YRKESHÖGSKOLAN
NOVIA

IoT Sensor Rail for Environmental Monitoring

Case study WSTAR - Wasa Zero Emission Data Centre

Nnanna Otuh

Degree Thesis

Thesis for a Bachelor of Engineering (UAS) degree

Information Technology

Vaasa 2025

DEGREE THESIS

Author: Nnanna Otuh
Degree Programme and place of study: Information Technology, Novia UAS
Specialisation: Information Technology
Supervisor(s): Hans Linden

Title: IoT Sensor Rail for Environmental Monitoring Case study WSTAR - Wasa Zero Emission Data Centre

---

Date: 27.05.2025 Number of pages: 34 Appendices: 10

Abstract

Cooling systems account for 30 – 40 % of the energy required in data centres. The thesis was developed as a part of the WSTAR project. WSTAR – Wasa Zero Emission Data Centre is a research facility that is built up in Technobothnia, enabling research for climate-neutral data centres for the future. Data centres require certain environmental conditions to operate optimally and ensure the durability and optimal functionality of the IT equipment they contain.

A study was conducted on a selection of various low-cost digital sensors. An IoT prototype device was developed using the selected sensors as a proof of concept.

Temperature at different altitudes within a data centre is of particular interest for maintaining optimal operating conditions for servers and other components housed in rack cabinets. The objective is to develop an IoT sensor that measures temperature in a linear vertical profile, using multiple temperature sensors evenly distributed from the ceiling to the floor.

Future improvements to the IoT sensor could include the integration of additional sensors, such as VOC and CO2 gas sensors. Only the proof of concept was done. The physical construction of the IoT device using a custom PCB and integration of the vertical temperature sensing unit were not completed.

Language: English
Key Words: IoT, ESP32, DS18B20, AM2320, LDR, PIR, temperature and humidity measurements

Table of Contents

Table of figures

Table of Listings

List of Abbreviations

CO2	Carbon dioxide
GPIO	General Purpose Input/Output
I^2C	Inter-Integrated Circuit
IC	Integrated circuit
IoT	Internet of Things
IDE	Integrated development environment
LDR	Light Dependent Resistor
MCU	Microcontroller Unit
PCB	Printed Circuit Board
PIR	Passive Infrared
SCL	Serial Clock Line
SDA	Serial Data Line
VOC	Volatile Organic Compound

1 Introduction

This thesis is about the design, development and evaluation of a low-cost IoT (Internet of Things) system for indoor environmental monitoring. To achieve this, open-source software is used in combination with low-cost, readily available hardware components to collect environmental data of indoor spaces. This is to show practical applications of IoT in indoor environmental data collection. These data could be used for environmental monitoring or integrated into an automation system for controlling heating and cooling within a space.

1.1 Background

Enclosed indoor spaces need accurate real-time environmental monitoring. Examples include data centres where optimal temperature and humidity values are required for equipment durability and functionality. Public spaces, for example, during the weekends when they are unoccupied, may not need as much cooling or heating to reduce operational costs. Other examples are remote or unoccupied indoor spaces like storage spaces that need to be maintained at a temperature slightly above freezing to prevent ice-related damage to pipes. In other cases, just enough cooling to prevent overheating. Temperature in an enclosed space can also vary depending on the height above the floor, often resulting in a vertical temperature gradient. Heat-producing sources like a server rack could also result in local variations in temperature. These are conditions where spaces need to maintain certain environmental parameters. Temperature variation can also demonstrate an inefficiency in the cooling or heating method used in an enclosed space.

All these examples require environmental measurements to help in decision-making. Traditional systems could easily become expensive and are usually not flexible, or require multiple equipment to take the measurements. It could also be difficult to collect the data and use it for other purposes.

There has been a lot of advancement in the field of IoT, and it is becoming very popular for environmental monitoring applications. IoT devices can provide real-time data and help with proactive decision making and evaluation or fine-tuning of cooling or heating systems.

1.2 Problem Statement

WSTAR – Wasa Zero Emission Data Centre is a small data centre currently being built in Technobothnia. It is a facility that enables research on climate-neutral data centres for the future. It is designed to operate without generating carbon emissions, which will reduce the impact of data processing and storage on the environment. Achieving zero emissions involves a lot of variables, one of which is energy efficiency. Cooling systems account for 30 to 40% of energy use in data centres. (Moghadampour, 2024).

To achieve this goal of zero emissions, it is important to monitor the indoor environmental conditions and evaluate the amount of energy required to maintain the optimal conditions in the data centre.

A scalable solution that can also be moved to different sites can identify hot spots or variations in environmental conditions.

1.3 Objectives

The goal of the thesis is to proffer a workable solution by designing, implementing and evaluating a scalable IoT-based indoor environmental monitoring system. This system will use multiple sensors connected to an ESP32 MCU. This thesis will focus on the information technology aspect, the development and coding using the Arduino IDE. Other areas that affect the project's success are also considered.

To account for the difference in temperature at different heights, a vertical IoT sensor rail is designed to measure the temperature. Additionally, sensors for humidity, light, motion and an LED are integrated as well to get a more general usage of the sensors. The data is transferred using MQTT over Wi-Fi.

The entire system must be cost-effective, scalable and adaptable for data centres and enclosed spaces in general. Specific objectives include:

- Developing a system capable of monitoring temperature at multiple heights using ten DS18B20 sensors.
- Integrating additional sensors (AM2320, EKMC1601111, LDR) to provide a comprehensive environmental overview.

- Utilising the ESP32 microcontroller and Arduino IDE for system control and software development.
- Implementing the MQTT protocol over Wi-Fi for efficient and reliable data transmission.
- Providing visual feedback on system status and alerts using an RGB LED.

1.4 Scope

The scope of this thesis includes the development of a functional prototype and the accompanying software, employing the specified hardware components: an ESP32 microcontroller, ten DS18B20 temperature sensors, an AM2320 humidity sensor, an EKMC1601111 PIR motion sensor, an LDR light sensor, and a common anode RGB LED. The system is designed to monitor key indoor environmental parameters, including temperature at various heights, humidity levels, motion detection, and ambient light intensity. Data transmission is done through the MQTT protocol over Wi-Fi. A watchdog timer is used to restart the system if the system hangs. All software development is conducted within the Arduino IDE.

The focus remains on the collection and potential use of this data for monitoring unoccupied buildings, shown by use cases such as adjusting temperature for energy conservation during weekends and maintaining temperatures above freezing during winter to prevent pipe damage. The selection of low-cost sensors ensures the accessibility and potential for widespread adoption of such a system, while the combination of different sensor types allows for a comprehensive assessment of the indoor environment.

Some aspects of environmental monitoring were considered but were later removed from the scope. These include the integration of VOC and CO2 sensors. Also excluded from the scope are any implementation of data visualisation or data analytics, integration of a database or a user interface. Developing an alert system by email or phone messaging is also outside the scope.

1.5 Significance of the Study

The significance of this study is in its contribution to the development of affordable and customizable solutions for indoor environmental monitoring. (Othman et al., 2024) This thesis can serve as a resource for individuals or organisations seeking to implement similar monitoring systems using readily available and cost-effective components. In addition, the environmental data provided by the system has the potential to help improve energy efficiency in buildings. (Freesi, 2023) Temperature data collected in real-time, especially when it is collected at different heights, helps with the control of heating and cooling systems, thereby reducing energy waste. Another advantage is that the system can improve the safety of buildings by providing information about unusual environmental conditions, for example, unexpected increase or decrease in temperature or the presence of motion during unoccupied periods. (Manx Technology Group, 2021) Such monitoring can provide early warnings of potential problems like heating or cooling system failures or unauthorised entry.

Students and researchers in the field of information technology can get practical learning experience in how to apply IoT technologies in environmental monitoring. Finally, this thesis goes along the trend of developing smart buildings by utilising the capabilities of IoT to obtain better safety and efficient management of resources.

2 Theoretical Background

The theoretical background covers the main knowledge base needed to understand the thesis. It includes the current activities in environmental monitoring, the role of IoT, technical information of the sensors and MCU, and finally, the gap this thesis plans to address.

2.1 Environmental Monitoring in Enclosed Spaces, e.g. Data Centres

Environmental monitoring is important in data centres because they contain expensive IT equipment that requires certain conditions to operate properly and be durable. Data centres generate a lot of heat, and keeping the temperature and humidity stable is necessary to prevent overheating, equipment failure, and downtime. The recommendation by The American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE) is a temperature range of 18-27°C and a relative humidity close to 60%, with the acceptable range of 20-80% for efficient operations. (ASHRAE Technical Committee 9.9., n.d.)

A good way to implement environmental monitoring systems in data centres is to place the temperature sensors strategically. Howell, (2024) recommends positioning sensors at the top, centre and lower sections of the server racks, both on the front and back sides. When such detailed data is not available, optimisation of energy efforts will be based on assumptions and can lead to overcooling or undercooling.

Current methods of monitoring usually involve a sensor placed by the door measuring temperature and airflow, and this does not really measure the real situation.

When environmental monitoring is effective, it not only ensures that the IT equipment stays reliable and lasts longer, but it also helps in energy efficiency by providing real-time data to help optimise cooling systems. In the case of zero-emission data centres, accurate environmental control is even more important. This ensures that cooling systems are operating effectively, as well as maintaining the recommended conditions without using excessive energy.

2.2 IoT in Environmental Monitoring

The Internet of Things (IoT) has helped to advance the field of environmental monitoring significantly by enabling the use of sensors and devices that can connect with each other to collect, transmit, and analyse environmental data in real time over the internet. (Sirre & Benítez, 2024)

IoT systems offer a lot of advantages over traditional monitoring methods, including an increase in accuracy and efficiency in data collection, the possibility for remote monitoring, and the possibility for automation and smart control. In environmental monitoring, IoT devices can measure a diverse range of parameters, for example, temperature, humidity, air quality, light levels, and motion, providing a holistic view of the environment. (Tsang et al., 2024)

The integration of IoT with environmental monitoring has led to the development of smart solutions for a lot of applications, including smart buildings, agriculture, and industrial process control. This has contributed to improvements in efficiency, sustainability, and decision-making. (Lueth, 2020)

2.3 ESP32 MCU, Sensors and Devices

2.3.1 ESP32 MCU

The ESP32 microcontroller is a popular choice for IoT projects because of its powerful features and low cost. (AZ-Delivery, n.d.) It includes a dual-core processor, integrated Wi-Fi and Bluetooth connectivity, a significant number of general-purpose input/output (GPIO) pins, in this case 38 pins, and analog-to-digital converters. The Wi-Fi capability is relevant for IoT applications because it allows for direct connection to internet services like MQTT brokers without the need for additional hardware. In addition, the ESP32 is designed for low-power operation, making it suitable for battery-powered devices, which is an important consideration for many IoT deployments. It has an average operating current of 80mA and an operating voltage of 3.0 – 3.6 V. (Espressif, n.d.-a) Further information on the device can be found in the datasheet in the References. The pin configuration can be found in Figure 20, in Appendix 3.

The ESP32 can be programmed in Arduino IDE, which is a user-friendly development environment. This makes the process of programming and deploying IoT applications simple. Figure 1 shows an image of the MCU:

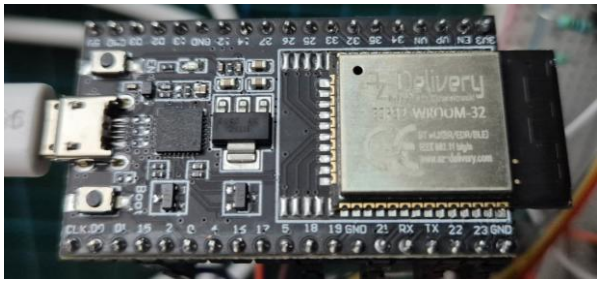


Figure 1: ESP-32 NodeMCU (Photograph by the author, 2025)

2.3.2 DS18B20

The DS18B20 is a digital temperature sensor known for its programmable resolution and ability to measure temperatures over a wide range. It has an accuracy of ±0.5°C for a temperature range of -10°C to +85°C. Its programmable resolution, ranging from 9 to 12 bits, allows users to choose between faster readings with lower accuracy or slower readings with higher accuracy, depending on the requirements. A key feature of the DS18B20, particularly relevant to multi-sensor systems, is its unique 64-bit serial code, which allows each sensor to be individually identified. This unique address is important for the project's requirement of knowing the location and height of each of the ten temperature sensors. The DS18B20 uses a 1-Wire interface, which needs only a single data pin for communication, making the wiring simple, especially when multiple sensors are connected to the same microcontroller.(Analog, n.d.-b) The operating voltage is 3.0 – 5.5V, and it requires an external pull-up resistor of 4.7kΩ. Figure 2 shows the pin configuration of the sensor.

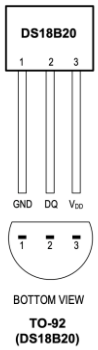


Figure 2: DS18B20 Pin Configuration (Analog, n.d.-b)

2.3.3 AM2320

The AM2320 is a digital sensor that measures both temperature and humidity and communicates using the I2C interface. The I2C protocol allows for communication with the ESP32 using two pins, the serial data line (SDA) and the serial clock line (SCL), which is an efficient way to connect multiple peripherals to the same two lines. The sensor measures relative humidity, which is useful for the developed system. It has an accuracy level of 3% for Humidity measurements and ±0.5°C for temperature measurements on a range of -40 to 80°C. The SDA and SCL pins need a pull-up resistor of 4.7kΩ (Adafruit, n.d.), making it suitable for a variety of indoor environmental monitoring applications. The operating voltage is 3.1 – 5.5V. The sensor has a default address of 0x5C, which cannot be changed. When using more than one of these sensors or similar sensors with fixed addresses, special measures must be taken, since two sensors with the same address are not allowed on the same wires (bus). Figure 3 shows the AM2320 sensor and pins.

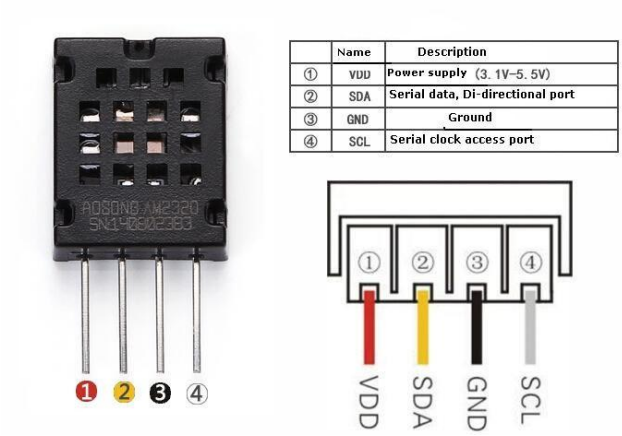


Figure 3: AM2320 pins (EngDial, n.d.)

2.3.4 PIR Sensor EKMC1601111

The EKMC1601111 is a Passive Infrared (PIR) motion sensor that outputs a digital signal when motion is detected. The digital output simplifies its interface with the ESP32, directly indicating the presence or absence of movement within its detection range. The sensor has a sensing range of up to 7 meters, which is appropriate for indoor environments. Its operating voltage of 3.0 -6.0 V and current consumption are suitable for low power applications. The PIR sensor has a defined detection area of 94° (±47°) on the horizontal and 82°(±41°) on the vertical, and the direction in which it is mounted can affect its sensitivity to movement.(Panasonic, 2021) PIR sensors are commonly used to detect occupancy in buildings, making them valuable for identifying whether a building is

unoccupied, which can inform energy-saving strategies and security measures. Figure 4 shows the pins of the PIR Sensor.

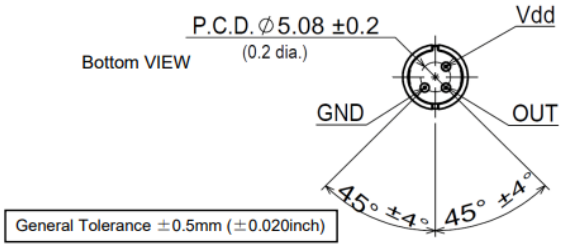


Figure 4: EKMC1601111 pins (Panasonic, 2021)

2.3.5 LDR PGM5526

A Light Dependent Resistor (LDR) is a sensor whose electrical resistance changes depending on the amount of light falling on it. (Soldered, n.d.) In dark conditions, the resistance of an LDR is high, and it decreases as the light level increases. LDRs are relatively simple and inexpensive components for basic monitoring of ambient light levels, although their response time to changes in light intensity can be slower compared to other types of light sensors.(LEDnique, n.d.) The LDR changes resistance depending on the light level, but resistance cannot be measured by the ESP32. Therefore, a voltage divider is needed to convert a resistance change to a voltage. A 10kΩ resistor was connected in series with the LDR. Figure 5 shows the LDR with dimensions.

Figure 5: LDR sensor (FindIC, n.d.)

2.3.6 Common Anode RGB LED

A common anode RGB LED is a type of LED that contains red, green, and blue light-emitting diodes within a single package, all sharing a common positive (+) terminal. (Urias, 2024) Figure 6 shows the pins for a common anode RGB LED.

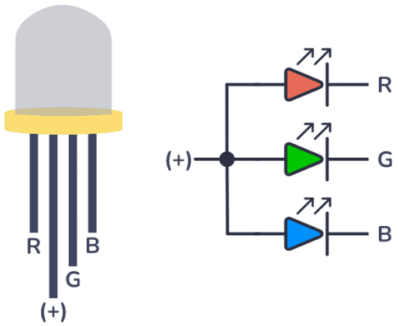


Figure 6: Common Anode RGB LED (Urias, 2024)

In this configuration, to turn on a specific colour, the corresponding negative (-) terminal (cathode) needs to be pulled low via a resistor to limit the current. Controlling the brightness of each colour component can be achieved using Pulse Width Modulation (PWM) if the microcontroller pins connected to the LED cathodes support this feature. RGB LEDs are versatile components that can be used to indicate various system states through different colours and blinking patterns. (Vogelman, 2023)

The LED needs resistors because the forward voltage for each colour channel is less than the operating voltage of the MCU. The Red channel is 1.8 – 2.4 V, green is 2.8 -3.6 V, and blue is 2.8- -3.6 V. The resistors limit the current, and the current controls the brightness.(Lumentheus, n.d.)

2.4 MQTT IoT Transport

The MQTT (Message Queuing Telemetry Transport) protocol is a lightweight messaging protocol based on the publish/subscribe model, making it well-suited for IoT applications, especially those involving devices with limited resources like low power and networks that are potentially unreliable. (Bender et al., 2021)

MQTT operates with three main components: a broker, which acts as a central server; publishers, which send messages on specific topics; and subscribers, which receive messages by subscribing to those topics. The use of topics allows for efficient organisation and filtering of data. MQTT also supports different Quality of Service (QoS) levels, which provide varying degrees of reliability in message delivery. For critical data, such as temperature alerts, a QoS level that ensures at least once delivery would be appropriate. (Soni & Makwana, 2017)

The publish/subscribe model of MQTT is particularly advantageous for this thesis, as the ESP32 (acting as a publisher of sensor data) can transmit information without needing to know the specific details or network address of the devices or systems that are receiving and processing the data (subscribers). Figure 7 shows the MQTT Publish-Subscribe Pattern.

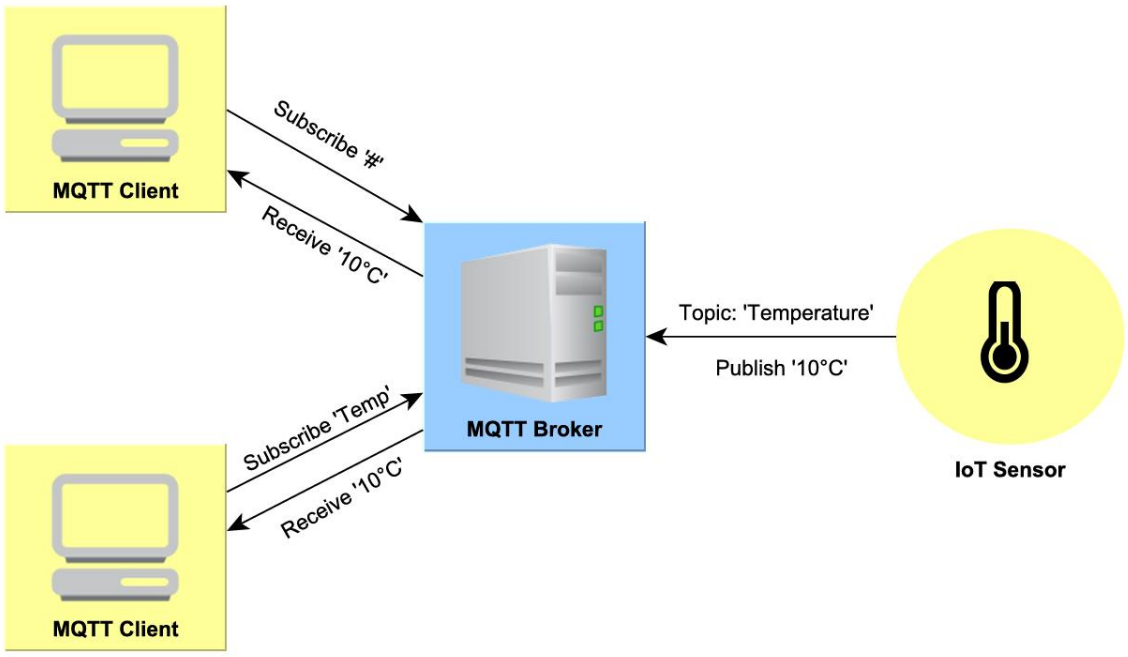


Figure 7: MQTT Publish-Subscribe Pattern (Bender et al., 2021)

2.5 Watchdog timers

Watchdog timers are designed to help embedded systems be self-reliant with minimal human intervention. It improves reliability by automatically detecting and recovering from system malfunctions or hangs by resetting the processor.(Murphy & Barr, 2001)

These timers work using a simple principle: there is a counter counting down to zero, and the main program must reset the counter periodically. If the counter gets to zero without being reset, it shows that the system may have encountered an error or malfunctioned, and the watchdog timer will reset the processor as if a human had done a power cycle. (Murphy & Barr, 2001) Figure 8 shows a typical watchdog setup.

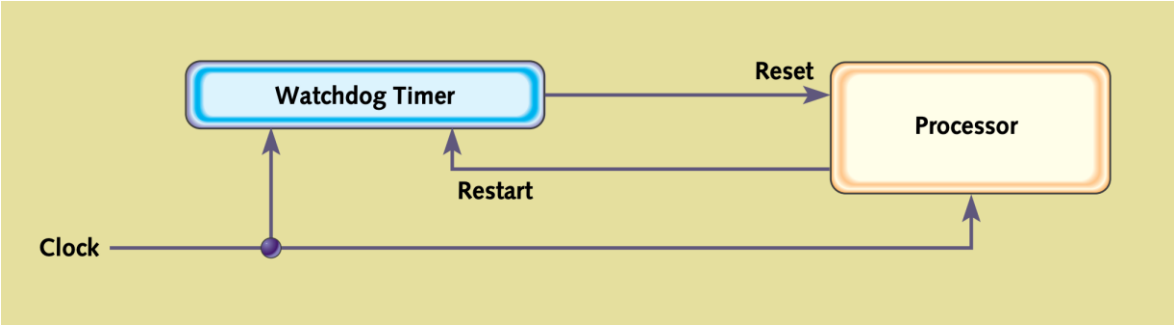


Figure 8: A typical watchdog setup (Murphy & Barr, 2001)

The ESP32 microcontroller includes different types of watchdog timers, such as the RTC watchdog and the Main System Watchdog. (Espressif, n.d.-b) The Main System Watchdog Timer will be used by the Task Watchdog Timer to monitor if the program is running. Using a watchdog timer in the indoor environmental monitoring system is necessary to ensure that it is reliable in the long run, especially when it is used in places where human intervention may not be available.

2.6 Gaps in Current Research

While there is a lot of research on IoT-based environmental monitoring systems, there are still gaps. Many existing studies focus on specific applications like smart homes or agriculture, with limited attention to the requirements of custom environments like zero-emission data centres. More research is needed on cost-effective and scalable systems that can be easily configured and used across different indoor environments where the environmental thresholds and monitoring needs change continuously. In addition, while individual sensor technologies are studied in depth, there is still a need to improve the analysis of the integration and performance of multi-sensor arrays for all-around assessment of indoor environmental conditions.

3 Methodology

3.1 System Design

The system has a hardware and software component to it. The hardware component is mainly centred on the electronics and hardware involved, and the software is centred on the code that drives the system.

3.1.1 Hardware Components

The indoor environmental monitoring system is designed with a two-part hardware architecture. The primary structure consists of a 2-meter vertical arrangement along which ten DS18B20 temperature sensors are positioned at regular intervals. This configuration allows for the collection of temperature data at varying heights, enabling the analysis of vertical temperature gradients within the monitored space. At the top of this vertical structure, a Printed Circuit Board (PCB) is located. This PCB serves as the central hub for the system, housing the ESP32 microcontroller, AM2320 humidity sensor, EKMC1601111 PIR motion sensor, LDR light sensor, and the common anode RGB LED. The ten DS18B20 temperature sensors are connected via wires to the PCB at the top, facilitating data transmission to the microcontroller. The entire system will require a power supply for the ESP32 and the connected sensors; the specific type of power supply (e.g., USB connection or an external power adapter) is determined based on the final deployment scenario. This vertical arrangement of temperature sensors is specifically intended to enable the creation of a temperature profile along the 2-meter structure, which can be particularly useful for analysing temperature at different heights in a room or for monitoring the heat generated along vertical equipment such as a server rack.

3.1.2 Software Components

The software architecture of the system is centred around Arduino code running on the ESP32 microcontroller. This code will perform several key functions:

- Initialising all the sensors and the RGB LED connected to the ESP32.
- Reading data from each sensor at predefined time intervals.

- formatting the collected sensor data into a structured JSON message format,
- Establishing a connection to an MQTT broker over a Wi-Fi network.
- publishing the formatted sensor data to specific MQTT topics.
- controlling the RGB LED to provide visual feedback on the system's transmission status and any detected errors; and
- Implementing a watchdog timer to continuously monitor the main program loop for potential hangs or malfunctions.

This structured software approach is essential for efficiently managing the data streams from multiple sensors and ensuring the overall reliable operation of the monitoring system. By clearly separating the tasks of sensor reading, data formatting, MQTT communication, LED control, and watchdog management within the software, the maintainability and long-term reliability of the code are significantly improved.

3.2 Development Process

The development process for this thesis involves several stages.

3.2.1 Prototype Design

Initially, the prototype is constructed on a breadboard to facilitate easy wiring and testing of the hardware components. A schematic diagram is shown in Figure 9Chapter 4, section 4.1, is created to ensure correct connections between the ESP32 and the other sensors, as well as the RGB LED. Eventually, the ten temperature sensors are mounted on a two-meter vertical structure at regular intervals along its length. A PCB (Printed Circuit Board) could be designed to house the ESP32 and other sensors and the RGB LED, with the temperature sensors wired to the appropriate pins on the PCB

3.2.2 Integration

After the hardware assembly, the software implementation phase follows. This involves setting up the Arduino IDE for ESP32 development and installing the necessary libraries required for interfacing with each of the sensors. A key aspect of the software development

is to uniquely identify each of the ten DS18B20 temperature sensors using their individual addresses. The DallasTemperature library provides functionalities to discover and retrieve these unique 64-bit addresses, which will then be stored in an array, associating each address with a specific height or identifier along the vertical structure. Functions are implemented to read temperature data from each sensor using its unique address, as well as to read humidity and temperature from the AM2320 sensor, the state of the PIR motion sensor, and the light level from the LDR sensor.

An MQTT client library is integrated into the code to enable communication with an MQTT broker. The sensor data is formatted into MQTT messages and published to designated topics, ensuring that temperature readings include the unique identifier of the originating sensor. Control logic for the common anode RGB LED is developed to blink the LED in a specific colour at regular intervals to indicate successful data transmission and to change the colour or pattern if a sensor reading fails or a malfunction is detected.

Finally, a watchdog timer is implemented using the ESP32's built-in capabilities to automatically reset the system if it becomes unresponsive. The software implementation phase requires careful attention to the communication protocols of each sensor, the structure of the MQTT messages, and robust error handling to ensure data accuracy, reliable transmission, and overall system stability.

3.2.3 Testing

The final stage of the development process involves thorough testing and validation of the system. This includes verifying the accuracy of individual sensor readings against known standards or calibrated devices, confirming the unique identification of each DS18B20 sensor, monitoring MQTT traffic to ensure data is being published correctly to the intended topics, testing the different RGB LED indicators for various system states, and evaluating the functionality of the watchdog timer by simulating system hangs. This rigorous testing and validation are essential to ensure the reliability and accuracy of the monitoring system, allowing for the identification and correction of any bugs or issues early in the development cycle.

3.3 Evaluation Methods

The performance of the indoor environmental monitoring system is evaluated based on several indicators. These include the accuracy of the individual sensors, the reliability of data transmission using the MQTT protocol. Sensor accuracy is assessed by comparing readings with datasheets and, if possible, calibrated instruments. Data transmission reliability is evaluated by monitoring the success rate of publishing sensor data to the MQTT broker. Case studies will involve deploying the prototype in an unoccupied room and a simulated data centre environment to evaluate its practical application.

4 System Design and Implementation

4.1 Hardware Setup

The hardware setup involves connecting each sensor and the RGB LED to the ESP32 microcontroller. For the DS18B20 temperature sensors, the data pin of each sensor is connected to GPIO15, a designated GPIO pin on the ESP32. Since the DS18B20 uses a 1-Wire protocol, multiple sensors can be connected to the same GPIO pin; however, for individual identification, each sensor's unique address is used in the software. A 4.7kΩ pull-up resistor is required on the data line. Figure 9 shows the schematics.

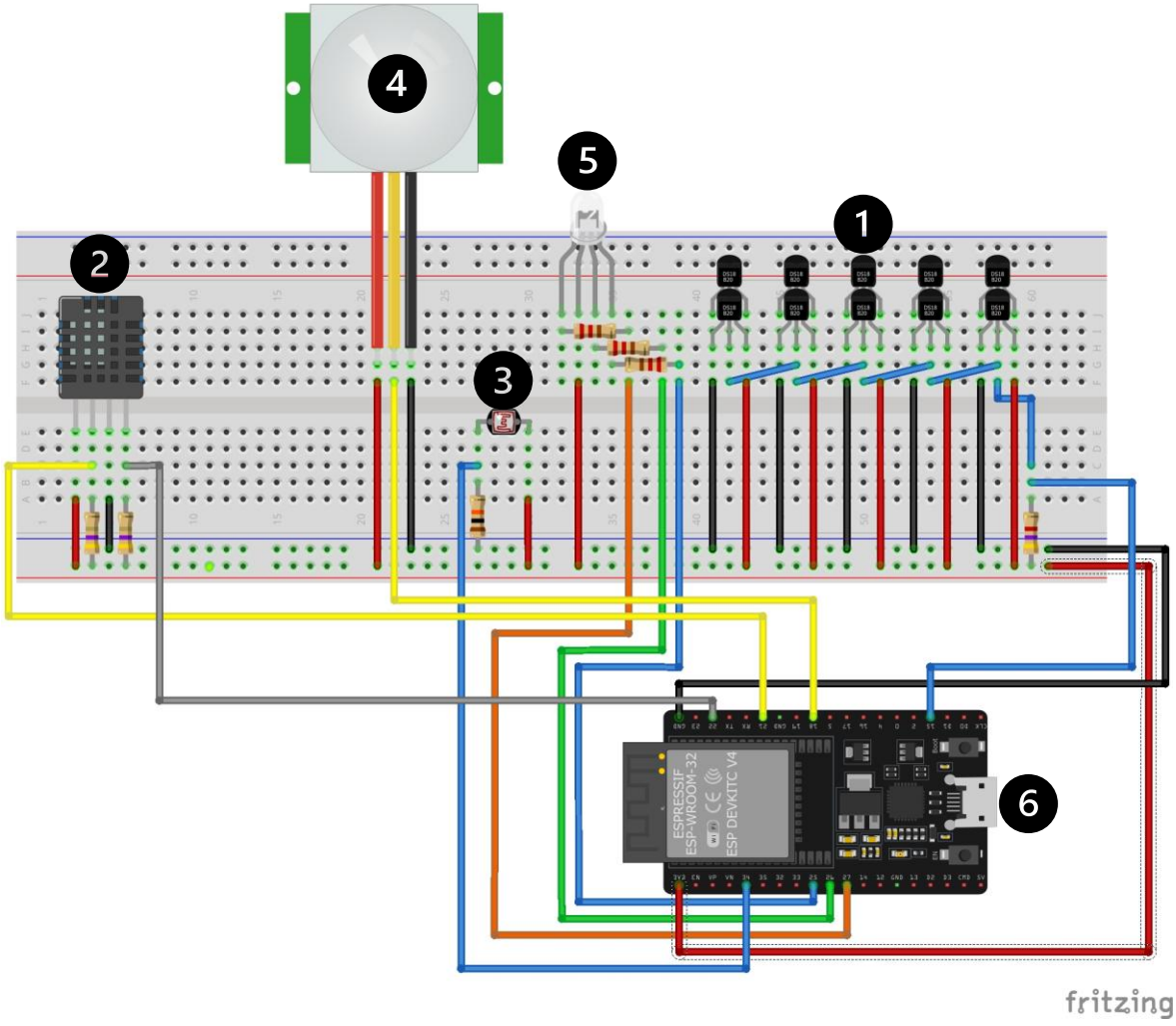


Figure 9: Schematics (1) Ten DS18B20s, (2) AM2320, (3) LDR, (4) PIR sensor, (5) RGB, (6) ESP 32. (Photograph by the author, 2025) made in the Fritzing application.

The AM2320 humidity and temperature sensor, utilising an I2C interface, will have its SDA pin connected to the ESP32's SDA pin GPIO21 and its SCL pin connected to the ESP32's SCL pin GPIO22. The ESP 32 has a dedicated SDA and SCL pin.

The EKMC1601111 PIR motion sensor, which provides a digital output, will have its output pin connected to a digital input pin on the ESP32 GPIO18. The LDR light sensor is connected in a voltage divider configuration with a 10kΩ fixed resistor, and the output of this voltage divider, which varies with light intensity, is connected to GPIO34, an analog input pin on the ESP32. The common anode RGB LED will have its common anode pin connected to the 3.3V power supply. The cathode pins for red, green, and blue are connected to separate digital output pins on the ESP32 GPIO27, GPIO26, and GPIO25, respectively, and a 220Ω current-limiting resistor is connected to each pin. Wiring diagrams detailing these connections are included in the appendices. Given the common anode configuration of the RGB LED, the control logic in the software will need to account for the fact that a low signal on a cathode pin will turn the corresponding LED on. Careful consideration is given to the physical arrangement of the ten temperature sensors along the 2-meter structure to ensure even spacing and representative temperature readings at different heights. Figure 10 shows the hardware setup on the breadboard.

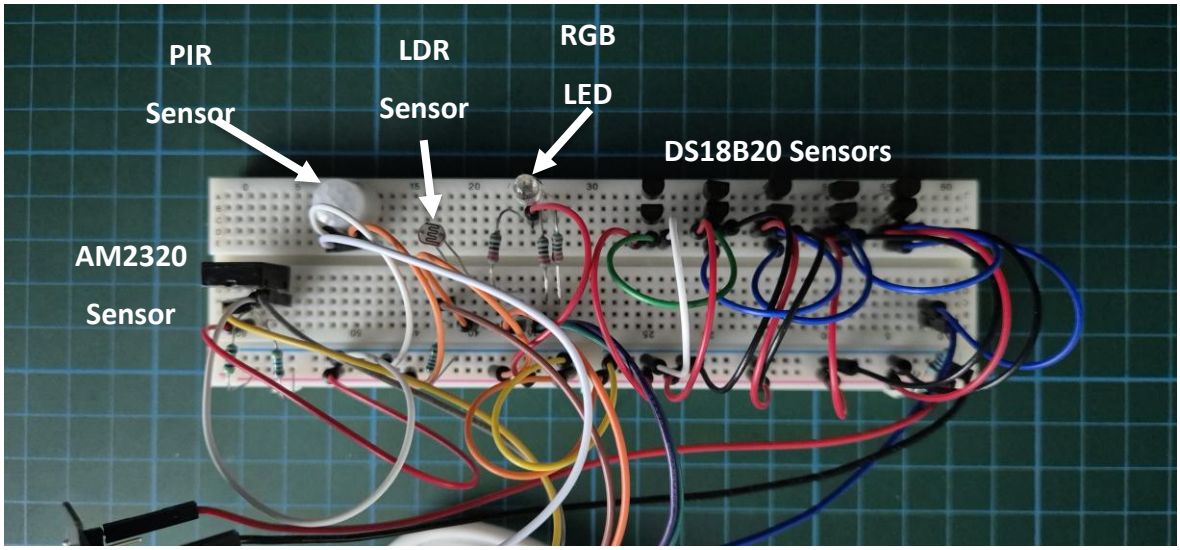


Figure 10: Hardware Setup (Photograph by the author, 2025)

4.2 Software Development

The software is developed in the Arduino IDE. Setting up the software was straightforward but required a little bit of carefulness. The ESP32 board requires a driver to enable it to communicate with the Port, and it may not be installed automatically when it is connected to the computer. The required driver for the board must be installed on the computer used to program it. The instructions for installation can be found in the datasheet of the board. (AZ-Delivery, n.d.)

Next is the initialisation of the various components. For the DS18B20 sensors, the OneWire library is initialised by specifying the GPIO pin connected to the data lines. Subsequently, the DallasTemperature library is initialised. The I2C interface for the AM2320 sensor is initialised using the AM232X library. The GPIO pin connected to the PIR motion sensor is configured as an input. The analog input pin for the LDR is configured for analog reading. The digital output pins connected to the RGB LED are configured as outputs. Figure 11 shows the libraries used.

Figure 11: Libraries used in the project (Photograph by the author, 2025) Snapshot Arduino IDE

An important part of the software is the identification of the DS18B20 sensors. The DallasTemperature library provides functions to discover the number of connected DS18B20 devices and to retrieve the 64-bit address of each sensor. This is implemented by scanning the 1-Wire bus and storing the addresses in an array. Each address will then be associated with a unique identifier, potentially based on its position along the 2-meter structure. When reading temperature data, the code will iterate through this array, using the unique address of each sensor to request and retrieve the temperature reading. The code can be found in Listing 1

Data is read from the AM2320 sensor using functions from its associated library to obtain both temperature and humidity values. The state of the PIR motion sensor is read using the digitalRead() function on the designated GPIO pin. The light level is read from the LDR sensor using the analogRead() function, which will return an analog value that can be correlated with the light intensity. The pin configuration can be seen in Figure 12.

```
32  const unsigned long interval = 5000;
33
34  // ---------------- Define sensor pins ------
35  #define pd_scl_dht 22
36  #define pd_sda_dht 21
37  #define ds18b20Bus 15
38  #define LDR 34  // ** Use an analog pin LDR :
39  #define pirPin 18
40  #define redPin 27
41  #define greenPin 26
42  #define bluePin 25
43
44  // ------------------ Sensor Objects ---------
```

Figure 12: Pin Configuration in Arduino IDE (Photograph by the author, 2025) snapshot Arduino IDE

For MQTT communication, the PubSubClient MQTT client library is used in the Arduino code. The code will establish a Wi-Fi connection to the internet and then connect to the designated MQTT broker. The sensor data is formatted into a JSON string, including sensor identifiers (especially for the temperature sensors), and the corresponding readings. This JSON object will then be published to specific MQTT topics. For example, temperature readings from each of the ten sensors are published to the topic wstar/ds18b20, while humidity, motion, and light data are published to their respective topics wstar/am2320, wstar/pir, and wstar/ldr. Figure 13 shows the flow of data.

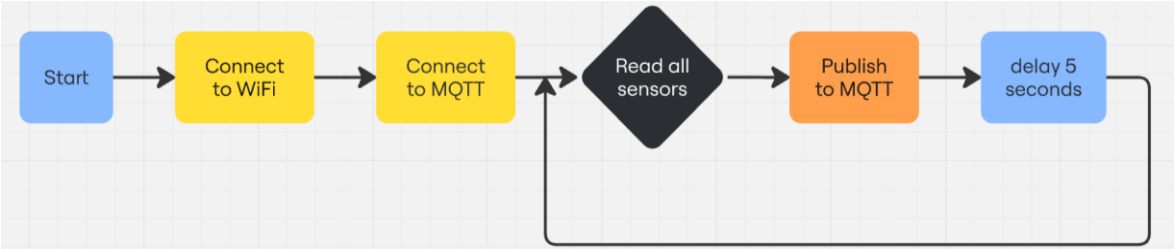


Figure 13: Flow Chart diagram of sensors. (Photograph by the author, 2025) made in Miro

The common anode RGB LED is controlled using the analogWrite() function to turn each color component on or off. To indicate successful data transmission, the LED will blink in a specific color, such as blue, every 5 seconds. If a sensor reading fails (e.g., returns an error code) or if a malfunction is detected, the LED color is changed to red, or a different blinking pattern is used to signal the error.

To enhance the system's reliability, the ESP32's built-in watchdog timer is implemented. This involves including the necessary header file and configuring the watchdog timer with a timeout period of 30 seconds. Within the main program loop, a function call to reset the

watchdog timer is included. If the main loop encounters an issue and fails to execute this reset within the timeout period, the watchdog timer will trigger an automatic restart of the ESP32.

Finally, for the system to function, an MQTT broker needs to be configured. This is done using Novia's cloud-based MQTT service. The configuration involves specifying the broker's address and port in the Arduino code. In the broker's settings, user authentication and the creation of relevant topics are required.

The full code can be found in Appendix 2. Listing 1 shows the code to get the addresses for the DS18B20 sensors and Listing 2 shows the full code for the Arduino IDE.

4.3 Deployment

The prototype is planned to be deployed in two simulated environments: an unoccupied room and a data centre. In the unoccupied room, the focus is on monitoring temperatures near freezing to detect potential pipe freezing risks. In the data centre simulation, the system monitors for overheating conditions. The RGB LED provides immediate visual feedback. The collected data is also published via MQTT for remote monitoring and analysis.

5 Results and Evaluation

As expected, the monitoring system collects data on temperature (from ten sensors), humidity, motion, and light levels.

5.1 Data Collected

Samples of the collected data can be found in figures 14 through 19 below

{ "Sensor" : "DS18B20s",

"Status": "all DS18B20 sensors ok",

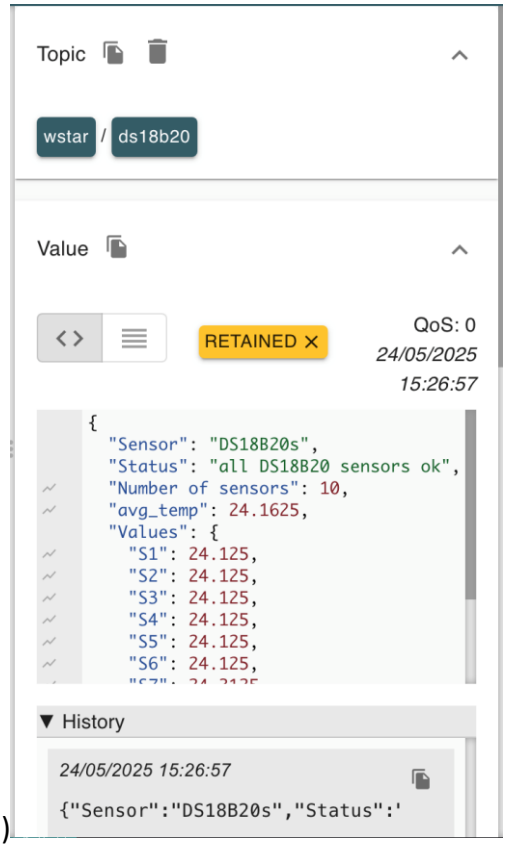
"Number of sensors" : 10,

"avg_temp" : 24.1625,

"Values": {"S1":24.125, "S2":24.125, "S3":24.125, "S4":24.125, "S5":24.125, "S6":24.125, "S7":24.3125, "S8":24.125, "S9":24.1875, "S10":24.25}

}

(a)



(b)

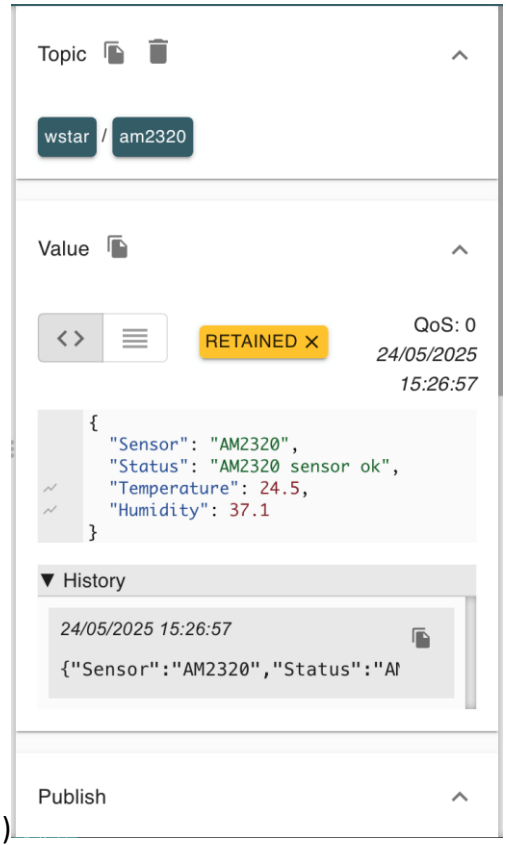


Figure 14: (a) Temperature readings (Snapshot from wstar/ds18b20 MQTT Explorer)

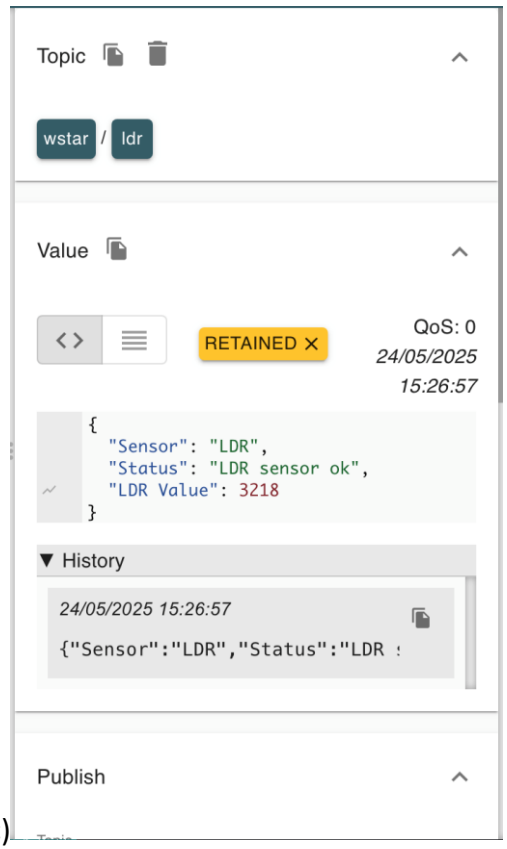
Figure 15: (b) Temperature and Humidity readings (Snapshot of wstar/am2320 MQTT Explorer)

Figure 16: (c) Resistor readings from LDR sensor. (Snapshot of wstar/ldr MQTT Explorer)

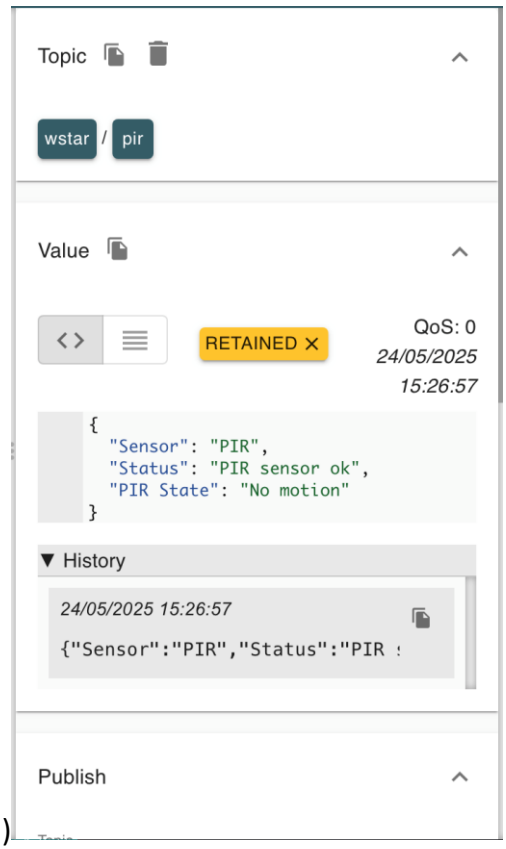
Figure 17: (d) Motion readings from PIR sensor. (Snapshot of wstar/pir MQTT Explorer)

Figure 18: (e) Colour readings from RGB LED. (Snapshot of wstar/rgb_led MQTT Explorer)

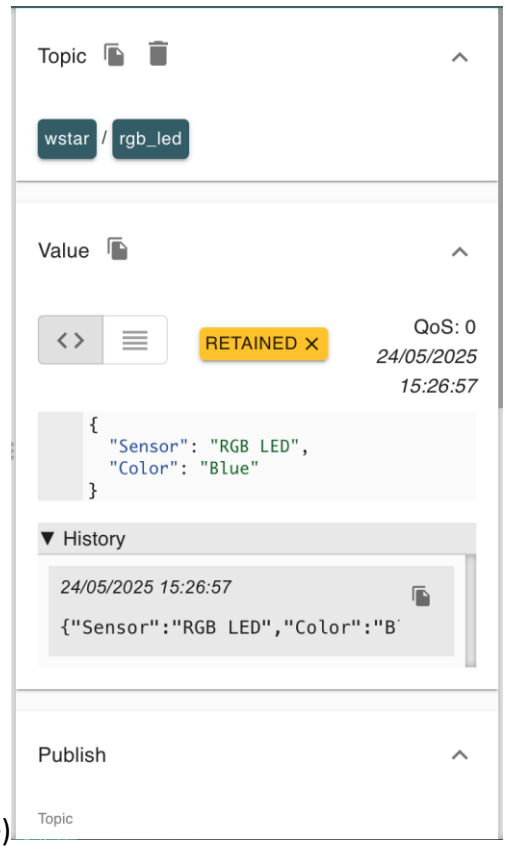
Figure 19: (f) Error readings from all sensors. (Snapshot of wstar/errors MQTT Explorer)



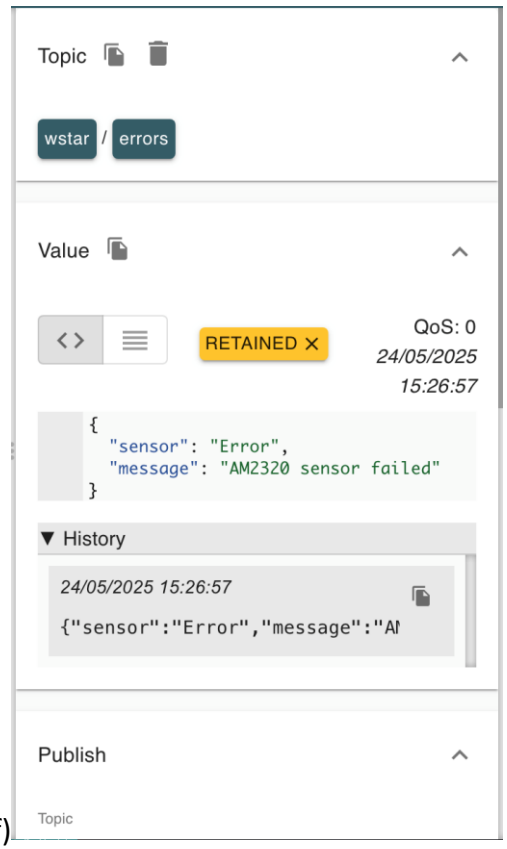
(c)



(d)



(e)



(f)

5.2 Performance Evaluation

The system's performance is evaluated based on its ability to continuously collect and transmit data without errors or interruptions. The reliability is assessed by monitoring the system's uptime and checking for any instances where the watchdog timer might have triggered a reset, indicating a potential software hang. The accuracy of the sensor readings, particularly temperature and humidity, can be evaluated by comparing the collected data with measurements from calibrated reference devices, if available. The power consumption of the system, if measured, will provide insights into its suitability for long-term deployments, especially if battery operation is considered in future iterations. The reliability and latency of the MQTT data transmission will be assessed by monitoring the data flow to the MQTT broker.

5.3 Cost Evaluation

By evaluating the Cost with what is commercially available, the system is considerably lower in cost. But the amount of work required to build it, and the skill level can be prohibitive to an average person. Considering the technology, however, and the simplicity

of the Arduino IDE, a hobbyist or a researcher can pick up the skills quite easily with some time investment.

6 Discussion

6.1 Key Findings

The key findings of this thesis are that the development of a low-cost indoor environmental monitoring system capable of collecting and publishing a range of environmental data, including detailed temperature profiles along a vertical structure, is possible. The integration of various sensors and the implementation of the MQTT protocol for data transmission function as designed. The system has the potential for providing valuable insights into indoor environmental conditions, particularly in the context of unoccupied buildings.

6.2 Limitations

As far as limitations go. The accuracy of the sensors, while generally sufficient for many applications, has inherent limitations as specified in their datasheets; examples are the DS18B20 with an accuracy of ±0.5°C within a certain range. (Analog, n.d.-a), and the AM2320 humidity with an accuracy of ±3% RH (Adafruit, n.d.). The communication range of the Wi-Fi connection may also be a limiting factor, depending on the deployment environment. The resolution of the LDR sensor provides a general indication of light levels but may not be suitable for precise lux measurements.

Challenges encountered during the development process include limitations in packet size for data transfer using MQTT. The libraries and the MQTT explorer have limits on how much data can be transmitted at one time. It will either be truncated or not sent at all. The ESP32 also has a memory limitation, and therefore, the system should be optimised. There are limited possibilities, for example, storing the data in a database or further processing the data in the ESP32.

6.3 Applications Beyond Data Centres

The potential applications of the developed system extend beyond the initial focus on monitoring unoccupied buildings. For example, the system could be adapted for environmental monitoring in smart homes to enhance comfort and optimise energy usage. The detailed temperature monitoring capabilities make it suitable for temperature profiling

in data centres or server rooms to aid in thermal management. With appropriate modifications, the system could also be used in agricultural settings, such as greenhouses, to monitor environmental conditions critical for plant growth.

6.4 Comparison with Existing Solutions

When compared to commercially available indoor environmental monitoring solutions, the developed system offers a unique advantage in its ability to provide detailed vertical temperature profiles using a cost-effective array of sensors. While commercial systems may offer more integrated features or higher accuracy in certain aspects, this thesis demonstrates the feasibility of building a customizable and informative monitoring system using low-cost, off-the-shelf components, tailored to specific monitoring needs.

7 Conclusion and Recommendations

7.1 Summary

This thesis project successfully focused on the design, implementation, and initial evaluation of a low-cost indoor environmental monitoring system. The primary objectives of developing a functional device, programming it to collect data from multiple sensors, including a vertical array of temperature sensors, implementing MQTT for data transmission, utilising an RGB LED for status indication, and incorporating a watchdog timer were achieved. The system demonstrates the capability to gather and publish a range of environmental data, providing insights into indoor conditions. While certain limitations exist, the thesis highlights the potential of using readily available and affordable IoT technologies for detailed environmental monitoring.

7.2 Future Work

At the time of writing the Thesis, the PCB and temperature rail itself were not built. The prototype was designed and tested. In the Future, the work can start from where the prototype stopped. In addition, future work could also involve integrating additional sensors to monitor other environmental parameters, such as air quality, for example, CO2, VOCs, and particulate matter. Implementing an advanced data analysis technique, such as anomaly detection algorithms, could further enhance the system's ability to identify unusual environmental events and improve the system's robustness. The implementation of AI is something that can be considered in future work. Integration with an external device like a Raspberry Pi will also help improve the robustness of the system. There could be some further post-processing done with the data, and the possibility of extra security added easily. There could also be some further integration with a database and a possibility for web development and alert messaging to mobile devices. Developing a user-friendly interface or dashboard for visualising the collected data would improve its accessibility and utility. Exploring alternative communication protocols, such as LoRaWAN, could extend the system's range and reduce power consumption for wider deployment scenarios.

7.3 Recommendations

Based on the findings, several recommendations can be made for implementing similar monitoring systems. Careful consideration should be given to sensor placement to ensure representative data collection for the intended application. Regular maintenance and calibration of sensors, where necessary, are important for maintaining data accuracy over time. For monitoring unoccupied buildings, the system can be effectively used to track temperature trends for energy optimisation and safety, as well as to detect unauthorised activity through motion and light sensors.

8 References

Adafruit. (n.d.). *Adafruit-am2320-temperature-humidity-i2c-sensor*. Retrieved 23 May 2025, from https://cdn-learn.adafruit.com/downloads/pdf/adafruit-am2320-temperature-humidity-i2c-sensor.pdf

Analog. (n.d.-a). *DS18B20 Datasheet and Product Info | Analog Devices*. Retrieved 23 May 2025, from https://www.analog.com/en/products/ds18b20.html

Analog. (n.d.-b). *DS18B20—Programmable Resolution 1-Wire Digital Thermometer*. https://www.analog.com/media/en/technical-documentation/data-sheets/DS18B20.pdf

ASHRAE Technical Committee 9.9. (n.d.). *2021 Equipment Thermal Guidelines for Data Processing Environments ASHRAE TC 9.9 Reference Card*. Retrieved 29 May 2025, from https://www.ashrae.org/File%20Library/Technical%20Resources/Bookstore/Supplemental%20Files/Therm-Gdlns-5th-R-E-RefCard.pdf

AZ-Delivery. (n.d.). *ESP-32_DevKit_C_V4_DataSheet_AZ-Delivery*. Retrieved 10 April 2025, from https://cdn.shopify.com/s/files/1/1509/1638/files/ESP-32_DevKit_C_V4_Datenblatt_AZ-Delivery_Vertriebs_GmbH_24ec770f-c65e-4bd3-92c9-cd64b4d070b8.pdf?v=1615364587

Bender, M., Kirdan, E., Pahl, M.-O., & Carle, G. (2021). Open-Source MQTT Evaluation. *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, 1–4. https://doi.org/10.1109/CCNC49032.2021.9369499

EngDial. (n.d.). *AM2320/AM2320_pins.jpg at master · EngDial/AM2320 · GitHub*. Retrieved 29 May 2025, from https://github.com/EngDial/AM2320/blob/master/AM2320_pins.jpg

Espressif. (n.d.-a). *Esp32-wroom-32d_esp32-wroom-32u_datasheet_en*. Retrieved 2 May 2025, from https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf

Espressif. (n.d.-b). *Watchdogs—ESP32——ESP-IDF Programming Guide v5.4.1 documentation*. Retrieved 30 May 2025, from https://docs.espressif.com/projects/esp-idf/en/v5.4.1/esp32/api-reference/system/wdts.html

FindIC. (n.d.). *PGM5526 datasheet PDF，Token Electronics Industry，Photoresistor; 0.1W(1/10W); 8÷20kΩ; 540nm; Mounting: THT; 150VDC; Ø:5mm，420KB -FindIC.us*. FindIC. Retrieved 31 May 2025, from https://www.findic.us/doc/browser/4LOoZRAZL?doc_id=49984661#locale=en-US

Freesi. (2023, March 29). Indoor Air Quality Monitoring: A Comprehensive Guide. *Freesi*. https://freesi.io/blog/indoor-air-quality-monitoring-a-comprehensive-guide-for-real-estate-professionals/

Howell, J. (2024, October 14). Data Center Environmental Monitoring Ultimate Guide [2024]. *ENCOR Advisors*. https://encoradvisors.com/data-center-environmental-monitoring/

LEDnique. (n.d.). Light dependent resistor (LDR). *LEDnique*. Retrieved 23 May 2025, from https://lednique.com/opto-isolators-2/light-dependent-resistor-ldr/

Lueth, K. L. (2020, July 8). *Top 10 IoT applications in 2020*. IoT Analytics. https://iot-analytics.com/top-10-iot-applications-in-2020/

Lumentheus. (n.d.). *Lumetheus LED Light Emitting Diode 5 mm RGB*. Retrieved 9 May 2025, from https://www.amazon.de/Lumetheus-Light-Emitting-Diode-Pack/dp/B07RRBNR7Q?ref_=ast_sto_dp&th=1

Manx Technology Group. (2021, April 2). Monitoring the indoor environment with IoT. *Manx Technology Group*. https://manxtechgroup.com/monitoring-the-indoor-environment-with-iot/

Moghadampour, D. G. (2024). WSTAR-Wasa Zero Emission Data Center. *AmiES 2024, Tallin, Estonia*, 25. https://international-symposium.org/amies_2024/proceedings_2024/Moghadampour_AmiEs_2024_Presentation.pdf

Murphy, N., & Barr, M. (2001). Watchdog Timers. *BEGINNER'S CORNER, 10/2001*. https://webpages.charlotte.edu/~jmconrad/ECGR4101Common/notes/Watchdog%20Timer%20-%20bcorner.pdf

Othman, H., Azari, R., & Guimarães, T. (2024). Low-Cost IoT-based Indoor Air Quality Monitoring. *Technology|Architecture + Design*, *8*(2), 250–270. https://doi.org/10.1080/24751448.2024.2405403

Panasonic. (2021). *Reference specification PIR MOTION SENSOR 'PaPIRs' EKMC160111*. https://api.pim.na.industrial.panasonic.com/file_stream/main/fileversion/250612

Sirre, A. L., & Benítez, R. (2024). *IOT ENVIRONMENTAL MONITORING SYSTEM USING ARDUINO AND NODEMCU ESP8266*.

Soldered. (n.d.). *CDS Photo Resistor (ldr)*. https://soldered.com/productdata/2015/02/Soldered_PGM5516-MP_datasheet.pdf

Soni, D., & Makwana, A. (2017). *A SURVEY ON MQTT: A PROTOCOL OF INTERNET OF THINGS(IOT)*.

Tsang, T.-W., Mui, K.-W., Wong, L.-T., Chan, A. C.-Y., & Chan, R. C.-W. (2024). Real-Time Indoor Environmental Quality (IEQ) Monitoring Using an IoT-Based Wireless Sensing Network. *Sensors (Basel, Switzerland)*, *24*(21), 6850. https://doi.org/10.3390/s24216850

Urias, O. M. (2024, January 30). A Simple Guide to RGB LEDs. *Build Electronic Circuits*. https://www.build-electronic-circuits.com/rgb-led/

Vogelman, V. (2023, November 1). Arduino RGB LED Guide: Easy Setup and Code Examples. *Build Electronic Circuits*. https://www.build-electronic-circuits.com/arduino-rgb-led/

Appendix 1. Code to get the addresses of the DS18B20 sensors

```
/*
 * Rui Santos
 * Complete Project Details https://randomnerdtutorials.com
 */

#include <OneWire.h>

// Based on the OneWire library example

OneWire ds(15);  //data wire connected to GPIO 15

void setup(void) {
  Serial.begin(115200);
}

void loop(void) {
  byte i;
  byte addr[8];

  if (!ds.search(addr)) {
    Serial.println(" No more addresses.");
    Serial.println();
    ds.reset_search();
    delay(3000);
    return;
  }
  Serial.print(" ROM =");
  for (i = 0; i < 8; i++) {
    Serial.write(' ');
    Serial.print(addr[i], HEX);
  }
  Serial.println("");
}
```

Listing 1: Code for getting DS18B20 Addresses

Appendix 2. Complete Arduino code

```
// ------------------------------------------------------------
#include <WiFi.h>
#include <PubSubClient.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <AM232X.h> // RobTillaart
#include <esp_task_wdt.h>
#include <ArduinoJson.h>
// #include <Wire.h>

// ------------- Configuration -------------------------------
const char* ssid = "WiFi ssid";  // use const to save RAM since it doesn't change
const char* password = "WIFi password";
const char* mqttServer = "iot.novia.fi";
const int mqttPort = 1883;
const char* mqttUser = "username";
const char* mqttPassword = "password";

// ------------------ Topics -------------------------------
const char* topicSummary = "wstar/summary";
const char* topicDS18B20 = "wstar/ds18b20";
const char* topicAM2320 = "wstar/am2320";
const char* topicLDR = "wstar/ldr";
const char* topicPIR = "wstar/pir";
const char* topicRGB = "wstar/rgb_led";
const char* topicErrors = "wstar/errors";

// ---------------- MQTT & WiFi ---------------------------------
WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastPublish = 0;
const unsigned long interval = 5000;

// ---------------- Define sensor pins ---------------------------
#define pd_scl_dht 22
#define pd_sda_dht 21
#define ds18b20Bus 15
#define LDR 34  // ** Use an analog pin LDR is connected to GPIO 34
#define pirPin 18
#define redPin 27
#define greenPin 26
#define bluePin 25

// ----------------- Sensor Objects ---------------------------
OneWire oneWire(ds18b20Bus);
DallasTemperature sensors(&oneWire);

TwoWire I2C_IMU = TwoWire(0);  // I2C1 bus
TwoWire I2C_DHT = TwoWire(1);  // I2C2 bus
AM232X AM2320(&I2C_DHT);

// --------------- RGB colours -----------------------------
int type = 1;  // 0 for common cathode, 1 for common anode RGB LEDs

int red, green, blue;  // variable to store color values

const int color[9][3] = {
```

```
  // Define colors
  { 255, 255, 255 },  // white
  { 255, 0, 0 },      // red
  { 0, 255, 0 },      // green
  { 0, 0, 255 },      // blue
  { 255, 255, 0 },    // yellow
  { 255, 0, 255 },    // magenta
  { 0, 255, 255 },    // aqua
  { 255, 125, 0 },    // orange
  { 0, 0, 0 }         // off
};

const String colorName[9] = {  // Define color names
  "White", "Red", "Green", "Blue", "Yellow", "Magenta", "Aqua", "Orange", "Off"
};

const int wdtTimeout = 30000;        // watchdog timeout in milliseconds set to half a minute
esp_task_wdt_config_t wdtConfig = {  // Initialize watchdog timer
  .timeout_ms = wdtTimeout,
  .idle_core_mask = (1 << portNUM_PROCESSORS) - 1,  // Monitor all cores
  .trigger_panic = true
};

// ----------- Variables to hold sensor readings ----------
float lastAM2320Temp = NAN;
float lastAM2320Humidity = NAN;
int ldrValue = NAN;
bool pirState = false;
float avgTempC = NAN;
String rgbColor = colorName[3];  // Default color

// ---------- Flags for sensor errors ----------
bool ds18b20Error = false;
bool am2320Error = false;
bool ldrError = false;
bool pirError = false;
/*
bool rgbError = false;
*/
bool sensorsOk = true;

// Device Addresses for ds18b20 sensors that way sensors can be identified
DeviceAddress sensor01 = { 0x28, 0x61, 0x64, 0x35, 0xC4, 0x7F, 0xE9, 0x3B };
DeviceAddress sensor02 = { 0x28, 0x61, 0x64, 0x35, 0xC4, 0x60, 0x86, 0xEB };
DeviceAddress sensor03 = { 0x28, 0x61, 0x64, 0x35, 0xC4, 0x2E, 0xD1, 0xF4 };
DeviceAddress sensor04 = { 0x28, 0x61, 0x64, 0x35, 0xC5, 0x96, 0x5B, 0xD9 };
DeviceAddress sensor05 = { 0x28, 0x61, 0x64, 0x35, 0x1F, 0xE, 0xB5, 0xE9 };
DeviceAddress sensor06 = { 0x28, 0x61, 0x64, 0x35, 0x1F, 0x2A, 0xC0, 0xD4 };
DeviceAddress sensor07 = { 0x28, 0x61, 0x64, 0x35, 0xC4, 0x7A, 0xC8, 0xB9 };
DeviceAddress sensor08 = { 0x28, 0x61, 0x64, 0x35, 0x1B, 0xB3, 0x50, 0x2A };
DeviceAddress sensor09 = { 0x28, 0x61, 0x64, 0x35, 0x1F, 0x31, 0x6, 0xC };
DeviceAddress sensor10 = { 0x28, 0x61, 0x64, 0x35, 0x1F, 0x4F, 0x6B, 0xFE };

uint8_t* sensorAddresses[] = { sensor01, sensor02, sensor03, sensor04, sensor05, sensor06, sensor07, sensor08,
sensor09, sensor10 };
const int numDS18B20Sensors = sizeof(sensorAddresses) / sizeof(sensorAddresses[0]);  // Get count dynamically
float tempDS18B20[numDS18B20Sensors];                                                // variable to store temperature array
const int startDS18B20Num = 1;

void setup_wifi() {  // We start by connecting to the WiFi network
```

```
  delay(100);
  Serial.println();
  Serial.print("Connecting to ...");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("Connected to WiFi");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void connectMQTT() {  // MQTT connection function
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    String clientId = "esp32-client-" + String(WiFi.macAddress());
    if (client.connect(clientId.c_str(), mqttUser, mqttPassword)) {
      Serial.println("connected");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      unsigned long startAttemptTime = millis();
      while (millis() - startAttemptTime < 5000) {
        delay(100);
      }
    }
  }
}

void setLEDColor(String inputColor, bool fade = false) {  // RGB LED color control function fades color
  for (int i = 0; i < sizeof(colorName) / sizeof(colorName[0]); i++) {
    if (inputColor.equalsIgnoreCase(colorName[i])) {
      red = color[i][0];
      green = color[i][1];
      blue = color[i][2];
      break;
    }
  }
  // Set the RGB LED color based on the color name
  rgbColor = inputColor;

  if (!fade) {
    if (type == 1) {
      red = 255 - red;
      green = 255 - green;
      blue = 255 - blue;
    }

    // Set the RGB LED color
    analogWrite(redPin, red);
    analogWrite(greenPin, green);
    analogWrite(bluePin, blue);
    return;
  }
  // Apply fading
  int steps = 50;
```

```
  int delayTime = 10;

  for (int i = 0; i <= steps; i++) {
    float factor = (float)i / steps;
    int r = red * factor;
    int g = green * factor;
    int b = blue * factor;

    if (type == 1) {
      r = 255 - r;
      g = 255 - g;
      b = 255 - b;
    }
    analogWrite(redPin, r);
    analogWrite(greenPin, g);
    analogWrite(bluePin, b);
    delay(delayTime);
  }

  for (int i = steps; i >= 0; i--) {
    float factor = (float)i / steps;
    int r = red * factor;
    int g = green * factor;
    int b = blue * factor;

    if (type == 1) {
      r = 255 - r;
      g = 255 - g;
      b = 255 - b;
    }
    analogWrite(redPin, r);
    analogWrite(greenPin, g);
    analogWrite(bluePin, b);
    delay(delayTime);
  }
}

void blinkLED(String targetColor, int times, int duration) {
  for (int i = 0; i < times; i++) {
    setLEDColor(targetColor);
    delay(duration);
    setLEDColor(colorName[8]);  // Off
    delay(duration);
  }
}

void readAllSensors() {  // Sensor reading functions
  sensorsOk = true;
  ds18b20Error = am2320Error = ldrError = pirError = false;
  /*
  rgbError = false;
  */

  sensors.requestTemperatures();  // Read DS18B20 sensors
  int validTempCount = 0;
  avgTempC = 0;
  for (int i = 0; i < numDS18B20Sensors; i++) {
    int num = i + startDS18B20Num;
    float temp = sensors.getTempC(sensorAddresses[i]);
    tempDS18B20[i] = temp;
```

```
    String key = "S" + String(num);
    if (temp == DEVICE_DISCONNECTED_C || temp < -55 || temp > 85 || temp == -127.0 || isnan(temp)) {
      ds18b20Error = true;
      sensorsOk = false;
      publishSensorError("DS18B20 sensor " + key + " failed");
    } else {
      avgTempC += temp;
      validTempCount++;
    }
  }
  avgTempC = (validTempCount > 0) ? (avgTempC / validTempCount) : NAN;

  int status = AM2320.read();  // Read the AM2320 sensor
  lastAM2320Temp = AM2320.getTemperature();
  lastAM2320Humidity = AM2320.getHumidity();
  if (status != AM232X_OK || isnan(lastAM2320Temp) || isnan(lastAM2320Humidity)) {
    sensorsOk = false;
    am2320Error = true;
    publishSensorError("AM2320 sensor failed");
  }

  ldrValue = analogRead(LDR);  // Read LDR value
  if (ldrValue < 0 || ldrValue > 4095) {
    sensorsOk = false;
    ldrError = true;
    publishSensorError("LDR sensor abnormal value");
  }

  pirState = digitalRead(pirPin);  // Read PIR value
  if (pirState != HIGH && pirState != LOW) {
    sensorsOk = false;
    pirError = true;
    publishSensorError("PIR sensor abnormal read");
  }
}

// Function to publish json with shared buffer
void publishJson(JsonDocument& doc, const char* topic) {
  static char buffer[1024];
  size_t len = serializeJson(doc, buffer);
  if (client.publish(topic, buffer, len)) {
    Serial.print("Published to ");
    Serial.println(topic);
    Serial.println(buffer);
  } else {
    Serial.print("Failed to publish to ");
    Serial.println(topic);
  }
}

// Function to publish sensor error messages
void publishSensorError(String message) {
  JsonDocument errorDoc;
  errorDoc["sensor"] = "Error";
  errorDoc["message"] = message;

  publishJson(errorDoc, topicErrors);
}

// Function to publish DS18B20 temperatures
```

```
void publishDS18B20Temperatures() {
  JsonDocument ds18b20Doc;
  ds18b20Doc["Sensor"] = "DS18B20s";
  ds18b20Doc["Status"] = ds18b20Error ? "DS18B20 error" : "all DS18B20 sensors ok";
  ds18b20Doc["Number of sensors"] = numDS18B20Sensors;
  ds18b20Doc["avg_temp"] = avgTempC;

  JsonObject values = ds18b20Doc.createNestedObject("Values");
  for (int i = 0; i < numDS18B20Sensors; i++) {
    int num = i + startDS18B20Num;
    String key = "S" + String(num);
    values[key] = tempDS18B20[i];
  }
  publishJson(ds18b20Doc, topicDS18B20);
}

void publishAM2320Values() {  // Function to publish AM2320 values
  JsonDocument am2320Doc;
  am2320Doc["Sensor"] = "AM2320";
  am2320Doc["Status"] = am2320Error ? "AM2320 error" : "AM2320 sensor ok";
  am2320Doc["Temperature"] = lastAM2320Temp;
  am2320Doc["Humidity"] = lastAM2320Humidity;

  publishJson(am2320Doc, topicAM2320);
}

void publishLDRValue() {  // Function to publish LDR value
  JsonDocument ldrDoc;
  ldrDoc["Sensor"] = "LDR";
  ldrDoc["Status"] = ldrError ? "LDR error" : "LDR sensor ok";
  ldrDoc["LDR Value"] = ldrValue;

  publishJson(ldrDoc, topicLDR);
}

void publishPIRValue() {  // Function to publish PIR value
  JsonDocument pirDoc;
  pirDoc["Sensor"] = "PIR";
  pirDoc["Status"] = pirError ? "PIR error" : "PIR sensor ok";
  pirDoc["PIR State"] = pirState ? "Motion detected" : "No motion";

  publishJson(pirDoc, topicPIR);
}

// Function to publish RGB LED state
void publishRGBState() {
  JsonDocument rgbDoc;
  rgbDoc["Sensor"] = "RGB LED";
  rgbDoc["Color"] = rgbColor;  // Change this to the current color

  Serial.print("Summary JSON size: ");
  Serial.println(measureJson(rgbDoc));

  publishJson(rgbDoc, topicRGB);
}

// Function to Publish all sensor data
void publishSummary() {
  JsonDocument doc;
```

```
  JsonObject temp = doc.createNestedObject("Temperature");
  temp["Temp sensors"] = "DS18B20s";
  temp["No. of sensors"] = numDS18B20Sensors;
  temp["avg_temp of sensors"] = avgTempC;

  JsonObject am2320 = doc.createNestedObject("AM2320");
  am2320["Temp Hum Sensor"] = "AM2320";
  am2320["Temperature"] = lastAM2320Temp;
  am2320["Humidity"] = lastAM2320Humidity;
  /*
*/
  JsonObject ldr = doc.createNestedObject("LDR");
  ldr["LDRSensor"] = "LDR";
  ldr["LDR Value"] = ldrValue;

  JsonObject pir = doc.createNestedObject("PIR");
  pir["PIRSensor"] = "PIR";
  pir["PIR"] = pirState ? "Motion detected" : "No motion";

  JsonObject rgb = doc.createNestedObject("RGB LED");
  rgb["RGBDevice"] = "RGB LED";
  rgb["Color"] = rgbColor;  // Change this to the current color

  Serial.print("Summary JSON size: ");
  Serial.println(measureJson(doc));

  publishJson(doc, topicSummary);
}

void setup() {
  Serial.begin(115200);
  while (!Serial) {
    delay(10);  // wait for serial port to connect. Needed for native USB
  }
  Serial.println("Serial Initialized.");

  setup_wifi();
  // configure MQTT client
  client.setServer(mqttServer, mqttPort);

  // Initialize DS18B20 sensors
  sensors.begin();
  Serial.println("DS18B20 system initialized...");

  // Initialize AM2320 sensor
  I2C_IMU.begin();
  I2C_DHT.begin(pd_sda_dht, pd_scl_dht, 100000ul);
  AM2320.wakeUp();
  delay(2000);
  Serial.println("Type,\tStatus,\tHumidity (%),\tTemperature (C)");

  // Initialize LDR sensor pin
  pinMode(LDR, INPUT);  // Set LDR pin as input
  Serial.println("LDR sensor initialized.");

  // Initialize PIR sensor pin
  pinMode(pirPin, INPUT);  // Set PIR pin as input
  Serial.println("PIR sensor initialized.");

  // Initialize RGB LED pins as outputs
```

```
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  setLEDColor(colorName[8]);  // Set initial color to off
  Serial.println("RGB LED initialized.");

  // deinitialize wdt and initialize it
  esp_task_wdt_deinit();
  esp_task_wdt_init(&wdtConfig);
  esp_task_wdt_add(NULL);  // Add current task to WDT
  Serial.println("Watchdog timer initialized.");
}

void loop() {  // Main loop
  if (!client.connected()) {
    connectMQTT();
  }
  client.loop();  // Keep MQTT alive

  unsigned long now = millis();
  if (now - lastPublish > interval) {
    lastPublish = now;
    readAllSensors();

    // ------ Publish sensor data -----------
    publishDS18B20Temperatures();
    publishAM2320Values();
    publishLDRValue();
    publishPIRValue();
    publishRGBState();
    delay(1000);
    publishSummary();

    if (sensorsOk) {  // Check for sensor errors
      Serial.println("All sensors are OK.");
      setLEDColor(colorName[2], true);  // Set to default color and fade
      delay(250);
      setLEDColor(colorName[3]);  // Turn LED blue
      delay(250);
    } else if (ds18b20Error) {
      Serial.println("DS18B20 sensor error detected.");
      blinkLED(colorName[4], 5, 300);   // Blink yellow LED
      setLEDColor(colorName[3], true);  // Turn LED blue
    } else {
      Serial.println("Sensor error detected.");
      blinkLED(colorName[1], 5, 300);   // Blink red LED
      setLEDColor(colorName[3], true);  // Turn LED blue
    }
  }
  esp_task_wdt_reset();  // Reset the watchdog timer
}
```

Listing 2: Full code for programming device

Appendix 3. ESP32 Pins

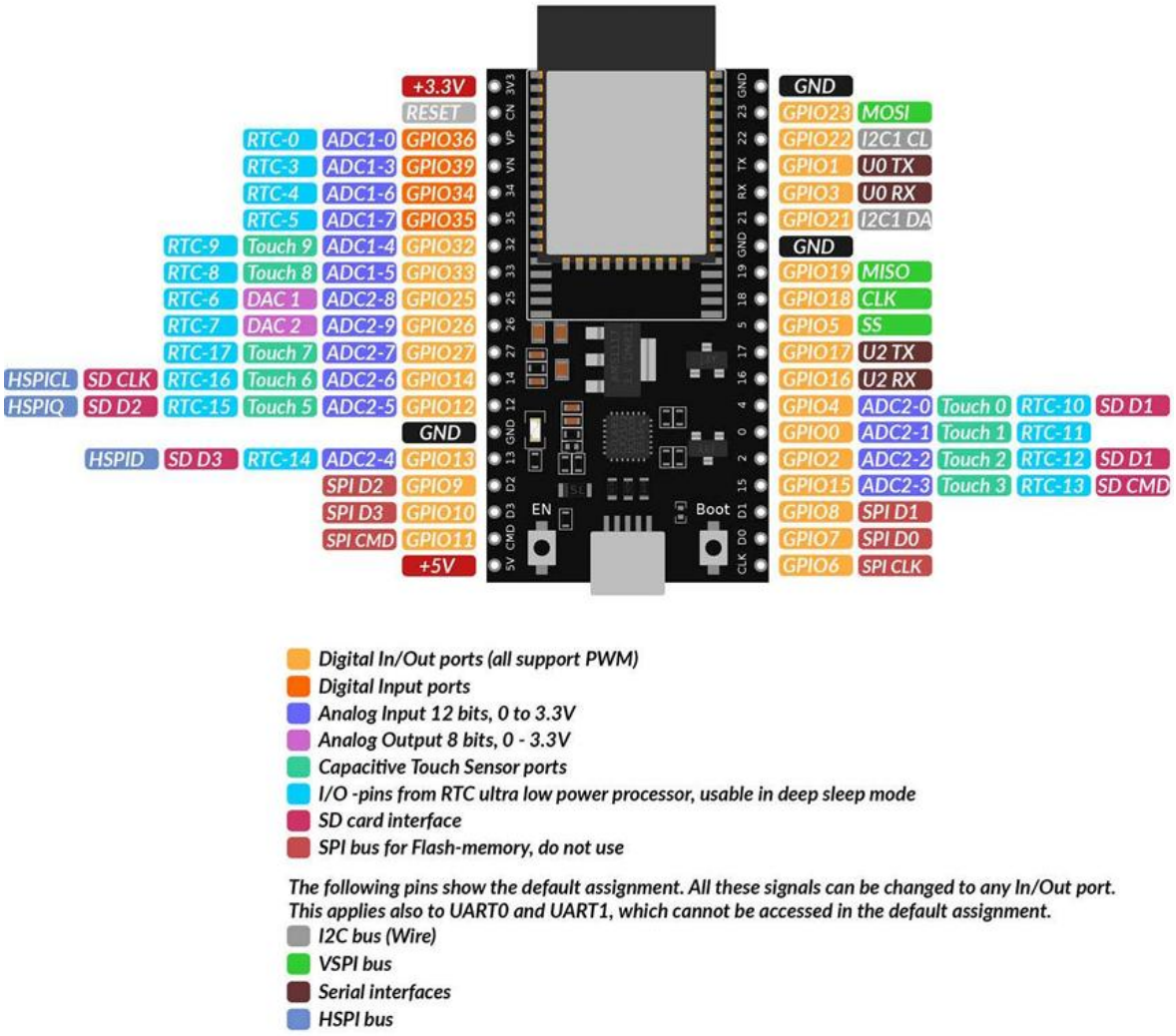


Figure 20: ESP32 Pinout (AZ-Delivery, n.d.)