NEST: Next-generation Edge System for Tracking

Architecture and Service Platforms

Alejandro Botas Bárcena
Javier Grimaldos Chavarría
Estela Mora Barba
Group 2

MIoT 2025-2026

Contents

List of Figures

List of Tables

Acronyms

ADC Analog-to-Digital Converter

AI Artificial Intelligence

API Application Programming Interface

CI Continuous Integration

FSM Finite State Machine

GPIO General Purpose Input-Output

GPS Global Positioning System

HTML HyperText Markup Language

IDE Integrated Development Environment

IoT Internet of Things

JSON JavaScript Object Notation

LED Light Emitting Diode

LoRaWAN Long Range Wide Area Network

MQTT Message Queuing Telemetry Transport

NEST Next-generation Edge System for Tracking

PWM Pulse Width Modulation

REST REpresentational State Transfer

RFID Radio Frequency Identification

RGB Red, Green and Blue

SoC System on a Chip

SPI Serial Peripheral Interface

TBEL ThingsBoard Expression Language

TLS Transport Layer Security

UID Unique Identifier

UML Unified Modeling Language

1 Overview and Introduction

1.1 Document Overview

This document provides a comprehensive technical description of the Next-generation Edge System for Tracking (NEST) project, developed as part of the Master of Science in Internet of Things. It covers the complete development lifecycle, starting from the identification of stakeholder requirements and system design using Unified Modeling Language (UML) modeling to the final implementation and validation results.

The report is structured to detail the hardware and software architectures, specifically focusing on the data fusion processes at the Edge level and the integration with Internet of Things (IoT) service platforms like ThingsBoard. Additionally, it includes the work organization and effort estimation, providing a transparent view of the engineering processes applied to solve real-world challenges in smart farming.

1.2 Project Description

NEST is an intelligent IoT solution designed for the automated management of poultry farms, with a specialized focus on egg quality monitoring and protection. The system utilizes a distributed architecture where intelligence is not centralized in the cloud, but deployed directly in the nodes to ensure low-latency responses and local autonomy.

By measuring environmental variables such as temperature and humidity periodically, the system ensures optimal conditions for incubation. Furthermore, the system implements complex logic to detect the concurrent presence of hens and eggs. This allows for the automation of physical security through nesting box closures and restricted access control via Radio Frequency Identification (RFID), ensuring that production remains protected from external threats or predators until the farmer collects them.

The primary objective is to implement a fully functional IoT ecosystem satisfying these technical milestones:

- **Autonomous Edge Decision-making:** Implementation of local procedures to trigger actuators (e.g., closing doors) without requiring a constant connection to the server.
- **Knowledge Extraction:** Transforming raw telemetry data from load cells and sensors into high-level actionable insights regarding the safety status of the farm.
- **Hybrid World Integration:** Achieving a seamless parallel operation where real physical ESP32 nodes work together with simulated nodes to form a unified network.
- **Remote Commanding:** Enabling the server-side infrastructure to send specific tasks or parameter updates to the network elements via standard protocols.

1.3 Methodology

The development of NEST follows a **Waterfall software engineering approach**, ensuring a disciplined and sequential flow through the following stages:

1. **Requirements Analysis:** Identification of mandatory system constraints, including sensing types, actuation needs, and distributed intelligence requirements.
2. **System Design:** Translation of requirements into technical blueprints. This includes architectural design (subsystems) and detailed design using UML diagrams to model the interaction between the physical and simulated worlds.
3. **Implementation:** Development of the firmware for the Edge nodes and configuration of the IoT service platforms using Message Queuing Telemetry Transport (MQTT) and JavaScript Object Notation (JSON) for data exchange.
4. **Testing and Validation:** Systematic verification of the system logic through test benches. UML are used to model and validate the system's response to both periodic data flows and non-periodical alarm events.

2 Project Specifications

2.1 Hardware Requirements

The hardware infrastructure must support a hybrid environment where physical nodes and simulated entities interact seamlessly.

Table 1: Hardware System Requirements

Requirement	Description
Hybrid Deployment	Implementation must work with a subset of real physical elements in parallel with simulated parts.
Sensing and Actuation	The network must contain both sensors and actuators to interact with the environment.
Periodic Sensing	The system must perform periodic measurements of at least two different types.
Node Infrastructure	Network intelligence must be deployed directly in the nodes to support Edge computing.
Commanding	Network elements must be capable of receiving and executing tasks commanded from the server side.

2.2 Software Requirements

The software layer is defined by the engineering standards and communication protocols required for a level IoT application.

Table 2: Software Specifications

Category	Requirement / Specification
Development Process	Sequential Waterfall model: requirements analysis, architectural design, detailed design, implementation, and testing.
Standard Modeling	Full utilization of UML for all design and modeling phases.
Communication Stack	Support for standard transport protocols (primarily MQTT) and JSON data formats.
Visualization	Custom dashboards for historical trend analysis and real-time telemetry visualization.
Software Deliverables	Mandatory submission of source code and complete documentation.

2.3 System Functional Requirements

These requirements define the intelligence, data processing capabilities, and operational logic of the NEST ecosystem.

Table 3: System Intelligence and Operational Logic

Feature	Requirement Description
Distributed Intelligence	Autonomous decision-making must be processed locally at the Edge node.
Knowledge Extraction	Actionable knowledge must be extracted from raw information obtained within the IoT network.
Data Fusion	The network must perform information fusion and aggregation at the Edge level.
Event Handling	Verification of non-periodical alarm triggers and periodic measurement cycles through testing.
Remote Control	Capability to manually override status and set parameters via remote requests from the server side.

3 Proposal Description

3.1 System Overview

The NEST system is designed as an end-to-end IoT solution that bridges the gap between physical farm management and digital monitoring. It leverages Edge Computing to provide real-time responses while maintaining a robust cloud presence for data analytics and remote management.

3.1.1 System Architecture

The NEST system architecture is designed as a modular, end-to-end IoT ecosystem that facilitates hybrid deployment, integrating physical hardware with virtual simulation environments. As illustrated in Figure 1, the architecture is structured into three main layers: the NEST Devices, the Edge Platform, and the User Interaction Layer.

The foundation of the system consists of a hybrid network of nodes:

- **Physical NEST Device**: Based on an ESP32 microcontroller, it integrates sensors for temperature, humidity, and weight, alongside actuators such as an intelligent lock (RFID), Light Emitting Diode (LED) indicators, and automatic doors (Servomotors).
- **Simulation Environment**: A Python-based software layer that replicates the behavior of multiple NEST nodes, allowing for system scalability testing and validation without the need for additional physical hardware.

At the core of the system, the **ThingsBoard** platform acts as the primary service orchestrator. Communication is established via the **MQTT protocol over port 8883** for secure data exchange. The platform is responsible for:

- **Data Aggregation and Storage**: Collecting raw telemetry in a dedicated database for historical analysis.
- **Rule Engine**: Processing incoming data to extract high-level knowledge and trigger automated actions or alarms based on predefined thresholds.
- **Digital Twin Management**: Handling **Shared Attributes** to synchronize the physical state of the nodes with their cloud representation, enabling remote commanding and parameter updates.

The final layer provides the farmer with actionable insights through two main channels:

- **Interactive Dashboard**: A multi-state web interface for real-time monitoring of environmental conditions and production status.
- **Telegram Integration**: An external notification service that sends critical alarms directly to the user's mobile device via REpresentational State Transfer (REST) Application Programming Interface (API), ensuring immediate awareness of any system anomalies.

The architecture is designed to transform raw sensor data into actionable knowledge. This process starts at the **NEST**, where the ESP32 performs sensors measurements. The integration between the physical world and the digital twin is mediated by a robust **Communication Stack**, sending those measurements to the **Edge platform** for processing.

The use of **MQTT** as the primary transport protocol allows the system to maintain a persistent, bidirectional connection with the ThingsBoard broker. This hybrid approach not only supports the current physical setup but also provides a scalable framework where hundreds of simulated nodes can be provisioned through the `run_all.bat` workflow, allowing for stress-testing of the platform's ingestion capabilities.

Beyond telemetry collection, the architecture emphasizes **Remote Commanding**. Through the use of **Shared Attributes**, the **Edge platform** acts as a centralized brain that can override local states or update operational parameters (such as the `telemetry_interval`) across the entire network. This bidirectional flow ensures that the farmer maintains full control through the interactive dashboard, while the **Rule Engine** automates complex scenarios, such as notifying the user via Telegram when specific production thresholds are reached.

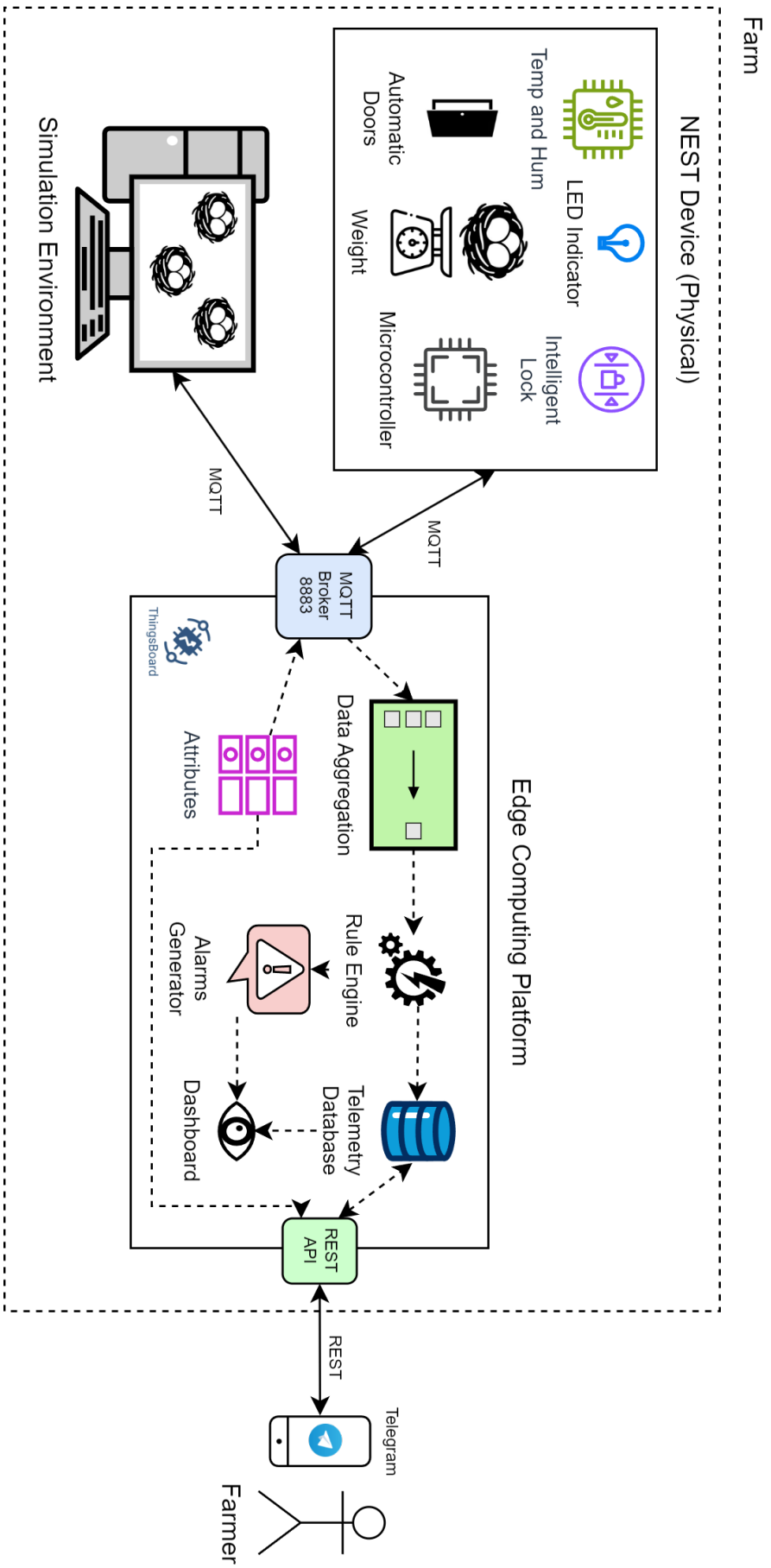

Figure 1: NEST System Architecture

3.1.2 Use Case Diagram

The primary interactions with the system involve the authorized personnel (farmer) and the biological entities (hens), as depicted in Figure 2.

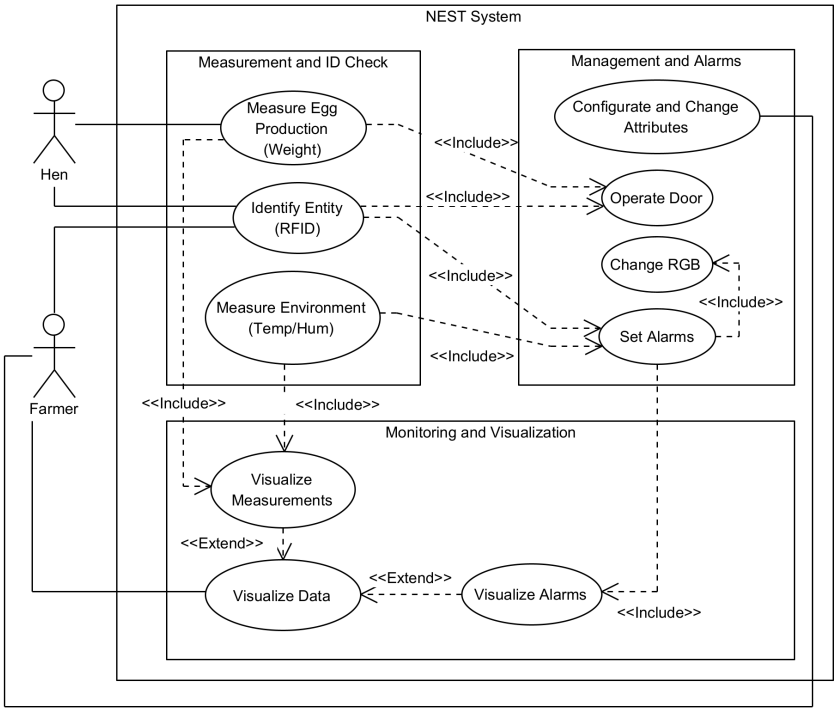

Figure 2: NEST System Use Case Diagram

3.1.3 Component Diagram

The architecture follows a modular approach where the NEST device acts as an intelligent gateway, and ThingsBoard serves as the integration broker.

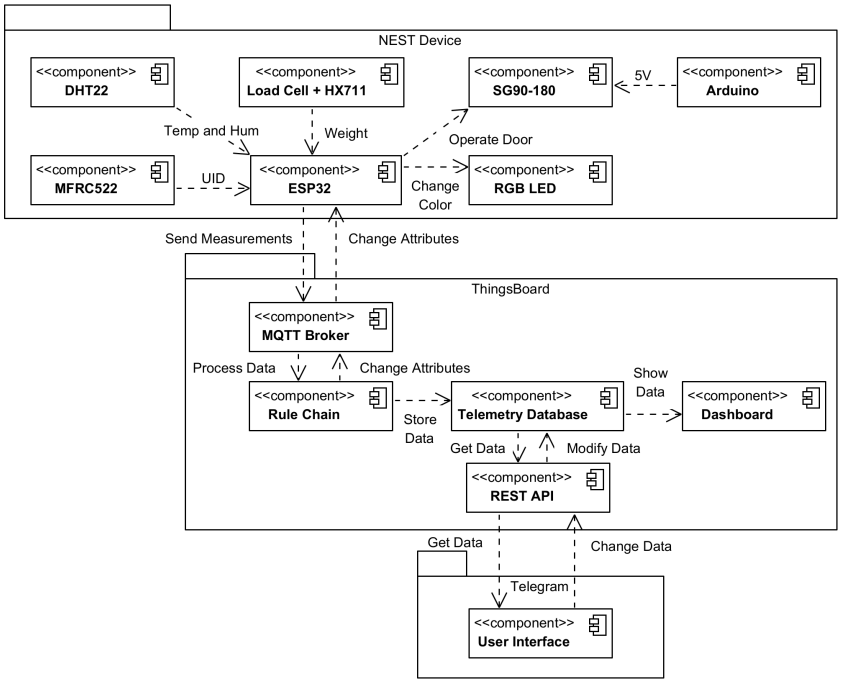

Figure 3: Component Diagram of the NEST Ecosystem

3.1.4 Deployment Diagram

The physical deployment illustrates the hybrid nature of the project, including the physical ESP32 node and the simulated Python environments.

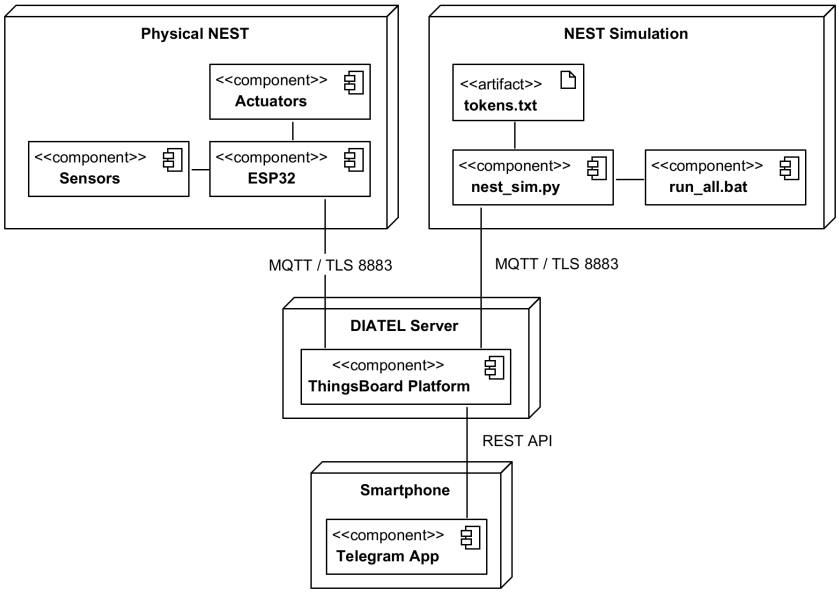

Figure 4: Deployment Diagram of the NEST System

3.2 System Workflow

The NEST system operates through a highly coordinated exchange of messages between the physical environment and the digital twin. This process ensures that every biological event—such as a hen entering the nest or an egg being laid—is recorded, analyzed, and acted upon in real-time.

As detailed in Figure 5, the workflow begins at the **NEST Device**. The ESP32 manages two concurrent execution threads using **FreeRTOS**. The `taskRFID` maintains a high sampling rate (250ms) to ensure an "Instant Publish" event as soon as a tag is detected.

Simultaneously, the `taskTelemetry` performs **Data Fusion** by gathering environmental readings and weight data, encapsulating them into a structured JSON payload every period (10s by default). To prevent hardware conflicts during these overlapping operations, **Semaphores** act as traffic controllers for the Serial Peripheral Interface (SPI) bus and the MQTT client.

Once the telemetry reaches the **ThingsBoard** platform, the workflow transitions to the **Rule Engine** logic depicted in Figure 6. The platform does not merely store data, it evaluates it against business logic:

- **Identity Validation**: The system distinguishes between a hen (triggering monitoring mode) and a staff member (triggering collection mode).
- **Autonomous Actuation**: If the sensors detect a weight increase consistent with an egg but the Unique Identifier (UID) indicates the hen has left, the platform pushes a **Shared Attribute** update.
- **Remote Sync**: This attribute update is received by the ESP32's `mqttCallback`, which immediately triggers the servos to close the NEST door, protecting the production until a person identifies themselves to collect it.

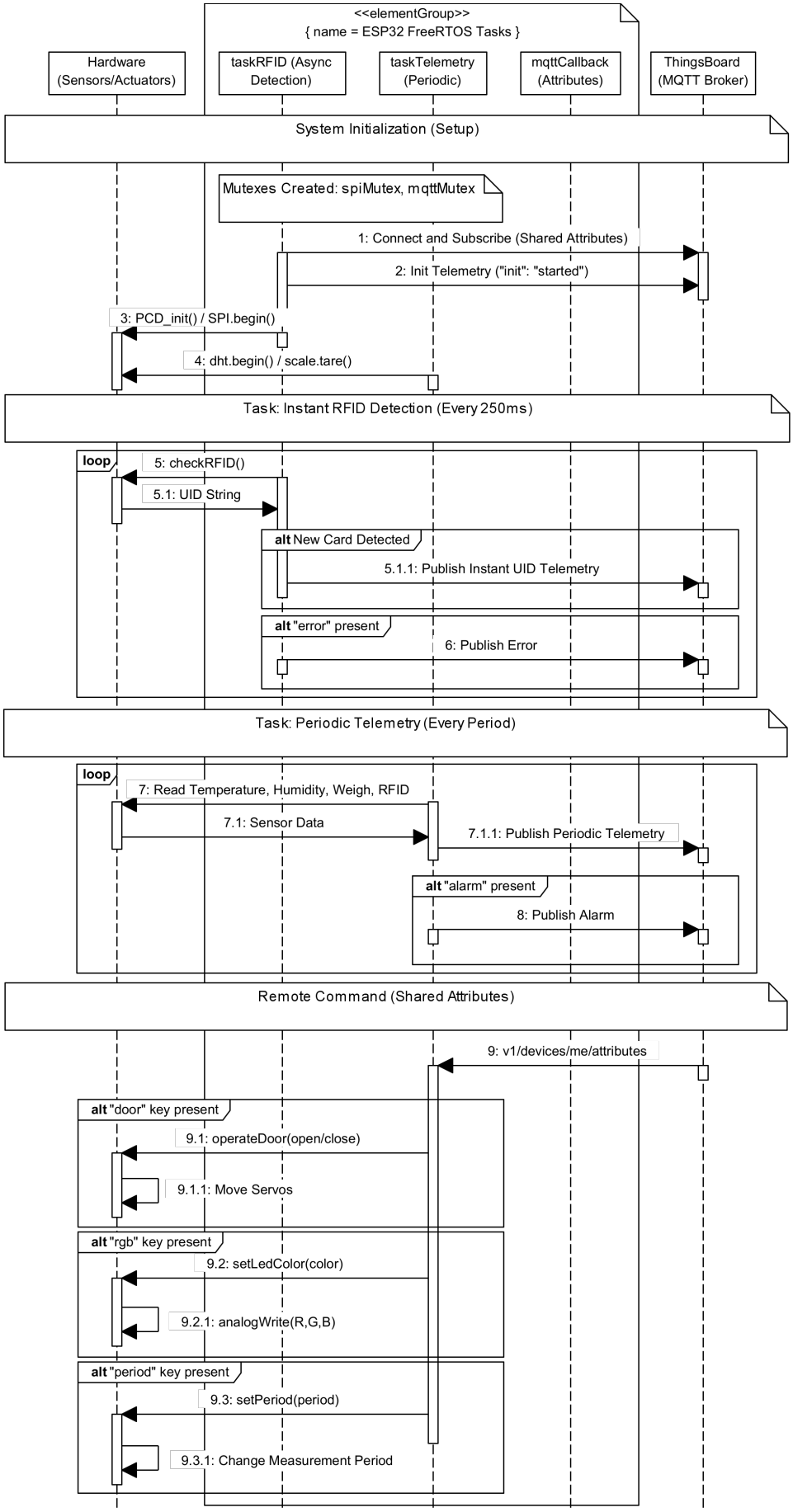

Figure 5: NEST Device Workflow Sequence Diagram

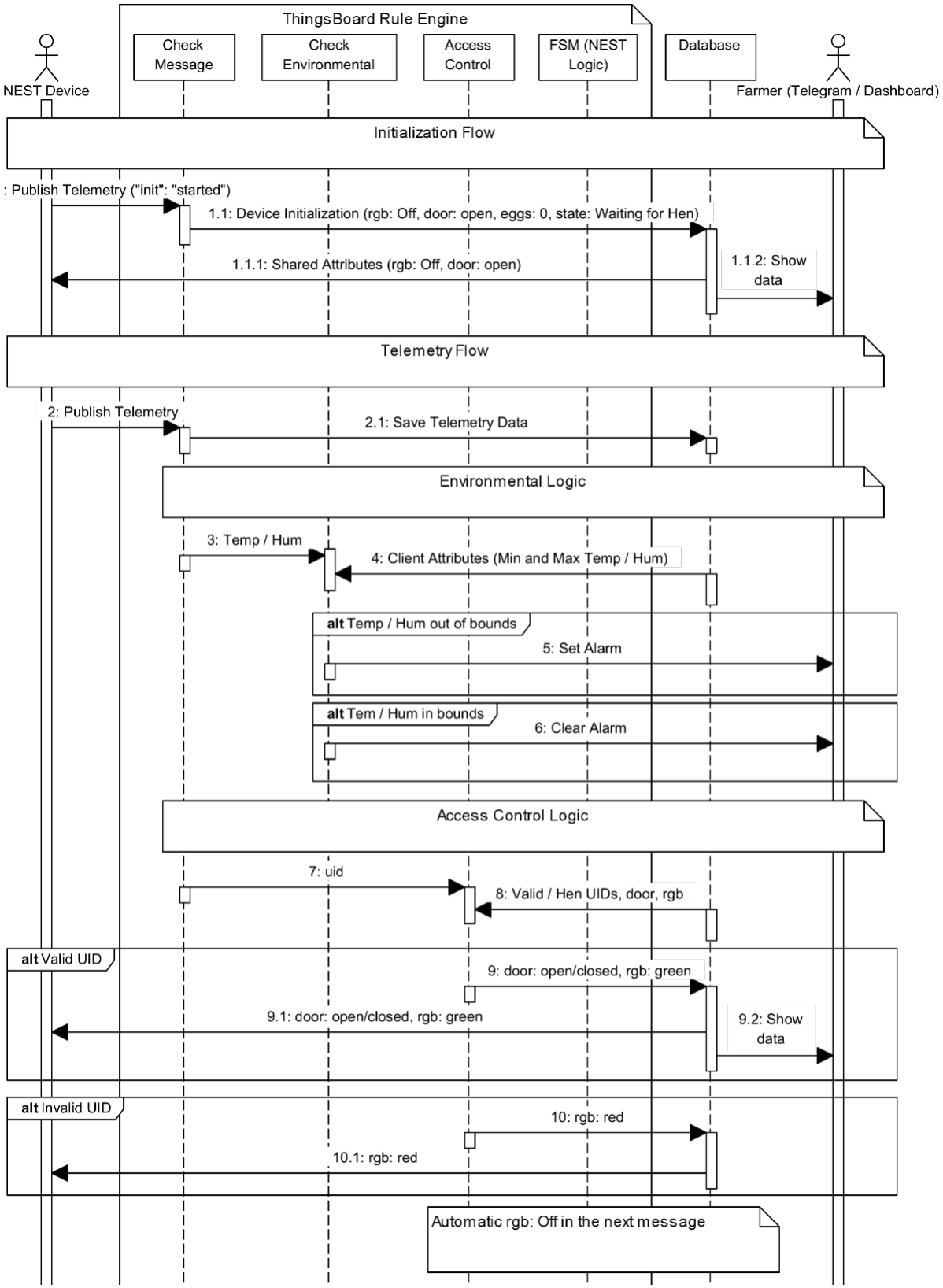

Figure 6: ThingsBoard Workflow Sequence Diagram (1)

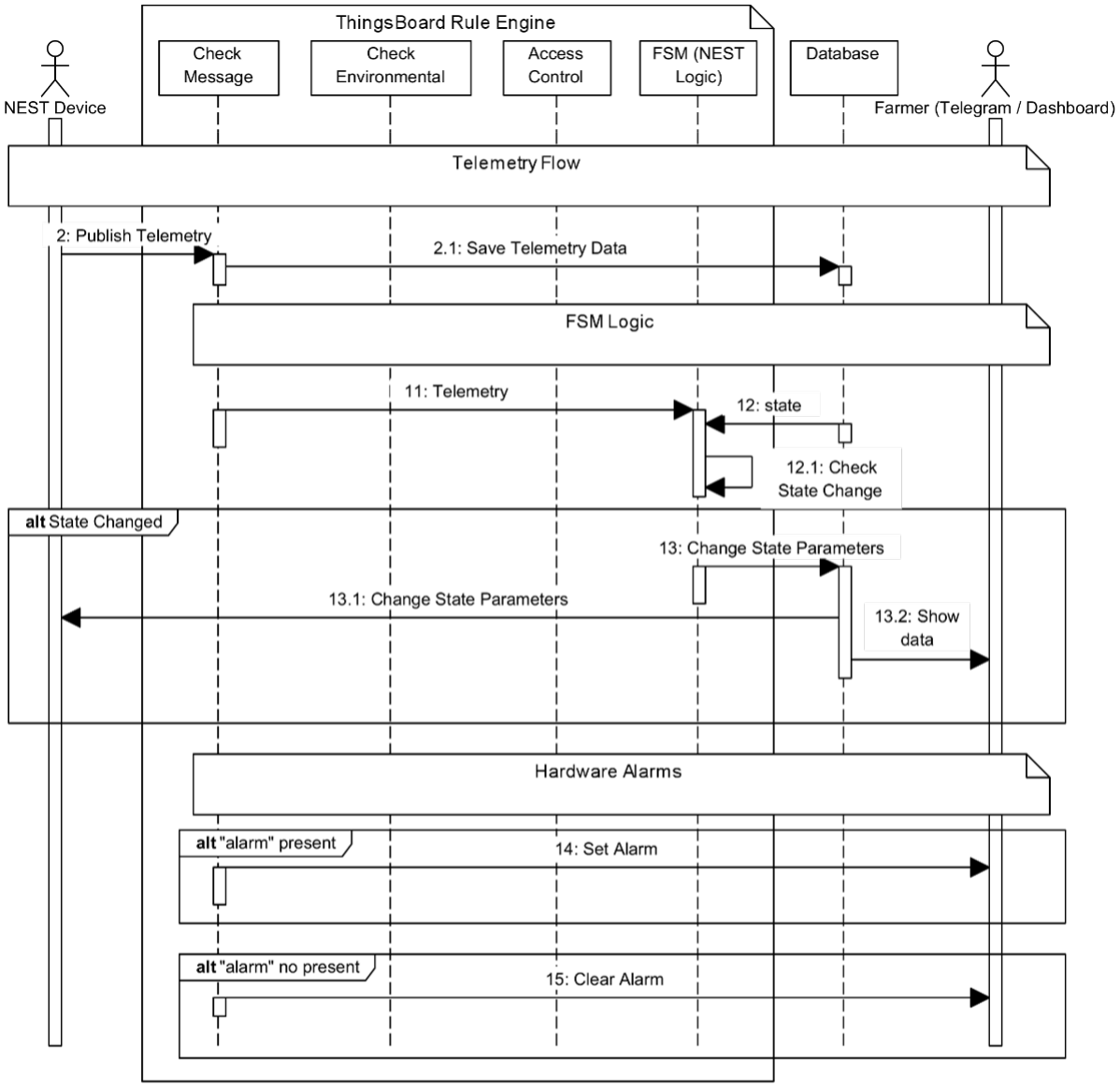

Figure 7: ThingsBoard Workflow Sequence Diagram (2)

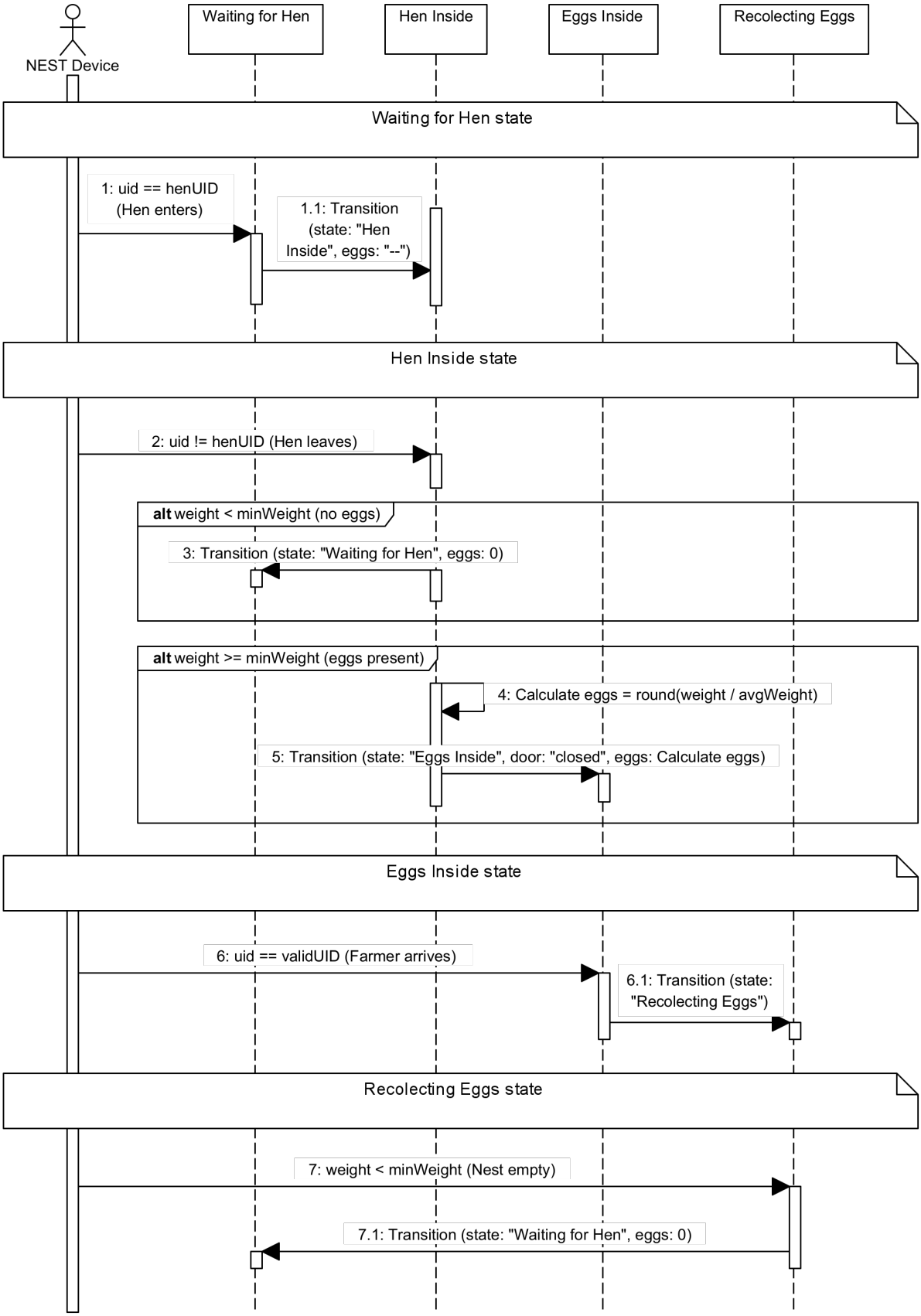

Figure 8: FSM Workflow Sequence Diagram

3.3 Hardware Device

3.3.1 Block diagram

The block diagram shown in Figure 9 provides an overview of the complete NEST hardware architecture. It illustrates how the ESP32 microcontroller acts as the central hub, interacting with the sensing layer (DHT22, Load Cell, and RFID) and the actuation layer (Servos and Red, Green and Blue (RGB) LED). Each peripheral is connected through specialized interfaces, allowing the system to process data at the edge and perform autonomous egg protection tasks.

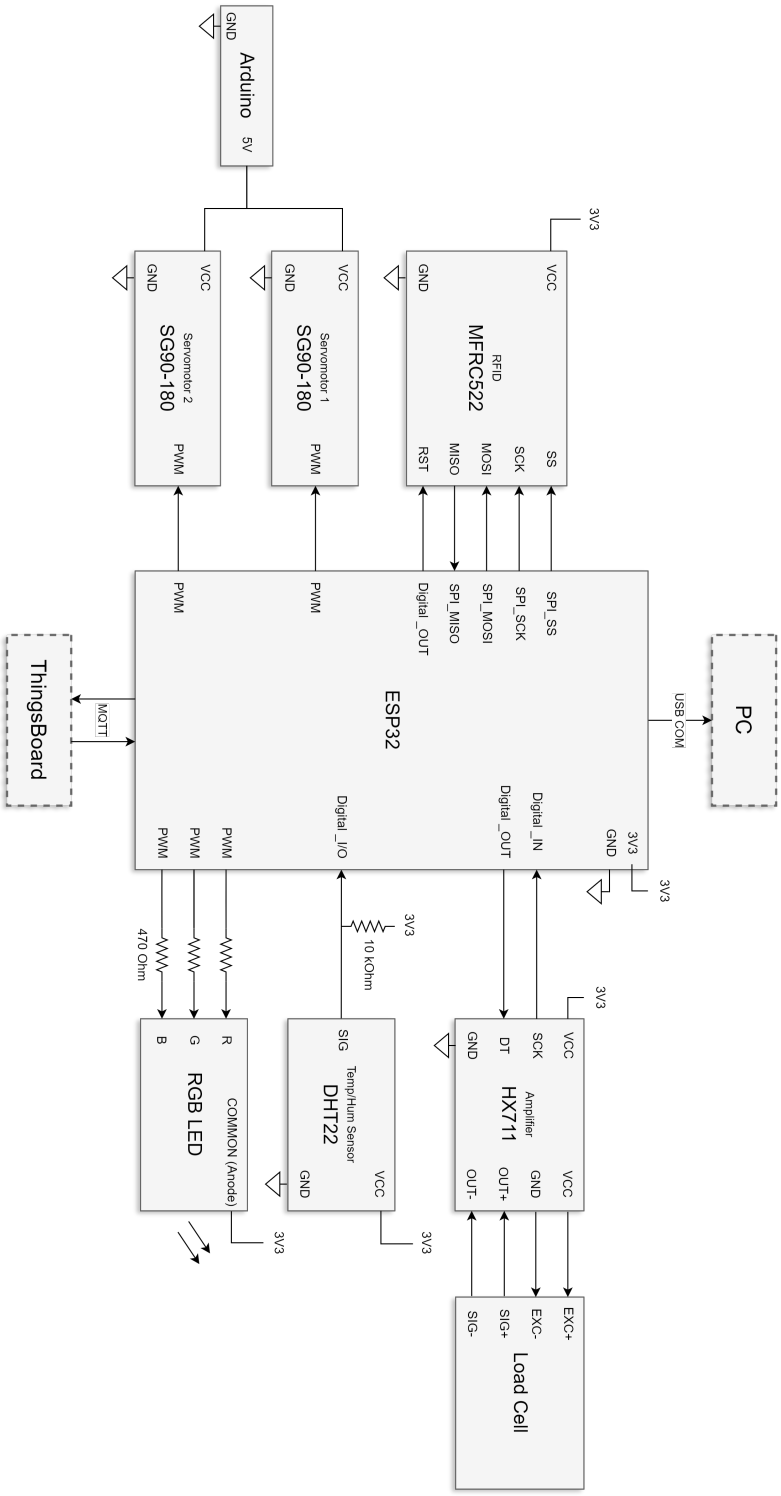

Figure 9: Block Diagram of the Hardware System

The ESP32 microcontroller manages the diverse hardware modules. A digital interface is used to gather periodic temperature and humidity data from the DHT22. The SPI bus is dedicated to the MFRC522 module for high-speed RFID communication to identify hens and farmers. Weight data is acquired through a serial interface with the HX711 amplifier, while Pulse Width Modulation (PWM) channels drive the SG90 servomotors for precise door positioning.

In addition, several General Purpose Input-Output (GPIO) pins drive the RGB LED to provide immediate visual feedback on the system's state. The hardware configuration utilizes the ESP32's 3.3V and Arduino's 5V power supplies. Finally, all the data collected from the sensors is sent to the ThingsBoard server via MQTT over a Wi-Fi connection, enabling remote monitoring and processing.

3.3.2 Hardware Components

- **ESP32 Microcontroller[1]**: The ESP32 is a high-performance System on a Chip (SoC) featuring dual-core processors and integrated Wi-Fi connectivity. In the NEST project, it is responsible for data acquisition from sensors, actuator Control and communication via MQTT.
- **5 kg Weight Scale[2] and HX711 Amplifier[3]**: For egg detection, a Load Cell is integrated with an HX711 24-bit Analog-to-Digital Converter (ADC) amplifier. This allows the system to detect the presence and quantity of eggs by weight.

  Physically, the load cell operates as a transducer that converts mechanical force into an electrical signal. It contains an internal **Wheatstone bridge** of strain gauges: When a weight is placed on the platform, the resistance of these gauges changes, producing a differential voltage in the millivolt range.

  Because this signal is too small for the ESP32 to process directly, the **HX711 amplifier** is used to amplify the signal and convert it into a 24-bit digital value. The HX711 connects to the ESP32 through GPIO 16 (DOUT) and GPIO 4 (SCK) using a two-wire serial interface.

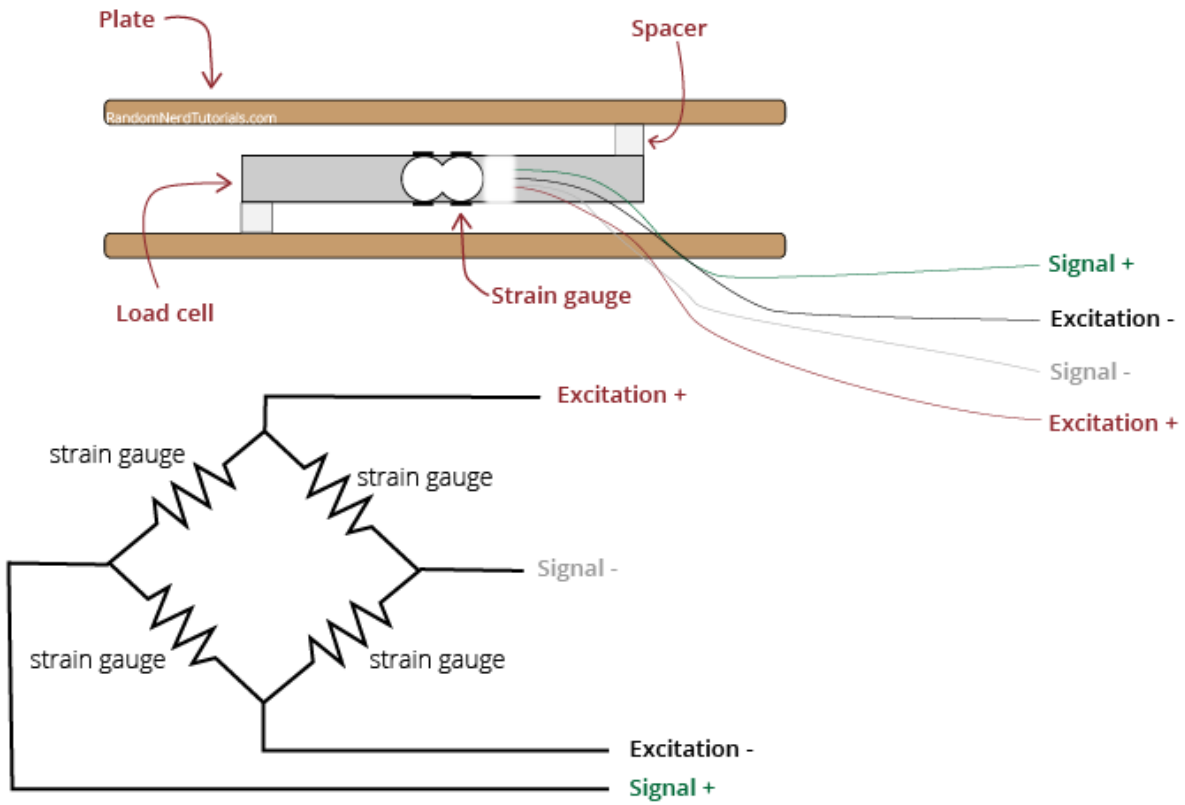

Figure 10: Load Cell Diagram[4]

- **DHT22 Temperature and Humidity Sensor[5]**: To monitor the incubation environment, the system integrates a DHT22 digital sensor. This sensor provides periodic measurements of ambient conditions, which are critical for egg quality monitoring. It communicates with the ESP32 via GPIO 15 using a digital single-wire protocol.

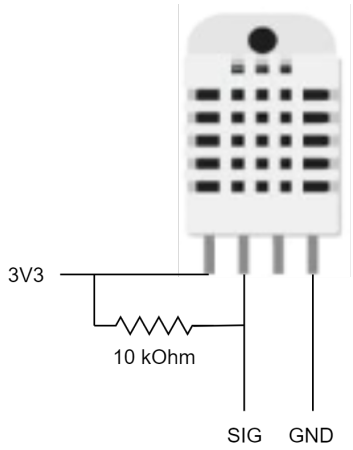

Figure 11: DHT22 Circuit

- **MFRC522 RFID Module[6]**: To handle hen detection, access control and identification, the system integrates an MFRC522 RFID reader. It communicates with the ESP32 via the SPI protocol using GPIO pins 18 (SCK), 19 (MISO), 23 (MOSI), and 5 (SS).
- **SG90-180 Servo Motors[7]**: The physical response of the system is managed by two SG90-180 micro servo motors acting as actuators. These motors are responsible for open and close the doors of the nest. The ESP32 controls these servos using PWM signals through GPIO 2 and GPIO 13. The use of actuators that can be triggered both by edge decisions and by remote commands.
- **RGB LED and 470 Ohm Resistors**: The system includes a common-anode RGB LED to provide local status indication. This LED is controlled via GPIO 25, 26, and 27. Each channel is connected in series with a 470 Ohm resistor to protect the ESP32 pins from overcurrent.

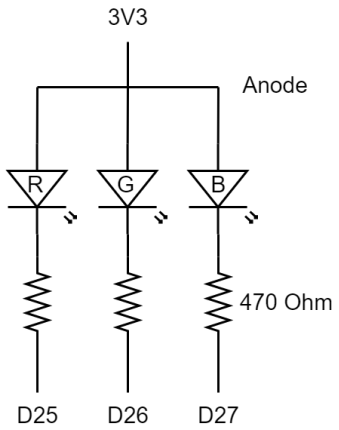

Figure 12: RGB LED Circuit

- **Arduino Microcontroller (Power Supply)**: Although the ESP32 acts as the central intelligence unit, an Arduino board has been integrated into the hardware architecture to function as a specialized power distribution bridge. The SG90 servo motors require a stable 5V supply to operate correctly under load. Since the ESP32 development board typically operates at 3.3V and its output pins cannot provide the necessary current or voltage for the actuators, the Arduino is utilized to leverage its 5V power rail.

3.3.3 Interfaces of the system

To ensure the reliability and modularity of the NEST platform, the hardware architecture is organized into distinct electrical domains. The following tables detail the pin-mapping for logic signals, the power distribution strategy, and the specific wiring for high-precision sensing.

The integration utilizes diverse communication protocols, including **Single-Bus** for environmental data, **SPI** for high-speed RFID identification, and **PWM** for precise mechanical control of the actuators.

Table 4: Main System Connections (ESP32)

Module	Pin Name	ESP32 Pin	Function/Protocol
DHT22 Temp/Hum	VCC	3.3V	Power Supply
	Data	GPIO 15	Digital Single-Bus
	GND	GND	Ground Reference
HX711 Amplifier	VCC	3.3V	Power Supply
	DOUT	GPIO 16	Serial Data Out
	SCK	GPIO 4	Serial Clock
	GND	GND	Ground Reference
MFRC522 RFID	VCC	3.3V	Power Supply
	SCK	GPIO 18	SPI Clock
	MISO	GPIO 19	SPI Master In
	MOSI	GPIO 23	SPI Master Out
	SS	GPIO 5	SPI Slave Select
	RST	GPIO 22	Reset
	GND	GND	Ground Reference
	IRQ	–	–
SG90-180 Servos	Servo 1	GPIO 2	PWM
	Servo 2	GPIO 13	PWM
	GND	GND	Ground Reference
LED RGB (Anode)	Common	3.3V	Power Supply
	R + 470 Ohm	GPIO 25	Red
	G + 470 Ohm	GPIO 26	Green
	B + 470 Ohm	GPIO 27	Blue

A critical aspect of the design is the **Power Decoupling** strategy. To prevent electrical noise and voltage drops caused by the SG90 servomotors from affecting the ESP32's logic, an Arduino board is used as a dedicated 5V power rail, sharing a common ground to maintain signal integrity.

Table 5: Arduino Connections

Device	Connection	Arduino Pin	Description
SG90 Servos	VCC	5V	High Current Power Supply
ESP32	GND	GND	Ground Reference Unification

For the production monitoring subsystem, the load cell is wired in a **Wheatstone Bridge** configuration. This setup allows the HX711 to detect minute changes in resistance and amplify them into 24-bit digital values for the ESP32.

Table 6: Load Cell Wiring (Wheatstone Bridge)

Wire Color	Signal Type	HX711 Pin	Description
Red	Excitation+	E+	Positive Voltage Supply
Black	Excitation-	E-	Negative Voltage Supply
Green	Signal+	A+	Differential Output Positive
White	Signal-	A-	Differential Output Negative

3.3.4 Final Prototype

The final prototype integrates all previous modules into a compact hardware unit, as shown in Figure 13. This assembly was designed to withstand the environmental conditions of a farm while maintaining high precision in identifying individual hens and measuring egg production.

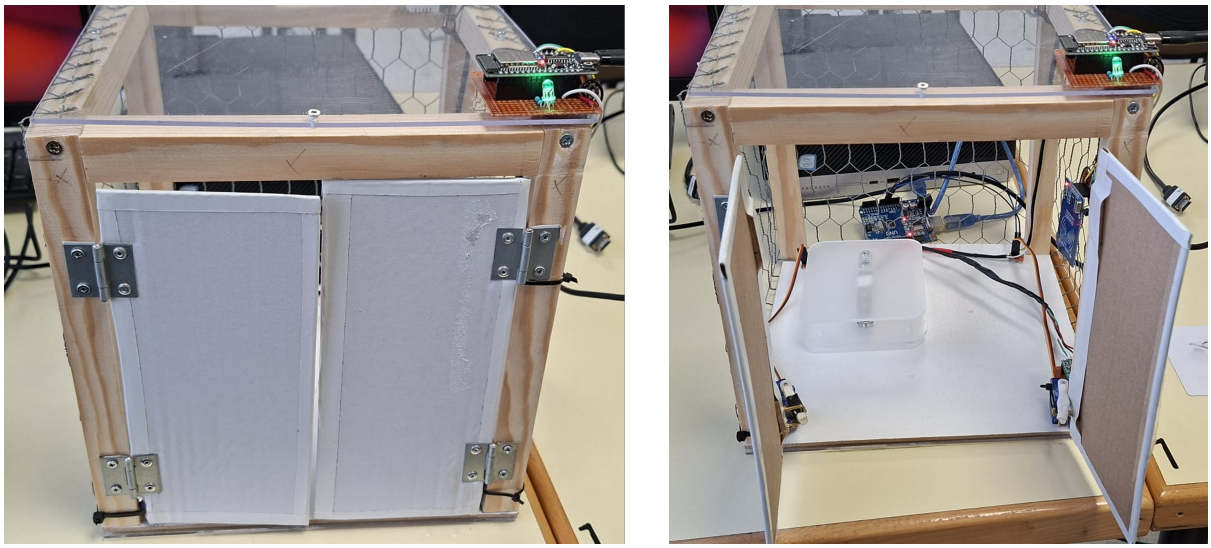

Figure 13: Final Prototype

Table 7: Hardware Specifications and Components

Component	**Function**
ESP32	Data acquisition, actuators control, and communication (MQTT).
DHT22	Periodic temperature and humidity sensing
Load Cell + HX711	Egg detection via weight measurement
MFRC522 (RFID)	Hens location, access control and identification
SG90-180 Servo	Actuator for automated nesting box closure
RGB LED	Local visual status notification and actuator feedback
Arduino	Secondary power distribution (5V)

Table 8 is a summary of the costs associated with each hardware component used in the NEST system.

Table 8: Hardware Budget and Costs

Component	**Price (€)**	**Amount**	**Final Price (€)**
ESP32	4.39	1	4.39
Load Cell + HX711 + Platform	4.42	1	4.42
DHT22	1.48	1	1.48
MFRC522 (RFID)	1.27	1	1.27
SG90-180 Servo	1.24	2	2.48
RGB LED	0.07	1	0.07
470 Ohm Resistors	0.01	3	0.03
Arduino	–	1	–
Electrical Materials	10	–	10
Box Materials	20	–	20
Total Price			44.14

3.4 Implementation and Development

This section details the technical realization of the NEST ecosystem, integrating the firmware development, the simulation environment, and the service platform configuration. The implementation has been designed following a hybrid deployment approach, where physical and virtual devices coexist to ensure both the scalability and robustness of the poultry monitoring system.

To achieve a seamless integration, communication protocols and data structures have been unified, ensuring that both real hardware and simulated nodes interact identically with the central server. The following sections describe the technological pillars of this phase: the real-time system-based firmware for the physical device, the cyclic simulation engine in Python, and the data management architecture within the ThingsBoard platform.

3.4.1 NEST Device

The firmware for the NEST device has been developed using the **Arduino Integrated Development Environment (IDE)** framework. The software architecture leverages the **FreeRTOS** kernel integrated into the ESP32 core to implement a multitasking environment that ensures the system meets the real-time requirements of a smart farm.

To prevent race conditions when multiple tasks access the same hardware resources, the system implements **Semaphores (Mutexes)**. This is particularly critical for the SPI bus, which is shared by the RFID reader, and the MQTT client used by both telemetry and attributes changes.

The system executes two main concurrent tasks to ensure efficient resource management:

- **Telemetry Task:** Handles the periodic measurement of temperature, humidity, weight, and UID. It transmits this data to the ThingsBoard platform every 10 seconds (configurable via remote attribute).
- **RFID Task:** Operates asynchronously with a high sampling rate (250ms) to provide instant detection. This task allows for "Instant Publish" events, sending the identified farmer's UID immediately upon detection without waiting for the telemetry cycle to be able to open the door.

The device implements a secure communication stack using **MQTT over Transport Layer Security (TLS) (Port 8883)** to interact with the ThingsBoard platform. This connection supports bidirectional traffic through a subscription to **Shared Attributes**, enabling "Commanding" from the server side as specified in the project requirements.

The firmware utilizes the `mqttCallback` function to receive attributes updates. The integration of the **ArduinoJson** library is fundamental to this process, as it facilitates the parsing of complex JSON payloads received from the server and the construction of structured telemetry messages for transmission.

The development of the NEST device firmware relies on several specialized libraries to interface with the hardware components and manage data serialization.

Table 9: External Libraries and Authorship

Library	Function	Author/Maintainer
DHT Sensor Library	Temp and Hum monitoring	Adafruit
HX711 Arduino Library	Load cell data acquisition	Bogdan Necula, Andreas Molt
MFRC522	RFID reader interfacing	Github Community
ESP32Servo	PWM control for SG90 motors	Kevin Harrington, John K. Bennett
PubSubClient.h	MQTT protocol implementation	Nick O'Leary
WiFi.h and WiFiClientSecure.h	Wireless connectivity and TLS encryption	Espressif Systems
SPI.h	SPI Bus management	Arduino / Espressif
ArduinoJson	JSON serialization for telemetry	Benoit Blanchon

3.4.2 NEST Simulation

To fulfill the requirement of a hybrid deployment where real and simulated worlds work together, a Python-based simulation environment has been developed. This tool allows the execution of multiple virtual nodes in parallel with the physical hardware, enabling scalability testing and verification of the system's logic across a larger network.

The simulation is built using the `paho-mqtt` library, which establishes a secure connection to the ThingsBoard platform via **MQTT over TLS (Port 8883)**. Each simulated instance runs a Finite State Machine (FSM) that replicates the life cycle of a nesting box and all its funtionalities. The defined states are:

- `WAITING_FOR_HEN`: The default idle state with zero weight and no identified UID.
- `HEN_INSIDE`: Simulates a hen's presence by generating a random weight (2000g-3500g) and assigning a specific UID.
- `EGGS_DEPOSITED`: Represents the state where the hen has left (UID: None), but the weight of the deposited eggs remains.
- `PERSON_COLLECTING`: Simulates an authorized user collecting the eggs, identified by a unique UID.

To evaluate the platform's performance with multiple devices, the system includes an automated deployment workflow. This consists of a `tokens.txt` provisioning file and a batch script (`run_all.bat`) that launches multiple simulation instances simultaneously, each with its unique credentials and virtual identity.

Table 10: Simulation Environment Components

Component	Function
`nest_sim.py`	Core script managing FSM logic, sensors simulation, and secure MQTT connectivity.
`tokens.txt`	Provisioning list linking ThingsBoard devices access tokens with device names.
`run_all.bat`	Automation script for parallel execution of multiple virtual nodes.

3.4.3 ThingsBoard

The centralized management of the NEST ecosystem is performed through the ThingsBoard IoT platform. This service acts as the core for data ingestion, device management, data visualization, and remote commanding.

Device Management

To ensure a standardized configuration across the network, a specific **Device Profile** named `NEST Device` has been created. This profile defines the common behavior, transport configurations, and alarm rules for all nodes in the coop. Currently, four operational devices have been provisioned under this profile:

- **NEST 1:** The primary physical node based on the ESP32 hardware.
- **NEST 2, NEST 3, and NEST 4:** Simulated nodes running the Python-based FSM to test hybrid deployment and scalability.

The platform utilizes **Shared Attributes** to enable remote commanding. These attributes are stored on the server and synchronized with the devices via MQTT. As shown in Figure 14, the following key attributes are managed:

- `door`: Controls the physical state of the nesting box (`open` or `closed`).
- `rgb`: Sets the color of the RGB LED (`Red, Green, Blue`).
- `period`: Defines the telemetry transmission interval in milliseconds.
- `state`: Current state of the FSM.
- `eggs`: Estimated amount of eggs.

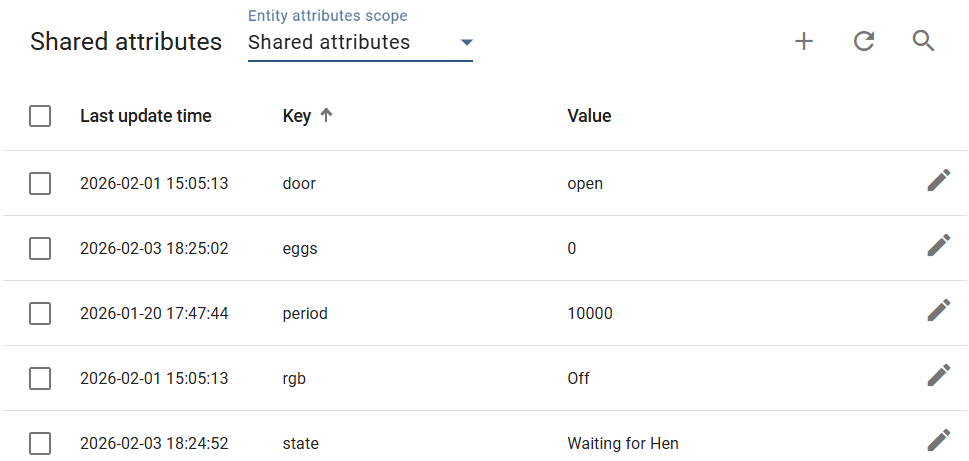

Figure 14: ThingsBoard Shared Attributes Configuration

The platform utilizes **Server Attributes** to maintain the digital twin of each device. These attributes are modified on the server with the Rule Engine. As shown in Figure 15, the following key attributes are managed:

- `nest_id`: A unique identifier for the specific physical location within the farm.
- `active`: Indicates whether the device is currently operational.
- `inactivityAlarmTime`: Time threshold for inactivity alarm generation.
- `lastActivityTime`: Timestamp of the last received telemetry data.
- `lastConnectTime`: Timestamp of the last successful connection to the server.
- `lastDisconnectTime`: Timestamp of the last disconnection from the server.

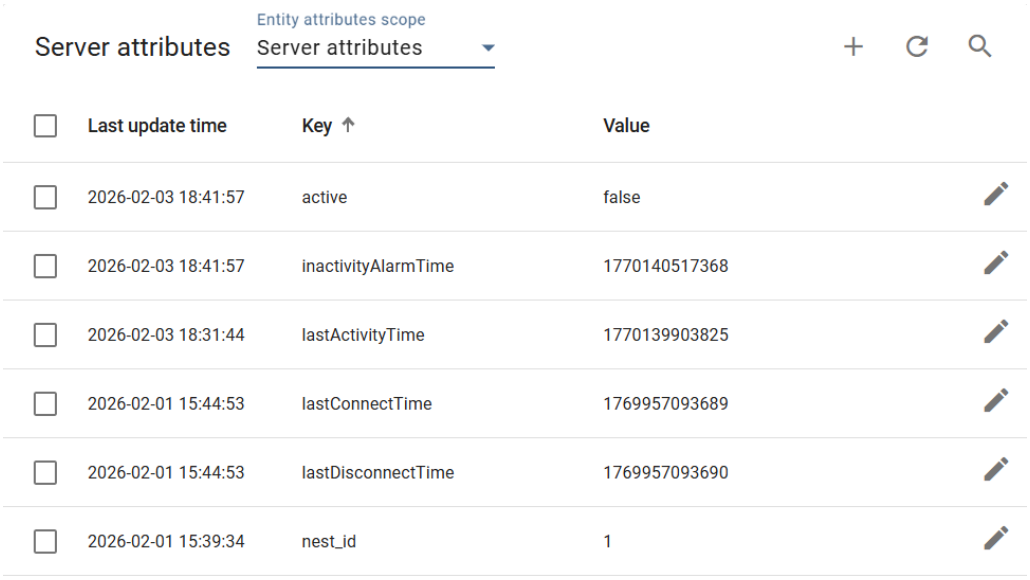

Figure 15: ThingsBoard Server Attributes Configuration

The client can modify the **Client Attributes** to change configuration data on the device. These attributes can be read on the server side. As shown in Figure 16, the following key attributes are managed:

- `latitude / longitude`: Geographic latitude and longitude coordinates of the device.
- `maxTemp / minTemp`: Maximum and minimum temperature thresholds for alarm generation.

- `maxHum / minHum`: Maximum and minimum humidity thresholds for alarm generation.
- `avgWeight / minWeight`: Average and minimum weight thresholds to estimate egg presence.
- `validUID`: Authorized UID for door access.
- `henUID`: UID of the hen.

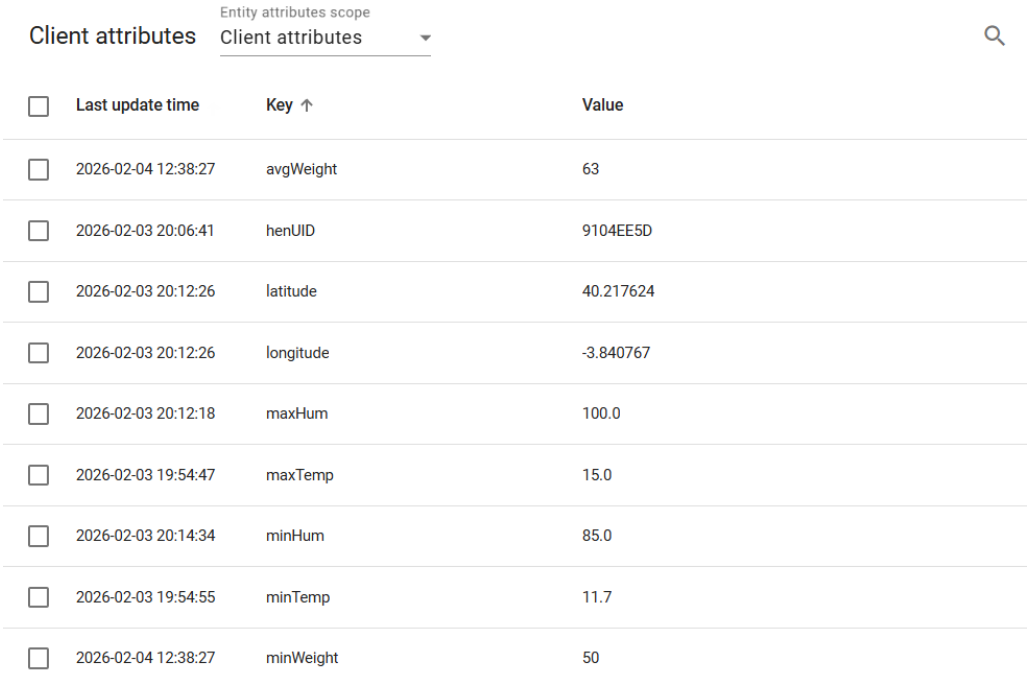

Figure 16: ThingsBoard Client Attributes Configuration

The system performs continuous data ingestion from both physical and simulated sources. The telemetry stream provides real-time visibility into the farm's conditions. Each device pushes a JSON payload containing:

- **Environmental Data**: `temperature` and `humidity` levels.
- **Production Data**: `weight` readings from the load cells to detect egg presence.
- **Identification Data**: The `UID` from the RFID reader to identify the hen or authorized personnel currently at the nest.
- **Error Reporting**: An `error` key is included in the payload if the device detects any hardware malfunction.

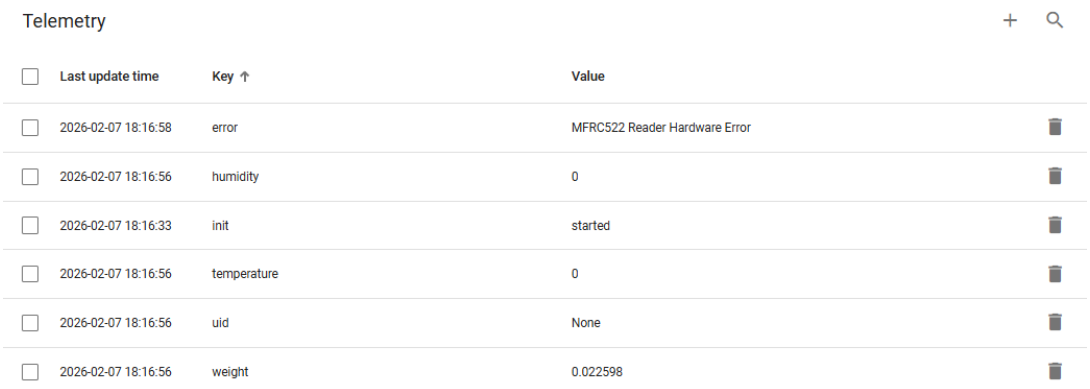

Figure 17: Real-Time Telemetry Data

Rule Chain

The centralized logic of the NEST ecosystem is governed by a custom rule chain named `NEST Device`. This engine processes incoming data through a series of filtering, enrichment, and action nodes to automate the NEST's behavior and notify the user of critical events in real-time.

When a message reaches the server, a **Message Type Switch** node segregates traffic based on its nature. The system primarily handles the following interaction streams:

- **Telemetry Processing:** Incoming telemetry—comprising `temperature`, `humidity`, `weight`, and `uid`—is validated by a **Check Telemetry** node. If the required keys are missing, a `MAJOR` alarm titled "Incorrect Message Syntax" is created. Valid data is persisted in the database via the **save data from sensor** node.
- **Attribute Management:** The system processes `Post attributes` messages to update client-side configurations, such as geographic coordinates or sensor thresholds, using the **Save Client Attributes** node.

The rule chain implements a comprehensive set of business logic to manage the nesting box lifecycle, environmental safety, and secure access:

- **Device Initialization**: To ensure the system starts in a known state, the **Check Initialization** node detects `init` messages. Upon detection, the **Device Initialization** transformation node resets the FSM to `Waiting for Hen`, clears the egg count, opens the door, and turns off the RGB LED.
- **Environmental Monitoring**: The system uses a "Fetch-and-Check" strategy. Metadata is enriched with thresholds (`maxTemp`, `minHum`, etc.) via the **TempRange** and **HumRange** nodes. Subsequently, **ThingsBoard Expression Language (TBEL)** scripts in the **Check Temperature** and **Check Humidity** nodes evaluate if the real-time values fall outside these safe bounds, triggering or clearing `MAJOR` alarms accordingly.
- **Access Control and Interactive Feedback**: Door security is managed by the **UIDs** attribute node. If the incoming `uid` matches the `validUID` (farmer), the **Open Door** node toggles the `door` shared attribute and sets the LED to `Green`. If the `uid` is unrecognized but not "None", the **Reject Door Use** node provides visual feedback by turning the LED `Red`. The **Turn Off RGB** node ensures the LED returns to an idle state after any interaction.
- **FSM Integration**: A sophisticated **Finite State Machine (FSM)** tracks the biological and operational cycle:
  - **Waiting for Hen / Hen Inside:** The **Check FSM State** node triggers a transition when the `henUID` is identified.
  - **Eggs Inside:** When the hen leaves and a residual weight is detected, the **Change FSM State** node calculates the number of eggs using the formula: round(weight/avgWeight). This triggers a `CRITICAL` **Eggs Inside** alarm.
  - **Recolecting Eggs:** The state transitions to `Recolecting Eggs` when an authorized `validUID` is detected while eggs are present. Once the weight drops below `minWeight`, the system resets to the initial state.
- **Telegram Notification System**: This implementation features a real-time alerting bridge. Whenever environmental alarms or `CRITICAL` production alerts (like egg detection) are triggered, a **Transformation Node** generates a formatted string. This payload is sent via a **REST API Call** node to the Telegram Bot API, providing the farmer with immediate status updates and device identification directly on their mobile device.
- **Error Handling**: Reliability is reinforced by the **Check Error** node. If the hardware reports an internal failure through an `error` key, the **Hardware Error** node creates a `CRITICAL` alarm containing the specific error message and a timestamp, immediately notifying the user via Telegram to prevent data loss or animal distress.

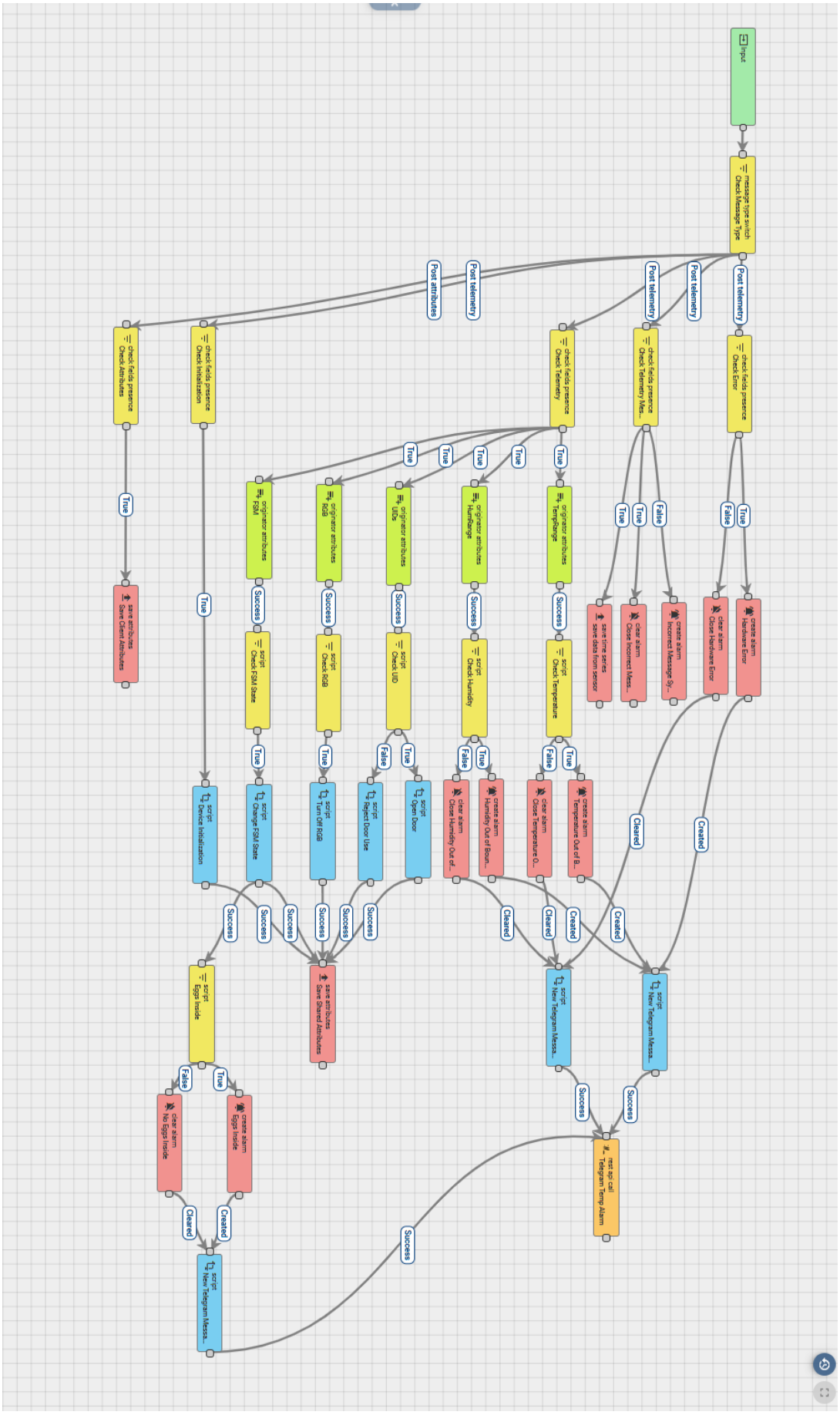

Figure 18: Rule Chain for NEST Device

Dashboard

The visualization layer is implemented through a multi-state interactive dashboard designed to provide both a global overview of the farm and detailed insights into individual nodes.

The primary dashboard state (see Figure 19) is designed for high-level monitoring. It integrates three main functional areas:

- **Geographic Map:** Utilizing the `OpenStreet Map` widget, all four devices (NEST 1-4) are geolocated based on their `latitude` and `longitude` attributes. The markers provide a quick visual reference of the sensors' physical distribution.
- **Entities Summary Table:** A real-time grid displaying the current `temperature`, `humidity`, `weight`, and `door status` for all devices in the network.
- **Global Alarms Feed:** A centralized list showing active alarms across the entire ecosystem.

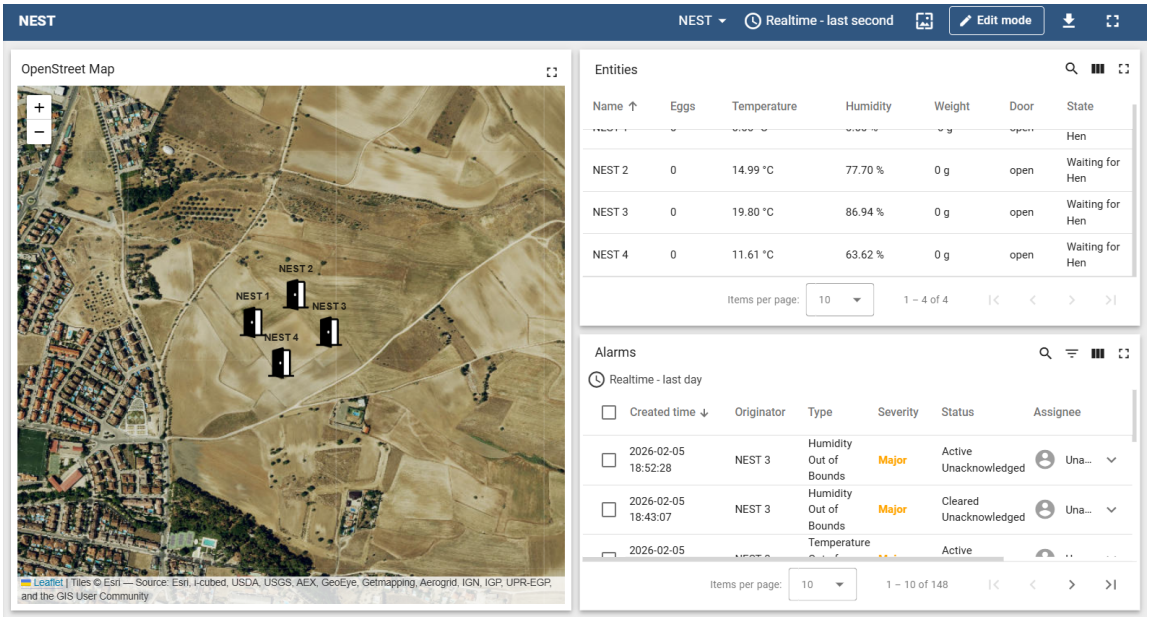

Figure 19: Main Dashboard View

To enable drill-down capabilities, the dashboard implements a **state-based navigation** logic. By clicking on a specific device name in the table or its marker on the map, the dashboard transitions to a "Details" state (see Figure 21).

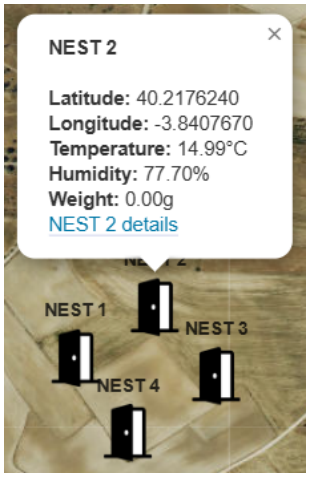

Figure 20: Marker Information Popup

This view focuses exclusively on the telemetry of the selected node:

- **Indicator Cards:** Large, high-visibility widgets displaying the latest values for `Temperature`, `Humidity`, `Weight`, and current `Door` status.
- **Historical Trend Analysis:** A time-series line chart that allows the farmer to visualize the evolution of environmental variables and weight over time, which is essential for identifying patterns in egg-laying behavior.

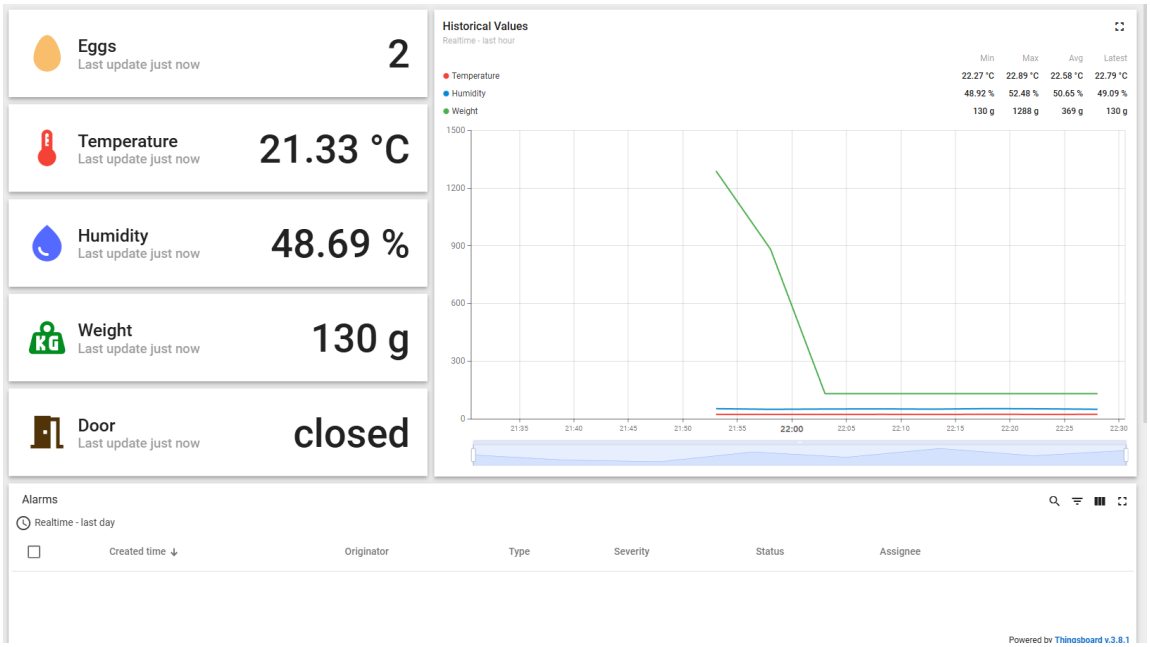

Figure 21: NEST Details Dashboard View

3.4.4 Telegram

As a way for visualization of the data and control the system, a Telegram bot named `NEST` has been created. In this bot, the farmer can see the data allocated in the telemetry fields, as well as the parameters configured in the attributes. Once the farmer has installed the bot, he will be receiving alarms from the NESTs. The more common are the sensor's bounds ones.

As seen in the figure 22, when an alarm is created where the value of any sensor is out of the parameters configured a notification message is sent to telegram to inform about the sensor that has sent the alarm. Once the value is stable again, another message is sent notifying that the alarm is unset.

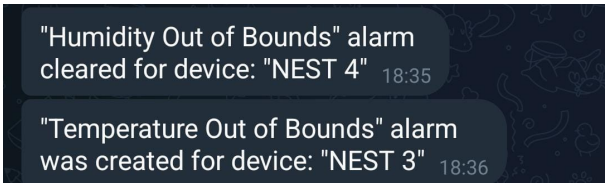

Figure 22: Telegram Messages Boundaries Notification

When an egg is laid, once the animal is out of the NEST, a notification with the estimated number of eggs is sent to the bot, as well, when the eggs are recollected, another notification is sent, as demonstrated in figure 23.

Figure 23: Eggs Telegram Messages

The telegram bots function with pre-programmed shortcuts to access some actions. For this project, five essential commands had been created, as display in figure 25:

- `/nest`: It is the principal command of the bot. When this command is typed, a list of the available NESTs, figure 24 is available to interact for the user.

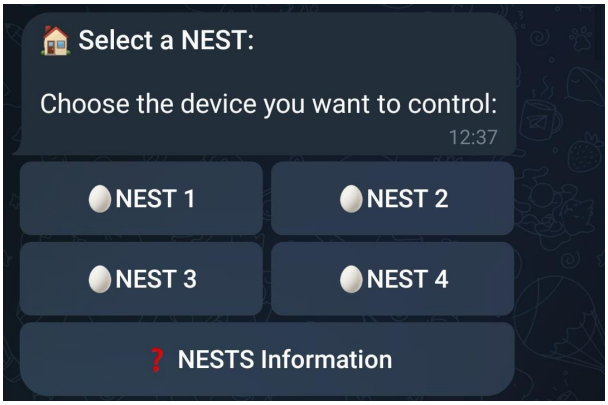

Figure 24: NEST Selection Telegram Menu

- `/login`: For some actions to use the system, a username and a password is required, with this command the user is able to log in the platform.
- `/logout`: If the user wants or needs to disconnect from the platform they can use this command.
- `/status`: Checks the status of the connection.
- `/help`: List all the commands that the bot can use with a little explanation of how to use it.

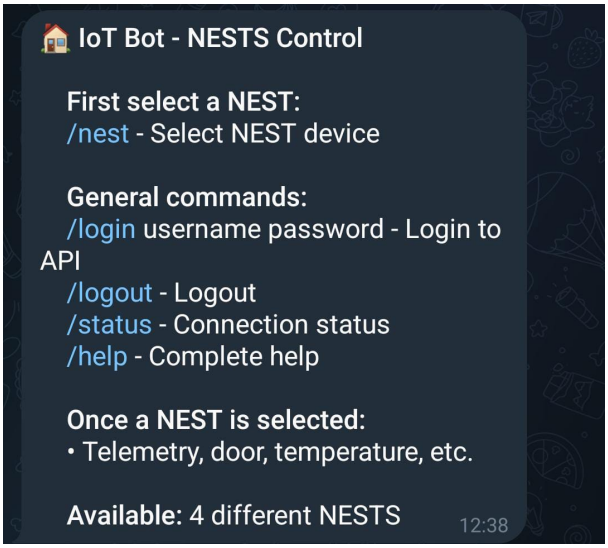

Figure 25: Main Telegram Menu

Once the NEST is selected, another menu to interact with the system is displayed, figure 26. The options in the menu are the following ones:

- **Telemetry**: If logged, this option gives back the information of the telemetry in the NEST selected.
- **Door Control**: Shows the current state of the door.
- **Temperature**: When selected, it shows the current temperature value and boundaries, it also shows how, if logged, change the boundaries of the temperature:

  - `/temperature max VALUE`: With this command the user can change the maximum value of the temperature boundaries.
  - `/temperature min VALUE`: With this command the user can change the minimum value of the temperature boundaries.

- **Humidity**: Similar as the Temperature option, does the same things as the other option but using `/humidity` instead.
- **RGB**: Shows the current state of the LED.
- **Eggs**: Shows the estimated quantity of eggs, that correspond with the selected animal. If logged, in this menu the user can change the type of animal that is using the NEST, changing, in this way, the internal parameters of the system for the calculus of the egg count.
- **Location**: Shows the location, in coordinates, of the NEST selected. If logged, the user can establish, by hand or using the Global Positioning System (GPS) location of the phone, the new coordinates of the NEST.
- **Change NEST**: A simple button for coming back to the previous menu for select the NEST.

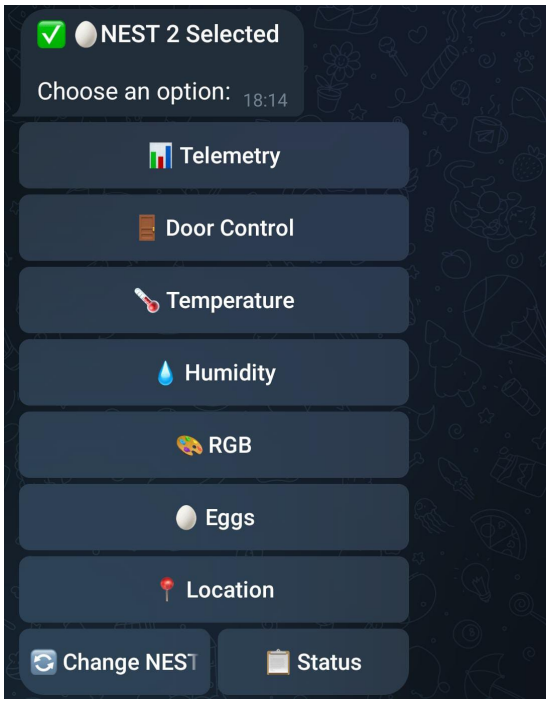

Figure 26: Main NEST Telegram Menu

3.5 Project Documentation

The technical documentation is structured using the **Sphinx** documentation generator. Sphinx provides a flexible and professional layout, leveraging reStructuredText to seamlessly integrate high-level project descriptions with detailed API technical specifications.

The documentation process is fully automated through a Continuous Integration (CI) workflow powered by GitHub Actions. This pipeline executes the Sphinx engine to extract structured information directly from source code docstrings and comments.

Upon every push to the repository, the GitHub Action triggers to regenerate the HyperText Markup Language (HTML) files and automatically host them via GitHub Pages. This automation eliminates discrepancies between the current software version and its public documentation, ensuring the manual is updated without the need for manual intervention.

The documentation is available at the following URL: `https://estelamb.github.io/NEST_ASP/`.

4 Results

4.1 Unitary Tests

To ensure the reliability of the NEST system, each hardware peripheral underwent a dedicated Unitary Testing phase. This process verified that both the physical wiring and the specific software drivers worked correctly before integrating them into the main FreeRTOS multitasking firmware.

Table 11: Hardware Unitary Testing Procedures

Component	Test Objective	Verification Method
DHT22	Data consistency and single-bus timing.	Measure temperature and humidity conditions.
Load Cell (HX711)	Accuracy and calibration of weight sensing.	Applied weights and verified the output in the Serial Monitor.
MFRC522 (RFID)	Successful UID identification and SPI stability.	Scanned multiple tags to ensure unique hex strings were captured correctly.
SG90 Servos	Precision of movement and power stability.	Commanded `open` and `closed` door positions.
RGB LED	Color mixing and PWM duty cycle validation.	Executed a test to display Red, Green, and Blue colors.

4.2 Local Test

A **Local Test** was performed to verify the integrity of the data acquisition. As shown in Figure 27, the system provides the measured values via the serial console.

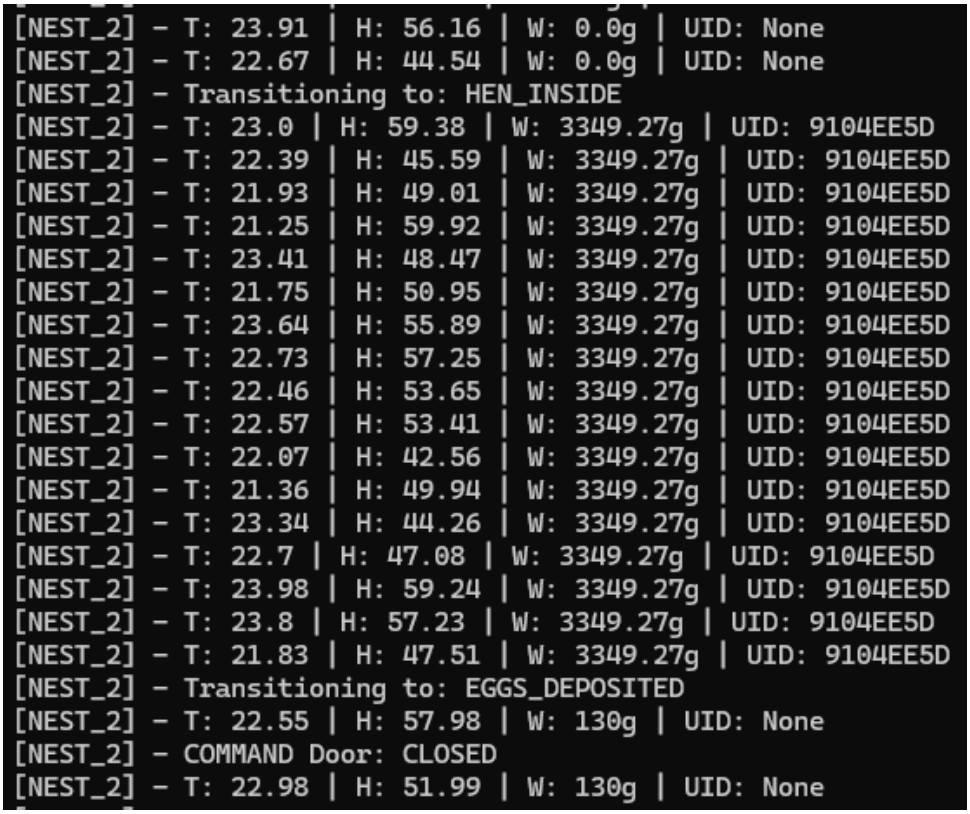

Figure 27: Local Feedback Test

4.3 Hardware Errors

The **Hardware Errors** test was designed to evaluate the system's robustness in handling hardware malfunctions. The system successfully detected and reported errors, such as sensor disconnections or power issues, ensuring that the user is promptly informed of any hardware-related problems.


```
{"temperature":0,"humidity":0,"weight":-0.037663,"uid":"None","error":"DHT22 Failure;"}
{"temperature":0,"humidity":0,"weight":0.03578,"uid":"None","error":"DHT22 Failure;"}
{"temperature":0,"humidity":0,"weight":0.084743,"uid":"None","error":"DHT22 Failure;"}
{"temperature":22.2,"humidity":93.8,"weight":0.037663,"uid":"None"}
```


Figure 28: Hardware Errors Test

4.4 ThingsBoard Test

The **ThingsBoard Test** focused on validating the connection between the NEST Devices and the IoT platform. As shown in Figure 29, the testing phase successfully verified the integrity of the secure MQTT stack and the real-time synchronization of the digital twin.

Telemetry

Last update time	Key	Value
2026-02-07 18:16:58	error	MFRC522 Reader Hardware Error
2026-02-07 18:16:56	humidity	0
2026-02-07 18:16:33	init	started
2026-02-07 18:16:56	temperature	0
2026-02-07 18:16:56	uid	None
2026-02-07 18:16:56	weight	0.022598

Figure 29: ThingsBoard Test

4.5 Attributes Changes

The **Attributes Changes** test validated the system's responsiveness to remote configuration updates. As shown in Figure 30, changes made to the device attributes on the ThingsBoard platform were successfully propagated to the NEST Device, demonstrating effective two-way communication.

```
[NEST_4] - COMMAND Door: CLOSED
[NEST_4] - COMMAND Door: OPEN
[NEST_4] - COMMAND LED Color: RED
[NEST_4] - COMMAND Period: 5.0s
```

Figure 30: Attributes Changes

4.6 Alarms Activation

The **Alarms Activation** test was designed to evaluate the system's ability to detect and report anomalies. The system successfully triggered alarms on the ThingsBoard platform, confirming its robustness in handling unexpected scenarios.

Alarms

Realtime - last day

Created time	Originator	Type	Severity
2026-02-07 18:27:40	NEST 1	Humidity Out of Bounds	Major
2026-02-07 18:27:40	NEST 1	Temperature Out of Bounds	Major
2026-02-07 18:27:08	NEST 1	Humidity Out of Bounds	Major
2026-02-07 18:27:08	NEST 1	Temperature Out of Bounds	Major

Figure 31: Alarms Activation

4.7 Telegram Bot

The **Telegram Bot** test validated the functionality of the user interface for remote monitoring and control. As shown in Figure 32, the bot successfully responded to user commands, providing real-time updates on the system's status and allowing for remote configuration changes.

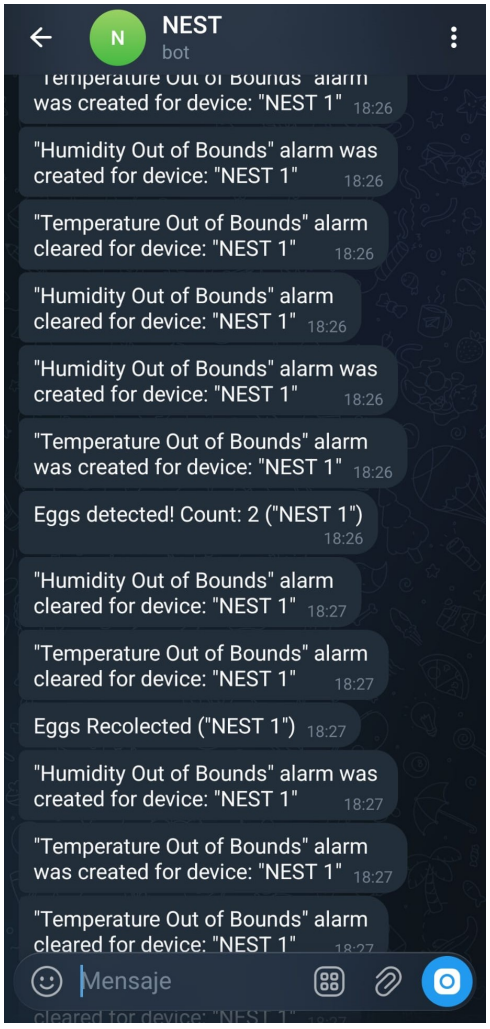

Figure 32: Telegram Bot Test

4.8 System Integration Test

The final validation of the project was conducted through a comprehensive **System Integration Test**, confirming the end-to-end functionality of the data pipeline, from physical sensing to ThingsBoard data management and visualization. As shown in Figure 33, the **ThingsBoard Dashboard** successfully aggregates all telemetry streams into a cohesive monitoring station.

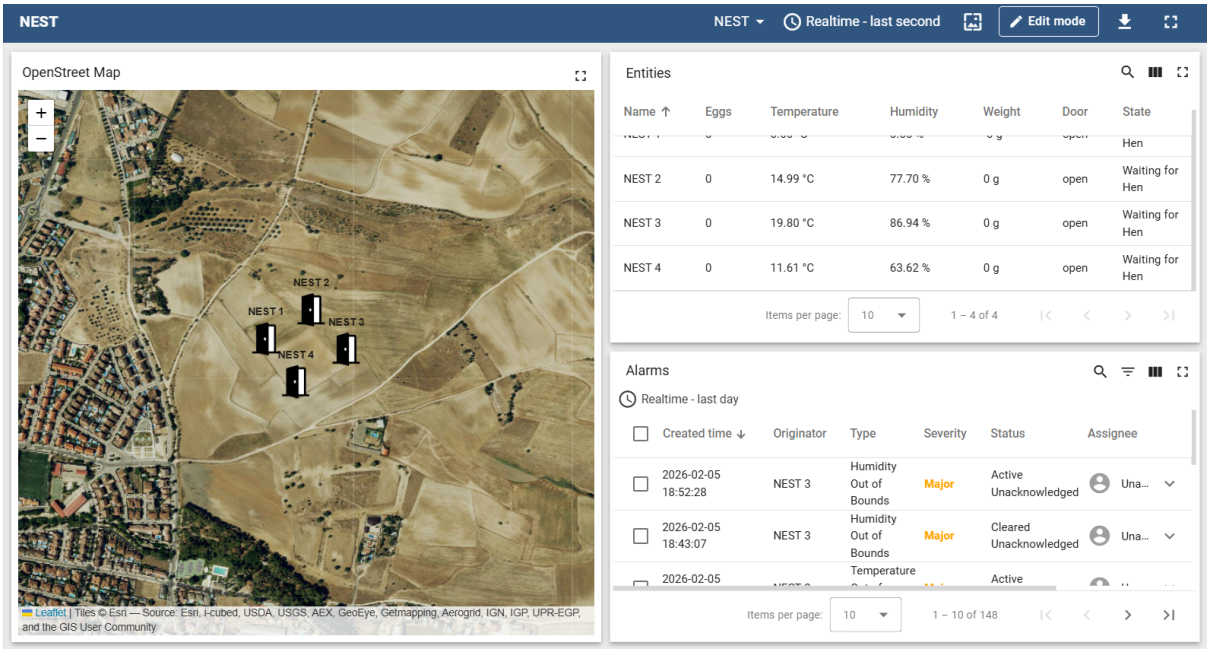

Figure 33: ThingsBoard Dashboard Result (1)

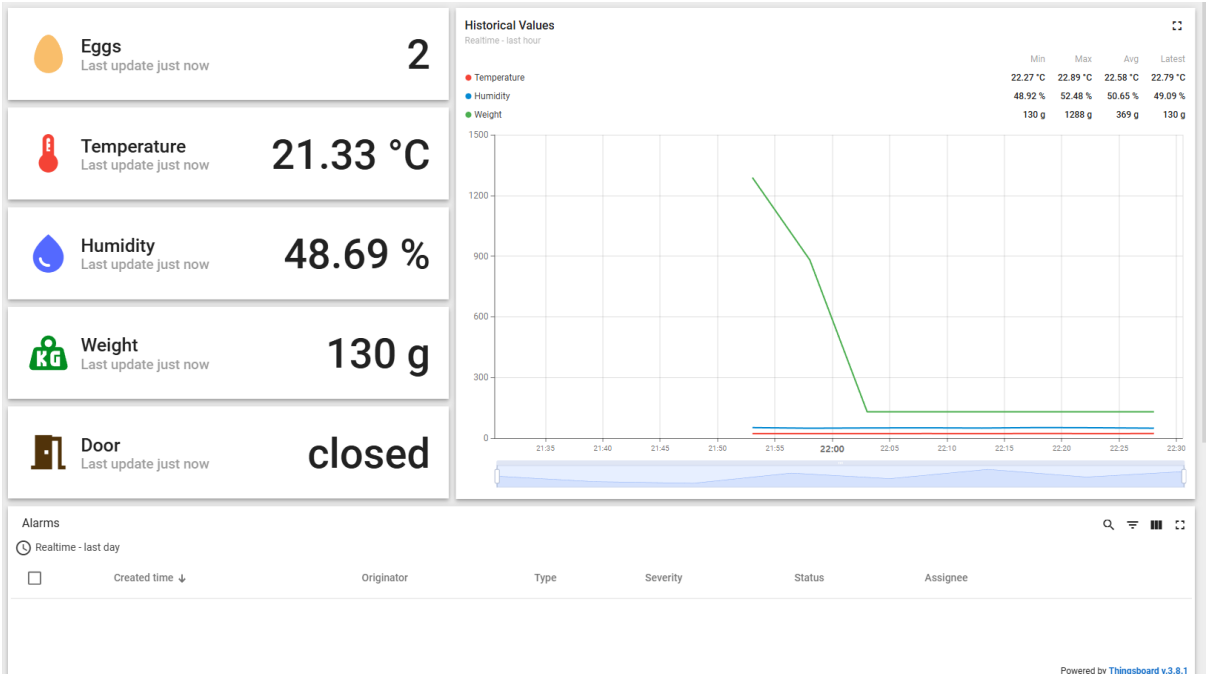

Figure 34: ThingsBoard Dashboard Result (2)

5 Conclusions and Future Works

5.1 Conclusions

The NEST project has successfully demonstrated the implementation of a comprehensive IoT ecosystem for the automated management of hen farms. By adopting a hybrid deployment strategy, the system effectively bridges the gap between the physical and virtual worlds, allowing real ESP32 nodes to operate in parallel with simulated entities. This architecture not only fulfills the project's scalability requirements, but also ensures a unified network capable of handling diverse data streams from various sensing and actuation layers.

A central achievement of the project is the successful deployment of autonomous Edge decision-making. By utilizing the FreeRTOS kernel on the ESP32 and implementing local operational logic, the system can trigger actuators, like the servomotors for the doors, without constant server connection. This decentralized intelligence ensures low latency and local autonomy vital for egg production.

Furthermore, the integration with ThingsBoard provides a robust infrastructure for data ingestion and remote commanding. Through the use of MQTT protocols and Shared Attributes, the system maintains a digital twin of each device, allowing for both historical trend analysis and real-time visualization via dashboards.

The project underscores the importance of a multi-channel interaction layer, where the interactive dashboard and the Telegram bot work together. By leveraging REST APIs for real-time notifications and Shared Attributes for remote commanding, the system ensures that critical production events, such as egg detection or environmental anomalies, are managed with high availability.

The Test Bench successfully validated this end-to-end data pipeline, proving that the hybrid approach is a viable and robust framework for modernizing smart farming operations. This project has effectively transformed raw telemetry data into actionable insights.

5.2 Future Work

Despite the successful implementation of the NEST ecosystem, the system remains a scalable framework designed for continuous evolution and technical refinement. The following points outline the roadmap for integrating more sophisticated local logic, enhanced user interfaces, and advanced data processing techniques to further optimize poultry farm management.

- **Predictive Analytics**: Leverage the historical data stored in the ThingsBoard Telemetry Database to implement machine learning models at the Edge for predicting egg-laying patterns and identifying early signs of hen illness based on environmental anomalies.
- **Edge Artificial Intelligence (AI) Integration**: Implement lightweight machine learning inference directly on the Edge to enable real-time anomaly detection and decision-making without relying on cloud connectivity, further enhancing the system's autonomy and responsiveness.
- **Telegram updates**: Better control of the users and actions that are able to be done with the bot, such us create UIDs for animals and users, controlling the door or implementing more types of birds.
- **Enhanced Dashboard**: Develop more sophisticated visualizations and alerting mechanisms on the ThingsBoard dashboard, including real-time heatmaps of environmental conditions and predictive maintenance alerts based on sensor data trends.
- **Power Optimization**: Explore deep-sleep modes for the ESP32 between periodic sensing cycles to enhance energy efficiency, which is critical for future deployments using battery or solar power in remote farm areas.
- **Sustainability and Energy Harvesting**: Research the integration of solar photovoltaic panels to create a maintenance-free energy cycle for the NEST device, reducing the reliance on external power bridges.
- **Multi-Protocol Redundancy**: Implement a fallback communication strategy using Long Range Wide Area Network (LoRaWAN) to ensure that critical alarms and autonomous edge decisions are transmitted even during Wi-Fi outages, which are common in rural farming environments.

6 Bibliography

[1] "ESP32 Wi-Fi & Bluetooth SoC — Espressif Systems," Accessed: Jan. 25, 2026. [Online]. Available: https://www.espressif.com/en/products/socs/esp32.

[2] "108322 Soldered — Mouser," Mouser Electronics, Accessed: Jan. 25, 2026. [Online]. Available: https://www.mouser.es/ProductDetail/Soldered/108322?qs=%252BXxaIXUDbq2Uuz4PwBDoag%3D%3D.

[3] "Load Cell Amplifier - HX711," Accessed: Jan. 25, 2026. [Online]. Available: https://grobotronics.com/load-cell-amplifier-hx711withoutheader.html?sl=en.

[4] S. Santos. "ESP32 with Load Cell and HX711 Amplifier (Digital Scale) — Random Nerd Tutorials," Accessed: Jan. 25, 2026. [Online]. Available: https://randomnerdtutorials.com/esp32-load-cell-hx711/.

[5] "DHT22 – Temperature and Humidity Sensor," Components101, Accessed: Jan. 25, 2026. [Online]. Available: https://components101.com/sensors/dht22-pinout-specs-datasheet.

[6] "MODULO RFID-RC522 KIT RFID NFC CON TARJETA Y LLAVERO," tiendatec.es, Accessed: Jan. 25, 2026. [Online]. Available: https://www.tiendatec.es/maker-zone/rf-rfid-nfc/394-modulo-rfid-rc522-kit-rfid-nfc-con-tarjeta-y-llavero-arduino-y-raspberry-pi-8403941180015.html.

[7] "MICROSERVO SG90 180º / 360º," tiendatec.es, Accessed: Jan. 25, 2026. [Online]. Available: https://www.tiendatec.es/maker-zone/servos/583-microservo-sg90.html.

[8] "Curso: Architectures and service plataforms — MOODLE UPM - OFICIALES 25-26," Accessed: Jan. 21, 2026. [Online]. Available: https://moodle.upm.es/titulaciones/oficiales/course/view.php?id=448.

[9] "ThingsBoard — Open-source IoT (Internet of Things) Platform," ThingsBoard, Accessed: Jan. 21, 2026. [Online]. Available: https://thingsboard.io/.

[10] "Arduino IDE — Arduino Documentation," Accessed: Jan. 21, 2026. [Online]. Available: https://docs.arduino.cc/software/ide/.

[11] "Egg Storage and Handling - Brinsea," Accessed: Jan. 18, 2026. [Online]. Available: https://www.brinsea.com/t-eggstorage.aspx.

Appendix A - GitHub

A.1 GitHub Repository - Source Code

The complete source code for the NEST System is hosted on GitHub. The repository, containing all project files and version history, is available at: `https://github.com/Estelamb/NEST_ASP`.

A.2 GitHub Pages - Documentation

Comprehensive technical documentation and guides are published via GitHub Pages. These resources can be accessed at: `https://estelamb.github.io/NEST_ASP/`.

A.3 GitHub Action - Workflow

Listing 1: GitHub Action workflow

```
name: Deploy Documentation

on:
  push:
    branches: [ main ]
  workflow_dispatch:

permissions:
  contents: write
  pages: write
  id-token: write

jobs:
  build-deploy:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout repository
      uses: actions/checkout@v4
      with:
        persist-credentials: false

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.x'

    - name: Install dependencies
      working-directory: Docs
      run: pip install -r requirements.txt

    - name: Build documentation
      working-directory: Docs
      run: |
        rm -rf build/ pdfbuild/
        sphinx-build -b html source build
        sphinx-build -b simplepdf source pdfbuild
        mkdir -p build # Ensure build directory exists before moving or touching
        mv pdfbuild/*.pdf build || echo "No PDF files to move"
        touch build/.nojekyll

    - name: Configure Git
      run: |
        git config --global user.name "GitHub Actions"
        git config --global user.email "actions@github.com"

    - name: Deploy to GitHub Pages
      uses: peaceiris/actions-gh-pages@v4 # UPDATED TO V4
```

```
      with:
        github_token: ${{ secrets.GITHUB_TOKEN }} # RECOMMENDED FIX
        publish_dir: Docs/build
        keep_files: false
        force_orphan: true
```