MAKERERE UNIVERSITY

SMART CLOUD-BASED SOIL ADVISOR

By BSE 25-28

EMBEDDED SYSTEMS SPECIFICATION

DEPARTMENT OF NETWORKS

COLLEGE OF COMPUTING AND INFORMATION SCIENCES

A Project Report Submitted to the School of Computing and Informatics Technology

For the Study Leading to a Project in Partial Fulfillment of the

Requirements for the Award of the Degree of Bachelor of Software Engineering of Makerere University.

Supervisor

Dr. Swaib Kyanda Kaawaase

Department of Networks

School of Computing and Informatics Technology, Makerere University, kswaibk@gmail.com

May, 2025

Declaration

We, group BSE 25-28, hereby declare hereby declare that this Project Report titled "Smart Cloud-Based Soil Advisor" is an original work authored by us and has never been submitted for an award to any university or institution of higher learning. We can confirm that where we have done consultations either from published material or the works of others, it has been attributed in this report.

#	NAME	REGISTRATION NUMBER	SIGNATURE
1	PRAJAPATI HELI ANILKUMAR	21/X/20194/PS	
2	MUGOYA MOSES	21/U/10447/PS	
3	MUHANGI ELIODA	22/U/6342	

Approval

This project report titled Smart Cloud-Based Soil Advisor has been submitted for examination with my approval as the supervisor of group BSE 25-28.

Dr. Swaib Kyanda Kaawaase

Department of Networks

School of Computing and Informatics Technology;

College of Computing and Information Sciences,

Makerere University

Signature: .... Date: ....

Supervisor

Dedication

We dedicate this work first of all to our parents who have invested plenty of effort for us to achieve this milestone. To our tutors, we dedicate it to them for they have taught us patience and hard work on any tasks that are presented to us.

To our supervisor, Dr. Swaib Kyanda Kaawaase, we owe immense gratitude for his unwavering guidance, technical insight, and continued encouragement. His mentorship played a crucial role in shaping our approach, refining our ideas, and helping us grow as engineers.

Finally, we also dedicate this project to the school communities and students across Uganda who inspired our mission. It is our hope that this system contributes to safer, smarter, and more efficient agricultural practices for all.

Acknowledgments

First and foremost, we are grateful to the Almighty God for without His graces and blessings, this project report would not have been possible.

We are deeply indebted to our project supervisor, Dr. Swaib Kyanda Kaawaase, for his invaluable mentorship, insightful feedback, and unwavering support during every phase of the project—research, development, and implementation. His expertise and encouragement were instrumental in shaping our work and overcoming challenges.

We sincerely thank Dr. Mary Nsabagwa, our Project Coordinator, for her unwavering support, valuable feedback, and effective leadership, which greatly improved our work.

Our sincere appreciation also goes to our dear parents who have extended the necessary facilitation required not only in this study but throughout our education life.

We are also grateful to our group members for their dedication, teamwork, and unique contributions, which made this project both productive and rewarding.

This project was made possible by the support and encouragement of our supervisor, family, and peers, for which we are deeply thankful.

Abstract

The Smart Cloud-Based Soil Advisor (CBSA) system integrates Internet of Things (IoT) sensors with cloud computing to provide real-time monitoring of soil conditions. By collecting data on parameters such as moisture, temperature, and nutrient levels, the system offers actionable insights and crop recommendations to farmers. The systems current version supports three major crops local maize, beans and Coffee(Arabica) to which proper recommendations can be given. This technology aims to enhance precision agriculture, optimize resource usage, and promote sustainable farming practices. Key functionalities of the system include multi-user management, compatibility with various IoT sensor devices, automated alerts and notifications, and secure cloud-based data processing.

The system is intended for deployment in agricultural settings across the country, assisting farmers in monitoring soil health and making informed decisions to improve crop productivity.

Table of Contents

List of Figures

List of Tables

Abbreviations/Acronyms

Acronym	**Meaning**
API	Application Programming Interface
CBSA	Cloud-Based Soil Advisor
CI/CD	Continuous Integration/Continuous Deployment
ESP32	Espressif Systems 32-bit Microcontroller
GND	Ground
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IoT	Internet of Things
JSON	JavaScript Object Notation
LAN	Local Area Network
LTE	Long-Term Evolution
NPK	Nitrogen, Phosphorus, Potassium
OS	Operating System
PC	Personal Computer
PWA	Progressive Web App
REST	Representational State Transfer
RS485	Recommended Standard 485
SRAM	Static Random-Access Memory
SRS	Software Requirements Specification
SDD	Software Design Document
UART	Universal Asynchronous Receiver/Transmitter
UI	User Interface
VCC	Voltage Common Collector
Wi-Fi	Wireless Fidelity

1 Introduction

1.1 Background and scope of the project

Agriculture remains a cornerstone of Uganda's economy, with a significant portion of the population engaged in farming activities [1]. However, many framers face challenges related to soil health, leading to suboptimal crop yields [2]. Traditional methods of soil assessment are often time-consuming, expensive, and inaccessible to smallholder farmers [3].

The Smart Cloud-Based Soil Advisor project aims to bridge this gap by leveraging Internet of Things (IoT) technologies and cloud computing to provide real-time soil monitoring and advisory services. By integrating sensors that measure soil moisture, temperature, and nutrient levels (Nitrogen, Phosphorous, and Potassium), the system collects vital data, which is then processed and analyzed to offer actionable insights to farmers. This empowers them to make informed decisions regarding crop selection, irrigation, and fertilization, ultimately enhancing productivity and sustainability [4].

The scope of this project encompasses the design and implementation of an embedded system for data collection, a cloud-based backend for data processing and storage, and a user-friendly frontend interface accessible via web and mobile platforms. Additionally, the system incorporates AI-driven recommendations to provide personalized farming advice based on the collected soil conditions on a particular farm [5].

1.2 Overview of the document

This document delineates the comprehensive process undertaken in the development of the Smart Cloud-Based Soil Advisor system. It encompasses the following sections:

- **Chapter 1: Introduction** - This section gives an overview of the document
- **Chapter 2: System Specifications** - Details the system requirements, inputs, outputs, functionalities, limitations, default settings, special requirements, and error handling mechanisms.

- **Chapter 3: Design Output** - Discusses the implementation process, including coding practices, development tools, device interfaces, and documentation.
- **Chapter 4: Inspection and Testing** - Outlines the testing strategies employed to ensure system reliability and performance.
- **Chapter 5: Installation and System Acceptance Test** - Describes the installation procedures and criteria for system acceptance.
- **Chapter 6: Performance, Servicing, Maintenance, and Phase-Out** - Provides insights into system maintenance, performance metrics, and strategies for future upgrades or decommissioning.
- **Chapter 7: Conclusion and Recommendations** - Summarizes the project's outcomes and offers recommendations for future work.
- **Appendix A: User Manual** - Offers a comprehensive guide for end-users to interact with the system effectively.

This structured approach ensures a thorough understanding of the project's objectives, methodologies, and outcomes, serving as a valuable resource for stakeholders and future developers.

2 System Specifications

2.1 Version of requirements and Version Control

The requirements for the Smart Cloud-Based Soil Advisory System evolved iteratively as development progressed. Initial requirements were documented in version 0.1, which focused on basic soil data acquisition and web display. Feedback from preliminary tests and stakeholder reviews led to updates in version 0.2, which added agentic recommendations and mobile responsiveness.

Version control was maintained using Git. All source code was managed in a Git repository, allowing for tracking of changes, collaborative development, and rollback capability when needed.

Version Control:

- **Tool Used:** Git
- **Repository Hosting:** Render
- **Versioning Convention:** Semantic Versioning (v0.1.0)
- **Branching Strategy:** Feature branches were created for new functionalities, which were then merged into the **Main** branch upon completion and review.

Task Management:

- **Tool Used:** ClickUp
- **Integration with GitHub:** Implemented to link GitHub activities (commits, pull requests) with ClickUp tasks for real-time updates and streamlined workflow.

Version History:

- **Version 0.1.0:** Initial requirements gathering and documentation.
- **Version 0.2.0:** Incorporated feedback from the supervisor, leading to the addition of AI agent integration using OpenAI and consideration of mobile responsiveness.

- **Version 0.3.0:** Refined system architecture diagrams and embedded system design based on further research and team discussions.

Each version was tagged in GitHub, and corresponding tasks were updated in ClickUp to reflect the changes and progress.

2.2 Input

The system receives input from both hardware and user interfaces. These include:

- **Sensor Inputs:**
    - **Soil Moisture Sensor:** Measures the volumetric water content of soil, providing a digital percentage value.
    - **Temperature Sensor:** Measures the ambient or soil temperature in degrees Celsius.
    - **NPK Sensor:** Analyzes soil nutrient levels, specifically Nitrogen (N), Phosphorus (P), and Potassium (K). Outputs values in mg/kg, which are used to assess soil fertility.

Sensor data is collected via microcontrollers (such as Arduino or ESP32) and transmitted wirelessly to a cloud database at regular intervals.

- **User Inputs:**
    - **Farm Profile Setup:** Users input details like farm location, size, and type of crops.
    - **System Queries**: Users may request crop recommendations from the system.
    - **Authentication Inputs:** Email and password are entered during login or registration.

- **Handling Invalid Inputs:**
    - Sensors are programmed to return null or default values when readings are outside expected ranges.
    - Form validation (both frontend and backend) prevents incomplete or incorrect data entries.

2.3 Output

The system provides several outputs for monitoring, decision-making, and reporting:

- **Real-Time Data Display:**
  - Visual dashboards show graphs and indicators for moisture, temperature, and NPK (Nitrogen, Phosphorous, Potassium) levels.
- **Advisory Recommendations:**
  - Based on sensor readings, the system suggests actions such as "Apply nitrogen-based fertilizer" or "Irrigate the soil within 6 hours."
  - Recommendations include timestamped records of recent readings.
- **Notifications and Alerts:**
  - Users receive in-app notifications when soil conditions are critical.

2.4 Functionality

The Smart Cloud-Based Soil Advisory System provides an intuitive platform designed specifically for farmers to manage their farms and make data-driven decisions. The system encompasses a range of features that enable farmers to easily monitor soil conditions and receive tailored crop recommendations.

1. Soil Monitoring

Utilizing IoT sensors, the system continuously captures soil parameters such as moisture, temperature, and nutrient levels (NPK). This data is transmitted to the cloud for processing and analysis, enabling timely interventions in agricultural practices.

2. Data Visualization

The system presents collected data through intuitive dashboards, offering graphical representations of soil conditions over time. This aids farmers in understanding trends and making informed decisions.

3. Crop Suitability Advisory

Based on analyzed soil data, the system recommends suitable crops that can thrive under current soil conditions, optimizing yield potential.

4. Device Integration

The system is compatible with various IoT sensor devices, facilitating automated data collection and ensuring scalability for different farming setups.

6. Automated Alerts and Notifications

Users receive real-time alerts regarding critical soil conditions, such as low moisture or nutrient deficiencies, enabling prompt corrective actions.

7. Cloud-Based Data Processing

Leveraging cloud computing, the system ensures secure storage of soil health records and employs predictive analytics for future soil condition forecasting

2.5 Limitations and safety

While the Smart Cloud-Based Soil Advisor system offers numerous benefits, it also has certain limitations and safety considerations:

Limitations

- **Dependence on Internet Connectivity**: The system requires a stable internet connection for real-time data transmission and access. In areas with poor connectivity, system performance may be hindered.
- **Power Supply Constraints**: IoT sensors and the ESP32 microcontroller rely on continuous power supply. In regions with unreliable electricity, alternative power solutions like solar panels may be necessary.
- **Sensor Calibration and Maintenance**: Regular calibration and maintenance of sensors are essential to ensure data accuracy. Neglect can lead to erroneous data and misinformed decisions.
- **Data Security Concerns**: As data is transmitted and stored in the cloud, there is a risk of unauthorized access or data breaches. Implementing robust security protocols is vital.
- **Initial Setup Costs**: The acquisition and installation of IoT devices and sensors may involve significant initial investment, which could be a barrier for small-scale farmers.

Safety Measures

1. **Input Validation**: The system incorporates validation checks to prevent erroneous data entry, ensuring the integrity of collected data.

2. **User Authentication**: Secure login mechanisms are in place to restrict access to authorized users only, safeguarding sensitive information.

3. **Fail-Safe Mechanisms**: In the event of sensor malfunction or data anomalies, the system triggers alerts and switches to predefined safe modes to prevent incorrect advisories.

4. **Compliance with Data Protection Regulations**: In alignment with global data standards, the system implements necessary measures to protect user's information and uphold user's rights.

2.6 Default settings

Upon initial power-up, the system initializes with the following default configurations:

- **Wi-Fi Configuration**: The ESP32 microcontroller attempts to connect to a predefined Wi-Fi network using default credentials. If unsuccessful, it establishes a local access point to allow manual configuration.

- **Sensor Calibration**: Soil moisture sensors are calibrated with default values representing 0% (dry) and 100% (saturated) moisture levels. These can be adjusted through the system's calibration interface.

- **Data Logging**: Data from sensors is logged at 15-minute intervals by default. This interval can be modified in the system settings.

- **Alert Thresholds**: Default thresholds for soil moisture, temperature, and NPK levels are set based on standard agricultural recommendations. Users can customize these thresholds to suit specific crop requirements.

2.7 Special requirements

To ensure the system's reliability, security, and integrity, the following special requirements are implemented:

- **Security and Confidentiality**:
  - **Data Encryption**: All data transmitted between the ESP32 and the backend server is encrypted using HTTPS protocols to prevent unauthorized access.
  - **Authentication**: User authentication mechanisms are in place to restrict access to authorized personnel only.
  - **Data Storage**: Sensitive data is stored securely on the server with appropriate access controls.
- **Change Control and Backup**:
  - **Version Control**: All code and configuration changes are tracked using Git, ensuring traceability and the ability to revert to previous versions if necessary.
  - **Task Management**: ClickUp is utilized for assigning tasks and tracking progress, facilitating organized development workflows.
  - **Data Backup**: Regular backups of sensor data and system configurations are scheduled to prevent data loss.
- **Error Handling and System Integrity**:
  - **Input Validation**: The system includes input validation checks to prevent erroneous data from causing system malfunctions.
  - **Watchdog Timers**: Implemented to reset the system in case of unresponsive states, ensuring continuous operation.
  - **Physical Protection**: Sensors and hardware components are housed in protective enclosures to shield against environmental hazards.

2.8 Errors and alarms

The system is designed to detect and handle various errors and trigger alarms as necessary:

- **Sensor Errors**:
  - **Disconnected Sensor**: If a sensor is disconnected or fails, the system logs the error and notifies the user through the interface.
  - **Out-of-Range Values**: Sensor readings outside expected ranges trigger alerts, prompting users to inspect the affected sensor.
- **Communication Errors**:
  - **Wi-Fi Disconnection**: Loss of Wi-Fi connectivity results in the system attempting to reconnect automatically. If unsuccessful, users are alerted to the issue.
  - **Server Unreachable**: Inability to communicate with the backend server triggers an alert, and data is stored locally until the connection is restored.
- **Power Supply Issues**:
  - **Low Battery**: For systems powered by batteries, a low battery level triggers an alert, advising users to replace or recharge the battery.
  - **Power Failure**: Unexpected power loss is logged, and the system attempts to resume normal operation upon restoration.
- **Alarm Mechanisms**:
  - **Visual Indicators**: LED indicators on the hardware provide immediate visual feedback on system status.
  - **User Notifications**: Alerts and error messages are displayed on the user interface, and critical issues may trigger email notifications.

3 Design output

This chapter documents the artifacts generated during the design and implementation of the Smart Cloud-Based Soil Advisor system, along with the development practices employed. It covers the tools, hardware, software environments, and coding methodologies used to build the system, ensuring a comprehensive record of the development process.

3.1 Implementation (coding and compilation)

Development Tools and Environment

The development of the Smart Cloud-Based Soil Advisor system utilized a suite of specialized tools to ensure efficient coding, compilation, and debugging across both embedded and web components:

- **Integrated Development Environment (IDE)**:
  - For embedded development (Arduino Nano and ESP32), we used the Arduino IDE (v2.3.2) for writing, compiling, and uploading firmware to the microcontrollers. The IDE supports C/C++ and adruino language which provides a streamlined environment for embedded systems development.
  - For web development (Django backend and React frontend), we employed Visual Studio Code (v1.89) as the primary IDE. VS Code offers robust support for Python, JavaScript, and TypeScript, with extensions like Prettier (for code formatting) and ESLint (for JavaScript linting) to maintain code quality.
- **Compiler**:
  - For the embedded system, we used the AVR-GCC compiler (bundled with Arduino IDE) to compile C/C++ code for the Arduino Nano (ATmega328P microcontroller). For the ESP32, the Xtensa GCC compiler (also bundled with Arduino IDE via the ESP32 board package) was used to generate machine code for the ESP32-WROOM-32 module. These compilers are optimized for their respective microcontrollers, ensuring efficient execution of firmware.

    - For the Django backend, Python code was interpreted using Python 3.11, with no separate compilation step required.

- **Debugger**:
    - For embedded debugging, we utilized the Serial Monitor in Arduino IDE to log real-time data from the Arduino Nano and ESP32. This allowed us to inspect sensor readings, network status, and error messages during development.
    - For the web application, we used Django's Debug Toolbar for backend debugging (e.g., inspecting database queries) and React Developer Tools (a Chrome extension) for debugging the frontend, enabling us to trace component states and props. Breakpoints and console logging were also used in VS Code for both backend and frontend debugging.

3.2 Device Interfaces and Equipment

This section details the physical hardware components of the system, their interfaces, and their roles in achieving the system's functionality, including devices used and supported during development and deployment.

- **Microcontroller**:
    - Arduino Nano: Used as the sensor node controller, powered by the ATmega328P microcontroller. It was chosen for its compact size, low cost, and sufficient I/O pins to interface with multiple sensors.
    - ESP32-WROOM-32: Acts as the gateway, receiving data from the Arduino Nano and uploading it to the cloud. The ESP32 was selected for its dual-core processor, built-in Wi-Fi, and support for multiple communication protocols.
- **Sensors**:
    - Capacitive Soil Moisture Sensor: Connected to the Arduino Nano's analog pin A0 to measure soil moisture levels via analog output (1.2V–3.0V).

    - DS18B20 Waterproof Temperature Sensor: Connected to digital pin D2 of the Arduino Nano using the OneWire protocol to measure soil temperature (-55°C to +125°C).
    - Soil NPK Sensor: Interfaces with the Arduino Nano via Modbus RS485 (using a MAX485 module) to measure Nitrogen, Phosphorus, and Potassium levels, communicating over UART (pins RX/TX).
- **Communication Modules**:
    - NRF24L01 Wireless Module: Facilitates wireless communication between the Arduino Nano (sensor node) and ESP32 (gateway) at 2.4GHz. It uses SPI (Serial Peripheral Interface) for communication with both microcontrollers (pins D11, D12, D13 on the Arduino Nano; GPIO 18, 19, 23 on the ESP32).
    - ESP32 Wi-Fi: The ESP32's built-in Wi-Fi module uploads data to ThingSpeak, using the HTTP protocol to send sensor readings to the cloud.
- **Devices Used and Supported:**
    - **Laptop**: A development laptop (e.g., Dell with Intel i5, 8GB RAM) was used for coding, compiling firmware with Arduino IDE, debugging via Serial Monitor, and deploying the backend and frontend using Visual Studio Code.
    - **Smartphone**: Used to access the dashboard via mobile browsers (e.g., Chrome on Android) for testing responsiveness and validating real-time data updates.

3.3 Hardware and Software Operating Environment

- **Hardware**:
    - Sensor Node: Arduino Nano with ATmega328P (16 MHz, 32 KB flash, 2 KB SRAM), powered via USB or a 9V battery.
    - Gateway: ESP32-WROOM-32 (dual-core 240 MHz, 4 MB flash, 520 KB SRAM), powered by a 5V external supply.

    - Sensors: Operate within specified voltage ranges (e.g., 3.3V–5V for moisture and temperature sensors, 9V–24V for the NPK sensor).

- **Operating System**:
    - The embedded system (Arduino Nano and ESP32) operates without an OS, using bare-metal programming for real-time performance.
    - The backend server runs on Ubuntu Server 20.04 LTS, providing a stable environment for the Django application and PostgreSQL database.
    - The frontend (React) is deployed on a serverless platform, and accessed via web browsers like Chrome.
- **Third-Party Software**:
    - ThingSpeak: Used for cloud storage and visualization of sensor data, accessible via API.
    - Chart.js: Integrated into the React frontend for rendering soil parameter graphs (e.g., moisture trends over time).
    - PostgreSQL v17.4: Manages user data, farm profiles, and historical sensor readings on the backend.

3.3.1.1 Coding Practices

- **Modularity**: Code was organized into separate modules for sensor reading, data transmission, and cloud integration. For example, sensor_reading.ino handles sensor data collection on the Arduino Nano, while cloud_upload.ino manages ThingSpeak uploads on the ESP32.
- **Error Handling**:
    - Embedded code includes checks for sensor failures (e.g., returning null for out-of-range values) and network disconnections (buffering data locally on the ESP32).

    - The Django backend uses try-catch blocks for database operations and validates API inputs.
    - The React frontend includes error boundaries to gracefully handle rendering issues.
- **Documentation**:
    - Embedded code uses inline comments and follows a consistent style (e.g., // Read soil moisture from A0).
    - Backend and frontend code uses JSDoc for JavaScript/TypeScript and docstrings for Python to document functions and classes.
- **Version Control**: All code was managed in a GitHub repository, with commits following a convention like feat: add NPK sensor integration or fix: resolve Wi-Fi reconnection issue.
- **Testing**: Unit tests were written for the Django backend using Python's unittest framework, and the frontend was tested using Jest and React Testing Library.

3.3.1.2 Artifacts Generated

- **Embedded Firmware**:
    - soil_sensor.ino: Firmware for the Arduino Nano to read data from moisture, temperature, and NPK sensors.
    - esp32_gateway.ino: Firmware for the ESP32 to receive data via NRF24L01 and upload to ThingSpeak.
- **Backend Code**:
    - Django project with models for users, farms, devices, and sensor readings.
    - API endpoints (e.g., /api/farms, /api/devices) for CRUD operations.
- **Frontend Code**:

    - React application with components for the dashboard, farm management, and notifications.
    - Graphs generated using Chart.js for visualizing soil trends.
- **Documentation**:
    - README.md with setup instructions.
    - User manual (Appendix A).
    - API documentation.

3.4 Documentation

3.4.1 Types of Documentation Provided

The project includes comprehensive documentation to support development, deployment, and user interaction:

Table 3.4.1: Documentation Provided:

#	**Document**	**Purpose**
1	Software Requirements Specification (SRS)	Baseline functional & non-functional requirements (Rev 0, 05, December 2024).
2	Software Design Document (SDD)	Architecture, sequence & deployment diagrams, database schema, SRS traceability.
3	Data-Collection Report	Raw & cleaned datasets from field trials, with statistical summaries.
4	Developer Guide	Details Thingspeak api reference endpoints, firmware structure, and setup.
5	Installation Guide	Steps for hardware assembly and software deployment.

6	Source Code Repository	Maintains versioned, structured codebase with comments and modular logic.
7	README.md file	Contains setup instructions.
8	User Manual	Guides end-users on system operation (Appendix A).
9	Installation Guide	Secure hardware setup, SIM provisioning, env-var templates, and dashboard installation.

3.4.2 Role of Documentation

- Facilitates understanding of system functionality for developers and stakeholders.
- Ensures consistency in coding standards and development practices.
- Enhances user experience by providing clear instructions for operation and troubleshooting.

Table 3.4.2: Design details

Topics	**Design output**	
Good programming practice *Efforts made to meet the recommendations for good programming practice...*	Source code is...	Source code contains...
	☑ Modulized	☐ Revision notes
	☑ Encapsulated	☑ Comments
	☑ Functionally divided	☑ Meaningfull names
	☐ Strictly compiled	☑ Readable source code
	☑ Fail-safe (handling errors)	☐ Printable source code

Topics	**Design output**
Dynamic testing *Step-by-step testing made dynamically during the implementation...*	☑ All statements have been executed at least once ☑ All functions have been executed at least once ☑ All case segments have been executed at least once ☑ All loops have been executed to their boundaries ☐ Some parts were not subject to dynamic test Comments: Employed dynamic testing methodologies to identify and rectify issues during the development phase.

4 Inspection and testing

4.1 Introduction

Inspection and testing are integral to ensuring the system meets specified requirements and functions reliably. A structured test plan was developed, encompassing various testing levels to address complexity, risk, and intended use.

Table 4.1.1: Inspection plan and performance

Topics	**3.3.1 Inspection plan and performance**	*Date / Initials*
Design output	☑ Program coding structure and source code ☑ Evidence of good programming practice ☑ Design verification and documented reviews ☐ Change-control reviews and reports Comments: Reviewed code structure, software design document, and requirements specification.	15/04/2025 / HP, MM, ME
Documentation	☑ System documentation, flow charts, etc. ☑ Test results ☑ User manuals, On-line help, Notes, etc. ☐ Contents of user manuals approved Comments: Manuals and supporting documents reviewed and found complete and user-friendly.	15/04/2025 / MM, HP

Topics	**3.3.1 Inspection plan and performance**	*Date / Initials*
Software development environment	☑ Data integrity ☑ File storage ☑ Access rights ☐ Code protection ☑ Installation kit, replication and distribution Comments: Environment was secure and well-configured for the project. All protocols followed.	15/04/2025 / MM, ME
Result of inspection	☑ Inspection approved Comments: All modules passed inspection. Testing covered core functionality and documented results were satisfactory.	15/04/2025 / HP, MM, ME

4.2 Test plan and performance

4.2.1 Test objectives

The primary objectives of the testing phase were:

- Functionality Verification: To confirm that the system operates as intended, meeting all specified functional requirements. Each feature was tested individually and then collectively to ensure cohesive operation.
- Data Visualization: To ensure that patient data can be effectively visualized through graphs based on regions and specified time ranges. This was tested by setting different time frames and verifying the graphical outputs.
- Requirement Compliance: To ascertain that all requirements outlined in the Software Requirements Specification (SRS) were fully implemented. This was achieved by cross-referencing the SRS with the system's functionalities.

4.2.2 Scope and Relevance of tests

In Scope:

- **Web Dashboard Accessibility:** The web dashboard is tested for accessibility and responsiveness on both desktop and mobile browsers.
- **User Authentication and Authorization:** Testing includes verifying user login, role-based access controls, and session management to ensure secure access to the system.
- **Data Visualization and Reporting:** The system's ability to generate reports and visualize data through graphs and charts is tested to confirm accurate representation of information.
- **PostgreSQL 17.4 Backend:** Testing includes all database operations such as data storage, retrieval, and management using PostgreSQL version 17.4.
- **Hardware Functionality**:
    - **Sensor Performance**: Testing ensures accurate readings from the capacitive soil moisture sensor, DS18B20 temperature sensor, and NPK sensor under various soil conditions (e.g., dry, wet, nutrient-rich).
    - **Microcontroller Operation**: Verifies the Arduino Nano and ESP32 functionality, including data processing and power management.
    - **Wireless Communication**: Confirms reliable data transmission between the Arduino Nano (via NRF24L01) and ESP32 gateway under different distances and interference levels.
    - **Hardware Durability**: Assesses the physical stability and environmental resilience of sensors and modules (e.g., resistance to moisture and temperature fluctuations).

Out of Scope:

- **Third-Party Integrations:** Interactions with external systems or third-party services are not part of the current testing activities.
- **Multi-Node Deployments:** The current testing phase does not cover notification functionalities or scenarios involving multiple IoT nodes operating simultaneously.

- **Extended Load Testing Beyond 24 Hours:** Stress testing or performance evaluation over extended periods exceeding 24 hours is not included in this testing scope.

Subsystem Coverage

Table 4.2.1: Subsystem Coverage

Subsystem	Coverage Details
Firmware	Sensor data acquisition and preprocessing Real-time clock synchronization Power management and sleep modes Error handling and recovery mechanisms
Backend	RESTful API endpoints for data ingestion and retrieval Business logic for data validation and processing Authentication and authorization services Logging and monitoring integrations
Database	Schema design and normalization Stored procedures and triggers for data integrity Backup and recovery procedures Performance tuning and indexing strategies
Dashboard	User interface components and navigation Data visualization using charts and graphs Responsive design for various screen sizes Accessibility features and compliance
Security	Encryption of data at rest and in transit Role-based access control mechanisms Audit trails and activity logging Vulnerability scanning and penetration testing

Deployment	Continuous integration and deployment pipelines Containerization using Docker Environment configuration and management

4.2.3 Levels of Tests

- **Module testing:** Ensured each software component performed its intended function.
- **Integration testing:** Verified seamless communication between IoT devices and backend.
- **System testing:** Validated overall performance and stability in a simulated real-world setup.

4.2.4 Types of Tests

- **Input validation tests:** These tests ensured that all data inputs from both users and sensors were validated correctly before processing. The system successfully rejected invalid entries and displayed appropriate messages.
- **Functionality tests:** This majorly focused on verifying that every feature performed its intended functionality as it was described in the SRS document. Ensured that functionalities like registration, login and farms, device registrations worked without failure. Triggered advisory generation by changing input parameters like soil condition and crop preference.
- **Performance tests:** A stress testing was performed to measure how the system handles load and resource usage under stress. We realized the system can maximally handle 1000 requests per minute.
- **Usability tests:** Evaluated how easily users could navigate the system and complete tasks. A group of target users (farmers and agricultural extension officers) was asked to perform common tasks such as Registering a new farm, viewing sensor data, Receiving crop recommendations.

4.2.5 Sequence of tests

Table 4.2.2: Sequence of tests

Test Case ID	***Test Procedure***	***Test Data***	***Result***
TC01	Simulate soil sensor sending data every 10 minutes	Soil Moisture = 45%, Temperature = 29°C	Data is received and stored in the database; dashboard updates in real-time
TC02	Enter invalid sensor values (e.g., letters instead of numbers)	Soil Moisture = "abc", Temperature = "NaN"	System rejects input and logs error; dashboard does not update
TC03	User logs in with correct and incorrect credentials	Username: farmer1 / Password: wrong123 and correctpass123	First attempt denied with message; second attempt logs user into dashboard
TC04	Generate crop advisory based on different soil profiles	Moisture: 70%, Temp: 22°C, pH: 6.5	System recommends moisture-tolerant crops suitable for pH 6.5
TC05	View dashboard on different screen sizes	Access web app from desktop, tablet, and smartphone	Interface adjusts responsively, no layout breakages
TC06	Sensor stops sending data (simulate network failure)	No sensor data received for 30 minutes	System displays warning “No data received”; sends alert to admin
TC07	User tries to access dashboard without login	Direct URL access without login	Redirect to login page; access denied
TC08	Run load test with 50 users simultaneously	50 simulated users with different farm profiles	System handles requests without crashing; response

	logging in and requesting advisories		time remains below 2 seconds
TC09	Submit form with missing required fields	Omit crop type or farm name	Form is rejected; validation messages are displayed
TC10	View historical data for the past 30 days	Select a past date range in dashboard filter	Graphs and tables correctly populate with archived sensor readings

4.2.6 Configuration and calculation tests

Table 4.2.3: Configuration and calculation tests

Test Category	Test Description	Platform/Environment	Expected Result
Platform Configuration	Verify that the system runs on Windows 10/11, Ubuntu 22.04 LTS	Web browser (Chrome/Firefox)	System loads without error, all features accessible across supported platforms
Network Configuration	Test system access over LAN and internet	Wi-Fi (home and institutional), localhost	System responds within 15 seconds; real-time data transmission confirmed
System Integration	Validate backend (Django) communication with PostgreSQL and IoT node	Backend server (Django), PostgreSQL DB	Sensor data is received, stored, and displayed on the dashboard in real-time

Calculation Test	Generate real-time usage graph from sensor data	Input: 24 hours of data	Graph plots correctly with accurate timestamps and values

4.3 Precautions

4.3.1 Anomalous conditions

- **Sensor reading latency** observed during simultaneous data transmission from multiple IoT devices.
- **Dashboard graph misalignment** when browser locale time zone differs from server configuration.
- **Occasional backend timeout errors** during high traffic simulation.
- **Frontend dropdown menu overlap issue** in certain browsers.
- **IoT device disconnection** in areas with unstable Wi-Fi signals.

4.3.2 Precautionary steps taken

- Optimized data transmission intervals and staggered polling per IoT node.
- Applied responsive web design principles for compatibility.
- Implemented caching and retry mechanisms for sensor data upload.
- Introduced fallback values and warnings when sensor readings are unavailable.
- Increased backend timeout threshold and added asynchronous task handling for large payloads.
- Updated CSS breakpoints and tested UI responsiveness across multiple devices.

4.4 Results and Approval

Summary of Testing:

- **Modules Tested**: Sensor data acquisition, backend API, admin dashboard, authentication, graph rendering, and user alerts

- **Environments**: Chrome, Firefox, Android browser, Django (v4.2), PostgreSQL (v14), Ubuntu Server 20.04

Test Execution Overview:

- Total Cases: 45
- Passed: 39
- Passed with Notes: 4
- Failed: 2

Remarks:
The failed tests were related to minor UI responsiveness on outdated browsers and a corner-case sensor reconnection scenario. These do not impact critical functionality or data integrity.

Approval Date: 3 May 2025

Release Tag: SmartSoil-prototype-v1.0-pilot

Decision: The system is cleared for controlled field deployment in partner testing zones. All tests, documentations, source code and validation results are archived under version v1.0 for future traceability and audit readiness.

Visualization of Test Results: During testing, the system successfully visualized soil sensor data on the dashboard, displaying graphs for soil moisture, temperature, and NPK levels. The image below shows the PC displaying these graphs, confirming the system's ability to process and present data effectively for user interpretation.

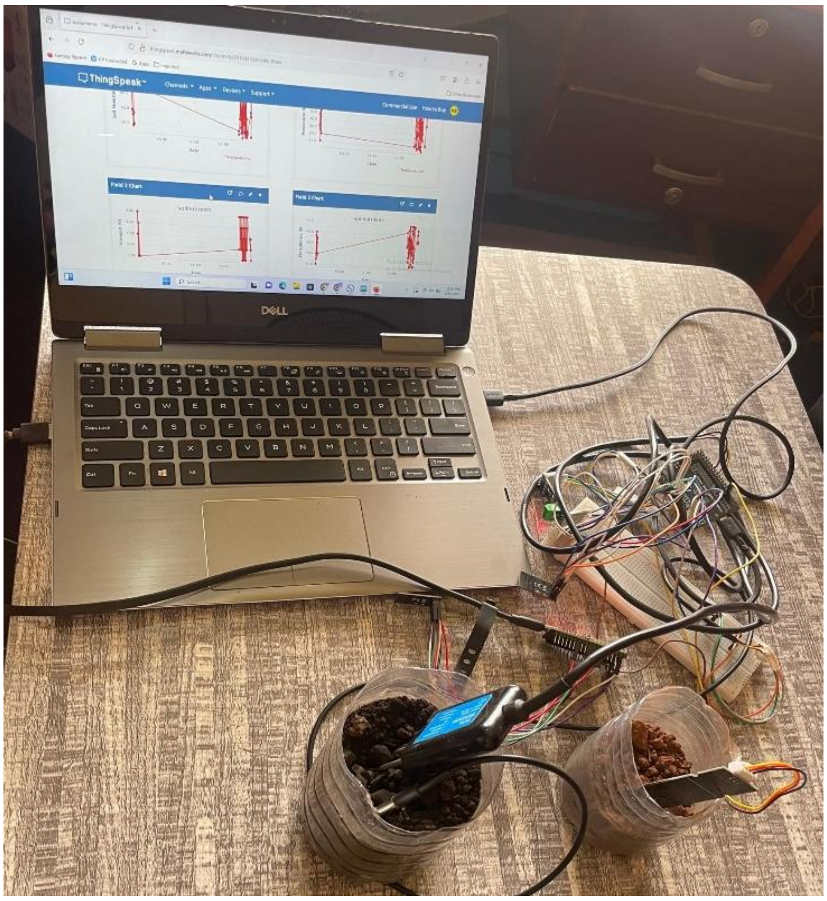


Figure 4.1: Real-life setup showing the PC displaying graphs of soil sensor data collected by the Smart Cloud-Based Soil Advisor system during testing with a sample soil.

5 Installation and system acceptance test

This chapter provides a clear and detailed explanation of how the Smart Cloud-Based Soil Advisor system is deployed, configured, and validated in its target operating environment. It outlines the components and files required for installation, describes the installation steps, and presents the procedures followed to verify that the system meets all functional and technical requirements. The goal is to ensure that all modules are properly integrated, operational, and safe for use in a real-world setting. Emphasis is placed on traceability, reproducibility, and system acceptance criteria based on defined test outcomes.

5.1 Input files

The following files were packaged in the installation archive and were essential in setting up the Smart Cloud-Based Soil Advisor system:

- setup_backend.sh: Shell script to set up the Django backend, configure environment variables, and initialize the database.
- client.zip: Compiled React frontend build used for UI rendering and interaction with the backend.
- requirements.txt: List of Python dependencies for the backend setup.
- config.env.example: Template file for environmental configuration variables.
- iot_firmware_v1.0.hex: Flashable firmware file for the IoT soil sensor node.
- db_init.sql: SQL file to initialize the PostgreSQL schema and seed base data.

5.2 Supplementary files

These auxiliary documents were included to support users, administrators, and developers:

- README.md: Overview of the system, including setup instructions and usage notes.
- LICENSE.txt: MIT license agreement for open-source use.
- changelog.txt: Describes changes across different software versions.
- example_config.json: Example configuration file for deploying the soil sensors.
- USER_MANUAL.pdf: End-user guide detailing how to navigate and use the system's dashboard.

5.3 Installation qualification

This section outlines steps to ensure the Smart Cloud-Based Soil Advisor system is installed correctly, validating functionality, safety, and compliance. The process is reproducible and traceable, with a checklist confirming success.

Steps:

1. **Hardware and Firmware Setup:**
   - ➢ Connect sensors (Moisture to A0, DS18B20 to D2, NPK via MAX485 to RX/TX) and NRF24L01 (D7, D8, D11-D13) to Arduino Nano. Wire ESP32 gateway with NRF24L01 (GPIO 4, 5, 18, 19, 23), power via USB. Upload iot_firmware_v1.0.hex using Arduino IDE (v2.3.2).
   - ➢ Verify via Serial Monitor (9600 baud): sensor readings (e.g., moisture: 60%, temp: 25.5°C, NPK: 180/22/120 mg/kg) and NRF24L01 transmission.
2. **Backend Setup:**
   - ➢ Set up Django + Celery + PostgreSQL + Redis backend using Render's render. yaml (Infrastructure as Code).
   - ➢ Configured web service, Celery worker, Celery beat, Redis broker, and Postgres DB, linked via environment variables.
   - ➢ Enabled automated deployments, background tasks, and scheduled jobs for farm data and crop recommendation system.
3. **Frontend Deployment:**
   - ➢ Extract client.zip, deploy to via GitHub (Node.js 18.x settings).
   - ➢ React app deployed as a static site on Render using the Static Site service.
   - ➢ Configured build command (npm run build) and publish directory (dist or build).
   - ➢ Connected to the Django back-end API via environment variables or direct API URL.
   - ➢ Verify: Access www.agrsns.com in Chrome, confirm dashboard loads.
4. **System Integration Test:**
   - ➢ Configure ESP32 with Wi-Fi and ThingSpeak API key, upload firmware, ensure nodes are within 100m range.
   - ➢ Verify: ThingSpeak updates (e.g., every 15 minutes) and dashboard reflects data within 30 seconds.

Table 5.3.1: Checklist of the Installation and system acceptance test

Topics	**Installation summary**
Installation method *Automatic or manual installation...*	☑ Automatic - installation kit located on the installation media ☐ Manual - Copy & Paste from the installation media Comments: Automatic installation via script and manual configuration for the backend
Installation media *Media containing the installation files...*	☐ Diskette(s) ☐ CD-ROM ☐ Source disk folder (PC or network) ☑ Download from the Internet Comments: The installation files are downloadable via the secure project repository.
Installed files *List of (relevant) installed files, e.g. EXE- and DLL-files, spreadsheet Add-ins and Templates, On-line Help, etc.*	• Python Files • .hex firmware • JSON configs • JSX build files • Environment templates

Table 5.3.2: Installation Procedure Check

Topics	**Installation procedure**	*Date / Initials*
Authorization *Approval of installation in actual environment.*	Person responsible: Mugoya Moses, Heli Prajapati	01/05/2025 / MM, HP

Topics	**Installation procedure**	*Date / Initials*
Installation test *The following installations have been performed and approved...*	☑ Tested and approved in a test environment ☑ Tested and approved in actual environment ☑ Completely tested according to test plan ☐ Partly tested (known extent of update) Comments: System passed all basic functionality tests.	01/05/2025 / MM, HP

6 Performance, servicing, maintenance, and phase out

This chapter outlines the ongoing operational support, routine servicing, and performance requirements of the Smart Cloud-Based Soil Advisor system. As an evolving platform deployed to assist farmers and agronomists with soil condition recommendations, the system requires proactive maintenance strategies, user support channels, and well-defined pathways for system upgrades, expansion, and eventual replacement.

6.1 Service and maintenance

To ensure reliable performance of the Smart Cloud-Based Soil Advisor system, the following servicing and maintenance activities are recommended:

- **Sensor and Device Maintenance:** Soil moisture, pH, and temperature sensors should be inspected and cleaned monthly. Calibration for analog sensors should be conducted bi-monthly using standardized reference soil conditions. Faulty sensors should be replaced immediately and logged via the maintenance interface.
- **Firmware and Software Updates:** Updates to the backend API and frontend dashboard are deployed using a CI/CD pipeline, with changelogs versioned and published in the system's documentation portal.
- **Cloud Integration Maintenance:** System integration with PostgreSQL, ThingSpeak, and other data pipelines should be reviewed quarterly to ensure smooth data ingestion, logging, and retrieval.
- **Support and Issue Tracking:** A centralized issue tracker (e.g., GitHub Issues) is used to monitor bugs, user-submitted issues, and enhancement requests. Support is divided into levels: Tier 1 (account setup, UI navigation), Tier 2 (sensor or network troubleshooting), and Tier 3 (data loss, corruption, or logic faults).

6.1.1 Performance and Support Requirements

The Smart Soil Advisor system is expected to meet the following performance benchmarks:

- **Data Ingestion Delay:** Sensor readings should be ingested and reflected in the dashboard within 30 seconds.
- **Advisory Generation Time:** Upon submission of a complete soil profile, the system must generate a recommendation within 20 seconds.
- **System Uptime:** The cloud backend must maintain 95.5% uptime monthly, with downtime events recorded and analyzed.

Support Strategy:

- **Documentation Access** including a quick-start tutorial, and instructional videos hosted on the system website.
- **Feedback Loop** to gather suggestions directly from end-users for future feature considerations.

6.1.2 System Upgrades and Expansion

Upgrades and future expansions are governed by structured release management and scalability goals:

- **Scheduled Feature Releases:** Feature updates are scheduled quarterly and managed under Git version control with semantic tagging (v1.1, v1.2, etc.).

- **Expansion Capability:** Support for additional IoT nodes (e.g., soil nutrient probes, rainfall sensors) will be integrated via modular firmware and API extension.

- **Platform Migration Support:** In the event of a backend switch (e.g., from PostgreSQL to another relational DB), migration scripts and validation checks will be implemented to maintain consistency and prevent data loss.

- **User Feedback Integration:** Suggested features such as SMS alerts or local language support will be evaluated for feasibility, prototyped, and released based on demand and system capacity.

6.1.3 Data Migration and System Phase-Out

To phase out an older version of the system or transition from an alternative soil monitoring tool, the following migration process is proposed:

i. **Data Export:** Export historical records (sensor readings, advisory logs) in CSV or JSON format from the legacy system.

ii. **Pre-Migration Check:** Validate schema compatibility and run field-matching scripts to detect discrepancies.

iii. **Parallel Deployment:** Operate both systems concurrently for one week, comparing outputs to ensure parity.

iv. **Final Transition:** Deactivate the legacy system after verification and compress its database for archival storage (retain for 3 years).

v. **Hardware Retirement:** Evaluate hardware for reuse. Clean and reassign working components; dispose of non-functional sensors and boards per e-waste guidelines.

vi. **Rollback Preparedness:** Maintain a rollback snapshot for at least 48 hours after migration in case of post-transition faults.

6.2 Performance and Maintenance

The system is engineered to deliver recommendations within 20 seconds after receiving complete sensor data inputs. This fast response supports timely interventions in crop management and irrigation.

Servicing includes sensor calibration, software patching, remote assistance for bug fixes or troubleshooting and checking integration points such as cloud data services and IoT firmware. Support is categorized to allow quick resolutions at different complexity levels.

System upgrades are introduced based on direct user feedback or emerging technological needs—such as the integration of AI-based error correction modules or improved management for IoT devices. Each release is documented and regression-tested prior to deployment.

Phase-out from legacy systems involves careful migration of historical data, operational overlap for validation, and eventual shutdown of the old setup, ensuring no disruption to user services or data continuity.

Table 6.2.1: Performance and maintenance details

Topics	**Performance and maintenance**	Date / Initials
Problem / solution	During early deployment, we noticed excessive resource usage due to the system	10/04/2025, M.E, M.M

	fetching sensor readings every 1 minute. We revised the interval to 1 hour to optimize performance. Additionally, the NPK sensor was occasionally producing false data; we traced this to unstable voltage levels and resolved it by calibrating and debugging the sensor firmware.	
Functional maintenance	In case of any system modification or feature change, including backend logic adjustments or frontend redesigns, will be announced to users via email 48 hours in advance.	28/04/2025, H.P, M.M
Functional expansion and performance im-provement	The following improvements are proposed for better system functionality: 1. Enable log-based alerts to monitor abnormal sensor readings. 2. Add multilingual support. 3. Implement progressive web app (PWA) capabilities to improve access on low-end devices.	01/05/2025, H.P, M.M

7 Conclusion and Recommendations

7.1 Conclusion

The Smart Cloud-Based Soil Advisor System has successfully met its primary objective of offering an accessible, efficient, and soil monitoring solution for Ugandan farmers. Built using a combination of IoT sensor nodes, a secure backend infrastructure, and a responsive web dashboard, the system enables users to monitor critical soil parameters such as temperature, soil moisture, and NPK levels from any location with internet access.

Throughout the development lifecycle, all major subsystems from the sensor firmware to the React-based frontend and Django backend were designed, implemented, and validated against defined performance and reliability requirements. Testing procedures ensured proper integration between hardware and software components, and system behavior was verified under various real-world conditions to ensure durability and functionality.

Key achievements include:

- Accurate soil condition readings transmitted via Wi-Fi to a centralized database.
- A user-friendly web dashboard accessible on desktop and mobile browsers.
- Soil and Crop Recommendations.
- Consistent data logging with minimal loss during connectivity disruptions.

Despite these successes, challenges such as occasional sensor misreading, delayed data uploads, and UI rendering inconsistencies were encountered and addressed through software patches, enhanced validation logic, and improved deployment practices.

Overall, the system has proven to be reliable and adaptable for deployment in low-resource agricultural settings, empowering farmers with real-time insights that can guide better land management and crop decisions.

7.2 Recommendations

To further improve the Smart Cloud-Based Soil Advisor System, the following recommendations are proposed:

1. **Improve Sensor Calibration and Accuracy:** Sensor performance, particularly with NPK readings, should be periodically verified with standard soil sample tests to ensure continued accuracy.

2. **Introduce a Machine Learning Model for Error Correction:** Incorporate a lightweight machine learning component in future versions to compare sensor data against verified soil sample inputs and correct for known inconsistencies or anomalies. This can help reduce false readings and improve overall diagnostic accuracy.

3. **Incorporate GPS Tagging for Readings:** Adding GPS functionality would allow for location-based soil mapping, enabling region-specific insights for farmers and agricultural officers.

4. **Optimize Power Management:** Since the devices are often deployed in the field, integrating solar charging or optimizing for low-power operation would enhance system longevity and reduce maintenance needs.

5. **Provide Deployment and Usage Guides:** To ensure proper setup and interpretation, clear user manuals or video tutorials should be created and distributed alongside the hardware kits.

6. **Diagnostic Recommendations and Feedback Loop:** A feedback mechanism should be added to allow users (especially agronomists) to flag unusual readings or manually input lab results. This helps improve ML model correction over time.

7. **Environmental Compliance Integration:** Consider integrating national agricultural regulations and fertilizer usage guidelines to ensure the recommendations comply with Uganda’s environmental policies.

8. **Design for Long-Term Scalability:** Ensure the backend architecture supports multi-node scalability, data export for research purposes, and integration with other environmental monitoring systems in the future.

8 References:

[1] Uganda Revenue Authority, "General Agricultural Sector Guide," 2022. [Online]. Available: https://ura.go.ug/en/general-agricultural-sector-guide/. [Accessed: 21-Apr-2025].

[2] World Bank, "Closing the Potential-Performance Divide in Ugandan Agriculture: Fact Sheet," 2018. [Online]. Available: https://www.worldbank.org/en/country/uganda/publication/closing-the-potential-performance-divide-in-ugandan-agriculture-fact-sheet. [Accessed: 21-Apr-2025].World Bank Group+1World Bank Group+1

[3] Daily Express, "Agriculture: The Base for Uganda's Economy," 2024. [Online]. Available: https://dailyexpress.co.ug/2024/11/06/agriculture-the-base-for-ugandas-economy/. [Accessed: 21-Apr-2025].Daily Express

[4] Saerd-Tech Consultants, "Monitoring & Sensing Systems – Smart Agriculture, Environmental Technologies & Research Development," 2024. [Online]. Available: https://saerdtechco.com/en/agric-iot-solutions/. [Accessed: 22-Apr-2025].

[5] Nile Post, "Jaguza AI Chatbot: Empowering Ugandan farmers!," 2023. [Online]. Available: https://nilepost.co.ug/agriculture/173010/jaguza-ai-chatbot-empowering-ugandan-farmers. [Accessed: 22-Apr-2025].

9 Appendices A

9.1 User Manual (User Interface and Operations)

About This Manual

This manual serves as a comprehensive guide for using the Smart Cloud-Based Soil Advisor system. It is designed for farmers, agricultural extension officers, and system administrators who intend to use the system for monitoring soil health and receiving tailored crop recommendations.

It outlines the hardware setup, system architecture, connectivity, dashboard usage, maintenance guidelines, and troubleshooting steps. This manual covers everything from powering the sensors to navigating the web interface and interpreting data.

Project GitHub Repository: https://github.com/HarrisonMoses/BSE25-28-CBSA

Blog Link: https://fyp-blog.onrender.com/

Thingspeak API: https://api.thingspeak.com/channels/2912443/feeds.json?api_key=7SEG1UOO84U6C6QF

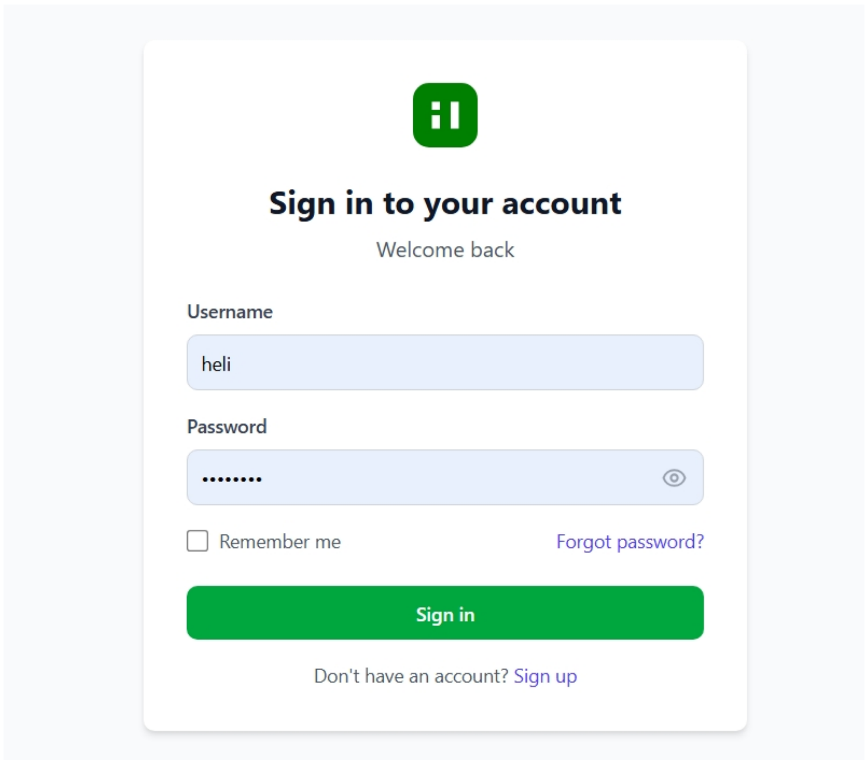
1. Getting Started

1.1 Sign in to Your Account

To access the platform:

- Open the application and navigate to the **Sign in** page.
- Enter your **Username** and **Password**.
- Optionally, check the **Remember me** box to save your login details.
- Click the green **SIGN IN** button.
- If you forget your password, click **Forget password?** to reset it.

- New user? Click Sign up to create an account.


Sign in to your account

Welcome back

Username: heli

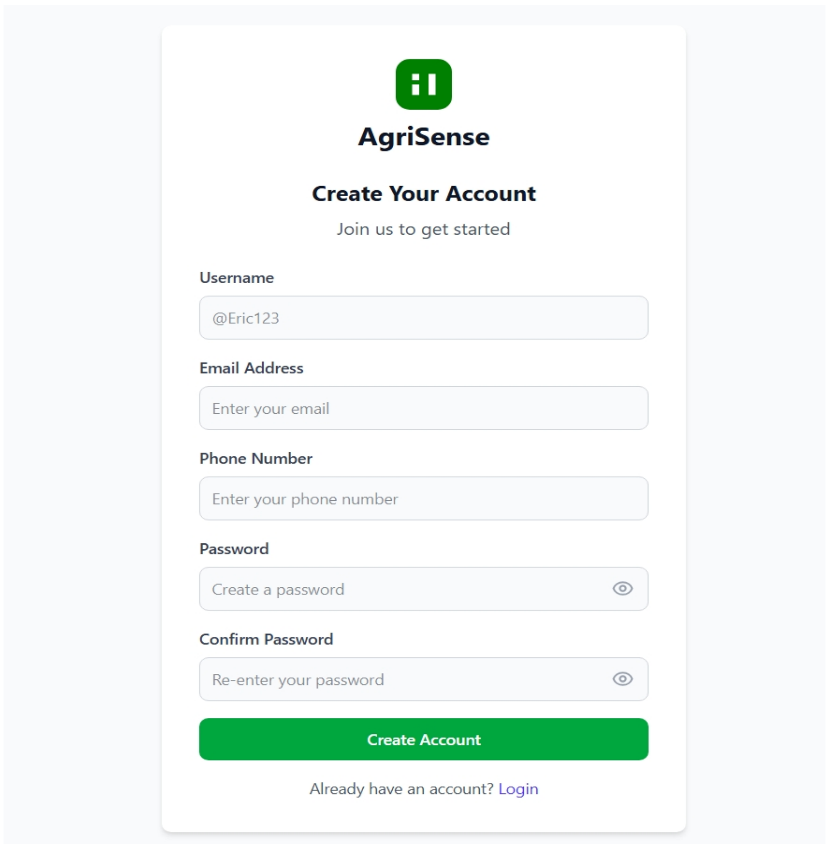
Password: ••••••••

Figure 9.1: Sign In Screen

1.2 Create a New Account

To create an account:

- Navigate to the **Create Your Account** page.
- Fill in the following fields:
    - **Username**: Choose a unique username (e.g., @eric123).
    - **Email Address**: Enter your email.
    - **Phone Number**: Provide your contact number.
    - **Password**: Create a secure password.
    - **Confirm Password**: Re-enter your password to confirm.
- Click the green **CREATE ACCOUNT** button.
- Already have an account? Click **LOGIN** to sign in.


AgriSense

Create Your Account

Join us to get started

Username

@Eric123

Email Address

Enter your email

Phone Number

Enter your phone number

Password

Create a password

Confirm Password

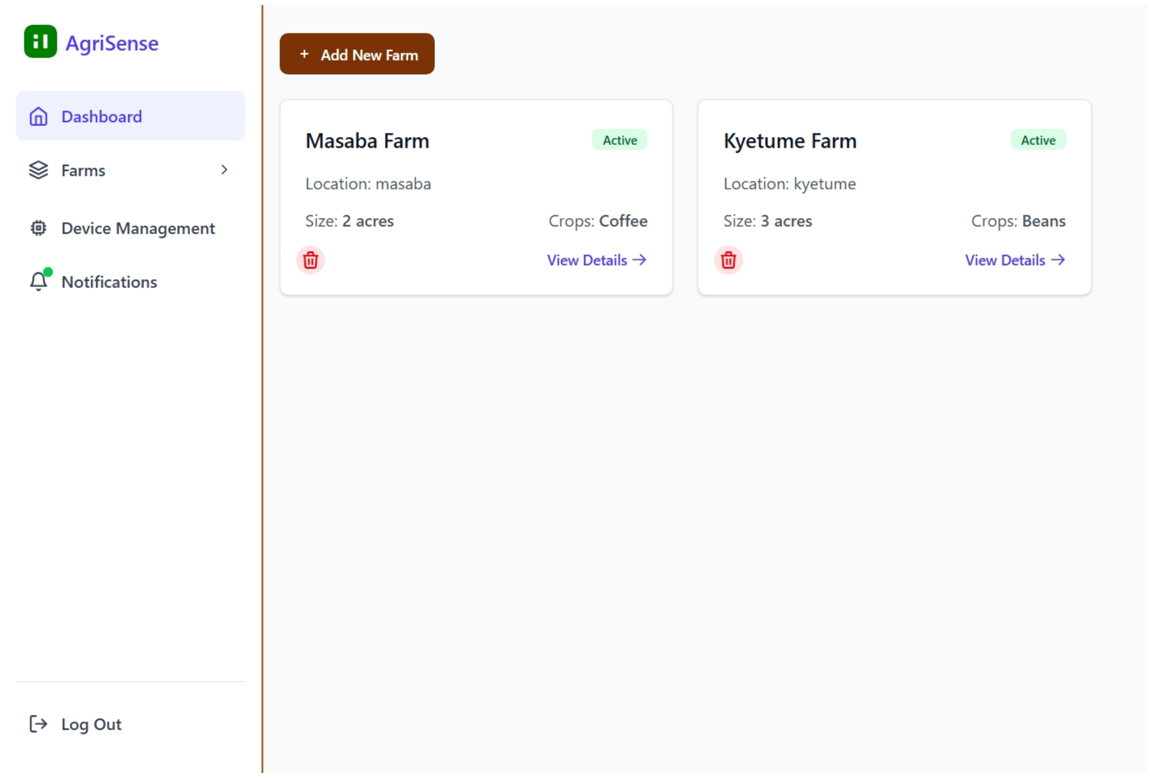
Re-enter your password

Create Account

Already have an account? Login

Figure 9.2: Register Screen

2. Managing Farms



Masaba Farm Active

Location: masaba

Size: 2 acres Crops: Coffee

View Details →

Kyetume Farm Active

Location: kyetume

Size: 3 acres Crops: Beans

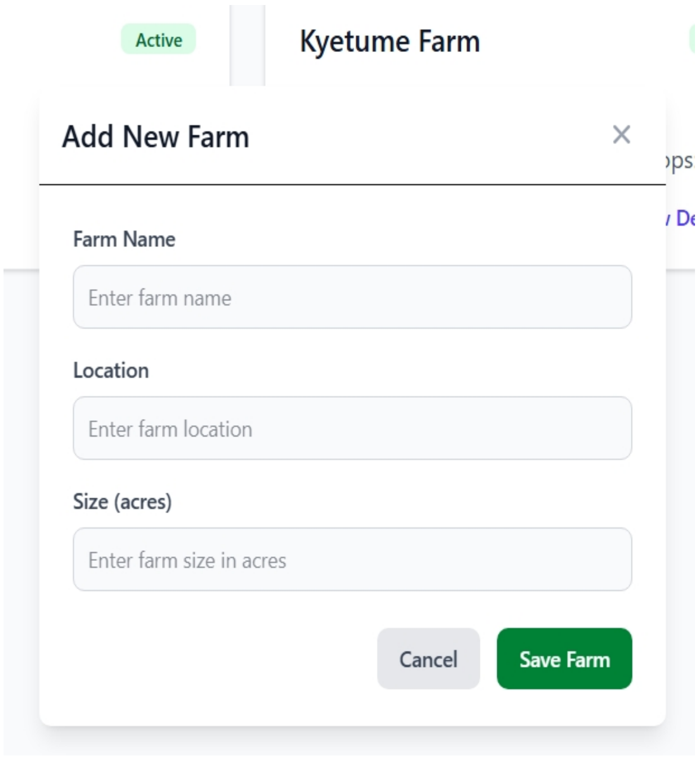
View Details →

Figure 9.3: Farm Dashboard Screen

2.1 Add a New Farm

To add a farm:

- From the **Farms** section, click the brown **+ Add New Farm** button.
- In the pop-up window, enter:
    - **Farm Name**: Name your farm (e.g., Masaba Farm).
    - **Location**: Specify the farm's location (e.g., masaba).
    - **Size (acres)**: Enter the farm size in acres (e.g., 2 acres).
- Click the green Save Farm button to save, or Cancel to discard.


Active

Kyetume Farm

Add New Farm

Farm Name

Enter farm name

Location

Enter farm location

Size (acres)

Enter farm size in acres

Figure 9.4: Add Farm Screen

2.2 View Farm Details

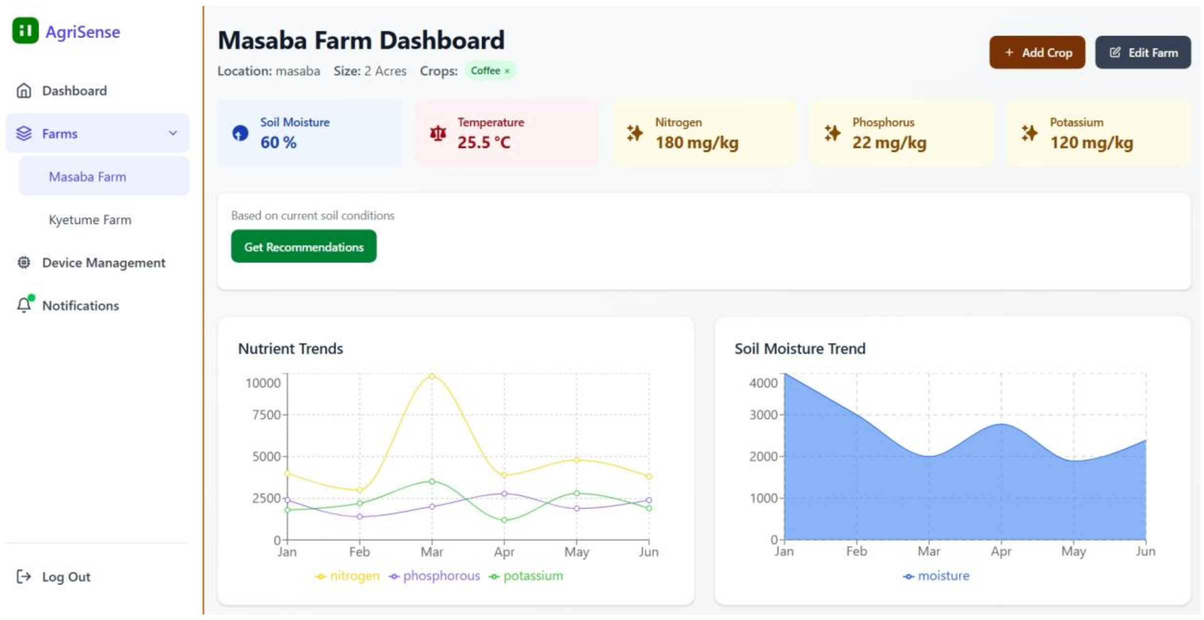
In the **Farms** section, you'll see a list of farms (e.g., Masaba Farm, Kyetume Farm).

Each farm card displays:

- Farm name, location, size, and crops.
- Status (e.g., Active).

Click **View Details** to see more information about a farm.

To delete a farm, click the red trash icon.



Masaba Farm Dashboard

Location: masaba Size: 2 Acres Crops: Coffee ×

Soil Moisture 60 %

Temperature 25.5 °C

Nitrogen 180 mg/kg

Phosphorus 22 mg/kg

Potassium 120 mg/kg

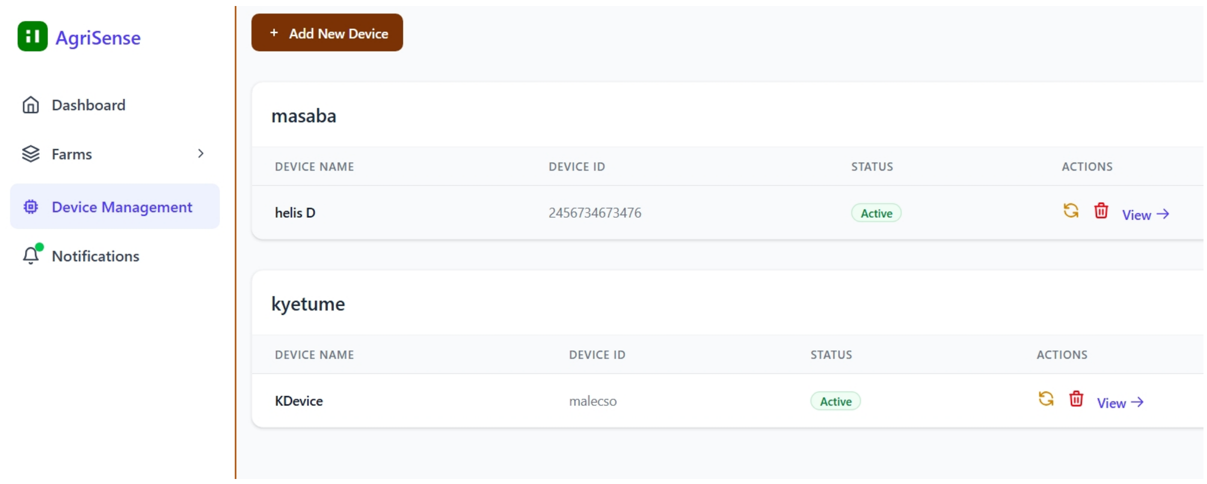
Based on current soil conditions

Figure 9.5: View Farm Details Screen

2.3 Edit a Farm

- From the farm dashboard (e.g., Masaba Farm Dashboard), click the **Edit Farm** button.
- Update the farm details as needed and save the changes.

3. Managing Devices



masaba

DEVICE NAME	DEVICE ID	STATUS	ACTIONS
helis D	2456734673476	Active	View →

kyetume

DEVICE NAME	DEVICE ID	STATUS	ACTIONS
KDevice	malecso	Active	View →

Figure 9.6: Device Management Screen

3.1 Add a New Device

To add a device to a farm:

- From the **Device Management** section, click the brown **+ Add New Device** button.
- In the pop-up window, enter:
  - **Device Name**: Name your device (e.g., helis D).
  - **Device ID**: Enter the unique device ID (e.g., 2456734673476).
  - **Select Farm**: Choose a farm from the dropdown (e.g., masaba).
- Click the green **Register Device** button to save, or **Cancel** to discard.

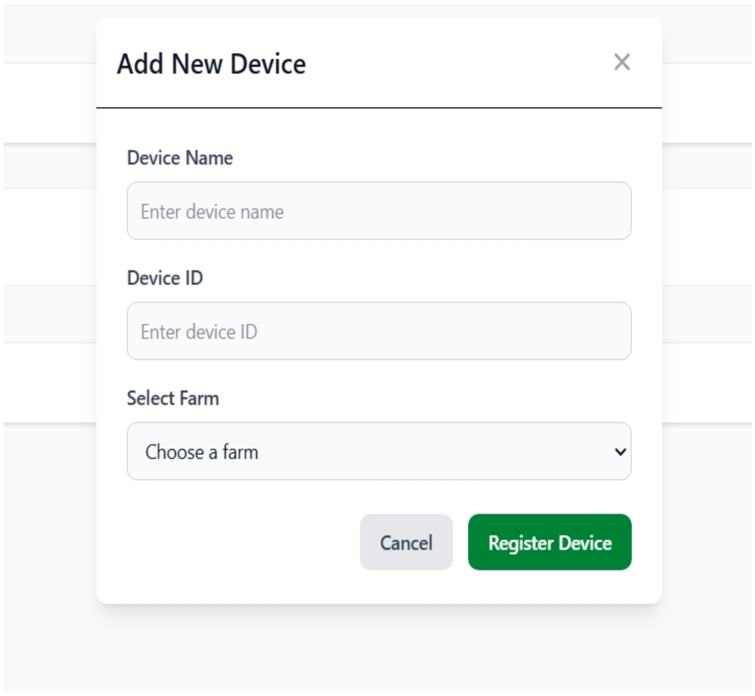


Figure 9.7: Add Device Screen

3.2 View Device Details

- In the **Device Management** section, devices are listed under their respective farms.
- Each device entry shows:
    - Device Name (e.g., helis D).
    - Device ID (e.g., 2456734673476).
    - Status (e.g., Active).
- Click **View** to see device details, or use the red trash icon to delete a device.

4. Monitoring Farm Conditions

4.1 Farm Dashboard

The farm dashboard provides a snapshot of soil conditions for a selected farm (e.g., Masaba Farm Dashboard):

- **Soil Moisture**: Displays the current soil moisture percentage (e.g., 60%).
- **Temperature**: Shows the current temperature (e.g., 25.5°C).
- **Nutrient Levels**: Displays levels of Nitrogen (e.g., 180 mg/kg), Phosphorus (e.g., 22 mg/kg), and Potassium (e.g., 120 mg/kg).
- **Trends**:
    - **Nutrient Trends**: A line graph showing Nitrogen, Phosphorus, and Potassium levels over time (e.g., Jan to Jun).
    - **Soil Moisture Trend**: A graph showing soil moisture levels over time (e.g., Jan to Jun).

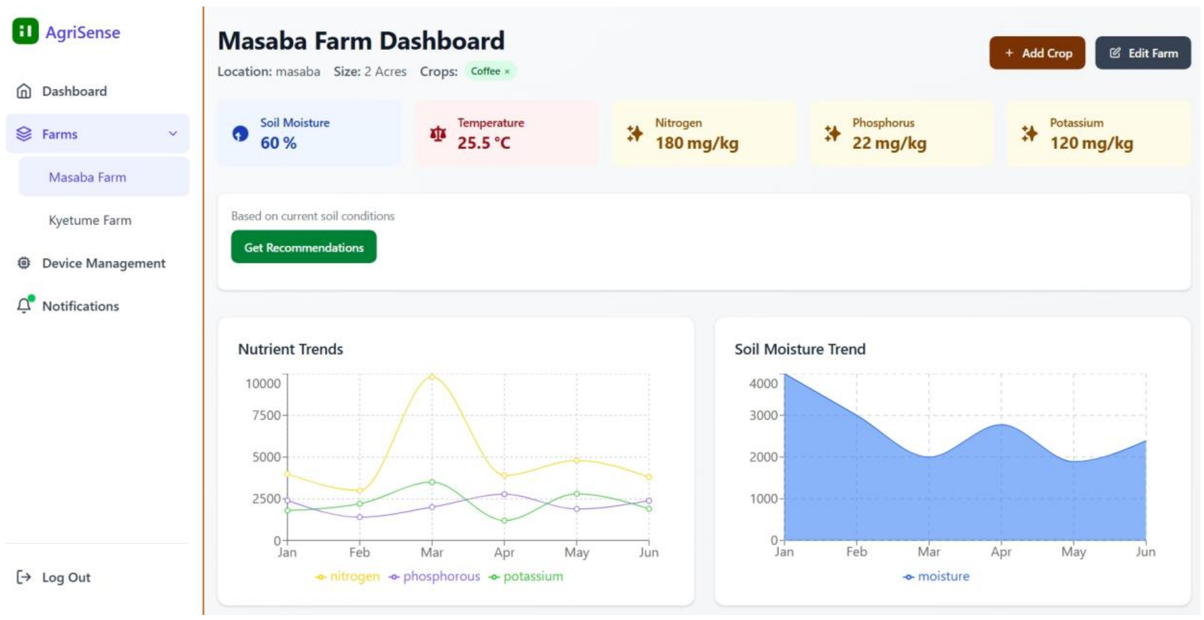


Figure 9.8: Farm Dashboard Screen

4.2 Crop Recommendations

- Based on current soil conditions, the dashboard suggests suitable crops (e.g., Tomatoes, Corn, Lettuce, Peppers).
- Click **Get Recommendations** to view more details.

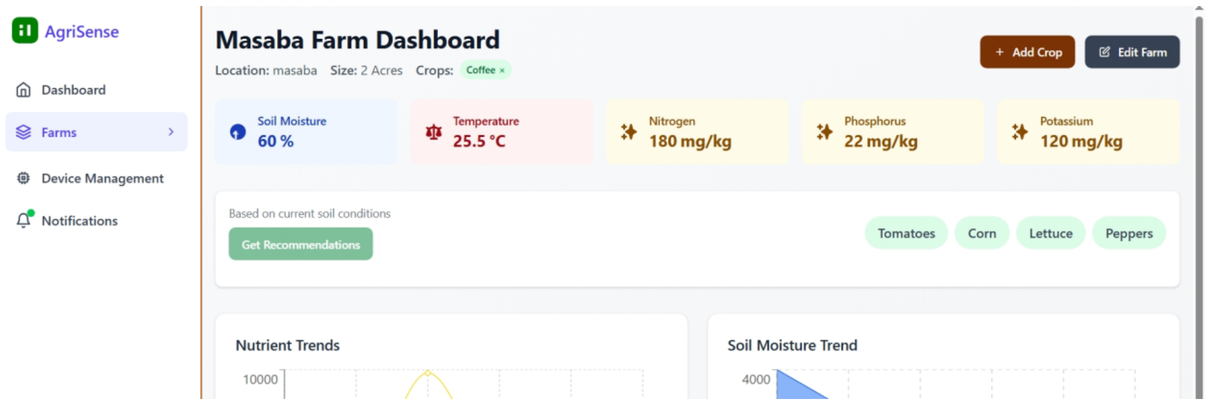


Figure 9.9: Crop Recommendations Screen

4.3 Add a New Crop

To add a crop to a farm:

- From the farm dashboard, click the brown **+ Add Crop** button.
- In the pop-up window, select a crop from the dropdown (e.g., Coffee).
- Click the red **Add Crop** button to save, or **Cancel** to discard.
- Added crops are displayed on the dashboard (e.g., Crops: Coffee).

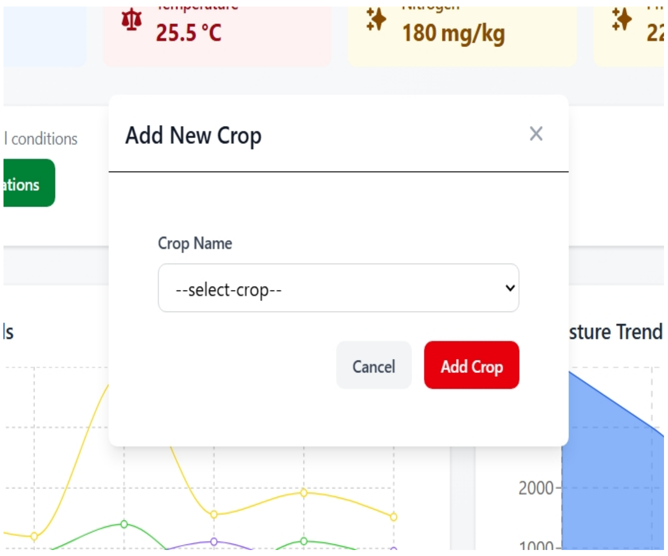


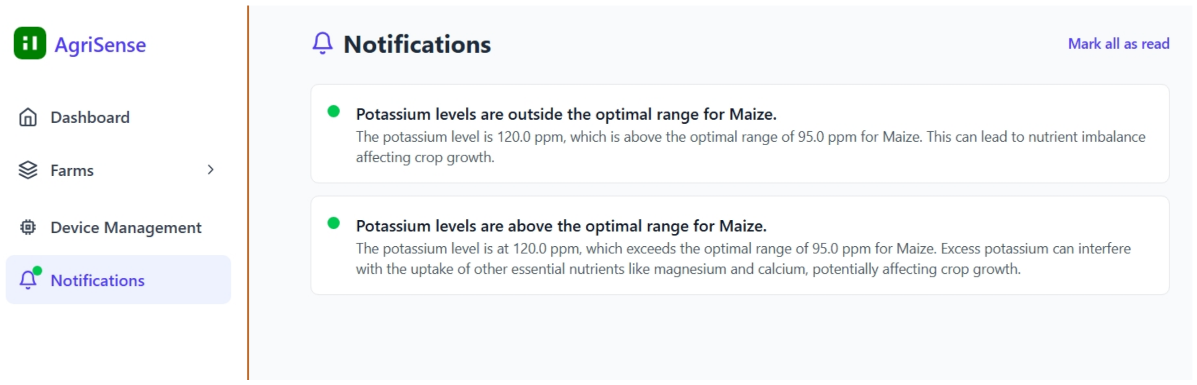
Figure 9.10: Add Crops Screen

5. Notifications

5.1 View Notifications

- Navigate to the **Notifications** section.

- Notifications alert you to important updates, such as:
    - Potassium levels being outside the optimal range for a crop (e.g., 120 ppm, above the optimal 95 ppm for Maize).
- Click **Mark all as read** to clear unread notifications.



Potassium levels are outside the optimal range for Maize.
The potassium level is 120.0 ppm, which is above the optimal range of 95.0 ppm for Maize. This can lead to nutrient imbalance affecting crop growth.

Potassium levels are above the optimal range for Maize.
The potassium level is at 120.0 ppm, which exceeds the optimal range of 95.0 ppm for Maize. Excess potassium can interfere with the uptake of other essential nutrients like magnesium and calcium, potentially affecting crop growth.

Figure 9.11: Notification Screen

6. Logging Out

To log out, click the **Logout** button at the bottom of the sidebar.

9.2 System Overview

The Smart Cloud-Based Soil Advisor is an integrated IoT system that collects real-time soil nutrient and environmental data using sensors and sends it to a cloud platform (ThingSpeak). The backend receives the data from the thingspeak api, processes it and formats the data to displays actionable insights on a web dashboard.

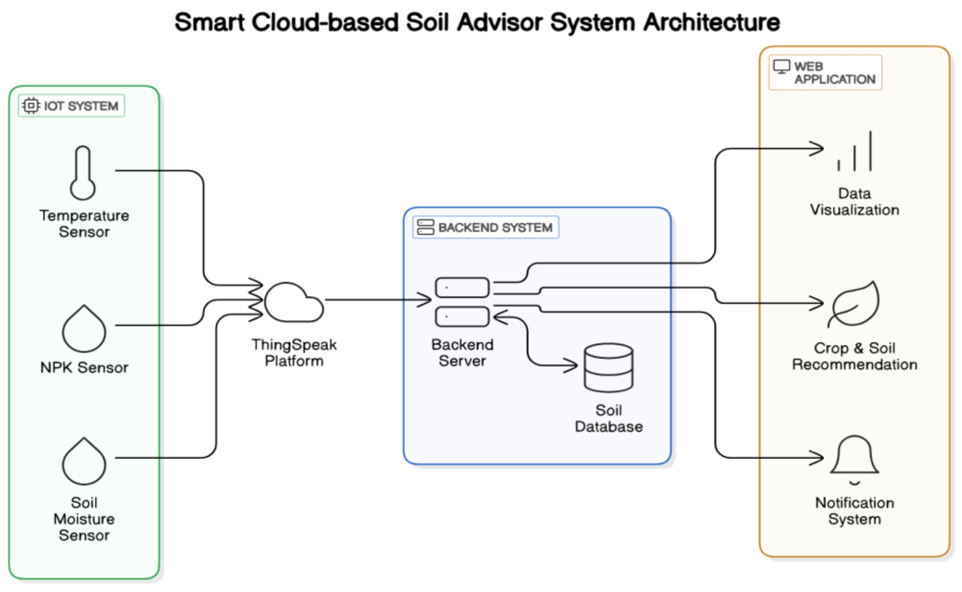


Figure 9.12: System Architecture Diagram

Key Components:

- **IoT Sensor Node:** Measures soil moisture, temperature, and nutrient (NPK) levels.
- **Gateway (ESP32):** Receives sensor data and uploads to the cloud.
- **ThingSpeak Platform:** Stores and visualizes real-time data.
- **Backend Server & Database:** Processes data, stores historical readings.
- **Web Interface:** Displays insights, provides recommendations and alerts to users.

9.3 Hardware Setup

9.3.1. Capacitive Soil Moisture Sensor

Purpose: Measures soil water content.

Voltage: 3.3V–5.5V

Output: Analog (1.2V–3.0V)

Connection: Arduino Nano analog pin A0.

Setup Steps:

i. Connect VCC to 3.3V on Arduino Nano, GND to GND.

ii. Connect OUT to A0 on Arduino Nano.

iii. Insert the probe into the soil (up to the marked line).
iv. Power on the Arduino and check readings via Serial Monitor (expected: 100-200).

Note: Keep the sensor’s electronics dry.

9.3.2. DS18B20 Waterproof Temperature Sensor

Purpose: Measures soil temperature.
Range: -55°C to +125°C.
Libraries: DallasTemperature, OneWire.
Pull-up Resistor: 4.7kΩ (DATA to VCC).
Connection: Digital pin D2 on Arduino Nano.

Setup Steps:

i. Install DallasTemperature and OneWire libraries in Arduino IDE.
ii. Connect VCC to 5V, GND to GND, DATA to D2 on Arduino Nano.
iii. Add a 4.7kΩ resistor between DATA and VCC.
iv. Place the probe in soil (5–10 cm deep).
v. Upload a test sketch and check Serial Monitor (expected: 20–30°C).

Note: Ensure the probe’s waterproof seal is intact.

9.3.3. Soil NPK Sensor

Purpose: Measures Nitrogen, Phosphorus, Potassium.
Connection: Modbus RS485 via MAX485 module to Arduino.
Voltage: 9V–24V
Resolution: 1 mg/kg.

Setup Steps:

i. Connect NPK sensor’s A and B to MAX485 A and B.
ii. Wire MAX485: VCC to 5V, GND to GND, RO to RX (pin 0), DI to TX (pin 1), RE/DE to D3 on Arduino Nano.
iii. Power the NPK sensor with 12V; connect GND to Arduino GND.
iv. Insert the probe into soil (10–15 cm deep).

v. Install ModbusMaster library, upload a test sketch, and check Serial Monitor (expected: N: 100–200, P: 20–50, K: 80–150 mg/kg).

Note: Use a stable 12V power supply.

9.3.4. NRF24L01 Wireless Module

Purpose: Sends sensor data wirelessly to the ESP32 gateway. **Frequency**: 2.4GHz.

Setup Steps:

i. **Arduino Nano (Sensor Node)**: Connect VCC to 3.3V, GND to GND, CE to D7, CSN to D8, SCK to D13, MOSI to D11, MISO to D12.
ii. **ESP32 (Gateway)**: Connect VCC to 3.3V, GND to GND, CE to GPIO 4, CSN to GPIO 5, SCK to GPIO 18, MOSI to GPIO 23, MISO to GPIO 19.
iii. Install RF24 library, upload test sketches to both devices (same channel/address).
iv. Check Serial Monitor on ESP32 for received data (expected: moisture, temp, NPK values).

Note: Use 3.3V power for NRF24L01 to avoid damage.

9.3.5. ESP32 Gateway

Purpose: Receives data via NRF24L01 and uploads to ThingSpeak using Wi-Fi.

Setup Steps:

i. Connect NRF24L01 to ESP32 (as above).
ii. Add Wi-Fi credentials and ThingSpeak API key to the sketch.
iii. Install ESP32 board, WiFi, and ThingSpeak libraries in Arduino IDE.
iv. Upload the sketch to ESP32.
v. Check Serial Monitor (115200 baud) for Wi-Fi connection and data upload (expected: “Uploaded to ThingSpeak”).

Note: Ensure a stable USB power source for the ESP32.

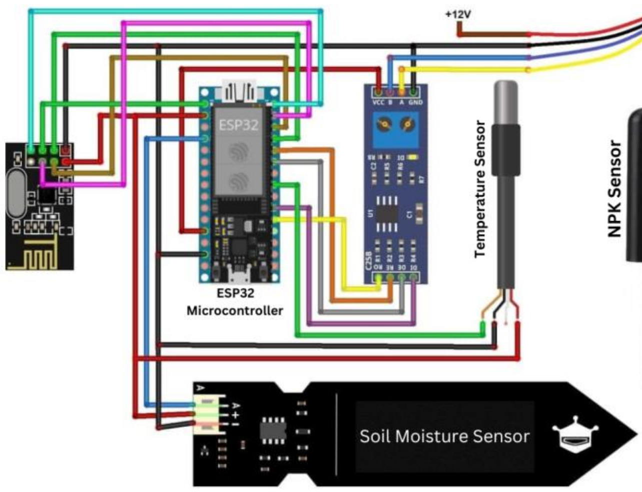


Figure 9.13: Sensor Node Circuit

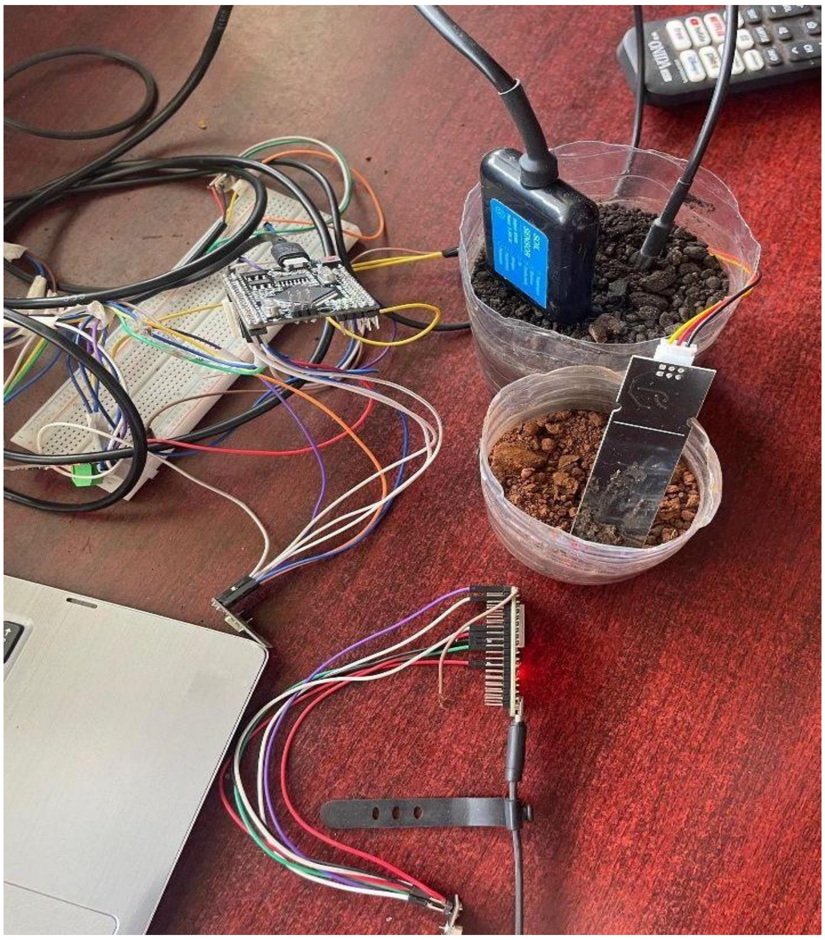


Figure 9.14: Real-life hardware setup of the Smart Cloud-Based Soil Advisor system, tested with a sample soil in a controlled environment, featuring sensors connected to the Arduino Nano and ESP32 gateway.

9.4 Firmware and Connectivity

9.4.1. Uploading Firmware

- Connect Arduino Nano and ESP32 to your PC via USB.
- Open Arduino IDE.
- Select the correct board and port for both the Arduino Nano and ESP32.
- Upload the sensor firmware (provided in GitHub repo).

9.4.2. Connecting to ThingSpeak

i. Go to ThingSpeak.com
ii. Create a channel and note your API key.
iii. Enter this key in the ESP32 firmware before uploading.

9.5 Maintenance & Calibration

- **Soil Moisture Sensor:** Clean sensor head monthly to avoid salt build-up.
- **NPK Sensor:** Recalibrate once every 6 months for accuracy.
- **Firmware Updates:** Check GitHub for the latest firmware.
- **Power Supply:** Ensure reliable power for uninterrupted data transmission.

9.6 Troubleshooting

Table 9.6.1: Troubleshooting

Issue	Possible Cause	Solution
No data on dashboard	ESP32 not connecting to Wi-Fi	Re-enter SSID and password in code
Sensor reads 0	Loose connection or faulty sensor	Check wiring, replace sensor if needed

ThingSpeak not updating	Wrong API key or quota exceeded	Verify API key, upgrade account

9.7 Frequently Asked Questions

Q1: Can the system work offline?

A: No, the ESP32 requires an internet connection to upload data to ThingSpeak.

Q2: How often is data sent to the cloud?

A: Data is sent every 60 minutes to reduce bandwidth and avoid sensory overload.

Q3: Can I integrate other sensors like EC?

A: Yes, as long as they communicate over I2C, UART, or analog, they can be added.

9.8 Safety

- Disconnect all power sources before wiring or making hardware changes.
- Handle NPK sensors carefully—excess moisture may affect calibration.
- Use protective gloves when handling soil samples or exposed circuitry.

Final approval for use	
Identification:	
Responsible for validation:	
Remarks:	
Date:	Signature: