WEATHER STATION

COURSEWORK FOR INTERNET OF THINGS FUNDAMENTALS

Ioannis Kozas

OVERVIEW

The purpose of this application is to monitor the physical environment using electronic sensors connected to the Raspberry Pi. The collected data is then presented on a locally running web page through a database and also accessible from any smartphone through the 'Blynk' IoT app.

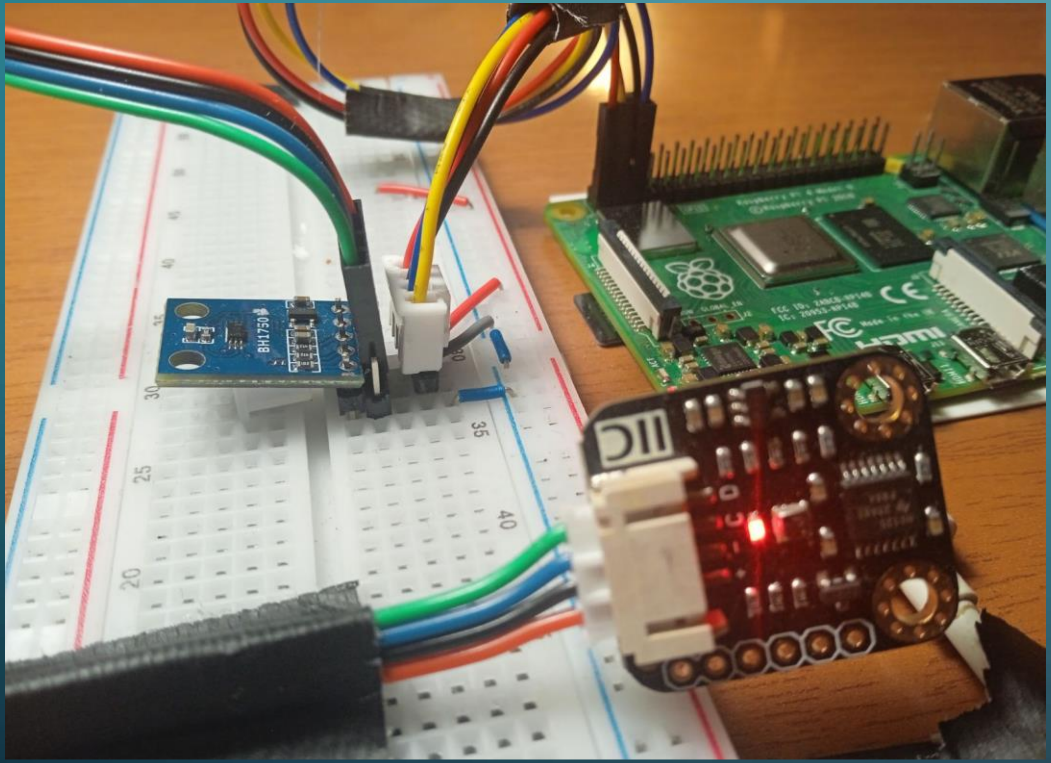
USE CASES

Weather stations play a crucial role in diverse applications. Energy industries leverage the data for efficient operation of renewable sources. In agriculture, these sensors aid farmers in optimizing crop growth by monitoring ideal environmental conditions. Urban planners utilize the information to design climate-resilient infrastructure. Such sensors contribute significantly to various research domains, enhancing our understanding of dynamic weather patterns.

APPLICATION SPECIFICATIONS

Hardware	• Raspberry pi, BME 280, BH 1750
Programming language	• Python
Database	• Docker with Postgres
IOT Platform	• Blynk IOT

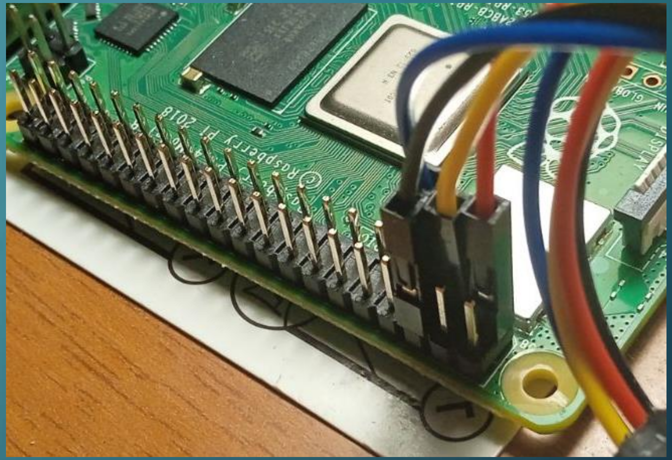
HARDWARE SETUP



- Both of sensors are communicating throw I2C(IIC) communication protocol with the Raspberry pi
- The advantage of the I2C protocol is that all the sensors using it can be connected in the same GPIOs, the SDA and SCL, which are the GPIO 3 and 5 on the Raspberry pi
- Both sensors are using 3.3v power so we can connect them to the same power and ground GPIO
- Therefore both sensors can be connected to the same GPIOs making the installation quite easy

HARDWARE SETUP

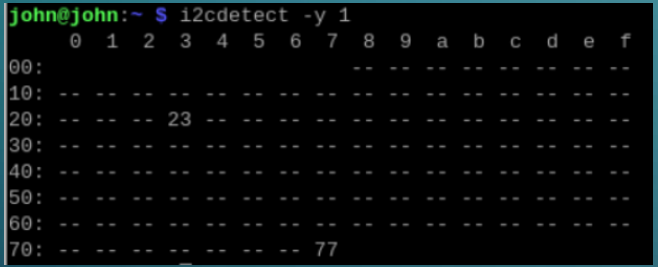
Description	GPIO RASPBERRY	BM 1750	BME 280
POWER	1	VCC	+
GROUND	6	GND	-
I2C SDA	3	SDA	D
I2C SCL	5	SCL	C
Address select	6	ADDR	



SOFTWARE SETUP FOR THE RASPBERRY PI

ENABLE I2C

1. Enable I2C communication
   - In the terminal type "sudo raspi-config"
   - Select interface options
   - Select and Enable I2C
2. Verify that the sensors are connected
   - In the terminal type "i2cdetect –y 1"
   - Get the I2C sensors

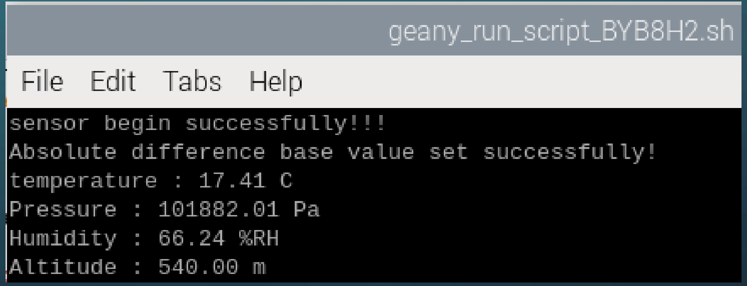


```
john@john:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- 23 -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- 77
```

- Each I2C device have a specific address
- BM1750: has the address 23
- BME280: has the address 77

GET DATA FROM BME280

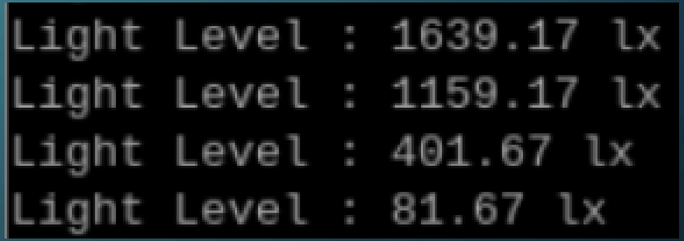
- The BME280 sensor that I use is manufactured by DFRobot, so to communicate with the sensor from the Raspberry Pi, we need to download the sensor's library.
- There is a GitHub page for the sensor where we can download the library. After downloading, we navigate to the Python folder, and within it, there is a Raspberry Pi folder containing examples of Python code that we can use.
- When we run the "read_data_i2c.py" example we get the result above:



```
sensor begin successfully!!!
Absolute difference base value set successfully!
temperature : 17.41 C
Pressure : 101882.01 Pa
Humidity : 66.24 %RH
Altitude : 540.00 m
```

GET DATA FROM BH1750

- The BH1750 is a simpler sensor that outputs one value, light level in lux (unit of illuminance)
- I found the code for sensor in this site: https://www.raspberrypi-spy.co.uk/2015/03/bh1750fvi-i2c-digital-light-intensity-sensor/
- When we run the code the output on the terminal is the above:



```
Light Level : 1639.17 lx
Light Level : 1159.17 lx
Light Level : 401.67 lx
Light Level : 81.67 lx
```

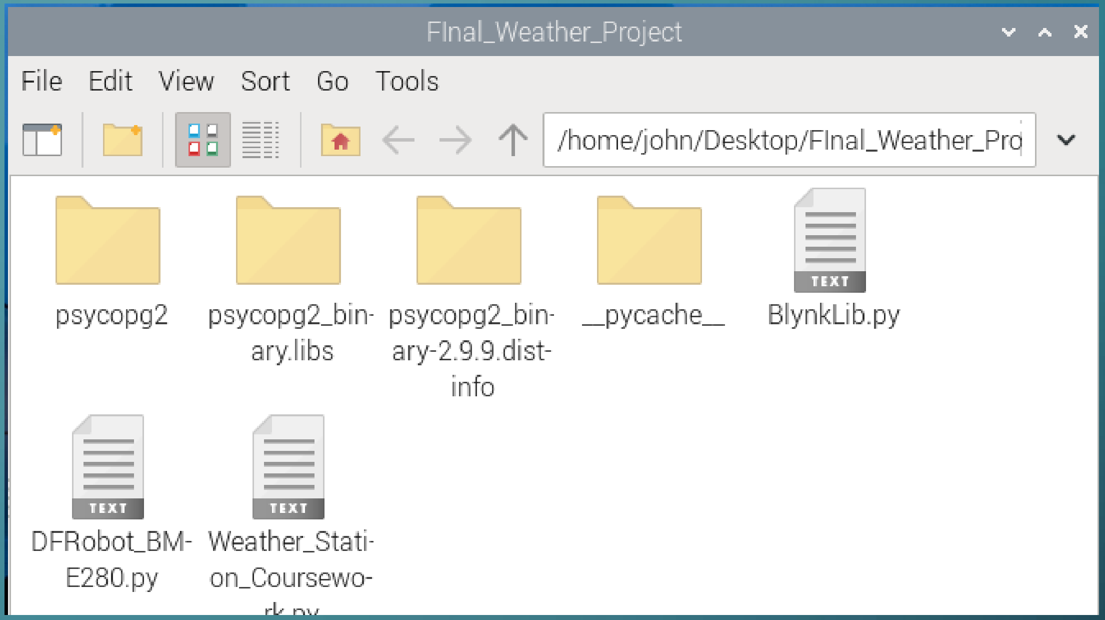
SOFTWARE SETUP FOR THE RASPBERRY PI

VIRTUAL ENVIRONMENTS

- To download and use libraries for the raspberry pi I used virtual environments.
- Virtual environments create isolated spaces for Python projects, allowing you to install and manage dependencies specific to each project.
- This prevents conflicts between different projects that might require different versions of the same library.

SOFTWARE SETUP FOR RASPBERRY PI

To the right, you can see the project folder and the libraries that I have utilized.


FInal_Weather_Project
File Edit View Sort Go Tools
/home/john/Desktop/FInal_Weather_Pro
psycopg2
psycopg2_binary.libs
psycopg2_binary-2.9.9.dist-info
__pycache__
TEXT
BlynkLib.py
TEXT
DFRobot_BME280.py
TEXT
Weather_Station_Coursework.py

LIBRARIES

DFRobot_BME280

The library of the BME 280 sensor

BlynkLib

Library for the Blynk IOT

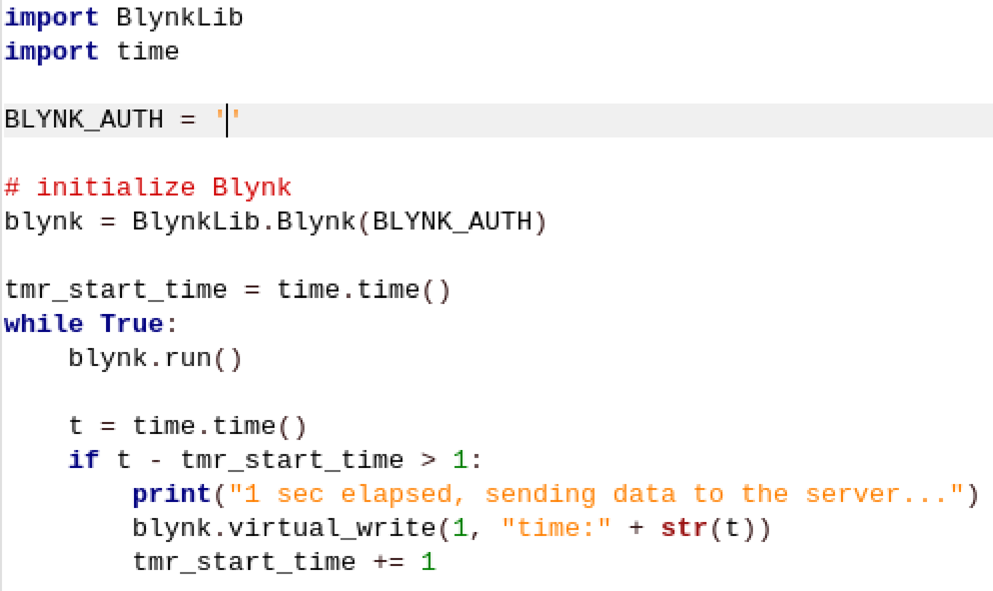
psycopg2

A PostgreSQL database adapter for python

SOFTWARE SETUP FOR THE RASPBERRY PI

COMMUNICATING WITH BLINK

- The Blynk IoT platform enables communication between single-board computers (SBC) and other devices like PCs and smartphones through the Blynk cloud.
- Upon registering on the blynk.io site, users access the "Dashboard" where they can view their projects, referred to as "templates." Each template has a unique Authorization token that is used in the code on the SBC to communicate with the Blynk cloud.



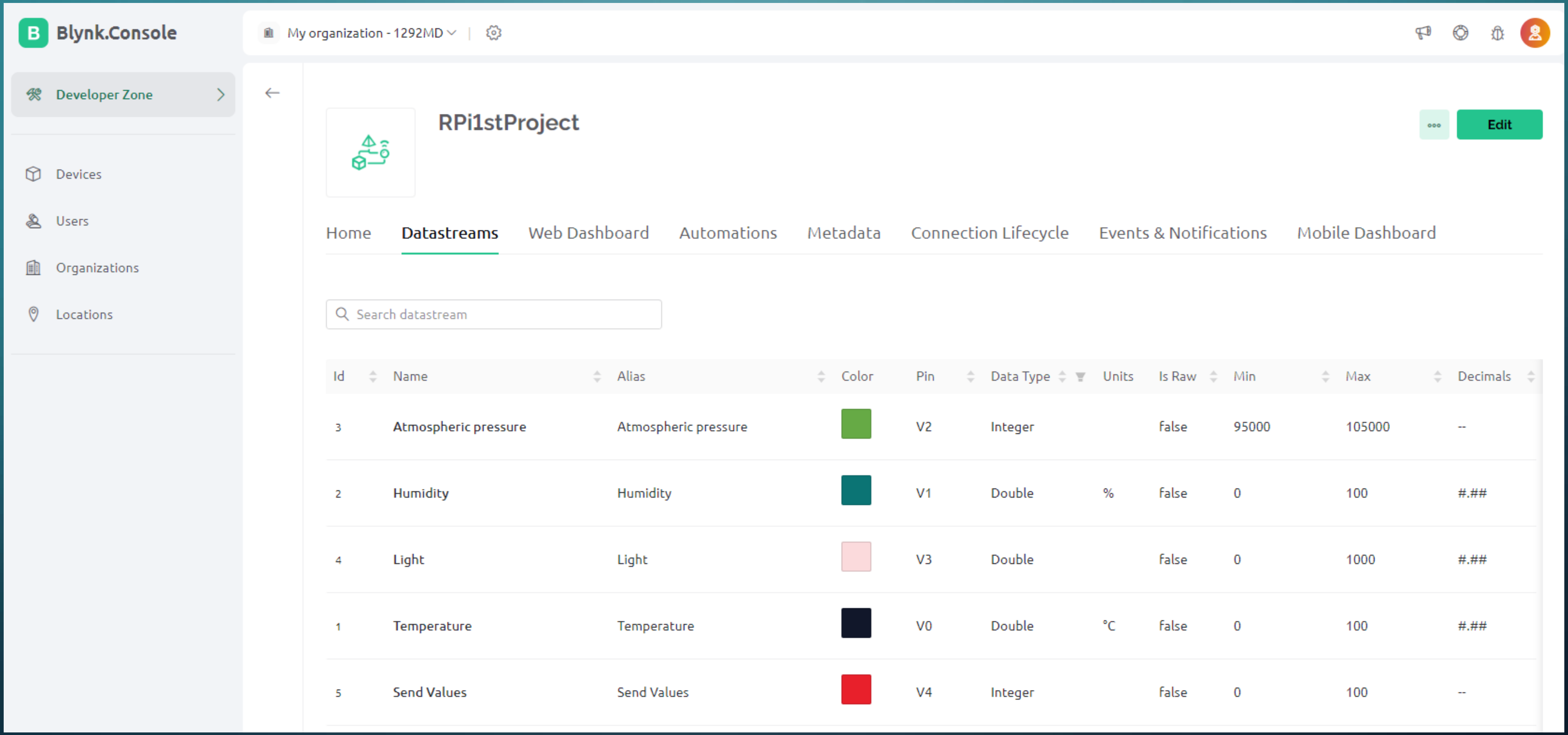
```python
import BlynkLib
import time

BLYNK_AUTH = ''

# initialize Blynk
blynk = BlynkLib.Blynk(BLYNK_AUTH)

tmr_start_time = time.time()
while True:
    blynk.run()

    t = time.time()
    if t - tmr_start_time > 1:
        print("1 sec elapsed, sending data to the server...")
        blynk.virtual_write(1, "time:" + str(t))
        tmr_start_time += 1
```

- The Python code for sending data to Blynk is illustrated in the image on the left.

1. To utilize the Blynk library, we need to import "BlynkLib".
2. Include the authorization token (BLYNK_AUTH) in the code.
3. In the "while True" loop, initiate the Rpi-Blynk communication with "blynk.run()."
4. Within the if loop, use "blynk.virtual_write()" to send data to the Blynk cloud. In the parentheses, the first parameter is the Virtual Pin number, and the second parameter is the value to set for that Pin.

SOFTWARE SETUP FOR BLYNK

This is developer zone in the Blynk console, here we can set and customize the virtual pins

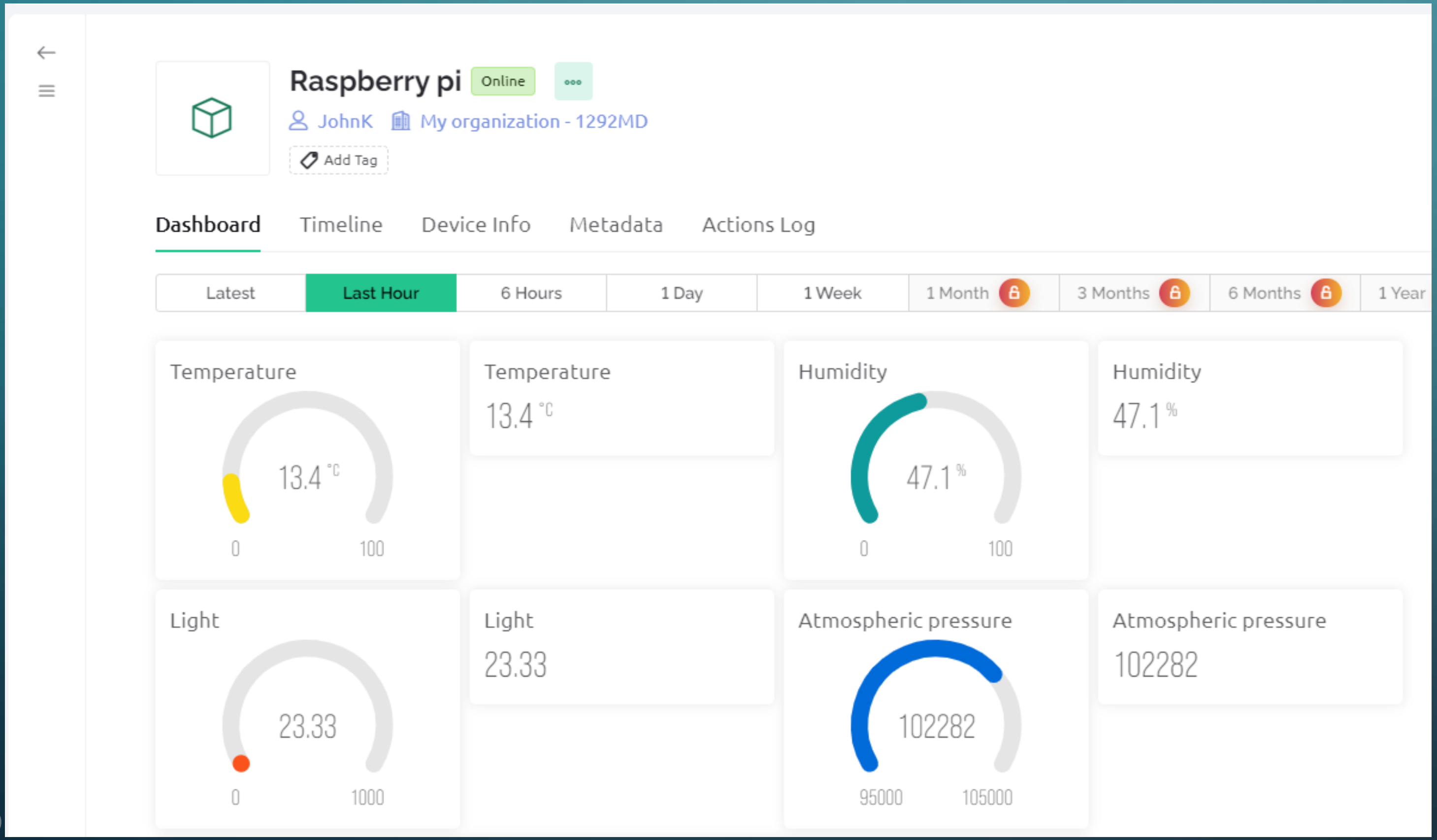


RPi1stProject

Id	Name	Alias	Color	Pin	Data Type	Units	Is Raw	Min	Max	Decimals
3	Atmospheric pressure	Atmospheric pressure	🟩	V2	Integer		false	95000	105000	--
2	Humidity	Humidity	🟦	V1	Double	%	false	0	100	#.##
4	Light	Light	⬜	V3	Double		false	0	1000	#.##
1	Temperature	Temperature	⬛	V0	Double	°C	false	0	100	#.##
5	Send Values	Send Values	🟥	V4	Integer		false	0	100	--

SOFTWARE SETUP FOR BLYNK

- The image on the right displays the Blynk IoT app on the smartphone.
- The image above illustrates how the data is represented in the Blynk console.



Raspberry pi Online

JohnK My organization - 1292MD

Add Tag

Temperature 13.4 °C

Temperature 13.4 °C

Humidity 47.1 %

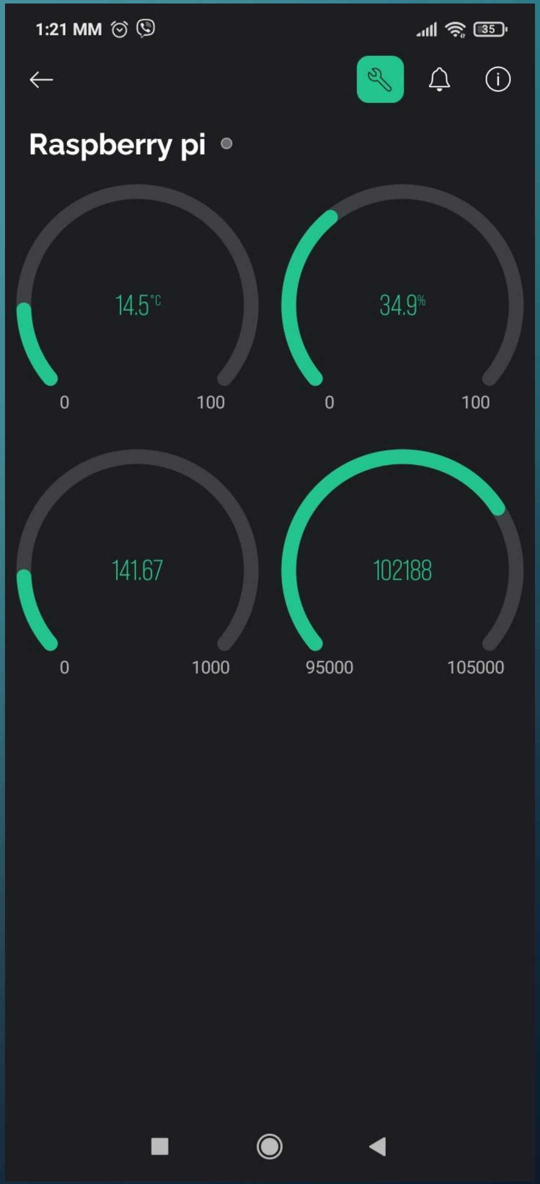
Humidity 47.1 %

Light 23.33

Light 23.33

Atmospheric pressure 102282

Atmospheric pressure 102282



Raspberry pi

14.5 °C

34.9 %

141.67

102188

DOCKER AND POSTGRES SETUP

DOCKER

- Docker is a containerization platform that allows you to package your application and its dependencies together in a container.
- Containers provide a consistent and isolated environment, ensuring that your application runs the same way across different environments.

POSTGRES

- PostgreSQL is a powerful, open-source relational database management system (RDBMS).
- PostgreSQL databases are commonly used in IoT projects to store and manage data efficiently.

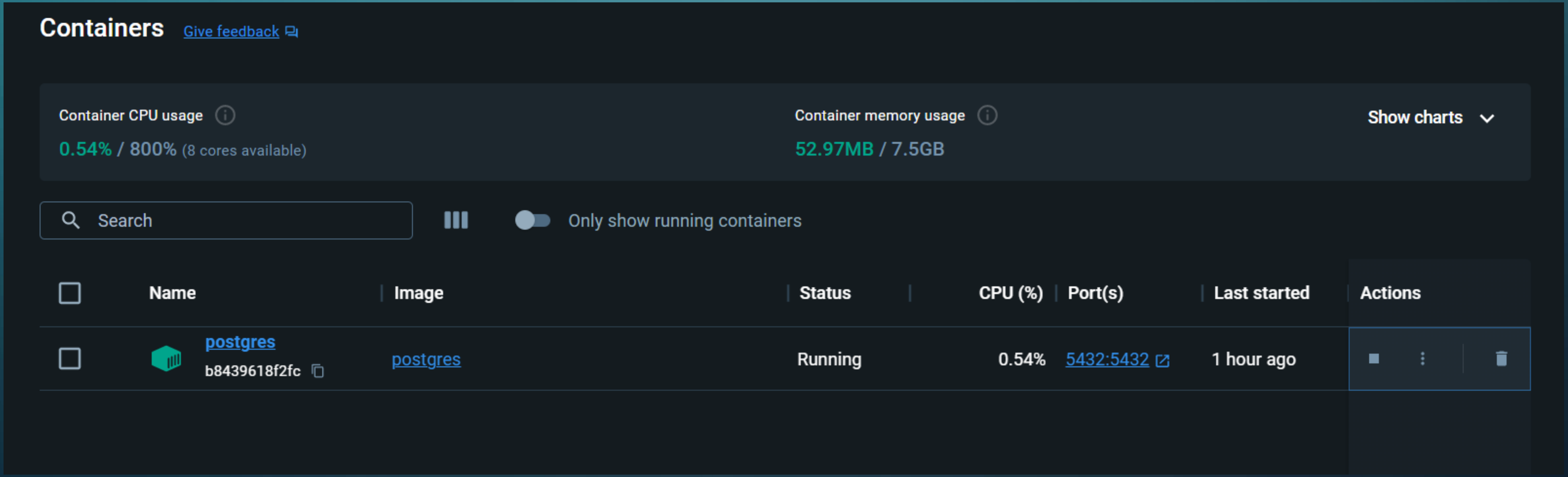
SETTING UP POSTGRES IN DOCKER

- I use a Docker image that includes PostgreSQL to create a containerized instance of the database.
- There is a dedicated page in docker for setting up Postgres which is here:
- https://hub.docker.com/_/postgres

DOCKER AND POSTGRES SETUP

- After downloading Docker, execute the following command in the command line. You can customize the database name, user, and password. Additionally, set a port using -p 5432:5432. To make the database accessible to your local network, include your local IP in the command:

-
  ```
  docker run -d --name your_postgres_container_name -p your_local_ip:5432:5432 -e POSTGRES_PASSWORD=your_password -e POSTGRES_USER=your_username -e POSTGRES_DB=your_database_name postgres:latest
  ```



Containers

Container CPU usage
0.54% / 800% (8 cores available)

Container memory usage
52.97MB / 7.5GB

	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
□							
□	postgres b8439618f2fc	postgres	Running	0.54%	5432:5432	1 hour ago	■ ⋮ 🗑

- This is now the database running local

DATABASE TABLE AND SETTING VALUES

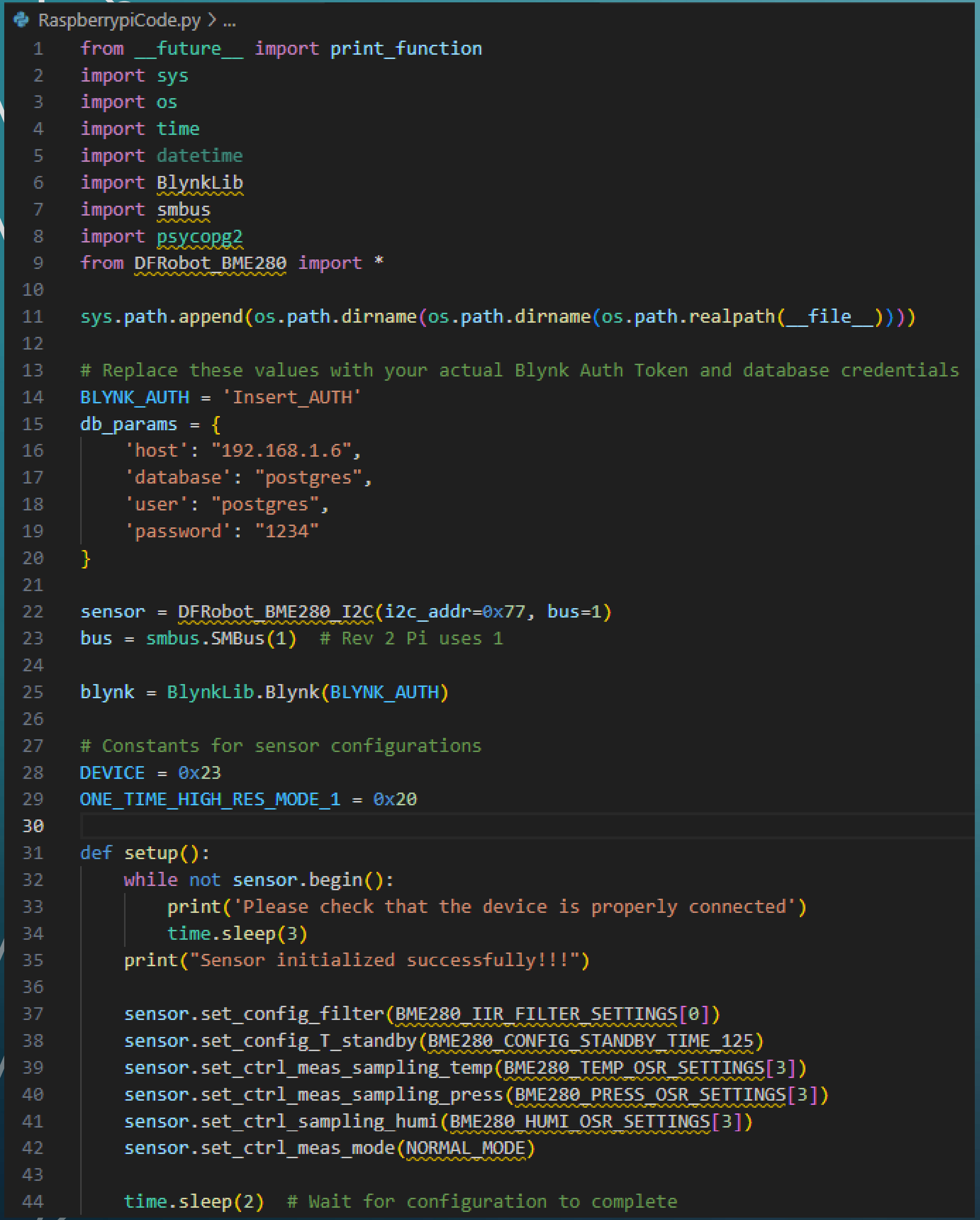
- After setting up the database, I used DBeaver to make a table named “weatherproject” with the data:
- Temperature, humidity, pressure and light.
- This is the SQL code that I used in the Dbeaver:

```
CREATE TABLE weatherproject (id SERIAL PRIMARY KEY, temperature INTEGER, humidity INTEGER, pressure
INTEGER, light INTEGER);
```

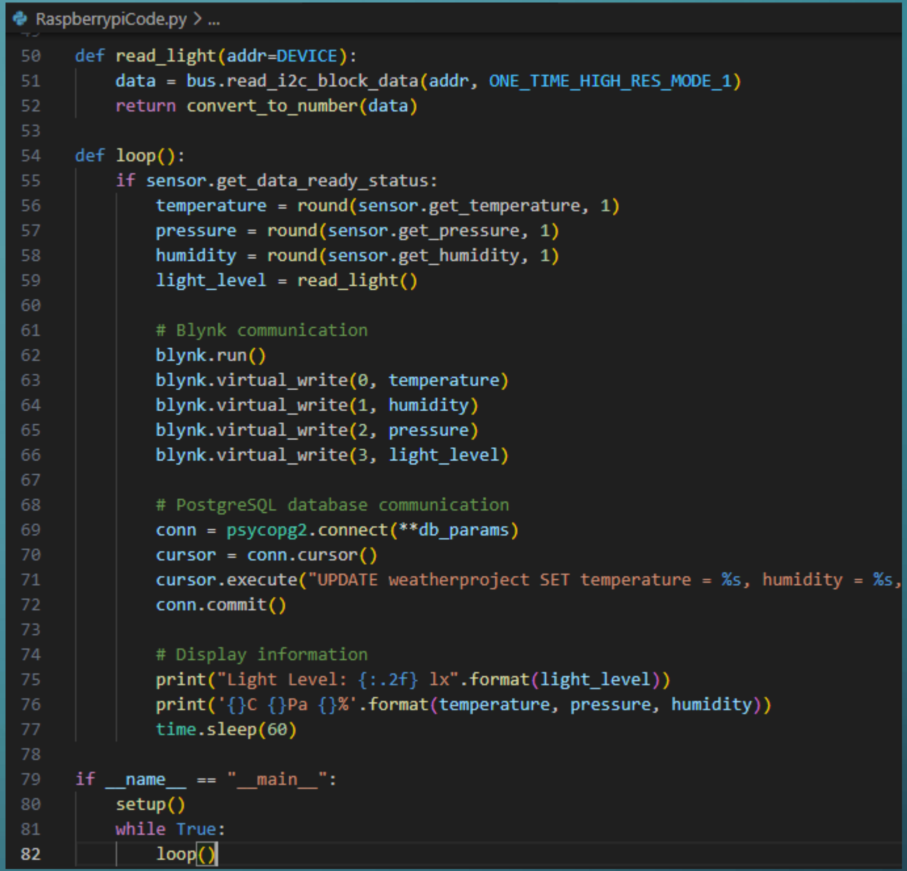
CONNECTING THE RASPBERRY PI WITH THE DATABASE

- After creating the table, I utilized psycopg2, a PostgreSQL database adapter for Python, to send values from our Raspberry Pi to the database.

COMBINING THE CODE TOGETHER IN THE RASPBERRY PI



```python
from __future__ import print_function
import sys
import os
import time
import datetime
import BlynkLib
import smbus
import psycopg2
from DFRobot_BME280 import *

sys.path.append(os.path.dirname(os.path.dirname(os.path.realpath(__file__))))

# Replace these values with your actual Blynk Auth Token and database credentials
BLYNK_AUTH = 'Insert_AUTH'
db_params = {
    'host': "192.168.1.6",
    'database': "postgres",
    'user': "postgres",
    'password': "1234"
}

sensor = DFRobot_BME280_I2C(i2c_addr=0x77, bus=1)
bus = smbus.SMBus(1)  # Rev 2 Pi uses 1

blynk = BlynkLib.Blynk(BLYNK_AUTH)

# Constants for sensor configurations
DEVICE = 0x23
ONE_TIME_HIGH_RES_MODE_1 = 0x20

def setup():
    while not sensor.begin():
        print('Please check that the device is properly connected')
        time.sleep(3)
    print("Sensor initialized successfully!!!")

    sensor.set_config_filter(BME280_IIR_FILTER_SETTINGS[0])
    sensor.set_config_T_standby(BME280_CONFIG_STANDBY_TIME_125)
    sensor.set_ctrl_meas_sampling_temp(BME280_TEMP_OSR_SETTINGS[3])
    sensor.set_ctrl_meas_sampling_press(BME280_PRESS_OSR_SETTINGS[3])
    sensor.set_ctrl_sampling_humi(BME280_HUMI_OSR_SETTINGS[3])
    sensor.set_ctrl_meas_mode(NORMAL_MODE)

    time.sleep(2)  # Wait for configuration to complete
```



```python
def read_light(addr=DEVICE):
    data = bus.read_i2c_block_data(addr, ONE_TIME_HIGH_RES_MODE_1)
    return convert_to_number(data)

def loop():
    if sensor.get_data_ready_status:
        temperature = round(sensor.get_temperature, 1)
        pressure = round(sensor.get_pressure, 1)
        humidity = round(sensor.get_humidity, 1)
        light_level = read_light()

        # Blynk communication
        blynk.run()
        blynk.virtual_write(0, temperature)
        blynk.virtual_write(1, humidity)
        blynk.virtual_write(2, pressure)
        blynk.virtual_write(3, light_level)

        # PostgreSQL database communication
        conn = psycopg2.connect(**db_params)
        cursor = conn.cursor()
        cursor.execute("UPDATE weatherproject SET temperature = %s, humidity = %s,
        conn.commit()

        # Display information
        print("Light Level: {:.2f} lx".format(light_level))
        print('{}C {}Pa {}%'.format(temperature, pressure, humidity))
        time.sleep(60)

if __name__ == "__main__":
    setup()
    while True:
        loop()
```

- The code initializes Blynk, the BME280 and BH1750 sensors, and establishes a connection to the PostgreSQL database.
- The setup function configures the BME280 sensor.
- The read_light function reads light level data from the BH1750 sensor.
- The loop function reads sensor data, updates Blynk widgets, updates the local database, and prints relevant information.
- The main block initializes the setup and enters an infinite loop for continuous sensor readings.

SOFTWARE SETUP FOR THE LOCAL PAGE

- I made a simple local page that snows the data of the database using python, flask and SQLAlchemy

FLASK

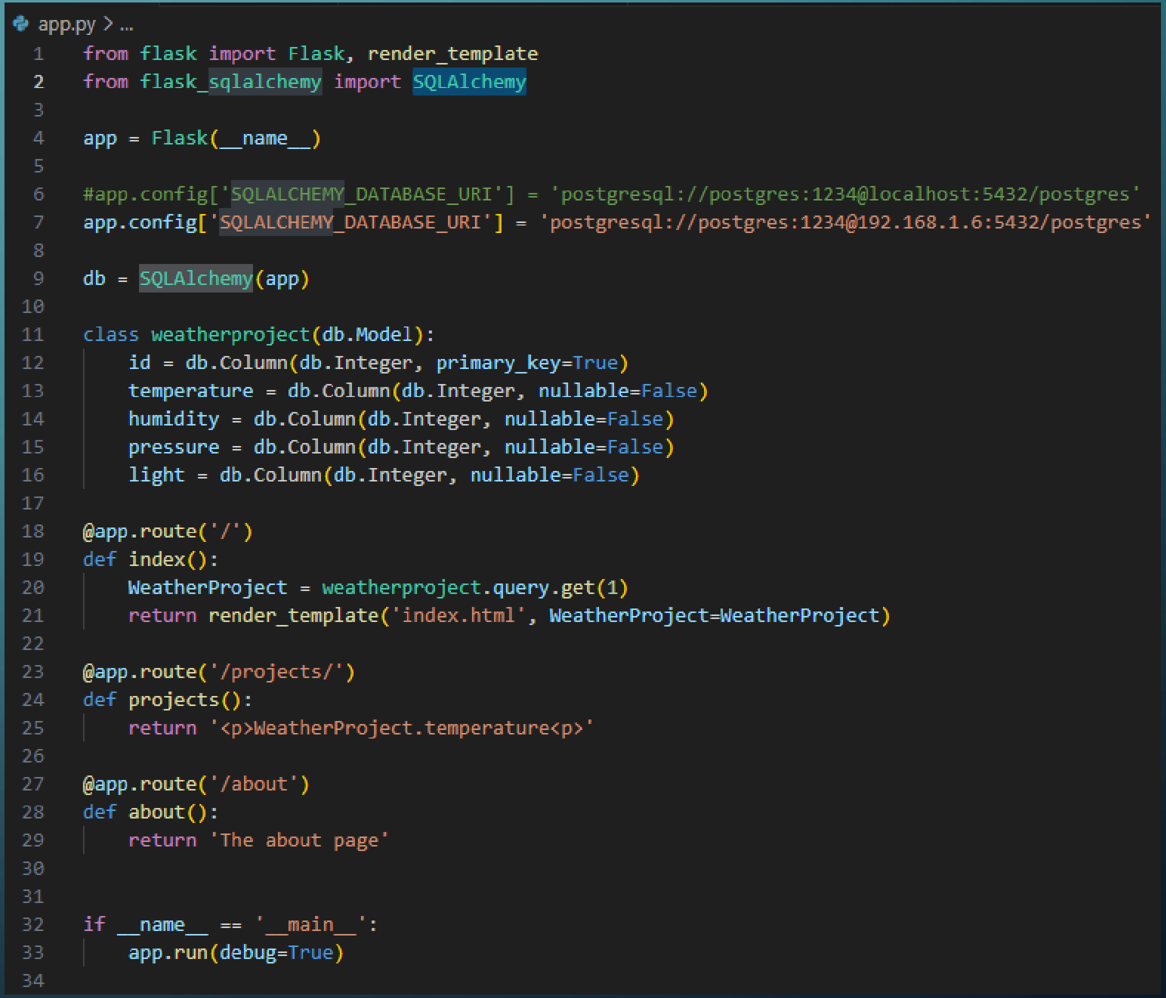
- Flask is a lightweight and flexible web framework for Python.
- It simplifies web development by providing tools for handling routing, templates, and requests.

SQLALCHEMY

- SQLAlchemy is a powerful and versatile SQL toolkit and Object-Relational Mapping (ORM) library for Python.
- It facilitates interaction with relational databases by providing a high-level, Pythonic interface.

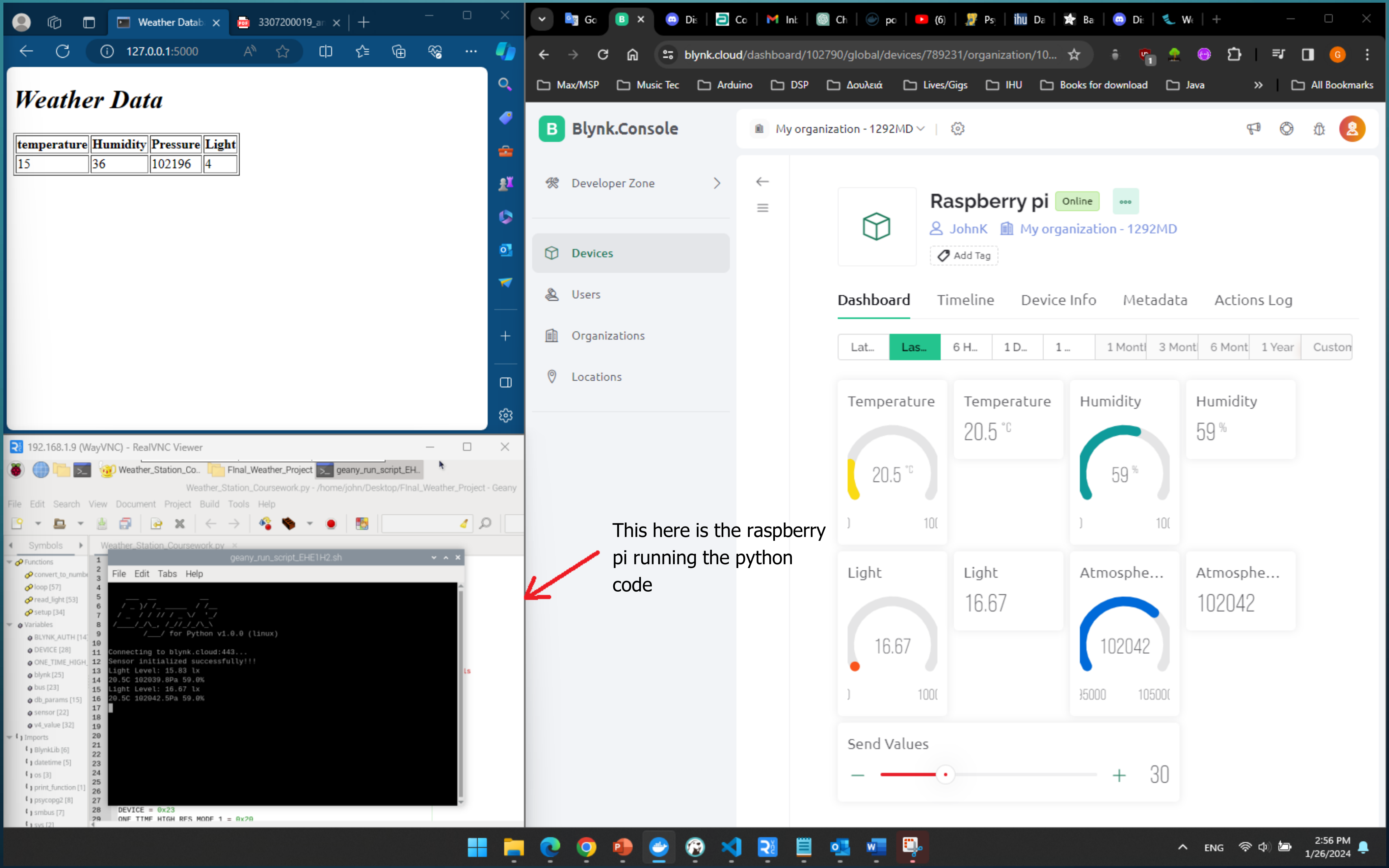
THE CODE

- The data from the database is assigned to variables with corresponding names and then rendered onto an HTML page, specifically 'index.html.' This HTML page consists of a simple table that visually represents the values.



```python
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)

#app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://postgres:1234@localhost:5432/postgres'
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://postgres:1234@192.168.1.6:5432/postgres'

db = SQLAlchemy(app)

class weatherproject(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    temperature = db.Column(db.Integer, nullable=False)
    humidity = db.Column(db.Integer, nullable=False)
    pressure = db.Column(db.Integer, nullable=False)
    light = db.Column(db.Integer, nullable=False)

@app.route('/')
def index():
    WeatherProject = weatherproject.query.get(1)
    return render_template('index.html', WeatherProject=WeatherProject)

@app.route('/projects/')
def projects():
    return '<p>WeatherProject.temperature<p>'

@app.route('/about')
def about():
    return 'The about page'


if __name__ == '__main__':
    app.run(debug=True)

```

RUNNING THE APPLICATION



127.0.0.1:5000

Weather Data

temperature	Humidity	Pressure	Light
15	36	102196	4

192.168.1.9 (WayVNC) - RealVNC Viewer

Weather_Station_Coursework.py - /home/john/Desktop/Final_Weather_Project - Geany

geany_run_script_EHE1H2.sh

```
    ___  __          __
   / _ )/ /_ _____  / /__
  / _  / / // / _ \/  '_/
 /____/_/\_, /_//_/_/\_\
        /___/ for Python v1.0.0 (linux)

Connecting to blynk.cloud:443...
Sensor initialized successfully!!!
Light Level: 15.83 lx
20.5C 102039.8Pa 59.0%
Light Level: 16.67 lx
20.5C 102042.5Pa 59.0%
```

```
DEVICE = 0x23
ONE_TIME_HIGH_RES_MODE_1 = 0x20
```

blynk.cloud/dashboard/102790/global/devices/789231/organization/10...

Blynk.Console — My organization - 1292MD

Raspberry pi — Online

JohnK — My organization - 1292MD

Temperature	Temperature	Humidity	Humidity
20.5 °C	20.5 °C	59 %	59 %

Light	Light	Atmosphe...	Atmosphe...
16.67	16.67	102042	102042

Send Values: 30

This here is the raspberry pi running the python code

MAKING THE HARDWARE PORTABLE

- Now that all the software and hardware are functional, I've made a simple modification to the weather station, making it portable.
- I utilized a power bank to supply power to the Raspberry Pi and enclosed all the hardware in a box.
- With these changes, the weather station is now portable, limited only by the range of the Wi-Fi connection.