

**Jordan University of Science and Technology**

**College of Computer Sciences & Information Technology**

**Smart Agri-Guard**

*A project submitted*

*in partial fulfillment of the requirements for the degree of*

*Bachelor in Software Engineering*

**by**

Mutasem Mustafa Al-Gharibeh (162289)

Batool Saber Abu Zaid (157902)

Osama Rad Al-Omari (161413)

Danial Ziad Yaseen (162683)

Naser Hani Mousa (164849)

**Supervised by**

Malik G. Qasaimeh

**Committee Member Names**

Hamza Alkofahi

Hasan Albzour

**March-2025**

## Undertaking

This is to declare that the project entitled “**Smart Agri-Guard**” is an original work done by undersigned, in partial fulfillment of the requirements for the degree “Bachelor in Software Engineering” at Software Engineering Department, College of Computer and Information Technology, Jordan University of Science and Technology.

All the analysis, design and system development have been accomplished by the undersigned. Moreover, this project has not been submitted to any other college or university.

Osama Rad Al-Omari

Naser Hani Mousa

Batool Saber Abu Zaid

Danial Ziad Yaseen

Mutasem Mustafa Al-Gharibeh

Note: sign across your name

## ABSTRACT

This project presents a smart agriculture system designed specifically for nursery management, aiming to monitor and improve plant health using sensor technology and artificial intelligence (AI). The system collects real-time environmental and soil data through ESP32-connected sensors and analyzes it using an AI model to predict plant conditions and recommend actions such as irrigation or fertilization. A mobile application built with Flutter allows farmers and nursery managers to monitor plants, receive alerts, and view historical trends. The backend, developed in ASP.NET Core, provides secure data handling, AI integration, and role-based access control. This solution is motivated by the need for efficient, localized monitoring in greenhouses, especially where resources and expertise are limited. The administrator of the system is the developer team, responsible for managing users, plants, and greenhouses. The primary users are farmers and managers within agricultural nurseries. The project demonstrates how affordable, intelligent systems can enhance crop care and sustainability in controlled environments.

## Acknowledgment

We would like to express our deepest gratitude to everyone who supported us throughout the development of this project.

First and foremost, we extend our sincere thanks to our supervisor,

**Dr. Malik Qasaimeh**, for his valuable guidance, continuous encouragement, and insightful feedback throughout every stage of this project.

We are also grateful to our instructors and the department staff for providing us with the technical foundation and tools needed to complete this work.

Special thanks go to our families and friends for their endless support, patience, and motivation, which helped us stay focused and committed.

Lastly, we would like to acknowledge our teammates for their collaboration, dedication, and collective efforts that made this project possible.

## CHAPTER 1: INTRODUCTION

### Overview

The **Smart Agri-Guard** project is a smart agricultural monitoring system designed to improve greenhouse management using IoT and AI technologies. The system uses sensors to monitor soil moisture, air temperature, humidity, soil acidity, nutrient levels, ensuring optimal growing conditions for plants. The data is analyzed and presented to farmers via a mobile app, enabling them to monitor and make decisions in real time.

### Project Motivation

**What are the reasons behind choosing this project?**

The **Smart Agri-Guard** project was developed to address key challenges in greenhouse farming, such as inconsistent monitoring, inefficient resource usage, and difficulties in maintaining plant health. Many farmers rely on manual observation, which can result in unnecessary water and fertilizer consumption. This project aims to automate greenhouse management and provide an efficient, data-driven approach to optimizing agricultural practices.

**Why is this project important?**

This project is essential as it introduces technological advancement in modern agriculture, enabling farmers to make more informed and precise decisions. By automating greenhouse monitoring, it reduces manual labor, increases accuracy, and enhances sustainability. Additionally, adopting smart farming techniques is crucial for meeting the growing food demand while conserving natural resources.

**What is the new idea proposed by this project?**

The innovative aspect of **Smart Agri-Guard** lies in its ability to automate greenhouse monitoring with AI-driven insights. Unlike traditional farming, which depends on manual observation and estimation, this system enables precision farming based on real-time data. The key features introduced include:

* AI-powered alerts for optimizing plant health.
* Proactive decision-making based on real-time sensor data.
* A user-friendly mobile app that provides easy access to greenhouse conditions.

### Problem Statement

Traditional greenhouse management faces multiple challenges that affect plant health, resource efficiency, and overall productivity. The key issues addressed by this project include:

* Inconsistent plant growth is due to variations in soil moisture, temperature, humidity, and nutrient levels.
* Inefficient use of water and fertilizers, leading to waste, increased costs, and environmental damage.
* Delayed detection of plant diseases and unfavorable conditions, resulting in crop losses and financial setbacks.

### Project Aim and Obj.ectives

**Q1: What is the goal of this project?**

The goal of **Smart Agri-Guard** is to develop an intelligent agricultural monitoring system that enhances greenhouse farming by ensuring optimal plant health and maximizing productivity. The system leverages IoT sensors and AI-driven analytics to provide farmers with real-time insights into soil and environmental conditions, enabling efficient resource management and early issue detection.

**Q2: How can this project achieve its goal?**

To achieve this goal, **Smart Agri-Guard** focuses on the following key objectives:

* Real-time monitoring: The system continuously tracks soil moisture, air temperature, humidity, soil acidity, nutrient levels through IoT sensors.
* AI-based analysis: The collected data is processed using machine learning algorithms to predict plant growth patterns and identify potential issues.
* Efficient resource utilization: The system optimizes water and fertilizer usage through data-driven irrigation schedules.
* User-friendly platform: The mobile app provides actionable insights, alerts, and recommendations, enabling farmers to make informed decisions easily.

### 1.5 Related Existing Systems

This project is inspired by several existing smart agriculture systems designed to monitor plant and environmental conditions. Two well-established systems relevant to greenhouse and nursery applications are **Arable** and **Phytech**. Below is a summary of their key features:

1.**Arable**

Arable is comprehensive crop intelligence device that provides multi-modal environmental and plant-level data. It combines weather monitoring, plant stress measurement, and predictive analytics into a single solar-powered sensor.

**Key Features:**

* Measures rainfall, solar radiation, temperature, humidity, wind, and leaf wetness.
* Includes a 5MP camera and 22-band spectrometer for canopy analysis.
* Provides crop models, yield forecasts, and disease risk assessments.
* Integrated with a dashboard and mobile tools for data visualization.

**Used in:** Commercial agriculture, research institutions, and smart greenhouses.

**2.Phytech**

Phytech is a plant-focused digital platform that monitors plant stress and irrigation effectiveness using real-time data collected from physical plant sensors.

**Key Features:**

* Uses dendrometers to track stem growth and detect water stress in plants.
* Monitors irrigation components such as pressure, flow, and tank levels.
* Sends real-time alerts for anomalies in irrigation performance.
* Delivers agronomic recommendations based on plant feedback and environmental factors.
* Supports integration with irrigation automation systems.

**Used in:** Orchards, vineyards, and controlled greenhouse environments.

**Project Differentiation:**

While Arable and Phytech are powerful and well-established agricultural monitoring systems, this project is uniquely tailored for nursery environments with specific functional and design goals. The key differentiators include:

1. **Target Environment**

* Unlike Arable and Phytech, which are optimized for large-scale farms, orchards, or vineyards, this project is designed specifically for agricultural nurseries and greenhouse-based plant production.
* It focuses on individual plant monitoring, making it ideal for nursery conditions where different plant types are grown in controlled sections.

1. **Role-Based Access and Plant Assignment**

* This system features a role-based mobile application where farmers only access and monitor plants assigned to them. Managers oversee the entire greenhouse and all associated plants. Administrators manage users, greenhouses and plant data.
* This level of user-specific interaction is not a core feature of Arable or Phytech.

1. **Integration of Specific Sensors**

* The system uses a combination of temperature & humidity (DHT22), soil moisture, pH, NPK, and light sensors — all of which are selected to provide essential insight for nursery-level decision-making.
* It supports flexible sensor integration with **ESP32**, allowing for **scalability** and **cost-effectiveness**.

1. **AI-Driven Predictions and Recommendations**

* The project includes a machine learning model trained to predict plant health status, recommend actions like irrigation or fertilization based on historical and real-time data.

1. **Expandable Design**

* It is structured to **expand over time**, allowing for future features like automation, drone integration, or centralized dashboards.

**CHAPTER 2: Planning phase**

**2.1 Scope of the project**

****

**Context Diagram**

**Overview:**

This project aims to develop a smart agriculture system for monitoring plant health in nurseries, using a mobile application integrated with a backend server. The system collects real-time environmental and soil data from sensors connected to ESP32 modules and uses AI algorithms to analyze this data. The AI provides insights such as plant condition, irrigation needs, and fertilization recommendations. The mobile app allows farmers to monitor onlytheirassigned plants, while nursery managers can access all plants within their greenhouse.

**Project Boundaries:**

* Development of a Flutter mobile application for farmers, managers, and admins.
* A backend server using ASP.NET Core with RESTful APIs for communication.
* Real-time sensor integration via ESP32 (temperature, humidity, pH, NPK, soil moisture).
* AI**-**basedpredictions of plant health and needs (irrigation, fertilization).
* Role-based access and plant assignments for farmers.

**Project Features:**

1. **Mobile App (Flutter):**

* User Authentication.
* Real-time monitoring of assigned plants (farmer) or full greenhouse (Manager).
* Sensor data visualization (charts, graphs).
* AI-driven insights and notifications.
* User-friendly interface for managing plants.
* Arabic user-friendly interface.

1. **Backend (ASP.NET Core):**

* RESTful API for mobile-app communication.
* Secure data handling and storage (SQL Server).
* AI module integration.
* User, plant and greenhouse management.
* Role-based access control.

1. **Sensor Network (ESP32):**

* Multiple IoT sensors (temperature, humidity, etc.)
* Data transmission via Wi-Fi to the backend server

1. **AI Model:**

* Predict plant health status (healthy, stressed, needs water/fertilizer)
* Generate tailored recommendations based on sensor thresholds and historical data.

**External Entities:**

* **Farmer:** Views and monitors only the plants assigned to them. Receives alerts.
* **Manager:** Manages farmers and monitors all plants in the greenhouse.
* **Admin:** Manages the system (users, plants, greenhouses, etc.).
* **ESP32 with sensors:** Collects environmental and soil data, sends it to backend via Wi-Fi.
* **Mobile App:** Used by all users to interact with the system and view data.

**Responsibilities:**

1. **Team Member Responsibilities:**

* **Mobile App:** Batool Saber, Mutasem Mustafa, Naser Hani
* **IOT/Hardware:** Osama Rad
* **AI Machine Learning:** Danial Zeyad, Osama Rad
* **Backend and Database:** Osama Rad.

1. **Verification and Approval Procedures:**

* **Feature Completion**: Each feature will be tested individually and integrated into the system.
* **System Testing:** Functional and non-functional testing including performance, security, and usability.
* **Approval:** Final testing results will be reviewed by the project advisor/stakeholders for approval.

### Project risks and Product risks

**Project Risks**

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk Category** | **Risk Description** | **Potential Impact** | **Mitigation Strategies** |
| **Schedule Risk** | Delays in hardware delivery or software development | Project delivery may be delayed | Use a flexible timeline with milestones to track progress |
| **Resource Risk** | Shortage of components (sensors, controllers, etc.) | Temporary halt in development or reduced prototype quality | Pre-order extra components and plan resource requirements carefully |
| **Technical Risk** | Difficulty integrating AI technologies with real-time sensor data | Reduced accuracy in decision-making | Gradual testing of models and using modular, adaptable software solutions |
| **Budget Risk** | Exceeding the allocated budget due to unexpected expenses | Inability to complete all features or purchase necessary components | Define a clear budget plan with a contingency buffer |
| **Team Risk** | Lack of expertise in AI, IoT | Slower progress or errors in implementation | Assign roles based on strengths, and consider external consultation or training |
| **Communication Risk** | Miscommunication among team members or with stakeholders | Misaligned expectations or rework needed | Schedule regular meetings and use collaboration tools like GitHub |

**Product Risks**

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk Category** | **Risk Description** | **Potential Impact** | **Mitigation Strategies** |
| **Hardware Risk** | Sensors might provide inaccurate or faulty readings | Poor decision-making and crop damage | Use calibration routines and test sensors thoroughly before deployment |
| **Software Risk** | Bugs in the mobile app or AI model misclassifications | Incorrect alerts or failure to notify about critical conditions | Conduct extensive testing and implement error-handling routines |
| **Data Risk** | Loss of data or connectivity interruptions | Missing critical information, poor insights | Use local backup systems and ensure stable data transmission protocols |
| **Usability Risk** | Mobile app may not be intuitive for farmers with limited tech experience | Low adoption or improper system use | Perform user testing and design a simple, localized user interface |
| **AI Model Risk** | AI predictions may be biased or trained on insufficient data | Inaccurate insights affecting plant health | Continuously train models with real, diverse datasets and monitor model performance |
| **Security Risk** | Unauthorized access to the system or data | Data theft or manipulation | Implement authentication, data encryption, and secure network practices |

### 2.3 Feasibility study

**Technical Feasibility**

The Smart Agri-Guard project is technically feasible due to the following factors:

|  |  |
| --- | --- |
| **Proven Technologies** | The system uses well-established hardware components (ESP32, DHT22, soil moisture, pH, NPK) that are widely available and tested in IoT applications.  The software stack (Flutter for mobile, ASP.NET Core for backend, Python for AI) consists of mature and well-documented frameworks. |
| **AI & IoT Integration** | Machine learning models for agricultural monitoring (e.g., predicting plant health based on sensor data) have been successfully implemented in similar projects.  ESP32’s Wi-Fi capabilities ensure reliable data transmission to the backend. |
| **Team Expertise** | The team has members with skills in mobile development (Flutter), backend (ASP.NET), Python and ready to learn machine learning and IOT to ensure all technical aspects are covered. |
| **Scalability & Modularity** | The system is designed in modular components (mobile apps, backend, AI, sensors), allowing incremental development and testing. |

**Schedule Feasibility**

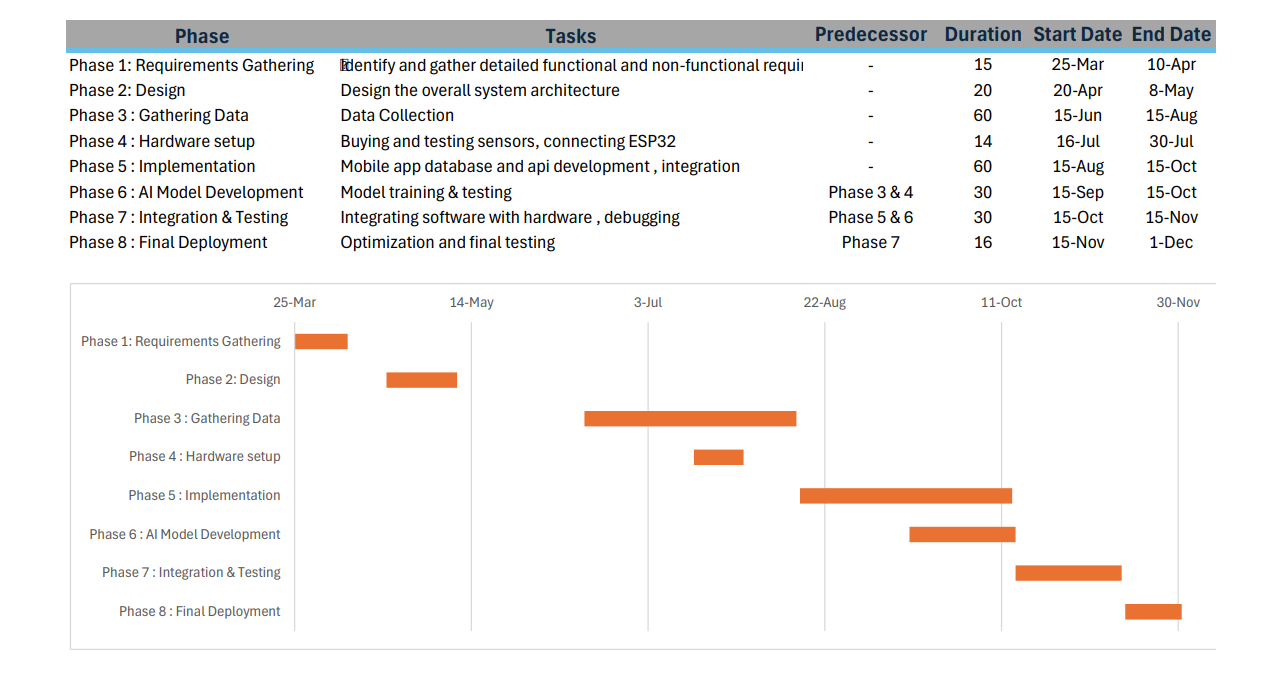
The project timeline is realistic and structured into clear phases:

|  |  |
| --- | --- |
| **Phase 1: Requirements Gathering (15 days)** | Identify and gather detailed functional and non-functional requirements from stakeholders using interviews, observation, and brainstorming techniques. |
| **Phase 2: Design**  **(20 days)** | Design the overall system architecture, create use case diagrams, class diagrams, ERD, and sequence diagrams based on the collected requirements. |
| **Phase** 3  Data Gathering (60 days) | Defining data sources, collection, validation, and storage ensures a solid foundation for AI training. |
| **Phase 4**  Hardware Setup (14 days) | Sensors and ESP32 integration are short but critical phase. |
| **Phase 5**  Implementation (60 days) | Mobile app (Flutter), backend (ASP.NET Core), and database setup are achievable within this period. |
| **Phase 6**  AI Model Development (30 days) | Data preprocessing, model selection, and training align with typical machine learning project timelines. |
| **Phase 7**  Integration & Testing (30 days) | System integration and bug fixing ensure a stable final product. |
| **Phase 8**  Final Deployment (16 days) | Perform final testing, validate system functionality, and prepare the system for presentation and evaluation. |

**Conclusion:**

The Smart Agri-Guard project is feasible in terms of technology, schedule, and budget. The team has the necessary skills, the timeline is well-structured, and the required components are cost-effective and reliable. Any risks (e.g., hardware delays, AI model accuracy) can be mitigated through iterative testing and alternative solutions.

**2.4 Project Schedule**

****

**2.5 Project Software and Hardware Requirements**

**Hardware Requirements:**

* **ESP32 Microcontroller**: Collects sensor data and sends it via Wi-Fi.
* **Capacitive Soil Moisture Sensor**: To measure the moisture level of the soil.
* **DHT22 / DHT11 Sensor**: Measures temperature and humidity.
* **PH Sensor**: Detects soil pH level.
* **NPK Soil Sensor**: Measures nitrogen, phosphorus, and potassium levels.
* **Battery / USB Power Supply**: Powers the system in the field.
* **Jumper wires, Breadboard, Resistors:** Required for hardware connections.
* **Mobile Phone (Android/iOS):** To run and test the Flutter mobile app.
* **Wi-Fi Router / 4G Module:** To provide internet access in the greenhouse area.

**Software Requirements:**

* **Arduino IDE**: To program the ESP32 microcontroller and sensors.
* **NET Core SDK:** Framework for building the server-side RESTful API.
* **Visual Studio 2022:** To develop the backend using ASP.NET Core Web API.
* **Postman:** To test and debug the RESTful APIs.
* **Firebase Cloud Messaging (FCM):** Notification service used to send real-time alerts to user's mobile apps.
* **SQL Server**: To store user data, plant data, sensor readings.
* **SQL Server Management Studio (SSMS)**: To create, manage, and interact with the SQL Server database.
* **Python:** To develop and train the AI model for plant health prediction.
* **TensorFlow / Keras:** For building and training machine learning models.
* **Jupiter Notebook**: Used for writing and running Python code for machine learning.
* **Flutter:** To build the cross-platform mobile application (Android/iOS).
* **Visual Studio Code:** IDE for mobile application.
* **Security Tools:** SSL/TLS encryption and authentication (token-based) to protect data in transit and rest.
* **GitHub**: Co-collaborate and managing the project’s source code.

## CHAPTER 3: Requirement Engineering and Analysis

# 3.1 Used Techniques for Requirements Collection

# Interviews

We conducted interviews with a diverse group of stakeholders, including farmers, farm owners, and agricultural managers. These interviews aimed to gain deeper insights into their specific needs, challenges, and expectations regarding the smart farming system. A wide range of open-ended questions was used to encourage participants to share detailed information based on their daily experiences.  
  
For example, we asked the farm manager questions such as:  
- "What kind of data do you need to make informed decisions about your farm operations?"

The answer was:

- "I find it difficult to collect data accurately because I have to measure moisture and temperature manually in different parts of the land. This takes a lot of time, and it's easy to make mistakes or forget to record important details. Tracking changes throughout the day is also challenging."  
  
- "How do you currently track environmental conditions, and what limitations do you face?"

The answer was:

- "Currently, we use manual measuring tools to monitor environmental conditions, but this does not provide continuous or precise data. Manual measurements take a lot of time and can delay decision-making. Also, storing the data manually becomes very complicated when managing multiple agricultural areas"

Meanwhile, the farmers were asked questions like:  
- "What challenges do you face when trying to monitor soil or weather conditions manually?"

The answer was:

*"We mainly need accurate and real-time data on soil moisture, general temperature, and pH levels. These parameters are crucial for managing irrigation schedules effectively and ensuring that crops are growing in suitable conditions. Without consistent monitoring, it's difficult to detect imbalances, which can lead to reduced crop quality and productivity."*

- "What kind of alerts or notifications would be most helpful to you during your work?"

The answer was:

"The most useful alerts would be those notifying me when there are significant changes in soil moisture or temperature, or when additional irrigation is needed. Real-time notifications would help me make quicker and more accurate decisions to avoid crop damage."

This interview process was crucial in helping us understand the different perspectives and operational needs across various roles, enabling us to define accurate system requirements that cater to all stakeholders.

# Direct Observation

As a second step, we conducted on-site observations to monitor farmers as they performed their routine agricultural tasks. This allowed us to witness firsthand how they interact with their current tools and workflows. We carefully recorded their actions, noting both efficient practices and areas of difficulty.  
  
For example, we observed that many farmers were manually recording environmental measurements such as soil moisture and temperature using notebooks or paper sheets. This method not only increased the risk of errors but also made it difficult to track historical data and trends. Additionally, we noticed that some tasks, such as checking irrigation needs or monitoring pest activity, required physical presence and repetitive manual inspection.  
  
Through this observation, we gained a better understanding of the challenges they face and identified opportunities where automation and real-time data access could significantly improve productivity, reduce effort, and minimize errors. These insights were essential in designing a system that directly addresses the pain points in current agricultural practices.

# Brainstorming

Another technique we used during the requirements collection process was brainstorming. Our development team held multiple sessions where we discussed and analyzed each part of the proposed system separately. We explored various real-world and hypothetical scenarios to better understand possible user interactions and system behaviors.  
  
One key outcome from these sessions was the idea that:  
"Each farmer should not be able to view all plant-related data in the system, but only the data that is relevant to their own crops, as managed and assigned by the farm manager."  
  
This concept helped shape the system’s role-based access control, ensuring that data visibility is aligned with user responsibilities. Brainstorming allowed us to creatively explore user needs from different angles and make better-informed design decisions.

**3.2 Functional Requirement & Modelling**

**3.2.1 List of System Functions (Features)**

**Sensor System:**

* Collect Sensor Data (e.g., temperature, humidity, NPK, etc.)
* Send Data to Server (via ESP32 using Wi-Fi)

**AI System:**

* Analyze Sensor Data
* Predict Plant Health Status
* Predict Plant Needs (e.g., water, nutrients)
* Generate Smart Recommendations

**Farmer:**

* Login to the System
* Monitor plant health
* Receiving Alerts & Notifications
* Log Out

**Manager:**

* Login to the System
* Monitor All Farmers’ Plants
* Generate Graphs and Reports
* Manage Farmer Accounts (Add/Remove/Update/assign for specific plants)
* View Plant Health History
* Log Out

**Admin:**

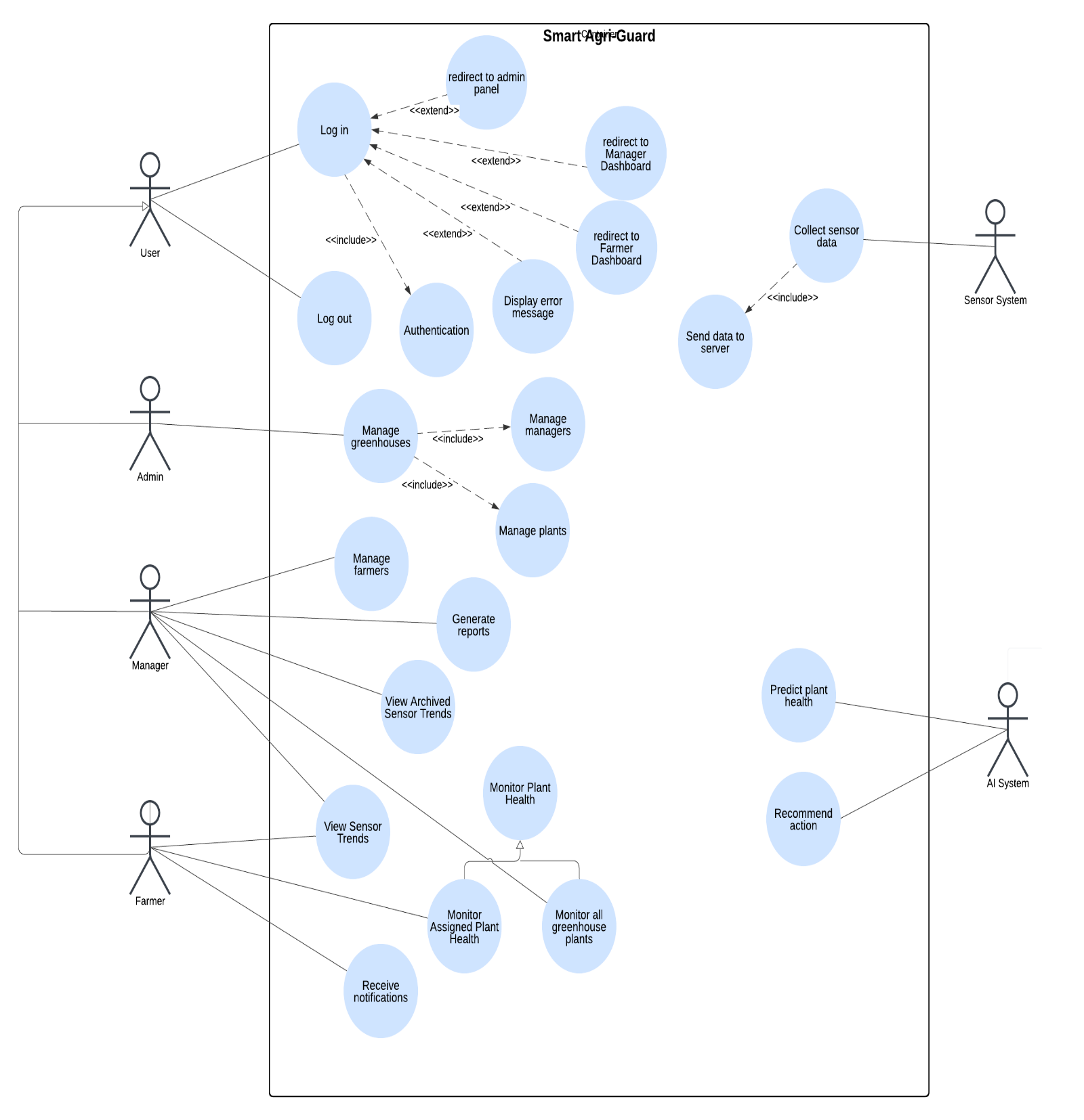
* Login to the System
* Manage Greenhouses
* Manage System Updates
* Manage Plant Types and Data
* Manage Manager Accounts (create, assign for greenhouse, remove)
* Log Out

**Common/System Features:**

* Authenticate Users (Login Validation)
* Encrypt Data in Transit and at Rest
* Display Error Messages
* Role-based Access Control (Redirect to correct dashboard)

**3.2.2 Use Case Diagram:**

**3.2.2 Use Case Diagram**

****

**3.2.3 Use Cases: Description & Details:**

**UC1: Log In**

**Actors:** Farmer, Manager, Admin

**Description:** Allows users to log into the system using their credentials.

**Main Flow:**

1. User opens the app.
2. User enters username and password.
3. The system verifies the credentials.
4. If valid, the system redirects users to their specific dashboard.
5. Session starts.

**Alternative Flows:**

* 3a. If credentials are invalid:
* System displays an error message: “Invalid username or password.”

**UC2: Collect Sensor Data.**

**Actor:** Sensor System

**Description:** Automatically collects data from sensors (e.g., temperature, humidity, etc.).

**Main Flow:**

1. Sensors read the environment periodically.
2. Sensor system gathers data.
3. Sensor system formats data for transmission.

**Alternative Flows:**

* a. Sensor is Disconnected or Malfunctioning:
* System detects that a sensor is not responding.
* Logs an error.
* Skips that sensor and proceeds with available data.
* Send a notification to the manager or admin about hardware failure.

**UC3: Send Data to Server**

**Actor:** Sensor System  
**Description:** Sends collected data to the backend server using HTTPS.  
**Main Flow:**

1. Sensor system establishes secure connection.
2. Sends encrypted JSON payload to backend API.
3. Server receives and stores the data.

**Alternative Flows:**

* 1a. If server is unreachable:
* Data is cached locally.
* Retry sending after delay.

**UC4:** Predict Plant Health

**Actor:** AI System

**Description:** The ai model predicts a plant’s health based on the latest sensor readings.

**Main Flow:**

1. The system retrieves the latest sensor readings for a specific plant.
2. The system preprocesses the data internally (e.g., cleaning, filtering, etc.)
3. The system runs the data through the trained machine learning model.
4. The system generates a prediction for this plant (e.g., healthy, Moderate stress, High stress).
5. The system stores the prediction results in the database.
6. The system makes the predictions available to the manager and/or assigned farmers.

**Alternative Flows:**

* A1: If sensor data is missing or incomplete, the system logs an error and skips prediction for that plant.
* A2: If model inference fails, a fallback mechanism is triggered (e.g., notify admin or retry).

**UC5: Recommend Actions**

**Actor:** AI System

**Description:** The ai model suggests corrective actions based on the latest sensor readings for specific plant.

**Main Flow:**

1. The system retrieves the latest sensor readings for a specific plant.
2. The system preprocesses the data internally (e.g., cleaning, filtering, etc.)
3. The system runs the data through the trained machine learning model.
4. The system generates a recommendation for this plant (e.g., it needs watering, it needs fertilizing).
5. The system stores the recommendation results in the database.
6. The system makes the recommendation available to the manager and/or assigned farmers.

**Alternative Flows:**

* A1: If sensor data is missing or incomplete, the system logs an error and skips prediction for that plant.
* A2: If the recommendation mapping fails or is unavailable, a generic message is shown (e.g., “Please consult with a specialist.”).

**UC6: Monitor Plant Health**

**Actors:** None directly (this is a generalized base use case)

**Description:** Shared logic to retrieve, process, and display plant health data.

**Used by:**

* Monitor Assigned Plant Health
* Monitor All Greenhouse Plants

**UC7: Monitor Assigned Plant Health**

**Actor:** Farmer

**Description:** The farmer views the health status of plants they are assigned to.

**Main Flow:**

1. The farmer logs into the system.
2. The system authenticates the user.
3. The system retrieves the list of assigned plants for this farmer.
4. The system fetches plant health status and displays the relevant metrics and insights.

**Alternative Flow**:

* If the farmer has no assigned plants, the system notifies: *"No plants assigned at the moment."*

**UC8: Monitor All Greenhouse Plants**

**Actor:** Manager

**Main Flow:**

1. The manager logs into the system.
2. The system authenticates the user.
3. The manager selects the greenhouse
4. The system retrieves all plants in the greenhouse.
5. The system displays each plant’s health status using sensor data and AI predictions.

**Alternative Flow**:

* If there are no plants registered in the system, display: *"No plant data available."*

**UC9: Receive Notifications & Alerts**

**Actors:** Farmer

**Description:** Get alerted about abnormal conditions or required actions.

**Main Flow:**

1. The system detects a critical issue (plant needs watering).
2. Send notification to a user device.
3. User receives and opens notifications to view details.

**UC10: Manage Farmers**

**Actors:** Manager

**Description:** Manage farmer accounts by performing actions such as adding, assigning farmers for specific plants, removing and deleting farmer from the greenhouse.

**Main Flow:**

1. Manager logs into the system.
2. Manager selects a particular greenhouse.
3. Manager navigates to the "Manage Farmers" section.
4. The system displays a list of existing farmers under the manager’s greenhouse.
5. The manager selects one of the following actions:

* Add a new farmer
* Edit a farmer’s information
* Delete a farmer

1. The system validates the action and processes the request.
2. System updates the database.
3. The system confirms the successful update to the manager.

**Alternative Flows:**

* 4c: Delete Farmer – Farmer Assigned to Plants:
* The system blocks the deletion and notifies the manager to reassign the plants first.

**UC11: View Sensor Trends**

**Actors:** Manager/Farmer

**Description:** Displays real-time and recent historical sensor data (e.g., temperature, moisture, pH) in graphical or tabular form.

**Main Flow:**

1. User selects a specific plant’s sensor trends.
2. User selects a date range (up to 2 months old) and sensors type.
3. The system displays sensor trends visually (charts, graphs).

**Alternative Flow:**

* 5a: No data found
* The system displays: "No sensor data available at the moment."

**UC12: View Archived Sensor Trends**

**Actors:** Manager

**Description:** Allows managers to access historical sensor data from archived storage for deeper analysis or seasonal trends.

**Main Flow:**

1. Manager logs in.
2. Manager selects a particular greenhouse.
3. Manager navigates to “Archived Sensor Trends”.
4. Manager selects a particular plant, date range (from 2 months up to 1 year) and sensor types.
5. The system displays sensor trends visually (charts, graphs).

**Alternative Flow:**

* A1: if the date range is less than 2 months old or more than 1 year old
* The system displays “too recent or older than 1 year”.
* A2: No data found
* The system displays “No data available for selected filters”.

**UC13: Generate Reports**

**Actor:** Manager

**Description:** The system compiles data and analytics into formatted reports for plant performance, health and predictions.

**Main Flow:**

1. Manager logs in.
2. Manager selects a particular greenhouse.
3. Manager navigates to “Generate Report”.
4. Manager selects specific plants, date range, sensor types and file type(PDF or CSV).
5. System retrieves relevant data from sensor readings from sensor data table or archived data table or both.
6. The System compiles and formats the report.
7. The system shows a downloadable link.

**Alternative Flow:**

* A1: date range not valid
* The system displays “The date range is not valid”
* A2: No data found
* The system displays “No data available for selected filters”.

**UC14: Manage Greenhouses**

**Actor:** Admin

**Description:** The admin can create, update, and delete greenhouses in the system.

**Main Flow:**

1. Admin logs in.
2. Admin navigates to the "Manage Greenhouses" section.
3. Admin selects an action: Add, Edit, or Delete.
4. The system performs the action and updates the database.

**UC15: Manage Managers**

**Actor**: Admin

**Description**: The admin can assign managers to greenhouses, and create, update, or remove manager accounts.

**Main Flow:**

1. Admin logs in.
2. Admin accesses the "Manage Managers" section.
3. Admin selects an action: Add, Update, or Remove manager.
4. If adding/updating, Admin fills in manager details and assigns a greenhouse
5. The system performs the action and updates the manager’s list.
6. System shows success message.

**UC16: Manage Plants**

**Actor:** Admin

**Description: a**dmin adds, updates, or removes plant types and their attributes (e.g., name, type)

**Main Flow:**

1. Admin logs in.
2. Admin navigates to **"**Manage Plants" section.
3. The system displays a list of greenhouses.
4. Admin selects a specific greenhouse.
5. The system displays a list of plants in the selected greenhouse.
6. Admin chooses to Add, Update, or Remove a plant.
7. Admin provides or updates plant data.
8. The system confirms the success and updates the greenhouse's plant list.

**UC17: Log Out**

**Actor:** Manager/Farmer/Admin

**Description:** The user clicks on log out to securely exit the system

**Main Flow:**

1. User selects log out
2. The system confirms the log out request
3. The system redirects the user to the log in screen
   1. **Nonfunctional Requirements: Quality & Constraints**

|  |  |  |  |
| --- | --- | --- | --- |
| **NFR Category** | **Requirement Description** | **Priority** | **Justification** |
| **Performance** | The system must process sensor data and provide health insights within 5 seconds. | High | Ensures timely feedback for farmers to take action quickly. |
| **Scalability** | The system must support up to 500 concurrent users and 1000 sensor nodes. | High | Supports future expansion of nursery operations and users. |
| **Usability** | Simple, intuitive UI for all user roles. | Medium | improves user experience and adoption among farmers. |
| **Maintainability** | Codebase must be modular to allow updates without affecting other parts of the system. | Medium | Facilitates long-term support and iterative development. |
| **Security** | User authentication, data transmission, and data storage must use secure protocols and data encryption (e.g., HTTPS, JWT) | High | Protects user data and prevents unauthorized access and ensures confidentiality and prevents data interception or tampering. |
| **Availability** | The system should be available 24/7 with minimal downtime to ensure constant monitoring and access to alerts and data. | High | Farmers and managers rely on real-time data for plant health monitoring and decision-making. Downtime could lead to missed issues. |
| **Reliability** | Ensure consistent reception, storage, and processing of sensor data. | High | Data loss or inconsistency can affect AI predictions and decision-making. |
| **Interoperability** | The backend must support RESTful APIs for integration with third-party tools. | Medium | Enables potential expansion with other agricultural platforms. |
| |  | | --- | | **Hardware Constraint** |  |  | | --- | |  | | |  | | --- | |  |  |  | | --- | | Limited memory and storage on ESP32 and sensors. | | High | It affects the complexity and size of data that can be handled locally. |
| **Connectivity** | Requires stable internet connection to sync data with the server. | High | Unstable connections may delay monitoring or AI-driven decisions. |
| **Access Control** | Role-based access for Admin, Manager, and Farmer. | Medium | |  | | --- | |  |  |  | | --- | | Prevents unauthorized access and ensures proper system usage. | |
| |  | | --- | | **Data Storage** |  |  | | --- | |  | | |  | | --- | |  |  |  | | --- | | The database must handle large historical datasets from multiple sensors. | | High | Historical data is needed for AI analysis, reports, and plant health tracking. |

**CHAPTER 4: Architecture & Design**

**4.1** **Software Architecture**

**A diagram of a cloud server

AI-generated content may be incorrect.**

This system architecture outlines a smart IoT-based platform designed to enable real-time monitoring and communication between hardware devices, a backend server, and a mobile application. At the center of this system is the ESP32 device, a compact microcontroller equipped with Wi-Fi capability. It serves as the physical interface with the environment, connected to various sensors that collect data such as temperature, humidity, and soil moisture. ESP32 communicates with the Backend Server through the Https protocol, sending real-time data.

The Backend Server acts as the brain of the operation. It serves as the bridge between the ESP32 hardware, the SQL Server Database, and the Mobile App. Whenever the ESP32 sends sensor readings, the server processes these requests and stores the information in the database. This server also takes on the responsibility of evaluating via integrated AI modules—and, when necessary, it triggers alerts or suggestions.

The Mobile App, which represents the user interface of the system, plays a critical role in user interaction. It enables users to receive alerts from FCM, monitor plants, and interact with the backend—whether to query past readings or manage their greenhouses.

To deliver real-time notifications to end users, the system leverages Firebase Cloud Messaging (FCM), that enables instant push messaging to the mobile devices. For instance, if the backend detects an abnormal sensor reading—such as high soil dryness or critical temperature—it can send a notification via FCM to the user’s mobile application.

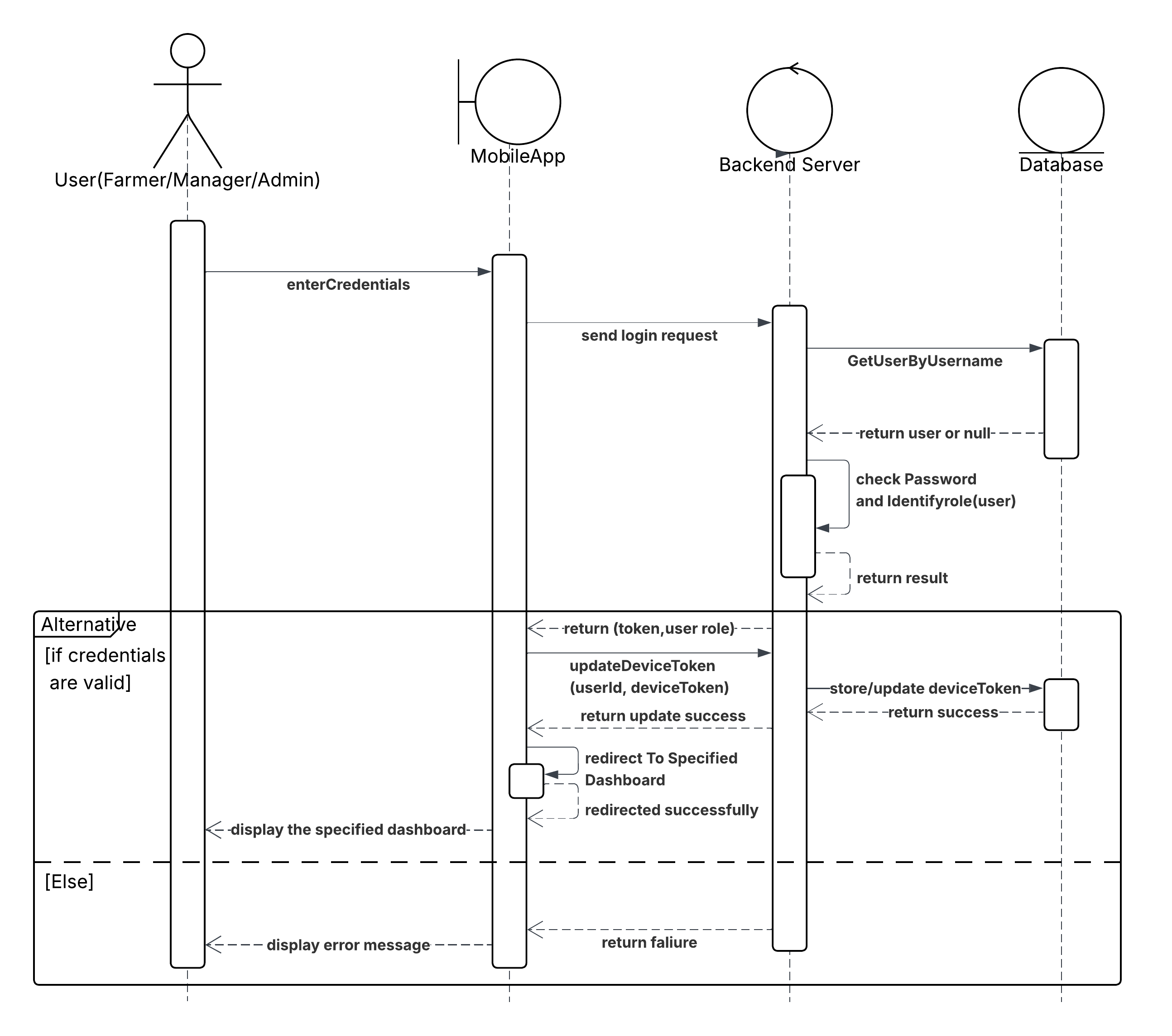
All structured data ranging from user accounts to historical sensor log is persistently stored in the MySQL Server. This relational database ensures that all records are well-organized and can be easily retrieved or updated as needed. It supports all major system functions including user authentication, greenhouse assignments, plant data tracking, and AI-based predictions.

In terms of data flow, the process begins with the ESP32 collecting environmental data and transmitting it via HTTP to the backend. The backend stores this data in the SQL Server database and evaluates whether any alerts or actions are needed. If so, it uses FCM to send real-time notifications to the mobile app. Users can also initiate interactions from the mobile app, which communicates with the backend to retrieve sensor readings or manage settings.

**4.2 Software Detailed Design**

#### 4.2.1 Use Cases Internal Interactions as Sequence Diagrams

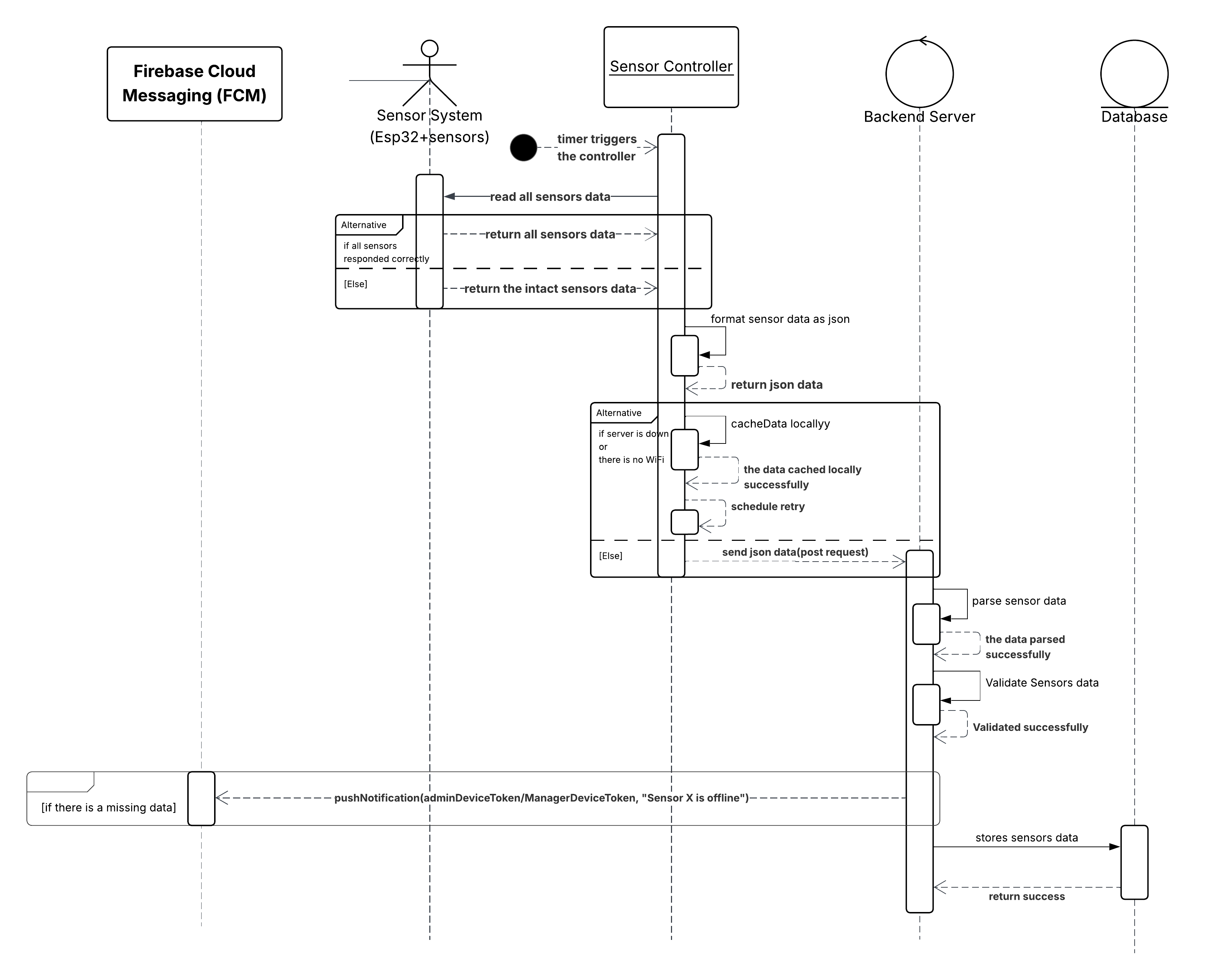
1. Log in:



This sequence diagram illustrates the login process for a user (who could be a Farmer, Manager, or Admin) accessing the system through a mobile application. The process begins when the user enters their credentials into the mobile app, which then sends a login request to the backend server. Upon receiving the request, the backend server queries the database to retrieve the user data using the provided username. The database returns either the user object or null if no such user exists. The backend server then verifies the password and identifies the user’s role (e.g., Farmer, Manager, or Admin).

Depending on the outcome, two scenarios are possible: if the credentials are valid, the backend server returns an authentication token and the user's role to the mobile app. The mobile app proceeds to update the device token by sending it along with the user ID to the backend server, which stores or updates this token in the database. Once the update is confirmed, the mobile app redirects the user to the appropriate dashboard based on their role and displays it accordingly. In the alternative case where the credentials are invalid, the backend returns a failure response, prompting the mobile app to display an error message to the user. The diagram effectively captures the interaction between the user, mobile app, backend server, and database during the login process, emphasizing both successful and failed authentication paths.

2. Collect sensor data and send them to the server:



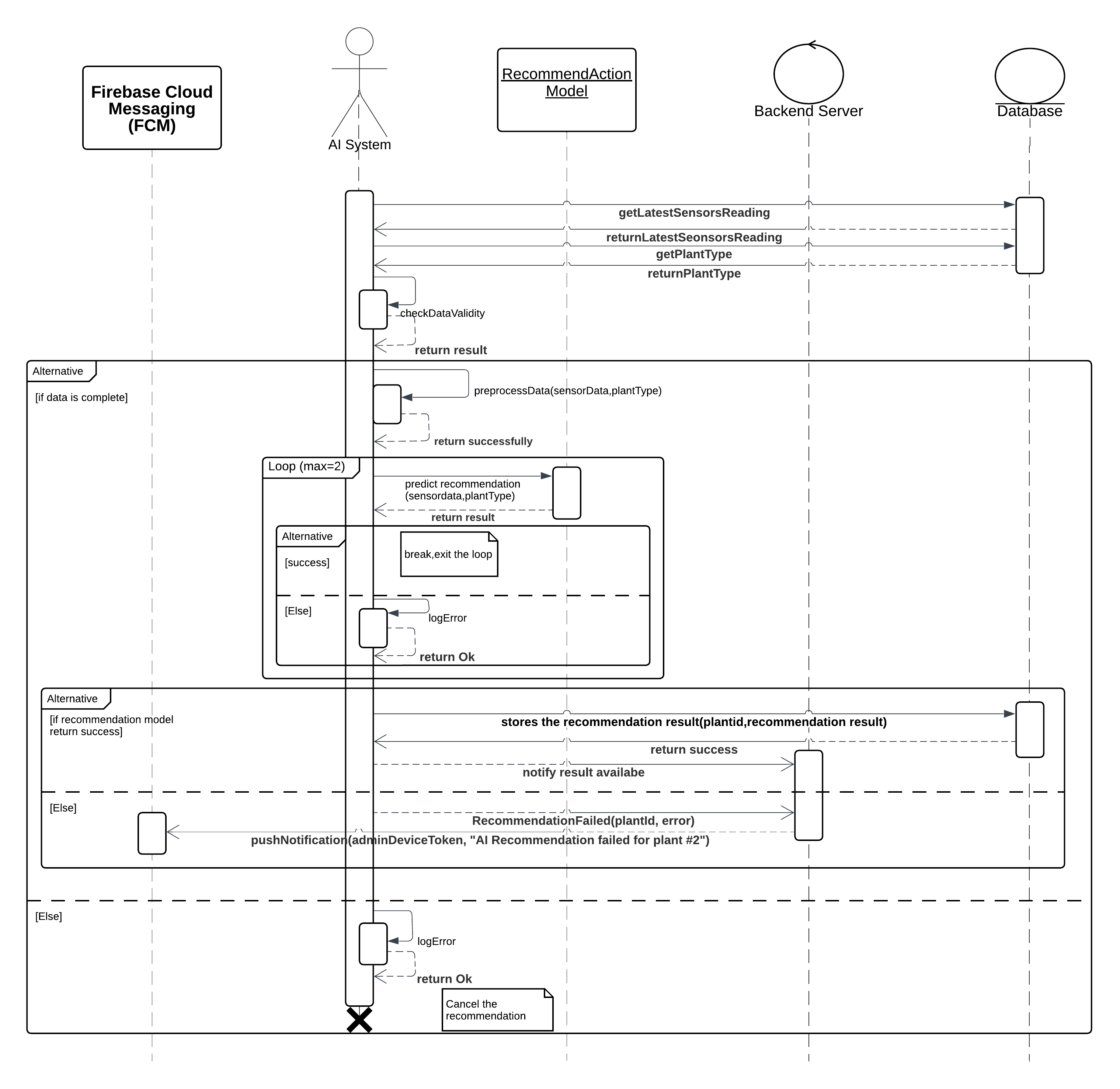
This sequence diagram illustrates the process of collecting sensor data from the ESP32-based sensor system and transmitting it to the backend server for storage in the database. The process is initiated by a timer that triggers the Sensor Controller to read data from all connected sensors. If all sensors respond correctly, the full dataset is returned; otherwise, only data from functional sensors is collected. The data is then formatted into JSON. If there is no internet connection or the server is unreachable, the data is cached locally, and a retry is scheduled. Otherwise, the data is sent to the backend server via a POST request. Upon receiving the data, the backend parses and validates it before storing it in the database. In case any sensor fails to provide data, a push notification is sent via Firebase Cloud Messaging (FCM) to inform the administrator or manager that a sensor is offline.

3. Predict plant health:



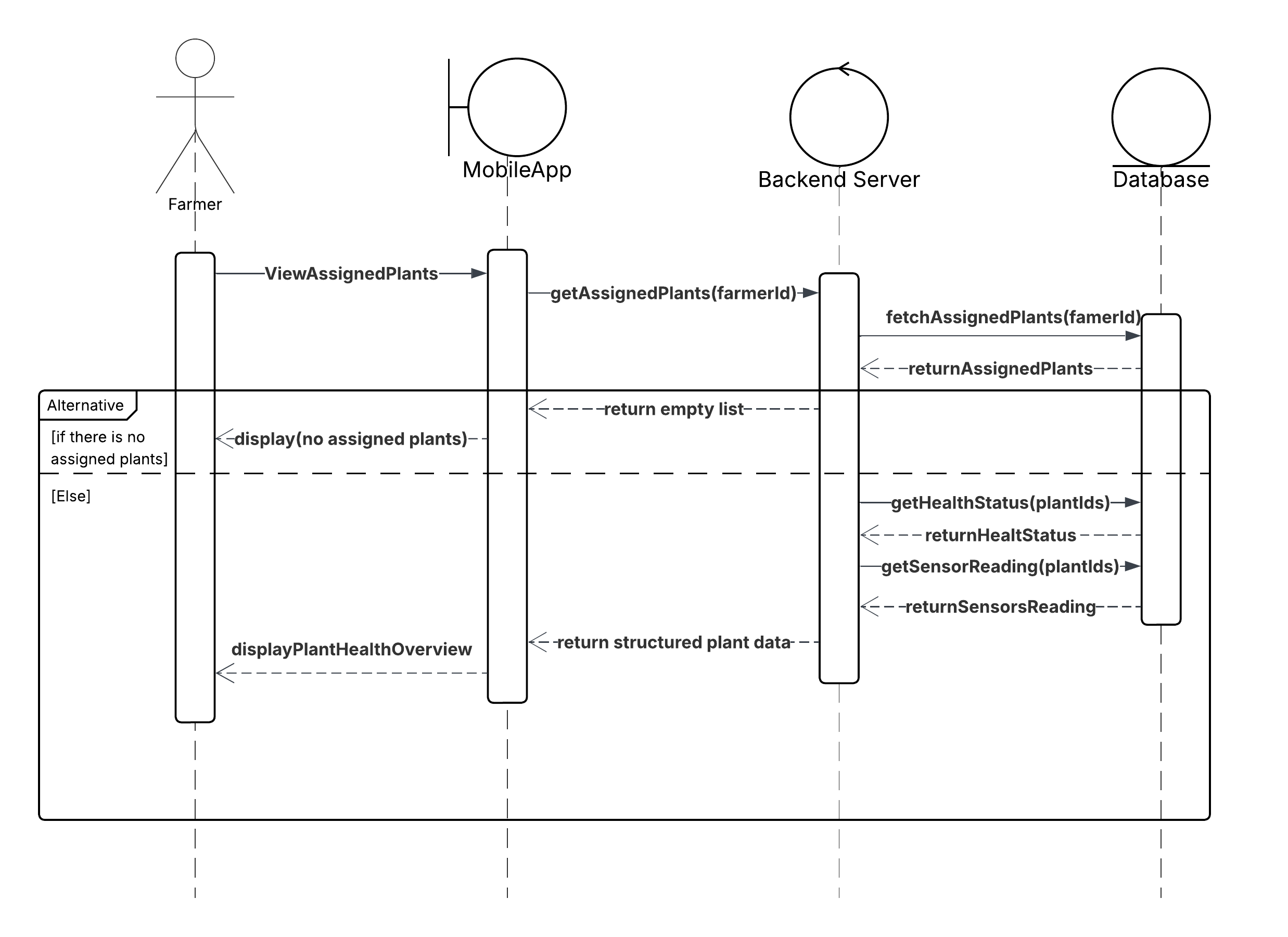
This diagram illustrates the AI-powered plant health prediction workflow, which operates autonomously and complements the system’s monitoring and management processes. The workflow begins with the system automatically retrieving the latest sensor readings and plant type information to feed into the AI model. The model performs data validation and executes a prediction algorithm to determine the plant’s health status. If the first attempt fails, a retry mechanism allows for a second execution. Depending on the outcome, the system either stores the prediction result or logs the failure. Firebase Cloud Messaging (FCM) is integrated to notify administrators of any failures encountered. Unlike other components of the system, this process is entirely automated, includes fault tolerance logic, and integrates proactive notification mechanisms. It highlights the system's intelligence layer and closes the loop between sensing, analysis, and actionable feedback.

4. Recommend action:



This diagram illustrates the AI-powered recommends action for plants, which operates autonomously and complements the system’s monitoring and management processes. The workflow begins with the system automatically retrieving the latest sensor readings and plant type information to feed into the AI model. The model performs data validation and executes a prediction algorithm to determine the plant’s needs. If the first attempt fails, a retry mechanism allows for a second execution. Depending on the outcome, the system either stores the prediction result or logs the failure. Firebase Cloud Messaging (FCM) is integrated to notify administrators of any failures encountered. Unlike other components of the system, this process is entirely automated, includes fault tolerance logic, and integrates proactive notification mechanisms. It highlights the system's intelligence layer and closes the loop between sensing, analysis, and actionable feedback.

5. Monitor assigned plant health:

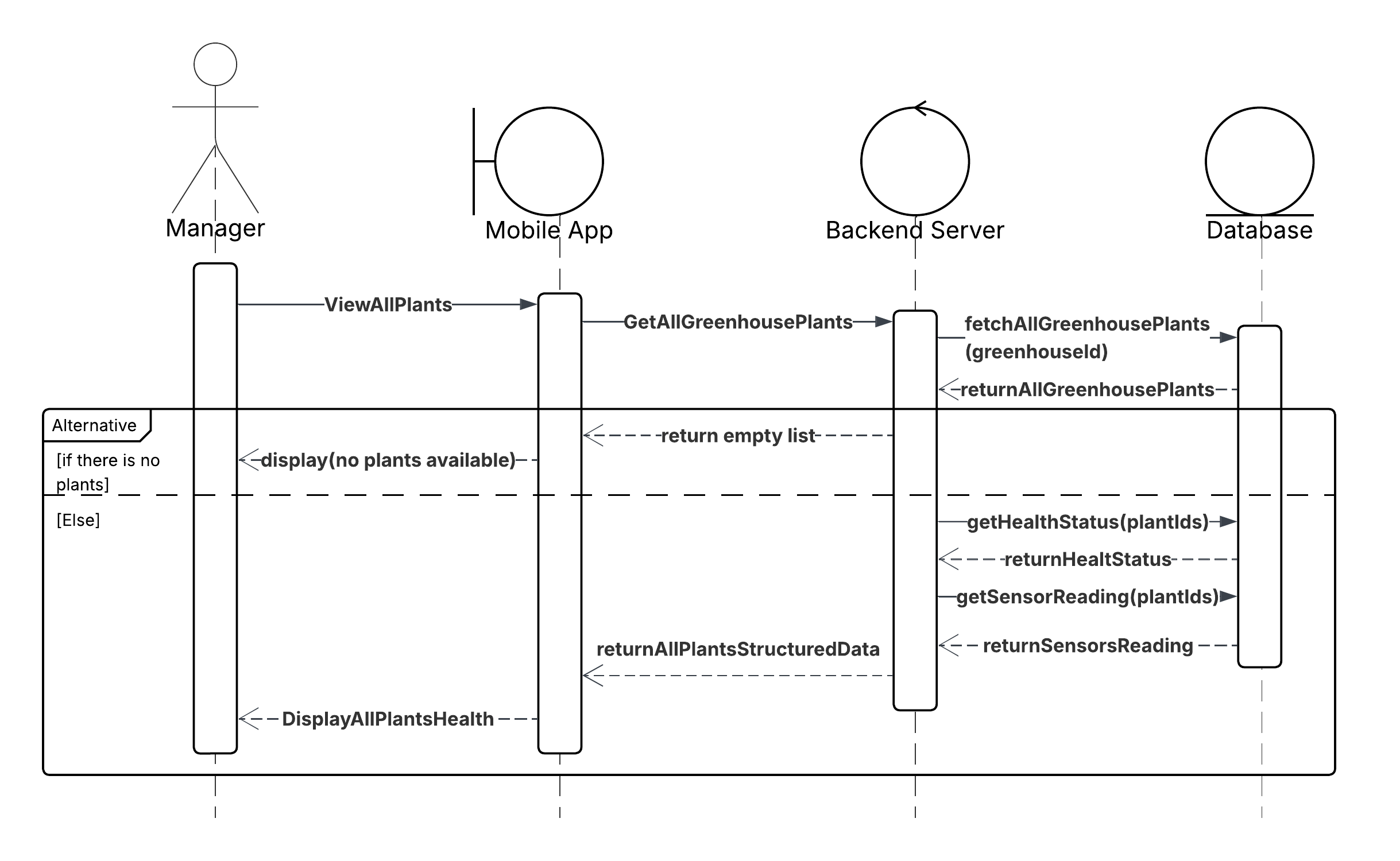


This diagram illustrates the workflow for monitoring the health of plants assigned to individual farmers. It is specifically designed to provide a personalized view that focuses solely on the plants each farmer is responsible for. Unlike administrative or general monitoring processes, this interface allows farmers to directly and effectively track the health of their own plants.

The flow begins when the farmer initiates a request through the mobile app to view their assigned plants. The app sends this request to the backend server, including the farmer’s unique identifier farmerId. The server processes the request by querying the database for all plants assigned to that farmer. If no assigned plants are found, the system responds with a clear message indicating that no plants are currently assigned, avoiding an empty or confusing interface.

If assigned plants are found, the system proceeds to gather relevant data for each one. First, it retrieves the basic information of the plants. Then, two key services are triggered: one to obtain the health status of each plant, and another to fetch sensor readings such as temperature, humidity, and soil moisture levels. These pieces of data are then combined into a structured and organized format. The mobile app receives this structured dataset and displays it in a health overview dashboard. This view provides the farmer with a comprehensive and visual representation of each plant’s condition, enabling informed and timely decision-making based on accurate and real-time information about both plant health and surrounding environmental factors. The clean and structured display enhances usability and responsiveness.

6. Monitor all greenhouse plants:



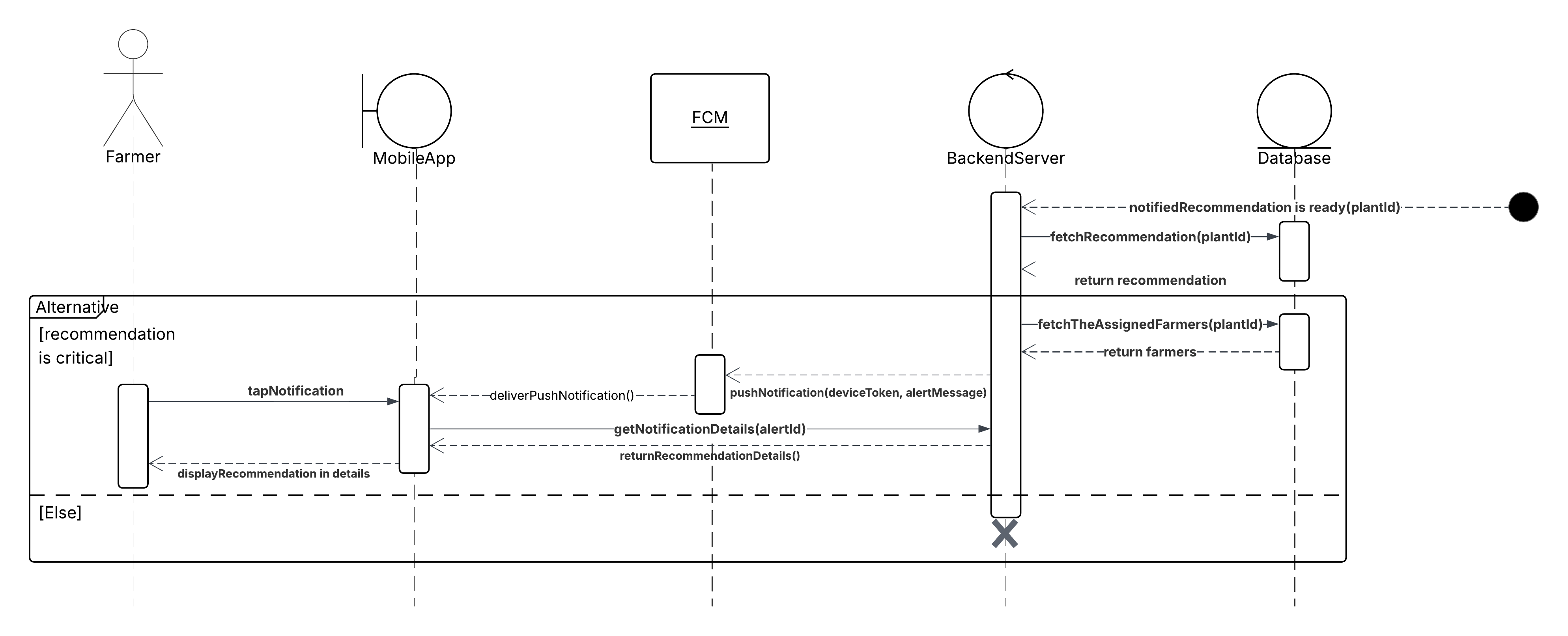
This diagram illustrates the plant monitoring workflow within the agricultural system, emphasizing data observation and analysis rather than the creation, modification, or deletion of records. Unlike previous diagrams focused on management, this workflow is tailored for the Manager role and centers around viewing comprehensive plant-related information to support informed decision-making.

The process begins when the Manager initiates a monitoring request through the mobile application. This request triggers the system to retrieve all plants associated with a specific greenhouse. If no plants are available, the system gracefully handles the empty state and informs the user. If plant data exists, the mobile app proceeds to display it in a structured and informative layout.

On the backend server, the workflow involves querying all plant records relevant to the Manager’s assigned greenhouse. Upon receiving the request, the backend interacts with the database to retrieve a list of plants. Once the basic data is obtained, the system performs data enrichment by invoking specialized functions: one to assess the health status of each plant and another to retrieve environmental sensor readings (e.g., temperature, humidity, soil moisture). These enriched data sets are then compiled into a structured response, which is sent back to the mobile app for presentation.

The Database supports this process by supplying comprehensive plant records and additional contextual data as needed. It is also responsible for identifying and returning an empty data set if no plant records are found for the specified greenhouse, enabling the system to respond appropriately.

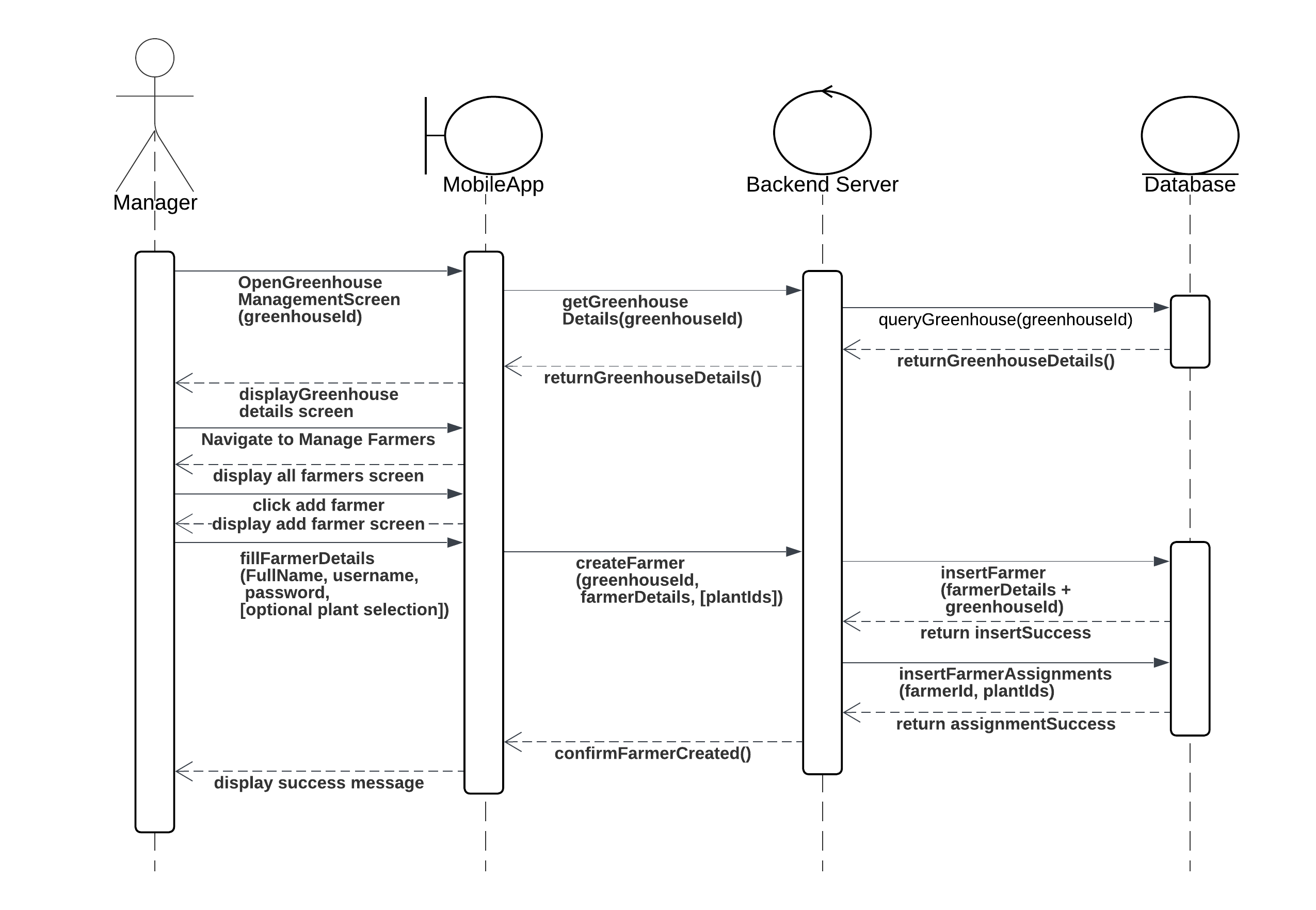
7. Receive notification & alert:



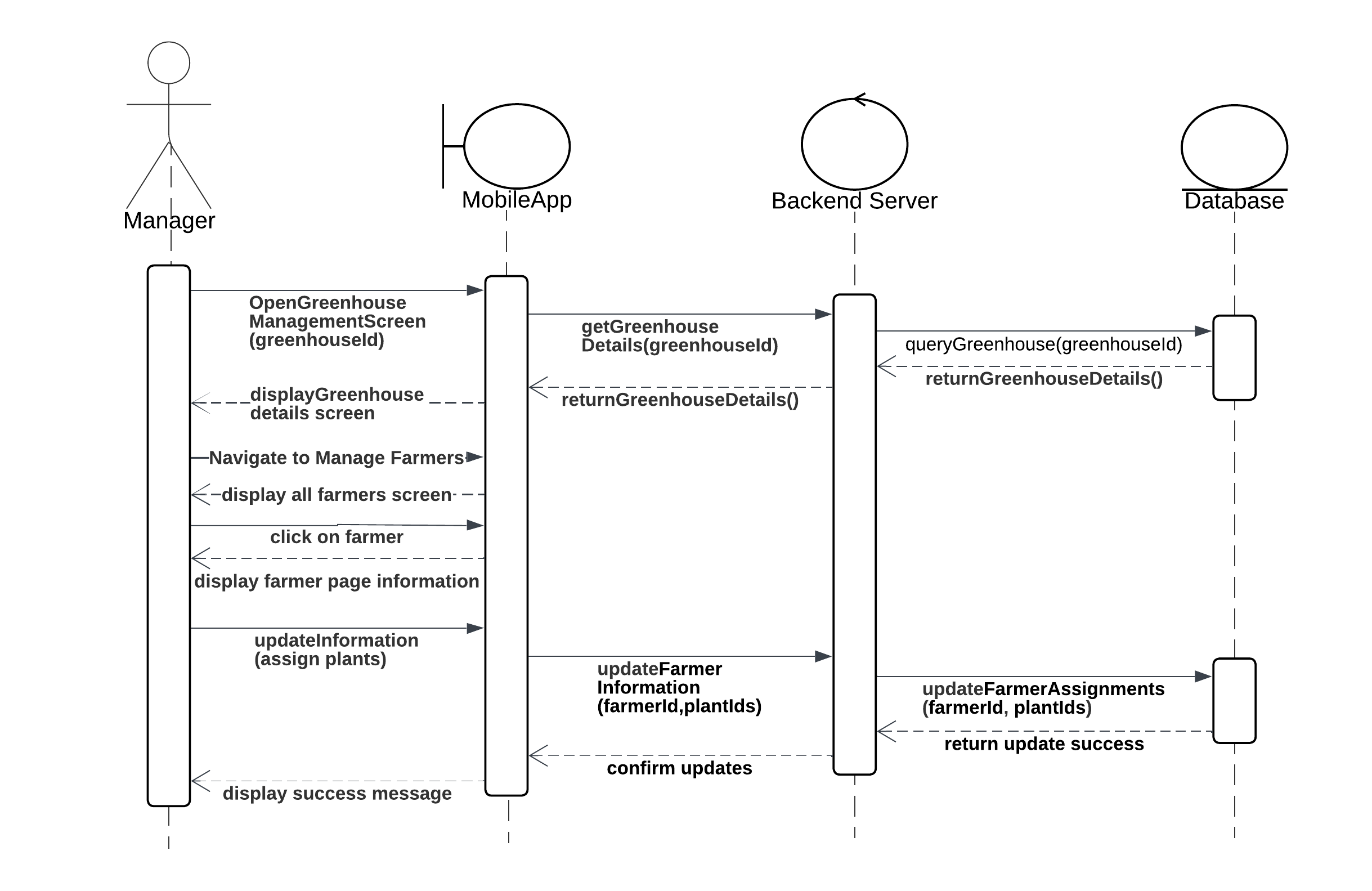
This diagram illustrates the notification delivery system, which introduces an event-driven communication workflow distinct from the system’s monitoring, prediction, and management processes. The process begins after the backend server notified that the recommendation is ready for specific plant, the backend server fetches the recommendation for this plant, if the recommendation is critical, then to deliver the notification for the plant’s assigned farmers, the backend server fetches the device tokens for these farmers and then send requests to FCM to push the notification to these farmers. After the farmer receives the notification label and he taps on it, the mobile app sends a request to the server to get all the details to display it to the farmer.

8. Manager Farmers:

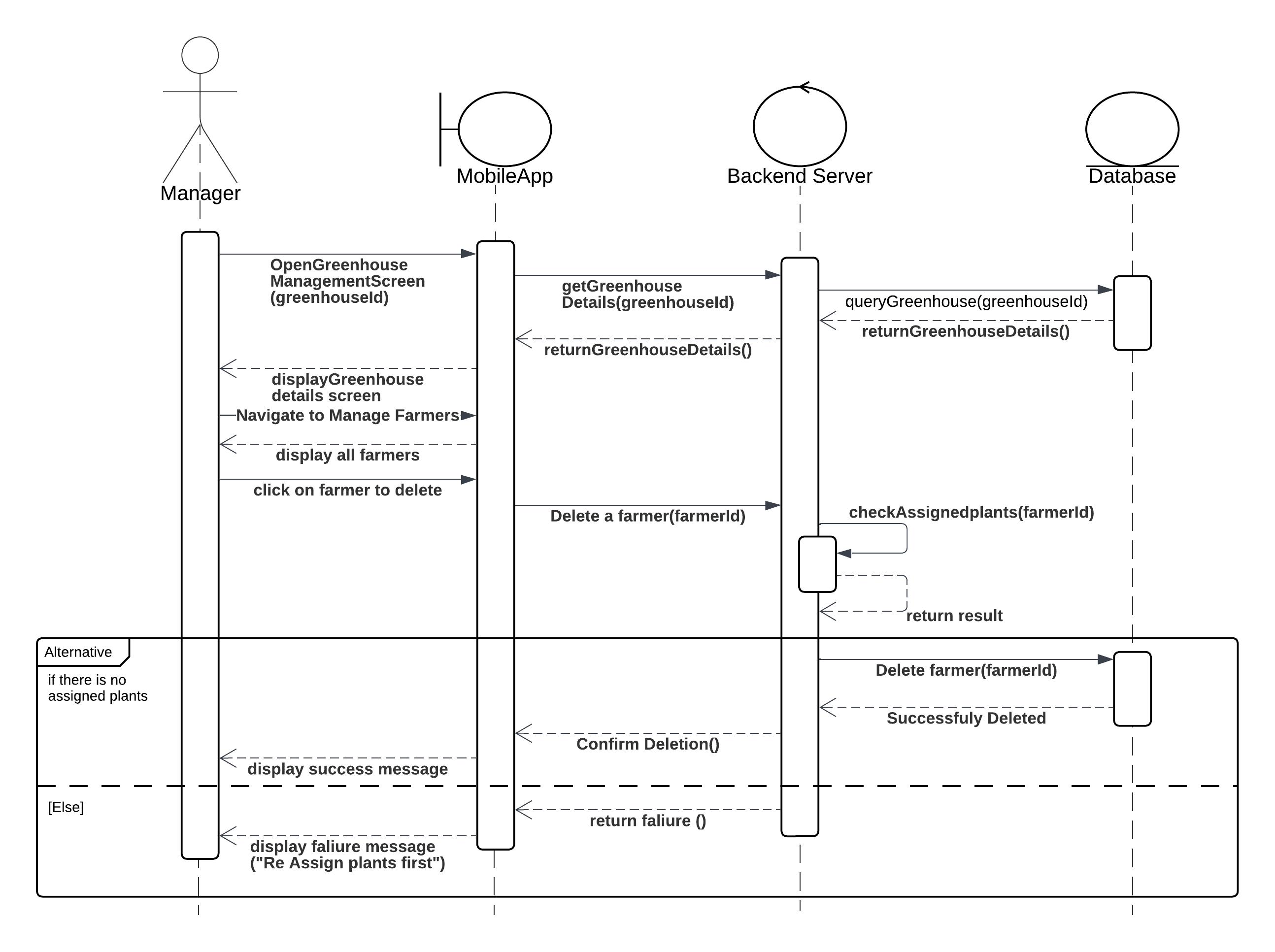
(Add)



(Update)



(Delete)



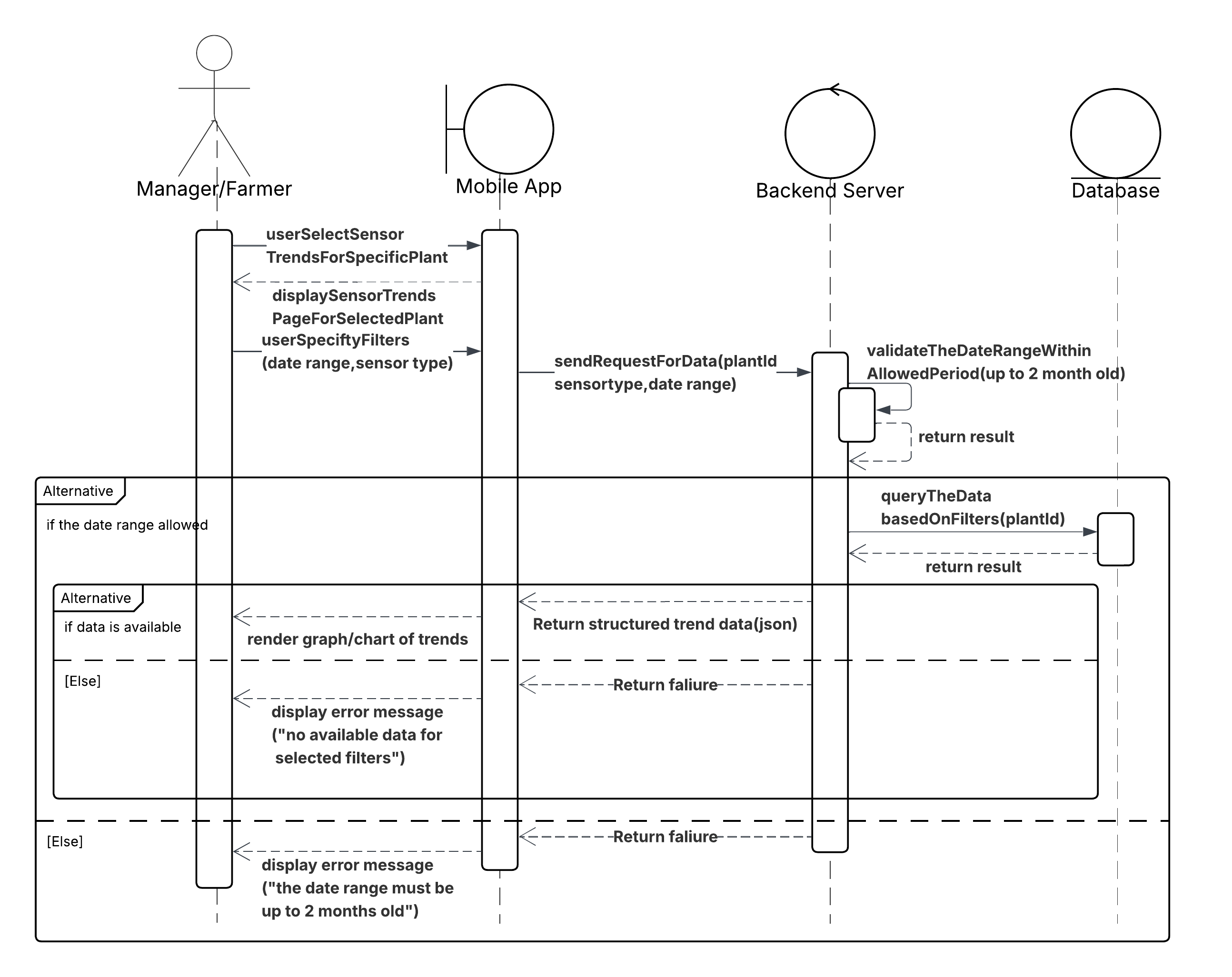
These three diagrams illustrate the complete process of managing farmers within the Smart Agri-Guard system, covering how farmers are added, updated, and deleted, along with their plant assignments inside a specific greenhouse.

The first diagram shows how a manager can add a new farmer to specific greenhouse. Through the mobile app, the manager selects a specific greenhouse, navigates to the farmer management section, the manager select add new farmer, the manager inputs the required details such as the farmer’s credentials and assigns specific plants during this step. Once the information is submitted, the backend processes the request, stores the data, and creates the new farmer profile along with any initial assignments.

The second diagram shows how a manager can update on specific farmer. After the manager navigates to manage farmers for specific greenhouse, the manager chooses a farmer to update his information, a farmer page information displayed, then the manager can update his information like adding new assignments, then sends the updated data to the backend, which stores the new assignments and confirms the success of the operation.

The third diagram shows how a manager can delete a specific farmer. Before allowing the deletion, the system checks whether the farmer still has any assigned plants. If so, the deletion is blocked, and the manager is informed that the farmer cannot be removed until all plant assignments are cleared. If there are no active assignments, the farmer is safely deleted from the system.

9. View sensor trends:



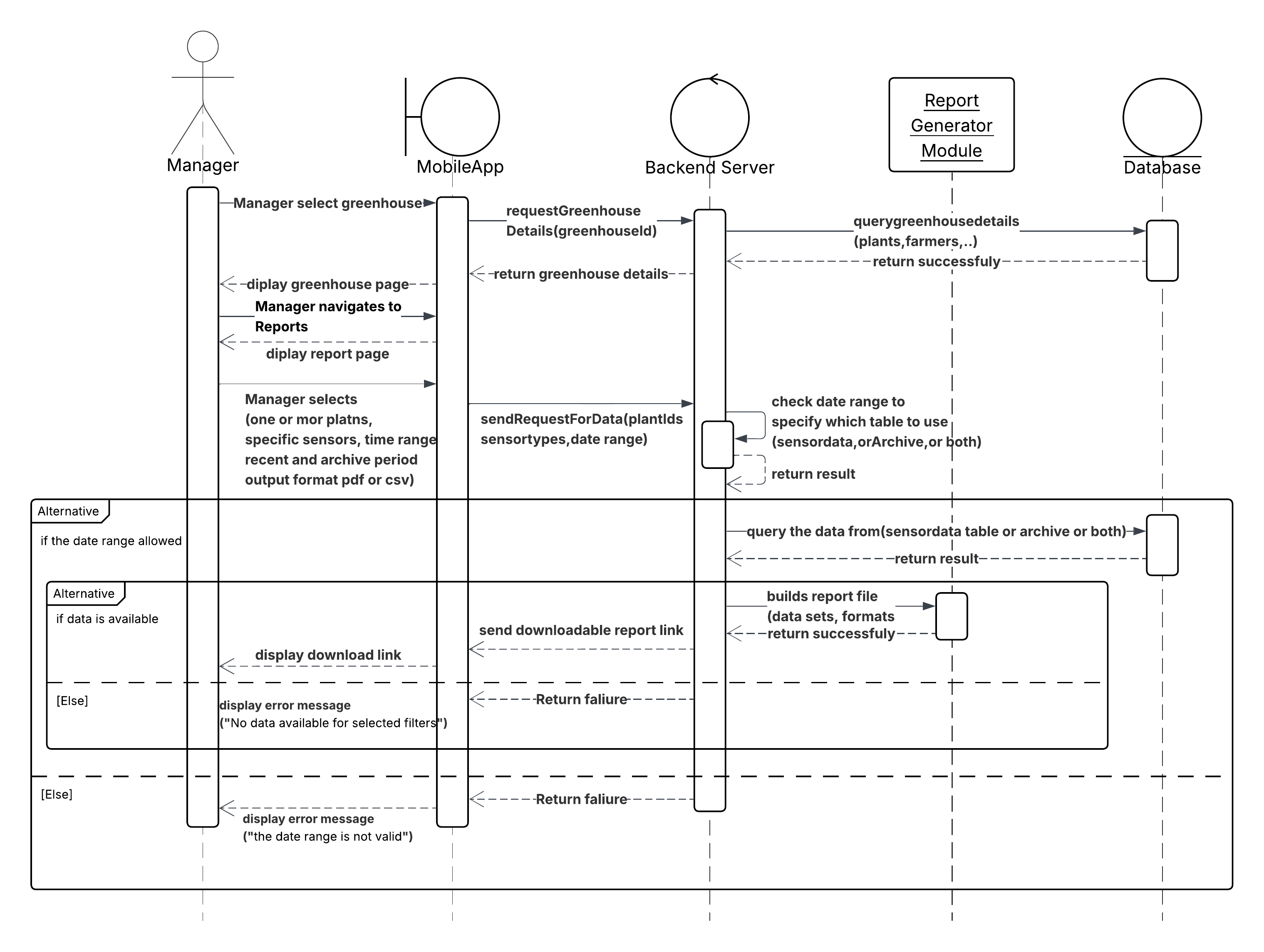
This diagram illustrates the workflow for viewing recent sensor data trends in the Smart Agri-Guard system. Unlike the archival access process, this workflow is limited to data that is up to two months old, enabling users—either Managers or Farmers—to quickly analyze short-term environmental patterns. The interaction begins when the user selects a plant and applies specific filters such as date range and sensor type. The mobile app sends a structured request to the backend server, which validates that the selected date range falls within the permitted two-month window. If valid, the server queries the database for relevant sensor readings based on the provided filters and returns the data in a JSON format suitable for trend visualization. If the data is available, the app renders a graph or chart to display the trends. Otherwise, appropriate error messages are returned to the user, either indicating an invalid date range or a lack of matching sensor data. This feature supports informed decision-making by enabling users to quickly assess plant conditions and environmental fluctuations over recent weeks.

10. View archived sensor trends:



This diagram illustrates the workflow for accessing historical sensor data within the Smart Agri-Guard system, focusing on archived trends rather than real-time monitoring. The process is initiated by a manager who selects a specific greenhouse and navigates to view archived sensor trends, then defines filters such as plant, sensor type, and a date range spanning from 2 months to 1 year in the past. The mobile application sends a structured request to the backend server, which first validates the date range against archival boundaries and ensures that relevant data exists for the selected filters. If valid, the backend queries a dedicated archival database, retrieves the appropriate time-series data, and formats it into a structured JSON response suitable for visualization. The system includes robust error handling to manage invalid date ranges or empty result sets, returning user-friendly messages when data is unavailable or outside the acceptable archival window. This workflow enables longitudinal analysis of environmental and plant conditions, complementing the system’s real-time monitoring by supporting data-driven, long-term agricultural decision-making.

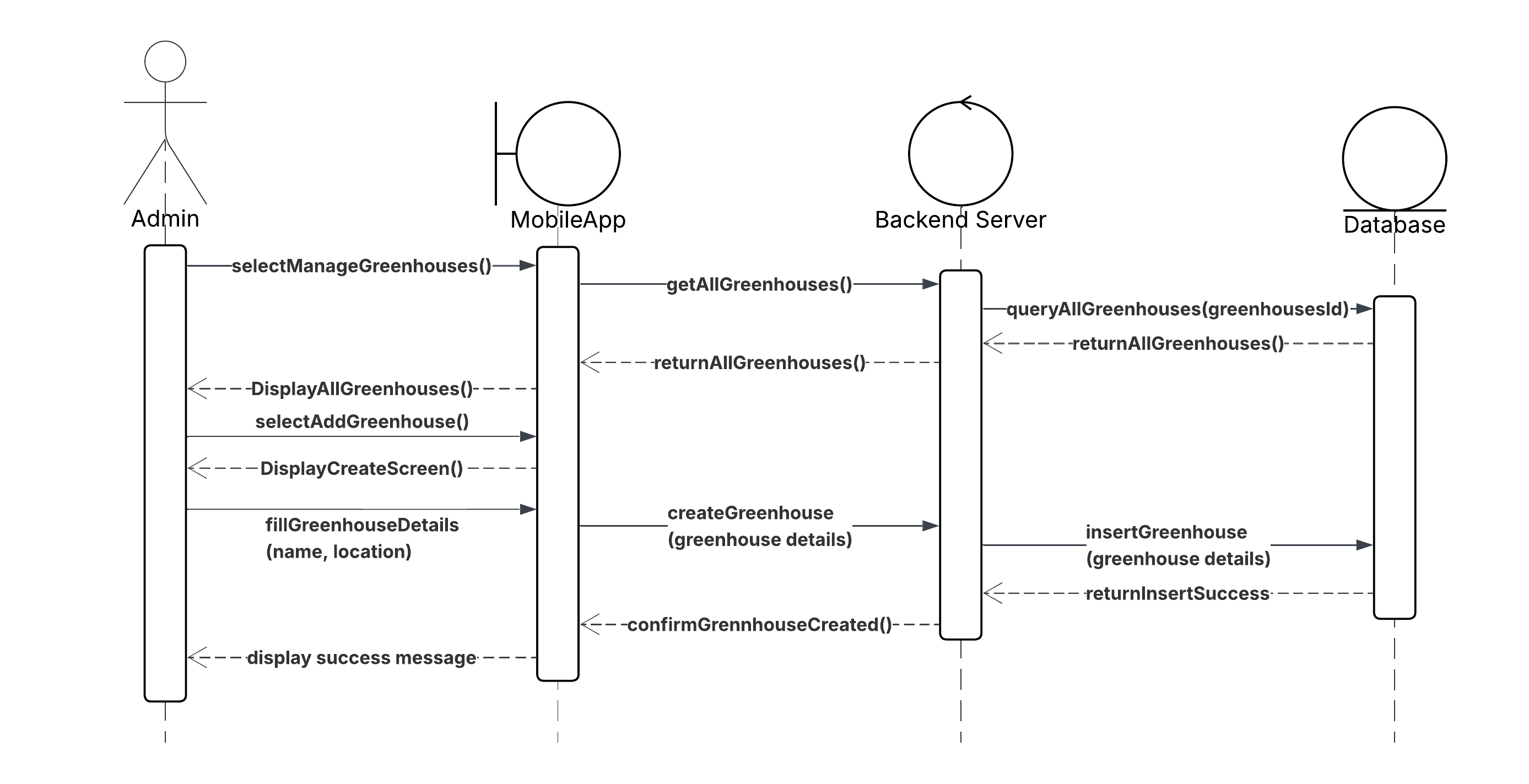
11. Generate reports:



This diagram illustrates the process of generating a report by a manager through the mobile application. The Manager begins by selecting a specific greenhouse and navigating to the Reports section. They then define report parameters such as selected plants, sensor types, time range (recent or archived), and desired output format (PDF or CSV). The mobile app sends a data request to the backend server, which verifies the date range to determine whether to retrieve data from the live sensor table, the archive, or both. The appropriate query is then performed on the database. If the date range is valid and data exists for the selected criteria, the Report Generator Module builds a downloadable report file and returns a download link to the user. If the data is unavailable, or if the provided date range is invalid, the system returns a descriptive error message to the mobile application.

12. Manage greenhouses:

(Add)



(Update)

A diagram of a software system

AI-generated content may be incorrect.

(Delete)

A diagram of a software project

AI-generated content may be incorrect.

This set of sequence diagrams illustrates the complete workflow for managing greenhouses within an agricultural management system, highlighting the collaboration between the Admin, the Mobile Application, the Backend Server, and the Database. The three main operations depicted are: adding a new greenhouse, deleting an existing greenhouse, and updating greenhouse information. Each diagram reflects a distinct process flow while maintaining a consistent interaction structure across system components.

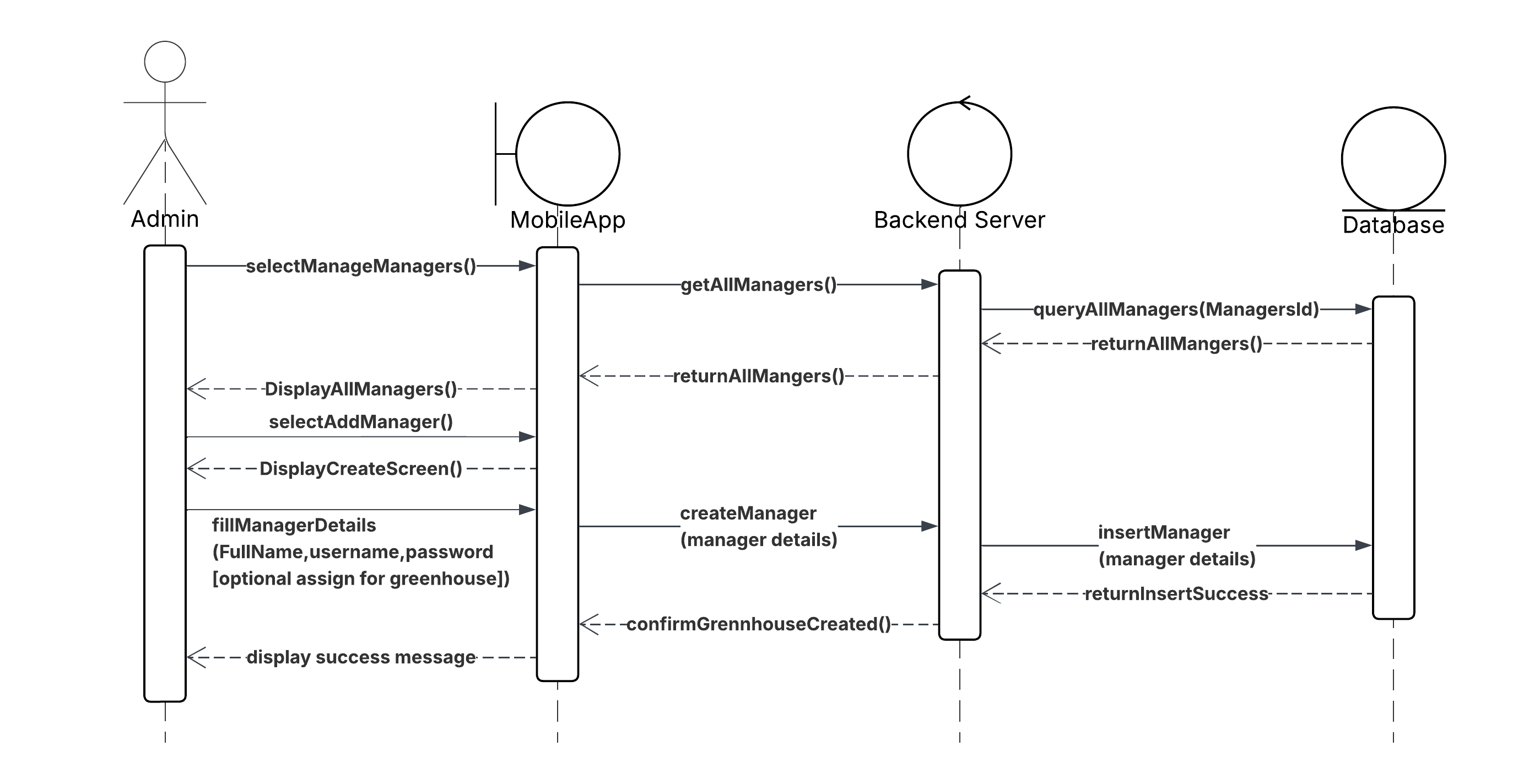
**In the "Add Greenhouse" diagram**, the process starts with the admin selecting the "Manage Greenhouses" option through the mobile application interface. The system displays a list of all greenhouses, after which the admin chooses to add a new one. Upon selecting this option, the interface presents a creation screen where the admin enters the required details such as the greenhouse name and location. These inputs are then sent by the mobile app to the backend server, which inserts the new record into the database. Once the insertion is successful, the server sends a confirmation message back through the app, and a success message is displayed to the admin, confirming the greenhouse has been added.

**In the "Update Greenhouse" diagram**, the admin again navigates to the greenhouse management section and views the list of existing greenhouses. After selecting one for updating, the system displays its current information. The admin may then modify specific details, such as the name. These changes are submitted through the mobile app to the backend server, which updates the relevant record in the database. Upon successful completion, the server returns a confirmation, and the app displays a message indicating the update was successful.

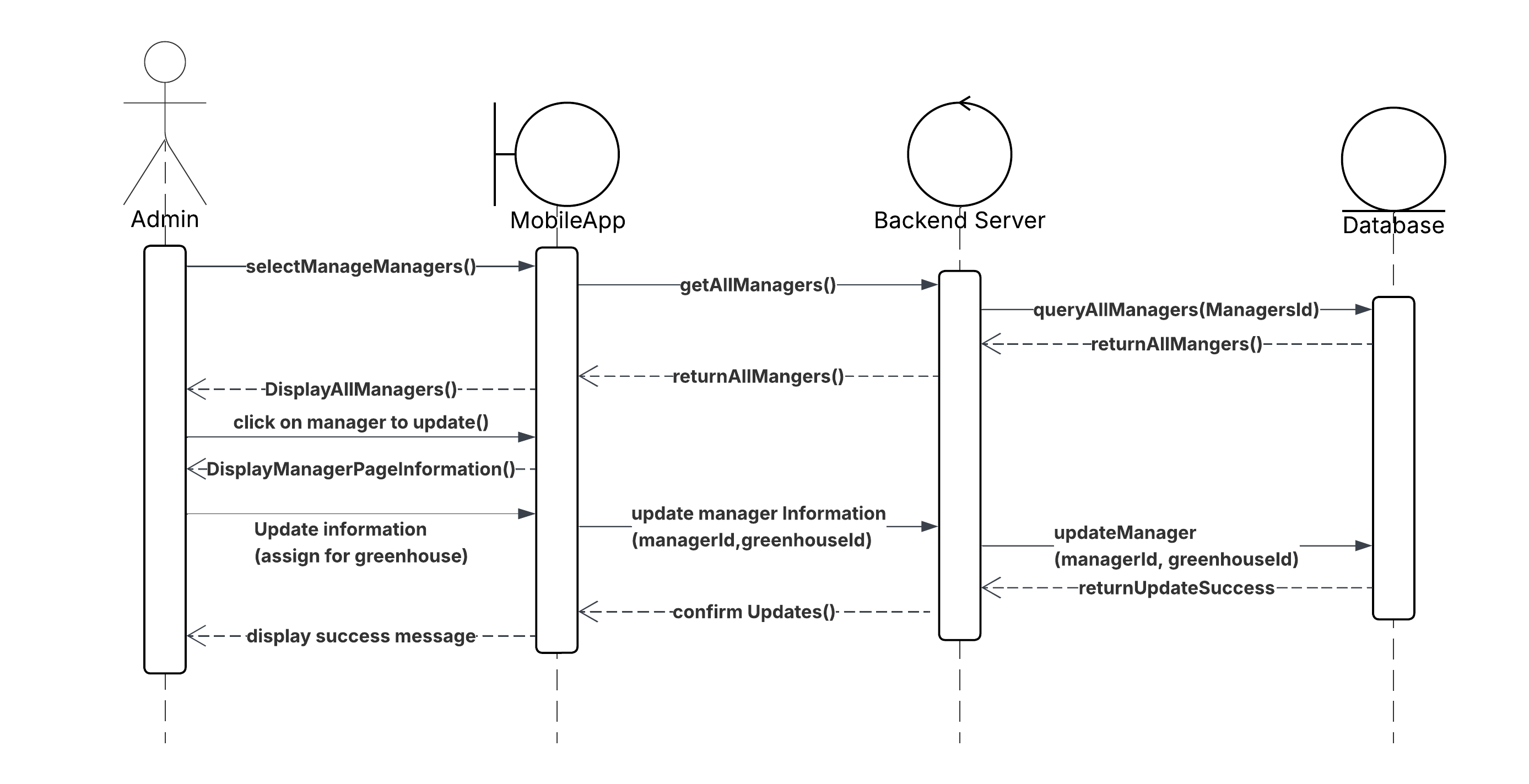
**In the "Delete Greenhouse" diagram**, the process again begins with the admin selecting "Manage Greenhouses" and viewing the current list. The admin then chooses a specific greenhouse for deletion. The mobile app sends this deletion request to the backend server, which performs a crucial validation step: it checks the database to determine whether the greenhouse is associated with any active plant or farmers. If no such associations exist, the server proceeds to delete the greenhouse and returns a success response, which is then shown to the Admin. However, if the greenhouse is still linked to plants or farmers, the deletion is aborted, and a failure message is sent back, instructing the admin to reassign or remove those dependencies first. This validation safeguards operational integrity by ensuring no greenhouse is deleted while still in use.

13. Manager managers:

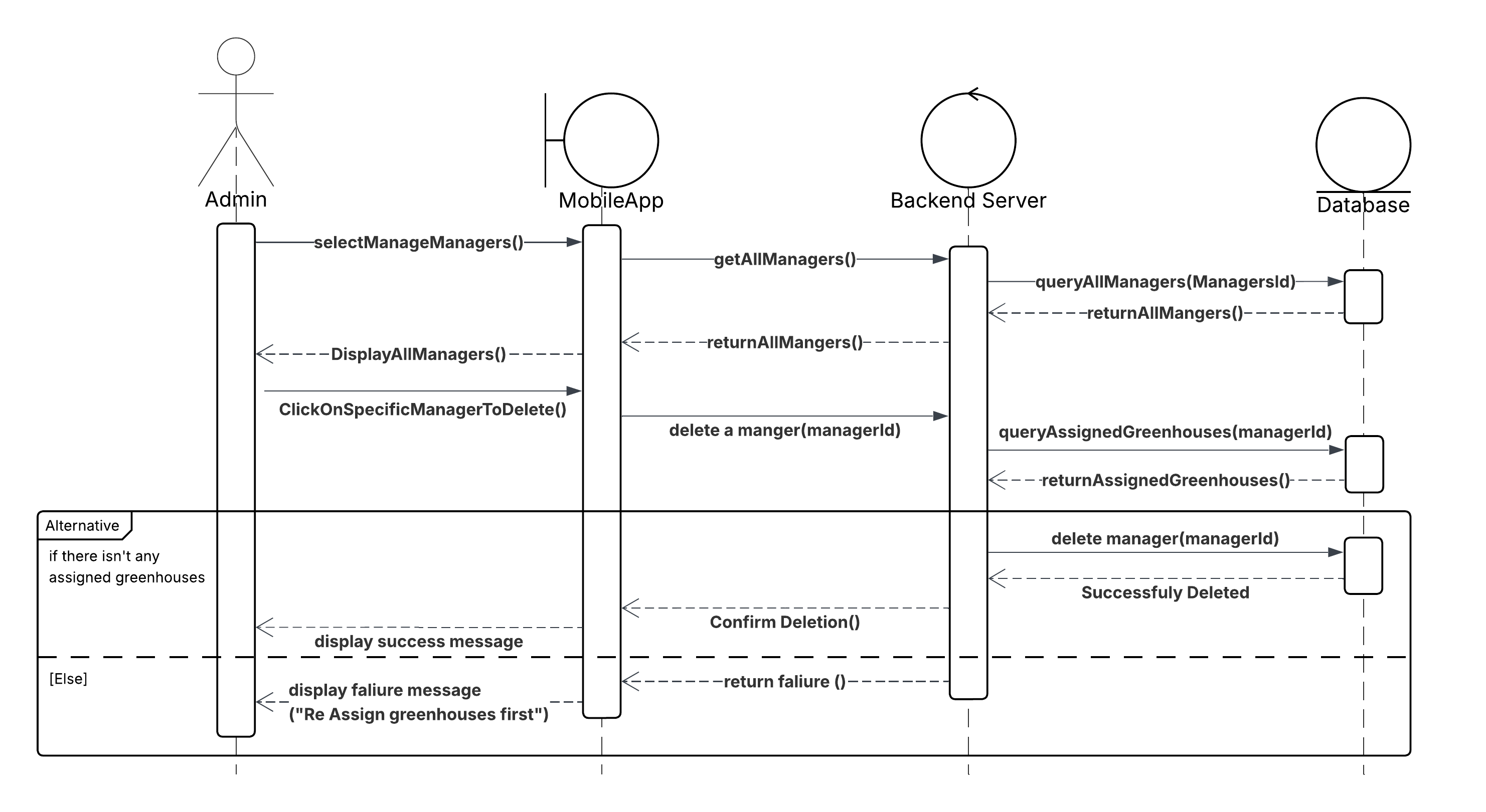
(Add)



(Update)



(Delete)



This set of sequence diagrams illustrates the complete workflow for managing manager accounts within an agricultural system, focusing on interactions between the Admin, the Mobile Application, the Backend Server, and the Database. The operations include adding a new manager, deleting an existing manager, and updating manager responsibilities. Each operation follows a clear and structured flow, ensuring data integrity and operational security.

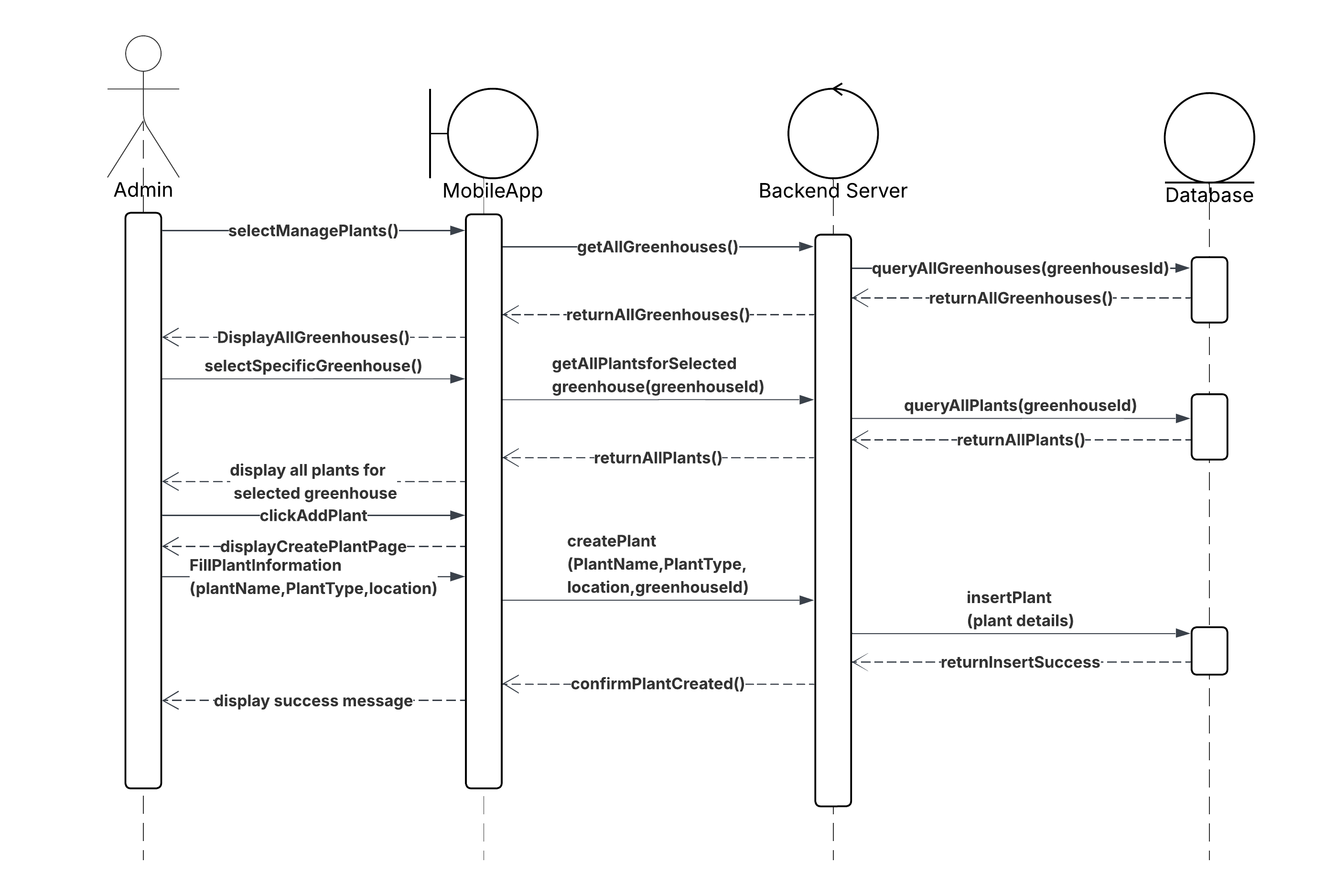
**In the "Add Manager" diagram**, the process starts when the admin selects the "Manage Managers" option via the mobile app. The system retrieves and displays a list of existing managers. The admin then chooses to add a new manager, prompting a form to appear for entering the manager’s full name, username, and password. After submission, the mobile app sends the data to the backend, which processes the creation request and stores the new manager’s credentials in the database. Upon successful insertion, a confirmation message is returned and displayed to the admin, indicating that the account has been successfully created.

**The "Update Manager" diagram** represents the flow for modifying a manager’s responsibilities. After selecting a manager from the list, the admin is presented with the current details of that account. The admin can then assign the manager for specific greenhouse, then the mobile app sends a request to the backend server to update the manager’s information. The backend updates the information in the database and returns a confirmation message indicating that the update was successful. This function is designed to support routine role changes without altering core account credentials.

**The "Delete Manager" diagram** shows the process of safely removing a manager from the system. The admin views the manager list and selects a specific manager for deletion. Before proceeding, the system performs a validation check to determine whether the manager has any active responsibilities. If no such responsibilities exist, the backend deletes the manager’s account and confirms the deletion. If the manager still has responsibilities, the deletion is blocked, and the admin is notified with an appropriate message. This check ensures that the system remains consistent and that responsibilities are not left unassigned.

14. Manage plants:

(Add)



(Update)

A diagram of a plant

AI-generated content may be incorrect.

(Delete)

A diagram of a software project

AI-generated content may be incorrect.

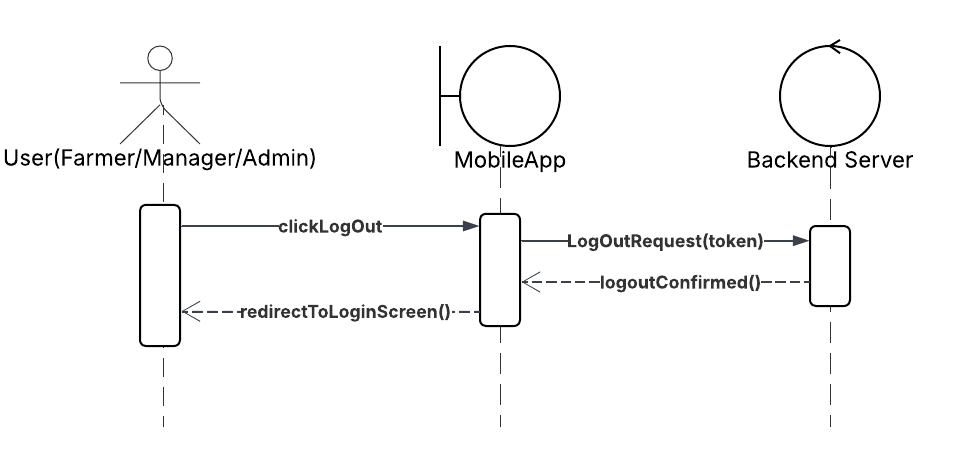
This set of sequence diagrams illustrates the complete workflow for managing plant records within an agricultural system, involving interaction between the Admin, Mobile Application, and Backend Server. The operations include adding a new plant, deleting an existing plant, and updating plant information. Each process follows a structured and consistent flow, ensuring operational control and maintaining data relationships, particularly regarding plant responsibilities.

**In the "Add Plant" diagram**, the workflow begins with the admin accessing the plant management interface through the mobile application. After selecting the desired greenhouse, the system retrieves and displays relevant plant data. The admin then initiates the add process, entering key information such as the plant's name, type, and location. Once submitted, the mobile app sends the data to the backend, which stores the new plant record in the database. A confirmation message is returned and displayed to notify the admin of successful creation.

**In the "Update Plant" diagram**, the admin begins by accessing the plant list and selecting the specific plant to be modified. The system retrieves the current plant details, the admin then modifies the location for example, and the update request is sent to the backend via the mobile app. The backend updates the plant's data and confirms the successful change, which is then displayed to the Admin.

**The "Delete Plant" diagram** represents the process of safely removing a plant from the system. After navigating through the plant management interface and selecting a specific plant, the system performs a validation check to determine if the plant is currently assigned to any farmers. If no such assignments exist, the backend proceeds with the deletion and confirms the success of the operation. However, if the plant is still linked to any farmer, the system blocks the deletion and returns a failure message, ensuring responsibilities are not left unassigned and maintaining operational integrity.

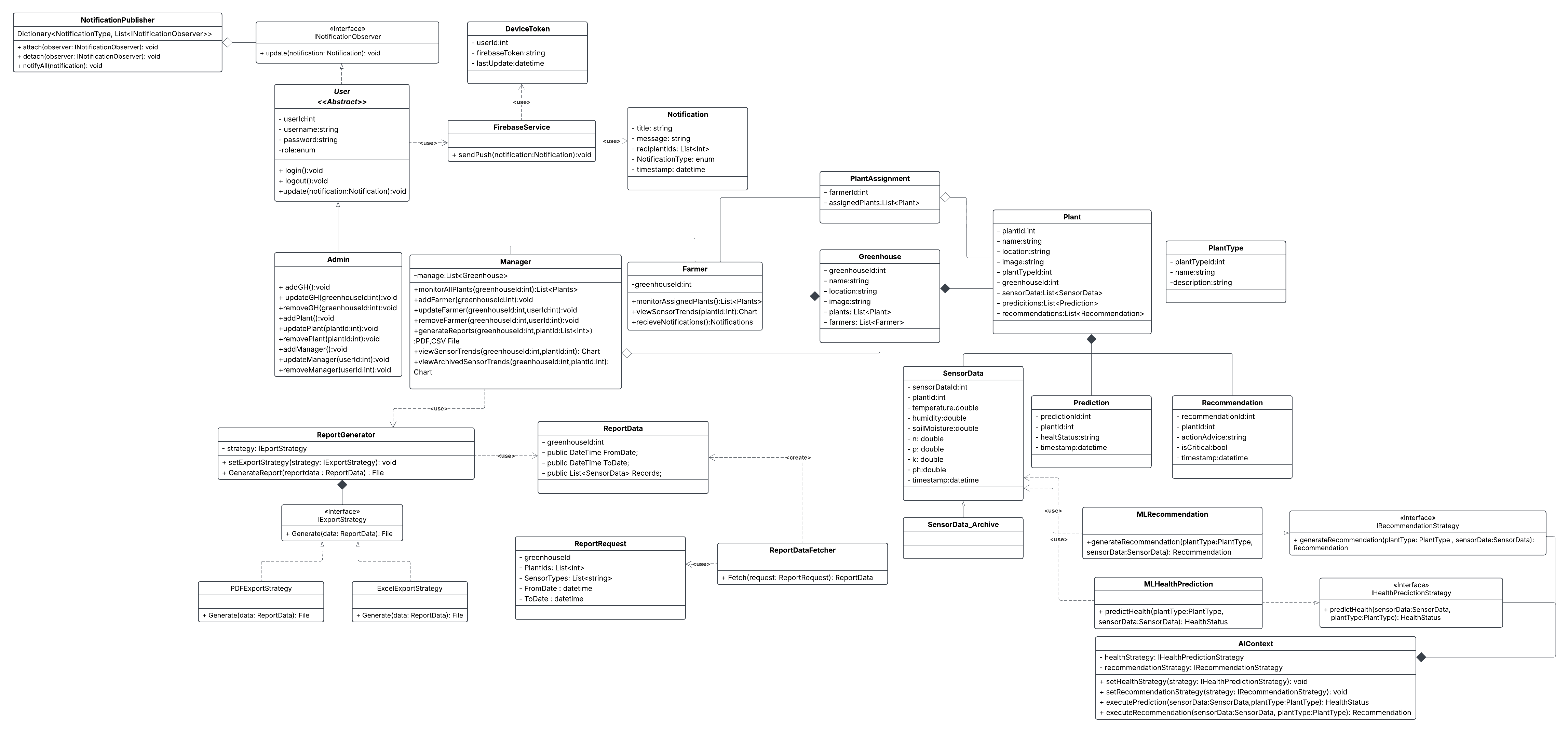
15. Log out:



This sequence diagram illustrates the logout process for a user (who could be a Farmer, Manager, or Admin) using a mobile application. The process begins when the user initiates a logout action by clicking the "Log Out" button in the app. This triggers the mobile application to send a Log Out Request, which includes the user’s authentication token, to the backend server. The backend server processes this request, confirming the logout by invalidating the token or terminating the session. Once the server confirms the logout operation, it sends a log out Confirmed response back to the mobile app. Upon receiving this confirmation, the mobile application redirects the user to the login screen, effectively completing the logout process.

#### 

#### 4.2.2 Class Diagram



This UML class diagram illustrates the architecture of the Smart Agri-Guard System, designed to support different user roles, monitor agricultural data through sensors, apply machine learning for prediction and recommendation, and provide real-time notifications and reporting features. The system aims to optimize greenhouse management and decision-making by offering a structured, automated, and intelligent platform.

At the core of the system is the User class, which serves as the base class for all users. It contains essential attributes such as userId, username, password and user role. From this class, three specialized roles inherit: the Admin, the Manager, and the Farmer. Each role has its own responsibilities and access rights. The admin is responsible for managing greenhouses, managers, plants and controlling system-wide configurations. The Manager oversees the operations of greenhouses, managing farmers and handling reporting tasks. On the other hand, the Farmer interacts directly with the physical environment, monitoring sensor data in their assigned greenhouses.

The Greenhouse class represents the physical farming environment. Each greenhouse contains attributes such as greenhouseId, name, and location. It is associated with a collection of plants and farmers. The assignment of plants to greenhouses is handled through the PlantAssignment class, which links a specific plant to farmers. The Plant class includes detailed information such as plantId, name, plantType, location in the greenhouse and other botanical data. Additionally, the Plant is related to the PlantType class, which describes the classification and characteristics of the plant.

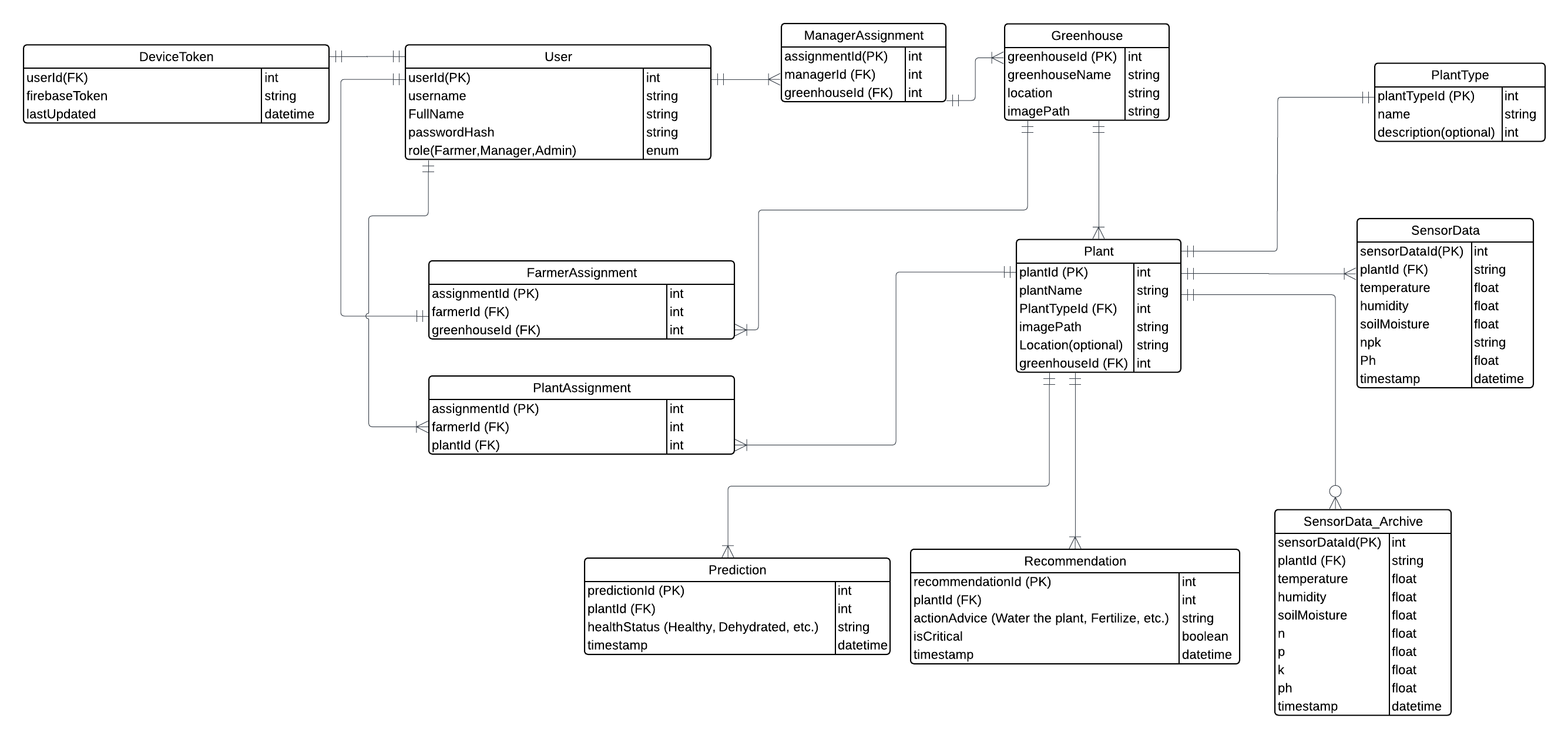
To monitor plants, the system uses the SensorData class. This class collects real-time data such as temperature, humidity, soil moisture, NPK, and soil PH. This sensor data is crucial for understanding the state of the environment in each greenhouse. For historical analysis, old data is stored in the SensorData\_Archive class, ensuring that trends can be tracked over time.

The AI system design leverages the Strategy Design Pattern to separate the logic for plant health prediction and recommendation generation. Two main strategy interfaces, IHealthPredictionStrategy and IRecommendationStrategy, define how predictions and actions are generated from sensor data and plant types. The AIContext class dynamically selects and executes these strategies, enabling flexibility to plug in different ML models (e.g., MLHealthPrediction, MLRecommendation). This structure ensures that the AI module remains modular, scalable, and easy to upgrade or replace as better models become available or new crops are supported.

The notification system in this design follows the Observer Design Pattern, where NotificationPublisher manages the subscription and broadcasting of notifications to observers implementing the INotificationObserver interface. Users (Admin, Manager, Farmer) act as observers and receive notifications via the update() method. Notifications are delivered through FirebaseService, which uses stored DeviceToken values to send push messages to user devices. This modular design allows flexible notification targeting based on roles and notification types, and it supports future expansion to additional delivery channels or custom logic per user role.

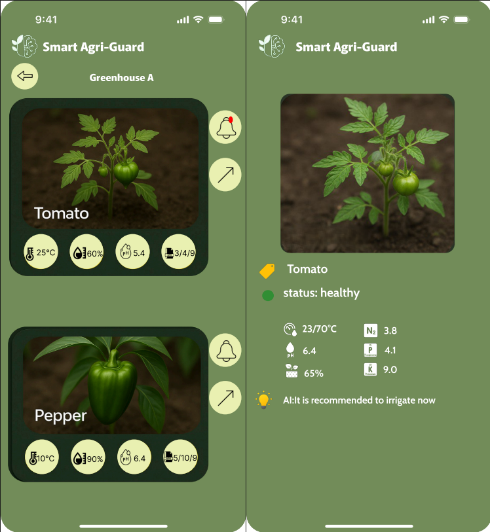
The report generation system applies the Strategy Design Pattern through the IExportStrategy interface, which defines a unified method for generating reports. The ReportGenerator class delegates the report creation to specific strategy implementations such as PDFExportStrategy and ExcelExportStrategy, allowing the user to choose the desired export format. This design supports generating reports from filtered ReportData, which is fetched using the ReportDataFetcher based on parameters in ReportRequest. This approach enables flexible and extensible reporting while maintaining a clean separation between data processing and export logic.

#### 4.2.3 Data Storage Organization



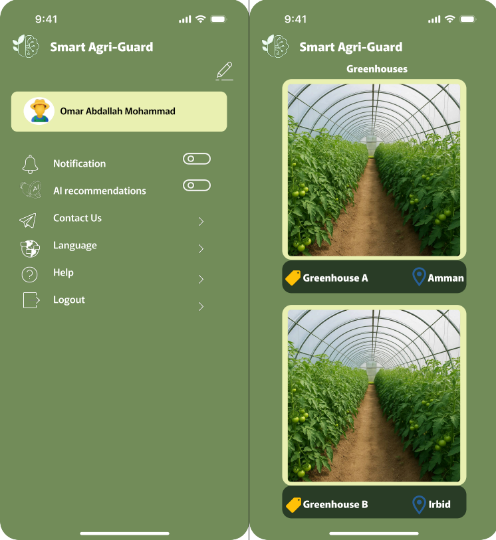
The database diagram defines the core structure for storing system data and is consistent with the class diagram in both logic and relationships. Each entity in the diagram—such as User, Greenhouse, Plant, SensorData, and Recommendation—maps directly to a corresponding class with similar attributes and foreign key relationships. The use of assignment tables like FarmerAssignment, ManagerAssignment, and PlantAssignment ensures proper many-to-many mappings, just as reflected in the class associations. Additionally, the DeviceToken table supports the notification mechanism aligned with the observer pattern in the class diagram. Overall, both diagrams work in harmony to support a modular, scalable, and maintainable system.

**4.3 User Interface Prototyping**

  A screenshot of a smart farm

AI-generated content may be incorrect.

(Assigned plant/all plant) (View Sensor Trends) (plants notification)

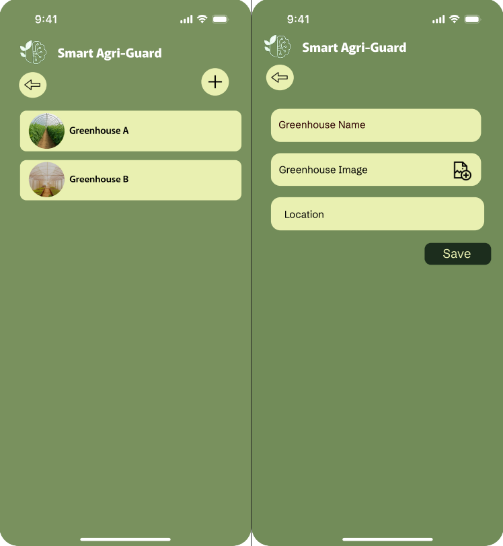
 Screens screenshot of a smart farm

AI-generated content may be incorrect. A screenshot of a phone

AI-generated content may be incorrect.

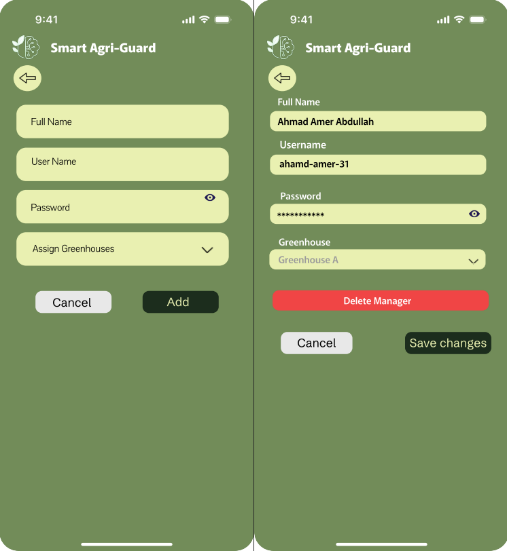
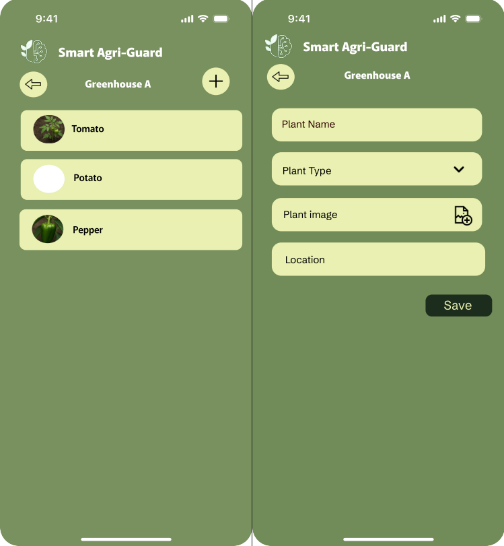
(archived sensor trends) (Generate report)

Screens screenshot of a green screen

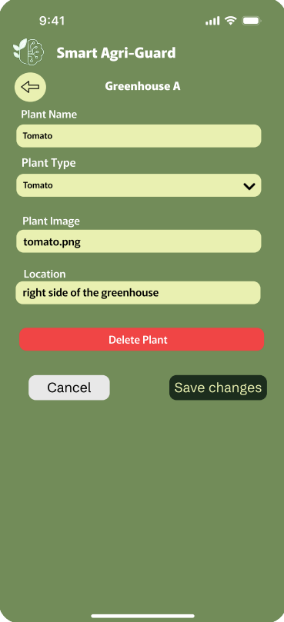
AI-generated content may be incorrect.  

(add/update/delete farmer) (system reports) (add greenhouse)

Screens screenshot of a green screen

AI-generated content may be incorrect.  

(update/delete greenhouse) (add/update/delete manager) (add plant)



(update/delete plant)

**CHAPTER 5: Testing Plan**

**5.1 Tools**

**1. API Testing: Postman**

**Purpose:**Testing the project’s RESTful APIs.  
**Explanation:**Used to send HTTP requests (GET, POST, etc.), inspect responses, and validate API functionality and integration.

**2. AI Model Testing: Jupyter Notebook**

**Purpose:**Developing and testing machine learning models. **Explanation:**A Python-based environment for writing, training, and evaluating AI models before deployment.

**3. Mobile App Testing: Flutter DevTools**

**Purpose:**Debugging and performance analysis of the Flutter mobile app. **Explanation:**Helps inspect UI, track app performance, and diagnose errors in real time.

**4. Flutter Test (Built-in Testing Library)**

**Purpose:** Writing unit and widget tests for the Flutter app.

**Explanation**: Ensures the UI components and business logic behave as expected across different devices and scenarios.

**5. Backend Unit & Integration Testing: Visual Studio 2022**

**Purpose:**The primary integrated development environment (IDE) for developing the backend of the project using ASP.NET Core. **Use:**It will be used for unit testing and integration testing of the API endpoints using tools like xUnit or NUnit to ensure correct functionality of the backend.

**6. Hardware Sensor Testing: Arduino IDE (Serial Monitor)**

**Purpose:**Monitoring sensor data from ESP32 modules.  
**Explanation:**Displays real-time sensor readings (e.g., temperature, humidity) via serial communication for validation.

**7. Database Testing: SQL Server Management Studio**

**Purpose:**Managing and testing the SQL Server database. **Explanation:**Executes queries, checks data integrity, and ensures proper CRUD operations.

**8. SSL Labs' SSL Test**

**Purpose:**Testing SSL/TLS encryption security. **Explanation:**Checks for proper HTTPS implementation and identifies certificate vulnerabilities.

**9. JWT.io Debugger**

**Purpose:**Validating and debugging JWT tokens. **Explanation:**Decodes and verifies authentication tokens for security compliance.

**10.** **Burp Suite**

**Purpose:** Burp Suite can be used for deeper penetration testing, especially for intercepting API requests and testing the authentication system.

**11.** **OWASP ZAP (Zed Attack Proxy)**

**Purpose:** It helps test for vulnerabilities in SSL/TLS communication, weak encryption, and any security flaws in the authentication system.

**5.2 System Testing (Black box)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Use Case** | **Test Description** | **Input** | **Expected Output** | **Pass Criteria** |
| TC1 | Log In | Verify valid login for all roles | Valid username and password | Redirect to role-based dashboard | Correct dashboard loads |
| TC2 | Log In | Handle invalid login credentials | Wrong username/password | Error message: "Invalid username or password" | Error shown |
| TC3 | Monitor Assigned Plant Health | Farmer views assigned plant health | Farmer logs in and selects assigned plant | Display of health status, predictions, and sensor readings | Health data shown |
| TC4 | Monitor All Greenhouse Plants | Manager views all plants in greenhouse | Manager login + greenhouse selection | List of all plant statuses and metrics | Complete data displayed |
| TC5 | Collect Sensor Data | Sensor system collects and formats data | Trigger reading from sensors | Data collected and prepared in correct format | Data is stored or transmitted |
| TC6 | Collect Sensor Data | Sensor system couldn’t collect all sensor data | Trigger reading from sensors | Data collected and assigned null value for the missing data and warned the manager/admin | Data is stored and the manager/admin gets notification for missing data |
| TC7 | Predict Plant Health | AI predicts health condition based on data and plant type | Sensor data input | Output: Healthy / Moderate stress /High stress | Prediction stored in DB |
| TC8 | Predict Plant Health | AI couldn’t predict plant health due to missing data | Sensor data input | Model refused to predict and notified the admin | The admin notified of failure in prediction for plant health |
| TC9 | Recommend Action | AI recommends action based on data and plant type | Sensor data input | Action: Water / Fertilize / None | Recommendation stored in DB |
| TC10 | Recommend Action | AI couldn’t recommend action due to missing data | Sensor data input | Model refused to recommend action and notified the admin | The admin notified of failure in recommendation for plant |
| TC11 | Receive Notification | Farmer receives notification | New critical recommendation | Push notification appears on device | Notification is shown |
| TC12 | View Sensor Trends | View filtered sensor data by plant and date | Date range, sensor type, plant ID (up to 2 months old) | Chart of filtered sensor values | Graph displayed correctly |
| TC13 | View archived sensor trends | Manager views archived sensor data for a specific plant/date | Plant ID, Sensor Type(s), Date Range (older than 2months) | Chart of filtered sensor values | Data retrieved from archive and shown |
| TC14 | Generate Report | Manager generates report by filters | Date range + sensor types + plants | Report file (PDF/Excel) generated | File downloaded successfully |
| TC15 | Manage Greenhouses | Admin adds/edits/deletes greenhouse | Greenhouse details form | DB updated and UI reflects changes | Operations succeed |
| TC16 | Manage Greenhouses | Admin tried to delete greenhouse with plants and farmers assigned to it | Request to delete greenhouse  (greenhouseId) | Refused to delete, showed appropriate message | Message has been shown of failure request |
| TC17 | Manage Farmers | Manager assigns farmers and plants | Farmer ID, plant ID, greenhouse ID | Assignment created and stored | Assignment shown in UI |
| TC18 | Manage farmers | Manage tries to delete a farmer with assigned plants | Request to delete farmer (farmerId) | Refused to delete, showed appropriate message | Message has been shown of failure request |
| TC19 | Manage Plants | Admin adds, edits, or deletes plant for a particular greenhouse | Plant name ,type ,image and location  (greenhouseId) | Plant record added, updated, or removed in the database | UI reflects change, DB updated |
| TC20 | Manage Plants | Admin tries to delete a plant with assigned farmer to it | Request to delete plant (plantId) | Refused to delete, showed appropriate message | Message has been shown of failure request |
| TC21 | Log Out | Log out from mobile app | Logout click | Redirect to login page and token removed | Session ends, login required |

**CONCLUSION**

This project addresses a critical need in modern agriculture—real-time monitoring and intelligent management of plant health in nurseries using smart technologies. Throughout the study, several challenges were identified, including the difficulty in manually tracking environmental factors and responding quickly to plant needs such as watering and fertilization. The proposed system integrates IoT-based sensors with an AI-powered backend to predict plant health and generate actionable recommendations, all accessible via a mobile app. By implementing features like real-time data tracking, predictive analytics, role-based access, and automated notifications, the system offers a scalable and practical solution for nursery managers and farmers. The integration of Firebase Cloud Messaging, machine learning models, and secure communication protocols further enhances usability and reliability. Based on this analysis, it is recommended to pursue and further develop this project, as it provides substantial benefits in optimizing agricultural productivity, reducing human error, and promoting sustainable farming practices.