Greenhouse Digital Twin Project | Version 2.0

Technical Specifications Document

Guiziou Maleaume | Juan Francisco Martinez

|  |  |  |
| --- | --- | --- |
|  |  | Text  Description automatically generated |

# Document Version Control

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version No.** | **Implemented By** | **Revision Date** | **Approved By** | **Approval Date** | **Description** |
| 0.1 | Andrew de Juan | 27/07/2023 | <Name> | <dd/mm/yyyy> | Initial draft |
| 0.2 | Andrew de Juan | 03/08/2023 |  |  | WBS headings and work packages |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Table of Contents

[Document Version Control 2](#_Toc204347771)

[Introduction 5](#_Toc204347772)

[Background 5](#_Toc204347773)

[Document Purpose 5](#_Toc204347774)

[Intended Audience 5](#_Toc204347775)

[Overview 6](#_Toc204347776)

[Project Scope 6](#_Toc204347777)

[Terminology 7](#_Toc204347778)

[1 Project Management 8](#_Toc204347779)

[1.1 Define Scope and Objectives – This is a Work Package 8](#_Toc204347780)

[1.1.1 Task 01 8](#_Toc204347781)

[1.1.2 Task 02 8](#_Toc204347782)

[1.1.3 Task 03 8](#_Toc204347783)

[1.2 Identify stakeholders and their roles 8](#_Toc204347784)

[1.2.1 Task 01- Intern 8](#_Toc204347785)

[1.2.2 Task 02- Supervisor 8](#_Toc204347786)

[1.2.3 Task 03- Future Developers 9](#_Toc204347787)

[2 Requirements Gathering & Analysis 10](#_Toc204347788)

[2.1 Identify sensor data sources 10](#_Toc204347789)

[2.1.1 Sensor Research and Selection 10](#_Toc204347790)

[2.1.2 Vendor Evaluation and Procurement 10](#_Toc204347791)

[2.1.3 Integration Planning 10](#_Toc204347792)

[2.1.4 Data Interfacing and Compatibility 10](#_Toc204347793)

[2.2 Gather functional requirements 11](#_Toc204347794)

[2.2.1 User Interviews and Surveys 11](#_Toc204347795)

[2.2.2 Use Case Definition 11](#_Toc204347796)

[2.2.3 Requirements Workshops 11](#_Toc204347797)

[2.2.4 Prototyping and Feedback 11](#_Toc204347798)

[2.2.5 Requirement Documentation 11](#_Toc204347799)

[2.3 Gather non-functional requirements 12](#_Toc204347800)

[2.3.1 Task 12](#_Toc204347801)

[2.4 Analyse and prioritize requirements 12](#_Toc204347802)

[3. Design & Architecture 13](#_Toc204347803)

[3.1 Define data schema and storage requirements 13](#_Toc204347804)

[3.2 Determine the technologies and frameworks for the frontend 13](#_Toc204347805)

[3.3.1 Unity 3D Game Engine 13](#_Toc204347806)

[3.3.2 Install Newtonsoft's Json.NET Package 13](#_Toc204347807)

[3.3 Design the front-end user interface 13](#_Toc204347808)

[3.2.1 Camera System 13](#_Toc204347809)

[3.2.2 Tooltip System 13](#_Toc204347810)

[3.4 Design the back-end architecture for data processing 14](#_Toc204347811)

[3.5 Plan data integration and APIs between sensors and the front-end 15](#_Toc204347812)

[3.6 Server deployment 15](#_Toc204347813)

[4 Sensor Data Acquisition 17](#_Toc204347814)

[4.1 Procure necessary sensors and hardware 17](#_Toc204347815)

[4.2 Develop or configure data collection software 17](#_Toc204347816)

[4.3 Test data acquisition system and validate sensor data 21](#_Toc204347817)

[5 Data Processing and Storage 22](#_Toc204347818)

# Introduction

## Background

The Internet of Things (IoT) enables the collection of environmental data through connected sensors, offering valuable insights for sustainable development, especially in precision agriculture. This greenhouse project is part of an IoT-driven initiative conducted at the Nimbus Research Centre in Cork (Ireland), and it aims to provide a complete monitoring system for greenhouse conditions.

Throughout this internship, I worked on developing a full-stack application capable of receiving, storing, and displaying sensor data coming from a set of LoRaWAN-based environmental sensors. The goal was to design and implement a robust and intuitive dashboard to visualize information such as temperature, humidity, CO₂ levels, light intensity, and soil parameters in real-time.

## Document Purpose

The purpose of this document is to provide sufficient detail on the overall process in achieving this end-to-end system architecture from sensor to front-end application design and development. This project is broken into multiple work packages which also function as headings in this document structure. The intended audience includes designers, developers, system integrators, and target end-users of the platform. This document is envisaged to serve as a blueprint with sufficient detail to replicate the workflow as well as commentaries on the developers’ experience working on platforms and systems.

It also explains the encountered challenges, the design choices made, and provides instructions on how to replicate and maintain the system. Unlike the previous version of the project, this new implementation was developed from scratch (except for the sensors and some APIs, as well as the files concerning the database and the login page), with a particular focus on modularity, maintainability, and user experience.

## Intended Audience

People with different skillsets are likely to see this document for their own purposes, as follows:

**Developers:** This group will use the document to understand the technical requirements specifications and how they will be transformed into a system design when the final software application is written.

**Testers:** This group will use the document during unit testing and system integration testing to understand all system features and how the system responds to external stimuli.

**End users:** System end users will be more interested in the overall description of the Greenhouse dashboard, which will help them understand the final product that the developers and testers will deliver and be able to propose other requirements that the development team might miss.

This document is primarily for developers looking into expanding the project and for technical users to understand the various functions and features of the GUI.

## Overview

This document is organized by key development steps. It describes the system architecture, from sensor registration on The Things Network (TTN), webhook configuration via Ngrok, to backend processing and frontend integration. It also contains testing procedures, known issues, and recommendations for future work.

## Project Scope

This project focuses on developing a web interface to visualize environmental data collected from a connected greenhouse using LoRa sensors. Unlike the initial version, which aimed to implement a full digital twin, my work was primarily centered on designing and deploying a functional dashboard to make sensor data easily accessible and understandable.

At the beginning of my internship, I received the code developed by a former intern, but it only included a login page with authentication. Since there was no existing interface for data visualization, I decided to start from scratch and build a new dashboard page from the ground up, displaying real-time data (temperature, humidity, luminosity, CO₂, etc.) through interactive charts.

In parallel, I designed a secondary “insight” page tailored to non-technical users working directly with the greenhouse. This page provides simplified interpretations of the data through automated summaries and alert messages, helping users make quick, informed decisions.

The scope of my work includes:

* Partial reuse of the previous project,
* The development of a complete and functional user interface,
* Real-time visualization of sensor data,
* And the creation of an additional insight-focused view for non-technical users.

This approach emphasizes reliability, accessibility, and scalability, with the aim of equipping greenhouse staff with a practical tool—without building a full-scale digital twin system.

To achieve this, the project leveraged existing sensors already installed in and around the greenhouse to collect environmental data such as temperature, humidity, luminosity, soil moisture, and CO₂ levels. These sensors communicated via LoRa through The Things Network (TTN), with the data being transmitted to a backend and stored in a structured PostgreSQL database.

My main contribution was focused on the development of a new front-end application that allows users to visualize the sensor data in real-time. This included building an interactive dashboard from scratch, as well as designing a secondary “insight” page for non-technical users, featuring automated summaries and simple alerts. The goal was to make the collected data easily accessible, readable, and actionable.

The system was tested and validated locally, ensuring stable data flow from sensors to backend to frontend. Once validated, the local version is expected to be deployed on the NUC server for production use.

Unlike the initial scope of developing a full digital twin, this version prioritizes functionality, clarity, and usability for the greenhouse staff. The project lays a solid foundation for future improvements, such as advanced data analysis or predictive features.

## Terminology

* **Graphical User Interface (GUI):** A type of interface that allows users to interact with a system using visual elements such as buttons, graphs, menus, and charts, rather than text-based commands. In this project, the GUI was built using React, a JavaScript library for building modern and dynamic web interfaces.
* **API (Application Programming Interface):** A set of rules and endpoints that allow different components of the application (frontend and backend) to communicate with each other. The APIs in this project were developed using Flask (Python) to serve sensor data to the React frontend.
* **Backend:** The part of the application that handles data processing, logic, and communication with external systems. In this project, the backend was developed using Flask and includes routes to receive, process, and send sensor data.
* **Frontend:** The visual and interactive part of the application that users interact with. The frontend in this project was created from scratch using React, and includes pages such as the dashboard and insights interface.
* **TTN (The Things Network):** A global LoRaWAN network that allows low-power devices to send data over long distances. It was used in this project to transmit sensor data from the greenhouse to the backend server.
* **LoRa / LoRaWAN:** A wireless communication protocol designed for long-range, low-power data transmission. The greenhouse sensors use this protocol to send environmental data to the TTN.
* **Ngrok:** A tool that creates secure tunnels to a local server, allowing it to be accessible from the Internet. It was used during development to expose the Flask backend to TTN without needing router access.
* **PostgreSQL:** An open-source relational database management system used to store and organize the data received from the sensors.
* **JWT (JSON Web Token):** A secure way to transmit user authentication information between frontend and backend. It was used to protect access to certain parts of the application, like the dashboard.

# 1 Project Management

## 1.1 Define Scope and Objectives – This is a Work Package

This work package focused on defining the goals, expected outcomes, and boundaries of the project, as well as identifying key milestones and deliverables.

The following are the list of tasks that need to be completed

•Task 01 – Analyze and understand the previous project and its limitations  
•Task 02 – Define the technical scope for a new frontend interface and data handling pipeline  
•Task 03 – Plan and schedule development tasks for the redesigned solution

### 1.1.1 Task 01

At the beginning of the internship, I received the code and documentation produced by a previous student. However, only the login and authentication page was accessible, and many parts of the system were either non-functional or poorly documented. I took time to understand what was reusable and what needed to be rebuilt.

### 1.1.2 Task 02

Given the limitations of the previous implementation, I decided to rebuild a new dashboard interface using React, and to progressively integrate the Flask backend and sensor data flow via TTN. The objective was to have a stable and modular system where environmental data could be visualized clearly and interactively.

### 1.1.3 Task 03

I organized the work in weekly objectives, starting with local setup and testing (using mock data), then implementing real-time data integration, building the dashboard layout, and finally planning the transition to server deployment. A Gantt chart was created to track these steps and identify priorities.

## 1.2 Identify stakeholders and their roles

### 1.2.1 Task 01- Intern

In charge of designing and developing the frontend dashboard, adapting the backend to receive data from TTN, creating API endpoints, and ensuring end-to-end testing of the system. Also responsible for documenting the whole process in this technical report.

### 1.2.2 Task 02- Supervisor

Provided guidance and validation throughout the development process, helped resolve infrastructure-related issues (e.g., NAS setup, Docker containers), and ensured that the work aligned with the broader research project on the greenhouse.

### 1.2.3 Task 03- Future Developers

Will use this report and the GitHub repository to understand the implemented structure, reuse existing components, or further extend the system (e.g., adding data insights, connecting additional sensors, deploying the project on the NUC, or integrating predictive analytics).

# 2 Requirements Gathering & Analysis

This section delineates the requirements that were identified during stakeholder engagement, particularly within the technical team that consists of experts in data modelling, back-end development, and software platform integration. The primary goal of this phase is to establish the content as well the scalability and extensibility of the graphical user interface (GUI) that will be presented to the end-users.

## 2.1 Identify sensor data sources

To successfully implement the sensor-to-front-end project, the first crucial step is to identify sensor data sources. This involves thoroughly assessing the various data points required to create a comprehensive digital twin of the greenhouse. Key sensor sources include temperature sensors to monitor ambient and soil temperatures, humidity sensors to track moisture levels, light intensity sensors to measure illumination, soil moisture sensors to assess the water content in the soil, and CO2 sensors to monitor carbon dioxide levels. Additionally, other relevant data sources, such as weather APIs for external weather data or connected devices for irrigation and ventilation systems, might be considered for integration. The identification of these sensor data sources lays the foundation for data collection, processing, and analysis, enabling the system to provide real-time and accurate insights into the greenhouse environment, empowering users to make informed decisions and optimize greenhouse operations effectively.

### 2.1.1 Sensor Research and Selection

I did not participate in the hardware selection phase. The sensors were already chosen and deployed before the beginning of my internship. My task was to work with the data already available through TTN and integrate it into a usable dashboard.

### 2.1.3 Integration Planning

At the beginning of the project, it was discovered that the batteries of all environmental sensors were depleted, preventing the collection of real-time data. To resolve this, all sensor batteries were replaced. Once operational, the sensors were carefully installed in and around the greenhouse, according to their respective measurement needs (e.g., soil moisture sensors placed in plant beds, CO₂ sensors positioned near air vents, etc.). This integration step was essential to restore communication with the sensors and validate the end-to-end data transmission through the LoRaWAN gateway and TTN.

<https://www.seeedstudio.com/Wireless-Smart-Agriculture-Kit-Greenhouses-p-4951.html>

### 2.1.4 Data Interfacing and Compatibility

Sensor data was received via TTN using webhooks and transmitted in JSON payloads. I ensured compatibility by decoding and parsing these payloads in the backend, validating the schema, and storing the values into a PostgreSQL database. Compatibility testing was essential to ensure that the format of each data type (temperature, humidity, etc.) matched the database structure and the frontend display logic.

## 2.2 Gather functional requirements

Gathering functional requirements is a vital phase in our project management plan to develop a sensor-to-front-end interface for visualizing greenhouse sensor data. During this stage, we will engage with stakeholders, including greenhouse operators, agronomists, and end-users, to elicit and document the specific functionalities and features they expect from the system. This process involves identifying the key user interactions, data visualization requirements, and performance expectations. Additionally, we will define the desired data processing capabilities, such as real-time data updates, historical data analysis, and predictive insights. By meticulously gathering these functional requirements, we will ensure that the resulting front-end interface meets the needs of our users and aligns with the project's objectives, fostering an intuitive and user-friendly experience to effectively leverage the potential of greenhouse sensor data for informed decision-making and enhanced agricultural practices.

### 2.2.1 User Interviews and Surveys

No formal interviews or surveys were conducted with end users during the project. Functional expectations were derived from discussions with the project supervisor, analysis of previous project deliverables, and general best practices in IoT dashboard design.

### 2.2.2 Use Case Definition

The main use cases identified included:

* A user opens the dashboard and views the latest sensor readings.
* A user selects a time range (day/week/month) to visualize historical trends.
* A user accesses the "Insights" page to quickly interpret summarized sensor trends or alerts.

Each of these use cases informed the frontend structure and API endpoints needed for data retrieval.

### 2.2.3 Requirements Workshops

No dedicated workshops were held. However, informal weekly discussions with the project supervisor helped validate development progress and refine the direction of the interface.

### 2.2.4 Requirement Documentation

The requirements were documented progressively within development files (code comments, README), and formalized in this report for future contributors. These include user goals (data visualization, real-time updates), interface elements, and API interactions.

**\*\*\***Specific ways to make the graphical user interface (GUI) more intuitive for the end-users:

* Ensure a consistent design pattern throughout the application, including color schemes, typography, and layout.
* Simplify navigation by using clear and descriptive labels for menu items and buttons.
* Provide tooltips and help sections to assist users in understanding features and functionalities of the GUI.
* Use visual aids such as icons and images to convey information quickly and effectively.
* Conduct user testing and gather feedback throughout the development process to identify areas for improvement and enhance the overall user experience.
* Prioritize the most important information and actions to be displayed prominently, while minimizing clutter and unnecessary details.
* Incorporate user-friendly error messages that clearly explain the problem and provide guidance on how to resolve it.
* Consider the context and goals of the end-users to ensure that the GUI is optimized for their specific needs.
* Continuously evaluate the effectiveness of the GUI and adjust as needed to improve the user experience.

## 2.3 Gather non-functional requirements

### 2.3.1 Task

## 2.4 Analyse and prioritize requirements

# 3. Design & Architecture

## 3.1 Define data schema and storage requirements

The data architecture of the Greenhouse Monitoring Platform is based on a relational database using MariaDB, chosen for its reliability, scalability, and ease of integration within a Dockerized environment. This database stores all key components of the system: sensor measurements, user credentials, and greenhouse management tasks. The design was structured to support real-time access, historical data queries, and future extensibility.

The schema consists of several main entities. A Users table stores login credentials and profile information, using hashed passwords and JWT-based authentication to ensure security and access control. Sensor measurements are organized by type, with each sensor (e.g., soil moisture, temperature, CO₂, light, etc.) having its own table containing timestamped values. Each entry includes an id (auto-incremented primary key), a datetime field, and one or more sensor-specific value fields such as valueSM, valueTemp, valueEC, or valueCO2.

For greenhouse task management, a Tasks table allows users to create and organize daily operations such as watering or ventilation. This table includes fields for the task’s title, description, date, time, priority level, category, color tag, and a foreign key linking the task to a specific user.

All data is persisted in a Docker volume (db-data) to ensure durability and ease of backup or restoration. The system is built with modularity in mind, allowing new sensor types to be added either through additional tables or by extending existing structures. Data access is provided through RESTful API endpoints, enabling frontend components to query measurements (with filters by type or date) and interact with the task management system.

This architecture ensures a robust, secure, and scalable backend, capable of handling both current needs and future extensions for the greenhouse monitoring system.

## 3.2 Determine the technologies and frameworks for the frontend

The frontend of the Greenhouse Monitoring Platform has been developed using a modern and efficient web stack designed for responsiveness, scalability, and maintainability. Although the project has not yet been deployed to a production server, the development environment is fully functional and prepared for future deployment.

The technologies and frameworks used include:

* **React**: The core JavaScript library for building the user interface. React’s component-based architecture and efficient state management were essential in creating a dynamic dashboard capable of displaying real-time environmental data.
* **Vite**: A fast build tool and development server that enables instant hot module replacement (HMR) and optimized production builds. It was used to scaffold and manage the entire React project during development.
* **React Router**: Implemented for client-side routing, allowing smooth transitions between pages such as the login page, dashboard, homepage, and the insights section.
* **Highcharts & Highcharts React**: These libraries were integrated for rendering interactive data visualizations (e.g., time series charts, gauges) to help users easily interpret sensor trends.
* **Tailwind CSS (via CDN) and custom CSS**: Used to style the interface quickly and efficiently. Tailwind provides utility-first classes for responsive design and ensures consistency across all UI components.
* **ESLint**: Integrated to maintain code quality and enforce consistent style across the codebase using recommended React and JavaScript rules.
* **Nginx** *(planned for production deployment)*: A lightweight web server considered for serving the frontend build within a Docker container when the application is ready for deployment.

This technology stack was selected for its excellent developer experience, active community support, and adaptability for future extensions such as user role management or advanced analytics modules.

## 3.3 Design the front-end user interface

This work package focuses on the design and development of the envisioned software application. There are multiple elements that considered important for the user to have a seamless experience with the

### 3.2.1 Camera System

Using the package

1. **Installing Cinemachine -** Use the Unity Package Manager (in the top menu: Window > Package Manager) to select Cinemachine for installation. If you don't see the package, look for a dropdown menu above the list and select “All packages”. Cinemachine is free and available for any project. If you already have it installed, you may update to the latest version. See link - <https://unity.com/unity/features/editor/art-and-design/cinemachine>
2. Once Cinemachine is imported into the scene, you can create a Virtual Camera game object.
3. The Virtual Camera can be set to ‘Look at’ and ‘Follow’ an object
4. Create a script that manages the switching between the views/Virtual Cameras

### 3.2.2 Tooltip System

* Designing a tooltip system to draw and display tooltips.
* Tooltip trigger component to handle interaction
* Dynamically size tooltip based on contents

1. Setup a separate canvas (Tooltip Canvas) in order for the panels to be on top of anything in the UI.
2. Set the sort order to the highest value to ensure it’ll render on top.
3. Add Canvas Scaler component and set the UI Scale Mode to ‘Scale With Screen Size’ with the reference resolution as 1920 x 1080.
4. Add an Image child game object which will contain the two TextMeshPro components. Adding a Vertical Layout Group with Control Child Size enabled.
5. Add Content Size Fitter component with Horizontal and Vertical Fit as ‘Preferred Size’
6. Add a Layout Element component with a Preferred Width of 500 to control the size of the text length.

## 3.4 Design the back-end architecture for data processing

The back-end is built using the Python Flask framework to create the APIs and MySQL for the database.

Python Flask is a micro web framework for building web applications in Python. It is designed to be lightweight and simple, providing the essentials needed to create web applications while allowing developers the flexibility to choose their tools and libraries for other components. <https://flask.palletsprojects.com/en/2.3.x/>

The Flask program is built as follows:

- “app.py” which is used to run the application and import the other files

- “auth.py” is the Python script that contains the ‘User’ class and the endpoint, “/api/users” which is used to create an account by posting a JSON with a ‘username’ and a ‘password’. This file also has the endpoint “/api/login” which is used to log in to an account by sending the same JSON than the account creation endpoint.

- There is also the “sensors” folder in the project tree. It contains all the sensors files.

- Each of the files inside the sensors folder are used to create several endpoints to access the sensor values that are stored in the database.

The following endpoints are contained in “sensorLightExt.py” which are as follows:

* “/api/sensors/sensor\_light\_ext/latest” – this is the endpoint to retrieve the latest value.
* “/api/sensors/sensor\_light\_ext/day/pic-average” – this endpoint returns the maximum light intensity during the day, the maximum light intensity during the night, and the average of these two values for the current day.
* ”/api/sensors/sensor\_light\_int/day/average” – this is the endpoint to retrieve the data of the last 24 hours, each hourly value is the average of the ten-minute interval.
* “/api/sensors/sensor\_light\_int/day” - this is the endpoint to retrieve all the data of the last 24 hours.
* “/api/sensors/sensor\_light\_ext/week” – this is the endpoint to retrieve the data of the last seven days. A list of the average of each hour over the last seven days is returned.
* “/api/sensors/sensor\_light\_ext/month” – this is the endpoint to retrieve the data of the last 30 days. Each day is divided into six 4-hour periods which are averaged.

This program uses an ORM (SQLAlchemy), which allows database tables to be created directly from classes created in Python.

The databases tables are simple. For the User table, there is an ID which is autoincrementing and which is the primary key, there is also a username and a password\_hash column. The password in encrypted with “werkzeug.security”.

The sensors tables are simple too. They have got an ID which is also autoincrementing and which is also the primary key, there is one or more value column depending of the sensor and a datetime column.

## 3.5 Plan data integration and APIs between sensors and the front-end

Sensors use LoRa to communicate. LoRa (Long Range) is a wireless technology for the Internet of Things (IoT), offering long-range coverage and low power consumption.

Sensors therefore transmit data frames containing information such as values, time, etc.

This data is received by a gateway, which then sends it to the Internet, in our case to TTN (The Things Network).

The Things Network registers the gateway as well as the various sensors.

The data is then sent via a webhook to the Flask backend application, where it is processed.

## 3.6 Server deployment

The backend (Flask and MariaDB) is hosted on a NUC Dell EMC Edge Gateway 3200.

Docker is used to deploy these services.

Docker is a lightweight virtualization technology for running portable, isolated applications in container form.

Since the NUC isn't just used to host this project, Docker makes it possible to keep each project running on the NUC separate.

**How to use the NUC :**

An ethernet cable must run from the NUC's first ethernet port to an internet router and a second cable from the NUC's 2nd ethernet port to the NAS's first ethernet port (LAN1).

Login : server

Password : 1MTUr3s34rch4DM1N ( same password for admin)

All the projects are in home/server/docker-containers/

Files shared with the NAS are in home/server/nas and home/server/bathorse

Docker commands :

* docker ps -> used to see what container is running
* docker-compose build -> build the project
* docker-compose up -d -> run the containers
* docker-compose down -> turn off the containers

docker ps can be used anywhere

All the docker-compose commands must be used in a directory which contains a docker-compose.yml file

**Ports used :**

|  |  |
| --- | --- |
| **App** | **Port** |
| Greenhouse frontend | 8079/TCP |
| Greenhouse backend | 8080/TCP |
| Projects page | 8085/TCP |
| VIB backend | 8084/TCP |
| Bathorse | 3005/TCP |
| BathorseUDP | 5550/UDP & 5555/UDP |

# 4 Sensor Data Acquisition

## 4.1 Procure necessary sensors and hardware

This project uses 8 LoRa sensors from Seeed :

* 2 Light intensity sensors
* 2 CO2, Temperature, Humidity sensors
* 4 Soil Moisture, Temperature, electro conductivity sensors

## 4.2 Develop or configure data collection software

The gateway sends sensor data to the TTN.

**Sensors Configuration**

To receive data from LoRaWAN sensors, each device must first be registered on **The Things Network (TTN)**.

**Steps to Register a Sensor on TTN:**

1. **Log in** to The Things Stack Console.
2. Navigate to your **application** or create a new one if needed.
3. In the application dashboard, click **"Add end device"**.
4. Choose the **manual option** to register the device manually.
5. Enter the following information:
   * **End device ID**: a unique name you choose for the sensor (e.g., light-sensor-ext1)
   * **Join EUI** (also known as App EUI): provided by the manufacturer (Seeed Studio)
   * **Dev EUI**: unique identifier of the sensor, printed on the device or provided by Seeed Studio
   * **App Key**: also provided by the manufacturer, used for OTAA activation
6. Select the **LoRaWAN version** and **regional parameters** (e.g., LoRaWAN 1.0.3, EU868 for Europe).
7. Finalize the registration and **save** the device.

Once registered, the sensor will be able to join the network via **Over-The-Air Activation (OTAA)**, and its messages will appear in the **Live Data** tab under the application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **ID** | **AppEUI** | **DevEUI** | **AppKey** |
| Light intensity out | eui-2cf7f1c04430094f | 11E62CA65AAD4B92 | 2CF7F1C04430094F | C48EDA4F132DC08FB6C739955B058E41 |
| Light intensity in | eui-2cf7f1c044300975 | 263F0C0CC52FAB2B | 2CF7F1C044300975 | 1B7C61DCE711FDD645D2AC9C3E444A59 |
| CO2, Temp, Hum out | eui-2cf7f1c044300436 | D87673FE226C96F3 | 2CF7F1C044300436 | B82D2A732905A8C7A75B7F3C2B635C78 |
| CO2, Temp, Hum in | eui-2cf7f1c0443004b1 | 8ABB1B9C9273EBF1 | 2CF7F1C0443004B1 | BE41B363444195613265D046DEF22A6C |
| Soil Moisture, Temp, EC 1 | eui-2cf7f1c0435006c8 | 8BAC255955815EB5 | 2CF7F1C0435006C8 | EC575A8F8893807DE07CD3CAE968CB7B |
| Soil Moisture, Temp, EC 2 | eui-2cf7f1c043500707 | 5E9C5CECCBDB86C4 | 2CF7F1C043500707 | E3EAA1A54F2A33E28618A49B814D3DD0 |
| Soil Moisture, Temp, EC 3 | eui-2cf7f1c043500681 | FAC375674CD1FD1C | 2CF7F1C043500681 | 2279F5260EA7B55D55F2C46599DAEBA9 |
| Soil Moisture, Temp, EC 4 | eui-2cf7f1c0435005e6 | C645419B93B20198 | 2CF7F1C0435005E6 | 34D1BF4CAF20FC39FF0ABD6254E5D0F5 |

**Webhook configuration :**

One of the key steps in ensuring that sensor data is transmitted to the backend is the configuration of a webhook on The Things Network (TTN). This webhook automatically forwards decoded LoRaWAN messages from TTN to the server responsible for processing and storing them (our Flask backend).

**Webhook Objective**

The goal is to send the sensor data received by TTN to a publicly accessible URL that the Flask backend listens to. In a production environment, this URL typically points to a public IP address or domain name associated with a server.

**Issue Encountered**

In my case, I did not have access to the router’s administration interface, which prevented me from configuring port forwarding to my local machine. As a result, TTN could not reach my backend directly.

**Solution: Using Ngrok**

To bypass this network limitation, I used Ngrok, a tool that creates a secure tunnel between a local port and the internet. Ngrok generates a temporary public URL (e.g., https://xxxx.ngrok.io) that forwards requests to a local server (e.g., http://localhost:8080).

**Configuration Steps Using Ngrok:**

1. Start the Flask backend locally (port 8080):
2. python app.py
3. Launch a tunnel using Ngrok:
4. ./ngrok http 8080
5. Ngrok will generate a public URL, such as:
6. https://xxxx.ngrok.io
7. In the TTN console, go to Webhooks > Add Custom Webhook.
8. Enter the Ngrok public URL followed by /api/ttn as the base URL:
9. https://xxxx.ngrok.io/api/ttn
10. Ensure that the Flask backend has a corresponding route, for example:
11. @app.route('/api/ttn', methods=['POST'])
12. Send test data from TTN and verify that the backend receives and processes the messages correctly.

**Important Notes:**

* Ngrok URLs expire each time it restarts unless you use a paid plan with a reserved domain.
* You’ll need to update the webhook URL in TTN every time Ngrok is restarted during development.

**Une image contenant texte, capture d’écran, logiciel, nombre

Le contenu généré par l’IA peut être incorrect.**

**Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement**

## 4.3 Test data acquisition system and validate sensor data

**Step 1 – Data Reception and Decoding**

The gateway forwarded the sensor payloads to TTN, which then relayed them via a webhook to my Flask backend. Using the ttn.py script, I decoded the incoming payloads based on the device ID and ensured proper parsing of each data field. I also checked the timestamp formatting and the insertion process into the appropriate database tables.

**Step 2 – Real-Time Visualization**

With the full pipeline in place — from sensors to TTN, to the backend, to the frontend — I was able to visualize real-time data on the dashboard. Values were displayed instantly under each section (Luminosity, Environment, Soil), and the data graphs updated correctly according to the selected period filters (Day, Week, Month).

**Validation Result**

The successful real-time visualization of sensor data validated the entire architecture. I checked:

* The accuracy and completeness of received payloads.
* The stability of the database insertions.
* The responsiveness and clarity of the web interface.
* The correct operation of the filtering system per sensor and time period.

This validation confirmed that the system is ready for deployment and future extension, once all sensors are installed around the greenhouse and fully operational.

# 5 Data Processing and Storage