UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Toomas Hendrik Raaga

Developing a Mesh Networked IoT System for Plant Health Monitoring

Bachelor's Thesis (9 ECTS)

Supervisors:
Ulrich Norbisrath
Matevž B. Zorec

Tartu 2025

Contents

Developing a Mesh Networked IoT System for Plant Health Monitoring

Abstract:

This thesis presents Smart Garden, a mesh-networked IoT system that addresses the challenge of determining the right time to water indoor plants. Unlike existing solutions that operate as standalone units, Smart Garden employs sensors in a mesh network, enabling monitoring in areas with poor coverage. The system continuously tracks soil moisture via capacitive sensors, transmits data through a mesh network, and notifies users through a web interface when plants need watering. Testing demonstrated 96% reliability over three months and successful communication through concrete barriers. Evaluation confirmed that the system effectively transforms the question of ”When do I water my plants?” into actionable notifications based on actual soil moisture data.

Keywords: IoT, mesh networking, plant monitoring, soil moisture, ESP32, network topology, indoor gardening

CERCS: T120 Systems engineering, computer technology. P175 Informatics, systems theory

Võrgustatud IoT-süsteemi arendamine taimede tervise jälgimiseks

Lühikokkuvõte:

Käesolev lõputöö tutvustab *Smart Garden'i* nimelist võrgustatud IoT-süsteemi, mis lahendab toataimede õige kastmisaja leidmise probleemi. Erinevalt olemasolevatest lahendustest, mis toimivad eraldiseisvate seadmetena, kasutab *Smart Garden* andureid silmusvõrgus, võimaldades taimede jälgimist ka halva levialaga piirkondades. Süsteem jälgib mahtuvuslike andurite abil mulla niiskust, edastab andmeid läbi silmusvõrgu ning teavitab kasutajaid läbi veebirakenduse, kui taimed vajavad kastmist. Peale kolmekuist testimisperioodi saime teada, et süsteem toimib 96% töökindlusega ning on võimeline edukalt edastama andmeid läbi betoonseina. Hindamine kinnitas, et süsteem muudab edukalt küsimuse "Millal pean oma taimi kastma?" kasulikeks märguanneteks, kasutades mullaniiskuse andmeid.

Võtmesõnad: IoT, silmusvõrk, taimede jälgimine, mullaniiskus, ESP32, võrgutopoloogia, toataimede hooldus

CERCS: T120 Süsteemitehnoloogia, arvutitehnoloogia. P175 Informaatika, süsteemiteooria

1. Introduction

Research on indoor plants has shown a steady increase in publications over a 25-year period, with studies documenting their positive effects on human physiological responses and cognitive functions [1]. As more people incorporate plants into their indoor environments to gain these benefits, they consistently face one fundamental challenge: determining the optimal time to water their plants. This seemingly simple question remains surprisingly difficult to answer with precision. Most plant owners rely on inconsistent manual methods, such as finger tests or visual cues, which often lead to either over-watering or under-watering. Both scenarios result in stressed plants, reduced health benefits, wasted resources, and frustrated owners who may eventually abandon plant care altogether.

While existing solutions exist to address this problem, they have significant limitations. Basic timers fail to account for environmental factors like humidity, temperature, and seasonal changes that affect a plant's water needs. Many current monitoring systems are constrained by their design and connectivity requirements. A common limitation is the requirement for direct WiFi connectivity for each sensor, which is problematic as WiFi signals often cannot reach all areas where plants are located in a home. Additionally, these solutions generally don't provide the ability to configure each sensor independently, such as setting different measurement intervals based on individual plant needs. These limitations can lead to monitoring that doesn't adequately address the specific requirements of different plants.

The modern lifestyle compounds these challenges. People often maintain busy schedules with irregular hours and frequent travel, making consistent plant care difficult. Additionally, even when people have basic knowledge about plant care, they may feel uncertain about when exactly to water their different plants, as each species has varying requirements. These factors create a need for a technological solution that provides clear guidance on plant watering timing.

This thesis presents Smart Garden, a distributed plant monitoring system designed to directly address the question of when to water plants through continuous, automated soil moisture tracking. The system employs a mesh network of ESP32 microcontrollers that enables monitoring across home environments without requiring direct WiFi connectivity for each plant sensor. This mesh approach allows sensors to communicate even when placed in areas with poor WiFi coverage, such as balconies or corners with limited signal strength. In the current prototype, each sensor node transmits soil moisture data through the network, which is then processed, stored, and

presented to users through an intuitive web interface. When moisture levels fall below defined thresholds, the system notifies users, providing clear guidance on when watering is needed.

Beyond addressing individual plant care challenges, the system contributes to environmental sustainability by improving plant survival rates. This directly reduces the environmental impact associated with disposing of dead plants and purchasing replacements. Many plant owners cycle through multiple plants before finding success, with each discarded plant contributing to landfill waste and resource consumption for replacement plants.

From a technical perspective, the implementation combines networked sensor nodes, distributed message routing, and modern web technologies to create a cohesive ecosystem for plant monitoring. The mesh network configuration offers advantages in coverage and reliability over traditional hub-and-spoke IoT architectures, particularly in home environments. The system's web application interface provides a seamless experience across devices while supporting push notifications.

Through Smart Garden, this thesis demonstrates how appropriately applied technology can solve an everyday problem that affects millions of plant owners worldwide, transforming the ambiguous question of ”When do I have to water my plants?” into a clear and actionable notification.

For text formatting and language refinement, Claude 3.7 Sonnet was used. Though capable of content generation, the AI was employed strictly for enhancing readability and grammatical structure of the original text.

2. Background and related work

2.1 Plant monitoring technologies

Indoor plant care presents a unique set of challenges for plant owners. Unlike outdoor plants that benefit from natural rain cycles, indoor plants rely entirely on human intervention for their water needs. This dependency creates a fundamental problem: determining the optimal watering schedule for each plant.

Traditional methods of determining watering needs include manual techniques such as inserting a finger into the soil to check moisture, visual inspection of soil color, or observing the plant for signs of wilting or leaf changes. These approaches have significant limitations: they are subjective, inconsistent, and often detect water needs only after the plant has already begun to experience stress. More importantly, these methods require regular attention that conflicts with busy lifestyles.

2.1.1 Resistive soil moisture sensors

Resistive soil moisture sensors measure electrical conductivity between two probes in soil. They work because water improves conductivity, a wetter soil allows more current to flow, lowering resistance. These low-cost sensors are straightforward, but may corrode with extended use. Their accuracy is generally adequate for basic monitoring but can be affected by soil salinity and temperature variations. [2]

2.1.2 Capacitive soil moisture sensors

Capacitive soil moisture sensors measure the dielectric properties of the soil, which change with water content. They use insulated electrodes that form a capacitor with soil as the dielectric medium. While also low-cost, these sensors typically last longer than resistive ones, since their probes do not make direct contact with the soil. Capacitive sensors generally provide more consistent readings across repeated measurements, although both types of sensors require calibration to convert voltage outputs to meaningful soil moisture values for agricultural applications. [2]

2.1.3 Professional soil moisture sensors

Laboratory grade soil moisture sensors include time-domain reflectometry (TDR), frequency-domain reflectometry (FDR), neutron probes, and gamma-ray attenuation devices. These professional instruments offer superior accuracy, stability, and reliability compared to consumer sensors, though at considerably higher cost. Due to this high cost, these instruments are rarely

used by regular consumers and are mainly found in research, agriculture, and industrial settings. [3]

2.2 Wireless sensor networks

Effective soil moisture monitoring requires reliable data collection and transmission. Microcontrollers interface with sensors via protocols like UART, I2C, or BLE, but since each sensor monitors only one plant, multiple plants require networked sensors. While wired connections are reliable, they introduce installation complexity and aesthetic concerns in residential settings. Wireless Sensor Networks (WSNs), communication systems designed to monitor and collect data from various environments, offer an efficient alternative without physical cabling limitations [4]. The topology of a WSN fundamentally determines how sensor nodes interact, communicate, and route information.

2.2.1 Star topology

In a star topology, all sensor nodes communicate directly with a central node or base station. This configuration is simple and straightforward, with each node having a single direct communication path. It's efficient for smaller networks but can become a bottleneck if the central node fails. [5]

2.2.2 Tree topology

Tree topology creates a hierarchical network structure where nodes are organized in a parent-child relationship. Data flows from child nodes to parent nodes, eventually reaching the root node or base station. This topology allows for efficient data aggregation and can cover larger areas than a star topology. [5]

2.2.3 Peer-to-peer topology

Peer-to-peer topology enables direct communication between sensor nodes without a strict hierarchical structure. Nodes can communicate and route data through multiple paths, providing more flexibility and resilience. This approach allows for more complex and adaptive network communications. [6]

2.2.4 Mesh topology

Mesh topology offers the most robust network configuration. Nodes can communicate with multiple other nodes and dynamically route data through different paths. If one node fails, the network can automatically reroute communications through alternative paths. This topology

provides the highest level of network reliability and coverage, making it ideal for complex monitoring environments. [6]

2.2.5 Communication protocols

Bluetooth provides short-range communication with moderate data transfer rates, but suffers from high power consumption, limiting its effectiveness in sensor networks. [7]

Wi-Fi provides high-speed data transfer but consumes significant power, rendering it less practical for most wireless sensor network applications requiring long-term, battery-powered monitoring. [8]

ZigBee offers low-power, long-battery-life communication with modest data transfer rates, making it ideal for wireless sensor networks that require extended, energy-efficient operation. [7]

2.3 Commercial solutions

Feature	**Xiaomi MiFlora**	**Flora Pod**	**Earthone**	**Gardy**	**OKO+**
Connectivity	BLE	WiFi	WiFi	BLE	WiFi
Power source	CR2032	Built-in	Built-in	Built-in	Built-in
Battery life	Not specified	∼3 months	∼4 months	Not specified	∼3 months
Network type	Standalone	Standalone	Standalone	Standalone	Standalone
Mobile app	Yes	Yes	Yes	Yes	Yes
Measurements	Soil-moisture, temperature, light, nutrients	Soil-moisture, temperature, light, humidity	Soil-moisture, temperature, light	Soil-moisture, temperature, light	Soil-moisture, temperature, pH, humidity

Table 1. Comparison of commercial plant monitoring solutions

The commercial plant monitoring solutions share similar core functionality but differ in sensor capabilities and connectivity methods. All operate as standalone units requiring direct connections to a smartphone or home network rather than functioning within a mesh network architecture.

Connectivity approaches split between Bluetooth Low Energy (MiFlora[1] and Gardy[2]) and WiFi (Flora Pod[3], Earthone[4], and OKO+[5]). BLE solutions typically offer lower power consumption but more limited range, while WiFi-based sensors provide better range at the potential cost of higher energy usage [9]. This tradeoff is reflected in the battery configurations, with most WiFi devices using built-in rechargeable batteries designed to last 3-4 months between charges.

The OKO+ offers the most comprehensive monitoring by including pH sensing and air humidity, while the MiFlora uniquely provides nutrient level monitoring through its conductivity sensor. The Flora Pod includes humidity sensing alongside the standard measurements.

These commercial options, while convenient for basic plant monitoring, all share a fundamental limitation: they require direct wireless connectivity to a central hub, router, or smartphone. This restricts their placement options and makes comprehensive coverage in larger homes or challenging environments difficult.

2.4 Open-source solutions

Unlike commercial products that operate as standalone units, open-source plant monitoring systems offer greater customization possibilities and often address the networking limitations found in commercial solutions.

2.4.1 In-plants

In-plants, developed by Nick Engmann, leverages Particle Mesh technology to create a network where only one Argon device connects directly to WiFi while other Xenon nodes communicate through the mesh network[6]. This architecture enables significant power savings as sensor nodes can enter sleep states between measurements. The system uses capacitive soil moisture sensors in 3D-printed enclosures, transmitting data to the Particle Cloud. While the Particle Mesh platform has been discontinued, the project demonstrates valuable principles for developing scalable plant monitoring networks.[7] [8]

---

[1] https://smarthomescene.com/reviews/xiaomi-miflora-plant-sensor-tuya-version-hhccjcy10-review/
[2] https://planteia.ca/products/smart-plant-care-sensor?shpxid=7b318922-bb81-4f64-842b-b5e4940103c3
[3] https://florasense.com/
[4] https://earthone.io/
[5] https://www.get-oko.com/products/oko-indoor-smart-sensor
[6] https://www.particle.io/blog/introducing-particles-next-generation-hardware-powered-by-particle-mesh/
[7] https://github.com/NickEngmann/in-plants
[8] https://www.particle.io/blog/learn-how-to-build-in-plants-a-mesh-connected-soil-monitoring-system/

2.4.2 OpenGrow

OpenGrow provides plant monitoring with autonomous maintenance capabilities, continuously tracking soil moisture and ambient light while offering automated watering through a PID control system[9]. The architecture combines a MERN stack web application[10], MQTT broker, and firmware running on STM32F4[11] and ESP8266 microcontrollers. It's distinctive feature is the integration of complete watering automation, setting it apart from monitoring-only systems.[12]

2.4.3 Chirp

Chirp takes a minimalist approach as a plant watering alarm that alerts users when soil becomes too dry. It employs capacitive humidity sensing to avoid electrode corrosion and soil electrolysis issues common in resistive sensors. Built around an ATTINY44A microcontroller[13] with I2C capabilities, Chirp can operate for up to a year on a single CR2032 battery. The device includes adaptive features like restricting alerts during nighttime through ambient light detection and adjustable alert thresholds for different plant species.[14]

2.4.4 MeshGarden

MeshGarden implements an ESP32/ESP8266-based mesh network where only a single bridge node requires WiFi connectivity while supporting numerous measurement nodes throughout the garden. Data collected from sensors transfers through the mesh to the bridge node, then to a Firestore[15] server, with user interaction via a Flutter[16] mobile application. The system supports various built-in sensors (temperature, humidity, soil moisture, UV) and allows custom sensor integration. Power-saving features include deep sleep cycles for battery-operated nodes, making it suitable for distributed monitoring across larger spaces.[17]

---

[9] https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID
[10] https://www.oracle.com/in/database/mern-stack
[11] https://www.st.com/en/microcontrollers-microprocessors/stm32f4-series.html
[12] https://github.com/esyywar/OpenGrow-IoT-Plant-Monitor
[13] https://www.microchip.com/en-us/product/attiny44a
[14] https://wemakethings.net/chirp/
[15] https://cloud.google.com/firestore/native/docs
[16] https://flutter.dev/
[17] https://github.com/m-schwob/MeshGarden

3. System architecture

3.1 Overview

The Smart Garden system implements a distributed architecture designed to address the fundamental challenges of indoor plant monitoring. While there are IoT systems for home use that rely on direct WiFi connectivity for each device, Smart Garden employs a mesh network approach that enables sensors to function in locations with poor or no WiFi coverage. This distinguishes it from both WiFi-dependent systems and those using other wireless protocols like Zigbee, Z-Wave, or BLE. The system consists of four primary components that work together to form a cohesive plant monitoring solution.

At the foundation lies the mesh network layer, composed of ESP32-based nodes that collect and relay soil moisture data throughout the home environment. This mesh approach allows sensors to communicate with each other, extending the effective range of the network beyond what would be possible with direct WiFi connections alone. The gateway layer bridges this local mesh network with cloud services, handling protocol translation between ESP-NOW and MQTT. The backend layer provides services for data processing, storage, and notification management, while the frontend layer delivers intuitive user interfaces for monitoring and configuration.

The architecture is guided by four key design principles. Reliability is ensured through the distributed communication network that can route messages through multiple paths, eliminating single points of failure. Accessibility is achieved through user-friendly interfaces that make complex data understandable and actionable. The system's scalability allows it to grow from monitoring a few plants to dozens without architectural changes. Finally, extensibility enables the system to incorporate new sensors, metrics, or features in the future with minimal disruption to existing components.

The architecture employs two databases for different purposes: InfluxDB for time-series sensor data and SQLite for configuration storage. This separation minimizes writes to sensor node flash memory, which has limited write cycles, while also reducing the amount of data transmitted through the mesh network for improved performance.

Figure 1. System architecture diagram

3.2 Nodes

3.2.1 Sensor nodes

Sensor nodes serve as the data collection points in the Smart Garden system, directly interfacing with capacitive soil moisture sensors to obtain readings. Each sensor node is built around an ESP32 microcontroller that manages the sensor interface, data processing, and network communication. The sensor integration connects to the ESP32's analog input, with the system designed to transmit raw ADC values through the network.

The sensor nodes implement deep sleep capabilities between measurements to reduce power consumption. The current implementation uses standard ESP32 development boards with USB

power, with the architecture designed to accommodate potential future battery operation. Each sensor node maintains a unique identifier and configuration parameters in non-volatile storage, enabling personalized settings for different plants and preserving these settings across power cycles.

The data acquisition cycle follows a precise sequence: wake from sleep, initialize the sensor, take readings, process and package the data, transmit through the mesh network, and return to sleep. This sequence is controlled by configurable intervals that can be adjusted by users based on individual plant requirements, accommodating the diverse watering needs of different plants.

3.2.2 Relay nodes

Relay nodes play a critical role in extending the network's reach by receiving and rebroadcasting messages. These nodes implement a minimalist design focused purely on message forwarding, with no sensor integration or data storage responsibilities. The forwarding mechanism employs a simple but effective approach: when a message is received, it is immediately rebroadcasted.

Network coverage extension is achieved through strategic placement of relay nodes in locations that bridge gaps between sensor nodes and the gateway. The system does not require manual configuration of routing tables; instead, it employs a straightforward message propagation approach. This design creates a self-forming network that can adapt to changes in the environment and node placement.

3.2.3 Master node

The master node functions as a specialized bridge between the mesh network and the gateway, implementing protocol translation between ESP-NOW and UART. This node is responsible for aggregating messages from the sensor network and forwarding them to the gateway, as well as receiving configuration updates from the gateway and distributing them to the appropriate sensor nodes.

The protocol translation design involves careful handling of message structures to ensure data integrity across different communication protocols. The master node transforms the ESP-NOW wireless packets into serial data for the UART connection, preserving all necessary information including node identifiers, measurement values, and configuration parameters. This bidirectional translation enables seamless communication between the mesh network and the internet-connected gateway.

3.2.4 Gateway node

The gateway node connects the local mesh network to the broader internet, managing WiFi connectivity and implementing MQTT protocol support. This node receives sensor data from the master node via UART and publishes it to an MQTT broker, making it accessible to the backend services. Similarly, it subscribes to configuration topics on the MQTT broker and forwards updates to the master node.

The data publishing mechanism implements a JSON-based message format that encapsulates the raw sensor readings along with the sensor identifier and configuration version. Timestamps are not included in these messages, as they are automatically added by InfluxDB when the data is stored. The gateway manages WiFi connectivity to ensure reliable data transmission to the MQTT broker.

3.3 Network architecture

The Smart Garden system implements a mesh topology using the ESP-NOW[18] protocol, a lightweight communication protocol developed by Espressif. This topology allows for flexible node placement throughout a home environment without requiring direct connections to a central hub. The zh_network[19] library provides the underlying protocol management, handling low-level details of peer discovery, connection management, and message delivery.

Message routing through the mesh network follows a flooding approach, where messages propagate through all available paths until reaching their destination. This simple but robust strategy eliminates the need for complex routing tables and adapts automatically to changes in network topology. When a sensor node broadcasts a message, nearby nodes receive and rebroadcast it, extending the effective communication range beyond what would be possible with direct transmission alone.

Communication paths from sensors to the user interface traverse multiple protocols and technologies. Data originates at the sensor nodes and travels through the ESP-NOW mesh network to the master node. From there, it passes via UART to the gateway node, which publishes it to an MQTT broker. The backend services process and store this data, making it available to the frontend application via RESTful API calls. This multi-stage path enables

[18] https://www.espressif.com/en/solutions/low-power-solutions/esp-now
[19] https://github.com/aZholtikov/zh_network

efficient data transfer while accommodating the different requirements of each segment of the system.

Fault tolerance is achieved through redundant communication paths within the mesh network. If one node fails or is temporarily unavailable, messages can route around the obstruction through alternative paths. The system does not rely on any single node for network functionality, except for the gateway node which serves as the connection point to the internet. This currently represents a single point of failure in the system architecture, which could be addressed in future versions through redundant gateway implementations.

Network coverage optimization for home environments considers the typical challenges of residential settings, including walls, furniture, and electronic interference. The mesh approach allows for strategic placement of relay nodes to overcome these obstacles, and the system can be easily extended with additional relay nodes to improve coverage in problematic areas without requiring changes to existing nodes or software.

3.4 Data flow

The data flow in the Smart Garden system follows a carefully designed path from sensors to storage and presentation. Sensor readings originate at the ESP32 microcontrollers, which capture analog values from capacitive moisture sensors. These values are converted to standardized moisture percentages and packaged into structured messages that include node identification and version information. The messages travel through the mesh network to the gateway, which publishes them to specific MQTT topics.

Node-RED subscribes to these topics and handles the initial data processing, performing validation and transformation before writing directly to InfluxDB[20] for long-term storage. The backend services have read access to InfluxDB, retrieving data as needed to fulfill API requests from the frontend application. This separation of write and read responsibilities helps optimize the system for the different patterns of data ingestion and retrieval.

The time-series data model implemented in InfluxDB organizes soil moisture readings by node identifier and timestamp, enabling efficient storage and retrieval of historical measurements. This model supports various time-based queries, from detailed recent data points to aggregated

[20] https://docs.influxdata.com/

historical trends. The system implements retention policies that balance data resolution with storage efficiency, keeping more detailed data for recent readings and aggregated data for historical analysis.

Configuration data follows a different model, stored in SQLite[21] and managed exclusively by the backend services. This relational model captures node configurations including names, measurement intervals, LED states, moisture thresholds, and associated plant information. The configuration data also includes user preferences and notification settings. Unlike the time-series measurement data, configuration changes are infrequent but require strict consistency guarantees, making SQLite an appropriate choice.

The notification data architecture links moisture measurements with user-defined thresholds and delivery preferences. When measurements cross these thresholds, the system generates notifications that include contextual information about the specific plant and its needs. Subscription information is stored in the SQLite database to ensure reliable notification delivery.

Data retention follows a tiered strategy that balances information value with storage constraints. Recent detailed measurements are retained for short-term analysis, while historical data undergoes automatic aggregation to preserve trends while reducing storage requirements. This approach ensures that users can access meaningful insights about their plants' health over time without requiring excessive storage capacity.

3.5 Application architecture

The application layer of the Smart Garden system consists of three main components: the backend API service, the Node-RED orchestration engine, and the frontend Progressive Web Application.

The backend API service, built with Express.js, serves as the central access point for system functionality. It implements a RESTful architecture with endpoints for retrieving plant data, managing configurations, handling image uploads, and managing notification subscriptions. The backend has exclusive write access to the SQLite configuration database and read access to the InfluxDB time-series database, acting as the single point of truth for configuration management. Remote access to the system is secured through Cloudflare Zero Trust Network.

[21] https://www.sqlite.org/docs.html

Node-RED[22] provides visual orchestration of data flows between system components. It subscribes to MQTT topics to receive sensor data, performs initial processing and validation, and writes directly to InfluxDB. When processing incoming messages, Node-RED checks for configuration version mismatches by querying the backend API. If it detects a need for configuration updates, it retrieves the latest configuration via API calls and publishes update messages to the appropriate MQTT topics. This orchestration approach provides flexibility and visibility into system operations while maintaining clean separation of responsibilities.

The frontend application, built with React[23] and Next.js[24], implements a Progressive Web App[25] architecture that provides a native-like experience across devices. It communicates exclusively with the backend API, never directly accessing databases or MQTT topics. The user interaction model focuses on simplicity and actionability, presenting plant status information in an intuitive format and guiding users through configuration options.

The push notification service architecture spans multiple components. The backend API manages subscription registration, storage, and message generation. The frontend implements the Web Push[26] protocol integration, handling permission requests, subscription management, and notification display through service workers. This architecture enables notifications to reach users even when the application isn't actively open in a browser.

Client-server communication follows a standard RESTful pattern for most interactions. For image uploads, the system uses multipart form data to efficiently transfer binary content. Together, these application components create a cohesive system that transforms raw sensor data into meaningful, actionable information for plant owners, addressing the fundamental question of when to water their plants through continuous, automated soil moisture tracking.

[22] https://nodered.org/docs/
[23] https://react.dev/reference/react
[24] https://nextjs.org/docs
[25] https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps
[26] https://www.w3.org/TR/push-api/

4. Implementation

The implementation of the Smart Garden system spans hardware, firmware, networking, backend, and frontend components which evolved from an initial plan for custom hardware to a more focused approach using ready-made components. This section details the technical choices and solutions for each part of the system.

4.1 Hardware

The ESP32-WROOM-32 microcontroller was selected due to its built-in WiFi capabilities, support for deep sleep mode, non-volatile storage, real-time clock, and ESP-NOW protocol. It was also already supported by the zh_network library, which significantly reduced development effort. Standard development boards were used for all node types in the system, configured to operate in station (STA) mode with transmit power set to 8dBm to balance communication range with power consumption.

Capacitive soil moisture sensors were chosen over resistive sensors due to their longer operational lifespan and resistance to corrosion. The sensors were connected to the ESP32's ADC pins with 12-bit resolution (0-4095 range) and 12dB attenuation to accommodate the 0-3.3V output range. The system converts raw ADC values to moisture percentages using a linear calibration determined through experimental testing with dry soil (∼3000 ADC value) and fully saturated soil (∼1500 ADC value).

```
function moistureToPercentage(moisture) {
    if (moisture <= 1500) return 100;
    if (moisture >= 3000) return 0;

    return Math.round(200 - (moisture / 15));
}
```

The current implementation uses USB power for all nodes, with the sensor nodes designed for future battery operation through the deep sleep functionality in the firmware. The GPIO pin assignments were selected to ensure compatibility with deep sleep mode, avoiding pins that might cause wake-up issues.

Component selection was pragmatic and driven by availability and ease of implementation. ESP-NOW was used as the communication protocol because it was already implemented in the

zh_network library, which eliminated the need to develop a custom networking solution from scratch.

The initial project plan included developing custom PCBs with integrated battery management systems (BMS) for wireless sensor nodes. A PCB was designed using EasyEDA[27] and manufactured by JLCPCB[28]. However, a design error in the ESP32 module connections prevented the board from being programmed. Given the thesis timeline constraints, the project pivoted to using ready-made ESP32 development boards, allowing focus on the core mesh networking functionality rather than hardware development challenges.

During hardware integration, an issue was encountered where UART communication between the master and gateway nodes experienced message corruption when only TX and RX lines were connected. This was resolved by establishing a common ground connection between the nodes, highlighting the importance of proper ground referencing in multi-board systems.

4.2 Firmware

The firmware was developed using ESP-IDF[29] (Espressif IoT Development Framework) with PlatformIO[30] in Visual Studio Code on Ubuntu, providing a consistent environment with dependency management. This approach offered direct access to the ESP32's capabilities without the abstraction layer of the Arduino framework.

The zh_network library provides a reliable mesh networking layer on top of ESP-NOW, handling peer discovery, message routing, and automatic retransmission. Sensor readings are acquired through the ESP32's ADC interface with error checking and validation. The readings are packaged into structured messages with node identification and version information, using packed structures for efficient memory layout and transmission.

The sensor nodes implement a sleep/wake cycle to conserve power. Each cycle wakes the ESP32, takes measurements, transmits data via ESP-NOW, waits briefly for configuration updates, and enters deep sleep for the configured interval. Node configuration is stored in the ESP32's non-volatile storage (NVS) to persist across power cycles and deep sleep, including a unique node

[27] https://easyeda.com/
[28] https://jlcpcb.com/
[29] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html
[30] https://docs.platformio.org/en/latest/

identifier, configuration version for change tracking, measurement interval, and LED indicator state for debugging.

4.3 Network

The mesh network implementation uses the zh_network library, which provides mesh capabilities on top of ESP-NOW. In the current implementation, only the broadcast functionality is used, where messages are sent to all nodes in the network rather than using unicast to target specific nodes. This simplifies the routing logic while ensuring that messages reach the gateway regardless of network topology.

The zh_network library implements a flooding algorithm for message routing where each message contains a unique identifier and the original sender's MAC address. Nodes check if they've seen a message before to prevent infinite loops, while ensuring messages can reach their destination via multiple paths without explicit route discovery.

The network implements auto-discovery and self-healing mechanisms, adapting to changes in node availability and physical placement without manual intervention. Error handling includes built-in retry mechanisms that attempt to resend failed messages up to a configurable number of times.

4.4 Backend

The backend API was implemented using Node.js and Express, providing RESTful endpoints for the frontend and handling data from the gateway node. The key API endpoints include:

Endpoint	Description
GET /online-nodes	Returns active sensor nodes with their latest readings
GET /config	Retrieves node configuration by ID
PUT /config	Updates node configuration
GET /current-measurements	Provides current sensor readings for a specific node
GET /history-measurements	Retrieves historical data for visualization in day, week, and month ranges
POST /upload-image	Handles plant image uploads
GET/PUT/DELETE /subscription	Manages push notification subscriptions
POST /subscription	Sends notifications to subscribed devices

Table 2. Backend API endpoints

InfluxDB was implemented for time-series storage of sensor readings, with queries optimized for different time ranges. The SQLite database model is structured as follows:

Column	Properties	Description
id	STRING, Primary Key	Unique node identifier
name	STRING	User-assigned name for the plant/sensor-node
version	INTEGER	Configuration version for tracking changes
interval	INTEGER	Measurement interval in seconds
led_state	BOOLEAN	Debug LED indicator status
imagePath	STRING	Path to uploaded plant image
soilMoistureThreshold	INTEGER	Alert threshold for low moisture
createdAt	DATETIME	Record creation timestamp
updatedAt	DATETIME	Record update timestamp

Table 3. Config table structure

Column	Properties	Description
id	STRING, Primary Key	Device identifier
data	STRING	Serialized push subscription data
lastSent	DATETIME	Timestamp of last notification
createdAt	DATETIME	Record creation timestamp
updatedAt	DATETIME	Record update timestamp

Table 4. Subscription table structure

Node-RED serves as the orchestration layer between the MQTT broker and the backend systems. The flow is organized in several stages:

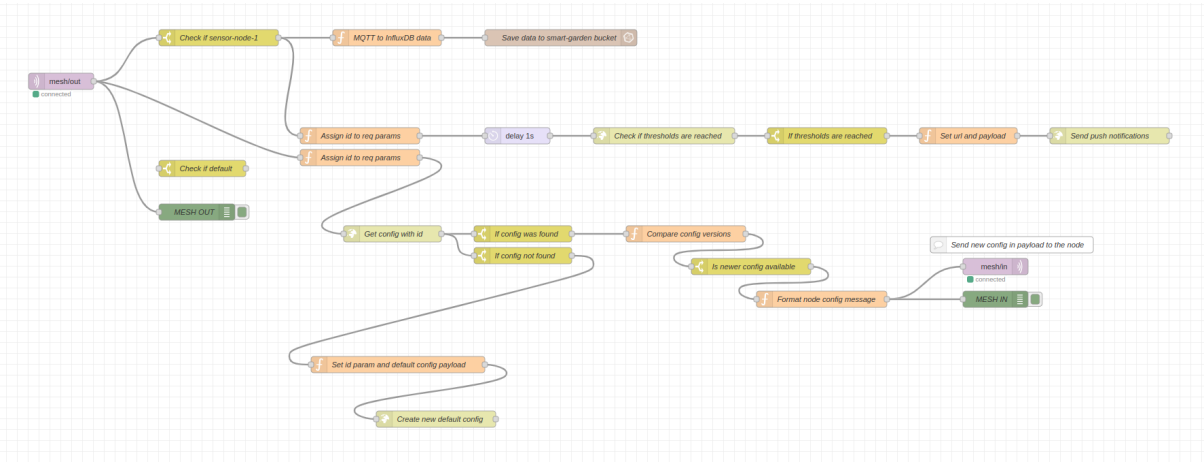

Figure 2. Node-RED flow

The Node-RED flow handles these key functions:

- Subscribes to the “mesh/out” MQTT topic to receive sensor readings from the gateway
- Parses and validates incoming JSON messages
- Checks configuration version against the API to detect updates
- Writes measurement data to InfluxDB with appropriate tagging
- Queries the API for any nodes below moisture thresholds
- Triggers push notifications when measurements fall below configured thresholds
- Publishes configuration updates to the “mesh/in” MQTT topic

The MQTT broker was configured with topics for sensor data (mesh/out) and configuration updates (mesh/in). The Web Push protocol was implemented for delivering notifications to users when soil moisture falls below configured thresholds.

4.5 Frontend

The frontend was implemented as a Progressive Web App using React and Next.js, providing a responsive and user-friendly interface for monitoring and managing plants. The PWA approach was chosen specifically to enable push notifications on iOS devices, which only supports notifications through PWAs or native apps. Given the lack of mobile app development experience, a PWA was the most practical solution.

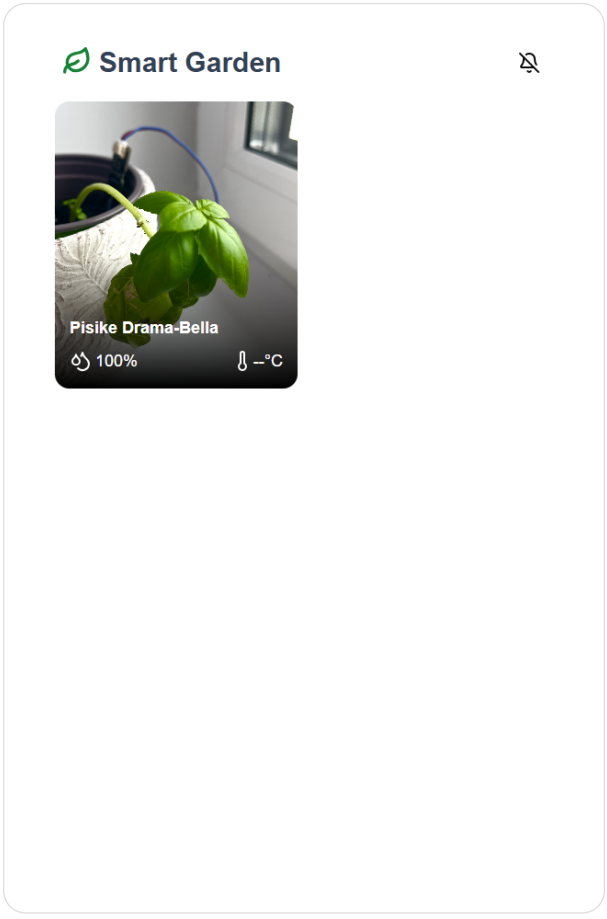

Figure 3. Screenshot of the dashboard view

The Dashboard page presents an overview of all monitored plants with current status:

- Plant cards showing the plant name and image
- Current moisture level displayed as a percentage
- Visual warning indicators for plants below moisture thresholds
- Navigation to detailed views for each plant
- Option to subscribe to push notifications

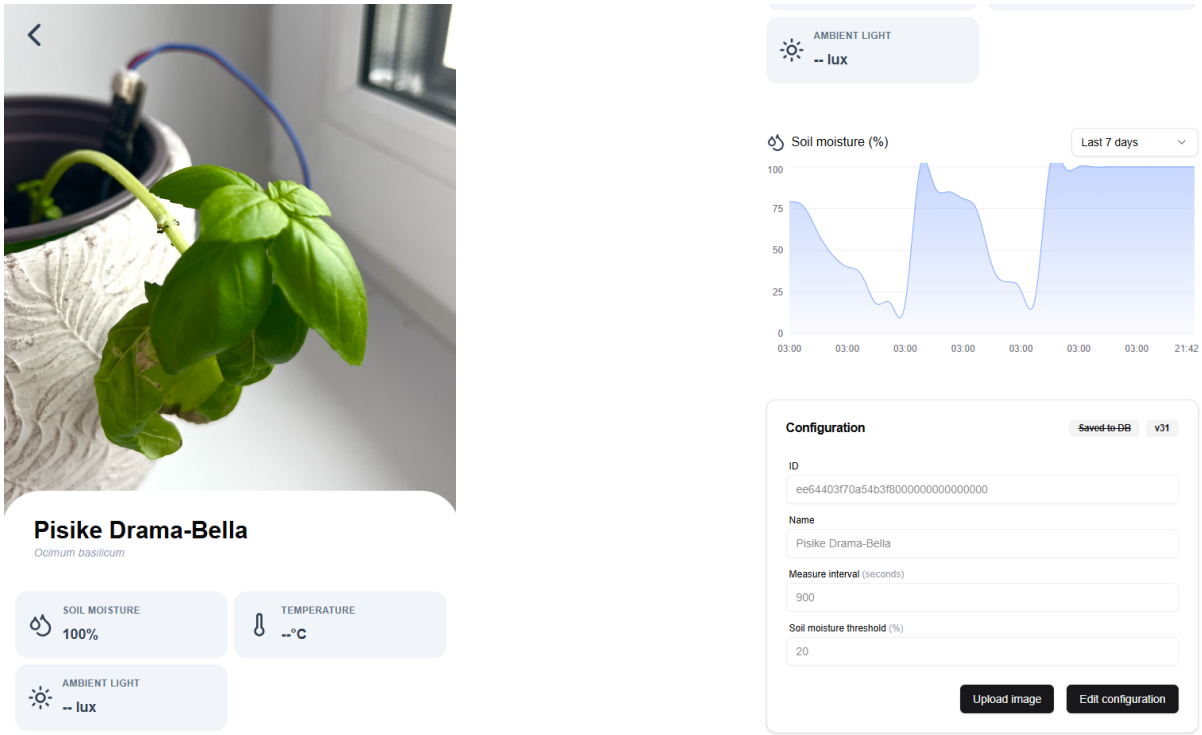

Figure 4. Screenshots of the details view

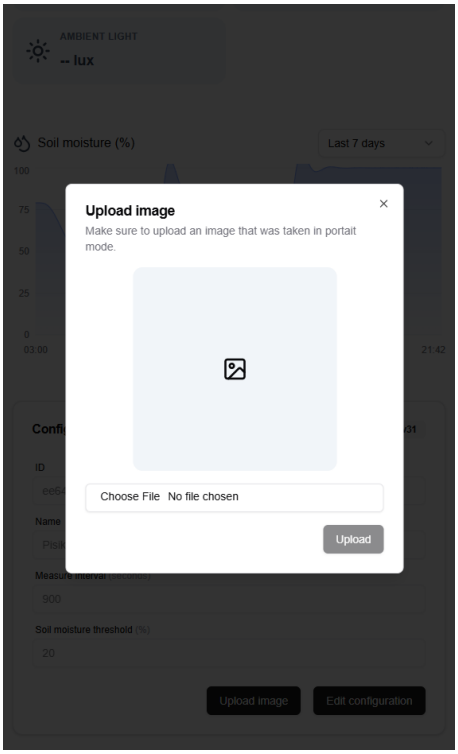

Figure 5. Screenshot of the upload image modal

The Plant Details and Configuration page provides:

- Large plant image at the top of the screen
- Current measurements displayed in cards with appropriate icons
- Historical charts showing moisture levels over different time periods (day, week, month)
- Change the plant name
- Set measurement intervals (how often the sensor takes readings)
- Configure moisture thresholds for notifications
- Upload new plant images

The PWA implementation provides push notification handling and responsive design that adapts to different screen sizes. Users can subscribe to notifications that alert them when plants need watering, making the system more practical for everyday use.

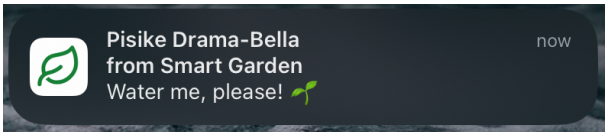

Figure 6. Screenshot of the push notification from iPhone

4.6 Deployment

The system is deployed across a Lenovo laptop running Ubuntu for the Express API, SQLite, and frontend, and a mini PC running Proxmox with Debian containers for the MQTT broker, Node-Red, InfluxDB, and Cloudflare Zero Trust Network[31] tunnel. IoTempower[32] was used to simplify the deployment of Node-RED and the MQTT broker, providing a convenient development environment for these services. ESP32 nodes are strategically positioned throughout the home environment to ensure comprehensive coverage while minimizing power consumption.

This implementation demonstrates the practical application of distributed systems principles in an IoT context, providing a robust foundation for plant monitoring while maintaining flexibility for future enhancements.

[31] https://developers.cloudflare.com/cloudflare-one/
[32] https://github.com/iotempire/iotempower

5. Evaluation

This section evaluates the system in terms of its network performance, sensor functionality, and overall effectiveness in meeting the original design objectives.

5.1 Performance

A multi-floor test was conducted in a residential apartment building to assess the mesh networking capability. A sensor node was placed on the first floor, a relay node on the second floor, and the master node on the third floor, with concrete floors and walls separating each level. The sensor node measurement interval was set to 5 seconds for testing purposes.

The test results confirmed that direct communication between the first-floor sensor node and third-floor master node was impossible due to the concrete structure blocking signal transmission. However, with the relay node on the second floor, the system successfully established communication between all nodes. This validated one of the core design principles: extending network coverage to areas unreachable with traditional direct WiFi connectivity.

While specific latency measurements were not conducted as they were not critical for the plant monitoring application, observational testing suggested that message transmission latency was minimal. For soil moisture monitoring where readings change gradually over hours, the system's transmission speed was more than adequate for the intended application.

5.2 Reliability

Analysis of soil moisture data collected between January 17 and May 1, 2025, revealed reliable monitoring performance. With 9,175 recorded measurements over approximately 105 days, the system achieved an effective reliability rate of 91% when comparing actual measurements to the theoretical maximum of 10,080 readings possible at 15-minute intervals. While several extended gaps (exceeding 2 hours) were observed in the dataset, these coincided with development activities rather than representing actual system failures. Excluding these intentional interruptions, the adjusted reliability rate increased to approximately 96%, with only occasional missed readings during normal operation. This reliability ensured accurate tracking of soil moisture trends for timely watering notifications.

5.3 Usability

The Progressive Web Application (PWA) was tested on an Ubuntu laptop and an iPhone, demonstrating cross-platform compatibility. The user interface rendered appropriately on

both devices, maintaining consistent functionality and appearance. Push notifications were successfully delivered to both platforms when soil moisture levels dropped below configured thresholds, confirming the effectiveness of the Web Push API implementation across different operating systems. This limited but effective testing demonstrated that the application functions as intended on both desktop and mobile platforms without requiring platform-specific code.

5.4 Plant growth monitoring

A single basil plant was monitored for a three and a half month period to assess both system longevity and effectiveness in supporting plant health. Figures 7, 8, 9 and 10 show the plant's growth progression at the start of each month from February to April. The system successfully alerted when watering was needed, leading to consistent soil moisture management. As evidenced by the plant's growth, the system effectively solved the core problem of determining optimal watering times.

Figure 7. Start of February

Figure 8. Start of March

Figure 9. Start of April

Figure 10. Start of May

The moisture level data collected during this period revealed clear patterns related to daily evaporation cycles and effects of plant growth on water consumption rates.

5.5 Limitations

The most significant limitation identified during testing relates to the implementation of Cloudflare Zero Trust Network (ZTN) for secure remote access. When a ZTN session expires, which can be set to a maximum of 1 month, users must re-authenticate. During this unauthorized period, push notifications do not function in the background, potentially causing missed watering alerts if the user is not actively using the application. This represents a usability challenge that should be addressed in future iterations.

Another limitation is the current power supply method. The system currently relies on wired power connections for all nodes. While this approach ensures reliable operation, it restricts placement options and reduces flexibility. Future implementations could benefit from battery-powered operation, particularly for sensor nodes, which would allow for more flexible positioning and truly wireless operation. A battery-powered configuration would make the system more adaptable to diverse indoor environments where power outlets may not be conveniently located near plants.

A third limitation concerns mesh network security. The ESP-NOW communication between nodes lacks encryption and authentication mechanisms, leaving the system vulnerable to data

interception or injection attacks. Production implementations would require node-level security measures to protect data integrity and prevent unauthorized network access.

5.6 Evaluation against original requirements

The system was evaluated against the original design requirements to determine how well it fulfilled its intended purpose.

Requirement	Evaluation	Status
Mesh networking to overcome WiFi limitations	Successfully demonstrated in multi-floor testing with concrete barriers	Achieved
Continuous soil moisture monitoring	System maintained 96% reliability (excluding development interruptions) over 105 days	Achieved
User-friendly interfaces	PWA implementation worked consistently on Ubuntu laptop and iPhone	Achieved
Individualized sensor configuration	Successfully implemented different measurement intervals for different nodes	Achieved
Push notifications for watering alerts	Implemented but affected by Cloudflare ZTN session expiration	Partially achieved
Scalability	System architecture supports scaling but tested with limited nodes	Partially verified
Environmental impact through improved plant survival	Demonstrated through successful growth of test plant over three months	Achieved

Table 5. Evaluation of system against original requirements

The mesh networking capability was conclusively proven in the multi-floor testing scenario with concrete barriers between nodes. The system maintained exceptionally reliable soil moisture monitoring with 96% uptime (excluding planned development interruptions) over a 105-day period from January to May 2025. The user interface functioned effectively on both desktop and mobile platforms.

Individual sensor configuration worked as designed, allowing customization for different plant needs. Push notifications functioned effectively when the user was authenticated but were impacted by Cloudflare ZTN session management limitations. While the system architecture theoretically supports larger deployments, it was primarily tested with a smaller number of nodes.

Overall, the system successfully addressed its primary objective: transforming the question of ”When do I water my plants?” into clear, actionable notifications based on actual soil moisture data.

6. Conclusion

This thesis presented Smart Garden, a mesh-networked IoT system that successfully addresses the challenge of determining optimal watering times for indoor plants. The system demonstrated several key strengths: effective mesh communication through concrete barriers, reliable operation with 96% uptime over three months, and intuitive user interfaces with functional push notifications. The evaluation confirmed that the mesh network approach effectively overcomes WiFi coverage limitations while enabling individualized sensor configuration.

The successful growth of the test plant validated that the system meets its primary objective of transforming the question of ”When do I water my plants?” into clear, actionable notifications. While limitations exist regarding power supply and authentication mechanisms, the implementation provides a solid foundation for future enhancements such as battery operation and additional environmental sensors. This project demonstrates that appropriately applied mesh networking technology can create practical solutions for everyday problems while contributing to environmental sustainability through improved plant care outcomes.

References

[1] Han K.-T., Ruan L.-W., and Liao L.-S. Effects of Indoor Plants on Human Functions: A Systematic Review with Meta-Analyses. *International Journal of Environmental Research and Public Health* 19.12 (Jan. 2022). Number: 12 Publisher: Multidisciplinary Digital Publishing Institute, p. 7454. DOI: 10.3390/ijerph19127454. https://www.mdpi.com/1660-4601/19/12/7454 (04/06/2025).

[2] Chowdhury S., Sen S., and Janardhanan S. Comparative Analysis and Calibration of Low Cost Resistive and Capacitive Soil Moisture Sensor. Oct. 6, 2022. DOI: 10.48550/arXiv.2210.03019. arXiv: 2210.03019[cs]. http://arxiv.org/abs/2210.03019 (04/06/2025).

[3] Singh A., Gaurav K., Sonkar G. K., and Lee C.-C. Strategies to Measure Soil Moisture Using Traditional Methods, Automated Sensors, Remote Sensing, and Machine Learning Techniques: Review, Bibliometric Analysis, Applications, Research Findings, and Future Directions. *IEEE Access* 11 (2023), pp. 13605–13635. DOI: 10.1109/ACCESS.2023.3243635. https://ieeexplore.ieee.org/document/10041136/ (04/06/2025).

[4] Buratti C., Conti A., Dardari D., and Verdone R. An Overview on Wireless Sensor Networks Technology and Evolution. *Sensors* 9.9 (Sept. 2009). Number: 9 Publisher: Molecular Diversity Preservation International, pp. 6869–6896. DOI: 10.3390/s90906869. https://www.mdpi.com/1424-8220/9/9/6869 (05/08/2025).

[5] Nimi T. and Samundiswary P. Comparative Performance evaluation on Priority based ZigBee Network with tree and mesh routing. *2018 4th International Conference on Electrical Energy Systems (ICEES)*. 2018 4th International Conference on Electrical Energy Systems (ICEES). Feb. 2018, pp. 691–695. DOI: 10.1109/ICEES.2018.8443184. https://ieeexplore.ieee.org/document/8443184 (05/08/2025).

[6] A J. S., Chakravarthy R., and L M. L. An Experimental study of IoT-Based Topologies on MQTT protocol for Agriculture Intrusion Detection. *Measurement: Sensors* 24 (Dec. 1, 2022), p. 100470. DOI: 10.1016/j.measen.2022.100470. https://www.sciencedirect.com/science/article/pii/S2665917422001040 (05/08/2025).

[7] Al-Sarawi S., Anbar M., Alieyan K., and Alzubaidi M. Internet of Things (IoT) communication protocols: Review. *2017 8th International Conference on Information Technology (ICIT)*. 2017 8th International Conference on Information Technology (ICIT). May 2017, pp. 685–690. DOI: 10.1109/ICITECH.2017.8079928. https://ieeexplore.ieee.org/document/8079928 (05/08/2025).

[8] Bhatia S., Jaffery Z. A., and Mehfuz S. A Comparative Study of Wireless Communication Protocols for use in Smart Farming Framework Development. *2023 3rd International Conference on Intelligent Communication and Computational Techniques (ICCT)*. 2023 3rd International Conference on Intelligent Communication and Computational Techniques (ICCT). Jan. 2023, pp. 1–7. DOI: 10.1109/ICCT56969.2023.10075696. https://ieeexplore.ieee.org/document/10075696 (05/08/2025).

[9] Eridani D., Rochim A. F., and Cesara F. N. Comparative Performance Study of ESP-NOW, Wi-Fi, Bluetooth Protocols based on Range, Transmission Speed, Latency, Energy Usage and Barrier Resistance. *2021 International Seminar on Application for Technology of Information and Communication (iSemantic)*. 2021 International Seminar on Application for Technology of Information and Communication (iSemantic). Semarangin, Indonesia: IEEE, Sept. 18, 2021, pp. 322–328. DOI: 10.1109/iSemantic52711.2021.9573246. https://ieeexplore.ieee.org/document/9573246/ (05/05/2025).

A. PCB Documentation

A.1 PCB Schematic

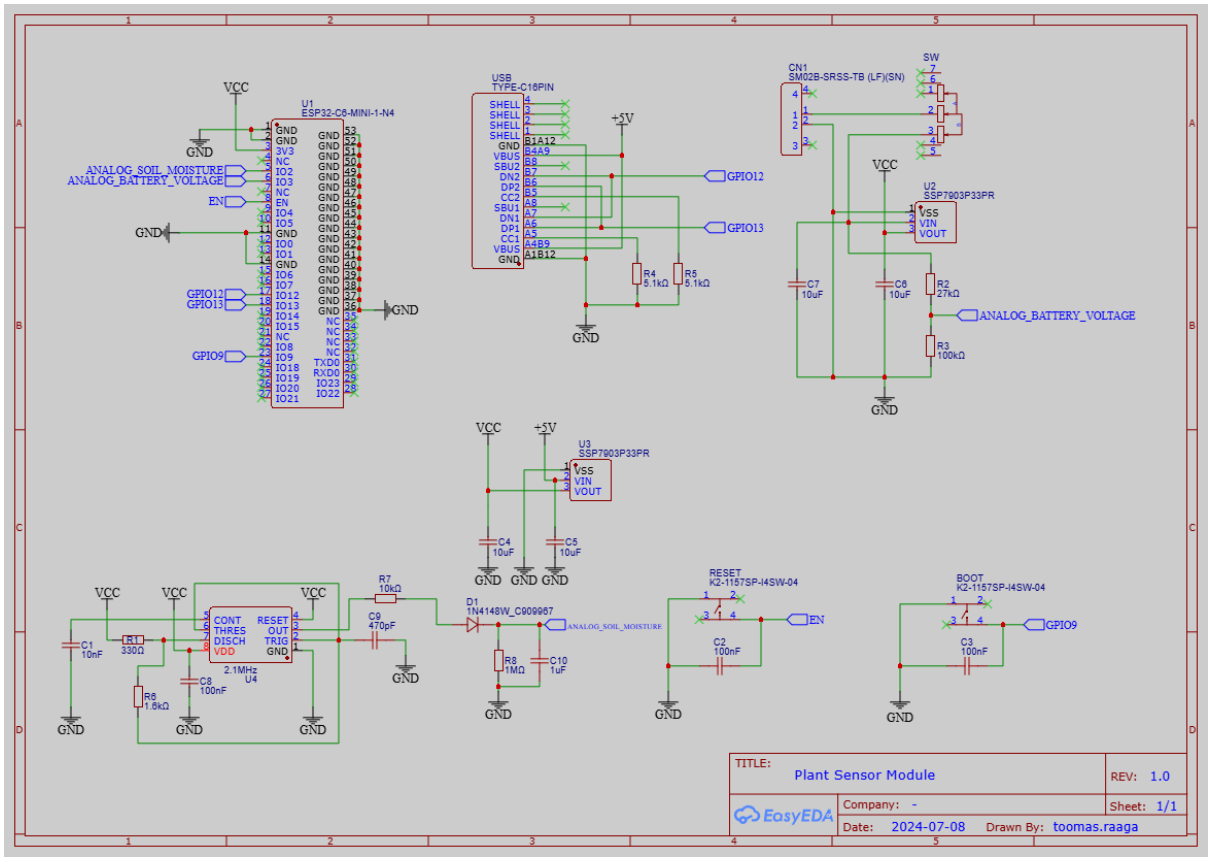

Figure 11. Complete circuit schematic of the PCB design

A.2 PCB Layout

Figure 12. PCB layout showing component placement and routing

A.3 Assembled PCB

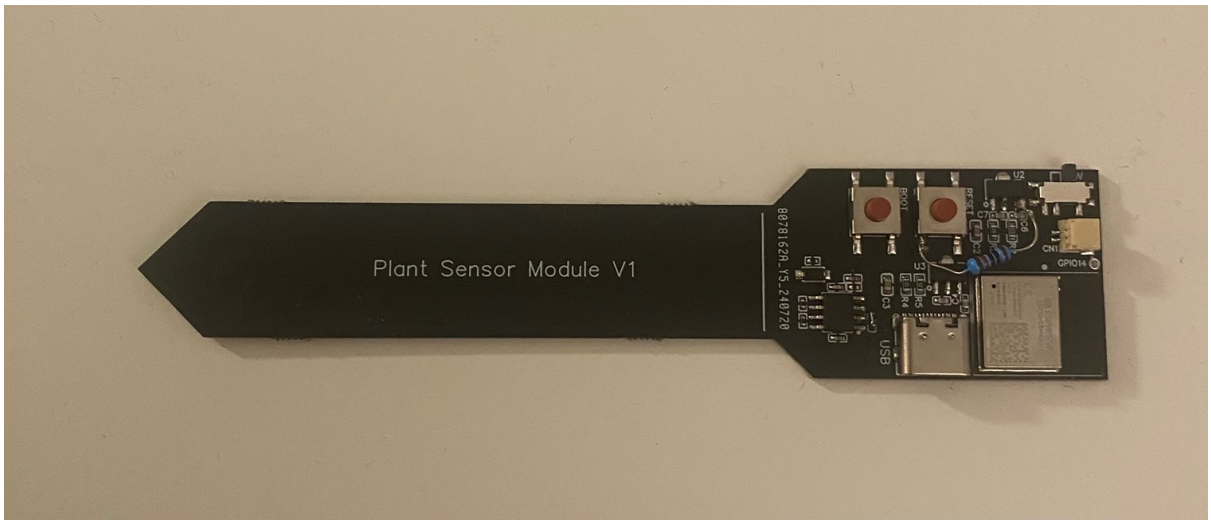

Figure 13. Photograph of the assembled PCB

B. GitHub repository

The complete source code for the Smart Garden project, including firmware, backend, and frontend implementations, is available in the following GitHub repository:

https://github.com/traaga/smart-garden

C. License

Non-exclusive licence to reproduce the thesis and make the thesis public

I, **Toomas Hendrik Raaga**

1. grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis,

   Developing a Mesh Networked IoT System for Plant Health Monitoring

   supervised by Ulrich Norbisrath and Matevž B. Zorec;

2. grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;

3. am aware of the fact that the author retains the rights specified in points 1 and 2;

4. confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.l

Toomas Hendrik Raaga

15/05/2025