International Journal of Simulation, Optimization & Modelling
ISSN: 3083-967X

Journal homepage: https://e-journal.scholar-publishing.org/index.php/ijsom

Integration of a Microcontroller-Based Temperature Sensor with SSH Scripting for Automated Server Power-Off Under Critical Conditions

Siska Amanda[1], Erdiwansyah[2], Teuku Andiansyah[1]

[1]Department of Computer Engineering, Universitas Serambi Mekkah, Banda Aceh 23245, Indonesia
[2]Department of Natural Resources and Environmental Management, Universitas Serambi Mekkah, Banda Aceh, 23245, Indonesia

Corresponding Author: siskaamandaiphon@gmail.com

Abstract

The advancement of Internet of Things (IoT) technology has enabled the development of monitoring systems that enhance the security and reliability of server infrastructure through real-time environmental data collection. This study aims to implement a server room temperature-monitoring system based on the ESP32 microcontroller integrated with a DHT22 temperature sensor, the Message Queuing Telemetry Transport (MQTT) protocol, the ThingsBoard platform, a Telegram Bot, and an automated server power-off mechanism via Secure Shell (SSH). The proposed system periodically measures temperature, transmits sensor data to a cloud platform via MQTT, displays real-time information on the ThingsBoard dashboard, and sends notifications to the administrator through Telegram when the temperature reaches a critical threshold. Under critical conditions, the system automatically executes a server power-off command through a Flask API and OpenSSH as a protective mechanism against hardware damage caused by excessive heat. Functional testing was conducted on all system components, including the DHT22 sensor, MQTT communication, the ThingsBoard dashboard, a 16×2 I2C LCD, LED indicator, buzzer, Telegram Bot, and the integration between Flask API and OpenSSH. The experimental results demonstrate that all components operated successfully and were fully integrated. Temperature data were transmitted and displayed on the dashboard in real time; Telegram notifications were delivered based on predefined conditions; and the automated server power-off simulation was successfully executed when the temperature exceeded the critical threshold. These findings indicate that integrating IoT technology with SSH automation enhances monitoring effectiveness and provides an automated protection mechanism for server rooms.

Article Info
Received: 10 June 2026
Revised: 11 July 2026
Accepted: 15 July 2026
Available online: 16 July 2026

Keywords
Internet of Things
ESP32
DHT22
MQTT
ThingsBoard

1. Introduction

Server rooms are essential components of contemporary organisational infrastructure because they accommodate computing equipment that supports data storage, network communication, digital services, and information-system operations. The reliability of these services is closely related to the environmental conditions surrounding the installed hardware. Excessive temperature can increase component stress, accelerate hardware degradation, trigger thermal throttling, reduce processing performance, and cause unexpected service interruptions. In severe conditions, prolonged overheating

©2026 The Author(s). Published by Scholar Publishing. This is an open access article under the CC BY license. Available online https://e-journal.scholar-publishing.org/index.php/ijsom

may damage processors, storage devices, power supplies, and other electronic components. International guidelines for data-centre facilities emphasise the need to maintain appropriate environmental conditions and continuously monitor temperature-related parameters as part of infrastructure protection and operational risk management (American Society of Heating, Refrigerating and Air-Conditioning Engineers [ASHRAE], 2021; International Organisation for Standardisation [ISO], 2021). Continuous temperature monitoring is therefore necessary to detect abnormal conditions before they develop into equipment failure or prolonged downtime (Kusuma & Waluyo, 2025).

The development of the Internet of Things (IoT) has expanded the capabilities of environmental monitoring systems by connecting sensors, embedded controllers, communication networks, and cloud-based applications. An IoT monitoring architecture enables continuous measurement of physical conditions and transmission to remote platforms without requiring administrators to remain in the server room (Al-Fuqaha et al., 2015; Gubbi et al., 2013). The ESP32 microcontroller is well-suited for this type of application because it integrates processing capabilities, Wi-Fi connectivity, configurable I/O interfaces, timers, and power management functions on a compact platform (Espressif Systems, 2025). In the proposed system, the ESP32 is connected to a DHT22 sensor to measure the server room temperature periodically. The DHT22 is widely used in embedded monitoring applications because it provides digital temperature and humidity measurements and offers greater measurement resolution than the DHT11 sensor (Saptadi, 2015).

Reliable data communication is also required to connect the sensing layer with the remote monitoring platform. Message Queuing Telemetry Transport (MQTT) employs a lightweight publish–subscribe communication model that is appropriate for resource-constrained devices and networks with limited bandwidth (ISO, 2016; Naik, 2017; OASIS, 2014). Through this protocol, the ESP32 publishes temperature data to an MQTT broker, while the ThingsBoard platform receives, stores, and presents the telemetry through numerical indicators, time-series graphs, and remotely accessible dashboards. This configuration allows administrators to observe current conditions and review historical temperature patterns through an internet-connected device. Previous research has demonstrated the suitability of MQTT and ThingsBoard for remote environmental data acquisition and visualisation (Nurwarsito & Adaby, 2024). The combination of ESP32, MQTT, and ThingsBoard consequently provides a scalable foundation for continuous server-room monitoring and centralised data management (ThingsBoard, Inc., n.d.).

Several studies have implemented IoT-based temperature-monitoring systems using ESP32 microcontrollers, DHT-series sensors, websites, mobile applications, and cloud dashboards. Saptadi (2015), for example, evaluated the measurement characteristics of the DHT11 and DHT22 sensors, while Ilmi et al. (2024) developed an IoT-based system for monitoring server room temperature and humidity. Related work has also integrated ESP32 devices with real-time notification capabilities for environmental monitoring applications (Inzagi et al., 2025). These studies demonstrate that low-cost embedded devices can effectively acquire and distribute environmental information. However, the systems reviewed in this study generally emphasise sensing, visualisation, data recording, and warning delivery. Once a hazardous temperature is detected, the administrator is still expected to interpret the warning and manually perform corrective action. Such dependence on human intervention can create a response delay when administrators are unavailable, when notification messages are overlooked, or when a temperature increase occurs rapidly.

A monitoring system intended to protect server infrastructure should therefore provide not only situational awareness but also an immediate protective response. The proposed system addresses this requirement by combining Telegram-based notification with an automated server power-off mechanism. When the temperature exceeds a predefined threshold, the ESP32 activates local warning devices and sends a notification through a Telegram Bot. It also sends an HTTP request to a Flask application programming interface that serves as a communication bridge between the microcontroller and the target Ubuntu server (Pallets Projects, n.d.; Telegram, n.d.). The Flask service subsequently establishes an OpenSSH connection and executes a controlled shutdown command. SSH provides encrypted remote login and command execution services with mechanisms for authentication, confidentiality, and data integrity protection (Cahyani, 2022; Ylonen & Lonvick, 2006a, 2006b). OpenSSH implements these standardised mechanisms, enabling commands to be executed remotely without relying on an unsecured communication channel (OpenBSD Project, n.d.).

©2026 The Author(s). Published by Scholar Publishing. This is an open access article under the CC BY license. Available online https://e-journal.scholar-publishing.org/index.php/ijsom

The resulting architecture combines local monitoring, cloud-based observation, remote notification, and automated server protection within a single system. Temperature readings are displayed locally through a 16 × 2 I2C liquid-crystal display and transmitted to ThingsBoard for remote visualisation. Normal and abnormal operating states are indicated through an LED and buzzer, while Telegram messages provide immediate information to the administrator. Under a critical temperature condition, the Flask–OpenSSH subsystem performs the server shutdown procedure without waiting for manual confirmation. System performance is evaluated through functional testing of the DHT22 sensor, ESP32 controller, display module, LED, buzzer, MQTT communication, ThingsBoard dashboard, Telegram notification, Flask API, OpenSSH connection, and automated power-off process. This evaluation approach assesses whether the hardware and software components operate as intended and whether the complete monitoring and protection workflow is successfully integrated (Ilmi et al., 2024).

The novelty of this study lies in the development of a closed-loop IoT protection system that transforms temperature measurements into an automated server-level response rather than limiting the system to monitoring and warning functions. Unlike conventional implementations that require administrators to respond manually after receiving an alert, the proposed architecture combines real-time sensing, local alarms, cloud visualisation, mobile notifications, API-based communication, secure remote command execution, and automated server shutdown. Specifically, the study aims to implement continuous temperature acquisition using ESP32 and DHT22; transmit and visualise telemetry through MQTT and ThingsBoard; generate local and Telegram-based warnings when the temperature reaches a predetermined threshold; execute an automated protective shutdown through Flask and OpenSSH; and verify the functional integration of all sensing, communication, notification, and automation components. This integrated approach is expected to reduce response delays, limit the duration of server exposure to excessive heat, and improve the reliability of server-room protection.

2. Methodology

This study employed an experimental research method to design, implement, and evaluate an Internet of Things (IoT)-based server-room environmental monitoring system integrated with an automated server power-off mechanism (Prasiani et al., 2022). The system combines an ESP32 microcontroller, a DHT22 temperature and humidity sensor, Wi-Fi connectivity, the Message Queuing Telemetry Transport (MQTT) protocol, the ThingsBoard platform, a Telegram Bot, a Flask application programming interface (API), and OpenSSH installed on an Ubuntu Server (Muliadi et al., 2020). The ESP32 serves as the main controller responsible for sensor data acquisition, network communication, threshold evaluation, and activation of the protection procedure. The DHT22 periodically measures the temperature and relative humidity in the server room, although the automated protection decision is primarily based on the measured temperature. The main objectives of the system are to provide continuous environmental monitoring, display telemetry data in real time, notify administrators of abnormal conditions, and automatically shut down the server when the temperature exceeds a predefined safety threshold. Through this integrated architecture, the system is designed not only to report server-room conditions but also to initiate an immediate protective response without waiting for manual administrator intervention.

As illustrated in **Fig. 1**, the operational process begins with initialising the ESP32, DHT22 sensor, Wi-Fi connection, MQTT communication, and SSH-related services. During this stage, the ESP32 verifies that the sensor and communication interfaces are available before entering the monitoring cycle. After successful initialisation, the DHT22 reads the current temperature and humidity values at predetermined intervals. The ESP32 then processes the sensor output and sends the acquired data to the ThingsBoard platform via MQTT. ThingsBoard receives published telemetry and presents it on a dashboard, allowing administrators to observe current values and changes in environmental conditions remotely. This data-transmission stage is repeated continuously so that the dashboard reflects the latest sensor readings. In addition to cloud-based visualisation, the continuous monitoring process provides the data required for the subsequent decision-making stage. The system therefore establishes a recurring sequence comprising sensor acquisition, MQTT-based transmission, dashboard visualisation, and temperature-condition evaluation.

©2026 The Author(s). Published by Scholar Publishing. This is an open access article under the CC BY license. Available online https://e-journal.scholar-publishing.org/index.php/ijsom

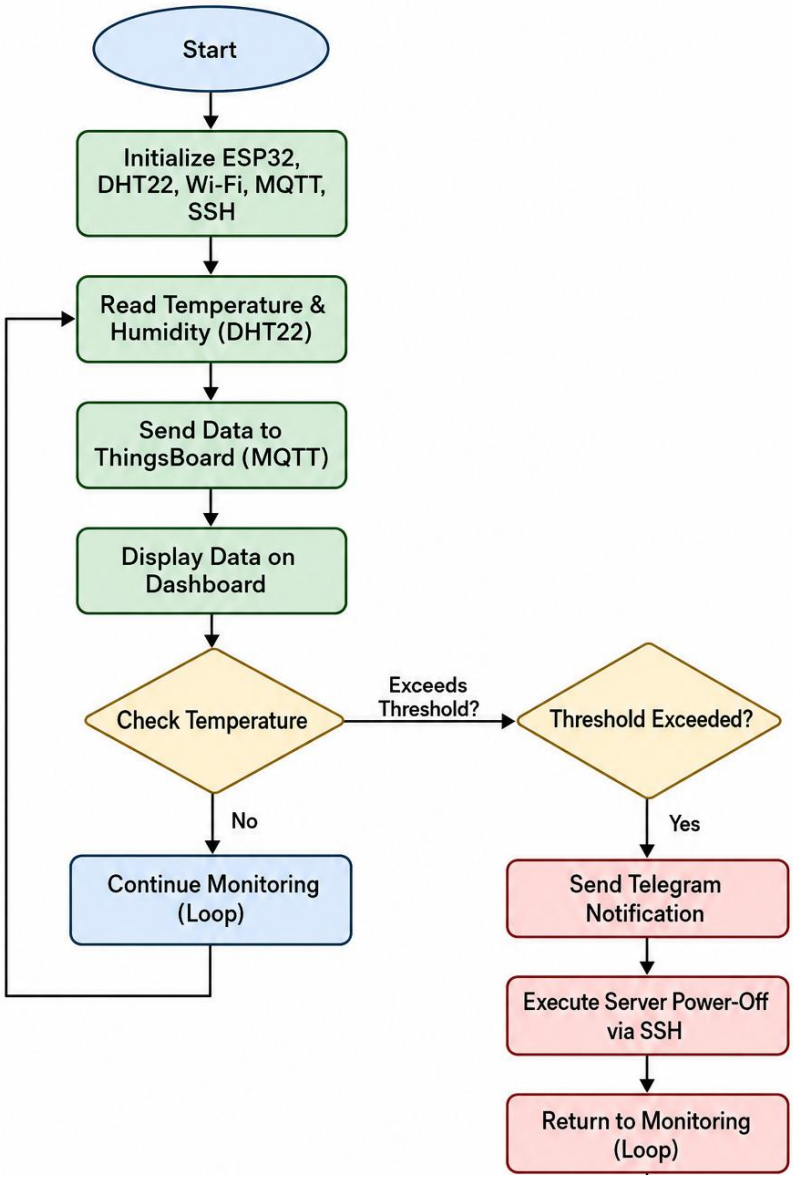

Fig. 1. Flowchart of the IoT-Based Server Room Temperature Monitoring and Automated Power Off System

Following data visualisation, the ESP32 compares the measured temperature with the predefined threshold, as shown in the decision process in Fig. 1. When the temperature is below or equal to the permitted limit, the condition is classified as normal, and the system proceeds to the "Continue Monitoring" stage. The workflow then returns to the DHT22 reading process, forming a continuous monitoring loop. When the measured temperature exceeds the threshold, the system enters the critical-condition branch. A warning notification is first sent via the Telegram Bot to inform the administrator that the server room temperature has reached an unsafe level. The ESP32 then sends a request to the Flask API, which acts as an intermediary between the monitoring device and the Ubuntu Server. Upon receiving the request, the Flask API initiates a secure connection through OpenSSH and executes the server power-off command. This sequence reduces the time between critical-temperature detection and protective action, thereby limiting the server's exposure to excessive heat. After the shutdown command is issued, the sensing subsystem returns to the monitoring loop and continues to monitor environmental conditions, keeping the temperature status available even after the protected server is powered off.

The hardware and software components used to implement the proposed monitoring and protection system are summarised in **Table 1**. The ESP32 DevKit V1 serves as the main controller, coordinating

©2026 The Author(s). Published by Scholar Publishing. This is an open access article under the CC BY license. Available online https://e-journal.scholar-publishing.org/index.php/ijsom

sensor acquisition, local output control, wireless communication, telemetry transmission, threshold evaluation, and activation of the automated protection mechanism. Its integrated 2.4 GHz Wi-Fi capability allows the system to connect directly to the local network and communicate with the MQTT broker without requiring an additional wireless module. The DHT22 sensor is connected to the ESP32 to measure server-room temperature within an operating range of −40°C to 80°C. The acquired temperature value is processed by the ESP32 and used both as telemetry data and as the primary input for determining whether the environmental condition remains within the predefined safe range. A 16 × 2 I2C LCD provides local visualisation of the measured temperature and system status. The I2C interface reduces the number of microcontroller pins required for the display, allowing the remaining pins to be allocated to other monitoring and warning components.

Table 1. Hardware and Software Specifications

No	Component	Specification	Function
1	ESP32 DevKit V1	Dual-Core Wi-Fi 2.4 GHz	Main controller
2	DHT22 Sensor	−40°C to 80°C	Measures server room temperature
3	16×2 I2C LCD	I2C Interface	Displays temperature and system status
4	Red LED	5 mm	Critical condition indicator
5	Active Buzzer	5 V	Audible warning device
6	ThingsBoard Community Edition	IoT Platform	Real-time monitoring dashboard
7	Telegram Bot	Telegram API	Sends notification messages
8	Ubuntu Server with OpenSSH	Ubuntu Server 24.04 LTS	Executes the remote power-off command
9	Flask API	Python Flask Framework	Bridges ESP32 and Ubuntu Server
10	Arduino IDE	C/C++ Development Environment	Firmware development

The local warning subsystem consists of a 5-mm red LED and a 5-V active buzzer. The red LED provides a visual indication when the measured temperature reaches a critical condition, while the active buzzer produces an audible warning that can immediately attract the attention of personnel near the server room. These components complement the cloud-based monitoring function by providing on-site alerts even when the administrator is not actively observing the dashboard. For remote monitoring, the system uses the ThingsBoard Community Edition as the IoT platform to receive, store, and visualise temperature telemetry. The ESP32 publishes sensor data to ThingsBoard via MQTT, enabling lightweight, efficient communication between the embedded device and the cloud platform (Nurfiqin et al., 2024). The dashboard allows administrators to view current temperatures and historical data in real time, while the Telegram Bot extends the warning mechanism by sending notifications to the administrator when the predefined threshold is exceeded. This combination of local and remote alerts improves the likelihood that abnormal temperature conditions will be detected promptly.

The automated protection subsystem is implemented on an Ubuntu Server 24.04 LTS system with OpenSSH and a Flask API. The Flask API acts as an intermediary between the ESP32 and the Ubuntu Server by receiving a request from the monitoring device when a critical temperature condition is detected. After validating the request, the Flask application initiates an SSH connection through OpenSSH and executes the remote server power-off command. The integration of Flask and OpenSSH, therefore, enables the system to move beyond passive monitoring and perform an immediate protective action without waiting for manual administrator intervention. OpenSSH also provides a secure channel for remote command execution, thereby reducing the risk of transmitting administrative commands over an unsecured network (Cahyani et al., 2022). Firmware for the ESP32 is developed and uploaded through the Arduino IDE using the C/C++ programming environment. Overall, the components listed in **Table** 1 form an integrated architecture comprising sensing, local display, warning, communication,

©2026 The Author(s). Published by Scholar Publishing. This is an open access article under the CC BY license. Available online https://e-journal.scholar-publishing.org/index.php/ijsom

visualisation, notification, and automated response layers. Their coordinated operation supports continuous monitoring of server room temperature and provides a structured mechanism to protect server hardware from prolonged exposure to excessive temperatures.

3. Result & Discussion

The results and discussion section evaluates the implementation and operational performance of the proposed IoT-based server room temperature-monitoring and automated protection system. The assessment focuses on the operation and integration of the ESP32 controller, DHT22 sensor, LCD, LED indicator, buzzer, MQTT communication, ThingsBoard dashboard, Telegram Bot, Flask API, and OpenSSH. Functional testing was conducted to verify the accuracy of temperature acquisition, the reliability of real-time telemetry transmission and visualisation, the delivery of warning notifications, and the execution of the automated server power-off command under critical temperature conditions. The results obtained are discussed by examining the responses of each hardware and software component, the effectiveness of communication among system layers, and the ability of the complete system to provide continuous monitoring and immediate protective action without requiring manual administrator intervention.

Fig. 2 presents the physical prototype of the proposed IoT-based server room temperature monitoring and automated power-off system during a simulated critical temperature condition. The prototype integrates an ESP32 DevKit V1 as the main controller, a DHT22 sensor for environmental data acquisition, a 16 × 2 I2C LCD for local information display, a red LED as a visual warning indicator, and an active buzzer for audible notification. The DHT22 sensor is positioned near the controller to measure the surrounding temperature, while the LCDs display the measured value and the corresponding system status. In the condition shown, the display indicates a temperature of approximately 35.6°C and presents the status "CRITICAL," confirming that the measured value has exceeded the predefined safety threshold. At the same time, the illuminated red LED indicates that the local warning mechanism has been successfully activated.

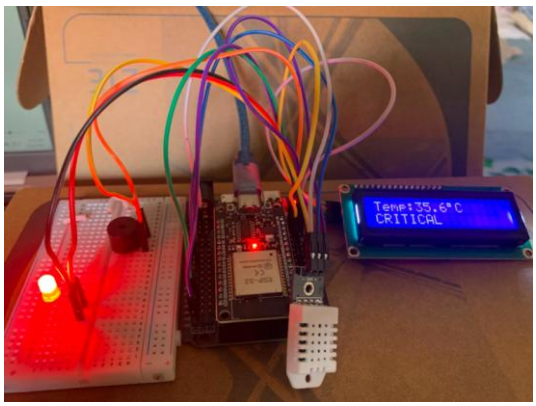

Fig. 2. Prototype of the IoT-Based Server Room Temperature Monitoring and Automated Power-Off System Under Critical Temperature Conditions

The prototype demonstrates the coordinated operation of the sensing, processing, display, and warning components under an abnormal environmental condition. After receiving the temperature reading from the DHT22, the ESP32 compares the measured value with the configured threshold and changes the system status from normal monitoring to critical response mode. This condition activates the LED and buzzer, updates the LCD information, and initiates subsequent remote protection processes, including transmitting a Telegram notification and executing the server power-off command via the Flask API and OpenSSH. The successful activation of the local indicators shown in **Fig. 2** confirms that the hardware subsystem responds correctly to excessive temperature. It also demonstrates that the prototype can provide immediate on-site warnings while supporting the automated protection mechanism designed to reduce the risk of server damage from prolonged overheating.

©2026 The Author(s). Published by Scholar Publishing. This is an open access article under the CC BY license. Available online https://e-journal.scholar-publishing.org/index.php/ijsom

Table 2 summarises the functional testing results for the principal hardware and communication components of the proposed IoT-based server room temperature monitoring system. The DHT22 temperature sensor successfully acquired environmental temperature data and transmitted the readings to the ESP32 controller for processing. The ESP32 also served as the central processing unit, receiving sensor data, comparing the measured temperature with the predefined threshold, controlling the output devices, and coordinating data transmission. The LCD correctly displayed the measured temperature and the corresponding system status, allowing users to observe the environmental conditions directly at the monitoring location. When the temperature exceeded the configured limit, the LED indicator and active buzzer were successfully activated to provide visual and audible warnings. These results confirm that the sensing, processing, display, and local alarm components operated according to the intended system logic during the functional tests.

Table 2. Functional Testing Results

No	Component	Test Result
1	DHT22 Temperature Sensor	Successful
2	ESP32 Controller	Successful
3	LCD Display	Successful
4	LED Indicator	Successful
5	Active Buzzer	Successful
6	MQTT Communication	Successful

The MQTT communication test was also successful, indicating that the ESP32 could establish a network connection and transmit temperature telemetry to the ThingsBoard platform using the publish–subscribe communication model. Successful MQTT operation is essential because it links the physical monitoring device with the remote dashboard and enables temperature information to be updated continuously. The combined results demonstrate that all tested components were integrated without functional conflicts and that data could move sequentially from the DHT22 sensor to the ESP32, local output devices, and cloud-based monitoring platform. However, the results in **Table 2** represent functional verification rather than a complete performance assessment. Therefore, the "Successful" classification confirms that each component performed its assigned function under the specified test conditions, but it does not independently establish measurement accuracy, communication latency, long-term stability, or system reliability under different network and environmental conditions.

Fig. 3 presents the real-time visualisation of temperature and humidity data received by the ThingsBoard platform from the ESP32-based monitoring device. The left side of the dashboard displays the latest environmental measurements in a pie chart, with humidity at approximately 73% and temperature at approximately 27% at the time of observation. This visualisation provides a simple overview of the DHT22 sensor's readings and allows the administrator to quickly identify the current server room condition. The successful display of both parameters on the dashboard confirms that the ESP32 acquired sensor readings, established MQTT communication, and published telemetry data to ThingsBoard. It also demonstrates that the cloud platform correctly received, processed, and displayed the transmitted measurements through a remotely accessible interface.

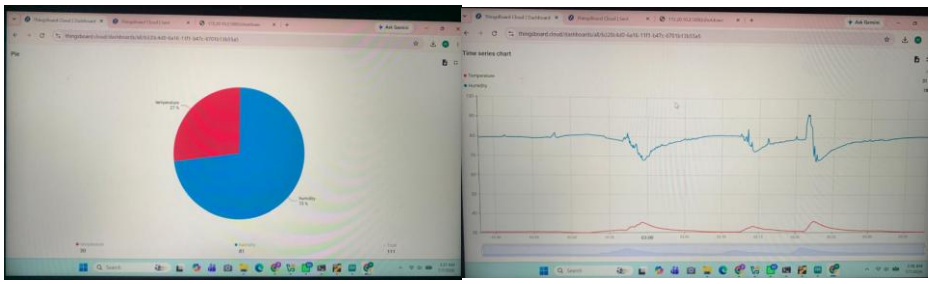

Fig. 3. Real-Time Temperature and Humidity Monitoring Visualisation on the ThingsBoard Dashboard

©2026 The Author(s). Published by Scholar Publishing. This is an open access article under the CC BY license. Available online https://e-journal.scholar-publishing.org/index.php/ijsom

The right side of **Fig. 3** displays the temperature and humidity measurements as time-series graphs, allowing changes in environmental conditions to be observed over the monitoring period. The blue line represents humidity, while the red line represents temperature. Variations in both curves indicate that the dashboard continuously updated the telemetry data as new measurements were received from the ESP32. The time-series visualisation is particularly useful for identifying gradual changes, sudden fluctuations, and recurring environmental patterns that may not be apparent from a single instantaneous reading. It also provides historical information to support the evaluation of server room conditions before and after a critical temperature event. Overall, the dashboard demonstrates that the ThingsBoard platform successfully performed real-time visualisation and historical recording, thereby providing administrators with both immediate environmental information and a broader view of temperature and humidity trends.

Fig. 4 illustrates the Telegram Bot notification interface used to report real-time server room temperature and humidity conditions to the administrator. The bot displays the monitoring status together with the latest environmental measurements obtained from the DHT22 sensor. Under normal conditions, the system reports temperatures of approximately 31.8–32.4°C and humidity of approximately 66.6–67.7%, indicating that the monitored environment remains below the predefined critical threshold. Each notification is automatically generated after the ESP32 processes the sensor readings and classifies the current condition according to the configured temperature limits. By presenting the status, temperature, and humidity in a structured message, the Telegram Bot enables the administrator to assess server room conditions remotely without having to access the ThingsBoard dashboard continuously.

Fig. 4. Telegram Bot Notifications for Normal, Warning, and Critical Server Room Temperature Conditions

The figure also demonstrates the escalation of notifications from normal to warning and critical conditions. When the measured temperature reaches approximately 34.7°C, the system issues a warning indicating that the environmental condition is approaching an unsafe level. When the temperature reaches approximately 35.0°C, the status changes to "CRITICAL SERVER," and a message indicates that the server power-off simulation has been initiated. This sequence confirms that the notification mechanism responds to temperature changes and provides progressively more urgent information as the measured value approaches or exceeds the configured threshold. The integration of Telegram notifications with the automated shutdown mechanism improves response effectiveness by allowing the administrator to receive immediate information while the system simultaneously initiates protective actions via the Flask API and OpenSSH. Therefore, the Telegram Bot functions not only as a remote alerting tool but also as an important component of the overall server protection workflow.

Fig. 5 presents the terminal-based execution of the automated server power-off mechanism through the integration of the Flask API and OpenSSH. The left terminal window shows the Ubuntu Server environment, which serves as the protected server and receives the remote shutdown instruction. It displays system information and command-line activity associated with the SSH service and server-side execution process. The right terminal window shows the Flask application operating as an intermediary between the ESP32 monitoring device and the Ubuntu Server. When the ESP32 detects that the measured temperature has exceeded the predefined critical threshold, it sends an HTTP request

©2026 The Author(s). Published by Scholar Publishing. This is an open access article under the CC BY license. Available online https://e-journal.scholar-publishing.org/index.php/ijsom

to the Flask API. The API receives the request, processes the associated endpoint, and initiates an SSH connection to the target server using the configured host address, username, and authentication credentials.

Fig. 5. Execution of the Automated Server Power-Off Command Through the Flask API and OpenSSH

The command-line output confirms the sequential operation of the automated protection mechanism, beginning with receipt of the shutdown request, followed by establishment of the SSH session and execution of the remote power-off command. The Flask application logs the request and displays the response generated during communication, while the Ubuntu Server terminal displays the resulting server-side command output. This implementation allows the system to perform protective action immediately after a critical temperature condition is detected, without requiring the administrator to log in manually and enter a shutdown command. The successful integration of the Flask API, OpenSSH, and Ubuntu Server demonstrates that the proposed system can connect environmental monitoring to secure remote server control. However, because the terminal output includes a warning associated with the Flask development server, deployment in an operational environment should use a production-grade web server, secure credential storage, restricted network access, and SSH key-based authentication to improve system security and reliability.

Table 3 summarises the functional testing results for the notification and automation subsystem of the proposed server room protection system. The Telegram notification function was deemed successful because the system delivered status messages to the administrator after processing temperature data from the DHT22 sensor. When the measured temperature reached the warning or critical range, the ESP32 generated a corresponding notification request, enabling the Telegram Bot to report the current status, temperature, and humidity. The Flask API communication test was also successful, confirming that the ESP32 could transmit an HTTP request to the Flask application when the predefined critical temperature threshold was exceeded. This communication stage is important because the Flask API serves as the bridge between the IoT monitoring device and the Ubuntu Server, translating the critical-condition signal into a server management action.

Table 3. Notification and Automation Testing Results

No	Parameter	Result
1	Telegram Notification	Successful
2	Flask API Communication	Successful
3	OpenSSH Connection	Successful
4	Automated Server Power-Off	Successful

The successful OpenSSH connection indicates that the Flask application established a secure remote session with the target Ubuntu Server and executed the configured administrative command. The automated server power-off test was likewise successful, demonstrating that the complete sequence from critical-temperature detection and Telegram notification to API communication, SSH connection, and remote shutdown execution operated in accordance with the proposed system design. These results show that the system can initiate protective action without requiring the administrator to perform the shutdown manually, thereby reducing the delay between overheating detection and protection of server hardware. Nevertheless, the “Successful” classification represents functional verification under the

©2026 The Author(s). Published by Scholar Publishing. This is an open access article under the CC BY license. Available online https://e-journal.scholar-publishing.org/index.php/ijsom

specified test conditions and does not by itself measure communication latency, long-term operational stability, authentication resilience, network-failure recovery, or the reliability of repeated shutdown executions. Further evaluation under different network conditions and extended operating periods would therefore be necessary to assess the robustness of the notification and automation subsystem.

The novelty of this study lies in the development of a closed-loop server room protection system that integrates environmental monitoring, cloud-based visualisation, remote notification, and automated server shutdown within a single IoT architecture. Unlike conventional monitoring systems that primarily display temperature data or send warnings and still depend on manual administrator intervention, the proposed system directly converts a critical temperature event into an immediate protective response. The integration of the ESP32 and DHT22 sensor with MQTT and ThingsBoard enables continuous data acquisition and real-time monitoring, while the Telegram Bot provides remote status notifications. More importantly, the Flask API and OpenSSH establish a secure communication channel to automatically execute a server power-off command when the predefined temperature threshold is exceeded. This coordinated sensing-to-action mechanism reduces response delays, limits server exposure to excessive heat, and strengthens the reliability of server room protection.

4. Conclusion

This study successfully designed and implemented an Internet of Things (IoT)-based server room environmental monitoring system integrated with an automated server power-off mechanism using Secure Shell (SSH). The developed system combines an ESP32 microcontroller, a DHT22 temperature and humidity sensor, MQTT communication, the ThingsBoard platform, a Telegram Bot, a Flask API, and OpenSSH running on an Ubuntu Server. This integrated architecture enables continuous environmental data acquisition, local status display, real-time cloud-based visualisation, remote warning delivery, and automated protective action when the measured temperature exceeds the predefined critical threshold. The functional testing results confirmed that all evaluated hardware, communication, notification, and automation components operated in accordance with the proposed system design. The DHT22 sensor successfully provided temperature and humidity readings to the ESP32, while the LCD, LED indicator, and active buzzer responded correctly to normal, warning, and critical conditions. Temperature and humidity telemetry was successfully transmitted through MQTT and displayed on the ThingsBoard dashboard in both numerical and time-series formats. The Telegram Bot delivered condition-based notifications to the administrator, and the integration of the Flask API with OpenSSH successfully established a remote connection and executed the automated server power-off command under a simulated critical temperature condition.

Overall, the proposed system demonstrates that IoT monitoring can be extended from passive observation to an automated closed-loop protection mechanism. By converting a critical temperature event into immediate notification and server shutdown actions, the system reduces dependence on manual administrator intervention and may limit prolonged server exposure to excessive heat. However, the present evaluation was limited to functional testing and did not assess sensor accuracy, communication latency, long-term operational stability, repeated shutdown reliability, or performance under network failure conditions. Future research should therefore include calibrated environmental measurements, extended reliability testing, redundant sensing, secure SSH key-based authentication, automatic cooling control, and predictive analytics based on historical temperature and humidity data.

Acknowledgement

The authors gratefully acknowledge the Department of Computer Engineering at Universitas Serambi Mekkah for providing the academic support, facilities, and resources required to conduct this research. The authors also extend their sincere appreciation to the lecturers, supervisors, technical staff, and colleagues who provided valuable guidance, constructive feedback, and assistance throughout the design, implementation, testing, and completion of this study.

©2026 The Author(s). Published by Scholar Publishing. This is an open access article under the CC BY license. Available online https://e-journal.scholar-publishing.org/index.php/ijsom

References

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of Things: A survey on enabling technologies, protocols, and applications. IEEE Communications Surveys & Tutorials, 17(4), 2347–2376. doi:10.1109/COMST.2015.2444095.

American Society of Heating, Refrigerating and Air-Conditioning Engineers. (2021). Thermal guidelines for data processing environments (5th ed.). ASHRAE.

Cahyani, I. D. (2022). Sistem enkripsi Secure Shell (SSH) untuk keamanan data. Publication details were not provided in the source manuscript.

Espressif Systems. (2025). ESP32 series datasheet (Version 5.2).

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, 29(7), 1645–1660. doi:10.1016/j.future.2013.01.010.

Ilmi, F. A., et al. (2024). Sistem monitoring suhu dan kelembaban pada ruang server berbasis Internet of Things. Saturnus: Jurnal Teknologi dan Sistem Informasi, 2(3), 95–105. doi:10.61132/saturnus.v2i3.186.

International Organisation for Standardisation. (2016). Information technology—Message Queuing Telemetry Transport (MQTT) v3.1.1 (ISO/IEC Standard No. 20922:2016).

International Organisation for Standardisation. (2021). Information technology—Data centre facilities and infrastructures—Part 4: Environmental control (ISO/IEC Standard No. 22237-4:2021).

Inzagi, F. M., et al. (2025). Sistem monitoring cuaca berbasis IoT dengan integrasi mikrokontroler ESP32 dan notifikasi real-time untuk penjemuran kerupuk (1), 1–6. Journal and publisher details were not provided in the source manuscript.

Kusuma, F., and Waluyo, S. (2025). Prototipe sistem monitoring suhu dan kelembaban ruang server berbasis IoT ESP32 dan DHT22. September issue, 122–129. Journal details were not provided in the source manuscript.

Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP, and HTTP. In 2017, IEEE International Systems Engineering Symposium (pp. 1–7). IEEE. doi:10.1109/SysEng. 2017.8088251.

Nurwarsito, H., and Adaby, R. W. (2024). Pengembangan Internet of Things (IoT) untuk perekaman data iklim mikro menggunakan platform ThingsBoard. Jurnal Teknologi Informasi dan Ilmu Komputer, 11(6), 1385–1398. doi:10.25126/jtiik.2024118987.

OASIS. (2014). MQTT version 3.1.1 (A. Banks & R. Gupta, Eds.). OASIS Standard.

OpenBSD Project. (n.d.). OpenSSH manual pages.

Pallets Projects. (n.d.). Flask documentation (Version 3.1.x).

Saptadi, A. H. (2015). Perbandingan akurasi pengukuran suhu dan kelembaban antara sensor DHT11 dan DHT22: Studi komparatif pada platform ATmega AVR dan Arduino. Jurnal Informatika, Telekomunikasi dan Elektronika, 6(2). doi:10.20895/infotel.v6i2.73.

Telegram. (n.d.). Telegram Bot API.

ThingsBoard, Inc. (n.d.). MQTT device API reference.

Ylonen, T., and Lonvick, C. (2006a). The Secure Shell (SSH) protocol architecture (RFC 4251). Internet Engineering Task Force. doi:10.17487/RFC4251.

Ylonen, T., and Lonvick, C. (2006b). The Secure Shell (SSH) transport layer protocol (RFC 4253). Internet Engineering Task Force. doi:10.17487/RFC4253.

©2026 The Author(s). Published by Scholar Publishing. This is an open access article under the CC BY license. Available online https://e-journal.scholar-publishing.org/index.php/ijsom