WEB BASED SMART IOT-BASED SYSTEM FOR OPTIMIZED JAPANESE MELON FARMING: DATA-DRIVEN APPROACH TO ENHANCE YIELD AND QUALITY

LIEW KE YING

UNIVERSITI TUNKU ABDUL RAHMAN

WEB BASED SMART IOT-BASED SYSTEM FOR OPTIMIZED JAPANESE MELON FARMING: DATA-DRIVEN APPROACH TO ENHANCE YIELD AND QUALITY

LIEW KE YING

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Science Software Engineering with Honours

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

September 2025

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Name : Liew Ke Ying

ID No. : 2105456

Date : 18/09/2025

COPYRIGHT STATEMENT

© 2025, Liew Ke Ying. All right reserved.

This final year project report is submitted in partial fulfilment of the requirements for the degree of Bachelor of Science (Honours) Software Engineering at Universiti Tunku Abdul Rahman (UTAR). This final year project report represents the work of the author, except where due acknowledgement has been made in the text. No part of this final year project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my supervisor, Ts. Dr. Sugumaran a/l Nallusamy, for his invaluable guidance, continuous support, and patience throughout the development of this project. His advice and encouragement have been instrumental in shaping the research direction and ensuring the successful completion of this work.

I would also like to extend my appreciation to the Lee Kong Chian Faculty of Engineering and Science (LKC FES), Universiti Tunku Abdul Rahman (UTAR), for providing the facilities, resources, and a conducive environment to carry out this project. My heartfelt thanks also go to UTAR for the financial sponsorship which made this research possible.

Special thanks are due to Ts. Dr. Loo Joo Ling for moderating and organizing the weekly meetings that ensured consistent progress at the melon farm, and to Dr. Kwan Ban Hoe for sharing his expertise and invaluable advice on the overall farming process. I am also deeply grateful to Tan Yi Jing and Loi Zhen Yee, students from other courses, for their contributions in handling the hardware and automation aspects, which played an essential role in the practical implementation of this project.

Lastly, I would like to acknowledge the encouragement and support from my family and friends, who have always stood by me throughout the course of my study. Their motivation has been a source of strength that contributed to the successful completion of this Final Year Project.

ABSTRACT

This project presents the design and implementation of a web-based smart IoT system for Japanese melon cultivation, addressing the critical need for real-time monitoring, actionable analytics, and decision support in high-value crop farming. The system integrates IoT sensors to capture environmental parameters such as soil moisture, pH, electrical conductivity, temperature, and light intensity, with data first ingested via ThingSpeak and subsequently synchronized into a Supabase PostgreSQL database through an automated Edge Function and Cron Job. The application layer, developed using Spring Boot, manages business logic including threshold-based rule evaluation and integrates with Firebase Cloud Messaging to deliver real-time alerts and recommendations. Angular, Ng Zorro, TailwindCSS, and embedded Grafana dashboards form the presentation layer, providing farmers with intuitive visualizations such as time-series graphs, Soil Health Index computation, and correlation heatmaps. System testing and evaluation demonstrated reliable data integrity (99.81% completeness), accurate threshold-based suggestions, and efficient performance with an average application start time of 1.55 seconds. Functional and integration test cases confirmed robust user management, sensor threshold configuration, and task scheduling features. The findings highlight that the system effectively transforms raw IoT data into interpretable insights, enabling timely interventions that improve yield consistency and fruit quality. While the study faced limitations in full-scale deployment and hardware connectivity, the outcomes establish a scalable, cost-effective foundation for precision agriculture. Future work is recommended to expand deployment across full cultivation cycles, incorporate predictive analytics, and integrate advanced automation for irrigation and ventilation control.

Keywords: smart farming; IoT; Japanese melon; Supabase; Grafana; Firebase; soil health index

Subject Area: T57.6–57.97

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 General Introduction

The global agricultural sector is undergoing a profound transformation driven by the demand for higher productivity, sustainable practices, and improved crop quality. Traditional farming methods, which often depend on manual observation and experience-based decision-making, face growing limitations in meeting these demands. Issues such as inconsistent monitoring, inefficient resource allocation, and vulnerability to environmental fluctuations can lead to reduced yield and compromised crop quality. With the rise of emerging technologies, agriculture is shifting toward smart farming systems that integrate Internet of Things (IoT), data analytics, and automation to optimize operations and decision-making.

Japanese melon cultivation serves as an excellent case study for this transformation. As a premium and high-value crop, Japanese melons require precise environmental control covering parameters such as temperature, humidity, soil health, and light intensity to ensure consistency in growth and sweetness. Even slight deviations from the optimal range can result in significant loss of quality and market value. This makes real-time monitoring and intelligent data-driven decision-making critical for farmers.

In response to these challenges, this project presents the development of a Web-Based Smart IoT System for Optimized Japanese Melon Farming. The system integrates IoT-enabled sensors, a cloud-based data pipeline, and a web application to provide real-time monitoring, visual analytics, and automated suggestions for corrective actions. By leveraging data-driven approaches such as threshold-based recommendations, soil health indexing, and correlation analysis, the system not only enhances productivity and decision-making but also contributes to sustainable resource management. Ultimately, this project demonstrates how the synergy of IoT and data analytics can bridge the gap between traditional farming and modern precision

agriculture, empowering farmers to achieve higher yields, superior quality, and long-term sustainability.

1.2 Importance of Study

The significance of this study lies in its ability to demonstrate how IoT, cloud computing, and real-time data analytics can address pressing challenges in modern agriculture, particularly for high-value crops such as Japanese melons. Agriculture today faces resource constraints, labour shortages, and increasing demands for sustainable practices, making technology-driven solutions essential. By providing farmers with real-time environmental monitoring, automated threshold-based alerts, and decision-support tools, this project exemplifies the role of IoT in enhancing both productivity and sustainability (Dhanaraju et al., 2022; Pathmudi et al., 2023).

1.2.1 Technological & Academic Advancement

This study contributes to academic discourse on precision agriculture by presenting a practical IoT-based framework that integrates sensor networks, cloud databases, and visual analytics dashboards. Unlike generic smart farming studies, this project focuses on a premium crop that demands strict environmental control. Recent studies highlight that IoT-enabled monitoring of soil and microclimate parameters significantly improves situational awareness and supports real-time decision-making (Singh and Sharma, 2024; Pathmudi et al., 2023). By implementing novel features such as a soil health index and correlation heatmaps, this study strengthens research on agricultural informatics and smart greenhouse management (Maraveas et al., 2022).

1.2.2 Practical Significance

For farmers, the system bridges the gap between measurement and timely action. IoT-driven greenhouse technologies have been shown to reduce water and fertiliser waste, stabilise microclimate conditions, and protect crop quality (Singh et al., 2024; Huynh et al., 2023). In Japanese melon cultivation, where even small deviations in soil moisture, pH, or temperature can lead to financial losses, the system's threshold-based alerts and push notifications provide practical, actionable guidance. The integration of real-time dashboards and

correlation analytics further enables farmers to detect patterns that would otherwise remain hidden, supporting better farm management (Maraveas et al., 2022).

1.2.3 Societal and Economic Significance

At a broader level, this study contributes to sustainable agriculture and national food security priorities. The Food and Agriculture Organization (FAO, 2021) emphasises that agrifood systems must become more resilient to shocks while improving efficiency. In Malaysia, the National Agrofood Policy 2021–2030 (NAP 2.0) highlights embracing modernisation and smart agriculture as a strategic thrust for enhancing productivity and farmer income (Ministry of Agriculture and Food Industries, 2021). By demonstrating a working IoT-based monitoring and decision-support system for premium crop cultivation, this project directly aligns with these international and national agendas.

1.3 Problem Statement

Agriculture is increasingly adopting digital and IoT-based technologies to improve productivity, sustainability, and crop quality. However, while research on smart farming has expanded, most implementations remain limited to generic crop monitoring and lack integrated decision-support features (Pathmudi et al., 2023; Singh and Sharma, 2024). For high-value crops such as Japanese melons, which require strict environmental control, these gaps become critical as they directly affect yield consistency and market competitiveness.

To address this issue, three main problems are identified: lack of real-time monitoring and decision support, fragmented data analytics with limited actionable insights, and inconsistent yield quality. These problems reflect gaps in both research and practice, highlighting the need for a comprehensive, IoT-based smart farming system that translates sensor data into reliable, actionable guidance for farmers.

1.3.1 Problem Statement 1: Lack of Real-Time Monitoring and Decision Support

Although IoT devices are increasingly applied in agriculture, many current systems are limited to basic data logging and trend visualization, without offering real-time decision support (Dhanaraju et al., 2022; Singh and Sharma, 2024). Farmers often detect issues such as nutrient imbalance or water stress only after symptoms appear, resulting in reduced yield and crop quality. This gap justifies the need for a system that integrates continuous monitoring with immediate alerts and threshold-based recommendations tailored to the specific requirements of Japanese melon farming.

1.3.2 Problem Statement 2: Fragmented Data Analytics and Limited Actionable Insights

Existing IoT solutions frequently provide raw sensor readings without contextual interpretation. Studies highlight that without advanced analytics, the collected data is underutilized (Pathmudi et al., 2023; Maraveas et al., 2022). Farmers therefore lack the ability to optimise irrigation, fertilisation, and environmental adjustments effectively. This gap demonstrates the importance of systems that transform data into actionable insights through dashboards, automated suggestions, and visual analytics.

1.3.3 Problem Statement 3: Inconsistent Yield Quality

Japanese melons are highly sensitive to environmental fluctuations, and inconsistent control often results in variable sweetness, texture, and appearance. Research indicates that current IoT-enabled greenhouse systems improve control but remain insufficiently precise for high-value crops requiring premium-grade consistency (Huynh et al., 2023; Singh et al., 2024). This inconsistency leads to significant financial losses for farmers. Therefore, there is a need for an IoT-based solution that ensures stable environmental management and supports consistent yield quality through real-time monitoring and corrective interventions.

1.4 Aim and Objectives

1.4.1 Project Aim

The aim of this project is to design and develop a web-based smart IoT system for optimized Japanese melon farming that integrates real-time monitoring, data-driven analytics, and automated decision support. The system aspires to enhance yield consistency and crop quality by providing farmers with timely insights, visualised trends, and actionable recommendations derived from environmental and soil data.

1.4.2 Project Objectives

The project objectives aim to define the scope and purpose of this endeavour, guiding its direction and intended outcomes. These objectives serve as a roadmap for achieving specific milestones, ensuring clarity, alignment, and measurability throughout the project lifecycle. By outlining clear and actionable objectives, resources can be allocated effectively, and success can be evaluated against predetermined criteria. The objectives are as follows:

1.4.2.1 To develop a web-based IoT system for real-time monitoring of environmental parameters in Japanese melon farming.

This objective focuses on establishing a sensor-driven web-based IoT system capable of capturing real-time environmental and soil parameters critical to Japanese melon cultivation, including air temperature, air humidity, soil moisture, soil temperature, soil pH, soil conductivity, total dissolved solids (TDS), and light intensity. The design will ensure seamless integration of multiple sensors with an IoT gateway, while data transmission will be directed to a secure cloud platform. Success will be measured by the system's ability to continuously capture accurate data under operational conditions within the greenhouse environment.

1.4.2.2 To develop and integrate a data-driven analytics pipeline with visualization and analysis

This objective aims to transform raw IoT sensor data into meaningful insights through the implementation of cloud-based data storage and analytical models. The pipeline will include features such as interactive dashboards, a soil health

index (SHI) to assess soil quality, and correlation heatmaps to reveal relationships between environmental parameters. Visualization will be provided via Grafana and embedded into the web application for user accessibility. Success will be determined by the system's ability to provide real-time visualization, accurate soil health index (SHI) calculations, and correlation heat map that support data-driven decision-making for melon farming.

1.4.2.3 To enhance farming yield and crop quality by implementing automated alerts and suggestions based on parameter thresholds.

This objective addresses the development of threshold-based rules that evaluate sensor readings against optimal ranges for Japanese melon growth. When deviations are detected, the system will generate context-specific suggestions (e.g., irrigation adjustment, nutrient correction, or ventilation changes) and deliver push notifications to farmers in real time. Success will be measured by the timeliness, accuracy, and relevance of the alerts and recommendations, as well as the system's ability to support farmers in making immediate corrective actions that minimise risks to yield and crop quality.

1.5 Scope and Limitation of the Study

1.5.1 Scope of the Study

The scope of this study is confined to the design and development of a web-based smart IoT system for Japanese melon cultivation, with the objective of enhancing yield consistency and crop quality through data-driven decision support. The focus is placed on software development, system integration, and data analytics rather than the construction of physical hardware. Specifically, the scope encompasses the following aspects:

i. **IoT Data Acquisition** - Integration of environmental and soil parameters, including air temperature, air humidity, soil moisture, soil temperature, soil pH, soil conductivity, total dissolved solids (TDS), and light intensity, into a centralised platform for continuous monitoring.

ii. **Cloud-Based Data Management and Analytics** - Utilisation of a cloud database (Supabase) to manage sensor data, coupled with analytical features such as soil health indices and correlation heatmaps to support data-driven insights.

iii. **Web-Based Dashboard** - Development of an Angular-based web application with embedded Grafana visualisations to display real-time and historical sensor data in a user-friendly format.

iv. **Automated Alerts and Decision Support** - Implementation of threshold-based rules to generate actionable recommendations, with real-time push notifications delivered to farmers via Firebase.

v. **System Evaluation** - Functional and non-functional testing of the system, including metrics such as application responsiveness, data latency, data completeness, and system reliability, to assess its effectiveness in supporting smart farming practices.

The scope therefore highlights the development of a data-driven smart farming platform tailored for Japanese melon cultivation, with hardware contributions managed by collaborating students from related engineering courses.

1.5.2 Limitations of the Study

Although the study demonstrates the feasibility of integrating IoT technologies, cloud analytics, and decision support in agriculture, several limitations must be acknowledged:

i. **Exclusion of Hardware Development** - The physical setup of IoT sensors, microcontrollers, and gateways was undertaken by collaborating students from other engineering disciplines. This study is restricted to software, integration, and analytics components.

ii. **Limited Deployment Period** - Due to time constraints inherent in the Final Year Project schedule, the system was not deployed across the full melon growth and harvesting cycle. As a result,

long-term evaluation over multiple planting seasons could not be conducted.

iii. **Absence of Automated Environmental Control** - The system is designed to provide monitoring, analytics, and recommendations only. It does not incorporate automated actuation for irrigation, ventilation, or nutrient delivery, which remain manual processes.

iv. **No Artificial Intelligence or Machine Learning Integration** - Predictive models were not implemented, as sufficient datasets were unavailable during the early stages of deployment. Decision support is instead based on predefined threshold rules.

v. **Crop-Specific Focus** - The system is tailored to Japanese melon cultivation. Application to other crops would require reconfiguration of threshold parameters, modification of decision rules, and potential adjustments to the system architecture.

1.6 Contribution of the Study

This study contributes to both academic research and practical applications in the domain of smart farming. By focusing on Japanese melon cultivation, which demands precise environmental control for premium quality, the project demonstrates how an IoT-based, data-driven system can bridge the gap between traditional farming practices and modern digital agriculture. The contributions of the study are outlined as follows:

1.6.1 Academic Contributions

One of the major academic contributions of this study is the development of a framework for IoT-based smart farming systems. The project establishes a reference model that integrates real-time environmental monitoring, cloud-based data management, and web-based visualization. This framework not only demonstrates the feasibility of combining these technologies but also provides a foundation for future research in agricultural informatics, where similar systems may be replicated or extended for different crops and contexts.

Another significant contribution is the introduction of novel analytical features within the system. Unlike conventional monitoring platforms that only

display raw sensor data, this study incorporates a Soil Health Index (SHI) and correlation heatmaps to transform data into actionable insights. These features enrich the academic discourse by showing how IoT-generated data can be processed to support precision agriculture through decision-making tools, thereby advancing research in data-driven farming technologies.

The study also contributes to academic knowledge by providing an empirical performance evaluation of the developed system. Key metrics such as application responsiveness, data latency, data completeness, and overall reliability were systematically tested and analyzed. These results offer benchmarks that can be used by future researchers to evaluate and compare similar smart farming systems, ensuring that this project adds measurable value to ongoing research in the field.

1.6.2 Practical Contributions

On a practical level, this study delivers a system that provides decision support for farmers through threshold-based recommendations and real-time push notifications using Firebase. This functionality enables farmers to receive timely alerts when environmental or soil parameters deviate from optimal ranges, helping them to take immediate corrective actions. Such decision-support features reduce risks associated with poor crop management and directly contribute to maintaining yield and quality.

The system also enhances farm management practices by consolidating multiple environmental and soil parameters into a single, user-friendly platform. With visual dashboards displaying real-time and historical trends, farmers can monitor the overall condition of their crops more efficiently. This integration reduces the reliance on manual observation, supports more effective allocation of resources such as water and fertilizers, and simplifies day-to-day management of farming operations.

Furthermore, the project contributes by offering a prototype tailored for high-value crop cultivation, specifically Japanese melons. These crops demand stricter quality control compared to many staple crops, and the system

demonstrates how IoT solutions can be customized to meet these requirements. This prototype can serve as a model for adapting IoT technologies to other premium horticultural crops, thereby extending its practical impact beyond the immediate study.

1.6.3 Societal and Policy Contributions

At the societal level, the system supports sustainable agriculture by promoting efficient monitoring and timely interventions. By reducing unnecessary use of water, fertilizers, and other inputs, the system encourages environmentally responsible practices. This contribution aligns with global sustainability goals, ensuring that agricultural productivity is balanced with resource conservation.

The project also contributes to the national agenda through its alignment with Malaysia's National Agrofood Policy 2021–2030 (NAP 2.0). The policy highlights the need to embrace smart agriculture technologies as part of a long-term strategy to modernize the agrifood sector, increase productivity, and enhance farmer income. This study's outcomes demonstrate how digital technologies can directly support these strategic goals, thereby reinforcing their relevance to current policy directions.

Finally, the study contributes to food security and quality assurance by addressing yield optimization and consistency in Japanese melon farming. By enabling more precise control of crop growth conditions, the system ensures a higher probability of producing premium-grade melons that meet market expectations. Indirectly, such systems also support broader societal efforts to secure reliable food production and meet consumer demand for high-quality agricultural products.

1.7 Outline of the Report

This report is organized into eight chapters, each addressing a specific component of the research and development process for the web-based smart IoT system for Japanese melon farming. The structure is as follows:

i. **Chapter 1: Introduction** – Presents the background of the study, importance, problem statements, research aim and objectives, scope and limitations, and contributions. This chapter establishes the rationale and foundation of research.

ii. **Chapter 2: Literature Review** – Reviews existing works on smart farming, precision agriculture, IoT applications, cloud platforms, data management, analytics techniques, and decision support systems. It highlights global trends, benefits, and limitations, while identifying research gaps and positioning the present study.

iii. **Chapter 3: Methodology and Work Plan** – Describes the development methodology adopted, namely Rapid Application Development (RAD), and outlines the project phases. It further details the work breakdown structure, Gantt chart, and tools used to guide systematic system development.

iv. **Chapter 4: Project Specification** – Defines the system requirements, both functional and non-functional, and provides use case diagrams and descriptions. It also introduces the conceptual prototype that serves as the blueprint for system design and development.

v. **Chapter 5: System Design** – Explains the architectural design of the system, including its three-tier structure (presentation, application, and data layers). It also covers the database design, entity-relationship diagram, and schema specifications.

vi. **Chapter 6: System Implementation** – Provides a detailed explanation of how the system was implemented, covering functional modules such as authentication, authorization, user management, task management, sensor threshold configuration, and data visualization. It also describes the implementation of business logic components (Edge Functions, Cron Jobs, suggestion logic, and notifications) and analytics modules (time-series graphs, Soil Health Index, and correlation analysis).

vii. **Chapter 7: System Testing and Evaluation** – Presents the testing approach and results, including functional, integration, and performance testing. It also evaluates data handling accuracy,

visualization and analytics validity, threshold-based suggestions, and responsiveness of the system.

viii. **Chapter 8: Conclusion and Recommendations** – Summarizes the study's key findings in relation to its objectives, highlights encountered problems and limitations and provides recommendations for future improvements such as full-cycle deployment, advanced predictive analytics, and enhanced IoT integration.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter reviews the growing body of research on smart farming and precision agriculture, focusing on how IoT, cloud platforms, and data analytics are reshaping modern cultivation practices. For high-value crops such as Japanese musk melons, small fluctuations in soil pH, moisture, or temperature can determine fruit quality, making real-time monitoring and decision support critical. The review is structured around four themes: smart farming concepts and global trends, the role of data management and analytics, the use of decision support systems, and comparative studies of existing smart farming solutions. By examining both benefits and limitations reported in recent literature, the chapter identifies gaps in crop-specific tailoring, actionable insights, and real-time usability—gaps that this project addresses through the design of a web-based IoT system tailored to Japanese melon farming.

2.2 Smart Farming and Precision Agriculture

2.2.1 Definition and Concepts

Smart farming and precision agriculture are transformative approaches that employ modern technologies to enhance agricultural practices beyond what traditional farming achieves. Precision agriculture (PA) is typically defined as a data-driven farming management approach that observes, measures, and analyses the variability within fields in order to guide resource application such as water, fertilisers, or pesticides only where and when they are needed to maximize crop yield, quality, and input efficiency (Monteiro et al., 2021; Padhiary et al., 2025). Smart farming builds upon precision agriculture by integrating IoT sensors, cloud platforms, dashboards, and decision support mechanisms to enable more responsive, real-time control over farming operations (Mansoor et al., 2025; Roy, 2025).

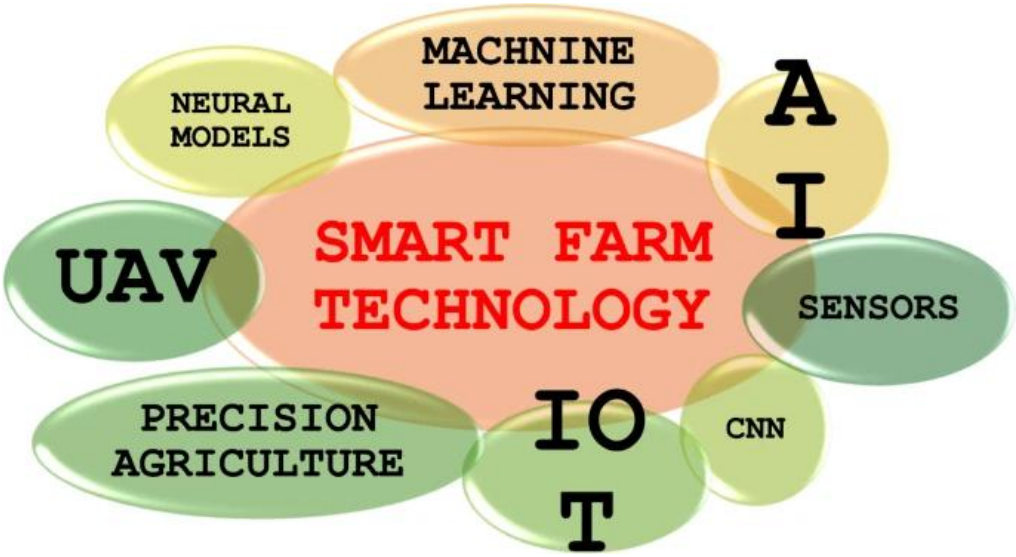


Figure 2.1: Conceptual framework of smart farming technology (Raj and Prahadeeswaran, 2025)

Traditional farming, in contrast, relies heavily on uniform inputs, manual observation, and fixed schedules without continuous feedback from the crop environment. This often leads to inefficiencies such as overuse or underuse of water, inconsistency in crop growth, and slower responsiveness to environmental changes (Monteiro et al., 2021). In the case of Japanese melon farming, these inefficiencies can manifest as variation in fruit sweetness, poor texture, or uneven maturity, because the crop is sensitive to microclimates and environmental fluctuations.

In summary, this section establishes that precision agriculture is about variability management and efficient input use, while smart farming is its broader technological extension with real-time monitoring and actionable intelligence. This conceptual groundwork is critical to justify the development of a smart IoT system for Japanese melon cultivation—one that captures environmental data, analyses it, and supports decisions in a responsive manner.

2.2.2 Global Trends (2020–2025)

In the period 2020-2025, global trends in smart farming reflect increased adoption of IoT, sensor technologies, and integrated analytics tools, driven by pressures such as climate change, resource scarcity, and the rising demand for

high-quality produce. A review by Mansoor et al. (2025) found that smart sensors for soil moisture, pH, and plant stress have become significantly more common in precision agriculture frameworks, particularly in regions facing water stress. Such sensors now often feed into systems that provide not only raw data but also data visualization dashboards and advisory outputs to farmers, indicating a shift toward systems that enable actionable insight rather than mere monitoring.

Another trend is the increasing use of hybrid connectivity models to ensure reliable IoT deployment in remote or rural farmland. A recent comparative study shows models combining LPWAN (e.g., LoRaWAN or NB-IoT) with 4G/5G cellular networks can improve network reliability and reduce costs in remote farm settings by up to 30% (Mohamed Rafi et al., 2025). This trend is important for scalability and robustness of IoT systems, particularly in regions with infrastructure limitations—factors highly relevant if systems are deployed outside urban centers or high-connectivity zones.

There is also a stronger emphasis on sustainability and efficiency. Reviews have underscored that IoT technologies are being increasingly paired with precision farming practices to optimize input use (water, fertilizer, energy) and reduce environmental pollution (Duguma et al., 2024). Efficiency gains are not only in production (yield) but also in resource use, aligning with global climate and sustainability goals. Countries with progressive agricultural policies are supporting smart farming via subsidies, technical training, and open-data initiatives to reduce the barrier to entry for farmers (Revolutionizing Agriculture: A Review, Raj & Prahadeeswaran, 2025).

Finally, there is a trend towards developing more integrated systems that combine real-time monitoring, analytics, and user engagement through dashboards and alerts. Systems are increasingly offering decision support, such as threshold-based alerts, predictive warnings, or advisory services, rather than simply collecting data. Although many of these systems still rely on machine learning or AI, there is growing recognition of the role that simpler, rules-based alerts (thresholds) can play, especially in early-stage or small-scale

deployments (Integration of Smart Sensors & IoT in Precision Agriculture, Mansoor et al., 2025).

2.2.3 Benefits and Limitations

The adoption of IoT technologies in agriculture offers a wide range of benefits that directly contribute to improved productivity, sustainability, and farm management. One of the most significant advantages is resource optimization, as IoT sensors enable precise monitoring of soil moisture, nutrient levels, and microclimatic conditions. This allows farmers to apply water and fertilizers only when required, reducing wastage and improving efficiency (Duguma et al., 2024). In addition, IoT-enabled systems support real-time monitoring, giving farmers continuous access to data that can enhance decision-making and improve responsiveness to sudden changes in the farm environment (Mansoor et al., 2025). This technology also contributes to labor reduction by automating data collection and reducing the need for manual field inspections (Raj & Prahadeeswaran, 2025). For high-value crops such as Japanese melons, these benefits are particularly important, as minor fluctuations in soil pH, humidity, or temperature can significantly affect fruit sweetness, texture, and market quality.

Despite these advantages, several limitations hinder the widespread adoption of IoT in agriculture. A major barrier is the high initial cost of sensors, connectivity infrastructure, and cloud service subscriptions, which may be prohibitive for smallholder farmers (Mohamed Rafi et al., 2025). Furthermore, IoT deployment in rural or greenhouse settings often faces connectivity challenges, as reliable networks such as 4G/5G or LPWAN may not always be accessible (Mansoor et al., 2025). Another limitation lies in data interoperability and complexity; the integration of heterogeneous sensor data into a unified platform can be technically demanding and requires expertise that many farmers may lack (Raj & Prahadeeswaran, 2025). Additionally, security and privacy issues surrounding cloud-stored farm data remain unresolved, raising concerns about data misuse or unauthorized access (Duguma et al., 2024).

In summary, IoT technologies in agriculture present substantial opportunities for enhancing efficiency and crop quality but face challenges of cost, connectivity, and technical accessibility. These benefits and limitations underscore the need for research on cost-effective, user-friendly, and crop-specific IoT solutions. The present study addresses this by focusing on a web-based IoT system tailored for Japanese melon farming, designed to provide actionable insights, reduce inefficiencies, and support consistent yield quality without overcomplicating the user experience.

Table 2.1: Benefits and Limitations of IoT in Agriculture

Aspect	Benefits	Limitations
Resource Management	Enables precise monitoring of soil moisture, nutrients, and microclimatic conditions, improving efficiency and reducing wastage (Duguma et al., 2024).	High initial cost of sensors, IoT gateways, and cloud services can be prohibitive, especially for smallholder farmers (Mohamed Rafi et al., 2025).
Decision-Making	Real-time monitoring provides continuous data access, enhancing decision-making and responsiveness to environmental changes (Mansoor et al., 2025).	Connectivity challenges in rural areas and greenhouses due to unreliable 4G/5G or LPWAN coverage (Mansoor et al., 2025).
Labor Efficiency	Automates data collection, reducing dependence on manual field inspections and lowering labor requirements (Raj & Prahadeeswaran, 2025).	Integration of heterogeneous sensor data into a unified platform is technically demanding and requires expertise (Raj & Prahadeeswaran, 2025).
Crop Status and Quality	Particularly beneficial for high-value crops like Japanese melons, where stable environmental control enhances fruit sweetness, texture, and consistency (Duguma et al., 2024; Mansoor et al., 2025).	Security and privacy concerns regarding storage of farm data on cloud platforms raise risks of misuse or unauthorized access (Duguma et al., 2024).

2.2.4 Identified Gap

Although IoT technologies have advanced rapidly and are increasingly applied in agriculture, several key gaps remain evident in the literature:

i. **Lack of crop-specific tailoring** - Most IoT applications are designed for general farming contexts and do not adequately address the unique environmental sensitivities of high-value crops such as Japanese melons, where small fluctuations in soil pH, humidity, and temperature critically affect yield and quality (Mansoor et al., 2025; Duguma et al., 2024).

ii. **Limited actionable insights** - Existing systems often focus on raw data collection and visualization but fall short in transforming these into decision-support features such as tailored recommendations or threshold-based alerts for farmers (Raj & Prahadeeswaran, 2025; Padhiary et al., 2025).

iii. **Barriers to accessibility and adoption** - High initial costs, unreliable connectivity, and technical challenges in integrating heterogeneous sensor data restrict the usability of IoT systems, particularly for smallholder farmers (Mohamed Rafi et al., 2025; Mansoor et al., 2025).

These gaps demonstrate the need for a cost-effective, user-friendly, and crop-specific IoT system that provides actionable decision support precisely what the present study aims to deliver for Japanese melon farming.

2.3 Data Management and Analytics in Smart Farming

2.3.1 Importance of Data Management

Effective data management is fundamental to the success of smart farming systems because IoT devices generate large volumes of heterogeneous data that must be properly structured, stored, and accessed to provide value. Without reliable data management, sensor reading may remain fragmented, inaccurate, or inaccessible, undermining the effectiveness of IoT adoption in agriculture. For high-value crops such as Japanese melons, which require

consistent monitoring of sensitive parameters like soil pH, moisture, and temperature, robust data management is particularly critical in ensuring that environmental variations are captured accurately and used for decision-making (Mansoor et al., 2025).

i. **Ensuring Data Accuracy and Reliability** – The quality of IoT data directly affects the trustworthiness of smart farming decisions. Poor data management may result in missing values, duplications, or delays, which in turn reduce system reliability. Studies emphasize that reliable data pipelines and structured storage mechanisms are essential to minimize latency and errors in real-time monitoring applications (Raj & Prahadeeswaran, 2025). This is especially important for greenhouse environments, where timely data access is needed for responsive actions such as irrigation adjustment or ventilation.

ii. **Supporting Scalability and Long-Term Use** – IoT in agriculture requires continuous monitoring across multiple crop cycles. Cloud-based data management ensures scalability, allowing systems to handle increasing amounts of sensor data while maintaining performance (Padhiary et al., 2025). Structured storage also enables longitudinal analysis, helping farmers and researchers to study correlations and seasonal patterns that are vital for long-term crop optimization.

iii. **Enabling Data-Driven Insights** – Well-managed data not only ensures accessibility but also enables transformation into actionable insights through analytics and visualization. Without structured data, advanced techniques such as soil health indices or correlation heatmaps—both applied in this study—would not be feasible (Duguma et al., 2024). Data management therefore serves as the foundation for bridging raw sensor readings with meaningful decision support.

iv. **Integration and Interoperability** – Smart farming often involves multiple sensors, platforms, and data formats. Robust data management frameworks enable integration of heterogeneous data

sources into a unified system, improving accessibility and usability (Mansoor et al., 2025). In Japanese melon farming, interoperability ensures that soil, environmental, and climate data are consolidated for holistic analysis, supporting yield and quality consistency.

2.3.2 Cloud Platforms in Agriculture

Cloud platforms form the backbone of smart farming systems by providing the infrastructure for sensor integration, real-time data ingestion, scalable storage, and advanced analytics. In this project, which focuses on Japanese melon cultivation, cloud platforms enable the seamless collection of environmental parameters such as soil moisture, pH, electrical conductivity, temperature, and light intensity. These platforms not only support remote monitoring through dashboards and mobile applications but also facilitate advanced analytics, including soil health indices and correlation heatmaps, that transform raw IoT data into actionable insights for optimized crop management. In the following section, several widely used cloud platforms for agricultural data management in the market will be discussed.

ThingSpeak, developed by MathWorks, is a lightweight IoT analytics platform that has been widely adopted in agricultural research and prototyping. It supports sensor data ingestion through REST APIs, enabling farmers and researchers to capture key environmental parameters such as soil moisture, humidity, and temperature. ThingSpeak also provides real-time visualization and basic analytics, which make it particularly suitable for applications such as irrigation scheduling, soil monitoring, and crop condition tracking. Due to its simplicity and accessibility, ThingSpeak is often applied in small-scale or experimental farming projects where rapid deployment and ease of use are prioritized (MathWorks, n.d.; Kadarabad et al., 2025).

Supabase is an emerging open-source Backend-as-a-Service platform that leverages PostgreSQL as its core database, offering structured relational storage, authentication, and real-time data streams. Although its adoption in agriculture remains limited compared to more established platforms like ThingSpeak and Firebase, its relational model makes it highly effective for

managing structured farm data. Examples include organizing sensor readings, maintaining detailed plot information, and storing farmer account profiles. This capacity for relational management provides greater flexibility for integrating diverse datasets, which can be especially beneficial in precision agriculture systems that rely on multi-parameter monitoring.

Firebase, developed by Google, has become one of the most widely used Backend-as-a-Service solutions for developing mobile and web dashboards in agriculture. It offers a wide range of services, including Firestore, a real-time database, authentication modules, and push notifications. These features allow farmers to visualize live sensor data and receive timely alerts via mobile devices, thereby enhancing decision-making and operational efficiency. Despite its advantages in rapid development and real-time updates, Firebase employs a NoSQL data model, which may pose challenges when handling relational agricultural datasets that require complex queries and structured analysis (Agarwal, 2025).

AWS IoT Core and its associated services provide enterprise-level scalability and advanced capabilities for agricultural applications. This platform enables secure device connectivity, real-time data streaming, and integration with other AWS services such as analytics and machine learning. Case studies have demonstrated AWS IoT being deployed in precision agriculture, particularly for greenhouse climate control, soil and crop sensor integration, and predictive analytics. The robustness and scalability of AWS make it a suitable choice for large-scale agricultural systems that demand both reliability and advanced data processing capabilities (AWS, 2018; AWS, n.d.).

Google Cloud IoT has also been successfully applied in agricultural projects, particularly where advanced analytics and machine learning integration are required. A notable example is the SpaceFarm initiative, which employed Google Cloud IoT services combined with predictive analytics to optimize greenhouse environments. This case demonstrates the platform’s ability to handle real-time monitoring, large-scale data management, and predictive modeling for agricultural optimization. By leveraging Google’s

global cloud infrastructure, farms can integrate IoT data with advanced machine learning pipelines to enhance decision-making and resource management (Google Cloud, n.d.; Google, 2022).

In summary, cloud platforms for agriculture vary in complexity, scalability, and suitability depending on farm size, objectives, and system design. ThingSpeak and Firebase are often preferred for small-scale deployments or prototypes due to their accessibility and real-time capabilities, while Supabase offers flexibility through its relational data management features. In contrast, AWS IoT and Google Cloud IoT are more suited for enterprise-level agricultural operations, where predictive analytics, automation, and advanced data integration are critical for achieving efficiency and scalability.

Table 2.2: Comparison of Cloud Platforms for Smart Farming

Platform	Key Features	Strengths	Limitations	Suitability in Agriculture
ThingSpeak (MathWorks)	REST API-based data ingestion, real-time visualization, basic analytics	Simple setup, widely used in research, suitable for rapid prototyping	Limited scalability, basic analytics only	Small-scale or experimental farms (e.g., irrigation scheduling, soil monitoring)
Supabase	Open-source BaaS, PostgreSQL relational storage, authentication, real-time streams	Strong relational model, flexible for structured datasets, open-source ecosystem, high free tier limit	Less adoption in agriculture, limited ecosystem maturity compared to Firebase or AWS	Precision agriculture projects requiring structured, multi-parameter datasets (e.g., soil + crop records)
Firebase (Google)	Real-time DB, authentication, push notifications, mobile/web dashboard integration	Fast deployment, strong mobile integration, real-time updates	NoSQL model complicates relational queries and structured analytics	Small to medium-scale farms needing dashboards and instant notifications
AWS IoT Core	Secure device connectivity, real-time	High scalability, robust ecosystem, advanced	Complex setup, higher cost	Enterprise-level smart farming (e.g., greenhouse

	data streaming, integration with AWS analytics & ML services	analytics & automation		automation, predictive analytics)
Google Cloud IoT	Device connectivity, ML/AI integration, predictive modeling, large-scale data pipelines	Strong analytics & ML support, global cloud infrastructure	More complex and costly than lightweight platforms	Large-scale or research-driven projects (e.g., greenhouse optimization, predictive farming)

2.3.3 Data Analytics Techniques

The rapid adoption of IoT in agriculture has resulted in a growing demand for data analytics techniques that can convert large volumes of sensor readings into actionable insights. In the current market, analytics solutions for smart farming can be broadly categorized into data visualization dashboards, time-series and trend analysis, composite indices and decision-support metrics, statistical and machine learning approaches, and predictive analytics frameworks. Each category offers unique advantages but also presents limitations depending on the scale of deployment, crop type, and local farming practices (Wolfert et al., 2017).

Dashboards and visualization platforms are among the most widely used analytics techniques in agriculture. Commercial and open-source tools such as Grafana, Power BI, and Google Data Studio are commonly deployed to present environmental parameters in real time. These dashboards provide farmers with accessible summaries of temperature, soil moisture, humidity, and nutrient levels. They also allow the integration of multiple data streams from cloud platforms such as AWS IoT Core and Google Cloud IoT. Studies show that dashboards improve user engagement and decision-making by presenting complex sensor data in an interpretable format (Mekonnen et al., 2021). However, dashboards are typically descriptive rather than predictive, and their value depends heavily on the underlying quality of data collected.

Time-series analysis and anomaly detection techniques are also extensively applied in the market. Vendors such as Microsoft Azure IoT and IBM Watson IoT integrate time-series databases and anomaly detection algorithms into their agricultural solutions, enabling farmers to detect abnormal fluctuations in soil moisture, pH, or light intensity. Academic studies have shown that time-series analysis supports irrigation management and greenhouse optimization by identifying cyclical patterns and abnormal readings (Singh et al., 2024). Despite these benefits, most applications still rely on historical trend monitoring, with limited predictive capacity for future conditions.

Composite indices and decision-support metrics have been increasingly promoted to simplify complex agricultural data. For example, Soil Health Indices (SHI) and crop stress indices combine multiple soil and environmental factors into a single score that reflects overall growing conditions (Singh et al., 2024). Commercial solutions such as CropX and Arable use similar composite indicators to provide farmers with holistic assessments of soil health and water efficiency. While such indices reduce the cognitive burden on farmers, they often lack universality and require localized calibration to specific soil types and crops, which restricts scalability across regions.

Statistical techniques and machine learning models represent another major area of analytics. Correlation and regression analyses are frequently used to identify relationships between environmental variables and crop performance, guiding targeted interventions in fertilization or irrigation (Khanna et al., 2020). At the commercial level, companies such as Prospera and Taranis deploy machine learning models that analyze sensor data, weather patterns, and satellite imagery to detect early signs of crop disease or predict yield outcomes. These approaches provide deeper insights than descriptive analytics, but they require high-quality datasets and computational infrastructure, which may not be feasible for smallholder farmers.

Predictive and prescriptive analytics frameworks are gaining traction as advanced solutions in the agricultural market. Predictive analytics leverages historical datasets combined with weather forecasts and soil models to anticipate future conditions such as drought stress, pest outbreaks, or nutrient deficiencies. Prescriptive analytics goes a step further by recommending specific interventions, such as adjusting irrigation frequency or applying fertilizer. Several agritech firms, including IBM Watson Decision Platform for Agriculture, have incorporated such capabilities into their solutions. However, these systems often demand high upfront investment and technical expertise, which can limit their adoption in developing regions.

As summarized in Table 2.3, dashboards, time-series visualization, and composite indices remain accessible and effective techniques for small- to medium-scale farms, while advanced predictive and prescriptive frameworks are more resource-intensive. In this project, emphasis is placed on dashboards, SHI, correlation analysis, and time-series visualization to balance feasibility with analytical depth.

Table 2.3: Comparison of Data Analytics Techniques in Smart Farming

Technique	Purpose	Advantages	Limitations
Dashboards & Visualization	Present multi-sensor data in real time through intuitive interfaces.	Easy to use, improve farmer engagement, and integrates multiple data streams.	Descriptive only; dependent on data quality.
Time-Series Analysis & Anomaly Detection	Identify temporal patterns, cycles, and abnormal fluctuations in sensor readings.	Supports irrigation scheduling, greenhouse optimization; enables anomaly detection.	Mostly retrospective; limited predictive capacity.
Composite Indices & Decision-Support Metrics	Aggregate multiple parameters into simplified indices for soil or crop health.	Provides holistic soil/crop assessment; reduces complexity for farmers.	Requires local calibration; limited universality.
Statistical & Machine Learning Models	Discover relationships or predict outcomes from complex datasets.	Deeper insights; supports targeted interventions.	High computation needs; costly for smallholders.
Predictive & Prescriptive Analytics	Anticipate future conditions and recommend corrective actions.	Proactive insights; supports resource optimization.	Require expertise, expensive, less accessible in developing regions.

2.3.4 Role of Data Analytics in Decision-Making

The integration of data analytics into agriculture has transformed how decisions are made, enabling more efficient, timely, and evidence-based practices. By analyzing IoT-generated sensor data, farmers are able to optimize the use of inputs, monitor crop growth environments, and respond effectively to emerging challenges. Weraikat et al. (2024) demonstrated that the use of electrical conductivity (EC) data correlated strongly with potassium levels in melon cultivation in southern Croatia, thereby allowing farmers to manage nutrient application more cost-effectively without relying solely on laboratory testing. Such findings highlight how analytics can reduce operational costs while maintaining crop quality.

In addition to nutrient management, data analytics supports broader operational decision-making such as irrigation scheduling, pest control, and greenhouse climate optimization. Thilakarathne et al. (2025) noted that the combination of IoT monitoring, and data analytics allows farmers to detect anomalies such as soil moisture deficits or pest infestations at an early stage, enabling corrective interventions before yield quality is compromised. This proactive approach improves resource efficiency and reduces the risk of crop failure.

Furthermore, analytics can enhance planning and risk mitigation by providing farmers with insights into alternative scenarios. Getahun et al. (2024) highlighted that predictive and scenario-based data analytics enable the evaluation of different irrigation or fertilization strategies, thus helping farmers to anticipate outcomes and manage risks. While advanced predictive techniques may extend beyond the scope of this project, their role in the wider market illustrates how analytics increasingly underpins agricultural decision-making.

For the purposes of this project, data analytics contributes directly to decision support through dashboards, soil health indices, correlation analysis, and threshold-based alerts. These techniques ensure that farmers can monitor

real-time conditions, understand soil health status, and receive actionable recommendations, thereby enhancing decision-making in Japanese melon cultivation.

2.3.5 Identified Gap

Although recent literature demonstrates the growing importance of data analytics in agriculture, several limitations persist that justify the direction of this project:

i. **Over-emphasis on advanced predictive models and machine learning** - Many studies have prioritized predictive frameworks for yield forecasting, disease detection, and risk assessment (Getahun et al., 2024; Thilakarathne et al., 2025). While effective, such approaches demand large datasets, high computational resources, and specialized expertise, making them less feasible for smallholder farmers or resource-constrained environments.

ii. **Limited practical implementation of composite indices** - Although composite measures such as the Soil Health Index (SHI) have been highlighted as valuable tools for simplifying multi-parameter soil quality evaluation (Weraikat et al., 2024), their use remains largely conceptual. Few studies demonstrate their integration into real-time, field-ready decision-support systems accessible to farmers.

iii. **Challenges in data latency, accessibility, and usability** - Connectivity limitations, delayed data transmission, and lack of user-friendly interfaces are frequently reported as barriers to adoption. Thilakarathne et al. (2025) emphasized that rural connectivity gaps hinder IoT applications, while Weraikat et al. (2024) noted that reliable, continuous data flow is critical for supporting timely decision-making in agricultural contexts.

In light of these gaps, this project seeks to develop an affordable and practical IoT-based system that integrates accessible analytics techniques—namely dashboards, SHI, correlation analysis, and alert mechanisms—into a

web-based platform for Japanese melon farming. By prioritizing interpretability, real-time usability, and cost-effectiveness, the project addresses the need for actionable insights without the complexity and resource intensity of advanced predictive models.

2.4 Decision Support Systems in Agriculture

2.4.1 Concept and Frameworks

Decision Support Systems (DSS) in agriculture are computer-based tools designed to aid farmers, agronomists, and stakeholders in making informed management decisions by combining data input, rules or models, and decision logic. Three main types/frameworks are common:

i. **Rule-based / Threshold-based systems** - These rely on predetermined thresholds or rule sets. For instance, when soil moisture drops below a certain level, trigger irrigation; or when a pest risk index exceeds a threshold, recommend pesticide application. Such systems are relatively simple to implement and transparent but may lack adaptability to varying conditions.

ii. **Advisory / Expert systems** - These integrate expert agricultural knowledge (often domain rules, crop models, historical data) to offer recommendations beyond just thresholds. They may incorporate soil and weather model simulations, or disease/pest risk predictions, offering advice such as nutrient management, scheduling, or crop protection strategies.

iii. **Hybrid and Model-based frameworks** - These combine threshold/rule-based logic with statistical, mechanistic, or even machine learning models to provide more sophisticated advice (e.g. predictions, scenario planning). They often handle multiple parameters (soil, weather, crop growth stage), allow what-if simulations, adapt over time with updated data.

These frameworks differ in complexity, data requirements, computational need, and usability. As more agriculture becomes connected (IoT, remote sensing), there is an increasing shift toward hybrid DSS that can

process real-time sensor data and provide dynamic advice rather than static guidelines.

2.4.2 Benefits and limitations of DSS

Decision Support Systems (DSS) provide several benefits in agriculture. They enable timely interventions by detecting risks such as pests, diseases, or soil stress early, allowing farmers to respond before serious damage occurs (Tratwal, 2025). DSS also reduces risks by combining forecasts, thresholds, and models to minimize uncertainty in farm management decisions (Tratwal, 2025). In addition, they improve resource efficiency, optimizing water, fertiliser, and pesticide use, which enhances productivity while reducing environmental impacts (Petraki et al., 2025).

However, DSS face notable limitations. Many lack real-time IoT integration, relying on periodic or forecast data that reduce responsiveness (Tratwal, 2025). Real-time alerts are also scarce; while some systems provide warnings, few offer mobile push notifications, with recent prototypes such as Jouini (2025) still limited in scope. Usability is another concern, as complex interfaces and poor connectivity hinder adoption in smallholder contexts (Petraki et al., 2025). Moreover, most DSS have a narrow focus, addressing single issues such as irrigation or pest control rather than delivering comprehensive, multi-parameter decision support.

In summary, DSS enhances agricultural decision-making through timely, risk-aware, and efficient interventions, but their effectiveness is constrained by gaps in real-time functionality, usability, and breadth of support.

2.4.3 Identified Gap

Based on recent literature (2020-2025), the gaps in Decision Support Systems for agriculture that this project aims to address are:

i. **Absence of systems combining real-time push notifications with comprehensive advisory support** - While warning systems and alerts exist in some DSS (e.g. pest/disease risk alerts), integration

of mobile push notifications triggered by IoT sensor thresholds across multiple parameters is rare.

ii. **Limited integration of full environmental and soil parameter sets in one DSS** - Many DSS focus on single or few parameters (e.g. pest risk + weather or irrigation only), but do not include a broad set like soil moisture, soil pH, conductivity, temperature together with advisory logic.

iii. **Poor usability and accessibility for farmers with constrained resources** - There is a gap in systems designed for user-friendly interaction, low infrastructure dependency, and operation under limited connectivity, especially in greenhouse or small-farm environment contexts.

iv. **Lack of systems validated under operational conditions for specific crops such as Japanese melon** - Few DSS studies are applied and evaluated for specific cultivars and under real greenhouse or controlled environments. Crop-specific validation is sparse.

2.5 Comparative Analysis of Related Works

2.5.1 Overview of Existing Smart Farming Systems

Recent smart-farming solutions converge on an IoT → (edge/fog) → cloud pipeline with web/mobile dashboards and varying levels of decision support. Reviews and systems papers consistently report multi-sensor deployments (soil moisture, EC, pH, temperature, humidity, light) streaming to cloud databases and dashboards for greenhouse and field control (Bersani et al., 2022; Maraveas, 2022; Soussi et al., 2024). Edge/fog architectures have emerged to cut latency and dependency on wide-area links, improving responsiveness for time-critical actions (Hong et al., 2024). At the DSS layer, integrative platforms such as DAKIS combine heterogeneous data (in-situ sensors, remote sensing, models) to support land-use and management choices but are often strategic rather than crop-specific operational. For cucurbits, greenhouse studies increasingly exploit data-driven methods; for example, climate forecasting in greenhouses with netted melons (a Japanese-melon type)

shows how analytics can anticipate environmental dynamics, albeit with research-grade ML rather than deployable farmer tools.

2.5.2 Comparative of recent smart farming system

i. Bersani et al. (2022) — IoT in smart greenhouses (state of the art).

Bersani and colleagues survey IoT approaches for greenhouse monitoring and control, synthesising common sensing stacks (soil moisture, temperature/humidity, light, CO_2), network protocols (MQTT/HTTP), and typical cloud dashboards/actuation loops for irrigation and ventilation. The review underlines tangible benefits (continuous observation, automation potential) but also recurrent challenges, notably integration and interoperability across heterogeneous devices and the tendency of deployments to plateau at descriptive monitoring rather than mature, farmer-facing decision support. This positions greenhouse IoT as technically robust yet often under-leveraged analytically (Bersani et al., 2022).

ii. Maraveas & Bartzanas (2021) — IoT for optimised greenhouse environments (review).

Maraveas and Bartzanas compile evidence on IoT-enabled optimisation of microclimate and irrigation, emphasising low-cost sensor integration, remote monitoring, and efficiency-oriented KPIs for greenhouse management. The review discusses scheduling/optimisation themes and reports actuation (e.g., irrigation/ventilation), while flagging constraints in generalisability and operational usability (connectivity, human factors). It concludes that, although IoT can improve energy and input efficiency, many systems need better decision logic and farmer-friendly interfaces to translate sensing into day-to-day actions (Maraveas & Bartzanas, 2021).

iii. Hong et al. (2024) — Fog-computing smart farm (implementation study).

Hong et al. implement an IoT smart-farm architecture that moves computation from cloud to fog/edge nodes. Through controlled experiments they compare communication protocols and network traffic, showing fog reduces overheads

and latency, enabling quicker responses for time-sensitive farm events. While the work evidence infrastructure-level gains and hints at faster operational decisions, the decision logic remains system-specific and the study foregrounds performance rather than farmer-facing advisory design (Hong et al., 2024).

iv. Soussi et al. (2024) — Smart sensors & smart data for precision agriculture (review).

Soussi and co-authors review sensing modalities and data pipelines in precision agriculture, with attention to mobile-accessible, cloud-backed visualisation. They highlight trends in data fusion and “smart data” practices while pinpointing persistent issues around data quality, integration, and standardisation that limit analytics depth. Although real-time monitoring is well covered, the review indicates that alerts and comprehensive decision support are less consistently embedded, especially for small and medium growers (Soussi et al., 2024).

Table 2.4: Feature comparison across recent smart-farming systems

Study / System	**IoT Integration**	**Data architecture**	**Analytics Capability**	**Decision Support Logic**	**Real time notifications**	**Key limitations**
Bersani et al. (2022) — review of IoT in smart greenhouses	Surveys soil & climate sensing (moisture, temp, RH, light, CO_2), typical MQTT/HTTP stacks	Device→ gateway→ cloud patterns consolidated	Mostly descriptive/diagnostic dashboards across surveyed works	Threshold/guideline logic referenced across cases	Mixed across surveyed cases	Many cases stop at monitoring; fragmented decision logic. (Bersani, 2022).
Maraveas & Bartzanas (2021) — IoT for optimised greenhouse environments	Emphasises microclimate & irrigation sensing; integration of low-cost sensors	Cloud-centric remote monitoring	Descriptive KPIs; efficiency metrics for climate/irrigation	Scheduling/optimisation themes discussed	Not a central focus	Operational usability + generalisability challenges flagged. (Maraveas & Bartzanas, 2021).
Hong et al. (2024) — fog	Standard sensors; tests of	Fog/edge nodes reduce	Low-latency processing; some	Faster operational decisions feasible	Enables quicker triggers (design	Strong latency results; decision

computing smart farm	HTTP/MQTT/CoAP	round-trip latency vs cloud	CV/AI classification at edge		shows potential)	rules still system-specific. (Hong et al., 2024). (MDPI)
Soussi et al. (2024) — "Smart Sensors & Smart Data"	Broad sensing landscape; phone-accessible monitoring	Cloud dashboards; mobile access	Real-time visualisation; data fusion trends	Operational insight emphasis	Alerts not primary emphasis	Highlights data quality/integration hurdles for analytics depth. (Soussi et al., 2024). (MDPI)

2.5.3 Strengths and Weaknesses of Prior Studies

Strengths:

i. **Robust sensing and pipelines** - Multi-sensor IoT stacks with cloud dashboards are well-documented, giving reliable environmental/soil monitoring at scale (Bersani et al., 2022; Maraveas, 2022; Soussi et al., 2024).
ii. **Latency-aware architectures** - Fog/edge deployments lower round-trip times and increase resilience when uplinks are unreliable (Hong et al., 2024).

Weaknesses / gaps:

i. **Operational DSS depth** - Many deployments stop at descriptive dashboards or generic alerts; composite soil indices and variable-relationship views (e.g., correlation heatmaps) are rarely integrated into day-to-day farmer workflows in peer-reviewed greenhouse DSS (Bersani et al., 2022; Soussi et al., 2024).
ii. **Real-time mobile or web push notifications** - While remote monitoring via smartphones or web is reported, unified push notifications tied to multi-parameter thresholds (soil + environment) are not consistently evidenced across greenhouse systems literature.

2.5.4 Positioning of the Present Study

Synthesizing the related works retained in this review shows a mature baseline for sensing and connectivity but uneven depth in farmer-facing decision support. Reviews of greenhouse IoT commonly report strong multi-sensor pipelines and cloud dashboards, yet many deployments plateau at descriptive monitoring with fragmented rules and limited, user-oriented advisory logic (Maraveas and Bartzanas, 2021; Bersani, Gennaro and Trobia, 2022). Edge/fog designs demonstrate latency advantages at the infrastructure layer (Hong et al., 2024), while broader “smart sensors/smart data” surveys

highlight persistent gaps in data quality, integration, and the embedding of actionable alerts for growers (Soussi et al., 2024). Against this backdrop, the present study is positioned as an operational, crop-focused DSS for Japanese melon greenhouses that bridges the space between simple dashboards and research-grade AI.

i. **End-to-end, crop-specific IoT integration** - The system instruments the greenhouse with commodity sensors aligned to melon agronomy (air temperature/humidity; soil moisture, temperature, pH, EC/TDS; light intensity) and streams data continuously to the cloud. This adheres to established device→gateway→cloud practice while tailoring parameters to a concrete cultivation context, addressing the "generic monitoring" bias noted by prior reviews (Maraveas and Bartzanas, 2021; Bersani, Gennaro and Trobia, 2022).

ii. **Mid-tier analytics for actionable interpretation** - Beyond time-series charts, the system computes a Soil Health Index (SHI) and correlation heatmaps that condense multi-variable soil–environment states into interpretable signals. This design deliberately targets the "monitoring-only" limitation—providing decision-ready summaries without the data/skill overhead of machine learning—thereby operationalising the "smart data" direction called for in recent surveys (Bersani, Gennaro and Trobia, 2022; Soussi et al., 2024).

iii. **Unified threshold-based advisory with real-time push notifications** - Calibrated multi-parameter thresholds (e.g., moisture/EC/pH bands and microclimate set-points at different growth stages) drive mobile push alerts that map directly to corrective actions (irrigation adjustment, fertigation checks, ventilation changes). Whereas related literature frequently reports remote viewing or simple warnings, comprehensive, push-style guidance tied to continuous IoT streams is inconsistent; the present study addresses this usability and responsiveness gap (Soussi et al.,

2024), while remaining compatible with edge-side checks where low latency is critical (Hong et al., 2024).

iv. **Evaluation under operational constraints and farmer usability** - The study evaluates responsiveness (sensor→dashboard latency), alert timeliness, data completeness and reliability, and dashboard usability—metrics that map directly to the project's objectives on real-time monitoring, analytics-with-visualization, and decision support. This emphasis on operational validity for a specific crop complements the largely technology-centric evaluations in the compared works (Maraveas and Bartzanas, 2021; Bersani, Gennaro and Trobia, 2022; Hong et al., 2024).

Collectively, these choices position the system as a practical, interpretable, and real-time DSS: it leverages the proven IoT/cloud backbone in the literature, incorporates mid-tier analytics that farmers can act on, and closes an identified gap in unified threshold-to-push decision support for controlled-environment Japanese melon cultivation (Soussi et al., 2024; Hong et al., 2024; Bersani, Gennaro and Trobia, 2022; Maraveas and Bartzanas, 2021).

2.5.5 Research Gap

The comparative review of recent smart-farming systems highlights clear progress in IoT-based sensing, cloud connectivity, and greenhouse monitoring dashboards. Studies such as Bersani, Gennaro and Trobia (2022) and Maraveas and Bartzanas (2021) confirm that multi-sensor deployments are technically mature and capable of providing reliable environmental and soil data streams. Fog and edge architecture has also been proposed to reduce latency and enhance responsiveness in smart farms (Hong et al., 2024). Furthermore, reviews of sensor and data practices show that mobile-accessible dashboards and cloud integration are becoming increasingly common (Soussi et al., 2024).

Despite these advances, several critical gaps remain:

i. **Descriptive monitoring without deeper analytics** - Most systems focus on dashboards and raw time-series visualization, but few integrate mid-tier analytics such as composite indices or correlation-based insights that convert raw values into interpretable indicators for day-to-day farm decisions (Soussi et al., 2024).

ii. **Limited real-time, multi-parameter notifications** - While some systems provide threshold warnings, unified push notifications that combine soil and environmental parameters in real time are largely absent, limiting their usefulness for immediate farmer response (Hong et al., 2024).

iii. **Lack of crop-specific operationalization** - Many solutions remain generic, designed for broad greenhouse contexts, without calibration for specific crops such as Japanese melon, whose growth requires finely tuned environmental and soil parameters (Maraveas and Bartzanas, 2021).

These gaps indicate the need for a smart-farming system that goes beyond generic monitoring by integrating accessible, interpretable analytics and real-time push-based decision support tailored to a specific crop. Addressing this gap is essential to ensure that IoT-enabled farming systems provide actionable knowledge rather than raw data, and that they are practical for adoption in resource-constrained greenhouse environments.

CHAPTER 3
METHODOLOGY AND WORK PLAN

3.1 Introduction

This chapter outlines the methodology and work plan adopted for the development of the smart farming system for Japanese melons. It describes the step-by-step approach taken to develop the system and covers discussion on selected software development methodology, breakdown of each development phases. Additionally, this chapter presents the work plan, detailing the project timeline, tasks, and milestones to ensure systematic and timely project execution.

3.2 System Development Methodology: Rapid Application Development (RAD)

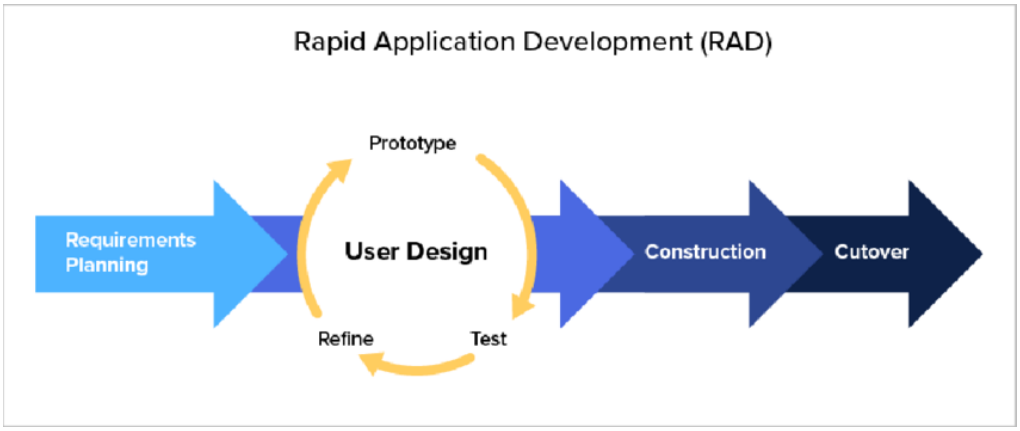


Figure 3.1: RAD methodology phases (Leonardo and Wiratama, 2023)

Rapid Application Development (RAD) was first introduced by James Martin in the 1980s while working at IBM (Rapid Application Development: RAD Methodology Roadmap, 2024). It is a software development methodology that emphasises rapid prototyping, iterative design, and continuous user involvement to deliver functional systems within shorter timeframes. Unlike traditional methodologies that rely on extensive upfront planning, RAD prioritises the early construction of working prototypes, which are refined through successive iterations based on stakeholder feedback (Information

Systems Development: Rapid Application Development | Saylor Academy, no date). This iterative nature makes RAD particularly suitable for projects where requirements may evolve or where flexibility is critical.

In the context of this smart farming system for Japanese melons, the adoption of RAD offers several advantages:

i. Accelerated prototyping and testing of IoT components, including sensors and user interfaces.
ii. Active incorporation of stakeholder feedback, particularly from the project supervisor to refine functionalities such as real-time data visualisation and automated irrigation.
iii. Enhanced adaptability to changing requirements, ensuring that the final system is not only technically sound but also aligned with practical user needs.

3.2.1 Requirements Planning Phases

The first phase of the Rapid Application Development (RAD) methodology is the Requirements Planning phase, which serves as the foundation for the smart farming system designed for Japanese melon cultivation. During this phase, the overall objectives, core functionalities, and project scope are systematically identified through close collaboration with key stakeholders, including supervisors and lecturers. A comprehensive literature review is conducted to examine existing approaches, technologies, and solutions available in the field of smart farming. This process provides critical insights into best practices, highlights how similar challenges have been addressed in prior research, and identifies opportunities for innovation in the present study.

Both functional and non-functional requirements are then elicited and documented to ensure clarity in system expectations. Based on these requirements, a Work Breakdown Structure (WBS) is developed to decompose the project into manageable tasks, thereby clarifying deliverables and organizing the overall project flow. Following this, use case diagrams are created to model the interactions between users and the system, while

accompanying use case descriptions provide detailed explanations of each interaction. To further operationalize the project plan, a Gantt Chart is employed to schedule the tasks identified in the WBS, establish milestones, allocate resources, and track progress.

To support requirement validation, conceptual mockups is developed to help visualize the intended system flow. The outputs of this phase therefore include a clearly defined system scope and objectives, validated functional and non-functional requirements, use case models, WBS, and a Gantt Chart, all of which establish a strong foundation for the rapid prototyping and iterative development to follow.

3.2.2 User Design Phase

The User Design phase builds upon the requirements identified earlier and focuses on the iterative development of software prototypes for the smart farming systems. In this phase, functional mock-ups of the web application are created and refined through multiple cycles of stakeholder feedback. The emphasis is placed on ensuring usability, system responsiveness, and the accuracy of data presentation.

Initial prototypes are developed for the frontend interface using Angular and Tailwind CSS, providing visualization of core features such as real-time sensor monitoring, threshold-based alerts, and task scheduling. To support this, the backend services are prototyped using Spring Boot, enabling the integration of Supabase for database management and authentication. Additionally, static images of Grafana dashboards are embedded to present analysis graph.

Throughout this phase, regular feedback is collected from the project supervisor to evaluate usability, clarity of data visualization, and intuitiveness of the overall interface. Identified issues such limited visual clarity in charts, or overly complex navigation are addressed in subsequent iterations.

Prototyping continues until the system achieves a level of stability and usability that aligns with both functional requirements and user expectations.

By adopting this iterative, user-centered approach, the User Design phase ensures that potential challenges are identified and resolved early in the development lifecycle. This reduces risks, strengthens the reliability of the system, and provides a solid foundation for the subsequent Rapid Construction phase.

3.2.3 Construction Phase

The Rapid Construction phase is the stage where the actual development of the smart farming system is undertaken. Building upon the validated prototypes from the User Design phase, this stage emphasizes the iterative coding, integration, and refinement of the software components. Development activities are carried out in short cycles, enabling quick incorporation of feedback and timely resolution of issues.

Key activities in this phase include the implementation of the system architecture, integration of the database and services, and the development of user interfaces into a cohesive application. Each software module is developed and tested incrementally to ensure that it functions correctly in isolation before being combined with other modules. Testing activities are embedded throughout the phase, comprising unit testing, integration testing and system testing. These activities collectively validate the accuracy, stability, and performance of the system.

A defining feature of this phase is its iterative nature. Any shortcomings identified during testing or stakeholder feedback sessions are promptly addressed in subsequent development cycles. This ensures that the system progressively evolves towards its intended quality, functionality, and usability. Continuous consultation with the supervisor further ensures that the development process remains aligned with project objectives and technical requirements.

3.2.4 Cutover Phase

The Cutover phase is the final stage of the RAD methodology and focuses on transitioning the developed system into an operational environment. This phase includes final testing, documentation, and presentation. Development work carried out in earlier phases is consolidated into a fully functional application that is ready for use and assessment.

Key activities in this phase include the deployment of the backend services onto a cloud platform and the hosting of the Angular-based frontend for seamless access across devices. Comprehensive integration and acceptance testing are conducted to verify that data flows smoothly through the system, ensuring reliability, stability, and usability. A demonstration session is also organized with the project supervisor to present the system's capabilities and gather final feedback. Any residual issues, such as usability concerns or minor bugs, are promptly addressed prior to submission.

In parallel, thorough documentation is prepared to support both academic evaluation and potential future system adoption. This includes the final FYP report, a user manual, updated architecture diagrams, and a reflection report outlining challenges encountered, lessons learned, and recommendations for future improvement.

3.3 Work Breakdown Structure (WBS)

1. Rapid Planning and requirement gathering
    - 1.1 Problem identification
        - 1.1.1 Identify current challenges
        - 1.1.2 Analyse limitations of existing solutions
        - 1.1.3 Define the real-world need for a smart farming system
    - 1.2 Define objectives and scope
        - 1.2.1 Determine expected project outcomes
    - 1.3 Literature Review
        - 1.3.1 Study existing musk melon planting techniques
        - 1.3.2 Study data collection, storage and visualisation techniques

1.3.3 Study existing smart faming websites and system

1.4 Define methodology

1.4.1 Survey and compare existing software development methodology

1.4.2 Choose software development methodology

1.4.2.1 Define RAD prototype iteration and goals

1.5 Work planning

1.5.1 Create project timeline with Gantt chart

1.5.2 Define key deliverables and milestones

1.5.3 Assign tentative deadlines for each task

1.6 Project specification

1.6.1 Define system requirements

1.6.2 Specify software tech stack

1.7 Initial proposal document

1.7.1 Prepare and submit proposal document

1.7.2 Prepare proposal presentation slides

1.7.3 Conduct initial presentation

2. Prototype 1: Sensor Integration & Data Acquisition

2.1 Select sensors and microcontroller

2.1.1 Choose sensors

2.1.2 Select suitable microcontroller

2.2 Define wiring and connection layout

2.3 Develop data acquisition script

2.3.1 Write microcontroller script for data reading

2.4 Integrate sensors and microcontroller

2.5 Set up IoT gateway for data transfer

2.5.1 Configure Wi-Fi module

2.5.2 Send data to temporary cloud endpoint

2.6 User review and feedback

2.6.1 Demonstrate sensor system to users or supervisors

2.6.2 Collect improvement suggestions

2.7 Refine based on feedback

2.7.1 Adjust scripts or hardware if needed

- 2.7.2 Finalize prototype 1 design

3. Prototype 2: Dashboard & Data Visualization
   - 3.1 Design simple website for dashboard
   - 3.2 Set up frontend and backend projects
     - 3.2.1 Initialize Angular project with Tailwind and Ng Zorro
     - 3.2.2 Set up Spring Boot backend
   - 3.3 Implement RESTful APIs and database connection
     - 3.3.1 Create CRUD endpoints for sensor data
     - 3.3.2 Integrate PostgreSQL database
   - 3.4 Build dashboard for environmental data
   - 3.5 Conduct usability testing with users
   - 3.6 Gather feedback and iterate improvements
     - 3.6.1 Modify UI elements based on feedback
4. Prototype 3: Notification & Report System
   - 4.1 Define report generation structure
     - 4.1.1 Determine daily, weekly, and monthly summaries
   - 4.2 Implement notification feature
     - 4.2.1 Define thresholds for each parameter
     - 4.2.2 Set up notification alert mechanisms
   - 4.3 Design report UI and export options
   - 4.4 User testing and feedback collection
   - 4.5 Refine and finalize modules
5. Continuous Cloud Integration & Deployment
   - 5.1 Select cloud provider
     - 5.1.1 Compare AWS, Firebase, and Azure
     - 5.1.2 Choose provider based on needs and free tier
   - 5.2 Set up cloud database and hosting
     - 5.2.1 Deploy database instance
     - 5.2.2 Create hosting environment for backend
   - 5.3 Containerize and deploy backend
     - 5.3.1 Push to cloud and test API endpoints
   - 5.4 Deploy frontend to cloud
     - 5.4.1 Upload Angular build to cloud storage/CDN

5.4.2 Configure DNS or Firebase Hosting

5.5 Implement secure API access

6. Iterative Testing & Feedback

6.1 Unit testing

6.2 Integration testing

6.3 System testing

6.4 Usability testing

6.5 User acceptance testing (UAT)

7. Documentation & Final Report

7.1 Prepare poster and final presentation slides

7.2 Set up demo environment

8. Project Closure & Reflection

8.1 Final deployment

8.2 Supervisor review and feedback

8.3 Reflection and lessons learned

8.3.1 Summarize challenges and resolutions

3.4 Gantt Chart

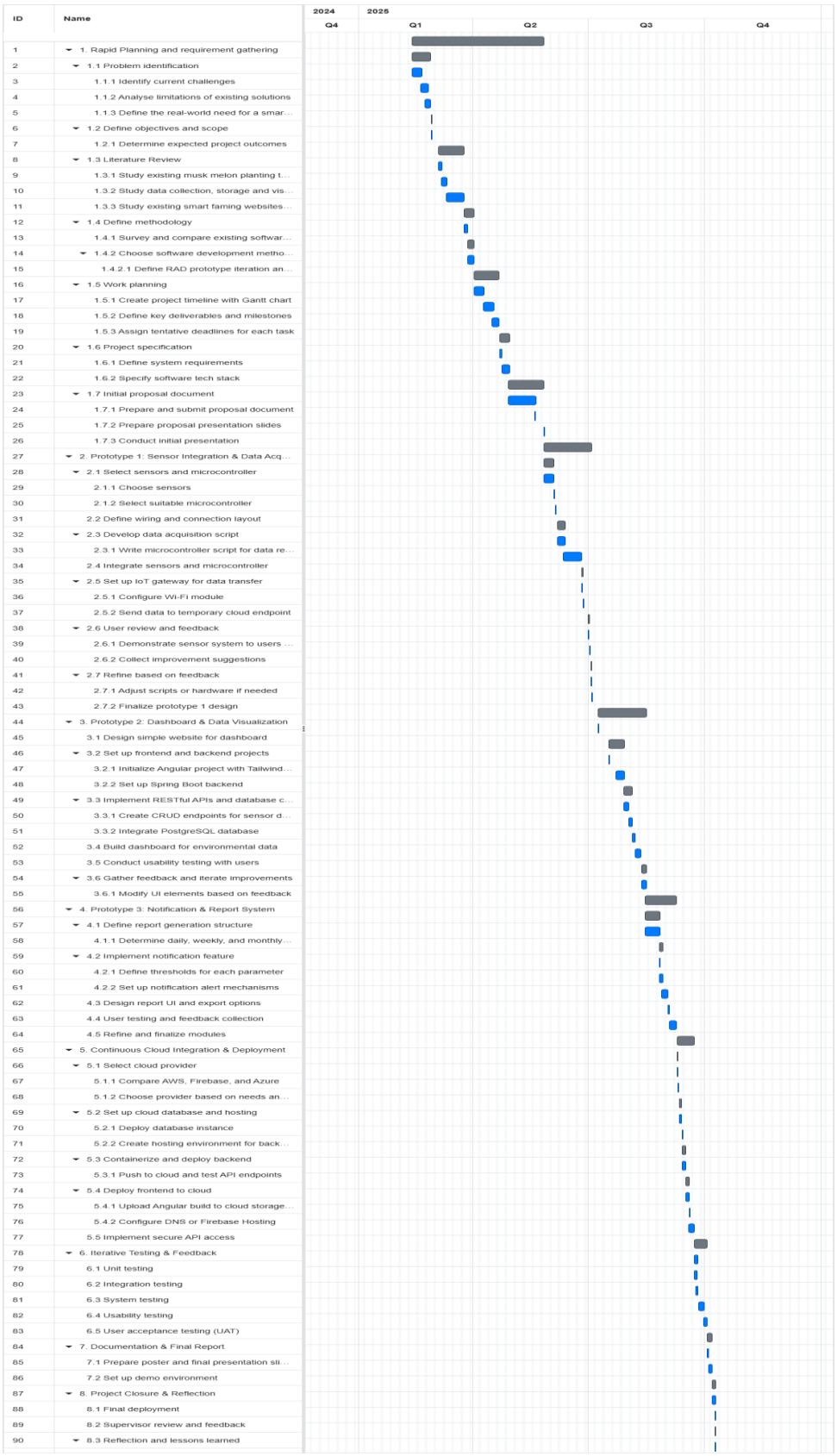
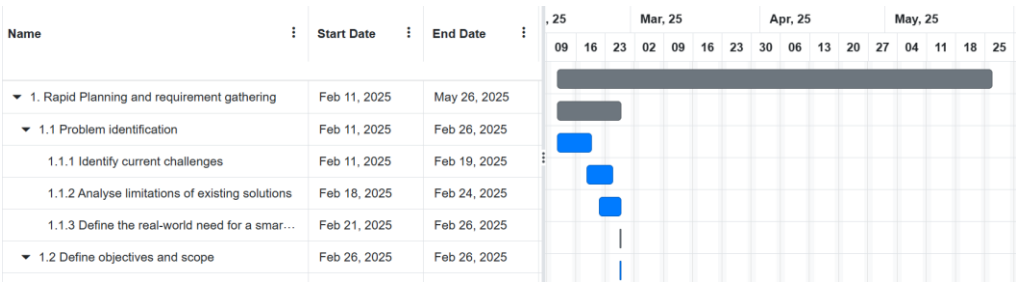
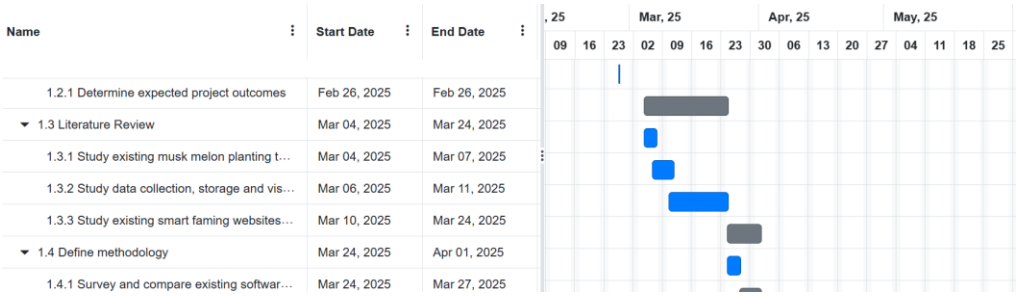


Figure 3.2: Gantt Chart overview



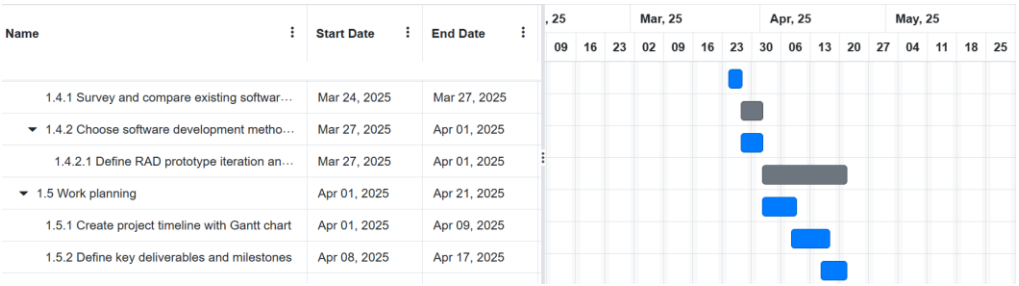
Name	Start Date	End Date
1. Rapid Planning and requirement gathering	Feb 11, 2025	May 26, 2025
1.1 Problem identification	Feb 11, 2025	Feb 26, 2025
1.1.1 Identify current challenges	Feb 11, 2025	Feb 19, 2025
1.1.2 Analyse limitations of existing solutions	Feb 18, 2025	Feb 24, 2025
1.1.3 Define the real-world need for a smar…	Feb 21, 2025	Feb 26, 2025
1.2 Define objectives and scope	Feb 26, 2025	Feb 26, 2025

Figure 3.3: Gantt Chart detail view 1



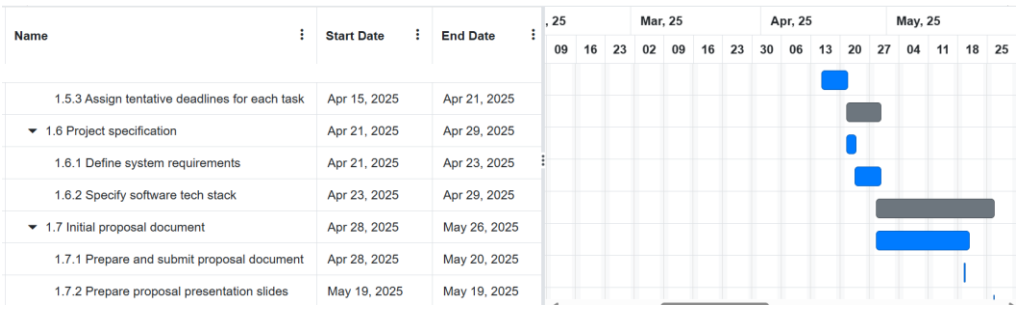
Name	Start Date	End Date
1.2.1 Determine expected project outcomes	Feb 26, 2025	Feb 26, 2025
1.3 Literature Review	Mar 04, 2025	Mar 24, 2025
1.3.1 Study existing musk melon planting t…	Mar 04, 2025	Mar 07, 2025
1.3.2 Study data collection, storage and vis…	Mar 06, 2025	Mar 11, 2025
1.3.3 Study existing smart faming websites…	Mar 10, 2025	Mar 24, 2025
1.4 Define methodology	Mar 24, 2025	Apr 01, 2025
1.4.1 Survey and compare existing softwar…	Mar 24, 2025	Mar 27, 2025

Figure 3.4: Gantt Chart detail view 2



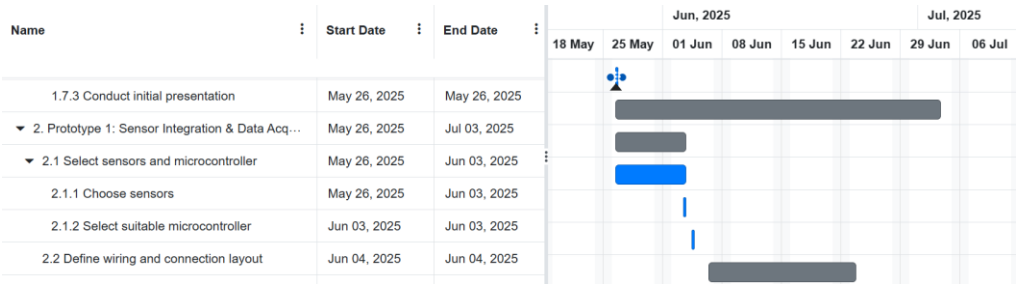
Name	Start Date	End Date
1.4.1 Survey and compare existing softwar…	Mar 24, 2025	Mar 27, 2025
1.4.2 Choose software development metho…	Mar 27, 2025	Apr 01, 2025
1.4.2.1 Define RAD prototype iteration an…	Mar 27, 2025	Apr 01, 2025
1.5 Work planning	Apr 01, 2025	Apr 21, 2025
1.5.1 Create project timeline with Gantt chart	Apr 01, 2025	Apr 09, 2025
1.5.2 Define key deliverables and milestones	Apr 08, 2025	Apr 17, 2025

Figure 3.5: Gantt Chart detail view 3



Name	Start Date	End Date
1.5.3 Assign tentative deadlines for each task	Apr 15, 2025	Apr 21, 2025
1.6 Project specification	Apr 21, 2025	Apr 29, 2025
1.6.1 Define system requirements	Apr 21, 2025	Apr 23, 2025
1.6.2 Specify software tech stack	Apr 23, 2025	Apr 29, 2025
1.7 Initial proposal document	Apr 28, 2025	May 26, 2025
1.7.1 Prepare and submit proposal document	Apr 28, 2025	May 20, 2025
1.7.2 Prepare proposal presentation slides	May 19, 2025	May 19, 2025

Figure 3.6: Gantt Chart detail view 4



Name	Start Date	End Date
1.7.3 Conduct initial presentation	May 26, 2025	May 26, 2025
2. Prototype 1: Sensor Integration & Data Acq…	May 26, 2025	Jul 03, 2025
2.1 Select sensors and microcontroller	May 26, 2025	Jun 03, 2025
2.1.1 Choose sensors	May 26, 2025	Jun 03, 2025
2.1.2 Select suitable microcontroller	Jun 03, 2025	Jun 03, 2025
2.2 Define wiring and connection layout	Jun 04, 2025	Jun 04, 2025

Figure 3.7: Gantt Chart detail view 5

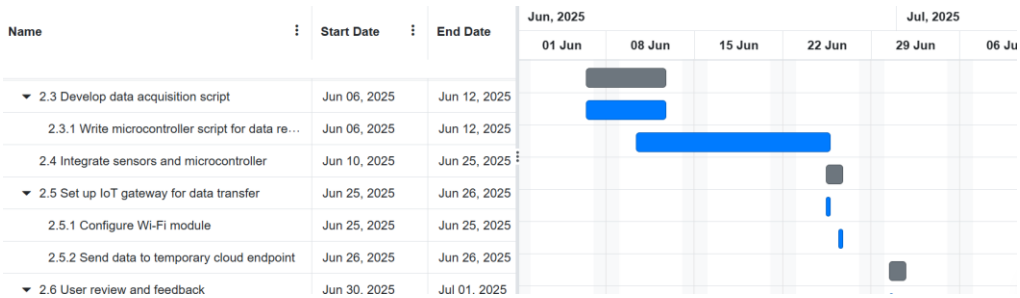


Figure 3.8: Gantt Chart detail view 6

Figure 3.9: Gantt Chart detail view 7

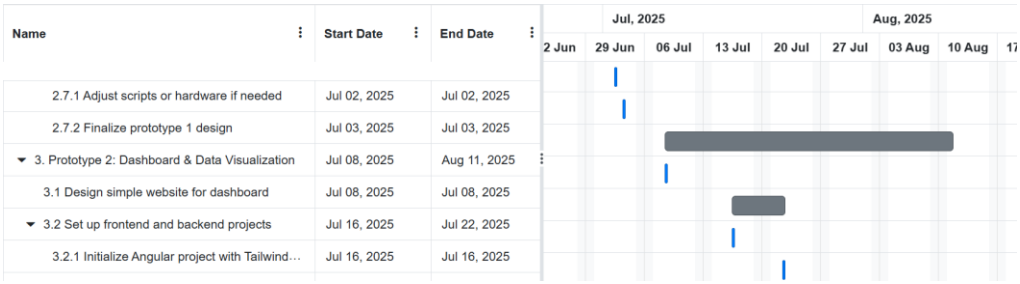


Figure 3.10: Gantt Chart detail view 8

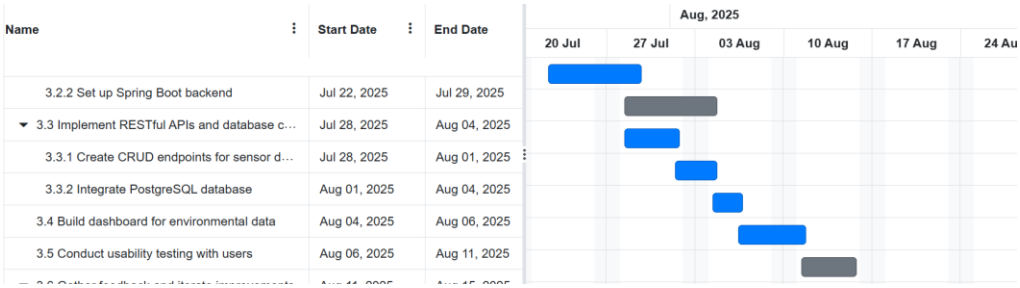


Figure 3.11: Gantt Chart detail view 9

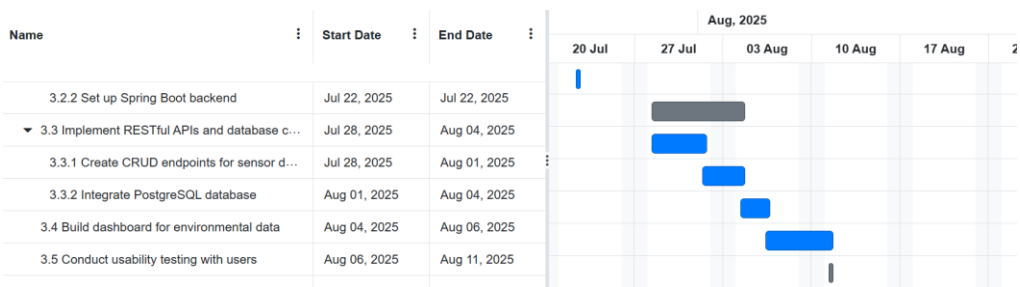


Figure 3.12: Gantt Chart detail view 10

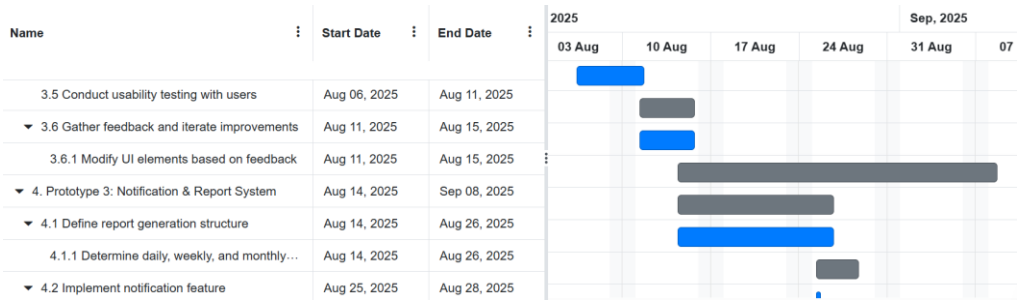


Figure 3.13: Gantt Chart detail view 11

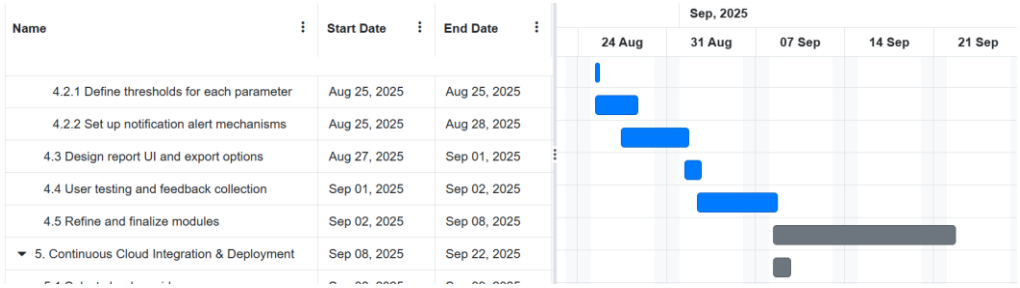


Figure 3.14: Gantt Chart detail view 12

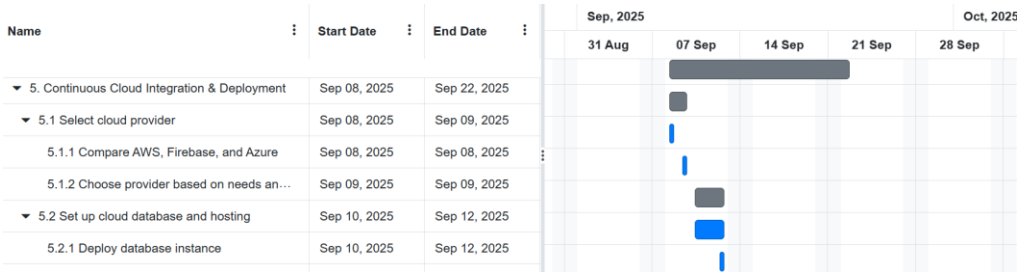


Figure 3.15: Gantt Chart detail view 13

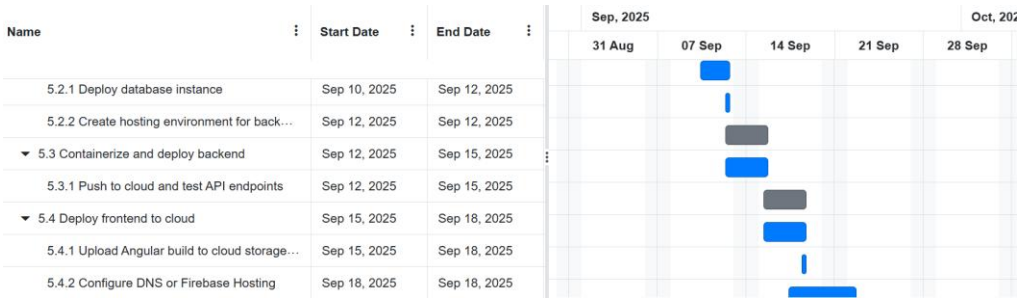


Figure 3.16: Gantt Chart detail view 14

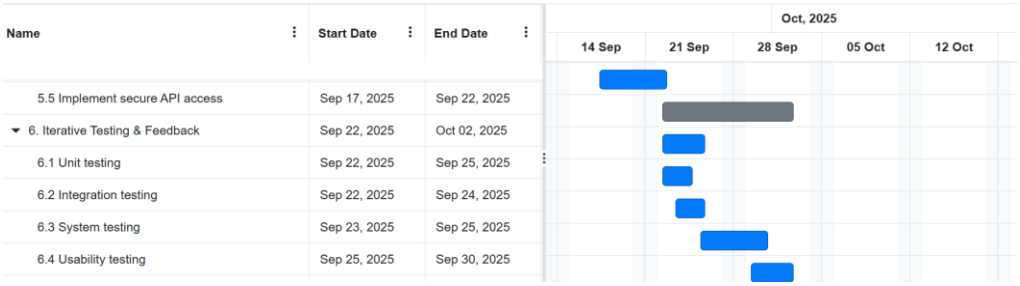


Figure 3.17: Gantt Chart detail view 15

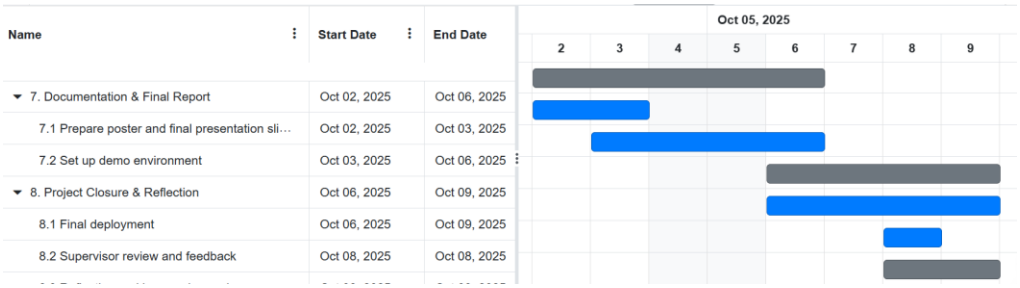


Figure 3.18: Gantt Chart detail view 16

Figure 3.19: Gantt Chart detail view 17

3.5 Development Tools

The development of the smart farming system for Japanese melon cultivation was supported by a range of software frameworks, cloud platforms, and auxiliary tools. Each tool was selected to address specific requirements of the system, including backend integration, frontend design, database management, analytics visualisation, and notification delivery. The following subsections describe the major tools employed and their roles in the project.

3.5.1 Backend Development Tools

The backend of the system was implemented using Spring Boot, a Java-based framework that simplifies the development of scalable and modular

applications. Spring Boot provided the foundation for building RESTful API endpoints, which handle the retrieval, storage, and processing of IoT sensor data. It also facilitated the implementation of the threshold evaluation and suggestion mechanism, ensuring that incoming data could be validated against predefined parameters. To streamline the build and deployment process, Gradle was used as the primary build automation tool. Gradle managed dependencies, compiled the project, and automated testing, which collectively improved the efficiency and consistency of backend development.

3.5.2 Frontend Development Tools

The frontend was developed using the Angular framework, chosen for its ability to support responsive, dynamic, and component-driven user interfaces. Angular enabled the creation of an interactive dashboard through which users could monitor real-time sensor data, view graphical trends, and access decision-support features. To enhance the visual design of the application, Tailwind CSS was integrated, allowing the implementation of a clean and consistent interface while maintaining flexibility in styling. In addition, Ng Zorro Ant Design was adopted as a UI component library, which accelerated development by providing ready-made, professional-grade interface components, ensuring both functionality and consistency in user interaction.

3.5.3 Database and Cloud Tools

Data storage and management were achieved through Supabase, an open-source platform built on PostgreSQL. Supabase served as the primary database for storing sensor readings, user profiles, and system configurations. It also provided built-in authentication services, simplifying the management of user access. To accommodate the limitations of the IoT hardware in transmitting secure HTTPS requests, ThingSpeak was introduced as an intermediary IoT gateway. Sensor data were first uploaded to ThingSpeak using HTTP protocols and later synchronised into Supabase through an Extract, Transform, Load (ETL) process. This integration ensured reliable and secure storage of real-time data within the central database.

3.5.4 Data Analytics and Visualisation Tools

For advanced data analytics and visualisation, Grafana was integrated into the system. Grafana provided interactive dashboards for time-series visualisation of environmental parameters, correlation heatmaps to identify relationships between variables, and a Soil Health Index (SHI) to present composite metrics. By embedding Grafana dashboards into the Angular application, the system offered both real-time monitoring and historical trend analysis, thereby supporting informed decision-making for melon cultivation.

3.5.5 Notification and Messaging Tools

To support real-time communication with users, Firebase Cloud Messaging (FCM) was employed as the push notification service. FCM enabled the system to send alerts whenever sensor readings exceeded the warning or critical thresholds defined in the parameter configuration. This ensured that users received timely updates, even when not actively logged into the system, thereby enhancing the reliability of the decision-support mechanism.

3.5.6 Project Management and Documentation Tools

Several additional tools were employed to facilitate project management and documentation. Git, together with GitHub, was used for version control, enabling systematic tracking of source code changes and collaborative development. For technical documentation and reporting, Microsoft Word was used to prepare the Final Year Project report, while Draw.io was employed to design system architecture diagrams, entity-relationship diagrams (ERD), and data flow diagrams (DFD). These tools supported the systematic organisation of development activities and ensured the production of professional documentation to accompany the implemented system.

CHAPTER 4
PROJECT SPECIFICATION

4.1 Introduction

This chapter presents the detailed specifications of the smart farming system developed for Japanese melon cultivation, serving as the foundation for the subsequent design and implementation. It begins by outlining the functional and non-functional requirements, which define the system's expected capabilities and quality attributes. To further specify the system's behaviour, use case models are introduced, consisting of a use case diagram that illustrates user interactions with the system and accompanying descriptions that elaborate on the roles, actions, and flows involved. In addition, conceptual prototypes are provided to visualise the initial interface design and demonstrate key functionalities, allowing early validation of system requirements. Collectively, these specifications establish a comprehensive blueprint that ensures the objectives, requirements, and user interactions of the smart farming system are well-defined prior to detailed design and development.

4.2 System Requirements

4.2.1 Functional Requirements

The smart farming system is designed to support various essential functionalities that contribute to efficient farm management, particularly for melon cultivation. Each function plays a critical role in ensuring data-driven decision-making, resource optimization, and task tracking. The following sections outline and elaborate on the core functional requirements of the system:

i. User authentication module

The system shall provide secure authentication to manage user access. It shall allow users to log in using their registered credentials, specifically an email address and password, which must be validated during authentication. The system shall also support password reset functionality in cases where users

forget their credentials. To maintain security, only administrators shall be permitted to register new users by using the user's email.

ii. User Profile Management Module

The system shall provide functionality for users to manage their personal profiles. Users shall be able to view and update basic information, including name, email, and contact details. Administrators shall have the ability to manage user roles to ensure that appropriate access levels are maintained. The system shall also allow the storage of device tokens linked to user accounts, enabling personalised push notifications through the notification service.

iii. Farming Activity Management Module

The system shall provide functionality to manage farming activities such as irrigation, fertilisation, and weed removal. Users shall be able to schedule activities for tracking and analysis. Integration with external weather APIs should be included to display forecasts, such as rain or extreme heat. Users shall be able to view, add, update, and delete farming activities.

iv. Sensor Data Collection, Storage, and Visualisation Module

The system shall collect environmental readings, including air temperature, humidity, soil moisture, soil pH, soil conductivity, soil temperature, soil TDS, and light intensity, from IoT sources. All data shall be securely stored in a cloud-based database and visualised through interactive charts and graphs to support monitoring. Alerts shall be triggered when sensor readings exceed defined thresholds, and these alerts shall be delivered via notifications.

v. Sensor Threshold Configuration Module

The system shall provide users with the ability to configure threshold values for each monitored parameter, including minimum, maximum, and warning levels. Users shall be able to view these thresholds to understand optimal ranges for cultivation. The system shall use the configured thresholds to generate automated alerts and decision-support suggestions whenever parameter values fall outside the acceptable limits. The configuration records

shall be stored in the database and updateable by authorised users, ensuring flexibility in adapting the system to changing farming conditions.

4.2.2 Non-Functional Requirements

The non-functional requirements define the quality attributes and operational constraints of the smart farming system. These requirements ensure that the system not only fulfils its intended functionalities but also delivers performance, reliability, and usability standards expected for real-world application.

i. Performance Requirements

The system shall process and update environmental sensor data with minimal delay to support near real-time monitoring. The dashboard shall load within acceptable time limits, with the Largest Contentful Paint (LCP) metric targeted at less than two seconds to provide a smooth user experience.

ii. Reliability Requirements

The system shall achieve a minimum of 95% data transmission success rate from the IoT gateway to the cloud database to ensure data completeness. In case of temporary network disruptions, mechanisms shall ensure that sensor data are synchronised once connectivity is restored. Critical functionalities such as threshold monitoring and notification delivery shall remain consistently available to guarantee continuous system reliability.

iii. Usability Requirements

The system shall provide a user-friendly and intuitive interface. Dashboards and charts shall employ clear visualisation techniques with appropriate labelling to assist interpretation. User interactions, such as scheduling tasks and configuring thresholds, shall be designed to require minimal training, thereby supporting adoption by non-technical users such as farmers.

iv. Security Requirements

The system shall enforce secure authentication through user credentials, with role-based access control to differentiate privileges between administrators and standard users. Sensitive data such as sensor readings, user details, and configuration records shall be stored securely in the cloud database.

4.3 Use Case Diagram

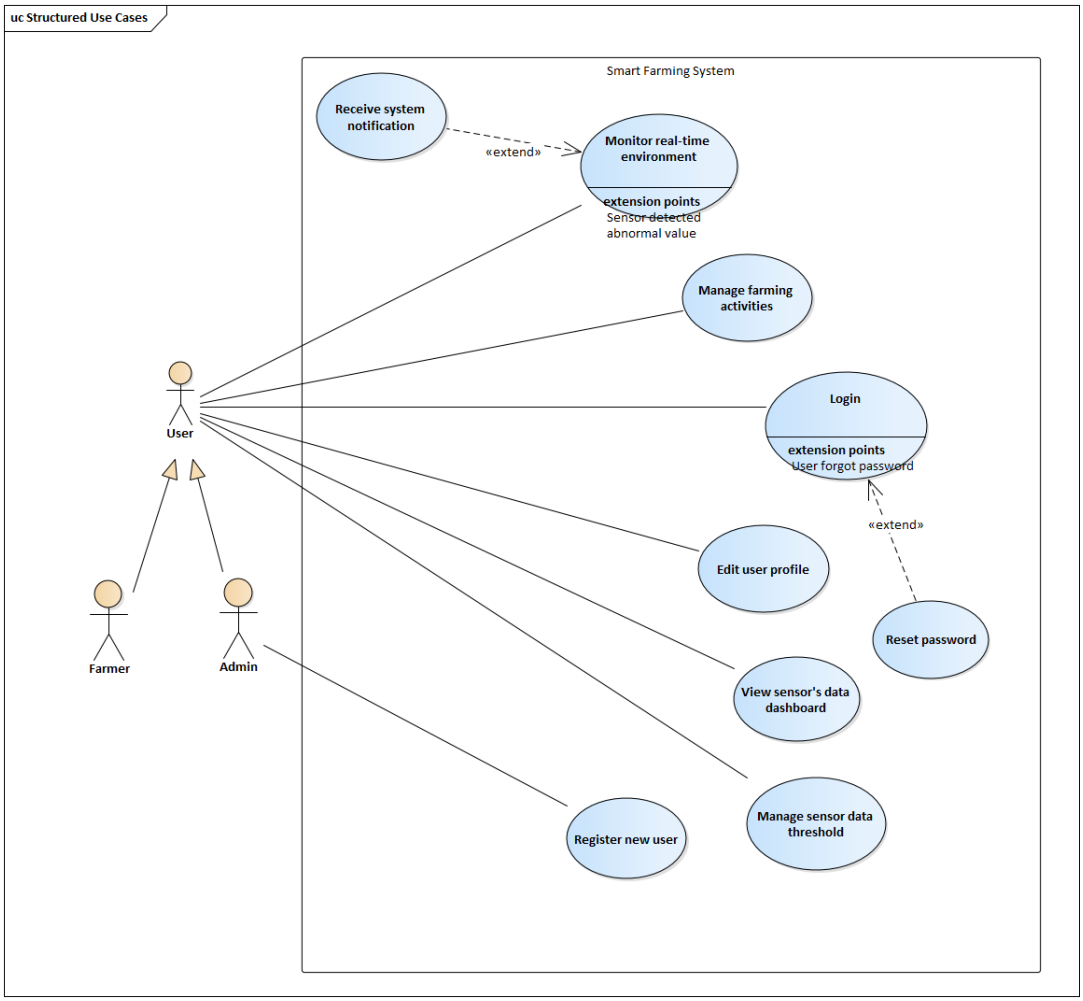


Figure 4.1: Use Case Diagram

4.4 Use Case Description

Use Case Name: User login	ID: UC001	Significance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User: Needs to access their personal account on the smart farming system by		

logging in.

Brief Description:

This use case outlines the process of a registered user signing into the smart farming platform.

Trigger: User navigates to the smart farming website.

Relationships:

Association: User

Include: N/A

Extend: Password Recovery

Generalization: N/A

Normal Flow of Events:

1. The user visits the smart farming website.
2. The website presents a login form.
3. The user enters valid email and password. If they cannot remember their password, the S-1 Password Recovery sub-flow is initiated.
4. The user pressed login button.
5. The system validates the provided credentials. If they are not correct, proceed to Exception Flow 6.1.

Sub-flows:

S-1 Recover account

1. The user selects the “Forgot Password” option on the login page.
2. The system shows the password reset interface.
3. The user provides their registered email address and submits the request.
4. The application generates a password reset link and sends to user’s email address.
5. The user accesses the reset link and sets a new password.
6. Return to step 2 of the main flow.

Alternate/Exceptional Flows:

6.1 Invalid Credentials

1. The system displays an error notification indicating incorrect login information.
2. The user is prompted to reattempt the login process.

<table>
<tr><td>Use Case Name:
Account Registration</td><td>ID: UC002</td><td>Significance Level: High</td></tr>
<tr><td colspan="2">Primary Actor: Admin</td><td>Use Case Type: Detail, Essential</td></tr>
<tr><td colspan="3">Stakeholders and Interests:
Admin: Responsible for creating accounts so that new users can gain access to the smart farming system.</td></tr>
<tr><td colspan="3">Brief Description:
The use case describes how the admin can register a new account for a new user.</td></tr>
<tr><td colspan="3">Trigger: Admin wants to add new user to the smart farming system.</td></tr>
<tr><td colspan="3">Relationships:
Association: Admin
Include: N/A
Extend: N/A
Generalization: N/A</td></tr>
<tr><td colspan="3">Normal Flow of Events:
1. The administrator navigates to the “Add New User” page from the side menu.
2. The administrator provides the new user’s email address on the registration form.
3. The administrator submits the registration details.
4. The system checks the validity of the input. <u>If the email is incorrectly formatted or already exists in the database, continue to Exception Flow 4.1.</u>
5. When the data is valid, the system displays a success message and sends a registration link to the new user’s email.</td></tr>
<tr><td colspan="3">Sub-flows:
-</td></tr>
<tr><td colspan="3">Alternate/Exceptional Flows:
4.1 Invalid or Duplicate Email Entry
1. The system shows an error indicating the email format is invalid or the address is already registered.</td></tr>
</table>

2. The administrator is asked to repeat the registration process with correct information.

Use Case Name: Edit user profile	ID: UC003	Significance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: Admin and user: Require the ability to modify and maintain accurate profile information.		
Brief Description: This use case explains how a registered user can update their profile details such as username, email address, and password.		
Trigger: The process starts when the user decides to change their profile information.		
Relationships: Association: User Include: N/A Extend: N/A Generalization: N/A		
Normal Flow of Events: 1. The user opens the profile page and selects the “Edit” option. 2. The user enters the updated information (username, email, or password). 3. The user confirms and submits the changes. If the input is in an invalid format or already in use, proceed to Exception Flow 3.1. 4. Once validated, the system shows a success notification and displays the revised profile information.		
Sub-flows:		
Alternate/Exceptional Flows: **3.1 Duplicated or Invalid Profile Information** 1. The system notifies the user with an error message indicating invalid or already existing credentials. 2. The system prompts user to enter their credentials again.		

<table>
<tr><td>Use Case Name:
Real-Time
Environmental
Monitoring</td><td colspan="2">ID: UC004</td><td>Significance Level: High</td></tr>
<tr><td colspan="2">Primary Actor: User</td><td colspan="2">Use Case Type: Detail, Essential</td></tr>
<tr><td colspan="4">Stakeholders and Interests:
User: Requires continuous access to live environmental data from the melon farm.</td></tr>
<tr><td colspan="4">Brief Description:
This use case outlines how a user can track environmental parameters of the farm in real time.</td></tr>
<tr><td colspan="4">Trigger: The process begins when the user chooses to monitor the farm’s environmental conditions.</td></tr>
<tr><td colspan="4">Relationships:
Association: User
Include: N/A
Extend: Receive push notifications.
Generalization: N/A</td></tr>
<tr><td colspan="4">Normal Flow of Events:
1. The user navigates to monitor sensor data page.
2. The system presents live readings of environmental factors, including air temperature, air humidity, soil temperature, soil moisture, soil pH, soil TDS, soil conductivity, and light intensity. If any parameter exceeds its defined threshold, Sub-flow S-1 is triggered. <u>If no sensor readings are available or a database error occurs, Exception Flow 2.1 is executed</u>.</td></tr>
<tr><td colspan="4">Sub-flows:
S-1 Receive push notifications
1. The user is notified through push notifications when sensor data values surpass the defined threshold.</td></tr>
<tr><td colspan="4">Alternate/Exceptional Flows:
2.1 Missing Data or Database Failure
1. The system displays a message to the user stating “No data</td></tr>
</table>

available.”

Use Case Name: View sensor’s data dashboard	ID: UC005	Significance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User: View sensor’s data dashboard.		
Brief Description: The user accesses the dashboard page to view real-time and historical sensor data including environment and sensor’s health data, which is presented in graphical format.		
Trigger: Users want to view sensor’s data in graphical format.		
Relationships: Association: User Include: N/A Extend: N/A Generalization: N/A		
Normal Flow of Events: 1. User navigates to the Dashboard page. 2. System retrieves the latest sensor data. If the sensor’s data is unavailable, perform exceptional flow 2.1 3. User can view the real-time, historical data in graphs and analysed data, gauge or charts on the dashboard.		
Sub-flows: **N/A**		
Alternate/Exceptional Flows: **Exceptional Flow 2.1: Data unavailable** 1. If sensor data cannot be retrieved, the system displays an error message indicating no data is available.		

Use Case Name:	ID: UC006	Significance Level: High

Manage farming activities		
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User: Manage melon farm's farming activities.		
Brief Description: This use case description describes how user manages farming activities.		
Trigger: Users want to manage farming activities.		
Relationships: Association: User Include: N/A Extend: N/A Generalization: N/A		
Normal Flow of Events: 1. User navigates to the farming activities page. 2. The system displays all scheduled activities in calendar view. 3. If users select to create or edit an activity, sub-flows S-1 will be performed. 4. If users want to delete an activity, sub-flows S-2 will be performed. 5. System saves or updates the activity in the database. If input information is invalid, exceptional flow 5.1 will be performed. 6. System displays the updated list of activities to the user.		
Sub-flows: **S-1 Creating and Editing Farming Activity** 1. User fills in or modifies necessary fields including activity title, description, date, time and assigned personnel. 2. Users click "Save" button to save the new or modified activity. 3. Back to main flow step 5. **S-2 Delete Farming Activity** 1. System pops up a confirmation window with "Confirm" and "Cancel" options.		

2. If user confirms the deletion, the system removes the selected activity from the database. 3. If user cancels, the system closes the confirmation window without making changes. 4. Back to main flow step 5.
Alternate/Exceptional Flows: **5.1 Invalid Input Information** 1. System displays an "Invalid input" message if there is missing required fields or incorrect date format.

Use Case Name: Register new user	ID: UC007	Significance Level: High
Primary Actor: Admin	Use Case Type: Detail, Essential	
Stakeholders and Interests: Admin: Add new user to the smart farming system.		
Brief Description: This use case description describes how admin add new user to the system.		
Trigger: Admin wants to add new user to the system.		
Relationships: Association: Admin Include: N/A Extend: N/A Generalization: N/A		
Normal Flow of Events: 1. Admins navigate to add new user page. 2. Admins enter the new user ‘s email and press confirm button, if user’s email are invalid, Exceptional Flow 2.1 will be performed. 3. User will receive an email link to register the smart farming system.		
Sub-flows: **N/A**		

Alternate/Exceptional Flows:

1.1 Invalid email entry

1. The system displays an error message stating “Invalid user credentials.”
2. The administrator is prompted to re-enter correct details and attempt the process again.

Use Case Name: Manage Sensor Data Thresholds	ID: UC008	Significance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	

Stakeholders and Interests:

Needs to configure threshold values and corresponding suggestions for each sensor parameter to ensure accurate system monitoring and recommendations.

Brief Description:

This use case describes how an user manages sensor data thresholds by adding or updating values such as optimal ranges, warning limits, and recommendation messages.

Trigger: The user needs to configure or adjust threshold values for one or more sensor parameters.

Relationships:

Association: User

Include: N/A

Extend: N/A

Generalization: N/A

Normal Flow of Events:

1. The user navigates to the threshold management page.
2. The user selects a parameter.
3. The user enters or updates values for optimal minimum, optimal maximum, warning minimum, and warning maximum, along with corresponding suggestion messages. If input are invalid, Exceptional Flow 3.1 will be performed.
4. The user confirms the update.
5. A confirmation message is displayed, and the updated thresholds are

applied to subsequent evaluations.
Sub-flows: **N/A**
Alternate/Exceptional Flows: **3.1 Invalid user's credentials** The system displays an error message prompting the administrator to re-enter valid thresholds.

4.5 Conceptual Prototype

This section focuses on the prototype of the smart farming website. The prototype demonstrates the core functionalities of the web system and serves as an early version for testing and further refinement based on user feedback.

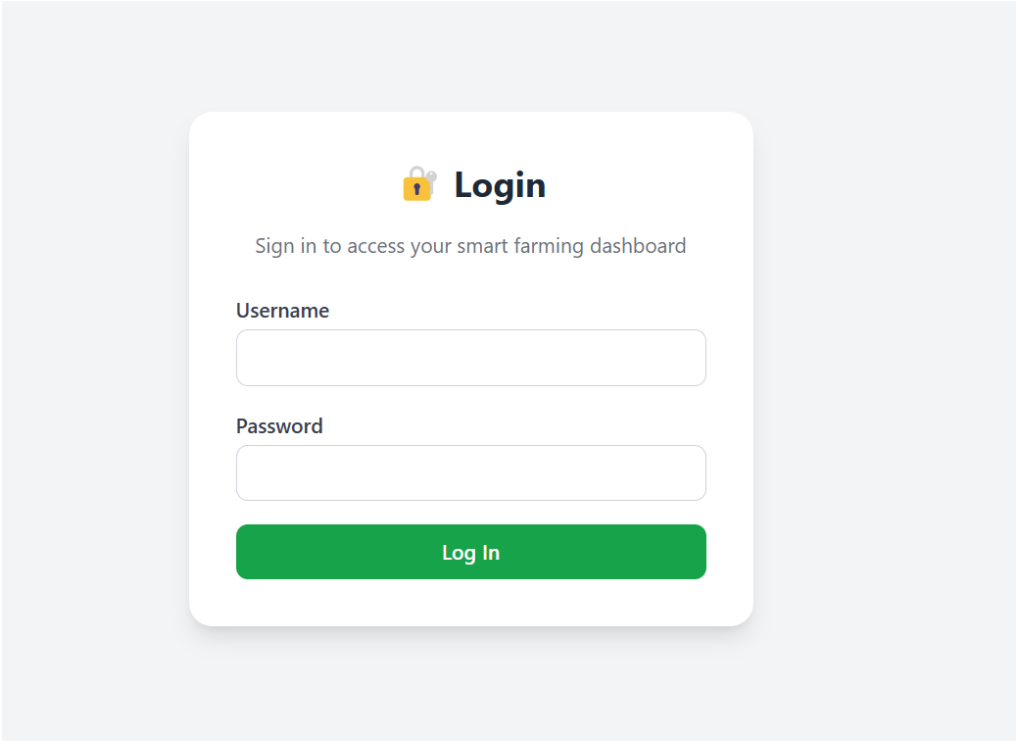


Figure 4.2: Prototype - User login interface

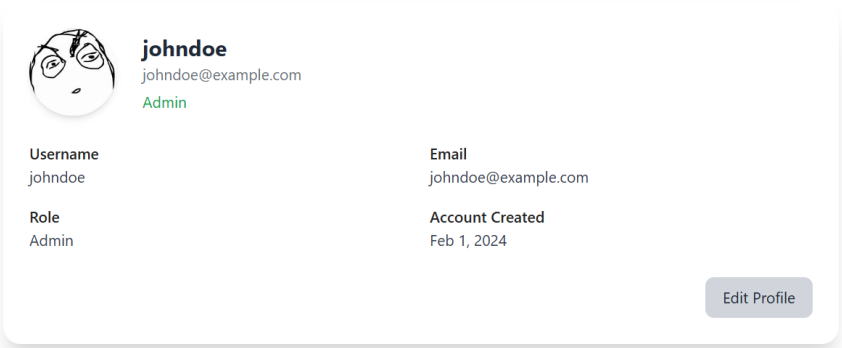


Figure 4.3: Prototype - User profile page

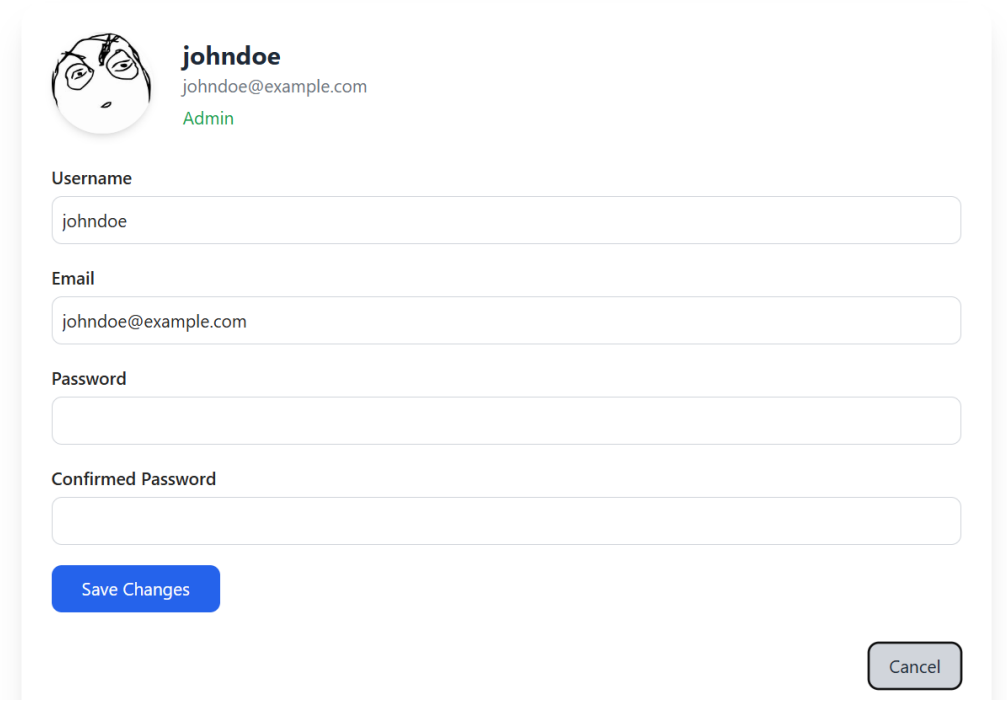


Figure 4.4: Prototype - Edit user credentials interface

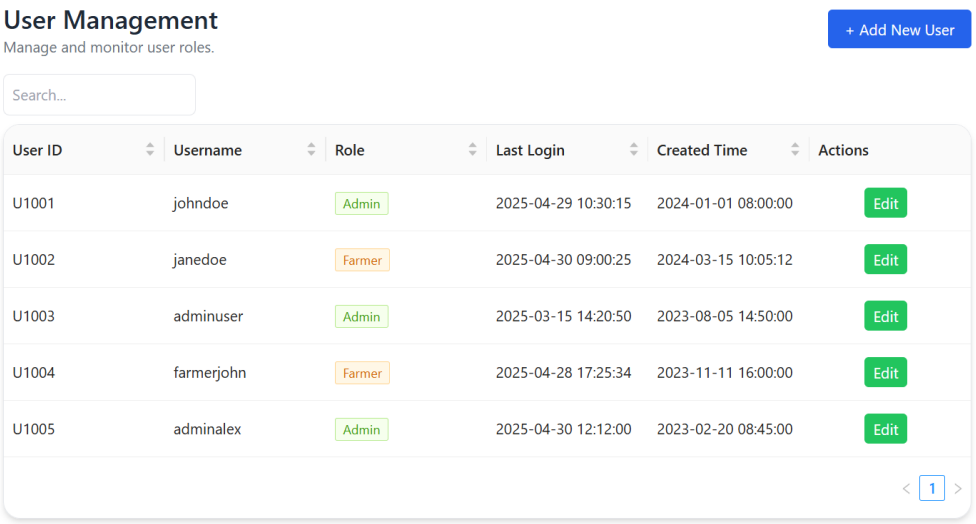


Figure 4.5: Prototype (Admin view) - User management interface

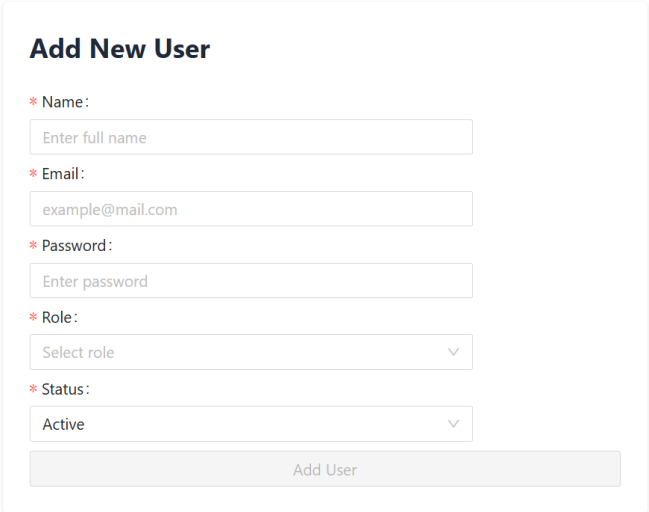


Figure 4.6: Prototype (Admin) - Add new user interface

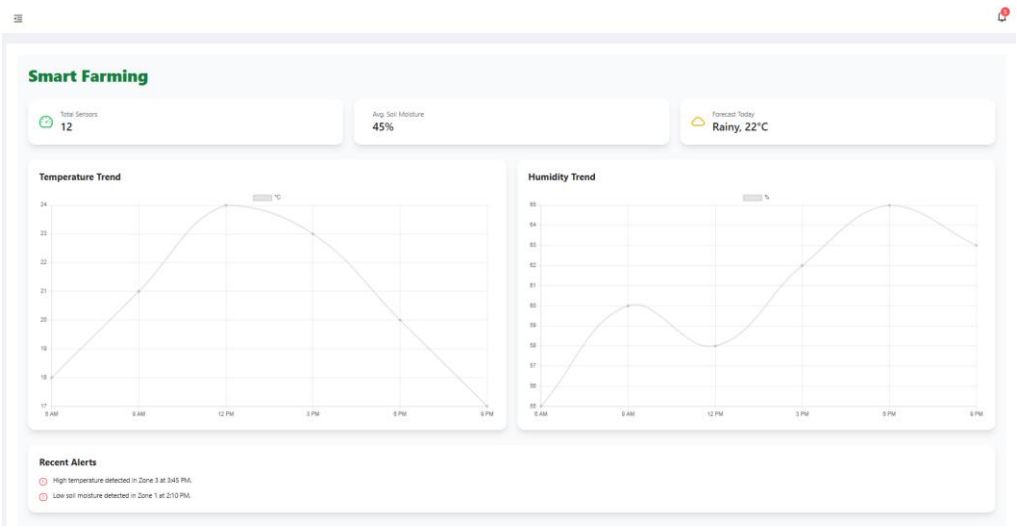


Figure 4.7: Prototype - Smart farming system home page

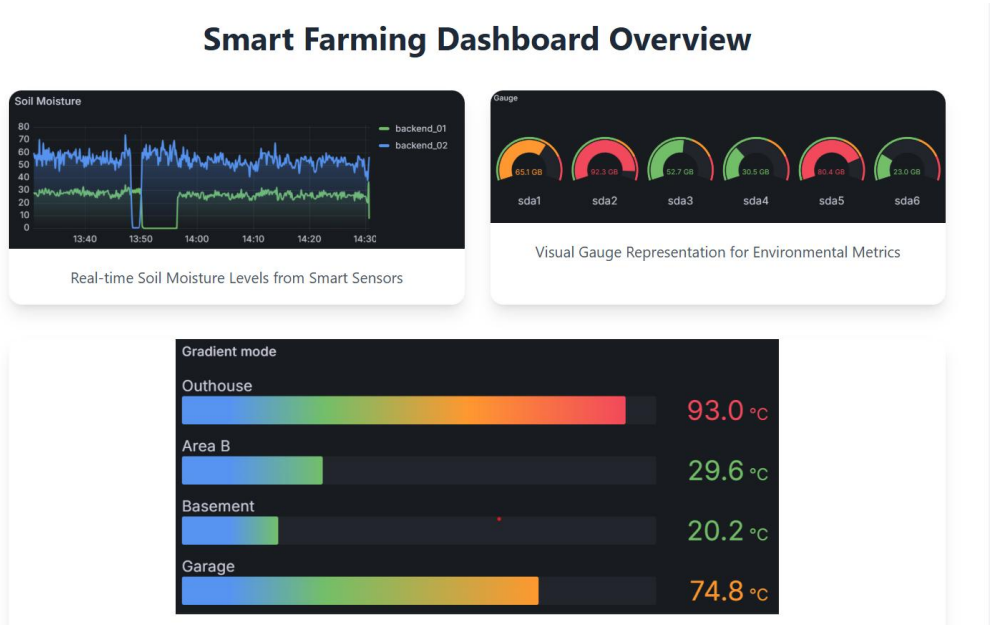


Figure 4.8: Prototype - Sensor dashboard overview

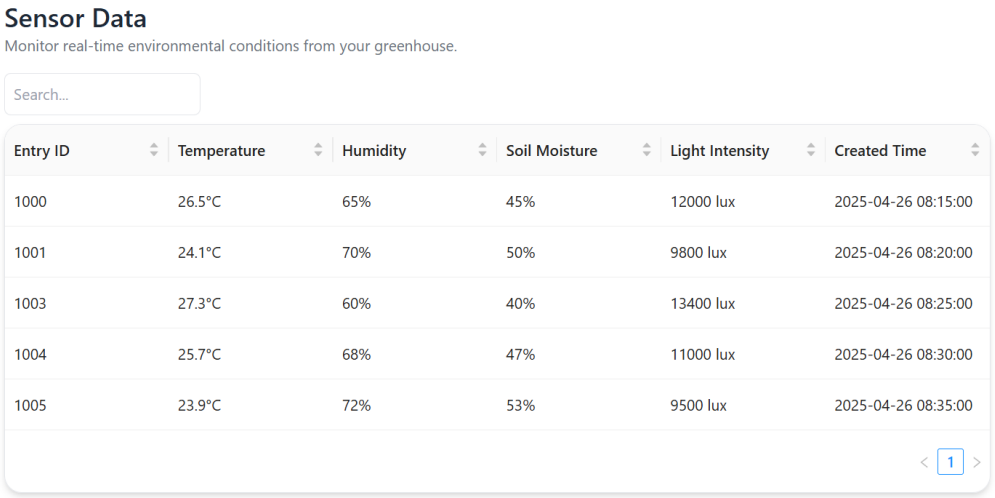


Figure 4.9: Prototype - Sensor data interface

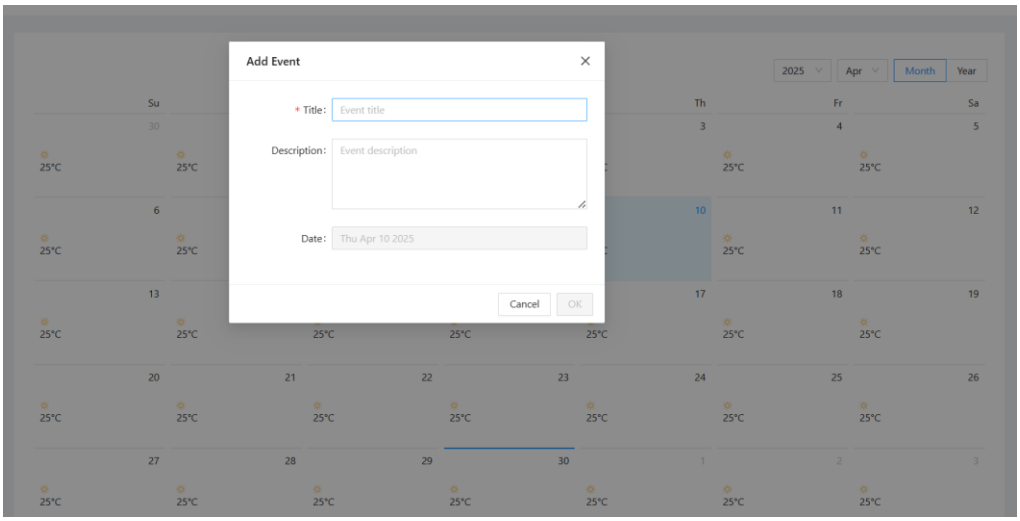


Figure 4.10: Prototype - Manage farming event interface

CHAPTER 5

System Design

5.1 Introduction

The system design defines the blueprint of the smart farming system, detailing how its components interact to achieve the objectives of real-time monitoring, automated decision support, and improved crop yield and quality. This chapter presents the design from three perspectives: the overall system architecture, the database design, and the functional modules. The architecture establishes the layered structure that governs communication between the presentation, application, and data layers, while the database design specifies entity relationships, schemas, and data dictionaries to ensure consistency and integrity of stored information such as sensor readings, thresholds, user accounts, and tasks. The functional modules, including sensor monitoring, threshold configuration, task management, and notification services, are described to illustrate how each supports the system's objectives. Collectively, these design decisions provide a scalable, maintainable, and cost-effective foundation for the subsequent implementation and evaluation of the smart farming solution.

5.2 System Architecture Design

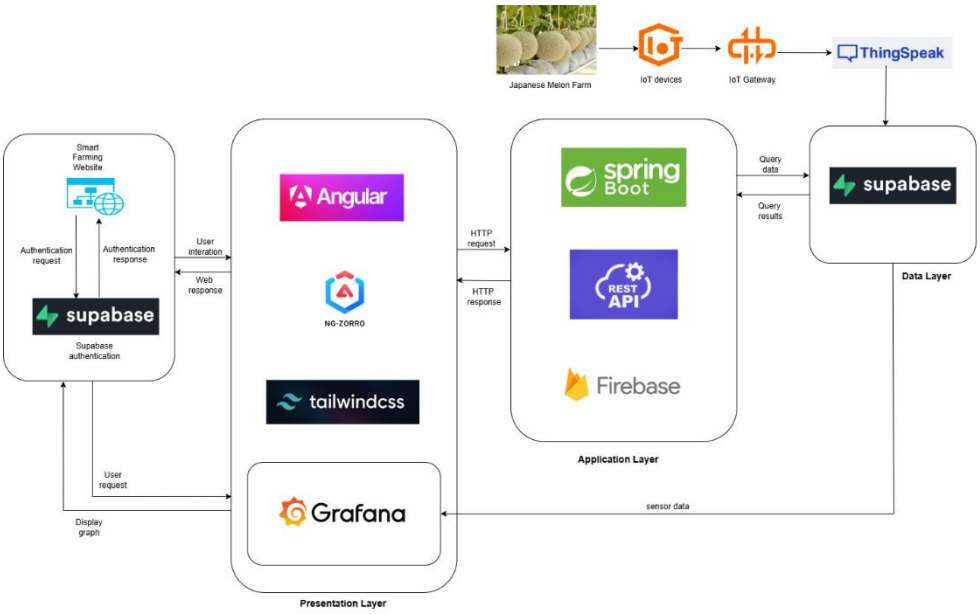


Figure 5.1: System Architecture Design

The smart farming system for Japanese musk melon cultivation was designed using a three-tier architecture, consisting of the Presentation Layer, the Application Layer, and the Data Layer. This layered approach was selected because it provides scalability, maintainability, and a clear separation of concerns, all of which are critical for systems that are expected to evolve alongside future farming requirements. The architecture supports both real-time insights and advanced analytics, thereby enhancing decision-making and crop quality. Figure 5.x illustrates the overall system architecture.

The Presentation Layer was implemented using Angular as the primary frontend framework. Angular was selected due to its modular design, strong ecosystem, and two-way data binding, which collectively supports the development of dynamic dashboards that update in real time. To complement this, Ng Zorro was adopted as a UI component library, providing professional-grade components that accelerate development and ensure design consistency. TailwindCSS was integrated to deliver a utility-first styling approach, resulting in a responsive and highly customizable dashboard accessible across devices.

In addition, Grafana was embedded within the presentation layer to provide advanced visualization capabilities. Grafana was selected because it offers powerful time-series analytics and reduces the need to develop complex charting modules manually. This enabled the system to deliver meaningful insights such as time-series graphs, Soil Health Index values, and correlation heatmaps directly within the Angular interface. The result is a user interface that not only displays data but also supports informed decision-making.

The Application Layer was developed using Spring Boot, chosen for its lightweight framework, modularity, and suitability for RESTful API development. This layer acts as middleware, ensuring standardized and loosely coupled communication between the frontend and the backend database. Such separation allows both the Angular frontend and the Supabase database to evolve independently without disrupting system stability. Spring Boot was also integrated with Firebase to enable real-time push notifications, ensuring that external services are managed at the middleware level rather than tied directly to the frontend. This approach increases robustness, maintainability, and long-term adaptability.

The Data Layer forms the foundation of the system and was designed with both technical and budget constraints in mind. Direct IoT-to-database integration was not feasible because the SIM-based IoT devices lacked support for secure HTTPS communication with a cloud-hosted PostgreSQL database. To address this constraint, ThingSpeak was selected as an intermediary platform for IoT data ingestion. ThingSpeak provides a reliable and cost-effective gateway that supports HTTP transmission, enabling the system to operate without requiring costly hardware upgrades.

From ThingSpeak, sensor data is synchronized into Supabase, which was chosen as the central cloud database due to its PostgreSQL foundation, schema management capabilities, and scalability. This two-step pipeline, ThingSpeak for ingestion and Supabase for structured storage ensures reliable data handling while remaining cost-effective. Grafana connects directly to

Supabase to deliver analytics, enabling both real-time monitoring and long-term trend analysis through tools such as correlation heatmaps and Soil Health Index visualizations.

In summary, the architecture combines Angular, Ng Zorro, and TailwindCSS for the presentation layer; Spring Boot and Firebase for the application layer; and ThingSpeak, Supabase, and Grafana for the data layer. Each technology was selected to balance feasibility under budget constraints with the need for scalability, usability, and analytical capability. Collectively, these choices enable the system to deliver a robust, data-driven solution that enhances melon yield and quality through cost-effective and sustainable smart farming practices.

5.3 Database Design

5.3.1 Entity Relationship Diagram (ERD)

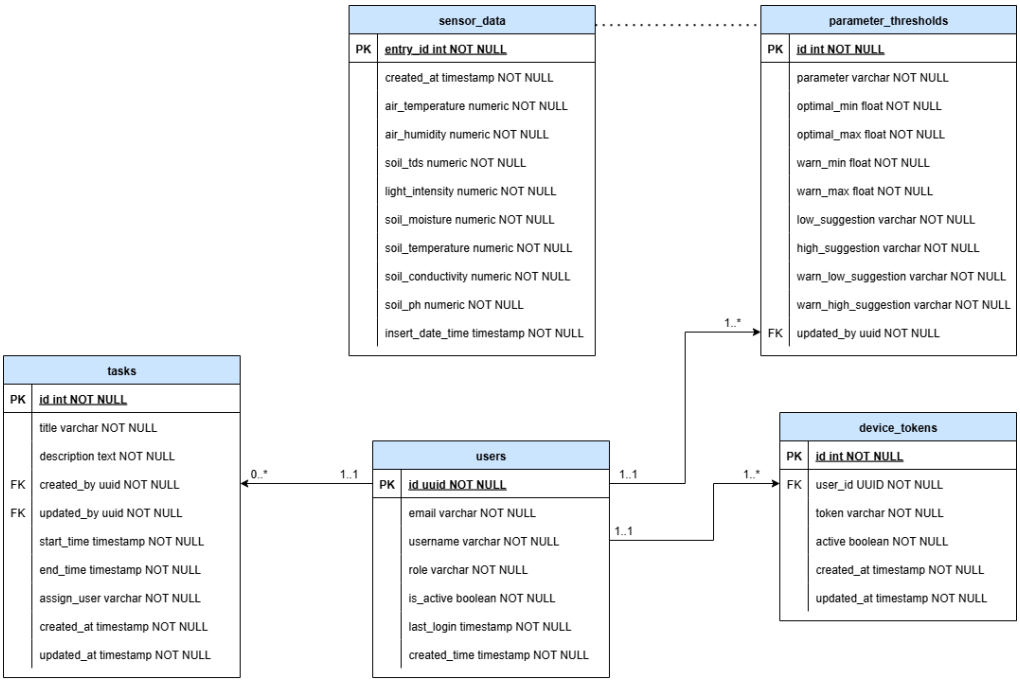


Figure 5.2: Entity Relationship Diagram

5.3.2 Schema Design

Users Entity Data Schema

Column Name	Definition	Data Type	Sample Value

user_id	Unique identifier for each user.	UUID	6aab053c-bdf4-4bcf-b71e-6aca36854b7d
email	Email address of the user.	Text	farmer01@example.com
role	Defines the user's role (e.g., admin, farmer).	Text	Farmer
username	Display name of the user.	Text	MelonMaster
is_active	Indicates whether the user account is active	Boolean	True
last_login	Records the last time the user logged into system.	Timetamp	2025-08-15 14:32:00
created_time	The time when the user account was created.	Timestamp	2025-03-15 14:32:00

Sensor_data Entity Data Schema

Column Name	Definition	Data Type	Sample Value
entry_id	Unique incremental ID of the reading, same as ThingSpeak entry ID.	Integer	1056
air_humidity	Measured humidity in the	Numeric	72.4

	air (%).		
air_temperature	Measured temperature in the air (°C).	Numeric	28.6
light_intensity	Measured sunlight/light intensity (lux).	Numeric	1350.0
soil_conductivity	Electrical conductivity of the soil (µS/cm).	Numeric	220.5
soil_moisture	Moisture content in the soil (%).	Numeric	44.3
soil_ph	pH level of the soil.	Numeric	6.8
soil_tds	Total dissolved solids in soil (ppm).	Numeric	550.0
soil_temperature	Temperature of the soil (°C).	Numeric	26.2
created_at	Original timestamp when data was recorded in ThingSpeak.	Timestamp	2025-08-15 00:10:57
insert_date_time	Timestamp when the data was inserted into the backend database.	Timestamp	2025-08-15 00:15:00

Task Entity Data Schema

Column Name	Definition	Data Type	Sample Value
id	Unique identifier for each task	Integer	1

	(primary key)		
title	Short title or name of the task	Varchar	Irrigation Check
description	Detailed explanation of the task	Text	Inspect and adjust drip irrigation system for melon beds
assign_user	The user assigned to carry out the task	varchar	Ali
created_by	User ID of the person who created the task	UUID	6aab053c-bdf4-4bcf-b71e-6aca36854b7d
updated_by	User ID of the person who last updated the task	UUID	6aab053c-bdf4-4bcf-b71e-6aca36854b7d
start_time	Scheduled start date and time of the task	Timestamp	2025-08-15 14:49:00
end_time	Scheduled end date and time of the task	Timestamp	2025-08-15 19:49:00
created_at	Timestamp when the task record was created	Timestamp	2025-08-12 02:44:40.2
updated_at	Timestamp when the task record was last modified	Timestamp	2025-08-15 02:44:40.2

Device Tokens Data Schema

Column Name	Definition	Data Type	Sample Value
id	Unique identifier for each device	Int	1

	token		
user_id	Unique identifier of the user associated with the device	UUID	6aab053c-bdf4-4bcf-b71e-6aca36854b7d
token	Push notification token generated by Firebase or similar	Varchar	dS7k-C0Wd9wvZo2XLJRl-p_80gEN3J7C9s4JvrjTWfYQlOUtkWDEZLXumy_fPm1e4Opo
active	Status flag indicating if the token is currently active	Boolean	True
created_at	Timestamp when the token record was first created	Timestamp	2025-09-09 15:50:22.31835 +00
updated_at	Timestamp when the token record was last updated	Timestamp	2025-09-09 15:50:22.31835 +00

Parameter Thresholds Data Schema

Column Name	Definition	Data Type	Sample Value
id	Unique identifier for each threshold record (primary key)	Int	1
Parameter	Name of the monitored parameter	Varchar	air_temperature
Optimal_min	Minimum value of the optimal range	Float	24
Optimal_max	Maximum value of the optimal range	Float	32
Warn_min	Minimum value for the	Float	22

	warning range (before becoming critical)		
Warn_max	Maximum value for the warning range (before becoming critical)	Float	35
Low_suggestion	Suggested corrective action when parameter falls below minimum	Varchar	Close vents or use heaters to raise temperature.
High_suggestion	Suggested corrective action when parameter exceeds maximum	Varchar	Improve greenhouse ventilation or install shading net.
Warn_low_suggestion	Suggested action when parameter approaches lower warning level	Varchar	Monitor, consider partial vent closing.
Warn_high_suggestion	Suggested action when parameter approaches higher warning level	Varchar	Keep ventilation running and monitor closely.
Updated_by	User ID who last updated the threshold entry	UUID	6aab053c-bdf4-4bcf-b71e-6aca36854b7d

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 Introduction

This chapter describes the software implementation of the Smart Farming System for musk melon cultivation, focusing on the transformation of the proposed system design into a functional application. The system is designed to enhance farming efficiency through IoT-based monitoring, where environmental sensor data are collected and transmitted to a ThingSpeak channel for subsequent processing and analysis. The implementation involves developing a software platform that retrieves data from ThingSpeak, processes it through cloud-based services, and presents it to users via an interactive and responsive interface. In addition to real-time data visualisation, the system supports environmental condition tracking and decision support features to maintain optimal growing conditions for musk melons. This chapter presents the implementation of the major software modules, the configuration of ThingSpeak integration, and the establishment of reliable communication between the cloud platform and the user interface.

6.2 System Module

The end-users for this system are categorized into two groups, that will be the administrative side and the non-administrative side. Since some functions are related to managerial tasks, a few submodules could be only accessed by administrative users.

Table 6.1: Module Overview by User Role

End-users	Module Name	Objective of Module
Admin	User and role management	Manage user roles, and system access.
	Add new user	Add new user to the smart farming system.
All user	Authentication	Secure login and access control for

		the system.
	Profile management	View and update personal details such as username, email, and password.
	Real-time sensor Monitoring	View live environmental data and historical trends.
	View suggestion	View rule-based suggestions for corrective actions.
	Task management	Manage farming task, keep track of pass and future farming activities.
	View analysed data dashboard	Explore graphical dashboards, soil health index, and correlation analytics for insights.
	Receive alert notifications	Get real-time alerts and notifications for abnormal conditions or threshold breaches.
	Manage sensor threshold	Configure, update, and maintain threshold values and respective suggestion for different environmental parameters.

6.3 Functional Module Implementation

6.3.1 Supabase Authentication

In order to provide secure access and enable personalized features within the Smart Farming System, Supabase Authentication (Auth) was implemented as the core user management module. Supabase Auth supports user registration, login, and session handling while integrating directly with the PostgreSQL backend, ensuring that only authorized users can access the farming dashboard and sensor data retrieved from the ThingSpeak channel. This implementation strengthens system security by restricting access to authenticated users and enabling role-based control of administrative functions.

The Supabase project was first created and configured through the Supabase dashboard. Within the authentication settings, the email and password option were enabled to support sign-up and login processes for farmers and administrators. Supabase automatically manages user credentials within its PostgreSQL database, reducing the need for additional custom authentication logic. During configuration, the system generated an API URL and an anonymous public key, which were subsequently integrated into the client application to establish secure communication with the Supabase backend.

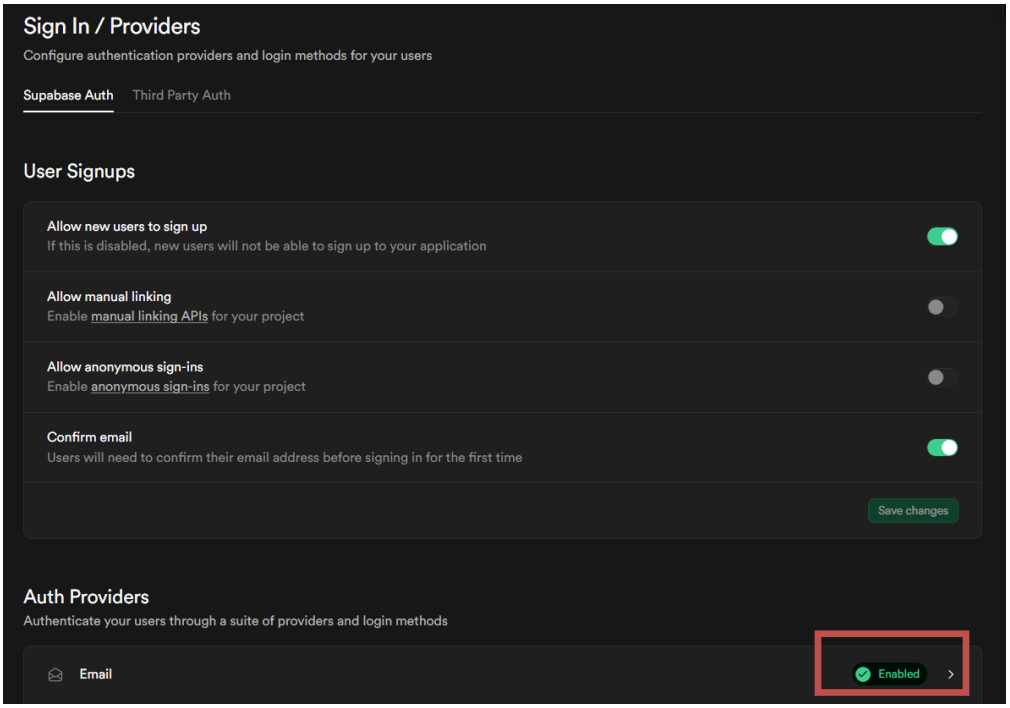


Figure 6.1: Enable auth providers (email) in supabase

The application integrates with Supabase Authentication through the @supabase/supabase-js client library, which provides a simple and secure interface for managing user sessions. The authentication flow consists of two primary functions:

i. **Sign Up process -** allows administrators to invite and register new users

ii. **Sign In process -** authenticates existing users and generates a valid session token.

```
signIn(email: string, password: string) : Promise<AuthTokenResponsePassword> { Show usages keyinglol
  return this.supabase.auth.signInWithPassword({ email, password });
}

signOut() : Promise<{ error: AuthError | null }> { Show usages keyinglol
  return this.supabase.auth.signOut();
}

getSession() : Promise<{ data: { session: Session }; err... { Show usages keyinglol
  return this.supabase.auth.getSession();
}
```

Figure 6.2: Code snippet for handling signs in, out and retrieve user's session

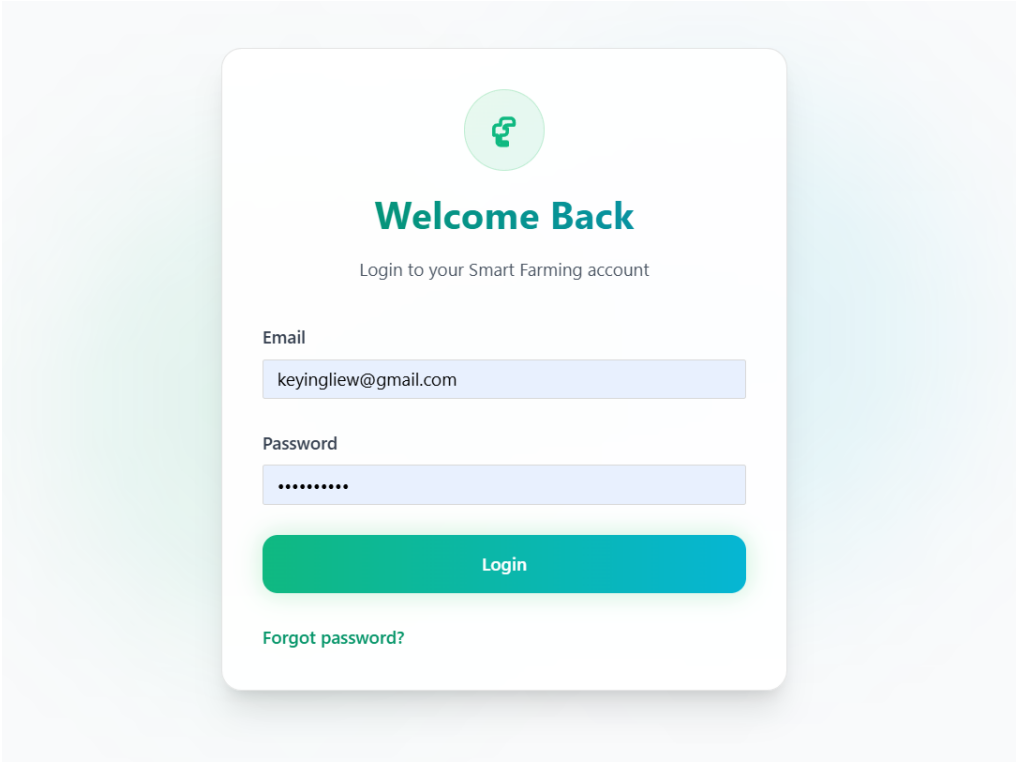


Figure 6.3: Sign in Page

During the sign-in process, users enter their email and password into the login form, and these credentials are transmitted securely to the Supabase Authentication API using the signInWithPassword() method. Supabase verifies the credentials against the user records stored in the PostgreSQL database. Upon successful authentication, Supabase generates a session object containing a JSON Web Token (JWT), which serves as proof of the user's identity. This token is stored locally by the application and is required for all subsequent requests to protected resources. In the event of invalid credentials,

Supabase returns an error response, preventing unauthorised access to the system.

To enhance usability, the login interface also includes a password recovery mechanism, whereby users can request a reset link sent to their registered email address. This ensures that forgotten credentials can be securely managed without compromising the integrity of the authentication system.

The sign-out process invalidates the active session and removes the locally stored token, ensuring that the user is fully logged out of the system. Together, these mechanisms provide a robust authentication framework that balances security, usability, and maintainability within the smart farming system.

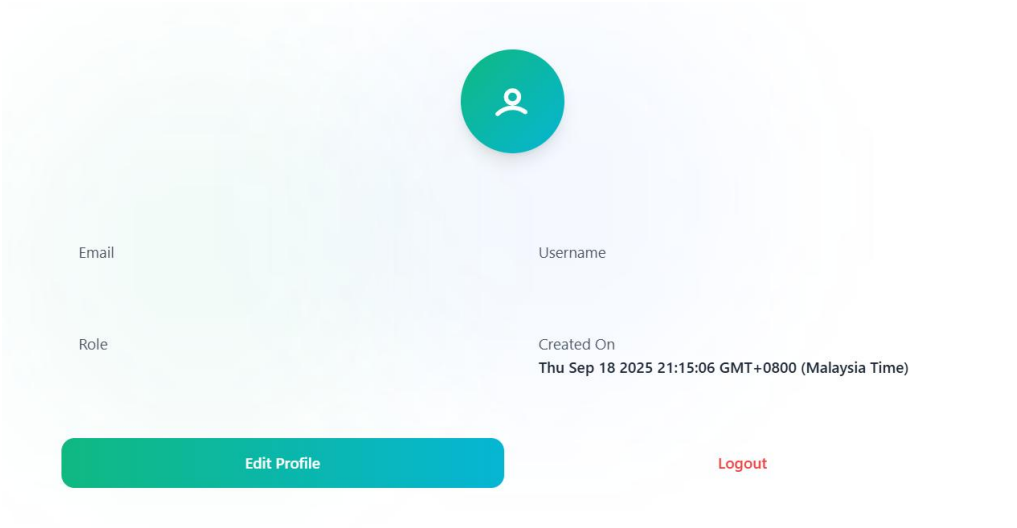


Figure 6.4: Sign out function for user in profile page

6.3.2 Authorisation

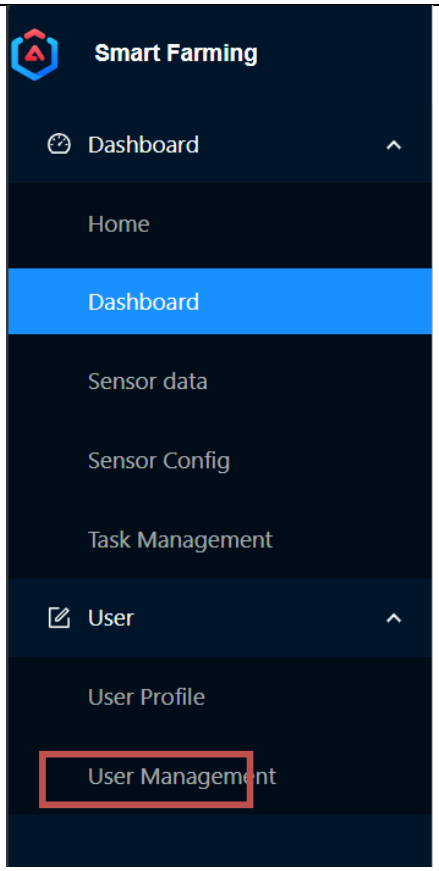


Figure 6.5: Admin navigation view

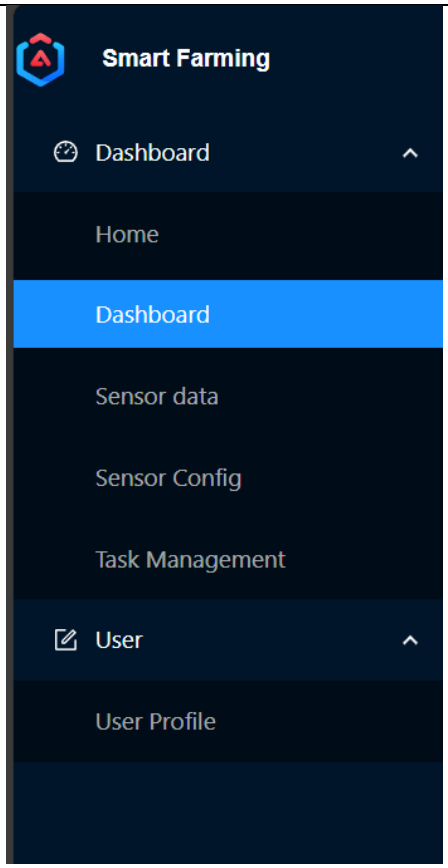


Figure 6.6: Normal user navigation view

Authorisation within the Smart Farming System is implemented through role-based access control, ensuring that users only have access to functions appropriate to their roles. Two main roles are defined in the system: administrator and normal user. Administrators are granted extended privileges which is the ability to register new users while normal users are restricted to essential functionalities. This role-based design not only strengthens system security by preventing unauthorised access to administrative functions but also enhances usability by presenting each user with a tailored interface aligned to their responsibilities.

6.3.3 Admin Sign Up

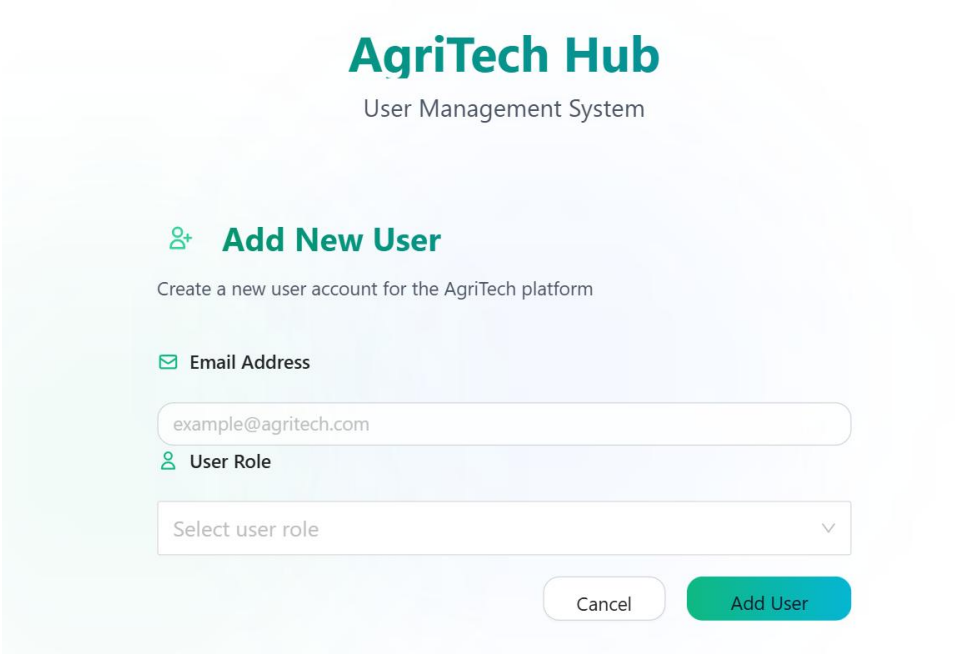


Figure 6.7: Add new user page

The sign-up process in the Smart Farming System is initiated by an administrator, who registers a new user by specifying the individual's email address and assigning an appropriate role, such as farmer or administrator. The application transmits these details securely to Supabase using the Admin Service Key, which authorises privileged operations restricted to administrative users. Supabase then generates a unique registration link and automatically dispatches it to the specified email address.

When the invited user accesses the link, they are redirected to the sign-up page, where they provide a password and complete the registration form. Once submitted, Supabase creates a new user record in the underlying PostgreSQL database, embedding the role assigned during registration. The system then issues a confirmation message to the user, indicating successful account creation. From this point, the user can proceed directly to the sign-in process to access the system's features.

Error-handling mechanisms are incorporated to ensure robustness during onboarding. If an invalid email address is provided or if the registration

link has expired, Supabase returns an error response. In such cases, the administrator is prompted to resend the invitation, thereby ensuring a smooth and reliable registration process.

The API for adding a new user leverages Supabase's administration endpoints and is secured through the Admin Service Key, ensuring that only authorised personnel can register new accounts. A code snippet illustrating this API call is provided in Figure 6.7 as supporting evidence of the implementation.

```
@Injectable({ providedIn: 'root' }) Show usages new *
export class AddUserService {

  private baseUrl : string = 'http://localhost:8085/api/admin/invite';

  constructor(private http: HttpClient) {} no usages new *

  inviteUser(email: string, role : string) : Observable<Object> { Show usages new *
    return this.http.post(this.baseUrl, { email, role });
  }
}
```

Figure 6.8: API call for add new user

6.3.4 User Profile Management

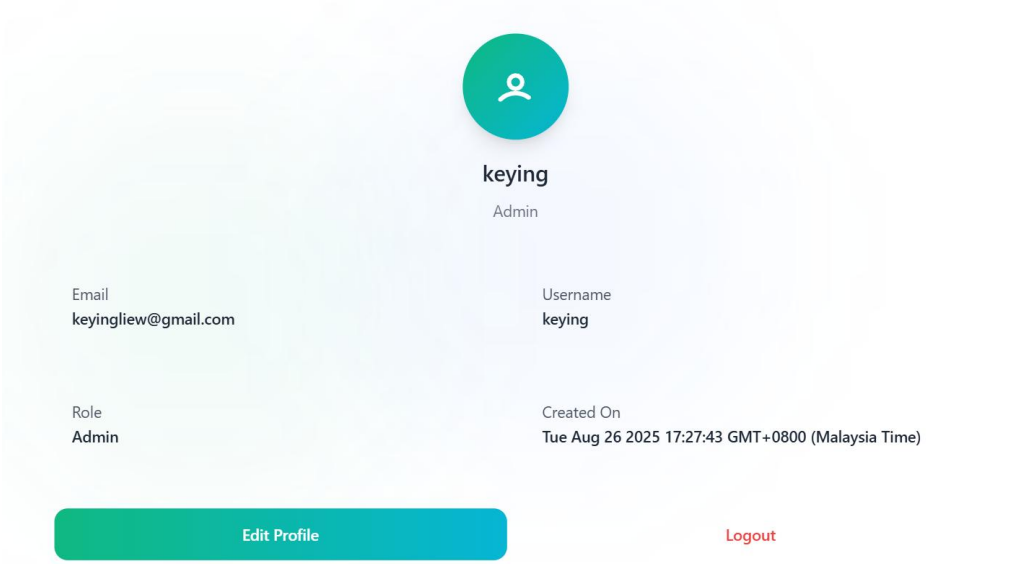


Figure 6.9: User profile management page

As shown in Figure 6.8, the User Profile Management page was implemented to enable authenticated users to view and update their personal information within the system. This feature enhances usability by allowing users to verify their registered details and make modifications, such as updating their username, when necessary. The module is integrated with Supabase Authentication and the corresponding user profile table in the PostgreSQL database, ensuring that any updates remain consistent across authentication records and application data. To maintain security, update requests are validated so that users are only permitted to modify their own profiles. Once approved, the revised details are committed to the database and reflected immediately in the interface, providing a seamless and secure profile management experience.

6.3.5 Task Management

The Task Management Module was developed to assist farmers and administrators in organizing and monitoring farm-related activities within the smart farming system. It enables the creation, updating, and deletion of farming tasks, while also supporting visualization of schedules in a calendar format. This feature is essential for ensuring that agricultural activities such as irrigation, fertilization, or equipment inspections are executed in a timely manner, thereby reducing the risk of overlooked or delayed operations.

```
export class TaskManagementComponent implements OnInit {
  calendarOptions: CalendarOptions = {
    plugins: [dayGridPlugin, timeGridPlugin, interactionPlugin],
    initialView: 'dayGridMonth',
    headerToolbar: {
      left: 'prev,next today',
      center: 'title',
      right: 'dayGridMonth,timeGridWeek,timeGridDay'
    },
    editable: true,
    selectable: true,
    events: (info, successCallback, failureCallback) => {
      successCallback(this.events);
    },
    eventDisplay: 'block',
    displayEventTime: true,
    displayEventEnd: true,
    select: this.onDateClick.bind(this),
    dateClick: this.onDateClick.bind(this),
    eventClick: this.onEventClick.bind(this)
  };
```

Figure 6.10: Code snippet for FullCalendar implementation

The module was implemented using the FullCalendar library integrated into Angular, which provides interactive and customizable scheduling capabilities. Farmers are able to view tasks in multiple modes including monthly, weekly, and daily perspectives. Each task entry consists of essential attributes such as title, description, start time, end time and assign user. Through the interface, users can click on a calendar date to add a new task or select an existing event to update or delete it. Modal dialogs powered by Ng-Zorro components provide a structured form for task entry and editing, ensuring consistent user experience.

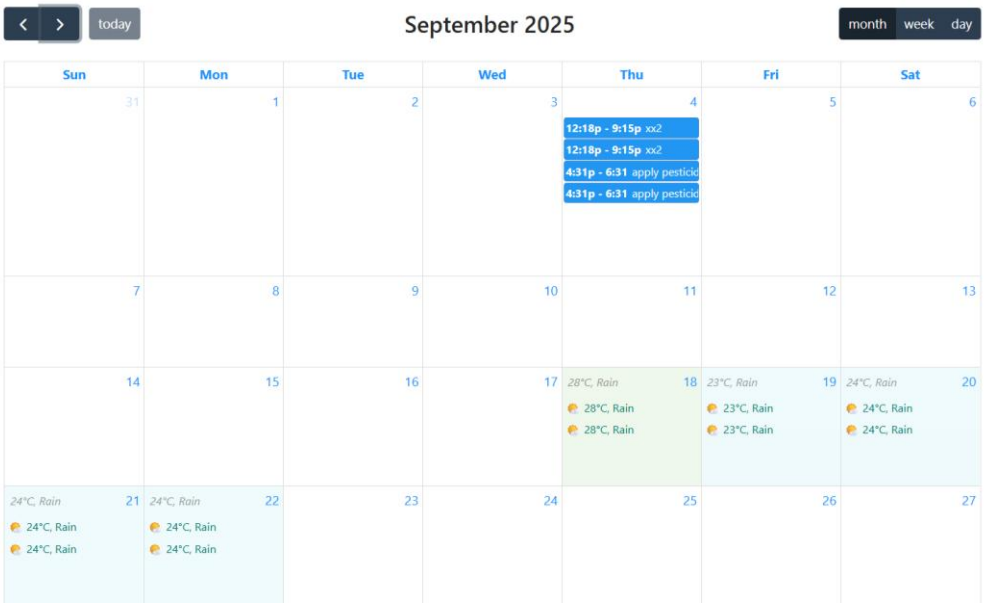


Figure 6.11: Calendar monthly view with weather forecast

Internally, the Task Service handles communication with the backend, where task information persisted in the Supabase database. The component retrieves tasks via the service and transforms them into calendar events for rendering. CRUD operations are supported, where updates to task data are reflected in real time on the calendar interface. The module also integrates weather forecast data, displayed alongside tasks as background and foreground events, thereby assisting farmers in planning activities according to environmental conditions.

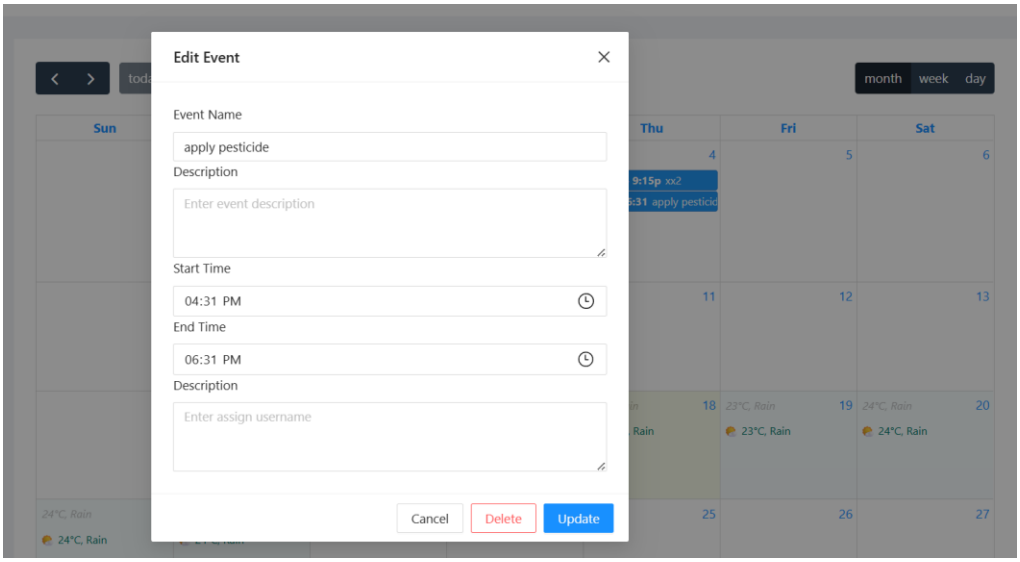


Figure 6.12: Calendar modal dialog for adding/editing a task

By combining calendar-based visualization with backend task management, the module enhances farm operation planning and contributes to resource efficiency. It provides farmers with a centralized interface to track past and upcoming activities, while also aligning with the system's overall objective of supporting decision-making through timely and actionable information.

6.3.6 Sensor threshold value configuration

The Sensor Threshold Value Configuration module was developed to allow users to manage the operating ranges of key environmental parameters within the smart farming system. Each sensor parameter is associated with predefined threshold ranges that determine its optimal, warning, and critical levels. These thresholds form the basis for the system's Suggestion Service, which generates corrective recommendations whenever sensor readings deviate from expected conditions.

The module provides a tabular interface, built using Angular and Ng-Zorro UI components, through which administrators can view, update, and configure threshold values. Each row corresponds to a specific parameter, displaying its associated threshold settings along with editable fields. In addition to numeric ranges (warning minimum, optimal minimum, optimal maximum, and warning maximum), the module also enables administrators to configure customized suggestion messages for each parameter and range. For example, users may specify corrective actions such as "Increase irrigation to restore soil moisture" or "Adjust ventilation to reduce air temperature." This design ensures flexibility, as messages can be modified directly through the front end without requiring backend code changes.

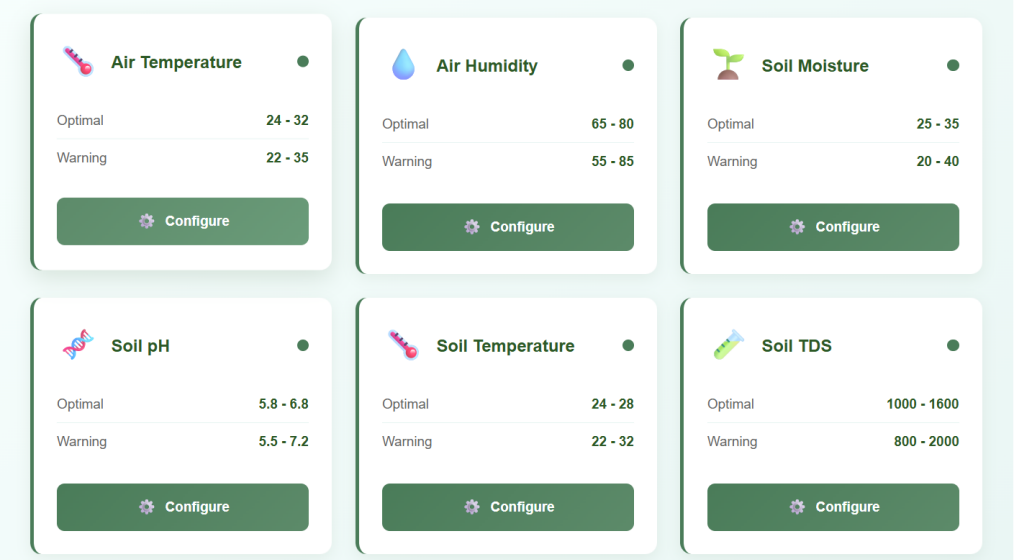
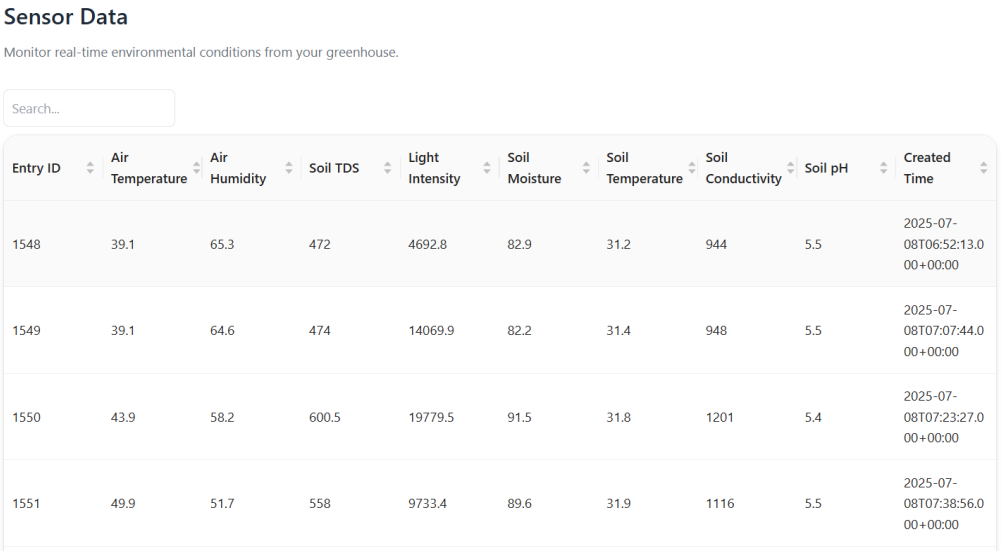


Figure 6.13: Threshold configuration page

All changes persisted in the parameter_thresholds table of the Supabase database. Once updated, these values are immediately utilized by the Suggestion Service in real time, ensuring that new recommendations and alerts are aligned with the latest configuration. This approach empowers administrators to adapt the system dynamically to varying cultivation requirements or seasonal conditions, thereby enhancing its practical utility.

6.3.7 Sensor Data Table View

The Sensor Data Table module was developed to provide administrators and farmers with an organized and interactive interface for viewing raw sensor readings collected from the greenhouse. This component displays environmental data including air temperature, humidity, soil moisture, soil temperature, soil pH, soil conductivity, total dissolved solids (TDS), and light intensity. Each entry in the table is linked to a unique identifier (entry_id) and timestamp (created_at), allowing users to trace the exact moment a reading was captured.



Entry ID	Air Temperature	Air Humidity	Soil TDS	Light Intensity	Soil Moisture	Soil Temperature	Soil Conductivity	Soil pH	Created Time
1548	39.1	65.3	472	4692.8	82.9	31.2	944	5.5	2025-07-08T06:52:13.000+00:00
1549	39.1	64.6	474	14069.9	82.2	31.4	948	5.5	2025-07-08T07:07:44.000+00:00
1550	43.9	58.2	600.5	19779.5	91.5	31.8	1201	5.4	2025-07-08T07:23:27.000+00:00
1551	49.9	51.7	558	9733.4	89.6	31.9	1116	5.5	2025-07-08T07:38:56.000+00:00

Figure 6.14: Sensor data table

The module was implemented in Angular, utilizing a custom TableComponent together with Ng-Zorro UI elements to provide advanced data handling capabilities. Users can sort, and search through the dataset to locate specific records or trends. For example, the sorting function allows the data to be ordered by attributes such as time of creation or sensor values. A search bar is also provided to refine results based on user queries, improving accessibility when dealing with large datasets.

All sensor data is retrieved dynamically from the Supabase backend through the SensorReadingService, which communicates with the database via API calls. Once retrieved, the readings are mapped into table rows, ensuring real-time synchronization between the underlying database and the frontend interface. This design ensures that farmers and administrators are always working with the most recent sensor readings, reducing the risk of outdated or inaccurate information.

By combining raw sensor visualization with interactive filtering and search functions, the Sensor Data Table module enhances the system's transparency and usability. Farmers can directly inspect the captured environmental data, while administrators can cross-verify whether parameter thresholds and generated suggestions align with actual sensor conditions. This

component thus serves as the foundational layer for higher-level analytics and decision-support features within the smart farming system.

6.4 Business Logic Implementation

6.4.1 Supabase Edge Function

A Supabase Edge Function was deployed to transfer IoT sensor data from ThingSpeak into the sensor_data table in Supabase. Running in a serverless environment, the function eliminates the need for a dedicated backend server while ensuring efficient and secure data synchronization.

The function queries the most recent entry_id stored in the database, fetches new entries from the ThingSpeak API using Axios, and applies a retry mechanism to handle transient errors. Retrieved data are transformed into the schema format, validated, and timestamped before being inserted into Supabase in bulk. Logging is included to track operations such as fetch attempts, inserted rows, and potential errors.

This design provides a lightweight ETL pipeline that reduces latency between data acquisition and storage, enabling near real-time updates on the dashboard. It demonstrates the effectiveness of serverless functions in bridging external IoT platforms with cloud databases, while supporting scalability and reliability in smart farming applications.

```
const fetchAndStoreThingSpeakData = async ()=>{
  console.log('[Function fetchAndStoreThingSpeakData] Invoked');
  try {
    console.log('[Fetching Last Entry ID from Supabase]');
    const { data: lastRow, error: fetchError } = await supabase.from('sensor_data').select('entry_id').order('entry_id', {
      ascending: false
    }).limit(1).single();
    if (fetchError && fetchError.code !== 'PGRST116') {
      console.error('[Fetch Last Entry ID] Error:', fetchError);
      return;
    }
    const lastEntryId = lastRow?.entry_id || 0;
    console.log('[Last entry_id fetched]:', lastEntryId);
    Windsurf: Refactor | Explain | Generate JSDoc | X
    const retry = async (fn, retries = 3)=>{
      let attempt = 0;
      while(attempt < retries){
        try {
          return await fn();
        } catch (err) {
          attempt++;
          if (attempt >= retries) throw err;
          console.warn(`[Retry ${attempt}/${retries}] Retrying due to:`, err.message);
        }
      }
    };
```

Figure 6.15: Supabase Edge function for fetch data from ThingSpeak

```
const response = await retry(()=>axios.get(`https://api.thingspeak.com/channels/${THINGSPEAK_CHANNEL_ID}/feeds.json`, {
    params: {
      results: RESULTS_LIMIT
    }
  }));
const feeds = response.data.feeds;
console.log(`[Fetched ${feeds.length} entries from ThingSpeak]`);
const newFeeds = feeds.filter((feed)=>feed.entry_id > lastEntryId).map((feed)=>{
  const temp = parseFloat(feed.field1);
  return {
    entry_id: feed.entry_id,
    created_at: new Date(feed.created_at).toISOString(),
    insert_date_time: new Date().toISOString(),

    air_temperature: isNaN(temp) ? null : temp,
    air_humidity: parseFloat(feed.field2) || null,
    soil_tds: parseFloat(feed.field3) || null,
    light_intensity: parseFloat(feed.field4) || null,
    soil_moisture: parseFloat(feed.field5) || null,
    soil_temperature: parseFloat(feed.field6) || null,
    soil_conductivity: parseFloat(feed.field7) || null,
    soil_ph: parseFloat(feed.field8) || null
  };
});
if (newFeeds.length === 0) {
  console.log('[No new data to insert]');
  return;
}
```

Figure 6.16: Supabase Edge function for store fetched data into database

6.4.2 Supabase Cron Job

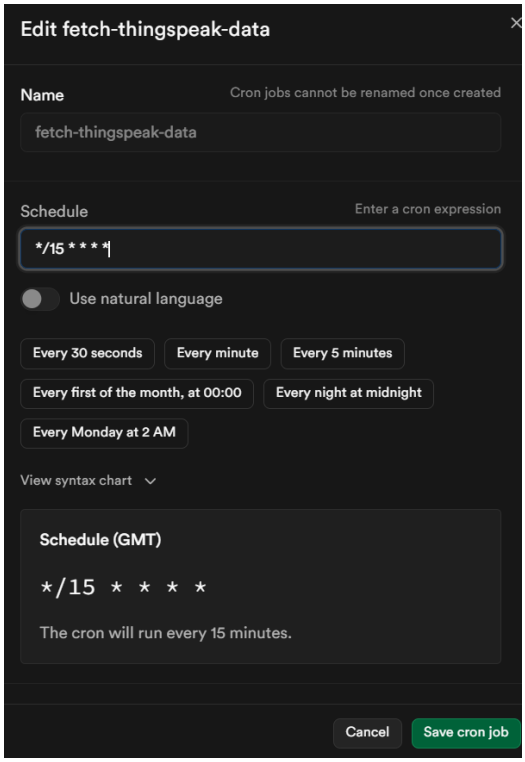


Figure 6.17: 15-minute interval cron job

To ensure the Supabase Edge Function operates continuously without manual intervention, a Supabase Cron Job was configured. The Cron Job automatically triggers the deployed Edge Function every 15 minutes, enabling

consistent retrieval of sensor readings from ThingSpeak and insertion into the sensor_data table.

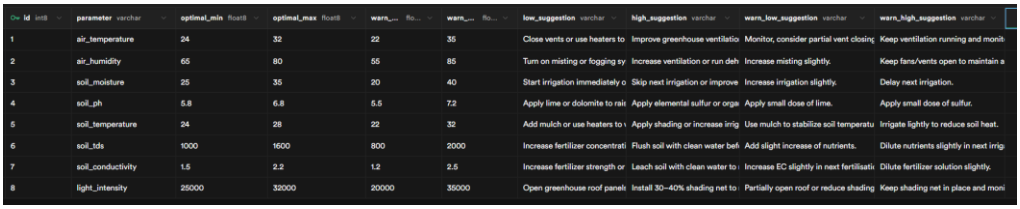
This scheduling frequency was selected to provide timely updates for near real-time monitoring while avoiding excessive API calls that could lead to redundant data collection or unnecessary resource usage. By combining serverless functions with scheduled execution, the system establishes a reliable and efficient data ingestion pipeline that supports the monitoring and analytics modules of the smart farming system.

6.4.3 Threshold-based rules suggestion logic

The Suggestion Service is a critical component of the smart farming backend, designed to analyze sensor readings and provide data-driven recommendations to farmers. Its primary objective is to evaluate recent environmental conditions against predefined threshold rules and generate suggestions that support the maintenance of optimal farming conditions. In this way, the service transforms raw sensor values into actionable insights that directly contribute to informed decision-making in the field.

entry_id int4	created_at timestamp	air_temperature numeric	air_humidi... numa...	soil_tds numeric	light_intensity numeric	soil_moisture numeric	soil_temperature numeric	soil_conductivity numeric	soil_ph numeric	insert_data_time timestamp
1548	2025-07-08 14:52:13	39.10	65.30	472.00	4692.80	82.90	31.20	944.00	5.50	2025-07-08 16:02:40
1549	2025-07-08 15:07:44	39.10	64.60	474.00	14069.90	82.20	31.40	948.00	5.50	2025-07-08 16:02:40
1550	2025-07-08 15:23:27	43.90	58.20	600.50	19779.50	91.50	31.80	1201.00	5.40	2025-07-08 16:02:40
1551	2025-07-08 15:38:56	49.90	51.70	558.00	9733.40	89.60	31.90	1116.00	5.50	2025-07-08 16:02:40
1552	2025-07-08 15:54:31	43.90	58.20	512.50	8549.30	88.20	32.10	1025.00	5.50	2025-07-08 16:02:40
1553	2025-07-08 16:10:03	42.30	62.50	496.50	3382.80	87.80	32.30	991.00	5.50	2025-07-08 16:15:01
1554	2025-07-08 16:25:33	39.10	64.60	493.50	4433.30	87.50	32.40	987.00	5.50	2025-07-08 16:30:02
1555	2025-07-08 16:41:05	37.50	71.80	490.50	5365.30	87.00	32.50	981.00	5.50	2025-07-08 16:45:01
1556	2025-07-08 16:56:34	38.00	68.90	489.00	2794.90	86.70	32.50	978.00	5.50	2025-07-08 17:00:02

Figure 6.18: Sensor data table



id int8	parameter varchar	optimal_min float8	optimal_max float8	warn_... flo...	warn_... flo...	low_suggestion varchar	high_suggestion varchar	warn_low_suggestion varchar	warn_high_suggestion varchar
1	air_temperature	24	32	22	35	Close vents or use heaters to	Improve greenhouse ventilatio	Monitor, consider partial vent closing	Keep ventilation running and monit
2	air_humidity	65	80	55	85	Turn on misting or fogging sy	Increase ventilation or run deh	Increase misting slightly.	Keep fans/vents open to maintain a
3	soil_moisture	25	35	20	40	Start irrigation immediately o	Skip next irrigation or improve	Increase irrigation slightly.	Delay next irrigation.
4	soil_ph	5.8	6.8	5.5	7.2	Apply lime or dolomite to rais	Apply elemental sulfur or orga	Apply small dose of lime.	Apply small dose of sulfur.
5	soil_temperature	24	28	22	32	Add mulch or use heaters to r	Apply shading or increase irrig	Use mulch to stabilize soil temperatu	Irrigate lightly to reduce soil heat.
6	soil_tds	1000	1600	800	2000	Increase fertilizer concentrati	Flush soil with clean water bef	Add slight increase of nutrients.	Dilute nutrients slightly in next irrig
7	soil_conductivity	1.5	2.2	1.2	2.5	Increase fertilizer strength or	Leach soil with clean water to	Increase EC slightly in next fertilisati	Dilute fertilizer solution slightly.
8	light_intensity	25000	32000	20000	35000	Open greenhouse roof panels	Install 30–40% shading net to	Partially open roof or reduce shading	Keep shading net in place and moni

Figure 6.19: Parameter threshold table

The service integrates two main data sources. First, it retrieves real-time sensor readings from the sensor_data table, which represents the actual conditions recorded by IoT devices within the greenhouse. Second, it references predefined threshold values stored in the parameter_thresholds table.

Each parameter is defined by four boundary values: minimum, optimal minimum, optimal maximum, and warning maximum, together with corresponding recommendation messages. This database-driven approach enhances flexibility, as administrators can modify thresholds and recommendations through the frontend interface without altering system code.

```
// Method to check a single parameter against its defined thresholds
private void checkParam(String paramName, Number value, List<SuggestionModel> suggestions) {  8 usages  new *
    // If the sensor value is missing, skip this parameter
    if (value == null) return;

    // Look up the threshold rule for the given parameter from the database
    Optional<ParameterThreshold> ruleOpt = parameterThresholdDao.findByParameter(paramName);

    // If no threshold rule exists, add a message indicating that it is undefined
    if (ruleOpt.isEmpty()) {
        suggestions.add(new SuggestionModel(paramName,  suggestion: "No threshold defined for " + paramName));
        return;
    }

    // Retrieve the threshold rule
    ParameterThreshold rule = ruleOpt.get();
    // Convert the sensor value to a double for comparison
    double v = value.doubleValue();
```

Figure 6.20: checkParam method

```
    // Case 1: Value is below the warning minimum → critical low condition
    if (v < rule.getWarnMin()) {
        String msg = rule.getLowSuggestion();
        suggestions.add(new SuggestionModel(paramName, rule.getLowSuggestion()));
        notificationService.sendNotificationToAll( title: "Alert: " + paramName,  body: msg + " (value=" + v + ")");

        // Case 2: Value is above the warning maximum → critical high condition
    } else if (v > rule.getWarnMax()) {
        String msg = rule.getHighSuggestion();
        suggestions.add(new SuggestionModel(paramName, rule.getHighSuggestion()));
        notificationService.sendNotificationToAll( title: "Alert: " + paramName,  body: msg + " (value=" + v + ")");

        // Case 3: Value is below the optimal range but above warning minimum → slightly low condition
    } else if (v < rule.getOptimalMin()) {
        String msg = rule.getWarnLowSuggestion();
        suggestions.add(new SuggestionModel(paramName, rule.getWarnLowSuggestion()));
        notificationService.sendNotificationToAll( title: "Warning: " + paramName,  body: msg + " (value=" + v + ")");

        // Case 4: Value is above the optimal range but below warning maximum → slightly high condition
    } else if (v > rule.getOptimalMax()) {
        String msg = rule.getWarnHighSuggestion();
        suggestions.add(new SuggestionModel(paramName, rule.getWarnHighSuggestion()));
        notificationService.sendNotificationToAll( title: "Warning: " + paramName,  body: msg + " (value=" + v + ")");
    }
```

Figure 6.21: getSuggestions() method

The service is structured around two core methods: getSuggestions() and checkParam(). The getSuggestions() method retrieves the latest sensor record and evaluates each parameter using the checkParam() function. Parameters without associated thresholds return a default message indicating that no rules are defined. The method compiles the evaluation results into a list

of suggestions; if all parameters fall within their optimal ranges, the service outputs a default message such as “All conditions optimal.”

The checkParam() method performs the core evaluation process. It verifies the availability of sensor readings, compares values against the stored thresholds, and appends appropriate recommendation messages to the results list. This design ensures that deviations are automatically translated into specific, actionable suggestions for the farmer.

6.4.4 Firebase Notification

The Notification Service, implemented using Firebase Cloud Messaging (FCM), is responsible for delivering real-time alerts to farmers whenever environmental parameters deviate from their defined optimal thresholds. While the Suggestion Service performs the evaluation of sensor data and generates context-specific recommendations, the Notification Service ensures that these critical insights are communicated promptly to end users through push notifications. This integration enhances the responsiveness of the smart farming system by enabling immediate corrective action when anomalies are detected.

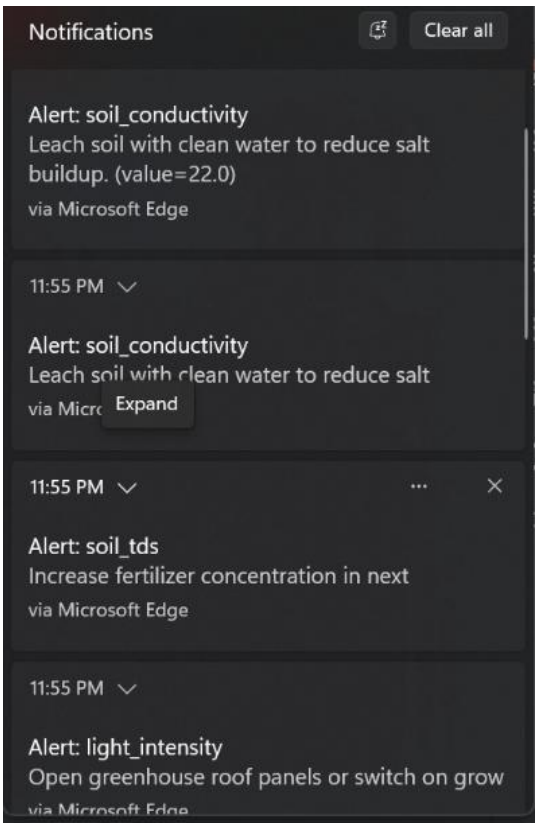


Figure 6.22: Notifications received by user

The service operates in close coordination with the Suggestion Service. Whenever a parameter value falls outside its designated range, a call is triggered to the Notification Service. Each notification contains two primary components: a title (e.g., “Alert: Soil Moisture”) and a message (e.g., “Critical low soil moisture detected. Immediate irrigation is required. Current value = 8%.”). These messages are dynamically generated based on real-time sensor readings and the corresponding threshold rules, ensuring that alerts remain both context-specific and actionable.

```java
// Send notification to all devices (stub, actual push handled elsewhere e.g. FCM)
public void sendNotificationToAll(String title, String body) {  5 usages  new *
    List<DeviceToken> tokens = getActiveTokens();

    tokens.forEach( DeviceToken t -> {
        try {
            Message message = Message.builder()
                    .setToken(t.getToken())
                    .setNotification(Notification.builder()
                            .setTitle(title)
                            .setBody(body)
                            .build())
                    .build();

            String response = FirebaseMessaging.getInstance().send(message);
            System.out.printf("Sent push to token=%s response=%s%n",
                    t.getToken(), response);

        } catch (Exception e) {
            System.err.printf("Failed to send push to token=%s error=%s%n",
                    t.getToken(), e.getMessage());
        }
    });
```

Figure 6.23: Function to send notification to all registered device

Technically, the Notification Service retrieves all registered device tokens stored in the database and forwards the notification payload to Firebase. FCM then distributes the alerts to all subscribed devices, independent of whether the mobile application is active in the foreground or running in the background. For instance, when greenhouse temperature surpasses the warning maximum, an immediate push notification is dispatched to the farmer's device, thereby supporting timely interventions to maintain crop health.

6.5 Data Analytics and Visualization

Grafana is employed in this project as the primary platform for real-time visualization of IoT sensor data. The system continuously collects and stores environmental parameters including air temperature, air humidity, soil moisture, soil pH, soil temperature, soil electrical conductivity (EC), pH value and light intensity. To transform this raw data into actionable insights, Grafana dashboards are organized into three main visualization components:

i. Time-series graphs

ii. Soil health index (SHI)

iii. Correlation heat maps

6.5.1 Time-Series Graphs for Individual Parameters

The first visualization component presents each environmental parameter in the form of a time-series graph. These graphs plot parameter values against a temporal axis, allowing farmers to observe fluctuations, identify recurring patterns, and compare variations across different time periods. Such visualizations are essential for detecting anomalies and understanding how specific conditions evolve throughout the cultivation process.

```
1  select
2    $__timeGroup(created_at, $__interval) as time,
3    avg(air_temperature) as "Air Temperature (°C)"
4  from public.sensor_data
5  where $__timeFilter(created_at)
6  group by 1
7  order by 1;
8
```

Figure 6.24: Query that demonstrates how Grafana retrieves and aggregates air temperature readings

For example, figure above showed air temperature readings are retrieved and aggregated through a query executed in Grafana, which produces a continuous line graph illustrating temperature changes over time. Similar queries are applied to other key parameters including humidity, soil moisture, pH, and conductivity, thereby providing farmers with a comprehensive and easily interpretable overview of environmental dynamics within the greenhouse.

Figure 6.25: Air temperature time series graph

By transforming raw data into intuitive visual representations, the time-series graphs enhance situational awareness and support proactive decision-making, ensuring that deviations can be identified and addressed before they negatively impact crop growth.

6.5.2 Soil Health Index (SHI)

To complement the visualization of individual parameters, the system implements a Soil Health Index (SHI) as a composite metric that consolidates multiple soil-related parameters into a single score. The SHI provides a holistic measure of soil condition by incorporating soil moisture, soil conductivity, soil temperature, soil tds, air temperature, light intensity, pH and air humidity. Each parameter is normalized against its respective threshold values, assigned a weighted sub-score, and aggregated to form the final index. Parameters with greater impact on melon growth, such as soil moisture and pH, are assigned higher weights to ensure their influence is reflected in the overall score.

```
  SELECT
    date_trunc('hour', created_at) AS bucket,
    AVG(air_temperature)    AS air_temp,
    AVG(air_humidity)       AS air_humidity,
    AVG(soil_tds)           AS soil_tds,
    AVG(light_intensity)    AS light,
    AVG(soil_moisture)      AS soil_moisture,
    AVG(soil_temperature)   AS soil_temp,
    AVG(soil_conductivity)  AS soil_cond,
    AVG(soil_ph)            AS soil_ph
  FROM sensor_data
  GROUP BY bucket
)
SELECT
  bucket AS "time",
  (
    GREATEST(0, (100 - (ABS(air_temp - 28) / 15.0 * 100))) * 0.2 +
    LEAST(air_humidity, 100) * 0.1 +
    LEAST(soil_moisture, 100) * 0.25 +
    GREATEST(0, (100 - (ABS(soil_ph - 6.5) / 3.0 * 100))) * 0.25 +
    GREATEST(0, (100 - (ABS(soil_temp - 25) / 15.0 * 100))) * 0.1 +
    GREATEST(0, (100 - (soil_tds / 2000.0 * 100))) * 0.05 +
    GREATEST(0, (100 - (soil_cond / 5.0 * 100))) * 0.025 +
    LEAST(light / 1000.0, 100) * 0.025
  ) AS soil_health_index
FROM data
ORDER BY bucket;
```

Figure 6.26: SQL query to compute SHI

The SHI is visualized in Grafana as a time-series graph, where the index is plotted against time. This approach enables farmers to monitor not only the current soil health but also its progression over different cultivation phases. By observing trends, farmers can identify gradual deterioration in soil conditions and take preventive measures before they affect crop growth. For example, a steadily declining SHI curve may indicate progressive nutrient depletion or moisture imbalance that requires corrective intervention.

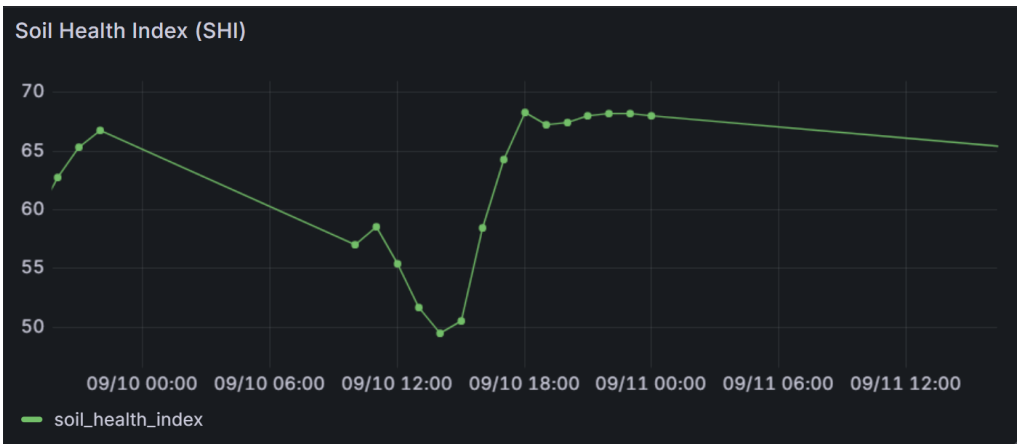


Figure 6.27: Soil Health Index Graph

Compared to analysing individual parameters in isolation, the SHI time-series graph simplifies decision-making by presenting a consolidated indicator of soil quality. This visualization provides farmers with an intuitive tool to assess the overall effectiveness of their soil management practices and supports proactive actions to sustain crop yield and quality.

6.5.3 Correlation Analysis

Correlation analysis was conducted to identify relationships among environmental parameters, enabling farmers to understand how variables interact and influence crop conditions. The computation was performed directly in PostgreSQL using the corr() function, which calculates Pearson correlation coefficients between pairs of parameters stored in the sensor_data table. The SQL query produced a correlation matrix, where each cell represents the degree of association between two variables (e.g., air temperature and humidity, soil moisture and conductivity). This matrix provides a structured dataset that quantifies the strength and direction of parameter relationships.

```sql
SELECT
  'air_temperature' AS variable,
  1.0 AS air_temperature,
  corr(air_temperature, air_humidity) AS air_humidity,
  corr(air_temperature, soil_tds) AS soil_tds,
  corr(air_temperature, light_intensity) AS light_intensity,
  corr(air_temperature, soil_moisture) AS soil_moisture,
  corr(air_temperature, soil_temperature) AS soil_temperature,
  corr(air_temperature, soil_conductivity) AS soil_conductivity,
  corr(air_temperature, soil_ph) AS soil_ph
FROM sensor_data
UNION ALL
```

Figure 6.28: SQL query using corr() function

The results of the correlation query were visualized as a correlation heatmap as shown in figure below, where coefficients are represented using a diverging color scale from strong negative (dark red) to strong positive (dark green), with weaker correlations shown in lighter shades of yellow and orange. As shown, air temperature and air humidity display a strong negative correlation (–0.957), reflecting their inverse relationship, while air temperature and soil temperature exhibit a strong positive correlation (0.730). Soil moisture and soil TDS also demonstrate a moderate positive correlation (0.673), suggesting that increased irrigation may elevate nutrient concentration levels in the soil. By consolidating all pairwise relationships into a single heatmap, the system provides farmers with an intuitive overview of environmental interactions, enabling more informed and data-driven greenhouse management decisions.

Correlation Heatmap

variable	air_temperature	air_humidity	soil_tds	light_intensity	soil_moisture	soil_temperature	soil_conductivity	soil_ph
air_temperature	1	-0.957	0.00318	0.621	-0.0964	0.730	0.00318	0.0366
air_humidity	-0.957	1	0.00551	-0.551	0.0781	-0.771	0.00551	-0.0141
soil_tds	0.00318	0.00551	1	0.180	0.673	-0.0228	1	0.143
light_intensity	0.621	-0.551	0.180	1	0.0956	0.268	0.180	0.0944
soil_moisture	-0.0964	0.0781	0.673	0.0956	1	-0.184	0.673	0.174
soil_temperature	0.730	-0.771	-0.0228	0.268	-0.184	1	-0.0228	-0.0886
soil_conductivity	0.00318	0.00551	1	0.180	0.673	-0.0228	1	0.143
soil_ph	0.0366	-0.0141	0.143	0.0944	0.174	-0.0886	0.143	1

Figure 6.29: Correlation analysis heatmap

CHAPTER 7

System Testing and Evaluation

7.1 Introduction

System testing and evaluation were conducted to ensure that the smart farming system for Japanese melon cultivation operates reliably, meets its functional requirements, and delivers accurate and timely decision support to farmers. The testing phase focused on validating the system's core functionalities, integration of components, data handling, threshold evaluation, visualization, and overall performance. A combination of functional testing, integration testing, and performance testing methods were employed. Functional testing was used to verify individual features such as user authentication, task management, sensor data visualization, and threshold configuration—worked according to specifications. Integration testing ensured that data pipelines between IoT devices, ThingSpeak, Supabase, and the web application functioned seamlessly, with no duplication, loss, or corruption of records. Performance testing was carried out to evaluate responsiveness and efficiency, particularly the system's ability to deliver real-time updates, trigger notifications, and render dashboards within acceptable time limits.

These testing methods were chosen because they collectively provide a comprehensive evaluation of the system's reliability and usability. Functional and integration testing validated correctness and robustness, while performance testing addressed timeliness, which is critical in greenhouse environments where rapid responses to anomalies directly influence crop yield and quality. Together, these methods ensure that the developed system not only functions as intended but also provides a practical, efficient, and farmer-friendly tool for precision agriculture.

7.2 Functional Test Case

Table 7.1: User Sign In Test Case

Test Case#	1	**Test Case Name**	User Sign In		
Test Case Summary	To test if registered users can successfully sign in and access the system.				
Pre-Conditions	User account exists in the database.				
Prepared & Executed By	Liew Ke Ying				
Scenario	**Test Procedure**	**Input Data**	**Expected Outcome**	**Observed Outcome**	**Evaluation (Fail/Pass)**
Valid login credentials	1.Navigate to login page. 2. Enter registered email and correct password. 3. Click “Login”.	Email: farmer01@example.com Password: correct123	System authenticates user and redirects to dashboard.	User successfully logged in and redirected.	Pass
Invalid login credentials	1. Navigate to login page. 2. Enter registered email with incorrect password.	Email: farmer01@example.com Password: wrong	System rejects login attempt and displays error	Error message shown: “Invalid credentials.”	Pass

	3. Click “Login”.		message.		
Empty fields	1. Leave email and/or password field empty. 2. Click “Login”	Email: - Password: -	System prompts user to fill required fields.	Validation message displayed.	Pass

Table 7.2: Add New User Test Case

Test Case#	2	**Test Case Name**	Add new user		
Test Case Summary	To test if admin can add a new user to the system.				
Pre-Conditions	Admin is logged in with role-based access.				
Prepared & Executed By	Liew Ke Ying				
Scenario	**Test Procedure**	**Input Data**	**Expected Outcome**	**Observed Outcome**	**Evaluation (Fail/Pass)**
Add valid user	1. Admin press to ‘Add User’ button. 2. Enter user’s email and role. 3. Press confirm button.	Email: farmer01@gmail.com Role: Farmer	User created successfully, invite email sent.	User added and email invite received.	Pass

Add with invalid email	1. Enter invalid email format. 2. . Click “Create new user” button.	Email: farmer01@wrong Role: Farmer	System rejects displays error message.	Error message shown: “Invalid email format.”	Pass
Duplicate email	1. Enter email that already exists. 2. Confirm	Email: farmer01@gmail.com Role: Farmer	System prevents duplicate creation.	Error message “User already exists” will be displayed.	Pass

Table 7.3: Configure Sensor Data Threshold Test Case

Test Case#	3	**Test Case Name**	Configure Sensor Data Threshold		
Test Case Summary	To test if admin can add/update threshold values and suggestion messages.				
Pre-Conditions	Admin logged in; parameter_thresholds table available.				
Prepared & Executed By	Liew Ke Ying				
Scenario	**Test Procedure**	**Input Data**	**Expected Outcome**	**Observed Outcome**	**Evaluation (Fail/Pass)**
Update	1. Select parameter “Air Temperature”.	Optimal Max = 32°C ->	New threshold will	New threshold	Pass

threshold	2. Update values. 3. Save.	35°C	be displayed.	value is updated and displayed.	
Invalid input	1. Enter empty values. 2. Save	Soil pH = -	System rejects and prompts error.	Error message " Invalid input" displayed	Pass

Table 7.4: Task Management Test Case

Test Case#	4	**Test Case Name**	Task Management		
Test Case Summary	To test create, update, and delete tasks in the calendar.				
Pre-Conditions	User logged in; Calendar module active.				
Prepared & Executed By	Liew Ke Ying				
Scenario	**Test Procedure**	**Input Data**	**Expected Outcome**	**Observed Outcome**	**Evaluation (Fail/Pass)**
Create new task	1. Open model. 2. Input task details. 3. Save	Title: "Irrigation Check" Time: 10:00-11:00	Task saved and shown in calendar.	Task created successfully.	Pass

Update task	1. Select existing task. 2. Change time. 3. Save.	Update to 09:00-10:00	Task updated in DB and calendar refreshed.	Updated successfully.	Pass
Delete task	1. Select task 2. Press Delete Button	"Irrigation Check"	Task removed from DB and calendar.	Task deleted successfully.	Pass

Table 7.5: View, Sort and Search Sensor Data Table Test Case

Test Case#	5	**Test Case Name**	View, sort and search sensor data table		
Test Case Summary	To test whether sensor readings are displayed and can be filtered.				
Pre-Conditions	Sensor data available in DB				
Prepared & Executed By	Liew Ke Ying				
Scenario	**Test Procedure**	**Input Data**	**Expected Outcome**	**Observed Outcome**	**Evaluation (Fail/Pass)**
Load sensor readings	1. Users click the sensor data navigation page.	Existing sensor data entries.	Data displayed in table format.	Displayed correctly.	Pass

Sort entries	1. Users press up sort button on entry id column. 2. Users then press down sort button on entry id column.	N/A	Column entry id is sorted in ascending then descending.	Sorted correctly.	Pass
Filter entries.	1. Users type an entry id in the search bar.	1034	Sensor data with entry id “1034” will be displayed.	Sensor data row retrieves and display correctly.	Pass

Table 7.6: View and Filter by Date on Time-Series Graph Test Case

Test Case#	6	**Test Case Name**	View and Filter by Date on Time-Series Graph		
Test Case Summary	To test whether the time-series graph displays parameter trends and supports filtering by date range.				
Pre-Conditions	Sensor data available in sensor_data table.				
Prepared & Executed By	Liew Ke Ying				
Scenario	**Test Procedure**	**Input Data**	**Expected Outcome**	**Observed Outcome**	**Evaluation (Fail/Pass)**

Load full graph	1. Navigate to dashboard. 2. Select time-series graph for "Air Temperature".	Sensor data over 1 week (by default)	Graph plotted with data points over full period.	Displayed correctly.	Pass
Apply date filter	1. Select date filter range. 2. Apply filter.	Start=2025-09-01, End=2025-09-07.	Graph updates to show data only in selected range.	Graph filtered successfully.	Pass

Table 7.7: View and Filter by Date on Correlation Heatmap Test Case

Test Case#	7	**Test Case Name**	View and Filter by Date on Correlation Heatmap		
Test Case Summary	To test whether the correlation heatmap updates correctly when filtered by date range.				
Pre-Conditions	Historical sensor data available.				
Prepared & Executed By	Liew Ke Ying				
Scenario	**Test Procedure**	**Input Data**	**Expected Outcome**	**Observed Outcome**	**Evaluation (Fail/Pass)**
Default heatmap	1. Open correlation heatmap view.	Default time range	Heatmap generated with correct	Displayed correctly.	Pass

			correlation values.		
Apply date filter	1. Select custom range. 2. Apply filter. 3. Click confirm button.	Start=2025-09-01, End=2025-09-07.	Heatmap recalculated for selected range.	Updated correctly.	Pass

Table 7.8: View and Filter by Date on Soil Health Index (SHI) Test Case

Test Case#	8	**Test Case Name**	View and Filter by Date on Soil Health Index (SHI)		
Test Case Summary	To test whether SHI time-series graph updates correctly with date filters.				
Pre-Conditions	SHI calculation configured in Grafana.				
Prepared & Executed By	Liew Ke Ying				
Scenario	**Test Procedure**	**Input Data**	**Expected Outcome**	**Observed Outcome**	**Evaluation (Fail/Pass)**
Load SHI graph	1. Users navigate to dashboard page.	Default time range	SHI graph generated with composite index.	Displayed correctly.	Pass

Filter SHI by date	1. Select custom range. 2. Apply filter. 3. Click confirm button.	Data from last 3 days.	Graph updates with SHI values in selected period.	Updated correctly.	Pass

Table 7.9: View Latest Sensor Values Test Case

Test Case#	9	**Test Case Name**	View latest sensor values		
Test Case Summary	To test whether the dashboard displays the most recent sensor readings.				
Pre-Conditions	New sensor entry inserted into sensor_data.				
Prepared & Executed By	Liew Ke Ying				
Scenario	**Test Procedure**	**Input Data**	**Expected Outcome**	**Observed Outcome**	**Evaluation (Fail/Pass)**
Display latest data	1. Users navigate to dashboard page.	N/A	Latest values of sensor parameter displayed in gauge form.	Displayed correctly.	Pass
Auto refresh	1. Wait until new entry inserted.	Entry updated in	Dashboard	Refreshed	Pass

latest value	2. Observe dashboard refresh.	ThingSpeak.	refreshed with newest reading.	correctly.	

Table 7.10: Admin Deactivate User Test Case

Test Case#	10	**Test Case Name**	Admins deactivate user		
Test Case Summary	To test whether admin can deactivate a user account.				
Pre-Conditions	User exists in system; admin logged in.				
Prepared & Executed By	Liew Ke Ying				
Scenario	**Test Procedure**	**Input Data**	**Expected Outcome**	**Observed Outcome**	**Evaluation (Fail/Pass)**
Deactivate user	1. Navigate to User Management. 2. Select user. 3. Set user status to false.	User: Farmer02@gmail.com	User marked inactive in database	User successfully deactivated.	Pass
Login after deactivation	1. User attempt login with deactivated account.	User: Farmer02@gmail.com	System rejects login and shows error.	Login blocked successfully.	Pass

Table 7.11: Admin Change User Role Test Case

Test Case#	11	**Test Case Name**	Admins change user role		
Test Case Summary	To test whether admin can update an existing user's role.				
Pre-Conditions	User exists in system; admin logged in.				
Prepared & Executed By	Liew Ke Ying				
Scenario	**Test Procedure**	**Input Data**	**Expected Outcome**	**Observed Outcome**	**Evaluation (Fail/Pass)**
Change user role	1. User navigate to user management. 2. Select user to change roles. 3. Change user role to admin. 4. Click 'Confirm' button.	User: Farmer02@gmail.com New role: Admin	User role updated in database and reflected in user management page.	Updated successfully.	Pass

7.3 Integration Test Case

Table 7.12: Fetch and Insert New Data Test Case

Test Case#	1	**Test Case Name**	Fetch and Insert New Data
Test Case	To test if the Edge Function fetches new ThingSpeak data and inserts into Supabase.		

Summary					
Pre-Conditions	Supabase sensor_data table is created.				
Prepared & Executed By	Liew Ke Ying				
Scenario	**Test Procedure**	**Input Data**	**Expected Outcome**	**Observed Outcome**	**Evaluation (Fail/Pass)**
When Supabase sensor_data table is empty.	1. Trigger the Edge Function using supabase dashboard. 2. Verified if the new data is inserted into the sensor_data table.	ThingSpeak Channel with new entry_id values	New entries inserted into Supabase sensor_data table	New entries inserted into Supabase sensor_data table	Pass
When Supabase sensor_data table already contains previous entries.	1. Trigger the Edge Function again after some rows already exist. 2. Verify if only new entries (greater entry_id) are appended, without duplicates.	Only new rows are inserted; no duplication of existing entries	Only new rows inserted successfully without duplicates	Only new rows inserted successfully without duplicates	Pass
When ThingSpeak channel has no	1. Trigger the Edge Function when ThingSpeak data is unchanged. 2. Check if no additional rows are added in	ThingSpeak channel without new data	No new rows are inserted; table remains unchange	No new rows were inserted	Pass

new entries since the last fetch.	Supabase.				
When ThingSpeak channel data has missing fields (e.g., null values in some sensors).	1. Trigger the Edge Function with entries having null fields. 2. Verify how Supabase stores incomplete records.	ThingSpeak channel entry with missing field values	Data inserted with null values preserved in corresponding columns	Data inserted with null values stored as expected	Pass

Table 7.13: Scheduled Data Fetch and Insert Test Case

Test Case#	2	**Test Case Name**	Scheduled Data Fetch and Insert		
Test Case Summary	To test if the Cron Job automatically triggers the Edge Function to fetch and insert new sensor data every 15 minutes.				
Pre-Conditions	Cron Job is configured in Supabase to run at every 15 minutes.				
Prepared & Executed By	Liew Ke Ying				
Scenario	**Test Procedure**	**Input Data**	**Expected Outcome**	**Observed Outcome**	**Evaluation (Fail/Pass)**

Scheduled Cron Job execution inserts new ThingSpeak data into Supabase	1. Wait for the Cron Job to trigger at the 15-minute schedule. 2. Verify if new data from ThingSpeak is inserted into the sensor_data table in Supabase.	ThingSpeak Channel with new entry_id values added between the last job and the current run	Cron Job triggers Edge Function, and new entries are automatically inserted into sensor_data table without manual intervention.	New entries successfully inserted into sensor_data table after 15 minutes.	Pass

Table 7.14: Fetch Current Weather Data Test Case

Test Case#	3	**Test Case Name**	Fetch Current Weather Data from OpenWeather API		
Test Case Summary	To test if the frontend successfully fetches live weather data from the OpenWeather API and displays it correctly.				
Pre-Conditions	A valid OpenWeather API key is configured in the frontend. Internet connection is available				
Prepared & Executed By	Liew Ke Ying				
Scenario	**Test Procedure**	**Input Data**	**Expected Outcome**	**Observed**	**Evaluation**

				Outcome	**(Fail/Pass)**
Weather API fetch and display validation	1. Open the frontend page with weather display. 2. Trigger the fetch request to OpenWeather API. 3. Observe whether weather data is displayed.	City = Sungai Long, Valid API Key	Weather data is successfully fetched from OpenWeather API and displayed correctly in frontend.	Weather data is fetched and displayed correctly in frontend UI.	Pass

Table 7.15: Test Notifications Test Case

Test Case#	4	**Test Case Name**	Test Notifications		
Test Case Summary	To test if the notifications able to receive successfully.				
Pre-Conditions	(i) Supabase device_tokens table is populated with valid device tokens. (ii) Firebase Cloud Messaging (FCM) service is configured correctly in backend. (iii) Application client is installed on a device, and notifications are enabled.				
Prepared & Executed By	Liew Ke Ying				
Scenario	**Test Procedure**	**Input Data**	**Expected Outcome**	**Observed Outcome**	**Evaluation (Fail/Pass)**

When a parameter value exceeds the warn_max threshold.	1. Insert a test sensor reading into sensor_data that exceeds threshold. 2. Verify if a notification is sent to registered device(s).	Sensor reading: air_temperature = 45°C (threshold max = 35°C)	Push notification is sent: "Alert: Air Temperature too high. Please ventilate."	Notification received successfully on client device	Pass
When a parameter value drops below the warn_min threshold.	1. Insert a test reading below threshold. 2. Verify notification.	Sensor reading: soil_moisture = 5% (threshold min = 15%)	Push notification is sent: "Alert: Soil moisture too low. Consider irrigation."	Notification received successfully	Pass
When parameter is within the optimal range.	1. Insert a normal reading. 2. Check if no unnecessary notification is triggered.	Sensor reading: soil_pH = 6.8 (within 6.5–7.0)	No notification should be sent.	No notification triggered	Pass

7.4 Data Handling and Accuracy

A key aspect of system testing was to ensure that sensor data transmitted from the IoT devices and first ingested into ThingSpeak was correctly synchronized into the Supabase database without loss, duplication, or corruption. Since Supabase serves as the primary data repository for analysis and visualization, maintaining accurate and reliable data transfer from ThingSpeak was essential for system functionality.

	A	B	C	D	E	F	G	H	I	J
80	2025-09-10 13:59:37 UTC	3474	27.3	94.7	37	0.6	29.3	27.1	74	5.7
81	2025-09-10 14:16:06 UTC	3475	27.3	94.7	37	0.6	29.2	27	74	5.7
82	2025-09-10 14:50:10 UTC	3476	27.3	95.4	36	0.6	29	26.8	72	5.8
83	2025-09-10 15:06:36 UTC	3477	27.3	95.4	36	0.6	28.9	26.8	72	5.8
84	2025-09-10 15:23:07 UTC	3478	27.3	95.4	35.5	0.6	28.9	26.7	71	5.8
85	2025-09-10 15:39:34 UTC	3479	27.3	95.4	35.5	0.6	28.8	26.6	71	5.7
86	2025-09-10 15:56:08 UTC	3480	26.9	95.4	35	0.6	28.6	26.6	70	5.7
87	2025-09-10 16:12:36 UTC	3481	27.3	95.4	35	0.6	28.6	26.5	70	5.7
88	2025-09-10 16:29:03 UTC	3482	27.3	95.4	34.5	0.6	28.6	26.5	69	5.7
89	2025-09-10 16:45:32 UTC	3483	27.3	95.4	34.5	0.6	28.5	26.4	69	5.7
90	2025-09-14 05:39:29 UTC	3484	29.5	91.8	0	8243.3	25.3	29.6	0	3.6
91	2025-09-14 05:52:36 UTC	3485	36.6	69.6	249.5	1974.8	93.5	27.5	499	4.9
92	2025-09-14 06:09:04 UTC	3486	34.1	69.6	158.5	4370.3	73.8	28.4	317	5.2
93	2025-09-14 06:25:33 UTC	3487	35.1	71.8	145.5	7355.3	69.9	28.6	291	5.4
94	2025-09-14 06:42:09 UTC	3488	40.3	62.5	137.5	7288.1	68.2	28.9	275	5.5
95	2025-09-14 06:58:34 UTC	3489	45.8	55.3	134	3427.5	67	29.3	268	5.5
96	2025-09-14 07:14:58 UTC	3490	40.2	61.8	131	1450	66	29.6	262	5.5
97	2025-09-14 07:31:24 UTC	3491	35.6	69.6	130	1159.8	65.3	29.8	260	5.5
98	2025-09-14 07:47:51 UTC	3492	34.6	73.9	129	1220.3	64.9	29.8	258	5.5

Figure 7.1: Thingspeak's sensor data

entry_id int4	created_at timestamp	air_temp... nu...	air_humidity numeric	soil_tds numeric	light_intensity numeric	soil_moisture numeric	soil_temperature numeric	soil_conductivity numeric	soil_ph numeric	insert_date_time times
3474	2025-09-10 13:59:37	27.30	94.70	37.00	0.60	29.30	27.10	74.00	5.70	2025-09-10 14:00:02.8
3475	2025-09-10 14:16:06	27.30	94.70	37.00	0.60	29.20	27.00	74.00	5.70	2025-09-10 14:30:02.4
3476	2025-09-10 14:50:10	27.30	95.40	36.00	0.60	29.00	26.80	72.00	5.80	2025-09-10 15:00:02.7
3477	2025-09-10 15:06:36	27.30	95.40	36.00	0.60	28.90	26.80	72.00	5.80	2025-09-10 15:15:02.2
3478	2025-09-10 15:23:07	27.30	95.40	35.50	0.60	28.90	26.70	71.00	5.80	2025-09-10 15:30:01.7
3479	2025-09-10 15:39:34	27.30	95.40	35.50	0.60	28.80	26.60	71.00	5.70	2025-09-10 15:45:02.4
3480	2025-09-10 15:56:08	26.90	95.40	35.00	0.60	28.60	26.60	70.00	5.70	2025-09-10 16:00:04.0
3481	2025-09-10 16:12:36	27.30	95.40	35.00	0.60	28.60	26.50	70.00	5.70	2025-09-10 16:15:02.4
3482	2025-09-10 16:29:03	27.30	95.40	34.50	0.60	28.60	26.50	69.00	5.70	2025-09-10 16:30:02.8
3483	2025-09-10 16:45:32	27.30	95.40	34.50	0.60	28.50	26.40	69.00	5.70	2025-09-10 17:00:03.4
3484	2025-09-14 05:39:29	29.60	91.80	NULL	8243.30	25.30	29.60	NULL	3.60	2025-09-14 05:45:03.0
3485	2025-09-14 05:52:36	36.60	69.60	249.50	1974.80	93.50	27.50	499.00	4.90	2025-09-14 06:00:03.1
3486	2025-09-14 06:09:04	34.10	69.60	158.50	4370.30	73.80	28.40	317.00	5.20	2025-09-14 06:15:02.9
3487	2025-09-14 06:25:33	35.10	71.80	145.50	7355.30	69.90	28.60	291.00	5.40	2025-09-14 06:30:02.0
3488	2025-09-14 06:42:09	40.30	62.50	137.50	7288.10	68.20	28.90	275.00	5.50	2025-09-14 06:45:03.5
3489	2025-09-14 06:58:34	45.80	55.30	134.00	3427.50	67.00	29.30	268.00	5.50	2025-09-14 07:00:02.8
3490	2025-09-14 07:14:58	40.20	61.80	131.00	1450.00	66.00	29.60	262.00	5.50	2025-09-14 07:15:02.4
3491	2025-09-14 07:31:24	35.60	69.60	130.00	1159.80	65.30	29.80	260.00	5.50	2025-09-14 07:45:02.1
3492	2025-09-14 07:47:51	34.60	73.90	129.00	1220.30	64.90	29.80	258.00	5.50	2025-09-14 08:00:04.1

Figure 7.2: Supabase sensor data table

The test focused on three main areas: data integrity and completeness. Data integrity was evaluated by comparing random samples of sensor readings recorded in ThingSpeak with those retrieved from the Supabase sensor_data table. Figure 7.5.1 illustrates an example of sensor readings as displayed in the

ThingSpeak channel, while Figure 7.5.2 shows the corresponding entries stored in Supabase after synchronization. The comparison revealed that all sampled values matched exactly across both platforms, resulting in an accuracy rate of 100%. This confirms that the synchronization process preserved the integrity of the sensor data without any corruption or modification during transfer.

```
64  WITH missing AS (
65      SELECT s1.entry_id + 1 AS missing_id
66      FROM sensor_data s1
67      LEFT JOIN sensor_data s2
68        ON s1.entry_id + 1 = s2.entry_id
69      WHERE s2.entry_id IS NULL
70  )
71  SELECT
72      (SELECT COUNT(*) FROM sensor_data) AS total_rows,
73      (SELECT COUNT(*) FROM missing) AS missing_rows,
74      ROUND(100.0 * ( (SELECT COUNT(*) FROM sensor_data)::numeric /
75                  ((SELECT COUNT(*) FROM sensor_data) + (SELECT COUNT(*) FROM missing)) ), 2) AS completeness_percent
76  ;
77
78
```

Results | Chart | Export | Source

total_rows	missing_rows	completeness_percent
1603	3	99.81

Figure 7.3: SQL to count completeness percentage

Completeness was verified by checking the sequence of entry_id values in the sensor_data table. Since ThingSpeak generates entries sequentially, any missing IDs would indicate a skipped or lost record. Out of a total of 1,603 rows, only 3 IDs were missing, representing a data loss rate of approximately 0.19% as shown in figure above. The results confirm that the data synchronization process between ThingSpeak and Supabase was highly reliable. The small discrepancy is likely due to temporary network or synchronization delays and is acceptable within the scope of this project.

7.5 Visualization and Analytics

The purpose of this test was to verify that the visualization and analytical components of the system accurately represented the data stored in Supabase and provided meaningful insights for farm management. Testing was carried out in three areas: time-series graphs in Grafana, Soil Health Index computation, and correlation analysis using heat maps.

For the Grafana dashboards, selected parameters such as air temperature and soil moisture were compared between raw database queries and their corresponding visualizations. The results confirmed that the plotted values aligned with the underlying data, ensuring that farmers could reliably observe environmental trends over time.

The Soil Health Index was tested by inserting controlled sample values into the database and verifying that the calculated index corresponded with expected soil conditions (e.g., optimal when all parameters were within defined thresholds, low when moisture and pH dropped below the minimum range).

Similarly, the correlation heat map was evaluated by analysing pairs of parameters with known relationships; for instance, soil moisture and conductivity were positively correlated, while air temperature and humidity displayed an inverse relationship.

7.6 Threshold Evaluation and Suggestions

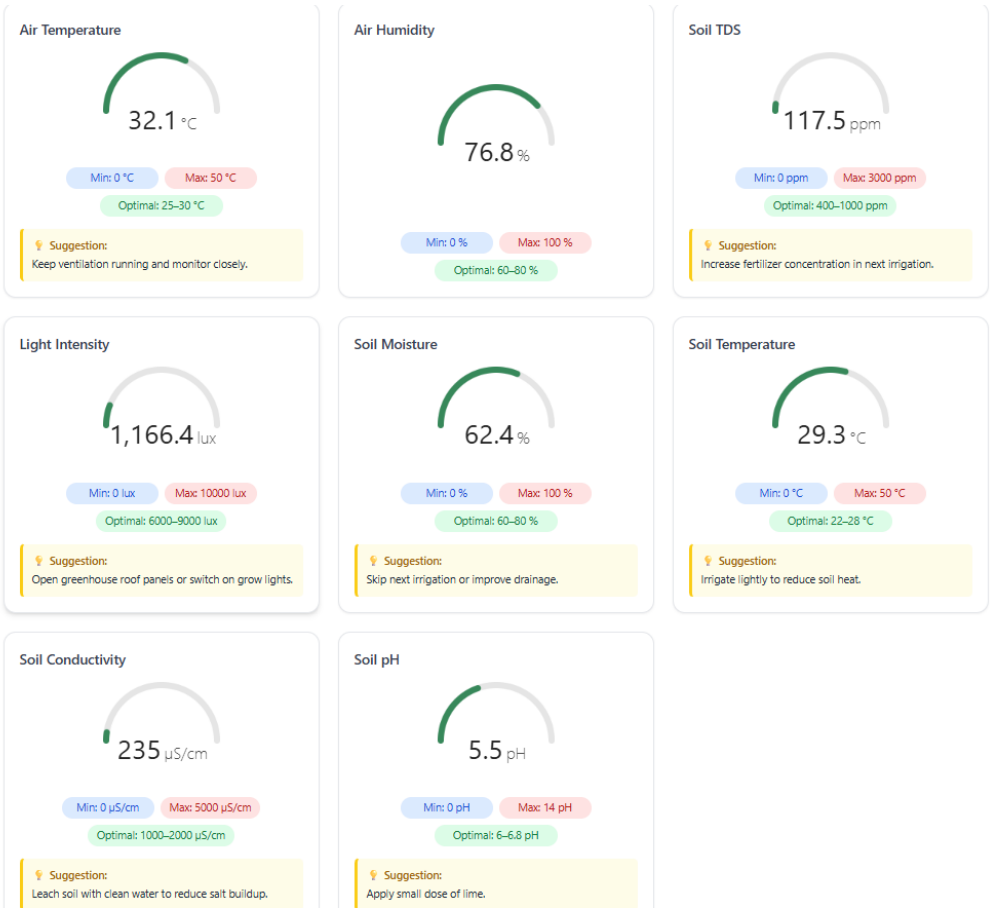


Figure 7.4: Angular dashboard displaying a suggestion

Results Chart Export

parameter	latest_value	optimal_min	optimal_max	warn_min	warn_max	expected_suggestion
air_temperature	32.10	24	32	22	35	Keep ventilation running and monitor closely.
air_humidity	76.80	65	80	55	85	Within optimal range
soil_tds	117.50	1000	1600	800	2000	Increase fertilizer concentration in next irrigation.
light_intensity	1166.40	25000	32000	20000	35000	Open greenhouse roof panels or switch on grow lights.
soil_moisture	62.40	25	35	20	40	Skip next irrigation or improve drainage.
soil_temperature	29.30	24	28	22	32	Irrigate lightly to reduce soil heat.
soil_conductivity	235.00	1.5	2.2	1.2	2.5	Leach soil with clean water to reduce salt buildup.
soil_ph	5.50	5.8	6.8	5.5	7.2	Apply small dose of lime.

Figure 7.5: Test results

```sql
WITH latest_readings AS (
    SELECT
        air_temperature, air_humidity, soil_tds, light_intensity,
        soil_moisture, soil_temperature, soil_conductivity, soil_ph,
        created_at
    FROM sensor_data
    ORDER BY entry_id DESC
    LIMIT 1
)
-- Air Temperature
SELECT
    'air_temperature' AS parameter,
    l.air_temperature AS latest_value,
    t.optimal_min, t.optimal_max, t.warn_min, t.warn_max,
    CASE
        WHEN l.air_temperature < t.warn_min THEN t.low_suggestion
        WHEN l.air_temperature > t.warn_max THEN t.high_suggestion
        WHEN l.air_temperature BETWEEN t.warn_min AND t.optimal_min THEN t.warn_low_suggestion
        WHEN l.air_temperature BETWEEN t.optimal_max AND t.warn_max THEN t.warn_high_suggestion
        ELSE 'Within optimal range'
    END AS expected_suggestion
FROM latest_readings l
JOIN parameter_thresholds t ON t.parameter = 'air_temperature'

UNION ALL
```

Figure 7.6: SQL to retrieve test results

This test was conducted to verify that the system correctly evaluated sensor readings against the predefined threshold values stored in the parameter_thresholds table and generated the appropriate suggestions. The latest sensor values from the sensor_data table were retrieved and compared with the optimal and warning ranges for each parameter. The SQL query above was executed to join the most recent sensor readings with their corresponding threshold definitions, automatically determining the expected suggestion for each case. For example, when the latest soil moisture reading fell below the warn_min value, the system correctly generated the suggestion to increase irrigation, while higher-than-expected air temperature values triggered recommendations to improve greenhouse ventilation. Above table in the figure summarizes the results of this test, showing the latest sensor values, the relevant threshold ranges, and the expected suggestions. The confirmed that the threshold evaluation logic functioned consistently across all parameters, with the generated suggestions matching the corrective actions defined in the database. This demonstrates that the system provides farmers with timely and context-specific guidance, enabling proactive interventions to optimize melon cultivation conditions.

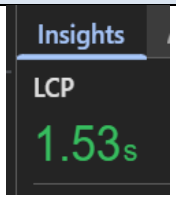
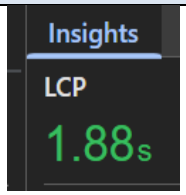
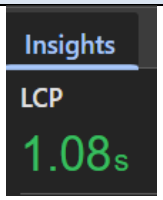
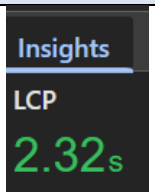
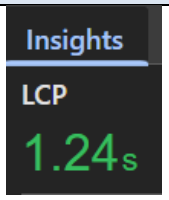
7.7 Performance Testing

Performance testing was conducted to evaluate the responsiveness and efficiency of the Smart Farming System. The goal was to measure how quickly the system reacts to sensor updates, processes notifications, and retrieves data for users. Since the system involves real-time monitoring, timely updates and alerts are critical to ensure farmers can respond promptly to abnormal farming conditions.

7.7.1 App Start Time

App Start Time testing was conducted to evaluate how quickly the web application loads and displays the dashboard after being launched. This test is important because loading speed directly affects user experience and system usability. In farming operations, where users often need to access the dashboard quickly to view real-time sensor readings, a delay in loading may hinder timely decision-making. To ensure reliability, the Largest Contentful Paint (LCP) metric was selected as the primary performance indicator. LCP, provided by Microsoft Edge DevTools, measures the time when the main content of a page becomes visible to the user. Since the dashboard is the central interface for monitoring greenhouse conditions, using LCP makes it a reliable representation of perceived load time and overall responsiveness of the application.

The testing process followed a systematic methodology. The web application was first opened in InPrivate (Incognito) mode to prevent cached data from influencing the measurement. The Microsoft Edge DevTools were then launched, and under the Performance tab, the Web Vitals feature was used to capture the LCP value. After refreshing the dashboard, the LCP time was recorded for each run. To increase the accuracy and consistency of the measurement, the process was repeated five times under the same conditions. Finally, the average App Start Time was calculated by dividing the sum of all LCP values by the number of runs. This approach ensured that the reported result reflected a consistent and reliable measure of system performance.

Test 1	Test 2	Test 3	Test 4	Test 5
				

Formula used:

$$\text{Average App Start Time} = \frac{Sum\ of\ LCP\ values}{Number\ of\ runs}$$

$$= \frac{1.53+1.88+1.08+2.32+1.24}{5}$$

$$= 1.554\ seconds$$

The results show that the application consistently loads within 1.554 seconds, which is well below the target threshold of 5 seconds for acceptable user experience. This indicates that the system is optimized and efficient in rendering the dashboard interface.

CHAPTER 8

CONCLUSION AND RECOMMENDATIONS

8.1 Overview

This chapter concludes the study by revisiting the objectives and outcomes of the web-based smart IoT system for Japanese melon farming, which was designed to improve efficiency, consistency, and crop quality through real-time monitoring and data-driven insights. The chapter first presents the research findings, evaluating how the system achieved its objectives. It then discusses the problems encountered during development and testing, highlighting both technical and coordination challenges faced along the way. This is followed by a review of the limitations of the project. Finally, the chapter outlines recommendations for future improvements and enhancements to ensure scalability, usability, and long-term effectiveness, reaffirming the system's potential to contribute to smart and sustainable melon farming practices.

8.2 Research Findings

This section reviews how the project's objectives were achieved by evaluating the outcomes of the developed system. It highlights how the IoT platform, data visualization tools, and automated alerts addressed the key challenges of Japanese melon farming, demonstrating the system's effectiveness and potential to improve efficiency, resource management, and crop quality.

8.2.1 Objectives 1: To develop a web-based IoT system for real-time monitoring of environmental parameters in Japanese melon farming.

The project successfully achieved this objective by designing and implementing a web-based platform that integrates IoT devices with cloud storage and visualization tools. Environmental parameters were collected and displayed real time in Grafana dashboard. This testing results also

demonstrates that the system is capable of providing reliable and continuous monitoring of environmental conditions in Japanese melon cultivation.

8.2.2 Objectives 2: To develop and integrate a data-driven analytics pipeline with visualization and analysis

This objective was met through the integration of Grafana into the system, enabling powerful data visualization and analytics. Time-series graphs were employed to illustrate fluctuations in key parameters, helping farmers identify environmental patterns over time. A Soil Health Index was developed by combining soil-related parameters into a single metric, providing farmers with a simplified yet comprehensive view of soil conditions. In addition, correlation heat maps were generated to highlight relationships between parameters.

8.2.3 Objectives 3: To enhance farming yield and crop quality by implementing automated alerts and suggestions based on parameter thresholds.

The system effectively addressed this objective by embedding a parameter threshold mechanism in the backend. Threshold values for each parameter were stored in the database and evaluated in real time by the Spring Boot application. When readings fell outside of the defined optimal ranges, the system automatically generated corrective suggestions. In addition, Firebase Cloud Messaging (FCM) was integrated to deliver instant push notifications to users, ensuring that farmers were alerted to anomalies without needing to constantly monitor the dashboard.

Although the system was not deployed continuously throughout a full cultivation cycle, it was tested under real greenhouse conditions and demonstrated functional reliability. During the evaluation period, sensor anomalies were detected correctly, and corresponding alerts and suggestions were generated as expected. Preliminary trials also resulted in the successful cultivation of four Japanese melons, indicating that the threshold-based mechanism and alert system can support farmers in maintaining stable growth conditions. While the short testing window limited the ability to conclusively

validate long-term yield improvements, the results provide credible evidence that the proposed system can enhance farming practices and contribute to more consistent crop quality when applied across multiple growth cycles.

8.3 Problem Encountered

During the development and testing of the smart farming system, several problems were encountered that affected both the technical implementation and project coordination. These challenges and their resolutions are discussed below.

8.3.1 Direct Integration from IoT Gateway to Supabase Cloud Database

The initial plan was for sensor readings to be transmitted straight from the gateway to Supabase; however, due to SIM card incompatibility and connectivity errors, this approach failed to establish a stable communication channel. As a solution, ThingSpeak was introduced as an intermediary platform for data ingestion. This allowed the IoT devices to successfully transmit data, which could then be synchronized with Supabase for structured storage and analysis.

8.3.2 Communication and Coordination with Hardware Team

Another problem encountered was related to communication and coordination with the hardware team members, who were responsible for sensor setup and calibration. Since the project involved multiple team members working on different components, occasional misalignment in timelines and unclear reporting of sensor performance created delays in backend and frontend integration. To address this, regular coordination meetings were established and shared documentation was introduced to streamline communication. This ensured that the hardware data formats, and collection processes were clearly defined, allowing smoother integration with the software components.

8.3.3 Integration Challenges Across Multiple Platforms

Integration challenges were also experienced across the multiple platforms used in the system, namely ThingSpeak, Supabase, Spring Boot, Angular, Firebase, and Grafana. Ensuring compatibility between APIs, authentication mechanisms, and data formats was complex and caused delays during development. Specific issues included CORS errors when connecting Spring Boot to Supabase and Firebase service worker registration failures when enabling push notifications. These problems were resolved through iterative debugging and careful configuration. For example, Spring Boot was updated with appropriate CORS headers to allow secure cross-origin requests, while Firebase documentation was consulted to correct service worker timing errors.

8.3.4 Limited Project Timeline and Testing Scope

The limited project timeline posed another challenge. Due to the constraints of the FYP schedule, the system could not be deployed throughout the entire Japanese melon cultivation cycle. This limited the scope of testing, meaning that while the system demonstrated feasibility and supported one successful melon harvest, its long-term impact on yield and fruit quality could not be conclusively validated. The short timeline, therefore, restricted comprehensive evaluation, and extended deployment across multiple cycles was identified as an important step for future research and system validation.

8.4 Limitations

Although the smart farming system achieved its objectives and demonstrated promising results, several limitations were encountered during development and testing. These limitations provide context for the findings and highlight opportunities for future work.

8.4.1 Partial Deployment Across Cultivation Cycle

Although the smart farming system achieved its objectives and demonstrated promising results, several limitations were encountered during development and testing. These limitations provide context for the findings and highlight opportunities for future work.

8.4.2 Hardware and Connectivity Constraints

Budget limitations restricted the use of more advanced IoT hardware and SIM cards capable of direct integration with cloud databases. As a workaround, ThingSpeak was used as an intermediary data ingestion platform before synchronizing with Supabase. While effective, this introduced additional steps that could affect real-time performance.

8.4.3 Dependence on Internet Connectivity

The system relies heavily on stable internet connectivity for transmitting sensor data, updating dashboards, and sending notifications. In rural or greenhouse environments with unstable networks, system performance and responsiveness may be reduced.

8.4.4 Usability Testing and User Adoption

The system's features were evaluated by the project team but not through extensive farmer-based usability testing. As such, the interface and workflow may need refinement to better align with actual farming practices and user expectations.

8.5 Recommendations

Based on the findings and limitations of this study, the following recommendations are proposed to improve the system and strengthen its impact in future implementations.

8.5.1 Full-Scale Deployment Across Cultivation Cycles

To validate improvements in yield and quality more conclusively, future work should deploy the system over multiple full cultivation cycles. Longitudinal data will allow for statistical evaluation of crop outcomes and help verify whether features such as threshold alerts and the soil health index consistently produce benefits over time.

8.5.2 Improved IoT Hardware and Direct Connectivity

Upgrading to IoT devices and SIM modules capable of direct integration with cloud databases (bypassing intermediate platforms like ThingSpeak) will reduce latency and simplify data flow. Studies have shown that precise data collection and optimized resource use are central to enhancing agricultural efficiency, particularly when connectivity and hardware are reliable. (AL Duguma et al., 2024)

8.5.3 Robustness to Connectivity Disruptions

The system should include mechanisms for offline data caching or local buffering to mitigate the effect of unstable or intermittent internet connection which is a common in rural or farm settings. Ensuring that data is not lost during outages improves reliability and trust in smart farming systems. Research into precision agriculture notes connectivity reliability as a frequent challenge and recommends architectural designs that include redundancy or hybrid connectivity models. (Mohamed Rafi et al., 2025)

REFERENCES

Agarwal, N. (2025) *Firebase vs Supabase: Which Backend Solution Wins?* https://www.wildnetedge.com/blogs/firebase-vs-supabase-which-backend-solution-wins.

AWS IoT-Driven Precision Agriculture | Amazon Web Services (2020). https://aws.amazon.com/blogs/iot/aws-iot-driven-precision-agriculture/.

Bersani, C. *et al.* (2022) 'Internet of Things Approaches for monitoring and control of smart greenhouses in Industry 4.0,' *Energies*, 15(10), p. 3834. https://doi.org/10.3390/en15103834.

Dhanaraju, M. *et al.* (2022) 'Smart Farming: Internet of Things (IoT)-Based Sustainable agriculture,' *Agriculture*, 12(10), p. 1745. https://doi.org/10.3390/agriculture12101745.

Duguma, A.L. and Bai, X. (2024) 'How the internet of things technology improves agricultural efficiency,' *Artificial Intelligence Review*, 58(2). https://doi.org/10.1007/s10462-024-11046-0.

Food and Agriculture Organization of the United Nations (2021) *THE STATE OF FOOD AND FOOD AND AGRICULTURE MAKING AGRI-FOOD SYSTEMS MORE RESILIENT TO SHOCKS AND STRESSES*, *Interacademies*. https://www.interacademies.org/sites/default/files/2021-11/The%20State%20of%20Food%20and%20Agriculture%202021_small.pdf (Accessed: September 17, 2025).

Getahun, S., Kefale, H. and Gelaye, Y. (2024) 'Application of Precision Agriculture Technologies for Sustainable Crop Production and Environmental Sustainability: A Systematic Review,' *The Scientific World JOURNAL*, 2024(1). https://doi.org/10.1155/2024/2126734.

Guidance for Building an Agricultural Sensor Network using IoT and Amazon DocumentDB (no date). https://aws.amazon.com/solutions/guidance/building-an-agricultural-sensor-network-using-iot-and-amazon-documentdb/.

Hong, S. *et al.* (2024) 'Implementation of smart farm systems based on Fog computing in artificial intelligence of things environments,' *Sensors*, 24(20), p. 6689. https://doi.org/10.3390/s24206689.

Huynh, H.X., Tran, L.N. and Duong-Trung, N. (2023) 'Smart greenhouse construction and irrigation control system for optimal Brassica Juncea development,' *PLoS ONE*, 18(10), p. e0292971. https://doi.org/10.1371/journal.pone.0292971.

Kadarabad, M.V., Vakacharla, D.H. and Palani, R.R. (2025) 'Real-Time Soil Health Monitoring with IoT and ThingSpeak Integration,' in *Atlantis highlights in engineering/Atlantis Highlights in Engineering*, pp. 401–408. https://doi.org/10.2991/978-94-6463-754-0_35.

Khanna, A. and Kaur, S. (2020) 'Internet of Things (IoT), Applications and Challenges: A Comprehensive review,' *Wireless Personal Communications*, 114(2), pp. 1687–1762. https://doi.org/10.1007/s11277-020-07446-4.

'Low-cost IoT-Based Smart Notification System for Rural Agriculture' (2022) *ResearchGate* [Preprint]. https://www.researchgate.net/publication/372304589_Low-cost_IoT-Based_Smart_Notification_System_for_Rural_Agriculture.

Mansoor, S. *et al.* (2025) 'Integration of smart sensors and IOT in precision agriculture: trends, challenges and future prospectives,' *Frontiers in Plant Science*, 16. https://doi.org/10.3389/fpls.2025.1587869.

Maraveas, C. *et al.* (2022) 'Applications of IoT for optimized greenhouse environment and resources management,' *Computers and Electronics in Agriculture*, 198, p. 106993. https://doi.org/10.1016/j.compag.2022.106993.

Maraveas, C. and Bartzanas, T. (2021) 'Application of internet of things (IoT) for optimized greenhouse environments,' *AgriEngineering*, 3(4), pp. 954–970. https://doi.org/10.3390/agriengineering3040060.

Mekonnen, Y. *et al.* (2019) 'Review—Machine Learning Techniques in Wireless Sensor Network Based Precision Agriculture,' *Journal of the Electrochemical Society*, 167(3), p. 037522. https://doi.org/10.1149/2.0222003jes.

Monteiro, A., Santos, S. and Gonçalves, P. (2021) 'Precision Agriculture for Crop and Livestock Farming—Brief review,' *Animals*, 11(8), p. 2345. https://doi.org/10.3390/ani11082345.

Mouratiadou, I. *et al.* (2023) 'The Digital Agricultural Knowledge and Information System (DAKIS): Employing digitalisation to encourage diversified and multifunctional agricultural systems,' *Environmental Science and Ecotechnology*, 16, p. 100274. https://doi.org/10.1016/j.ese.2023.100274.

Padhiary, M., Kumar, A. and Sethi, L.N. (2025) 'Emerging technologies for smart and sustainable precision agriculture,' *SPRINGER NATURE*, 1(1). https://doi.org/10.1007/s44430-025-00006-0.

Pathmudi, V.R. *et al.* (2023) 'A systematic review of IoT technologies and their constituents for smart and sustainable agriculture applications,' *Scientific African*, 19, p. e01577. https://doi.org/10.1016/j.sciaf.2023.e01577.

Petraki, D. *et al.* (2025) 'Digital tools and decision support systems in Agroecology: Benefits, challenges, and practical implementations,' *Agronomy*, 15(1), p. 236. https://doi.org/10.3390/agronomy15010236.

Prathibha, S.R., Hongal, A. and Jyothi, M.P. (2017) 'IOT Based Monitoring System in Smart Agriculture,' *IEEE Xplore*, pp. 81–84. https://doi.org/10.1109/icraect.2017.52.

Rafi, M.S.M., Behjati, M. and Rafsanjani, A.S. (2025) *Reliable and Cost-Efficient IoT Connectivity for Smart agriculture: A Comparative Study of LPWAN, 5G, and Hybrid Connectivity models*. https://arxiv.org/abs/2503.11162.

Raj, M. and Prahadeeswaran, M. (2025) 'Revolutionizing agriculture: a review of smart farming technologies for a sustainable future,' *Deleted Journal*, 7(9). https://doi.org/10.1007/s42452-025-07561-6.

Singh, G. and Sharma, S. (2024) 'A comprehensive review on the Internet of Things in precision agriculture,' *Multimedia Tools and Applications* [Preprint]. https://doi.org/10.1007/s11042-024-19656-0.

Singh, N. *et al.* (2024) 'IoT-based greenhouse technologies for enhanced crop production: a comprehensive study of monitoring, control, and communication techniques,' *Systems Science & Control Engineering*, 12(1). https://doi.org/10.1080/21642583.2024.2306825.

Soussi, A. *et al.* (2024) 'Smart Sensors and Smart Data for Precision Agriculture: A review,' *Sensors*, 24(8), p. 2647. https://doi.org/10.3390/s24082647.

Talbott, C. (2022) 'Helping farmers with cloud technology, up close and global,' *Google*, 3 June. https://blog.google/products/google-cloud/helping-farmers-with-cloud-technology-up-close-and-global.

TeamSpace Farm Case Study | Google Cloud (no date). https://cloud.google.com/customers/spacefarm.

THE MINISTER OF AGRICULTURE AND FOOD INDUSTRIES (2021) *NATIONAL AGROFOOD POLICY 2021-2030 (NAP 2.0) Agrofood Modernisation: Safeguarding the future of National Food Security*, *www.kpkm.gov.my*. My Gogoprint Sdn. Bhd. https://www.kpkm.gov.my/images/04-dasar-agromakanan/national_agrofood_policy_2021-2030_nap%202.0.pdf (Accessed: September 17, 2025).

Thilakarathne, N.N. *et al.* (2025) 'Internet of Things enabled smart agriculture: current status, latest advancements, challenges and countermeasures,' *Heliyon*, 11(3), p. e42136. https://doi.org/10.1016/j.heliyon.2025.e42136.

ThingSpeak for Smart Farming - ThingSpeak IoT (no date). https://thingspeak.mathworks.com/pages/smart_farming.

Tratwal, A., Jakubowska, M. and Pietrusińska-Radzio, A. (2025) 'Decision support systems in Integrated pest and Disease Management: Innovative elements in sustainable agriculture,' *Sustainability*, 17(18), p. 8111. https://doi.org/10.3390/su17188111.

Weraikat, D. *et al.* (2024) 'Data Analytics in Agriculture: Enhancing Decision-Making for crop yield optimization and sustainable practices,' *Sustainability*, 16(17), p. 7331. https://doi.org/10.3390/su16177331

Project title:	Mobile Based Smart IoT-Based System for Optimized Japanese Melon Farming: A Data-Driven Approach to Enhance Yield and Quality
Student Name:	LIEW KE YING
Supervisor Name:	Sugumaran a/l Nallusamy
Examiner Name:	Lee Chen Kang

Key Assessment for Project Proposal	Supervisor Comments/Remarks	Examiner Comments/Remarks
Project Description - Is the problem or need to be addressed clearly presented? - Is the proposed approach or solution clearly presented and justified?	Nil	The problem statements are clearly defined. More elaborations should be provided if possible. The propose solutions are OK.
Project Scope and Objectives - Is the scope of the project clearly defined? - Are the objectives of the project clearly specified? - Are the project scope and objectives appropriate for a final year project?	Refine the objective	Recommend to focus on only 3 objectives rather than 6 objectives The project scope is sufficient.
Literature Review / Fact Finding for Benchmarking / Verification of Project - Are sources for literature review / fact finding appropriate? - Is information from literature review / fact finding relevant and adequate? - Is information from literature review / fact finding clearly presented and discussed?	Nil	The sources for literature reviews needs to be properly cited. The literature review contains very few in-text citations, and this section requires significant improvement to meet academic standards.
Research/Development Methodology and Development Tools - Is the methodology for the project clearly described and discussed? - Are the required development tools clearly described and discussed? - Are the stated methodology and development tools appropriate?	Nil	The development methodology is clearly described. The ER diagrams and acitvity diagrams are missing. Do put the diagrams inside as part of the design specification.
Project Plan - Are the phases and tasks of the project properly defined and planned? - Are the phases and tasks consistent with the methodology of the project?	Nil	The phases and tasks of the project is properly defined and planned.
Initial Deliverables - Are deliverables (e.g. use case diagrams and descriptions) of initial phases of the project plan included in the report?	Nil	The initial deliverables are approriate and sufficient.
Report Structure and References - Is the report organised in a logical structure? - Are references listed in accordance to Harvard format?	Nil	[redacted]
Language and Clarity of Writing - Are the sentences concise and understandable? - Are there spelling and grammar issues?	Nil	

Appendix 1: FYP1 feedback