hardware

MDPI

Article

Low Cost Embedded Systems Applications for Smart Cities

Victoria Alejandra Salazar Herrera [1], Hugo Puertas de Araújo [1], César Giacomini Penteado [2], Mario Gazziro [1] and João Paulo Carmo [3,*]

[1] Federal University of ABC (UFABC), Av. dos Estados, 5001, Santo André 09210-580, Brazil; victoria.herrera@ufabc.edu.br, hugo.puertas@ufabc.edu.br, mario.gazziro@ufabc.edu.br
[2] Universidade de Marília (UNIMAR), Avenida Hygino Muzzy Filho, 1001, - Marília 17525-902, SP - Brasil; cesargiacomini@gmail.com
[3] Group of Metamaterials Microwaves and Optics (GMeta), University of São Paulo (USP), Avenida Trabalhador São-Carlense, Nr. 400, Parque Industrial Arnold Schimidt, São Carlos 13566-590, Brazil; jcarmo@sc.usp.br
* Correspondence: jcarmo@sc.usp.br

Abstract: The Internet of Things (IoT) represents a transformative technology that allows interconnected devices to exchange data over the Internet, enabling automation and real-time decision-making in a variety of areas. A key aspect of the success of IoT lies in its integration with low-resource hardware, such as low-cost microprocessors and microcontrollers. These devices, which are affordable and energy-efficient, are capable of handling basic tasks such as sensing, processing, and data transmission. Their low cost makes them ideal for IoT applications in low-income communities where the government is often absent. This review aims to present some applications - such as a flood detection system; a monitoring system for analog and digital sensors; an air quality measurement system; a mesh video network for community surveillance; and a real-time fleet management system - that use low-cost hardware such as ESP32, Raspberry Pi, and Arduino, and the MQTT protocol to implement low-cost monitoring systems applied to improve the quality of life of people in small cities or communities.

Keywords: Internet of Things - IoT; Smart Cities; Low-end Hardware; Social Applications.

Citation: To be added by editorial staff during production.

Academic Editor: Firstname Lastname

Received: date
Revised: date
Accepted: date
Published: date

1. Introduction

The Internet of Things (IoT) has revolutionized how we interact with technology, creating a vast network of interconnected devices that collect, share, and act on data. This concept, which began with the connection of a vending machine to the internet in 1982 [1], has evolved to encompass billions of devices across numerous domains, including healthcare, industrial automation, agriculture, and smart cities [2].

In healthcare, IoT devices - including wearables [3] - enable remote patient monitoring, fitness activities tracking, and smart medical equipment integration, enhancing treatment outcomes and reducing costs [4]. Industrial settings benefit from IoT through predictive maintenance, efficient resource allocation, and improved safety measures. The integration of IoT in urban environments has given rise to the concept of Smart Cities [5], where advanced technology and data analytics optimize city operations and enhance the quality of urban living.

Smart cities utilize IoT devices and networks to gather and analyze vast amounts of data from various sectors, including energy, transportation, public safety, and healthcare. Integrating data from devices through the cloud (fog) facilitates real-time decision-making, resource management, and improved service delivery [6]. For instance, urban mobility is enhanced through IoT-powered smart traffic management systems that reduce congestion and improve public transportation efficiency.

Energy management in smart cities is transformed through smart grids that dynamically adjust energy supply based on consumption data, reducing waste and carbon foot-prints. Public safety is also improved, with IoT-enabled surveillance systems and predictive policing helping police respond more efficiently, especially in underserved areas.

While smart cities often leverage high-performance IoT systems, there is a growing trend towards utilizing low-end hardware such as microcontrollers and low-cost sensors in IoT applications [7,8], especially in resource-constrained settings. These devices, including microprocessors like the ESP32 or Arduino boards, offer a cost-effective solution to connect everyday objects, creating affordable IoT systems. Low-end hardware is particularly beneficial in scenarios where cost and simplicity are critical.

Despite their limited processing power, these components can efficiently handle real-time data collection and communication, thanks to improvements in lightweight IoT protocols such as MQTT [9] and CoAP. In regions with limited technological infrastructure or low-income populations, the use of low-end IoT hardware provides an avenue for enhancing productivity and connectivity. From improving access to healthcare to enhancing agricultural yields through smart sensors, these technologies make the IoT more accessible, fostering digital inclusion and societal benefits [10].

The rapid expansion of IoT technology presents opportunities and challenges, particularly in ensuring data privacy, security, and scalability. Nevertheless, IoT remains a critical driver of technological innovation and societal transformation, representing the next step in the evolution of urban spaces and addressing modern challenges while improving sustainability and enhancing the lives of citizens, by enabling the policy-makers to be more proactive instead of just reacting to the events.

By addressing these challenges and opportunities, we can unlock the full potential of IoT applications in low-income populations, promoting greater access to essential services and improving the overall quality of life, mainly in areas where the State is absent. Enforcing the use of low-cost hardware such as ESP32, Raspberry Pi, and Arduino and the MQTT protocol for the implementation of low-cost monitoring systems applied to improve the quality of life of people in small cities or communities.

The following sections present some applications: the implementation of a monitoring system for analog and digital sensors based on ESP32 and Raspberry Pi Pico; a flood detection system based on Arduino hardware and AWS cloud services; an air quality mea- surement system based on ESP32 and Raspberry Pi; a mesh video network for community surveillance; and a real-time fleet management system.

2. Design

The Low Cost Embedded Systems Applications for Smart Cities presented in this work are designed based on several devices among them: ESP32 (US$ 9,99), Arduino Uno (US$ 27,00), Raspberry Pi (US$ 25,00) and Raspberry Pi Pico W (US$ 6,00).

Raspberry Pi and Raspberry Pi Pico W are much better options than Arduino Uno both from performance than price, despite the fact that Pico W actually runs Python nowadays.

3. Build Instruction

3.1. Building a Monitoring System for Analog and Digital sensors

People with physical disabilities may need a certain level of automation at home to facilitate daily activities. An example of this is presented in [11], which describes a pilot study conducted with people with physical disabilities at home to investigate the effectiveness of home modification to improve the quality of life. In some cases, simple technologies are enough, for example [12] indicates that Alexa helps consumers with special needs to regain their independence and freedom.

Inspired by such necessity, we work on how to monitor some variables for applications on remote activation of devices at home. Specifically, we describe the use of the ESP32 and Raspberry Pi Pico W in conjunction with MQTT to create a remote monitoring system for sensors. The goal is to promote a communication and monitoring system for variables, whether they are analog or digital, as well as to enable the remote activation of devices, such as turning on lamps, using low-cost components, and the available Wi-Fi network in residences.

For communication via the MQTT protocol, the use of a broker is necessary, which serves as a kind of server that will manage communication between the devices connected to the network. As mentioned before, these devices will be the ESP32, the Raspberry Pico W, and the dashboard that will be used to monitor the variables.

As shown in Figure 1, both the broker and the dashboard will utilize mobile applications that can be downloaded for free, which helps reduce the project's costs.

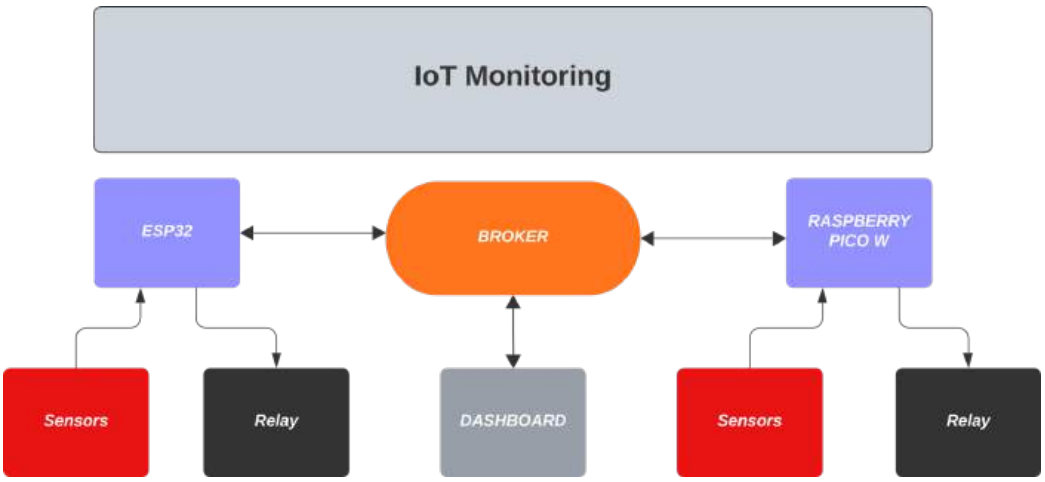

Figure 1. Block diagram of the assembled MQTT network, with the main components.

3.1.1. Materials and technologies applied for the system implementation

For the implementation were used the ESP32 and Raspberry Pico W microcontrollers, as well as temperature and humidity sensors, a reed switch sensor, and a relay for output control. All of them are described in the following.

The DHT22 is a humidity and temperature sensor with a calibrated digital output, ensuring reliability and stability. It has an 8-bit chip and is calibrated in a precise chamber, with the coefficient stored in OTP memory. The sensor is small, low-power, and allows for long-distance transmission (up to 100 m).

The reed switch is a magnetic field sensor that closes its contacts when a magnet approaches and opens them when it moves away. To monitor the opening and closing of doors and windows, a model with a plastic casing was used in order to facilitate installation.

Microcontrollers operate with voltages ranging from 3.3 VDC to 5 VDC, making it necessary to use relays to control devices that require higher voltages. The project used a 5 VDC electromechanical relay, with contacts that can handle up to 8 A, to actuate the door/window latch.

The ESP32 and Raspberry Pico W microcontrollers were used to demonstrate the versatility of the MQTT protocol, which is open and low-cost. Programming was performed in MicroPython, a lightweight version of the Python language optimized for microcontrollers and available under the MIT license. The Thonny software was used to transfer the code to the microcontrollers.

As it is an application for residential use, it is important to have a simple way to control the system for any kind of user, for this reason, some Smartphone Applications

are needed. The MQTT protocol requires a broker to manage messages between connected clients and a dashboard to monitor network information. An application is being used to display temperature values, humidity, reed sensor status, and relay output.

The Narada MQTT Broker is an open-source broker for Android, ideal for prototyping MQTT servers on smartphones. The choice of Android is based on data from Statcounter as of April 2024, which shows that 81.38% of smartphones in Brazil use Android OS, while 18.39% use iOS, [13]. The application allows for configuring access to the MQTT network, including the router's IP address, username, and password for security.

A dashboard can be a mobile application, a PDF file, or a physical screen displaying information. In the context of information technology, it is a visual panel that centralizes indicators and metrics, assisting in decision-making. Technical dashboards analyze device performance and availability, while business management dashboards provide an overview of organizational performance. In this application, the MQTT Dashboard is a mobile application that, when connected to the Broker, receives data such as temperature, humidity, and door status in real-time, also allowing for the configuration of a button to trigger the relay output of the prototype [14].

3.1.2. Implementation steps

We begin this section by describing the electrical diagrams of the circuits that represent the clients controlled by the ESP32 and Raspberry Pico, embedded in them. These circuits will process the sensor signals and send them to the broker via the MQTT protocol. One of the circuits will use the ESP32 as a base, as shown in Figure 2, while the other will use the Raspberry Pico W, as shown in Figure 3, thus demonstrating that the protocol can be used on different platforms.

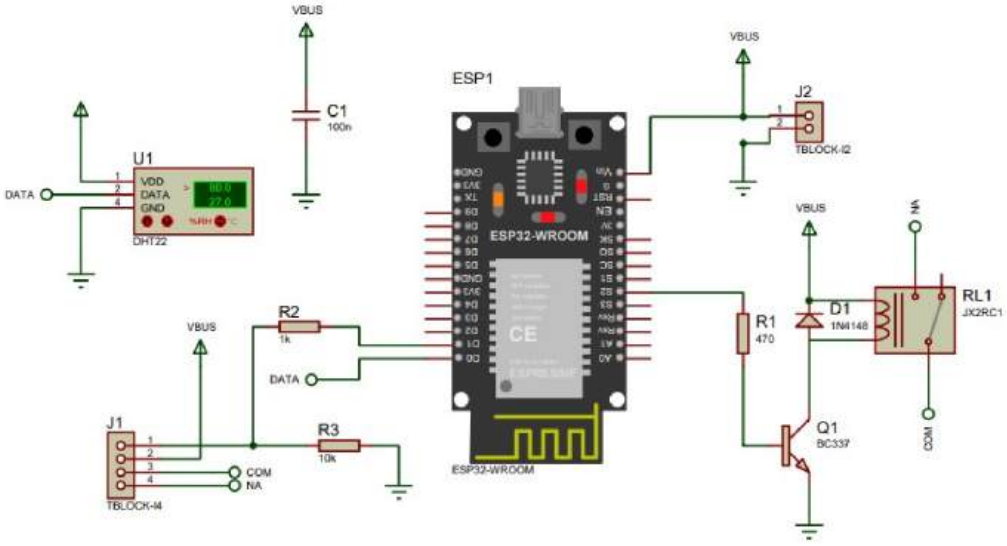

Figure 2. Electrical diagram of the MQTT client that is controlled by the ESP32.

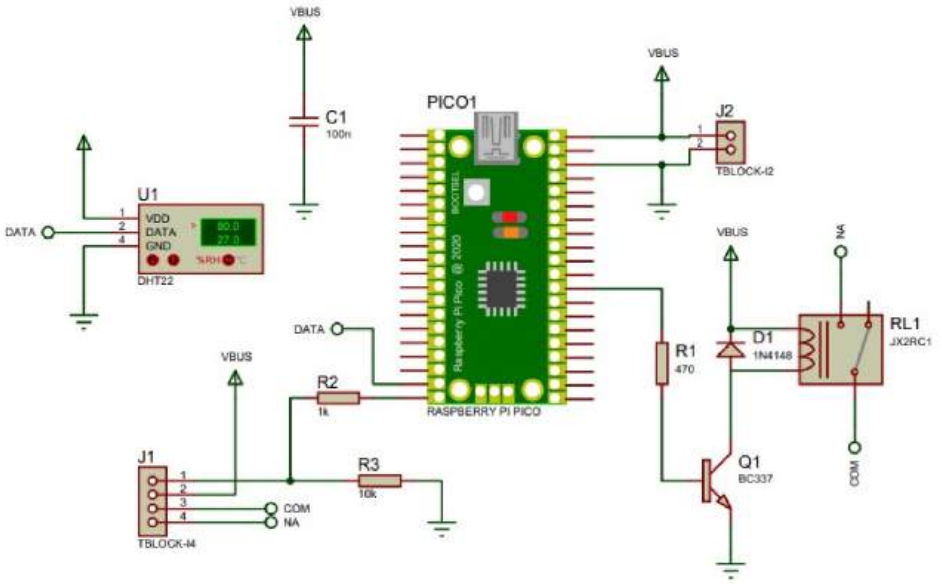

Figure 3. Electrical diagram of the MQTT client that is controlled by the Raspberry Pico W.

For the MQTT Broker, the mobile application Narada MQTT was configured. After the broker's configuration and initialization, we can observe the broker's connection with the prototypes, as it is possible to monitor the activities of publishing and subscribing to the defined topics (C1 = Client1 (RASPBERRY); C2 = Client2 (ESP32); T = Temperature; U = Humidity; R = Reed Switch; S = Relay Output).

Subsequently, the configuration and assembly of the dashboard were carried out. To this end, the mobile application MQTT Dashboard was used. After linking the dashboard icons with the topics, the temperature and humidity data began to be displayed, along with the status icon of the reed switch and the relay output activation icon. After this step, tests were initiated for activating the relay output and testing the status of the reed switch. Example of configuration screens are shown in Figure 4.

Figure 4. Smartphone application interface.

When activating the locking icon of Client1 (relay output, identified as "lock" on the dashboard), it publishes the payload value defined in the program stored on the RASPBERRY to the topic C1/S/. In the case of the ESP32, it will publish to the topic C2/S/ as Client2. The microcontrollers, configured to perform polling every 5 seconds, check for changes in the topic and execute the action defined in the program. In this case, the defined action is to turn on the relay and, consequently, activate the load, which, in this case, is an LED.

Thus, it was concluded that the commands via the dashboard for activating the relay output of both prototypes worked as designed. For the reed switch test, a structure was built in the prototype aimed at simulating the operation of a door or window.

The goal is to show that when the "door" is opened, the reed switch will open its internal electrical contact. Thus, a signal will be sent to an input of the microcontroller that, identifying the change in status, will publish the payload value defined in the program to the topic C1/R/. In this case, the dashboard will receive this information and change the icon representing the door as open/closed.

It can also be observed on the dashboard that the monitoring of temperature and humidity is working normally, updating every 5 seconds.

This time can be adjusted to allow, this and other variables, to poll more frequently. Therefore, the full functionality of the reed switch and its respective indication on the dashboard was confirmed.

It should be noted that the topics identified with "C1" refer to the RASPBERRY prototype and the topics identified with "C2" refer to the ESP32 prototype. The MQTT protocol operated adequately during the tests of the connected devices.

No connection failures between the components were observed, and all commands for activating devices and receiving the status of the variables were executed satisfactorily.

Finally, in Figure 5, we present the diagram showing the entire data flow of the network using the MQTT protocol.

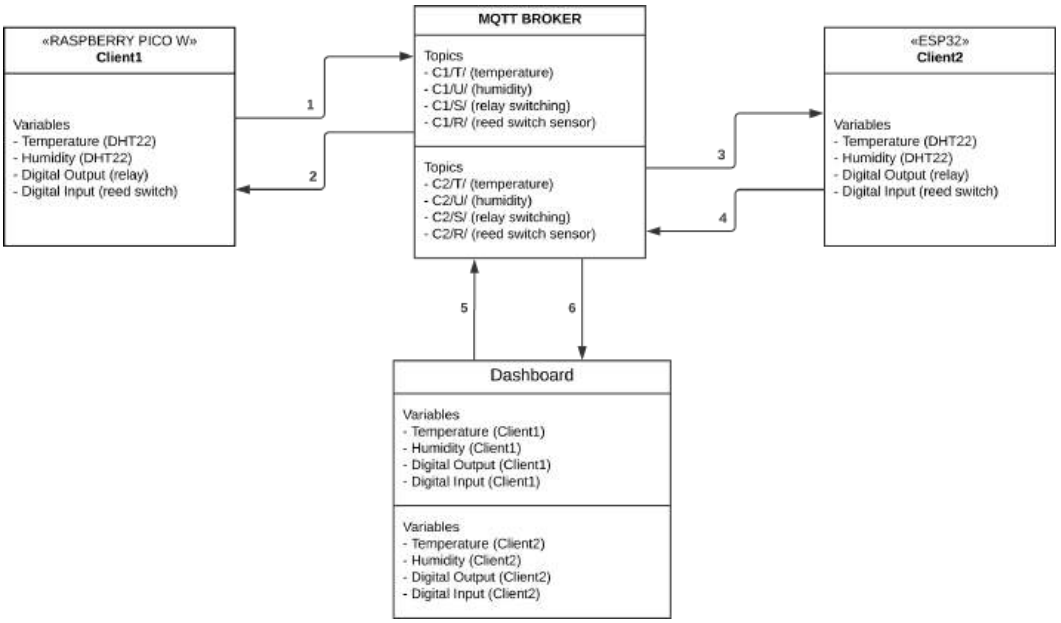

Figure 5. Diagram of the MQTT network showing in detail the communication between the connected components.

For cost and security reasons, a residential router completely disconnected from the internet was used. The system can be used for online monitoring using servers, for example, but additional costs must be considered (use of private brokers, as using public

brokers may lead to data leakage) and also the quality of service (QoS) level, aiming to prevent possible external attacks on the network.

The use of two distinct microcontrollers aims to demonstrate that the protocol can be executed on different hardware, providing more options for evaluation concerning cost/performance factors.

As a point of improvement, it is suggested to implement battery-powered supply sources for all network components in cases where there is a brief power outage. Some attention is needed to the number of devices connected to the MQTT broker to avoid excessive latency, data loss from the topics, or connectivity issues.

3.2. Urban precipitation monitoring system

Climate change has intensified in recent years, and the lack of adequate planning in urban areas further increases the risk of flooding. Given this context, it is crucial to develop more effective ways to alert the population about the danger of flooding and, simultaneously, find ways to minimize the negative impacts that these events can cause. To this end, this study developed a prototype of an Urban Precipitation Monitoring System (UPMS) that integrates hardware, software, and cloud solutions to identify flood events, alerting the community in the affected region as quickly as possible. The system uses sensors connected to an Arduino microcontroller, which processes data in real-time, triggering audible and visual alarms on-site, while sending the data to a public web server. In the cloud, data storage and analysis are performed through Amazon Web Services (AWS), ensuring information security and also sending automatic notifications by email. This work offers an alternative solution that helps people, in a flood event, to make quick decisions to protect their safety and belongings.

3.2.1. System-level description

The Urban Precipitation Monitoring System (UPMS) was developed to detect flooding and send warning messages about imminent danger to residents in affected areas. The system continuously monitors water levels (of a local river, for instance) and when they reach a critical point, automatically triggers alarms, ensuring a fast and effective response in the region where the UPMS is installed.

Recognizing that part of the population may not have access to the internet and, therefore, be more vulnerable in risky situations, the UPMS alarms were divided into two distinct groups, as shown in Figure 6.

Local Group: The alerts are transmitted via an LCD connected to the microcontroller, which displays the flood level, along with an audible warning, alerting people in the area to move away. For this group, the UPMS does not require an internet connection, ensuring that the alert is issued directly on-site.

Mobile Group: The data is transmitted via an Ethernet module connected to the microcontroller, which creates a local Web server. This server is replicated to a public Internet Protocol (IP) Web Server. Using AWS cloud services, data is processed and an alert is generated in the form of an e-mail sent to people registered in the Mobile Group system.

Figure 6 also shows the design of the entire system. A physical device is placed close to the body of water to be monitored. Such a device is composed of an Arduino UNO board, a water-level sensor, an LCD, and a buzzer for visual and audible alarms.

In this simple PoC (Proof-of-Concept) project, the Arduino UNO board was connected to an ethernet shield to provide easy internet access for the system. In a real-world scenario, some form of long-range communication would likely be required, which could range from a simple WiFi connection to a nearby Access Point to a more elaborate solution like LoRA, Sigfox, or even a satellite link. This would depend on the geographic location of the monitored body of water and the local availability of communications technology.

A dedicated web server is deployed locally on the device and its purpose is to communicate (all internet stack: HTML, javascript, CSS, etc.) with another web server hosted elsewhere in the cloud (AWS EC2 or NGROK, for production). The purpose of this server is to provide a public IP for the users' access.

Through the public IP, the application gets access to AWS services, like the ones described in the sequence. AWS IAM is a service that allows you to securely control access and permissions to AWS resources. AWS EventBridge is a serverless service that makes it easy to build applications by connecting components together through events. In this work, EventBridge was used because of its Schedule functionality, which allows you to trigger other AWS services based on time settings. Specifically, it was configured to trigger AWS Lambda every 1 minute.

Amazon S3 is an object storage service that offers scalability, availability, security, and performance. In this work, it was necessary to store the algorithm code in a bucket in S3. Since the package, including the required libraries, exceeded the 10 MB limit allowed for direct upload to AWS Lambda, it was stored in S3. AWS Lambda was configured to point to the package's private link in the bucket, ensuring its execution. Amazon DynamoDB is a fully managed, serverless Not Only SQL (NoSQL) database that delivers sub-ten-millisecond latency performance at any scale. In this application, DynamoDB was chosen for its speed and easy integration with AWS Lambda. The database is used by the algorithm to store water level status, a crucial piece of information for the Lambda function's business logic.

Amazon SNS enables users to receive notifications via push, email, SMS, and other channels, making it easy to send messages to recipients subscribed to a topic. In the context of this project, SNS acts as the output channel, sending email alerts to users subscribed to the topic whenever the Lambda function algorithm detects a flooding situation, as defined in the business rules.

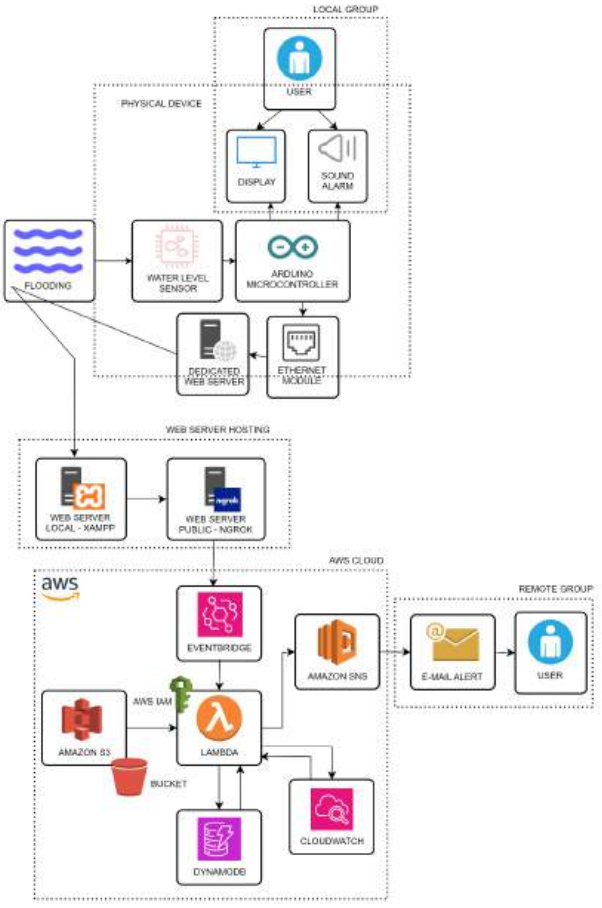

Figure 6. Details of the UPMS components.

Amazon CloudWatch monitors resources and applications running on AWS in real time, allowing the collection and monitoring of metrics that help evaluate the performance of these resources and applications. During the execution of this work, CloudWatch was essential for debugging errors, in addition to providing real-time logs that ensured the observation of application responses.

Finally, the system works in serverless mode, meaning the entire flow of commands and actions is coordinated through lambda functions, instead of a central server. AWS Lambda executes code in response to events, automatically managing compute resources without the need for servers. In this project, AWS Lambda executes code that processes water-level data from the public Web Server. When triggered by EventBridge, Lambda orchestrates the execution, interacting with other AWS services to send the alert. Serverless operation increases the system's scalability, allowing new services to be added, or current services to be expanded, without having to change the fundamental structure of the application.

3.2.2. Hardware implementation

Figure 7 shows a detailed Proteus simulation of the hardware designed for this application.

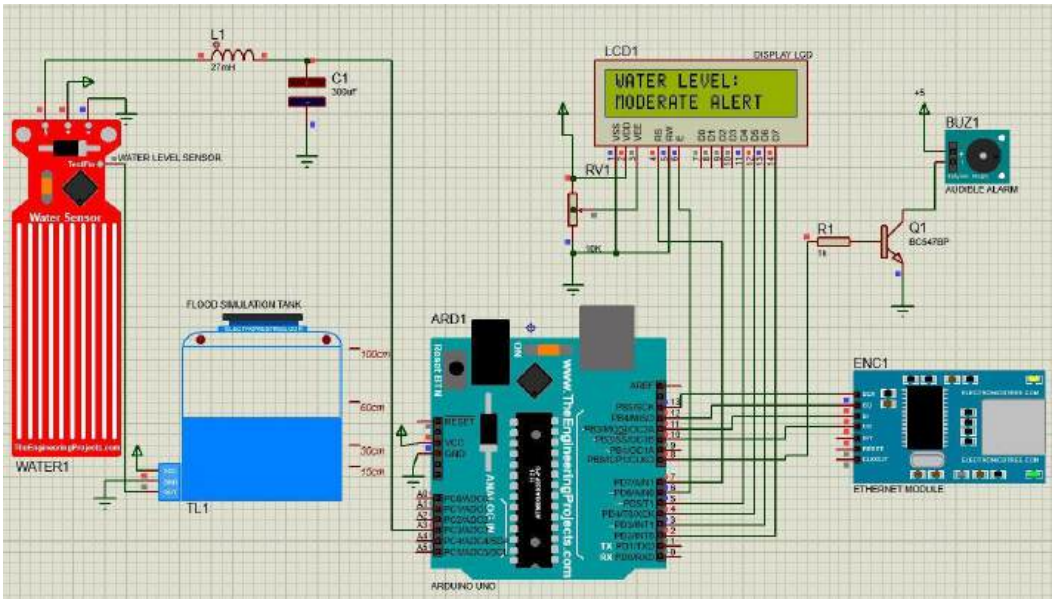

Figure 7. Proteus' simulation of the UPMS.

As mentioned earlier, the Arduino UNO board is the central piece of the system. It executes the embedded code; activates the buzzer delivering a digital high-level voltage to the base of the transistor Q1; sends graphic messages to the LCD; negotiates the communication protocol with the ethernet shield ENC1; and receives water-level data from the sensor WATER1.

Since the water level sensor sends analog signals (a variation in ohmic resistance depending on the presence of water), an LC low-pass filter was provided before the signal reaches the Arduino's analog input port. As can be noticed, the system monitors up to four predefined water levels.

In the circuit of figure 7, RV1 is a potentiometer used to adjust the LCD backlight. In line with the goals of this project, all components are affordable and readily available at local electronics stores.

This ensures that even low-income communities can implement a flood monitoring system, regardless of government support. Receiving early warnings about floods can save lives and help preserve people's property.

3.3. Air Quality Measurement Station

Growing concerns about air quality and its impacts on human health and the environ- ment have driven the search for affordable and effective monitoring solutions, especially for resource-limited communities that face additional challenges in accessing information about air quality and the presence of harmful particles.

For this project, a low-cost air quality monitoring station was created, especially aimed at underprivileged communities without access to environmental measurement data. The idea is to understand how pollution in these places affects air quality and, consequently, the health of the people who live there.

To this end, sensors connected to a microcontroller (Arduino) were used to measure pollutants such as NO2, CO2, smoke, and ozone, as well as temperature and humidity, which are also essential for good air quality.

The collected data is sent to a Raspberry Pi, which functions as a data storage system in a local MySQL database. The Raspberry Pi also has an instance of Grafana installed, which is software that allows the collected data to be viewed on dashboards clearly and in real-time. This way, it will be possible to monitor air quality and take measures to improve the environment.

This monitoring station is a practical and affordable solution that aims to provide essential data to improve air quality and the health of people in underprivileged areas. In addition to monitoring particle concentrations and the presence of harmful gasses, the system also aims to identify pollutants generated by human agglomerations, thus expanding its usefulness in densely populated urban environments. To achieve this purpose, different sensors compatible with the Arduino platform were used, such as the Particle Sensor PMS5003 and the MQ Gas Sensor, allowing detailed and comprehensive monitoring.

3.3.1. System-level description

Figure 8 presents the overall structure of the system. Sensor data is acquired by an Arduino UNO board and sent to a Node-RED instance running in a local server embedded in the Raspberry Pi board. RPi also serves as an instance of Grafana dashboard software.

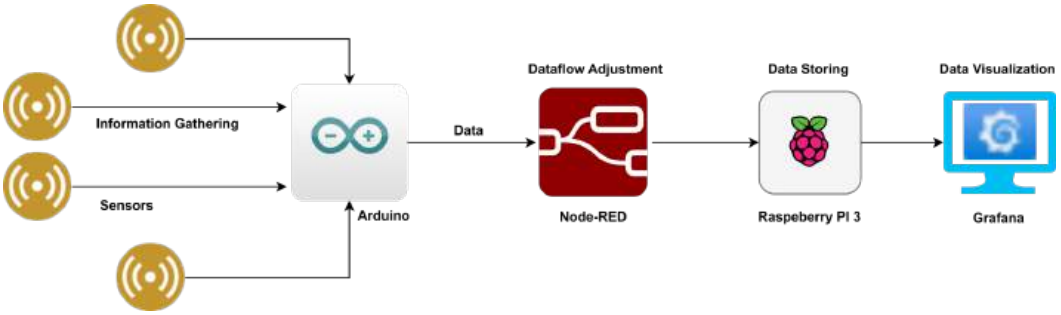

Figure 8. System components.

This setup assumes that all sensor data preprocessing, packaging, and transmission 310 are done by the Arduino UNO system (HW and SW). While the RPi is responsible for data 311 storage and presentation. Here are the software components of the system:

- **Maria DB:** MariaDB is a high-performance, open-source database developed by the community as a fork of MySQL. It is widely used to store data in a variety of formats. Its applications include operations monitoring, IoT sensor data collection, and real-time analytics.
- **Node-RED:** Node-RED is a flow-based, low-code development tool designed for visual programming. The platform is used to wire together hardware devices, APIs,

and online services, enabling developers to build IoT (Internet of Things) applications, automation systems, and other data-driven workflows with minimal coding. Node-RED presents a browser-based editor, allowing users to drag and drop nodes that represent different actions.

- **Grafana:** Grafana is an open-source platform designed for data visualization, monitoring, and analytics. It allows users to query, visualize, and alert on metrics, logs, and traces, regardless of where the data is stored. Grafana is commonly used to create interactive dashboards composed of panels that display data in various forms, such as graphs, charts, and other customizable visualizations.

The system components interact with each other through Node-RED, which orchestrates the acquisition of data from Arduino and its ingestion into MariaDB. They are installed on the Raspberry Pi board.

Figure 9. Node-RED web-based graphical interface (image from Node-RED Programming Guide).

Figure 10. Grafana web-based graphical interface.

Figures 9 and 10 show the graphical interfaces of Node-RED and Grafana. Both are web-based and are accessible through a browser pointing to https://localhost:1880 and https://localhost:3000, respectively.

The Node-RED module plays a crucial role in integrating and storing data collected by sensors. Its main functions are: receiving data from the measurement device, converting this information to JSON format, filtering and formatting the data to make it compatible with database tables, and finally storing it in the MariaDB database instance.

Along with the software components described above, mention should also be made of the Operating System on the Raspberry Pi and the firmware on the Arduino, which was developed using the Arduino IDE as a programming platform.

3.3.2. Hardware Implementarion

Figure 11 shows the physical implementation of the Air Quality Measurement Station in a breadboard. Here, the Arduino board is the central piece of the system, connecting and providing power to the sensors.

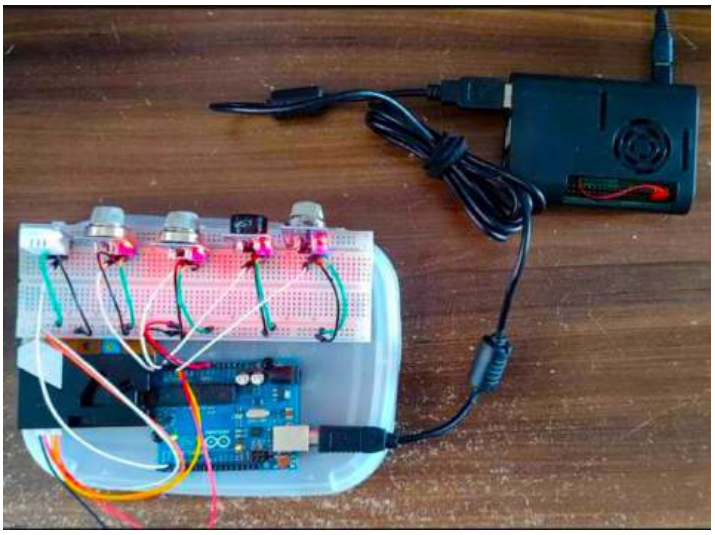

Figure 11. Air Quality Measurement Station.

Below is a list of the sensors that are used in the system. All of them can be easily purchased from local electronics stores at low cost.

- MQ-135: To measure the concentration of gasses such as NO2 and other pollutants.
- MQ-7: For the detection of carbon monoxide (CO).
- MQ-5: To detect a range of gasses including smoke and combustible gasses.
- MQ-131: For ozone (O3) measurement.
- DSM501-A: To measure airborne particles such as PM10 and PM2.5.
- AM2302: To measure the temperature and humidity of the environment.

Sensor measurements are transmitted through WiFi to the server in the Raspberry Pi board. For the interaction with the Arduino hardware, some third-party libraries were used, like the *GasSensor.lib*, for instance. All libraries are pretty available through the IDE interface. Arduino's firmware flow diagram can be seen in figure 12 and is listed below.

- Initializes all Arduino devices and sensors.
- Uses specific library functions to define conversion curve parameters.
- Starts reading the Arduino's analog ports to receive the values measured by the sensors.
- Calls the functions that determine the correct concentration of each pollutant in ppm.

• Inserts the collected data into an object in bytes, which is placed in the transmission queue to be sent to the application in Node-RED.

• After a one-minute delay, the process of reading sensor measurements is repeated in a loop.

Arduino's firmware is implemented in C-language.

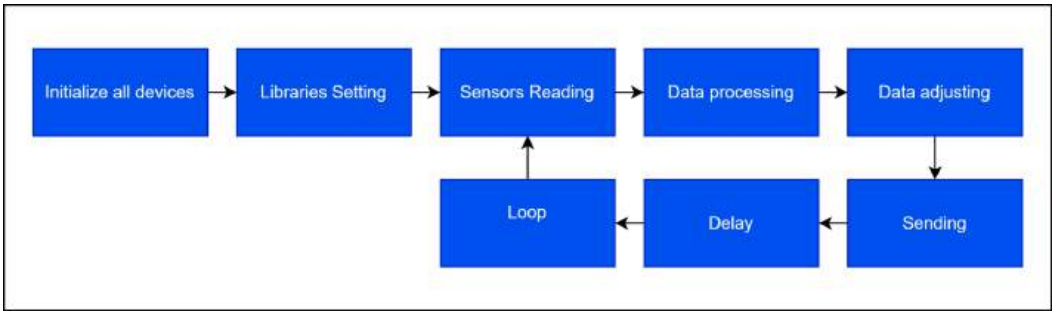

Figure 12. Arduino's firmware flow diagram.

3.3.3. Some results

This section provides some results obtained during the tests of the system. Data monitoring was carried out for eight days in a closed environment, using the prototype mounted on a breadboard. The objective was to observe the variations and performance of the device over this period. The collected measurements were used to calculate the average of the values recorded during the eight days, which were later exported to a spreadsheet for analysis and documentation.

Table 1 presents the results of 8 days of monitoring air quality data and figure 13 shows the graphic for the presence of microparticles.

Table 1. Final results after 8 days of monitoring.

Date	PM2.5 (μg/m^3)	PM10 (μg/m^3)	Smoke (ppm)	CO (ppm)	NO_2 (ppm)	O_3 (ppm)	SO_2 (ppm)	Tmp (ºC)	Hum (%)
07.01.24	4.09	3.49	4.87	4.87	4.81	4.87	0.00	23.6	43
07.02.24	4.08	3.54	4.87	4.87	4.82	4.88	0.01	23.6	44
07.03.24	4.08	3.51	4.88	4.86	4.81	4.87	0.03	23.6	43
07.04.24	4.09	3.49	4.87	4.87	4.81	4.87	0.03	23.6	43
07.05.24	4.09	3.49	4.88	4.87	4.81	4.88	0.04	23.7	43
07.06.24	4.09	3.49	4.87	4.87	4.81	4.87	0.03	23.6	42
07.07.24	4.09	3.49	4.87	4.87	4.81	4.87	0.02	19,2	43
07.08.24	4.09	3.49	4.88	4.87	4.81	4.88	0.01	22.0	43

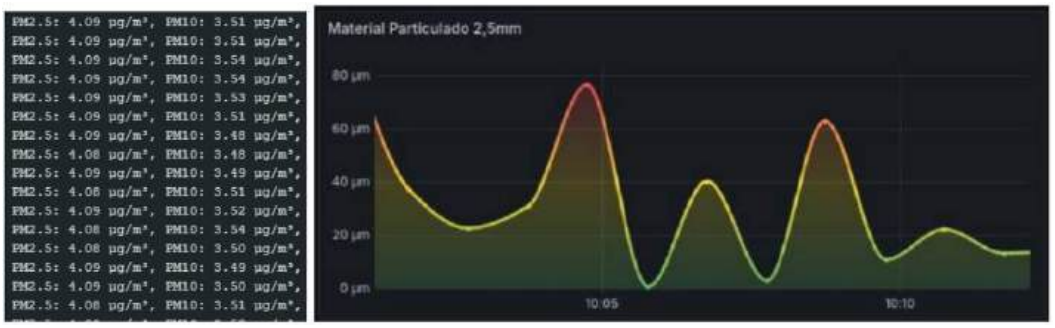

Figure 13. Microparticles of 2.5 μg/m^3 and 10 μg/m^3.

3.4. Mesh Video Network for Community Surveillance

Based on the enormous number of surveillance devices already connected to the internet and the great adherence of people to social networks, it is possible to share images and videos from these devices among members of the same social network - with the appropriate authorizations granted by device owners - to expand the scope of the monitored area, as well as sharing information on past events.

With social networks, combined with applications and information made available by IoT we can have a considerable amount of information according to the article by Luigi Atzori, Antonio Iera, Giacomo Morabito and Michele Nitti [15] and connect communities to a universe of ubiquitous computing, as described by Antonio M. Ortiz, Dina Hussein, Soochang Park, Son N. Han, Noel Crespi [16] and a combination of the Internet Of Things and Social Networks as explained in the article by Lianhong Ding, Peng Shi, and Bingwu Liu [17]. For the development of this mesh video surveillance community, we proposed a hub as shown in figure 14.

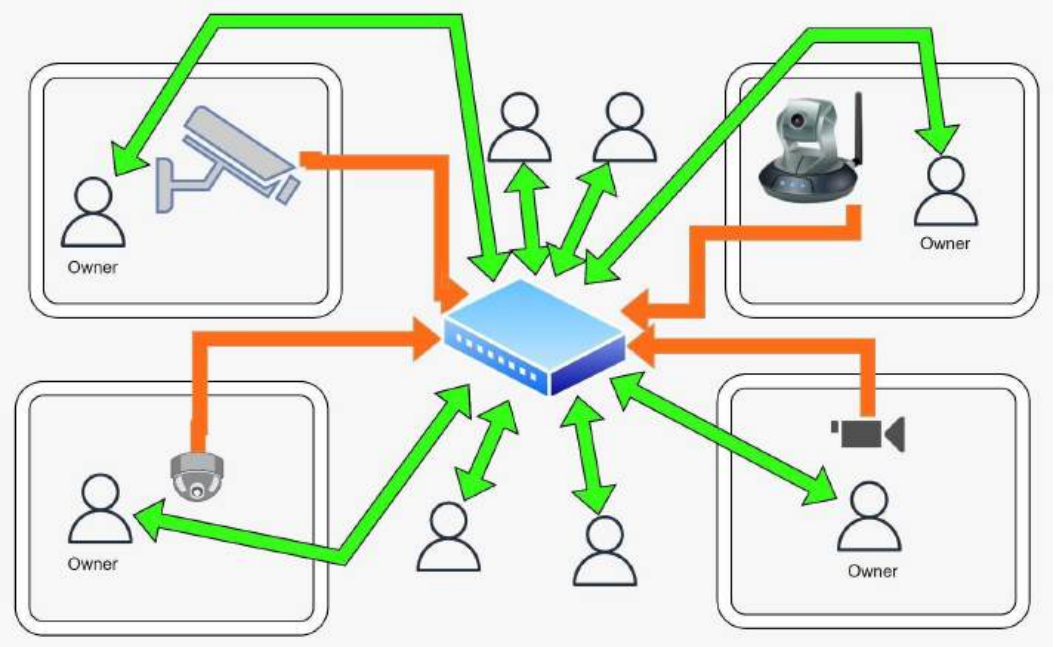

Figure 14. Mesh Community Surveillance Hub with Owners and non-owners (guests).

Spring Tools 4 will be used as an IDE, PostgreSQL 16.2 and Java 11 database. The software was developed using microservices architecture with SpringBoot 3.0.

All modules were placed in Docker images, to abstract the configuration needs of the machine where the hub will be installed. Basically, we have two modules:

- **WrapperGate:** Connection to the company's surveillance and exposure devices and their service interface.

- **MemberWatch:** Membership and device permissions manager registered.

WrapperGate must extract and interact with the service interface of a camera connected to an Orange pi i96 (figure 15). All camera interfaces must be encapsulated by the Wrap- perGate and this will be responsible for abstracting the entire authentication, buffering, and provisioning flow for users who have authorization to extract information output of the device. WrapperGate must export the interface of the device used as a web services standard or REST.

By sharing information captured by security devices, we can increase safety within our communities simply by allowing community members to get access to these images. So any granted members can monitor the surroundings of the community and prevent or inhibit threats.

We can extend this coverage to a whole city and allow authorities, so they can act assertively and preventively to increase the safety of communities.

Also, future image analysis algorithms can identify dangerous situations or threats to the surroundings, receiving as data input the images captured by any device connected to the system.

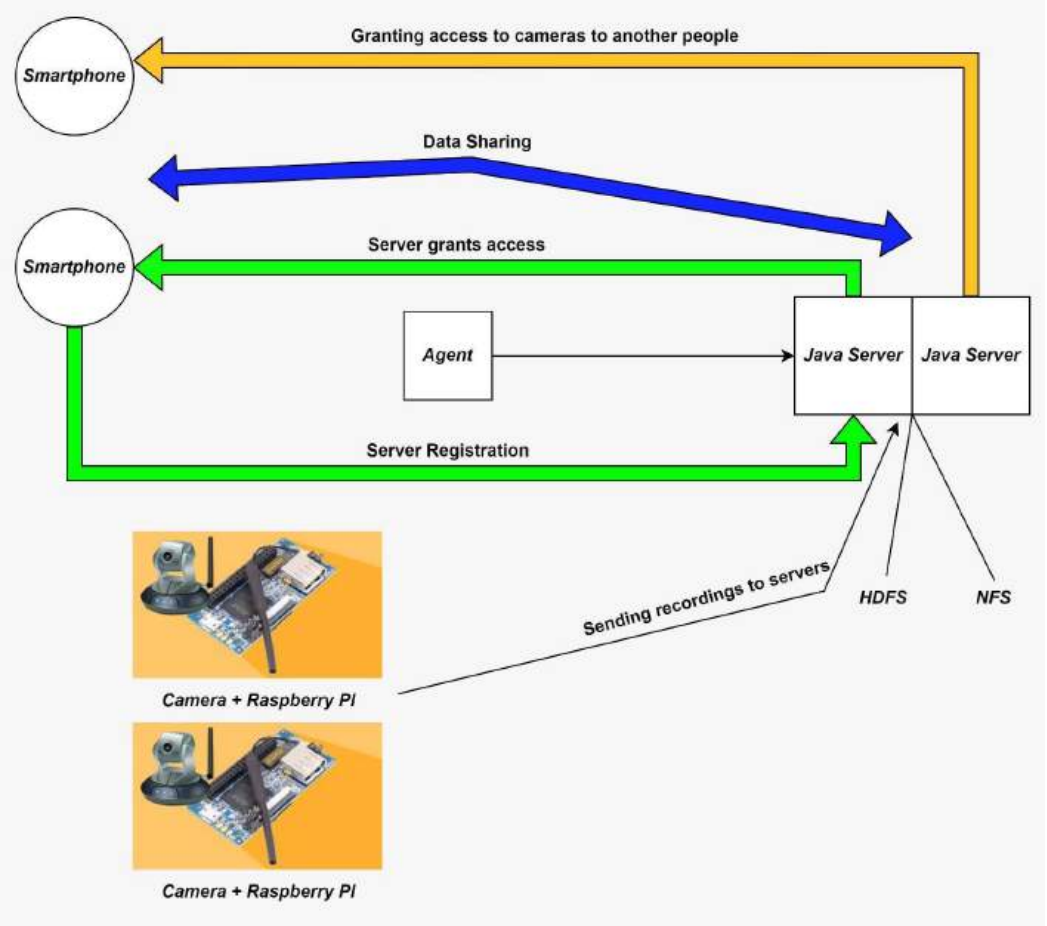

Figure 15. Dataflow architecture of the proposed mesh community surveillance network system.

3.5. Fleet management systems

This application aims to implement real-time Data Acquisition and analysis in Fleet Management Systems, focused on distribution centers where forklifts are monitored to analyze their behavior, allowing for the prediction of maintenance needs.

This application is listed in this paper because we notice the potential future application for monitoring vehicles for mailing delivery in small cities or communities. For such applications, some improvements are needed to ensure real-time data availability for users.

However, at this initial stage, we are focused on describing how telemetry works, specifically in the acquisition and analysis of operational machine data in real-time.

The system uses low-cost technologies, such as ESP32 microcontrollers and Raspberry Pi servers, to collect current and voltage data, which are then transmitted via the MQTT protocol to a database and visualized on dashboards, allowing for more efficient and informed management.

3.5.1. Technologies applied on the implementation

This section presents some details of the technologies applied for implementing the telemetry system.

- **ESP32 Microcontroller:** Open-hardware development board based on a 32-bit dual-core processor, 520 KB of flash memory, integrated Wi-Fi, and Bluetooth, facilitating integration with other modules. It is used for monitoring operational machine data, such as current and voltage, and allows communication with local servers.
- **Raspberry Pi 3 B+ Server:** It works as a local server that centralizes the data sent by the ESP32 microcontrollers. It can perform basic pre-processing or filtering of the data before sending them to the main server. It is a low-cost, compact solution ideal for telemetry applications.
- **MQTT Protocol:** A lightweight and efficient communication protocol designed for real-time communication. It is especially suitable for networks with limited or unstable bandwidth, ensuring reliable data transmission between devices.
- **Prometheus Database:** A monitoring and alerting system that stores data in a time-series format. It is ideal for continuous operational data, allowing real-time collection and querying of metrics.
- **Grafana:** An open-source platform used for data visualization and analysis. It allows the creation of interactive and customizable dashboards, making it easy to visualize the collected data in real-time via a web browser.
- **Docker Compose:** A tool for defining and running multi-container Docker applications. It simplifies system service configuration and management, allowing all components to run in an integrated manner.
- **Current/Voltage Sensors:** The sensors are fundamental for acquiring the data that feed the telemetry system. The current/voltage sensor was based on a shunt resistor and an INA226 circuit. Shunt Resistor is used to measure electric current by creating a small voltage drop proportional to the current flowing through it. This approach allows for accurate measurements and is easy to install, contributing to a plug-and-play concept of the system. Current sensor INA226 is a digital current monitor that communicates with the microcontroller via the I2C bus. It is responsible for taking current and voltage readings and transmitting the collected data to the telemetry system. These sensors were chosen for the accuracy they offer in measurements and for their ease of integration into the developed system.

3.5.2. Architecture of the telemetry implemented system

The architecture of the implemented system is synthesized in Figure 16.

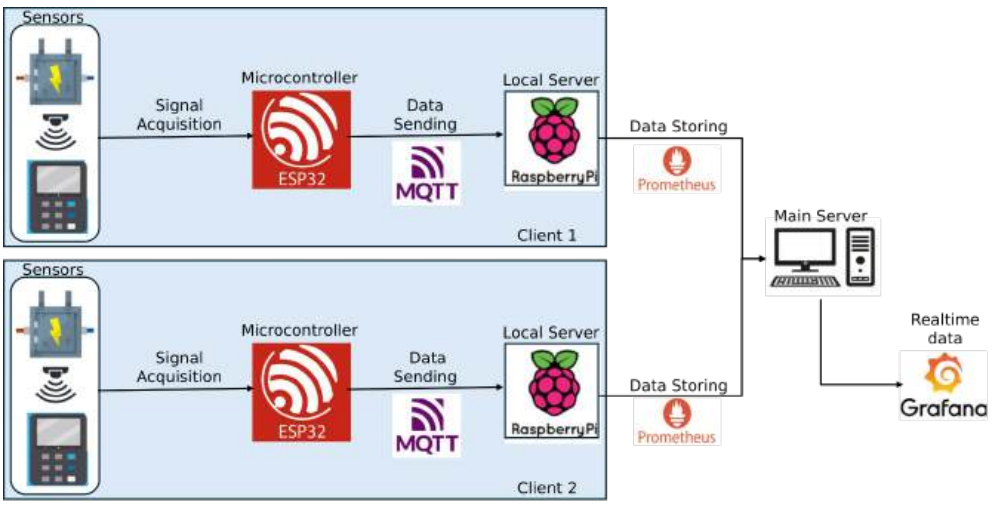

Figure 16. Architecture of the telemetry implemented system.

The Sensors represent the forklifts deployed across the distribution center, equipped with ESP32 microcontrollers. These microcontrollers monitor various operational data from forklifts, such as current, voltage, and other critical parameters. Based on these readings, the system implements control measures using hour meters, which help manage and optimize the equipment maintenance schedule.

The Local Communication involves the data collected by the ESP32 microcontrollers being transmitted to local servers, composed of Raspberry Pi Single Board Computers. Communication between the microcontrollers and the Raspberry Pi servers is handled via the MQTT protocol. MQTT is a highly efficient solution for real-time communication, particularly suited for networks with limited or unstable bandwidth.

The Local Servers are implemented using Raspberry Pi. Each local Raspberry Pi server receives and centralizes data from the ESP32 microcontrollers. Additionally, the Raspberry Pi can perform basic pre-processing or data filtering before forwarding the information to the main server.

The main server receives data from all local servers. This data is stored using Prometheus, a robust monitoring and alerting system designed to handle time-series data, making it ideal for continuously monitored operational metrics.

Visualization and Monitoring works with Prometheus which makes the collected data available for visualization. Grafana, an open-source platform, is used to create interactive, customizable dashboards, allowing real-time data visualization and analysis through a web browser.

3.5.3. Some results

The system blocks that constitute the technological framework designed for the project, from the measurement of signals and acquisition of information in the forklifts to the final user dashboard, are represented below in Figure 17.

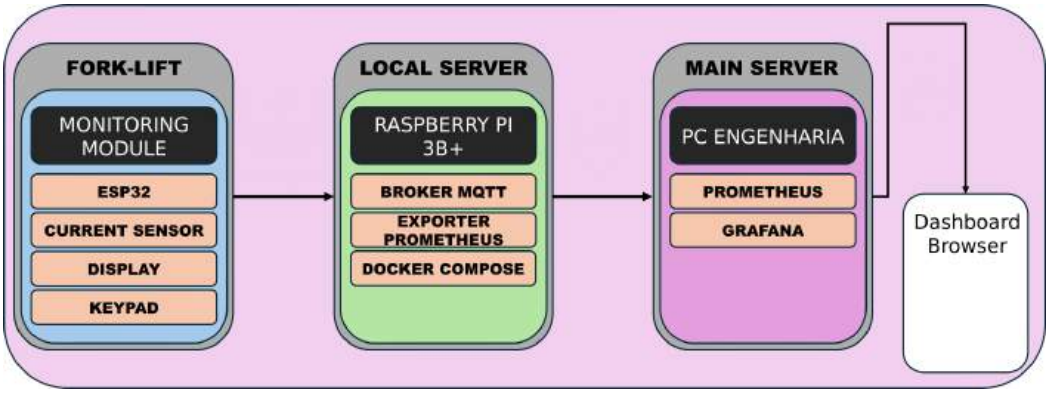

Figure 17. Telemetry system architecture.

The test applied to the system demonstrated the effectiveness of the telemetry system developed for real-time monitoring of operational data from machines. The main highlights include, the Operator Panel (OP) projected as an interactive panel created to display voltage and current readings, allowing operators to view critical information clearly and accessibly The Display and Keyboard Setup were configured to use an LCD and a matrix keypad, facilitating user interaction with the system and data entry.

4. Operating Instructions

Operations Checklists are performed using screens, implemented to assist in managing and monitoring operations, ensuring that procedures are followed correctly. The combination of ESP32 microcontrollers, Raspberry Pi servers, and the Prometheus and Grafana platforms resulted in a low-cost solution with easy integration, meeting the monitoring needs in small environments.

Although the system proved effective, challenges related to Wi-Fi coverage and stability in large warehouses and distribution centers were identified, suggesting the need for improvements for large-scale applications. The results indicate that the system is viable for telemetry applications, with the potential to optimize fleet management and equipment maintenance.

The implementations of the five applications briefly described in this paper show the feasibility of using low-cost hardware and the MQTT protocol in IoT systems, enabling real-time data collection and communication in resource-constrained settings. The authors intend to show that the developed prototypes can provide affordable and effective solutions for resource-limited communities to monitor and respond to flood events, air quality issues, community surveillance, and automation of simple devices at home by monitoring analog and digital sensors as a way to help people with physical necessities and fleet management and equipment maintenance. All those applications have a goal to improve the quality of life in economically disadvantaged cities or communities.

5. Validation

The study demonstrates the feasibility of using Arduino microcontrollers, sensors, and cloud services to develop low-cost and effective monitoring systems for flood events and air quality issues, the successful implementation of low-cost IoT-based solutions for smart monitoring and video surveillance, using devices such as ESP32 microcontrollers, Raspberry Pi servers, and sensors.

6. Conclusions

The limitations for developing technological solutions to improve the quality of life in our communities in the past were primarily cost. Nowadays these costs have been reduced, and technologies such as Arduino, Raspberry Pi, and ESP32 microcontrollers together with open source software allow people to develop their own solutions for local needs. We hope that such projects can inspire other amateur hardware developers around the world.

Funding: This work was partially supported by the CNPq (Conselho Nacional de Desenvolvimento Cientifico e Tecnológico) through the project with the reference 402752/2023-6. Professor João Paulo Carmo was supported by a PQ scholarship with the reference CNPq 305858/2023-8.

Data Availability Statement: All data will be made available upon request to the authors.

Acknowledgments: The authors would like to thank the coordinator of the TSI-UAB-UFABC Guiou Kobayashi and the students Carlos Augusto Duru Pacheco, Fernando Moreno Nhoqui, Jhony Soares Martins da Silva, Marcelo Fernandes de Barros, Breno Matos Carvalho Soares.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. in Wikipedia. Internet of things, 2024.
2. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems 2013, 29, 1645–1660. https://doi.org/10.1016/j.future.2013.01.010.
3. Li, W.; Chai, Y.; Khan, F.; Jan, S.R.U.; Verma, S.; Menon, V.G.; Li, X. A Comprehensive Survey on Machine Learning-Based Big Data Analytics for IoT-Enabled Smart Healthcare System 2021. https://doi.org/10.1007/s11036-020-01700-6/Published.
4. Selvaraj, S.; Sundaravaradhan, S. Challenges and opportunities in IoT healthcare systems: a systematic review, 2020. https://doi.org/10.1007/s42452-019-1925-y.
5. Syed, A.S.; Sierra-Sosa, D.; Kumar, A.; Elmaghraby, A. Iot in smart cities: A survey of technologies, practices and challenges. Smart Cities 2021, 4, 429–475. https://doi.org/10.3390/smartcities4020024.
6. Apat, H.K.; Nayak, R.; Sahoo, B. A comprehensive review on Internet of Things application placement in Fog computing environment. Internet of Things (Netherlands) 2023, 23. https://doi.org/10.1016/j.iot.2023.100866.

7. Vujovic´, V.; Maksimovic´, M. Raspberry Pi as a Sensor Web node for home automation. Computers and Electrical Engineering 2015, 44, 153–171. https://doi.org/10.1016/j.compeleceng.2015.01.019.
8. Ojo, M.O.; Giordano, S.; Procissi, G.; Seitanidis, I.N. A Review of Low-End, Middle-End, and High-End Iot Devices, 2018. https://doi.org/10.1109/ACCESS.2018.2879615.
9. D'ortona, C.; Tarchi, D.; Raffaelli, C. Open-Source MQTT-Based End-to-End IoT System for Smart City Scenarios. Future Internet 2022, 14. https://doi.org/10.3390/fi14020057.
10. Park, S.; Rosca, E.; Agarwal, N. Driving social impact at the bottom of the Pyramid through the internet-of-things enabled frugal innovations. Technovation 2022, 118. https://doi.org/10.1016/j.technovation.2021.102381.
11. Asvadi, A.; Mitriakov, A.; Lohr, C.; Papadakis, P. Digital Twin Driven Smart Home: A Feasibility Study. In Proceedings of the Participative Urban Health and Healthy Aging in the Age of AI; Aloulou, H.; Abdulrazak, B.; de Marassé-Enouf, A.; Mokhtari, M., Eds., Cham, 2022; pp. 18–29.
12. Ramadan, Z.; F. Farah, M.; El Essrawi, L. From Amazon.com to Amazon.love: How Alexa is redefining companionship and interde- pendence for people with special needs. Psychology & Marketing 2021, 38, 596–609, https://onlinelibrary.wiley.com/doi/pdf/10.1002 https://doi.org/https://doi.org/10.1002/mar.21441.
13. GlobalStats, S. Mobile Operating System Market Share Worldwide. https://gs.statcounter.com/os-market-share/mobile/worldwide, 2024.
14. Wexler, S.; Shaffer, J.; Cotgreave, A. The big book of dashboards: visualizing your data using real-world business scenarios; John Wiley & Sons, 2017.
15. Atzori, L.; Iera, A.; Morabito, G.; Nitti, M. The Social Internet of Things (SIoT) – When social networks meet the Internet of Things: Concept, architecture and network characterization. Computer Networks 2012, 56, 3594–3608. https://doi.org/10.1016/j.comnet.2012.07.010.
16. Ortiz, A.M.; Hussein, D.; Park, S.; Han, S.N.; Crespi, N. The Cluster Between Internet of Things and Social Networks: Review and Research Challenges. IEEE Internet of Things Journal 2014, 1, 206–215. https://doi.org/10.1109/JIOT.2014.2318835.
17. Ding, L.; Shi, P.; Liu, B. The clustering of Internet, Internet of Things and social network. In Proceedings of the 2010 Third International Symposium on Knowledge Acquisition and Modeling, 2010, pp. 417–420. https://doi.org/10.1109/KAM.2010.5646274.