Mestrado Informática

Pós-Graduação - Data Science and Digital Transformation

Instituto Politécnico Portalegre

Internet das Coisas e Sistemas Embebidos

Projeto Final – Sensor de Temperatura para IoT

João Alex Arruda da Silva

1 Introduction

This academic report explores a project that uses an ESP32 microcontroller and a DHT11 sensor to monitor temperature and humidity in real-time. The report highlights the importance of precise environmental data collection and the use of advanced techniques, such as moving average calculations, to improve accuracy.

The project focuses on the ESP32, a versatile and capable device that seamlessly interacts with a temperature and humidity sensor. Although the DHT11 sensor has limitations, it plays a pivotal role in the hardware amalgamation and contributes to the system's overall functionality.

This project incorporates a moving average calculation methodology to mitigate the impact of transient fluctuations in the collected data. The aim is to provide a more stable and accurate representation of the environmental conditions. The meticulous approach ensures that the sensor readings offer a comprehensive view of temperature and humidity trends over time.

The data collected is transmitted to a Mosquitto broker, which provides a scalable and efficient means of communication within an IoT ecosystem. The ESP32 communicates with the broker using the MQTT (Message Queuing Telemetry Transport) protocol, establishing a robust data transfer mechanism.

To ensure a user-friendly and informative interface, the project includes a local web server data visualization. This aspect of the project highlights the significance of presenting information in a clear and concise manner, catering to both technical and non-technical stakeholders.

This project can also be found at a remote repository on GitHub.

2 Objectives

The project aims to use the ESP32 microcontroller for environmental monitoring by integrating a temperature and humidity sensor, the DHT11. Another important objective is to implement a sophisticated algorithm that calculates the moving averages of temperature and humidity data, which will enhance the precision of the acquired information. In addition, the use of the Mosquitto MQTT broker is essential to the project, ensuring efficient and scalable data communication. Finally, the project aims to provide clear visualizations using a web server created locally.

3 Setting up the environment

In this section, we'll go through the steps done such as setting up the IDE, installing the MQTT Broker and all other steps needed to be able to replicate the results. It's important to say that this is not the only possible way to do this, but one of the easiest.

3.1 VSCode

The operating system used for this is Fedora, which is a Linux based system. It's possible to accomplish this in any operating system, though.

The first step is to install and setup either the VSCode IDE or the Arduino IDE. VSCode is the one used in project, and it has several advantages over Arduino. It's much

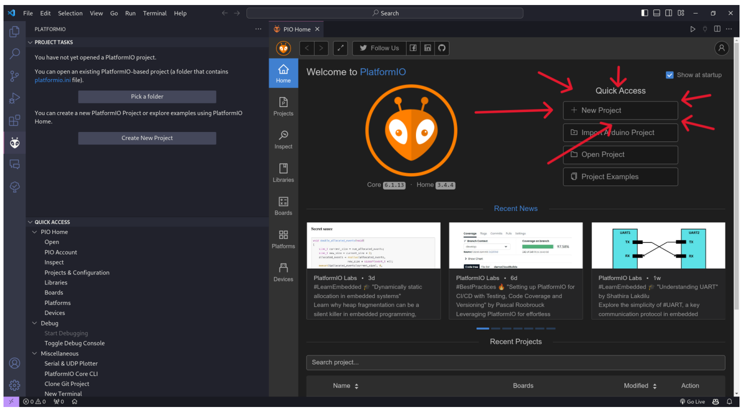
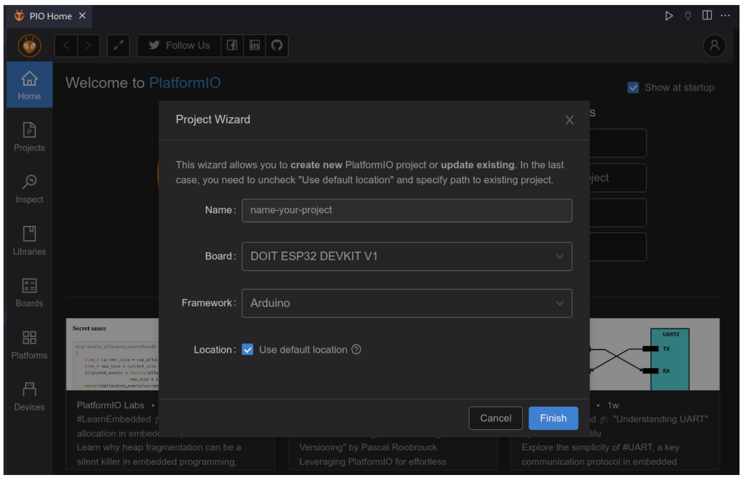


Figure 1 – PlatformIO Tab



Project Wizard

This wizard allows you to **create new** PlatformIO project or **update existing**. In the last case, you need to uncheck "Use default location" and specify path to existing project.

Name: name-your-project

Board: DOIT ESP32 DEVKIT V1

Framework: Arduino

Location: ✓ Use default location

Figure 2 – Create new project

easier to use, and has a lot of features, such as code completion, code linting, git integration, etc.

With VSCode installed, the PlatformIO extension is needed. PlatformIO is an open-source ecosystem for IoT (Internet of Things) development. It is designed to simplify and streamline the process of developing software for embedded systems, including microcontrollers and other hardware platforms. PlatformIO supports a wide range of development boards, frameworks, and platforms, making it versatile for various hardware configurations. Figures 1, 2 and 3 show the necessary steps to create a new project and add libraries to it. Notice that the boards on Figure 2 may differ, there are many ESP32 models, so it's important to look up the specifications.

The libraries that should be added will be discussed later, but they are:

- Adafruit Unified Sensor
- DHT Sensor Library
- PubSubClient

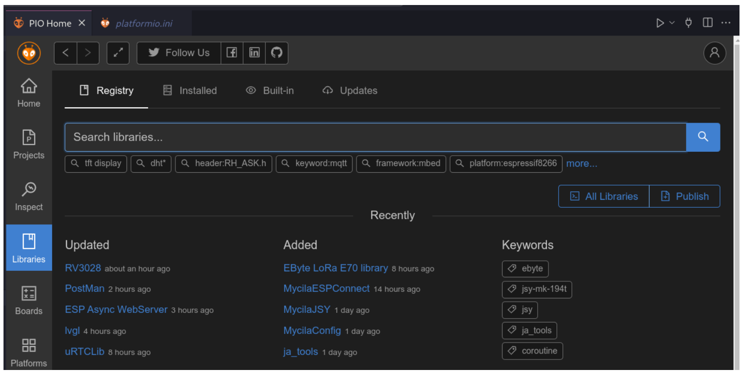


Figure 3 – Libraries

After adding the libraries, there is nothing to do at VSCode. You can close it untill the next steps are concluded. The next steps involve Mosquitto.

3.2 Mosquitto

Mosquitto is an open-source message broker that implements the MQTT protocol. MQTT is a lightweight and efficient messaging protocol designed for low-bandwidth, high-latency, or unreliable networks. This communication method is frequently utilized in IoT (Internet of Things) and other applications where devices require a publishing/subscribing approach. After downloading and installing Mosquitto, a change has to be made to its configuration file. The documentation provides an in-depth explanation, but to simplify, for any Linux system, the following command line should be executed in the terminal:

```
sudo nano /etc/mosquitto/mosquitto.conf
```

The file is then opened, and as Figure 4 shows, the lines below should be added at the end of the file.

- listener 1883 0.0.0.0
- allow_anonymous true

Then this command should be executed for the changes to take effect:

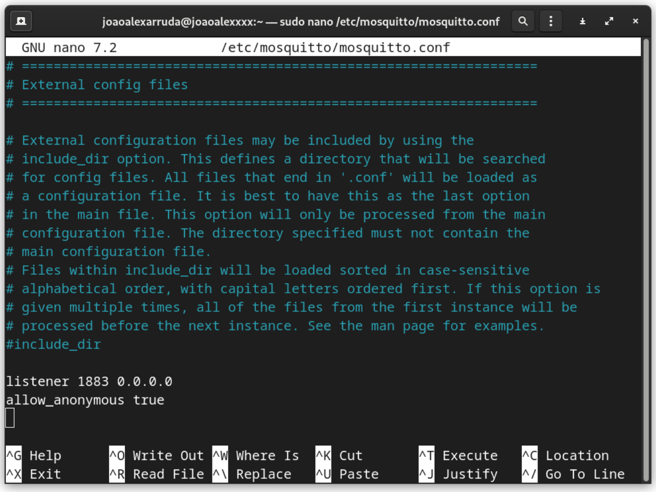
```
sudo systemctl restart mosquitto
```

To start Mosquitto, it's required to run the following command, and it will start, as seen in Figure 5:

```
mosquitto
```

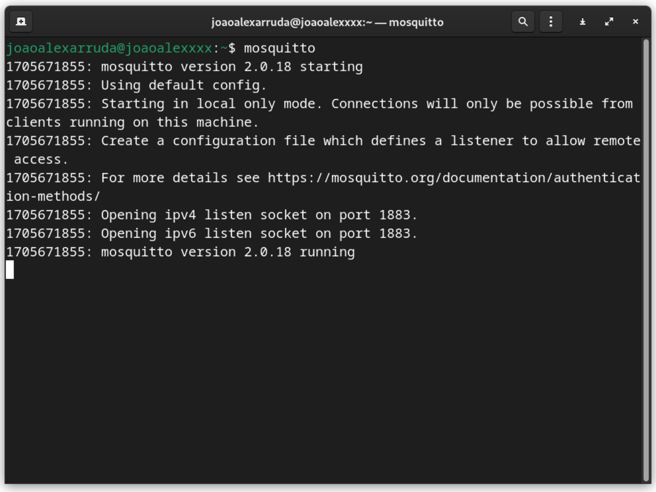
Ideally, a Raspberry Pi should be used to run Mosquitto, but that's not possible for not owning one.

After concluding those steps, hardware should be set next.



```
joaoalexarruda@joaoalexxxx:~ — sudo nano /etc/mosquitto/mosquitto.conf
  GNU nano 7.2                 /etc/mosquitto/mosquitto.conf
# =================================================================
# External config files
# =================================================================

# External configuration files may be included by using the
# include_dir option. This defines a directory that will be searched
# for config files. All files that end in '.conf' will be loaded as
# a configuration file. It is best to have this as the last option
# in the main file. This option will only be processed from the main
# configuration file. The directory specified must not contain the
# main configuration file.
# Files within include_dir will be loaded sorted in case-sensitive
# alphabetical order, with capital letters ordered first. If this option is
# given multiple times, all of the files from the first instance will be
# processed before the next instance. See the man page for examples.
#include_dir

listener 1883 0.0.0.0
allow_anonymous true


^G Help         ^O Write Out ^W Where Is  ^K Cut          ^T Execute    ^C Location
^X Exit         ^R Read File ^\ Replace   ^U Paste        ^J Justify    ^/ Go To Line
```

Figure 4 – mosquitto.conf



```
joaoalexarruda@joaoalexxxx:~ — mosquitto
joaoalexarruda@joaoalexxxx:~$ mosquitto
1705671855: mosquitto version 2.0.18 starting
1705671855: Using default config.
1705671855: Starting in local only mode. Connections will only be possible from
clients running on this machine.
1705671855: Create a configuration file which defines a listener to allow remote
 access.
1705671855: For more details see https://mosquitto.org/documentation/authenticat
ion-methods/
1705671855: Opening ipv4 listen socket on port 1883.
1705671855: Opening ipv6 listen socket on port 1883.
1705671855: mosquitto version 2.0.18 running
```

Figure 5 – Mosquitto running

4 Setting up the hardware

Besides the computer running Mosquitto, there are a few simple items required to be able to build this project:

- ESP32 Microcontroller
- DHT11 Temperature and Humidity Sensor
- 10 kΩ resistor
- Breadboard
- A few jumper wires

Connect the DHT11 sensor to your ESP32. The connections are as follows:

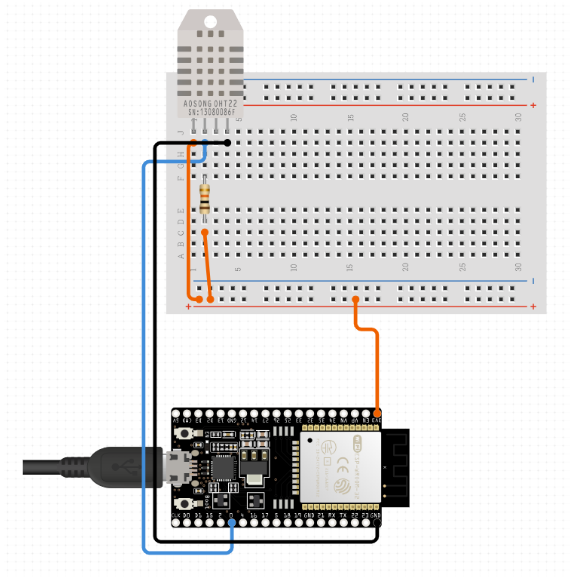


Figure 6 – Project Scheme

Figure 7 – Hardware

- DHT11 VCC to ESP32 3V3
- DHT11 GND to ESP32 GND
- DHT11 DATA to ESP32 GPIO 4 (or another pin)
- DHT11 DATA to the 10 kΩ resistor
- 10 kΩ resistor to ESP32 3V3

Figure 6 shows the project scheme and connections, and Figure 7 exhibits the hardware utilized.

4.1 ESP32

The ESP32 is a microcontroller belonging to the ESP (Espressif) family of chips. It is commonly used in developing Internet of Things (IoT) devices, embedded systems, and other applications requiring low-power and wireless connectivity. The ESP32 is known for its processing power, peripherals, and built-in support for Wi-Fi and Bluetooth communication. It has a robust development ecosystem, with support for popular frameworks and platforms such as the Arduino IDE, PlatformIO, Espressif IDF (IoT Development Framework), and MicroPython.

4.2 DHT11

The DHT11 is a digital sensor that measures temperature and humidity. It is commonly used in DIY electronics and IoT projects due to its simplicity, affordability, and ease of use. The sensor is manufactured by various companies and is widely available in module form with a small circuit board around it.

5 Code structure and explanation

The code is nicely formatted and mostly follows Microsoft C++ Standards, although not entirely. Indentation takes fours spaces, and variables and functions are camel Case. Every block of the code contained in src/main.cpp will be explained next:

```cpp
// Include the necessary headers/libraries
#include <Adafruit_Sensor.h>
#include <Arduino.h>
#include <DHT.h>
#include <PubSubClient.h>
#include <WiFi.h>
#include <deque> // To be able to add and remove elements from both ends
#include <numeric>
#include "ESPAsyncWebServer.h"
```

As the comment suggests, the first part of the code includes all necessary headers for the project to work.

- Adafruit_Sensor.h: provides a common interface for various sensors, simplifying sensor integration in Arduino projects developed by Adafruit
- Arduino.h: it includes the basic declarations and functions required for an Arduino sketch, such as the setup and loop functions. It also includes definitions for common Arduino functions and types
- DHT.h: used for interfacing with DHT series sensors, such as DHT11 and DHT22, to measure temperature and humidity in electronic projects
- PubSubClient.h: facilitates communication with MQTT (Message Queuing Telemetry Transport) brokers, enabling devices to publish and subscribe to topics in IoT (Internet of Things) applications
- WiFi.h: provide functions for connecting and managing WiFi networks, allowing Arduino devices to communicate wirelessly over Wi-Fi
- deque: allows fast insertion and deletion at both the beginning and the end of the sequence. It stands for "double-ended queue" because it supports efficient insertion and removal of elements from both ends

- numeric: provides several numeric algorithms that operate on ranges of elements, such as the std::accumulate function, which is used to compute the sum of elements
- ESPAsyncWebServer.h: provides functionality for creating asynchronous web servers on ESP8266 and ESP32 platforms

```
// Wifi details: SSID and password
const char *ssid = "joaoalex1";
const char *password = "joao1579";

// MQTT broker details: IP address and port
const char *mqttServer = "192.168.29.165";
const int mqttPort = 1883;
```

Those are self-explained, although it's good to remind that the mqttServer is the IP address of the machine running it. On Linux, it's possible to check the IP address by entering **hostname -I** in the terminal. The mqttPort defines the standard port for communication between IoT devices and MQTT brokers.

```
// DHT11 sensor details: pin and type
const int DHTPIN = 4;
const int DHTTYPE = DHT11;
DHT dht(DHTPIN, DHTTYPE);
```

This defines the pin used and the dht type.

```
// Deque to store the last temperature and humidity readings
// and calculate the moving average
std::deque<float> temperatureReadings;
std::deque<float> humidityReadings;

// Size of the moving average, i.e., how many readings to consider
const int MOVING_AVERAGE_SIZE = 10;
```

- temperatureReadings and humidityReadings: These are instances of the std::deque template class, specialized for storing floating-point values (float). A deque (double-ended queue) is a dynamic array-like container that allows fast insertion and deletion at both ends.
- MOVING_AVERAGE_SIZE: This constant represents the size of the moving average. A moving average is a statistical calculation that is commonly used to analyze data over a certain period. In this case, it specifies how many previous readings (temperature or humidity) will be considered when calculating the moving average.

```
// Time to keep track of the last reading
unsigned long lastReadingTime = 0;
```

The variable is intended to keep track of the time when the last reading (temperature or humidity) was taken.

```
// Create an instance of the AsyncWebServer class
// to serve the web page
AsyncWebServer server(80);

// WifiClient and PubSubClient instances,
// to connect to the MQTT broker
WiFiClient espClient;
PubSubClient client(espClient);
```

Setup for creating a web server on an ESP8266 or ESP32 microcontroller using the ESPAsyncWebServer library. The web server typically listens on port 80, which is the default port for HTTP communication.

These objects are instantiated to facilitate communication over a Wi-Fi network and, specifically, for MQTT messaging. The WiFiClient object (espClient) will handle the low-level details of the Wi-Fi connection, while the PubSubClient object (client) will handle the MQTT protocol for publishing and subscribing to topics.

```
// Function to setup the Wi-Fi connection
void setupWifi() {
    delay(10);
    Serial.println("Connecting to " + String(ssid) + "...");
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connecting...");
    }
    Serial.println("Connected to " + String(ssid) + " network!");
}
```

The setupWifi() function attempts to connect to a Wi-Fi network with the provided SSID and password, printing connection messages to the Serial Monitor, and confirming successful connection.

```
// Function to reconnect to the MQTT broker
void reconnect() {
    while (!client.connected()) {
        Serial.print("Trying to connect to MQTT broker...");
        if (client.connect("ESP32Client")) {
            Serial.println("Connected!");
        } else {
            Serial.print("Failed, rc=");
            Serial.print(client.state());
            Serial.println(" Retrying in 5 seconds...");
            delay(5000);
        }
    }
}
```

The reconnect() function attempts to connect to an MQTT broker using the PubSubClient object named client and retries with a 5-second delay if the connection fails.

```
// Read the temperature from the DHT11 sensor and
// calculate the moving average of the last readings
float readTemperatureAndCalculateMovingAverage() {
    float currentTemperature = dht.readTemperature();
    // If the reading is NaN, return the last valid reading
    if (isnan(currentTemperature)) {
        return temperatureReadings.back();
    }

    // Add the current reading to the readings deque
    temperatureReadings.push_back(currentTemperature);

    // If we have more readings than we want to consider for the moving average,
    // remove the oldest one
    if (temperatureReadings.size() > MOVING_AVERAGE_SIZE) {
        temperatureReadings.pop_front();
    }

    // Calculate the sum of the readings
    float sum = std::accumulate(temperatureReadings.begin(),
                                temperatureReadings.end(), 0.0f);

    // Calculate and return the moving average
    return sum / temperatureReadings.size();
}
```

Reads the temperature from a DHT11 sensor, handles potential errors, and calculates the moving average of the last temperature readings stored in a deque (temperatureReadings).

- dht.readTemperature(): This line reads the current temperature from the DHT11 sensor using the readTemperature() function provided by the DHT library.
- Checking for NaN: If the reading is NaN, which stands for "Not a Number" (indicating a failed or invalid reading), the function returns the last valid temperature reading stored in the deque. This helps in handling situations where the sensor may provide unreliable readings.
- Adding Current Reading to Deque: The current valid temperature reading is added to the temperatureReadings deque.
- Deque Size Control: If the number of readings in the deque exceeds the specified MOVING_AVERAGE_SIZE, the oldest reading is removed to maintain the deque's size.
- Calculating Moving Average: The code then calculates the sum of all readings in the deque using std::accumulate from the <numeric> header. Finally, the function returns the calculated moving average by dividing the sum by the number of readings in the deque.

```
// Read the humidity from the DHT11 sensor and
// calculate the moving average of the last readings
float readHumidityAndCalculateMovingAverage() {
    float currentHumidity = dht.readHumidity();
    // If the reading is NaN, return the last valid reading
    if (isnan(currentHumidity)) {
        return humidityReadings.back();
    }

    // Add the current reading to the readings deque
    humidityReadings.push_back(currentHumidity);

    // If we have more readings than we want to consider for the moving average,
    // remove the oldest one
    if (humidityReadings.size() > MOVING_AVERAGE_SIZE) {
        humidityReadings.pop_front();
    }

    // Calculate the sum of the readings
    float sum =
        std::accumulate(humidityReadings.begin(), humidityReadings.end(), 0.0f);

    // Calculate and return the moving average
    return sum / humidityReadings.size();
}
```

Works exactly like the temperature code.

```
// HTML code for the web page served by the ESP32 microcontroller
// The placeholders %TEMPERATURE% and %HUMIDITY% will be replaced by the actual
// values
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<script src="https://kit.fontawesome.com/1b6e98d141.js"
    crossorigin="anonymous"></script>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <style>
  body {
    font-family: Arial, sans-serif;
    background-color: #121212;
    color: #ffffff;
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    height: 100vh;
    }
    .header { display: flex; align-items: center; gap: 20px; }
    .header img { width: 160px; height: auto; }
    .header h2 { font-size: 2.5rem; color: #ffffff; }
    p { background-color: #1f1f1f; border-radius: 5px; padding: 20px; margin:
        10px; width: 80vw; display: flex; align-items: center; justify-content:
        space-between; }
    .units { font-size: 1rem; }
    .fa-solid, .fas { margin-right: 10px; }
    .dht-labels{
      flex-grow: 1;
      text-transform: uppercase;
      letter-spacing: 1px;
    }
    footer {
      position: fixed;
      left: 0;
      bottom: 0;
      width: 100vw;
      background-color: #1f1f1f;
      color: white;
      text-align: center;
      padding: 10px 0;
    }
    footer a {
      color: white;
      text-decoration: none;
    }
    footer a:hover {
      color: #ddd;
    }
  </style>
</head>
<body>
  <div class="header"><img
      src="https://i.imgur.com/23DiEOf.png"><h2>ESP32</h2></div>
  <p>
    <i class="fa fa-temperature-high" style="color:#9e0505;"></i>
    <span class="dht-labels"> TEMPERATURE</span>
    <span id="temperature">%TEMPERATURE%</span>
    <span class="units">&deg;C</span>
  </p>
  <p>
    <i class="fas fa-tint" style="color:#00add6;"></i>
    <span class="dht-labels"> HUMIDITY</span>
    <span id="humidity">%HUMIDITY%</span>
    <span class="units">&percnt;</span>
  </p>
  <footer>
    <a href="https://github.com/joaoalexarruda" target="_blank"><i class="fab
        fa-github fa-2x"></i></a>
  </footer>
</body>
<script>
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("temperature").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/temperature", true);
  xhttp.send();
}, 10000 ) ;
```

```
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("humidity").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/humidity", true);
  xhttp.send();
}, 10000 ) ;
</script>
</html>)rawliteral";
```

HTML code to be displayed on the web server.

```
// Function to replace the placeholders in the HTML code
// with the actual values
String processor(const String &var) {
    // Serial.println(var);
    if (var == "TEMPERATURE") {
        return String(readTemperatureAndCalculateMovingAverage());
    } else if (var == "HUMIDITY") {
        return String(readHumidityAndCalculateMovingAverage());
    }
    return String();
}
```

Serves as a callback function for the ESPAsyncWebServer library. The purpose of this function is to replace placeholders in an HTML code template with actual values. The placeholders are identified by specific strings (e.g., "TEMPERATURE" and "HUMIDITY"). The actual values are obtained by calling functions (readTemperatureAndCalculateMovingAverage() and readHumidityAndCalculateMovingAverage()) and converting the results to strings using String().

```
// Setup function
// This function is called only once when the microcontroller starts
void setup() {
    Serial.begin(115200);                 // Initialize serial communication
    dht.begin();                          // Initialize DHT sensor
    setupWifi();                          // Setup Wi-Fi connection
    client.setServer(mqttServer, mqttPort); // Setup MQTT broker

    // Setup the web server, and define the routes
    server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
        request->send_P(200, "text/html", index_html, processor);
    });
    server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request) {
        request->send_P(
            200, "text/plain",
            String(readTemperatureAndCalculateMovingAverage()).c_str());
    });
    server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request) {
        request->send_P(
            200, "text/plain",
            String(readHumidityAndCalculateMovingAverage()).c_str());
    });

    // Start server
    server.begin();
}
```

This setup function combines the initialization of serial communication, DHT sensor, Wi-Fi connection, MQTT broker, and the web server. The web server responds to specific routes, serving HTML pages and providing temperature and humidity information via HTTP requests.

```
// Loop function
// This function is called repeatedly
void loop() {
    // Check if the client is connected to the MQTT broker
    if (!client.connected()) {
        reconnect();
    }

    // Every 3 seconds, read the temperature and humidity from the DHT11 sensor,
    // calculate the moving average, and publish the values to the MQTT broker
    // This interval allows the microcontroller to perform other tasks
    // and not be constantly occupied with reading sensor data.
    unsigned long currentMillis = millis();
    if (currentMillis - lastReadingTime >= 3000) {
        // Read the temperature and humidity from the DHT11 sensor and
        // calculate the moving average
        float avgTemperature = readTemperatureAndCalculateMovingAverage();
        float avgHumidity = readHumidityAndCalculateMovingAverage();

        // Convert the float values to strings
        char avgTempStr[8];
        dtostrf(avgTemperature, 2, 2, avgTempStr);
        char avgHumStr[8];
        dtostrf(avgHumidity, 2, 2, avgHumStr);

        // Name of the topics to publish the values
        char avgTempTopic[] = "esp32/moving_average_temperature";
        char avgHumTopic[] = "esp32/moving_average_humidity";

        // Publish the values to the MQTT broker
        client.publish(avgTempTopic, avgTempStr);
        client.publish(avgHumTopic, avgHumStr);

        // Print the values and topics to the serial monitor for debugging
        Serial.println();
        Serial.println("+~~~~~~~~~~~~~~~~~~~+~~~~~~~~~~~~~~~~~~~+");
        Serial.println("|           DEBUGGING            |");
        Serial.println("+~~~~~~~~~~~~~~~~~~~+~~~~~~~~~~~~~~~~>~~~+");
        Serial.printf("| Local IP Address | %-17s |\n",
                      WiFi.localIP().toString().c_str());
        Serial.println("+~~~~~~~~~~~~~~~~~~~+~~~~~~~~~~~~~~~~~~~+");
        Serial.println("| Parameter     | Value           |");
        Serial.println("+-------------------+-------------------+");
```

```
        Serial.printf("| Moving Avg Temp.  | %-17.2f |\n", avgTemperature);
        Serial.printf("| Moving Avg Humid. | %-17.2f |\n", avgHumidity);
        Serial.println("+-------------------+-------------------+");
        Serial.println("|         Published Topic         |");
        Serial.println("+-------------------+-------------------+");
        Serial.println("| " + String(avgTempTopic) + " |");
        Serial.println("| " + String(avgHumTopic) + "    |");
        Serial.println("+-------------------+-------------------+");

        // Update the last reading time
        lastReadingTime = currentMillis;
    }
}
```

The block within if (currentMillis - lastReadingTime >= 3000) ensures that the DHT sensor readings and MQTT publishing occur at intervals of 3 seconds. This interval is controlled by the 3000 milliseconds condition.

Within the conditional block, it reads the temperature and humidity from the DHT11 sensor and calculates the moving averages.

The dtostrf function is used to convert the floating-point values of temperature and humidity to strings with a specific format.

The client.publish function is used to publish the temperature and humidity values to the MQTT broker on the topics created.

The lastReadingTime variable is updated with the current timestamp (currentMillis) to keep track of the time of the last sensor reading.

Note that there is a nice formatted table being printed to the serial monitor for debugging purposes.

6 Validation tests and visualizing the data

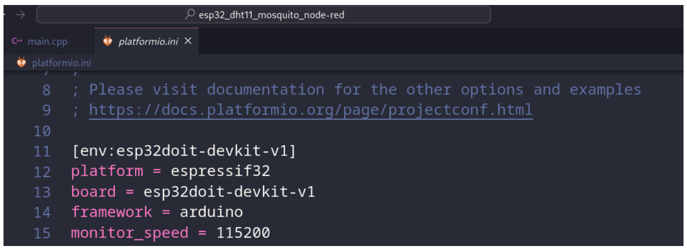
6.1 Serial monitor

Now it is time to upload the code to the ESP32 to start visualizing the temperature and humidity on a nice web page. After building and uploading, if there are no code errors, it's a good idea to check if the code is working properly. That can be accomplished by paying attention to the serial monitor. When using PlatformIO, it's required to add the line "monitor_speed = 115200" in the platformio.ini file, as seen in Figure 8. The platformio.ini file allows users to configure various settings for their projects, such as the target platform, build options, dependencies, and more.

Now it's possible to see the messages on the serial monitor.

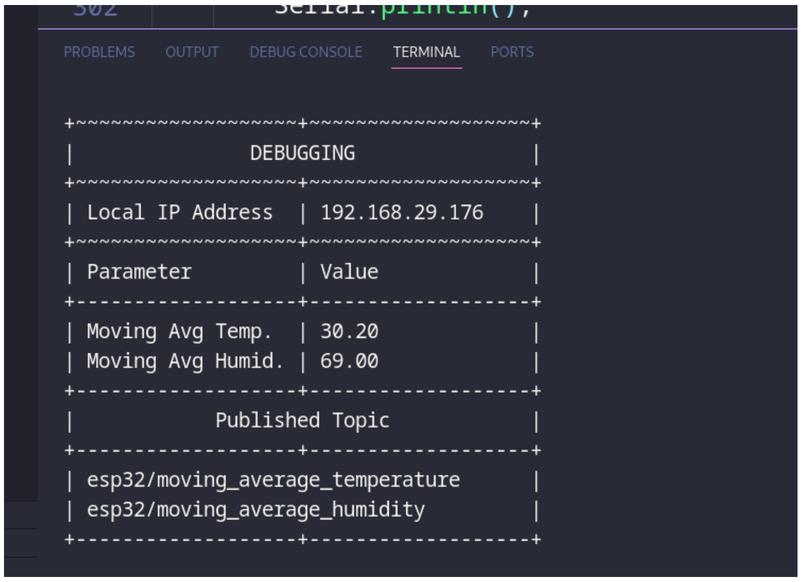
As seen in the previous section, the loop function continuously prints useful information for debugging in a nicely formatted table, as seen in Figure 9. If weird symbols start being printed, the reset button on the ESP32 should be pressed once.

The table confirms that the sensor is working and also says that the data is being published with their respective topics. As seen in Figure 10, if something is wrong, it's possible to know from looking at the serial monitor as well. Those are two possible situations that could happen during the execution of the loop, but there are more. The first is when



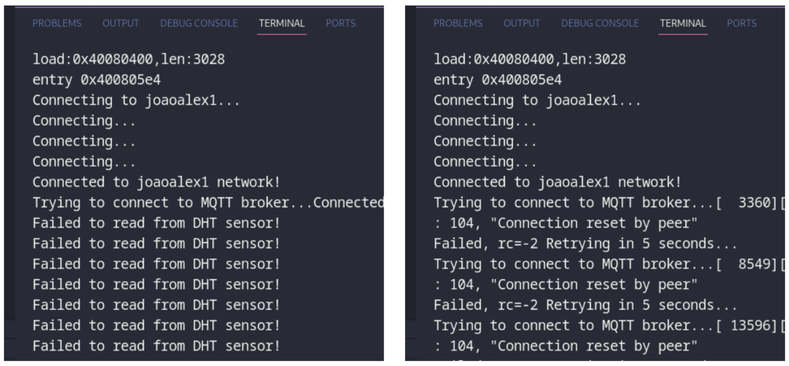
```
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:esp32doit-devkit-v1]
platform = espressif32
board = esp32doit-devkit-v1
framework = arduino
monitor_speed = 115200
```

Figure 8 – platformio.ini



```
+~~~~~~~~~~~~~~~~~~~~+~~~~~~~~~~~~~~~~~~~~+
|               DEBUGGING                 |
+~~~~~~~~~~~~~~~~~~~~+~~~~~~~~~~~~~~~~~~~~+
| Local IP Address  | 192.168.29.176      |
+~~~~~~~~~~~~~~~~~~~~+~~~~~~~~~~~~~~~~~~~~+
| Parameter         | Value               |
+-------------------+-------------------+
| Moving Avg Temp.  | 30.20               |
| Moving Avg Humid. | 69.00               |
+-------------------+-------------------+
|            Published Topic              |
+-------------------+-------------------+
| esp32/moving_average_temperature        |
| esp32/moving_average_humidity           |
+-------------------+-------------------+
```

Figure 9 – Debugging with Serial Monitor



```
load:0x40080400,len:3028
entry 0x400805e4
Connecting to joaoalex1...
Connecting...
Connecting...
Connecting...
Connected to joaoalex1 network!
Trying to connect to MQTT broker...Connected
Failed to read from DHT sensor!
Failed to read from DHT sensor!
Failed to read from DHT sensor!
Failed to read from DHT sensor!
Failed to read from DHT sensor!
Failed to read from DHT sensor!
Failed to read from DHT sensor!
```

```
load:0x40080400,len:3028
entry 0x400805e4
Connecting to joaoalex1...
Connecting...
Connecting...
Connecting...
Connected to joaoalex1 network!
Trying to connect to MQTT broker...[  3360][
: 104, "Connection reset by peer"
Failed, rc=-2 Retrying in 5 seconds...
Trying to connect to MQTT broker...[  8549][
: 104, "Connection reset by peer"
Failed, rc=-2 Retrying in 5 seconds...
Trying to connect to MQTT broker...[ 13596][
: 104, "Connection reset by peer"
```

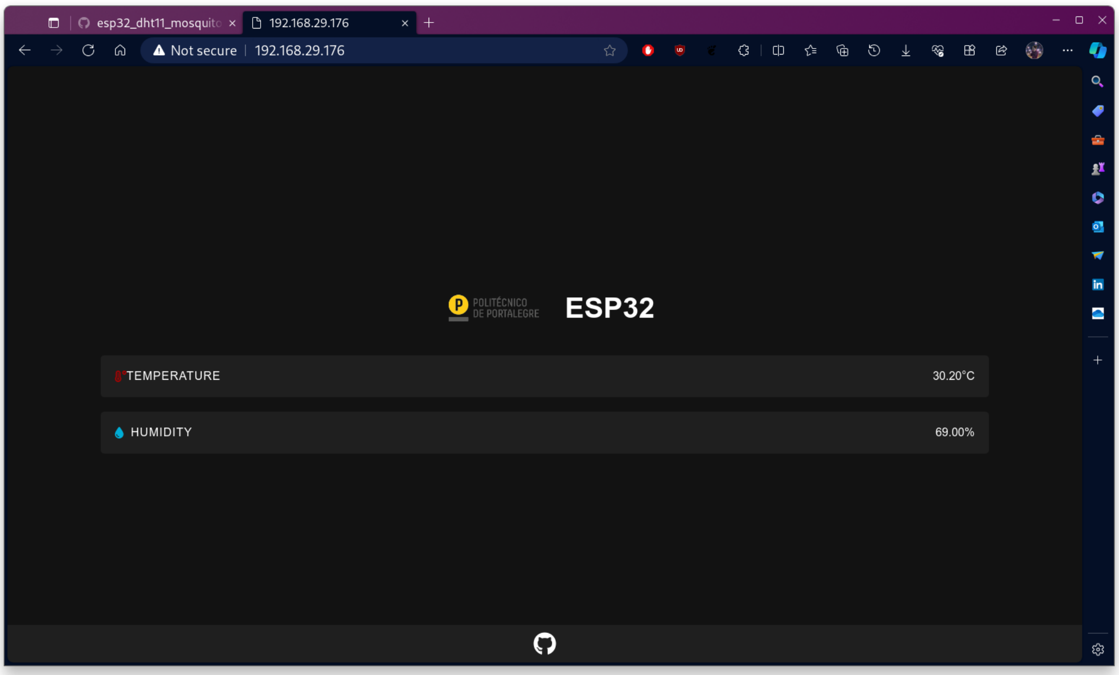
Figure 10 – Debugging

the sensor is not even assembled, and the latter is when the Mosquitto MQTT Broker is not running.

It's time to visualize what's been done in a more intuitive way. For that, the previously demonstrated HTML code will be seen on the browser.

6.2 Web server

In order to see the project on the web page, it's necessary to put the local IP address on the browser. Once that's done, the page is displayed as seen in Figure 11.



TEMPERATURE 30.20°C

HUMIDITY 69.00%

Figure 11 – Web Page

This page is displaying both temperature and humidity moving averages. It has a nice Instituto Politécnico de Portalegre logo and an icon linking to my GitHub profile.

7 Conclusion and future improvements

The DHT11 sensor is a very cheap sensor. It's not very accurate, and it's not very reliable. It's a good sensor to start with, but it's not recommended for real-world applications.

It's very easy to install everything if you are running Linux. If you are running Windows, you may have some problems. I recommend using Fedora.

VS Code + PlatformIO is preferred over the Arduino IDE. It's much easier to use, and it has a lot of features, such as code completion, code linting, git integration, etc.

The ESP32 is a very powerful microcontroller. It's very easy to use, and it has a lot of features. It's a good choice for this project.

In this project, the capacity of the IoT becomes tangible, and with the results presented, it's easy to think of future improvements:

- Connect another ESP32 with a DHT11 sensor to be able to calculate the temperature and humidity differences between two rooms, for example.
- Get a Raspberry Pi and install the Mosquitto MQTT Broker and Node-RED on it. This way you can have a dedicated device for this project, and also visualize the data on nice dashboards.
- Subscribe multiple devices to the same MQTT broker so they can do actions based on the data received.
- Use a good sensor instead of a cheap DHT11 sensor.
- Implement a better way to handle errors.
- Utilize these sensors in a real-world scenario, like a smart home. For example, turn on the air conditioner when the temperature is too high, or turn on the humidifier when the humidity is too low.