Mestrado Informática

Pós-Graduação - Data Science and Digital Transformation

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 Node-RED dashboard for data visualization. Node-RED's intuitive visual programming environment enables users to create dynamic and responsive dashboards that provide real-time insights into the collected temperature and humidity data. 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. It's an improvement over this simpler project, which only works on a local network with no dashboard nor broker, but also presents the data on a nicely formatted screen. Take a look at the old one to see the differences.

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 and insightful visualizations using Node-RED and Node-RED dashboards. This will enable users to easily interpret and comprehend the environmental data through dynamic graphs and gauges. These objectives contribute to the development of a robust and integrated IoT solution for temperature and humidity monitoring.

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, node-red and all other steps needed to be able to replicate the results.

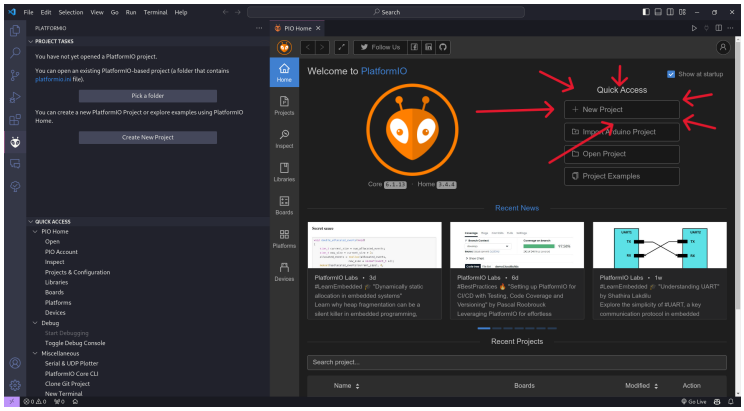

Figure 1 – PlatformIO Tab

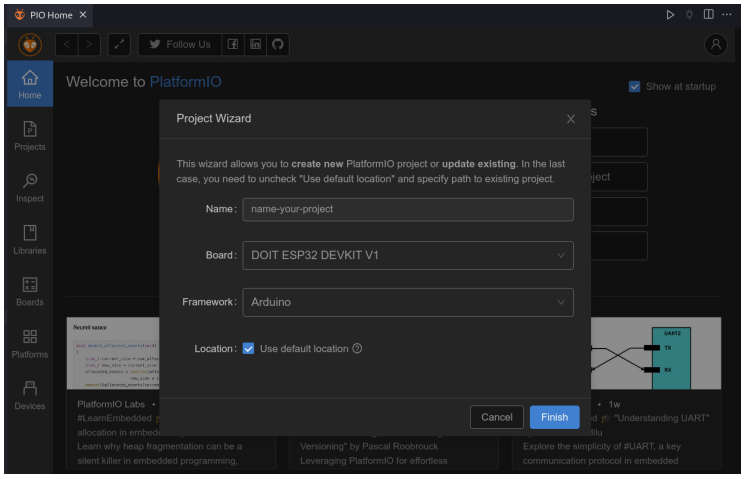

Figure 2 – Create new project

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 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.

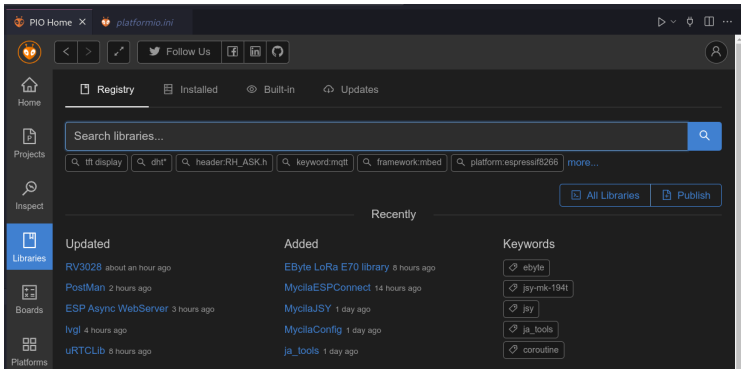

Figure 3 – Libraries

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

- Adafruit Unified Sensor
- DHT Sensor Library
- PubSubClient

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 and Node-red.

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:

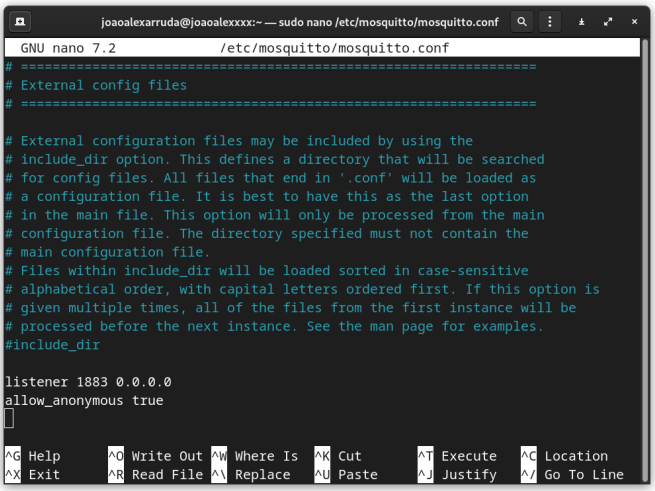

Figure 4 – mosquitto.conf

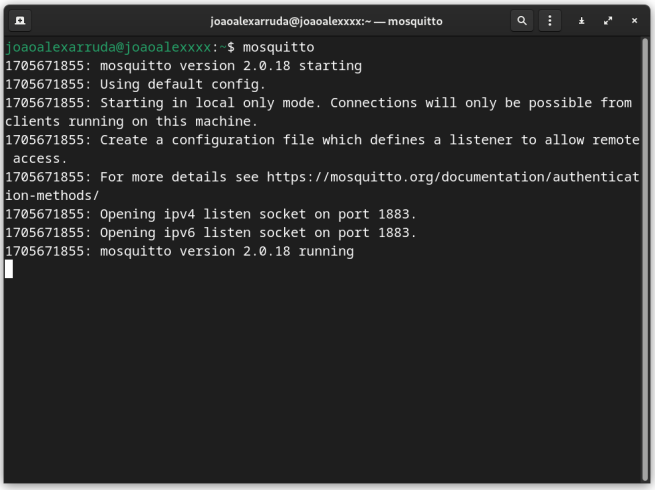

Figure 5 – Mosquitto running

```
mosquitto
```

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

After concluding those steps, node-red should be installed next.

3.3 Node-red

Node-RED is an open-source flow-based development tool used for visual programming of Internet of Things (IoT) and other event-driven systems. It provides a browser-based editor that allows users to easily connect devices, APIs (Application Programming Interfaces), and online services. Developed by IBM Emerging Technology, Node-RED is built on top of Node.js and is widely used for creating automation and integration solutions.

The documentation provides different methods of installation. More about the node-red when it's time to use it.

3.3.1 Node-red-dashboard

It's also required to install Node-RED Dashboard, which is a module that complements Node-RED, providing nodes and a web-based user interface to create real-time dashboards for visualizing data and controlling IoT devices and systems. It simplifies the process of creating interactive and responsive dashboards without requiring extensive web development skills.

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:

- 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.

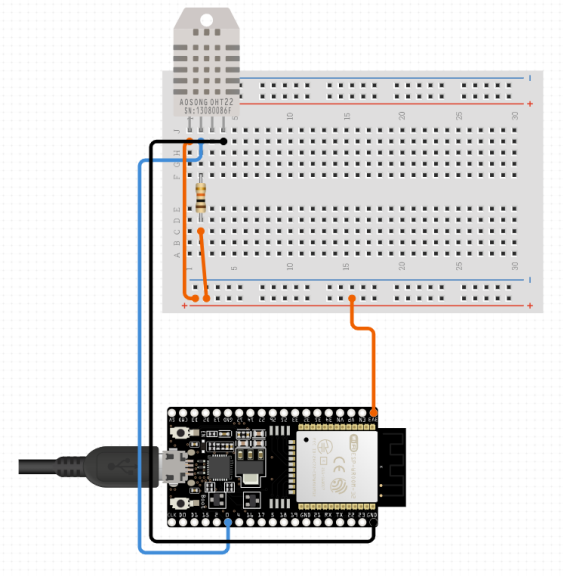

Figure 6 – Project Scheme

Figure 7 – Hardware

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:

```
// Include the necessary headers/libraries
#include <Adafruit_Sensor.h>
```

```
#include <Arduino.h>
#include <DHT.h>
#include <PubSubClient.h>
#include <WiFi.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

```
// SSID and password for the Wi-Fi network
const char *ssid = "joaoalex1";
const char *password = "joao1579";

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

// Number of readings to calculate the moving average
const int numReadings = 5;
int readingsIndex = 0;
```

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.

```
/*
 * These two lines of code are declaring and initializing arrays of floats
 * with all elements set to -1.0. It is a placeholder value to indicate that
 * the array element has not been set yet. Later, when we read the temperature
 * and humidity from the DHT11 sensor, we will update the array with the new
 * values.
 */
float temperatureReadings[numReadings] = {-1.0};
float humidityReadings[numReadings] = {-1.0};
```

As mentioned in the comment, these arrays are intended to be updated later in the code when actual temperature and humidity readings are obtained from the DHT11 sensor. The initial use of -1.0 makes it clear that the array values are not valid until updated.

```
// Defines the DHT pin and type
#define DHTPIN 4
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
```

This structure is common in Arduino programming, where macro definitions are used to set configuration parameters at the beginning of the code. It provides a convenient way to make adjustments to hardware configurations without modifying the rest of the code extensively.

```
/* This line creates an instance of the WiFiClient class which represents a TCP
   client that can connect to a specified internet IP address and port. */
WiFiClient espClient;
PubSubClient client(espClient);
```

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.

```
// Function to update the readings array
void updateReadings(float *readings, float newValue) {
    readings[readingsIndex] = newValue;
    readingsIndex = (readingsIndex + 1) % numReadings;
}
```

The updateReadings() function updates an array with a new value, maintaining a circular buffer of readings.

- readings[readingsIndex] = newValue;: updates the element at the current index (readingsIndex) of the readings array with the provided newValue.
- readingsIndex = (readingsIndex + 1) % numReadings;: increments the index (readingsIndex) circularly, ensuring it wraps around to 0 when it reaches the maximum number of readings (numReadings).

By updating the array and cycling through indices, the function effectively creates a circular buffer that overwrites old readings when the buffer is full.

```
/*
 * Function to calculate the moving average of the last 5 readings.
 * It takes an array of floats as an argument and returns a float.
 * The function iterates over the array and calculates the sum of all
 * the values that are not equal to -1.0. Then it divides the sum by
 * the number of values that are not equal to -1.0 and returns the result.
 */
float calculateMovingAverage(float *readings) {
    float sum = 0;
    int count = 0;

    for (int i = 0; i < numReadings; i++) {
        if (readings[i] != -1.0) {
            sum += readings[i];
            count++;
        }
    }

    if (count > 0) {
        return sum / count;
    } else {
        return -1.0;
    }
}
```

Explained in the comment. This function is commonly used in scenarios where a smooth representation of data is needed by averaging out fluctuations over a specified number of readings.

```
// Main function
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
}
```

- Serial.begin(115200);: used for debugging
- client.setServer(mqttServer, mqttPort);: sets up the MQTT broker connection by specifying the server address (mqttServer) and port (mqttPort) for the PubSubClient object named client

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

    // Read temperature and humidity from the DHT11 sensor
    float humidity = dht.readHumidity();
    float temperature = dht.readTemperature();

    // Check if any reading is invalid
    if (isnan(humidity) || isnan(temperature)) {
        Serial.println("Failed to read from DHT sensor!");
        delay(2000);
        return;
    }

    // Update the readings array
    updateReadings(temperatureReadings, temperature);
    updateReadings(humidityReadings, humidity);

    // Calculate the moving average of the last 5 readings
    float avgTemperature = calculateMovingAverage(temperatureReadings);
    float avgHumidity = calculateMovingAverage(humidityReadings);

    // Convert the float values to strings
    char tempStr[8];
    dtostrf(temperature, 2, 2, tempStr);
    char humStr[8];
    dtostrf(humidity, 2, 2, humStr);
    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 tempTopic[] = "esp32/temperature";
    char humTopic[] = "esp32/humidity";
```

```
    char avgTempTopic[] = "esp32/moving_average_temperature";
    char avgHumTopic[] = "esp32/moving_average_humidity";

    // Publish the values to the MQTT broker
    client.publish(tempTopic, tempStr);
    client.publish(humTopic, humStr);
    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("| Parameter      | Value            |");
    Serial.println("+-------------------+--------------------+");
    Serial.printf("| Temperature    | %-17.2f |\n", temperature);
    Serial.printf("| Humidity       | %-17.2f |\n", humidity);
    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(tempTopic) + "                |");
    Serial.println("| " + String(humTopic) + "                   |");
    Serial.println("| " + String(avgTempTopic) + " |");
    Serial.println("| " + String(avgHumTopic) + "    |");
    Serial.println("+-------------------+--------------------+");

    // Wait 3 seconds before reading the sensor again
    delay(3000);
}
```

The loop() function continuously reads temperature and humidity from a DHT11 sensor, calculates moving averages, publishes values to MQTT topics, and prints information to the Serial Monitor for debugging.

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 nice temperature and humidity graphs. 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

Figure 8 – platformio.ini

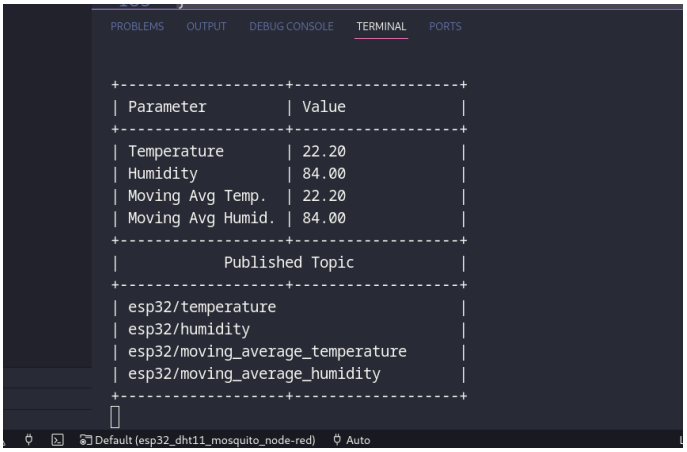

Figure 9 – Debugging with Serial Monitor

Figure 10 – Debugging

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

Back to the best case scenario, where everything is working flawlessly, it's time to visualize what's been done in a more intuitive way. For that, the previously installed node-red and node-red-dashboard will be used.

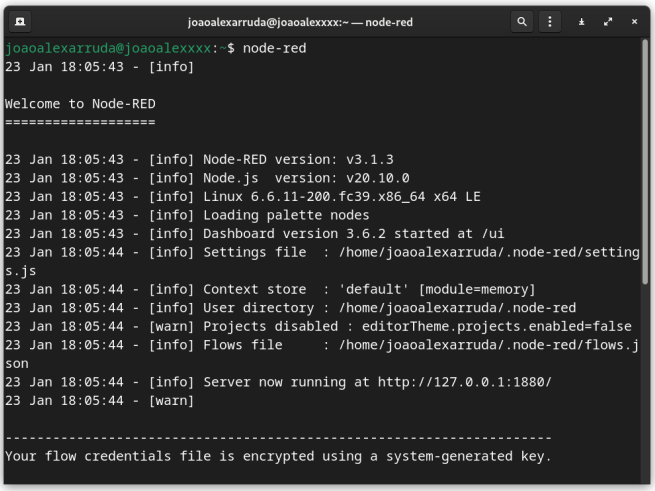

Figure 11 – Node-red in the terminal

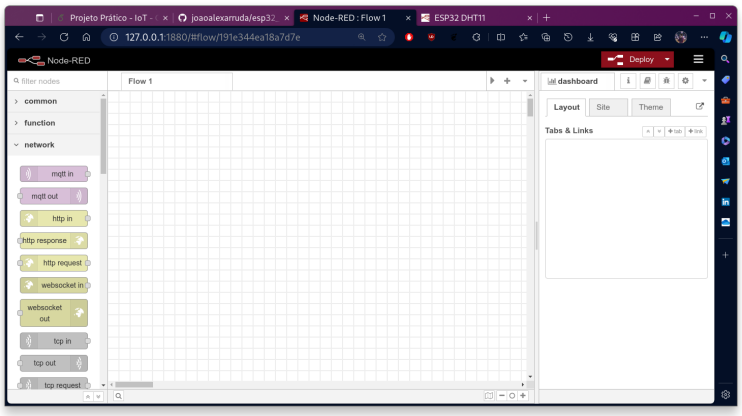

Figure 12 – Node-red on the first use

6.2 Node-red flow

To start node-red, just enter "node-red" in your terminal, as seen in Figure 11, and the server will start running. In this case, it says "Server now running at http://127.0.0.1:1880/".

When that link is clicked, the default browser will open and node-red flow is the first thing visible, as seen in Figure 12.

The UI/UX is very straightforward and intuitive. The panel on the left exhibits the nodes it can use. When one of those is dragged to the flow and clicked, a few options appear, as Figure 13 shows. Be sure to have the right IP:port and topic there.

Figure 14 shows the completed flow for this project. Notice that on the right side there are a few options to set up. But before that, it is interesting to check if node-red is really capable of getting the data from the topics just entered. To do so, hit deploy and then the debugging option, as seen in Figure 15.

As it is possible to see, the data is being published correctly. It's the same data

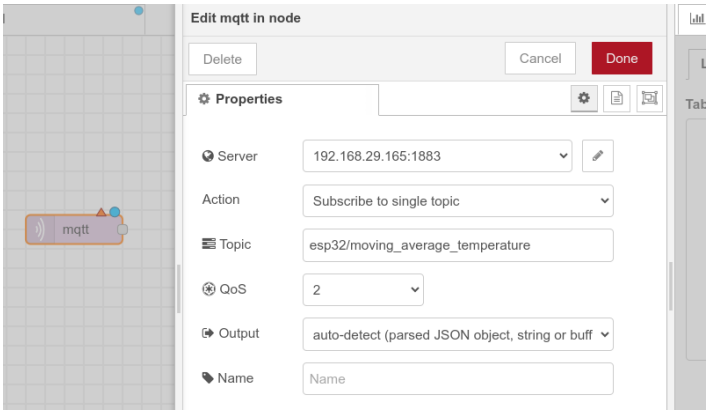

Figure 13 – Editing MQTT in node

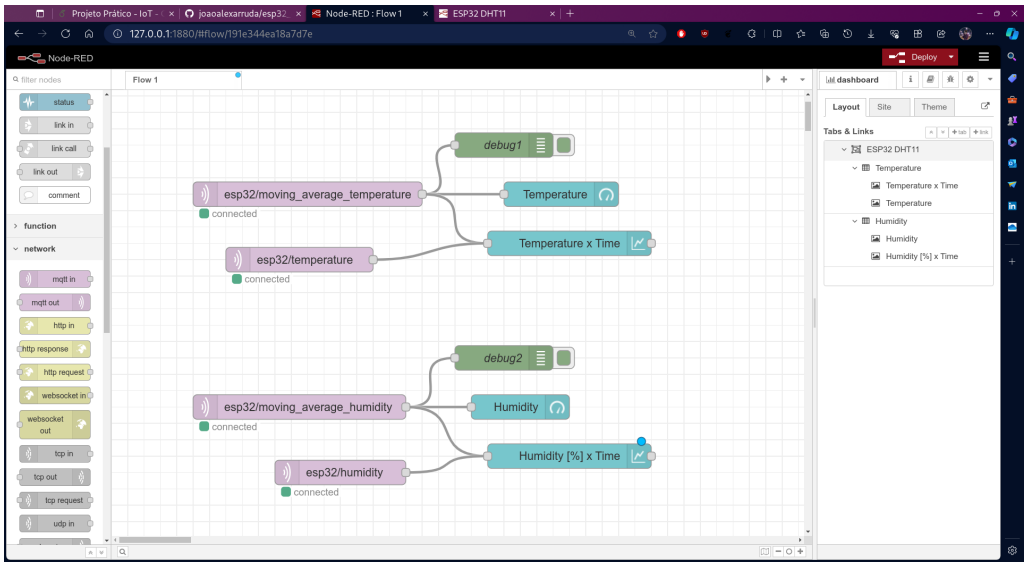

Figure 14 – Completed node-red flow

seen in the serial monitor from Figure 9.

Now it's time to see all that on a dashboard. When clicking the arrow in the top right corner, it's possible to access the dashboard options, as on Figure 16.

There are some tweaks to be made on the dashboard, like setting units, axis range, theme, layout etc. Those options make the dashboard friendly. Figure 17 presents that. The most important one is the layout, so it's actually possible to see the plots and gauges.

After hitting deploy, all it's needed is to enter the following URL in the browser, "http://localhost:1880/ui". Figure 18 presents the page just opened. There, it's possible to see gauges for the temperature and humidity and plots containing both parameters and also the moving averages of them. Notice that the difference is very small, but exists, as shown in Figure 19.

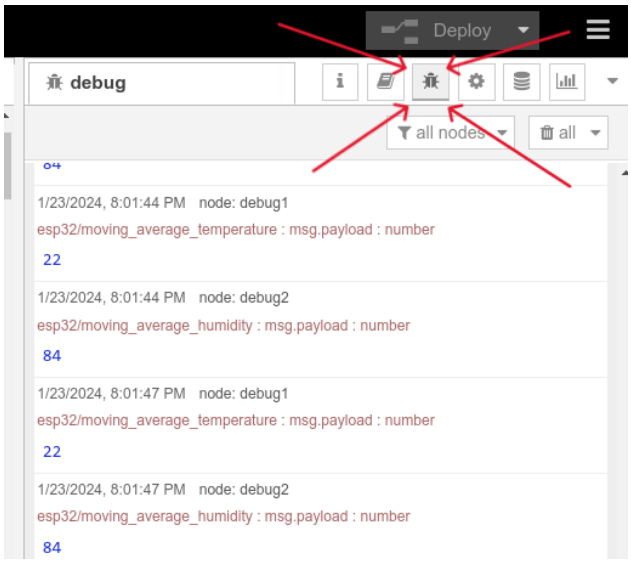

Figure 15 – Validating node-red

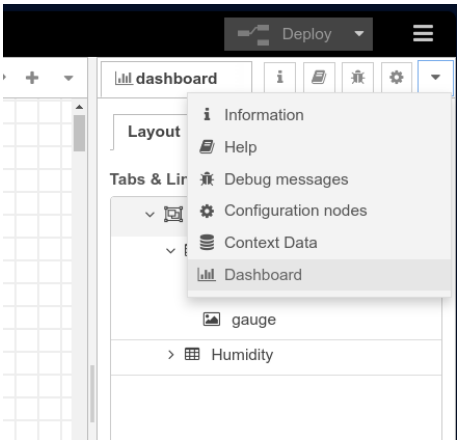

Figure 16 – Dashboard option

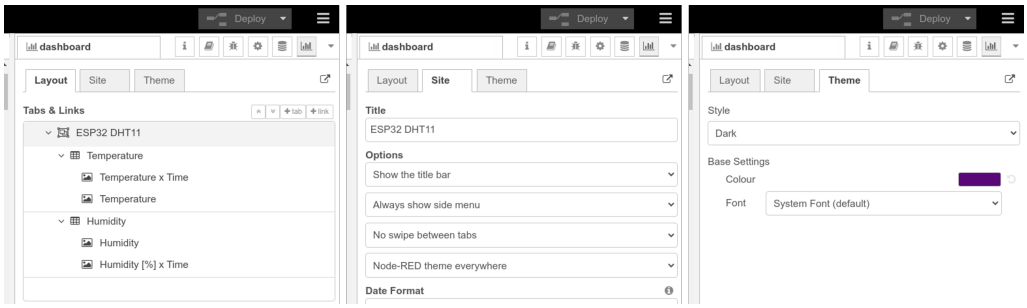

Figure 17 – Dashboard tweaks

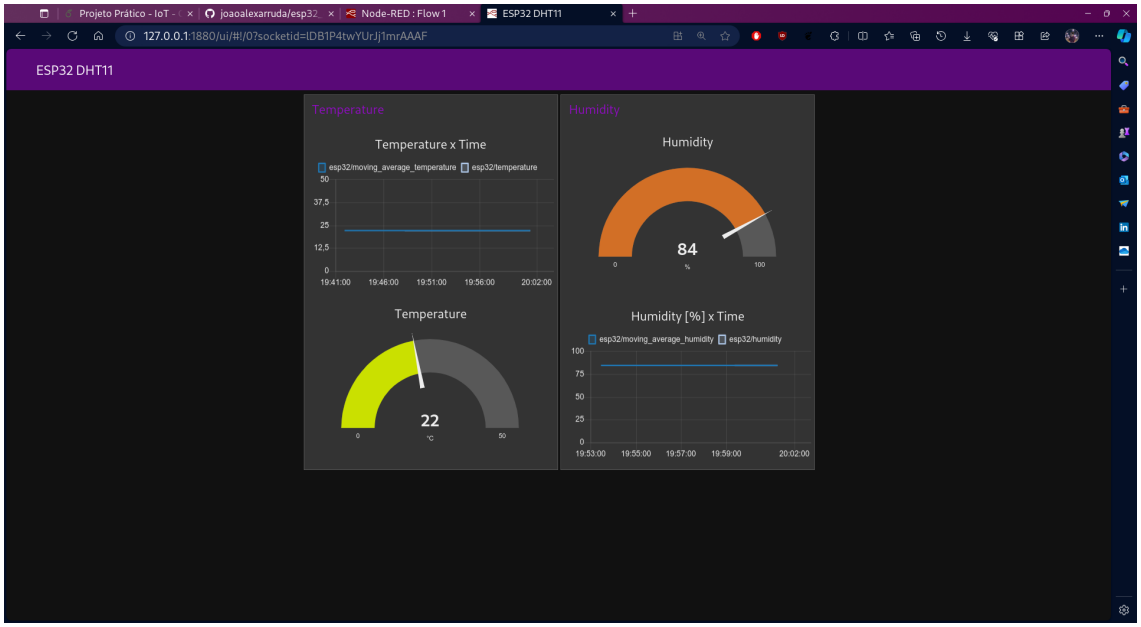

Figure 18 – Node-red-dashboard

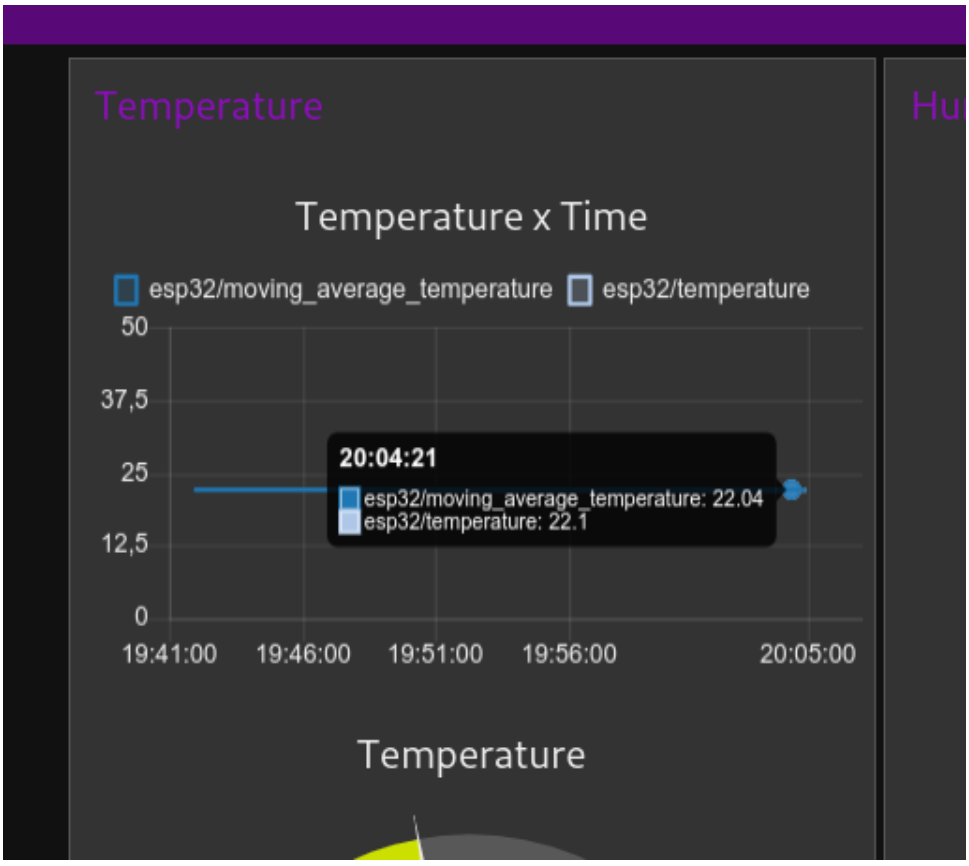

Figure 19 – Both temperature and its moving average plotted

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 the same MQTT broker and display the data on the same dashboard.
- Subscribe multiple devices to the same MQTT broker so they can do actions based on the data received.
- 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.
- 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.