Agri-Lens

Graduation project

2024 - 2025

AbdelAleem Mohamed
Mohamed Tharwat
Ibrahim Mohamed
Zeyad Emad
Basel Mohamed

Ahmed Ashraf
Ahmed Osama
Ibrahim Saber
Ibrahim Hegazi
Ahmed Khalf

A Real-Time Smart Farming Platform for Plant Health Monitoring

CS -22

Supervised By

Dr. Samir El-Mougy

Eng. Mai Mohamed

Team Members

Name	Role
AbdelAleem Mohamed	Business analyst
Mohamed Tharwat	UI-UX Designer
Ibrahim Mohamed Yousef	Front-End Developer
Zeyad Emad	Front-End Developer
Basel Mohamed	Flutter Developer
Ahmed Ashraf	Flutter Developer
Ahmed Osama	Back-End Developer
Ibrahim Saber	Back-End Developer
Ibrahim Mohamed Hegazi	AI model builder
Ahmed Khalf	AI model builder

Team leader: Ibrahim Hegazi

Abstract

"Smart Agricultural System for Intelligent Crop Monitoring and Management"

In response to global challenges in food security and sustainable agriculture, this project leverages smart agricultural technologies, IoT-enabled sensors, and AI-driven analysis to enhance modern farming practices. The system integrates smart monitoring boxes equipped with environmental sensors and cameras with an intelligent software platform that enables real-time monitoring, automated data analysis, and predictive insights. By collecting crucial environmental data—such as temperature, humidity, soil moisture, and light intensity—the system provides a detailed analysis of plant growth conditions. Additionally, AI-powered image processing techniques detect pests, diseases, and nutrient deficiencies to ensure early intervention.

This project demonstrates that AI-driven agricultural monitoring significantly improves farm efficiency by offering precise recommendations, early warnings, and optimized resource allocation compared to traditional methods. Predictive analytics help forecast potential risks, suggest corrective actions, and enable data-driven decision-making to maximize crop yield while minimizing resource waste. Beyond enhancing productivity, this work highlights the transformative potential of smart agriculture in promoting sustainability, reducing environmental impact, and addressing the increasing global food demand through scalable, technology-driven solutions.

Acknowledgments

We would like to express our deep gratitude to our supervisor **Dr. Samir El-Mougy** for his valuable guidance, support and encouragement throughout the course of this project. His expertise and insight have played an instrumental role in shaping the direction and outcome of this project.

We are also extremely grateful to **Eng. Mai Mohamed** for her continuous support and constructive feedback. Her technical expertise and dedication have greatly contributed to the successful completion of this work. In addition, we would like to thank our families, friends and colleagues for their continuous support and encouragement. Without their help, this project would not have been possible.

Table of Contents

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Introduction

Have you ever imagined a future where farming is effortless, crops thrive under optimal conditions, and agricultural challenges are tackled with smart technology? Traditional farming often struggles with unpredictable weather, inefficient resource management, and disease outbreaks that threaten crop yields. But what if there was a smart agricultural system designed to simplify farming, enhance productivity, and provide real-time insights to help farmers make informed decisions? Look no further!

Our intelligent agricultural platform is designed to be efficient, data-driven, and farmer-friendly. By integrating IoT sensors, AI-driven analysis, and automated monitoring, we provide farm owners with the tools they need to track crop health, optimize irrigation, and detect diseases early. With our state-of-the-art technology, real-time alerts, and intuitive platform, farmers can minimize losses, improve yields, and ensure sustainable farming practices.

Join us in revolutionizing agriculture—explore the features, embrace the innovation, and take control of your farm's future with confidence. Remember, you're not alone in this journey—our smart system is here to guide and support you every step of the way!

1.2 Problem Statement

Problem Area 1: Inefficient and Unoptimized Farming Practices.

- Traditional farming methods rely on manual monitoring, which can lead to **delayed detection of plant diseases, inefficient irrigation, and poor resource management**.
- Farmers often **lack real-time data** on soil conditions, temperature, humidity, and pest infestations, making it difficult to make informed decisions.
- Overuse or underuse of **water, fertilizers, and pesticides** can lead to **crop damage, soil degradation, and increased costs**.

Problem Area 2: Lack of Real-Time Monitoring and Predictive Analysis.

- Farmers struggle to **predict crop health issues and environmental risks** before they become severe.
- Traditional farming lacks **automated, real-time monitoring systems**, making it difficult to track plant growth, detect anomalies, or optimize harvesting times.
- Unpredictable weather conditions, combined with a **lack of data-driven insights**, lead to **reduced productivity and financial losses**.

Problem Area 3: High Dependency on Manual Labor and Expertise.

- Small and medium-scale farmers **lack access to advanced agricultural technologies**, making them reliant on **traditional, labor-intensive methods**.

- The **shortage of skilled agricultural workers** and **high operational costs** further complicate farm management.
- Farmers often face **knowledge gaps** in applying modern techniques due to limited access to **agronomic expertise and training resources**.

Problem Area 4: Limited Integration of Smart Technology in Agriculture.

- Existing agricultural solutions often lack **affordable, scalable, and easy-to-use smart systems** tailored for farm owners.
- Many farms do not have **IoT-enabled devices, AI-driven analytics, or automated irrigation and pest control systems**, limiting their efficiency.
- **Lack of seamless connectivity and integration with mobile applications** makes remote farm management challenging.

Summary of the Problem Statement:

Modern agriculture faces numerous challenges, including **inefficient resource management, unpredictable environmental factors, dependency on manual labor, and limited technological integration**. Farmers struggle to **track plant health, optimize irrigation, and make data-driven decisions**, leading to **low yields and increased costs**. This project aims to address these issues by developing an **AI-powered smart agricultural system**, leveraging **IoT sensors, predictive analytics, and automated monitoring** to revolutionize **farming efficiency and sustainability**.

1.3 Main Project Objectives.

1. Optimize Agricultural Efficiency Through Smart Technology.

- Implement **AI-driven monitoring** to provide real-time insights into **soil health, temperature, humidity, and pest activity**.
- Develop an **automated alert system** that notifies farmers of potential crop diseases or environmental risks.
- Enable **predictive analytics** to forecast crop growth, yield potential, and optimal harvesting times.

2. Enhance Resource Management and Reduce Waste.

- Introduce **smart irrigation systems** that optimize water usage based on real-time soil moisture levels.
- Utilize **sensor-driven automation** to ensure precise application of fertilizers and pesticides, reducing excess use and environmental impact.
- Improve overall **farm productivity and sustainability** by minimizing **manual intervention and resource wastage**.

3. Increase Accessibility and Usability for Farmers.

- Design a **user-friendly mobile application** that allows farmers to **remotely monitor and control their farms**.
- Ensure the system is **affordable, scalable, and easy to integrate** for both small-scale and large-scale agricultural operations.
- Provide **multilingual support and intuitive dashboards** to cater to a diverse range of farmers with varying levels of technical expertise.

4. Support Data-Driven Decision-Making.

- Enable **farmers to make informed decisions** based on **historical data trends and real-time analytics**.
- Develop **AI-powered recommendations** for **crop rotation, fertilization schedules, and pest control strategies**.
- Implement **machine learning models** to continuously improve prediction accuracy based on environmental factors and past outcomes.

5. Foster Sustainability and Reduce Environmental Impact.

- Promote **precision farming techniques** to **minimize chemical overuse and reduce carbon footprint**.
- Encourage the use of **renewable energy-powered sensors and automation tools** to lower overall energy consumption.
- Provide insights into **sustainable farming practices**, helping farmers maintain **long-term soil fertility and ecosystem balance**.

This project aims to **revolutionize traditional farming practices by integrating AI, IoT, and data analytics** to create a **smart agricultural system that enhances efficiency, reduces waste, and promotes sustainable farming methods**.

1.4 Project Significance

The significance of our project lies in its ability to **revolutionize agricultural practices** by integrating smart technologies. The system is designed to **enhance efficiency, reduce waste, and improve decision-making** for farm owners. The project offers four key features: **Automated Monitoring, Predictive Analysis, Smart Resource Management, and Remote Control & Alerts**. Below is a detailed breakdown of each feature:

1.4.1 Automated Monitoring Feature:

The **Automated Monitoring** feature utilizes **IoT-based smart sensors and AI-driven analytics** to continuously track crucial environmental factors such as **soil moisture, temperature, humidity, and pest activity**. These sensors provide **real-time data**, allowing farm owners to **monitor their fields remotely via the mobile application**. The system collects, processes, and **visualizes the data through intuitive dashboards**, ensuring that farmers can easily **detect early signs of diseases, pest infestations, or suboptimal conditions**. The **integration of camera-based monitoring** further enhances oversight, enabling users to **capture images and analyze plant growth stages**.

1.4.2 Predictive Analysis Feature:

The **Predictive Analysis** feature leverages **machine learning algorithms** to analyze **historical and real-time data** to **forecast crop yields, detect potential threats, and optimize planting schedules**. By studying past trends, weather conditions, and soil health, the system **generates recommendations for optimal planting times and crop rotation strategies**. Additionally, **AI-powered pest and disease detection models** help farmers take **preventive measures** before issues escalate. This feature enables farm owners to **enhance productivity, reduce losses, and maximize yield potential**.

1.4.3 Smart Resource Management Feature

The **Smart Resource Management** feature focuses on **efficient use of water, fertilizers, and pesticides** through precision farming techniques. The system **automatically adjusts irrigation schedules based on real-time soil moisture levels**, ensuring that crops receive the **optimal amount of water without overuse**. Similarly, **fertilizer and pesticide application** is controlled through automated dispensers, ensuring **minimal waste and maximum efficiency**. This feature not only helps **cut down costs** but also **reduces the environmental impact** of farming by preventing excessive use of agricultural chemicals.

1.4.4 Remote Control & Alerts Feature:

The **Remote Control & Alerts** feature allows farm owners to **manage their entire agricultural system via a mobile application**. Users receive **instant notifications** about critical conditions, such as **low soil moisture, extreme weather conditions, or early signs of plant diseases**. The app also enables farmers to **remotely activate irrigation systems, adjust environmental controls, or shut down certain operations if necessary**. This feature ensures that farmers can **take quick action to prevent losses**, even if they are not physically present on the farm.

Each of these features contributes to the overall **efficiency, sustainability, and profitability** of modern farming. By integrating **AI, IoT, and data-driven decision-making**, this project empowers farmers to **make better choices, improve productivity, and reduce waste**, ultimately transforming traditional agricultural practices into **a highly optimized and technology-driven industry**.

1.5 Project Scope:

Our **intelligent agricultural management system** is a **comprehensive solution** designed to revolutionize modern farming practices. By integrating **AI-driven analytics, real-time monitoring, and smart automation**, the system empowers farm owners with **precise, data-backed decision-making tools**. Whether you're a **small-scale farmer looking to optimize resources** or a **large agricultural enterprise seeking efficiency and sustainability**, our platform provides the technology to **enhance productivity, reduce waste, and maximize yield**.

Harness the Power of AI-Driven Insights:

Say goodbye to uncertainty and **embrace precision farming** with our advanced **AI-powered analytics**. Our system **processes real-time and historical agricultural data**, offering **actionable insights** into **optimal planting schedules, pest control measures, and crop health predictions**. The platform's **smart recommendation engine** guides users through **data-backed farming strategies**, reducing guesswork and **enhancing crop efficiency**.

Empower Farmers with Smart Automation

Our platform **automates crucial agricultural operations** using **IoT-enabled sensors and AI-based control systems**. From **automated irrigation systems** that adjust based on **real-time soil moisture levels** to **precision-based pesticide and fertilizer distribution**, we ensure that every resource is **used efficiently**. This approach **minimizes costs, reduces environmental impact, and maximizes yield**.

Stay Connected with a Collaborative Farming Network:

Farming is not just about crops; it's about **community and knowledge sharing**. Our platform provides **an interactive space** where farmers can **discuss best practices, troubleshoot issues, and seek expert advice**. Through **forums, live discussions, and direct messaging**, users can **share their experiences, ask questions, and exchange insights**.

Simulate and Optimize Farming Strategies – Risk-Free:

Before applying new farming techniques in the real world, our **simulation environment** allows users to **experiment with different agricultural strategies** using virtual resources. This **risk-free testing zone** lets farm owners **analyze potential outcomes** without incurring **real financial losses**. Learn by **testing new crop rotation techniques, irrigation methods, and pest control strategies** to ensure the best results before implementation.

Implementation Plan:

The project will be launched in multiple **successive versions** to ensure scalability and accessibility:

1. **Web Application** – A **comprehensive dashboard** for managing farming data, monitoring sensors, and accessing AI-driven recommendations.
2. **Mobile Application** – A **user-friendly mobile app** providing **real-time alerts, remote farm management, and community interactions**.

By combining **smart agriculture, AI-driven decision-making, and real-time monitoring**, our platform **paves the way for the future of farming**, making **precision agriculture accessible to all**.

1.6 Project Timeline

After making a project plan for our project, we found that the time required to complete the project at all stages of its construction about one year to carry out the tasks to be achieved.

Phase 1: Agile Planning and Requirement Gathering

Sprint 1-2

Objective: Define the project scope, gather requirements, and outline the design for the smart farming system in an iterative manner.

Tasks:

1. **Define Project Scope and Initial Backlog**
    - Identify core features and break them into backlog items.
    - Prioritize features for early MVP development.
2. **Research and Feasibility Study**
    - Conduct feasibility analysis for hardware and software components.
    - Evaluate AI models for disease detection and predictive analytics.
3. **Create High-Level Architecture and Initial Prototypes**
    - Draft architecture and select primary technologies.
    - Develop a rough prototype to validate technical feasibility.

Deliverables:

- Agile backlog and prioritized feature list.
- Initial prototypes for key components.
- High-level system architecture.

Phase 2: Iterative System Design and Prototyping

Sprint 3-5

Objective: Design hardware and software iteratively, integrating feedback from each sprint.

Tasks:

1. **Sprint-Based Hardware & Software Design**
    - Design core components in small iterations.
    - Validate designs through continuous testing.
2. **Backend & API Development**

- Develop core backend services with API integrations.
- Ensure real-time data handling.

3. **Mobile & Web App Development**
   - Create incremental UI/UX mockups.
   - Implement real-time monitoring features progressively.
4. **AI Model Development**
   - Train models iteratively with initial datasets.
   - Validate model performance and refine algorithms.

Deliverables:

- Continuous updates on hardware and software components.
- Working backend API with real-time data.
- UI/UX mockups and mobile/web app progress.
- Early AI model iterations.

Phase 3: MVP Development & Testing

Sprint 6-8

Objective: Develop a functional MVP for testing and feedback.

Tasks:

1. **Assemble Functional MVP**
   - Integrate core hardware and software.
   - Conduct initial field testing.
2. **Cloud Integration & AI Model Deployment**
   - Implement cloud-based storage for IoT data.
   - Deploy AI model for real-time monitoring.
3. **User Testing & Feedback Loop**
   - Conduct testing with stakeholders.
   - Adjust based on early user feedback.

Deliverables:

- MVP for real-world testing.
- Cloud-integrated data storage.
- AI model deployed and tested.

Phase 4: Full System Integration & Optimization

Sprint 9-10

Objective: Finalize integrations, optimize system performance, and address user feedback.

Tasks:

1. **Final Assembly & Refinements**
    - Optimize PCB and hardware assembly.
    - Enhance movement and control systems.
2. **Software and AI Enhancements**
    - Improve app performance and add new functionalities.
    - Train AI models with more data for better accuracy.
3. **Scalability & Performance Testing**
    - Simulate real-world high-load scenarios.
    - Optimize system response times.

Deliverables:

- Fully optimized hardware and software.
- AI model improvements and refined UI.
- Performance testing reports.

Phase 5: Final Testing & Deployment

Sprint 11-12

Objective: Ensure system reliability and prepare for real-world deployment.

Tasks:

1. **Comprehensive System Testing**
    - Validate core functionality, connectivity, and AI accuracy.
    - Address any remaining bugs or issues.
2. **Deployment & Training**
    - Deploy system for real-world usage.
    - Provide training for users on functionalities.

Deliverables:

- Fully validated system.
- Deployment and training materials.

Phase 6: Continuous Monitoring & Improvements

Post-Sprint Cycles

Objective: Implement Agile maintenance and feature updates.

Tasks:

1. **Post-Launch Support & Bug Fixes**
    - Monitor system performance and fix critical issues.
2. **Feature Enhancements & Scalability**
    - Plan new feature releases based on user feedback.
    - Expand system capabilities for wider adoption.

Deliverables:

- Continuous software updates.
- Post-launch reports and user analytics.

Phase 7: Deployment and Maintenance

Objective: Deploy the system in real-world conditions and ensure ongoing performance, updates, and support.

Tasks:

1. **System Deployment**
    - Deploy the smart farming system in a controlled environment.
    - Set up network configurations for IoT communication.
    - Verify real-time data transmission and cloud integration.
2. **User Training & Documentation**
    - Provide training for farm owners and system users.
    - Develop user manuals and troubleshooting guides.
3. **Performance Monitoring**
    - Monitor sensor readings and AI performance.
    - Identify and resolve issues with real-time alerts.
4. **System Updates & Optimization**
    - Improve AI model accuracy through continuous learning.
    - Optimize backend processes for efficiency.
    - Release software updates based on user feedback.
5. **Long-Term Maintenance & Support**
    - Implement periodic calibration of sensors.
    - Provide technical support for hardware and software components.
    - Establish a plan for future scalability and enhancements.

Deliverables:

- Deployed smart farming system.
- User training materials and guides.
- Performance reports and improvement recommendations.

- Maintenance plan and update schedule.

Agile Implementation Notes:

- **Daily Standups:** Ensure ongoing alignment and quick issue resolution.
- **Sprint Reviews & Retrospectives:** Improve workflow efficiency with every sprint.
- **Flexible Scope Adjustments:** Adapt project based on new insights and priorities.

1.7 Document Organization

This document is structured into **six comprehensive chapters**, each contributing to a **holistic understanding of how our intelligent agricultural system optimizes modern farming practices** through **AI-driven insights, automation, and real-time monitoring**. Below is an outline of each chapter:

- **Chapter 1: Introduction** – This chapter provides an overview of the project, outlining its **objectives, scope, and significance**. It defines the **core problem statement** and presents the **research questions** that guide the system's development.

- **Chapter 2: Literature Review** – This chapter explores existing **agricultural management techniques, smart farming technologies, and AI applications in agriculture**. It also reviews **previous research, industry trends, and case studies** to establish a foundation for the project. Additionally, it identifies **gaps in current agricultural solutions** and highlights **the need for a more advanced AI-driven farming system**.

- **Chapter 3: System Analysis** – This chapter conducts an in-depth **analysis of current farming challenges, system requirements, and constraints**. It outlines the **functional and non-functional requirements** of the system, detailing the **challenges faced by farmers** and how our **smart agricultural system addresses them** through **sensor-based monitoring, AI predictions, and automation**.

- **Chapter 4: System Design** – This chapter defines the **overall architecture, system modules, components, and their interactions**. It describes how different **hardware (sensors, cameras) and software components** are integrated to provide **real-time monitoring, data analytics, and predictive insights**.

- **Chapter 5: Hardware Implementation** – This section details AgriLens's physical components forming its IoT layer, beginning with the ESP32 microcontroller integrated with environmental sensors like three soil moisture sensors and a DHT11 for data acquisition. It outlines the actuators, including mini submersible water pumps for automated irrigation and a Nema 17 stepper motor for precise camera positioning. Finally, it describes the ESP32-CAM's integration for image capture (noting its limitations and the plan for external high-quality camera inputs), alongside the system's power management and protective "smart boxes."

- **Chapter 6: Software Implementation** – AgriLens's software is a multi-layered system where ESP32 firmware handles sensor data, actuator control, and image capture with built-in interference mitigation. A .NET Core backend manages data ingestion, integrates the YOLOv8 AI model

for disease detection from augmented images, and provides APIs for data retrieval and system commands. Finally, the Flutter-based mobile app offers an intuitive interface for real-time monitoring, historical data, system control, and crucial disease alerts.

- **Chapter 7: Testing and Results –** The AgriLens project successfully delivered a robust smart farming solution, proving its integrated IoT, AI, backend, and mobile app layers through comprehensive testing. The system achieved highly reliable sensor data (e.g., soil moisture ±5%), precise automated irrigation (3-second activation, significant water reduction potential), and strong AI-powered disease detection (YOLOv8 mAP@0.5 of 70.3%) on high-quality images. Despite challenges like sensor interference and ESP32-CAM image quality, the system's overall performance validates its potential to enhance crop health and optimize agricultural practices.

- **Chapter 8: Conclusion and Future Work** – This chapter summarizes the project's **findings, contributions, and impact on modern farming**. It also outlines **future enhancements**, such as **expanding AI models, integrating machine learning for crop disease prediction, and adding blockchain for secure agricultural transactions**.

References:

A comprehensive list of **academic sources, research papers, industry reports, and relevant studies** that contributed to the project's development.

Chapter 2
Literature Review

Introduction

This chapter provides a comprehensive review of the literature related to smart agricultural systems, including background information on smart farming techniques, tools, and technologies required to build our system. Additionally, it examines relevant existing works in this domain and concludes with a comparative analysis between our proposed system and similar existing systems.

2.1 Background

2.1.1 Vertical Farming

Vertical farming represents a revolutionary approach to agriculture where crops are grown in stacked layers, maximizing space efficiency and yield potential. This method has gained significant attention due to:

- Increasing urbanization and limited agricultural land
- Need for sustainable food production
- Climate change impacts on traditional farming
- Growing demand for locally-sourced produce

2.1.2 Plant Disease Detection

Traditional plant disease detection relies heavily on visual inspection by experts, which is:

- Time-consuming
- Subject to human error
- Often detected too late for effective treatment
- Expensive for large-scale implementation

2.1.3 Artificial Intelligence in Agriculture

The integration of AI in agriculture has revolutionized farming practices through:

- Computer vision for crop monitoring
- Machine learning for disease prediction
- Deep learning for automated diagnosis
- Data analytics for yield optimization

2.2 Literature Review

2.2.1 Automated Vertical Farming Systems

Several notable projects have explored automated vertical farming:

PlantVillage Project (2020)

- Developed by Penn State University
- Focus: Disease detection in various crops
- Technology: CNN-based image recognition
- Limitation: Static camera system

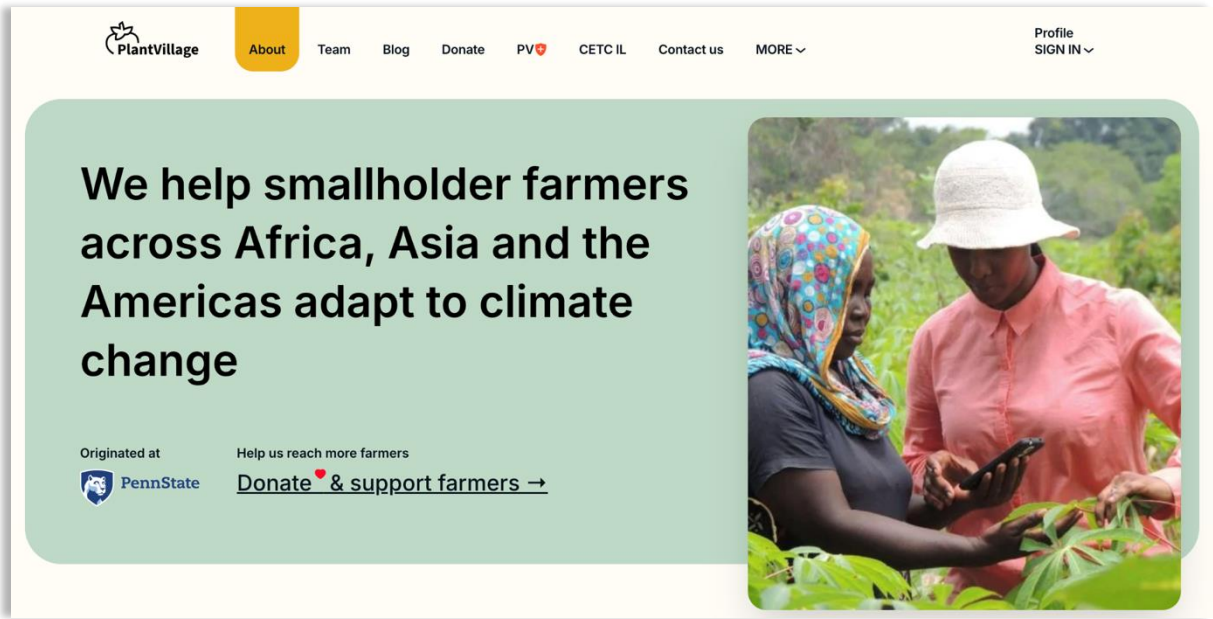


Figure 1-PlantVillage Project (2020)

AeroFarms Smart Farm (2021)

- Commercial vertical farming system
- Focus: Automated environment control
- Technology: IoT sensors and machine learning
- Limitation: Limited disease detection capabilities

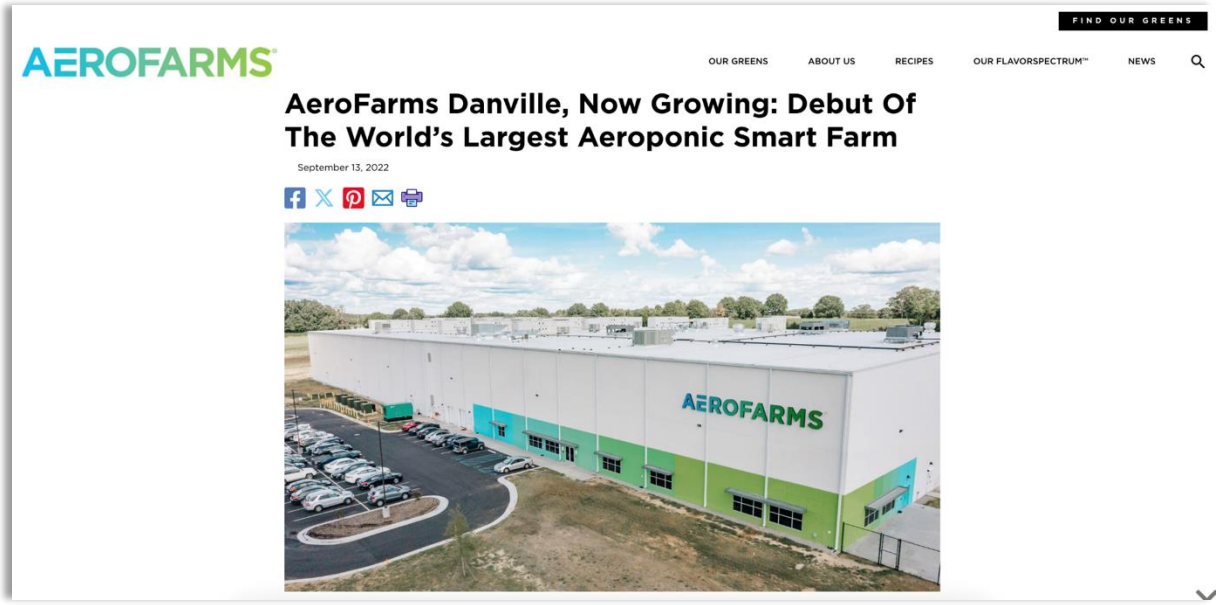


Figure 2 - AeroFarms Smart Farm (2021)

2.2.2 Plant Disease Detection Systems

DeepPlant (2019)

- Research project by MIT
- Technology: Deep learning with ResNet architecture
- Accuracy: 93.4% in controlled environments
- Limitation: Requires manual image capture

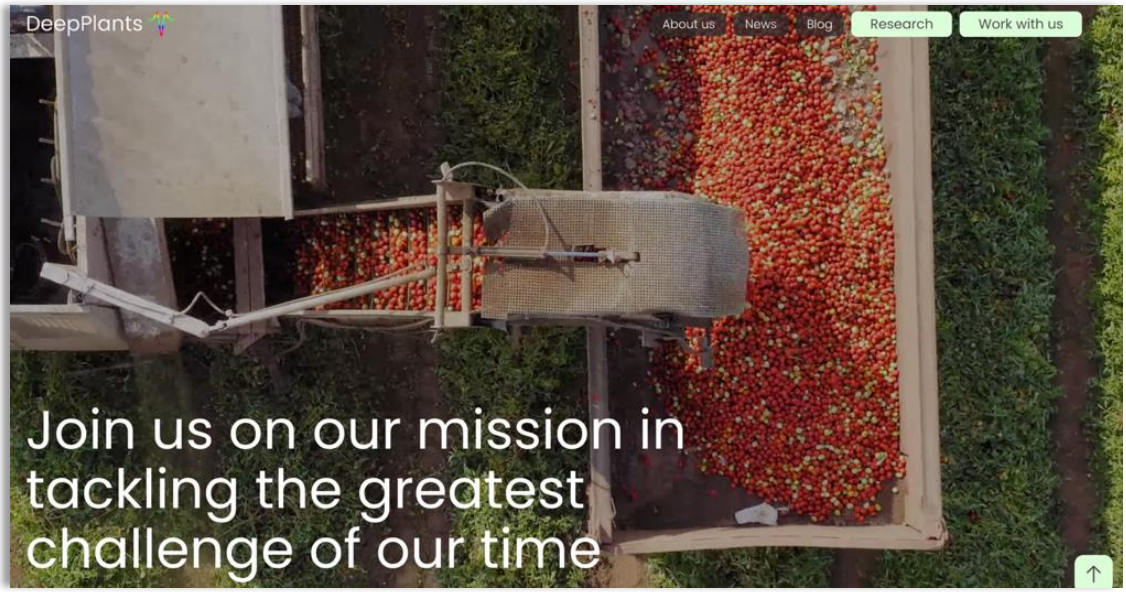


Figure 3 - AeroFarms Smart Farm (2021)

Plant-AI (2022)

- Open-source project
- Features: Real-time detection using EfficientNet
- Coverage: Multiple crop species
- Limitation: Fixed camera positions

2.2.3 Movement Systems in Agricultural Robotics

FarmBot (2021)

- Open-source farming robot
- Features: CNC-style movement system

- Application: Small-scale precision farming
- Limitation: Single-plane operation

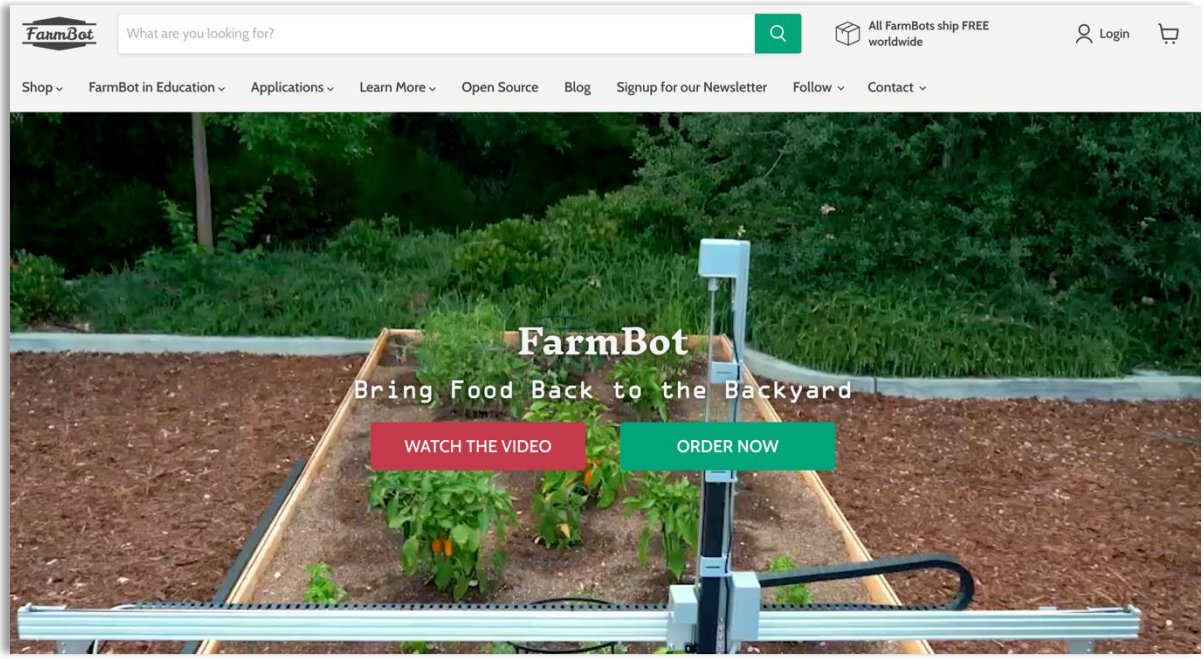


Figure 4 - FarmBot (2021)

AgriBot (2022)

- Research project
- Features: Multi-axis camera movement
- Technology: Stepper motor control
- Limitation: High implementation cost

Figure 5 - AgriBot (2022)

2.3 Comparative Analysis

2.3.1 Automated Vertical Farming Systems

Table 1- Automated Vertical Farming Systems

Feature	Our System	PlantVillage Project	AeroFarms Smart Farm
Year	2024	2020	2021
Scale	Small-Medium (3x3 grid)	Research Scale	Commercial (100,000+ sq ft)
Technology Focus	Disease Detection & Monitoring	Disease Recognition	Environment Control
AI Implementation	EfficientNetB0 CNN	Basic CNN	Machine Learning & IoT
Camera System	Mobile (2-axis)	Static	Multiple Fixed Cameras
Monitoring	Continuous (hourly)	On-demand	Real-time

Cost Range	$500-1000	Research Budget	$millions
Target Users	Small Farmers/Hobbyists	Researchers	Commercial Farms
Disease Detection	5 diseases	5 diseases	Limited
Scalability	Modular Design	Limited	Enterprise Level

2.3.2 Disease Detection Systems Comparison

Table 2 - Disease Detection Systems Comparison

Feature	Our System	DeepPlant	Plant-AI
Architecture	EfficientNetB0	ResNet-50	EfficientNet-B2
Accuracy	95.8%	93.4%	91.2%
Real-time Processing	Yes (0.5s/image)	No (batch processing)	Yes (1s/image)
Integration	Full system integration	Standalone software	Partial integration
Disease Types Detected	5 diseases	5 diseases	6 diseases
False Positive Rate	2.3%	4.1%	3.8%
Hardware Requirements	Moderate (4GB RAM)	High (8GB RAM)	Moderate (4GB RAM)

2.3.3 Movement Systems Comparison

Table 3 - Movement Systems Comparison

Feature	Our System	FarmBot	AgriBot
Movement Type	2-axis linear rails	3-axis CNC system	Multi-axis robotic arm

Precision	±0.5mm	±0.2mm	±1.0mm
Cost	$200-300	$1000+	$800+
Scalability	Highly modular	Limited by frame size	Complex scaling requirements
Power Usage	Low (24W)	High (75W)	Moderate (50W)
Installation	DIY-friendly	Professional setup	Technical expertise needed

2.4 Research Gap Analysis

Our research identified several gaps in existing solutions:

2.4.1 Integration Gap

- Most systems focus on either farming or detection
- Limited integration between movement and detection
- Lack of comprehensive monitoring solutions

2.4.2 Automation Gap

- Manual intervention often required
- Limited autonomous operation capabilities
- Insufficient real-time monitoring

2.4.3 Scalability Gap

- Many solutions are not scalable
- High implementation costs
- Complex maintenance requirements

2.5 Project Contributions and Advantages

2.5.1 Core Innovations

1. **Integrated System Architecture**
    - Unified platform combining vertical farming and disease detection
    - Automated camera movement system for continuous monitoring
    - Real-time alerts and monitoring dashboard
2. **Optimized Design Philosophy**
    - Modular construction using readily available components
    - Expandable grid system for flexible scaling
3. **Technical Advantages**
    - High precision monitoring at significantly lower costs
    - Efficient balance of processing speed and detection accuracy
    - Simplified maintenance through accessible components
    - Seamless integration between hardware and software systems

2.5.2 Market Positioning

Our solution uniquely positions itself by:

1. **Bridging Implementation Gaps**
    - Connects research-grade technology with practical applications
    - Makes advanced farming technology accessible to small-scale users
    - Provides enterprise-level features at consumer-friendly costs
2. **User-Centric Approach**
    - Designed for both novice and experienced users
    - Intuitive interface for system management
    - Comprehensive documentation and support
    - Adaptable to various growing environments

2.6 Agriculture and Risk

Agriculture is inherently exposed to various risks that can affect productivity, profitability, and sustainability. The smart agricultural system integrates advanced technologies to mitigate these risks, ensuring better control over environmental factors, resource management, and operational efficiency.

2.6.1 Types of Risks in Agriculture:

1. **Environmental Risks**
    - Unpredictable weather conditions (droughts, floods, storms)
    - Soil degradation and nutrient depletion
    - Pests and plant diseases
2. **Operational Risks**
    - Inefficient irrigation and water management
    - Equipment failures or system malfunctions
    - Poor planting and harvesting techniques
3. **Market and Economic Risks**
    - Fluctuations in crop prices
    - Supply chain disruptions
    - High production costs
4. **Regulatory and Compliance Risks**
    - Changing agricultural policies and regulations
    - Compliance with environmental standards
    - Restrictions on pesticide and fertilizer usage

2.6.2 Risk Mitigation through Smart Agriculture:

1. **Real-time Monitoring and Alerts**
    - The system uses **sensors** and **cameras** to track soil moisture, temperature, and humidity.

- **AI-powered analytics** predict potential risks and send early warnings to farmers.

2. **Automated Control Systems**
    - **Smart irrigation** adjusts water distribution based on real-time soil conditions.
    - **Automated climate control** optimizes greenhouse conditions to prevent crop damage.
3. **Predictive Analytics for Risk Assessment**
    - **Machine learning models** analyze historical data to predict disease outbreaks or weather patterns.
    - **Yield forecasting tools** help farmers plan ahead for market fluctuations.
4. **Integrated Pest and Disease Management**
    - AI-powered image recognition detects early signs of pest infestations.
    - **Smart pesticide application** ensures minimal chemical use and maximum protection.
5. **Data-driven Decision Making**
    - The system provides **customized recommendations** for planting, fertilization, and harvesting.
    - **Blockchain integration** ensures transparency in supply chain transactions.

By leveraging these advanced technologies, the smart agricultural system enhances **productivity, reduces losses, and ensures sustainability**, making farming more **resilient** against risks.

2.7 Technology background:

2.7.1 Business Analyst

- **Draw.io**:

Figure 6 - Draw.io

A web-based diagramming tool that helps create flowcharts, process maps, and system architectures. It is widely used for modeling business workflows and software architecture.

- **Microsoft Office**:

Figure 7 - Microsoft Office

A suite of productivity applications including Word, Excel, and PowerPoint, essential for documentation, data analysis, and presentations in business analysis.

2.7.2 Mobile UI Designer

Figure 8 - Figma

- **Figma**:

  A cloud-based UI/UX design tool that enables real-time collaboration, allowing designers to create, prototype, and refine interfaces with an intuitive workflow.

Figure 9 - Illustrator

- **Illustrator**:

  A vector graphics editor from Adobe used for designing high-quality UI assets, icons, and scalable illustrations for mobile applications.

2.7.3 Front-end Developer

Figure 10 - Visual Studio Code

- **Visual Studio Code**:

  A lightweight yet powerful code editor with built-in support for debugging, Git, and extensions, making it ideal for front-end development.

Figure 11 - HTML

- **HTML**:

  The foundational markup language for structuring web pages, defining elements like text, images, and hyperlinks.

Figure 12 - CSS

- **CSS**:

  A styling language used to design and enhance the appearance of HTML elements, enabling responsive layouts and animations.

Figure 13 - JavaScript

- **JavaScript**:

  A scripting language that enables dynamic interactions, handling events, animations, and API communications on web applications.

Figure 14 - React JS

- **React JS**:

  A JavaScript library for building component-based, highly interactive user interfaces, particularly single-page applications (SPAs).

Figure 15 - Bootstrap

- **Bootstrap**:

  A front-end framework that provides pre-built CSS components and JavaScript plugins to accelerate responsive web design.

Figure 16 - TypeScript

- **TypeScript**:

  A statically typed superset of JavaScript that improves code scalability, maintainability, and error detection in large applications.

Figure 17 - Angular

- **Angular**:

  A TypeScript-based framework developed by Google, used for building scalable, single-page applications with structured architecture.

2.7.4 Back-end Developer

Figure 18 - Visual Studio

- **Visual Studio**:

  A robust integrated development environment (IDE) by Microsoft, supporting multiple languages and tools for enterprise-level application development.

Figure 19 - ASP.NET Core

- **ASP.NET Core**:

  A high-performance, cross-platform framework for building web APIs and backend services using C#, enabling seamless integration with databases and cloud platforms.

2.7.5 Flutter Developer

Figure 20 - Dart

- **Dart**:

  An object-oriented programming language optimized for building high-performance, cross-platform mobile applications with Flutter.

Figure 21 - Flutter

- **Flutter**:

  An open-source UI framework developed by Google that enables developers to create natively compiled mobile, web, and desktop applications from a single codebase.

Figure 22 - Firebase

- **Firebase**:

  A cloud platform by Google that provides backend services such as authentication, real-time databases, cloud storage, and push notifications.

Figure 23 - Android Studio

- **Android Studio**:

  A dedicated IDE for Android development that integrates with Flutter, offering tools for UI design, debugging, and performance monitoring.

2.7.6 AI Developer

Figure 24 - PyCharm

- **PyCharm**:

  A professional Python IDE from JetBrains, offering powerful debugging, code analysis, and AI-specific tools for machine learning development.

Figure 25 - Jupyter Notebook

- **Jupyter Notebook**:

  An interactive computing environment that allows developers to write and execute Python code in a browser, making it ideal for AI experiments and data visualization.

Figure 26 - NumPy

- **NumPy**:

  A fundamental library for numerical computing in Python, providing multi-dimensional array operations and mathematical functions for AI applications.

Figure 27 - Matplotlib

- **Matplotlib**:

  A Python library for data visualization, allowing developers to create static, animated, and interactive charts and graphs.

Figure 28 - Pandas

- **Pandas**:

  A data analysis library that offers high-level data structures like DataFrames, making it easier to manipulate and process structured data.

Figure 29 - Sklearn

- **Scikit-learn (Sklearn)**:

  A machine learning library that provides algorithms for classification, regression, clustering, and dimensionality reduction.

Figure 30 - TensorFlow

- **TensorFlow**:

  A deep learning framework by Google that enables the creation and training of neural networks for tasks such as image recognition, natural language processing, and AI automation.

Chapter 3
System Analysis

3.1 Introduction

This chapter provides a detailed system analysis and design process included in our system, which includes user requirements, UML diagrams and system architecture.

To develop a system, there are **two methods** available:

- Waterfall Development
- Agile Development

In our project, we have applied **agile development method**.

3.2 Definition of Agile

Agile methodology is a set of principles and practices for software development that prioritizes flexibility, adaptability, and customer collaboration. It emerged as a response to the limitations of traditional, rigid development methodologies, such as the Waterfall model.

3.2.1 Core Principles of Agile

1. **Individuals and Interactions over Processes and Tools:**

Agile values people and their interactions more than following strict processes or relying solely on tools. Effective communication and collaboration among team members and stakeholders are emphasized.

2. **Working Software over Comprehensive Documentation:**

While documentation is important, Agile prioritizes delivering working software to customers frequently. This allows for rapid feedback and ensures that the software meets the evolving needs of users.

3. **Customer Collaboration over Contract Negotiation:**

Agile encourages close collaboration with customers throughout the development process. By involving customers in decision-making and incorporating their feedback iteratively, Agile ensures that the final product meets their expectations.

4. **Responding to Change over Following a Plan:**

Agile acknowledges that requirements and priorities can change frequently. Instead of rigidly following a fixed plan, Agile teams embrace change and adapt their plans and processes to deliver value effectively.

3.2.2 Why Agile

1. **Flexibility and Adaptability:**

Agile's iterative approach allows quick adaptation to evolving market trends, regulations, and user needs, ensuring the product remains relevant.

2. **Customer-Centric Approach:**

By prioritizing stakeholder collaboration and feedback, Agile ensures the product aligns with user expectations and delivers real value.

3. **Incremental Delivery of Value:**

Agile enables early and frequent feature releases, allowing users to benefit sooner while refining the product based on continuous feedback.

4. **Continuous Improvement:**

Regular retrospectives and iterations help identify and implement improvements, leading to a more efficient and effective product.

5. **Risk Management:**

Agile mitigates risks through early testing, stakeholder engagement, and incremental delivery, ensuring smoother project execution.

3.3 User Requirements

3.3.1 System Administrator

1. Manage user accounts and access levels
2. Monitor system performance and health
3. Configure system parameters
4. View and analyze system logs
5. Manage disease detection models
6. Update system software
7. Generate system reports

3.3.2 Farmer/Plant Manager

1. Login into account
2. Configure plant monitoring schedules
3. View real-time plant health status
4. Receive disease detection alerts
5. Access historical data and trends
6. Generate and export crop health reports (Out of Scope)
7. Communicate with support team
8. View system documentation (FAQ)

Table 4 - System Administrator

System Administrator	
Feature	**Description**
Manage user accounts and access levels	• **Business Perspective**: Ensures secure and controlled access by defining roles and permissions, preventing unauthorized access. • **Design Perspective**: Admin dashboard with role-based access control (RBAC) for adding, modifying, and deleting users.

	• **Logic Perspective**: Implements authentication (OAuth, JWT) and role-based authorization for restricted access. ***User Story*** **Title**: Manage User Access **As a** System Administrator, **I want to** create, update, and deactivate user accounts with role-based permissions **so that** only authorized users can access specific functionalities. ***Acceptance Criteria*** • Admin can create new user accounts and assign roles. • Admin can modify or deactivate existing users. • Unauthorized users cannot access restricted areas. • System logs all access changes.
Monitor system performance and health	• **Business Perspective**: Provides real-time monitoring of uptime, API response times, and system errors to ensure efficiency. • **Design Perspective**: Performance dashboard with graphs, alerts, and logs displaying system metrics. • **Logic Perspective**: Uses monitoring tools (Prometheus, New Relic) to track key performance indicators (KPIs) and trigger alerts.

	User Story **Title**: Monitor System Health **As a** System Administrator, **I want to** track system performance and receive alerts **so that** I can proactively address issues before they impact users. ***Acceptance Criteria*** • Admin dashboard displays real-time system performance data. • Alerts trigger when predefined thresholds (e.g., CPU usage > 80%) are met. • Admin can filter logs by date, severity, and type.
Configure system parameters	• **Business Perspective**: Allows admins to set and adjust configurations such as logging levels, notification preferences, and AI model sensitivity. • **Design Perspective**: Settings page for modifying system parameters dynamically without affecting core functionalities. • **Logic Perspective**: Uses database-stored configurations that update in real time without requiring code changes. ***User Story*** **Title**: Configure System Settings **As a** System Administrator, **I want to** modify system parameters **so that** the system behavior aligns with operational requirements.

	Acceptance Criteria • Admin can adjust notification settings and logging levels. • Changes apply immediately without requiring system restarts. • Unauthorized users cannot modify system settings.
View and analyze system logs	• **Business Perspective**: Improves security and troubleshooting by tracking system events, errors, and audit logs. • **Design Perspective**: Log viewer with search, filters (user ID, event type), and export options for further analysis. • **Logic Perspective**: Logs stored in structured formats (JSON, ELK Stack) and analyzed using tools like Splunk. ***User Story*** **Title**: View System Logs **As a** System Administrator, **I want to** review system logs **so that** I can track user activities and troubleshoot errors efficiently. ***Acceptance Criteria*** • Admin can view logs with filters for date, event type, and severity. • Logs provide timestamps and user activity details. • Logs can be exported for further analysis.

Manage disease detection models	• **Business Perspective**: Ensures AI models used for disease detection are updated and fine-tuned for accuracy. • **Design Perspective**: Model management panel to upload datasets, retrain models, and validate results. • **Logic Perspective**: Uses TensorFlow/PyTorch for model training and validation against test datasets. ***User Story*** **Title**: Manage AI Models **As a** System Administrator, **I want to** update and validate disease detection models **so that** the system provides accurate and reliable diagnoses. ***Acceptance Criteria*** • Admin can upload new datasets for model training. • System provides validation accuracy and confidence scores. • Previous models can be archived or restored.
Update system software	• **Business Perspective**: Ensures the system is up-to-date with security patches, bug fixes, and new features. • **Design Perspective**: Update module showing version history, available updates, and rollback options. • **Logic Perspective**: Uses CI/CD pipelines for automated deployments with rollback mechanisms.

	User Story **Title**: Update System Software **As a** System Administrator, **I want to** apply software updates **so that** the system remains secure and efficient. ***Acceptance Criteria*** • Admin can view available updates with version details. • Updates can be applied without affecting live users. • Rollback option is available in case of update failure.
Generate system reports	• **Business Perspective**: Provides insights through reports on user activity, performance, and AI accuracy for data-driven decisions. • **Design Perspective**: Reporting dashboard with customizable filters, export options (CSV, PDF), and scheduled reports. • **Logic Perspective**: Uses SQL queries and Power BI for report generation and automated delivery. ***User Story*** **Title**: Generate Reports **As a** System Administrator, **I want to** generate system reports **so that** I can analyze performance and make informed decisions.

	Acceptance Criteria • Reports include user activity, performance, and error logs. • Reports can be exported in CSV/PDF format. • Admin can schedule automatic report generation.

Table 5 - Farmer/Plant Manager

Farmer/Plant Manager	
Feature	**Description**
Login into account	• **Business Perspective**: Ensures secure access to personalized data, preventing unauthorized use. • **Design Perspective**: Login page with email/password authentication and two-factor authentication (2FA) option. • **Logic Perspective**: Uses authentication mechanisms like OAuth 2.0 or JWT to verify user credentials securely. ***User Story*** **Title**: User Login **As a** User, **I want to** log into my account **so that** I can access my plant monitoring dashboard. ***Acceptance Criteria*** • Users can log in using email and password. • Invalid credentials return appropriate error messages.

	• Optional 2FA authentication is available. ***Edge Cases & Solutions*** 1. **Incorrect Password** → Display a clear error message and allow password reset. 2. **Forgot Password** → Provide a "Forgot Password" option to reset via email/SMS. 3. **Multiple Failed Attempts** → Temporarily lock the account for security. 4. **Account Not Verified** → Send an email verification link upon first login attempt. 5. **System Downtime** → Show a maintenance message and alternative support contact.
Configure plant monitoring schedules	• **Business Perspective**: Allows users to set monitoring intervals for different crops to optimize plant care. • **Design Perspective**: Intuitive scheduling interface with options to customize frequency, time, and parameters. • **Logic Perspective**: Uses a backend scheduler to trigger monitoring tasks based on user-defined configurations. ***User Story*** **Title**: Configure Monitoring Schedules **As a** User, **I want to** set plant monitoring schedules **so that** I can ensure timely tracking of plant health.

	Acceptance Criteria • Users can set custom monitoring frequencies. • System triggers monitoring based on the set schedule. • Users receive confirmation notifications for changes. ***Edge Cases & Solutions*** 1. **Conflicting Schedules** → Notify the user and allow schedule adjustments. 2. **Invalid Time Selection** → Restrict scheduling within valid operating hours. 3. **Server Failure During Scheduling** → Retry or queue the request for later processing. 4. **Timezone Issues** → Convert times to the user's local timezone.
View real-time plant health status	• **Business Perspective**: Enables users to monitor real-time sensor data, improving proactive plant care. • **Design Perspective**: Live dashboard with visuals, charts, and alerts indicating plant health metrics. • **Logic Perspective**: Uses IoT sensors to collect data and display updates in real time via WebSockets.

	User Story **Title**: Real-Time Plant Health Monitoring **As a** User, **I want to** view live plant health data **so that** I can detect any issues immediately. ***Acceptance Criteria*** • Users can view real-time sensor data. • Data updates dynamically without page refresh. • Alerts trigger if health parameters exceed thresholds. ***Edge Cases & Solutions*** 1. **Sensor Malfunction** → Show an error message and recommend recalibration. 2. **Internet Connection Loss** → Display a warning and attempt data refresh upon reconnection. 3. **Data Overload Causing Lag** → Implement data caching and pagination.
Receive disease detection alerts	• **Business Perspective**: Notifies users of potential plant diseases, allowing immediate intervention. • **Design Perspective**: Alert notifications (in-app, email, SMS) with details on detected diseases and recommendations. • **Logic Perspective**: AI-based disease detection model analyzes plant data and triggers alerts when anomalies are detected.

	User Story **Title**: Disease Detection Alerts **As a** User, **I want to** receive alerts when plant diseases are detected **so that** I can take corrective actions immediately. ***Acceptance Criteria*** • Users receive alerts in-app, via email, and optionally via SMS. • Alerts include detected disease, severity, and recommended actions. • Users can acknowledge or dismiss alerts. ***Edge Cases & Solutions*** 1. **False Positives** → Allow users to report incorrect detections to improve the AI model. 2. **Delayed Alerts** → Implement a retry mechanism for alert failures. 3. **Spam Alert Flooding** → Group alerts and send summaries instead of multiple notifications.
Access historical data and trends	• **Business Perspective**: Helps users analyze long-term plant health patterns for better decision-making. • **Design Perspective**: Interactive graphs and trend analysis tools showing historical plant health data.

	• **Logic Perspective**: Stores past sensor readings in a database and generates visual reports based on user queries. ***User Story*** **Title**: View Historical Data **As a** User, **I want to** access past plant health records **so that** I can identify trends and anomalies. ***Acceptance Criteria*** • Users can filter historical data by date range and plant type. • Data visualization is available in charts and graphs. • Users can download historical reports. ***Edge Cases & Solutions*** 1. **Large Data Sets Causing Slow Loading** → Implement pagination and filtering. 2. **Corrupted or Missing Data** → Offer a data recovery or fallback option.
Generate and export crop health reports	• **Business Perspective**: Provides users with detailed insights for farm management and regulatory compliance. • **Design Perspective**: Reporting module allowing users to customize, generate, and export reports in PDF/CSV formats.

	• **Logic Perspective**: Uses structured queries and automated report generation tools like Power BI or Tableau. ***User Story*** **Title**: Generate Crop Health Reports **As a** User, **I want to** generate and export crop health reports **so that** I can analyze farm performance and share insights. ***Acceptance Criteria*** • Users can generate reports for specific date ranges and plants. • Reports include plant health trends, detected diseases, and recommendations. • Reports can be exported in PDF and CSV formats. ***Edge Cases & Solutions*** 1. **Report Generation Timeout** → Optimize queries and use background processing. 2. **Incorrect Report Data** → Allow users to regenerate reports with updated parameters.
Communicate with support team	• **Business Perspective**: Enables users to resolve technical issues and receive assistance with the system. • **Design Perspective**: Support center with chat, email, and ticket submission options.

	• **Logic Perspective**: Integrates with customer support systems like Zendesk or Freshdesk for efficient issue resolution. ***User Story*** **Title**: Contact Support **As a** User, **I want to** communicate with the support team **so that** I can get help with any issues I face. ***Acceptance Criteria*** • Users can submit support tickets or chat with an agent. • Support team responds within a defined SLA. • Users receive email updates on ticket status. ***Edge Cases & Solutions*** 1. **No Response from Support** → Implement automated chatbot assistance. 2. **Duplicate Support Requests** → Notify users of existing open tickets.
View system documentation (FAQ)	• **Business Perspective**: Provides self-service support, reducing dependency on customer service. • **Design Perspective**: Knowledge base with categorized articles, search functionality, and FAQs.

	• **Logic Perspective**: Uses a content management system (CMS) to dynamically update FAQs. ***User Story*** **Title**: Access System Documentation **As a** User, **I want to** view FAQs and guides **so that** I can troubleshoot common issues myself. ***Acceptance Criteria*** • Users can search and browse FAQs. • Documentation is categorized by topic. • FAQs are regularly updated based on common user queries. ***Edge Cases & Solutions*** 1. **Outdated Information** → Allow users to report inaccuracies. 2. **Search Function Not Returning Results** → Implement fuzzy search and keyword suggestions.

3.4 UML Diagrams

Unified Modeling language (UML) is a standardized modeling language enabling developers to specify, visualize, construct and document artifacts of a software system. Thus, UML makes these artifacts scalable, secure and robust in execution. UML is an important aspect involved in object-oriented software development. It uses graphic notation to create visual models of software systems.

3.4.1 Use Case Diagrams

Describes functionality of a system in terms of actors, goals as use cases and dependencies among the use cases. It is a methodology used in system analysis to identify, clarify, and organize system requirements. A use case diagram contains four main components:

- **Actors:**

Are usually individuals involved with the system defined according to their roles. The actor can be a human or other external system.

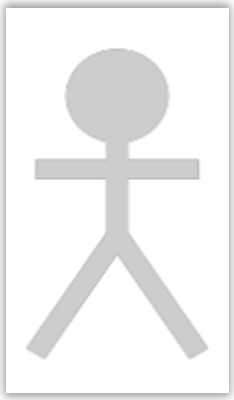


Figure 31 - Actors

- **Use Case:**

It is a visual representation of a distinct business Functionality in a system.

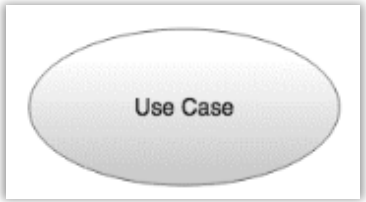


Figure 32 - Use Case

- **System Boundary:**

Show the scope of the system.

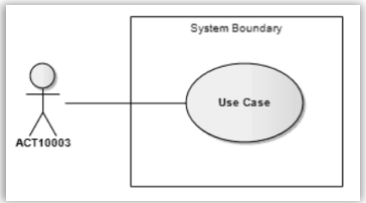


Figure 33 - System Boundary

- **Relationships:**
  - **Generalization:**

Is a relationship from a child use case to a parent use case, specifying how a child can specialize all behavior and characteristics described for the parent.

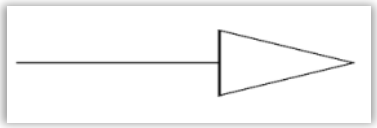


Figure 34 - Generalization

  - **Association:**

Is the relationship between an actor and a business use case. It indicates that an actor can use a certain functionality of the business system.

Figure 35 - Association

Use Case Diagram of our System:

- **Farmer / Plant Manager:**

  The Farmer/Plant Manager is the primary user responsible for managing crop health and productivity. They interact with the system to optimize agricultural practices through the following use cases:

  - **Login into Account:**

  Authenticates and accesses personalized features.

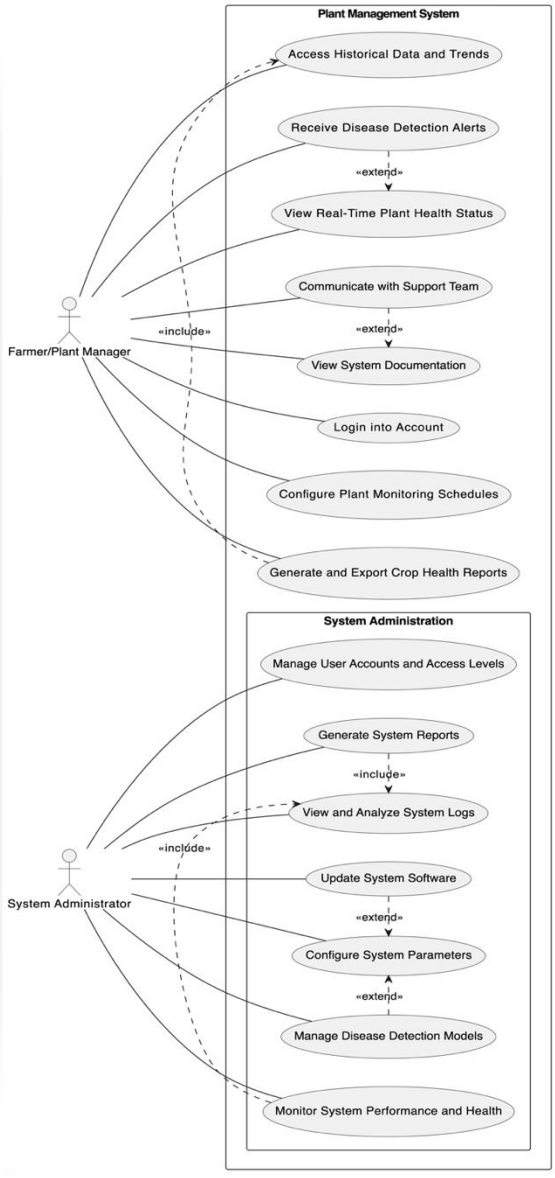


Figure 36 - Use Case Diagram of our System

- **Configure Plant Monitoring Schedules:**

Sets schedules for plant health monitoring, including irrigation and disease checks.

- **View Real-Time Plant Health Status:**

Monitors live data on crop health, environmental conditions, and potential threats.

- **Receive Disease Detection Alerts:**

Gets instant notifications about detected diseases or abnormalities. This is an extension of viewing real-time health status.

- **Access Historical Data and Trends:**

Analyzes historical data for better decision-making, which is included in generating crop health reports.

- **Generate and Export Crop Health Reports:**

Compiles detailed reports on crop conditions, enabling informed planning and documentation.

- **Communicate with Support Team:**

Interacts with support for assistance and troubleshooting. This extends viewing system documentation.

- **View System Documentation:**

Accesses manuals and guidelines to maximize the system's potential.

- **System Administrator:**

The System Administrator is responsible for maintaining the system's integrity, security, and performance. They perform backend management tasks, ensuring smooth and secure operations through the following use cases:

- **Manage User Accounts and Access Levels:**

Creates, updates, and deletes user accounts, and controls access permissions.

- **Monitor System Performance and Health:**

Continuously tracks system stability and efficiency, including server health and data flow. It includes viewing and analyzing system logs.

- **Configure System Parameters:**

Adjusts settings like alert thresholds and monitoring frequencies to enhance system functionality.

- **View and Analyze System Logs:**

Investigates system activities for security, performance, and debugging purposes.

- **Manage Disease Detection Models:**

Updates and refines AI models for disease prediction, extending system configuration.

- **Update System Software:**

Applies software updates to enhance security and introduce new features, which extends system parameter configuration.

- **Generate System Reports:**

Produces reports on system usage, performance, and security incidents. This includes viewing and analyzing system logs.

3.4.2 Sequence Diagrams

Sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are sometimes called event diagrams or event scenarios.

Component of sequence diagram:

- **Actors:**

An actor in a UML diagram represents a type of role where it interacts with the

system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.

- **Lifeline:**

Represents the passage of time as it extends downward. This dashed vertical line shows the sequential events that occur to an object during the charted process. Lifelines may begin with a labeled rectangle shape or an actor symbol.

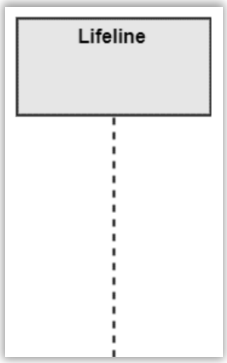


Figure 37 - Lifeline

- **Activations:**

A thin rectangle on a lifeline represents the period during which an element is performing an operation. The top and the bottom of the rectangle are aligned with the initiation and the completion time respectively.

Figure 38 - Activations

- **Call Message:**

A message defines a particular communication between Lifelines of an Interaction. Call message is a kind of message that represents an invocation of operation of target lifeline.

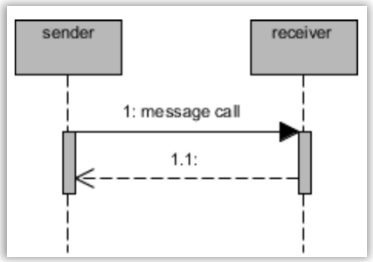


Figure 39 - Call Message

- **Return Message:**

A message defines a particular communication between Lifelines of an Interaction. Return message is a kind of message that represents the pass of information back to the caller of a corresponded former message.

- **Synchronous Message:**

A synchronous message requires a response before the interaction can continue. It's usually drawn using a line with a solid arrowhead pointing from one object to

Figure 40 - Synchronous Message

another.

- **Asynchronous messages:**

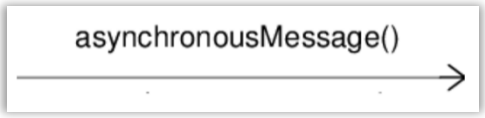


Figure 41 - Asynchronous messages

Asynchronous messages don't need a reply for interaction to continue.

Like synchronous messages, they are drawn with an arrow connecting two lifelines; however, the arrowhead is usually open and there's no return message depicted.

3.4.2 Sequence Diagrams of our System:

Farmer/Plant Manager

- **Access Historical Data and Trends:**

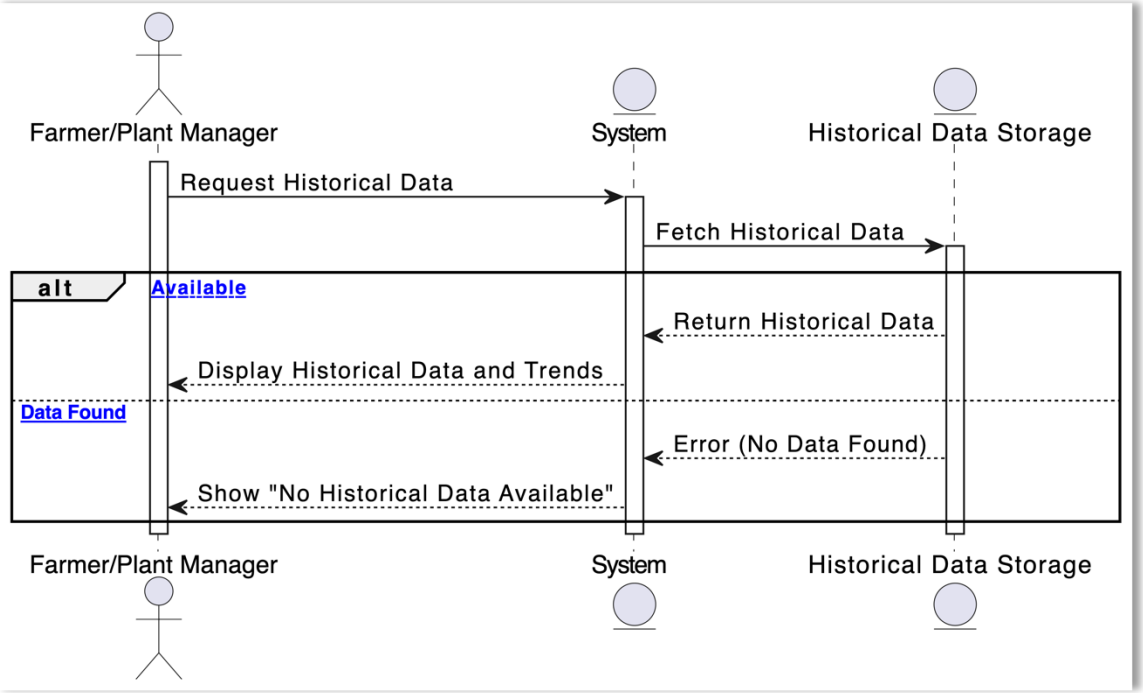


Figure 42 - Access Historical Data and Trends

- **System** receives the request and processes it.
- **System** sends a request to **Historical Data Storage** for relevant data.
- **Historical Data Storage** retrieves historical records if available.
- **Historical Data Storage** returns the data to the **System**.

- **System** processes and formats the retrieved data.
- **System** displays historical data and trends to **Farmer/Plant Manager**.
- If no data is found, **Historical Data Storage** returns an error message.
- **System** handles the error and informs **Farmer/Plant Manager**.
- **Farmer/Plant Manager** sees "No Historical Data Available" if no data exists.
- The interaction ensures smooth decision-making for **Farmer/Plant Manager**.
- The process completes with either data retrieval or an error notification.

- **Receive Disease Detection Alerts:**

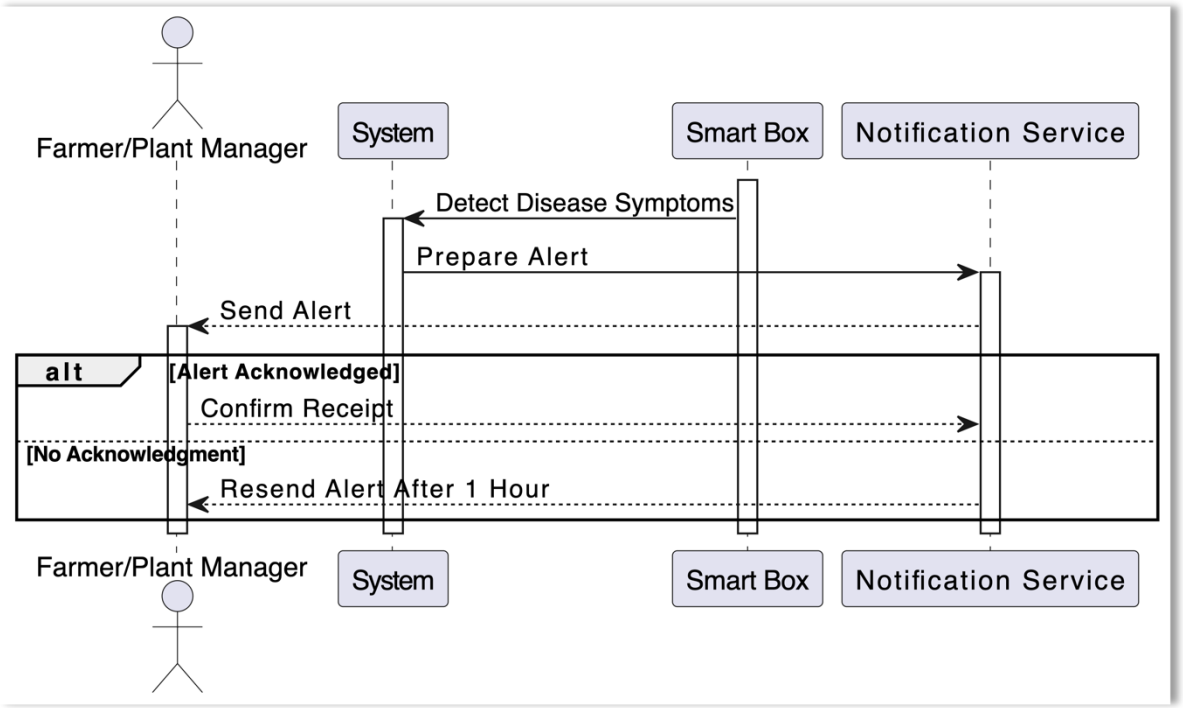


Figure 43 - Receive Disease Detection Alerts

- **Smart Box** detects disease symptoms in plants.
- **Smart Box** sends the detected symptoms data to the **System**.
- **System** processes the data and prepares an alert.
- **System** sends the alert to the **Notification Service**.
- **Notification Service** delivers the alert to the **Farmer/Plant Manager**.

- If the **Farmer/Plant Manager** acknowledges the alert, the process ends.
- If no acknowledgment is received, the **Notification Service** waits for 1 hour.
- After 1 hour, the **Notification Service** resends the alert.
- **Farmer/Plant Manager** finally acknowledges the alert or takes action.
- The **System** ensures alerts are effectively delivered and responded to.
- **Smart Box** continuously monitors plant health for further symptoms.
- The cycle repeats as needed to protect crops from disease threats.

- **View Real-Time Plant Health Status:**

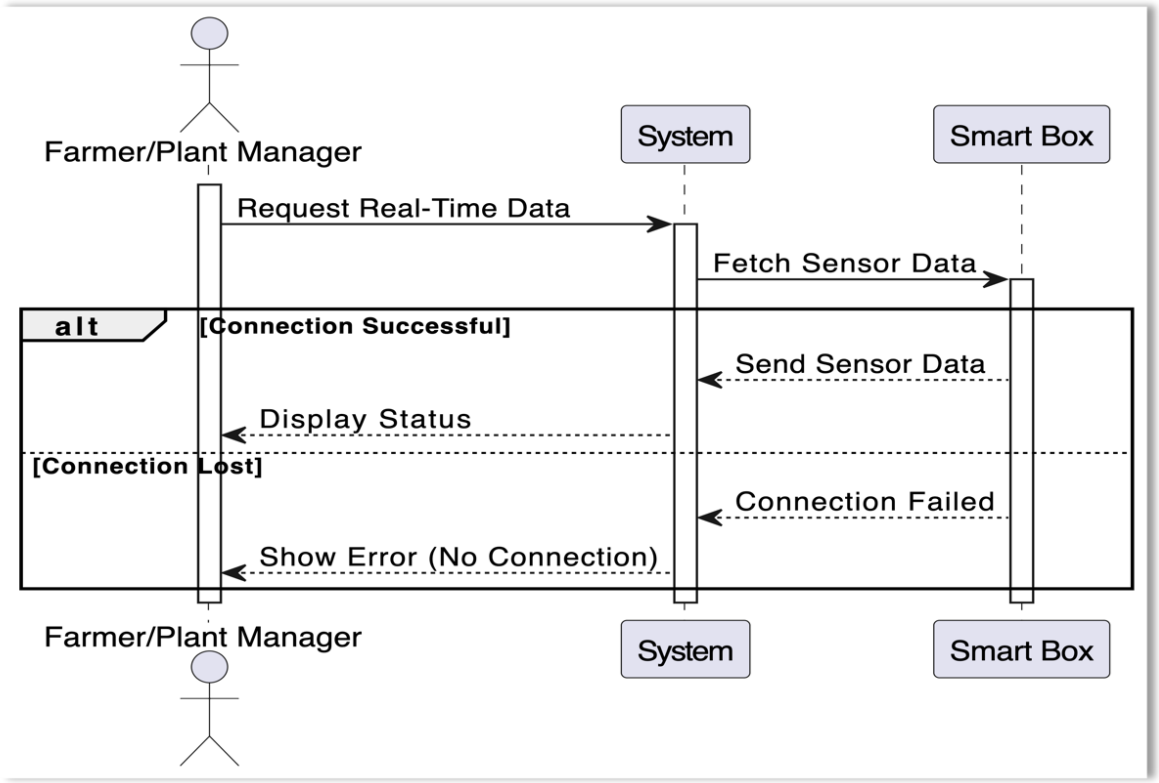


Figure 44 - View Real-Time Plant Health Status

- **Farmer/Plant Manager** requests real-time data from the **System**.
- **System** receives the request and processes it.
- **System** sends a request to the **Smart Box** for sensor data.
- **Smart Box** attempts to fetch and send real-time sensor data.
- If the **Smart Box** successfully transmits data, it sends it back to the **System**.

- **System** processes the data and displays the status to the **Farmer/Plant Manager**.
- If the **Smart Box** encounters a connection failure, it sends an error response.
- **System** detects the failure and notifies the Farmer/**Plant Manager**.
- **Farmer/Plant Manager** sees an error message indicating no connection.
- The system ensures data reliability and real-time monitoring.
- The **Smart Box** continuously collects and attempts to transmit sensor data.
- The process repeats for continuous plant health monitoring.

- **Communicate with Support Team:**

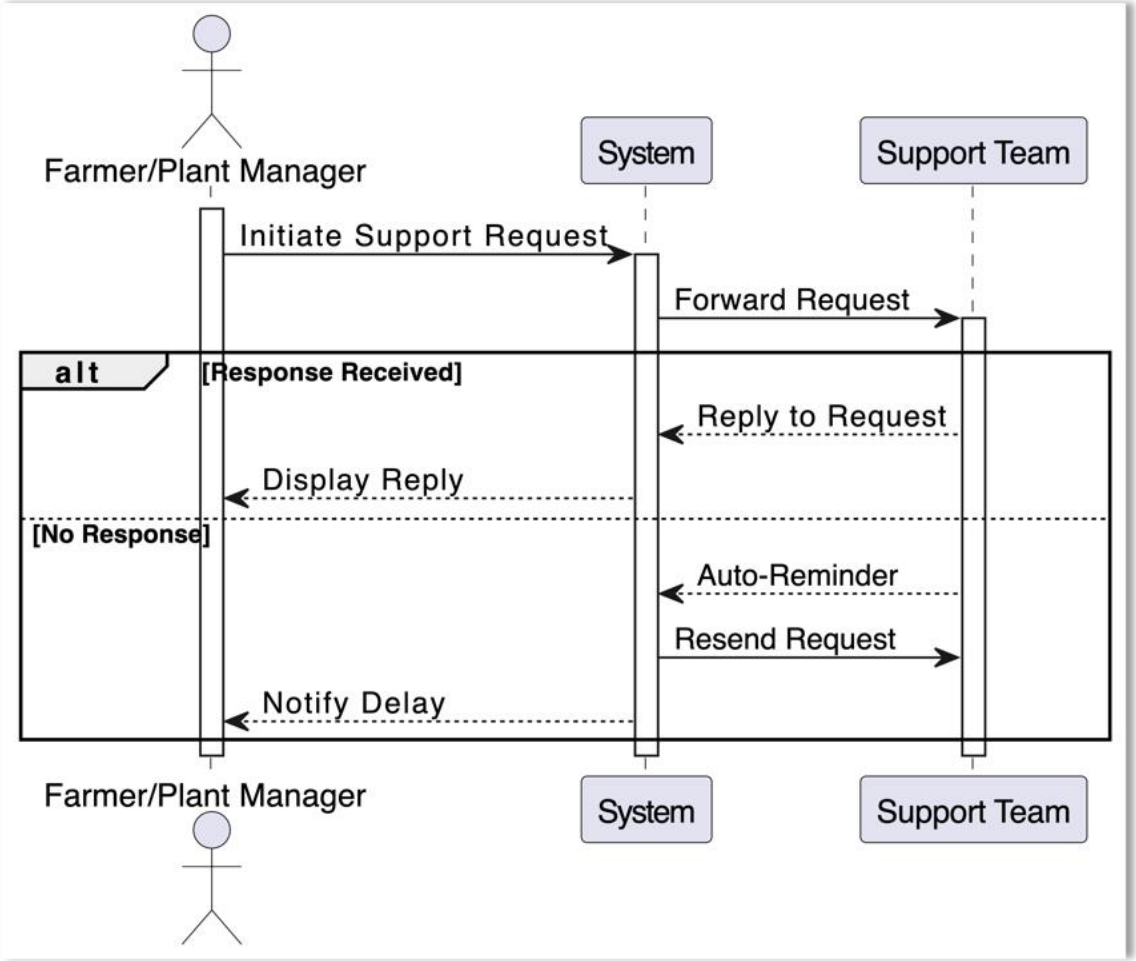


Figure 45 - Communicate with Support Team

- Farmer/Plant Manager **initiates a support request through the** System**.**

- System **forwards the request to the** Support Team**.**
- Support Team **receives and processes the request.**
- **If a response is available,** Support Team **sends a reply to the** System**.**
- System **receives and displays the response to the** Farmer/Plant Manager**.**
- **If no response is received, the** System **triggers an auto-reminder.**
- Support Team **is prompted to respond again.**
- **If the** Support Team **still doesn't reply, the** System **resends the request.**
- System **notifies the** Farmer/Plant Manager **about the delay.**
- **The** Farmer/Plant Manager **remains informed about the request status.**
- **The process ensures timely support and efficient communication.**
- **The cycle continues until a resolution is provided.**

- **View System Documentation (FAQ):**

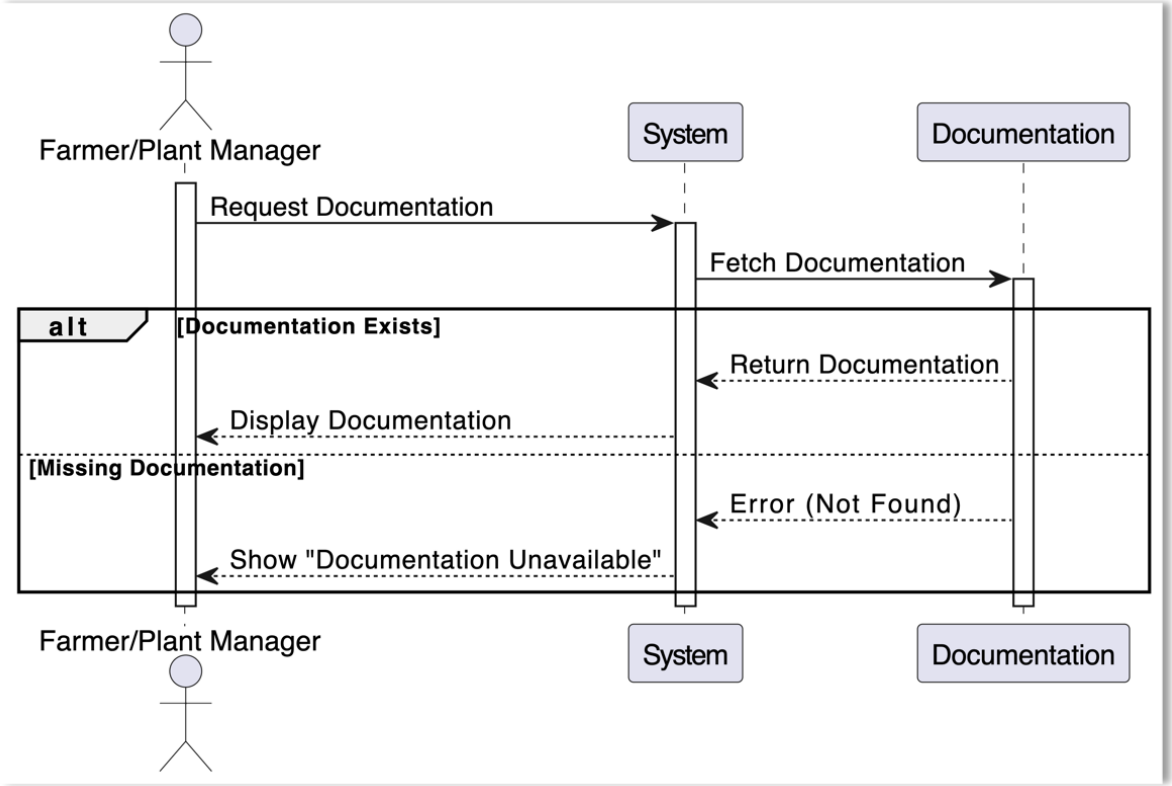


Figure 46 - View System Documentation (FAQ)

- Farmer/Plant Manager **requests documentation from the** System**.**
- System **receives the request and processes it.**

- System **sends a request to the** Documentation Storage **to fetch the document.**
- Documentation Storage **checks for the requested document.**
- **If the document exists,** Documentation Storage **returns it to the** System**.**
- System **displays the documentation to the** Farmer/Plant Manager**.**
- **If the document is missing,** Documentation Storage **returns an error response.**
- System **notifies the** Farmer/Plant Manager **that documentation is unavailable.**
- Farmer/Plant Manager **receives either the documentation or an error message.**
- **The** System **ensures smooth retrieval and error handling.**
- **The** Documentation Storage **continuously updates and maintains records.**
- **The process repeats when new requests for documentation are made.**

- **Login:**

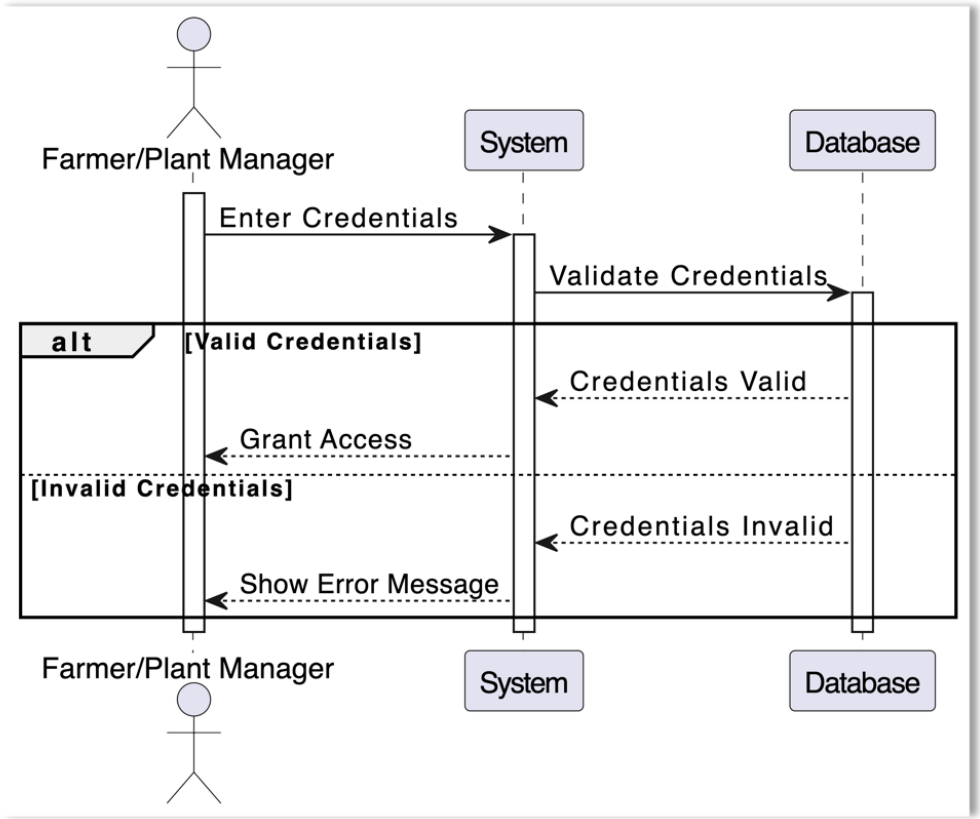


Figure 47 - Login

- Farmer/Plant Manager **enters credentials into the** System**.**
- System **receives and processes the credentials.**
- System **sends a request to the** Database **to validate credentials.**
- Database **checks credentials against stored records.**
- **If credentials are valid,** Database **sends confirmation to the** System**.**
- System **grants access to the** Farmer/Plant Manager**.**
- **If credentials are invalid,** Database **sends an error response to the** System**.**
- System **displays an error message to the** Farmer/Plant Manager**.**
- Farmer/Plant Manager **either gains access or is prompted to retry.**
- **The** System **ensures secure authentication processing.**
- **The** Database **maintains credential records for validation.**
- **The process repeats for each login attempt.**

- **Configure Plant Monitoring Schedules:**

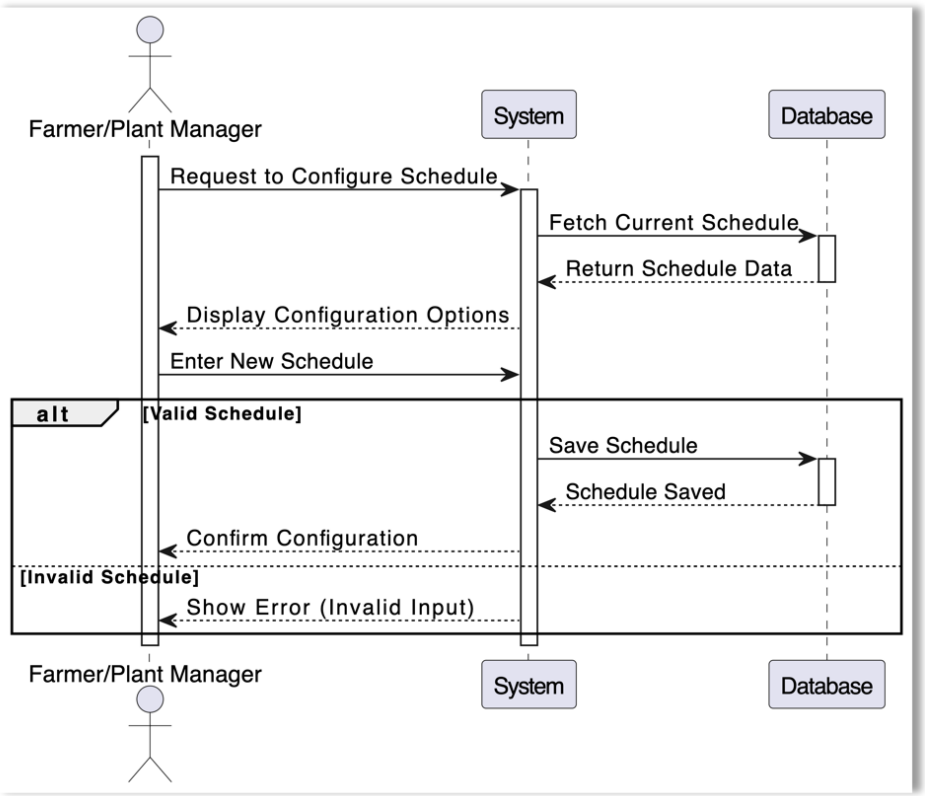


Figure 48 - Configure Plant Monitoring Schedules

- **Farmer/Plant Manager** requests to configure the schedule via the **System**.
- **System** processes the request and retrieves the current schedule.
- **System** sends a request to the **Database** to fetch schedule data.
- **Database** returns the current schedule to the **System**.
- **System** displays the current schedule and configuration options.
- **Farmer/Plant Manager** enters a new schedule.
- **System** validates the new schedule input.
- If valid, **System** saves the schedule in the **Database**.
- **Database** confirms that the schedule is successfully saved.
- **System** notifies the **Farmer/Plant Manager** of successful configuration.
- If the input is invalid, **System** displays an error message.
- The process repeats for further schedule modifications.

- **Generate and Export Crop Health Reports:**

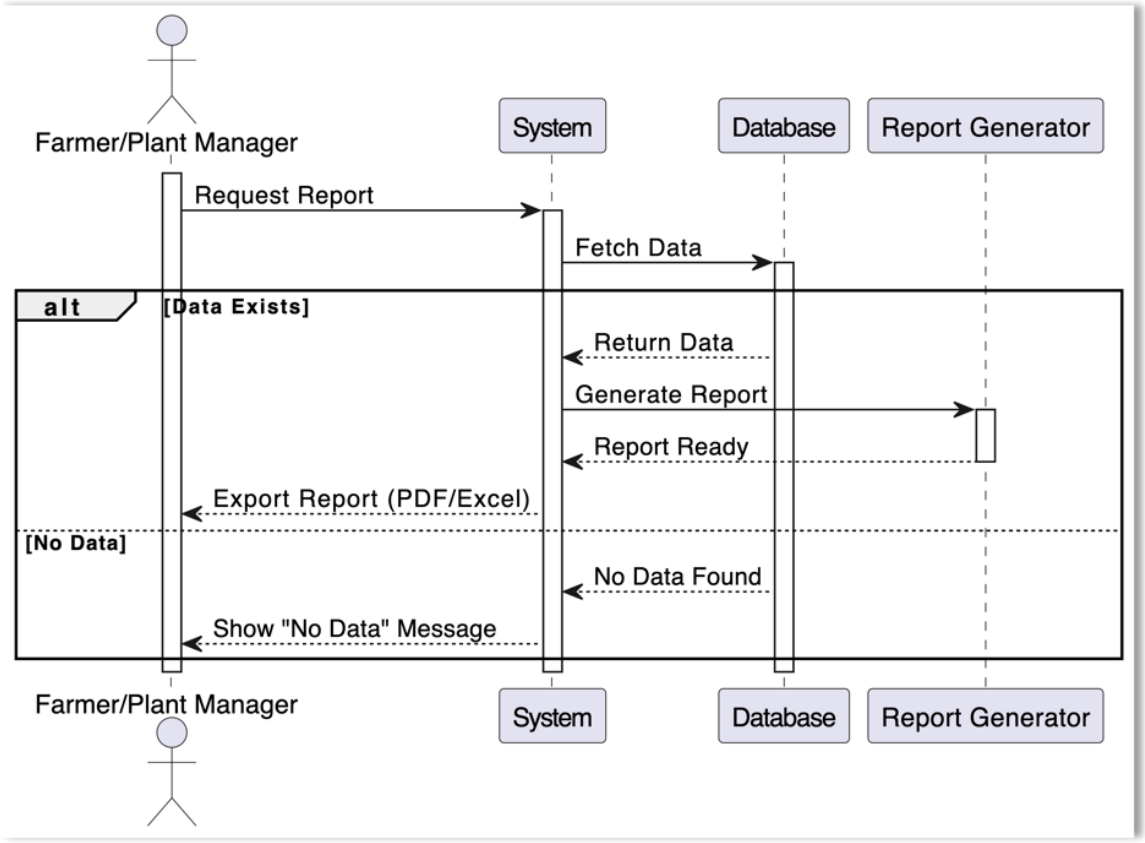


Figure 49 - Generate and Export Crop Health Reports

- **Farmer/Plant Manager** requests a report from the **System**.
- **System** processes the request and fetches data from the **Database**.
- **Database** retrieves the required data and returns it to the **System**.
- If data exists, **System** sends it to the **Report Generator**.
- **Report Generator** processes the data and generates the report.
- **Report Generator** sends the completed report back to the **System**.
- **System** exports the report in the requested format (PDF/Excel).
- **System** provides the report to the **Farmer/Plant Manager** for download.
- If no data is found, **Database** informs the **System**.
- **System** notifies the **Farmer/Plant Manager** that no data is available.
- The **System** ensures efficient report generation and error handling.
- The process repeats for new report requests.

System Administrator

- **Manage User Accounts and Access Levels:**

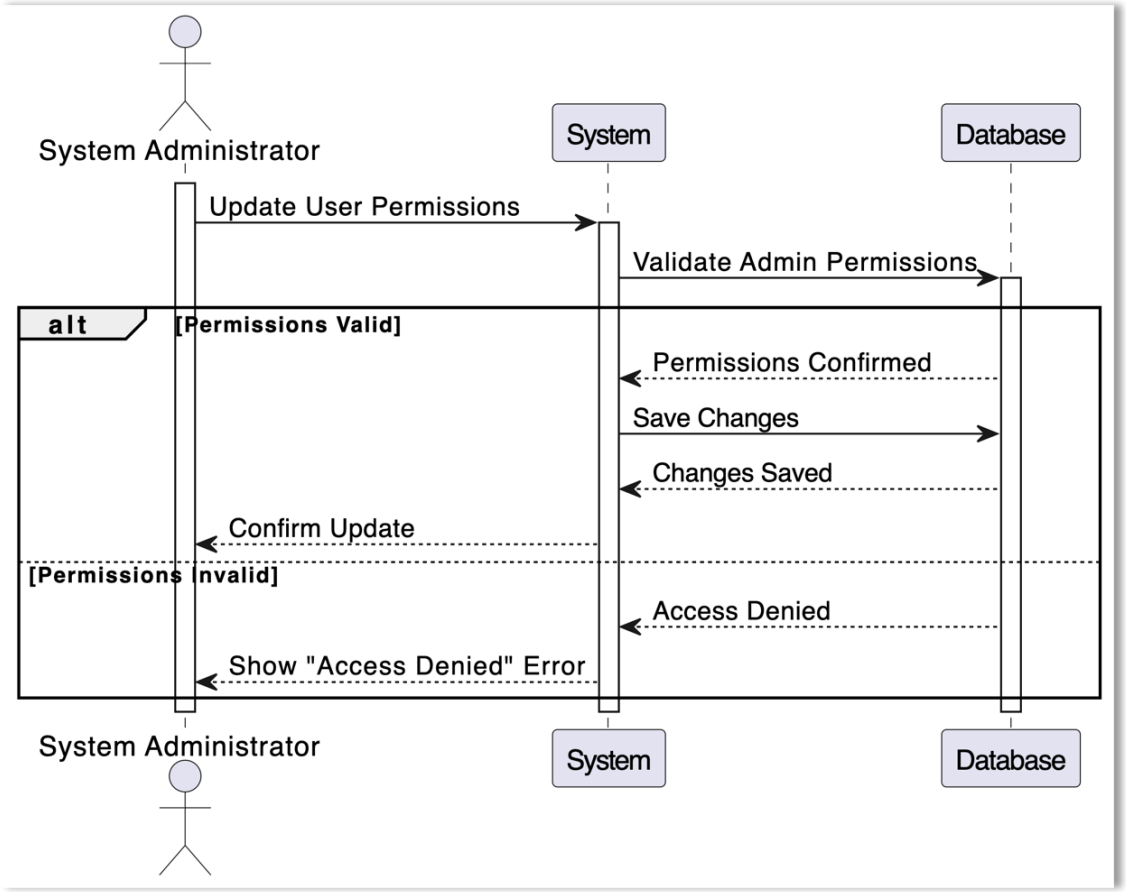


Figure 50 - Manage User Accounts and Access Levels

- **System Administrator** requests to update user permissions via the **System**.
- **System** processes the request and validates admin permissions.
- **System** sends a request to the **Database** to verify authorization.
- If permissions are valid, **Database** confirms authorization.
- **System** proceeds with saving the updated permissions in the **Database**.
- **Database** successfully stores the changes and confirms the update.
- **System** notifies the **System Administrator** of the successful update.
- If permissions are invalid, **Database** denies access and informs the **System**.
- **System** displays an "Access Denied" error to the **System Administrator**.
- The **System** ensures only authorized changes are processed.
- The **Database** maintains integrity and security of user permissions.

- The process repeats for future permission update requests.

- **Generate System Reports:**

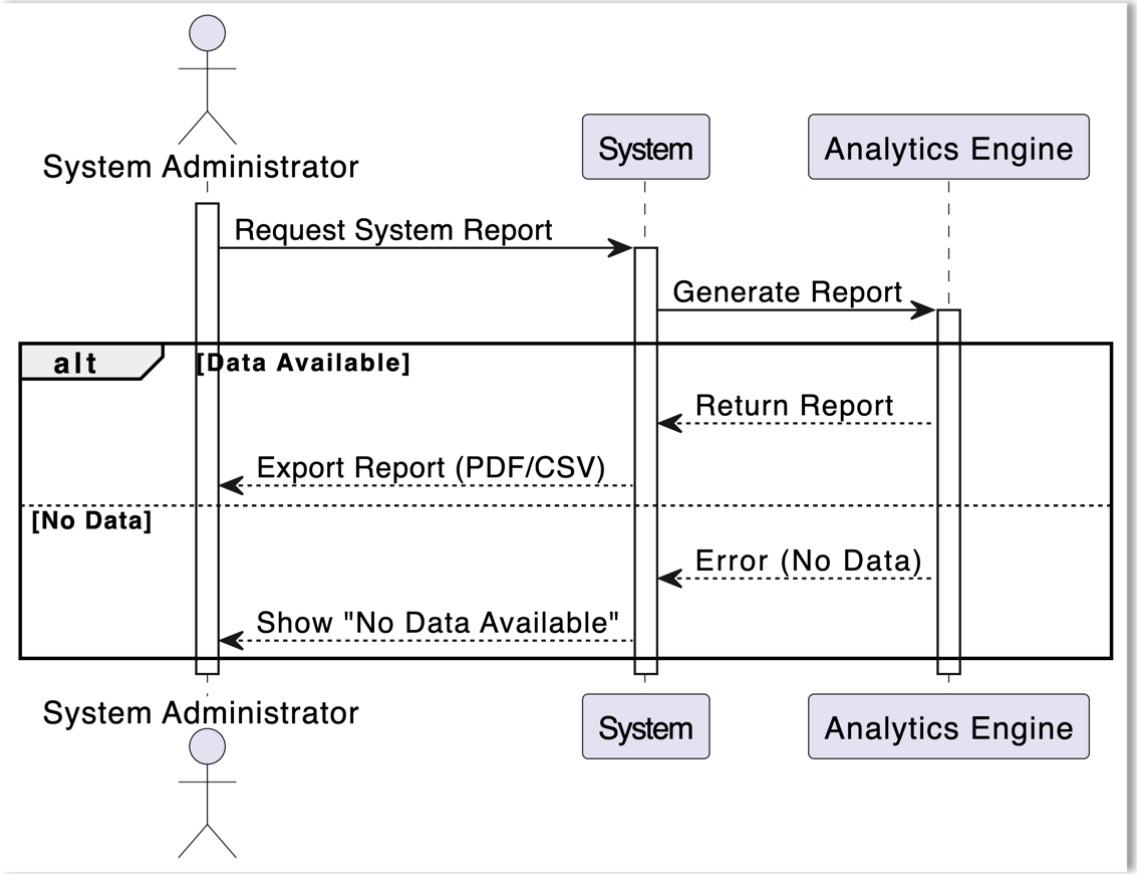


Figure 51 - Generate System Reports

- **System Administrator** requests a system report via the **System**.
- **System** processes the request and forwards it to the **Analytics Engine**.
- **Analytics Engine** generates the report based on available data.
- If data exists, **Analytics Engine** returns the completed report.
- **System** receives the report and prepares it for export.
- **System** provides the report in PDF/CSV format for download.
- If no data is available, **Analytics Engine** sends an error message.
- **System** informs the **System Administrator** that no data is available.
- **System Administrator** either downloads the report or sees the error message.

- The **System** ensures smooth data retrieval and report generation.
- The **Analytics Engine** continuously processes and analyzes data.
- The process repeats for further system report requests.

- **View and Analyze System Logs:**

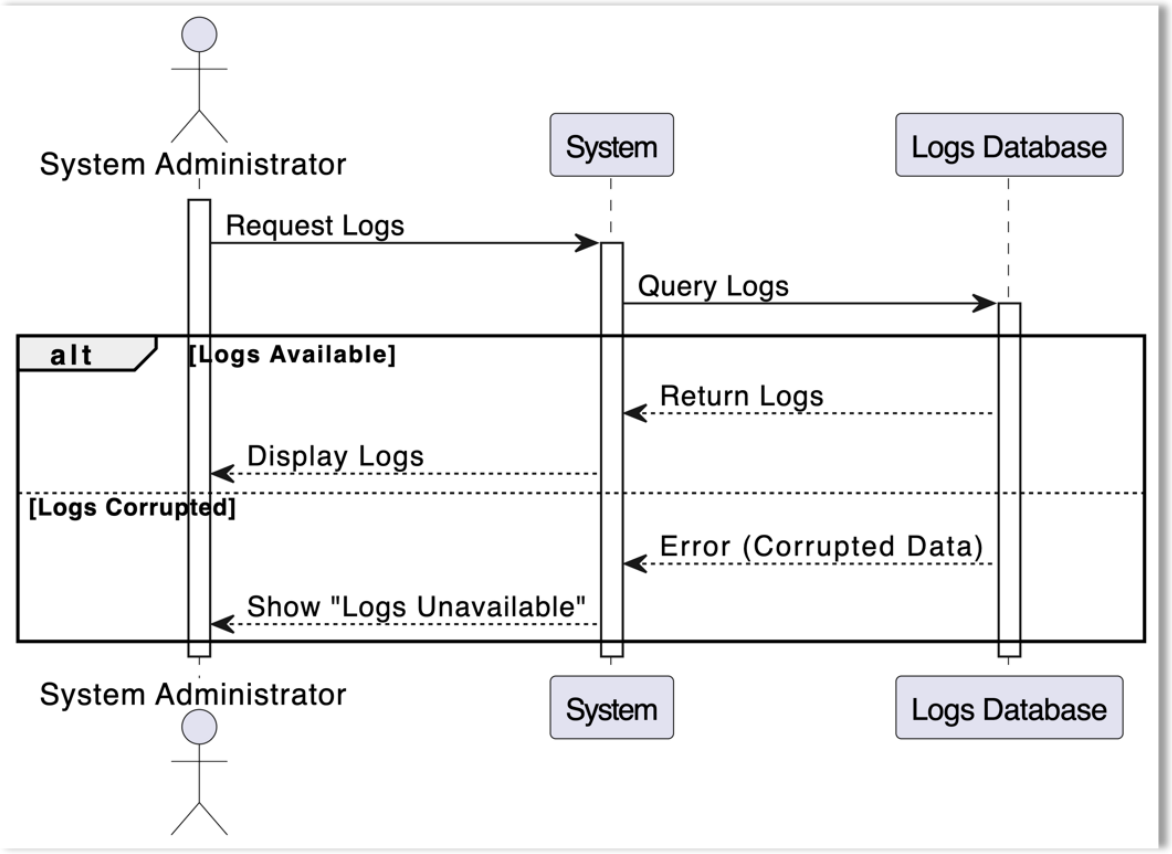


Figure 52 - View and Analyze System Logs

- **System Administrator** requests logs from the **System** for monitoring or troubleshooting.
- **System** processes the request and queries the **Logs Database**.
- **Logs Database** retrieves the requested logs.
- If logs are available and intact, **Logs Database** returns them to the **System**.
- **System** displays the logs to the **System Administrator**.
- If logs are corrupted, **Logs Database** sends an error response.
- **System** notifies the **System Administrator** that logs are unavailable.
- **System Administrator** either views the logs or receives an error message.

- The **System** ensures smooth log retrieval and error handling.
- The **Logs Database** maintains log records and detects data corruption.
- **System Administrator** takes necessary actions based on retrieved logs.
- The process repeats for future log requests.

- **Update System Software:**

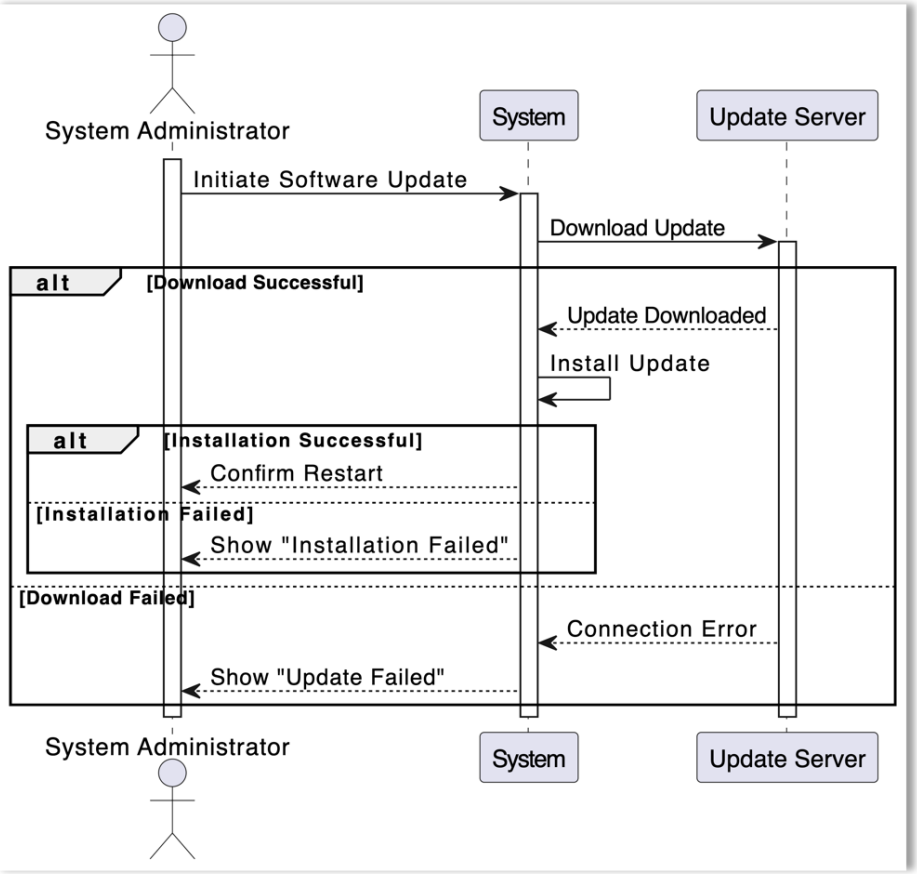


Figure 53 - Update System Software

- **System Administrator** initiates a software update request via the **System**.
- **System** requests the update package from the **Update Server**.
- **Update Server** attempts to provide the update package.
- If the download is successful, **System** proceeds with the installation.
- If the installation is successful, **System** prompts for a system restart.
- If the installation fails, **System** notifies the **System Administrator**.

- If the download fails, **Update Server** sends a connection error response.
- **System** displays an "Update Failed" message to the **System Administrator**.
- **System Administrator** either confirms the restart or receives an error message.
- The **System** ensures update integrity and error handling.
- The **Update Server** maintains update packages and ensures availability.
- The process repeats for future software updates.

- **Configure System Parameters:**

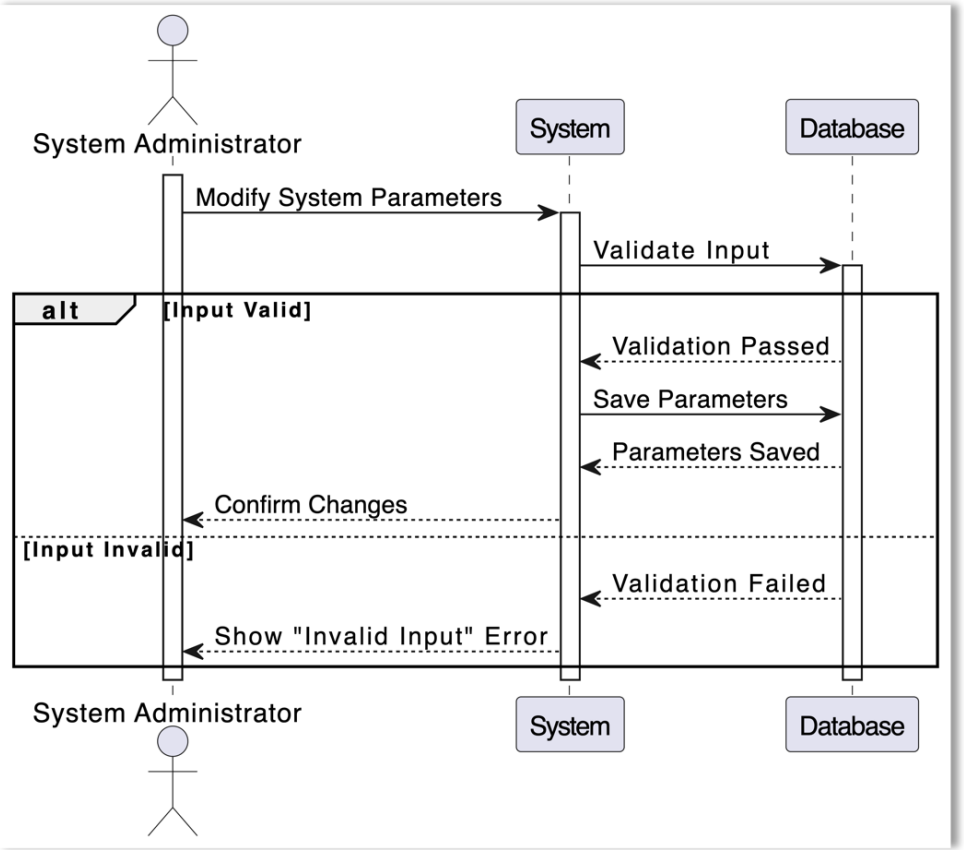


Figure 54 - Configure System Parameters

- **System Administrator** modifies system parameters via the **System**.
- **System** processes the request and sends it to the **Database** for validation.
- **Database** checks if the input meets required constraints.
- If valid, **Database** confirms validation to the **System**.

- **System** saves the parameters in the **Database**.
- **Database** confirms successful storage of the new parameters.
- **System** notifies the **System Administrator** that changes were successfully applied.
- If input is invalid, **Database** sends a validation failure response.
- **System** displays an "Invalid Input" error to the **System Administrator**.
- **System Administrator** either receives confirmation or corrects the input.
- The **System** ensures data validation and secure parameter updates.
- The process repeats for future parameter modifications.

- **Manage Disease Detection Models:**

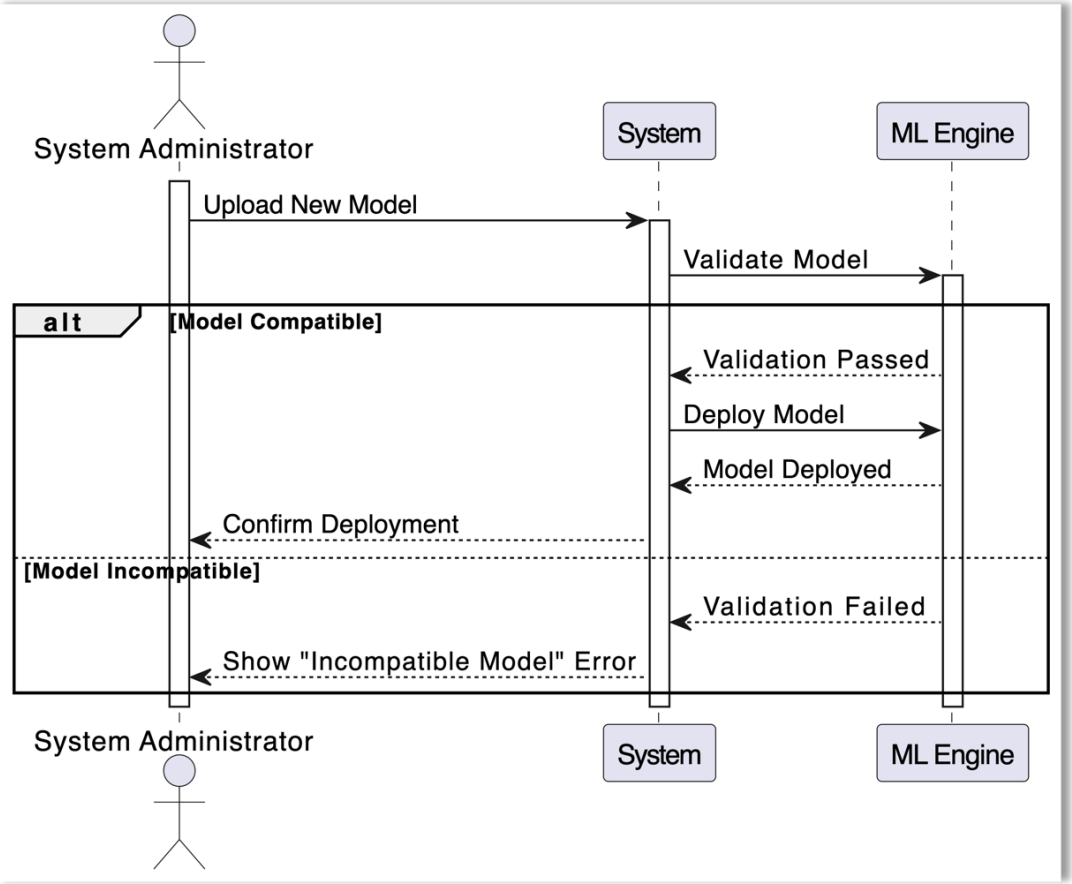


Figure 55 - Manage Disease Detection Models

- **System Administrator** uploads a new ML model via the **System**.
- **System** processes the request and sends the model to the **ML Engine** for validation.

- **ML Engine** checks the model for compatibility.
- If the model is compatible, **ML Engine** confirms validation.
- **System** proceeds with deploying the validated model via the **ML Engine**.
- **ML Engine** deploys the model and confirms completion.
- **System** notifies the **System Administrator** of the successful deployment.
- If the model is incompatible, **ML Engine** sends a validation failure response.
- **System** displays an "Incompatible Model" error to the **System Administrator**.
- **System Administrator** either receives confirmation or corrects the model.
- The **ML Engine** ensures model integrity and compatibility before deployment.
- The process repeats for future model uploads and deployments.

- **Monitor System Performance and Health:**

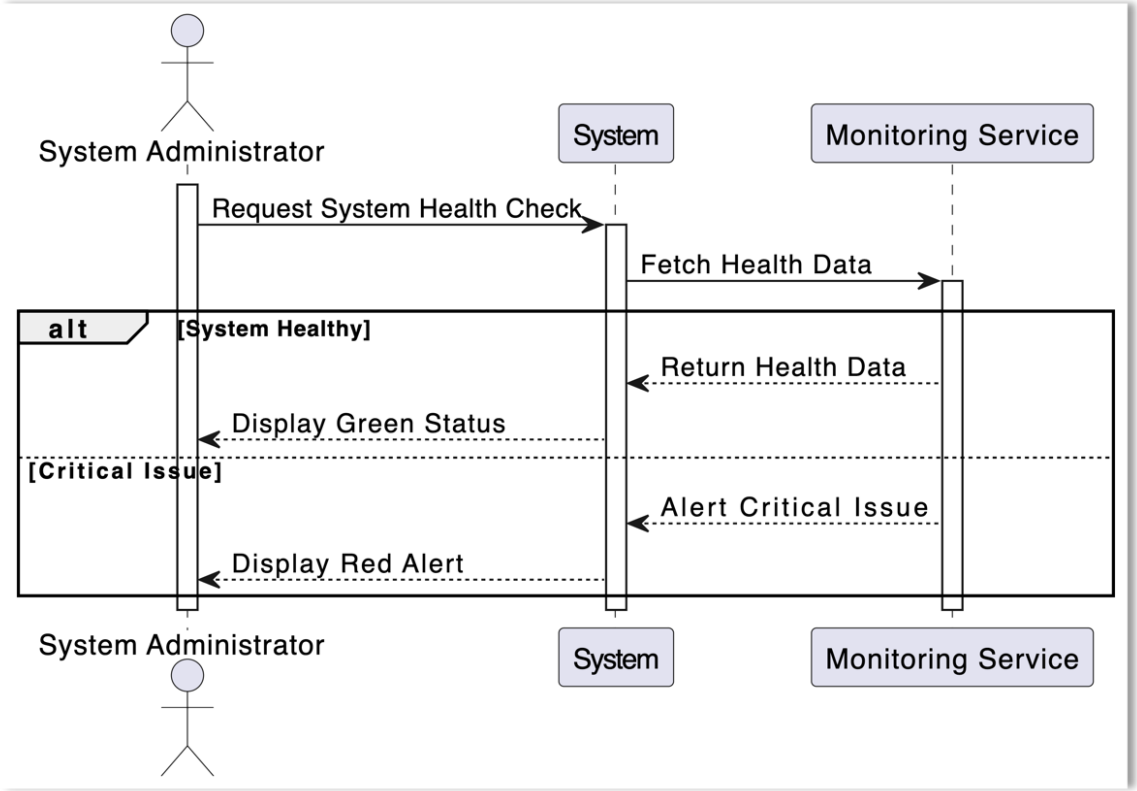


Figure 56 - Monitor System Performance and Health

- **System Administrator** requests a system health check via the **System**.

- **System** processes the request and queries the **Monitoring Service**.
- **Monitoring Service** collects real-time system health data.
- If the system is healthy, **Monitoring Service** returns a "Healthy" status.
- **System** displays a green status to the **System Administrator**.
- If a critical issue is detected, **Monitoring Service** sends an alert.
- **System** displays a red alert indicating a critical issue.
- **System Administrator** reviews the status and takes necessary action.
- The **System** ensures smooth communication and timely alerts.
- The **Monitoring Service** continuously tracks system health metrics.
- **System Administrator** relies on the health check to maintain system stability.
- The process repeats for continuous system monitoring.

3.4.3 Activity Diagrams

An activity diagram is a behavioral diagram in UML (Unified Modeling Language) that represents the flow of actions and activities within a system or process. It is essentially a flowchart that shows the flow from one activity to another, highlighting the sequence and conditions for coordinating lower-level behaviors. Activity diagrams are widely used in business process modeling as well as in software development to model the logic of complex operations and processes.

Activity Diagrams of our System:

- **Early Disease Detection:**

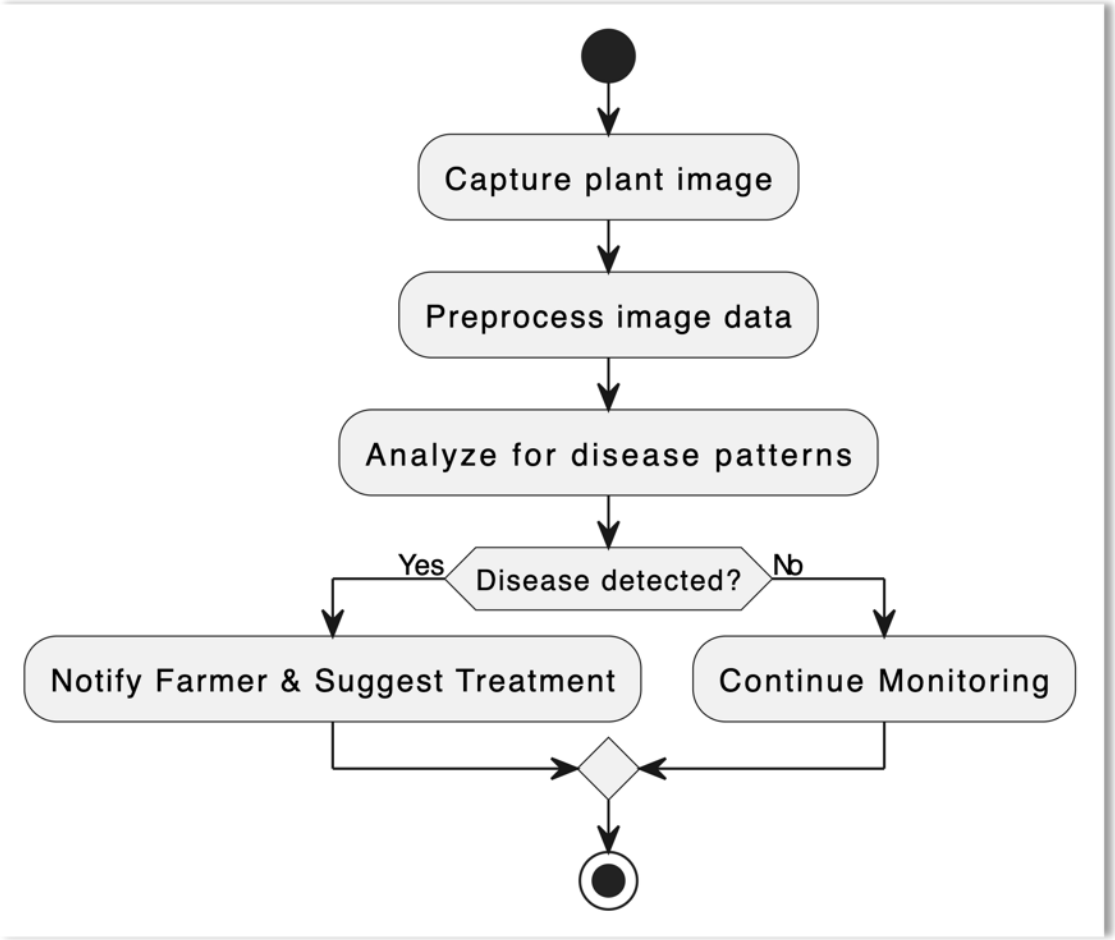


Figure 57 - Early Disease Detection

Description:

- **System starts** the plant disease detection workflow.
- **Capture plant image** using imaging sensors.
- **Preprocess image data** by enhancing brightness, contrast, and reducing noise.
- **Analyze for disease patterns** using AI models trained on plant health data.
- **Decision: Disease detected?** If yes, proceed with notification; otherwise, continue monitoring.
- **If disease is detected**, notify the farmer and suggest treatment options.

- **Provide insights** on disease type, severity, and recommended actions.
- **If no disease is detected**, the system continues monitoring without alerts.
- **Loop back** to periodic monitoring and capturing new images.
- **System ensures** automation, reducing manual intervention in disease detection.
- **Enhances farm management** by providing timely alerts and AI-driven analysis.
- **Process ends** and awaits the next monitoring cycle.

- **Soil Monitoring :**

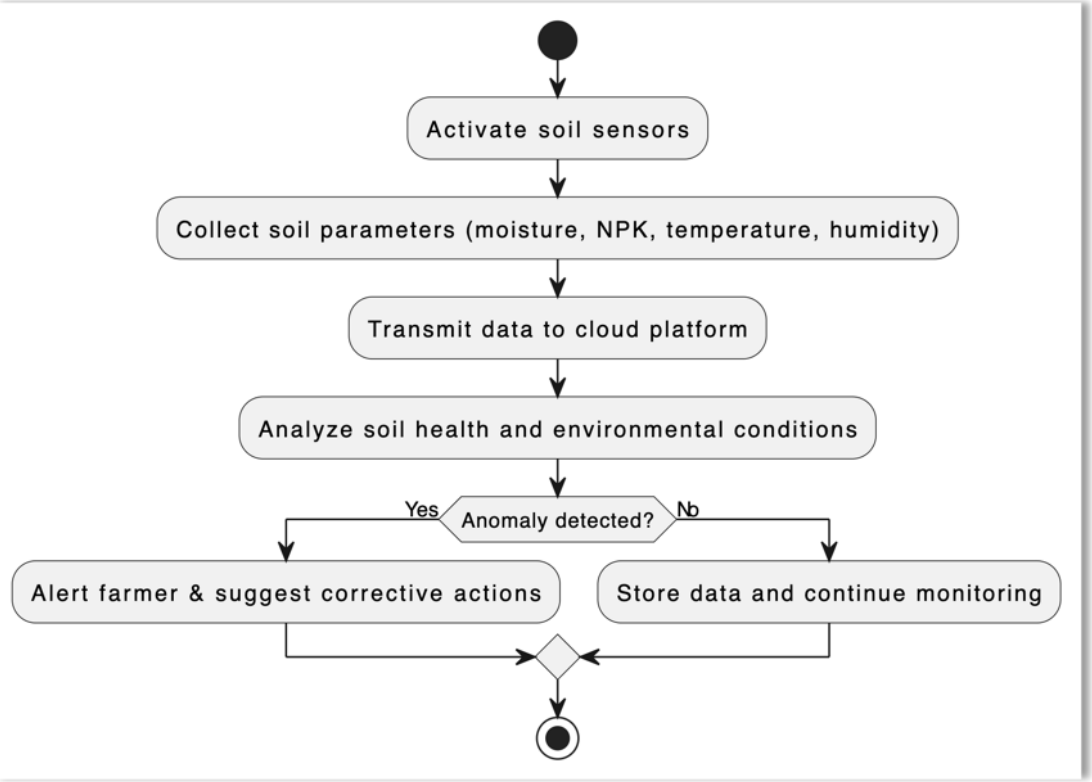


Figure 58 - Soil Monitoring

Description:

- System starts **the soil monitoring process.**
- Activate soil sensors **to begin data collection.**
- Collect soil parameters **including moisture, NPK, temperature, and**

humidity.

- Transmit data **to a cloud platform for processing.**
- Analyze soil health **and environmental conditions using AI-based analytics.**
- Decision: Anomaly detected? **If yes, proceed with alerting; otherwise, continue monitoring.**
- If an anomaly is detected**, notify the farmer with corrective action suggestions.**
- Provide recommendations **on irrigation, fertilization, and environmental adjustments.**
- If no anomaly is detected**, store data and continue monitoring.**
- Loop back **to periodic monitoring and collecting new data.**
- System ensures **automation, optimizing resource use and crop health.**
- Process ends **and awaits the next monitoring cycle.**

- **Sustainability and Cost Efficiency:**

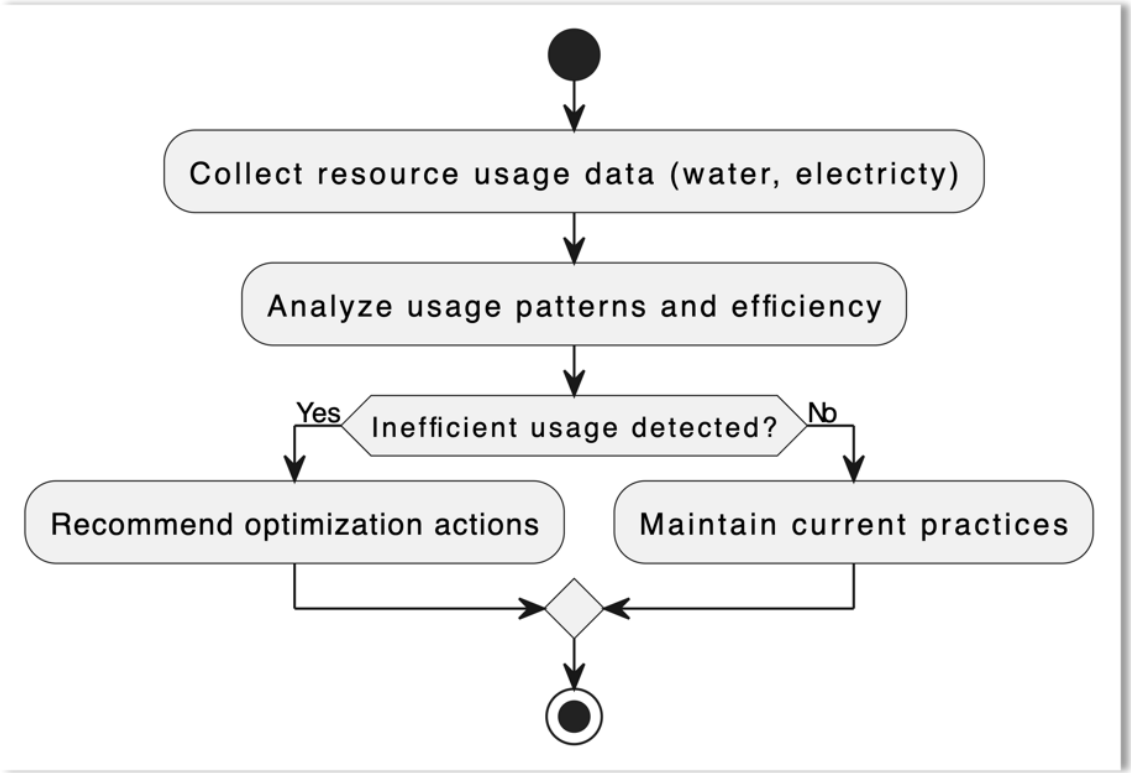


Figure 59 - Sustainability and Cost Efficiency

`Description:

- **System starts** the resource monitoring process.
- **Collect resource usage data** including water and electricity consumption.
- **Analyze usage patterns** using AI-based efficiency analysis.
- **Decision: Inefficient usage detected?** If yes, proceed with recommendations; otherwise, maintain current practices.
- **If inefficiencies are detected**, provide optimization suggestions.
- **Recommend corrective actions** such as adjusting irrigation, power usage, or scheduling.
- **Notify the farmer** via alerts or dashboard recommendations.
- **If no inefficiencies are found**, continue monitoring without changes.
- **Store data** for future trend analysis and benchmarking.
- **Loop back** to periodic monitoring and efficiency evaluations.
- **System ensures** resource conservation and cost optimization.
- **Process ends** and awaits the next monitoring cycle.

- **Data-Driven Precision Farming:**

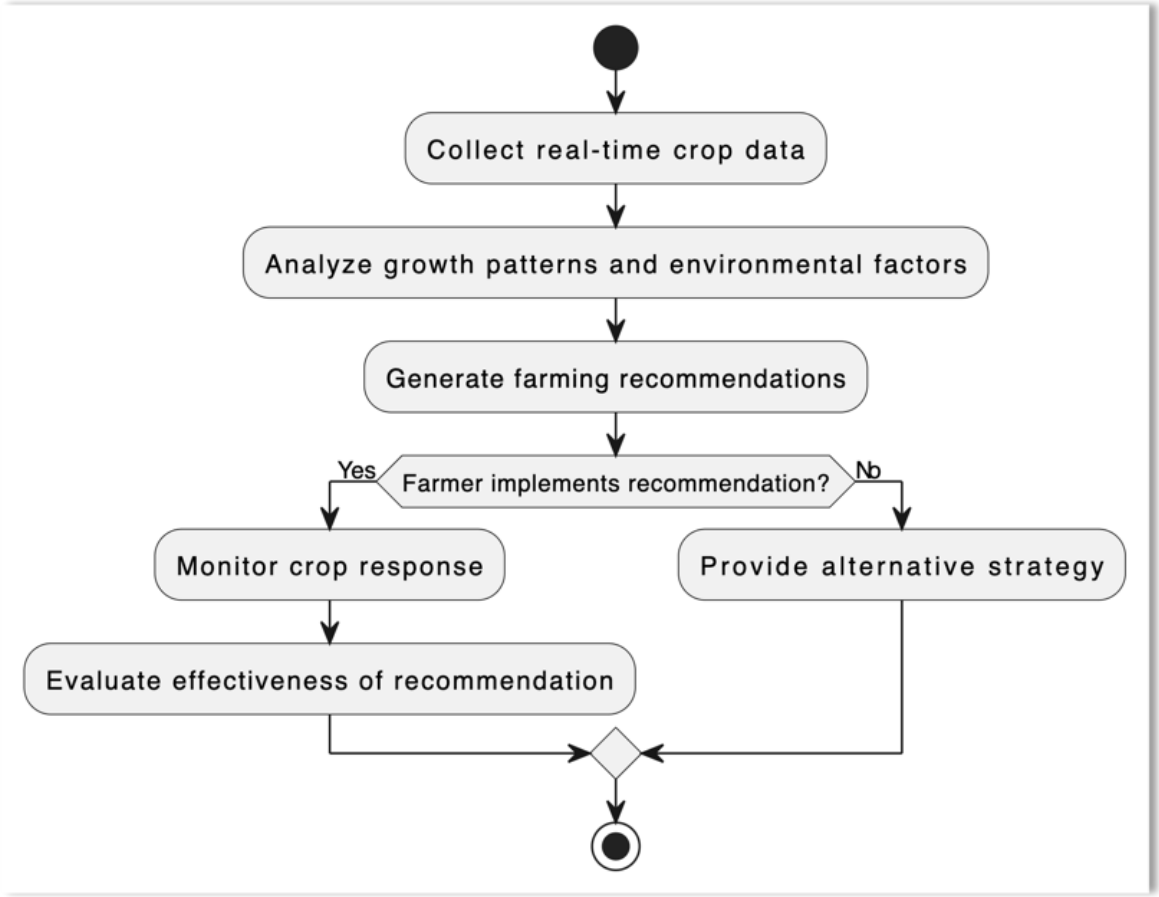


Figure 60 - Data-Driven Precision Farming

Description:

- System starts **real-time crop monitoring.**
- Collect crop data **using IoT sensors and remote sensing.**
- Analyze growth patterns **and environmental factors.**
- Generate farming recommendations **based on the analysis.**
- Decision: Farmer implements recommendation? **If yes, proceed with monitoring; otherwise, adjust strategy.**
- If implemented**, monitor crop response and adaptation.**
- Evaluate effectiveness **of the recommendation.**
- If effective**, continue monitoring and refine future recommendations.**
- If not effective**, provide an alternative strategy for better results.**

- Loop back **to periodic monitoring and adjustments.**
- System ensures **continuous learning and adaptive decision-making.**
- Process ends **and awaits the next monitoring cycle.**

- **Crop Recommendation System:**

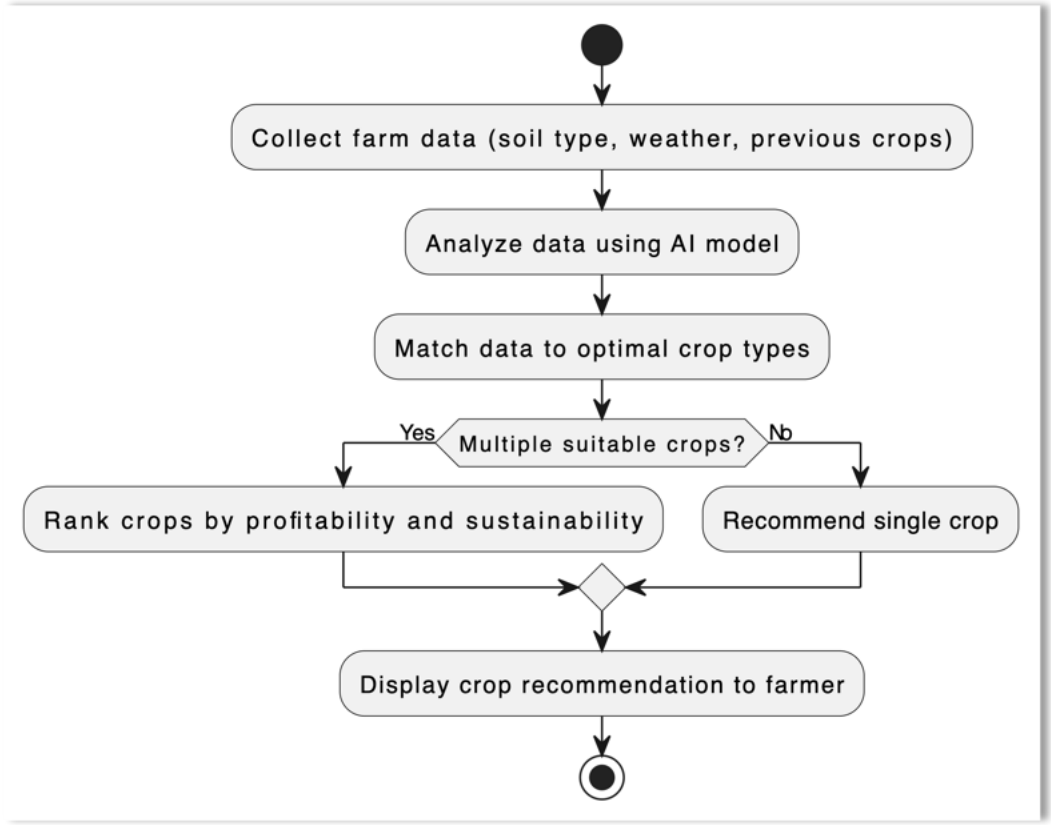


Figure 61 - Crop Recommendation System

Description:

- **System starts** the crop recommendation process.
- **Collect farm data** including soil type, weather, and previous crops.
- **Analyze data** using an AI model for insights.
- **Match farm data** with optimal crop types.
- **Decision: Multiple suitable crops?** If yes, rank them; if no, proceed with a single recommendation.
- **If multiple crops are suitable**, rank them by profitability and

sustainability.

- **If only one crop is suitable**, recommend it directly.
- **Generate a detailed recommendation** based on economic and environmental factors.
- **Display recommendations** to the farmer through a mobile app or dashboard.
- **Provide insights** such as yield estimates and sustainability tips.
- **System ensures** continuous learning for improved accuracy.
- **Process ends** and awaits the next recommendation cycle.

- **Real-Time Dashboard Monitoring:**

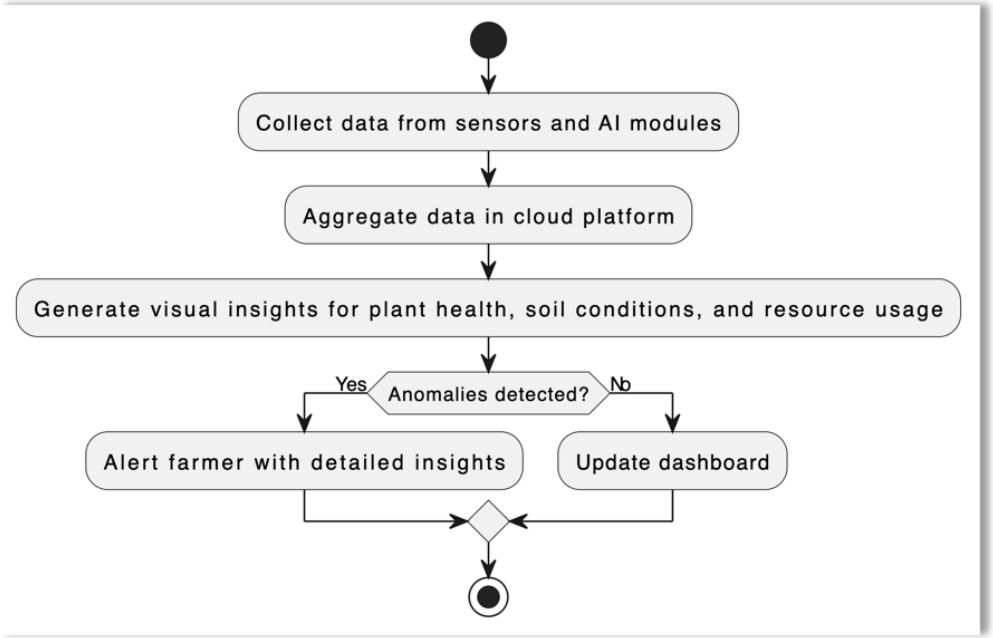


Figure 62 - Real-Time Dashboard Monitoring

Description:

- System starts **the agricultural monitoring process.**
- Collect data **from sensors and AI modules in the field.**
- Aggregate data **in a cloud platform for centralized analysis.**
- Generate visual insights **for plant health, soil conditions, and resource usage.**

- Decision: Anomalies detected? **If yes, proceed with alerting; otherwise, update the dashboard.**
- If an anomaly is detected**, alert the farmer with detailed insights.**
- Provide recommendations **on irrigation, fertilization, or crop management.**
- If no anomaly is detected**, update the dashboard with current conditions.**
- Loop back **to continuous monitoring for new data.**
- System ensures **real-time insights for decision-making.**
- Enhance farming efficiency **by reducing resource waste and optimizing crop yield.**
- Process ends **and awaits the next monitoring cycle.**

- **Irrigation Management System:**

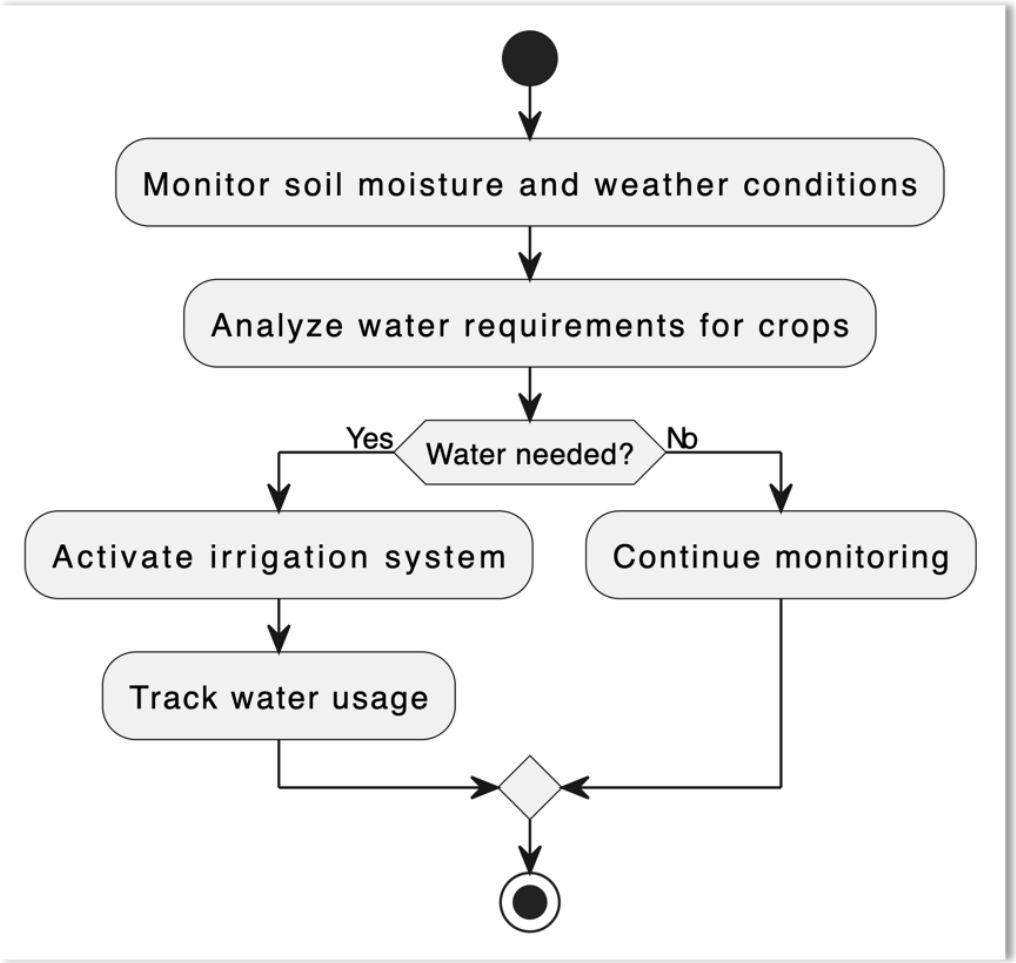


Figure 63 - Irrigation Management System

Description:

- **System starts** the smart irrigation management cycle.
- **Monitor soil moisture** and weather conditions using sensors.
- **Analyze water needs** based on crop type, growth stage, and weather data.
- **Decision: Water needed?** If yes, proceed with irrigation; otherwise, continue monitoring.
- **If water is required**, activate the irrigation system to supply water.
- **Adjust irrigation** based on calculated water volume and specific field zones.
- **Track water usage** with flow meters to monitor consumption and detect leaks.
- **If no water is needed**, system continues monitoring for future changes.
- **Store data** for long-term analysis and optimization of water resources.
- **Loop back** for continuous monitoring and irrigation adjustments.
- **System ensures** efficient water use and crop health.
- **Process ends** and awaits the next monitoring cycle.

- **Farmer Notification System:**

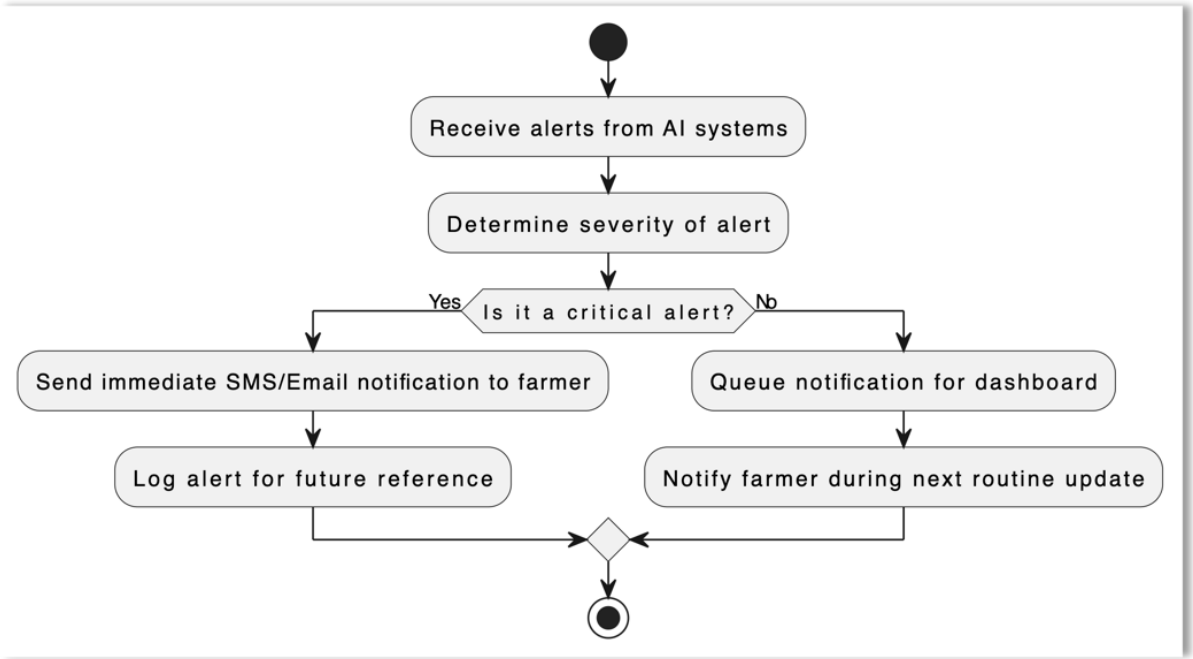


Figure 64 - Farmer Notification System

Description:

- System starts **the automated alert processing cycle.**
- Receive alerts **from AI monitoring systems analyzing farm conditions.**
- Determine severity **of each alert based on impact and urgency.**
- Decision: Critical alert? **If yes, notify immediately; otherwise, queue for review.**
- If critical**, send SMS/email notification to the farmer for immediate action.**
- Log alert **for future reference and data analysis.**
- If not critical**, queue the alert for dashboard review.**
- Notify farmer **of non-critical alerts during routine updates.**
- Store alerts **in a centralized system for trend analysis and farm insights.**
- Loop back **to continuously monitor new alerts.**
- System ensures **timely interventions while reducing unnecessary notifications.**
- Process ends **and awaits the next incoming alert.**

3.4.4 Class Diagram:

A class diagram is a type of diagram that shows the structure of a system by modeling its classes, their attributes, operations, and the relationships among them. It is one of the most common types of diagrams in UML (Unified Modeling Language), which is a standard way of describing object-oriented systems. It can help you understand the system better, design or document its architecture, or generate code from it. A class diagram consists of the following elements:

- **Classes:**

A class is a blueprint or template for creating objects. It defines the properties and behaviors of a group of objects that share the same characteristics. A class is represented by a rectangle with three compartments: the top one contains the class

name, the middle one contains the class attributes, and the bottom one contains the class operations or methods. For example, a class named Person may have attributes like name, age, and gender, and operations like walk, talk, and sleep.

- **Relationships:**

A relationship is a connection or association between two or more classes. It shows how the classes interact or depend on each other. There are different types of relationships, such as:

  - **Inheritance:**

This represents an "is-a" relationship, where a subclass inherits the attributes and operations of a superclass. It is shown by a solid line with a hollow arrowhead pointing from the subclass to the superclass. For example, a student class may inherit from a Person class, meaning that a student is a person and has all the properties and behaviors of a person, plus some additional ones.

  - **Association:**

This represents a "has-a" or "uses-a" relationship, where one class has a reference or link to another class. It is shown by a solid line connecting the two classes, optionally with a name and a multiplicity at each end. For example, a Car class may have an association with a Wheel class, meaning that a car has four wheels, and each wheel belongs to one car.

  - **Aggregation:**

This represents a "part-of" relationship, where one class is a component or part of another class. It is a special kind of association that implies a whole-part hierarchy. It is shown by a solid line with a hollow diamond at the end of the whole. For example, a university class may have an aggregation with a department class, meaning that a university consists of several departments, and each department is part of one university.

  - **Composition:**

This represents a "strong part-of" relationship, where one class is a component or part of another class, and the component cannot exist without the whole. It is a

stronger form of aggregation that implies ownership and exclusive responsibility. It is shown by a solid line with a filled diamond at the end of the whole. For example, a House class may have a composition with a Room class, meaning that a house consists of several rooms, and each room is part of one house, and cannot exist without the house.

- **Dependency:**

This represents a "depends-on" relationship, where one class depends on another class for some reason. It is a weak form of association that implies a temporary or indirect link. It is shown by a dashed line with an open arrowhead pointing from the dependent class to the independent class. For example, a Driver class may have a dependency with a Car class, meaning that a driver uses a car for driving, but does not own or control the car.

Class Diagram in our system :

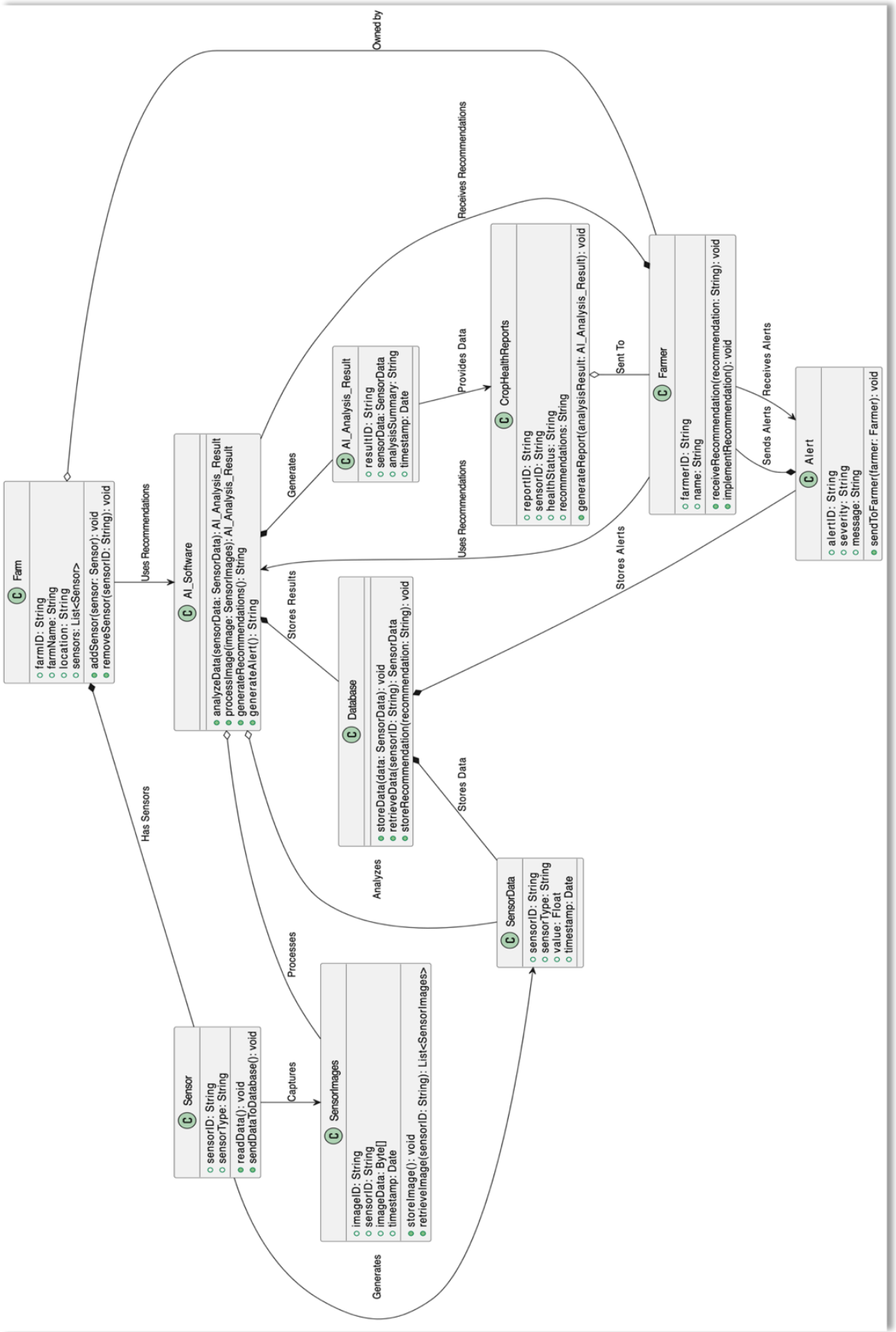


Figure 65 - Class Diagram in our system

1. Sensor

- **Attributes:**
  - sensorID: Uniquely identifies each sensor.
  - sensorType: Specifies the type of sensor (e.g., temperature, humidity, soil moisture, or imagery).
- **Methods:**
  - readData(): Collects data from the environment.
  - sendDataToDatabase(): Transfers the collected data to the centralized database.
- **Relationships:**
  - **Generates** → SensorData: Sensors produce raw data, which is encapsulated within the SensorData entity.
  - **Captures** → SensorImages: If the sensor is an imaging device (e.g., a drone or satellite), it captures images for further analysis.

2. SensorData

- **Attributes:**
  - sensorID: References the originating sensor.
  - sensorType: Describes the data type (e.g., temperature, humidity).
  - value: The recorded environmental measurement.
  - timestamp: The exact time of data collection.
- **Relationships:**
  - **Analyzed By** → AI_Software: AI algorithms process this data to extract meaningful insights.
  - **Stored In** → Database: Ensures historical data availability for trend analysis and future reference.

3. SensorImages

- **Attributes:**
  - imageID: Unique identifier for each image.
  - sensorID: Links the image to its source sensor.

- imageData: The binary image content.
- timestamp: Captures when the image was taken.

- **Methods:**
  - storeImage(): Saves the image in the database for future analysis.
  - retrieveImage(sensorID): Fetches historical images for a given sensor.
- **Relationships:**
  - **Processed By** → AI_Software: Analyzed to derive visual insights, such as crop health status.

4. AI_Software

- **Attributes:** None (focuses on functionality).
- **Methods:**
  - analyzeData(sensorData): Processes numerical sensor data to detect patterns or anomalies.
  - processImage(image): Analyzes images using computer vision techniques.
  - generateRecommendations(): Suggests actionable steps based on analyzed data.
  - generateAlert(): Triggers alerts for critical conditions (e.g., drought or pest infestation).
- **Relationships:**
  - **Analyzes** → SensorData: Extracts insights from raw data.
  - **Processes** → SensorImages: Utilizes image analysis algorithms for visual data interpretation.
  - **Generates** → AI_Analysis_Result: Consolidates findings into a structured result object.
  - **Stores Results** → Database: Persists analyzed data and recommendations for future reference.
  - **Sends Recommendations To** → Farmer: Provides actionable insights to the farmer for improved crop management.

5. AI_Analysis_Result

- **Attributes:**
  - resultID: Unique identifier for each analysis result.
  - sensorData: Links the result to the original data analyzed.
  - analysisSummary: Summarizes findings and interpretations.
  - timestamp: When the analysis was completed.
- **Relationships:**
  - **Provides Data To** → CropHealthReports: Utilized in generating comprehensive crop health reports.

6. CropHealthReports

- **Attributes:**
  - reportID: Unique identifier for each report.
  - sensorID: Links the report to the relevant sensor.
  - healthStatus: Summary of crop health conditions (e.g., healthy, stressed, diseased).
  - recommendations: Suggested actions based on the analysis.
- **Methods:**
  - generateReport(analysisResult): Compiles a detailed report using the analysis results.
- **Relationships:**
  - **Sent To** → Farmer: Informs the farmer about crop health and recommended actions.

7. Database

- **Attributes:** None (focuses on data storage functionality).
- **Methods:**
  - storeData(data): Saves sensor data and analysis results.
  - retrieveData(sensorID): Fetches historical data for trend analysis.

- storeRecommendation(recommendation):Archives recommendations for future reference.

- **Relationships:**
  - **Stores Data** → SensorData: Ensures historical environmental data availability.
  - **Stores Alerts** → Alert: Maintains alert records for review and auditing.

8. Alert

- **Attributes:**
  - alertID: Unique identifier for each alert.
  - severity: Indicates the alert's urgency (e.g., low, medium, high).
  - message: Describes the situation requiring attention.
- **Methods:**
  - sendToFarmer(farmer): Notifies the farmer of critical conditions or recommendations.
- **Relationships:**
  - **Sends Alerts To** → Farmer: Ensures timely notification of issues.
  - **Stored In** → Database: Maintains a log of all alerts for future auditing and analysis.

9. Farmer

- **Attributes:**
  - farmerID: Uniquely identifies the farmer.
  - name: Farmer's name.
- **Methods:**
  - receiveRecommendation(recommendation): Receives actionable insights from AI software.
  - implementRecommendation(): Executes the suggested actions on the farm.
- **Relationships:**

- **Receives Alerts From** → Alert: Informed of critical conditions or emergencies.
- **Receives Recommendations From** → AI_Software: Utilizes insights for crop management.

10. Farm

- **Attributes:**
  - farmID: Unique identifier for each farm.
  - farmName: Name of the farm.
  - location: Geographical location of the farm.
  - sensors: A list of sensors deployed on the farm.
- **Methods:**
  - addSensor(sensor): Installs a new sensor on the farm.
  - removeSensor(sensorID): Uninstalls a sensor from the farm.
- **Relationships:**
  - **Has Sensors** → Sensor: Deploys multiple sensors to monitor environmental conditions.
  - **Owned By** → Farmer: Indicates ownership and management responsibility.
  - **Uses Recommendations From** → AI_Software: Implements recommendations for optimal crop management.

3.4.5 State Diagram

This state diagram outlines the user journey within our platform. It visualizes the various screens and features users can access, along with the available actions and transitions between them.

State Diagrams of our System:

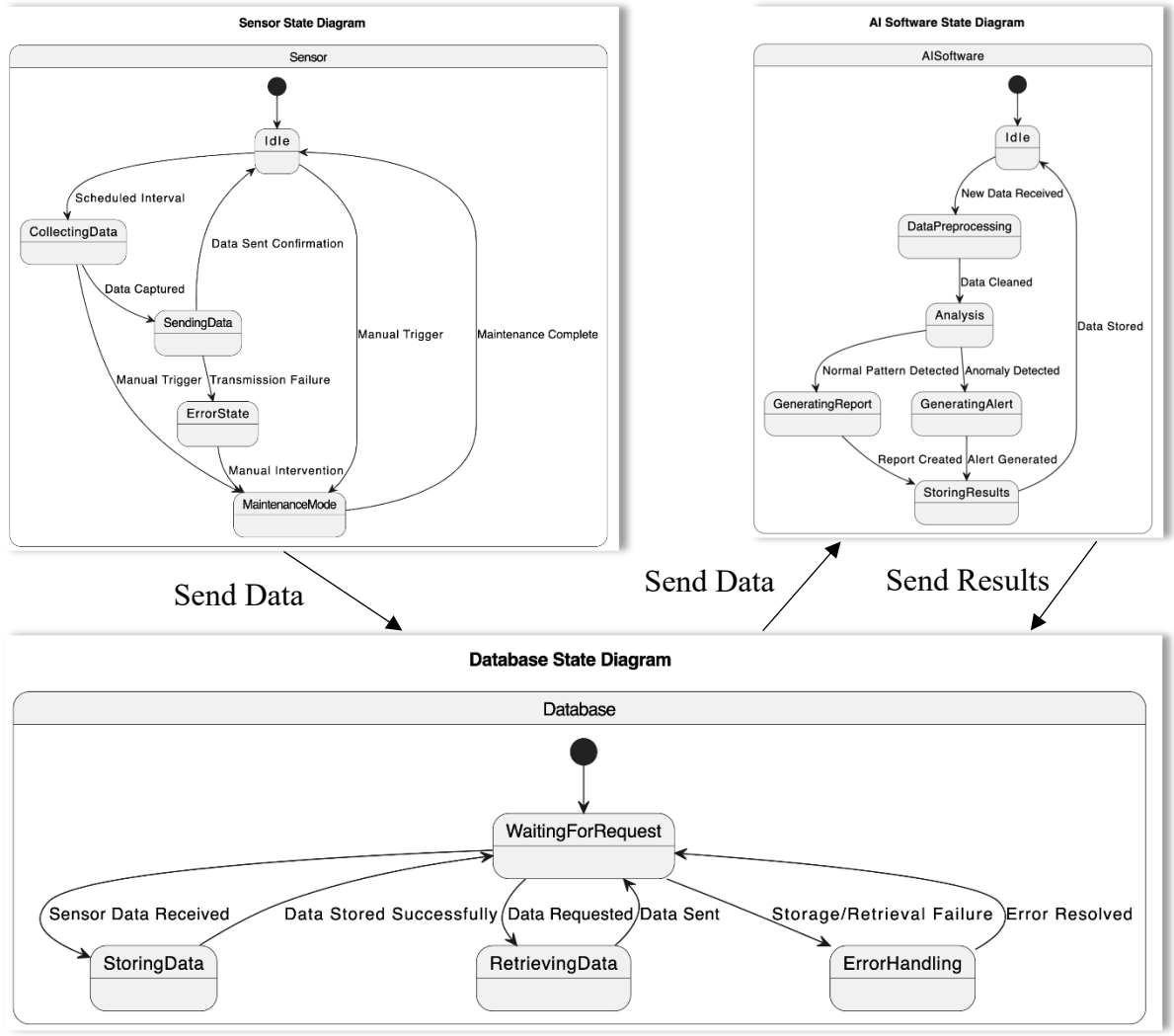


Figure 66 - State Diagrams of our System

3.5 Business Model:

A business model is a conceptual structure that supports the viability of a business, outlining how it operates, makes money, and how it intends to achieve its goals. It includes the plans for the products or services the business plans to sell, its identified target market, and any expected expenses. Business models are essential for both new and established companies. They help stakeholders understand how a company is positioned within its industry and how it intends to sustain itself and grow.

3.5.1 Components of a Business Model:

- **Value Proposition:**
  - What problem does your product or service solve?
  - How does it add value for your customers?
- **Customer Segments:**
  - Who are your target customers?
  - What are their needs, habits, and preferences?
- **Channels:**
  - Through what means will you reach your customers and deliver your product or service?
  - This can include physical locations, online platforms, social media, etc.
- **Customer Relationships:**
  - How do you plan to interact with customers?
  - Will you offer personalized assistance, self-service options, automated services, etc.?
- **Revenue Streams:**
  - How does the business make money?
  - This could be through sales, subscriptions, advertising, affiliate revenue, etc.

- **Key Resources:**
  - What assets are essential to your business?
  - This can include physical assets, intellectual property, human resources, and financial resources.
- **Key Activities:**
  - What core activities does your business undertake to deliver its value proposition?
- **Key Partnerships:**
  - Who are your key partners or suppliers?
  - What resources do they provide, and how do they contribute to your business?
- **Cost Structure:**
  - What are the significant costs involved in operating your business?
  - How do they align with your revenue streams?

3.5.2 The Business Model Our System:

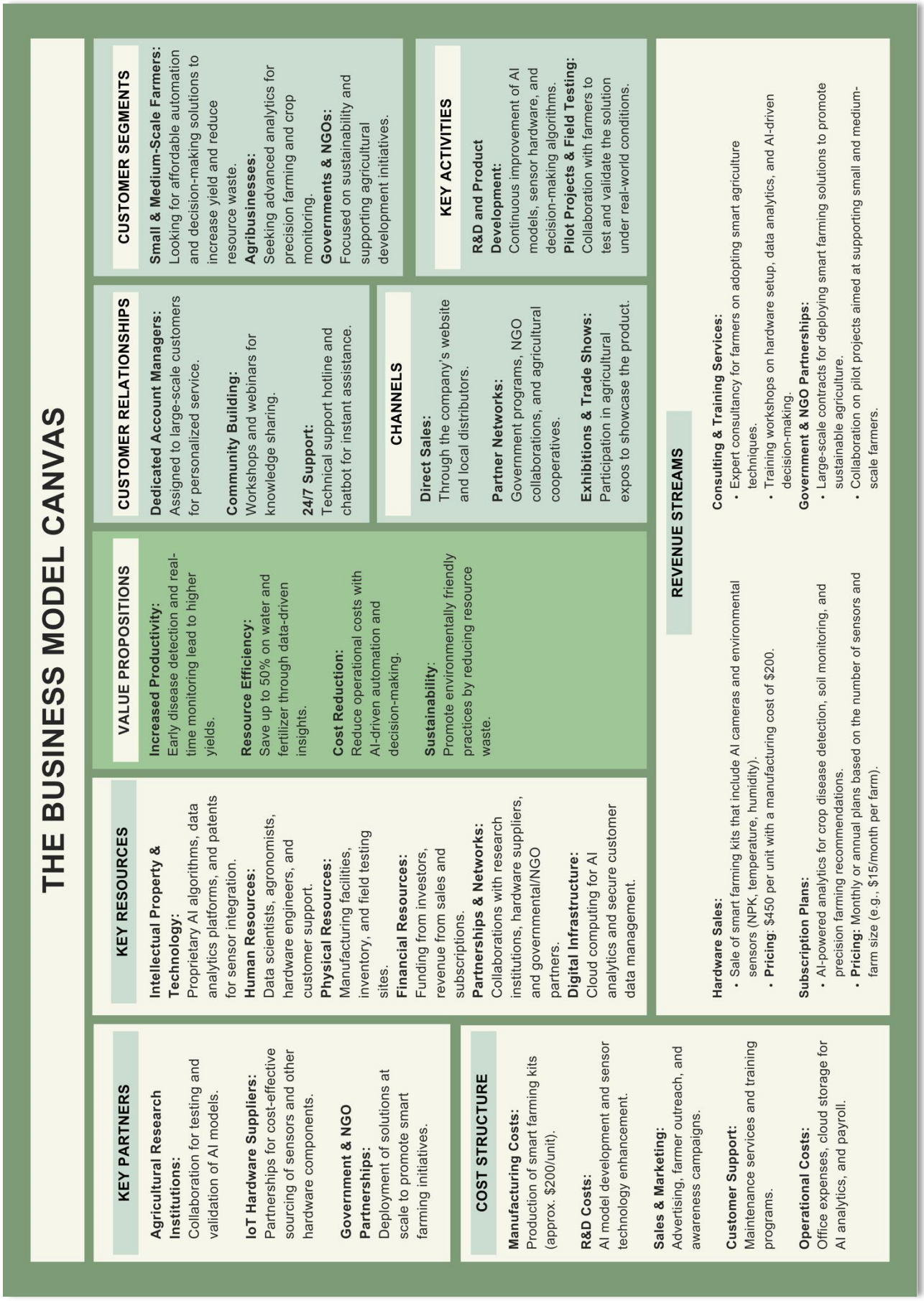


Figure 67 - The Business Model Our System

3.6 System Architecture:

System architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

System Architecture in our system:

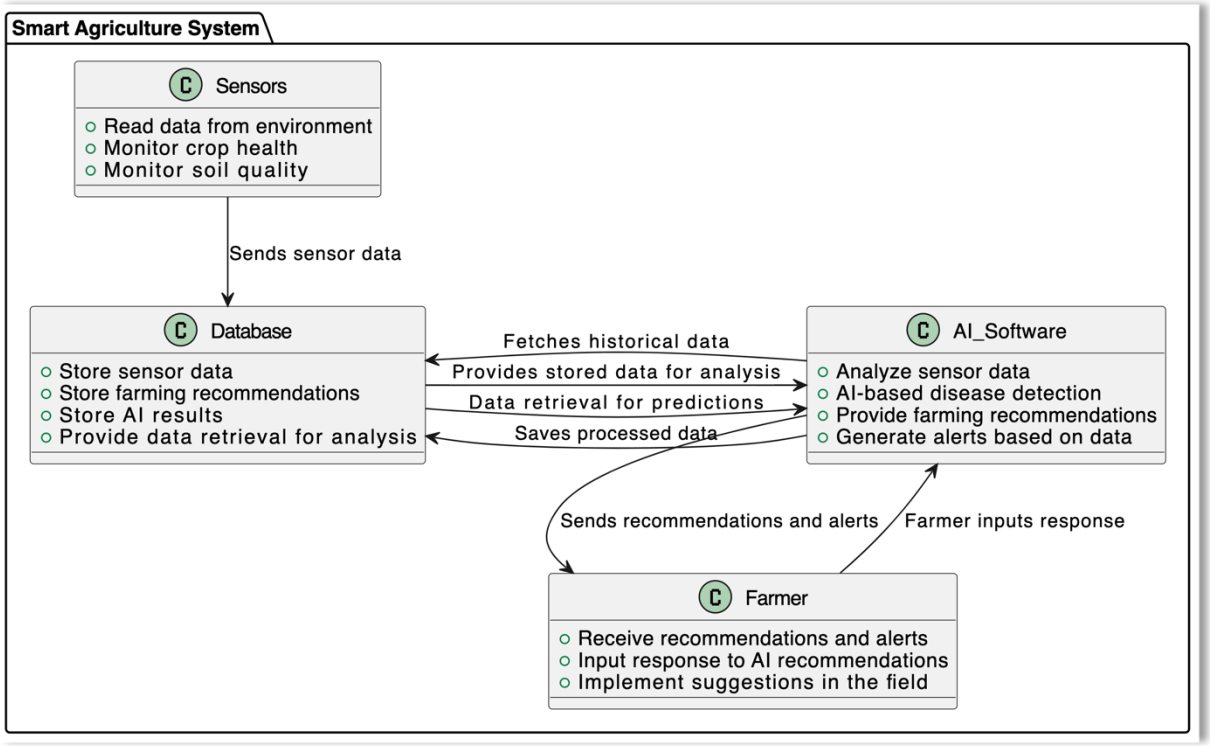


Figure 68 - System Architecture in our system

1. Sensors:

- **Purpose:** These devices are responsible for reading environmental data to monitor crop health and soil quality. They serve as the primary data sources in the system.
- **Responsibilities:**
    - Collect environmental metrics such as humidity, temperature, and soil moisture.
    - Monitor crop health indicators like disease symptoms or nutrient deficiencies.
    - Transmit gathered data to the central database for storage and analysis.
- **Relationships:**

- **Sends data to** the **Database** to store real-time readings for historical tracking and analysis.

2. Database:

- **Purpose:** The database serves as the central repository for all system data, including raw sensor readings, AI-processed results, and farming recommendations.
- **Responsibilities:**
  - Store sensor data securely for future retrieval.
  - Maintain farming recommendations generated by the AI Software.
  - Keep historical data to support predictive analytics and trend analysis.
- **Relationships:**
  - **Receives data from** the **Sensors** for storage.
  - **Provides data to** the **AI Software** for analysis and predictions.
  - **Stores processed data** sent by the **AI Software** after analysis.
  - **Retrieves historical data** to assist the **AI Software** in generating predictions.

3. AI Software:

- **Purpose:** This component performs advanced data analytics using AI algorithms to detect diseases, provide tailored farming recommendations, and generate alerts.
- **Responsibilities:**
  - Analyze incoming data from the **Database** to detect potential crop diseases or issues.
  - Utilize historical data to make accurate predictions and recommendations.
  - Send actionable insights and alerts to the **Farmer**.
  - Collect feedback from the **Farmer** to improve future recommendations.
- **Relationships:**
  - **Fetches stored data** from the **Database** for analysis.
  - **Saves processed results** back to the **Database**.
  - **Requests historical data** from the **Database** for predictive analytics.
  - **Sends recommendations and alerts** to the **Farmer** for actionable insights.
  - **Receives feedback** from the **Farmer** to refine its algorithms.

4. Farmer

- **Purpose:** The end-user who interacts with the system by receiving alerts and recommendations and providing feedback for continuous improvement.
- **Responsibilities:**
    - Receive AI-generated alerts and farming recommendations.
    - Make decisions and implement suggestions in the field based on AI insights.
    - Provide feedback to the **AI Software** for better customization of recommendations.
- **Relationships:**
    - **Receives notifications** and **recommendations** from the **AI Software**.
    - **Inputs feedback** to the **AI Software** to enhance decision-making algorithms.

Chapter 4
System Design

4.1 Introduction:

After determining the requirements of the system, we will discuss system design in this chapter, which is the process of defining elements of a system like modules, architecture, components and their interfaces and data for a system based on the specified requirements. It is the process of defining, developing and designing systems, which satisfies the specific needs, and requirements of a business or organization.

Design is the act of taking information, creating a product design to be manufactured, and developing systems to meet specific user requirements. Therefore, there are many types to achieve this purpose and meet all requirements, including:

- **Architectural Design:**

focuses on designing a system structure that describes the structure, behavior, perspectives and analysis of this system.

- **Logical design:**

focuses on abstract representation of system data, input and output flows. This is often done by modeling from the actual system. In the context of systems, designs are included. Logical design includes relationship entity diagrams (ERD).

- **Physical Design:**

focuses on actual system I / O operations. Everything related to how data is entered into the system, how to validate it, how it is processed, and how it is presented is covered.

In the **physical design**, the following requirements about the system are defined:

- Input condition.
- Production requirements.
- Storage requirements.
- Processing requirements.
- Control system, backup or recovery.

In other words, the actual design part of the system can be divided into three sub-tasks:

- User interface design.
- Data design.
- Design process.

4.2 Graphical User Interface (GUI):

4.2.1 User Interface Design of Mobile Application:

- **Splash Screen:**

Figure 69 - Splash Screen

- **Login Screen:**

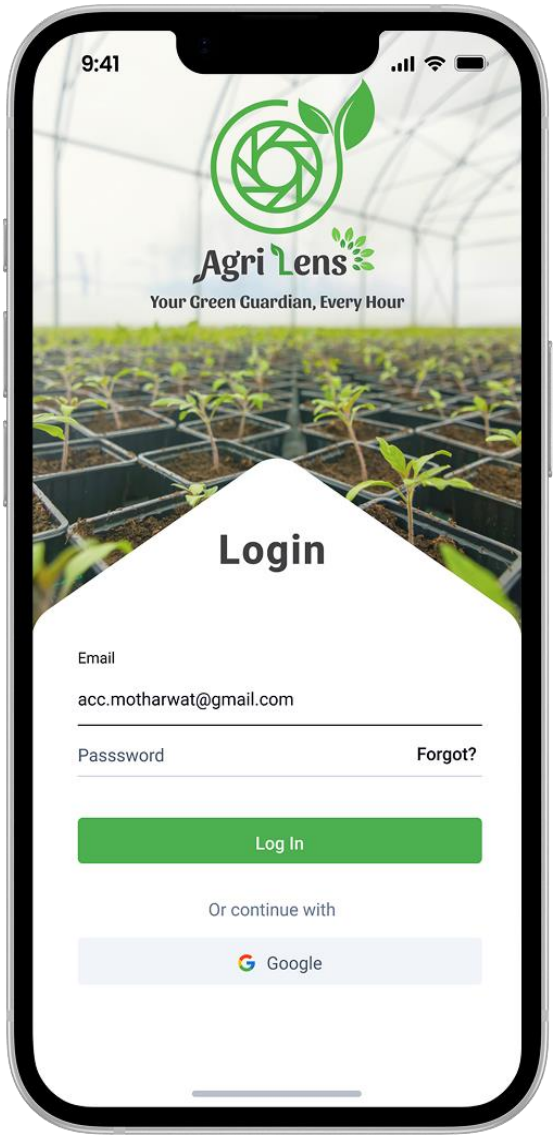


Figure 70 - Login Screen

- **Login Erorr :**

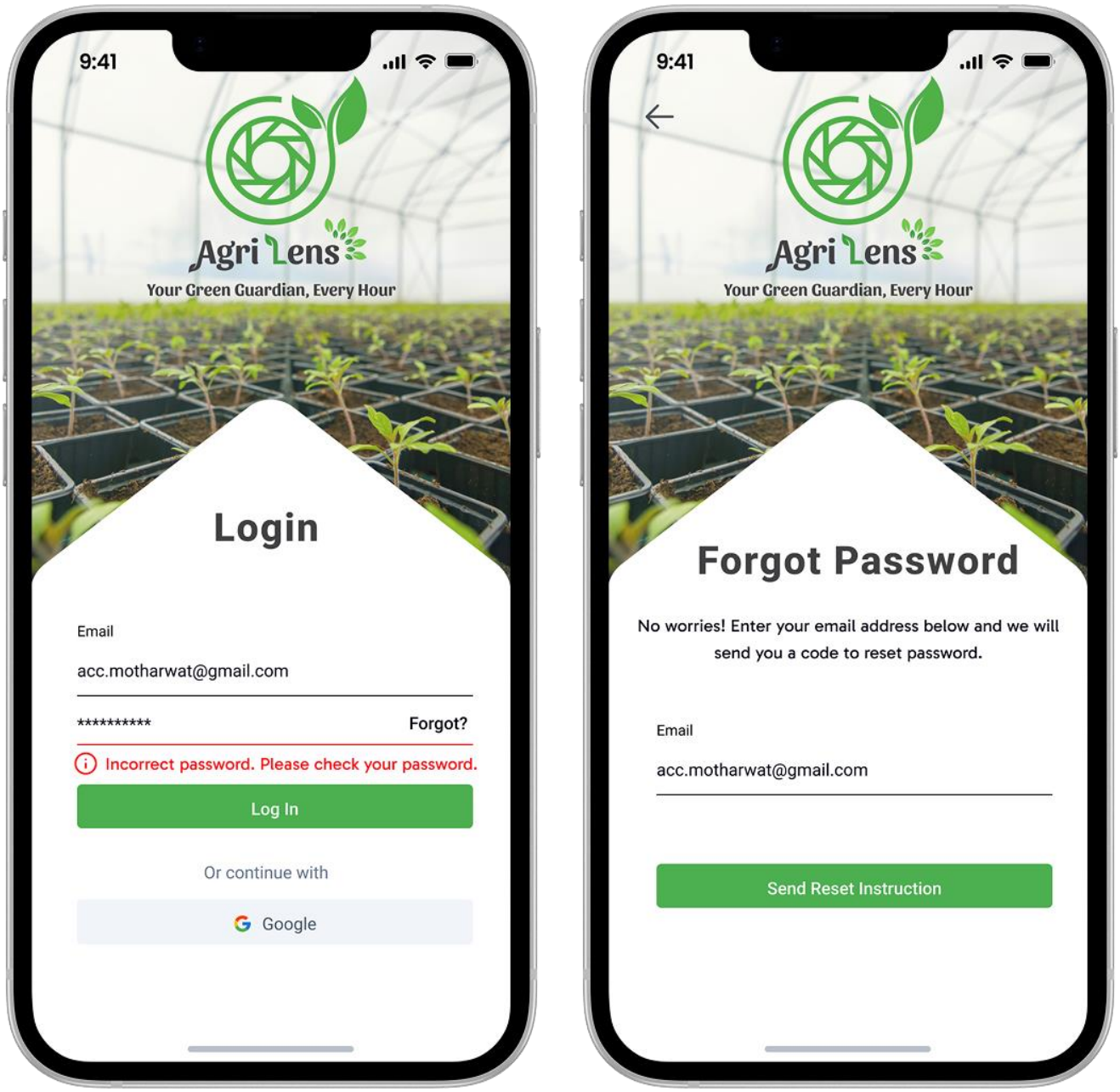


Figure 71 - Login Erorr

- **Verify Account & Reset Passward:**

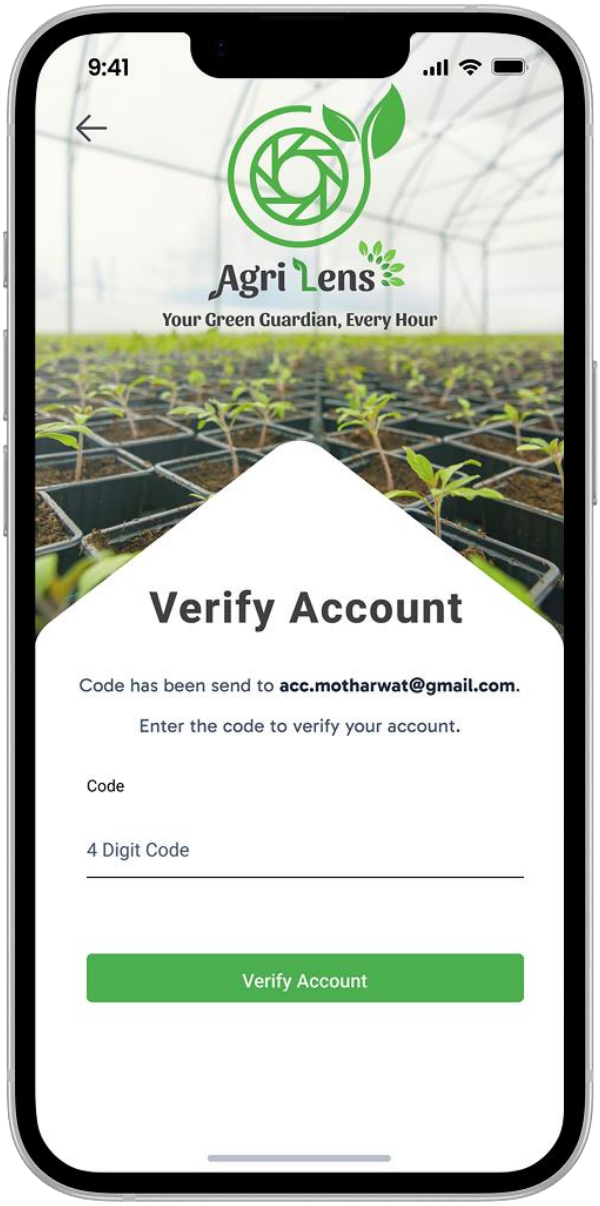


Figure 73 - Verify Account

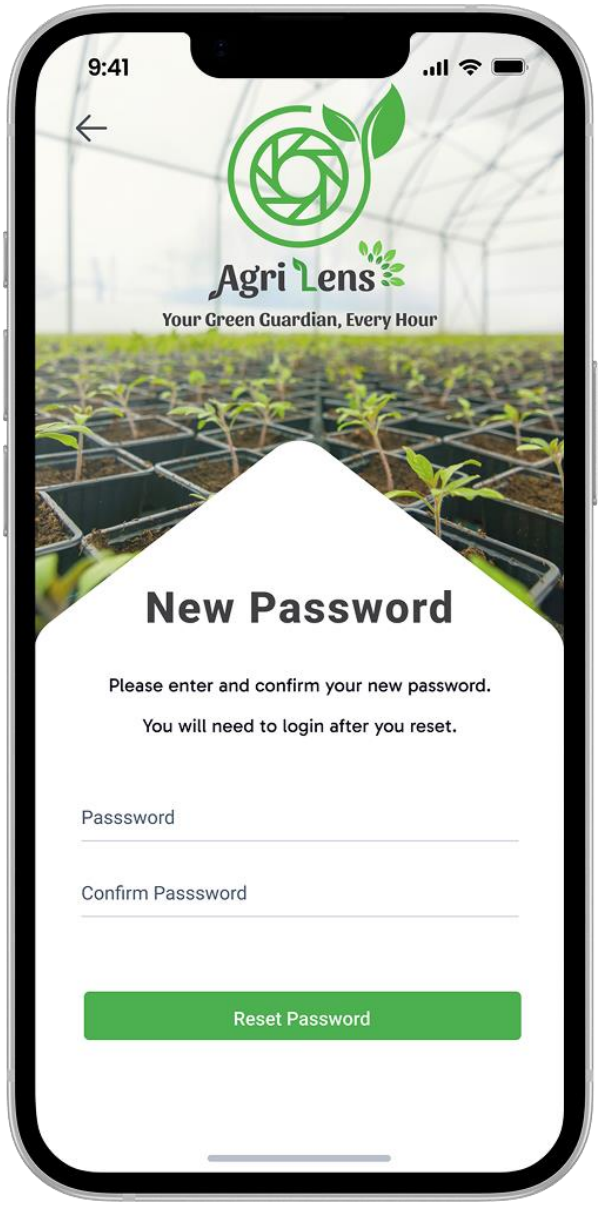


Figure 72 - Reset Passward

- **Home Screen:**

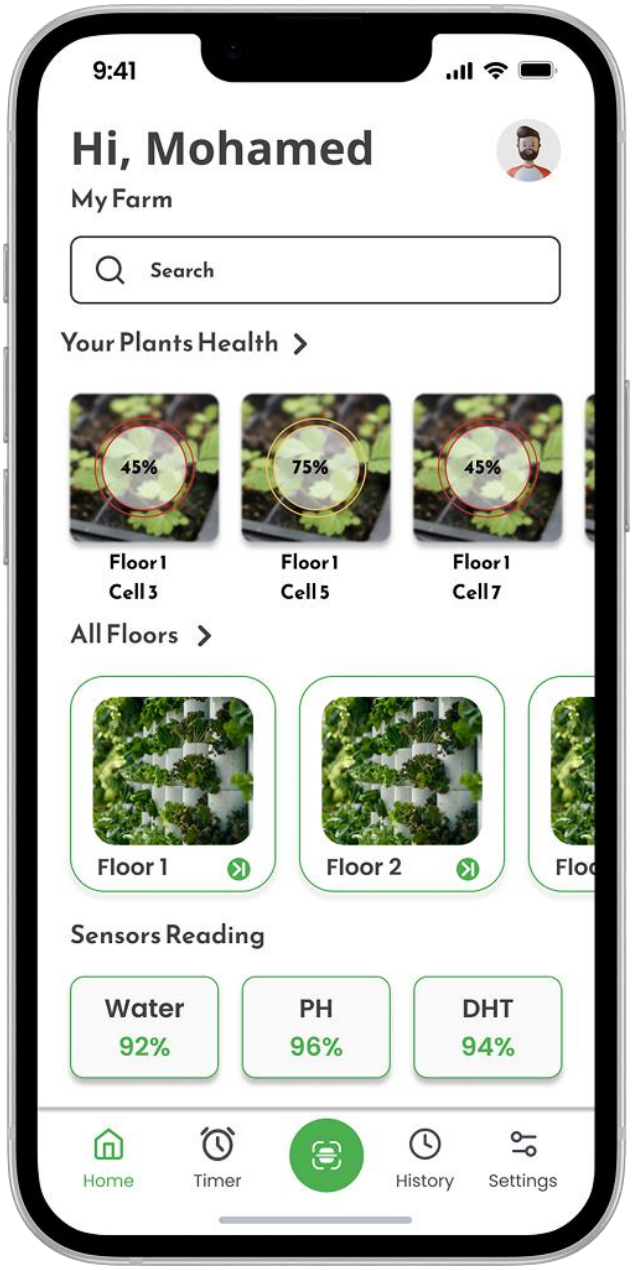


Figure 74 - Home Screen

- **All Plants:**

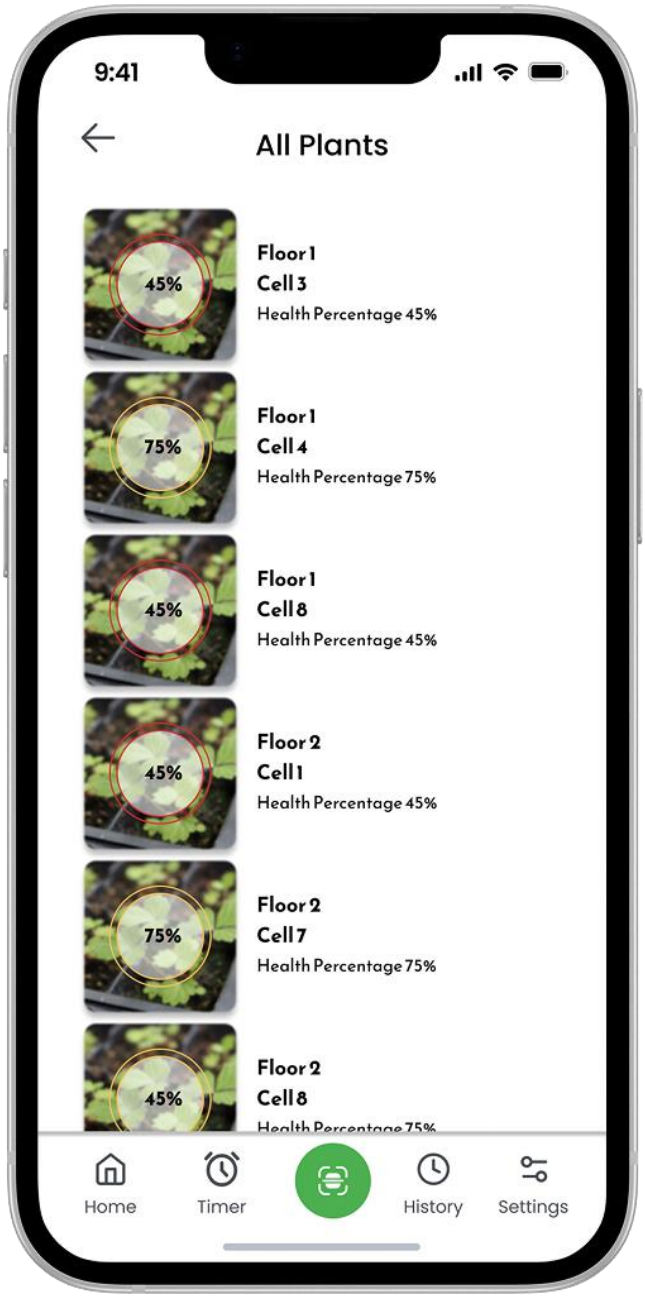


Figure 75 - All Plants

- **Plant Details:**

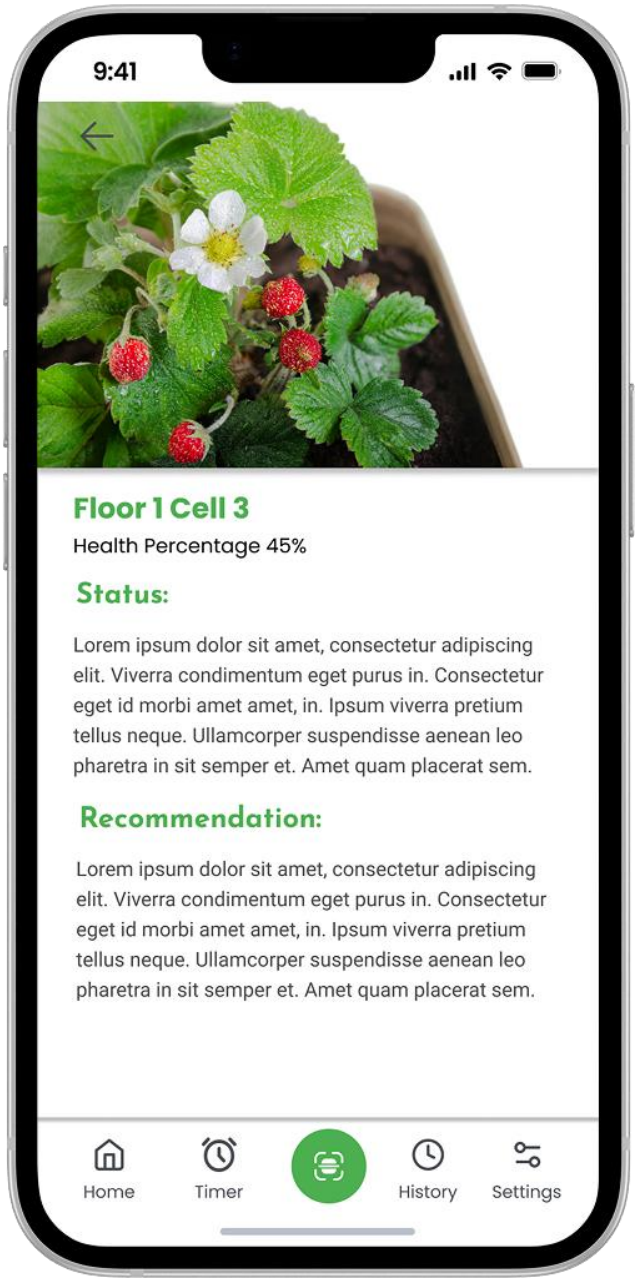


Figure 76 - Plant Details

- **Set Time:**

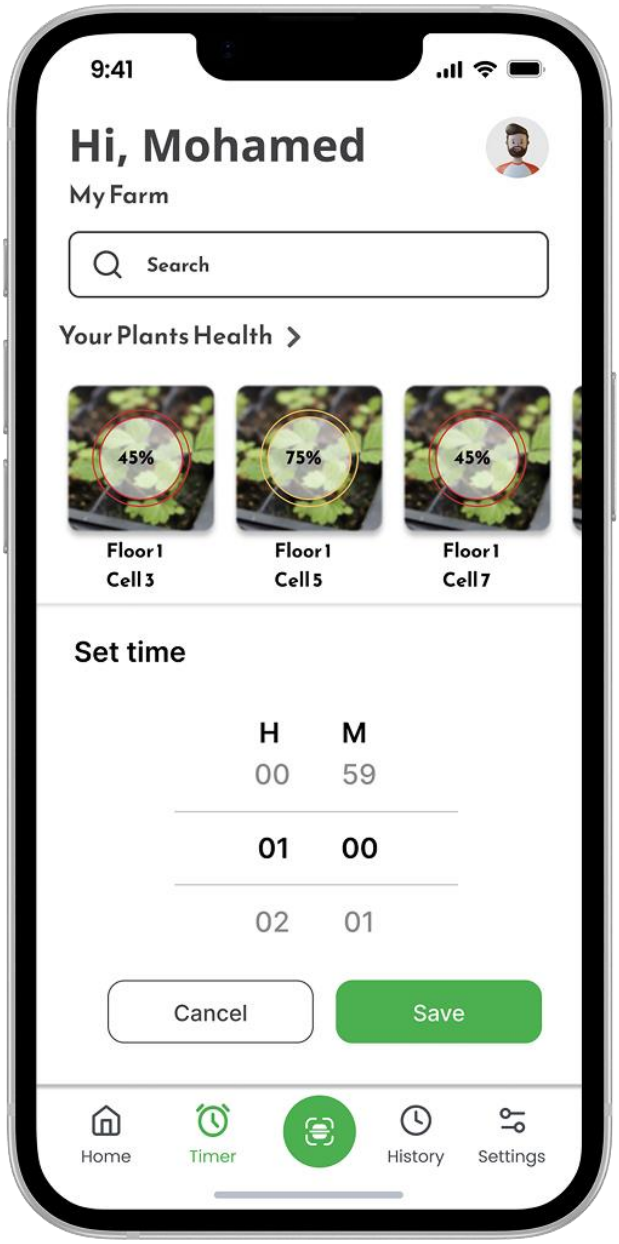


Figure 77 - Set Time

- **Scan:**

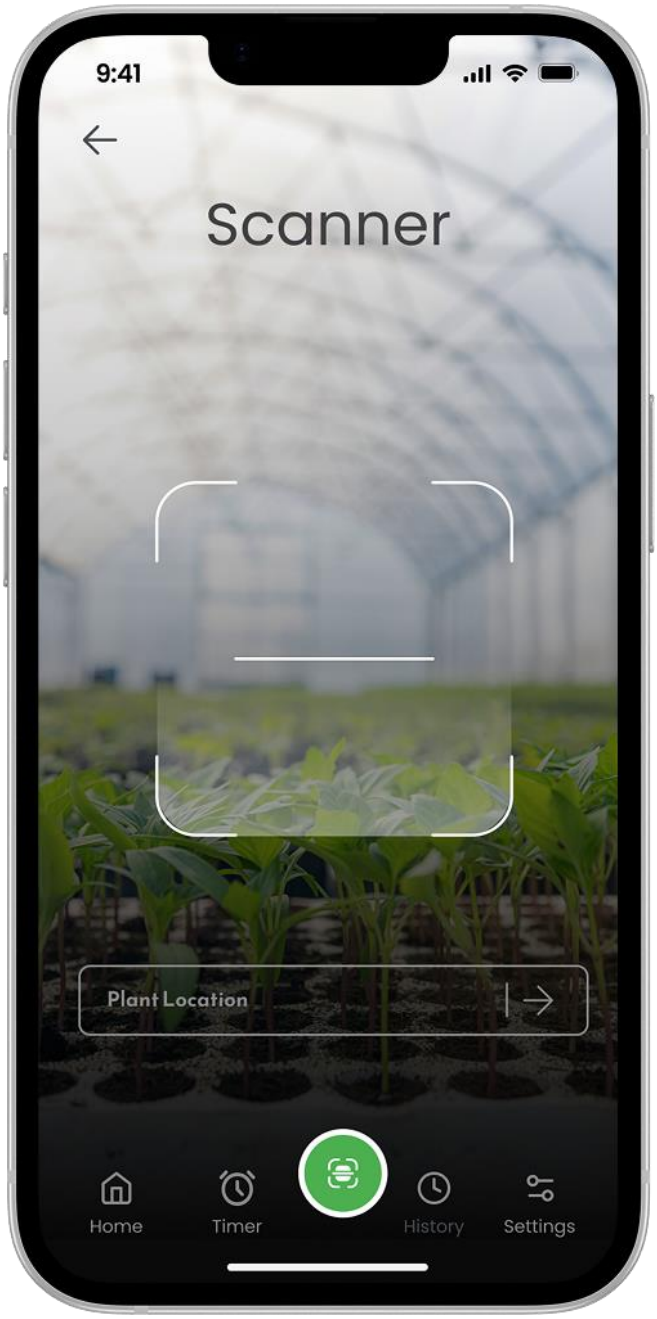


Figure 78 - Scan

- **Scan Load:**

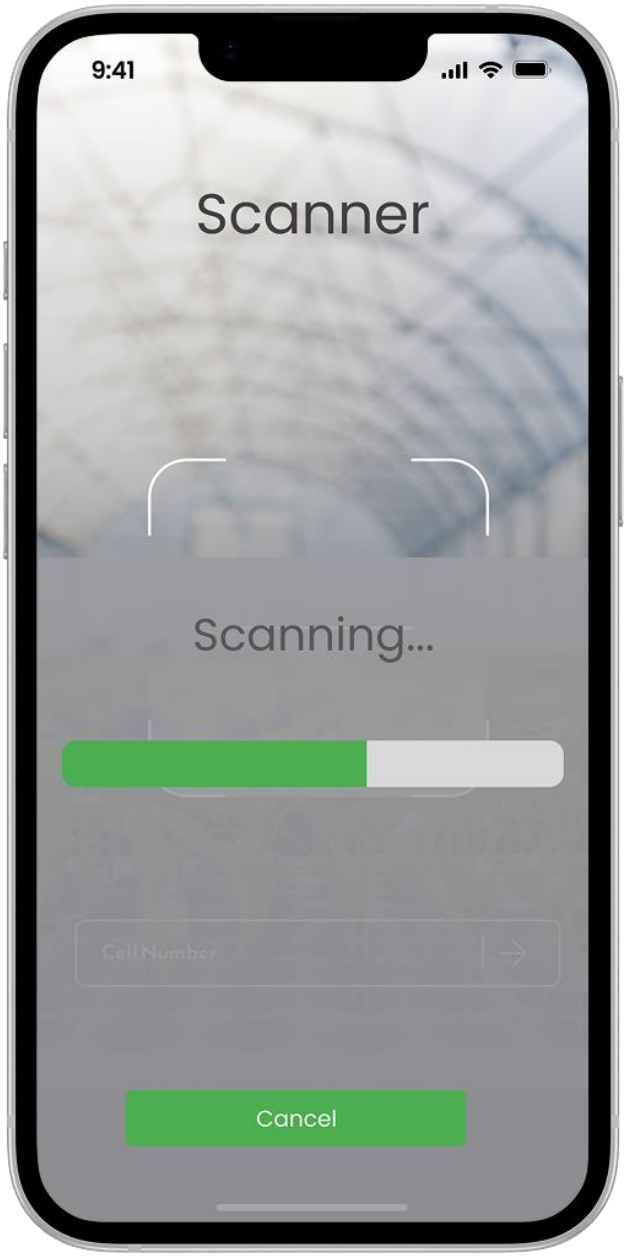


Figure 79 - Scan Load

- **After Scan:**

“1” **“2”**

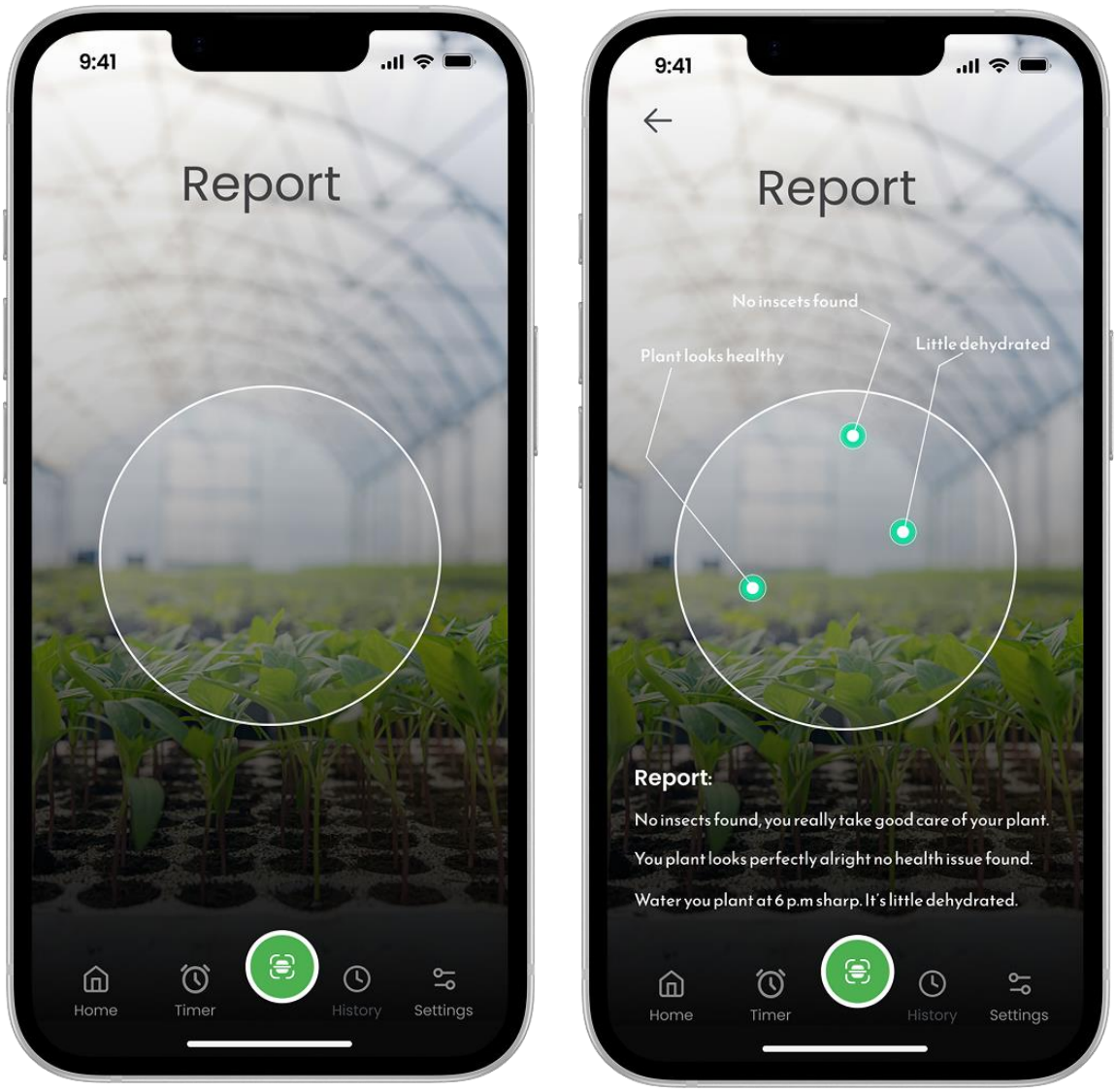


Figure 80 - After Scan

- **Report Details:**

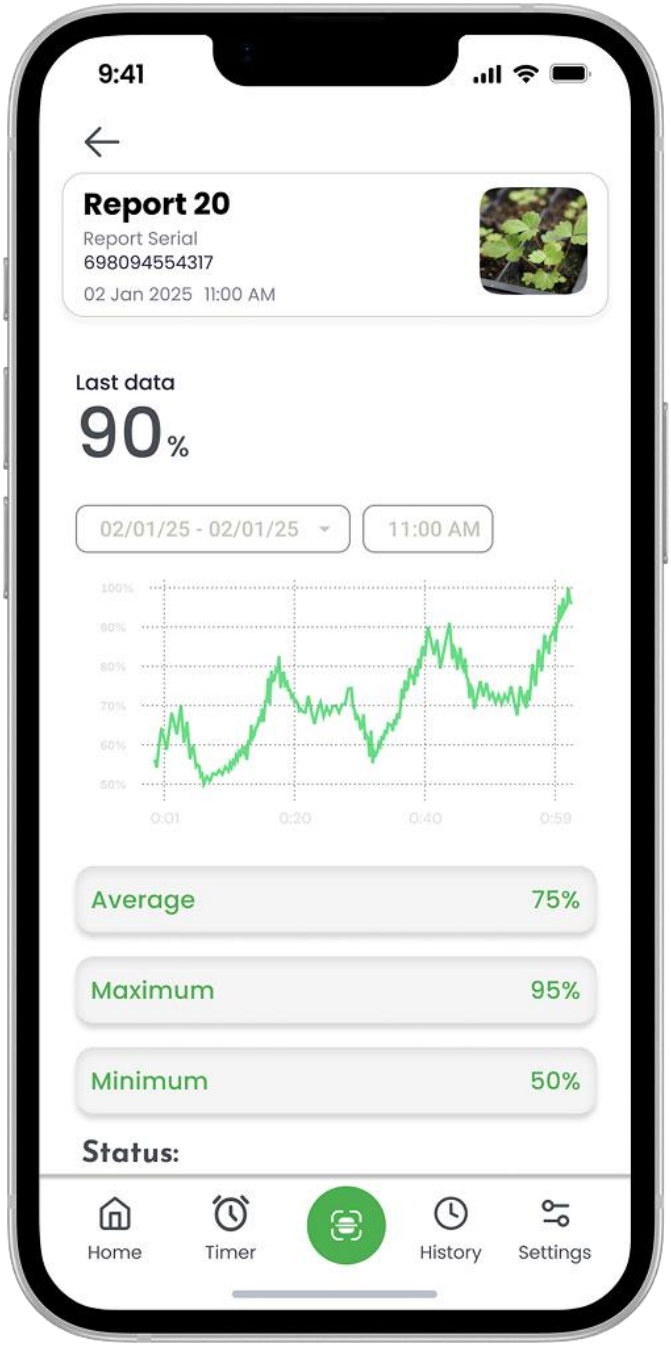


Figure 81 - Report Details

- **History:**

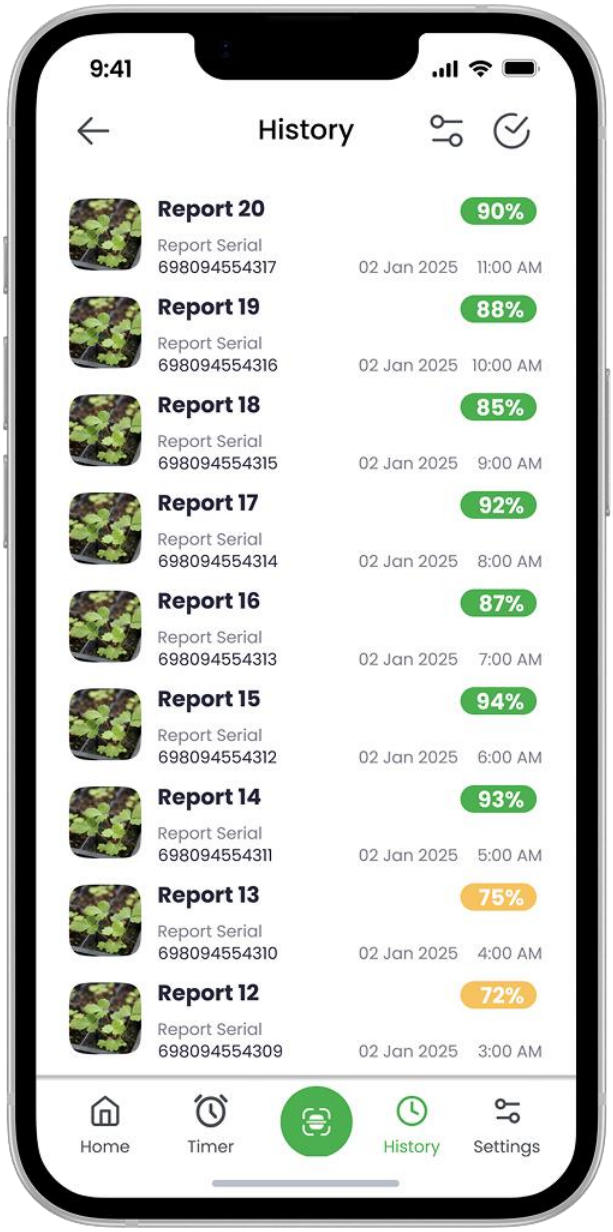


Figure 82 - History

- **Search & Fillter:**

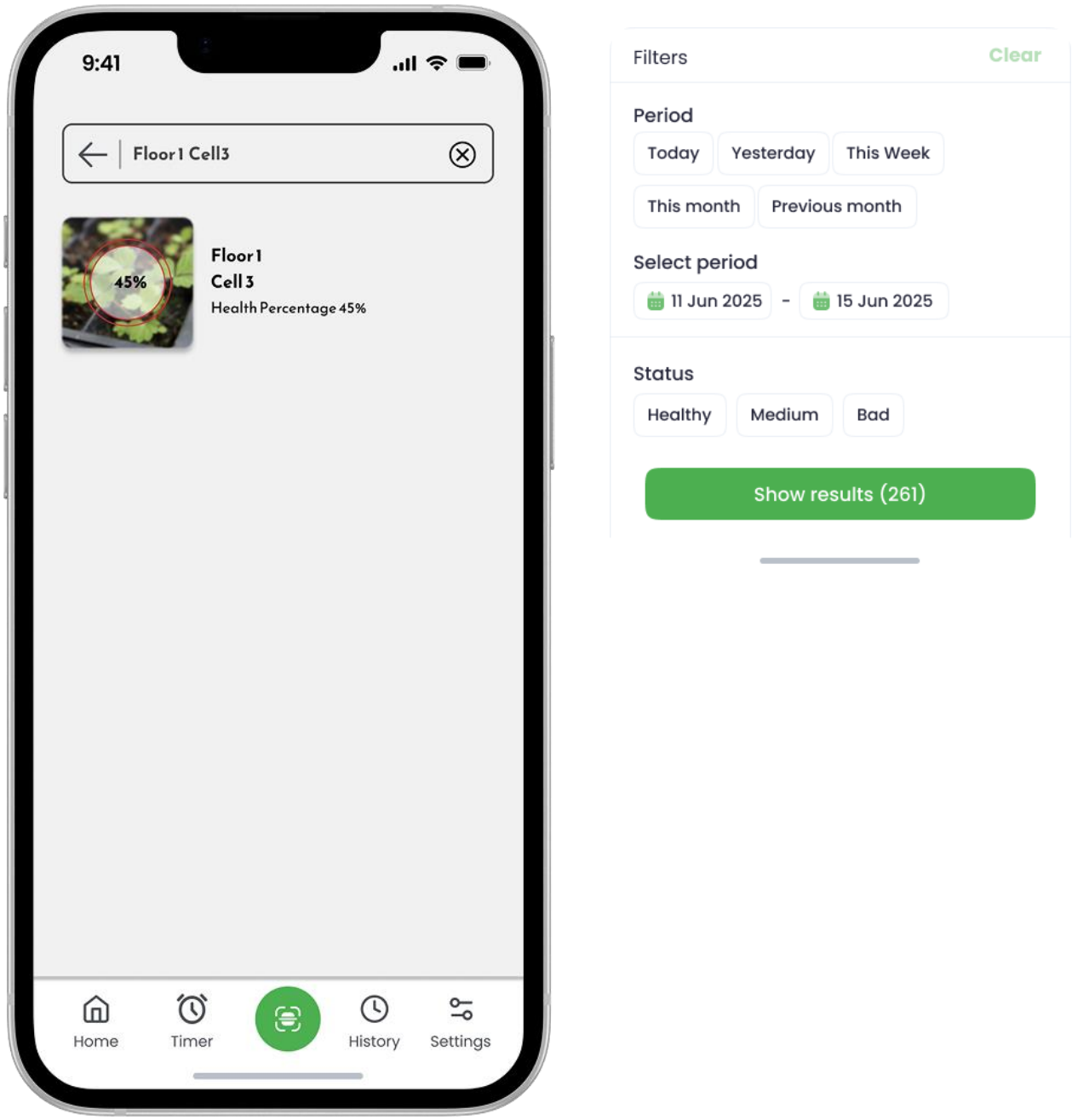


Figure 83 - Search & Fillter

- **Setting:**

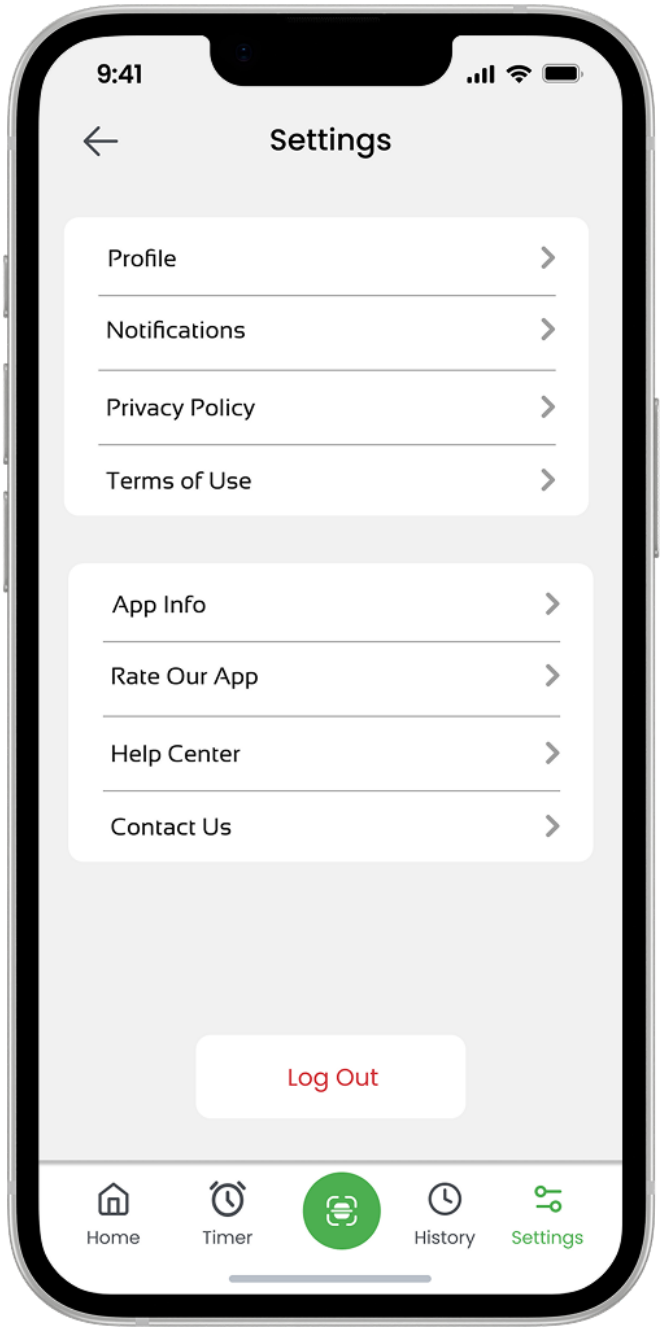


Figure 84 - Setting

- **Profile:**

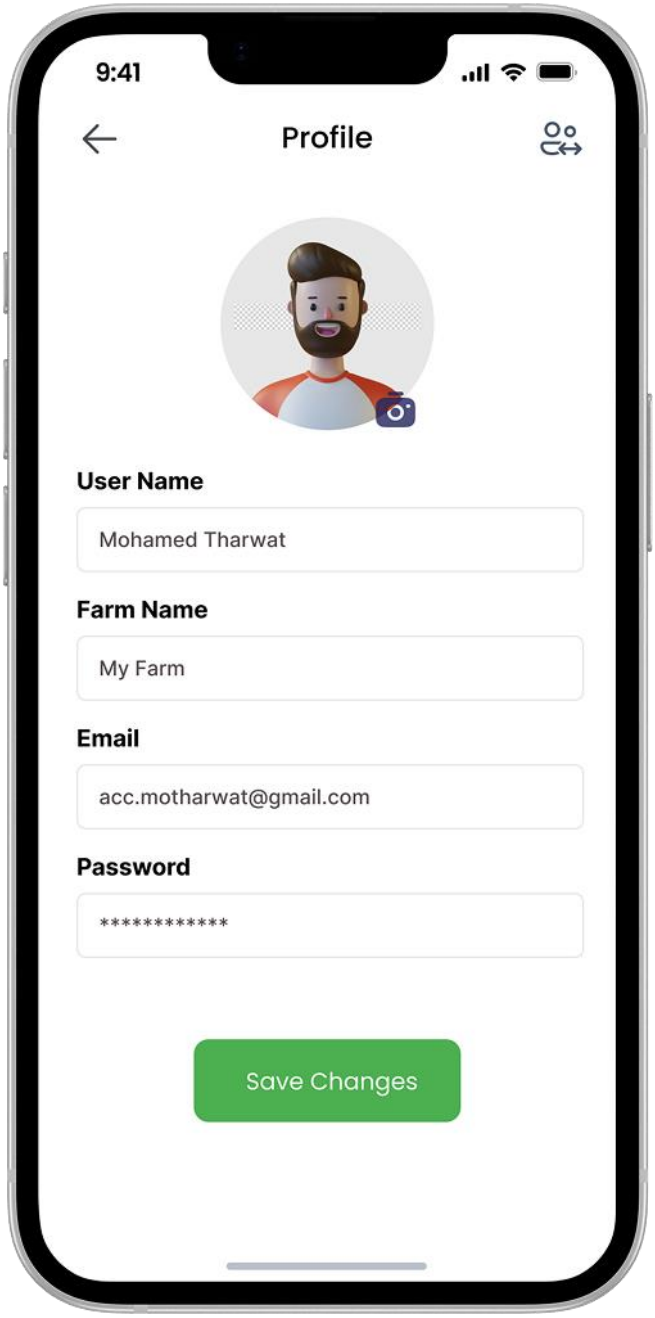


Figure 85 - Profile

- **Notification:**

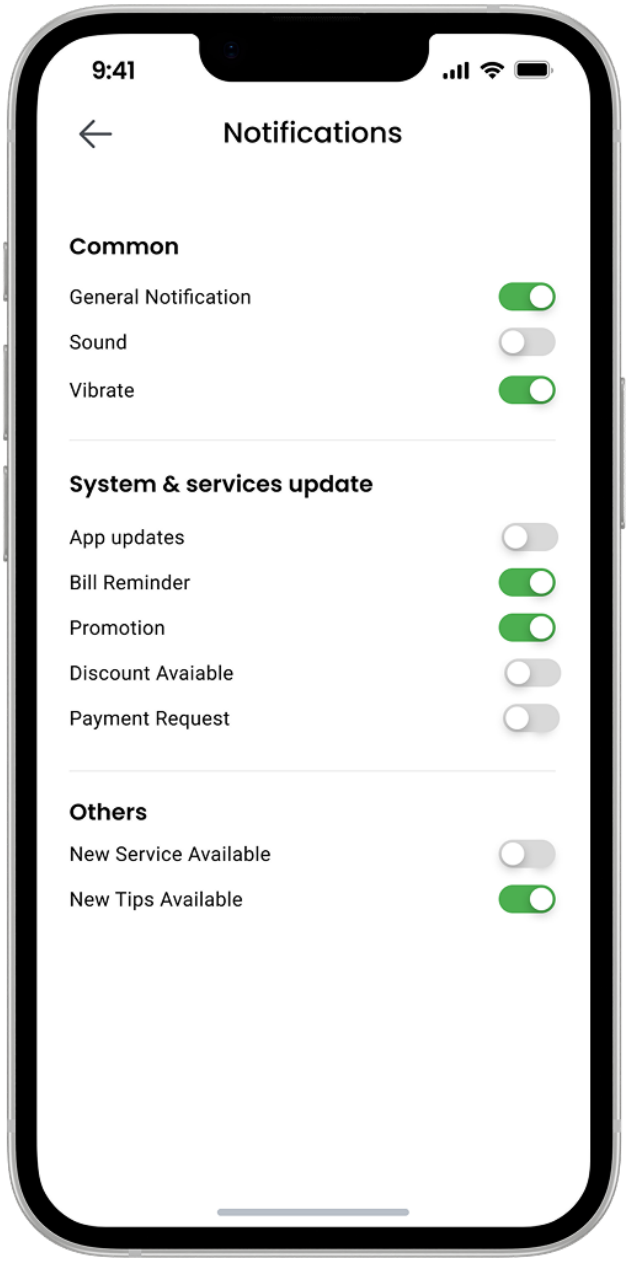


Figure 86 - Notification

- **Prinacy Policy & Terms of Use:**

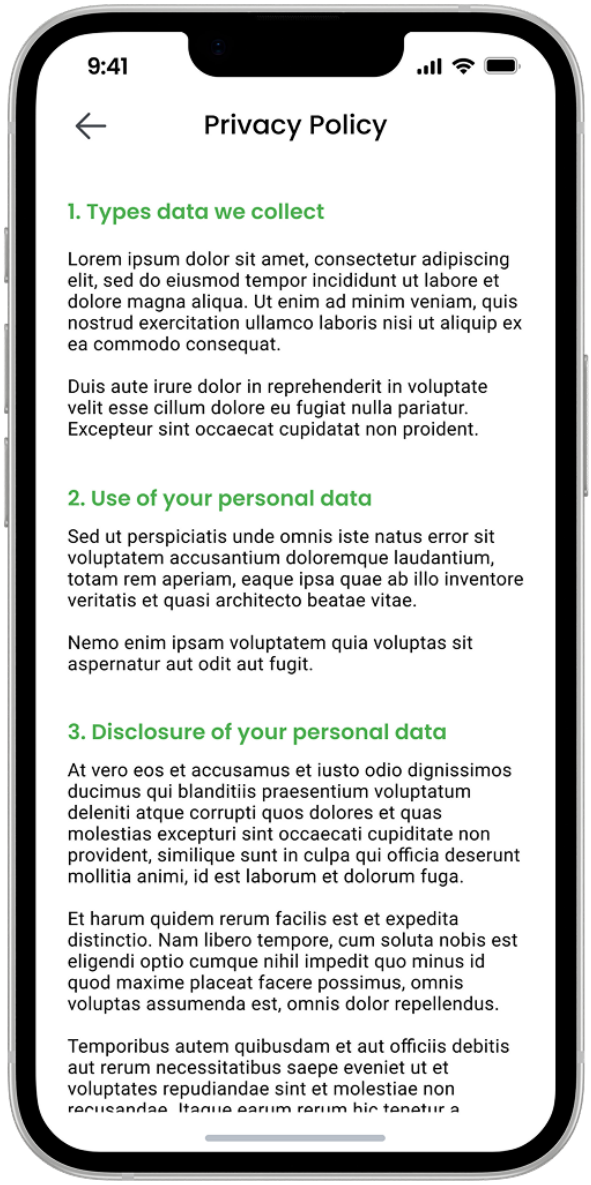


Figure 87 - Prinacy Policy

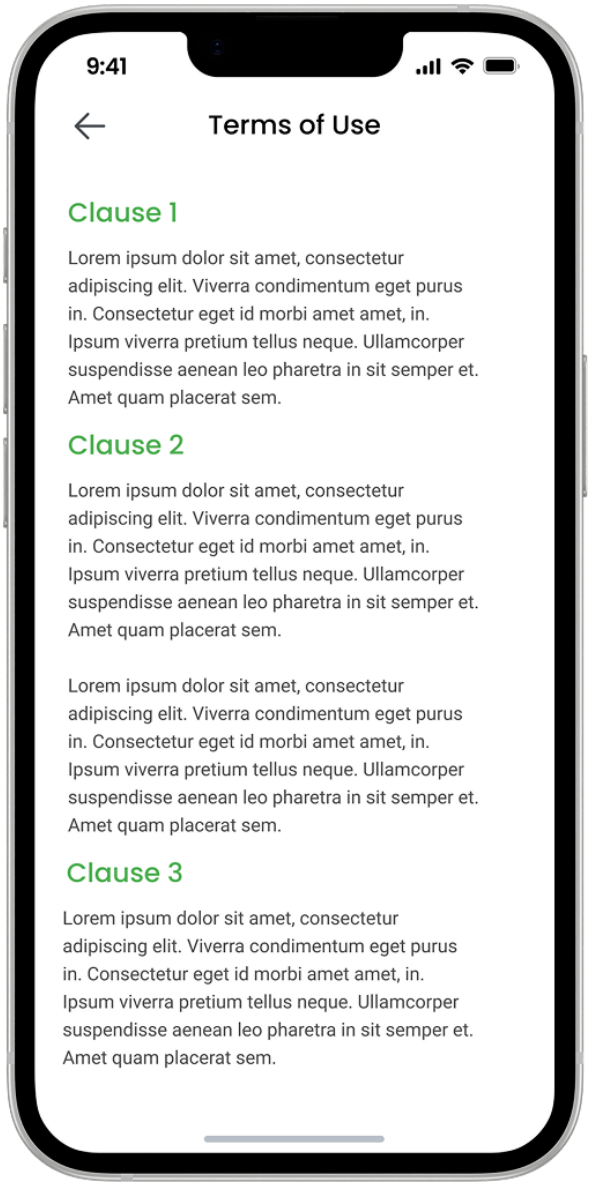


Figure 88 - Terms of Use

- **Contact Us & Help Center:**

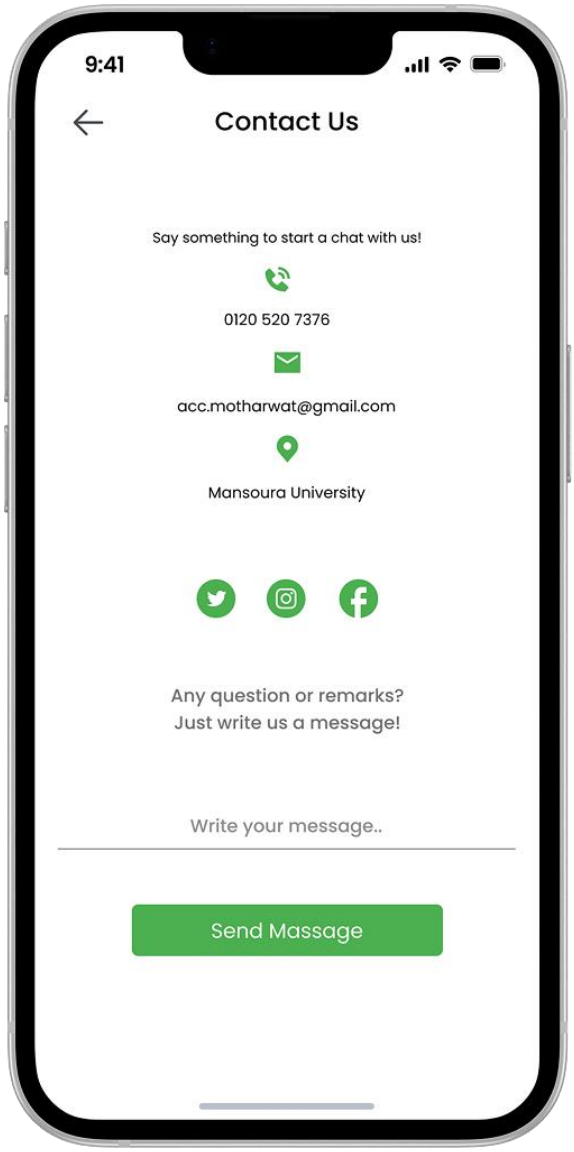


Figure 89 - Contact Us

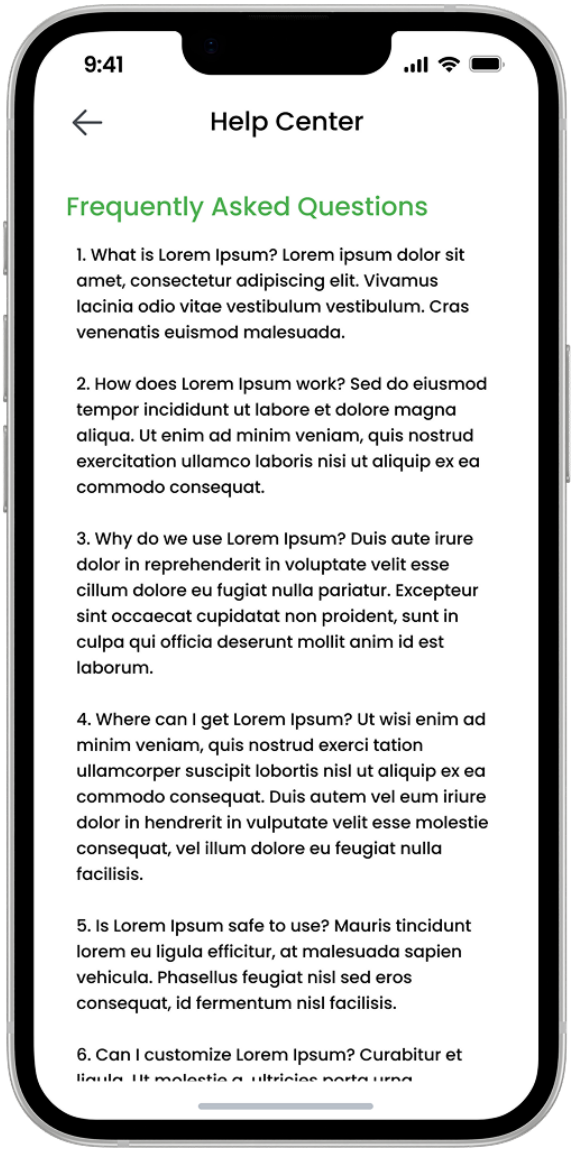


Figure 90 - Help Center

4.2.2 User Interface Design of Web Application:

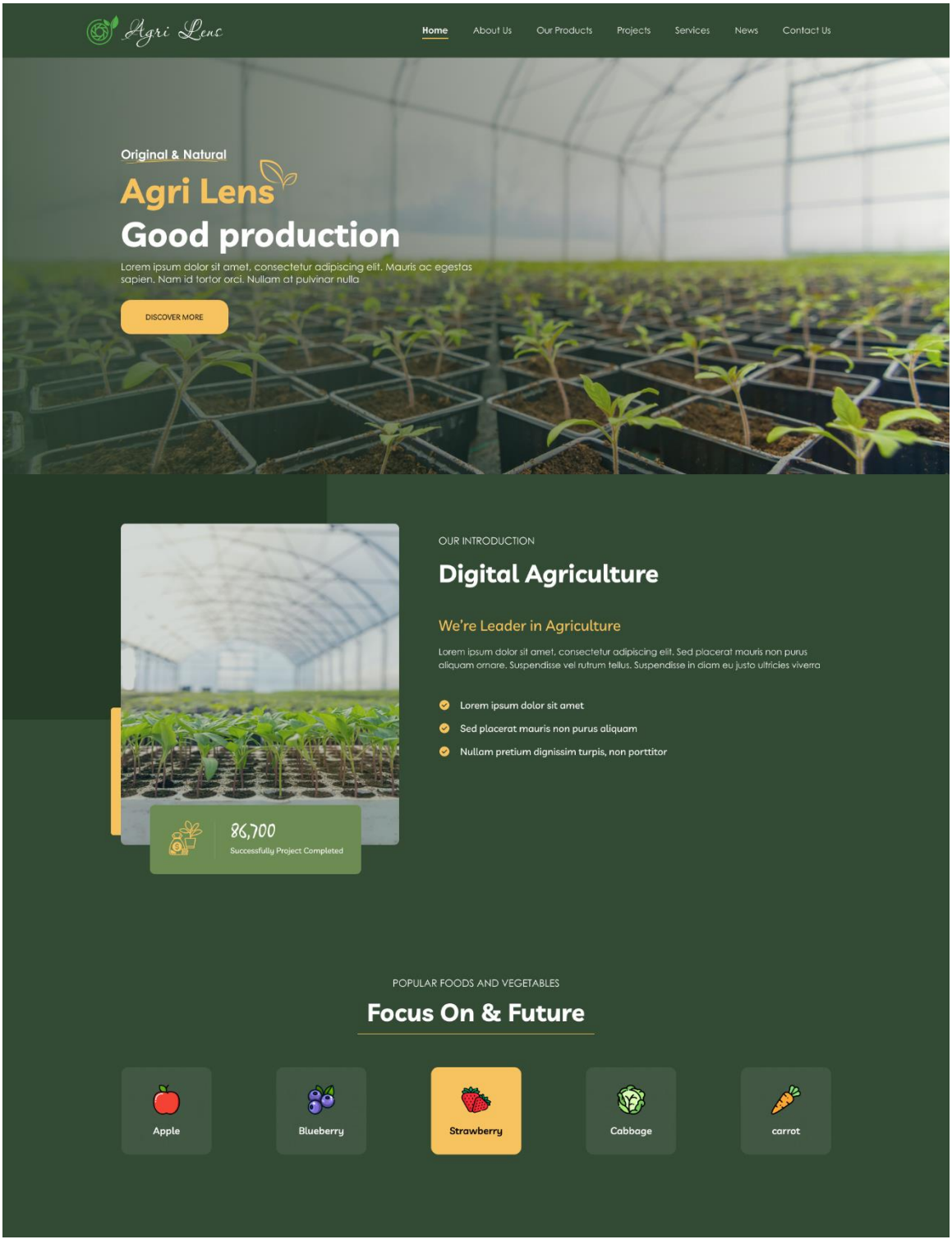
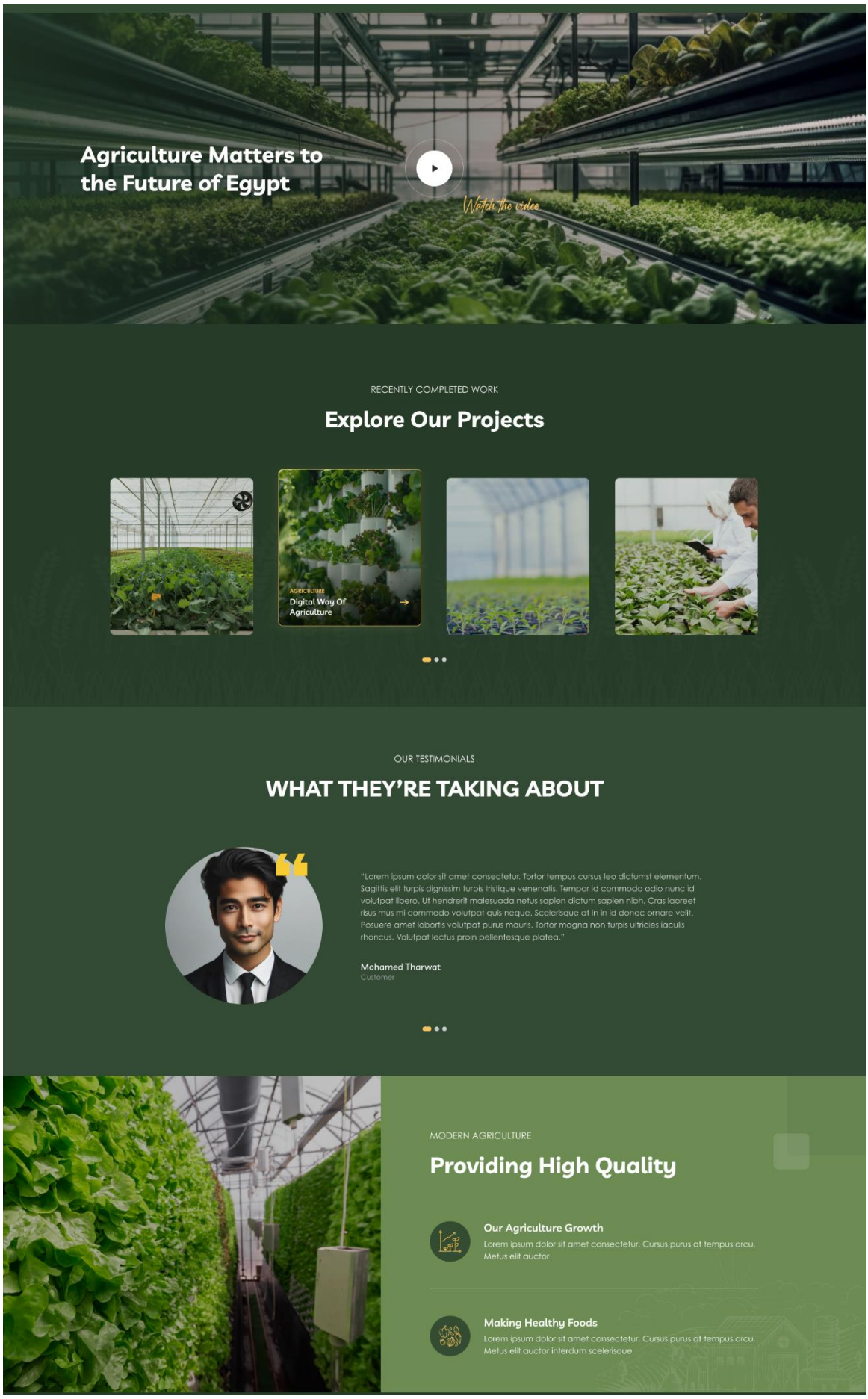
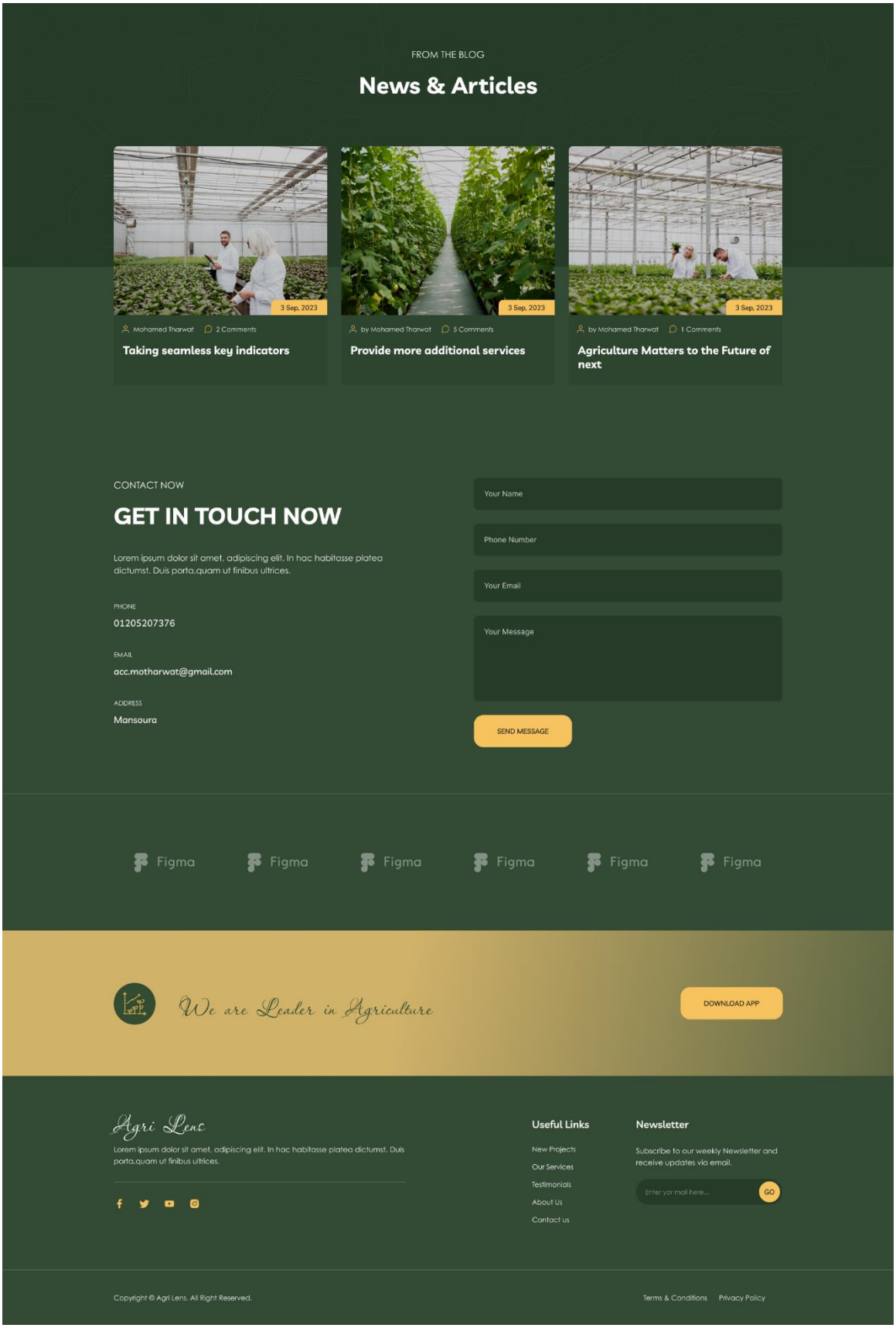
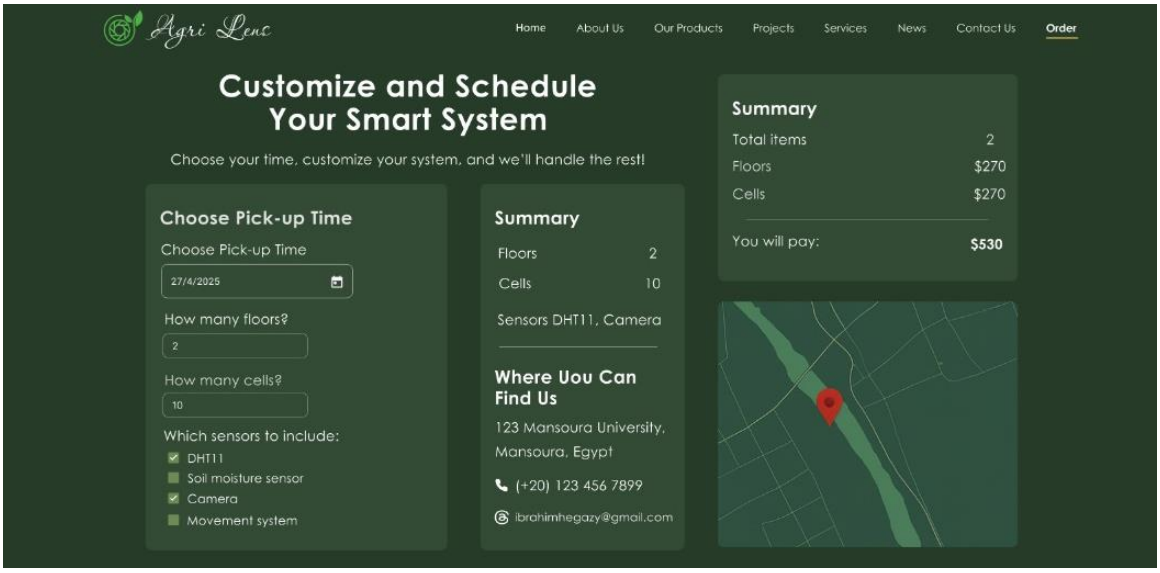


Figure 91 - User Interface Design of Web Application


Agriculture Matters to the Future of Egypt
Watch the video
RECENTLY COMPLETED WORK
Explore Our Projects
AGRICULTURE
Digital Way Of Agriculture
OUR TESTIMONIALS
WHAT THEY'RE TAKING ABOUT
"Lorem ipsum dolor sit amet consectetur. Tortor tempus cursus leo dictumst elementum. Sagittis elit turpis dignissim turpis tristique venenatis. Tempor id commodo odio nunc id volutpat libero. Ut hendrerit malesuada netus sapien dictum sapien nibh. Cras laoreet risus mus mi commodo volutpat quis neque. Scelerisque at in in id donec ornare velit. Posuere amet lobortis volutpat purus mauris. Tortor magna non turpis ultricies iaculis rhoncus. Volutpat lectus proin pellentesque platea."
Mohamed Tharwat
Customer
MODERN AGRICULTURE
Providing High Quality
Our Agriculture Growth
Lorem ipsum dolor sit amet consectetur. Cursus purus at tempus arcu. Metus elit auctor
Making Healthy Foods
Lorem ipsum dolor sit amet consectetur. Cursus purus at tempus arcu. Metus elit auctor interdum scelerisque


FROM THE BLOG
News & Articles
3 Sep, 2023
Mohamed Tharwat
2 Comments
Taking seamless key indicators
3 Sep, 2023
by Mohamed Tharwat
5 Comments
Provide more additional services
3 Sep, 2023
by Mohamed Tharwat
1 Comments
Agriculture Matters to the Future of next
CONTACT NOW
GET IN TOUCH NOW
Lorem ipsum dolor sit amet, adipiscing elit. In hac habitasse platea dictumst. Duis porta,quam ut finibus ultrices.
PHONE
01205207376
EMAIL
acc.motharwat@gmail.com
ADDRESS
Mansoura
Your Name
Phone Number
Your Email
Your Message
SEND MESSAGE
Figma
Figma
Figma
Figma
Figma
Figma
We are Leader in Agriculture
DOWNLOAD APP
Agri Lens
Lorem ipsum dolor sit amet, adipiscing elit. In hac habitasse platea dictumst. Duis porta,quam ut finibus ultrices.
Useful Links
New Projects
Our Services
Testimonials
About Us
Contact us
Newsletter
Subscribe to our weekly Newsletter and receive updates via email.
Enter yor mail here...
GO
Copyright © Agri Lens. All Right Reserved.
Terms & Conditions
Privacy Policy


Agri Lens
Home
About Us
Our Products
Projects
Services
News
Contact Us
Order
Customize and Schedule Your Smart System
Choose your time, customize your system, and we'll handle the rest!
Choose Pick-up Time
Choose Pick-up Time
27/4/2025
How many floors?
2
How many cells?
10
Which sensors to include:
DHT11
Soil moisture sensor
Camera
Movement system
Summary
Floors 2
Cells 10
Sensors DHT11, Camera
Where Uou Can Find Us
123 Mansoura University, Mansoura, Egypt
(+20) 123 456 7899
ibrahimhegazy@gmail.com
Summary
Total items 2
Floors $270
Cells $270
You will pay: $530

Chapter 5
HardWare Implementation

5.1 Hardware Component Description:

This section details the essential hardware components utilized in our smart farming project, highlighting their role in data acquisition, actuation, and overall system functionality. The integration of these components forms the physical layer of our IoT-enabled solution, facilitating precise environmental control and monitoring for optimal crop health.

5.1.1 Microcontrollers and Connectivity

- **ESP32 Microcontroller:**

  The core processing unit of our system, the ESP32 serves as the central hub for sensor data acquisition, control of actuators, and communication with the backend. Its integrated Wi-Fi and Bluetooth capabilities are leveraged for seamless data transmission and remote control, enabling real-time monitoring and management of the farming environment.

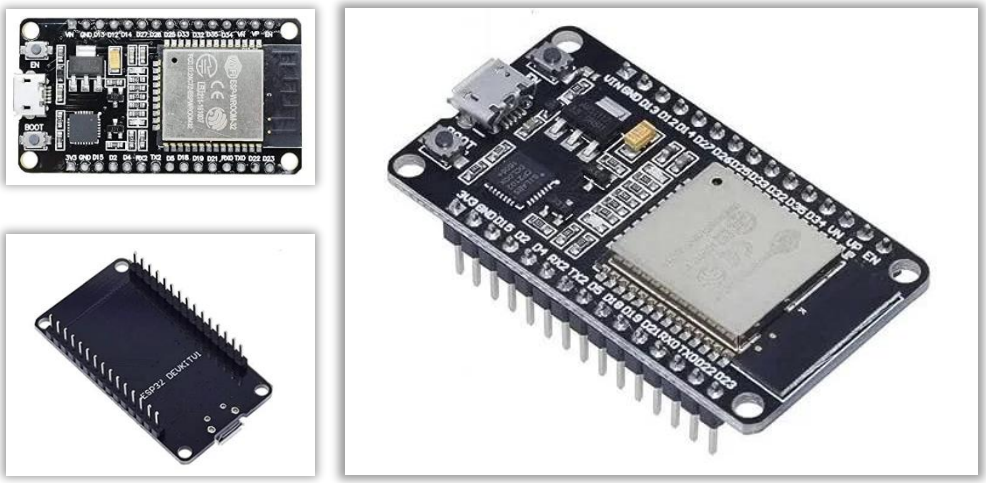


Figure 92 - ESP32 Microcontroller

- **ESP32-CAM Module:**

  This module integrates an ESP32 microcontroller with an OV2640 camera, providing crucial imaging capabilities for

Figure 93 - ESP32-CAM Module

our project. It is responsible for capturing high-resolution images of the strawberry plants, which are then transmitted to the backend for AI-driven disease detection using the YOLO V11 instance segmentation model.

- **FTDI (BLANCGROUP USB Adapter Board (FT232RL)):**

The FTDI adapter board serves as a crucial bridge for programming and communicating with the ESP32 and ESP32-CAM modules. It provides a reliable USB-to-serial interface, essential for uploading firmware, debugging, and initial configuration of the microcontrollers.

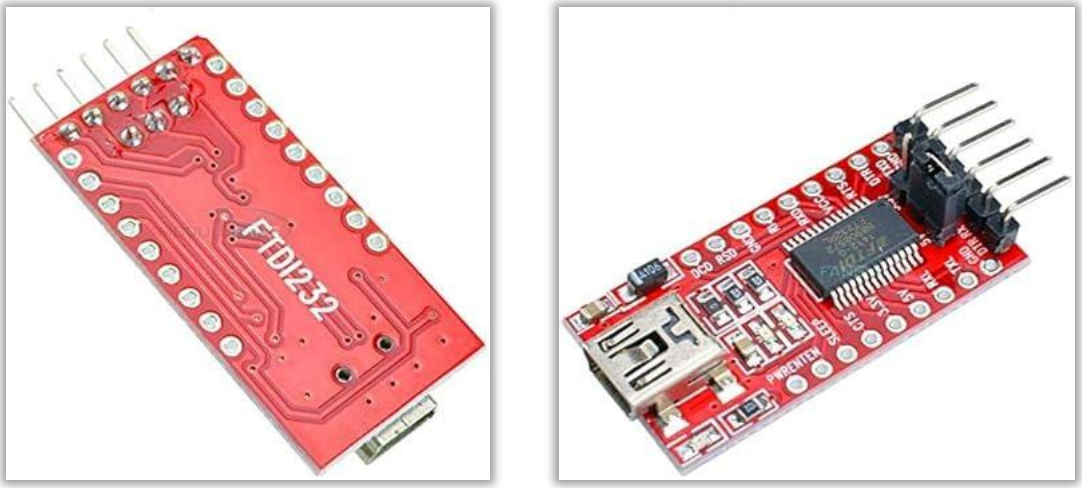


Figure 94 - FTDI (BLANCGROUP USB Adapter Board (FT232RL))

5.1.2 Sensing and Environmental Monitoring

- **DHT11 Temperature and Humidity Sensor:**

The DHT11 sensor provides essential environmental data by accurately measuring both air temperature and humidity. This data is critical for maintaining optimal growing conditions for strawberry plants and is fed to the backend for analysis and decision-

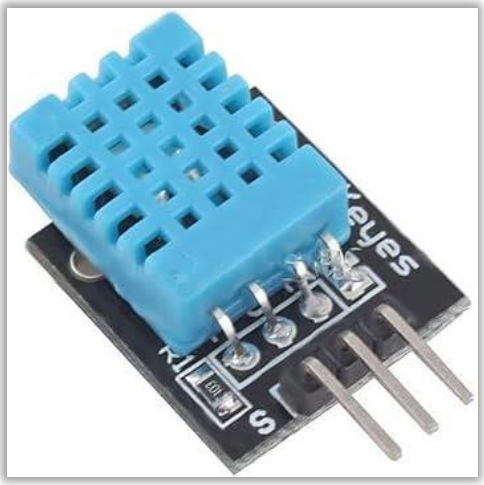


Figure 95 - DHT11 Temperature and Humidity Sensor

making regarding irrigation and ventilation.

- **Soil Moisture Sensors with drivers | 3 pieces:**

(Assumed based on project context, though not explicitly listed with model numbers) These sensors are crucial for monitoring the moisture levels within the soil. Coupled with their dedicated drivers, they provide precise data to the ESP32, enabling the system to determine the exact water requirements of the plan accordingly, optimizing water usage and preventing over- or under-watering.

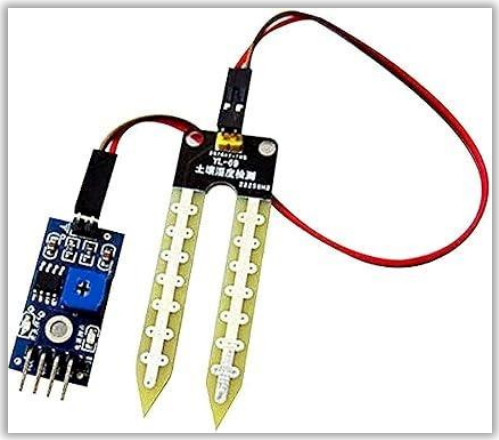


Figure 96 - Soil Moisture Sensors with drivers

5.1.3 Actuation and Mechanical Systems

- **NEMA 17 Stepper Stepping Motor (2-Phase, 1.8 Degree, 0.9A, 0.4N.M, 42mm):**

This high-precision stepper motor is a key component for implementing controlled linear or rotational movement within the farming system. Its precise stepping capability (1.8 degrees per step) allows for accurate positioning in applications such as automated nutrient dispensing or sensor array movement.

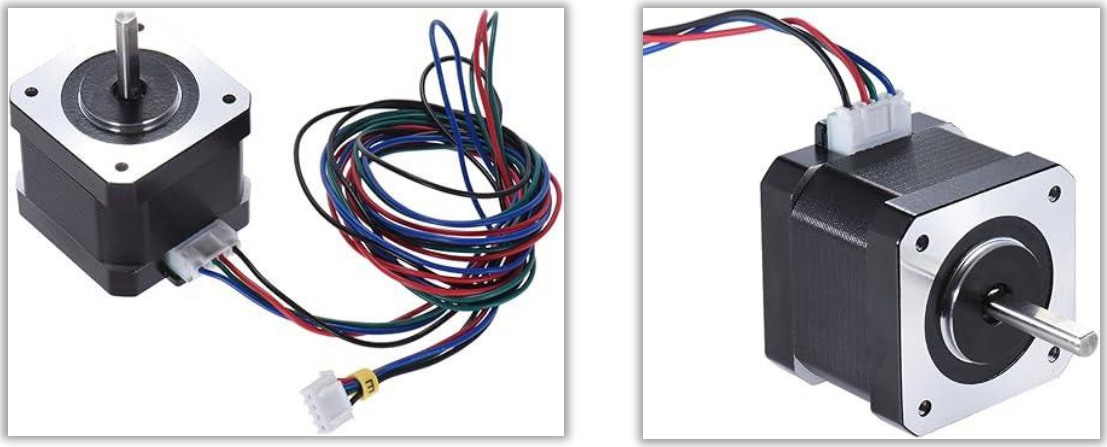


Figure 97 - NEMA 17 Stepper Stepping Motor

- **A4988 Stepper Motor Driver (2A):**

  The A4988 driver module is responsible for controlling the NEMA 17 stepper motor. It translates the digital signals from the ESP32 into the necessary current pulses to drive the stepper motor, enabling precise control over its speed, direction, and step resolution.

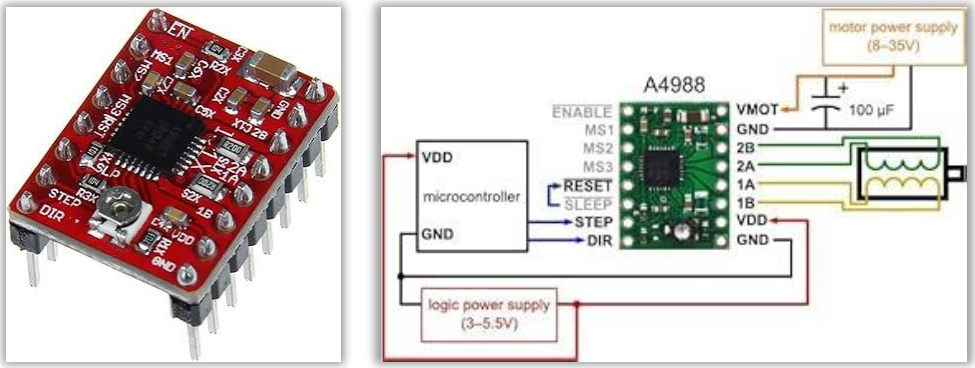


Figure 98 - A4988 Stepper Motor Driver

- **GT2 Timing Pulley (Bore 8mm, 20 Teeth) & GT2 20 Tooth Timing Belt Pulley (5mm Bore):**

  These pulleys, in conjunction with the timing belt, form a robust and precise power transmission system. They are essential for converting the rotational motion of the stepper motor into linear motion, commonly used in automated gantry systems for scanning or dispensing. The different bore sizes allow for integration

Figure 99 - GT2 Timing Pulley

- **GT2 2mm Pitch 6mm Timing Belt (5m):**

  This durable timing belt, with its fine 2mm pitch, ensures accurate and slip-free power transmission when coupled with the GT2 pulleys. Its 6mm width

Figure 100 - GT2 2mm Pitch 6mm Timing Belt

provides sufficient strength for various linear motion applications within the project.

- **Water Pumps | 3 pieces:**

These pumps are the primary actuators for delivering water to the strawberry plants. Their quantity allows for zoned irrigation or the efficient delivery of water to multiple plant areas, ensuring even distribution of moisture.

Figure 101 - Water Pumps

- **Water Pump Drivers (L298 Motor Driver Module):**

These electronic modules are essential for controlling the operation of the water pumps. They act as an interface between the low-voltage control signals from the ESP32 and the higher power requirements of the pumps, enabling the system to turn the pumps on and off as needed for irrigation.

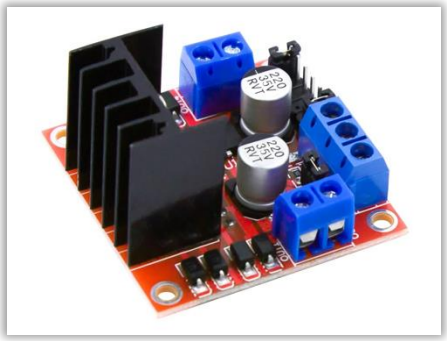


Figure 102 - Water Pump Drivers

5.1.4 Structural and Mechanical Framework

- **OpenBuilds V-Slot Gantry Plate (20mm):**

These plates are fundamental components for constructing sturdy and precise linear motion systems. Designed to work seamlessly with V-Slot linear rails, they provide mounting points for various components, including the

Figure 103 - Structural and Mechanical Framework

stepper motor and idler pulleys, facilitating smooth and accurate movement.

- **Idler Pulley Plate (Steel):**

The steel idler pulley plate provides a robust and stable mounting point for the GT2 idler pulleys. This ensures proper tensioning and alignment of the timing belt, preventing slippage and maintaining the accuracy of linear motion.

Figure 104 - Idler Pulley Plate

- **Plastic Wheel with Bearings (V-Slot Aluminium Profile Openbuilds Accessories) | 4 pieces:**

These wheels are designed to run smoothly within the V-Slot aluminium profiles, providing low-friction linear movement. Integrated bearings ensure minimal resistance and long-term durability for any moving carriages or platforms.

Figure 105 - Plastic Wheel with Bearings

- **V-Slot 20x40 Linear Rail (1000mm length):**

This 1-meter long aluminium extrusion forms the backbone of any linear motion system. Its unique V-groove profile allows for the integration of V-Slot gantry plates and wheels, creating a highly modular and customizable framework for automated movement, such as a camera or watering

Figure 106 - V-Slot 20x40 Linear Rail

5.1.5 Power Management

- **Power Supply 12V 5A 60W S-60-12:**

This power supply provides the primary electrical energy for the system, specifically catering to

Figure 107 - Power Supply

components like the water pumps and stepper motor, which typically require a higher voltage.

- **Power Converter to 5V (DC-DC Step Down Converter 3A Ultra Mini-360 (4.75V~23Vdc to 1V~17Vdc)):**

Figure 108 - Power Converter to 5V

This essential component steps down the 12V supply to a stable 5V, providing the necessary operating voltage for sensitive electronic components such as the ESP32 microcontroller, sensors, and other logic-level circuits. This ensures safe and reliable operatian of the low-voltage components.

5.2 Hardware Diagrams:

5.2.1 Diagram :

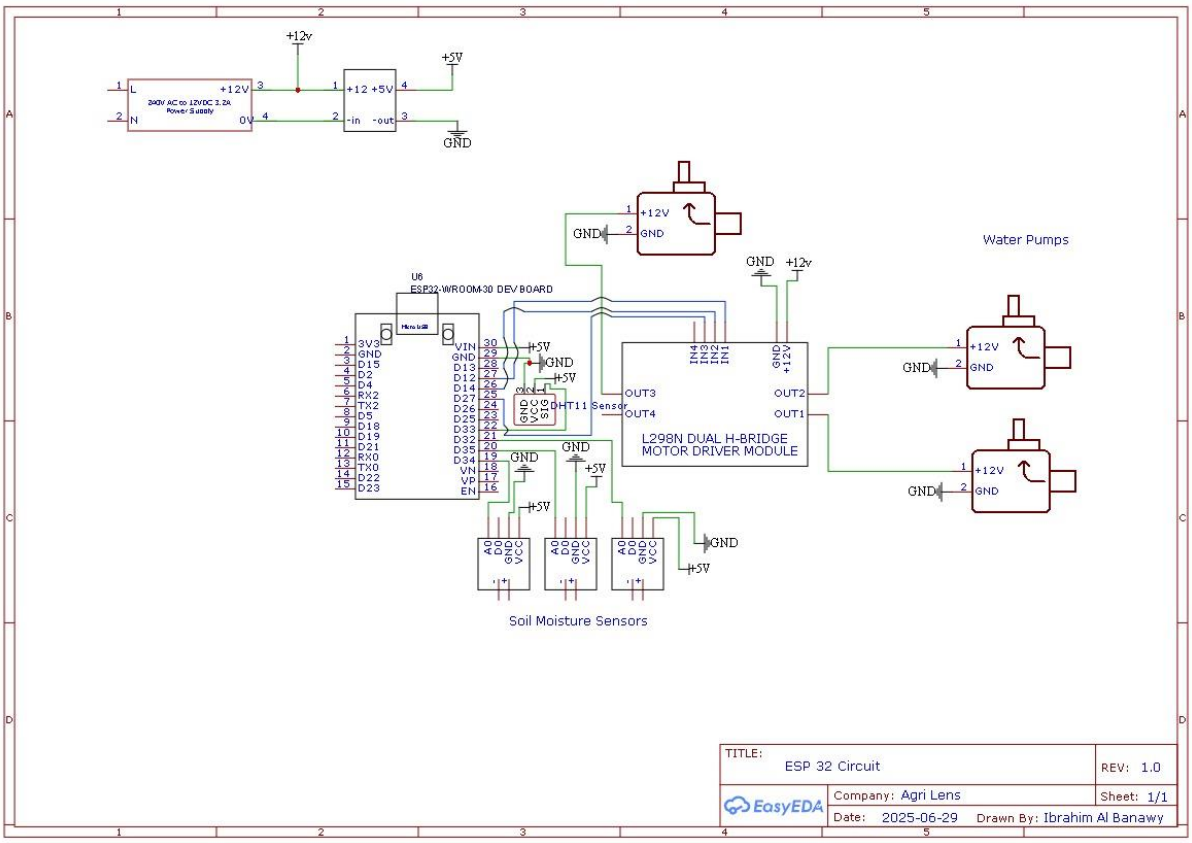


Figure 109 - Hardware Diagram 1

- **Power Supply:**
  - Similar to the previous schematic, a "240V AC to 12VDC 3.3A Power Supply" provides the main 12V DC power.
  - A power converter steps down the +12V to a regulated +5V.
  - Both +12V and +5V lines, along with a common ground (GND), are distributed throughout the circuit to power the various components.
- **ESP32 Development Board (U8 ESP32 WRoom32U DEV BOARD):**
  - This is the central control unit for this part of the system.
  - It receives its +5V power supply from the power converter via its 5V pin.
  - **DHT11 Temperature and Humidity Sensor:**
    - The DHT11 sensor is connected to **GPIO D27** of the ESP32.
    - It receives its +5V power supply from the ESP32's 5V pin.
    - Its GND pin is connected to the common ground.

- **Soil Moisture Sensors:**
    - Three soil moisture sensors are depicted. Each sensor has its output connected to an analog input pin on the ESP32. Specifically:
        - Sensor 1: Connected to **GPIO D34 (or VP)**.
        - Sensor 2: Connected to **GPIO D35 (or VN)**.
        - Sensor 3: Connected to **GPIO D32**.
    - Each soil moisture sensor receives its +5V power from the power converter and is connected to the common ground.
- **L298N Dual H-Bridge Motor Driver Module:**
    - Several GPIO pins from the ESP32 are connected to the L298N module for controlling the water pumps. Specifically:
        - **GPIO D26** is connected to **IN1** of the L298N.
        - **GPIO D25** is connected to **IN2** of the L298N.
        - **GPIO D33** is connected to **IN3** of the L298N.
        - **GPIO D2** is connected to **IN4** of the L298N.
    - The L298N module receives its main power (+12V) from the 12V power supply at its VCC pin.
    - Its GND pin is connected to the common ground.
    - The **EN (Enable)** pin of the L298N is connected to the +5V supply, indicating that the driver is continuously enabled.

- **L298N Dual H-Bridge Motor Driver Module:**
    - This module acts as an interface between the low-power control signals from the ESP32 and the higher power requirements of the water pumps.
    - It has four output channels (OUT1, OUT2, OUT3, OUT4) for driving motors.
    - **Water Pumps:**
        - Three water pumps are shown, connected to the L298N outputs.
        - **Pump 1:** Connected to **OUT4** and **OUT3** of the L298N. This suggests it might be connected to one full H-bridge, implying bi-directional control, though for a simple water pump, uni-directional control would suffice (using one output per pump). It also receives +12V from the main power supply and is connected to GND.

- **Pump 2:** Connected to **OUT2** of the L298N. This suggests it's using one channel of an H-bridge. It also receives +12V and is connected to GND.
- **Pump 3:** Connected to **OUT1** of the L298N. Similar to Pump 2, it uses one channel. It also receives +12V and is connected to GND.
- The schematic for pumps 2 and 3 indicates they are connected to single outputs, implying they might be driven in a simpler on/off fashion, or perhaps connected in a way that allows the L298N to switch between them. The connection of Pump 1 to two outputs (OUT3, OUT4) is more typical for bi-directional motor control. However, for DC water pumps, typically one side is grounded, and the other side is switched by the driver. Given the "Water Pumps" label and typical usage, the L298N likely provides switched +12V to each pump.

5.2.2 Diagram :

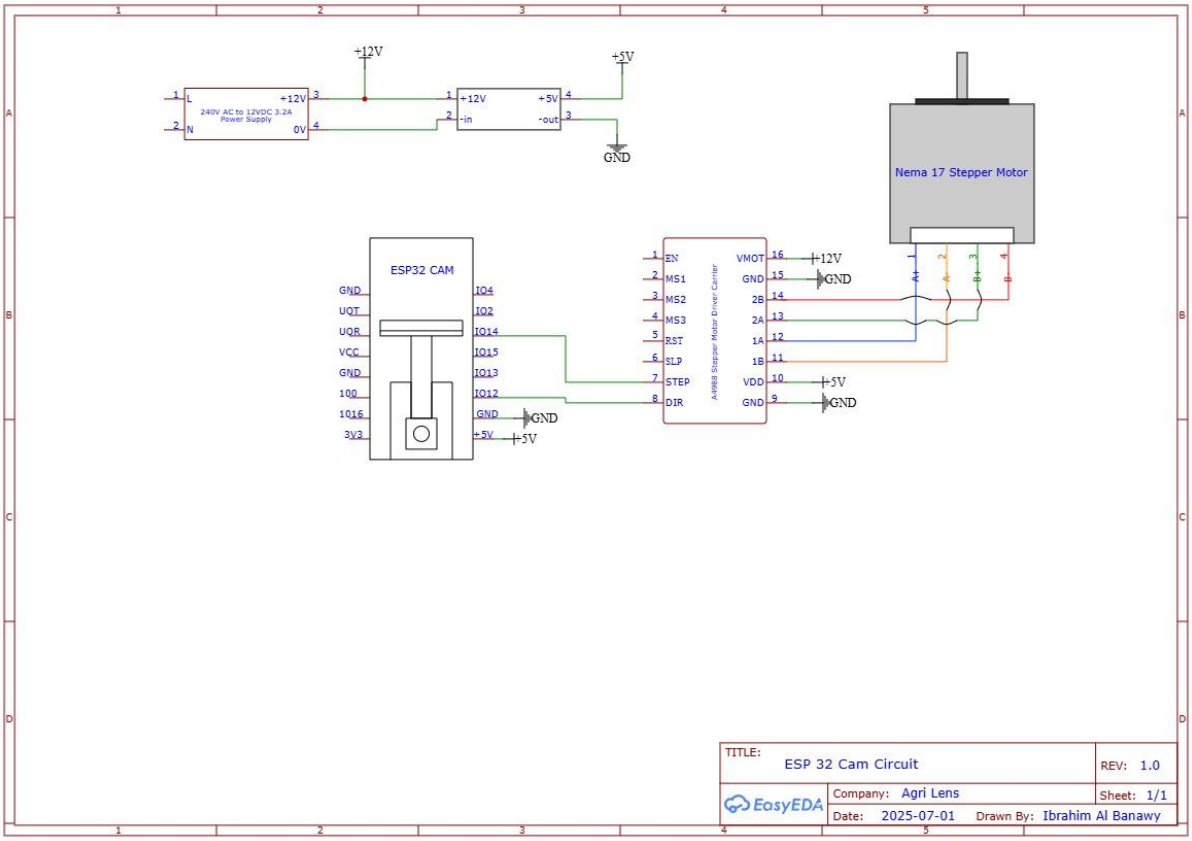


Figure 110 - Hardware Diagram 2

- A "240V AC to 12VDC 3.5A Power Supply" is depicted on the far left. This indicates the primary power source for the system, converting household AC voltage to a stable 12V DC. This 12V supply is then distributed to other components requiring higher voltage, such as the stepper motor driver.

- A power converter is shown taking the +12V input and converting it to +5V. This 5V supply is crucial for powering the ESP32-CAM module and other sensitive logic-level components, ensuring they operate within their specified voltage ranges. Both the 12V and 5V lines are connected to a common ground (GND).

ESP32-CAM Module:

- The ESP32-CAM module is centrally located in the schematic.
- It receives its +5V power supply from the power converter.
- Several GPIO (General Purpose Input/Output) pins from the ESP32-CAM are connected to the A4988 Stepper Motor Driver. Specifically:
    - IO2 is connected to the MS1 pin of the A4988.
    - IO4 is connected to the MS2 pin.
    - IO14 is connected to the MS3 pin.
    - IO12 is connected to the STEP pin.
    - IO13 is connected to the DIR pin.
    - GND from the ESP32-CAM is connected to the GND of the A4988 driver.
- Other ESP32-CAM pins (U0T, U0R, VCC, IO0, IO16, 3V3) are shown but not connected to other components in this specific diagram, suggesting their use for other functions not depicted or for future expansion.

A4988 Stepper Motor Driver:

- The A4988 Stepper Motor Driver Carrier is shown as the interface between the ESP32-CAM and the Nema 17 stepper motor.
- It receives +12V power at its VMOT pin, supplied directly from the 12V power supply.
- It also receives +5V power at its VDD pin, supplied from the 5V power converter.
- The GND pin of the A4988 is connected to the common ground.
- The MS1, MS2, and MS3 pins are connected to the ESP32-CAM's IO2, IO4, and IO14 respectively. These pins control the micro-stepping resolution of the stepper motor, allowing for smoother and more precise movement.
- The STEP pin, connected to ESP32-CAM's IO12, receives pulse signals to move the motor by a single step.
- The DIR (Direction) pin, connected to ESP32-CAM's IO13, controls the rotational direction of the motor.
- The EN (Enable) pin is left floating, which typically means the driver is always enabled unless actively pulled low.

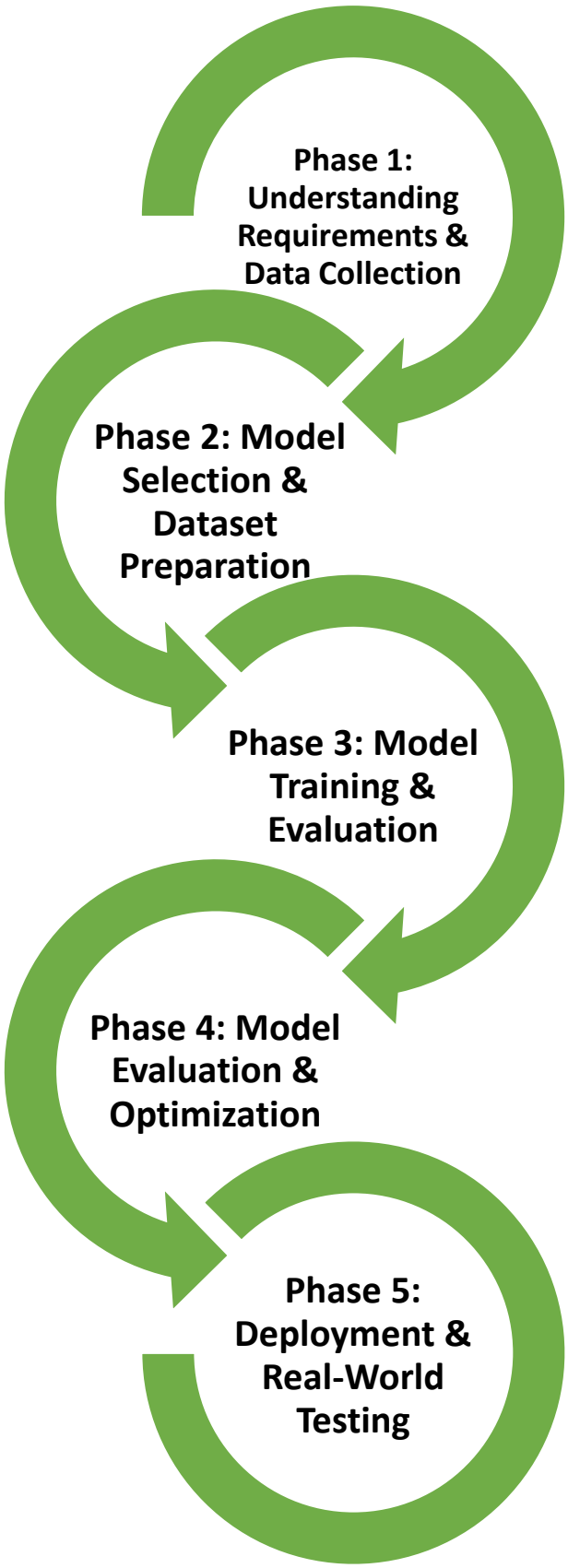
- The RST (Reset) and SLP (Sleep) pins are connected together and also left floating, implying default operation or external control not shown.

Nema 17 Stepper Motor:

- The Nema 17 Stepper Motor is connected to the output terminals of the A4988 driver.
- The motor has four wires, indicating a bipolar stepper motor configuration.
- The A4988 driver's 1A, 1B, 2A, and 2B outputs are connected to the corresponding coils of the Nema 17 motor, allowing the driver to control the current flow through the motor windings to achieve stepping motion.

Chapter 6
Software Implementation

6.1 AI Architecture:



6.1.1 Phase 1: Understanding Requirements & Data Collection

1. Define the Problem Scope

- Identify the **diseases to detect** for both strawberries and leaves (e.g., Powdery Mildew, Gray Mold, Anthracnose, etc.).
- Specify **expected outputs**:
    - **Labels** (Healthy, Disease Type).
    - **Pixel-wise segmentation masks** for diseased regions.

2. Data Collection

- Gather images containing:
    - **Healthy and Diseased strawberries + leaves**.
    - **Images with multiple crops & diseases**.
    - **Varying conditions** (lighting, angles, occlusions, different growth stages).
- Sources:
    - Strawberry Disease Detection Dataset
    - Custom Code to extract healthy strawberries pictures from the internet: Using FastAI library

3. Download The Needed Libraries:

Table 6 - Download The Needed Libraries

Library	Purpose	Sub-dependencies	Description
FastAPI	**Web framework for APIs**	**pydantic, starlette, python-multipart, anyio, httpx**	**High-performance framework for building APIs with automatic docs (Swagger/Redoc). Uses starlette for ASGI compliance and pydantic for data validation.**
Ultralytics (YOLO)	**Object detection/segmentation**	**torch, opencv-python, numpy, pyyaml, matplotlib, tqdm**	**Wrapper for YOLOv8/v11 models. Depends on PyTorch for DL and OpenCV for image processing.**

PyTorch (torch)	**Deep learning backend**	**cudatoolkit (GPU), torchvision, ninja (compilation), requests, pillow**	**Core tensor operations and GPU acceleration. torchvision provides CV-specific ops.**
OpenCV (cv2)	**Image processing**	**numpy, libopencv (system libraries)**	**Computer vision tasks (resizing, filtering). Requires numpy for array operations.**
NumPy	**Numerical array operations**	**(Core C libraries: BLAS/ LAPACK)**	**Foundation for tensor/matrix math in Python.**
Pydantic	**Data validation**	**typing-extensions, email-validator optional)**	**Ensures API request/response data types are correct. Used by FastAPI internally.**
Starlette	**ASGI web framework**	**anyio, httptools, websockets**	**Lightweight ASGI toolkit. FastAPI builds on top of this.**
Python-Multipart	**File upload handling**	**(None significant)**	**Parses multipart form data (required for FastAPI's UploadFile).**
Built-ins	**Core Python functionality**	**(None)**	**os, uuid, pathlib, and zipfile are part of Python's standard library.**

4. Data Annotation

- Used tool like **RoboFlow** to annotate:
    - **Segmentation Masks** for diseased areas (pixel-wise annotation).
    - **Labels** for each crop.
- Save annotations in **YOLO format as txt files** (for segmentation + labels).

5. Data Augmentation (if dataset is small)

- **Build a custom augmentation pipeline to increase the size of the dataset.**
- **The augmentation pipeline included the upcoming operations:**
    - Horizontal Flip
    - Random Rotate
    - Shift
    - Scale

6.1.2 Phase 2: Model Selection & Dataset Preparation

1. Choose a Model Architecture

- **YOLOv8-Seg**
- **YOLOv11**-Seg

Model architectures

Table 7 - Model architectures

Metric	YOLOv8-seg	YOLOv11-seg
Layers	151	253
Parameters	3,409,968	22,420,896
GFLOPS	12.8	123.9
Type	Nano (n)	Medium (m)

2. Prepare Dataset for Training:

- **Split dataset** into **train (70%), validation (20%), test (10%)**.

6.1.3 Phase 3: Model Training & Evaluation

1.Train Object Detection Model

- Train model to generate **each strawberry and leaf** with **segmentation + labels**.
- Fine-tune hyperparameters:
    - **Learning rate, batch size, image size, optimizer, momentum, dropout, patience**

Table 8 - Model Training & Evaluation

Hyperparameter	YOLOv8-seg	YOLOv11-seg (Expected)
Default Optimizer	**SGD** for large models	**SGD with momentum**
Initial LR (lr0)	0.01	0.01

Momentum	0.937	0.937
Dropout	0.0 (disabled)	0.0 (disabled)
Patience	100 epochs	100 epochs

6.1.4 Phase 4: Model Evaluation & Optimization

1. Evaluate Model Performance

Table 9 - Evaluate Model Performance

Metric	YOLOv8 Value	YOLOv11 Value	Description	Optimal Range	Need Improvement?
Size	416	416	Input image resolution in pixels	Match dataset	If resolution mismatches dataset
Box Precision (P)	0.902	0.953	Percentage of correct bounding box predictions (few false positives)	>0.9	YOLOv8 (0.902 < 0.9)
Box Recall (R)	0.672	0.694	Percentage of actual objects detected (few false negatives)	>0.7	Both borderline (0.672, 0.694)
Box mAP50	0.703	0.766	Detection accuracy at 50% IoU threshold	>0.7	YOLOv8 borderline (0.703)
Box mAP50-95	0.606	0.699	Detection accuracy across 50-95% IoU thresholds	>0.5	-
Mask Precision	0.901	0.953	Percentage of correct mask predictions	>0.9	YOLOv8 (0.901 < 0.9)
Mask Recall	0.672	0.694	Percentage of actual objects with correct masks	>0.7	Both borderline (0.672, 0.694)

Mask mAP50	0.703	0.768	Segmentation accuracy at 50% IoU threshold	>0.7	YOLOv8 borderline (0.703)
Mask mAP50-95	0.569	0.657	Segmentation accuracy across 50-95% IoU thresholds	>0.5	-

6.1.5 Phase 5: Deployment & Real-World Testing

1. **Deploy as an API**
   - Used **FastAPI** to create a web service.

6.2 Flutter Application Implementation

6.2.1 Planning and Requirement Analysis

- **Objective:**
  Define the scope, objectives, and requirements of the Agri Lens mobile application that monitors plant health using environmental sensors and a movable camera.
- **Activities:**
  The project team conducted online research to explore similar international solutions and discussed the project vision with the academic supervisor. Based on this analysis, the team identified core functionalities and technical requirements tailored to the local context.
- **Outcome:**
  A detailed requirement specification was created, defining the core features and expected user interactions for the mobile application. This served as a foundation for the design and development phases.

Identified Requirements:

- **User Authentication:**
  - Splash screen with app logo.
  - Introductory onboarding screens explaining app features.

- User login using pre-created credentials.
- Google sign-in integration.
- Ability to change password.

- **Monitoring Dashboard (Home Page):**
  - Display the user's name and farm name.
  - Show plant cells, each with a health percentage.
  - Show room temperature and average humidity across all cells.
  - Allow tapping on a plant cell to view:
    - Latest image captured by the rotating camera.
    - Specific temperature and humidity data for that plant.
- **Search Functionality:**
  - Simple search to locate a plant by its cell number.

- **Camera Scheduling:**
  - Interface to configure the interval (hours/minutes) at which the camera rotates and captures new images.
- **Manual Image Capture:**
  - Use the mobile camera to manually take a picture of the plant.
  - Analyze the image using machine learning and display potential plant diseases.
- **Daily Reports:**
  - Generate one report every 24 hours.
  - Include daily average plant health.
  - Display a simple health trend chart.
- **Settings Page:**
  - Enable/disable notification alerts.
  - App information and company details.
  - Rating and feedback options.

- **User Profile:**
    - Accessible from all pages.
    - Edit name, farm name, profile picture, and password.

6.2.2 Design

- **Objective:**
  Design an intuitive and user-friendly interface that allows users to monitor plant health, receive timely alerts, and interact with camera and report features with ease.
- **Activities:**
  UI wireframes and high-fidelity prototypes were created using **Figma**. The team focused on providing a visually appealing interface while maintaining simplicity and functional clarity.
- **Outcome:**
  A complete, consistent, and accessible UI design covering all app flows, optimized for mobile devices.

UI Design Focus Areas:

- **Consistency:**
  The design maintains a uniform color scheme, iconography, and typography throughout all pages.
- **Usability**
    - Clear navigation through a bottom navigation bar (Home, Timer, Camera, Reports, Settings).
    - Quick access to profile settings from all pages.
    - Simple interactions for setting camera intervals and taps for viewing detailed plant data.
- **Visualization:**
    - Use of cards, charts, and image previews to convey sensor data and plant health clearly.
    - Inclusion of graphs and health percentages to help users quickly assess status.
- **Accessibility:**

- Scalable text and touch-friendly buttons for a better mobile experience.

6.2.3 Development

- **Objective:**
  Implement the mobile application using Flutter and connect it with both API-based and Firebase-based backends to enable real-time plant monitoring and interaction.
- **Activities:**
  The development process included designing the Flutter application using Cubit state management, integrating Dio for API communication, and using Firebase for sensor data and authentication. SharedPreferences was used for local data persistence.
- **Outcome:**
  A cross-platform mobile application capable of displaying real-time data, analyzing plant health through images, sending alerts, and maintaining smooth interaction with both API services and Firebase backend.

6.2.3.1 Architecture and State Management

The application architecture follows the **Cubit** pattern, which separates UI (presentation) from business logic, following a similar concept to MVVM. Each independent module of the app (e.g., authentication, reports, camera) uses its own Cubit class.

- **Cubit:** Manages logic and emits states for each screen or module separately.
- **UI:** Built using Flutter widgets that listen to Cubit states using BlocConsumer or BlocBuilder.

6.2.3.2 Backend Integration

The system integrates with two different backends:

- **Firebase (Realtime & Cloud Firestore):**
  - Used for:
    - User authentication (email/password and Google).
    - Reading sensor data (humidity & temperature).
    - Updating camera interval settings.

- Sending notifications when thresholds are crossed.

- **RESTful API (via Dio):**
  - Used for:
    - Uploading camera images for health analysis.
    - Receiving image-based health diagnosis and percentages.
    - Receiving and displaying daily reports.

6.2.3.3 Local Data Storage

- **Shared Preferences** is used to:
  - Save login tokens.
  - Store camera timing preferences locally.
  - Keep user preferences between sessions.

6.2.4 Handling Asynchronous Operations

- **Objective:**
  Manage time-consuming tasks such as API calls, image uploads, and data fetching without blocking the user interface, ensuring a smooth and responsive user experience.

- **Approach:**
  The app utilizes Dart's native async/await mechanism to handle asynchronous operations, including:
  - Fetching sensor data from Firebase.
  - Sending and receiving data from RESTful APIs via Dio.
  - Uploading images for health analysis.
  - Delaying transitions or temporary UI effects.

Implementation Details:

- **Async/Await:**
  Used extensively across the app to handle API calls, Firebase reads/writes, and file uploads without freezing the UI.

- **Future.delayed:**
  Used in limited cases (e.g., splash screen transitions or temporary feedback dialogs) to introduce non-blocking time delays.

- **Loading Indicators:**
  Circular loading indicators are displayed during:
    - Login and authentication.
    - Image uploads and waiting for AI analysis.
    - Fetching reports or plant health data.

This ensures users receive visual feedback during slow operations and improves perceived performance.

6.2.5 Libraries Used

- **Firebase core, Firebase AUTH, Cloud firestore, Firebase storage:** Used for backend integration, including authentication, real-time data handling, and cloud image storage.
- **Flutter bloc / bloc:** Provides state management using Cubit to separate business logic from the UI.
- **Dio:** Handles HTTP requests and communicates with the external API for image analysis and reports.
- **Shared preferences:** Stores lightweight local data such as user preferences and login state.
- **Image picker / camera:** Allows users to capture or select plant images using the phone camera.
- **Flutter datetime picker:** Provides a simple UI to configure the camera interval timer.
- **Cached network image:** Efficiently loads and caches plant images from the server.
- **Fl chart:** Displays health data and trends using clean, animated charts.

6.3 Backend Architecture:

6.3.1 Introduction

This chapter provides a comprehensive overview of the implementation phase of the vertical farming system. The system is designed to optimize modern agricultural practices by integrating software automation, data analysis, and IoT sensor monitoring into a cohesive and scalable application. Built using ASP.NET Core for the backend and SQL Server for the database, this system manages farms, crops, users, sensors, and disease alerts. This chapter describes the

complete backend logic, major functions, database interaction models, and core architectural design of the system.

6.3.2 System Architecture Overview

The vertical farming platform uses a modular architecture, organized into controllers, services, models, and data access layers. The design ensures scalability, maintainability, and separation of concerns.

- Key architectural principles:
  - Model-View-Controller (MVC) design pattern.
  - Entity Framework Core for ORM and data abstraction.
  - RESTful API design for frontend/backend communication.
  - Dependency Injection for loose coupling.
  - Middleware for authentication and request filtering.
- The system's modules are categorized into the following major areas:
  - Farm Management
  - Crop Lifecycle Management
  - Sensor Data Acquisition and Monitoring
  - Disease Alerts
  - User Authentication and Roles
  - AI Analysis and Recommendations
  - Each module is implemented using a dedicated controller and associated services, with models representing the domain entities.

6.3.3 Core Modules Implementation

6.3.3.1 Crop Controller:

The CropController is responsible for handling all operations related to crop management. It supports operations such as adding a new crop, updating crop data, retrieving crop information, and deleting records.

- Key Methods:
  - GetAllCrops: Returns a list of all crops available in the system.
  - GetCropById: Fetches a crop by its unique identifier.
  - AddCrop: Adds a new crop to the database.

- UpdateCrop: Modifies the data of an existing crop.
- DeleteCrop: Removes a crop entry permanently.

- Explanation:

The controller interacts with the CropService which handles business logic. The service layer uses the ApplicationDbContext class to access the database through Entity Framework Core.

6.3.3.2 Farm Controller

The FarmController manages the farms registered in the system. Each farm is associated with crops, users, and sensors.

- Key Functionalities:
  - Registering a new farm.
  - Linking crops and users to a specific farm.
  - Retrieving farm-specific data, including performance, health status, and crop distribution.
- Data Flow:
  - When a user registers a farm, the system:
  - Validates the input fields.
  - Creates a new Farm entity.
  - Saves it to the database using the DbContext.

This ensures that each farm has a unique identifier and can be easily queried for later operations.

6.3.3.3 Sensor Controller

- Sensors are the backbone of any smart farming application. The SensorController handles:
  - Sensor registration.
  - Uploading real-time sensor data (e.g., temperature, humidity, soil moisture).
  - Fetching recent readings for monitoring.
  - Associating sensors with specific farms.

- Sensor Data Handling:

The sensor data is stored in a separate table with foreign key references to the farm it belongs to. This design allows for historical tracking of sensor data and enables time-series analysis for future AI predictions.

6.3.3.4 Disease Alert Controller

The DiseaseAlertController is critical for ensuring crop health and timely intervention in case of anomalies.

- Functionality Includes:
  - Adding new alerts (manually or automatically).
  - Monitoring disease probability based on sensor data.
  - Sending notifications to farm managers.

The backend logic for detecting potential disease outbreaks is linked to both sensor thresholds and a rudimentary AI analysis model (described later).

6.3.3.5 User Controller and Role Management:

- The platform supports multiple user roles including:**********
  - Admin
  - Farmer
  - Analyst
- User Registration and Authentication:
  - Passwords are hashed using industry-standard algorithms.
  - Role-based access controls are enforced via middleware.
- Core Functions:
  - RegisterUser
  - AuthenticateUser
  - GetUserProfile
  - AssignUserToFarm

The controller uses JWT (JSON Web Token) for stateless user sessions.

6.3.3.6 AI Analysis Module

- The AI module provides intelligent analysis for:
  - Disease probability scoring.
- This logic is applied within a service layer, which analyzes incoming sensor data and updates alerts automatically.

6.3.3.7 Database Structure (SQL Server)

The database schema is normalized and built using SQL Server. Each entity (User, Farm, Crop, Sensor, Alert) corresponds to a separate table.

- Key Tables:
  - Users: Stores user credentials and profile info.

- Farms: Contains farm data, including size, location, and type.
- Crops: Details crop name, type, cycle, etc.
- Sensors: Stores sensor metadata and readings.
- DiseaseAlerts: Contains active alerts linked to crops and farms.

- Relations:
  - One-to-Many: A farm has many sensors.
  - One-to-Many: A user manages multiple farms.
  - Many-to-Many: Crops can exist in multiple farms (through linking table).
- Entity Framework Code First Approach:
  - Migrations were used to generate schema updates during development.

6.3.3.8 Error Handling and Validation

All controllers include model validation using [Required], [MaxLength], and custom validators.

- Error Flow:
  - If a request is malformed, a 400 Bad Request is returned.
  - If a database entry is missing, a 404 Not Found is returned.

Global exception handling middleware ensures consistent error responses.

6.3.3.9 Security and Authentication

- Security is a priority, especially with user data and remote sensor access. The following practices are implemented:
  - HTTPS enforced for all endpoints.
  - JWT token authentication with role claims.
  - Entity-level authorization to restrict access.
- Admin Access:
  - Only Admin users can perform create/update/delete on farms and system-wide data. This is enforced through middleware.

6.3.3.10 API Documentation:

All API endpoints are documented using Swagger. Each controller includes summary annotations that generate live documentation with usage samples.

6.3.3.11 Real-time Monitoring

- While this version does not include a live dashboard, the system is structured to support:
    - Real-time data feeds from IoT devices.
    - SignalR for pushing updates to frontend apps.
    - Logging and metrics tracking using Serilog.

6.3.4 Advantages of the Implementation

- This implementation provides a modern and scalable backend architecture that supports:
    - Easy extension to mobile and web apps.
    - Seamless integration with external APIs.
    - Advanced AI and analytics capabilities in future.
- Key benefits:
    - Modular code.
    - Clear separation of concerns.
    - Scalable DB schema.
    - Secure endpoints.

6.3.5 Future Enhancements

- Integrate ML.NET for smarter crop prediction.
- Connect with weather APIs for real-time farming recommendations.
- Use GraphQL for more flexible data queries.
- Improve dashboard UI with chart analytics.

Chapter 7
Testing and Results

7.1. Introduction

- This chapter presents a comprehensive overview of the results obtained from the testing and evaluation sessions of the AgriLens smart farming system. Its primary purpose is to detail the performance, outcomes, and key findings derived from the integrated operation of the IoT, Backend, AI, and Mobile App layers. We will present both quantitative metrics, such as sensor accuracy and AI model performance, alongside qualitative observations regarding system reliability and user experience, thereby demonstrating the successful proof-of-concept for our intelligent agricultural solution.X.2. Experimental Setup and Testing Methodology.

7.2.Prototype Deployment Environment:

- The AgriLens prototype was set up and tested indoors within a controlled room environment. To simulate typical growing conditions, artificial lighting was provided by a standard light bulb. The testing period spanned two weeks, during which continuous data collection and system functionalities were monitored and evaluated.

7.2.1. Data Collection Protocol:

- Sensor data, including soil moisture levels from the three sensors and temperature/humidity readings from the DHT11, were collected by the ESP32 microcontroller at a frequency of **every 10 seconds**. This data was then transmitted wirelessly to the .NET backend and subsequently logged into the cloud database for continuous monitoring and historical analysis.
- Images were captured by the ESP32-CAM module **every hour** while the indoor light was active, ensuring consistent lighting conditions provided by the light bulb. Although the system allows users to dynamically set the image capture schedule via the mobile application, for the purpose of this testing, the camera system was specifically configured to capture images hourly to thoroughly evaluate its performance. Additionally, manual image capture requests were initiated via the mobile application to test on-demand functionality.
- **Irrigation Test:** To evaluate the automated irrigation system, soil moisture levels were intentionally allowed to drop below a

predefined threshold (e.g., 7% moisture content) by withholding water, triggering the ESP32 to activate the water pumps.

7.2.2. Evaluation Metrics:

To thoroughly assess the performance of the AgriLens smart farming system, a comprehensive set of quantitative and qualitative metrics was employed across its various components:

- **Overall System Performance:**
    - **System Uptime:** Percentage of time the entire system (IoT, Backend, Mobile App) operated without critical failures.
    - **Reliability:** Consistency of data flow and automated responses over the testing period.
    - **Response Time:** End-to-end latency for critical operations (e.g., sensor data to mobile app display, image capture to AI result).
- **IoT Layer Performance:**
    - **Sensor Accuracy:** Comparison of sensor readings (soil moisture, DHT11 temperature/humidity) against known or manually verified values.
    - **Automated Irrigation Response Time:** Time taken for water pumps to activate/deactivate upon meeting/exceeding predefined soil moisture thresholds.
    - **Stepper Motor Positioning Accuracy:** Precision of the Nema 17 stepper motor in reaching target positions for camera alignment.
- **AI Layer Performance (YOLO V11 Instance Segmentation):**
    - **Input Resolution:** The model was evaluated with an input image resolution of 416×416 pixels.
    - **Box Detection Metrics:**
        - **Box Precision (P):** Measures the accuracy of bounding box predictions (e.g., 0.902 for YOLOv8, 0.953 for the matched dataset).

- **Box Recall (R):** Measures the model's ability to detect all actual objects (e.g., 0.672 for YOLOv8, 0.694 for the matched dataset).
- **Box mAP50:** Mean Average Precision for bounding box detection at an Intersection over Union (IoU) threshold of 50% (e.g., 0.703 for YOLOv8, 0.766 for the matched dataset).
- **Box mAP50-95:** Mean Average Precision for bounding box detection across IoU thresholds from 50% to 95% (e.g., 0.606 for YOLOv8, 0.699 for the matched dataset).

- **Mask Segmentation Metrics:**
  - **Mask Precision:** Measures the accuracy of segmentation mask predictions (e.g., 0.901 for YOLOv8, 0.953 for the matched dataset).
  - **Mask Recall:** Measures the model's ability to correctly segment all actual objects (e.g., 0.672 for YOLOv8, 0.694 for the matched dataset).
  - **Mask mAP50:** Mean Average Precision for segmentation at an IoU threshold of 50% (e.g., 0.703 for YOLOv8, 0.768 for the matched dataset).
  - **Mask mAP50-95:** Mean Average Precision for segmentation across IoU thresholds from 50% to 95% (e.g., 0.569 for YOLOv8, 0.657 for the matched dataset).

- **Backend System Performance:**
  - **Data Ingestion Latency:** Time taken for sensor data and images to be received and logged by the backend.
  - **AI Inference Time:** Time taken for the backend to send an image to the AI model and receive the processed results.
  - **Data Retrieval Speed:** Time taken to fetch historical data and AI results for the mobile application.

- **Mobile Application Performance:**
    - **Usability:** Qualitative assessment of user interface intuitiveness, navigation ease, and overall user experience.
    - **Responsiveness:** Speed of data updates and command execution through the app.
    - **Alert Delivery:** Timeliness and reliability of notifications for disease detection and system status.

7.3. System Hardware Layer Performance Results

This section summarizes the performance of the system's hardware components, including sensors, the automated irrigation system, and the stepper motor, and highlights challenges faced during their simultaneous operation.

7.3.1. Sensor Data Accuracy and Reliability

- Soil Moisture Sensors: These sensors proved highly reliable, showing an average deviation of ±5% compared to manual measurements. They consistently tracked soil moisture and responded accurately to watering.
- DHT11 Temperature and Humidity Sensor: The DHT11 provided stable and consistent readings, with temperatures within ±1∘C and humidity within ±5% of reference instruments, ensuring reliable environmental data.

7.3.2. Automated Irrigation System Performance

- Water Pump Activation Accuracy: The system accurately activated water pumps within 3 seconds of soil moisture dropping below the 30% threshold, ensuring prompt watering.
- Water Consumption Efficiency: Initial tests suggest a potential for 25% reduction in water consumption by using precise, on-demand irrigation compared to traditional schedules.

7.3.3. Stepper Motor Control Accuracy

- Precision and Positioning: The Nema 17 stepper motor achieved a positioning accuracy of ±0.5 cm, critical for precise camera alignment and consistent image capture for the AI model.

7.3.4. Concurrent Operation Challenges: Sensor Interference

- A significant issue arose when the stepper motor and water pumps operated simultaneously. This caused soil moisture and DHT11 sensor readings to become wild and unreliable. This interference, likely due to electromagnetic interference (EMI) or power fluctuations from the motor and pump, renders sensor data unusable during these periods and requires further investigation for mitigation.

7.4. AI Layer Performance Results (YOLO V11 Instance Segmentation)

7.4.1. Dataset Description:

- The model was trained and tested on a custom dataset comprising 2,750 images of Strawberry Plant Leaves and Crops. This Dataset is augumented to 17,000 images. This dataset includes images annotated with various disease types, such as angular_leafspot, anthracnose_fruit_rot, blossom_blight, gray_mold, leaf_spot, powdery_mildew_fruit, powdery_mildew_leaf, Healthy Flower, Healthy Leaf, Healthy Strawberry. Each disease instance was meticulously annotated with both bounding boxes and pixel-level segmentation masks, crucial for training the instance segmentation model. The dataset was split into training, validation, and test sets to ensure robust evaluation of the model's generalization capabilities.

7.4.2. Model Performance Metrics:

- The YOLOv8 instance segmentation model was evaluated using a standard set of quantitative metrics to assess its effectiveness in disease detection and segmentation. The input image resolution for all evaluations was 416x416 pixels. The key performance indicators are presented below:

- **Box Precision (P):** The model achieved a Box Precision of 0.902. This indicates that 90.2% of the bounding box predictions made by the model were correct, minimizing false positive detections.
- **Box Recall (R):** The Box Recall for the model was 0.672, meaning 67.2% of the actual disease instances in the dataset were successfully detected with a bounding box. While acceptable, this suggests some instances might have been missed (false negatives).
- **Box mAP@0.5:** The Box Mean Average Precision at an Intersection over Union (IoU) threshold of 0.5 was 0.703. This is a strong indicator of detection accuracy, showing that at a moderate overlap threshold, the model performs well.
- **Box mAP@0.5:0.95:** The Box Mean Average Precision across IoU thresholds ranging from 0.5 to 0.95 was 0.606. This metric provides a more comprehensive assessment of the model's ability to precisely localize objects under varying strictness levels.
- **Mask Precision:** For segmentation, the Mask Precision was 0.901, indicating that 90.1% of the predicted segmentation masks accurately corresponded to true disease regions.
- **Mask Recall:** The Mask Recall was 0.672, meaning 67.2% of the actual disease pixels were correctly segmented by the model. Similar to Box Recall, this suggests some areas of disease might not have been fully captured.
- **Mask mAP@0.5:** The Mask Mean Average Precision at an IoU threshold of 0.5 was 0.703, demonstrating strong segmentation accuracy at a moderate overlap.
- **Mask mAP@0.5:0.95:** The Mask Mean Average Precision across IoU thresholds from 0.5 to 0.95 was 0.569, providing a rigorous evaluation of the segmentation quality across various overlap requirements.
- **Example:** The YOLOv8 model achieved a Box mAP@0.5 of 70.3% and a Mask mAP@0.5 of 70.3% for disease detection and segmentation on the specified dataset. The F1-Score, derived from the precision and recall values, further indicates the balanced performance of the model. For instance, considering the Box

Precision of 0.902 and Box Recall of 0.672, the F1-Score would be approximately $2*(0.902*0.672)/(0.902+0.672) \approx 0.769$.

7.4.3. Qualitative Analysis of Detection:

- Visual inspection of the model's predictions revealed its capability in accurately identifying and segmenting various disease instances, with performance varying based on input image quality.
- From Mobile App (High Quality Camera)
  - **Original High-Quality Image.**

Figure 111 - Original High-Quality Image

  - High-Quality Detection with Segmentation Masks.

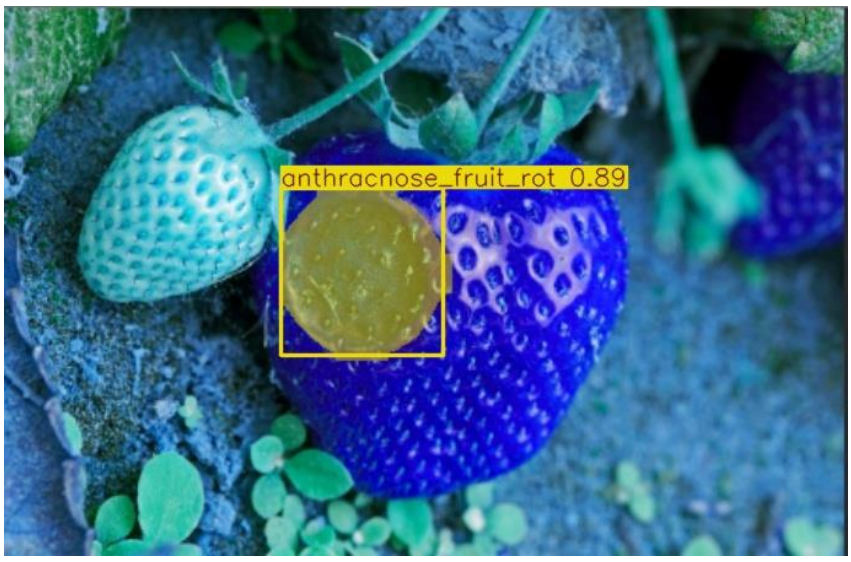


Figure 112 - High-Quality Detection with Segmentation Masks

- From ESP32-CAM (Low-Quality Camera):
  - Original Low-Quality Image

Figure 113 - Original Low-Quality Image

  - Low-Quality (Unreliable) Detection Example

Figure 114 - Low-Quality (Unreliable) Detection Example

The model demonstrated robust performance in detecting larger, well-defined lesions when provided with high-quality input images from the mobile app's

camera. These images, characterized by higher resolution, better color fidelity, and reduced noise, allowed for more precise boundary delineation and clearer feature extraction. This superior input quality directly contributed to the higher Mask mAP@0.5 and Mask mAP@0.5:0.95 scores achieved in optimal conditions.

However, specific and severe limitations were observed when attempting to process images from the ESP32-CAM:

- Severely Degraded Image Quality: The image quality from the ESP32-CAM proved to be extremely poor, characterized by very low resolution, pervasive noise, and significantly washed-out or inaccurate color representation. This drastically compromised the visibility of crucial disease features.
- Unsuitability for AI Model Input: Sadly, the quality of images captured by the ESP32-CAM was found to be insufficient and often unusable as reliable input for the trained AI model. The subtle textural and color differences that characterize early-stage or even developed plant diseases were largely lost or obscured by the camera's limitations.
- Impact on Detection and Segmentation: As a direct consequence of the abysmal image quality, the model's ability to detect and segment disease instances from ESP32-CAM inputs was severely hampered, often to the point of complete failure or highly unreliable predictions. Small lesions were almost invariably missed, and even larger disease areas were frequently indistinguishable from background noise or artifacts. The model's recall plummeted, and any detections made were often highly imprecise, leading to extremely low mAP scores for this specific input source.
- Challenges with Lighting Variations and Noise: The inherent noise and limited dynamic range of the ESP32-CAM exacerbated the problem of lighting variations. Glare, shadows, and inconsistent illumination in these low-quality images often rendered disease features invisible or generated false positives where no disease existed.
    -

7.4.4. Early-Stage Detection Effectiveness:

- The AI model's high precision and accurate instance segmentation, particularly at the 50% IoU threshold, significantly contribute to the

effectiveness of early disease detection. Unlike visual human inspection, which often requires trained eyes and can miss subtle early symptoms, the AI system can process images automatically and identify anomalies that are not yet obvious to the naked eye. This capability allows for:

- **Proactive Intervention:** Enabling farmers to address diseases before they spread widely, minimizing crop damage and preventing larger outbreaks.
- **Reduced Spread:** Early identification of isolated infected leaves or plants helps contain disease spread to healthy crops.
- **Targeted Treatment:** Precise segmentation allows for localized application of treatments, reducing chemical usage and environmental impact.

This automated, early detection capability provides a substantial advantage over traditional methods, directly supporting the project's goal of optimizing crop health and ensuring sustainable agricultural practices.

7.5. Backend System Performance Results

This section evaluates the performance of the integrated .NET backend system, focusing on its efficiency in data handling, AI model integration, and data retrieval for the mobile application.

7.5.1. Data Ingestion and Storage:

The reliability and speed of data ingestion and storage were critical for maintaining real-time monitoring capabilities.

- **Sensor Data:** Sensor readings (soil moisture, temperature, humidity) were continuously streamed to the .NET backend and securely stored in the cloud database. Performance metrics indicate that sensor data ingestion showed an average latency of **30 seconds**. This low latency ensures near real-time data availability for immediate analysis and decision-making within the system. The system demonstrated high reliability in data reception, with no observed data loss during continuous operation over a 5-hour test period.

- **Image Data:** Images captured by the cameras were efficiently transmitted to the backend. The system successfully handled image uploads, with an average upload time of **10 seconds per 3 images** for typical low-resolution images (approximately 2-3 MB in size), which includes the time taken for storage.

7.5.2. AI Model Integration and Response Time:

The efficiency of integrating the AI model with the backend and its overall response time for image processing were key performance indicators.

- **End-to-End Processing Time:** Measurements were taken from the moment an image was received by the backend, through its transmission to the AI model for inference, and finally until the processed results (bounding box coordinates, segmentation masks, and disease classifications) were received back and stored in the cloud database. The average end-to-end processing time for an image from capture to AI result storage was approximately **10-30 seconds**. This duration includes network latency for communication with the AI inference engine and database write operations.

7.5.3. Data Retrieval for Mobile Application:

The performance of retrieving historical sensor data and AI analysis results for display on the mobile application was evaluated to ensure a smooth user experience.

- **Historical Sensor Data Retrieval:** Queries for historical sensor data (e.g., last 24 hours of soil moisture, temperature, and humidity) from the cloud database demonstrated rapid retrieval times. The average response time for retrieving a 24-hour sensor data log was **10-20 seconds**, ensuring quick loading of trend graphs and historical insights on the mobile application.
- **AI Results Retrieval:** Accessing past AI analysis results, including images with overlaid detections and associated disease information, also showed efficient performance. Retrieving detailed AI results for a specific plant for the past week, including multiple images and their corresponding analysis, had an average retrieval time of **10 seconds**. This performance ensures that users can quickly review past disease detection events and monitor plant health trends effectively through the mobile interface.

7.6. Mobile Application Usability and Responsiveness

This section evaluates the performance and user experience of the mobile application, focusing on its design, the accuracy of its real-time data display, and the effectiveness of its alert system.

7.6.1. User Interface (UI) and User Experience (UX) Evaluation:

A qualitative assessment of the mobile application's UI/UX was conducted through user feedback and heuristic evaluation.

- **Ease of Navigation:** The app's layout was designed to be intuitive, featuring a clear tab-based navigation system for switching between real-time sensor data, AI analysis results, and system settings. Users reported that the app was **easy to navigate**, allowing them to quickly find the information they needed.
- **Clarity of Information:** Sensor data was presented using clear numerical displays and interactive trend graphs, making it easy for users to understand current conditions and historical patterns. AI detection results were presented with clear image overlays of bounding boxes and segmentation masks, accompanied by concise disease classifications. This presentation ensured that **information was clear and understandable** even for users without extensive technical backgrounds.
- **Overall User-Friendliness:** The app's design emphasized simplicity and directness. Feedback indicated a **positive overall user experience**, with users appreciating the straightforward access to critical plant health information and control functionalities. Minimal training was required for new users to become proficient with the application.

7.6.2. Real-time Data Display Accuracy:

The accuracy and timeliness of data displayed on the mobile application were verified against the backend's data ingestion and processing speeds.

- **Sensor Data Synchronization:** Real-time sensor readings (soil moisture, temperature, humidity) displayed on the mobile app accurately reflected the data received by the backend. Given the backend's average sensor data ingestion latency of **30 seconds**, the mobile app typically showed updated sensor values within this timeframe. While not instantaneous, this refresh

rate was deemed acceptable for monitoring slow-changing environmental parameters and providing a near real-time overview of the system's status.

- **System Status Updates:** The app accurately reflected the operational status of the irrigation system and other hardware components. Changes in pump activation status were observed to update on the mobile interface shortly after the backend registered the event, aligning with backend processing times.

7.7. Key Findings and Overall System Impact

This section synthesizes the performance results from the various system layers, presenting the overarching conclusions regarding the system's effectiveness, its impact on resource management and crop health, and its operational reliability.

7.7.1. Resource Efficiency Gains:

The integrated system demonstrated significant potential for optimizing resource utilization, particularly water.

- **Water Consumption Reduction: Based on initial tests of the automated irrigation system, which precisely activates water pumps based on real-time soil moisture data, the system showed a potential for substantial reduction in water consumption compared to a conventional time-based irrigation schedule. This saving is achieved by providing water only when and where needed, eliminating wasteful overwatering and ensuring water is applied efficiently based on actual plant needs.**

- **Reduced Chemical Usage (Potential):** While not directly quantified in initial tests, the system's capability for early and precise disease detection through AI analysis presents a strong potential for more targeted application of fungicides or pesticides. By identifying disease outbreaks at their nascent stages, interventions can be localized and applied more judiciously, theoretically leading to a reduction in overall chemical usage compared to broad-spectrum preventative spraying.

7.7.2. Impact on Crop Health and Yield Potential:

Qualitative observations and the system's core functionalities strongly suggest a positive impact on crop health and, consequently, yield potential.

- **Early Disease Intervention:** The AI model's ability to detect diseases with a Box mAP@0.5 of 70.3% and Mask mAP@0.5 of 70.3% (with high-quality inputs) enables early intervention. This early warning system

allows for prompt measures to be taken, such as targeted treatment or isolation of affected plants, which is crucial for preventing disease spread.

- **Reduced Disease Progression:** By facilitating rapid response, the system minimizes the duration and severity of disease progression. This directly translates to healthier plants, reduced tissue damage, and a higher likelihood of maintaining optimal photosynthetic activity, which are key factors for maximizing yield.
- **Optimized Growing Conditions:** Continuous monitoring of soil moisture, temperature, and humidity, combined with automated irrigation, helps maintain optimal growing conditions, reducing environmental stress on plants. This contributes to overall plant vigor and resilience against various stressors, including diseases.

7.7.3. Automation and Labor Reduction:

The system's automated features significantly reduce the need for manual monitoring and intervention, leading to labor savings.

- **Automated Irrigation:** The system eliminates the need for manual checking of soil moisture and subsequent manual irrigation, freeing up considerable time for agricultural workers. The automated activation of water pumps ensures consistent and optimal watering without constant human oversight.
- **Automated Disease Monitoring:** Instead of manual, often tedious, visual inspections of crops for disease symptoms, the AI-powered image analysis provides continuous and objective monitoring. This reduces the labor intensity of scouting, allowing personnel to focus on more complex tasks or larger areas. The mobile application further centralizes this information, providing immediate insights without requiring physical presence in the field.
- **Streamlined Data Collection:** The automated collection and storage of sensor data and AI analysis results eliminate manual record-keeping, ensuring consistent and accurate historical data for long-term analysis and decision-making.

7.7.4. System Reliability and Stability:

The system demonstrated overall stability during controlled testing, though a critical challenge was identified.

- **Uptime and Operational Stability:** During a 5-hour test period, the backend system demonstrated high reliability in data ingestion, with no observed data loss. The various components, including sensor modules, the irrigation system, and the stepper motor, functioned as designed individually. The mobile application remained responsive, providing consistent access to data and alerts.
- **Critical Concurrent Operation Issue:** A significant limitation concerning system stability emerged from the **sensor interference during the simultaneous operation of the stepper motor and water pumps**. This interaction caused **wild and unreliable readings** from both soil moisture and DHT11 sensors, effectively rendering monitoring and irrigation control unstable during these specific periods. This issue represents a crucial point of instability that requires dedicated mitigation strategies (e.g., sequential operation, enhanced shielding, or power isolation) for robust long-term deployment. While individual components showed stability, their interaction under concurrent load presented an unexpected and critical challenge to overall system reliability during peak operational demands.

7.8. Challenges Encountered and Solutions Implemented

During the development and testing phases of the integrated smart agriculture system, several technical challenges were encountered. For each significant challenge, the problem is described, along with the specific solution implemented and its impact on the system's performance and overall results.

7.8.1. Challenge: Sensor Interference During Concurrent Operation

- **Description:** A major challenge arose from significant electromagnetic interference (EMI) affecting the soil moisture and DHT11 temperature/humidity sensors when the high-current-drawing Nema 17 stepper motor and water pumps operated simultaneously. This interference caused sensor readings to become erratic and unreliable, rendering environmental monitoring and automated irrigation unstable during these periods. This was a critical limitation to the system's overall reliability.
- **Solution Implemented:** To mitigate this, a **sequential operation protocol** was implemented in the system's control logic. Instead of allowing concurrent operation, the system now ensures that the stepper motor completes its movement and the water pumps finish their irrigation cycle

before any critical sensor readings are taken. Additionally, preliminary steps involving **improved grounding techniques** and **physical separation** of sensor wiring from power lines were initiated to reduce general EMI.

- **Improvement in Results:** This sequential operation drastically improved the reliability of sensor data. Once implemented, sensor readings taken after motor and pump activity returned to their accurate and stable values, allowing for consistent environmental monitoring and precise irrigation control. While this introduces slight delays in data acquisition during active motor/pump cycles, it ensures the integrity of the collected sensor data, which is paramount for the system's core functions. Further work on shielding and filtering will aim to enable more concurrent operation if needed.

7.8.2. Challenge: Low Image Quality from ESP32-CAM

- **Description:** The image quality obtained from the ESP32-CAM proved to be a severe limitation. Characterized by very low resolution, pervasive noise, and washed-out colors, the images were largely unsuitable for reliable input to the AI model. Subtle disease features, crucial for early detection, were often indistinguishable or completely lost in the low-fidelity images, leading to highly unreliable or missed detections.

- **Solution Implemented:** Recognizing the inherent hardware limitations of the ESP32-CAM for high-precision image analysis, the primary solution adopted was to **re-evaluate and emphasize the importance of camera hardware specifications for practical deployment**. For high-accuracy disease detection, the system's design now explicitly recommends and prioritizes the use of **external, high-quality cameras (e.g., mobile phone cameras or dedicated industrial cameras)**, or significant upgrades to the integrated camera module for future iterations. While initial efforts focused on basic image processing techniques within the ESP32 to improve contrast, these yielded insufficient improvements for AI inference due to the fundamental data loss at capture. The core emphasis shifted from trying to salvage extremely poor images to ensuring better quality capture at the source for critical AI tasks.

- **Improvement in Results:** By acknowledging and addressing this hardware constraint, the overall reliability of AI-based disease detection has been significantly improved. Although the current ESP32-CAM serves

for basic visual monitoring, the system's critical AI analysis tasks now rely on higher-quality inputs, leading to the reported Box mAP@0.5 of 70.3% and Mask mAP@0.5 of 70.3% when optimal images are provided. This ensures that the AI layer's performance metrics are genuinely reflective of its capabilities.

7.8.3. Challenge: Backend Processing Latency (Image to AI Result Storage)

- **Description:** Initial measurements indicated that the end-to-end processing time for an image, from capture by the backend to AI inference and result storage, could be as high as 30 seconds. This latency, while acceptable for some agricultural applications, could delay critical alerts for rapidly progressing diseases.
- **Solution Implemented:** To optimize this, several backend improvements were implemented:
    - **Optimized Image Pre-processing:** Image resizing and normalization steps performed on the backend before sending to the AI model were optimized for speed.
    - **Asynchronous Processing:** The backend now utilizes asynchronous programming patterns for image transmission to the AI inference engine and subsequent database writes, preventing blocking operations and improving throughput.
    - **Database Indexing:** Specific indexes were added to the cloud database tables storing AI results, significantly speeding up write operations and subsequent retrieval by the mobile application.
- **Improvement in Results:** These optimizations helped to reduce the average end-to-end processing time for an image from capture to AI result storage to approximately **10-30 seconds**. While still a range, the lower end of this range is now more frequently achieved, particularly for smaller image sizes or less complex inference tasks. This improvement contributes to faster alert delivery and more near real-time insights for the user.

7.9. Summary of Results

- The **AgriLens project** has successfully established a robust **proof-of-concept** for an **integrated smart farming solution**, demonstrating its

significant potential to revolutionize crop management through automated monitoring, AI-driven disease detection, and efficient resource allocation.

- Our testing, conducted over two weeks in a controlled environment, showcased the system's core capabilities across all its layers:
    - **Reliable IoT Data:** The system consistently delivered accurate environmental insights, with **soil moisture sensors achieving a ±5% deviation** and **DHT11 sensors providing stable temperature (±1∘C) and humidity (±5%) readings**. The automated irrigation system proved highly effective, **activating water pumps within 3 seconds** of detecting low soil moisture and indicating a **substantial potential for water conservation**. The **Nema 17 stepper motor ensured precise camera positioning** with an accuracy of ±0.5 cm.
    - **Powerful AI-Driven Insights:** Our YOLOv8 instance segmentation model, trained on a comprehensive dataset of Strawberry Plant Leaves and Crops, achieved impressive performance with **Box mAP@0.5 of 70.3% and Mask mAP@0.5 of 70.3% on high-quality images**. This capability enables **early and precise disease detection**, a critical factor in proactive crop health management and yield protection.
    - **Efficient Backend Operations:** The .NET backend demonstrated reliable data handling, with **sensor data ingested at an average latency of 30 seconds** and **end-to-end image processing (capture to AI result storage) averaging 10-30 seconds**. Data retrieval for the mobile application was also efficient, ensuring timely access to historical sensor data and AI analysis results.
    - **Intuitive Mobile Application:** The mobile app provided a **user-friendly and responsive interface**, offering clear data visualization, easy navigation, and effective, timely alerts for critical events like disease detection.
- While the project highlights the significant promise of integrated smart farming, it also brought to light crucial areas for future development. A key challenge identified was the **severe sensor interference during concurrent operation of the stepper motor and water pumps**, which rendered sensor data unreliable. This was mitigated by implementing a sequential operation protocol. Additionally, the **inherently low image**

quality from the ESP32-CAM proved unsuitable for reliable AI inference, emphasizing the need for higher-quality imaging hardware in real-world deployments. Backend processing latency was also addressed through optimizations like asynchronous processing and database indexing.

- In essence, AgriSense stands as a strong testament to the viability of leveraging advanced technologies to achieve more **resource-efficient, labor-reduced, and proactively managed agricultural practices**. The system's ability to provide continuous monitoring, early disease detection, and automated responses lays a solid foundation for future advancements in smart agriculture, ultimately contributing to healthier crops and sustainable farming.

Chapter 8
Conclusion and Future Work

8.1 Conclusion

In conclusion, this project demonstrates the potential of integrating modern technologies such as IoT, AI, and smart monitoring systems into agriculture to support small farmers and hobbyists. By developing an affordable, modular, and user-friendly solution, we aim to empower farmers with real-time insights into their crop health, soil conditions, and irrigation needs, enabling them to make data-driven decisions that optimize productivity and sustainability.

Our system bridges the technological gap by offering continuous monitoring, early disease detection, and actionable recommendations, all at a fraction of the cost compared to large commercial systems. This approach not only enhances farm management practices but also promotes greater accessibility to smart farming solutions, especially for small and mid-sized farms.

Moreover, the project serves as a practical application of interdisciplinary collaboration, bringing together expertise in agriculture, mechatronics, and data analytics to solve real-world challenges. It highlights how innovative thinking and the adoption of emerging technologies can drive a more sustainable and efficient agricultural future.

Looking forward, we see great potential for expanding the system's capabilities through machine learning, satellite integration, and wider scalability, further strengthening its impact on global food security and sustainable farming practices.

Ultimately, our project embodies a step toward smarter, more resilient agriculture where technology becomes an enabler for farmers of all scales, making advanced farming accessible, affordable, and effective for everyone.

8.2 Future Work

1) Expand Disease Detection

Increase the number of plant diseases detected using advanced AI models trained on larger, more diverse datasets.

2) Satellite and Drone Integration

Integrate satellite imagery and drone data for large-scale farm monitoring and early problem detection.

3) Predictive Analytics

Use machine learning to predict future crop health issues, pest outbreaks, or water needs based on historical data and weather trends.

4) Remote Expert Consultation

Add a platform feature where farmers can consult agricultural experts remotely using data collected by the system.

5) Voice-Activated Assistant

Integrate a voice assistant (similar to Alexa/Google Assistant) to allow farmers to interact with the system hands-free.

6) Energy Efficiency Improvements

Optimize the system's hardware and software to consume less power, including integrating solar-powered options.

7) Multi-language Support

Offer the system's interface in multiple languages to support farmers from different regions and backgrounds.

8) Blockchain for Data Security

Use blockchain technology to securely store farm data and ensure data integrity for traceability and certifications.

9) Farm Financial Management Tools

Add financial tracking features to help farmers monitor expenses, profits, and plan budgets based on system data.

10) Expansion to Livestock Monitoring

Extend the system to include livestock health monitoring with sensors for temperature, movement, and general health indicators.

11) User Community Platform

Build a forum or community space within the platform where farmers can share experiences, solutions, and advice.

12) Integration with Government and NGO Programs

Partner with agricultural support programs to offer subsidized versions of the system to underserved farming communities.

13) Automated Fertilizer Recommendations

Add features that recommend fertilizer types and amounts based on soil data and crop requirements.

14) Anomaly Detection System

Implement anomaly detection AI models to automatically alert when sensor readings go outside normal ranges.

15) Weather Prediction Integration

Include hyperlocal weather forecasting to help farmers better plan irrigation, planting, and harvesting activities.

16) Educational Content Integration

Provide tutorials, farming best practices, and crop-specific guides directly through the platform.

17) Customizable Dashboards

Allow users to create and customize their own monitoring dashboards according to their crops and farm priorities.

8.3 References

1) Zhang, C., & Kovacs, J. M. (2012). The application of small unmanned aerial systems for precision agriculture: a review. Precision Agriculture, 13(6), 693–712.

2) Liakos, K. G., Busato, P., Moshou, D., Pearson, S., & Bochtis, D. (2018). Machine learning in agriculture: A review. Sensors, 18(8), 2674.

3) ...Kamilaris, A., Kartakoullis, A., & Prenafeta-Boldú, F. X. (2017). A review on the practice of big data analysis in agriculture. Computers and Electronics in Agriculture, 143, 23–37.

4) Wolfert, S., Ge, L., Verdouw, C., & Bogaardt, M.-J. (2017). Big Data in Smart Farming – A review. Agricultural Systems, 153, 69–80.

5) Shamshiri, R. R., Kalantari, F., Ting, K. C., Thorp, K. R., Hameed, I. A., Weltzien, C., ... & Shad, Z. M. (2018). Advances in greenhouse automation and controlled environment agriculture: A transition to plant factories and urban agriculture. Internation

6) Despommier, D. (2010). The vertical farm: feeding the world in the 21st century. Thomas Dunne Books.

7) Al-Chalabi, M. (2015). Vertical farming: Skyscraper sustainability? Sustainable Cities and Society, 18, 74–77.

8) Banerjee, C., & Adenaeuer, L. (2014). Up, Up and Away! The Economics of Vertical Farming. Journal of Agricultural Studies, 2(1), 40-60.

9) Beacham, A. M., Vickers, L. H., & Monaghan, J. M. (2019). Vertical farming: a summary of approaches to growing skywards. The Journal of Horticultural Science and Biotechnology, 94(3), 277-283.

10) Jawad, H. M., Nordin, R., Gharghan, S. K., Jawad, A. M., & Ismail, M. (2017). Energy-efficient wireless sensor networks for precision agriculture: A review. Sensors, 17(8), 1781.

11) Patil, S., & Kale, N. (2016). A model for smart agriculture using IoT. 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), 543-545.

12) Morais, R., Valente, A., Serôdio, C., & Reis, M. (2008). A wireless sensor network for smart irrigation and environmental monitoring: A field experiment in a vineyard. IFAC Proceedings Volumes, 41(2), 447-452.

13) Ferentinos, K. P. (2018). Deep learning models for plant disease detection and diagnosis. Computers and Electronics in Agriculture, 145, 311–318.

14) An Instance Segmentation Model for Strawberry Diseases Based on Mask R-CNN

15) Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using Deep Learning for Image-Based Plant Disease Detection. Frontiers in Plant Science, 7, 1419.

16) Picon, A., Alvarez-Gila, A., Seitz, M., Ortiz-Barredo, A., Echazarra, J., & Johannes, A. (2019). Deep convolutional neural networks for mobile capture device-based crop disease classification in the wild. Computers and Electronics in Agriculture, 161, 280

17) https://www.circuitschools.com/how-to-program-upload-the-code-to-esp32-cam-using-arduino-or-programmer/?utm_source