

SipSip - Requirements Analysis & Design

Table of Contents

1.0 Introduction.....	2
1.1 Purpose.....	2
1.2 Scope.....	2
2.0 System Overview.....	4
2.1 Project Perspective.....	4
2.2 System Context.....	4
2.3 General Constraints.....	5
2.4 Assumptions & Dependencies.....	5
3.0 Functional Requirements.....	6
3.1 Features.....	7
3.1.1 Feature #1 - Plant Profile Management.....	7
3.1.2 Feature #2 - Plant Identification.....	8
3.1.3 Feature #3 - Plant Care Reminders.....	9
3.1.3 Feature #4 - Shared Household Plant Care.....	10
3.2 Use Cases.....	12
3.2.1 Use Case #1 - Creating Plant Profile.....	12
3.2.2 Use Case #2 - Identifying a Plant.....	13
3.2.3 Use Case #3 - Completing a Shared Task.....	14
3.3 Data Modelling & Analysis.....	15
3.3.1 Normalized Data Model Diagram.....	15
3.3.2 Activity Diagrams.....	16
3.3.3 Sequence Diagrams.....	18
3.3.4 UML Class Diagram.....	20
3.4 Process Modelling.....	21
4.0 Non-Functional Requirements.....	21
5.0 Logical Database Requirements.....	22
6.0 Approval.....	22

1.0 Introduction

The SipSip project is an Android mobile application designed to assist casual plant owners and gardening enthusiasts in managing their plant care routines effectively. The application aims to address common plant care challenges such as missed watering schedules, improper environmental assessment, and insufficient plant-specific care guidance. Developed by a team at George Brown College, SipSip targets busy individuals and households with shared plant care responsibilities, providing personalized reminders, plant identification, and educational resources. The project follows an agile methodology within a two-semester academic timeline under the guidance of instructor Anjana Shah.

1.1 Purpose

The purpose of this document is to:

- Define the functional requirements of the SipSip system
- Specify non-functional performance requirements
- Model data structures, workflows, and interactions (design artifacts)
- **Align** development with the approved **Project Vision** and **MVP scope**

Intended Audience:

- Development team (Jayden Lewis, Uzma Shaikh, Aidan Repchik)
- Instructor Anjana Shah
- Future reviewers and stakeholders

1.2 Scope

In Scope

- Android application (target API 35, backward compatible to API level 24, Android 7.1).
- Core MVP features include:
 - Plant profile CRUD (S01).

- Plant identification via camera/upload with Google Cloud Vision API (S02).
- Dashboard featuring summary cards and urgent care tasks (S03).
- Playful, reliable reminders with snooze and mark-done options (S04).
- Differentiation between indoor and outdoor plants with specific care logic including weather alerts (S05).
- Beginner and advanced plant care tips displayed daily and on demand (S06).
- Customizable notification schedules including do-not-disturb periods (S07).
- Shared household plant care allowing task assignment and activities log (S09).
- Activity logging for tracking plant care history (S10).
- Offline functionality with local data persistence and notification capabilities (S16).
- Local data storage managed with Room Persistence Library.
- Optional future data synchronization across household users using Supabase.
- Use of free-tier third-party APIs for plant identification, weather, and notifications.

Out of Scope

- iOS, web, or desktop application versions.
- Advanced AI diagnostics or chatbot functionality.
- Subscription or paid model features.
- Real-time cloud synchronization in the MVP version.
- Integration with smart devices or third-party hardware.

Objectives & Goals

- Deliver a functional minimum viable product (MVP) by March 27, 2026.
- Support beginners and busy users with an intuitive, playful user experience.
- Enable offline-first plant care with core features working without internet access.

- Establish a foundation for future scalability, such as API integrations and business partnerships.

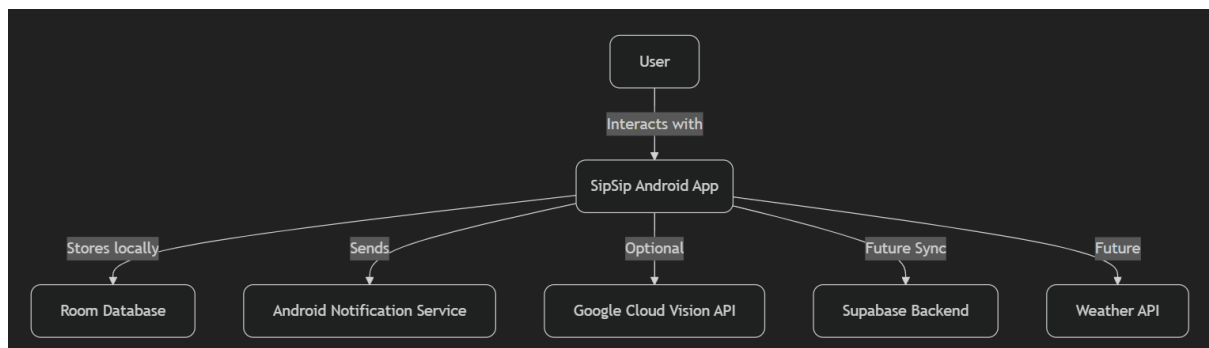
2.0 System Overview

2.1 Project Perspective

SipSip is a new, self-contained mobile application built from scratch using Kotlin and Android Jetpack. It is not a replacement for existing systems but draws inspiration from apps like Planta and Vera, while differentiating through:

- Zero-cost entry (no signup, no paywall)
- Playful, non-intrusive reminders
- Full offline capability
- Household sharing (future sync)

2.2 System Context



The app operates **entirely offline** for core functions. Internet is required only for:

- **Plant identification** (Google Cloud Vision)
- **Future household sync** (Supabase)
- **Optional weather-based tips**

2.3 General Constraints

Constraint	Impact
Android-only	All UI, testing, and deployment must target the Android SDK
Target API 35	Must compile against Android 15 and support devices with API level 24 and above
Two-semester timeline	Fixed project deadline on March 27, 2026; requires strict feature prioritization and scheduling
Zero budget	Only free or low-cost development tools and APIs can be used
Material Design	UI must follow Google's Material Design guidelines for consistency and usability
Team size = 3	Tasks must be balanced to avoid single points of failure and prevent overloading any team member

2.4 Assumptions & Dependencies

#	Assumption / Dependency	Reference
1	Users have Android device (API 24+) with camera & notifications	Vision 2.3
2	Team has continuous access to Android Studio, Kotlin, GitHub	Plan 5
3	Instructor feedback within 1 week of submission	Charter 6
4	Google Cloud Vision API key remains valid and rate-limited	Backlog T07
5	Supabase free tier sufficient for household sync (1GB, 5GB transfer)	Draft 5.0
6	Two-week sprint cycle maintained	Plan 8
7	Room DB handles all local persistence	Features 3.1.1

3.0 Functional Requirements

ID	Requirement	Linked Story	Priority
FR-01	The system shall support CRUD operations on plant profiles with required fields: Name, Species, Environment (Indoor/Outdoor)	S01	High
FR-02	The system shall allow photo upload/capture and auto-fill via Google Cloud Vision API	S02	High
FR-03	The system shall display a dashboard with urgent tasks and plant summaries	S03	High
FR-04	The system shall schedule and send playful, non-intrusive reminders with snooze/mark-done	S04, S13	High
FR-05	The system shall differentiate indoor vs. outdoor plants and adjust logic (e.g., weather alerts)	S05	Medium
FR-06	The system shall display daily rotating care tips and advanced lessons	S06	Medium
FR-07	The system shall support customizable notification schedules and do-not-disturb	S07	Low
FR-08	The system shall enable shared household plant care with task assignment and activity log	S09, S10	High
FR-09	The system shall support offline access to all core features	S16	High
FR-10	The system shall allow future sync via Supabase for household groups	S09	Low (Post-MVP)

3.1 Features

3.1.1 Feature #1 - Plant Profile Management

Description:

Users will be able to manage plant profiles based on their personal or shared plants using CRUD-based operations in a user-friendly way.

Input:

- Plant Name
- Species
- Outdoor or Indoor
- Plant Photo
- Last Watered
- Location
- Acquisition date
- Plant Collection

Processing:

- Create:
 1. Frontend validates required fields are filled
 2. Backend validates the plant model
 3. Data is saved to the local database
 4. Confirmation message is shown after successful creation
- View All:
 1. Backend retrieves all plant profiles from the local database
 2. Data is sent to the frontend via a list of plant profiles
- View Singular:
 1. Backend retrieves plant data by ID from the local database
 2. Object data is sent to the frontend for display
- Update:
 1. Frontend pre-fills current plant details in an editable form
 2. Backend validates and updates modified data in the local database
- Delete:
 1. System displays a confirmation prompt to prevent accidental deletion
 2. Upon confirmation, the backend removes the plant record from the local database

Output:

- Create: Confirmation message on the status of plant creation
- View All: All plant profiles are displayed in a list or grid format with thumbnails
- View Singular: Data is displayed in a detailed manner, with all information available
- Update: UI updates to reflect the update and displays a status message
- Delete: UI updates to reflect the deletion and displays a status message

3.1.2 Feature #2 - Plant Identification**Description:**

Plant identification via camera or photo upload. Optional automatic completion of plant information for a plant profile from the identified plant.

Input:

- Image

Processing:

1. Using a secure API key, a request for identification is sent to Google Cloud Vision API with the image attached
2. Google Cloud Vision analyzes the image and returns the most likely plant species and related metadata
3. Backend processes the response and extracts relevant information
4. If the user chooses, the system auto-fills the corresponding fields in the plant profile creation form

Output:

- Display of the identified plant species and confidence percentage
- Option to automatically fill plant profile fields with the identified information
- An error message is displayed if identification fails or the plant cannot be recognized
- Confirmation message if auto-completion is successfully applied

3.1.3 Feature #3 - Plant Care Reminders

Description:

Allows users to set and manage reminders for plant care activities such as watering, fertilizing, pruning, or repotting. Notifications help users maintain their plants' health by alerting them when specific actions are due.

Input:

- "Do-not-disturb" Times
- Completion Confirmation

Processing:

1. Background scheduler or system alarm service monitors stored reminders
2. At specified times, the service cross-references plant profile data from the local database
3. Reminder timing is adjusted based on species type, environmental conditions, calculated watering interval, and do-not-disturb times
4. The system triggers notifications for upcoming plant care activities such as watering, fertilizing, or pruning

Output

- Notification alert at the scheduled time with quick actions (Mark Done, Snooze, Open Plant)
- Confirmation messages when reminders are created, updated, completed, or deleted
- Updated reminders list showing upcoming and completed tasks

3.1.3 Feature #4 - Shared Household Plant Care

Description:

Enables collaborative plant care management between multiple household/workplace members. Users can assign, share, and track plant care responsibilities within a shared collection. Each user can view, complete, or reassign tasks.

Input:

- Household Name
- Member Email or Invite Code
- Shared Plant Selection
- Task Details
- Member Tasks
- Notification Preferences

Processing:

- Create Household:
 1. Owner creates a household group and generates an invite code or link.
 2. Backend stores the household data and sets the owner as the administrator.
- Add Members:
 1. Invited users join using the invite link or code.
 2. Backend verifies the code and adds the member to the household database.
- Share Plants:
 1. Selected plant profiles are linked to the household so all members can view and interact with them.
- Assign Tasks:
 1. Members can assign plant care tasks to themselves or others in the household.
 2. Backend saves the task details to the local database and syncs them to the cloud for shared access.
- Update & Track:
 1. Task completion status is updated by any authorized member.

2. Backend syncs updates across devices so all members see real-time changes.
- Notifications:
 1. The system sends notifications to members when a task is assigned, updated, or completed.

Output:

- Confirmation message after household creation or member addition
- Shared plant list displaying plants linked to the household
- Task list showing assigned activities and responsible members
- Notifications for task assignments and completion updates
- Visual status indicators for completed, upcoming, or overdue tasks

3.2 Use Cases

3.2.1 Use Case #1 - Creating Plant Profile

Trigger	The user taps on the <i>Create Plant Profile</i> icon on the <i>Dashboard</i> screen.
---------	---

Precondition	The user must be logged in and on the <i>Dashboard</i> screen.
Basic Path	1. The user is presented with the <i>Create Plant Profile</i> screen with input fields where the user can enter their plant's information. 2. After entering information into all required fields, the user may tap the create button, which will save all data to the local database on the device. 3. After tapping the create button, the user will be prompted to add the newly created <i>Plant Profile</i> to a <i>Collection</i> of plants. 4. The user will be returned to the <i>Dashboard</i> with their newly created <i>Plant Profile</i> visible in its designated <i>Collection</i>.
Alternative Path	In step 1, the user can optionally upload an image of the plant they are making the <i>Plant Profile</i> for, which will prompt the app to access the device's image storage to allow for easy selection.
Postcondition	The <i>Plant Profile</i> has been added to the local database.
Exception Path	The user may cancel this operation at any time and be returned to the <i>Dashboard</i> .
Other	None

3.2.2 Use Case #2 - Identifying a Plant

Trigger	The user taps on the "Identify Plant" icon on the Dashboard or the Add Plant screen.
---------	--

Precondition	The user must be logged in and have granted camera or photo library permissions.
Basic Path	1. The user is prompted to either take a photo of a plant using the device's camera or select an existing image from their gallery. 2. Once an image is selected, the app uploads the image to the Google Cloud Vision API for analysis. 3. The backend receives the identification result, processes it, and displays the most likely plant species with a confidence percentage. 4. The user is shown an option to automatically fill plant information fields (name, species, category) using the identification data. 5. The user can then choose to create a Plant Profile directly using this pre-filled data.
Alternative Path	In step 2, if the device is offline, the app stores the image and queues the identification request to be processed once a connection is available.
Postcondition	The plant identification result is displayed, and the user may optionally save it as a new Plant Profile.
Exception Path	If the Google Cloud Vision API cannot identify the plant, the user is notified and returned to the Identify Plant screen to try again.
Other	None

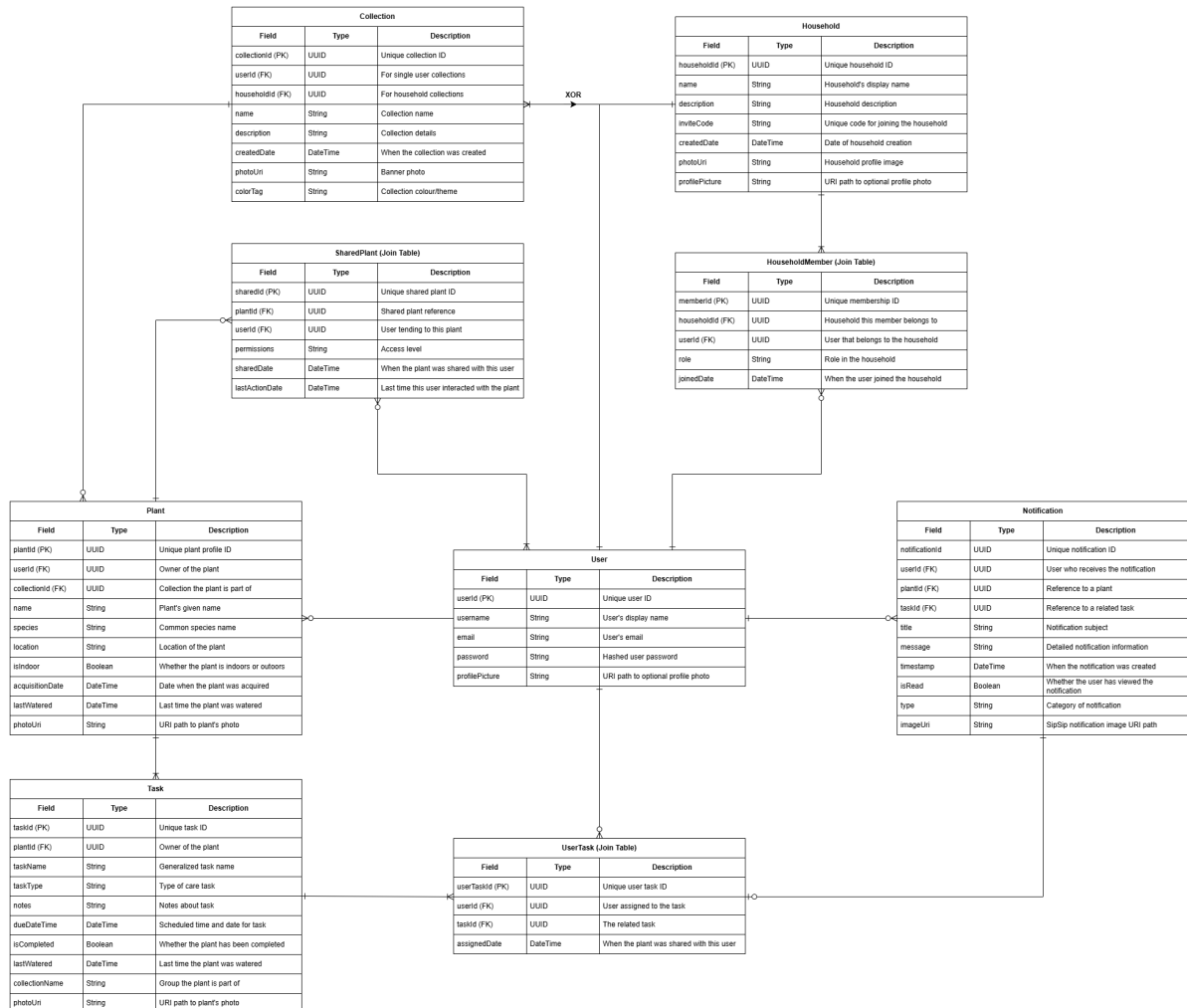
3.2.3 Use Case #3 - Completing a Shared Task

Trigger	A household member taps on a reminder notification or opens a task from the Shared Household screen.
----------------	--

Precondition	The user must be logged in and a member of a household group with assigned shared tasks.
Basic Path	1. The user navigates to the Shared Household screen and selects a plant care task assigned to them. 2. The app displays task details, including the plant name, activity type, and due date. 3. The user taps the "Mark as Done" button to complete the task. 4. The task's status is updated in the local database and synced to the cloud. 5. Other household members receive a notification that the task has been completed.
Alternative Path	In step 3, the user can reassign the task to another household member instead of completing it.
Postcondition	The shared task's status is updated to "Completed," and all members see the updated status on their devices.
Exception Path	If the sync fails due to no internet connection, the task completion is queued and synced automatically once the device reconnects.
Other	None

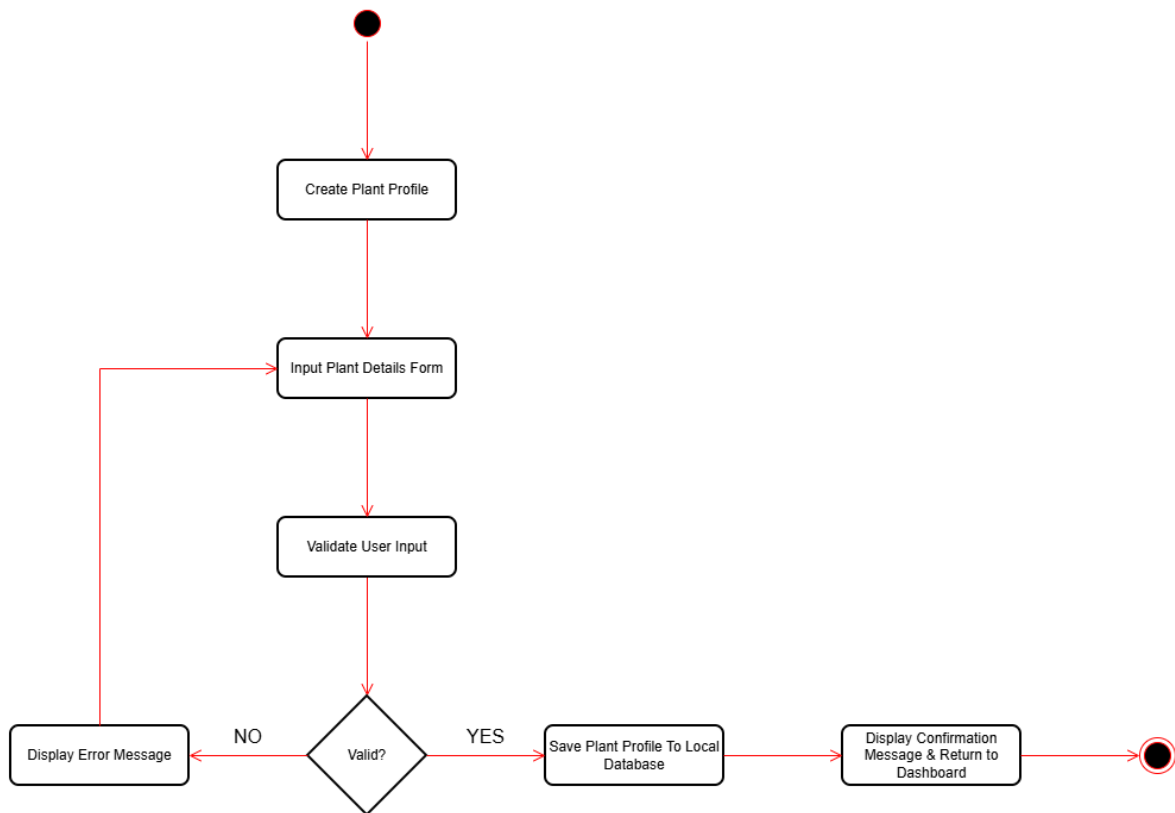
3.3 Data Modelling & Analysis

3.3.1 Normalized Data Model Diagram

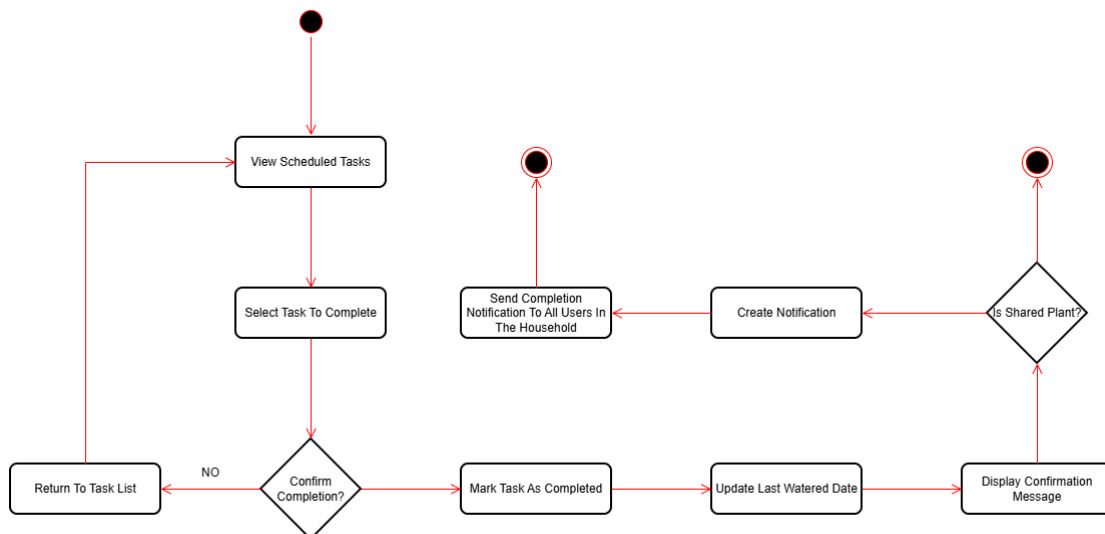


3.3.2 Activity Diagrams

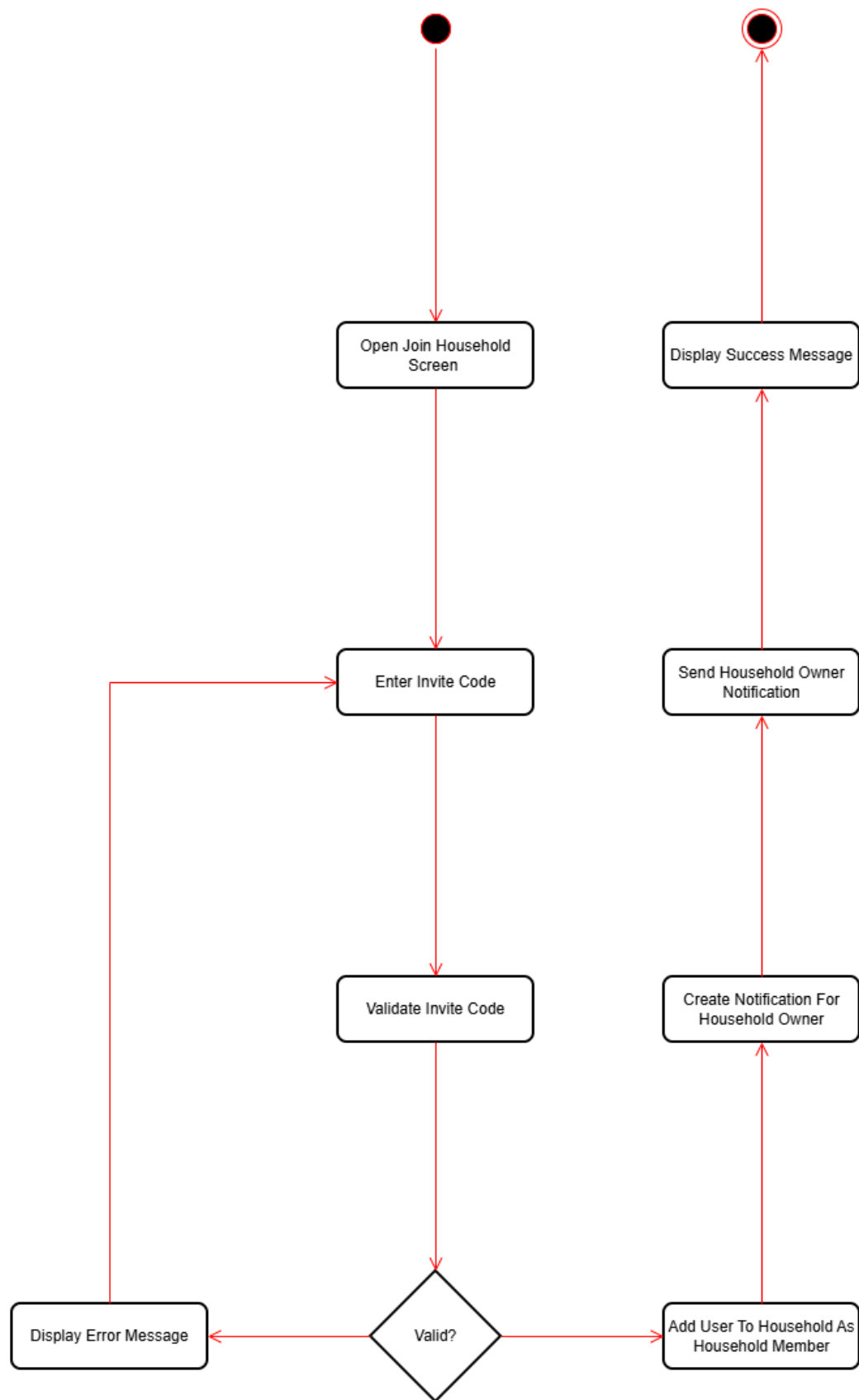
Create Plant Profile



Complete Plant Care Task

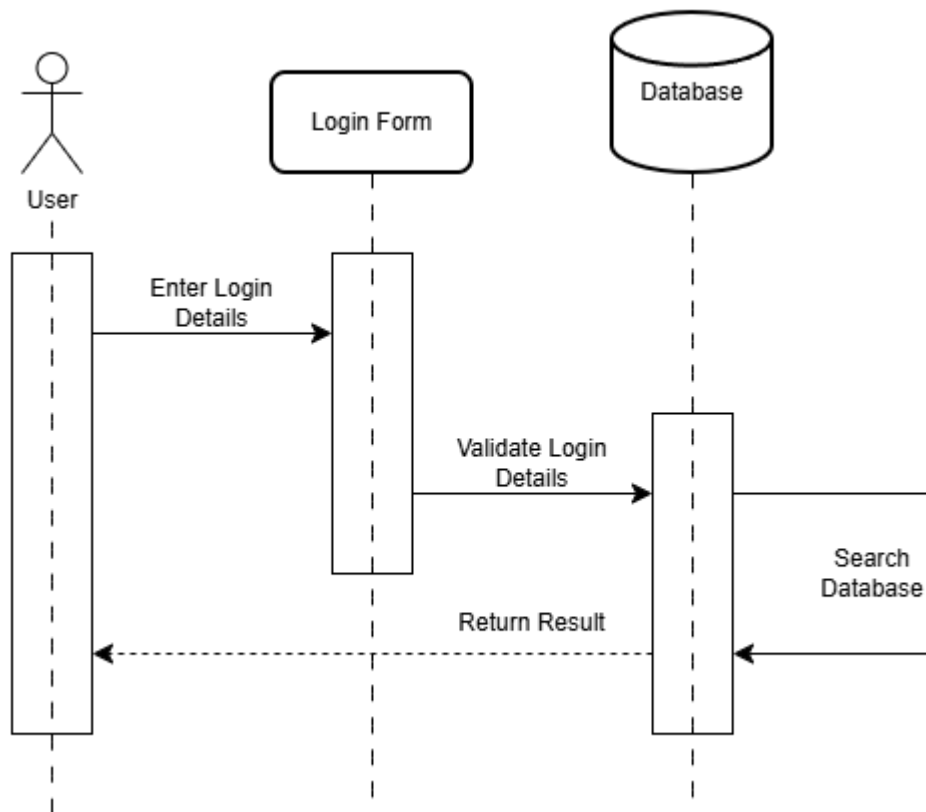


Join Household

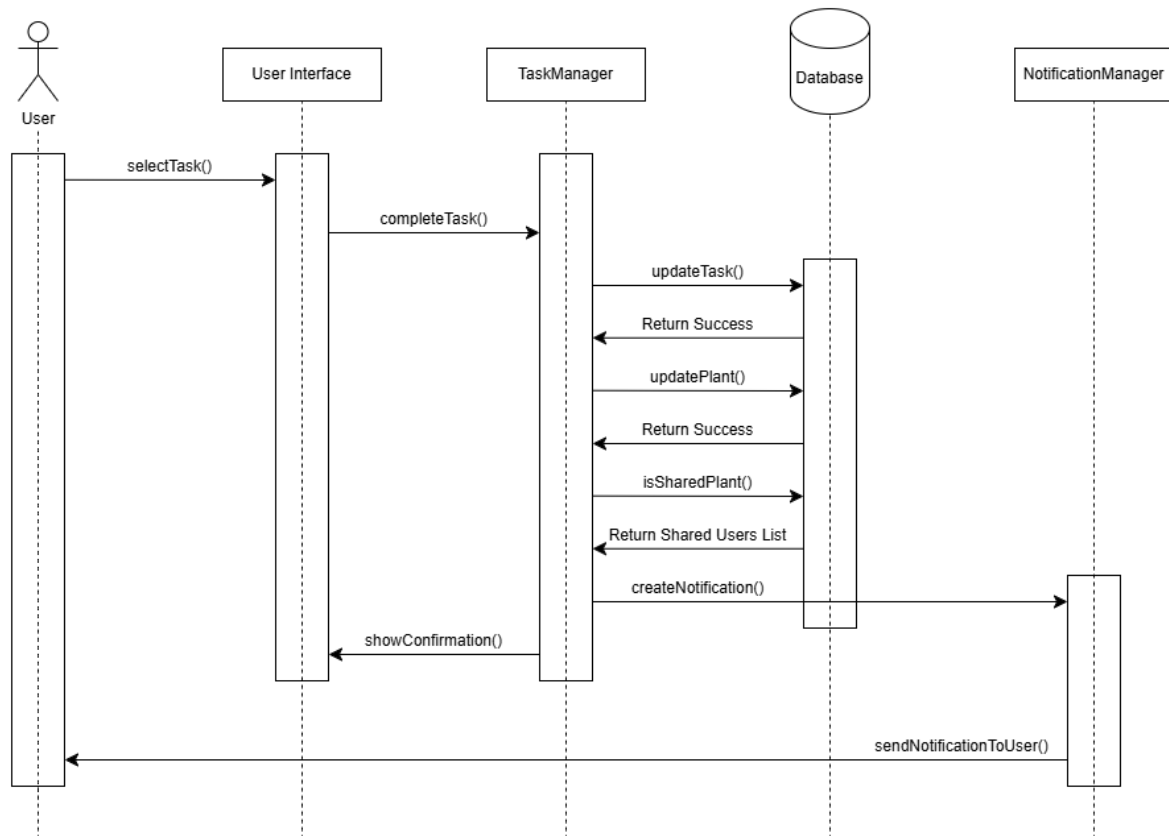


3.3.3 Sequence Diagrams

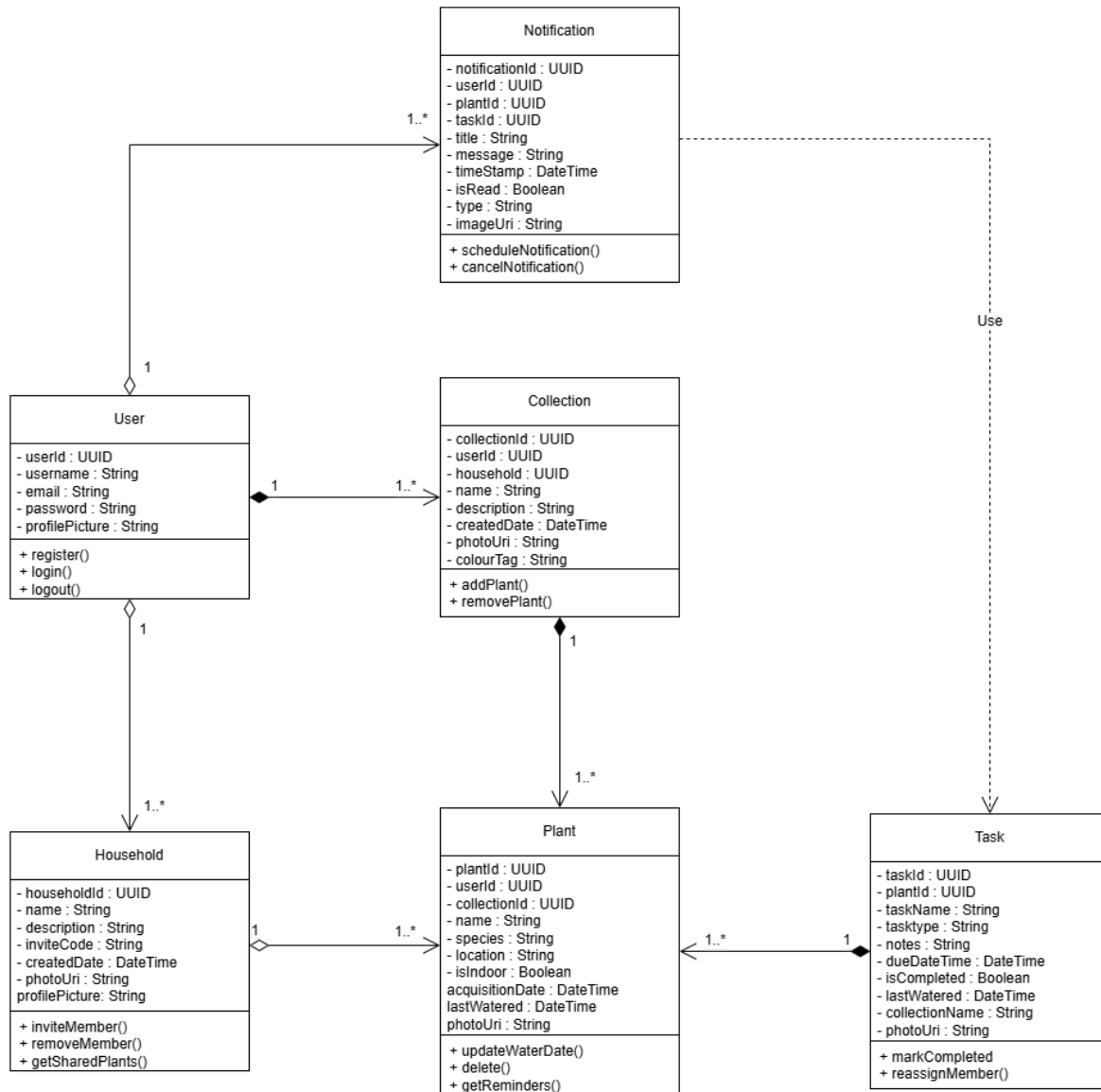
User Login



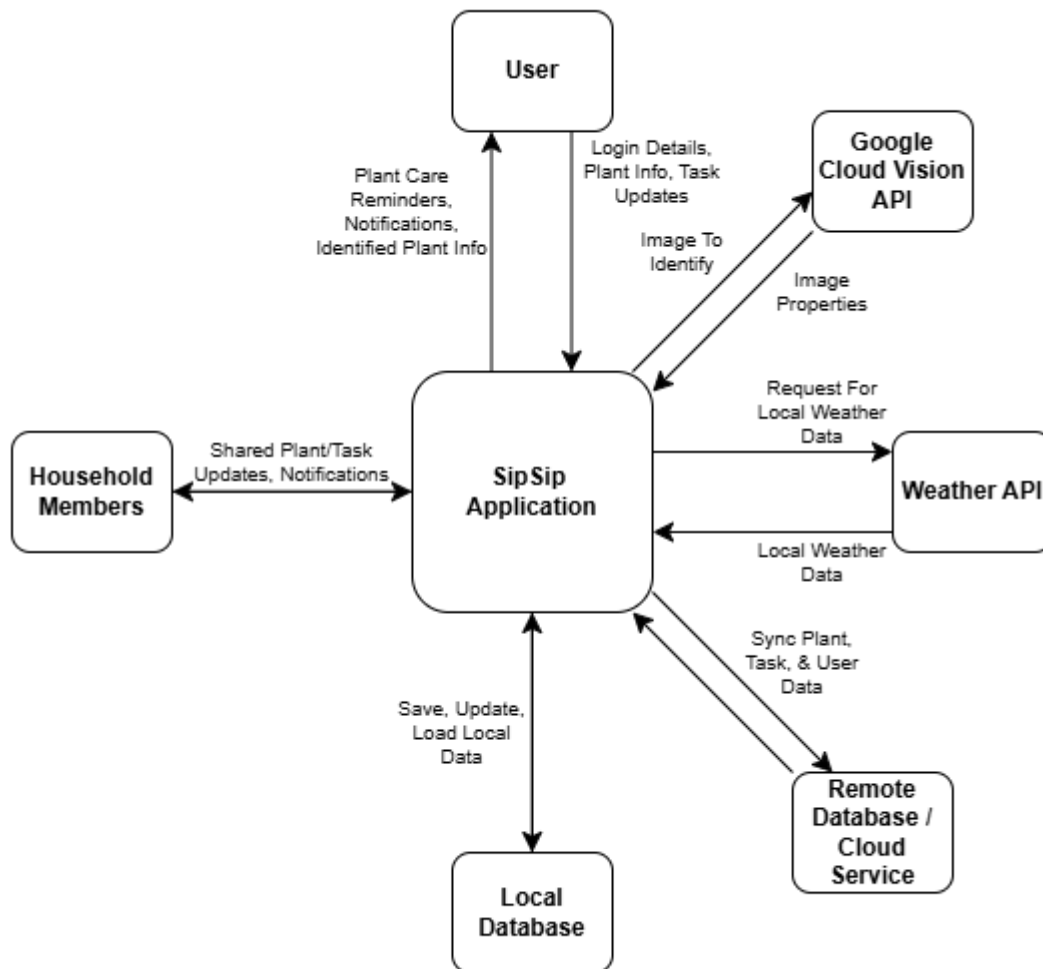
Complete Plant Care Task



3.3.4 UML Class Diagram



3.4 Process Modelling



4.0 Non-Functional Requirements

Performance:

The app should respond to user actions within 2 seconds under normal conditions. Data synchronization between the device and cloud will only occur when updating data between devices on the same household group

Reliability:

System errors should occur in less than 0.1% of operations. In case of app crashes or connection loss, the app should recover the last saved state within 10 seconds.

Availability:

Most data and functionality will be stored and processed locally on the user's device, allowing the app to remain usable without an internet connection. The server will be used primarily for syncing data between grouped devices, ensuring that household group updates occur automatically once connectivity is restored.

Maintainability:

The system should be modular and well-documented to allow easy updates and bug fixes. Critical issues should be resolved within 72 hours of detection.

Portability:

The app should run on Android devices and adapt smoothly to different screen sizes and orientations.

5.0 Logical Database Requirements

Sipsip will use a room data base for managing local storage as well as a Supabase remote database to handle the data syncing between users in the same household. The data storage on the remote database will be 1gb with a limit of 5 gb of data transfer per month. The database will store user profile data and uploaded images. To ensure that the capacity of the database is not exceeded the image data will be erased after syncing with the other members of a household.

6.0 Approval

Sprint 3 Role	Name	Signature	Date
Project Developer	Aidan Repchik	Aidan Repchik	10/11/2025
Project Manager	Jayden Lewis	Jayden Lewis	10/11/2025
Project Engineer	Uzma Shaikh	Uzma Shaikh	10/11/2025