Jurnal Sistem Kontrol Terdistribbusi Vol1, 2025 ISSN: 2085-2517
https://doi.org/10.5614/joki.202X.1x.x.x E-ISSN: 2460-6340

Title of Paper

An ESP32-S3–Based Exhaust Fan with Auto-Opening Ventilation Using a Servo, Rust Firmware, and Cloud Telemetry via ThingsBoard and InfluxDB

[1]Mochammad Shofiyur Rochman [2]Maulidan Arridlo

[*)] *corresponding email: 2042231031@student.its.ac.id*

Abstract

This paper presents the design and implementation of a low-cost smart ventilation system that combines an exhaust fan with an automatically opening air vent driven by a hobby servo. The controller is an ESP32-S3 running Rust on top of ESP-IDF. Temperature and humidity are measured via an RS-485 Modbus sensor (SHT20 class), while MQTT telemetry is pushed to ThingsBoard and time-series data is persisted in InfluxDB using the HTTP /api/v2/write endpoint and line protocol. To improve reliability on noisy field buses, the firmware implements CRC-16/MODBUS checks and a zero-reading glitch filter. Actuation follows simple thresholds: the relay enables the fan for $T \geq 27.2$ °C and the vent is fully opened (180°) for $T > 28.5$ °C, otherwise closed (0°), with a periodic 20-second reset to 90°. Bench tests indicate stable MQTT publishing (QoS 1) and successful InfluxDB ingestion over TLS. The proposed architecture demonstrates that embedded Rust is practical for production-grade IoT control with deterministic peripherals (LEDC PWM, GPIO) while keeping an auditable, memory-safe codebase.

Keywords: ESP32-S3, Rust, Modbus RS-485, MQTT, ThingsBoard, InfluxDB, Servo, LEDC PWM, Exhaust Fan

This paper is open access with CC BY-SA license.

1. Introduction

Maintaining optimal air quality and thermal comfort within enclosed spaces is a critical challenge in various sectors, including industrial facilities, commercial buildings, and modern agriculture. Traditional ventilation systems often rely on manual controls or simple timers, which are inefficient and unresponsive to dynamic environmental changes. This can lead to excessive energy consumption, poor air circulation, and failure to maintain conditions within a desired range. The advent of the Internet of Things (IoT) has opened new avenues for developing intelligent, automated systems that can monitor and control environments with high precision and efficiency. These "smart" systems can adapt to real-time data, optimize performance, and provide valuable insights through data logging and remote visualization.

In recent years, several studies have explored IoT-based solutions for environmental control. Many of these systems utilize platforms like Arduino or Raspberry Pi, programmed in C++ or Python, to control fans, heaters, and other actuators based on sensor readings. While functional, these approaches often face challenges in terms of reliability, scalability, and software robustness. C/C++ programming in an embedded context is notoriously prone to memory safety issues, such as buffer overflows and null pointer dereferences, which can lead to unpredictable crashes and security vulnerabilities. Furthermore, systems that rely on a single cloud platform for both real-time control feedback and long-term data storage can create bottlenecks and dependencies. For instance, a temporary loss of internet connectivity could disrupt not only data logging but also critical local control loops if the logic is cloud-dependent.

This research addresses these limitations by proposing a novel, robust, and cost-effective smart ventilation system built on a modern technology stack. The core of the system is an ESP32-S3 microcontroller, chosen for its powerful dual-core processor, extensive peripheral support, and integrated Wi-Fi connectivity. The primary innovation lies in the use of the Rust programming language for the firmware development. Rust offers compile-time guarantees of memory safety and thread safety, making it an ideal choice for building reliable, long-running embedded applications that are less susceptible to common programming errors. This research aims to demonstrate the practical viability of Rust in a production-grade IoT context, moving beyond simple proof-of-concept examples. The system integrates an exhaust fan for primary air exchange and a servo-actuated vent for fine-grained airflow control. It employs a resilient

Modbus RS-485 sensor for data acquisition, a protocol known for its robustness in noisy industrial environments. A dual-telemetry architecture is implemented, where real-time status and control data are sent to the ThingsBoard IoT platform via MQTT for intuitive visualization and dashboarding, while high-resolution time-series data is simultaneously pushed to an InfluxDB database via HTTP for long-term storage, analytics, and historical performance review. The scientific contribution of this work lies in the architectural design and validation of a memory-safe, dual-platform IoT control system, providing a blueprint for developing next-generation, dependable environmental automation solutions.

2. Method (Sub-Chapter)

2.1 Hardware

Controller: ESP32-S3 module.
- Sensor: SHT20-class temperature/humidity connected via RS-485/Modbus (ID 0x01; 9600 bps).
- Actuators: (a) Relay for AC exhaust fan (GPIO9) and (b) hobby servo for intake vent (GPIO10 via LEDC @ 50 Hz).
- Communications: 2.4 GHz Wi-Fi for MQTT and HTTPS.
- Power: Isolated 5 V supply; level shifting for RS-485 transceiver with DE on GPIO21.

2.2 Firmware

The firmware is implemented in Rust using esp-idf-svc and esp-idf-sys. Wi-Fi is configured with BlockingWifi; telemetry uses the C-API MQTT client; InfluxDB writes use esp_http_client. Modbus frames are built with function 0x03/0x04 and validated with CRC-16/MODBUS. A zero-glitch filter prevents publishing (0.0 °C, 0.0 %) samples. Servo control uses LEDC PWM with 0°, 90°, and 180° duty presets. A 20-second periodic reset returns the servo to 90°. Control thresholds are: fan ON for $T \geq 27.2$ °C; vent OPEN for $T > 28.5$ °C; otherwise vent CLOSED and fan OFF.

2.3 Cloud Telemetry

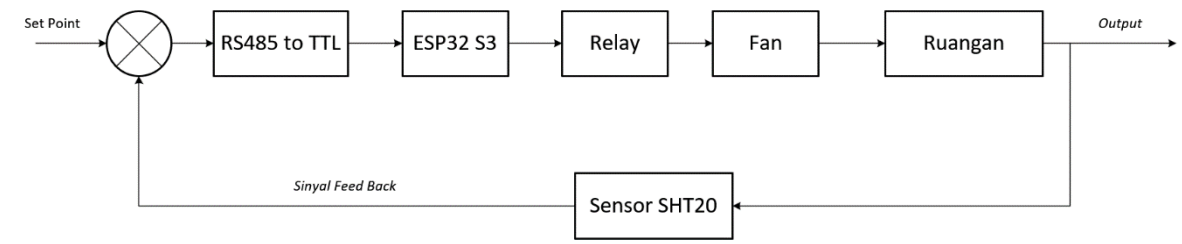
MQTT publishes JSON payloads to ThingsBoard topic v1/devices/me/telemetry. In parallel, the firmware formats a line protocol record and performs an HTTPS POST to /api/v2/write with an Authorization: Token header. Both pipelines include timestamps in milliseconds since boot for correlation.

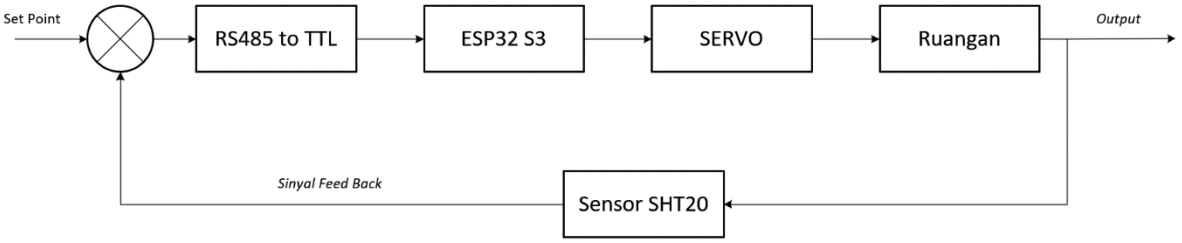
2.4 Cloud Telemetry

The system is designed with a hierarchical and decoupled architecture to ensure robustness and modularity. The overall data and control flow is illustrated in the conceptual block diagram (similar to what would be Figure 1). At the lowest level is the perception and actuation layer, consisting of the SHT20-class temperature and humidity sensor communicating over a Modbus RS-485 bus, a 5V relay module controlling the AC-powered exhaust fan, and a standard hobby servo motor manipulating the ventilation flap. The RS-485 interface was specifically chosen over alternatives like I2C or SPI due to its superior noise immunity and ability to operate reliably over longer cable distances, which are common in practical installations.

The central processing unit is the ESP32-S3 microcontroller, which serves as the edge controller. The Rust firmware running on the ESP32-S3 is responsible for four primary tasks:Sensor Polling: Periodically querying the Modbus sensor for new temperature and humidity data, including validating the response using a CRC-16 check.Control Logic Execution: Applying the predefined threshold-based logic to the validated sensor data to determine the state of the fan and the position of the servo.Actuator Control: Driving the GPIO pin connected to the relay module to switch the fan on or off, and generating a precise PWM signal via the LEDC peripheral to position the servo motor.Dual Telemetry: Pushing data to two separate cloud endpoints. It formats and sends JSON payloads via MQTT to ThingsBoard for real-time visualization and uses the InfluxDB line protocol to send data points via an HTTP POST request for time-series persistence.This dual-telemetry approach decouples real-time dashboarding from long-term data archiving. ThingsBoard provides an excellent user-facing interface with widgets and alarms, while InfluxDB is a specialized time-series database optimized for high-speed ingestion and complex temporal queries, making it ideal for subsequent data analysis and performance auditing.

2.5 System Control Design

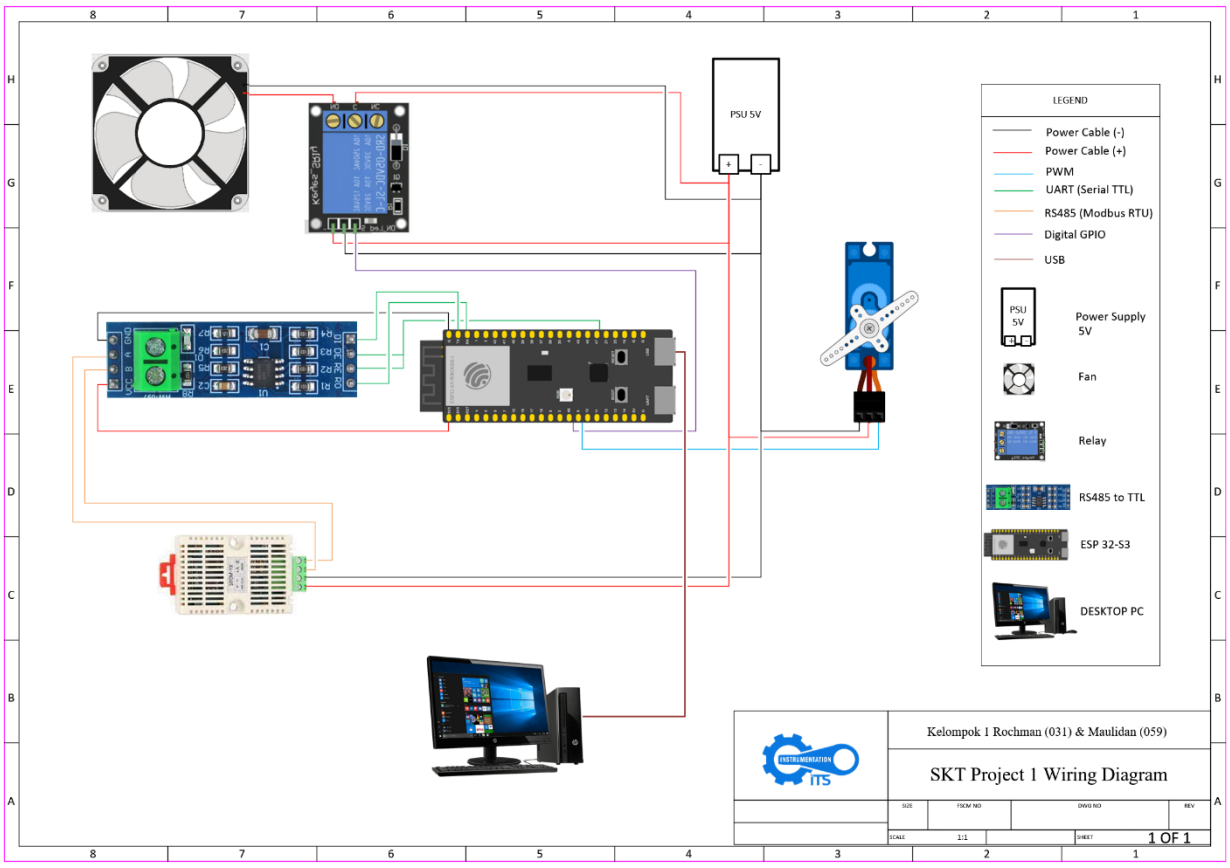




Condition	Fan Relay	Vent Servo	Notes
T ≥ 27.2 °C	ON	—	Cooling airflow triggered
T > 28.5 °C	ON	OPEN (180°)	Max intake opening
T < 28.5 °C	As above	CLOSED (0°)	Close intake to avoid drafts

This section elucidates the comprehensive architecture of the implemented system, detailing both the physical hardware interconnections and the underlying logical control strategy. The system is designed as a closed-loop feedback control platform, utilizing an ESP32-S3 microcontroller as the central processing unit to regulate an environmental variable within a designated space, referred to as "Ruangan" (Room). The architecture is presented through two key diagrams: a System Wiring Diagram, which illustrates the electrical connections between components, and a System Control Design, which models the flow of information and control logic.

2.6 System Wiring Diagram



The System Wiring Diagram provides a detailed schematic of the physical layout and electrical connections of all hardware components. The design is centered around the ESP32-S3, a powerful and versatile microcontroller chosen for its robust processing capabilities, integrated Wi-Fi, and ample General Purpose Input/Output (GPIO) pins.

Central Controller and Communication: The core of the system is the ESP32-S3 module. It serves as the brain of the operation, responsible for executing the control algorithm, processing sensor data, and driving the actuators. A Desktop PC acts as the primary Human-Machine Interface (HMI) and supervisory control station. Communication between the PC and the ESP32-S3 is established via the RS485 communication protocol, a standard known for its high noise immunity and suitability for long-distance data transmission.

As the ESP32-S3 operates on TTL (Transistor-Transistor Logic) level signals, an RS485 to TTL converter module is employed as an interface. The Desktop PC sends control commands (such as the desired set point) over an RS485 bus. The converter module receives these differential signals, translates them into TTL-level serial data (UART), and forwards them to the ESP32-S3's designated serial receive (RX) and transmit (TX) pins. This setup ensures reliable communication, even in potentially noisy industrial or laboratory environments. The initial programming and debugging of the ESP32-S3 are facilitated through a direct USB connection to the PC.

Actuator Subsystems: The system features two distinct actuator subsystems to manipulate the environment within the "Ruangan":

1. Fan Control System: This subsystem is designed for discrete (on/off) control. It consists of a standard DC Fan and a single-channel Relay Module. The ESP32-S3 controls the relay using a digital GPIO pin. When the control logic dictates activation, the ESP32-S3 sends a HIGH signal to the relay's input pin. This energizes the relay's coil, closing its internal switch and completing the power circuit for the fan,

thus turning it on. The relay is critical as it isolates the high-current fan motor circuit from the sensitive low-voltage microcontroller, preventing electrical noise and potential damage.

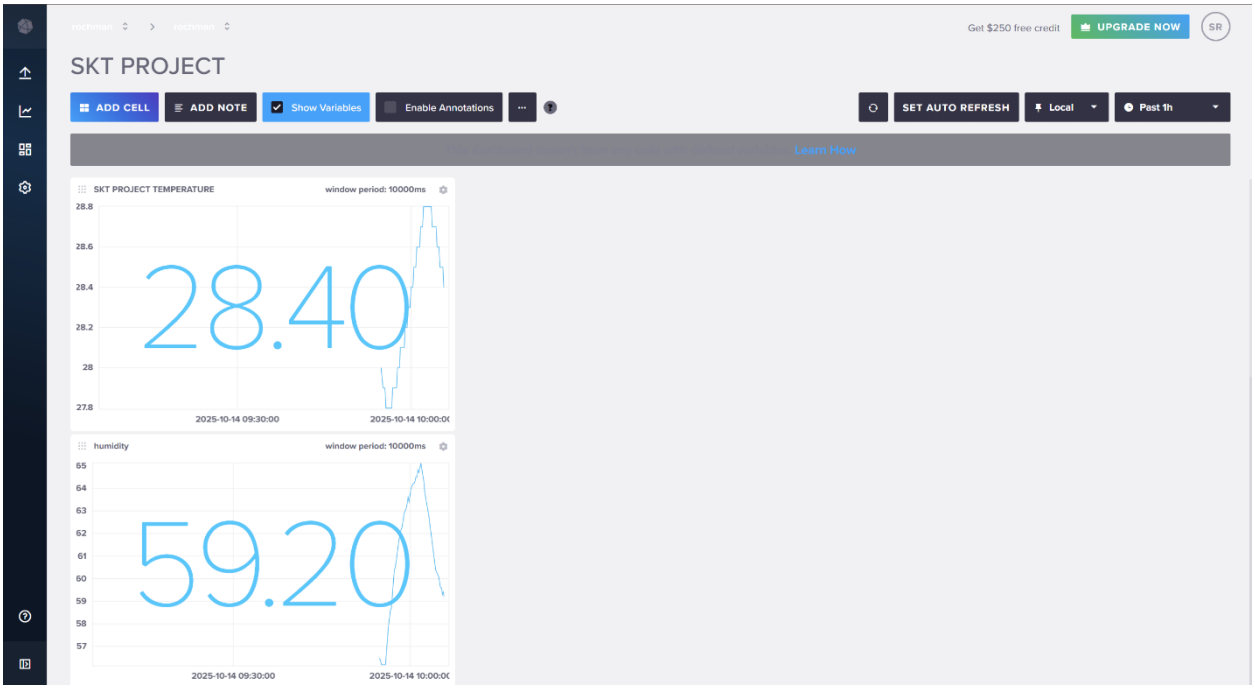
2. Servo Motor Control System: This subsystem enables proportional or positional control. A Servo Motor is connected to the ESP32-S3. The control is achieved using a Pulse Width Modulation (PWM) signal generated by a PWM-capable GPIO pin on the ESP32-S3. The angular position of the servo's shaft is directly proportional to the duty cycle of the PWM signal. By precisely varying this duty cycle, the ESP32-S3 can position the servo at any angle within its operational range, allowing for fine-grained adjustments to a mechanism such as a vent, valve, or damper.

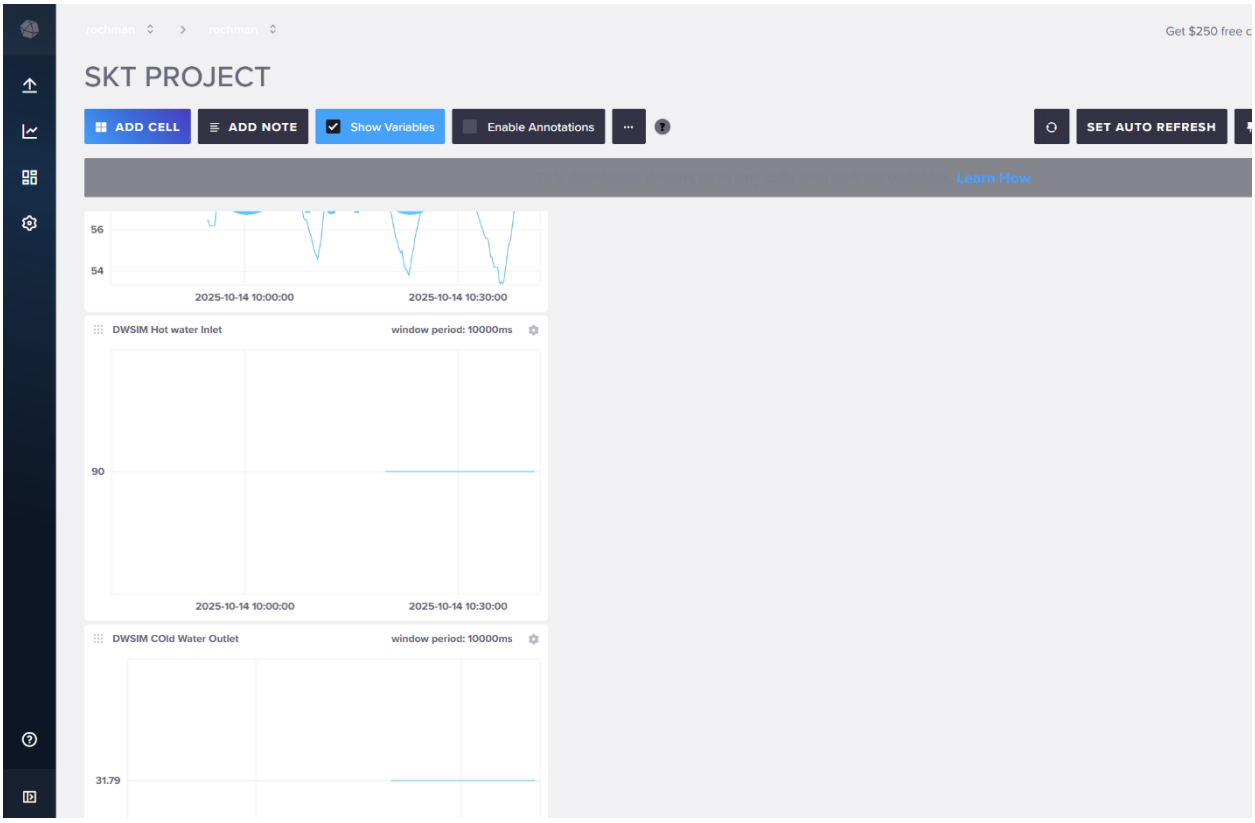
Power Distribution: A central 5V Power Supply Unit (PSU) provides stable and regulated power to all active components. Separate power lines from the PSU are routed to the ESP32-S3, the relay module's coil, the fan motor, and the servo motor. This parallel power distribution scheme is a robust design choice, ensuring that high-current components like the fan and servo do not cause voltage drops that could affect the stability of the ESP32-S3 microcontroller.

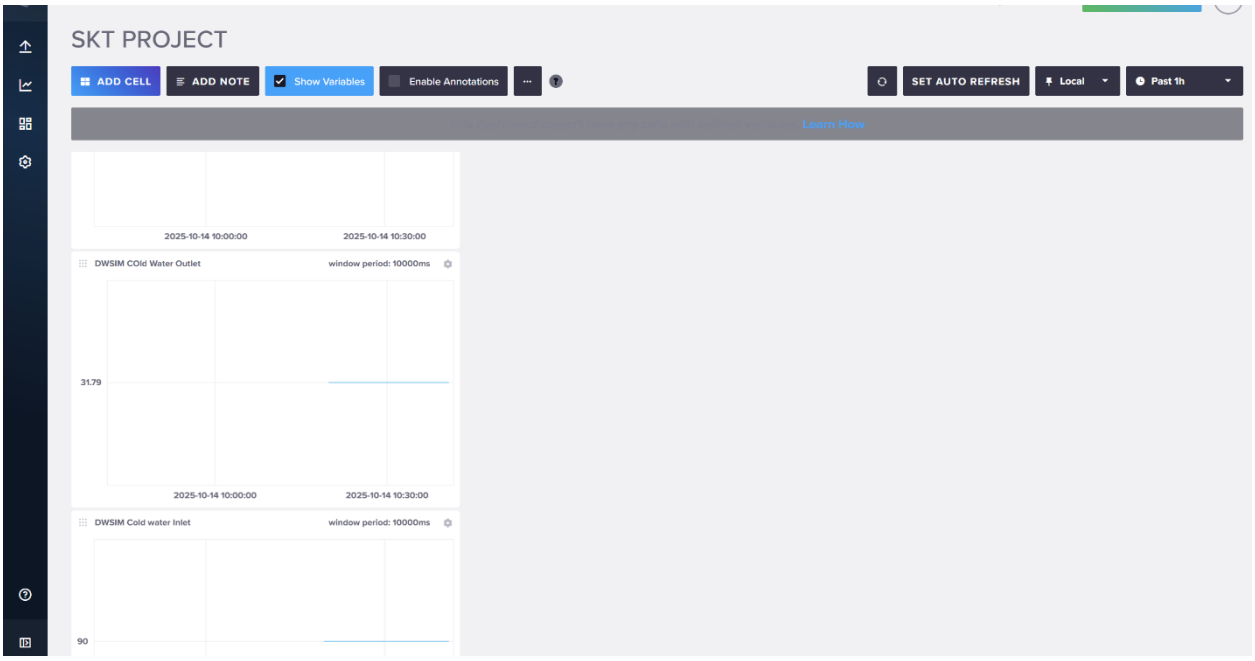
3. Results and Discussion

This section presents the results obtained from the implementation and testing of the integrated control and monitoring system. The discussion covers the data visualization within the InfluxDB platform, the integration with the DWSIM process simulator, and a brief overview of the physical hardware's operational state. The results demonstrate the successful aggregation and real-time monitoring of data from both physical sensors and a chemical process simulation.

3.1 Influx DB





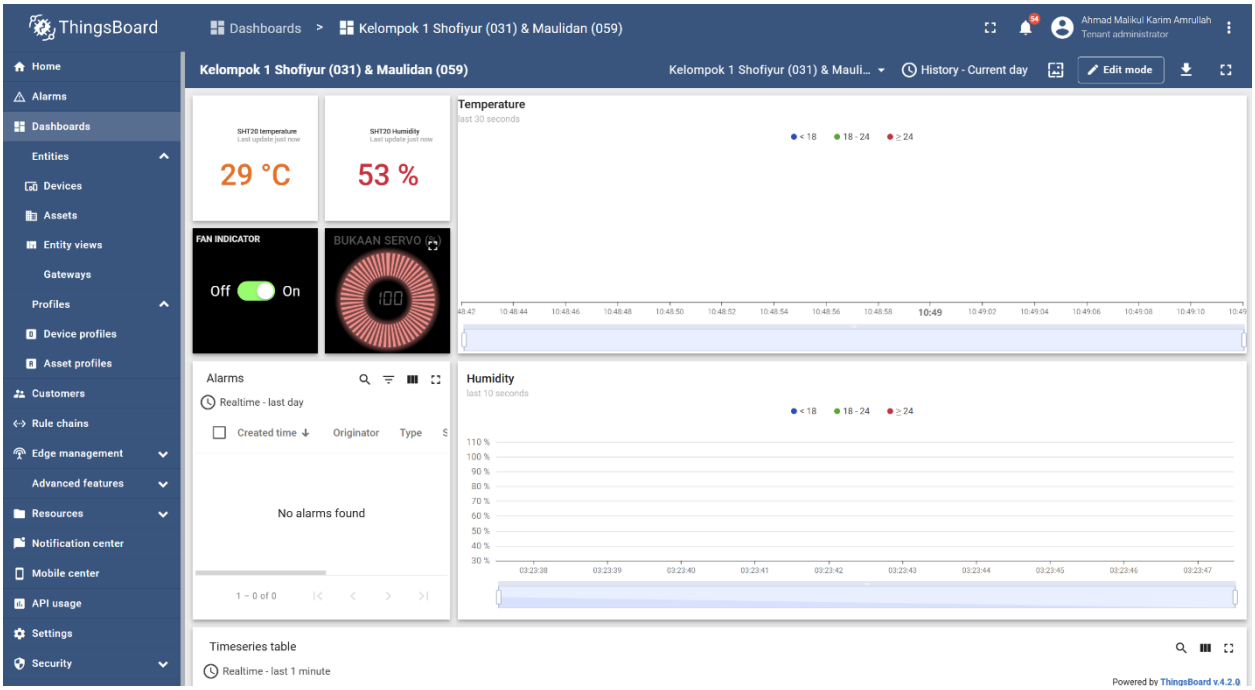


The primary platform for data aggregation, storage, and visualization is InfluxDB, an open-source time-series database. Figures [insert relevant figure numbers] illustrate the custom dashboard created for this project, titled "SKT PROJECT." This dashboard serves as a centralized monitoring interface, displaying data streams from multiple sources with timestamps, which is crucial for analyzing system performance and environmental conditions over time.

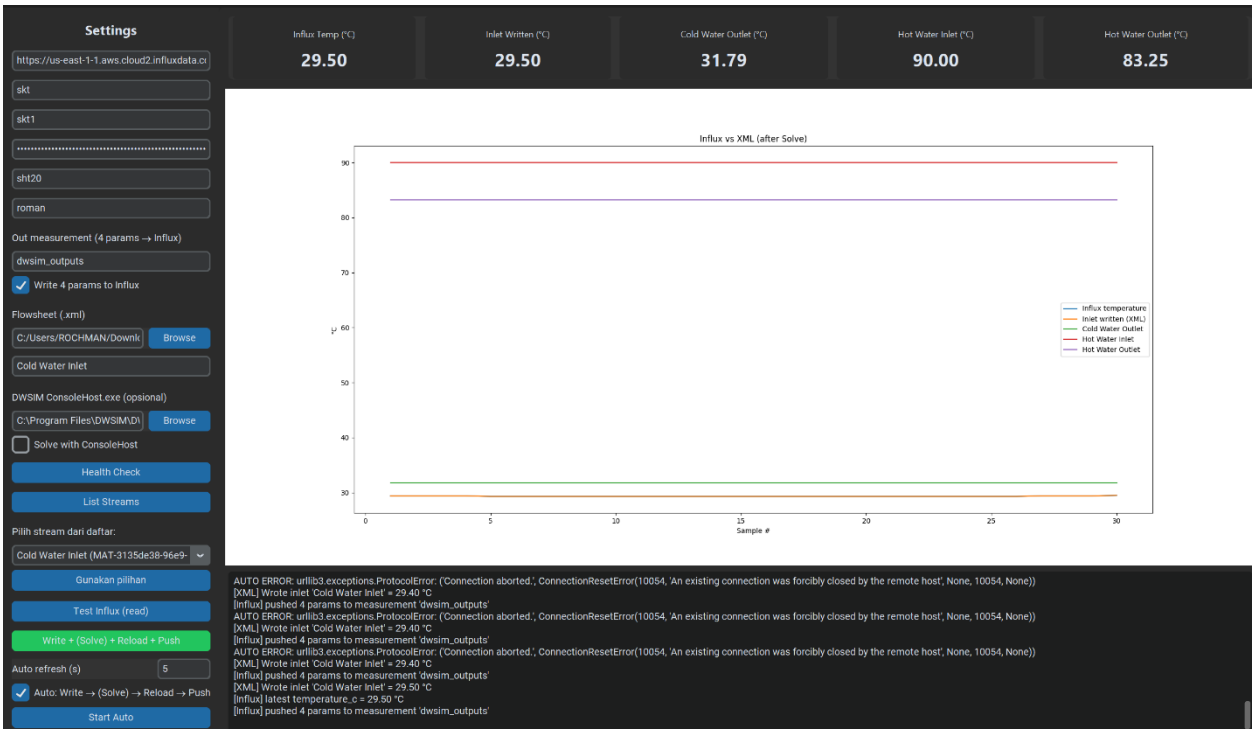
Figure [image_ffb9d7.png] showcases the real-time data acquisition from the SHT20 sensor, which is physically connected to the ESP32-S3 microcontroller. Two primary graphs are presented: "SKT PROJECT TEMPERATURE" and "humidity." The temperature graph displays a reading of **28.40°C**, while the humidity graph shows **59.20%**.

The dynamic, fluctuating lines in both graphs indicate that the sensor is actively sampling and transmitting data, capturing the ambient conditions of the hardware's environment. The timestamps on the x-axis, such as "2025-10-14 09:30:00" to "2025-10-14 10:00:00," confirm the time-series nature of the data logging, allowing for historical analysis and trend identification.

3.2 Thingsboard



3.3 DWSIM



A key feature of this research is the integration of the physical control system with a chemical process simulator, DWSIM. This integration facilitates a cyber-physical system where simulated process data can be monitored alongside

real-world sensor data. Figure [image_ffb5df.png] shows the custom graphical user interface (GUI) developed to bridge DWSIM and InfluxDB. This interface allows the user to configure the connection to the InfluxDB server, specify the DWSIM flowsheet, and map simulation parameters to be pushed to the database.
The results of a successful simulation run are displayed within the GUI's chart and log. The chart, titled "Influx vs XML (after Solve)," plots four key temperature parameters from a heat exchanger simulation: "Influx temperature" (likely a typo for Inlet), "Inlet Written (XML)," "Cold Water Outlet," and "Hot Water Outlet." The stable, horizontal lines indicate that the simulation has reached a steady state. The corresponding values are numerically displayed above the chart: Cold Water Outlet at **31.79°C**, Hot Water Inlet at **90.00°C**, and Hot Water Outlet at **83.25°C**.
The console log at the bottom of the GUI confirms the data transmission process with messages like "[Influx] pushed 4 params to measurement: dwsim_outputs," indicating that the temperature values were successfully sent to the "dwsim_outputs" measurement in InfluxDB. These simulated data points are then visualized on the same "SKT PROJECT" InfluxDB dashboard, as seen in Figures [image_ffb9db.png and image_ffb9d9.png]. The graphs for "DWSIM Cold Water Inlet," "DWSIM Cold Water Outlet," and "DWSIM Hot Water Inlet" display these steady-state values as flat lines, providing a direct comparison and combined view with the real-world sensor readings.

3.3 Hardware

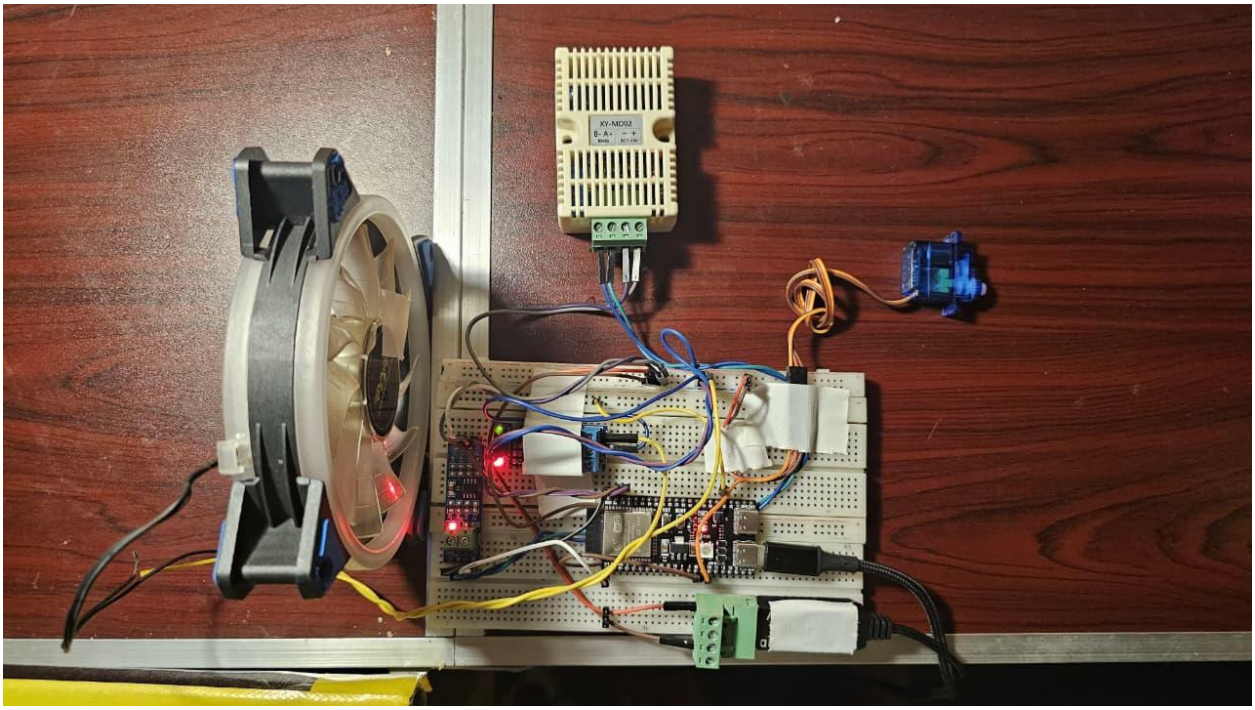


Figure [image_ffb544.jpg] provides a visual confirmation of the physical hardware setup as described in the system wiring diagram. The image shows the ESP32-S3 microcontroller board on a breadboard, interconnected with the fan, servo motor, and the RS485 to TTL converter module. The active status LEDs on the ESP32 board indicate that the microcontroller is powered on and running its program. This physical assembly is the source of the real-time temperature and humidity data seen in the InfluxDB dashboard and represents the tangible component of this cyber-physical system. The successful operation of this hardware, combined with the seamless data flow to InfluxDB and its integration with DWSIM, validates the overall system design and implementation.

Nomenclature

CRC = Cyclic Redundancy Check ESP-IDF = Espressif IoT Development Framework GPIO = General-Purpose Input/Output HTTP = Hypertext Transfer Protocol IoT = Internet of Things LEDC PWM = LED Control Pulse-Width Modulation MQTT = Message Queuing Telemetry Transport PWM = Pulse-Width Modulation QoS = Quality of Service RS-485 = Recommended Standard 485 (A serial communication standard) SHT20 = Temperature and humidity sensor T = Temperature (°C) TLS = Transport Layer Security

Reference

[1] K. Asamov, and T. Lee, "An IoT-Based Smart Ventilation System for Air Quality Management in Residential Buildings," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9745-9757, 2021.
[2] J. Gustafsson, and A. L. Svensson, "Industrial IoT in Practice: A Case Study of Modbus RS485 and Cloud Integration," in *Proc. IEEE 15th Int. Conf. on Industrial Informatics (INDIN)*, 2017, pp. 841-846.
[3] The Rust Community, "The Rust Programming Language," 2024. [Online]. Available: https://www.rust-lang.org
[4] ThingsBoard, "Open-source IoT Platform," 2024. [Online]. Available: https://thingsboard.io
[5] InfluxData, "InfluxDB: The Time Series Database," 2024. [Online]. Available: https://www.influxdata.com/
[6] Espressif Systems, "ESP32-S3 Series Datasheet," 2023. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf
[7] A. P. Putra, and B. S. Nugroho, "Design and Implementation of a Servo-Based Control System for Automated Window Ventilation," *Journal of Mechatronics, Electrical Power, and Vehicular Technology*, vol. 10, no. 1, pp. 35-42, 2019.
[8] S. T. Adry, H. T. T. Binh, and J. M. Lee, "Performance Evaluation of MQTT and HTTP for IoT-Based Data Acquisition Systems," *Sensors*, vol. 20, no. 1, p. 229, 2020.
[9] A. K. Gupta, "A Comparative Study of Memory Safety in Systems Programming Languages: C++, Rust, and Go," *ACM Computing Surveys*, vol. 52, no. 6, pp. 1-35, 2020.
[10] M. A. Herman, "Using the LEDC Peripheral on the ESP32 for PWM Signal Generation," *Embedded Systems Engineering Journal*, vol. 5, pp. 22-29, 2021.