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

API Application Programming Interface

CI Continuous Integration

FSM Finite State Machine

GPIO General Purpose Input-Output

HTML HyperText Markup Language

IDE Integrated Development Environment

IoT Internet of Things

JSON JavaScript Object Notation

LED Light Emitting Diode

MQTT Message Queuing Telemetry Transport

NEST Next-generation Edge System for Tracking

PWM Pulse Width Modulation

RFID Radio Frequency Identification

RGB Red, Green and Blue

SoC System on a Chip

SPI Serial Peripheral Interface

TLS Transport Layer Security

UID Unique Identifier

UML Unified Modeling Language

Todo list

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

3.1.1 System Architecture

3.1.2 System Workflow

3.2 Hardware Device

3.2.1 Block diagram

The block diagram shown in Figure 1 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) Light Emitting Diode (LED)). Each peripheral is connected through specialized interfaces, allowing the system to process data at the edge and perform autonomous egg protection tasks.

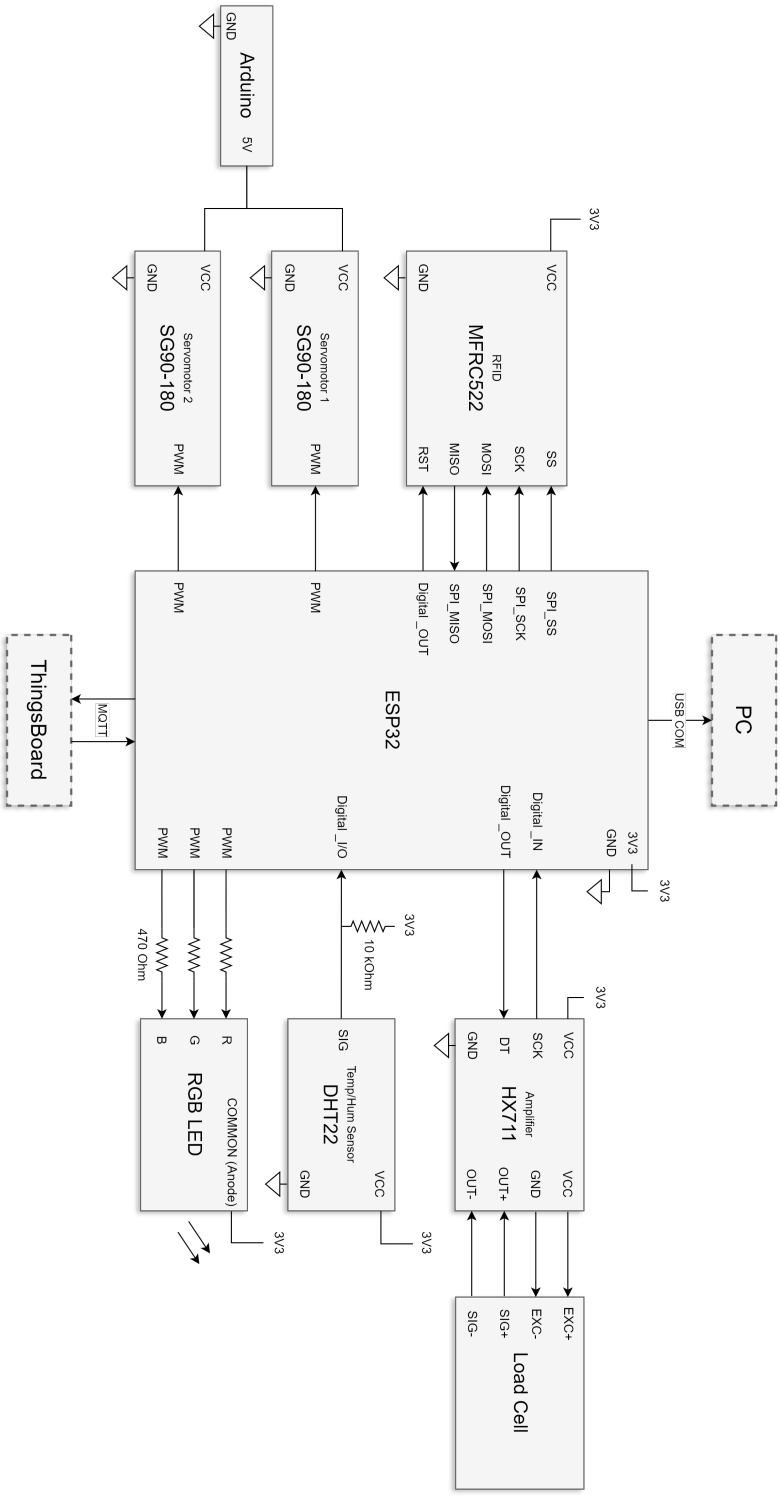

Figure 1: Block Diagram of the Hardware System

The ESP32 microcontroller manages the diverse hardware modules. A digital interface is used to gather periodic temperature and humidity data from the DHT22. The Serial Peripheral Interface (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.

3.2.2 Hardware Components

citas

arreglar

- **ESP32 Microcontroller**: The ESP32 is a powerful System on a Chip (SoC) featuring dual-core processors and integrated Wi-Fi and dual-mode Bluetooth. In the NEST project, it serves as the central processing unit and communication hub.
- **DHT22 Temperature and Humidity Sensor**: 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.
- **Weight Scale and HX711 Amplifier**: 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. The HX711 connects to the ESP32 through GPIO 16 (DOUT) and GPIO 4 (SCK) for serial data communication.
- **MFRC522 RFID Module**: 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**: 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.

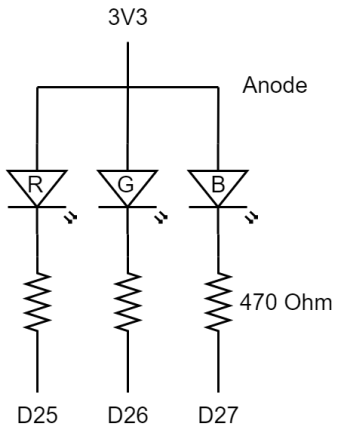

Figure 2: RGB LED circuit diagram

- **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.

3.2.3 Hardware Summary

arreglar

Table 4: Hardware Specifications and Components

Component	Function
ESP32	Central processing and distributed Edge intelligence
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)

3.2.4 Interfaces of the system

Table 5: 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

Table 6: Arduino Connections

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

Table 7: 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 Implementation and Development

introducción

3.3.1 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 Unique Identifier (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.

comprobar

Table 8: External Libraries and Authorship

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

3.3.2 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 production, 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 9: 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.

3.3.3 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 maintain the digital twin of each device and enable remote commanding. These attributes are stored on the server and synchronized with the devices via MQTT. As shown in Figure 3, 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.
- `location`: Static geographic coordinates (`latitude` and `longitude`) for asset tracking.
- `nest_id`: A unique identifier for the specific physical location within the farm.

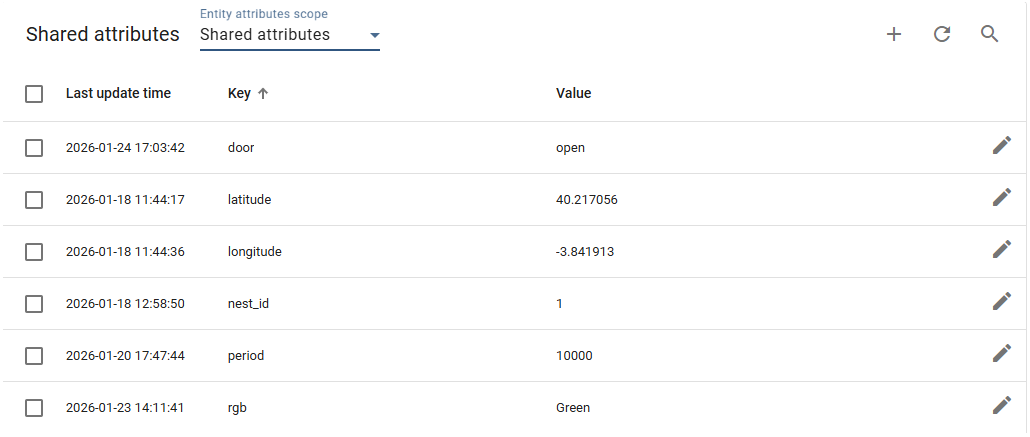

Figure 3: ThingsBoard Shared Attributes configuration for NEST devices

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.

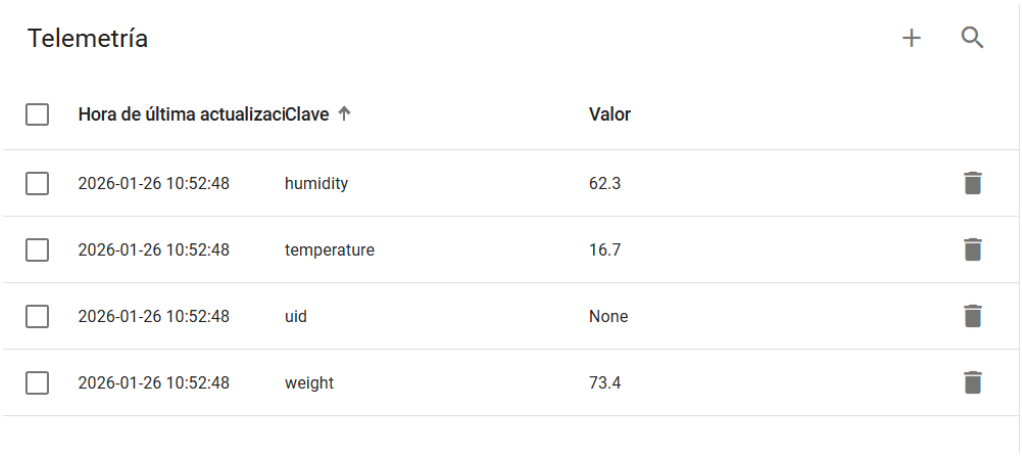

Figure 4: Real-time telemetry data

Rule Chain

The main characteristic of ThingsBoard are their **Rule chains**, which when an input is detected execute a series of actions determinated by its nodes and the configuration. For this project, a specific **Rule Chains** has been developed named `NEST Device`, which can be seen below (Figure 5).

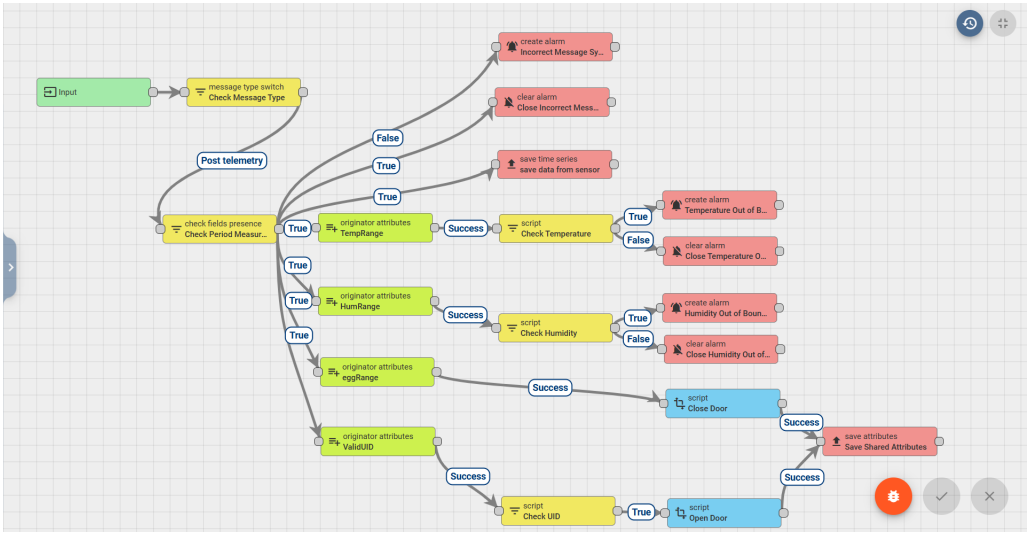

Figure 5: Real-time telemetry data

When messages arrive, the rule chain filters for *Post telemetry* messages and checks for the expected fields, and saves the data in the device's latest telemetry. If none of the fields are present an alarm is created to inform of a problem. Then the message is enriched in different ways with the device's shared attributes that define the ranges of the telemetry values. First, temperature and humidity are checked to be in the accepted range, if not another alarm is created.

The weight, if it is above a minimun threshold it checks how many eggs are expected base on the averge weight. If there are eggs and no UID is detected, it sets the door attribute to "closed" and the sets the expected amount of eggs. If a valid UID is detected, meanwhile the door is closed, it sets the attribute to "open".

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 6) 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.

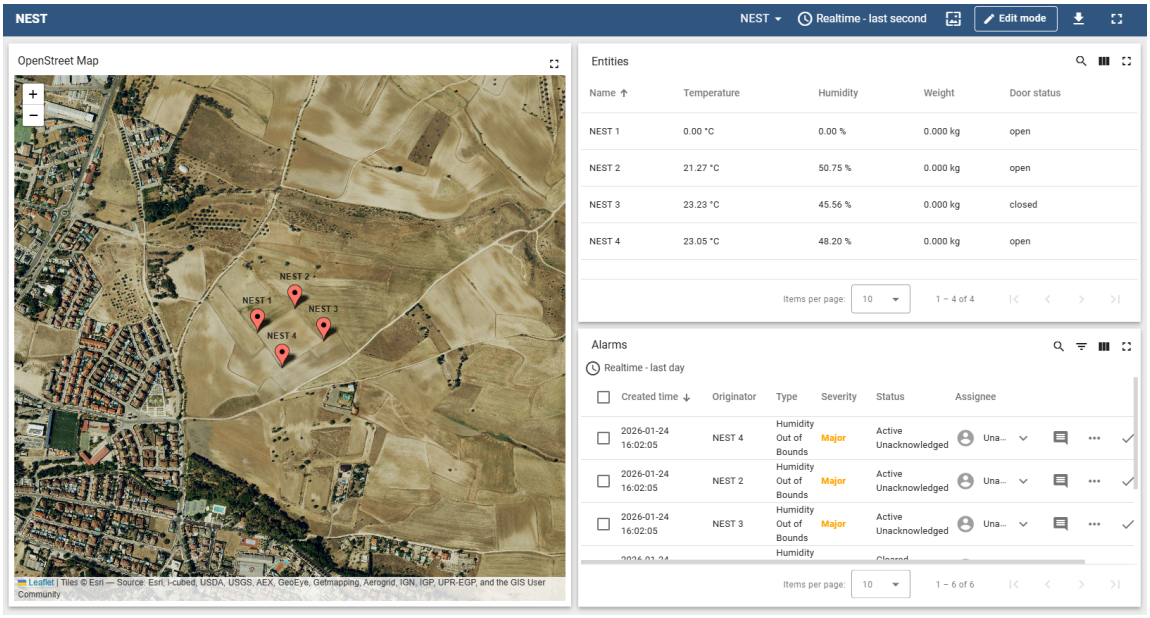

Figure 6: Main Dashboard View

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 8).

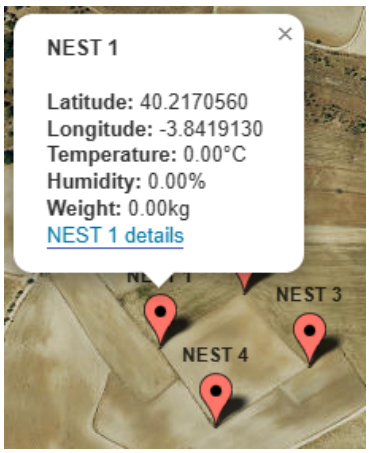

Figure 7: Marker Information Popup

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.

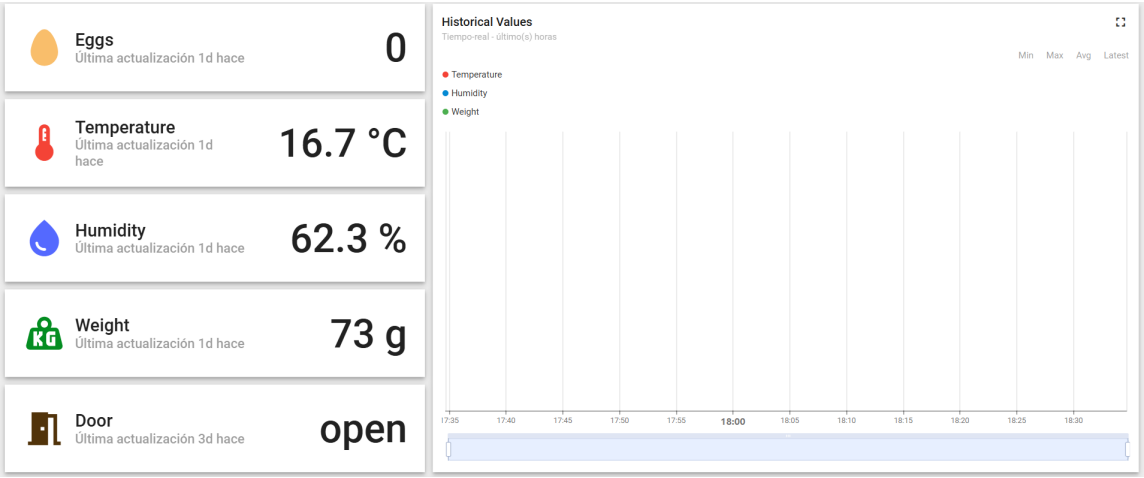

Figure 8: NEST Details Dashboard View

3.4 Project Documentation

For the next development stage, technical documentation will be structured using the **Sphinx** documentation generator. Unlike traditional tools, Sphinx allows for a highly flexible and professional layout, utilizing reStructuredText or Markdown to integrate both high-level project descriptions and technical Application Programming Interface (API) details.

In the upcoming **second version** of the project, the documentation process will be fully automated through a Continuous Integration (CI) workflow using a GitHub Action. This planned workflow will execute the Sphinx engine to extract structured information directly from the docstrings and comments within the source code. This ensures that every module, class, and function is documented at the point of implementation, keeping the technical manual perfectly synchronized with the evolving codebase.

Once the second version is deployed, every push to the repository will trigger the GitHub Action to regenerate the HyperText Markup Language (HTML) files and automatically host them via GitHub Pages. This prevents discrepancies between the current firmware and its public description while removing the need for manual updates.

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

4 Results

5 Conclusions and Future Works

5.1 Conclusions

5.2 Future Work

6 Bibliography

[1] "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`

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

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

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