IoT

MDPI

Article

Shower–IoT: An Internet of Things System for Monitoring Electric Showers

Helder Holanda Prezotto [1], Natássya Barlate Floro da Silva [2], Lucas Dias Hiera Sampaio [2] and Rogério Santos Pozza [2,*]

[1] Independent Researcher, Boa Vista, Barueri 06460-000, Brazil; helderprezotto@alunos.utfpr.edu.br
[2] Department of Computer, Federal Technological University of Paraná, Cornélio Procópio 86300-000, Brazil; natassyasilva@utfpr.edu.br (N.B.F.d.S.); ldsampaio@utfpr.edu.br (L.D.H.S.)
* Correspondence: pozza@utfpr.edu.br

Abstract: The electric shower is the main form of heating water for bathing in Brazilian homes and one of the significant appliances related to electricity and water consumption. Internet of Things (IoT) projects make it possible to connect objects to the Internet and collect data from machines remotely. In this work, we developed an Internet of Things system for monitoring an electric shower, called Shower–IoT, whose sensor data are water temperature, electric tension, electric current, and water flow. To implement the software infrastructure, we used the services present in cloud computing, such as a broker, processing, and storage, in which the information about the electric shower was made available through an Android application. The results demonstrate that our system can monitor an electric shower integrated with cloud services, allowing the users to visualize its behavior in real time and detect possible failures by comparing sensor data from previous evaluations.

Keywords: Internet of Things; IoT; ESP32; cloud computing; electric shower

Academic Editor: Tareq Hasan Khan

Received: 27 November 2024
Revised: 16 January 2025
Accepted: 25 January 2025
Published: 8 February 2025

Citation: Prezotto, H.H.; da Silva, N.B.F.; Sampaio, L.D.H.; Pozza, R.S. Shower–IoT: An Internet of Things System for Monitoring Electric Showers. *IoT* **2025**, *6*, 11. https://doi.org/10.3390/iot6010011

1. Introduction

Electric showers are devices developed for heating water, used mainly for bathing, and are relatively low-cost equipment widely used in Brazil. According to Eletrobras [1], in 2019, about 40% of Brazilian households had electric showers. In the South and Southeast Regions, this index reached 95% and 79%, respectively. Among households with a heating source, the electric shower is present in 94% of them, making it the primary source for heating water for bathing in households. For Sangoi [2], residential electric showers consume a large amount of electricity in Brazil, especially at times considered to be of most effective use, such as early morning and between late afternoon and early evening.

With the advances in electronics, many everyday devices now contain electronic components that connect to other objects and the Internet. This connectivity allowed for new functionalities and made it possible to control objects at a distance, with the help of cloud computing, as well as to produce several metrics for users, who can visualize and manipulate information through smartphone applications [3].

Cloud computing is a business model that enables computing resources on-demand, such as processing, data storage, and network services. Availability, scalability, and resiliency services make cloud computing a strategic tool for Internet of Things (IoT) applications, which enable applications to support multiple connected devices, as well as tools for processing the generated data [4].

Internet of Things (IoT) systems for monitoring electric showers and power consumption have gained attention due to their potential to optimize resource usage and enhance

safety. These systems can track water and electricity consumption, control temperature, and even provide historical data management [5,6]. IoT-based smart showers help reduce energy costs and water waste while offering real-time data analysis [5]. Low-cost architectures that incorporate various sensors enable efficient water management and safety monitoring [6]. Additionally, IoT-enabled electric current monitoring devices can analyze users' bathing habits and, when anonymized, allow data sharing with others [7]. Research on the monitoring of electrical appliances highlights how IoT technologies can be leveraged to improve energy efficiency and detect operational failures, thereby contributing to the sustainable management of resources. Moreover, the data generated by these devices can contribute to a robust foundation for the development of innovative applications.

However, research to date [5–10] has restricted shower's measurement to limited parameters, such as water flow, water temperature, or electrical current, without integrating all key variables in a single system. These studies often focus on isolated aspects, such as water consumption monitoring or temperature regulation, and lack a comprehensive approach to simultaneously monitor temperature, voltage, current, and water flow, which are important for a better understanding of an electric shower's performance.

Currently, electric showers in the Brazilian market generally lack interfaces for communication with embedded devices. In this context, this work proposes connecting the circuits of an electric shower to an Arduino, enabling the collection and dissemination of information from this equipment via a cloud-based application. This integration allows for continuous monitoring of the shower's operational data, contributing to more efficient management and offering practical benefits for residential and commercial establishments, like hotels.

The specific objective of this work is to develop a system called Shower–IoT, consisting of an IoT device (electric shower) and a mobile application. The IoT device sends the temperature, voltage, current, and water volume data to a cloud service, responsible for the processing, storage, and visualization of these data. The mobile application registers the IoT device in the cloud and receives the data sent by it. The results show that our system can monitor an electric shower integrated with cloud services, enabling users to visualize its behavior in real-time and detect potential failures by comparing sensor data from prior evaluations.

The remaining part of this paper is organized as follows: Section 2 describes and compares previous studies from the literature related to Shower–IoT. Section 3 details the development of Shower–IoT, the technologies and tools used, the cloud application development process, and the Android application. Section 4 presents the results. Section 5 discusses our conclusions and future work.

2. Related Work

The related works presented in this section show the implementation of systems for monitoring and controlling the parameters of an electric shower and ways to store and consult this information.

Coelho et al. [8] proposes a system for monitoring water consumption with a focus on reducing water waste in homes. The authors used the YF-S201 sensor, an Arduino UNO, and an ESP8266 to measure the water flow and transmit data to the server. An Ubuntu server saved the sensor data, and an Android application allowed the user to query the accumulated consumption in real time. Although the system demonstrated its efficiency and provided an intuitive visualization tool, it did not include data regarding the shower power.

Vanelli et al. [6] presents a Cloud infrastructure for storing information generated by IoT devices. This infrastructure has three layers: the sensor layer, responsible for carrying

out the communication and reading data produced by sensors in the environment; the network layer, responsible for grouping the sensor information and sending it to the Cloud server; and the cloud application layer, which is responsible for receiving requests from the network layer and storing the data in the database to guarantee persistence. The authors used a non-relational database as a service (Microsoft Azure Tables), and, as sensors, used DHT11 (temperature and humidity), LDR (brightness), and PIR (infrared movement). As a limitation, this study does not specifically address showers, nor does it provide tools for visualizing sensor data.

Rodrigues et al. [5] describes the development of an IoT smart shower intending to monitor the user's shower and obtain shower parameters, such as water temperature, ambient temperature, and water flow. To regulate the temperature, the project used a dimmer circuit to adjust the power of the shower. Also, the ZigBee protocol was chosen for communication, with the shower information being displayed on a computer terminal. Despite being one of the most comprehensive works in terms of data collection, the system is unable to measure the shower's power consumption and lacks a user-friendly data visualization tool, with data only accessible in a raw format on a computer terminal.

Fabricio et al. [9] illustrates the use of an IoT system to monitor the electrical current of manufacturing equipment to detect possible failures and carry out preventive maintenance. The system consists of three electrical current sensors, an Arduino Uno, and a supervisory system. The information generated from the sensors is sent to a computer via a USB port and stored in a MySQL database, which the supervisory system can access. It contributes to the advancement of monitoring systems and highlights the importance of storing historical data to detect failures based on the electrical consumption of equipment. However, it is not specifically designed for showers and lacks tools for visualizing the collected data.

Kwon et al. [7] developed an IoT system to monitor users' bathing habits considering water temperature, shower water flow, shower movement, and the weight of products used in the bath. The system consists of sensors and a Raspberry Pi that stored the data generated through the sensors. The work aims to identify how users deal with bath data and the risks and benefits of using these data to promote products and services. It demonstrated that providing users with data about their bathing routines can encourage reflection on water consumption, including through interactions with partners. Additionally, such data could potentially be shared with service providers, provided it is sufficiently abstract and anonymized. However, the prototype developed for evaluation focused on measuring water flow rather than power consumption.

Rocha [10] developed a system to monitor the temperature, voltage and current of an electric shower and control the temperature through phase power control. The ESP32 microcontroller was used in conjunction with the Mosquitto broker to process the shower parameters and control the temperature. The Home Assistant software was used to view the data. Although the system focused on power consumption for showers, it lacked sensors to measure water flow, which is important for evaluating water usage.

Table 1 compares the Shower–IoT with other related works and shows the technologies used for reading and controlling the parameters, communicating, and visualizing data. As noted, the related works have sensors in common, which indicates a standard for monitoring the parameters of electric showers. Between them, it is possible to highlight the LM35, DS18B20, and MAX6675 sensors for temperature measurement, the SCT-013 sensor for the measurement of electrical currents, the ZMPT101B sensor for measuring electrical voltage, and the YF-S201 sensor for measuring water flow.

Table 1. Comparison between related works and Shower–IoT.

Characteristics	Electrical Shower Systems						
	[8]	[6]	[5]	[9]	[7]	[10]	Shower–IoT
Water temperature sensor	✗	✗	✓	✗	✓	✓	✓
Ambient temperature sensor	✗	✓	✓	✗	✗	✗	✗
Voltage sensor	✗	✗	✗	✗	✗	✓	✓
Power sensor	✗	✗	✗	✓	✗	✓	✓
Water flow sensor	✓	✗	✓	✗	✓	✗	✓
Wireless Communication	✓	✓	✓	✗	✓	✓	✓
Broker	✗	✗	✓	✗	✗		✓✓
Data Visualization	✓	✓	✓	✓	✓	✓	✓
Temperature control	✗	✗	✓	✗	✗	✓	✗

From the perspective of sensor data, Shower–IoT is more comprehensive than related works, as it integrates water temperature, voltage, electric current, and water flow measurements into a single system. This holistic approach enables a detailed analysis of electric shower performance and resource consumption. However, unlike Rodrigues et al. [5] and Rocha [10], Shower–IoT does not include a temperature control mechanism, representing an opportunity for future development.

In terms of data visualization, Rodrigues et al. [5] displays shower parameters on a computer terminal, while [10] employs the *Home Assistant*, a residential automation software, and Kwon et al. [7] and Coelho et al. [8] use a mobile application. Shower–IoT, on the other hand, leverages an Android application, which enables users to configure hardware via Bluetooth and access real-time and historical data more conveniently.

Regarding communication, Shower–IoT distinguishes itself by leveraging AWS cloud infrastructure, which offers scalability, reliability, and seamless integration with other IoT services. In contrast, Rocha [10] employs a local implementation with a Mosquitto broker, database, and the Home Assistant hosted on local servers, which may limit scalability and remote accessibility.

3. Materials and Methods

Figure 1 shows the general architecture of the Shower–IoT complete system with the MQTT broker, API, and database implementation using IoT Core, EC2, and RDS services, all belonging to AWS. The Shower-IoT device sends the sensor's data inside messages and receives commands from the MQTT broker (AWS IoT Core) via a Wi-Fi connection to the Internet. The broker then distributes the data to clients connected through the Web Socket to the Ruby-On-Rails API and the Lambda function responsible for storing the message information in the database (RDS).

To register a new device, shown in Figure 1, the user must first pair the shower's Bluetooth in the Android application. The token is sent by the device through Bluetooth to the Android application, which performs the registration request in the users' accounts. Also, the device is able to connect to the Wi-Fi via the network name and password information that is sent via Bluetooth during registration by the Android application.

All the stages followed for the development of Shower–IoT system, with its main device, its communication with the infrastructure services and the Android application being presented in the following subsections.

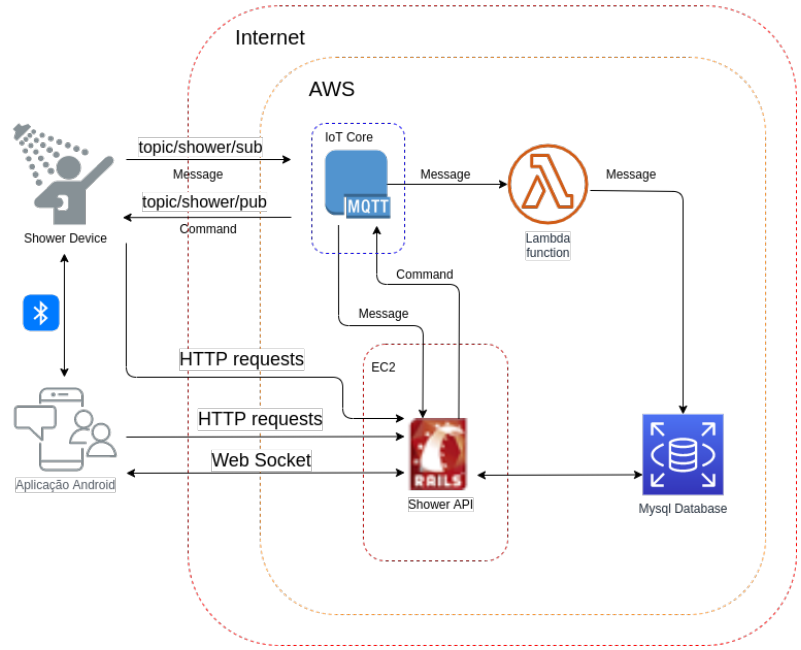


Figure 1. Architecture of Shower–IoT system.

3.1. Shower–IoT Device

The Shower–IoT device contains an ESP32 Dev V1 board and an Arduino UNO. The ESP32 was chosen because it features several useful interfaces for an IoT device, such as Wi-Fi, Bluetooth, Analog-to-Digital Converter (ADC), Digital-to-Analog Converter (DAC), Serial Peripheral Interface (SPI) and digital pins. This work used Wi-Fi, Bluetooth, and SPI peripherals, but it was not possible to use the ESP's ADC due to limitations on the reference voltage measurements. Thus, to avoid the need for calibration of the module to adjust the input signal for every device, an Arduino Uno was used for the analog measurements of voltage and current of the Shower-IoT device.

The ESP32's ADC contains an intrinsic characteristic to the manufacture of an integrated circuit, due to the ADC reference voltage having values between 1000 mV to 1200 mV, and this project has a value of 1100 mV. Thus, to use the ADC present in ESP32, it was necessary to calibrate the module and adapt the input signal to the operating range. However, even after performing the calibration, the results obtained with the ESP32 were not satisfactory. Thus, to avoid the limitations of the ADC present in ESP32, we used an Arduino Uno for the analog measurements of voltage and current sensors.

ESP32 is the component responsible for grouping all monitored variables and sending them to the AWS MQTT broker, which stores these values and makes them available to the Android application. This work used a real-time operating system, FreeRTOS [11], to manage all activities executed in ESP32, and it developed four tasks: MAX6675 reading, YF-s201 sensor reading, Universal Receiver/Transmitter (UART) reading, and sending the MQTT message.

Figure 2 shows that the ESP32 is the central component of the device and is responsible for acquiring the temperature data from the MAX6675 sensor through the SPI protocol, counting the pulses generated in the water flow sensor (YF-s201), and receiving the voltage and current values through UART from the Arduino UNO. Finally, after acquiring all monitored variables, ESP32 sends a message to the MQTT broker on AWS.

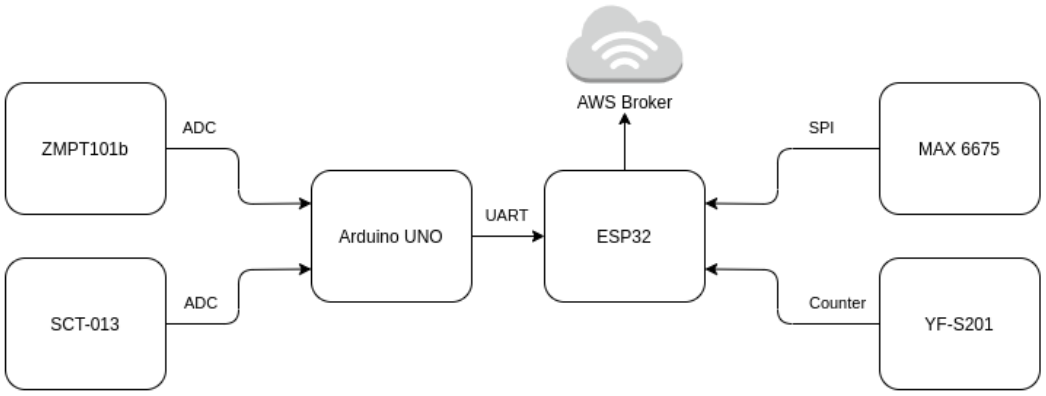


Figure 2. Hardware components' diagram.

To obtain the values of the mains voltage and the electric current, we implemented an Arduino UNO program to store the values read from channels ADC0 and ADC1 (Figure 2) into buffers. When the number of samples reaches the buffer size, the program calculates the root mean square (RMS) values of the samples and performs the conversion to the respective units (volts or amperes). After calculating the RMS values, the program calculates an average to generate a message every second.

To sample the signals in Arduino, firstly, it is necessary to define the respective sampling frequency, obtained by multiplying the number of samples per sinusoidal cycle by the frequency of the electrical network. Our work defined the number of 20 samples per cycle of the electrical network. As the frequency of the electrical network in Brazil is equal to 60 Hz, the sampling frequency must be equal to 1200 Hz. In this way, it was possible to better visualize the waveform generated in the voltage and current sensors.

After defining the value of the sampling frequency, we used the Arduino UNO Timer1 to generate the interruptions responsible for starting the ADC conversion process and for controlling the definition of the read and write buffers. Figure 3 shows the flowchart of Arduino's Timer1 interrupt.

To read the value of the ADC conversion, we used the completion interruption present in this converter. This function executes whenever a conversion ends, and it is responsible for storing the conversion value in the writing buffer and performing the change from the ADC0 channel to the ADC1. Figure 4 shows the flowchart of the converter interruption process.

Therefore, the process of acquiring the values present in the voltage and current sensors follows these steps:

- **Timer1 Interruption:** Select the ADC0 channel and starts the ADC conversion process.
- **First, ADC interruption:** Stores the conversion value in the write buffer of ADC0, selects ADC1, and starts the ADC conversion process.
- **Second, ADC Interruption:** Stores the conversion value in the ADC1 write buffer, selects ADC0, and increments the sample counter.

It takes 1200 samples per channel to store one second of data at a sampling frequency of 1200 Hz, which results in 2400 memory positions. However, the ATmega328P, Arduino Uno's microcontroller, has 2 Kbytes of SRAM memory, making it impossible to program a single buffer for each channel. To overcome this limitation, we programmed four buffers with 150 positions each: two were used for channel 0 and two for channel 1. When a buffer is complete with samples, our program calculates the RMS and stores this value. After 8 iterations, the 1200 samples are complete, and it is possible to average the obtained RMS values, where the average is sent to the ESP32 through the serial port. The entire

process of calculating RMS values and averages is performed in the main loop so as not to spoil the acquisition process and the intervals, as they run like interruptions.

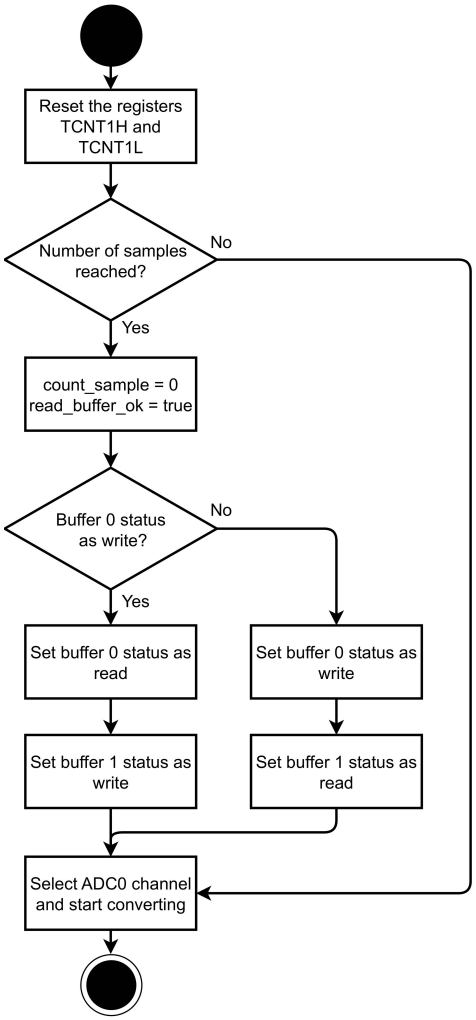


Figure 3. Timer1 flowchart.

The entire process of calculating RMS values and averages is performed in the main loop so as not to spoil the acquisition process and the intervals, as they run like interruptions. Figure 5 shows the main loop flowchart.

The program also uses the technique of multi-buffers or Ping Pong Buffers to ensure that the samples were in the correct ranges and that the calculations did not interfere with this process. In this technique, there are two buffers per information flow: a read and a write buffer. Next, one code (consumer) can read the values from the read buffer, while another code (producer) writes the new values to the write buffer. The read and write operations used pointers; thus, it was possible to change the write-to-read and read-to-write buffers [12].

ESP32 is the component responsible for grouping all monitored variables and sending them to the AWS MQTT broker, which stores these values and makes them available to the Android application. This work used a real-time operating system, FreeRTOS [11], to manage all activities executed in ESP32, and it developed four tasks: MAX6675 reading, YF-s201 sensor reading, UART reading, and sending the MQTT message.

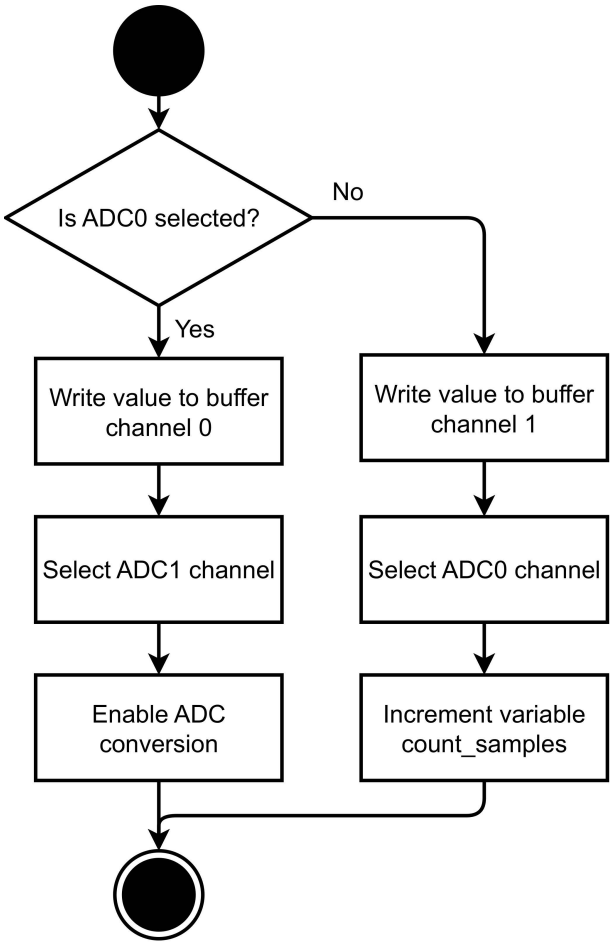


Figure 4. ADC converter flowchart.

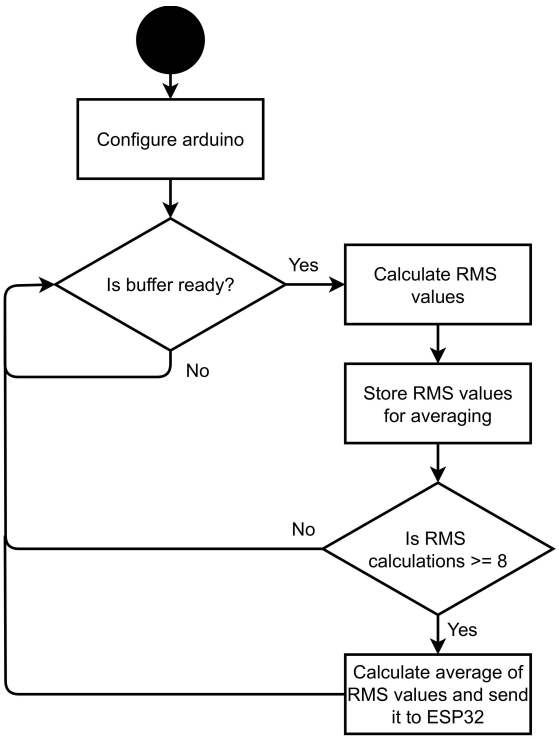


Figure 5. Main loop flowchart.

We developed a communication task with the MAX6675 module to obtain the shower temperature values through the ESP32 HSPI module, which implements the SPI protocol

with pins 12, 13, 14, and 15, respectively, the MISO, MOSI, CLK, and CS signals. Figure 6 shows the connection of the MAX 6675 module with the ESP32.

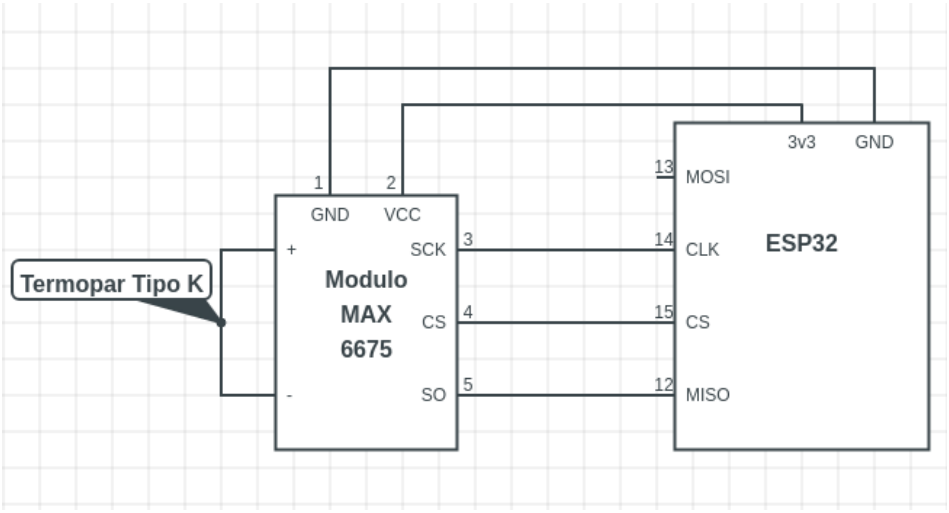


Figure 6. Connection diagram between ESP32 and MAX6675.

Figure 7 shows that the first bits to be transmitted are the most significant bits (MSBs). However, in the ESP32 registers, these bits are stored as being the least significant bits (LSBs). Therefore, it is necessary to exchange the first byte with the second before interpreting the temperature values.

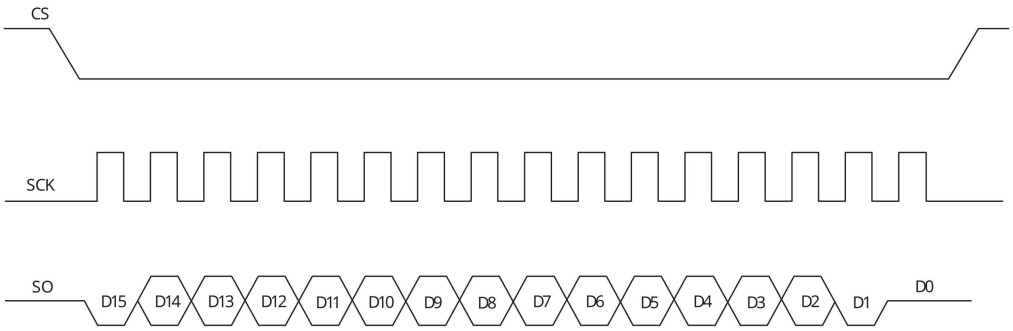


Figure 7. MAX6675 signal sequence.

After correcting the byte positions, it was possible to obtain the sensor information and the read value, as shown in Figure 8. Bit 2 indicates whether the sensor is coupled, where a value of 0 indicates that the sensor is missing and a value of 1 indicates that the sensor is present.

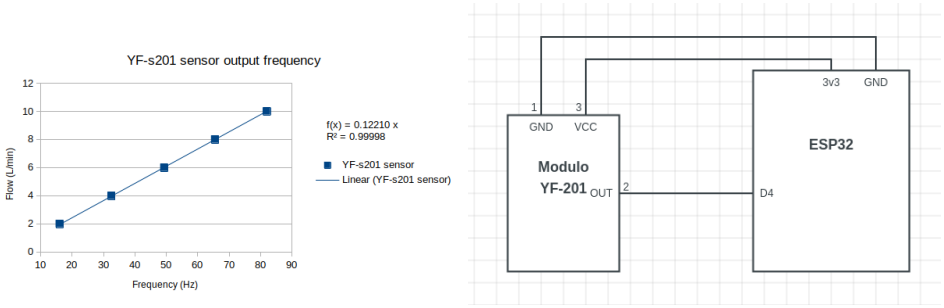
The MAX6675 stores the temperature value in bits 3 to 14, and we calculate the final value by multiplying the read value with the value of the module resolution (0.25 °C).

BIT	DUMMY SIGN BIT	12-BIT TEMPERATURE READING												THERMOCOUPLE INPUT	DEVICE ID	STATE
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	MSB											LSB		0	Three-state

Figure 8. MAX6675 output.

The YF-s201 sensor performs the water flow measurement, which emits pulses as the water passes through it; a counter present in the ESP32 reads these pulses, analyzes them every second, and then starts from zero. Figure 9a shows the graph elaborated through the datasheet data of the YF-s201 component [13]. The points on this graph form a regression liner used to obtain the water flow.

$$flow(L/min) = 0.122098 * frequency\ (Hz) \quad (1)$$



(**a**) (**b**)

Figure 9. YF-s201 sensor. (**a**) YF-s201 sensor output frequency. (**b**) Connection flowchart of ESP32 and YF-s201.

Figure 9b shows the electrical connections necessary for the water flow meter to function and the connection from the signal output pin of the YF-s201 to the pin D4 of the ESP32, which is the pin of counter entry.

The UART protocol is used by the communication between the Arduino UNO and the ESP32, with a transmission rate of 115 Kbps. This communication uses two lines: TX (transmission) and RX (receive). However, due to the difference between the operating voltage level of Arduino (5 V) and ESP32 (3.3 V), we added a voltage divider in the transmission line from Arduino UNO to ESP32, as shown in Figure 10. This voltage divider is straightforward to construct as it uses two identical resistors, each with a value of 1 KΩ. It reduces the 5 V signal to 2.5 V, which falls within the ESP32's operating voltage range of 2.2 to 3.6 V.

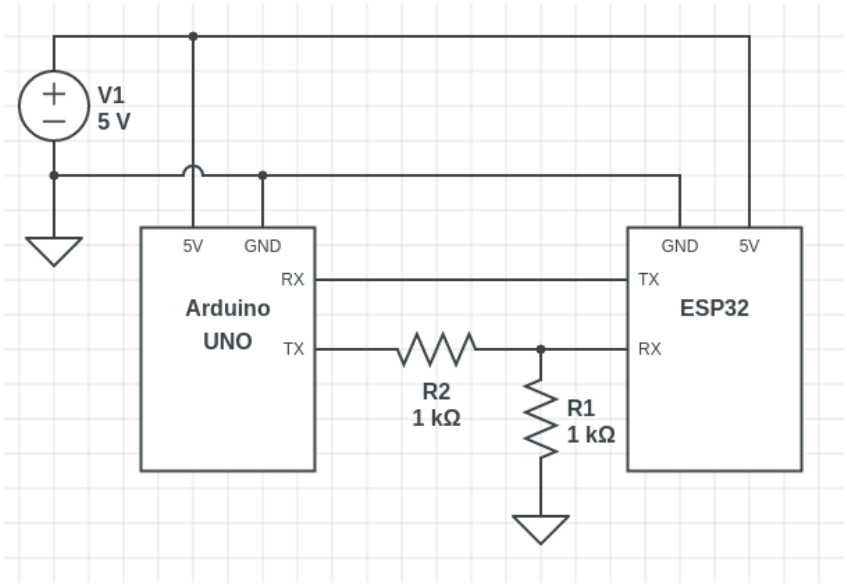


Figure 10. Connection flowchart of ESP32 and Arduino UNO.

The message sending is responsible for establishing it with the monitored attributes and transmitting it to the broker in AWS. For that, we used aws-iot-device-sdk-embedded-C SDK, which has functions for IoT device communication with aws-IoT. To implement authentication on the broker, we used security certificates provided when registering the device with AWS.

After configuring and linking with the user account, the device starts sending messages to the broker, using the topic in the format `shower-iot/:account_id/:device underline id/pub`. Figure 11 shows the example JSON code used for sending messages to MQTT.


```
"device_id": "device_123",
"voltage": 127.5,
"current": 31.2,
"temp": 32.25,
"flow": 2.1
```


Figure 11. Sample message for MQTT.

To develop the prototype (hardware), we used a plastic box with a seal on the lid, which minimizes the contact of water vapor with the electronic components; moreover, to make connections between components, we used a breadboard and wires. Figure 12 shows the components and sensors used in prototype development.

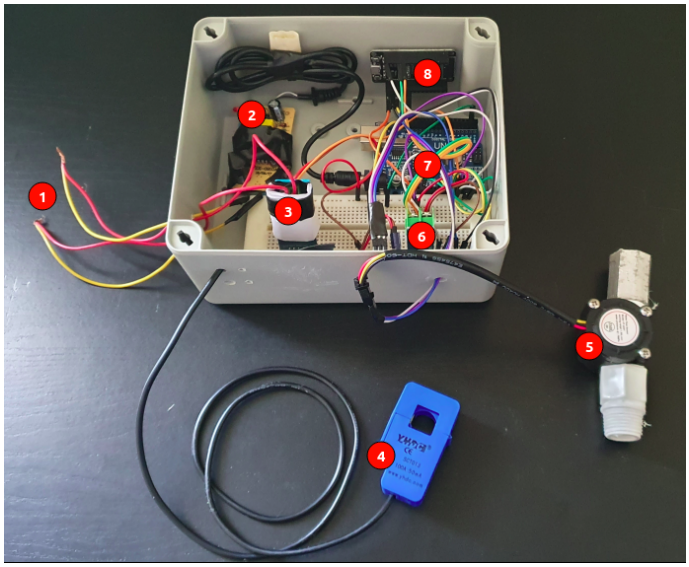


Figure 12. Hardware prototype.

As shown in Figure 12, the items used were, in addition to wires (1) for the connection to the electrical network and the power supply (2), an electrical voltage sensor (ZMPT101b) (3), an electrical current sensor (SCT -013) (4), a water flow sensor (YF-s201) (5), a MAX6675 module (6), an Arduino UNO (7) and an ESP32 (8).

Figure 13 shows the items used: electric shower (1), mains connection wires (2), sensor terminals (3), electrical power adjustment (4), and sensor installation location (5).

To read the parameters, we used an electric shower (Figure 13) with power adjustment, which makes it possible to change the monitored parameters, such as the applied electric current and the water temperature.

Figure 14 shows the installation of the temperature sensor in the shower water outlet, where there is an opening for the installation of the sensor and the use of epoxy resin for sealing.

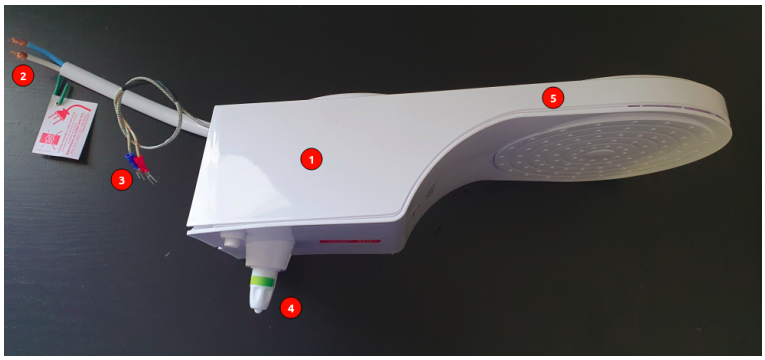


Figure 13. Electric shower.

As shown in Figure 13, the items used were, in addition to the electric shower (1), the wires (2) for the connection to the electrical network, the terminals of the thermocouple sensor (3), the power adjustment mechanism (4), and the installation location of the thermocouple sensor (5).

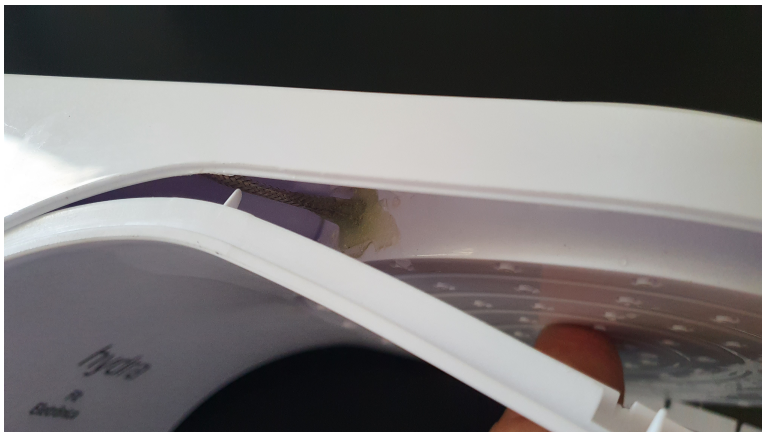
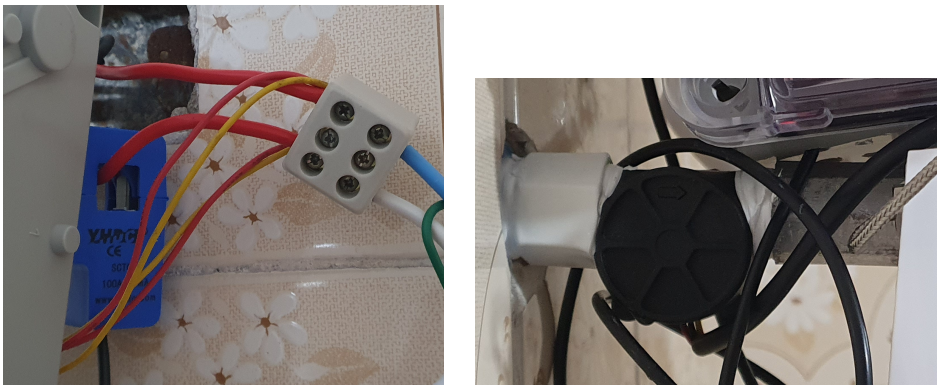


Figure 14. Detail of the installation of the temperature sensor.

Figure 15 shows how we completed the integration of the prototype to the electric shower. Figure 16a,b show the installation of the water flow and electrical current sensor.

Figure 15. Installation of the shower together with the prototype.



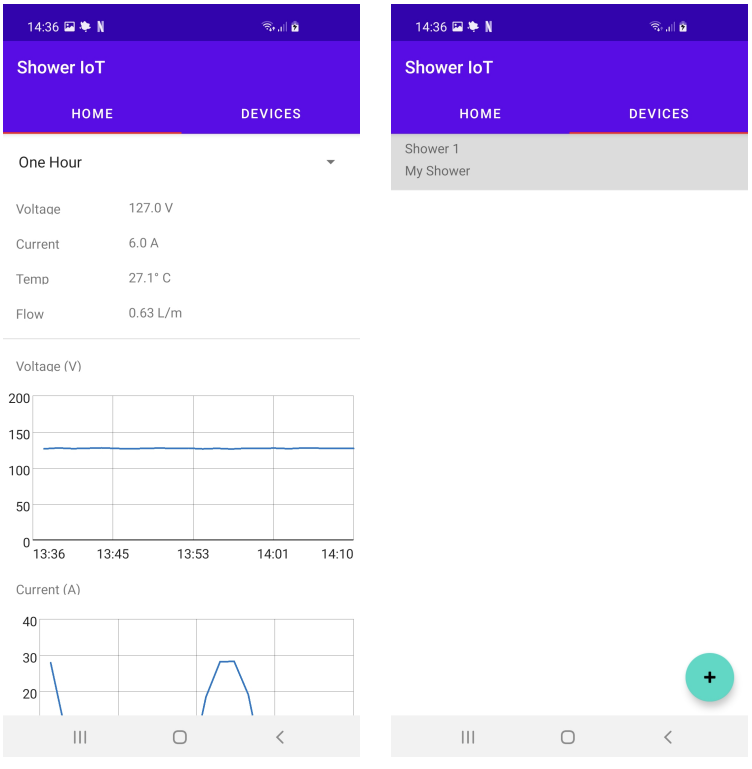
(**a**) (**b**)

Figure 16. Electric current and water flow sensor installation detail. (**a**) Installation of electrical current sensor. (**b**) Installation of water flow sensor.

3.2. Shower–IoT Application

Our application is composed of four main packages: *activity*, which stores the classes used to interact with the main screens; *API_services*, which contain the classes to interact with the API; *data*, which contain the classes to map the entities present in the database; and *fragments*, which are the classes used in the interaction with the main screen tabs. The application version is 1.0.

The main screen is composed of two tabs: the *Home* tab (Figure 17a), to view the summarized parameters of all devices present in the account, and the *Devices* tab (Figure 17b), showing the list with all devices registered to the user account and the icon for registering new devices. The *Home* tab shows the average values at the top of the screen, followed by graphs of each parameter over time.



(**a**) (**b**)

Figure 17. Main screen. (**a**) Summarized data screen. (**b**) Devices' list screen.

By using the list of devices shown in Figure 17b, it is possible to access the details of a device (Figure 18a) and its registration information, such as its real-time parameters. This screen has four graphs related to the data sent by the device.

To register new devices, the user must access the configuration screen (Figure 18b), where he will select the device using the selector at the top of the screen and enter the name and description of the device, as well as the Wi-Fi network name and password.

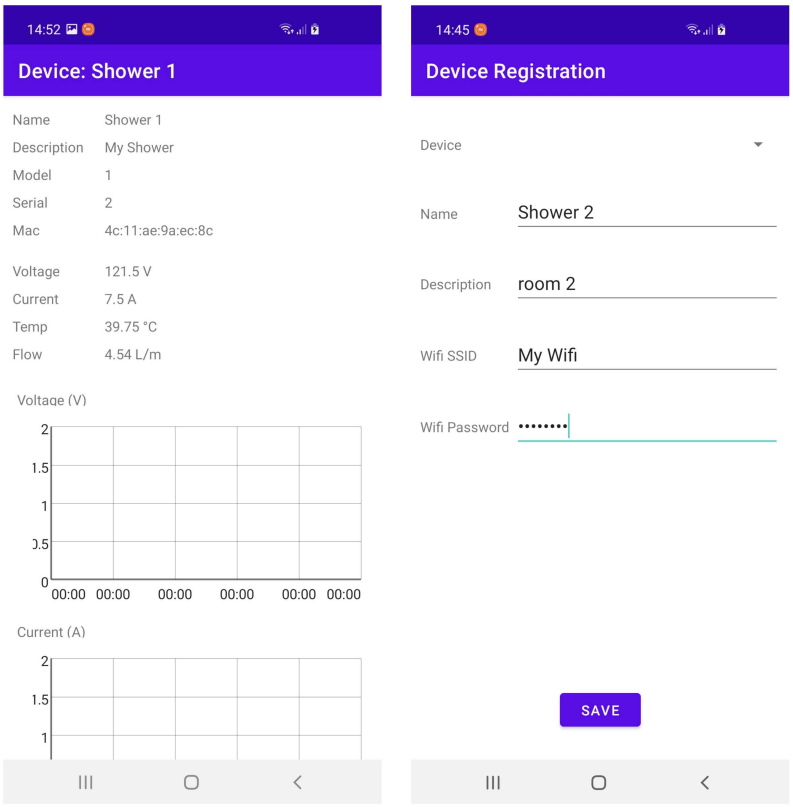


Figure 18. Device screen. (**a**) Device details. (**b**) Device registration.

For our application, to obtain the messages generated by the device, we used a WebSocket connection. We chose this connection because it allows two-way communication between the server and the app, which allows the server to send messages from the device without requiring the app to make a request. Therefore, when the server receives a new message, it immediately sends it to the user. When the WebSocket establishes a connection, the API performs the subscription to the MQTT topic of the device. Thus, when the device publishes a new message, the API server receives a copy of this message, sending it to the application over the WebSocket connection. Figure 19 shows the sequence diagram of this process.

The REST API is the component responsible for handling the user's HTTP requests and registering new devices. We used the Ruby-On-Rails framework for its implementation, which is based on the Ruby language and the MVC (model-view-controller) project pattern and used to develop web applications and APIs.

We used classes called Controllers to process the requests, which are responsible for validating the requests, accessing the Models to interact with the database, and, finally, generating the requests' responses. We developed the following Controllers to meet the needs of this project:

- **Api::V1::DeviceRegisterController**: This class resolves requests referring to the registration of a new device to the user's account.

- **Api::V1::DevicesController**: This Controller is responsible for requests related to obtaining device information, such as the user device list and the details of a specific device.
- **Api::V1::RegistrationsController**: This class is responsible for requests related to the registration of a new user, in which the user must provide an e-mail, password, password confirmation, and first and last name.
- **Api::V1::SessionsController**: This class performs the requests in the user login and logout process in the Android application.
- **Api::V1::SummaryInfosController**: This Controller executes the requests referring to obtaining the data used in the application's graphics.
- **Api::V1::UsersController**: This Controller is responsible for requests for obtaining and changing user information.

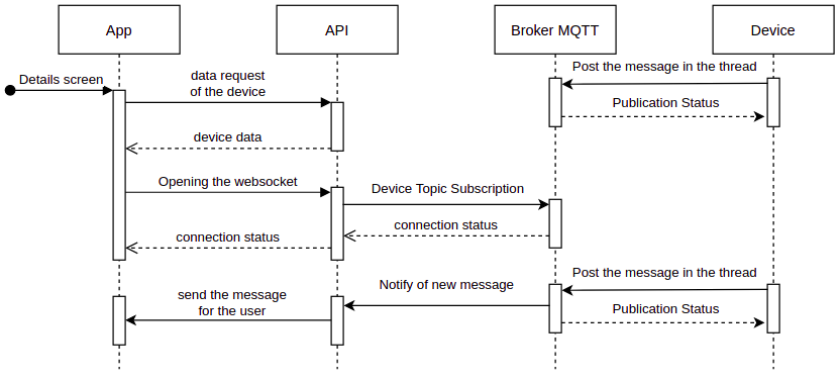


Figure 19. Device message sequence diagram.

To start using a new device, users must register it in their account. To do so, the user selects the device from the list of available Bluetooth devices and provides the Wi-Fi network name, password, device name, and a description. The application then makes an API request to obtain a registration token, which is sent to the device via Bluetooth along with the user-provided information. The device connects to the Wi-Fi network and sends a registration request to the API, including the token and device information for validation. In response, the API returns the MQTT topics to which the device should connect. Figure 20 shows this process.

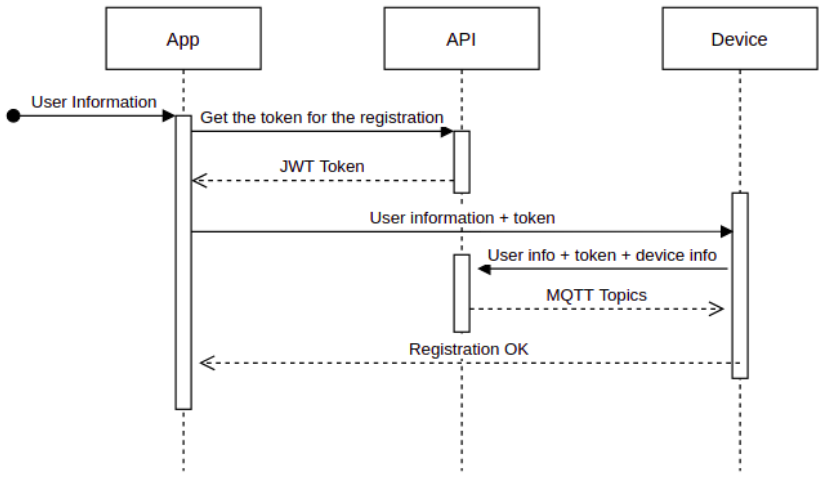


Figure 20. Flow for device registration.

We used the MySQL relational database to store events generated on devices and data related to users and devices. Figure 21 shows the entity relationship diagram developed for this project.

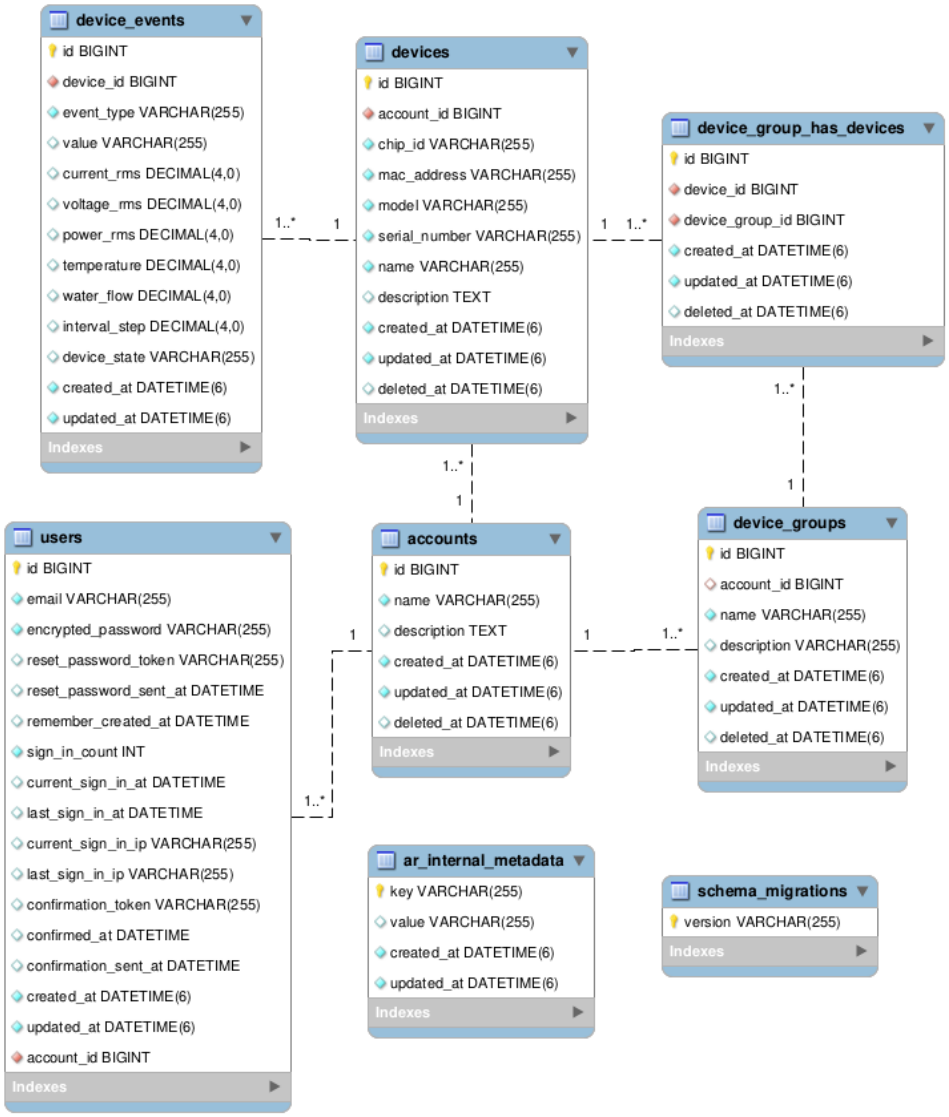


Figure 21. Entity relationship diagram.

In the *accounts'* table, the system's clients are registered. It may have several associated users (*users*), devices (*devices*) and device groups (*device_groups*). Device events are stored in the *device_events* table, and the *schema_migrations* and *ar_internal_metadata* tables are used by the Ruby-On-Rails framework to control scripts database migration process.

4. Results

To analyze the results of the prototype's work, we carried out software and hardware tests. We analyzed the functionality of including the device via Bluetooth, sending the sensor data to the cloud, and storing the data in the software tests.

We performed hardware tests to compare the accuracy between information collected from sensors sent to the cloud by the Shower-IoT and data collected from these sensors on a bench connected to the electrical grid. We describe each test below:

- Zmpt101b (electrical voltage)—to analyze the values obtained on the Zmpt101b sensor, we connected an RMS multimeter in parallel to the sensor input, with the alternating

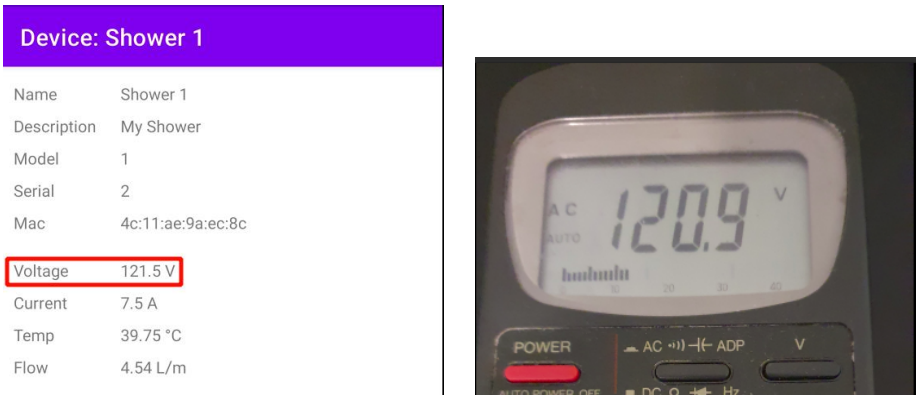
voltage reading configuration, so it was possible to compare the values obtained on the multimeter with the values obtained in the app.

- Sct-013 (electric current)—to analyze the electric current values, we connected the multimeter in series with the test load and the Sct-013 sensor to one of the wires; thus, it was possible to obtain the values of the current flowing through the test circuit.
- Yf-s201 (water flow)—to calculate the water flow, we implemented an experiment that analyzed the variation of the water mass over time; for this, we used a faucet with a constant flow and, after analyzing the data, we obtained the average value of the tap water flow, which we compared with the value presented in the app.
- Max6675 (temperature)—to test the Max6675 sensor, we used an analog thermometer, in which we measured the temperature of the water at the shower outlet and compared it with the values sent to the application.

The Max6675 temperature sensor has a precision of 0.25 degrees Celsius, the Yf-s201 water flow sensor has a precision of 0.007 L/min, the Zmpt101v voltage sensor has a precision of 0.97 volts, and the Sct-013 current sensor has a precision of 0.097 A.

The use of Arduino Uno allows us to read analog signals when the voltage is between 0 and 5 volts, an interval in which the electrical interference generated on the protoboard is minimized, which results in more stable RMS values and the possibility of viewing the waveform of the captured signal.

We used the Zmpt101b sensor for measuring the electrical voltage, which showed a response with low distortion. Figure 22a shows the value obtained in the prototype, and Figure 22b shows the value obtained in a True RMS multimeter.



(**a**) (**b**)

Figure 22. Zmpt101b sensor voltage values. (**a**) Voltage information in the app. (**b**) Voltage value on the multimeter.

Figure 23 shows the graph of the values of the samples obtained from the Zmpt101b sensor. We used 60 samples to generate the graph data, in which it was possible to count three cycles of a sinusoid. The graph in Figure 23 shows that the shape of the sinusoid is well delineated and has low distortion.

To validate the electric current values, we used the same methodology applied to the validation of the electric voltage values. Figure 24a shows the current value obtained in the application, and Figure 24b shows the value obtained in the TrueRMS multimeter.

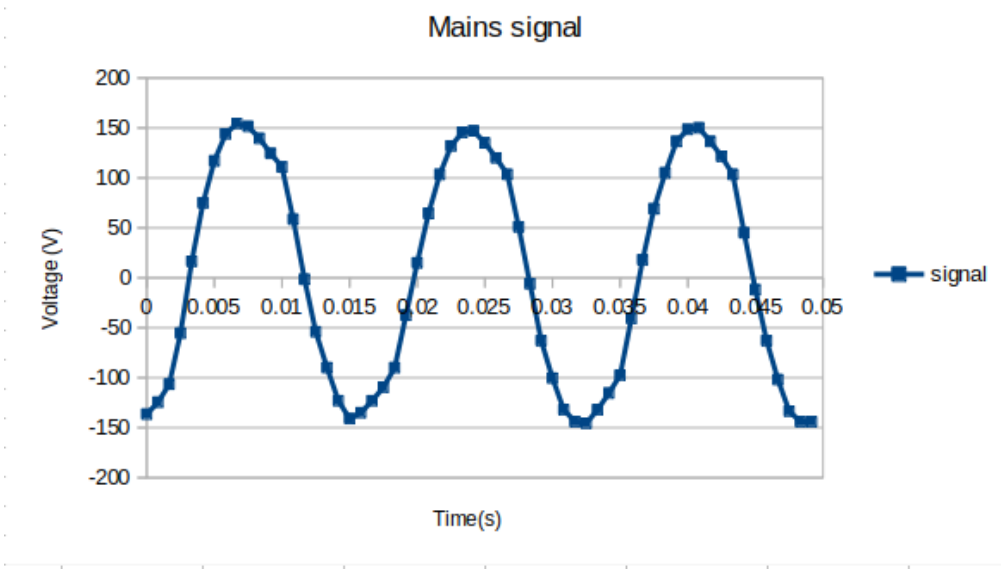
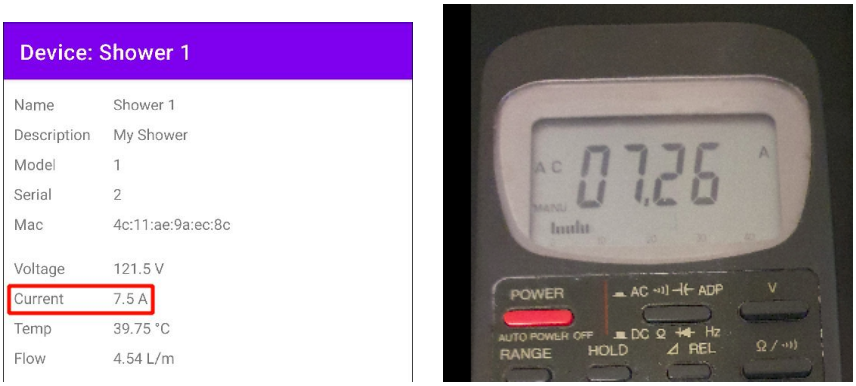


Figure 23. Voltage values obtained from Zmpt101b sensor.



(**a**) (**b**)

Figure 24. Current value on the multimeter. (**a**) Electric current information in the app. (**b**) Electric current information on the multimeter.

Figure 25 shows the current waveform, and there is low distortion in one of the sine values, but this distortion did not affect the RMS value of the electric current.

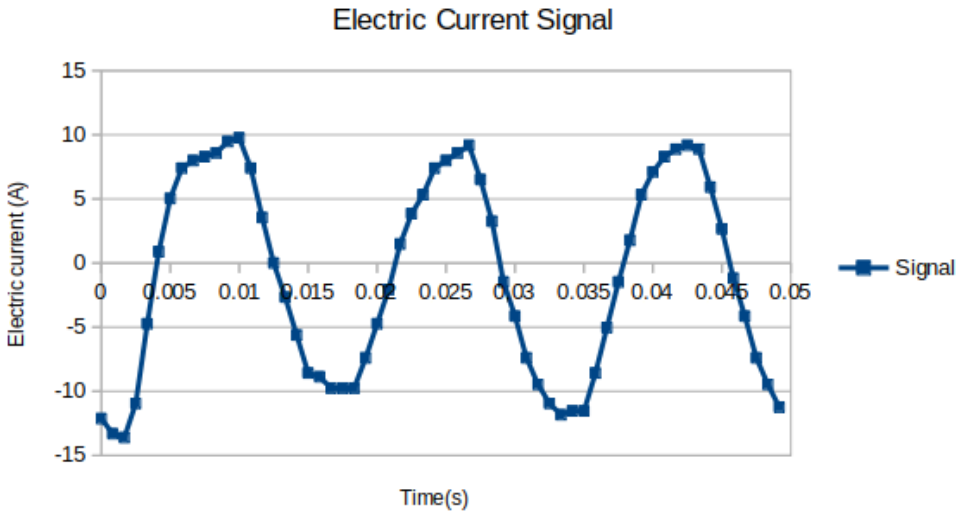


Figure 25. Graph of the values obtained from the Sct-013 sensor.

To analyze the water flow, we used a digital scale to measure the mass of water present in the container and, with the help of a digital camcorder, determine the variation in the mass of water over time. Figure 26 shows the obtained values and the linear regression, and the resulting value of 4.07 L per minute. In the application, Figure 27, the average value was 4.54 L per minute (red frame).

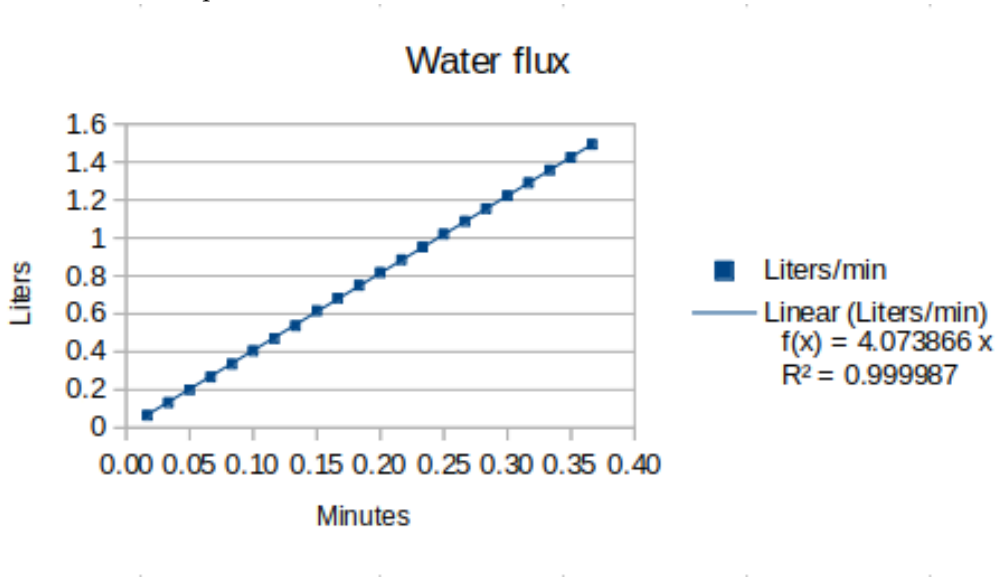
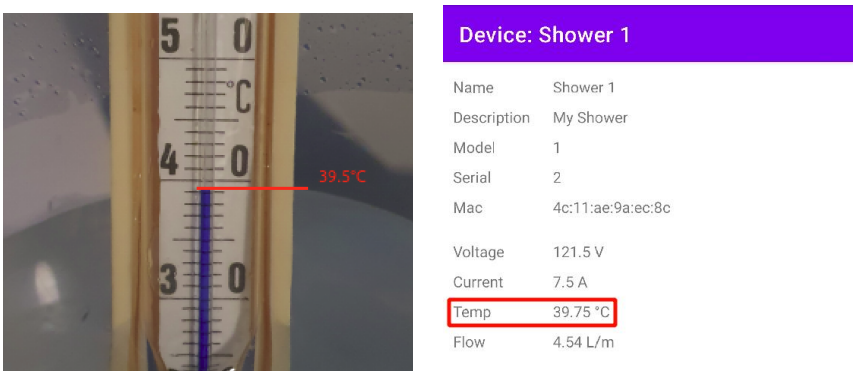


Figure 26. Graph to determine water flow.

Figure 27. Value of water flow in the app.

Figure 28a,b show the temperature value comparisons, in which we used an analog thermometer to measure the temperature of shower water. The value obtained in the application is close to the value of the thermometer.

ESP32 should use the parameters of peripherals and tasks. Moreover, we developed an application with FreeRTOS and the PlatformIO plugin. PlatformIO projects on ESP32 have 1 MB of memory space, but this amount is insufficient for our project requirements. We changed the memory mapping configuration file to work around this limitation and take advantage of all available ESP32 memory.



(**a**) (**b**)

Figure 28. Information about the temperature sensor values. (**a**) Thermometer temperature value. (**b**) Application temperature value.

With a sampling rate of 2400 Hz, the network sinusoid did not present distortions, which indicates that the software embedded in the Arduino UNO obtained a satisfactory response. To obtain this answer, we used ATmega328P interrupt mechanisms, which made it possible to obtain this waveform of analog signals and RMS values. From the RMS values of current and voltage, the apparent power can be calculated by multiplying these values. This calculation allows users to monitor the electrical consumption of their equipment, enabling them to track usage patterns and potentially identify malfunctions or inefficiencies.

The API that we developed with the Ruby-On-Rails framework made it possible to access user information, access data from linked devices, and link new devices in the user's account. We used a WebSocket to obtain messages from devices in real time as an alternative to not authenticating users on the AWS platform. In this process, users had to authenticate to the API the first time. We also achieved a more transparent system architecture.

Through WebSocket, the Android application allows the user to view the history of device parameters stored in the database, real-time information, and use Bluetooth to register new devices. Our application used a relational database (MySQL) which returned historical data within 1 second.

The integration of AWS cloud services allows users to access historical data and analyze consumption trends over time. This feature is particularly valuable for detecting anomalies in usage patterns, predicting maintenance needs, and improving overall energy efficiency.

The implementation of AWS cloud infrastructure enabled the development of an application with security mechanisms, such as authentication certificates and encryption protocols (SSL/TLS). It also allowed integration with other services for IoT devices, such as implementing cloud status, firmware update distribution, and integration with systems for sending notifications to users.

5. Conclusions

The proposed Shower-IoT system successfully integrates multiple parameters' monitoring—water temperature, electrical voltage, current, and flow—into a single, cohesive platform. The system demonstrated accuracy, with voltage and current readings deviating less than 3% from standard measurement tools. Moreover, AWS cloud services provide a scalable, robust, and accessible solution for data storage and analysis, enhancing the system's usability for end users.

Compared to existing works, the Shower-IoT system offers a more comprehensive approach. Unlike [5,10], which focused on temperature control or partial monitoring, this

system captures all key parameters in real time. Additionally, the adoption of AWS cloud infrastructure differentiates this solution by enabling remote access, scalability, and robust data storage, whereas other works rely on locally hosted servers or limited connectivity options.

ESP32 met expectations for task execution and message delivery to the AWS broker. However, its analog–digital converter exhibited limited reliability due to a narrow operating range and susceptibility to noise. To address this, we integrated an Arduino UNO for analog signal readings, ensuring consistent results and reliable data processing.

This combination of hardware adjustments and cloud integration underscores the system's capacity to deliver reliable data while remaining adaptable to diverse environments.

The AWS infrastructure provided functionalities critical to IoT projects, such as security certificates for communication, remote update mechanisms, and device management systems. These tools ensured reliable message delivery, real-time data access, and scalable storage, positioning the system for large-scale IoT deployments.

The Shower-IoT system addresses key gaps in the literature by combining extensive parameter monitoring with cloud-based data management. This integration facilitates a deeper understanding of electric shower performance and empowers users to optimize resource consumption, reducing water and energy waste. The system's modular design and cloud connectivity make it adaptable for broader applications in residential and commercial contexts.

The Shower-IoT system facilitates real-time monitoring and provides users with actionable insights to optimize resource usage and enhance safety. For instance, water flow data help identify leaks, while voltage and current monitoring ensure the safe operation of the electric shower. These capabilities are critical for residential and commercial settings, allowing for proactive maintenance and resource planning.

Despite its strengths, the current system lacks temperature control functionality, which could further enhance user comfort and resource optimization. Our future work will focus on integrating water temperature control using ESP32 and a dimmer circuit, enhancing user experience and resource efficiency. Additionally, a web-based system for monitoring electric showers in commercial establishments, such as gyms and hotels, will be developed. This extension will include hardware capable of controlling water flow and tracking individual consumption, thereby expanding the system's applicability and sustainability. Furthermore, tests will be conducted over extended periods to assess the system's robustness.

Author Contributions: Conceptualization, H.H.P. and R.S.P.; methodology, H.H.P.; software, H.H.P.; validation, H.H.P., L.D.H.S., N.B.F.d.S. and R.S.P.; formal analysis, H.H.P., L.D.H.S., N.B.F.d.S. and R.S.P.; investigation, H.H.P. and R.S.P.; resources, H.H.P. and R.S.P.; data curation, L.D.H.S.; writing—original draft preparation, H.H.P. and R.S.P.; writing—review and editing, L.D.H.S., N.B.F.d.S. and R.S.P.; visualization, H.H.P., L.D.H.S. and N.B.F.d.S.; supervision, R.S.P.; project administration, R.S.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

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

References

1. Eletrobras. Pesquisa de Posse e Hábitos de Uso de Equipamentos Elétricos na Classe Residencial. 2019. Available online: https://eletrobras.com/pt/AreasdeAtuacao/BRASIL.pdf (accessed on 15 January 2025).
2. Sangoi, J.M. Análise Comparativa Do Desempenho De Sistemas De Aquecimento De água Em Edificações Residenciais. 2015. Available online: https://repositorio.ufsc.br/bitstream/handle/123456789/169435/338189.pdf (accessed on 15 January 2025).

3. Rayes, A.; Salam, S. *Internet of Things From Hype to Reality: The Road to Digitization*, 2nd ed.; Springer Publishing Company, Incorporated: Berlin/Heidelberg, Germany, 2018.
4. Samuel, A.; Sipes, C. Making Internet of Things Real. *IEEE Internet Things Mag.* **2019**, *2*, 10–12. [CrossRef]
5. Rodrigues, R.R.; Rodrigues, J.J.; da Cruz, M.A.; Khanna, A.; Gupta, D. An IoT-based automated shower system for smart homes. In Proceedings of the 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Bangalore, India, 19–22 September 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 254–258.
6. Vanelli, B.; da Silva, M.P.; Manerichi, G.; Pinto, A.S.R.; Dantas, M.A.R.; Ferrandin, M.; Boava, A. Internet of things data storage infrastructure in the cloud using NoSQL databases. *IEEE Lat. Am. Trans.* **2017**, *15*, 737–743. [CrossRef]
7. Kwon, H.; Fischer, J.E.; Flintham, M.; Colley, J. The connected shower: Studying intimate data in everyday life. *Proc. ACM Interactive, Mobile, Wearable Ubiquitous Technol.* **2018**, *2*, 1–22. [CrossRef]
8. Coelho, E.M.; e Souza, D.E.S.; dos Santos Sá, J.; de Souza Farias, F. Water Manager: A System Based on Hardware and Software for User Consumption Monitoring. *IEEE Lat. Am. Trans.* **2019**, *17*, 1879–1886. [CrossRef]
9. Fabricio, M.A.; Behrens, F.H.; Bianchini, D. Monitoring of industrial electrical equipment using IoT. *IEEE Lat. Am. Trans.* **2020**, *18*, 1425–1432. [CrossRef]
10. Rocha, J.C. Smart Shower: Connected to an IoT Network. Undergraduate Thesis, Federal University of Santa Catarina, Araranguá, Brazil, 2021. Available online: https://repositorio.ufsc.br/handle/123456789/223733 (accessed on 15 January 2025).
11. FreeRTOS. Mastering the FreeRTOS—Real Time Kernel. 2016. Available online: https://www.freertos.org/media/2018/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf (accessed on 15 January 2025).
12. Neves, F. Ping-Pong Buffer Para Sistemas Embarcados. 2016. Available online: https://www.embarcados.com.br/ping-pong-buffer/ (accessed on 15 January 2025).
13. YIFA The Plastics Ltd. Sensor de Fluxo de Água—YF-S201. 2021. Available online: https://cdn-shop.adafruit.com/product-files/828/C898+datasheet.pdf (accessed on 15 January 2025).