MINISTRY OF EDUCATION AND TRAINING

**VAN LANG UNIVERSITY**

**SCHOOL OF TECHNOLOGY**

A red and white logo

Description automatically generated

**CAPSTONE PROJECT – GROUP 1**

d&c

**SOFTWARE REQUIREMENT**

**SPECIFICATION**

Smart Home Garden Irrigation System

**Document version: 1.3**

# **REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Description | Authorize by |
| February 28, 2025 | 1.0 | Initial creation of the document | Nguyen Quang Tri  Nguyen Toan Thang |
| March 3, 2025 | 1.1 | Update document format | Nguyen Trung Hau |
| March 7, 2025 | 1.2 | Update content | Nguyen Trung Hau  Do Van Duong  Huynh Cong Tan  Chau Gia Kien |
| March 29, 2025 | 1.3 | Update content &  Revise format | Nguyen Toan Thang |

Table 1: Document Revision History

**TABLE OF CONTENTS**

[REVISION HISTORY 1](#_Toc1545685230)

[TABLE OF CONTENTS 2](#_Toc153869684)

[LIST OF TABLES 3](#_Toc835555622)

[LIST OF FIGURES 3](#_Toc995355839)

[I. INTRODUCTION 3](#_Toc1081772935)

[1. Purpose 4](#_Toc1245429039)

[2. Scope 4](#_Toc1179167089)

[3. Definition 4](#_Toc1621782868)

[II. SYSTEM CONTEXT DIAGRAM 5](#_Toc1918780372)

[1. Overview 6](#_Toc282459298)

[2. System Boundary 6](#_Toc1571871098)

[3. External Entities 6](#_Toc1619488890)

[4. Interactions 7](#_Toc1852715288)

[III. BUSINESS REQUIREMENT 8](#_Toc232658164)

[1. Business goals 9](#_Toc1140402286)

[2. Business constraints 10](#_Toc1686034276)

[3. Business criteria 12](#_Toc2003507556)

[IV. USER REQUIREMENT 13](#_Toc1489940768)

[1. Functional Requirements list 14](#_Toc1779383449)

[2. Use Cases Diagram 17](#_Toc1653317922)

[V. SYSTEM & SOFTWARE REQUIREMENT 49](#_Toc1466449230)

[1. Functional System Requirement 49](#_Toc570093581)

[2. Non-Functional System Requirements 51](#_Toc392034550)

[3. System Constraints 53](#_Toc1635783115)

[4. Backend Requirements 54](#_Toc385397829)

[5. Frontend Requirements 57](#_Toc1196788909)

[6. ESP32 Firmware Requirements 59](#_Toc403927122)

# **LIST OF TABLES**

[Table 1: Document Revision History 1](#_Toc198228164)

[Table 2: Functional Requirement List 17](#_Toc198228165)

[Table 3: Priority Table 18](#_Toc198228166)

[Table 4: Complexity table 18](#_Toc198228167)

[Table 5:Use case Description – Register account 24](#_Toc198228168)

[Table 6: Use case Description - Login account 25](#_Toc198228169)

[Table 7: Use case Description - Recover password 27](#_Toc198228170)

[Table 8:Use case Description - Change personal information 30](#_Toc198228171)

[Table 9: Use case Description – Search Personal Garden 32](#_Toc198228172)

[Table 10: Use case Description – Add Garden 33](#_Toc198228173)

[Table 11: Use case Description – Remove Garden 34](#_Toc198228174)

[Table 12: Use case Description – Update Garden Information 36](#_Toc198228175)

[Table 13: Use case Description – Create Watering Schedule 38](#_Toc198228176)

[Table 14: Use case Description – Update Watering Schedule 39](#_Toc198228177)

[Table 15: Use case Description – Delete Watering Schedule 41](#_Toc198228178)

[Table 16: Use case Description – Create Light Schedule 43](#_Toc198228179)

[Table 17: Use case Description – Update light Schedule 44](#_Toc198228180)

[Table 18: Use case Description – Delete Light Schedule 46](#_Toc198228181)

[Table 19: Use case Description – Update Moisture-based Watering Rule 48](#_Toc198228182)

[Table 20: Use case Description – Updated Light-based Lighting Rule 51](#_Toc198228183)

[Table 21: Use case Description – Generate Garden Statistics 51](#_Toc198228184)

[Table 26: Functional System Requirement 54](#_Toc198228185)

[Table 27: Non-Functional System Requirements 55](#_Toc198228186)

[Table 28: System Constraints 57](#_Toc198228187)

[Table 29: Backend Software Requirements 60](#_Toc198228188)

[Table 30: Frontend Software Requirements 62](#_Toc198228189)

[Table 31: ESP32 Firmware Software Requirements 64](#_Toc198228190)

# **LIST OF FIGURES**

[Figure 1: System Context Diagram - GreenSprout 7](#_Toc198228199)

[Figure 2: Notations 19](#_Toc198228200)

[Figure 3: Use case diagram – System overview 20](#_Toc198228201)

[Figure 4: Use case diagram – Login account 21](#_Toc198228202)

[Figure 5: Use case diagram – Change personal information 28](#_Toc198228203)

[Figure 6: Use case diagram – Modify Gardens 30](#_Toc198228204)

[Figure 7: Use case diagram – Modify Watering Schedules 36](#_Toc198228205)

[Figure 8: Use case diagram – Modify Light Schedules 41](#_Toc198228206)

[Figure 9: Use case diagram – Modify Moisture-Based Watering Modes 46](#_Toc198228207)

[Figure 10: Use case diagram – Modify Light-Based Lighting Modes 49](#_Toc198228208)

# **I. INTRODUCTION**

## **1. Purpose**

The purpose of this project is to provide a garden management system that can be managed on a mobile application and website. Responding to the needs of agriculture, farmers can manage their gardens remotely. Allowing users to control their gardens remotely, turn on and off systems, and control information in the garden

This document is used as a commitment between the customer and the development team about the output product and the rules of the development process.  
The expected audiences of this document are:

* Mrs. Thanh: the customer of the project
* Group 1: the developers of the project

## **2. Scope**

This project aims to control everything with one device, view information about temperature, humidity, turn on and off devices remotely, report and statistics about the amount of water in the garden. In addition, it is possible to schedule automatic turning on and off devices, easy to manage the garden.

## **3. Definition**

The garden management system serves two main groups of users: Garden managers and garden staff. Garden managers can perform actions such as turning irrigation systems on or off and scheduling automatic watering for the garden. Viewing reports, watering frequency statistics, and assigning permissions to system managers. On the other hand, garden managers can view information and can turn devices on and off if the manager grants permission.

- Customer:

• Users can create accounts to log in.

• They can view garden status information and reports, statistics on watering frequency.

• Schedule watering by day or week, assign permissions to member accounts.

- Member accounts:

* Create an account to log in.
* Generate garden information and its statistics.
* Turn on and off the irrigation, light, wind system and lights.

- Owner accounts:

* Update garden information
* Moderate users permission

# **II. SYSTEM CONTEXT DIAGRAM**

## **1. Overview**

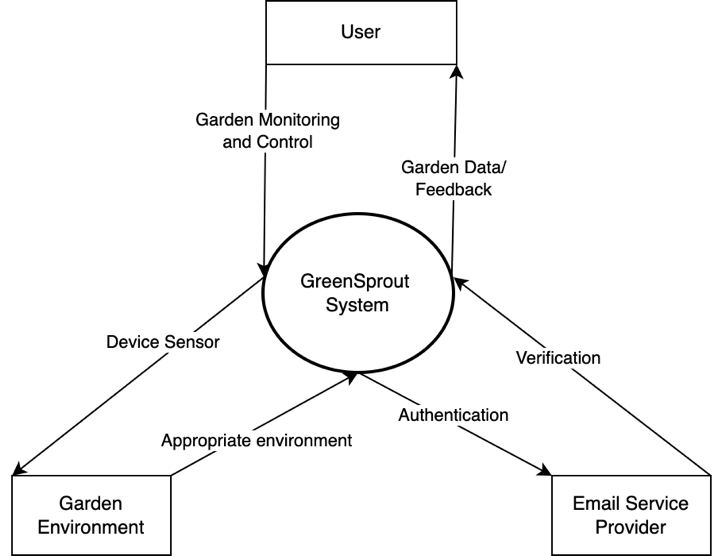
**The System Context Diagram for the GreenSprout project provides a high-level,   
visual representation of the GreenSprout System and its interactions with external entities. Designed as an IoT-based automated irrigation system, GreenSprout aims to optimize garden management by integrating hardware, software, and user interfaces to monitor and control garden environments efficiently.

Figure 1: System Context Diagram - GreenSprout

## **2. System Boundary**

The GreenSprout System encompasses all components developed and controlled by the project team, forming a cohesive unit for garden management:

* **Hardware**: ESP32 microcontrollers equipped with soil moisture and temperature sensors, along with relays to control irrigation, deployed across garden sections. Software on the ESP32 boards that handles sensor data collection and relay control, communicating with the backend over HTTP using Wi-Fi.
* **Backend**: A Node.js/Express server with a MongoDB database (hosted on MongoDB Atlas) that manages user accounts, device configurations, sensor data, and irrigation schedules via RESTful APIs.
* **Frontend**: React-based web and React Native mobile applications that provide user interfaces for monitoring, control, and data visualization.

These components are internal to the system, as they are designed, implemented, and maintained by the GreenSprout team.

## **3. External Entities**

The diagram identifies four external entities that interact with the GreenSprout System, representing the primary actors and systems outside its boundary:

1. **User:** Users responsible for overseeing garden operations, managing information garden, monitoring garden status, scheduling irrigation, setting threshold and assigning ownership to other user.
2. **Garden Environment**: The physical garden environment, including soil, plants, and irrigation setups, which the system monitors and controls through sensors and actuators.
3. **Email Service Provider**: A third-party service used by the system to send authentication codes for user registration and password recovery, ensuring secure access.

## **4. Interactions**

Each external entity interacts with the GreenSprout System through bidirectional flows, illustrating data or commands sent to and received from the system. The transition names are designed to be simple, noun-based, and strategically focused for high-level stakeholders:

* **Owner (Garden Managers)**:
  + **System Management and Control**: Commands sent from Owner to the system, such as setting schedules, configuring devices, or managing permissions, reflecting their strategic control.
  + **Garden & Member Data**: Data received by Owner from the system, including user permission & garden information.
* **Member (Garden Observator)**:
  + **Garden Monitoring and Control**: Requests sent from Member to the system, such as generating garden status, scheduling irrigation, or issuing manual control commands.
  + **Garden Data/ Feedback**: Information received by Member, including   
    real-time sensor data, statistical reports, and alerts (e.g., low moisture notifications), providing insights into garden performance.
* **Garden**:
  + **Appropriate environment**: Indicating whether the garden is suitable for installing device sensors based on environmental factors (e.g., soil type, space, or connectivity). This reflects the system’s initial evaluation to determine sensor deployment feasibility.
  + **Device Sensor**: Directing the next step of installing device sensors   
    (e.g., ESP32 units with sensors and relays) into the garden, enabling monitoring and control.
* **Email Service Provider**:
  + **Authentication**: Codes sent from the system to the Email Service Provider for user registration verification or password recovery, supporting secure access.
  + **Verification:** Confirmation received by the system from the Email Service Provider, indicating the success or failure of email delivery, ensuring reliable communication.

# **III. BUSINESS REQUIREMENT**

## **1. Business goals**

Business goals articulate the high-level objectives that GreenSprout aims to achieve, reflecting its purpose of transforming garden management into a smart, sustainable practice. These goals are derived from the ConOps’ executive summary and goals section, emphasizing market demand and user needs. Here, we detail four key goals, each with specific objectives and rationales:

### *1.1 Optimize Water Usage*

* **Objective**: Reduce water waste by enabling data-driven irrigation decisions, targeting a more efficiency improvement for prototype users compared to manual methods.
* **Rationale**: This goal addresses global water scarcity and aligns with market trends in precision agriculture, as noted in ConOps 1.2, where the smart irrigation market is projected to grow significantly. For GreenSprout, this means leveraging ESP32 sensors to collect real-time data, enabling users to water only when necessary, thus reducing waste.
* **Detail**: The target will be measured through a controlled test, comparing water usage in a GreenSprout-managed garden section against a manually irrigated section over two weeks, ensuring quantifiable results for increment validation.

### *1.2 Deliver an Affordable Prototype*

* **Objective**: Create a functional Minimum Viable Product (MVP) by May 2025 using low-cost components (e.g., ESP32 ~$5, free-tier MongoDB Atlas) within a $100 hardware budget.
* **Rationale**: This goal targets underserved small-scale gardeners and farmers, differentiating GreenSprout from high-cost competitors like Rain Bird, as highlighted in ConOps 1.2. It leverages affordable hardware and free-tier cloud services to ensure accessibility, fitting the project’s student-led, resource-constrained context.
* **Detail**: The budget includes ESP32 boards, sensors, and relays, with software development relying on open-source tools like GitHub and Node.js, ensuring cost-effectiveness while meeting the May 2025 deadline

### *1.3 Enhance Plant Care*

* **Objective**: Improve plant health through real-time monitoring and control, aiming for positive feedback from 80% of test users on usability and effectiveness.
* **Rationale**: This goal meets user needs for smarter gardening, supporting market trends in IoT and cloud integration, as per ConOps 1.2. It ensures GreenSprout provides actionable insights (e.g., moisture alerts, watering schedules) via React/React Native apps, enhancing user experience and plant health.
* **Detail**: Test users, including team members, mentor, and 5 external gardeners, will trial the prototype for a week, with surveys rated on a 5-point scale for ease of use and effectiveness, targeting ≥80% rating 4 or 5

### *1.4 Establish Scalability Potential*

* **Objective**: Design a modular system supporting up to 5 devices per user in the prototype, with a foundation for future expansion (e.g., 50+ devices).
* **Rationale**: This goal positions GreenSprout for growth from home gardens to small farms, capitalizing on IoT scalability trends, as noted in ConOps 6.2. It ensures the architecture can handle increased device loads, supporting future market expansion.
* **Detail**: The prototype will demonstrate modularity by managing 5 devices, each with independent sensor and relay functions, with the backend designed to scale via MongoDB and cloud services, tested incrementally in increment.

## **2. Business constraints**

Business constraints define the limitations that shape GreenSprout’s development, ensuring goals are achievable within the team’s resources and timeline. These are drawn from ConOps 3.3 and reflect practical realities as of March 03, 2025.

### *2.1 Limited Budget*

**Constraint**: Total hardware cost must not exceed $100, relying on low-cost, open-source tools (e.g., MongoDB Atlas free tier).

**Impact**: Excludes premium hardware (e.g., industrial sensors) or paid cloud services, focusing on ESP32 boards (~$5 each), basic sensors, and relays. This ensures affordability but limits advanced features like solar power, as noted in ConOps 3.1.

**Detail**: The budget covers 2 ESP32 units, 8 sensors (4 per device), and 6 relays, with software development using free-tier GitHub, aligning with the team’s cost-effective approach. (table)x1.5 price

### *2.2 Tight timeline*

* **Constraint**: Prototype must be completed by May 2025 (4-5 months from January 10, 2025), limiting scope to core features.
* **Impact**: Prioritizes MVP over advanced features like weather integration, requiring efficient Increnmental to meet the deadline. This timeline aligns with the team’s academic schedule and market entry goals.
* **Detail**: The project is planned for 6-8 two-week increment, starting February 15, 2025, with final testing and demo by late April, ensuring delivery by May for validation.

### *2.3 Small team size*

* **Constraint**: Six members with overlapping roles (developers, testers) restrict concurrent task capacity.
* **Impact**: To manage workload effectively, the project plan emphasizes structured task allocation and clearly defined phases, prioritizing core functionality such as device management and monitoring in the initial increments. This structured approach mitigates risks associated with limited parallel development.
* **Detail**: Roles include Nguyen Quang Tri as PM/Team leader, Nguyen Trung Hau, Do Van Duong, Chau Gia Kien, Nguyen Toan Thang for app, web development, Huynh Cong Tan for hardware lead, and others for testing, ensuring balanced coverage despite size constraints.

### *2.4 Dependency on free tools*

* **Constraint**: Development limited to free-tier services (e.g., MongoDB Atlas ~512 MB, GitHub) and team skills (e.g., Python/Node.js, React).
* **Impact**: Caps storage and performance, leveraging familiar tech for speed. This ensures rapid prototyping but may limit scalability testing in the MVP.
* **Detail**: The backend uses Node.js for REST APIs, React for UI, and MongoDB for data storage, all within free-tier limits, with GitHub hosting code and CI pipelines, aligning with team expertise.

## **3. Business criteria**

Success criteria provide measurable outcomes to evaluate whether GreenSprout meets its business objectives, bridging goals to deliverables. These criteria are derived from the ConOps 3.2 and are systematically defined for each increment, ensuring alignment with project objectives and requirements. Each criterion is validated upon completion of the respective increment to confirm conformance before proceeding to the next phase.

### *3.1 Functional prototype delivery*

* **Criterion**: Deploy a working prototype by May 2025 with 5 devices operational in a small garden test (2-3 sections), supporting all core features.
* **Measure**: No critical failures (defined as any core feature downtime exceeding 1 hour) during a 1-week test period, monitored via logs and user trials.
* **Detail**: The test will involve setting up 5 ESP32 devices in a controlled garden, ensuring user login, device management, real-time monitoring, and irrigation control work seamlessly, validated by the team and mentor by late April 2025.
* **Criterion**: Achieve positive feedback from ≥80% of test users (5 external testers) on ease of use and plant care benefits.
* **Measure**: Survey results post-prototype demo, with key questions rated 4 or 5 on a 5-point scale (e.g., “Easy to monitor moisture?” “Helped improve plant health?”).
* **Detail**: Testers, including local gardeners, will trial the system for a week, with surveys conducted via Google Forms, ensuring ≥4 ratings for usability and effectiveness, aligning with sprint feedback in May 2025.

### *3.2 Water efficiency*

* **Criterion**: Demonstrate a 20% reduction in water usage compared to manual irrigation in a controlled test.
* **Measure**: Conduct a 2-week test with two identical garden sections: one managed by GreenSprout (e.g., automated based on moisture data) and one with manual irrigation (e.g., fixed schedule). Calculate the percentage reduction in water usage, logged in MongoDB.
* **Detail**: The test will measure total liters used, aiming for GreenSprout using 0.8 liters/day vs. 1 liter/day manually, validated in Sprint 6.

### *3.3 On time and on budget delivery*

* **Criterion**: Complete the prototype by May 2025 within the $100 hardware budget.
* **Measure**: Project completion date (tracked via Trello) and total hardware expenditure (logged in team budget sheet), ensuring no overspend.
* **Detail**: Track expenses weekly, with final validation by PM Nguyen Quang Tri, ensuring delivery aligns with sprint milestones and budget limits

# **IV. USER REQUIREMENT**

## **1. Functional Requirements list**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Name** | **Description** | **Priority** | **Levels of complexity** |
| UC\_01 | Register Account | The system allows actors to register accounts to grant permission to interact with the system | 1 | 3 |
| UC\_02 | Login Account | The system allows actors to log into the system | 1 | 4 |
| UC\_03 | Recover Password | The system allows the actor to recover their password | 2 | 3 |
| UC\_04 | Update User Information | The system allows actors to change personal information | 2 | 3 |
| UC\_05 | Search Personal Garden | The system allows actors to search their own garden in their garden list | 1 | 3 |
| UC\_06 | Add Garden | The system allows actors to add new garden | 1 | 2 |
| UC\_07 | Remove Garden | The system allows actors to remove garden | 2 | 4 |
| UC\_08 | Update Garden Information | The system allows actors to change garden information | 2 | 3 |
| UC\_09 | Create Watering Schedule | The system allows actors to create a schedule to automatically turn the water pump on/off. | 1 | 2 |
| UC\_10 | Update Watering Schedule | The system allows actors to modify an existing water pump schedule. | 2 | 3 |
| UC\_11 | Delete Watering Schedule | The system allows actors to delete a scheduled water pump action. | 2 | 3 |
| UC\_12 | Create Lighting Schedule | The system allows actors to create a schedule to automatically turn garden lights on/off. | 1 | 2 |
| UC\_13 | Update Lighting Schedule | The system allows actors to modify an existing lighting schedule. | 1 | 3 |
| UC\_14 | Delete Lighting Schedule | The system allows actors to delete a scheduled lighting action. | 2 | 3 |
| UC\_15 | Update Watering Threshold | The system allows actors to modify the existing watering threshold. | 2 | 3 |
| UC\_16 | Update Lighting Threshold | The system allows actors to modify an existing lighting threshold. | 2 | 3 |
| UC\_17 | Generate Garden Statistics | The system allows users statistical reports on the garden's wastewater volume, by day, week, month. | 1 | 2 |
| UC\_18 | Control Users Permission | The system allows the Owner to view user profiles and ban/unban accounts. | 1 | 2 |
| UC\_19 | Update Garden Information | The system allows the Owner to update detailed information about any garden, including name and image. | 1 | 4 |
| UC\_20 | Transfer Ownership role | The garden owner transfer their ownership role for the specific member in the same garden | 1 | 4 |

Table 2: Functional Requirement List

|  |  |
| --- | --- |
| **Level** | **Description** |
| 1 | Must do |
| 2 | Should do |
| 3 | Depends on |
| 4 | Should not |

Table 3: Priority Table

|  |  |
| --- | --- |
| **Level** | **Description** |
| 1 | Extremely complex |
| 2 | Very complex |
| 3 | Normal |
| 4 | Easy |
| 5 | Extremely easy |

Table 4: Complexity table

## **2. Use Cases Diagram**

### ***2.1. Notations***

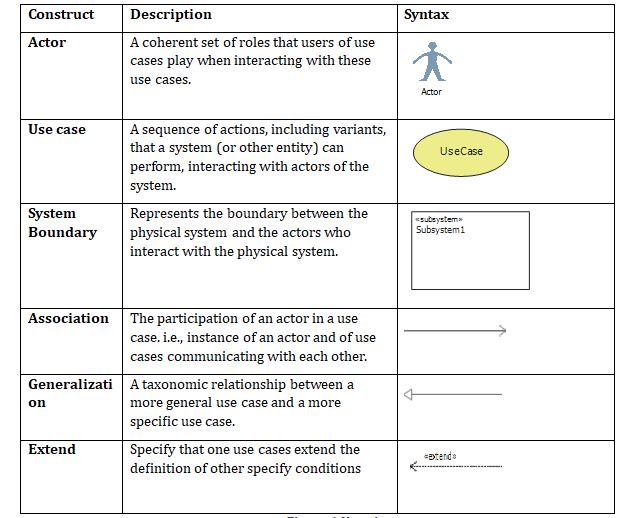


Figure 2: Notations

### ***2.2. System Overview***

A black background with a black square

AI-generated content may be incorrect.

Figure 3: Use case diagram – System overview

### ***2.3 Use case Authentication***

#### 2.3.1. Use Case Detail

A diagram of a user authentication

AI-generated content may be incorrect.

Figure 4: Use case diagram – Login account

#### 2.3.2. Use Case Description

##### *a) Use case Register account*

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_01** |
| ***Use Case Name:*** | Register account |
| ***Brief Description:*** | The actor performs account registration to gain access to the system's features. |
| ***Actor:*** | Customer |
| ***Pre-conditions:*** | The system must be operational and capable of processing account registration requests. |
| ***Post-conditions:*** | The actor has successfully created an account, and the data is stored in the system's database. |
| ***Main Success Flow:*** | 1. The actor selects "Register" on the system's homepage.  2. The system displays the registration screen.  3. The actor enters the required information, including:  - Full name  - Email address  - Password (and re-enter password)  4. The actor confirms "Register".  5. The system displays the email confirmation screen, allowing the actor to check and re-enter the email if incorrect.  6. The actor confirms by selecting the “Send Code".  7. The system displays the verification code input screen, where the actor receives a code via email.  8. The actor enters the verification code.  9. The actor confirms by selecting "Verify".  10. The system notifies the actor that the account has been successfully created. |
| ***Alternative Flows:*** | None |
| ***Exception Flows:*** | **E1: Existing Account**  In step 2 of the main flow: If the actor already has an account, they can select "Already have an account" on the functionality screen, redirecting to use case UC2 (Login Use Case).  **E2: Incomplete Information**  In step 4 of the main flow: If the actor provides incomplete information, the system displays an error message: "Please enter complete information" and prompts the actor to complete the missing fields before proceeding.  **E3: Duplicate Email**  In step 4 of the main flow: If the actor enters an email that already exists, the system displays an error message: "Account already exists" and prompts the actor to use a different email.  **E4: Invalid Email Format**  In step 4 of the main flow: If the actor enters an invalid email format, the system displays an error message: "Invalid email" and prompts the actor to enter a correctly formatted email address.  **E5: Mismatched Passwords**  In step 4 of the main flow: If the actor re-enters a password that does not match the initial password, the system displays an error message: "Passwords do not match" and prompts the actor to re-enter the password.  **E6: Incorrect Email Confirmation**  In step 5 of the main flow: If the actor enters an incorrect email during confirmation, the system displays an error message: "Invalid email" and allows the actor to re-enter the email.  **E7: Verification Code Not Received**  In step 7 of the main flow: If the actor does not receive the verification code, they can select "Resend Code" to request a new one.  **E8: Incorrect Verification Code**  In step 8 of the main flow: If the actor enters an incorrect verification code, the system displays an error message: "Invalid verification code" and allows the actor to re-enter the code. |

Table 5:Use case Description – Register account

##### *b) Use case Login account*

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_02** |
| ***Use Case Name:*** | Login account |
| ***Brief Description:*** | The actor logs into their account to access the system's features. |
| ***Actor:*** | Customer |
| ***Pre-conditions:*** | 1. The actor must have a valid account with a registered username and password.  2. The system must be operational and accessible.  3. The actor must have an active network connection to access the system's login interface. |
| ***Post-conditions:*** | The actor has successfully logged into their account and gained access to the system's features. |
| ***Main Success Flow:*** | 1. The system displays the login screen. 2. The actor enters their registered email and password. 3. The actor confirms by selecting "Login". 4. The system verifies the credentials. 5. The system grants access and redirects the actor to the main dashboard. |
| ***Alternative Flows:*** | None |
| ***Exception Flows:*** | **E1: Incorrect Credentials**  In step 4 of the main flow: If the actor enters an incorrect email or password, the system displays an error message: "Invalid email or password" and prompts the actor to re-enter valid credentials.  **E2: Unregistered Email**  In step 4 of the main flow: If the actor enters an email that is not registered, the system displays an error message: "Account not found" and prompts the actor to register an account.  **E3: Forgot Password**  In step 2 of the main flow: If the actor forgets their password, they can select "Forgot Password?" to initiate the password recovery process.  **E4: Account Locked**  In step 4 of the main flow: If the actor exceeds the allowed number of failed login attempts, the system temporarily locks the account and displays a message: "Account locked due to multiple failed login attempts. Try again later or reset your password."  **E5: Disabled Account**  In step 4 of the main flow: If the actor's account has been disabled, the system displays an error message: "Your account has been disabled. Please contact support for assistance." |

Table 6: Use case Description - Login account

##### *c) Use case Recover the password*

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_03** |
| ***Use Case Name:*** | Recover the password |
| ***Brief Description:*** | The actor requests a password recovery to regain access to their account. |
| ***Actor:*** | Customer |
| ***Pre-conditions:*** | 1. The actor has an existing registered account.  2. The system is operational and accessible.  3. The actor has access to the registered email associated with the account. |
| ***Post-conditions:*** | 1. The actor successfully resets their password, and the new password is stored in the system.  2. The actor can use the new password to log in to the system. |
| ***Main Success Flow:*** | 1. The actor selects "Forgot Password" on the login screen. 2. The system displays the password recovery screen. 3. The actor enters their registered email address. 4. The system verifies if the entered information is associated with an existing account. 5. If valid, the system sends a password reset code to the actor’s registered email. 6. The actor enters the verification code received. 7. The system verifies the code and, if correctly, allows the actor to create a new password. 8. The actor enters and confirms the new password. 9. The system updates the password and notifies the actor that the password has been successfully reset. |
| ***Alternative Flows:*** | None |
| ***Exception Flows:*** | **E1: Invalid Email or Phone Number**  In step 4 of the main flow: If the actor enters an unregistered email or phone number, the system displays an error message: "Account not found" and prompts the actor to re-enter valid credentials.  **E2: Verification Code Not Received**  In step 6 of the main flow: If the actor does not receive the verification code, they can select "Resend Code" to request a new one.  **E3: Incorrect Verification Code**  In step 7 of the main flow: If the actor enters an incorrect verification code, the system displays an error message: "Invalid verification code" and allows the actor to re-enter the correct code.  **E4: Mismatched New Password**  In step 8 of the main flow: If the actor’s new password and confirmation password do not match, the system displays an error message: "Passwords do not match" and prompts the actor to re-enter them. |

Table 7: Use case Description - Recover password

### ***2.4. Use case Change personal information***

#### 2.4.1. Use Case Detail

A diagram of a change

AI-generated content may be incorrect.

Figure 5: Use case diagram – Change personal information

#### 2.4.2. Use Case Description

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_04** |
| ***Use Case Name:*** | Change personal information |
| ***Brief Description:*** | The actor updates their personal information to keep their account details accurate. |
| ***Actor:*** | Customer |
| ***Pre-conditions:*** | 1. The actor has an existing account and is logged into the system.  2. The system is operational and accessible. |
| ***Post-conditions:*** | 1. The actor's updated information is successfully saved in the system.  2. The system reflects changes in the actor’s profile. |
| ***Main Success Flow:*** | 1. The actor selects **"Edit Profile"** in the user settings.  2. The system displays the **User Information Update** screen.  3. The actor updates their information, including:   * Full name * Email address * Date of birth * Address   4. The actor confirms the update by selecting **"Save Changes"**.  5. The system validates the input and updates the actor’s profile.  6. The system displays a confirmation message: **"Profile updated successfully."** |
| ***Alternative Flows:*** | None |
| ***Exception Flows:*** | **E1: Missing Required Information**  In Step 4 of the main flow: If the actor leaves any required field empty, the system displays an error message:" Please fill in all required fields." The actor must complete the missing fields before proceeding.  **E2: Invalid Email Format**  In Step 3 of the main flow: If the actor enters an incorrectly formatted email, the system displays an error message: "Invalid email format. Please enter a valid email address."  **E3: Duplicate Email**  In Step 3 of the main flow: If the entered email already exists in the system for another user, the system displays an error message: "This email is already in use. Please enter a different one."  **E4: System Error**  In Step 5 of the main flow: If the system encounters an error while updating the information, it displays an error message: "An error occurred while updating your profile. Please try again later." |

Table 8:Use case Description - Change personal information

### ***2.5. Use case Modify Gardens***

#### 2.5.1. Use Case Detail

A diagram of a garden

AI-generated content may be incorrect.

Figure 6: Use case diagram – Modify Gardens

#### 2.5.2. Use Case Description

##### *a) Search Personal Garden*

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_05** |
| ***Use Case Name:*** | Search Personal Garden |
| ***Brief Description:*** | The actor searches for a personal garden in the system to view or manage its details. |
| ***Actor:*** | Customer |
| ***Pre-conditions:*** | 1. The actor has a registered account and is logged into the system.  2. The system is operational and accessible.  3. The actor has at least one personal garden stored in the system. |
| ***Post-conditions:*** | The system successfully retrieves and displays the search results based on the actor’s input. |
| ***Main Success Flow:*** | 1. The actor selects the **"Search"** option in the personal garden section.  2. The system displays the **Search Personal Garden** interface with a search input field.  3. The actor enters search criteria (e.g., **garden name, location, plant type**).  4. The actor confirms the search by selecting **"Search"**.  5. The system processes the request and retrieves matching results.  6. The system displays the list of gardens that match the search criteria.  7. The actor selects a garden from the results to view its details. |
| ***Alternative Flows:*** | **A1: No Input Provided**  In **Step 4** of the main flow: If the actor attempts to search without entering any criteria, the system displays a message: "Please enter at least one search criterion." |
| ***Exception Flows:*** | **E1: No Matching Results**  In Step 6 of the main flow: If no gardens match the search criteria, the system displays a message: "No gardens found. Please refine your search."  **E2: Error System**  In Step 5 of the main flow: If the system encounters an error while processing the request, it displays an error message: "An error occurred while searching. Please try again later." |

Table 9: Use case Description – Search Personal Garden

##### *b) Add Garden*

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_06** |
| ***Use Case Name:*** | Add Garden |
| ***Brief Description:*** | The actor adds a new garden to the system to manage their plants and track its details. |
| ***Actor:*** | Customer |
| ***Pre-conditions:*** | 1. The actor has a registered account and is logged into the system.  2. The system is operational and accessible. |
| ***Post-conditions:*** | The system successfully stores the newly added garden in the database. |
| ***Main Success Flow:*** | 1. The actor selects "Add Garden" in the garden management section.  2. The system displays the **Add Garden** form.  3. The actor enters the required garden information, including:   * Garden name * Status * Id ESP   4. The actor confirms the entry by selecting **"Save"**.  5. The system validates the input and saves the garden details.  6. The system displays a confirmation message: "Garden added successfully." |
| ***Alternative Flows:*** | **A1: Cancel Addition**  In Step 4 of the main flow: If the actor cancels the operation, the system discards the entered data and returns to the garden management screen. |
| ***Exception Flows:*** | **E1: Incomplete Information**  In Step 4 of the main flow: If the actor provides incomplete information, the system displays an error message: "Please fill in all required fields."  **E2: Error System**  In Step 5 of the main flow: If the system fails to save the garden due to an internal error, it displays an error message: "An error occurred while adding the garden. Please try again later." |

Table 10: Use case Description – Add Garden

##### *c) Remove garden*

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_07** |
| ***Use Case Name:*** | Remove garden |
| ***Brief Description:*** | Allow a customer to remove a registered garden from the system. |
| ***Actor:*** | Customer |
| ***Pre-conditions:*** | 1. The actor has logged into the system.  2. The actor must have internet connectivity.  3. The actor has at least one garden registered in the system. |
| ***Post-conditions:*** | The selected garden is successfully removed from the customer’s account in the system. |
| ***Main Success Flows:*** | 1. The actor navigates to the "Manage Gardens" section of the system.  2. The system displays a list of gardens currently registered under the actor’s account.  3. The actor selects the garden they wish to remove.  4. The system prompts the actor to confirm the removal of the selected garden.  5. The actor confirms the removal.  6. The system updates the database and removes the garden from the actor’s account.  7. The system displays a confirmation message: "The garden has been successfully removed." |
| ***Alternative Flows:*** | None |
| ***Exception Flows:*** | **E1: Cancellation of Removal**  In step 5 of the main flow: If the customer cancels the removal (e.g., by clicking "No" or "Cancel"), the system returns to the "Manage Gardens" section without making any changes. |

Table 11: Use case Description – Remove Garden

##### *d) Update garden information*

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_08** |
| ***Use Case Name:*** | Update garden information |
| ***Brief Description:*** | Allows a customer to update the information of a registered garden in the system. |
| ***Actor:*** | Customer |
| ***Pre-conditions:*** | - The actor has logged into the system.  - The actor must have internet connectivity.  - The actor has at least one garden registered in the system. |
| ***Post-conditions:*** | - The selected garden’s information is successfully updated in the customer’s account in the system. |
| ***Main Success Flows:*** | 1. The actor navigates to the "Manage Gardens" section of the system. 2. The system displays a list of gardens currently registered under the actor’s account. 3. The actor selects the garden they wish to update. 4. The system displays the current information for the selected garden and provides editable fields. 5. The actor modifies the desired information and submits the changes. 6. The system prompts the actor to confirm the updates to the selected garden. 7. The actor confirms the updates. 8. The system updates the database with the new garden information. 9. The system displays a confirmation message: "The garden information has been successfully updated." |
| ***Alternative Flows:*** | None |
| ***Exception Flows:*** | **E1: Cancellation of Updates** • In step 7 of the main flow: If the customer cancels the update (e.g., by clicking "No" or "Cancel"), the system discards the changes and returns to the "Manage Gardens" section without updating the garden information.  **E2: Invalid Input** • In step 5 of the main flow: If the actor enters invalid data the system displays an error message and prompts the actor to correct the input before proceeding. |

Table 12: Use case Description – Update Garden Information

### ***2.6. Use case Modify Watering Schedules***

#### 2.6.1. Use Case Detail

A diagram of a water schedule

AI-generated content may be incorrect.

Figure 7: Use case diagram – Modify Watering Schedules

#### 2.6.2. Use Case Description

##### *a) Use case Add Watering Schedules*

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_09** |
| ***Use Case Name:*** | Create Watering Schedule |
| ***Brief Description:*** | Allows a customer to create a new watering schedule for a registered garden in the system. |
| ***Actor:*** | Customer |
| ***Pre-conditions:*** | - The actor has logged into the system.  - The actor must have internet connectivity.  - The actor has at least one garden registered in the system. |
| ***Post-conditions:*** | -A watering schedule is successfully created and associated with the selected garden in the customer’s account. |
| ***Main Success Flows:*** | 1. The actor navigates to the "Manage Gardens" section of the system. 2. The system displays a list of gardens currently registered under the actor’s account. 3. The actor selects the garden for which they wish to create a watering schedule. 4. The system presents an option to "Create Watering Schedule" for the selected garden. 5. The actor selects the "Create Watering Schedule" option. 6. The system displays a form with fields for watering schedule details (e.g., start time, duration, frequency, or amount of water). 7. The actor enters the desired watering schedule details and submits the form. 8. The system prompts the actor to confirm the creation of the watering schedule. 9. The actor confirms the creation. 10. The system saves the watering schedule to the database and associates it with the selected garden. 11. The system displays a confirmation message: "The watering schedule has been successfully created." |
| ***Alternative Flows:*** | None |
| ***Exception Flows:*** | **E1: Cancellation of Creation**  • In step 8 of the main flow: If the customer cancels the creation (e.g., by clicking "No" or "Cancel"), the system discards the entered details and returns to the "Manage Gardens" section without creating the watering schedule.  **E2: Invalid Schedule Input** • In step 7 of the main flow: If the actor enters invalid data (e.g., negative duration, negative water amount, start time in the past, or missing required fields), the system displays an error message (e.g., "Please enter valid watering schedule details") and prompts the actor to correct the input before proceeding. |

Table 13: Use case Description – Create Watering Schedule

##### *b) Use case Update watering schedule*

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_10** |
| ***Use Case Name:*** | Update watering schedule |
| ***Brief Description:*** | Allows a customer to update an existing watering schedule for a registered garden in the system. |
| ***Actor:*** | Customer |
| ***Pre-conditions:*** | - The actor has logged into the system.  - The actor must have internet connectivity.  - The actor has at least one garden registered in the system.  - The selected garden has at least one existing watering schedule. |
| ***Post-conditions:*** | - The selected watering schedule is successfully updated in the customer’s account in the system. |
| ***Main Success Flows:*** | 1. The actor navigates to the "Manage Gardens" section of the system. 2. The system displays a list of gardens currently registered under the actor’s account. 3. The actor selects the garden with the watering schedule they wish to update. 4. The system displays the existing watering schedules associated with the selected garden. 5. The actor selects the specific watering schedule they want to update. 6. The system displays the current details of the selected watering schedule in editable fields (e.g., start time, duration, frequency, or water volume). 7. The actor modifies the desired details and submits the changes. 8. The system prompts the actor to confirm the updates to the watering schedule. 9. The actor confirms the updates. 10. The system updates the database with the new watering schedule details. 11. The system displays a confirmation message: "The watering schedule has been successfully updated." |
| ***Alternative Flows:*** | None |
| ***Exception Flows:*** | **E1: Cancellation of Updates** • In step 9 of the main flow: If the customer cancels the update (e.g., by clicking "No" or "Cancel"), the system discards the changes and returns to the "Manage Gardens" section without updating the watering schedule.  **E2: Invalid Schedule Input** • In step 7 of the main flow: If the actor enters invalid data (e.g., negative duration, illogical frequency, or missing required fields), the system displays an error message (e.g., "Please enter valid watering schedule details") and prompts the actor to correct the input before proceeding. |

Table 14: Use case Description – Update Watering Schedule

##### *c) Use case Delete watering schedule*

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_11** |
| ***Use Case Name:*** | Delete watering schedule |
| ***Brief Description:*** | Allows a customer to delete an existing watering schedule for a registered garden in the system. |
| ***Actor:*** | Customer |
| ***Pre-conditions:*** | - The actor has logged into the system.  - The actor must have internet connectivity.  - The actor has at least one garden registered in the system.  - The selected garden has at least one existing watering schedule. |
| ***Post-conditions:*** | - The selected watering schedule is successfully removed from the customer’s account in the system. |
| ***Main Success Flows:*** | 1. The actor navigates to the "Manage Gardens" section of the system. 2. The system displays a list of gardens currently registered under the actor’s account. 3. The actor selects the garden with the watering schedule they wish to delete. 4. The system displays the existing watering schedules associated with the selected garden. 5. The actor selects the specific watering schedule they want to delete. 6. The system prompts the actor to confirm the deletion of the selected watering schedule. 7. The actor confirms the deletion. 8. The system updates the database and removes the watering schedule from the selected garden. 9. The system displays a confirmation message: "The watering schedule has been successfully deleted." |
| ***Alternative Flows:*** | None |
| ***Exception Flows:*** | **E1: Cancellation of Updates** • In step 7 of the main flow: If the customer cancels the deletion (e.g., by clicking "No" or "Cancel"), the system returns to the "Manage Gardens" section without making any changes. |

Table 15: Use case Description – Delete Watering Schedule

### ***2.7. Use case Modify Light Schedules***

#### 2.7.1. Use Case Detail

A diagram of light schedule

AI-generated content may be incorrect.

Figure 8: Use case diagram – Modify Light Schedules

#### 2.7.2. Use Case Description

##### *a) Use case Create light schedule*

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_12** |
| ***Use Case Name:*** | Create light schedule |
| ***Brief Description:*** | Allows a customer to create a new light schedule for a registered garden in the system. |
| ***Actor:*** | User |
| ***Pre-conditions:*** | - The actor has logged into the system.  - The actor must have internet connectivity.  - The actor has at least one garden registered in the system. |
| ***Post-conditions:*** | - A new light schedule is successfully created and associated with the selected garden in the customer’s account. |
| ***Main Success Flows:*** | 1. The actor navigates to the "Manage Gardens" section of the system. 2. The system displays a list of gardens currently registered under the actor’s account. 3. The actor selects the garden for which they wish to create a light schedule. 4. The system presents an option to "Create Light Schedule" for the selected garden. 5. The actor selects the "Create Light Schedule" option. 6. The system displays a form with fields for light schedule details (e.g., start time, duration or frequency). 7. The actor enters the desired light schedule details and submits the form. 8. The system prompts the actor to confirm the creation of the light schedule. 9. The actor confirms the creation. 10. The system saves the light schedule to the database and associates it with the selected garden. 11. The system displays a confirmation message: "The light schedule has been successfully created." |
| ***Alternative Flows:*** | None |
| ***Exception Flows:*** | **E1: Cancellation of Creation** • In step 9 of the main flow: If the customer cancels the creation (e.g., by clicking "No" or "Cancel"), the system discards the entered details and returns to the "Manage Gardens" section without creating the light schedule.  **E2: Invalid Schedule Input** • In step 7 of the main flow: If the actor enters invalid data (e.g., negative duration, start time in the past, or missing required fields), the system displays an error message (e.g., "Please enter valid light schedule details") and prompts the actor to correct the input before proceeding. |

Table 16: Use case Description – Create Light Schedule

##### *b) Use case Update light schedule*

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_13** |
| ***Use Case Name:*** | Update light schedule |
| ***Brief Description:*** | Allows a customer to update exist light schedule for a registered garden in the system. |
| ***Actor:*** | User |
| ***Pre-conditions:*** | - The actor has logged into the system.  - The actor must have internet connectivity.  - The actor has at least one garden registered in the system. |
| ***Post-conditions:*** | - A light schedule is successfully updated and associated with the selected garden in the customer’s account. |
| ***Main Success Flows:*** | 1. The actor navigates to the "Manage Gardens" section of the system. 2. The system displays a list of gardens currently registered under the actor’s account. 3. The actor selects the garden for which they wish to update the light schedule. 4. The system displays the existing light schedule details with an option to "Update Light Schedule." 5. The actor selects the "Update Light Schedule" option. 6. The system displays a form pre-filled with the current light schedule details (e.g., start time, duration, or frequency). 7. The actor modifies the desired light schedule details and submits the form. 8. The system prompts the actor to confirm the update of the light schedule. 9. The actor confirms the update. 10. The system saves the updated light schedule to the database and associates it with the selected garden. 11. The system displays a confirmation message: "The light schedule has been successfully updated." |
| ***Alternative Flows:*** | None |
| ***Exception Flows:*** | **E1: Cancellation of Update** • In step 8 of the main flow: If the customer cancels the update (e.g., by clicking "No" or "Cancel"), the system discards the modified details and returns to the "Manage Gardens" section without updating the light schedule.  **E2: Invalid Schedule Input** • In step 7 of the main flow: If the actor enters invalid data (e.g., negative duration, start time in the past, or missing required fields), the system displays an error message (e.g., "Please enter valid light schedule details") and prompts the actor to correct the input before proceeding. |

Table 17: Use case Description – Update light Schedule

##### *c) Use case Delete light schedule*

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_14** |
| ***Use Case Name:*** | Delete light schedule |
| ***Brief Description:*** | Allow a customer to delete an existing light schedule for a registered garden in the system. |
| ***Actor:*** | User |
| ***Pre-conditions:*** | - The actor has logged into the system.  - The actor must have internet connectivity.  - The actor has at least one garden registered in the system. |
| ***Post-conditions:*** | -The light schedule is successfully removed from the selected garden in the customer’s account. |
| ***Main Success Flows:*** | 1. The actor navigates to the "Manage Gardens" section of the system. 2. The system displays a list of gardens currently registered under the actor’s account. 3. The actor selects the garden for which they wish to delete the light schedule. 4. The system displays the existing light schedule details with an option to "Delete Light Schedule." 5. The actor selects the "Delete Light Schedule" option. 6. The system prompts the actor to confirm the deletion of the light schedule (e.g., "Are you sure you want to delete this light schedule?"). 7. The actor confirms the deletion. 8. The system removes the light schedule from the database and disassociates it from the selected garden. 9. The system displays a confirmation message: "The light schedule has been successfully deleted." |
| ***Alternative Flows:*** | None |
| ***Exception Flows:*** | **E1: Cancellation of Deletion**  • In step 6 of the main flow: If the customer cancels the deletion (e.g., by clicking "No" or "Cancel"), the system retains the light schedule and returns to the "Manage Gardens" section without deleting it. |

Table 18: Use case Description – Delete Light Schedule

### ***2.8. Use case Modify Moisture-Based Watering Modes***

#### 2.8.1. Use Case Detail

A diagram of a water mode

AI-generated content may be incorrect.

Figure 9: Use case diagram – Modify Moisture-Based Watering Modes

#### 2.8.2. Use Case Description

##### *a) Use case Update Moisture-Based Watering Rule*

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_15** |
| ***Use Case Name:*** | Update Moisture-based Watering Rule |
| ***Brief Description:*** | Allow a customer to update an existing moisture-based watering rule for a registered garden in the system. |
| ***Actor:*** | User |
| ***Pre-conditions:*** | -The actor has logged into the system.  -The actor must have internet connectivity.  -The actor has at least one garden registered in the system.  -The garden has a moisture sensor integrated and registered in the system.  -The selected garden has an existing moisture-based watering rule. |
| ***Post-conditions:*** | - The moisture-based watering rule is successfully updated and associated with the selected garden in the customer’s account. |
| ***Main Success Flows:*** | 1. The actor navigates to the "Manage Gardens" section of the system. 2. The system displays a list of gardens currently registered under the actor’s account. 3. The actor selects the garden for which they wish to update the moisture-based watering rule. 4. The system displays the existing moisture-based watering rule details with an option to "Update Moisture-Based Watering Rule." 5. The actor selects the "Update Moisture-Based Watering Rule" option. 6. The system displays a form pre-filled with the current rule details (e.g., moisture threshold percentage, watering duration, and amount of water). 7. The actor modifies the desired moisture-based watering rule details and submits the form. 8. The system prompts the actor to confirm the update of the watering rule. 9. The actor confirms the update. 10. The system saves the updated moisture-based watering rule to the database and associates it with the selected garden. 11. The system displays a confirmation message: "The moisture-based watering rule has been successfully updated." |
| ***Alternative Flows:*** | None |
| ***Exception Flows:*** | **E1: Cancellation of Update** • In step 8 of the main flow: If the customer cancels the update (e.g., by clicking "No" or "Cancel"), the system discards the modified details and returns to the "Manage Gardens" section without updating the watering rule.  **E2: Invalid Rule Input** • In step 7 of the main flow: If the actor enters invalid data (e.g., moisture threshold outside 0-100%, negative duration, negative water amount, or missing required fields), the system displays an error message (e.g., "Please enter valid moisture-based watering rule details") and prompts the actor to correct the input before proceeding. |

Table 19: Use case Description – Update Moisture-based Watering Rule

### ***2.9. Use case Modify Light-Based Lighting Modes***

#### 2.9.1. Use case Detail

A diagram of a light-based lighting mode

AI-generated content may be incorrect.

Figure 10: Use case diagram – Modify Light-Based Lighting Modes

#### 2.9.2. Use case description

##### *a) Use case Update Light-Based Lighting Rule*

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_16** |
| ***Use Case Name:*** | Update Light-Based Lighting Rule |
| ***Brief Description:*** | Changes the percentage of natural light to how low the light is before it is turned on, and when the natural light reaches a certain level the light is turned off |
| ***Actor:*** | User |
| ***Pre-conditions:*** | - The actor has logged into the system.  - The actor must have internet connectivity.  - The actor has at least one garden registered in the system.  - The light is successfully associated with the selected garden in the customer’s account. |
| ***Post-conditions:*** | The actor changes the brightness level successfully, and the data is stored in the system's database. |
| ***Main Success Flow:*** | 1. The actor navigates to the "Home Page" section of the system.  2. The system displays the Home Page screen.  3. The actor selects the garden for which they wish to change the brightness  4. The form display system has fields for detailed information about the garden.  5. The actor performs pressing on "change brightness level on/off".  6. The system displays a form with fields for details on what percentage of natural light the light is on, and what percentage of natural light is the auto light off.  7. The actor enters the desired light level change and submits the form.  8. The system prompts the actor to confirm the light level change.  9. The actor confirms the change.  10. The system saves the new light update to the database.  11. The system displays a confirmation message: "The light level change has been updated successfully". |
| ***Alternative Flows:*** | None |
| ***Exception Flows:*** | **E1: Cancel Change**  In step 9 of the main flow: If the customer cancels the change (e.g. by clicking "No" or "Cancel"), the system will cancel the entered details and return to the "Garden Details" section without the change level light. |

Table 20: Use case Description – Updated Light-based Lighting Rule

##### *b) Use case Generate Garden Statistics*

|  |  |
| --- | --- |
| ***Use Case ID:*** | **UC\_17** |
| ***Use Case Name:*** | Generate Garden Statistics |
| ***Brief Description:*** | Displays statistical reports on the garden's wastewater volume, by day, week, month |
| ***Actor:*** | User |
| ***Pre-conditions:*** | - The actor has logged into the system.  - The actor must have internet connectivity.  - The actor has at least one garden registered in the system.  - The watering system has been connected to the device. |
| ***Post-conditions:*** | The system displays clear statistical reports |
| ***Main Success Flow:*** | 1. The actor navigates to the "Statistical Reports" section of the system.  2. The system displays the Statistical Reports screen.  3. The actors select the garden they want to see the statistical reports  4. The system displays the Report Details screen according to the selected garden.  5. The actor performs the operation of selecting the date, month, and year to view the report.  6. The system displays the Report Details screen according to the selected date, month, and year. |
| ***Alternative Flows:*** | None |
| ***Exception Flows:*** | None |

Table 21: Use case Description – Generate Garden Statistics

# **V. SYSTEM & SOFTWARE REQUIREMENT**

To ensure clarity and manageability, we categorize the system requirements into functional requirements (what the system must do), non-functional requirements (how well it must perform), and system constraints (limitations and conditions). This approach aligns with the thinking trace’s recommendation to use multiple techniques for comprehensive coverage, facilitating iterative development in sprints.

## **1. Functional System Requirement**

Functional requirements specify the core capabilities of GreenSprout, ensuring it meets user and business needs. These are derived from the ConOps document’s functional requirement analysis and objectives, focusing on the system’s behavior as a whole

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Descriptions | Rationale | Verifiability | Detail |
| SR-F1: Data Collection | The system collect sensor data (moisture percentage and temperature in Celsius) every 5 minutes from up to 5 ESP32 devices, each equipped with moisture and temperature sensors. | Supports system for real-time environmental monitoring, ensuring data for irrigation decisions. | Test by deploying 5 devices, logging collection times, and verifying data against sensor readings, achieving 100% collection over a 24-hour period. | Data includes device ID, timestamp, moisture level (0-100%), and temperature (-40 to 125°C), stored in MongoDB |
| SR-F2: User Management | The system shall allow users to register and log in using email and password, supporting up to 10 concurrent users in the prototype. | Enabling personalized device management. | Test registration and login functionality, ensuring no duplicate emails and secure sessions, with 100% success rate. | Uses HTTP over Wi-Fi, storing user data in MongoDB with password hashing. |
| SR-F3: Device Management | The system shall enable users to add, remove, and manage up to 5 devices per user, associating each device with a garden section. | Ensuring scalability for small gardens. | Test adding/removing devices, verifying data association in MongoDB, with 100% accuracy. | Devices identified by unique IDs, managed via REST API calls. |
| SR-F4: Irrigation Control | The system shall allow users to control irrigation manually or schedule it (up to 5 schedules per device), with ESP32 relays executing commands within 1 second of receipt. | Enabling automated and on-demand watering for water efficiency. | Test manual toggle and schedule execution, measuring relay activation time, achieving 95% success rate. | Schedules stored in MongoDB, checked every 5 minutes by backend, sent via HTTP GET to ESP32. |
| SR-F5: Data Analytics | The system shall provide daily water usage and 7-day moisture trends for each device, accessible via the app. | Aiding users in optimizing water use. | Verify app displays correct charts, matching MongoDB data, with 100% accuracy over a week. | Uses Chart.js for rendering, pulling data via REST API. |

Table 26: Functional System Requirement

## **2. Non-Functional System Requirements**

Non-functional requirements specify the quality attributes and performance metrics, ensuring GreenSprout operates reliably and efficiently.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Descriptions | Rationale | Verifiability | Detail |
| SR-NF1: Uptime and Reliability | The system shall achieve 99% uptime for data collection and transmission over stable Wi-Fi (≥1 Mbps), measured over a 30-day period. | Ensures reliability, critical for continuous monitoring. | Monitor system logs, calculate uptime percentage, ensuring ≥99% over 30 days. | Handles Wi-Fi failures by retrying up to 3 times, logging errors. |
| SR-NF2: Performance | The system shall respond to user app requests (e.g., data fetch, control commands) within 5 seconds, 95% of the time, under load with 5 concurrent users. | Ensures usability, meeting user expectations for responsiveness. | Conduct load testing, measuring response times, achieving ≤5 sec for 95% of requests. | Includes API calls and UI updates, tested on Android 8+ and iOS 12+ devices. |
| SR-NF3: Security | All data transmissions between system components (ESP32, backend, apps) shall be encrypted using HTTPS/TLS, ensuring data integrity and confidentiality. | Protects user data, meeting security needs for IoT systems. | Verify encryption via network traffic analysis, ensuring HTTPS usage. | Includes user credentials, sensor data, and control commands. |
| SR-NF4: Scalability | The system shall support up to 5 devices per user without performance degradation, with potential for future expansion to 50+ devices. | Ensures scalability, aligning with market growth. | Test with 5 devices, measuring response times and data sync rates, ensuring no degradation. | Backend uses MongoDB Atlas free tier (~512 MB), designed for modular growth. |

Table 27: Non-Functional System Requirements

## **3. System Constraints**

System constraints define the limitations and conditions under which GreenSprout must operate, derived from ConOps assumptions and dependencies, ensuring feasibility within resources.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Descriptions | Rationale | Impact | Verifiability |
| SR-C1: Network Dependency | The system shall rely on Wi-Fi for all communications, with no offline mode in the MVP, requiring stable connections (≥1 Mbps). | Reflects hardware constraints, aligning with IoT setup. | Limits operation to Wi-Fi environments, affecting reliability in remote areas. | Test under various Wi-Fi conditions, ensuring functionality with ≥1 Mbps. |
| SR-C2: Hardware Limitations | The system will operate within ESP32 hardware constraints, including ~520 KB SRAM and 4 MB flash, and consume <100mW average power. | Ensures feasibility, fitting low-cost IoT design. | Limits firmware complexity, prioritizing efficiency over advanced features. | Measure memory usage and power consumption, confirming within limits. |
| SR-C3: Cloud Storage Limits | The system shall use free-tier MongoDB Atlas (~512 MB) for data storage, limiting database size and performance. | Ensuring cost-effectiveness. | Caps data retention, requiring periodic cleanup for long-term use. | Monitor storage usage, ensuring within 512 MB. |
| SR-C4: Timeline and Scope | The system shall deliver a functional prototype by May 2025, focusing on core features (e.g., monitoring, control) within 6-8 sprints. | prioritizing MVP over advanced features. | Excludes weather integration, advanced analytics in MVP, per ConOps 3.1. | Track sprint completion, ensuring prototype ready by May. |

Table 28: System Constraints

GreenSprout aims to optimize garden irrigation through real-time sensor data and automated control, targeting small-scale gardeners and farmers. The system integrates ESP32 microcontrollers for data collection, a Node.js/MongoDB backend for processing, and React/React Native apps for user interaction. The software requirements must support the system’s functional and non-functional needs, derived from business goals (e.g., water efficiency), user needs (e.g., remote monitoring), and system specifications (e.g., 5-minute data collection). The presentation emphasizes that software requirements are specific to the developer’s perspective, ensuring alignment with system architecture and user needs, and must be correct, complete, clear, concise, consistent, relevant, feasible, and verifiable.

To ensure clarity and manageability, we categorize the software requirements into backend, frontend, and ESP32 firmware components, reflecting GreenSprout’s distributed nature. This approach aligns with the presentation’s recommendation to use multiple techniques and models for comprehensive coverage, facilitating iterative development in sprints.

## **4. Backend Requirements**

The backend, built with Node.js and MongoDB, handles data processing, user management, and device control.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Requirement Types | ID | Descriptions | Rationale | Verifiability |
| Functional  Requirements | **SWR1** | The DeviceService shall handle HTTP POST requests from ESP32 boards, processing up to 5 concurrent requests, each containing sensor data (moisture and temperature), and store the data in MongoDB with 100% success rate | Supports system requirement collect sensor data every 5 minutes, ensuring reliable data sync. | Test by simulating 5 ESP32 boards sending data, checking MongoDB for 100% storage. |
| **SWR2** | The backend shall provide a REST API for user registration, login, device management, and data retrieval, with endpoints including /api/user/signup, /api/user/login, /api/devices, and /api/devices/:id. | Meets user registration/login and device management, enabling user interaction. | Verify API responses for each endpoint, ensuring correct data handling. |
| **SWR3** | The backend shall have a scheduler that checks irrigation schedules in MongoDB every 5 minutes and sends HTTP GET requests to the appropriate ESP32 boards to activate or deactivate the relay based on the schedules. | Supports irrigation control, ensuring automated watering. | Monitor scheduler logs and ESP32 relay state changes every 5 minutes. |
| Non-Functional Requirements | **SWR4** | The backend API shall respond to requests within 2 seconds for 95% of all requests, measured under load testing with 5 concurrent users | Ensures performance, critical for user experience. | Conduct load tests, measuring response times. |
| **SWR5** | All communication between ESP32 boards and the backend shall be over HTTPS to ensure data integrity and confidentiality. | Addresses security, protecting user data. | Verify HTTPS usage via network traffic analysis. |
| **SWR6** | User passwords shall be stored in MongoDB using a secure hashing algorithm | Enhances security, meeting user trust needs. | Review database schema and test password hashing. |
| Interface Requirements | **SWR7** | The backend shall expose REST API endpoints as specified, with JSON format for requests and responses (e.g., POST /api/sensor-data accepts “device\_id": "1", "moisture": 45}). | Ensures interoperability with apps and devices. | Test API calls with Postman, checking JSON parsing. |
| Software Constraints | **SWR8** | The backend shall be developed using Node.js 16+ and Express.js, leveraging free-tier MongoDB Atlas (~512 MB). | Aligns with project constraints (BC4, free tools), ensuring cost-effectiveness | Confirm environment setup and database limits |

Table 29: Backend Software Requirements

## **5. Frontend Requirements**

The frontend, using React for web and React Native for mobile, provides user interaction for monitoring and control.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Requirement Types | ID | Descriptions | Rationale | Verifiability |
| Functional  Requirements | **SWR9** | The React/React Native apps shall poll the backend API every 60 seconds to fetch the latest sensor data and update the user interface accordingly, displaying moisture and temperature readings. | Supports real-time data, ensuring users see current conditions. | Test by simulating 5 ESP32 boards sending data, checking MongoDB for 100% storage. |
| **SWR10** | The apps shall allow users to manually toggle irrigation on or off for any device, sending the corresponding command to the backend via API calls (e.g., POST /api/settings/:device\_id). | Meets manual control, enhancing user control | Test toggle functionality, verifying backend updates. |
| **SWR11** | The apps shall display daily water usage and 7-day moisture trends for each device, using data retrieved from the backend via the API, rendered using Chart.js. | Supports data analytics, aiding water efficiency. | Verify chart rendering and data accuracy against MongoDB. |
| Non-Functional Requirements | **SWR12** | The apps shall load within 5 seconds on Android 8+ and iOS 12+ devices with basic hardware, ensuring usability. | Meets user constraints, ensuring accessibility. | Test load times on target devices. |
| **SWR13** | The apps shall handle up to 5 concurrent users polling the API without significant lag, supporting small-scale testing. | Ensures scalability for prototype. | Conduct user load tests, measuring response times. |
| Interface Requirements | **SWR14** | The apps shall consume REST API endpoints at api.greensprout.com (or localhost for prototypes), handling JSON responses for data display. | Ensures integration with backend. | Test API calls, verifying JSON parsing. |
| Software Constraints | **SWR15** | The frontend shall be developed using React 18+ for web and React Native for mobile, compatible with free-tier GitHub hosting. | Ensuring rapid development | Confirm environment setup and deployment. |

Table 30: Frontend Software Requirements

## **6. ESP32 Firmware Requirements**

The ESP32 firmware, written in C++ using the Arduino framework, manages sensor data collection and relay control.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Requirement Types | ID | Descriptions | Rationale | Verifiability |
| Functional  Requirements | **SWR16** | The ESP32 firmware shall collect sensor data (moisture and temperature) every 5 minutes using attached sensors | ensuring data collection for monitoring. | Log collection times, verify data against sensors. |
| **SWR17** | The ESP32 firmware shall send the collected sensor data to the backend via HTTP POST to the /api/sensor-data endpoint, including device ID and readings. | Ensure data sync. | Monitor backend logs for received data. |
| **SWR18** | The ESP32 firmware shall accept HTTP GET requests from the backend to activate or deactivate the relay for irrigation control, executing the command within seconds. | Enabling automated watering. | Test relay activation/deactivation, measuring response time. |
| Non-Functional Requirements | **SWR19** | The ESP32 firmware shall consume less than 100mW average power, measured under normal operation. | Ensuring low-power IoT operation. | Use power meter to measure consumption. |
| **SWR20** | The firmware should handle Wi-Fi connection failures by retrying HTTP requests up to 3 times before logging into an error. | Ensures reliability | Simulate Wi-Fi drop, verify retry behavior |
| Interface Requirements | **SWR21** | The ESP32 firmware shall communicate with the backend via HTTP over Wi-Fi, using JSON format for data (e.g.,"device\_id": "1", "moisture": 45}) | Ensures interoperability. | Test data format and transmission. |
| Software Constraints | **SWR15** | The ESP32 firmware shall be developed using C++ with the Arduino framework, compatible with ESP32’s ~520 KB SRAM and 4 MB flash. | Aligns with hardware constraints, ensuring feasibility. | Confirm memory usage within limits. |

Table 31: ESP32 Firmware Software Requirements