SAMSUNG

IoT Course

Capstone Project

Final Report

‹ Developing an IoT system to monitor smart greenhouse environment via sensor network›

‹ Date (16/07/2025) ›

GROUP_2_TNT

‹Hà Minh Nghĩa›

‹Hà Đức Thắng›

‹Phạm Văn Trung›

‹Hoàng Quốc Việt Quang›

‹Lương Thế Vinh›

Content

1. Introduction

1.1. Background Information

The agricultural sector today faces a critical turning point. While agriculture remains a cornerstone of global food security and rural development, it is increasingly under pressure from both environmental and socio-economic factors. Many farming communities continue to rely on outdated tools, techniques, and intuition-driven decision-making processes. These traditional practices, though time-tested, are becoming increasingly insufficient in the face of modern challenges. A transformation towards data-driven and technology-assisted agriculture is not just desirable but essential.

One of the foremost challenges confronting traditional agriculture is climate instability. Farmers are grappling with the growing frequency and intensity of extreme weather phenomena such as droughts, floods, heatwaves, and unexpected frosts. These climate anomalies disrupt planting and harvesting schedules, reduce crop yields, and can even result in total crop failures. The unpredictability of weather patterns makes long-term planning nearly impossible without reliable environmental data.

In addition, inefficient resource management continues to plague traditional farming systems. Conventional methods of irrigation, fertilization, and pest control are often guided by routine or guesswork, rather than precise measurements. This results in overuse or underuse of vital resources such as water and fertilizers. Over time, these practices lead to economic inefficiencies, environmental degradation, and loss of soil fertility. Fertilizer runoff contaminates local water sources, pesticide overuse harms beneficial insects and biodiversity, and unchecked irrigation can lead to waterlogging or salinization.

Equally problematic is the lack of access to accurate, real-time environmental data. Farmers, especially in smallholder and rural settings, often operate without reliable information on soil moisture levels, ambient temperature, humidity, or

light intensity. Decisions about planting, watering, fertilizing, or harvesting are made based on experience, which—while valuable—cannot always account for rapidly changing environmental conditions. This absence of data results in reactive rather than proactive farm management.

As a result of these compounding issues, many farms experience suboptimal yields and inconsistent produce quality. Environmental stresses that could have been mitigated go undetected. Diseases or pest infestations are only identified when damage is already extensive. The quality and quantity of produce become unpredictable, affecting marketability and profitability.

Together, these factors highlight an urgent need for a systemic shift towards smart agriculture. By embracing Internet of Things (IoT) technologies and precision farming tools, agriculture can become more responsive, efficient, and sustainable. IoT-enabled monitoring systems can provide farmers with granular, real-time insights into their crops and environmental conditions, allowing them to make informed, timely decisions. Smart agriculture is not simply a technological upgrade; it is a pathway to resilience, sustainability, and food security in an era of uncertainty.

1.2. Motivation and Objectives

The motivation for this project stems from a desire to close the gap between traditional farming practices and modern agricultural technologies. Specifically, the project aims to explore how low-cost, accessible IoT solutions can empower farmers to better monitor and control their greenhouse environments, leading to improved crop health, increased productivity, and reduced environmental impact.

This project proposes the design and implementation of a comprehensive IoT-based environmental monitoring system tailored for smart greenhouse applications. The system will serve as a model for integrating real-time data collection, storage, visualization, and intelligent decision-making into a compact and practical platform suitable for small- to medium-sized greenhouses.

Key Objectives:

1. Hardware Development:

- Design and assemble a reliable hardware infrastructure that includes a network of sensors such as DHT11 (temperature and humidity), soil moisture sensors, and light-dependent resistors (LDRs).
- Utilize an ESP32 microcontroller as the central processing unit to collect and transmit data.
- Ensure all hardware is power-efficient, robust, and appropriately calibrated for accurate measurements in real greenhouse conditions.

2. Communication and Data Infrastructure:

- Implement MQTT (Message Queuing Telemetry Transport) protocol for lightweight and efficient communication between the ESP32 and the server.
- Set up a local or cloud-based server equipped with InfluxDB, a high-performance time-series database optimized for IoT data.
- Develop routines for reliable data acquisition, buffering, and storage.

3. Visualization and User Interface:

- Create a real-time dashboard using Grafana to visualize temperature, humidity, soil moisture, and light levels.
- Include threshold-based alerting mechanisms that notify users when environmental variables fall outside optimal ranges.
- Ensure the interface is user-friendly and accessible from mobile and desktop devices.

4. Foundations for Automation:

- Although full automation is outside the immediate scope of this version, the system will be designed with extensibility in mind, enabling future integration with actuators (e.g., water pumps, ventilation systems).
- Establish groundwork for decision-making algorithms that can be added to trigger automatic environmental control based on sensor input.

5. Impact and Practical Relevance:

- Provide farmers and greenhouse operators with actionable insights that were previously inaccessible.
- Promote water conservation, reduce unnecessary chemical usage, and improve crop consistency.
- Contribute to the broader movement toward sustainable and climate-resilient agriculture.

By accomplishing these objectives, this project serves as a stepping stone toward the adoption of smart farming practices in rural and developing regions. It demonstrates that advanced agricultural technologies do not have to be expensive or complex—they can be accessible, scalable, and tailored to local needs, thereby making a meaningful impact on food production systems worldwide.

1.3. Members and Role Assignments

This project was completed by a three-member team, each responsible for a specific domain of the system:

- Hà Minh Nghĩa – Team Leader

  As the team leader, Nghĩa was primarily responsible for the overall architecture and integration of the system. He designed the data pipeline from the ESP32 microcontroller to the server infrastructure, ensuring reliable and real-time communication via the MQTT protocol. In addition, he configured and managed the InfluxDB time-series database used for storing sensor data efficiently.

  Nghia also built a custom Grafana dashboard, which allowed for real-time visualization of temperature, humidity, and soil moisture data. Beyond technical implementation, he oversaw the project's progress, assigned tasks, led group discussions, and played a key role in preparing the final presentation materials.

- Phạm Văn Trung – Embedded System Developer

  Trung took charge of the firmware development for the ESP32 microcontroller. He wrote the embedded C/C++ code to interface with sensors such as the DHT11 and Soil Moisture Sensor, enabling the ESP32 to read analog/digital values and publish them through MQTT to the server.

  His work included writing non-blocking loops, configuring GPIO pins, handling sensor delays, and implementing reconnection logic to ensure data was transmitted even after network interruptions. He also supported system testing and debugging to ensure firmware stability throughout.

- Hà Đức Thắng – Hardware Developer

  Thắng was responsible for the hardware development and physical assembly of the prototype. He carefully selected compatible components, including voltage regulators and resistors, and designed the circuit layout to ensure clean wiring and safe power distribution.

  Working closely with Trung, he helped connect the sensors to the ESP32, verified power stability, and tested signal outputs using basic electronic tools such as multimeters. His attention to detail in hardware ensured the entire circuit was physically robust and electrically reliable.

  Responsible for circuit design and hardware assembly, including the selection and configuration of sensors (DHT11 and soil moisture sensor).

- Hoang Quoc Viet Quang

  Quang played a key role in the visual and presentation aspects of the project. He designed a clean and informative poster that summarized the project's objectives, system architecture, features, and results in a way that was both technically accurate and visually appealing.

  In addition, he contributed to preparing the final demo setup, including organizing the table display, supporting the team during the live demonstration, and helping Nghĩa refine the presentation slides. His work ensured that the team's technical efforts were effectively communicated to the audience.

- Luong The Vinh

  Vinh was in charge of project documentation and team coordination. She kept a detailed record of daily progress, meeting notes, and testing outcomes. Throughout the project, she maintained a shared folder of images, source code backups, schematic drafts, and intermediate reports.

  She also acted as a timeline manager, sending regular reminders to ensure deadlines were met. Finally, she compiled all relevant materials into a complete final report, organized in a logical, easy-to-follow format ready for submission and archiving.

Each member contributed to the planning, development, testing, and reporting phases to ensure a successful project delivery.

1.4. Schedule and Milestones

Member	Detailed Tasks	Time Period
Ha Minh Nghia (Team Leader, Full-stack Developer)	- Take lead on all technical development - Select suitable sensors for environmental monitoring (e.g., temperature, humidity, light) - Program the complete firmware for the ESP32 microcontroller to read and transmit sensor data - Configure MQTT Broker for real-time data transmission - Set up InfluxDB for time-series data storage - Design and implement a real-time dashboard using Grafana - Prepare presentation slides, script, and lead final demo	01/07 – 05/07: Sensor selection & ESP32 firmware 06/07 – 08/07: MQTT + InfluxDB setup 09/07 – 11/07: Grafana dashboard implementation 12/07 – 16/07: Presentation slide + rehearsal
Pham Van Trung (Hardware Developer)	- Assemble the entire hardware prototype - Solder and wire components such as sensors, ESP32, resistors, and jumpers - Ensure all sensor modules are firmly connected and placed correctly - Assist with powering up and validating each hardware part - Conduct initial tests to confirm hardware responds correctly - Write the hardware development section in the final report (challenges, solutions, circuit summary)	01/07 – 04/07: Circuit assembly and wiring 05/07 – 06/07: Support in testing and validation 13/07 – 15/07: Report writing (hardware section)
Ha Duc Thang (Hardware Developer)	- Double-check and verify all electronic components	01/07 – 03/07: Component verification

	before usage - Support Trung in soldering and wiring - Ensure all electrical connections are neat, correct, and labeled - Measure voltage and test sensor values manually for accuracy - Document full list of components and their specifications - Draw final circuit schematic using software (e.g., Fritzing) - Summarize test results and insert into final report	04/07 – 05/07: Hardware prototyping support 14/07 – 15/07: Report writing (technical specs & test results)
Hoang Quoc Viet Quang (Presentation & Visual Support)	- Design visual materials such as a project poster or infographic - Build a small model/mockup of the system for demo purposes (if required) - Assist in capturing demo footage or helping edit video - Support slide formatting and ensure smooth visual transitions - Coordinate with Nghia to refine content for live presentation	10/07 – 13/07: Poster & model design 14/07 – 16/07: Presentation support (slide, visuals, demo setup)
Luong The Vinh (Documentation & Logistics)	- Take daily notes of team meetings and development sessions - Keep record of progress (including photos of circuit stages, screenshots of code/dashboard) - Ensure all team members follow deadlines and provide reminders - Organize and archive all source files, reports, and test logs	01/07 – 16/07: Ongoing documentation and team progress tracking

2. Project Execution

2.1. IoT Service Model

To build our smart environment monitoring system, we selected core hardware components based on their efficiency and cost-effectiveness. Our system includes:

- ESP32- 3C Microcontroller: This is the brain of our system, chosen for its integrated Wi-Fi and Bluetooth capabilities, which make it easy to collect and transmit data to a server.

- DHT11 Temperature and Humidity Sensor: A common and reliable sensor used to measure two of the most critical environmental parameters in a greenhouse.

- Light Sensor: Used to measure light intensity, providing us with a comprehensive view of the external environmental conditions to make optimal decisions for the crops.

- Relay Module: Used to activate the water pump, allowing for automated irrigation when the soil moisture drops below a set threshold.

This combination of components allows us to create a complete IoT system capable of gathering accurate data and automating necessary tasks.

2.1.1. ESP32-3C

ESP32-C3 is a single-core Wi-Fi and Bluetooth 5 (LE) microcontroller SoC, based on the open-source RISC-V architecture. It strikes the right balance of power, I/O capabilities and security, thus offering the optimal cost-effective solution for connected devices. The availability of Wi-Fi and Bluetooth 5 (LE) connectivity not only makes the device's configuration easy, but it also facilitates a variety of use-cases based on dual connectivity.

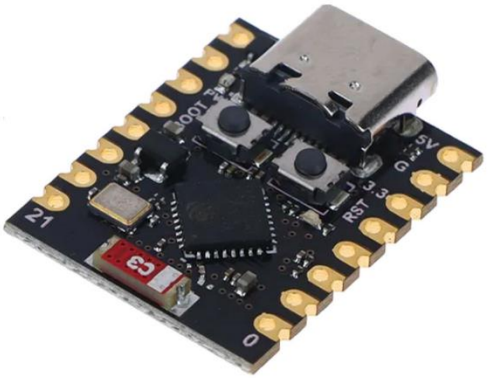


Figure 2.1. ESP-3C

- Application: Acts as the main controller of the system. It collects sensor data, processes environmental information, controls the relay to activate devices (e.g., pump, fan), and sends data to the cloud or local server via Wi-Fi for monitoring.
- In the greenhouse system:
- Connects all sensors and actuators.
- Enables wireless communication for remote monitoring and control.
- Executes decision-making logic (e.g., turn on pump when humidity is low).

2.1.2. DHT11

The DHT11 is a basic, ultra low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air and spits out a digital signal on the data pin (no analog input pins needed). It's fairly simple to use but requires careful timing to grab data. The only real downside of this sensor is you can only get new data from it once every 2 seconds, so when using our library, sensor readings can be up to 2 seconds old.

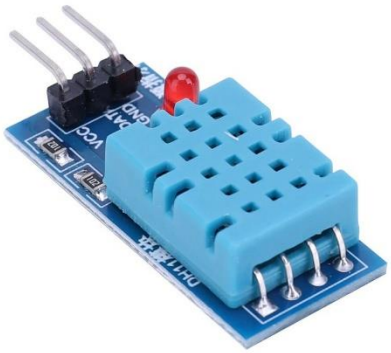


Figure 2.2. DHT11

- Application: Provides essential data on environmental conditions, allowing the system to assess whether the greenhouse is too hot or too dry.
- In the greenhouse system:
  - Monitors air temperature and humidity.
  - Triggers actions such as turning on a fan or water pump when values exceed or fall below thresholds.
  - Helps maintain a stable microclimate for plant development.

2.1.3. Module Relay

A relay module is a switching device, the control circuit that operates with low-power signals. It enables a low-power supply circuit to switch on or regulate a high-power supply circuit without integrating it with the same circuit or electrical appliance. In other words, relay modules are employed to break the different parts of the given system to mitigate the problems of electrical coupling or failure.

Figure 2.3. Module Relay

- Application: Serves as a switch that controls high-power electrical devices based on control signals from the ESP32-C3.
- In the greenhouse system:
  - Controls the water pump to irrigate plants automatically.

– Can be extended to control fans, lights, or heating systems.
– Enables automation of greenhouse equipment for consistent operation.

2.1.4. Light sensor

Light Sensor V1.2 is an analog ambient light sensor module that uses a photoresistor (LDR – Light Dependent Resistor) to detect the intensity of light in the environment. Its resistance varies based on the amount of light: the brighter the light, the lower the resistance, and vice versa. This change in resistance is converted into a voltage signal that can be read by an analog input on a microcontroller. Due to its simplicity and low power consumption, it is widely used in light-sensitive applications.

The sensor operates on 3.3V to 5V, making it compatible with a wide range of development boards, including ESP32, Arduino, and STM32. It provides real-time analog output corresponding to the ambient light level, which can be used for threshold-based control systems or data logging.

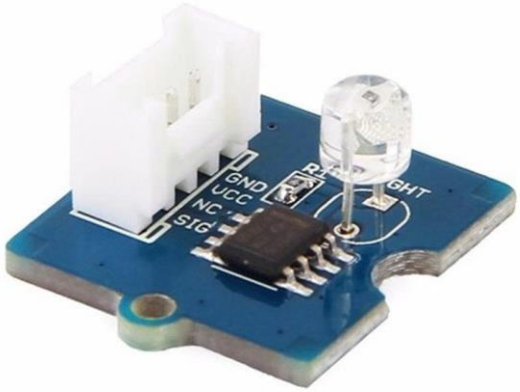


Figure 2.4. Light sensor

➢ Application: Light Sensor V1.2 is used to monitor ambient light conditions. It provides the system with information about environmental brightness, enabling automated control based on light levels (e.g., switching lights on at night or closing shades when it's too bright).
➢ In the greenhouse system:
– Measures ambient light intensity to determine sunlight levels inside the greenhouse.
– Assists in automating environmental controls such as adjusting grow lights or shading mechanisms.
– Provides real-time light data for logging and analysis in monitoring dashboards.

2.2. Software Tools Selection

To build an efficient and scalable greenhouse monitoring and control system, the team selected a set of modern and reliable software tools. These tools support embedded development, real-time data transmission, time-series data storage, and automation workflows — all essential for an IoT-based system.

2.2.1. Rust Programming Language

Rust is used for programming the ESP32-C3 microcontroller. It is a modern systems programming language that can serve as a safer and more reliable alternative to C/C++ for embedded development.

- Key Features:
  - Memory safety without requiring a garbage collector.
  - High performance, comparable to C/C++.
  - Strong compile-time error checking, minimizing runtime bugs.
  - Good support for embedded development (e.g., esp-idf, embedded-hal, embassy).
  - Async programming model for efficient concurrency.
- Application in the system:
  - Rust ensures a safe, efficient, and robust firmware for ESP32, helping the embedded system run reliably over long periods without crashes or memory leaks.

2.2.2. MQTT Protocol

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol designed for IoT devices to communicate over low-bandwidth or unreliable networks.

- Key Features:
  - Based on the publish/subscribe model.
  - Low overhead, ideal for embedded systems.
  - Supports different Quality of Service (QoS) levels.
  - Simple implementation and wide community support.

- Application in the system:
  - Used for transmitting sensor data from ESP32 to a server or processing platform. Also supports remote device control by subscribing to command topics.

2.2.3. InfluxDB (Time-Series Database)

InfluxDB is a high-performance time-series database designed to store and query data that changes over time — such as temperature, humidity, and light levels in a greenhouse.

- Key Features:
  - Optimized for high write and query performance.
  - Supports powerful queries using Flux or InfluxQL.
  - Easily integrates with visualization tools and dashboards.
  - Open-source and scalable for large datasets.
- Application in the system:
  - Stores sensor data over time to monitor environmental trends. Enables analytics, threshold checking, and long-term data storage for further optimization of greenhouse conditions.

2.2.4. n8n (Workflow Automation Tool)

n8n is an open-source workflow automation platform that allows developers to create logic flows using a visual, node-based interface.

- Key Features:
  - Drag-and-drop workflow editor.
  - Supports integrations with MQTT, InfluxDB, email, Telegram, HTTP APIs, and more.
  - Can run self-hosted or in the cloud.
  - Extensible with JavaScript functions and custom nodes.
- Application in the system:

  Automates tasks such as:
  - Receiving MQTT data and storing it in InfluxDB.
  - Sending alerts via email or messaging apps when thresholds are exceeded.
  - Scheduling irrigation events or generating periodic reports.

2.3. Block Diagram

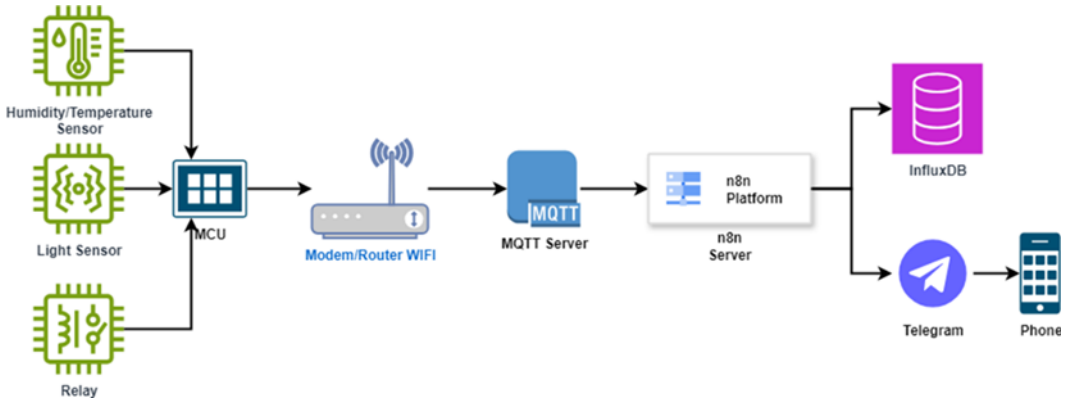


Figure 2.5. Block Diagram

The diagram above illustrates the overall architecture of the smart greenhouse monitoring system, which leverages IoT technologies to collect, transmit, store, and notify users of key environmental data in real time. The system is designed to be modular, scalable, and efficient, combining both embedded hardware and cloud-based services to automate data handling and provide meaningful insights.

1. Sensor Layer
   - At the foundation of the system are three core hardware components:
   - Humidity/Temperature Sensor: This sensor (e.g., DHT11 or DHT22) continuously monitors the ambient air temperature and relative humidity within the greenhouse. These parameters are critical for plant health and must remain within optimal ranges to prevent disease or growth delay.
   - Light Sensor: This component (e.g., a Light Dependent Resistor or photodiode) detects the level of light intensity inside the greenhouse. Light data is useful for analyzing plant photosynthesis conditions and determining the need for artificial lighting or shading systems.
   - Relay Module: The relay acts as a control interface for actuators. In future system upgrades, it may be used to automate equipment like irrigation pumps, fans, or lighting systems based on sensor readings.
   - All of these sensors are connected to a Microcontroller Unit (MCU), such as the ESP32, which serves as the brain of the embedded system.
2. Microcontroller Unit (MCU)
   - The MCU is responsible for reading sensor data through its GPIO pins and processing the input using custom firmware. After gathering and

formatting the environmental data, the MCU uses Wi-Fi to transmit the information wirelessly to the local network.

- The microcontroller is programmed to publish this data using the MQTT (Message Queuing Telemetry Transport) protocol, which is lightweight and ideal for IoT applications. The data is sent to the MQTT server hosted on the same or an external network.
- Communication Layer (MQTT Server + Wi-Fi Router)
- The Wi-Fi Router provides local wireless connectivity for the ESP32 and acts as the communication backbone. It facilitates the secure transmission of MQTT packets from the MCU to the MQTT Broker, which is typically hosted on a Raspberry Pi, local server, or cloud platform.
- The MQTT Server (broker) receives, buffers, and distributes sensor messages to subscribers. In this case, the subscriber is the n8n automation platform, which will further process the incoming data.

3. n8n Platform (Automation Server)
   - The n8n platform serves as an automation and data routing engine. Once MQTT data is received, n8n executes a predefined workflow that parses the sensor data, logs it to the database, and checks whether any alert conditions are met.
   - This low-code/no-code workflow platform allows for custom logic to be applied, such as:

     Checking if the humidity or temperature exceeds set thresholds.

     Triggering alerts via messaging platforms.

     Routing data to external systems like cloud dashboards or spreadsheets.
4. Data Storage: InfluxDB
   - InfluxDB, a high-performance time-series database, is used to store the incoming sensor data efficiently. It is optimized for fast writes and queries of time-stamped values, which makes it ideal for tracking greenhouse parameters over time.
   - This database can later be connected to data visualization tools (e.g., Grafana) to create interactive dashboards that display trends, anomalies, and system health metrics.
5. Notification System: Telegram

- In parallel with data storage, the n8n server can also trigger real-time notifications via Telegram. This ensures that farmers or greenhouse managers receive immediate alerts when environmental conditions become abnormal, allowing them to take prompt action.
- The alert message is pushed to the Telegram API, which then sends it to the user's mobile phone through the Telegram app. This provides an accessible and efficient way to stay informed remotely.

6. User Interface: Mobile Phone
   - The end user (farmer or operator) receives alerts via Telegram and can also access live or historical data through a web or mobile dashboard connected to the InfluxDB or MQTT stream. In future expansions, the mobile interface can also allow control commands (e.g., turning on/off irrigation manually).

2.4. Flow of Algorithm

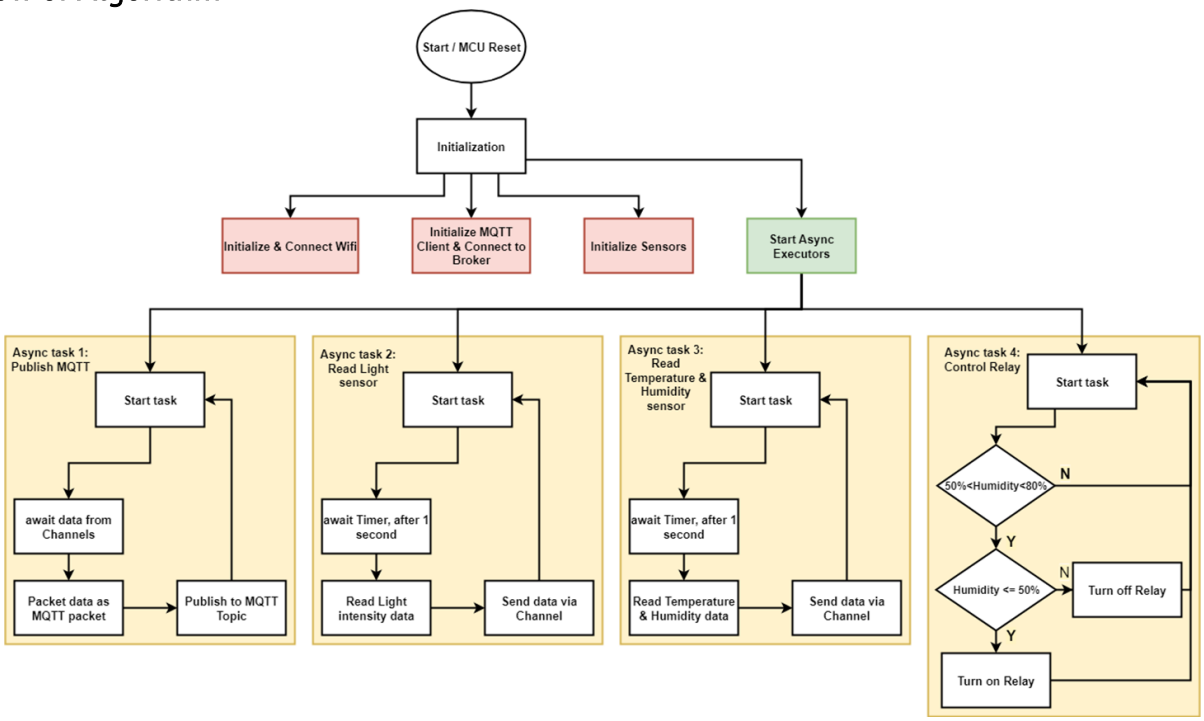


Figure 2.6. Flow of Algorith

The diagram above illustrates the internal control flow of the microcontroller unit (MCU), specifically how it asynchronously handles multiple operations: data acquisition from sensors, data publishing via MQTT, and relay-based automation. This multitasking design ensures non-blocking operation, real-time performance, and modularity in logic.

System Initialization Phase

Upon startup or manual reset, the MCU enters the Initialization phase, during which it performs the following key steps:

1. **Connect to Wi-Fi:**
   The device establishes a connection to the local Wi-Fi network, ensuring that the ESP32 has internet access or local LAN access to communicate with the MQTT broker.
2. **Initialize MQTT Client & Connect to Broker:**
   The MQTT client is configured with the broker address, topic names, and credentials (if needed). A successful connection allows the system to start publishing and receiving MQTT messages.
3. **Initialize Sensors:**
   All sensors, including the DHT11 (for temperature and humidity) and the light sensor (e.g., LDR), are configured with the correct GPIO pins and operating modes.
4. **Start Async Executors:**
   After all system components are initialized, the asynchronous task scheduler (e.g., using FreeRTOS or custom coroutine logic) is started. This allows multiple independent tasks to execute in parallel without blocking the main thread.

❖ Async Task 1: MQTT Publisher

This task is responsible for packaging and sending all environmental data to the MQTT topic.

- It continuously waits for sensor data that is sent via channels (e.g., queue, buffer).
- Once new data is received, it is structured into a JSON MQTT packet.
- The packet is then published to the configured MQTT topic at regular intervals.

This design ensures decoupling between data collection and data transmission, allowing each module to operate independently.

❖ Async Task 2: Light Sensor Reader

This task handles reading light intensity values from the light sensor.

- It operates on a timer-based loop, triggered every 1 second.
- After the wait, it reads analog or digital values from the sensor.

- The measured light intensity is then sent to a shared channel for use by other tasks (like the MQTT publisher).

❖ Async Task 3: Temperature and Humidity Reader

This task periodically collects data from the temperature and humidity sensor (DHT11).

- Like Task 2, it also runs on a 1-second interval timer.
- Upon triggering, it reads temperature and humidity values from the sensor.
- The data is forwarded to other modules via shared channels.

This asynchronous polling ensures consistent updates while avoiding blocking the MCU's main thread.

❖ Async Task 4: Relay Controller (Actuator Logic)

This task monitors environmental conditions and triggers actuator control based on humidity thresholds.

- It continuously listens for updated humidity values.
- The logic follows a two-step conditional check:
- If humidity > 50% and < 80%, the system checks further:
- If humidity ≤ 50%, it turns ON the relay (e.g., start irrigation).
- Otherwise, it turns OFF the relay (e.g., stop irrigation).
- The decision-making logic is kept simple for responsiveness and clarity.

This task is a prototype for future closed-loop automation, where actions are triggered in real time based on data trends.

2.5. Circuit connection diagram

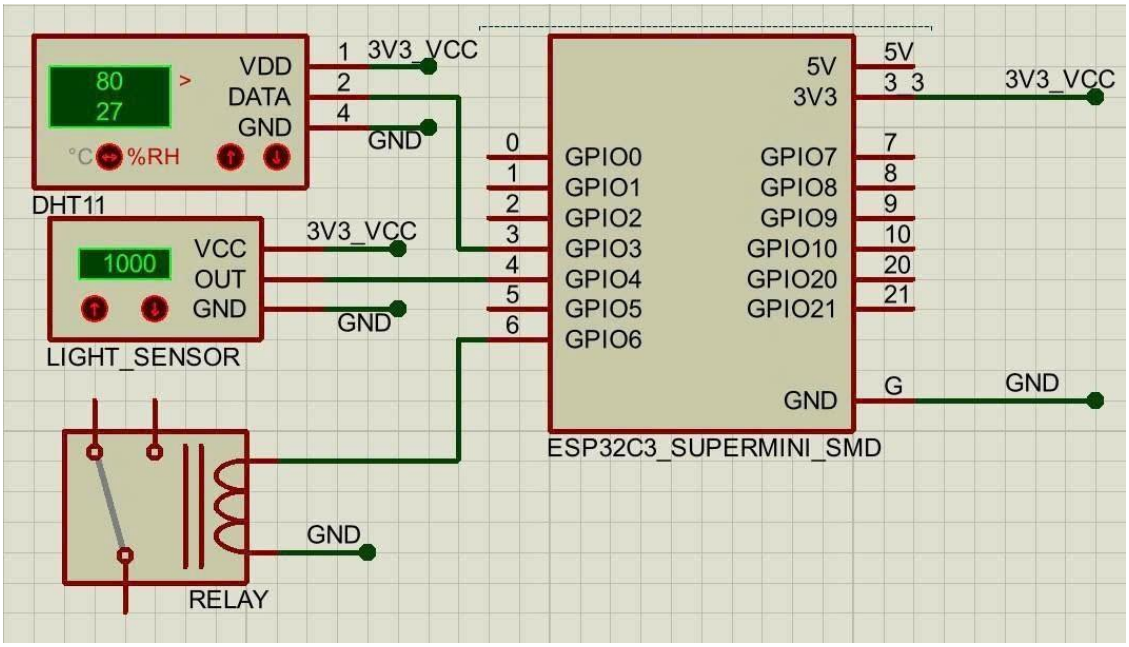


Figure 2.7. Circuit connection diagram

The schematic above shows the hardware wiring of the system using the **ESP32-C3 SuperMini** microcontroller. The main components are:

- **DHT11 Sensor:**
  Connected to **GPIO2**, powered by 3.3V. It measures temperature and humidity.
- **Light Sensor:**
  Connected to **GPIO4**, powered by 3.3V. It provides analog or digital light intensity readings.
- **Relay Module:**
  Connected to **GPIO5**, used to control external devices such as water pumps or fans based on environmental data.
- **Power Supply:**
  All components are powered using the **3.3V** and **GND** rails from the ESP32 board.

This setup ensures reliable data acquisition and allows for future expansion with minimal changes to the wiring.

3. Results

3.1. Data Acquisition (Sensor, Actuator, Controller)

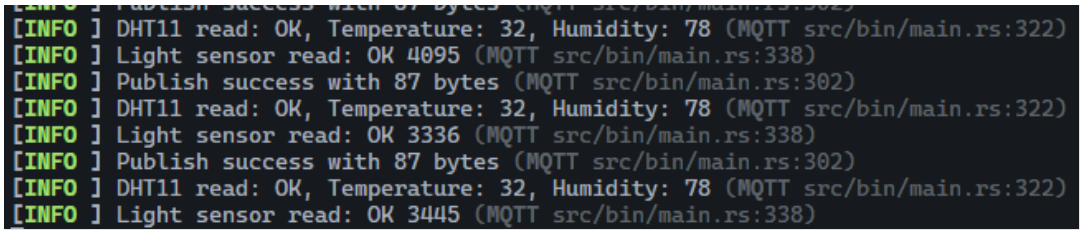


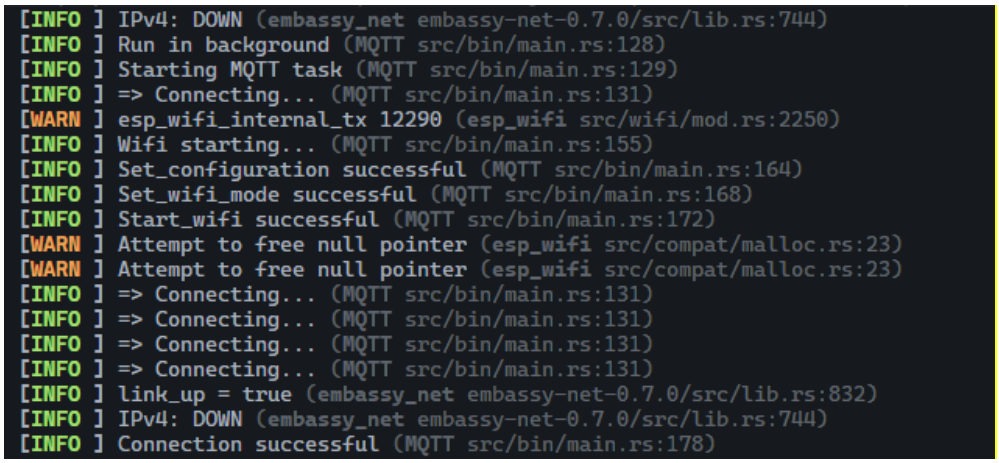
Figure 3.1. Data Acquisition

The image shows real-time console logs captured during the execution of the smart greenhouse monitoring system. These logs demonstrate successful readings and MQTT data transmission:

- **DHT11 Sensor Readings:**
  The temperature is consistently **32°C** and the humidity is **78%**. The logs confirm that the data was read successfully from the sensor.
- **Light Sensor Readings:**
  The light intensity values fluctuate (e.g., **4095**, **3336**, **3445**), indicating that the light sensor is actively measuring changes in lighting conditions.
- **MQTT Data Transmission:**
  Each successful data collection cycle ends with a message:
  "Publish success with 87 bytes",
  confirming that the environmental data is packaged and transmitted correctly via the MQTT protocol.

These logs verify that the sensor nodes, data formatting, and MQTT communication pipeline are working as expected in real time.

3.2. Network and Communication



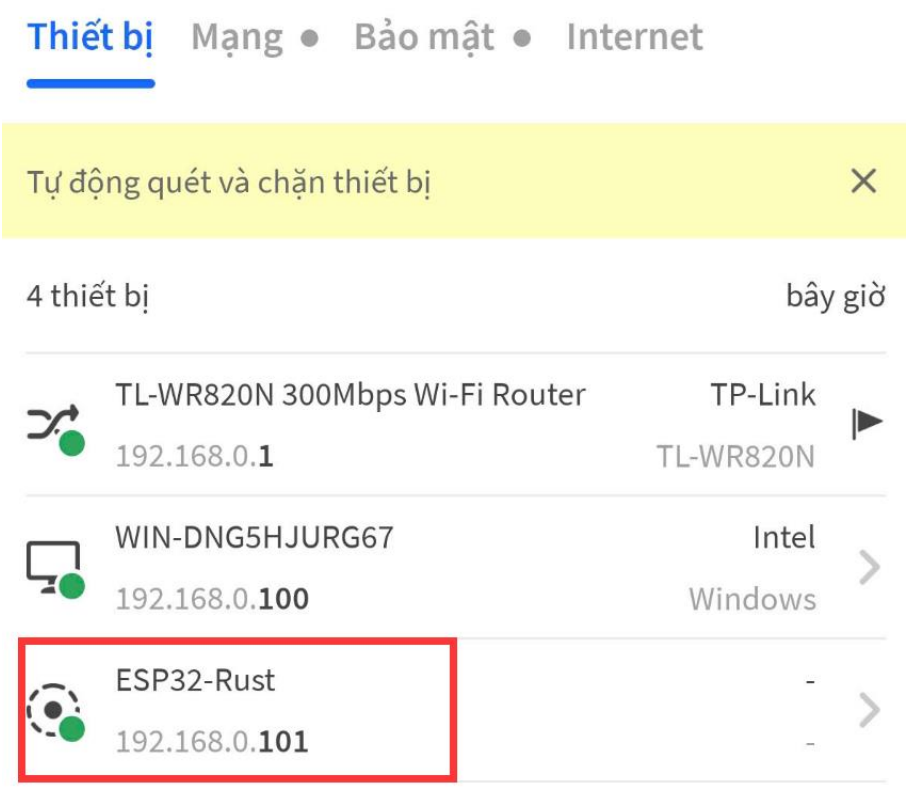


Figure 3.2. Network and Communication

The two figures illustrate the successful initialization and network connection of an ESP32 development board running firmware developed in Rust.

In the first figure, terminal logs indicate that the device successfully initiates Wi-Fi setup and connects to the MQTT broker. Key steps include configuration, Wi-Fi mode setup, and repeated connection attempts. A warning related to a null pointer is shown but does not interrupt the process. The final log line confirms that the MQTT connection is established successfully.

The second figure shows the ESP32 device ("ESP32-Rust") appearing in the router's connected device list with the assigned local IP address 192.168.0.101, confirming that the device has successfully joined the local network.

3.3. Hardware Implementation

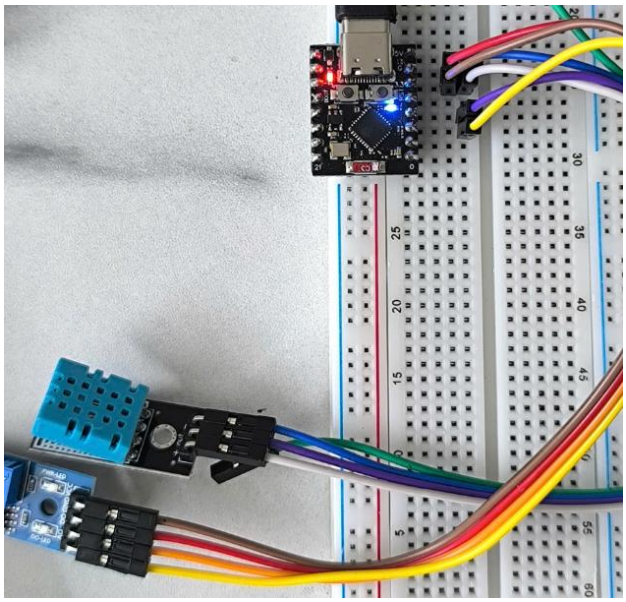


Figure 3.3. Hardware Implementation

3.4. Data Visualization

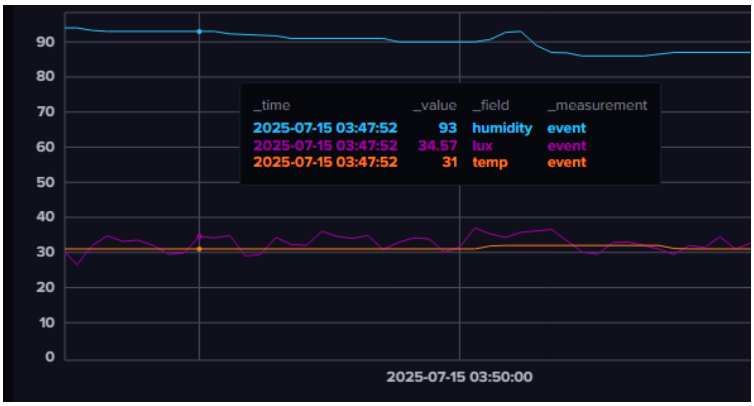


Figure 3.4. Data Visualization

Displays a time-series chart visualizing environmental sensor data collected by the ESP32-C3 system and stored in an InfluxDB database. The data is plotted using a dashboard (likely Grafana or similar), allowing real-time monitoring of three key parameters:

- Humidity (humidity):
  Represented in blue, with a value of 93% at timestamp 2025-07-15 03:47:52.
- Light Intensity (lux):
  Represented in purple, with a value of 34.57 lux, indicating ambient light levels.
- Temperature (temp):
  Represented in orange, with a value of 31°C at the same timestamp.

Each data point corresponds to a measurement recorded and transmitted by the ESP32 through MQTT, then written into the time-series database under the measurement name "event". This setup enables continuous logging and visualization of environmental changes, useful for applications such as smart agriculture, indoor climate control, or environmental monitoring.

3.5. Testing and Improvements

Figure 3.5. Testing and Improvements

The figure shows an automated Telegram message generated using n8n, a workflow automation tool. The message reports the average environmental conditions over the past hour, based on data collected and processed from sensors connected to the ESP32-C3 microcontroller.

The reported values include:

- Average Temperature: 31°C
- Average Humidity: 87%

This message was automatically triggered and formatted by an n8n workflow that retrieves data from a time-series database (e.g., InfluxDB), calculates hourly averages, and sends the results to a designated Telegram chat. This feature enables real-time environmental monitoring and alerting without user intervention, demonstrating the integration of IoT devices with messaging platforms for smart notification systems.

4. Projected Impact

4.1. Conclusion

In this project, we have successfully designed, implemented, and deployed a functional prototype of an IoT-based environmental monitoring system utilizing the ESP32-C3 SuperMini microcontroller. The system was built to measure key environmental parameters such as temperature and humidity using appropriate sensors (e.g., DHT11), and transmit this data in real-time to a cloud-based MQTT broker. From there, the data was forwarded to an InfluxDB time-series database, enabling structured storage and querying of historical sensor readings.

For visualization purposes, the collected data was displayed on dynamic dashboards, allowing users to observe trends and changes over time. Additionally, a notification system was implemented using the n8n automation platform, which sends alerts via Telegram when environmental conditions exceed preconfigured thresholds (e.g., high temperature or humidity levels). This enables timely responses and enhances user awareness.

Overall, the project demonstrates the feasibility, scalability, and practicality of using lightweight, low-cost IoT components in conjunction with open-source tools to achieve real-time environmental monitoring and intelligent alerting. The proposed architecture can be adapted for various smart agriculture, greenhouse automation, or environmental sensing applications where continuous, remote, and autonomous monitoring is required.

4.2. Future Improvements

To enhance the performance, usability, and intelligence of the system, the following future directions are proposed:

- Integration of Machine Learning (ML) or AI:

  Implement predictive models to forecast environmental trends (e.g., temperature rise or humidity drop), optimize irrigation schedules, detect anomalies in sensor data, and enable autonomous decision-making for

actuators like pumps or fans. This would transform the system from a reactive to a proactive model.

- Development of a User-Friendly Mobile or Web Application:

Building a dedicated cross-platform application would enhance the user experience by allowing real-time access to data, device status, and manual control. Features like data export, historical charting, and custom alert configurations could be included for advanced users.

- Expansion of Sensor Network:

Adding more sensor types such as soil moisture, light intensity (lux), CO_2 concentration, air quality (e.g., PM2.5), and rain detection would offer a more comprehensive understanding of the monitored environment, supporting more refined automation strategies.

- Edge Computing Capabilities:

Incorporating edge analytics directly on the ESP32 or via a local gateway (e.g., Raspberry Pi) would allow preliminary data processing, filtering, and decision-making to be done locally. This would reduce response latency, save bandwidth, and ensure system resilience during temporary internet outages.

- Energy Optimization and Sustainability:

For long-term deployments in remote or off-grid locations, the system could adopt solar panels, battery power, and use low-power modes of the ESP32 to minimize energy consumption. Smart scheduling of data transmission and sensor polling could further improve efficiency.

- Modular and Scalable Architecture:

Refactoring the current system into a more modular codebase would make it easier to add new sensors or outputs (e.g., relay controls). Implementing OTA (Over-The-Air) updates would allow firmware upgrades without physical access to the device.

- Security Enhancements:

As the system involves network communication, incorporating TLS encryption, authentication mechanisms, and secure OTA updates would help protect against unauthorized access or data breaches, especially when deployed in production.

5. Team Member Review and Comment

<ATTACH A TEAM PICTURE HERE>

NAME	REVIEW and COMMENT

6. Instructor Review and Comment

CATEGORY	SCORE	REVIEW and COMMENT
IDEA	__/10	
APPLICATION	__/30	
RESULT	__/30	

PROJECT MANAGEMENT	__/10	
PRESENTATION & REPORT	__/20	
TOTAL	__/100	