Research Article

How to Build an IoT System Using AI and Drones to Prevent Wildfires in California

Jun Wang[1]

1. Arizona State University, United States

In the wildfire monitoring system, the DHT22 and BMP180 sensors will be used to collect temperature, humidity, pressure, and altitude data. This data will be sent to a MySQL database for storage via AWS MQTT, allowing for SMS and email notifications to monitor analysts and enabling alerts to fire stations and the data for future research. The Raspberry Pi 4 is utilized in this project as a compact and portable computer to connect the DHT22 and BMP180 sensors. The Raspberry Pi 4 will have AWS-IOT-MQTT installed to transmit data from the Pi to a Restful server. The Restful server, built with FastAPI, will persist the data into the MySQL database and send email notifications to users.

This paper will demonstrate the proper use of sensors to construct your IoT system. The cost of the project can be influenced by the price of different sensors, and using reliable sensors can save time during debugging by helping developers identify errors in the code rather than in the hardware. The project also involves the use of several libraries on the Raspberry Pi, including Adafruit_DHT, Adafruit_Python_DHT, Adafruit_CircuitPython, and Adafruit_Python_BMP. These libraries are essential for retrieving data from DHT22 and BMP180 sensors.

By following this paper, readers will understand how to effectively build and manage an IoT weather monitoring system, taking into consideration the cost and reliability of sensors, as well as the interaction of various software libraries and drones system. Also, Drones have become a crucial tool for rescue operations. With continuous advancements in science and technology, their functionalities are steadily increasing. In rescue missions, drones can monitor wildfire in real time, scout various hazardous environments, and assist firefighters in firefighting and rescue efforts. Additionally, modern building structures are complex, and multiple factors influence fire incidents. Firefighters often cannot promptly assess the specific situation of a fire scene or safely enter to conduct rescue operations. By utilizing drones, remote-controlled reconnaissance of the fire scene can be conducted, providing better support for firefighters in their firefighting efforts.

 doi.org/10.32388/DNC09I.2

Introduction

IoT weather monitoring systems leverage cutting-edge IT technologies, including advanced sensors, portable mini-computers, AWS Cloud services, Drones, high-level programming languages, databases, and AI, to create versatile systems applicable in various scenarios. These systems can record data for climate change studies and support future research through AI models like ANN (Artificial Neural Networks) or the Random Forest method, aiding in weather prediction and natural disaster prevention.

For instance, an IoT weather monitoring system could be deployed in California to help prevent forest fires. Historical data shows that the top seven worst wildfires in California each resulted in several billion dollars in insured losses. The 2018 Camp Fire alone caused estimated damages of $10 billion, or approximately $10.38 billion in 2020 value. Deploying an IoT weather monitoring system in forests could enable early detection of high temperatures through sensors, which would then alert fire stations or weather monitoring stations. These agencies could respond promptly to these alerts, potentially preventing fires in their early stages.

This paper will demonstrate the process of building an IoT weather monitoring system. How to select appropriate sensors, connect them to a Raspberry Pi, test their functionality, and use Python code to read data from these sensors and how to integrate advanced drone in IoT system. Additionally, how to transmit data to a server via MQTT and troubleshoot any errors that arise. By the end of this paper, a comprehensive understanding of IoT systems with advanced Drone system. The knowledge and experience gained from this project can be applied to other IoT applications. Also, An Unmanned Aerial Vehicle (UAV) is an aircraft controlled by a wireless remote-control device or a pre-programmed system. It does not require a pilot to operate it from the cockpit, as its flight process is automatically controlled by electronic equipment. Since no pilot-related equipment needs to be installed on the aircraft, space can be effectively saved for carrying application devices to complete various assigned tasks.

The biggest difference between drones and manned aircraft is that a drone cannot complete any task solely by itself. It requires a strict control system and various application devices depending on the mission requirements. Therefore, drones are also referred to as unmanned aircraft systems (UAS).

The application of drone technology has overcome the challenges of fire rescue operations, allowing for more accurate acquisition of fire scene information and the implementation of effective rescue

strategies, significantly improving the efficiency of firefighting and rescue work. Drones offer distinct advantages in fire rescue operations and hold great practical significance.

Design Process

- Software and Hardware Preparation

Before we enter the design process, the software and hardware requirement need to show first.

Software requirement	Hardware requirement
Python Charm IDE	Raspberry Pi 4
MySQL database	DHT22 Sensor
Restful - fastAPI	BMP180 Sensor
Python3	Breadboard
BMP180 and DHT22 library(adafruit)	Jumper wires
DJI Pilot 2	DJI Matrice 300 RTK
DJI Pilot 2	DJI Mavic 3T
FreeFlight 6	Parrot Anafi Thermal
Ardupilot	Gaia 160
K-MAX UAS Ground Control Station	Kaman K-MAX UAS

When building a weather IoT system, cost is a crucial factor. Different sensors come at varying prices, and using unreliable sensors can lead to frequent replacements, increasing both time and budget costs. Poor-quality sensors can also send inaccurate data to servers, affecting weather information accuracy and future research outcomes.

For purchasing reliable sensors and devices, Amazon and other E-commence websites are good choices for reviews from other users and compare prices to make informed decisions. For instance, HiLetgo offers high-quality DHT22 and BMP180 sensors. Purchase 5 BMP280 sensors for $6 and 2 DHT22 sensors for $13. Additionally, AITRIP provides 10 BMP180 sensors for $8. All these options are

reliable.To cover wildfire-prone areas in the Los Angeles region (about 1,500–2,000 square miles), an IoT wildfire detection system would require 750–2,000 temperature sensors, assuming a density of 1 sensor per 1–2 square miles. Basic IoT sensors cost $50–$200 each, while advanced AI-enabled sensors range from $500–$2,000 each, bringing total sensor costs to $75,000–$4 million. Additional expenses for connectivity (LoRaWAN, LTE, satellite), power (solar, batteries), cloud AI processing, and maintenance push the overall budget to $200,000, depending on coverage, sensor type, and infrastructure needs for a fully integrated wildfire and drone monitoring system.

- IoT Level

In the project, the IoT level 3 is chosen for building the weather monitor system. A level 3 IoT system has a single node. Data is stored and analyzed in the cloud and the application is cloud-based. Level 3 IoT systems are suitable for solutions where the data involved is big and the analysis requirements are computationally intensive.[1] Based on these characteristics, IoT level 3 is suitable for this IoT monitor system. Also, the system can integrate smoke concentration monitoring, flame detection, and temperature monitoring for forest fires, forming a multi-sensor data fusion wireless sensor network based on DHT22. By deploying multiple sensor nodes, the system continuously monitors forest fire conditions and periodically uploads monitoring data to the monitoring terminal.

When no fire is detected, the DHT22 module in the nodes remains in sleep mode, while the sensors monitor environmental parameters at regular intervals to conserve energy and extend the lifespan of the monitoring nodes. In the event of a fire, the DHT22 module is awakened to transmit real-time sensor data and relay location information through the node network to the monitoring center. This enables rapid fire situation assessment and precise fire location tracking, ensuring timely response to forest fires and minimizing losses.

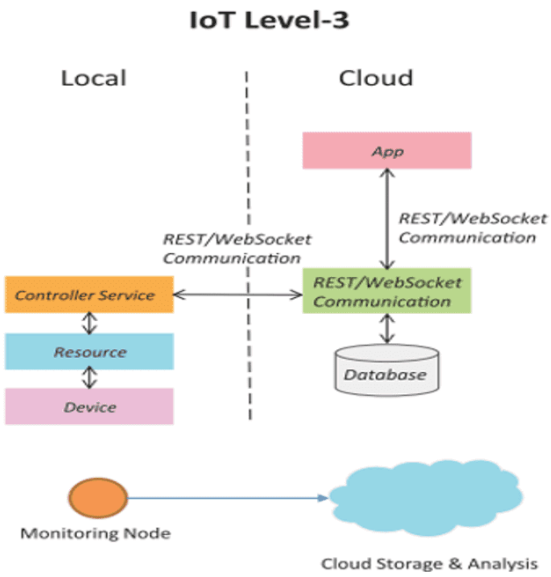


Figure 1. [1]

- Block diagram

doi.org/10.32388/DNC09I.2

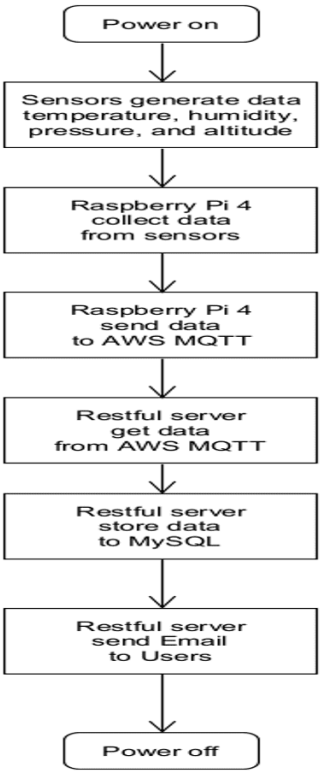


Figure 2.

- Data flow charts

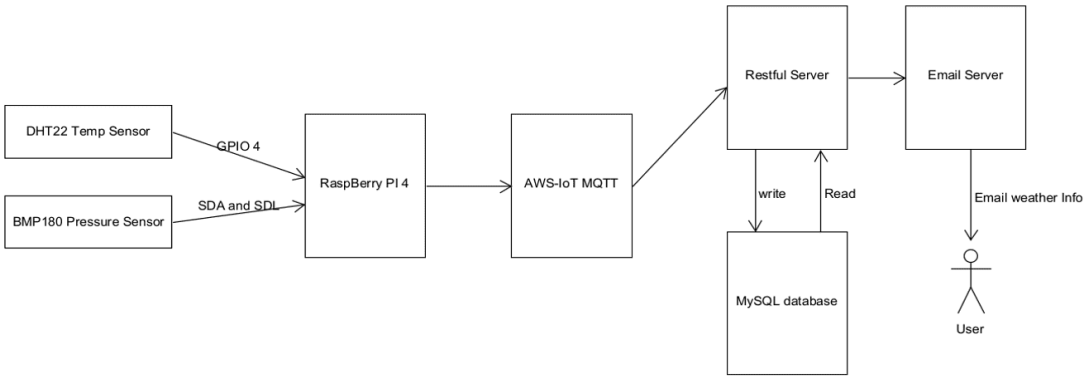


Figure 3.

doi.org/10.32388/DNC09I.2

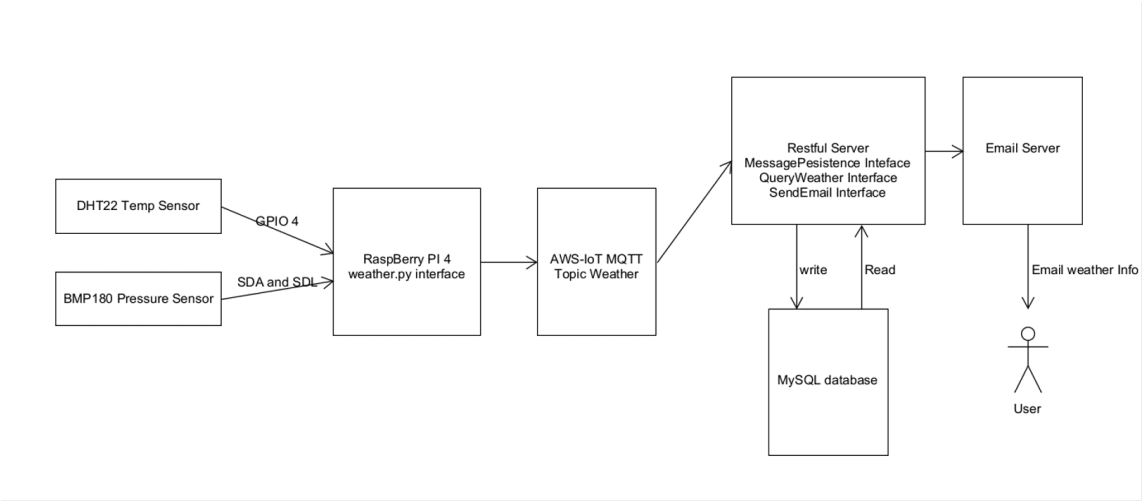
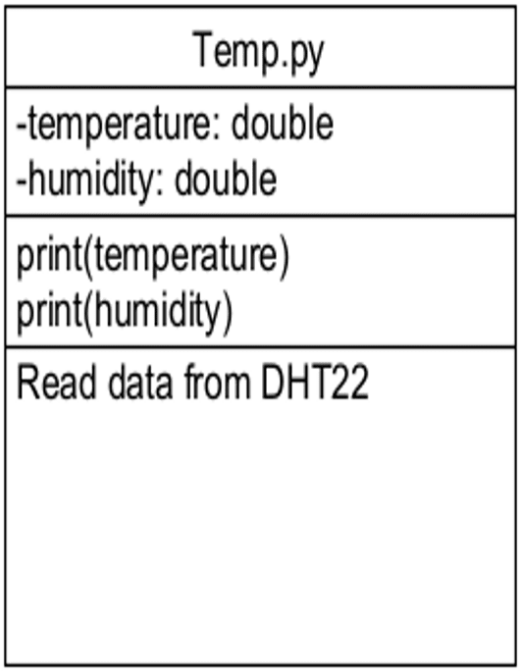
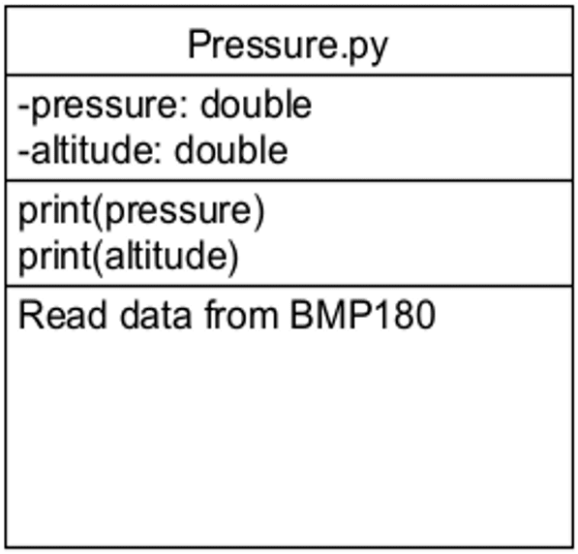
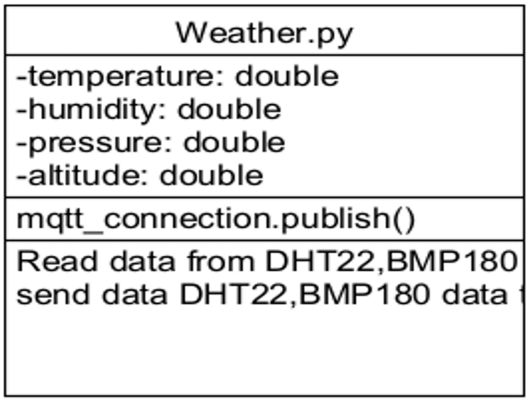
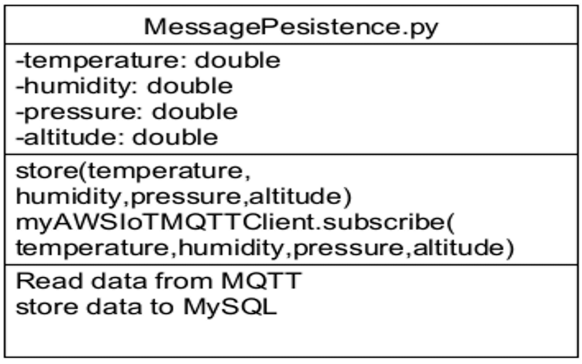


Figure 4.

- Interface design

Raspberry Pi 4 side:

doi.org/10.32388/DNC09I.2

Temp.py	Pressure.py
	
Weather.py	MessagePesistence.py
	
Main.py	QueryWeather.py

 doi.org/10.32388/DNC09I.2

Development Process

In the Figure below, DHT22 installed on the breadboard. The DHT22 is a powerful sensor to record temperature and humility. Some jumper wires are used to connect Raspberry PI 4.

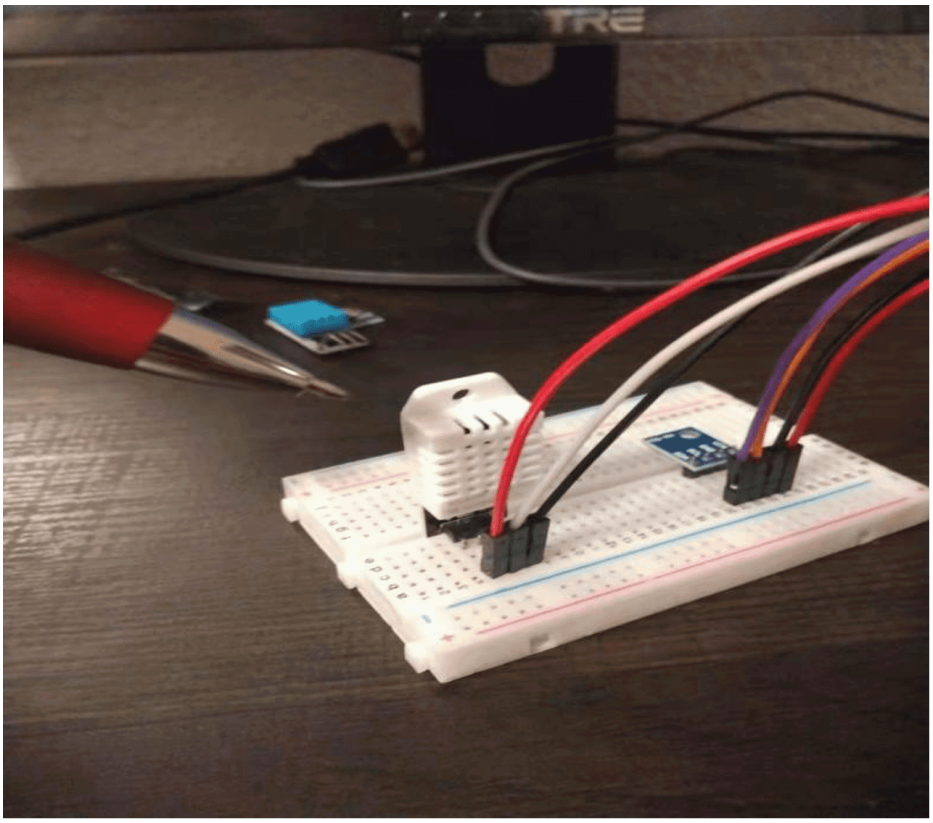


Figure 5.

For connection to Raspberry PI 4, as the below Figure 6, the positive port in the DHT22 must connect to the 5V power port. Negative port must connect to the ground port through jumper wires. Here, data out ports can connect to GPIO4.

doi.org/10.32388/DNC09I.2

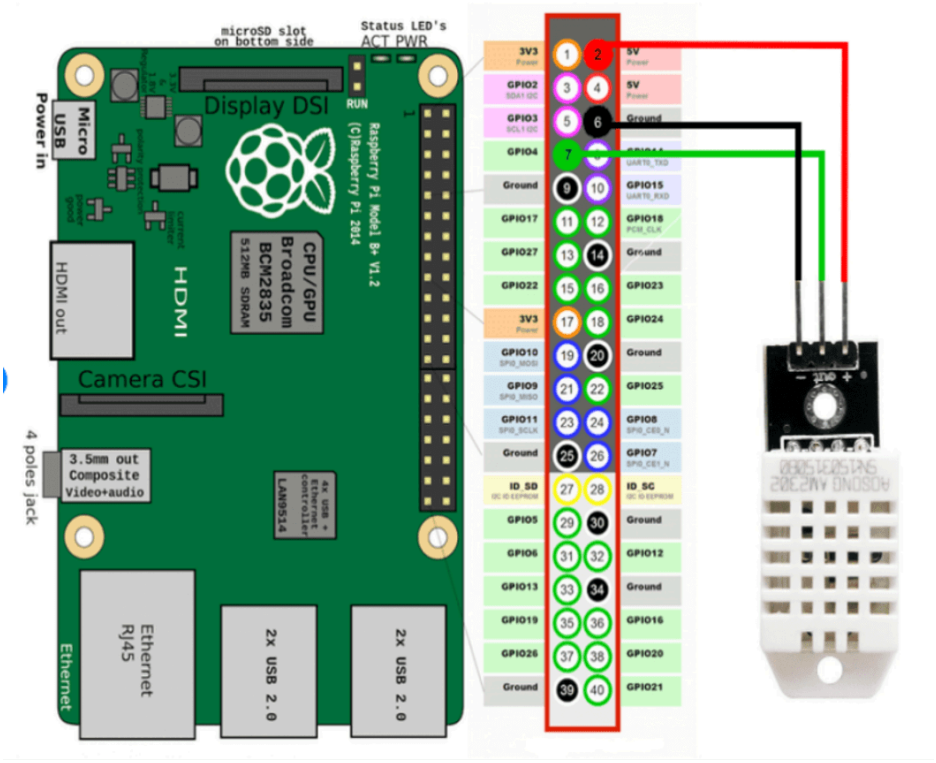


Figure 6.

To test whether DHT22 is working or not, first, install Adafruit_DHT. And then, add the code below to Adafruit_DHT/platform_detect.py

```
elif match.group(1) == 'BCM2711':
   #Pi 4b
   return 3
```

You can write the code as below Figure 7. The code means read data from DHT22 and GPIO4 return in variable humidity and temperature.

```
humidity,temperature = Adafruit_DHT.read_retry(22,4)
```

Figure 7.

```
humidity, temperature = Adafruit_DHT.read_retry(22,4)
```


doi.org/10.32388/DNC09I.2

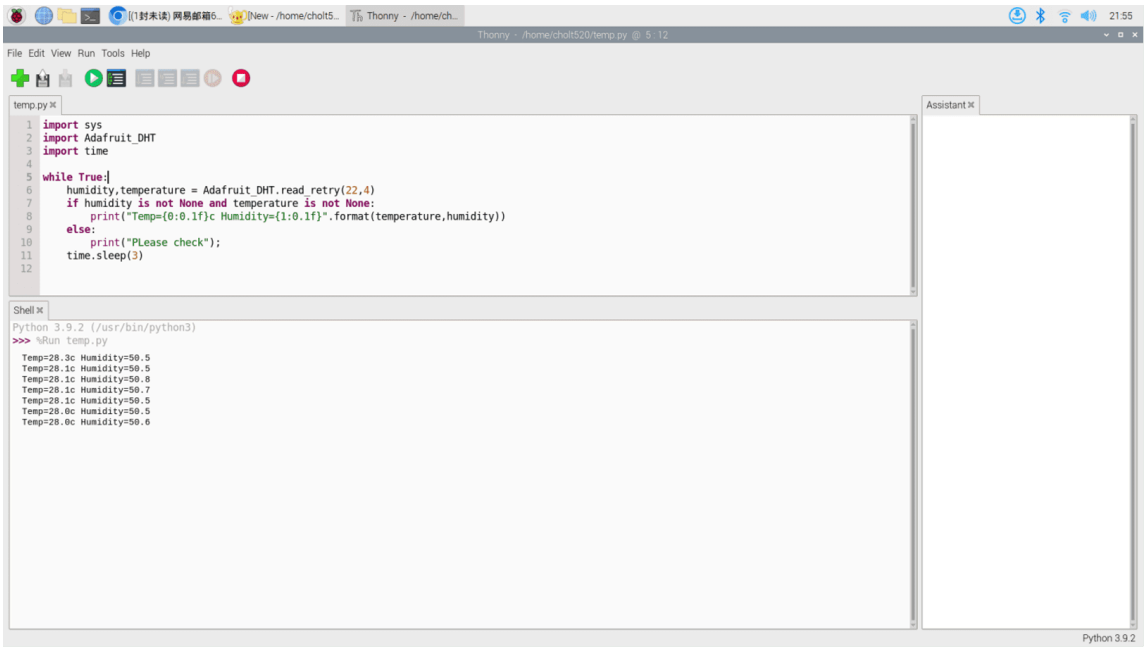


Figure 7.

In the below Figure 8, BMP180 installed on the breadboard. The BMP180 is a powerful sensor to record pressure, altitude, and temp. Some jumper wires are used to connect Raspberry PI 4.

doi.org/10.32388/DNC09I.2

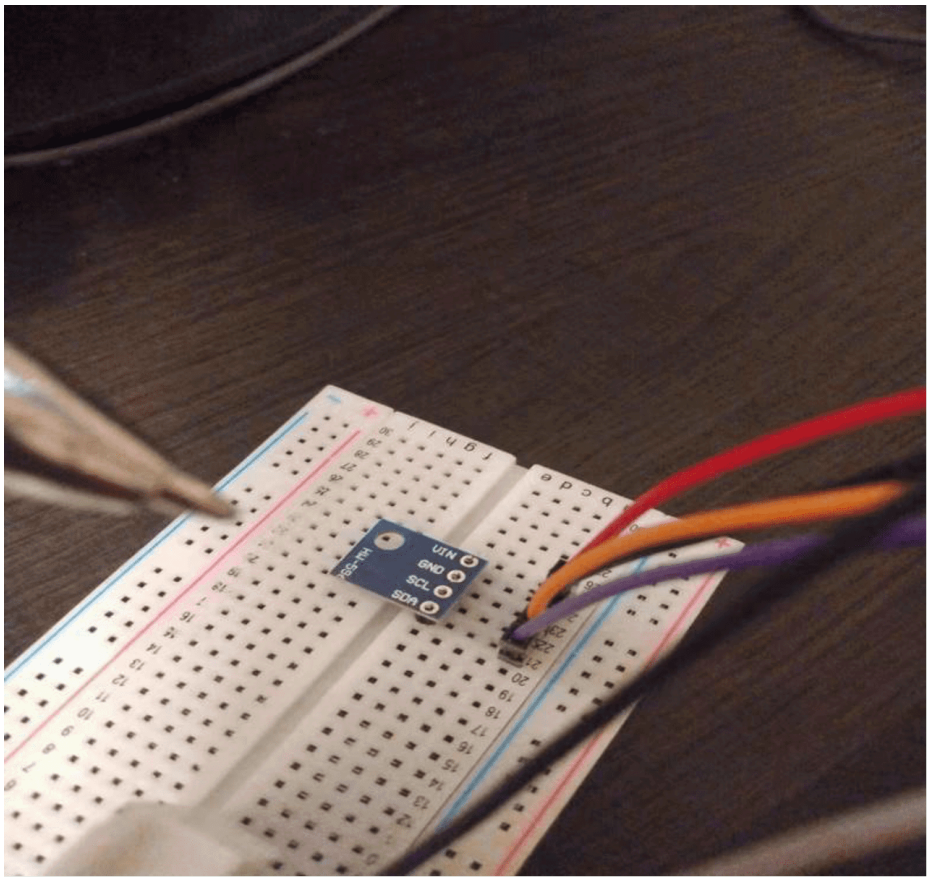


Figure 8.

For connection to Raspberry PI 4, as the below Figure 9, the positive port in the BMP180 must connect to the 3V3 power port. Negative ports must connect to the ground port through jumper wires. Here, data out ports can connect to SCL and SDA ports.

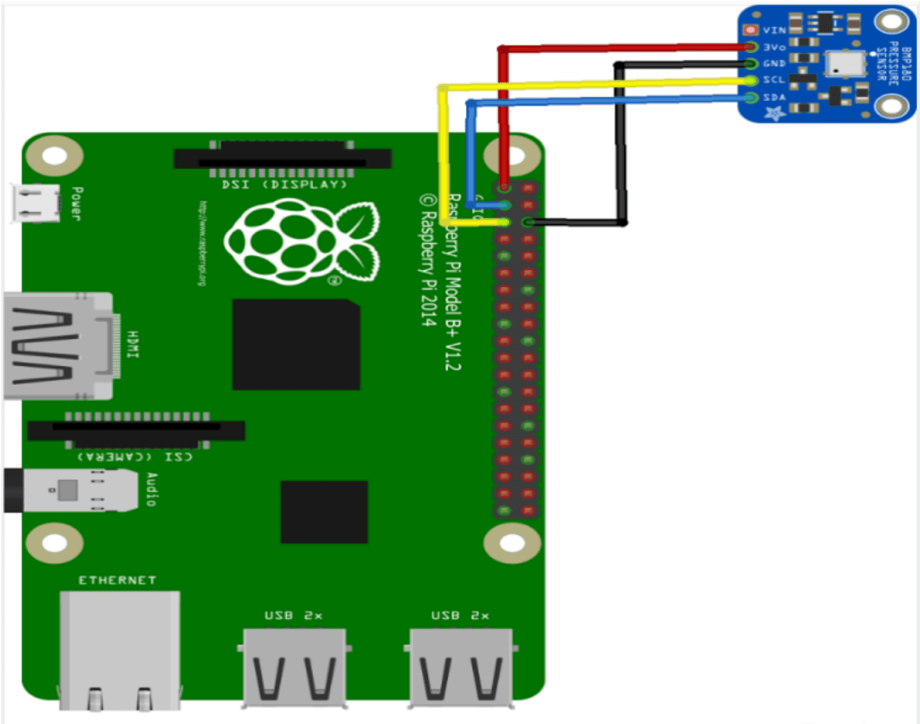


Figure 9.

To test whether the BMP180 is working or not, first, you need to install Adafruit_BMP.

Write the code below Figure 10. The code means to read data from BMP180 return in variable pressure and altitude.

```
temp,pressure,altitude = bmpsensor.readBmp180()
```

temp, pressure, altitude = bmpsensor.readBmp180()

doi.org/10.32388/DNC09I.2

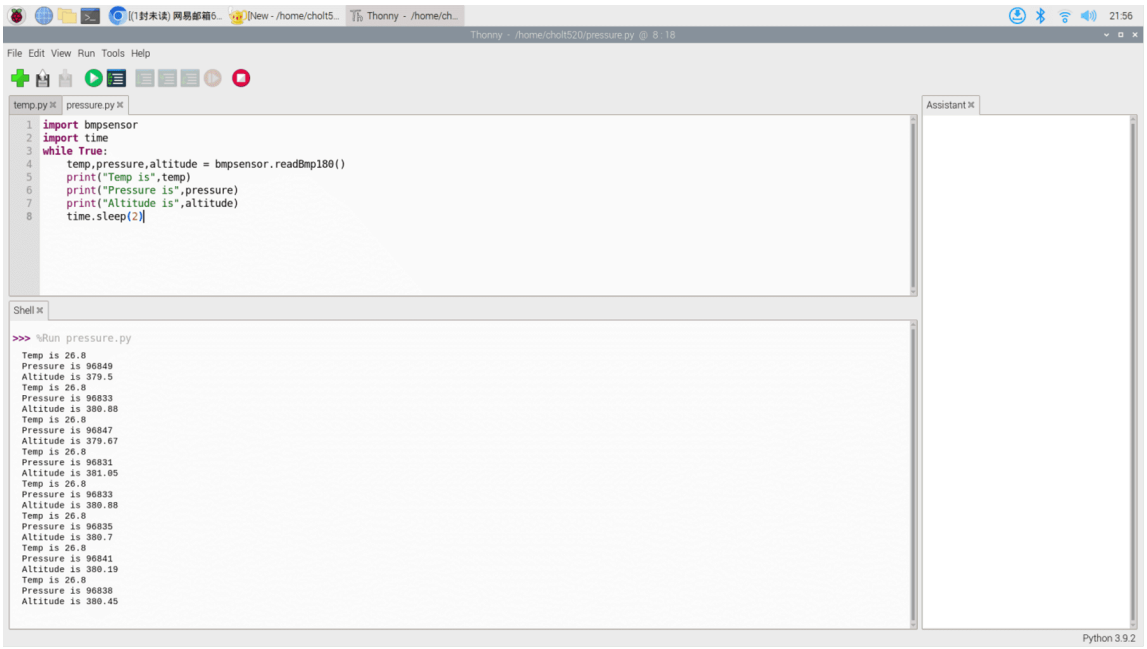


Figure 10.

When finishing the previous work, it can get the IoT hardware like below Figure 11.

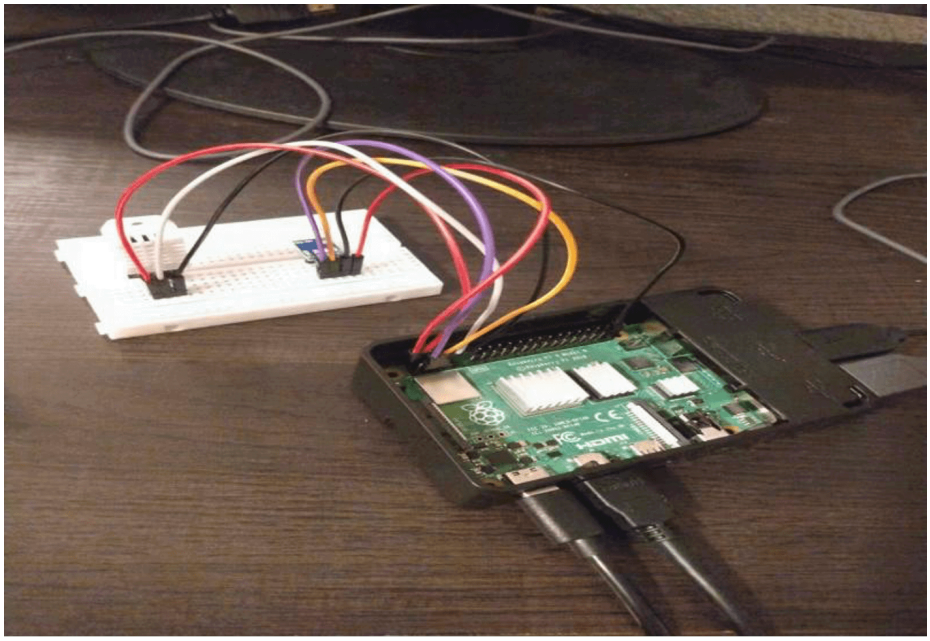


Figure 11.

doi.org/10.32388/DNC09I.2

Here, Raspberry PI 4 needs to send data to AWS-IOT-MQTT. First, you need to install awsiotsdk. The below code will send data from sensors to MQTT through mqtt_connect.publish() function.

```
publish_count = 1
while (publish_count <= message_count) or (message_count == 0):
    humidity,temperature = Adafruit_DHT.read_retry(22,4)
    temp,pressure,altitude = bmpsensor.readBmp180()
    message = "Temp={0:0.1f} Humidity={1:0.1f}".format(temperature,humidity,publish_count)
    message = message + " Pressure=" +str(pressure) + " Altitude=" + str(altitude)
    print("Publishing message to topic '{}': {}".format(message_topic, message))
    message_json = json.dumps(message)
    mqtt_connection.publish(
        topic=message_topic,
        payload=message_json,
        qos=mqtt.QoS.AT_LEAST_ONCE)
    time.sleep(1)
    publish_count += 1
```

```
publish_count = 1
while (publish_count <= message_count) or (message_count == 0):
    humidity, temperature = Adafruit_DHT.read_retry(22,4)
    temp, pressure, altitude = bmpsensor.readBmp180()
    message = "Temp={0:0.1f} Humidity={1:0.1f}".format(temperature, humidity, publish_count)
    message = message + " Pressure=" +str(pressure) + " Altitude=" + str(altitude)
    print("Publishing message to topic '{}': {}".format(message_topic, message))
    message_json = json.dumps(message)
    mqtt_connection.publish(
        topic=message_topic,
        payload=message_json,
        qos=mqtt.QoS.AT_LEAST_ONCE)
    time.sleep(1)
    publish_count += 1
```

Run command:

python3 weather.py --topic weather --ca_file ~/certs/AmazonRootCA1.pem --cert ~/certs/certificate.pem.crt --key ~/certs/private.pem.key --endpoint a38fdsv9in84d7-ats.iot.us-west-2.amazonaws.com

doi.org/10.32388/DNC09I.2

Now, the data is being sent to AWS-MQTT

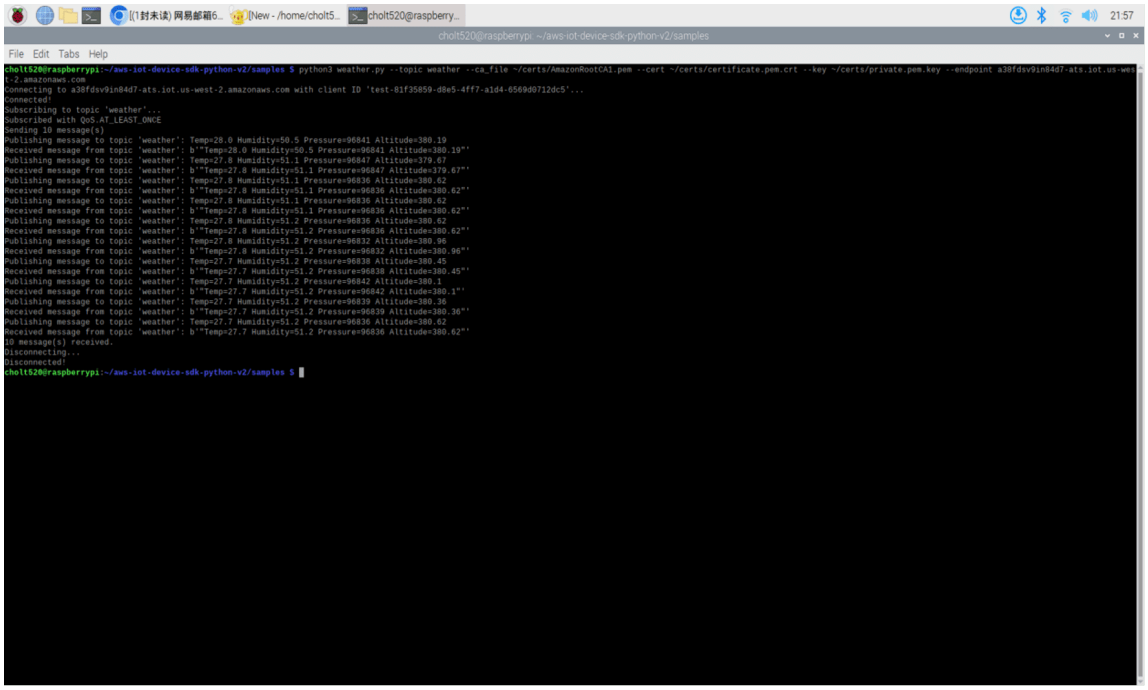


Figure 12.

For the AWS-MQTT console, subscribe to the topic: weather. And then the data comes from IoT devices.

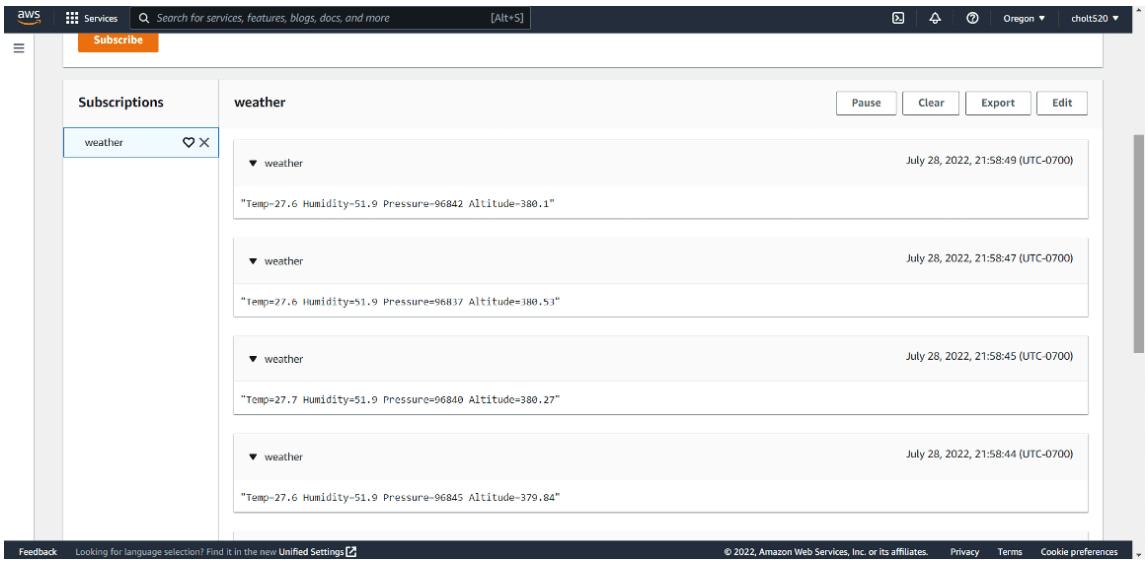


Figure 13.

In the below Figure 14, it shows the Restful server reading data from MQTT. First, put certificate.pem.crt, private.pem.key and AmazonRootCA1.pem into the same folder with code. And then use myAWSIoTMQTTClient.subscribe() to read data from MQTT.

doi.org/10.32388/DNC09I.2

```
ENDPOINT = "a38fdsv9in84d7-ats.iot.us-west-2.amazonaws.com"
CLIENT_ID = "testDevice"
PATH_TO_CERTIFICATE = "certificate.pem.crt"
PATH_TO_PRIVATE_KEY = "private.pem.key"
PATH_TO_AMAZON_ROOT_CA_1 = "AmazonRootCA1.pem"
TOPIC = "weather"
RANGE = 20

myAWSIoTMQTTClient = AWSIoTPyMQTT.AWSIoTMQTTClient(CLIENT_ID)
myAWSIoTMQTTClient.configureEndpoint(ENDPOINT, 8883)
myAWSIoTMQTTClient.configureCredentials(PATH_TO_AMAZON_ROOT_CA_1,
PATH_TO_PRIVATE_KEY, PATH_TO_CERTIFICATE)

myAWSIoTMQTTClient.connect()
myCallbackContainer = CallbackContainer(myAWSIoTMQTTClient)
myAWSIoTMQTTClient.subscribe(TOPIC, 1, myCallbackContainer.messagePersistence);
time.sleep(10)

myAWSIoTMQTTClient.disconnect()
```

```
ENDPOINT = "a38fdsv9in84d7-ats.iot.us-west-2.amazonaws.com"

CLIENT_ID = "testDevice"

PATH_TO_CERTIFICATE = "certificate.pem.crt"

PATH_TO_PRIVATE_KEY = "private.pem.key"

PATH_TO_AMAZON_ROOT_CA_1 = "AmazonRootCA1.pem"

TOPIC = "weather"

RANGE = 20myAWSIoTMQTTClient = AWSIoTPyMQTT.AWSIoTMQTTClient(CLIENT_ID)


myAWSIoTMQTTClient.conFigure Endpoint(ENDPOINT, 8883)

myAWSIoTMQTTClient.conFigure Credentials(PATH_TO_AMAZON_ROOT_CA_1,

PATH_TO_PRIVATE_KEY, PATH_TO_CERTIFICATE)


myAWSIoTMQTTClient.connect()

myCallbackContainer = CallbackContainer(myAWSIoTMQTTClient)

myAWSIoTMQTTClient.subscribe(TOPIC, 1, myCallbackContainer.messagePersistence);

time.sleep(10)


myAWSIoTMQTTClient.disconnect()
```

When get data from MQTT successfully, print them out in the Pycharm console as below Figure 14.

doi.org/10.32388/DNC09I.2

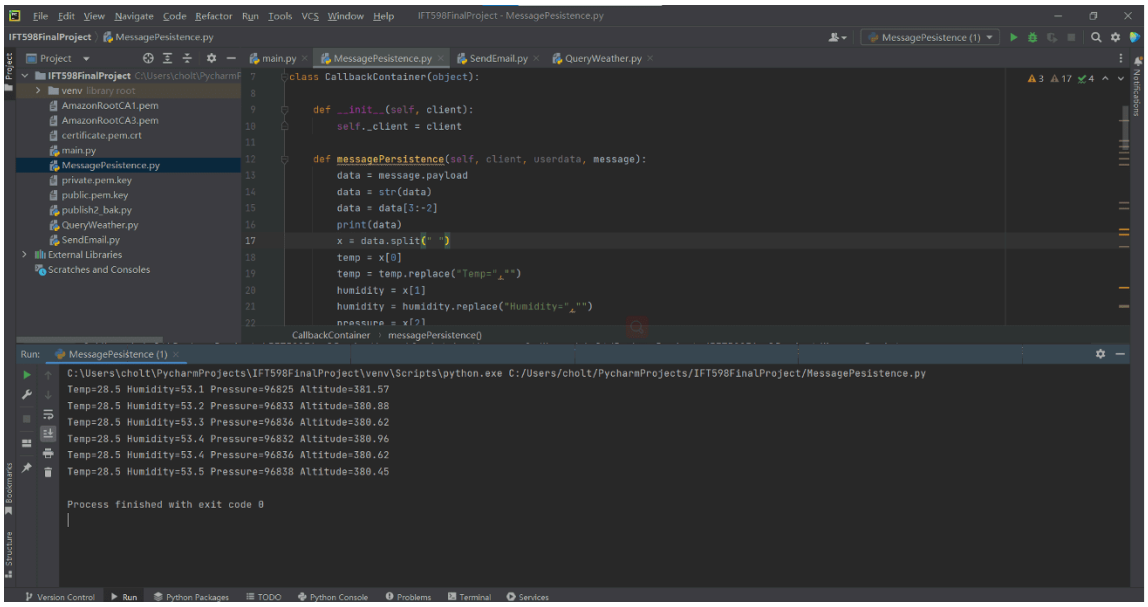


Figure 14.

To store the data into MySQL, you must use the callback function below code.

First, parse the message. This is because the message cannot be directly used without format.

doi.org/10.32388/DNC09I.2

```
def messagePersistence(self, client, userdata, message):
    data = message.payload
    data = str(data)
    data = data[3:-2]
    print(data)
    x = data.split(" ")
    temp = x[0]
    temp = temp.replace("Temp=","")
    humidity = x[1]
    humidity = humidity.replace("Humidity=","")
    pressure = x[2]
    pressure = pressure.replace("Pressure=","")
    altitude = x[3]
    altitude = altitude.replace("Altitude=","")
    ts = time.time()
    reportTime = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
    location = "300 E main street"
```


```
def messagePersistence(self, client, userdata, message):
    data = message.payload
    data = str(data)
    data = data[3:-2]
    print(data)
    x = data.split(" ")
    temp = x[0]
    temp = temp.replace("Temp=","")
    humidity = x[1]
    humidity = humidity.replace("Humidity=","")
    pressure = x[2]
    pressure = pressure.replace("Pressure=","")
    altitude = x[3]
    altitude = altitude.replace("Altitude=","")
    ts = time.time()
    reportTime = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
    location = "300 E main street"
```


Then, use the code below to store data into MySQL database.

```
try:
   connection = mysql.connector.connect(host='localhost',
                                        database='ift598finalprojectweather',
                                        user='root',
                                        password='')

   cursor = connection.cursor()
   query = "INSERT INTO weather(temperature,
humidity,altitude,pressure,reportTime,location,city,country,state) " \
           "VALUES(%s,%s,%s,%s,%s,%s,%s,%s,%s)"
   args = (temp, humidity, pressure, altitude, reportTime,location,city,country,state)
   cursor.execute(query, args)
   connection.commit()

except mysql.connector.Error as e:
   print("Error reading data from MySQL table", e)
finally:
   if connection.is_connected():
      connection.close()
      cursor.close()
```

```
try:
   connection = mysql.connector.connect(host='localhost',
                                        database='ift598finalprojectweather',
                                        user='root',
                                        password='')

   cursor = connection.cursor()
   query = "INSERT INTO weather(temperature, humidity, altitude, pressure, reportTime, location, city,
country, state) " \
           "VALUES(%s,%s,%s,%s,%s,%s,%s,%s,%s)"
   args = (temp, humidity, pressure, altitude, reportTime, location, city, country, state)
   cursor.execute(query, args)
   connection.commit()

except mysql.connector.Error as e:
   print("Error reading data from MySQL table", e)
finally:
   if connection.is_connected():
```


doi.org/10.32388/DNC09I.2

```
    connection.close()
    cursor.close()
```

When successful, the data in the weather table is below Figure 15.

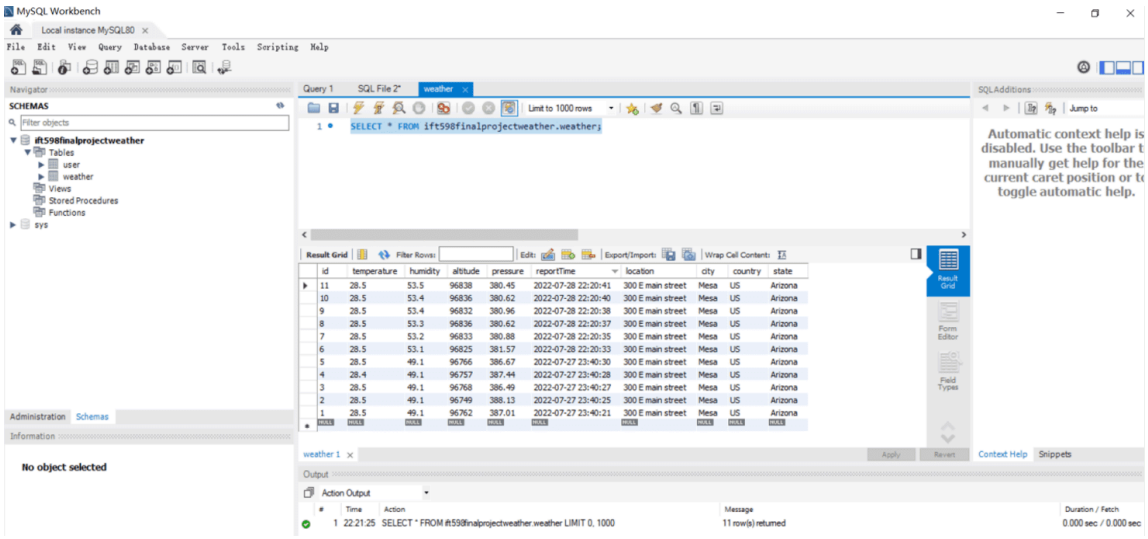


Figure 15.

In the Figure below, install fastAPI as a restful server. If you install successfully, use command: uvicorn main:app to start the Restful server.

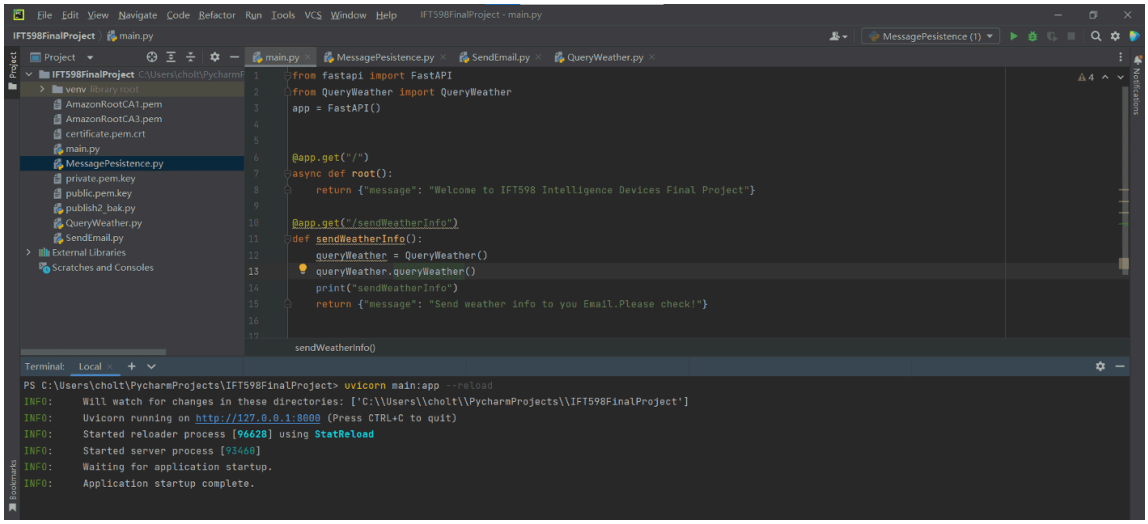


Figure 16.

doi.org/10.32388/DNC09I.2

Open browser, Go to localhost:8000. the index page as below Figure 17.

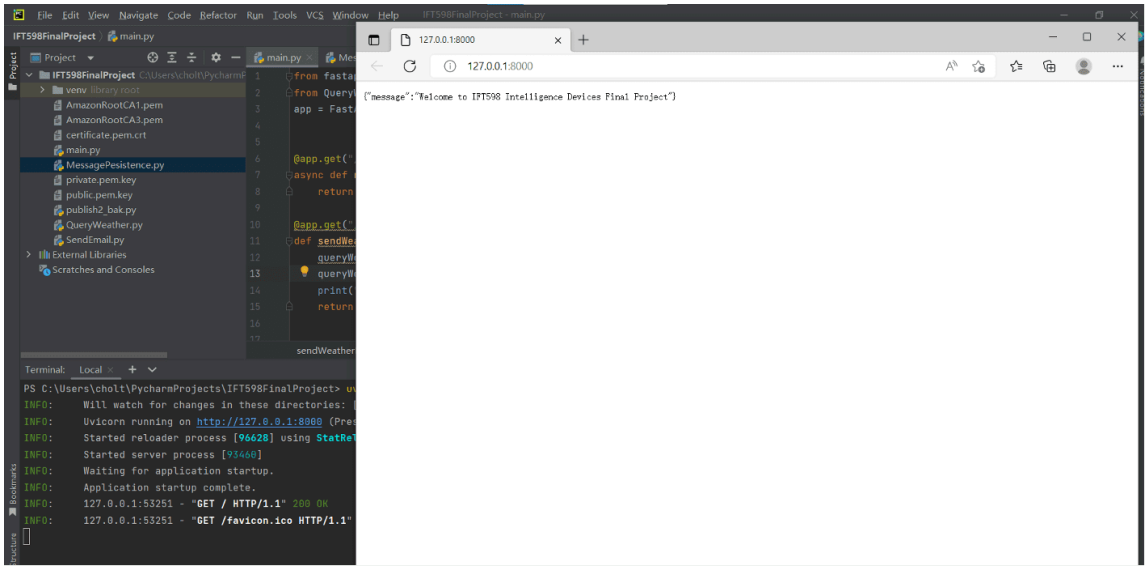


Figure 17.

To send emails to users, first we need to query data from MySQL, the below code will do this.

doi.org/10.32388/DNC09I.2

```
connection = mysql.connector.connect(host='localhost',
                                     database='ift598finalprojectweather',
                                     user='root',
                                     password='cholt666')

sql_select_Query = "SELECT * FROM weather"
cursor = connection.cursor()
cursor.execute(sql_select_Query)
# get all records
records = cursor.fetchall()
```


```
connection = mysql.connector.connect(host='localhost',
                                     database='ift598finalprojectweather',
                                     user='root',
                                     password='cholt666')


sql_select_Query = "SELECT * FROM weather"
cursor = connection.cursor()
cursor.execute(sql_select_Query)
# get all records
records = cursor.fetchall()
```


And then, invoke the send email function as below code:

doi.org/10.32388/DNC09I.2

```
def __init__(self):
    self.port = 465
    self.smtp_server_domain_name = "smtp.gmail.com"
    self.sender_mail = "wj4507657@gmail.com"
    self.password = "zigfttbjyteetsro"

def send(self, emails, subject, content):
    ssl_context = ssl.create_default_context()
    service = smtplib.SMTP_SSL(self.smtp_server_domain_name, self.port, context=ssl_context)
    service.login(self.sender_mail, self.password)

    for email in emails:
        result = service.sendmail(self.sender_mail, email, f"Subject: {subject}\n{content}")
    service.quit()
```

```
def ___init___(self):
    self.port = 465
    self.smtp_server_domain_name = "smtp.gmail.com"
    self.sender_mail = "wj4507657@gmail.com"
    self.password = "zigfttbjyteetsro"

def send(self, emails, subject, content):
    ssl_context = ssl.create_default_context()
    service = smtplib.SMTP_SSL(self.smtp_server_domain_name, self.port, context=ssl_context)
    service.login(self.sender_mail, self.password)

    for email in emails:
        result = service.sendmail(self.sender_mail, email, f"Subject: {subject}\n{content}")
    service.quit()
```

The restful server code as below:

doi.org/10.32388/DNC09I.2

```python
@app.get("/sendWeatherInfo")
def sendWeatherInfo():
    queryWeather = QueryWeather()
    queryWeather.queryWeather()
    print("sendWeatherInfo")
    return {"message": "Send weather info to you Email.Please check!"}
```


@app.get("/sendWeatherInfo")

def sendWeatherInfo():

queryWeather = QueryWeather()

queryWeather.queryWeather()

print("sendWeatherInfo")

return {"message": "Send weather info to you Email.Please check!"}

Open browser, Go to localhost:8000/sendWeatherInfo. the return page as below Figure 18.

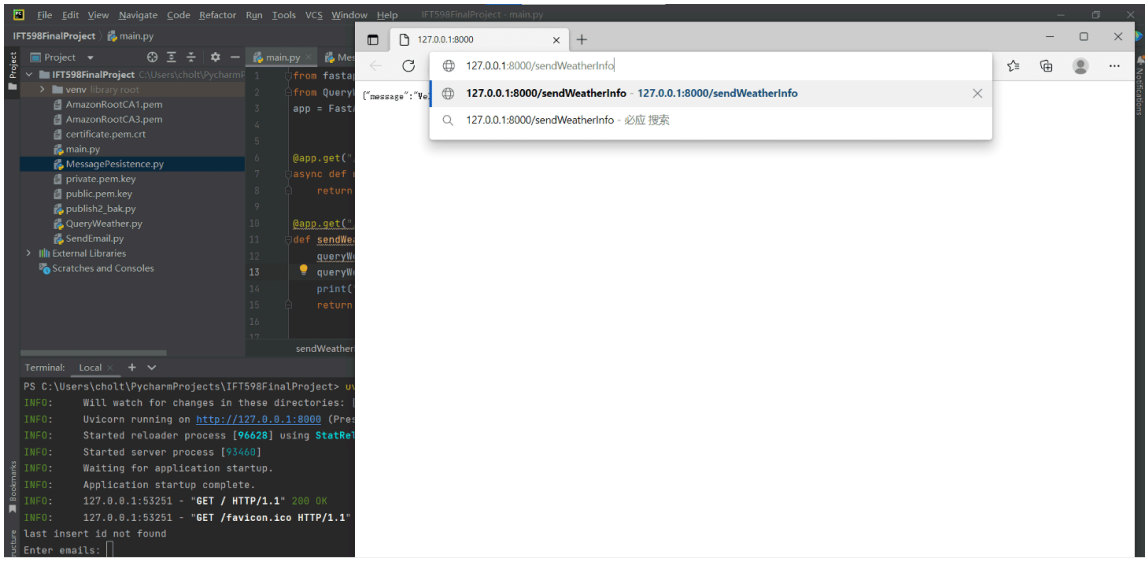


Figure 18.

For testing whether the email was sent successfully or not, open Gmail, the email notification as below Figure 19.

doi.org/10.32388/DNC09I.2

Figure 19.

After sending an email or phone message to notify monitoring personnel, monitor analyst can receive the fire's location, including latitude and longitude. Monitor analyst deploys a detection drone to the specified coordinates to capture images and transmit them back for analysis. Advanced drones, such as the DJI Mavic 3 Thermal Enterprise, can be utilized to determine whether wildfire is present.

The DJI Mavic 3 Thermal Enterprise is a professional drone designed for wildfire detection, emergency response, and industrial inspections. Equipped with a 640 × 512 px thermal sensor and a 48MP visual camera, it enables precise heat detection and high-resolution imaging. With 20x digital zoom, 56x hybrid zoom, and an optional RTK module for centimeter-level accuracy, it provides detailed monitoring capabilities. Its 45-minute flight time and omnidirectional obstacle sensing ensure extended and safe operations, while real-time live streaming and data transmission support rapid decision-making. This makes it an invaluable tool for detecting and analyzing wildfires efficiently.

The DJI Mavic 3 Thermal Enterprise can be seamlessly integrated into a wildfire detection system to enhance real-time monitoring and rapid response. By deploying drones immediately after receiving wildfire alerts, whether from satellite data, IoT sensor networks, or emergency reports—firefighters and analysts can quickly assess fire locations. The drone's thermal imaging detects heat anomalies, while its high-resolution camera captures visual confirmation. Coupled with AI-powered analysis, the system can automatically identify fire intensity, spread direction, and potential risks. Its RTK precision positioning ensures accurate mapping, while live data streaming enables immediate

doi.org/10.32388/DNC09I.2

decision-making. This integration significantly improves wildfire detection efficiency, reducing response times and minimizing damage.

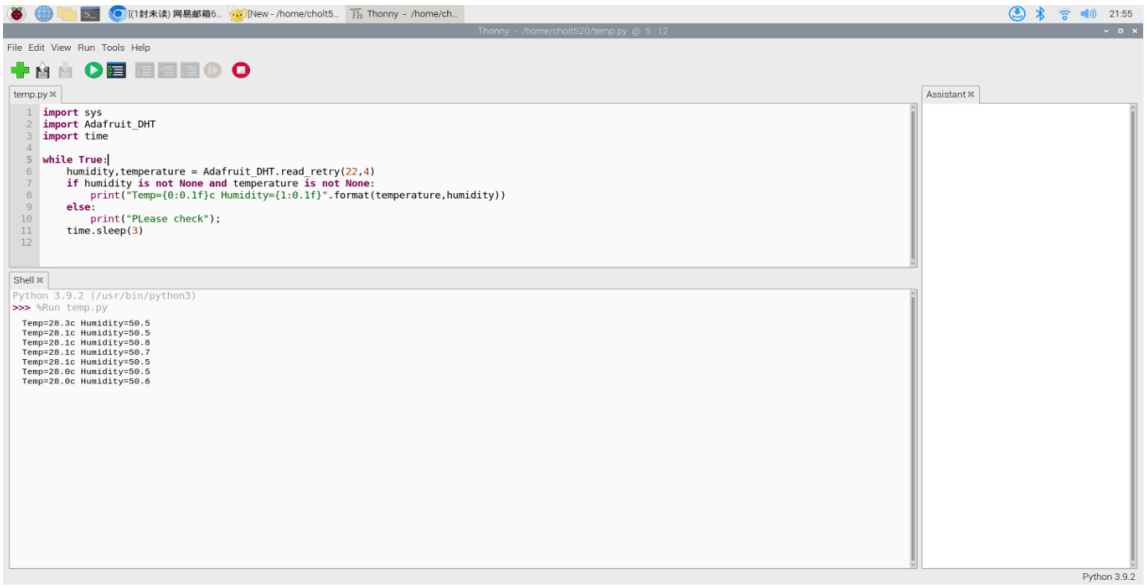
Figure 20.

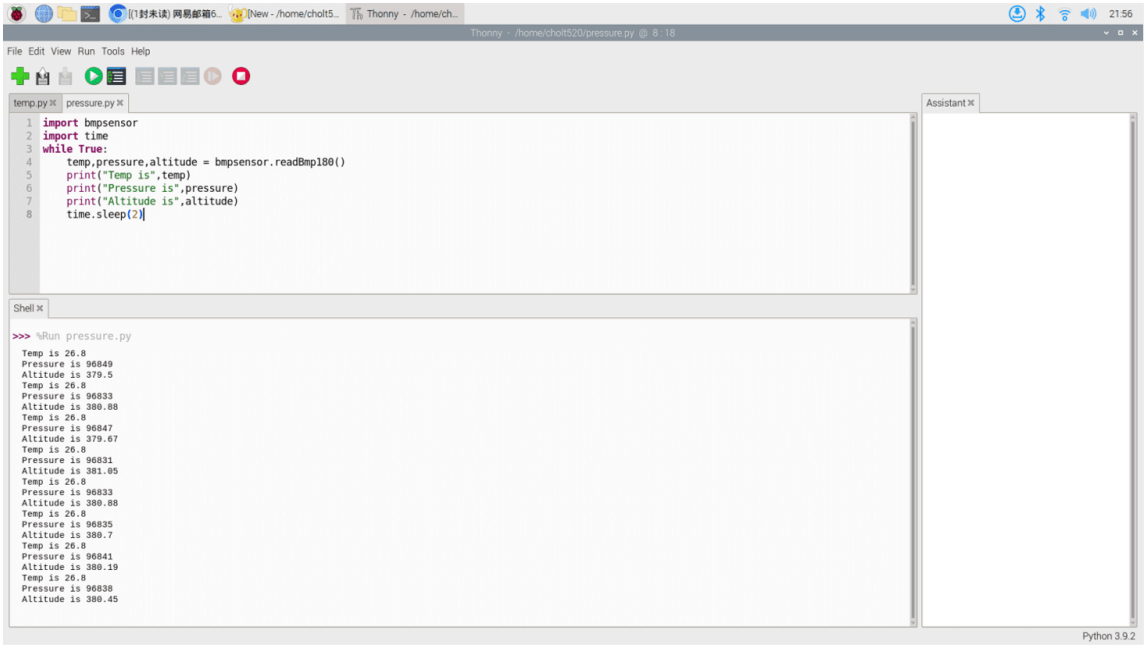
Figure 21.

Testing and Result

For test DHT22 and BMP180, the below code can be used as the test part. The sample way to test temperature and humidity change. blow your breath to DHT22, as a result, the change immediately happens.

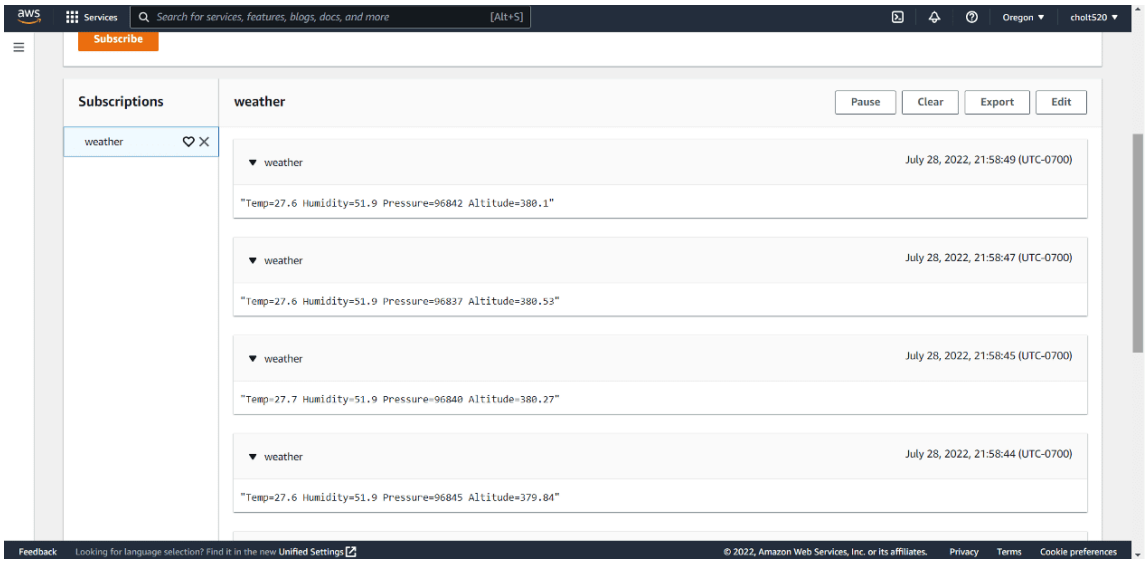
doi.org/10.32388/DNC09I.2



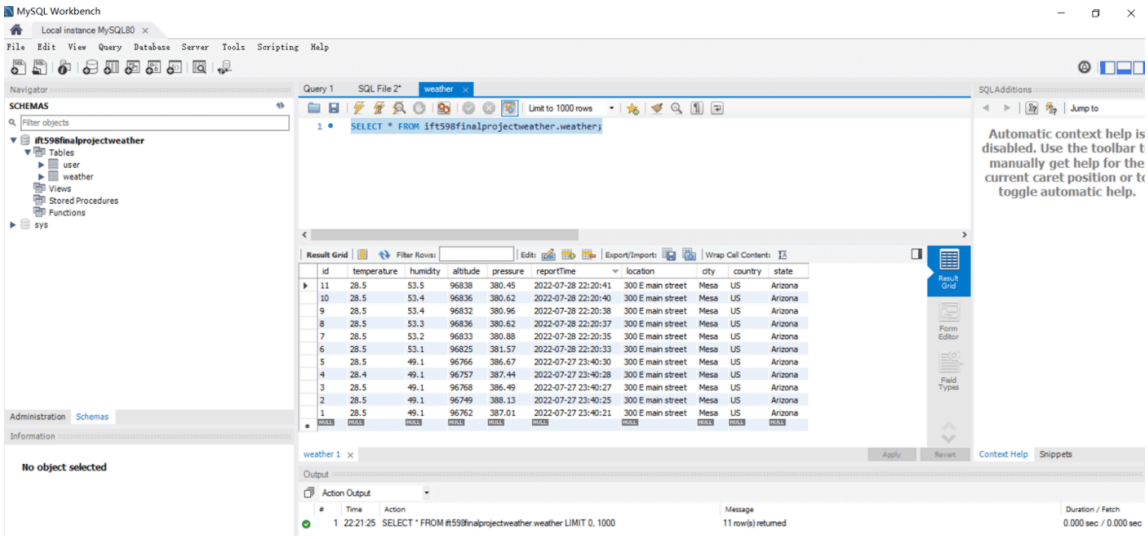


Login into AWS MQTT console, subscribe to the topic. test the data whether it is transferred successfully.

doi.org/10.32388/DNC09I.2

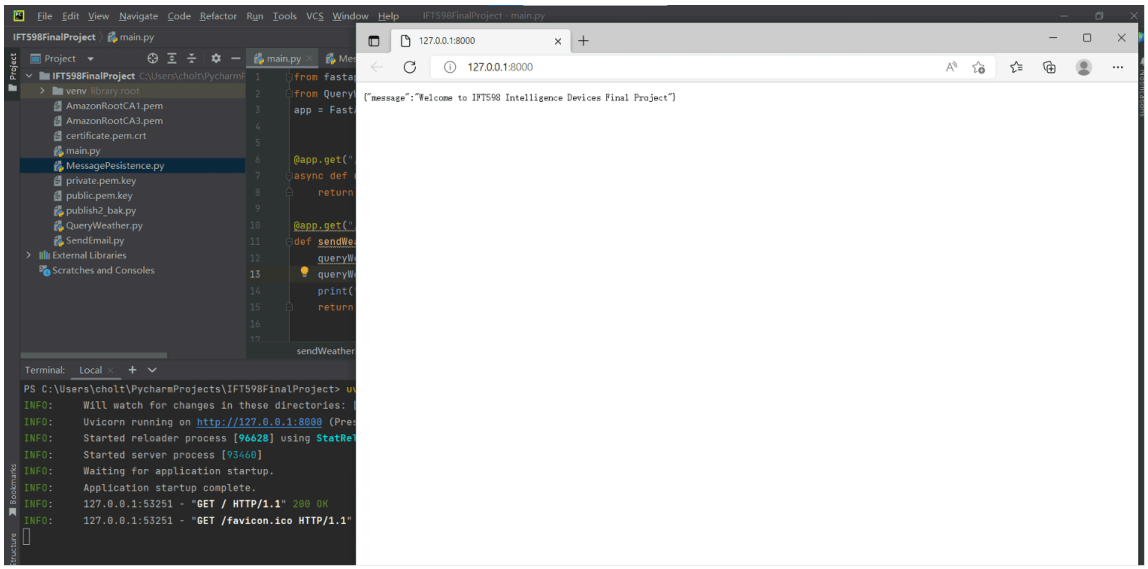


Using MySQL Workbench can easily see the data whether it is stored successfully.



Open browser, Go to localhost:8000. It can test the restful server to see whether it is installed successfully.

doi.org/10.32388/DNC09I.2



Open your Gmail account, email notification which test the email was sent successfully or not.

Weather monitor system Integration testing:

To test the performance of the IoT weather monitoring system, sensors can be placed in various locations. For instance, the system can be set up indoors to measure temperature, humidity, and pressure. Initially, the system can be tested over a 24-hour period, then extended to one week, one month, and so on, to assess its long-term reliability.

Testing should also include evaluating the portable power supply. If deploying the IoT system in a remote field, it is essential to determine how long the system can operate without losing power and how long the portable power can sustain the system. Additionally, it is crucial to test the system's

doi.org/10.32388/DNC09I.2

scalability by deploying hundreds or thousands of sensors to ensure it continues to function correctly under increased load.

The DHT22 sensor, with its wide temperature range of -40 to 80°C, can operate effectively in harsh environments, making it suitable for testing in various conditions.

Summary and Conclusion

This project is designed and developed for IoT weather monitoring, utilizing weather parameters such as temperature, humidity, pressure, and altitude. The IoT weather monitoring system collects real-time data and sends it to a server for analysis and notification purposes.

During the design process, a list of requirements will help readers understand what is needed for the system. Block diagrams and data flow charts provide a deeper understanding of the entire process. The project interface displays all parameters and functions to developers, aiding them in easily building their own IoT system.

The development process details the core code of the IoT system, explaining how to send data from sensors and receive data from RESTful servers. The testing and results section describes how to test the entire system to ensure it works properly and provides strategies for troubleshooting errors. Drones have become essential in rescue operations, especially for firefighting. With advancements in technology, they can monitor disasters in real time, scout hazardous environments, and assist firefighters in rescue efforts. Modern building complexities and unpredictable fire conditions make it difficult for firefighters to assess situations safely. Drones, as unmanned aerial vehicles (UAVs), provide remote-controlled reconnaissance, improving decision-making and safety. Unlike manned aircraft, drones require a strict control system and specialized equipment to complete tasks, making them part of a larger unmanned aircraft system (UAS). Their use in fire rescue enhances situational awareness, enables effective strategies, and significantly improves firefighting efficiency.

This IoT weather monitoring system is designed and developed for weather prediction and forest fire prevention. In California, accurate weather monitoring is crucial for residents' safety and preparedness.

References

1. a, b *Bahga A, Madisetti V (2014). Internet of Things: A hands-on approach. VPT.*

Declarations

Funding: No specific funding was received for this work.

Potential competing interests: No potential competing interests to declare.