of the BMP180 sensor:

1. High accuracy: The BMP180 sensor, which measures atmospheric pressure, has an accuracy of up to 0.03 hPa and a measurement range of 300 to 1100 hPa.

2. The sensor uses extremely little power and is built to work at low voltages, making it the perfect option for battery-operated devices.

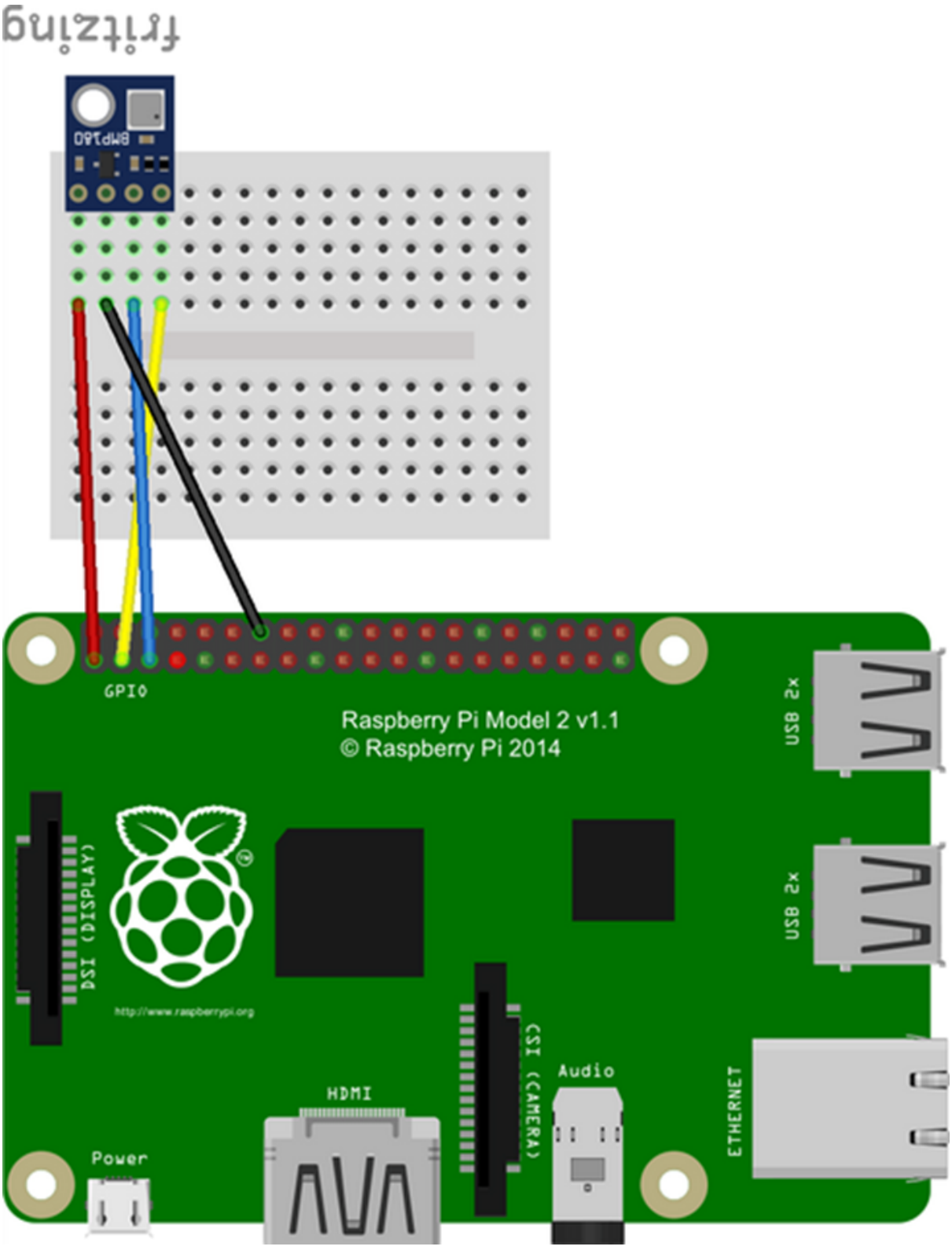
3. Temperature compensation: The BMP180 sensor can account for temperature fluctuations, allowing it to measure pressure accurately even in surroundings with a range of temperatures.

4. Compact size: The sensor's light weight and tiny size make it simple to integrate into a range of applications, including portable devices.

5. Digital interface: The BMP180 sensor connects to other electronic devices and microcontrollers by means of a digital interface.

6. Flexibility: The BMP180 sensor has a wide range of uses, including interior navigation systems, altitude monitoring, and weather forecasting.

7. Affordable: The BMP180 sensor is an alternative for monitoring pressure and temperature that is reasonably priced, making it available to professionals, students, and hobbyists alike.


fritzing
BMP180
GPIO
Raspberry Pi Model 2 v1.1
© Raspberry Pi 2014
USB 2x
USB 2x
DSI (DISPLAY)
http://www.raspberrypi.org
CSI (CAMERA)
Audio
ETHERNET
HDMI
Power

Chapter 3: Setting up the Raspberry Pi

We will go over how to configure the Raspberry Pi for this project in this chapter. We will go over the necessary hardware and software requirements and give detailed setup instructions for the Raspberry Pi.

Hardware requirements

- Ethernet cable
- GPIO connector
- GPIO breadboard
- Computer / Monitor
- Power Up cable
- A Secure Digital Card (SD card)
- Mouse
- Keyboard

Software requirements

- VNC viewer

Components of a Raspberry Pi:

The Raspberry Pi consists of several components, including:

1. CPU: The Raspberry Pi's CPU serves as the system's brain, processing data and commands while directing the other parts of the system.

2. RAM: The Raspberry Pi uses its RAM (Random Access Memory) to temporarily store data and instructions while it processes them.

3. GPIO pins: The GPIO pins are used to link a variety of peripherals and parts, including LEDs, motors, and sensors. Through programming, users can interact and control these components using the GPIO pins.

4. USB ports: Users can connect external devices, such as keyboards, mouse, and storage devices, using the USB ports.

5. Ethernet port: Users can connect the Raspberry Pi to a network and use the Ethernet connection to access the internet.

6. HDMI port: Users can attach the Raspberry Pi to a monitor or TV via the HDMI port to view the system's output.

Connecting a Raspberry Pi:

Connecting a Raspberry Pi has a few steps to it be is generally a straight forward process . Here are the steps:

1. We will need the necessary equipments put together for this project to be able to work: First, we will need a Raspberry Pi board, a microSD card which holds the installed operating system, which for us in this case we had to install Raspberry Pi Os using Raspberry Pi Imager**.**

   A power supply, a keyboard, a mouse, a computer or monitor, and an ethernet cable.

2. Insert the microSD card: Insert the microSD card into the Raspberry Pi's microSD card slot so that the software can be accessed and be used to edit and run the code needed for the project.

3. Connect the peripherals: Connect the keyboard, mouse, and monitor or TV to the Raspberry Pi's USB and HDMI ports, respectively.

4. Connect the power supply: Connect the power supply to the Raspberry Pi's power port .

5. Power on the Raspberry Pi: Power on the Raspberry Pi by plugging in the power supply and make sure that the switch is on for the raspberry Pi address to be accessed from the VNC viewer.

6. Configure the operating system by following the on-screen prompts to configure the operating system settings, such as language and Wi-Fi.

7. You can start using the Raspberry Pi once the configuration is complete, you can start using the Raspberry Pi. You can use the command line interface or graphical user interface to interact with the system, install software, and start projects.

Chapter 4: Connecting the Sensors

We will go over how to connect the DHT11 and barometer sensors to the Raspberry Pi in this chapter. The pin connections and wiring needed for each sensor will be described in great detail.

THE DHT11 Sensor

A well-liked digital temperature and humidity sensor is the DHT11 model. A microcontroller or other devices can be interfaced with using just a few connectors. Total pin count for the sensor is four: VCC, GND, DATA, and NC (No Connection).

VCC (Power): The power source for the sensor is delivered by this pin. It needs to be connected to a 5V power supply that is regulated. Make that the power source can deliver enough current for the sensor to function.

GND (Ground): Join this pin to your system's ground (GND). It acts as the sensor's reference voltage.

DATA (Data): The microcontroller and sensor communicate in both directions using the DATA pin. To ensure proper signal communication, a pull-up resistor is needed on this pin. Between the sensor's DATA pin and VCC pin, attach a 10k resistor.

As this pin is not in use, it can be left disconnected (NC).

These steps should be followed to connect the DHT11 sensor to a microcontroller:

Connect your GPIO breadboard 5V power source to the DHT11's VCC pin.

Connect your GPIO breadboard ground (GND) to the DHT11's GND pin.

Connect your GPIO breadboard digital input/output (I/O) pin to the DHT11's DATA pin.

Put a 10k pull-up resistor between the DHT11's VCC pin and DATA pin.

Note: Depending on the microcontroller or development board you are using, the pin configurations and numbers may change. For exact pin assignments and specifications, always refer to your microcontroller's datasheet or documentation.

By implementing the proper communication protocol in your microcontroller code, you can now read temperature and humidity data from the DHT11 sensor after these connections have been made.

To achieve precise and trustworthy readings, make sure to manage the sensor's data gathering and communication timing requirements as outlined in the DHT11 datasheet.

The pin connections required to connect to the DHT11 sensor are listed above. Please refer to the manufacturer's datasheet or reference materials if you need more specific information.

THE BMP180 Sensor

A typical digital barometric pressure and temperature sensor used in many applications is the BMP180. Instead of the usual six pins, some sensors only have four. The four-pin version of the BMP180 has the following pinout: VCC, GND, SDA, and SCL.

VCC (Power): The power source for the sensor is delivered by this pin. Join it to a 3.3V power supply that is regulated. Verify that the power source can deliver enough current for the sensor to function. Don't go beyond the voltage limit listed in the sensor's datasheet, please.

GND (Ground): Join this pin to your system's ground (GND). It acts as the sensor's reference voltage.

SDA (Serial Data): The I2C (Inter-Integrated Circuit) protocol uses the SDA pin for bidirectional communication. Connect it to your microcontroller's or I2C bus's matching SDA pin.

SCL (Serial Clock): In the I2C protocol, clock synchronization is accomplished using the SCL pin. Connect it to the I2C bus or microcontroller's matching SCL pin.

The methods below describe how to use I2C to connect a microcontroller and the four-pin BMP180 sensor:

Connect your microcontroller's 3.3V power supply to the BMP180's VCC pin.

Connect your microcontroller's ground (GND) pin to the GND pin on the BMP180.

Connect the BMP180's SDA pin to the I2C bus or microcontroller's corresponding SDA pin.

Connect the BMP180's SCL pin to the I2C bus or microcontroller's corresponding SCL pin.

Please take note that depending on the particular microcontroller or development board you are using, the pin names (SDA, SCL) and configurations may change. For exact pin assignments and specifications, always refer to your microcontroller's datasheet or documentation.

Once the connections are made, you can use your microcontroller to implement the necessary code to communicate with the BMP180 sensor using the I2C protocol.

For accurate readings and dependable operation, make sure to adhere to the timing and communication requirements listed in the BMP180 datasheet.

The pin connections required to connect to the four-pin BMP180 sensor are listed above in their entirety. Please refer to the manufacturer's datasheet or reference materials if you need more specific information.

Image of the DHT11 sensor

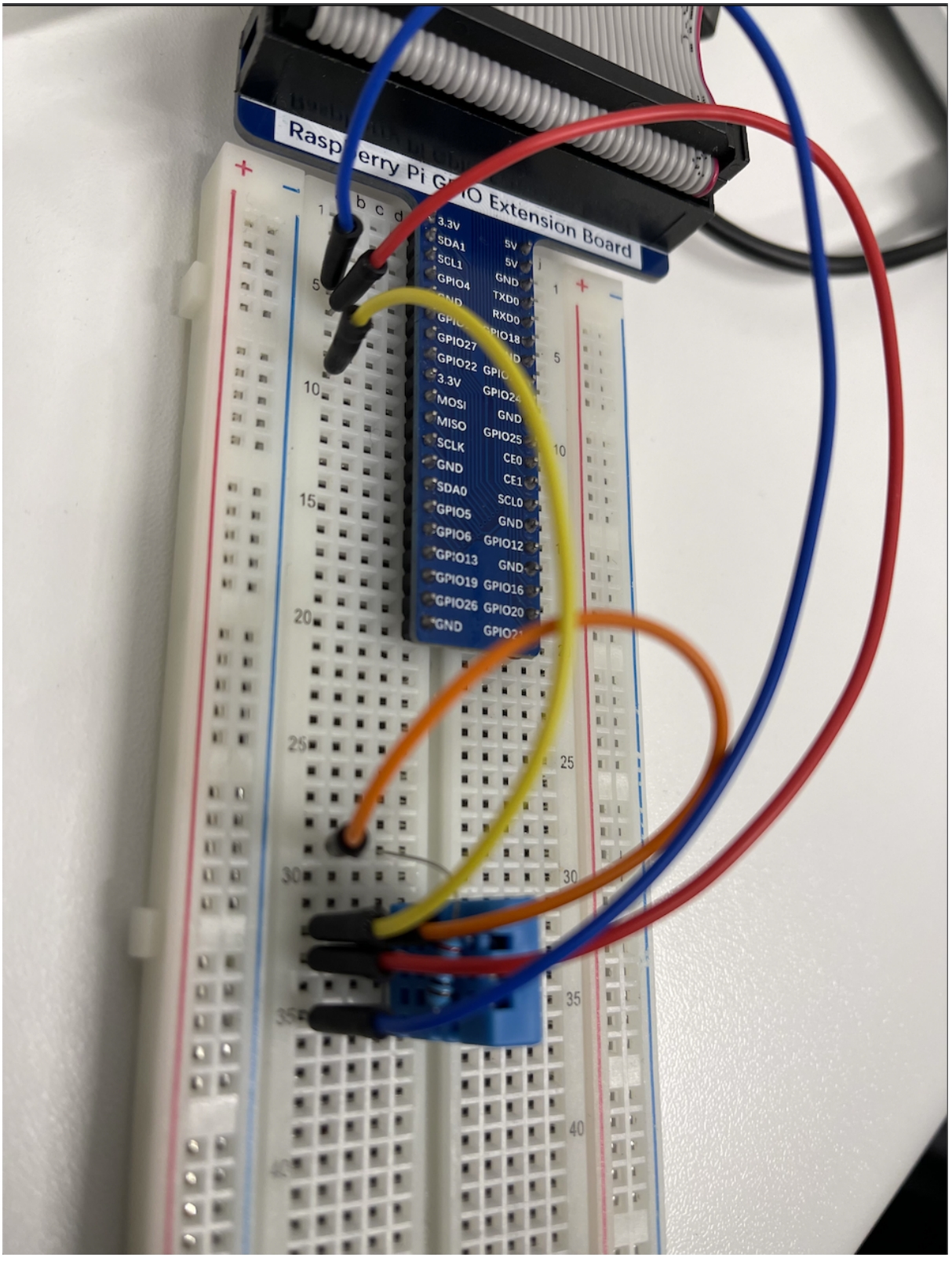
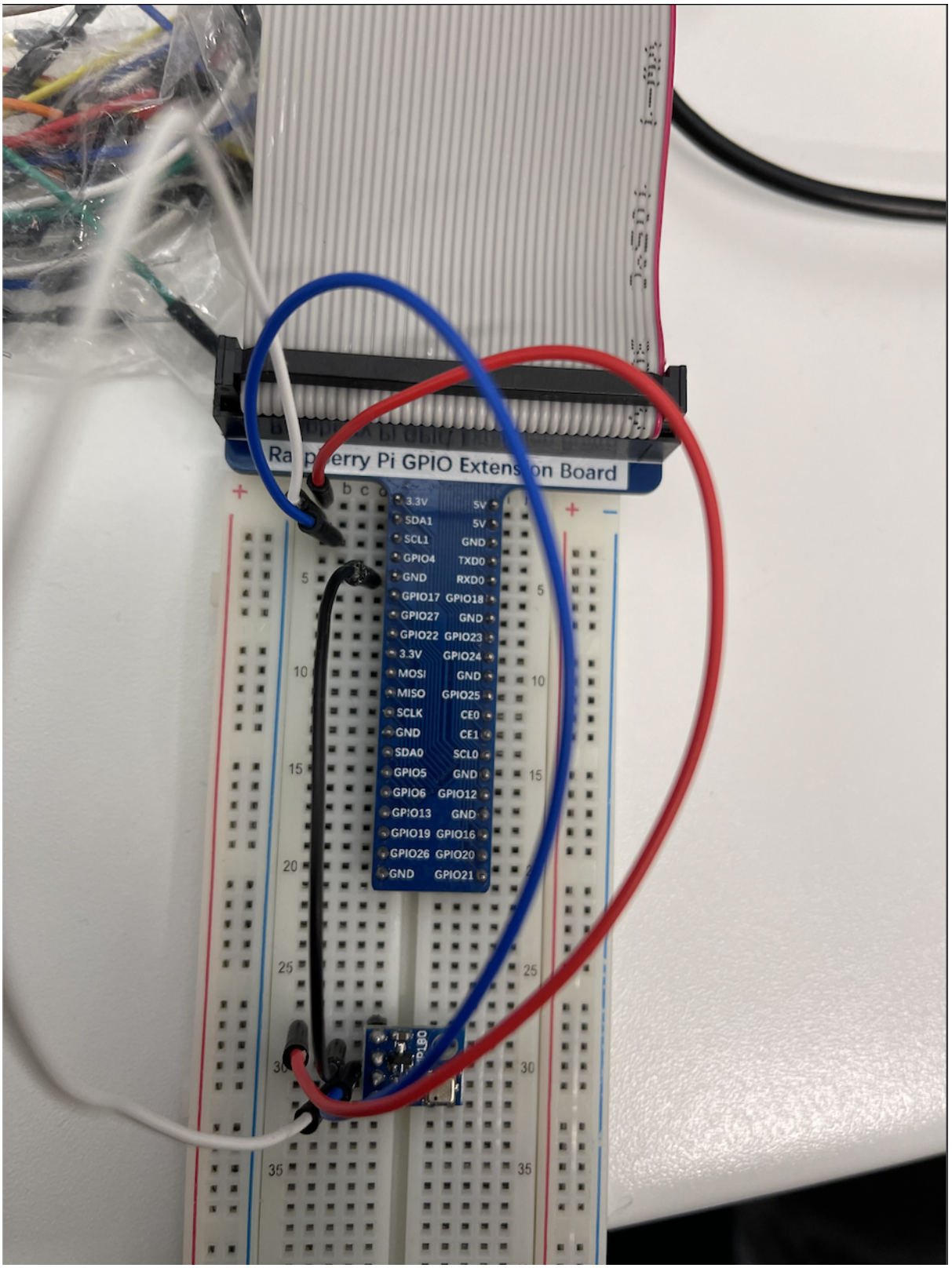


Image of BMP180 sensor



Chapter 5: Raspberry Pi programming

```
# Weather_Station.py

import RPi.GPIO as GPIO
import time
import Project_Testing as DHT
import smbus
import time
from ctypes import c_short
from time import sleep
from pyrebase import pyrebase
import base64
from datetime import datetime


DHTPin = 11     #define the pin of DHT11
DEVICE = 0x77 # Default device I2C address
GPIO.setwarnings(False)

#bus = smbus.SMBus(0)  # Rev 1 Pi uses 0
bus = smbus.SMBus(1) # Rev 2 Pi uses 1

config = {
  "apiKey": "AIzaSyBML8cxQ8MJN0Zvz4s_n-de8heivfPOqfw",
  "authDomain": "group-project-4b008.firebaseapp.com",
  "databaseURL":
"https://group-project-4b008-default-rtdb.firebaseio.com/",
  "storageBucket": "group-project-4b008.appspot.com"
}
firebase = pyrebase.initialize_app(config)
auth = firebase.auth()
user =
auth.sign_in_with_email_and_password("immanuelsy2112@gmail.com",
"123456")
db = firebase.database()



def convertToString(data):
  # Simple function to convert binary data into
```

```
  # a string
  return str((data[1] + (256 * data[0])) / 1.2)

def getShort(data, index):
  # return two bytes from data as a signed 16-bit value
  return c_short((data[index] << 8) + data[index + 1]).value

def getUshort(data, index):
  # return two bytes from data as an unsigned 16-bit value
  return (data[index] << 8) + data[index + 1]

def readBmp180Id(addr=DEVICE):
  # Chip ID Register Address
  REG_ID     = 0xD0
  (chip_id, chip_version) = bus.read_i2c_block_data(addr, REG_ID,
2)
  return (chip_id, chip_version)

def readBmp180(addr=DEVICE):
  # Register Addresses
  REG_CALIB  = 0xAA
  REG_MEAS   = 0xF4
  REG_MSB    = 0xF6
  REG_LSB    = 0xF7
  # Control Register Address
  CRV_TEMP   = 0x2E
  CRV_PRES   = 0x34
  # Oversample setting
  OVERSAMPLE = 3    # 0 - 3

  # Read calibration data
  # Read calibration data from EEPROM
  cal = bus.read_i2c_block_data(addr, REG_CALIB, 22)

  # Convert byte data to word values
  AC1 = getShort(cal, 0)
  AC2 = getShort(cal, 2)
  AC3 = getShort(cal, 4)
  AC4 = getUshort(cal, 6)
  AC5 = getUshort(cal, 8)
  AC6 = getUshort(cal, 10)
  B1  = getShort(cal, 12)
```

```
  B2  = getShort(cal, 14)
  MB  = getShort(cal, 16)
  MC  = getShort(cal, 18)
  MD  = getShort(cal, 20)

  # Read temperature
  bus.write_byte_data(addr, REG_MEAS, CRV_TEMP)
  time.sleep(0.005)
  (msb, lsb) = bus.read_i2c_block_data(addr, REG_MSB, 2)
  UT = (msb << 8) + lsb

  # Read pressure
  bus.write_byte_data(addr, REG_MEAS, CRV_PRES + (OVERSAMPLE <<
6))
  time.sleep(0.04)
  (msb, lsb, xsb) = bus.read_i2c_block_data(addr, REG_MSB, 3)
  UP = ((msb << 16) + (lsb << 8) + xsb) >> (8 - OVERSAMPLE)

  # Refine temperature
  X1 = ((UT - AC6) * AC5) >> 15
  X2 = (MC << 11) / (X1 + MD)
  B5 = X1 + X2
  temperature = int(B5 + 8) >> 4

  # Refine pressure
  B6  = B5 - 4000
  B62 = int(B6 * B6) >> 12
  X1  = (B2 * B62) >> 11
  X2  = int(AC2 * B6) >> 11
  X3  = X1 + X2
  B3  = (((AC1 * 4 + X3) << OVERSAMPLE) + 2) >> 2

  X1 = int(AC3 * B6) >> 13
  X2 = (B1 * B62) >> 16
  X3 = ((X1 + X2) + 2) >> 2
  B4 = (AC4 * (X3 + 32768)) >> 15
  B7 = (UP - B3) * (50000 >> OVERSAMPLE)

  P = (B7 * 2) / B4

  X1 = (int(P) >> 8) * (int(P) >> 8)
  X1 = (X1 * 3038) >> 16
```

```
  X2 = int(-7357 * P) >> 16
  pressure = int(P + ((X1 + X2 + 3791) >> 4))

  return pressure/100.0

def loop():
    dht = DHT.DHT(DHTPin)   #create a DHT class object
    counts = 0 # Measurement counts
    now = datetime.now()
    current_time = now.strftime("%Y-%m-%d %H:%M")

    for j in range(10):
        counts += 1
        print("Measurement counts: ", counts)
        pressure=readBmp180()
        for i in range(0,15):
            chk = dht.readDHT11()     #read DHT11 and get a return
value. Then determine whether data read is normal according to the
return value.
            if (chk is dht.DHTLIB_OK):      #read DHT11 and get a
return value. Then determine whether data read is normal according
to the return value.
                print("DHT11,OK!")
                break
            time.sleep(0.1)
        print("Humidity : %.2f"%(dht.humidity))
        print("Temperature: %.2f"%(dht.temperature))
        print("Pressure: %.2f mbar\n" %(pressure))
        data = {
                "timestamp":current_time,
                "humidity":dht.humidity,
                "temperature":dht.temperature,
                "pressure":pressure
                }
        results = db.child("measurement").push(data,
user['idToken'])
        time.sleep(2)



if __name__ == '__main__':
    print ('Program is starting ... ')
```

```
    try:
        loop()
    except KeyboardInterrupt:
        GPIO.cleanup()
        exit()
    print("Program Finished!")
```

The code above is the code for reading the DHT11(Pressure) sensor and the BMP180(Temperature and Humidity) sensor, then displays the data onto the terminal while also pushing the data to the database that we implemented, which is Firebase.

Chapter 6: Storing the Sensor Data

```
#Weather_Station.py

import base64
from datetime import datetime

config = {
  "apiKey": "AIzaSyBML8cxQ8MJN0Zvz4s_n-de8heivfPOqfw",
  "authDomain": "group-project-4b008.firebaseapp.com",
  "databaseURL":
"https://group-project-4b008-default-rtdb.firebaseio.com/",
  "storageBucket": "group-project-4b008.appspot.com"
}
firebase = pyrebase.initialize_app(config)
auth = firebase.auth()
user =
auth.sign_in_with_email_and_password("immanuelsy2112@gmail.com",
"123456")
db = firebase.database()

data = {
                "timestamp":current_time,
                "humidity":dht.humidity,
                "temperature":dht.temperature,
                "pressure":pressure
                }
        results = db.child("measurement").push(data,
user['idToken'])
```

The code above is part of the Weather_Station.py(Not a different file) that was shown before, this is just to show how the database will be implemented. First import the necessary database that we will be using, which is Firebase. After that, initializing the configurations of the database by giving it the API key, authentication domain, database URL, and storage bucket. With all these the data will then be pushed to the correct database with the configurations provided.

The data that will be pushed to the database includes timestamps, humidity, temperature, and pressure.

Chapter 7: Visualizing the Data

```
/* index.php */

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <link rel="stylesheet" type="text/css" href="styling.css">
</head>

<body>

    <?php
    session_start();
    ?>

    <div style="margin-left:50px;margin-right:50px;">

        <table>
            <thead>
                <h1>Weather Station</h1>
                <tr>
                    <th>Measurements</th>
                    <th>Humidity</th>
                    <th>Temperature</th>
                    <th>Pressure</th>
                    <th>Timestamp</th>
                </tr>
            </thead>
            <tbody>
                <?php
                include 'dbcon.php';

                $ref_table = 'measurement';
                $fetchdata =
$database->getReference($ref_table)->getValue();
```

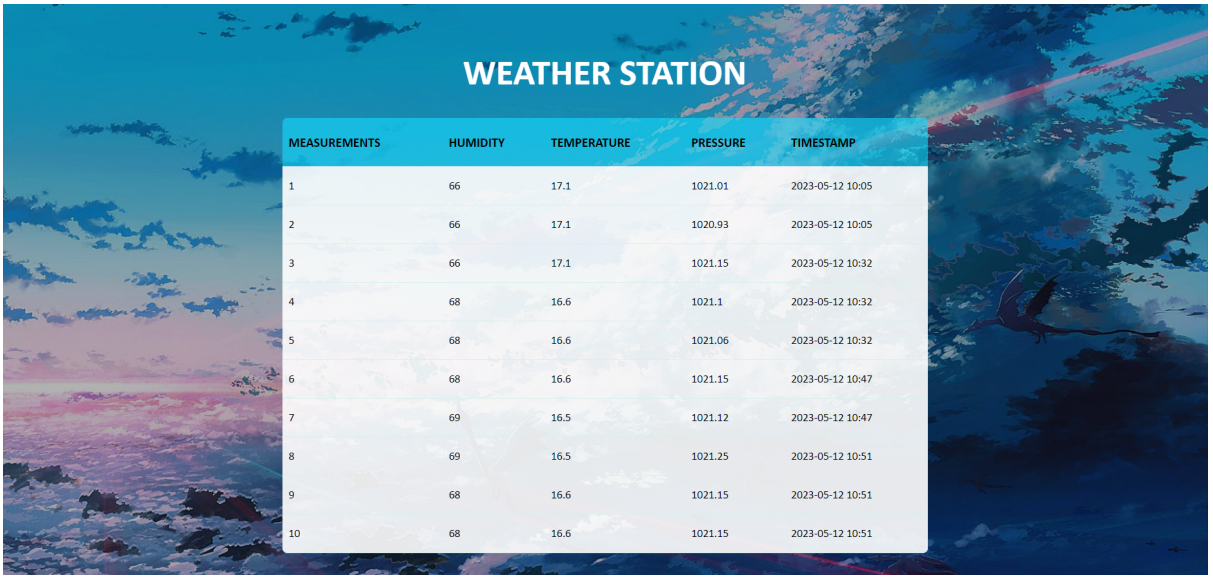
```
                if ($fetchdata > 0) {
                    $i = 1;
                    foreach ($fetchdata as $key => $row) {
                        ?>
                        <tr>
                            <td>
                                <?= $i++; ?>
                            </td>
                            <td>
                                <?= $row['humidity']; ?>
                            </td>
                            <td>
                                <?= $row['temperature']; ?>
                            </td>
                            <td>
                                <?= $row['pressure']; ?>
                            </td>
                            <td>
                                <?= $row['timestamp']; ?>
                            </td>
                        </tr>
                        <?php
                    }
                } else {
                    ?>
                    <tr>
                        <td>No Record Found</td>
                    </tr>
                    <?php
                }
                ?>

            </tbody>
        </table>
    </div>

</body>

</html>
```

The code above is the code that will be doing the frontend of the project. Basically it takes the data that was previously pushed to the database and displays it onto the web browser. The output will be shown as below.



As said before, after sending the data from the backend to the database, the frontend will then access the database and display the data onto the web browser. The data that will be displayed includes humidity, temperature, pressure, and timestamp.

Chapter 8: Final Verdict