Environment Monitoring System

Table of Contents

- Introduction
- List of Components
- Hardware Requirements
- Software Requirements
- Setting up Thonny for ESP32
- Programming ESP32 with MicroPython
- Web Server Setup with MicroWebSrv
- MQ Sensor for Air Quality
- DHT Sensor for Temperature and Humidity
- Creating a Real-Time Web Dashboard
- Testing and Debugging
- Deployment and Monitoring

1. Introduction

Project Overview: The Environment Monitoring System is designed to measure and monitor environmental parameters such as air quality, temperature, and humidity using an ESP32 microcontroller. This system enables real-time data collection and remote access to environmental data, offering a cost-effective solution for various applications, including indoor air quality monitoring and climate control.

Purpose and Objectives: The primary purpose of this project is to create a system that can continuously measure and report environmental data, enabling users to make informed decisions regarding their surroundings. The objectives include real-time data monitoring and remote access to data through a web interface, which provides user-friendly visualization.

List of Components:

- ESP32 microcontroller
- MQ sensor
- DHT11/DHT22 temperature and humidity sensor
- Breadboard
- Jumper wires
- Resistors
- Power supply

2. Hardware Requirements

Key hardware components include the **ESP32** microcontroller, **MQ sensors** for air quality measurement, **DHT sensors** for temperature and humidity, and an appropriate **power supply**. Proper **wiring and connections** are crucial to ensure accurate data collection. In some cases, an **enclosure** may be necessary for outdoor deployments.

3. Software Requirements

The software stack comprises **MicroPython** for ESP32 programming, the **Thonny IDE** for code development, the **MicroWebSrv** library for web server functionality, and web technologies like **HTML and JavaScript** for creating the web interface. End-users need a **web browser** to access and visualize data.

4. Setting up Thonny for ESP32

Thonny installation on the computer is the starting point. It's an integrated development environment for MicroPython and simplifies code writing and uploading. Next, **MicroPython firmware** is flashed onto the ESP32 via Thonny. A **serial connection** is configured to enable code uploading and interactive sessions for testing and debugging.

5. Programming ESP32 with MicroPython

Developers begin by writing a simple MicroPython program for the ESP32. This program serves as a foundation for more complex applications. They learn and use **basic MicroPython commands** to control hardware components and upload scripts to the ESP32 through Thonny.

6. Web Server Setup with MicroWebSrv

Understanding the **MicroWebSrv** library is essential for setting up a web server on the ESP32. Developers configure the server with specific routes for handling different types of web requests. This involves defining how the server responds to specific **HTTP request methods** such as GET and POST. **Route handling** and the creation of custom route handlers are key components of web server setup.

7. MQ Sensor for Air Quality

MQ sensors are introduced as crucial components for **air quality measurement** and pollution detection. The documentation explains the role of MQ sensors, provides a guide to **wiring the sensor**, and includes Python code for **reading data from the sensor**. The actual code may vary depending on the MQ sensor model used.

8. DHT Sensor for Temperature and Humidity

DHT sensors are chosen for **temperature and humidity monitoring**, and the documentation elaborates on their significance. Proper **wiring and connection** methods are explained to ensure precise data collection. Additionally, Python code for reading temperature and humidity data is provided, focusing on data accuracy.

9. Creating a Real-Time Web Dashboard

Developing the **web-based dashboard** for real-time data visualization is a critical aspect. The documentation discusses the principles of **Server-Sent Events (SSE)** for real-time updates and guides developers through HTML and JavaScript for **web interface development**. This section emphasizes user-friendliness and data responsiveness.

10. Testing and Debugging

To guarantee the accuracy of sensor data, developers carry out **data accuracy testing**. This process helps identify and rectify any discrepancies in data. The documentation also covers **common issues** that may arise during testing and provides guidance on resolving them.

11. Deployment and Monitoring

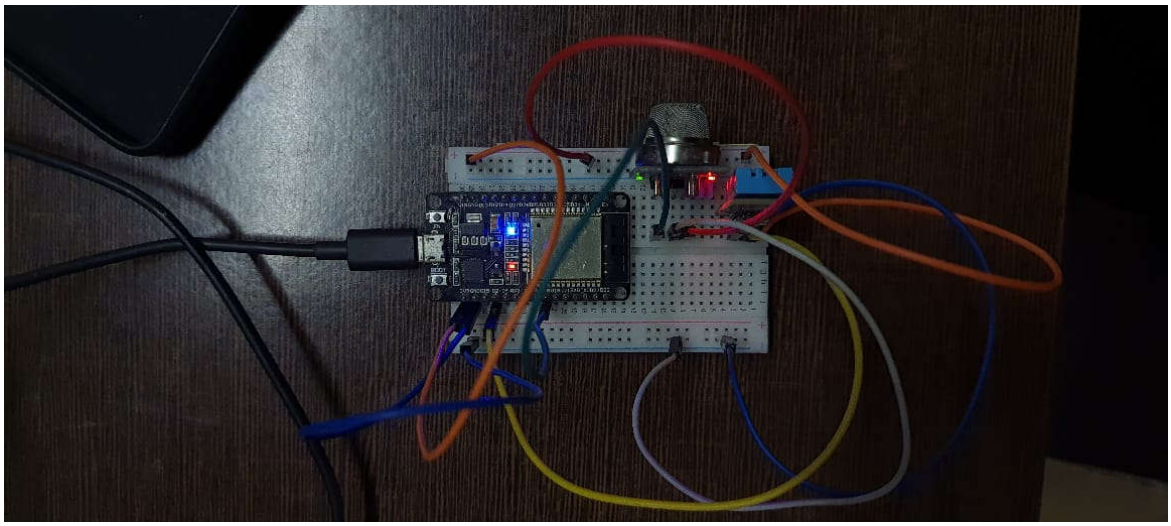
The project's final stages involve **preparing for deployment**. Once testing is complete, the system is ready for field deployment, which may entail installing it in the intended environment. This phase may include hardware setup and network configurations to ensure reliable and continuous data collection.

System, providing an insight to hardware and software setup, coding, server configuration, and data display.

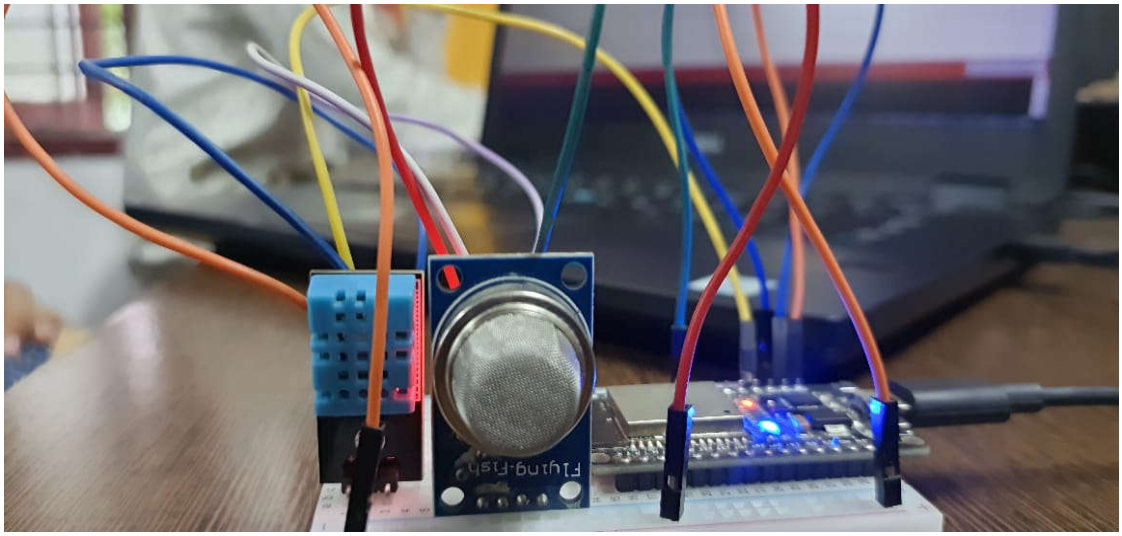
The Environment Monitoring System provides a detailed guide to setting up and programming the ESP32 microcontroller to measure and monitor air quality, temperature, and humidity. With a robust hardware and software foundation, this system offers a cost-effective solution for a range of applications, from indoor air quality monitoring to climate control in various environments. The real-time web interface ensures that end-users can easily access and visualize the collected data, enabling informed decision-making.

DESIGN:

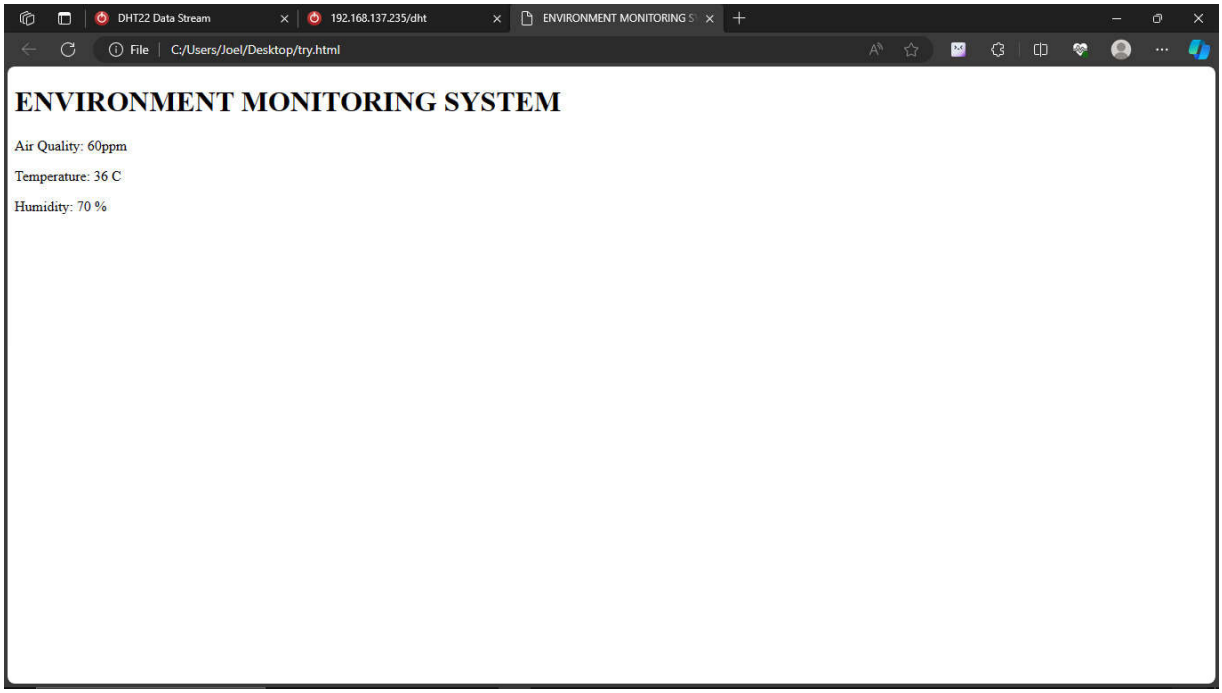
TOP VIEW:



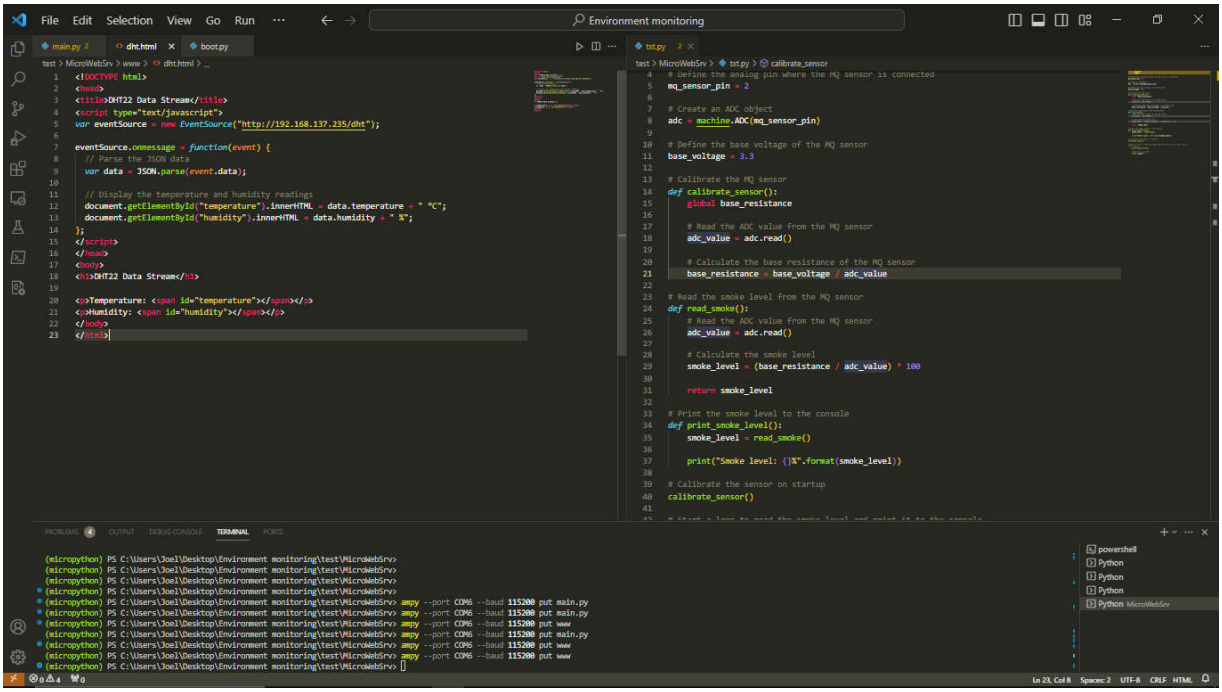
SIDE VIEW:



SERVER:


DHT22 Data Stream
192.168.137.235/dht
ENVIRONMENT MONITORING
File
C:/Users/Joel/Desktop/try.html
ENVIRONMENT MONITORING SYSTEM
Air Quality: 60ppm
Temperature: 36 C
Humidity: 70 %

CODE:


File Edit Selection View Go Run
Environment monitoring
main.py 2
dht.html
boot.py
txt.py 2
test > MicroWebSrv > www > dht.html > ...
test > MicroWebSrv > txt.py > calibrate_sensor
<!DOCTYPE html>
<html>
<head>
<title>DHT22 Data Stream</title>
<script type="text/javascript">
var eventSource = new EventSource("http://192.168.137.235/dht");
eventSource.onmessage = function(event) {
// Parse the JSON data
var data = JSON.parse(event.data);
// Display the temperature and humidity readings
document.getElementById("temperature").innerHTML = data.temperature + " °C";
document.getElementById("humidity").innerHTML = data.humidity + " %";
};
</script>
</head>
<body>
<h1>DHT22 Data Stream</h1>
Temperature:
Humidity:
</body>
</html>
# Define the analog pin where the MQ sensor is connected
mq_sensor_pin = 2
# Create an ADC object
adc = machine.ADC(mq_sensor_pin)
# Define the base voltage of the MQ sensor
base_voltage = 3.3
# Calibrate the MQ sensor
def calibrate_sensor():
global base_resistance
# Read the ADC value from the MQ sensor
adc_value = adc.read()
# Calculate the base resistance of the MQ sensor
base_resistance = base_voltage / adc_value
# Read the smoke level from the MQ sensor
def read_smoke():
# Read the ADC value from the MQ sensor
adc_value = adc.read()
# Calculate the smoke level
smoke_level = (base_resistance / adc_value) * 100
return smoke_level
# Print the smoke level to the console
def print_smoke_level():
smoke_level = read_smoke()
print("Smoke level: {}%".format(smoke_level))
# Calibrate the sensor on startup
calibrate_sensor()
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS
(micropython) PS C:\Users\Joel\Desktop\Environment monitoring\test\MicroWebSrv>
(micropython) PS C:\Users\Joel\Desktop\Environment monitoring\test\MicroWebSrv> ampy --port COM6 --baud 115200 put main.py
(micropython) PS C:\Users\Joel\Desktop\Environment monitoring\test\MicroWebSrv> ampy --port COM6 --baud 115200 put www
(micropython) PS C:\Users\Joel\Desktop\Environment monitoring\test\MicroWebSrv> ampy --port COM6 --baud 115200 put main.py
(micropython) PS C:\Users\Joel\Desktop\Environment monitoring\test\MicroWebSrv> ampy --port COM6 --baud 115200 put www
(micropython) PS C:\Users\Joel\Desktop\Environment monitoring\test\MicroWebSrv>
powershell
Python
Python
Python
Python MicroWebSrv
Ln 23, Col 8 Spaces: 2 UTF-8 CRLF HTML