A picture containing plant, green, leaf, vegetable

Description automatically generated

**Garden App**

**Report 3 (Part 1)**

**CSCI 441**

**GitHub URL:** <https://github.com/Kirapants07/CSCI441_VA_Group1_Spring2023_GardenApp>

**April 12, 2023**

**Group Kira: Daniel Dietrich, Gavin Cyr, Kira Luethe, Richard Williams, Todd Wood**

# Individual Contributions Breakdown

## Responsibility Matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Task** | **Daniel** | **Gavin** | **Kira** | **Richard** | **Todd** |
| Work Assignment, Glossary of Terms | 20% | 20% | 20% | 20% | 20% |
| Functional Requirement Specification, Use Cases (3a,3b,3c.i 3c.ii, 3c.iii, 3c.iv, 3d), Interaction Diagrams, Classes: (User & UserAuth), Test Case Designs 1, 2, 3, 4, 5 | 100% |  |  |  |  |
| Business Goals Diagram, System Architecture and Project Size Estimation, System Operation Contracts, Classes: Zipcode, PlantingZone, Test Coverage, Integration Testing |  | 100% |  |  |  |
| History of Work, User Interface Graphics, User Interface Specification, Project Management and Team Leader |  |  | 100% |  |  |
| Classes: All API Classes and diagram, Section 2 Data Model and Persistent Data Storage, Decomposition into Sub-Problems, Database Development, References |  |  |  | 100% |  |
| Conceptual Model (parts i,ii,iii, and iv), Classes: Plants, PlantCalcs, Traceability Matrix, Class diagram, Test Cases 6,7,8,9,10, Customer Problem Statement, Plan of Work, System Architecture |  |  |  |  | 100% |

# Table of Contents

[Individual Contributions Breakdown 2](#_Toc132801406)

[Responsibility Matrix 2](#_Toc132801407)

[Table of Contents 4](#_Toc132801408)

[Summary of Changes 7](#_Toc132801409)

[1. Customer Statement of Requirements 7](#_Toc132801410)

[Decomposition into Sub-Problems 10](#_Toc132801411)

[2. Glossary of Terms 10](#_Toc132801412)

[3. System Requirements 12](#_Toc132801413)

[a. Business Goals 12](#_Toc132801414)

[b. Enumerated Functional Requirements 12](#_Toc132801415)

[c. Enumerated Nonfunctional Requirements 12](#_Toc132801416)

[d. User Interface Requirements 13](#_Toc132801417)

[4. Functional Requirements Specification 13](#_Toc132801418)

[a. Stakeholders 13](#_Toc132801419)

[b. Actors and Goals 13](#_Toc132801420)

[c. Use Cases 14](#_Toc132801421)

[i. Casual Description 14](#_Toc132801422)

[ii. Use Case Diagram 15](#_Toc132801423)

[iii. Traceability Matrix 15](#_Toc132801424)

[iv. Fully-Dressed Description 16](#_Toc132801425)

[d. System Sequence Diagrams 22](#_Toc132801426)

[5. Effort Estimation Using Use Case Points 30](#_Toc132801427)

[a. User Effort Estimation 31](#_Toc132801428)

[UC-1: Create a User Account 31](#_Toc132801429)

[UC-3: Filter by Zip Code 32](#_Toc132801430)

[UC-4: Search by Plant 32](#_Toc132801431)

[UC-5: Enter Planting Date 32](#_Toc132801432)

[6. Domain Analysis 33](#_Toc132801433)

[a. Conceptual Model 33](#_Toc132801434)

[I. CONCEPT DEFINITIONS 34](#_Toc132801435)

[ii. ASSOCIATION DEFINITIONS 35](#_Toc132801436)

[iii. ATTRIBUTE DEFINITIONS 36](#_Toc132801437)

[iv. TRACEABILITY MATRIX 37](#_Toc132801438)

[b. System Operation Contracts 37](#_Toc132801439)

[7. Interaction Diagrams 40](#_Toc132801440)

[a. Interaction Diagrams 40](#_Toc132801441)

[8. Class Diagrams and Interface Specification 48](#_Toc132801442)

[a. Class Diagram 48](#_Toc132801443)

[b. Data Types and Operation Signatures 48](#_Toc132801444)

[c. Traceability Matrix 62](#_Toc132801445)

[d. Design Patterns 64](#_Toc132801446)

[e. OCL Contracts 65](#_Toc132801447)

[9. System Architecture and System Design 67](#_Toc132801448)

[a. Identifying Subsystems 67](#_Toc132801449)

[b. Architecture Styles 68](#_Toc132801450)

[c. Mapping Subsystems to Hardware 69](#_Toc132801451)

[d. Connectors and Network Protocols 69](#_Toc132801452)

[e. Global Control Flow 69](#_Toc132801453)

[i. Execution Orderness 69](#_Toc132801454)

[ii. Time Dependency 70](#_Toc132801455)

[f. Hardware Requirements 70](#_Toc132801456)

[10. Algorithms and Data Structures 70](#_Toc132801457)

[a. Algorithms 70](#_Toc132801458)

[b. Data Structures 70](#_Toc132801459)

[c. Concurrency: 70](#_Toc132801460)

[11. User Interface Design and Implementation 70](#_Toc132801461)

[a. Initial User Interface Requirements Diagrams 71](#_Toc132801462)

[b. User Interface Specification 77](#_Toc132801463)

[i. Preliminary/Final Designs 77](#_Toc132801464)

[UC-1: Create a User Account 77](#_Toc132801465)

[UC-3: Filter by Zip Code 81](#_Toc132801466)

[UC-4: Search by Plant 85](#_Toc132801467)

[UC-5: Enter Planting Date 89](#_Toc132801468)

[12. Design of Tests 92](#_Toc132801469)

[a. Test cases 92](#_Toc132801470)

[b. Test Coverage 98](#_Toc132801471)

[c. Integration Testing 98](#_Toc132801472)

[13. History of Work, Current Status, and Future Work 99](#_Toc132801473)

[a. Merging the Contributions from Individual Team Members 99](#_Toc132801474)

[b. History of Work 100](#_Toc132801475)

[c. Breakdown of Responsibilities 101](#_Toc132801476)

[14. References 101](#_Toc132801477)

# Summary of Changes

- Functional Requirements: REQ-2 and REQ-14 removed from requirements list.

- Functional Requirement Traceability Matrix: REQ-2, REQ-14, UC-2 and UC-6 removed as these features will not be implemented prior to Demo 2

- Function Requirements REQ-9, REQ- 10, REQ- 11 removed. They will be attempted, however due to the current data they may not be able to be properly implemented. Implementation may require refactoring of the data stored in the database and time does not allow for that level of change. Thus, they may not end up in the final Demo 2.

- Interaction Diagrams: UC-1, UC-3, UC-4, UC-5. UC-6, UC-7, UC-8, UC-9 and UC-10 revised to better reflect their current implementation. UC-2 and UC-6 removed as these features will not be implemented prior to Demo 2.

-Added final app design screenshots to User Interface Specification

# Customer Statement of Requirements

Gardening is a universal way to grow your own food, and live healthier and more sustainably. For many it is a hobby, and for some it is a way to grow much needed food. Regardless of the reason behind it, many Americans have some form of garden, whether it be a small box on a porch or a large field in the backyard.

Growing a garden can be a particularly challenging proposition. It is easy to throw seeds in the soil and hope they come up, but planning a garden and tracking the plants is a more challenging prospect. Many people are busy and do not have time to document everything they do, or research what type of seeds a person should use for their region or figure out when they should plant those seeds. After planting, people often do not know when they can expect those plants to start to grow, or when they will be ready to harvest.

People try various methods to keep track of the garden. They will write it all on a piece of paper, or they will place markers in the garden. Some will just not bother and will hope for the best. None of these solutions give the gardener the tools to plant the best possible garden. A better solution would be an application that can help the average gardener research their plants, plan their garden and gain information without a significant amount of writing, internet sleuthing and data collection.

How does the average person determine what seeds they will plant? Many times, the person will go to the store and look through racks of seeds and try to figure out what might work. They then plant them in hopes that the plants will work well. Or they go to the local nursery hoping to find something that will work well in their region. Ideally an application would allow the person to sit down and enter their zip code and from their computer or mobile device browser be able to search or look through a list of plants that work well in their region.

But the person does not want a general database they have to refer to repeatedly. Nobody wants to have to re-look up their plants or their garden over and over. Instead, an application should allow the person to create an account and save the gardeners' plant choices so that they can easily retrieve the information about those plants by simply logging back in whenever they desire.

What about after the person goes to the store and finds the seeds or plants, they want to put in the ground? When do they put them in the ground? Do they plant them right away or do they wait? The package often tells them when it is best to plant, but that is often a challenge for the average gardener to keep track of that information. The application should have information about plants already loaded and allow the gardener to see when the plants they have chosen are best planted. It should tell them the time of year and roughly when the best time is to put them in the ground.

Once the gardener plants their garden they often forget what they did and when. The gardeners will find themselves going out periodically to check and see if their plants have started to grow. But when can that person determine that this seedling failed? Has the seed not germinated? Is it not growing? For the average gardener this is a guessing game. The application should allow the gardener to enter a date for planting. The application should store the planting dates and save them until the gardener decides that they no longer need that information and chooses to delete it.

Additionally, the application should be able to do the rough estimates and tell the gardener when to expect germination or when they can expect to see the newly growing plants. This way the gardener can determine when something has gone wrong and may choose to replant or choose a different planting.

The application should also allow the gardener to see roughly when the plantings will have grown to full maturity. While growing times are not an exact science the application should at least give the gardener a generalized window of when they can expect to reap the rewards of their hard work.

What about next time? When the garden has run its course and it is time to put up the hoe for the winter, will the plantings that the gardener just used be available or will they have to start over next year? The application should give the gardener the options to start over or to save their plantings to be able to use them again next year. That way if the garden worked well and they enjoyed their crops the gardener can simply log in the next season and see what they did before and do it again.   
 But the gardener does not want to spend time working on a database or dealing with a lot of text documents. That will be as much work as just doing it on paper. The application should sport a graphical user interface that is easy to use and understand.

It should include an intuitive user login screen so that the gardener can quickly get to their account. The menu should be simple with only relevant options. Once logged in the interface should be simple with easy use areas for researching their plants and adding them to their own list. The plant list should be instantly accessible, with planting dates, germination dates, and harvest dates listed in an easy to read and understandable format. The gardener should not have to dig to find all their information. The application should populate the gardener’s active list on log in so that they do not have to dig around to find their current information.

Future iterations of the application interface should include graphics of the plants, and easy to use charts and graphs including US zones, and growth charts.

There should be ongoing improvements in this application so that there is additional functionality in the future. For example, the application should eventually let the gardener know which plantings complement other plantings, also known as companion planting. For example, what types of tomatoes grow best with other types of tomatoes? The functionality should extend to making suggestions for the gardener to consider.

Eventually the application should migrate to a handheld device or cell phone app. This will allow the gardener to have the information in their pocket and be able to add or remove things in real time while they are building their garden.

In the future the application should include a garden planner graphical garden that will allow the user to not only pick their plantings, but also decide where in the garden they should go. This will allow the gardener to see their crops, the layout, and where they placed each planting.

Finally, the application should eventually move to include offline functionality. In the ideal world the gardener should not need to be connected to the internet to see their plantings and the information regarding those plantings. Allowing local storage of the user information will improve the ability of the gardener to have the info on hand even when there is no internet in the garden.

### Decomposition into Sub-Problems

The larger problem was first broken down into its two pieces, the website and the API. The website features are further broken down by their importance and the API into two parts, the public data domain and the private user data / user access authorization.

# Glossary of Terms

**Application** – Website built to display information to and be interacted with by the end user.

**Companion planting** – growing plants together where one or both plants benefit the other in ways that include pest control, pollination, attracting beneficial insects, or creating other favorable conditions.

**Database** – Online repository of information

**Gardener** – User of the application

**Germination** – Point in time from initial planting of seed to the time it sprouts from the seed.

**Germination date** – Time from initial sowing of seed to germination.

**GUI (Graphic User Interface)** - The web page, or app page that users will use in order to interact with the software.

**Harvest dates** – The date at which a plant is fully developed and is ready to be harvested.

**Local Storage** – Data storage located physically on the device in question.

**Mobile friendly** - an application that can be used across devices, such as a computer, tablet, and cell phone, without losing functionality.

**Planting date** – Date of initial sow of seed into the ground or source of dirt.

**Seedling** – A seed after planting but before germination.

**USDA Plant Hardiness zones** – A collection of geographic areas separated by average annual minimal temperature.

# System Requirements

### Business Goals

Diagram

Description automatically generated

### Enumerated Functional Requirements

|  |  |  |
| --- | --- | --- |
| **Identifier:** | **Priority:** | **Requirement:** |
| REQ-1 | 5 | The user can create an account. |
| REQ-3 | 5 | The application should allow users to input a zip code. |
| REQ-4 | 5 | The user can search for a plant by name. |
| REQ-5 | 3 | When selecting a plant by name, the application displays plant information. |
| REQ-6 | 4 | The application shall filter plant by zone. |
| REQ-7 | 5 | The application should display results based on filtering criteria. |
| REQ-8 | 5 | The user can input the date of planting. |
| REQ-12 | 5 | The application should save user’s entered planting dates. |
| REQ-13 | 5 | The application should display all plants for a given user. |

### Enumerated Nonfunctional Requirements

|  |  |  |
| --- | --- | --- |
| **Identifier:** | **Priority:** | **Requirement:** |
| REQ-15 | 5 | The application should require online connectivity. |
| REQ-16 | 2 | The application shall display relevant image of plant. |
| REQ-17 | 1 | The application shall store a local cache of user data. |
| REQ-18 | 3 | The application shall use Browser assisted location detection. |
| REQ-19 | 5 | The application shall reliably store user data with a loss of user data less than 10%. |
| REQ-20 | 4 | The application shall be easy for users to use. |

### User Interface Requirements

|  |  |  |
| --- | --- | --- |
| **Identifier:** | **Priority:** | **Requirement:** |
| REQ-21 | 5 | The application should require the user to enter a valid email and password. |
| REQ-22 | 5 | If the user forgets the account password, the applications UI should show the reset password features in the login page. |
| REQ-23 | 4 | The application shall require the user to first log in using appropriate login credentials to edit user information. |
| REQ-24 | 5 | If the user does not log in to a valid account, the application should allow the user to access basic search features that do not require saved user information. |
| REQ-25 | 2 | The application shall allow users to disable their account. |

# Functional Requirements Specification

### a. Stakeholders

**Development Team** – The interest of the Development Team in this project is primarily focused on the software engineering process and on the performance and functionality of the end project.

**Course Instructor** - The interest of the Course Instructor in this project is primarily focused on the collaboration and development process followed by the Development Team. The Course Instructor also has interest in the end result.

**End Users** – The interest of the End Users in this project is primarily focused on the functionality and ease-of-use of the final product.

### Actors and Goals

**User:**

|  |  |  |
| --- | --- | --- |
| Actor | Goal | Use Case Name |
| User | To create a new user account. | Create User Account (UC-1) |
| User | To create and save note to personal garden. | Take Note (UC-2) |
| User | To filter plant data by Hardiness Zone using Zip Code. | Filter by Zip Code (UC-3) |
| User | To search for planting information on a specific plant. | Search by Plant (UC-4) |
| User | To enter and save a planting date, calculate a germination date and calculate a harvest date. | Enter Planting Date (UC-5) |
| User | To design and save a graphic of their personal Garden Box. | Design Garden Box (UC-6) |
| User | To log in to the system and access their stored data. | Log In (UC-7) |
| User | To reset the password associated with their user account. | Password Reset (UC-8) |
| User | To update the stored user information. | Update User Information (UC-9) |
| User | To disable their user account and remove their saved data from the database. | Disable User Account (UC-10) |

**Database** – Primarily a participating actor that acts as storage for data used and modified by the System and by User.

### Use Cases

#### Casual Description

|  |  |  |
| --- | --- | --- |
| Use Case | Description | Requirements |
| UC-1: Create User Account | Allows users to create an account, setting a valid username and password combination. | REQ-1, REQ-15 |
| UC-3: Filter by Zip Code | Allows user to filter plants by zone by entering a valid Zip Code. | REQ-3, REQ-6, REQ-7, REQ-17 |
| UC-4: Search by Plant | Allows user to search for information on a plant by entering a valid plant name. | REQ-4, REQ-5, REQ-7, REQ-16 |
| UC-5: Enter Planting Date | Allows user to enter planting date to be stored in the application database. Future implementation will be added for entering a date for a given plant calculates the approximate germination and harvest date and display them for the user. | REQ-8, REQ-12, REQ-13, REQ-17, REQ-19 |
| UC-7: Log In | Allows user to sign in to account using valid credentials. | REQ-21 |
| UC-8: Password Reset | Allows user to reset the password for their account. | REQ-22, REQ-20 |
| UC-9: Update User Information | Allows user to update user information once they have signed in using valid credentials. | REQ-21, REQ-23 |
| UC-10: Disable User Account | Allows user to disable their account, removing all their stored data from the database. | REQ-25 |

#### Use Case Diagram

Diagram

Description automatically generated

#### iii. Traceability Matrix

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **REQ ID** | **PW** | UC-1 | UC-3 | UC-4 | UC-5 | UC-7 | UC-8 | UC-9 | UC-10 |
| REQ-1 | 5 | X |  |  |  |  |  |  |  |
| REQ-3 | 5 |  | X |  |  |  |  |  |  |
| REQ-4 | 5 |  |  | X |  |  |  |  |  |
| REQ-5 | 3 |  |  | X |  |  |  |  |  |
| REQ-6 | 4 |  | X |  |  |  |  |  |  |
| REQ-7 | 5 |  | X | X |  |  |  |  |  |
| REQ-8 | 5 |  |  |  | X |  |  |  |  |
| REQ-9 | 3 |  |  |  | X |  |  |  |  |
| REQ-10 | 4 |  |  |  | X |  |  |  |  |
| REQ-11 | 4 |  |  |  | X |  |  |  |  |
| REQ-12 | 5 |  |  |  | X |  |  |  |  |
| REQ-13 | 5 |  |  |  | X |  |  |  |  |
| REQ-15 | 5 | X |  |  |  |  |  |  |  |
| REQ-16 | 2 |  |  |  |  |  |  |  |  |
| REQ-17 | 1 |  | X | X | X |  |  |  |  |
| REQ-18 | 3 |  |  |  |  |  |  |  |  |
| REQ-19 | 5 |  |  |  | X |  |  |  |  |
| REQ-20 | 4 |  |  |  |  |  | X |  |  |
| REQ-21 | 5 |  |  |  |  | X |  | X |  |
| REQ-22 | 5 |  |  |  |  |  | X |  |  |
| REQ-23 | 4 |  |  |  |  |  |  | X |  |
| REQ-24 | 5 |  |  |  |  |  |  |  |  |
| REQ-25 | 2 |  |  |  |  |  |  |  | X |
| **MAX PW** | | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 2 |
| **TOTAL PW** | | 10 | 15 | 14 | 32 | 5 | 9 | 9 | 2 |

#### iv. Fully-Dressed Description

|  |  |
| --- | --- |
| **Use Case UC-1:** | **Create User Account** |
| **Related Requirements:** | REQ-1, REQ-15 |
| **Initiating Actor:** | User |
| **Actor’s Goal:** | To create a user account with |
| **Participating Actors:** | Database |
| **Preconditions:** | User has accessed the Sign In page for the website via a device with internet connectivity, such as a computer or smartphone. |
| **Postconditions:** | New user account is created and stored in the database. |
| **Flow of Events for Main Success Scenario:** | |
| ® | 1. **User** selects the on-screen “Make an Account” option |
| ¬ | 1. **System** displays a page on which the user can enter the email, password, name and zip code that they would like to use for the account. |
| ® | 1. **User** enters email, password, name and zip code in the appropriate fields and selects the on-screen “Create” option. |
| ¬ | 1. **System** (a) verifies in Database that the entered zip code is valid; (b) saves the new account data in **Database** and signals account creation. |
| **Flow of Events for Extensions (Alternate Scenarios):**  4a. System is unable to filter by appropriate Hardiness Zone. | |
| ¬ | 1. **System** displays page with message to **User** stating that the system was unable to filter data and to ensure that the entered zip code is valid. |

|  |  |
| --- | --- |
| **Use Case UC-3:** | **Filter by Zip Code** |
| **Related Requirements:** | REQ-3, REQ-6, REQ-7, REQ-17 |
| **Initiating Actor:** | User |
| **Actor’s Goal:** | To filter the plant data by Hardiness Zone by entering a valid US Zip Code. |
| **Participating Actors:** | Database |
| **Preconditions:** | User has accessed the Plant data search page using a device with internet connectivity, such as a computer or smartphone. |
| **Postconditions:** | The data page is updated with data filtered by the valid zip code entered by the user. |
| **Flow of Events for Main Success Scenario:** | |
| ® | 1. **User** enters a valid zip code in the appropriate text entry field and selects the “Filter” option. |
| ¬ | 1. **System** (a) determines the appropriate Hardiness Zone based on the entered zip code; (b) displays an updated data search page with information filtered by the appropriate hardiness zone. |
| **Flow of Events for Extensions (Alternate Scenarios):** | |
| 2a. System is unable to filter by appropriate Hardiness Zone. | |
| ¬ | 1. **System** displays page with message to **User** stating that the system was unable to filter data and to ensure that the entered zip code is valid. |

|  |  |
| --- | --- |
| **Use Case UC-4:** | **Search by Plant** |
| **Related Requirements:** | REQ-4, REQ-5, REQ-7, REQ-16 |
| **Initiating Actor:** | User |
| **Actor’s Goal:** | To search for data on a specific plant. |
| **Participating Actors:** | Database |
| **Preconditions:** | User has accessed the Plant data search page using a device with internet connectivity, such as a computer or smartphone. |
| **Postconditions:** | The data page is updated with data on the plant specified in the search criteria. |
| **Flow of Events for Main Success Scenario:** | |
| ® | 1. **User** enters a valid plant name in the search field and selects the on-screen “Search” option. |
| ¬ | 1. **System** (a) searches **Database** based on the entered plant name; (b) displays an updated data search page with information on the specified plant. |
| **Flow of Events for Extensions (Alternate Scenarios):** | |
| 2a. System is unable to filter by submitted plant name. | |
| ¬ | 1. **System** displays page with message to **User** stating that the system was unable to filter data and to ensure that the entered plant name is valid. |

|  |  |
| --- | --- |
| **Use Case UC-5:** | **Enter Planting Date** |
| **Related Requirements:** | REQ-8, REQ-9, REQ-10, REQ-11, REQ-12, REQ-13, REQ-17, REQ-19 |
| **Initiating Actor:** | User |
| **Actor’s Goal:** | To enter the planting date for a specified plant. |
| **Participating Actors:** | Database |
| **Preconditions:** | * User has logged in with valid user credentials. * User has accessed the Garden Management page using a device with internet connectivity, such as a computer or smartphone. * User has entered a valid zip code and a Hardiness Zone has been associated with the User’s account. |
| **Postconditions:** | Planting date, Germination date and Harvest date for the specified plant have been saved to the database for the user’s account and are displayed on the Garden Management page. |
| **Flow of Events for Main Success Scenario:** | |
| ® | 1. **User** enters a valid plant name in the Plant Name text entry field and a valid date in the Planting Date text entry field and selects the “Add Plant” option. |
| ¬ | 1. **System** (a) searches the database for information based on the entered plant name; (b) calculates the Germination date and Harvest date based on data for specified plant; (c) saves the Plant Name, Planting Date, Germination date and Harvest Date for the entry to **Database** for the user account; (d) displays the Garden Management page with all added plants for the current user along with accompanying dates. |
| **Flow of Events for Extensions (Alternate Scenarios):** | |
| 2a. System is unable retrieve data based on entered Plant Name. | |
| ¬ | 1. **System** displays Garden Management page with message to **User** stating that the system was unable to create entry and to ensure that the entered plant name is valid, followed by the other contents of the page. |

|  |  |
| --- | --- |
| **Use Case UC-7:** | **Log In** |
| **Related Requirements:** | REQ-21 |
| **Initiating Actor:** | User |
| **Actor’s Goal:** | To log in to system using valid user credentials. |
| **Participating Actors:** | Database |
| **Preconditions:** | * User has accessed the Main page using a device with internet connectivity, such as a computer or smartphone. * User has previously created an account using an email and password combination. |
| **Postconditions:** | User is signed into system and the main page is updated and the user’s stored information is accessible via the Garden Management Page. |
| **Flow of Events for Main Success Scenario:** | |
| ® | 1. **User** selects the on-screen “Sign In” option on the main page. |
| ¬ | 1. **System** displays the Sign In page with an Email text entry field, a password text entry field, a “Forgot Password” hyperlink, a “Make New Account” hyperlink and a “Sign In” button. |
| ® | 1. **User** enters a valid email and password into the appropriate fields and selects the “Sign In” button |
| ¬ | 1. **System** (a) validates user credentials from **Database** of user accounts; (b) displays the Garden Management page to **User**. |
| **Flow of Events for Extensions (Alternate Scenarios):** | |
| 4a. Invalid Email/Password entered – Unable to validate User. | |
| ¬ | 1. **System** displays the Log In page with message to **User** stating that an invalid email or password was entered, followed by the Log In form. |

|  |  |
| --- | --- |
| **Use Case UC-8:** | **Password Reset** |
| **Related Requirements:** | REQ-22, REQ-20 |
| **Initiating Actor:** | User |
| **Actor’s Goal:** | To reset the password associated with their account. |
| **Participating Actors:** | Database |
| **Preconditions:** | * User has accessed the Sign In page using a device with internet connectivity, such as a computer or smartphone. * User has previously created an account using an email and password combination. |
| **Postconditions:** | The password associated with the given account has been changed and updated in the database. |
| **Flow of Events for Main Success Scenario:** | |
| ® | 1. **User** selects the on-screen “Forgot Password” option on the Sign In page. |
| ¬ | 1. **System** displays the Reset Password page with an Email text entry field and a “Send Link” button. |
| ® | 1. **User** enters a valid email associated with the account and selects the “Send Link” button |
| ¬ | 1. **System** (a) validates user email from **Database** of user accounts; (b) sends an email to the provided email address containing a link to reset the user password. |
| ® | 1. **User** (a) uses the link provided in the email to navigate to the Set Password page containing a New Password text entry field, a Confirm Password text entry field, and an “Update Password” button; (b) enters the new password into the New Password and Confirm Password fields and select the “Update Password” Button. |
| ¬ | 1. **System** updates the password associated with the user email in **Database** and displays the Sign In page. |
| **Flow of Events for Extensions (Alternate Scenarios):** | |
| 4a. Invalid Email – Unable to validate User. | |
| ¬ | 1. **System** displays the Reset Password page with message to **User** stating that an invalid email was entered, followed by the Reset Password form. |

|  |  |
| --- | --- |
| **Use Case UC-9:** | **Update User Information** |
| **Related Requirements:** | REQ-21, REQ-23 |
| **Initiating Actor:** | User |
| **Actor’s Goal:** | To update user information such as Name, Email, Zip Code and Password. |
| **Participating Actors:** | Database |
| **Preconditions:** | * User has logged in with valid credentials. * User has accessed the User Profile page using a device with internet connectivity, such as a computer or smartphone. |
| **Postconditions:** | The edited user information field is updated on the User Profile page and in the database. |
| **Flow of Events for Main Success Scenario:** | |
| ® | 1. User selects edit icon for the information that they would like to update. |
| ¬ | 1. System displays a prompt box to user containing a text entry field for the updated information. |
| ® | 1. User enters the updated information into the text entry field and selects the “Update” option. |
| ¬ | 1. System updates the corresponding information in **Database** for the user to reflect the change, then displays and updated User Profile page to reflect the change made. |
| **Flow of Events for Extensions (Alternate Scenarios):** | |
| 3a. User selects the “Cancel” option. | |
| ¬ | 1. **System** does not change any user data and displays the User Profile page. |

|  |  |
| --- | --- |
| **Use Case UC-10:** | **Disable User Account** |
| **Related Requirements:** | REQ-25 |
| **Initiating Actor:** | User |
| **Actor’s Goal:** | To disable the User’s account and remove all related data from database. |
| **Participating Actors:** | N/A |
| **Preconditions:** | * User has logged in with valid credentials. * User has accessed the User Profile page using a device with internet connectivity, such as a computer or smartphone. |
| **Postconditions:** | The User account credentials and all data related to the account have been removed from the database. |
| **Flow of Events for Main Success Scenario:** | |
| ® | 1. **User** selects the “Disable Account” option. |
| ¬ | 1. **System** displays a prompt to **User** to confirm if they would like to disable their account. |
| ® | 1. **User** selects the “Disable Account” option located in the prompt box. |
| ¬ | 1. **System** (a) deletes all of the store user data from **Database**; (b) displays the Sign In page |
| **Flow of Events for Extensions (Alternate Scenarios):** | |
| 3a. User selects the “Cancel” option in the prompt box. | |
| ¬ | 1. **System** does not change any user data and displays the User Profile page. |

### System Sequence Diagrams

A picture containing text

Description automatically generated

A picture containing text

Description automatically generated

A picture containing diagram

Description automatically generated

Text

Description automatically generated with medium confidence

A picture containing text

Description automatically generated

Text

Description automatically generated with medium confidence

A picture containing text

Description automatically generated

A picture containing text

Description automatically generated

# Effort Estimation Using Use Case Points

|  |  |  |  |
| --- | --- | --- | --- |
| Actor name | Description of relevant characteristics | Complexity | Weight |
| User | User is interacting with the program via a graphical user interface. | Complex | 3 |
| Database | Database is another system interacting through a protocol. | Average | 2 |
| Unadjusted Actor Weight(UAW) | | | 5 |

|  |  |  |  |
| --- | --- | --- | --- |
| Use case | Description | Category | Weight |
| Create User Account (UC-1) | Complex user interface. 4 steps for the main success scenario. One participating actor (Database). | Average | 10 |
| Take Note (UC-2) | Moderate user interface. 2 steps for the main success scenario. One participating actor (Database). | Average | 10 |
| Filter by Zip Code (UC-3) | Complex user interface. 2 steps for the main success scenario. One participating actor (Database). | Average | 10 |
| Search by Plant (UC-4) | Complex user interface. 2 steps for the main success scenario. One participating actor (Database). | Average | 10 |
| Enter Planting Date (UC-5) | Simple user interface. 2 steps for the main success scenario. One participating actor (Database). | Simple | 5 |
| Design Garden Box (UC-6) | Complex user interface. 4 steps for the main success scenario. One participating actor (Database). | Average | 10 |
| Log In (UC-7) | Simple user interface. 4 steps for the main success scenario. One participating actor (Database). | Simple | 5 |
| Password Reset (UC-8) | Simple user interface. 6 steps for the main success scenario. One participating actor (Database). | Average | 10 |
| Update User Information (UC-9) | Complex user interface. 4 steps for the main success scenario. One participating actor (Database). | Average | 10 |
| Disable User Account (UC-10) | Complex user interface. 4 steps for the main success scenario. No participating actors. | Average | 10 |
| Unadjusted Use Case Weight (UUCW) | | | 90 |

Unadjusted Use Case Points (UUCP) = 95

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Technical Factor | Description | Weight | Perceived Complexity | Calculated Factor |
| T1 | Distributed, Web-based system | 2 | 3 | 6 |
| T2 | Users expect good performance but nothing exceptional | 1 | 3 | 3 |
| T3 | No exceptional demands for end-user efficiency | 1 | 3 | 3 |
| T4 | Internal processing is relatively simple | 1 | 1 | 1 |
| T5 | No requirement for reusability | 1 | 0 | 0 |
| T6 | Ease of install not required | 0.5 | 0 | 0 |
| T7 | Ease of use is very important | 0.5 | 5 | 2.5 |
| T8 | Minimal portability concerns | 2 | 2 | 4 |
| T9 | Easy to change minimally required | 1 | 1 | 1 |
| T10 | Concurrent use is required | 1 | 4 | 4 |
| T11 | Security is an average concern | 1 | 3 | 3 |
| T12 | No direct access to third parties | 1 | 0 | 0 |
| T13 | No unique training needs | 1 | 0 | 0 |
| Technical Factor Total: | | | | 27.5 |

Technical Complexity Factor (TCF) = 0.875

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Environmental Factor | Description | Weight | Perceived Impact | Calculated Factor |
| E1 | Beginner familiarity with the UML-based development | 1.5 | 1 | 1.5 |
| E2 | Some familiarity with application problem | 0.5 | 2 | 1 |
| E3 | Some knowledge of object-oriented approach | 1 | 2 | 2 |
| E4 | Beginner lead analyst | 0.5 | 1 | 0.5 |
| E5 | Average motivated team members | 1 | 3 | 3 |
| E6 | Stable requirements expected | 2 | 3 | 6 |
| E7 | No part-time staff will be involved | -1 | 0 | 0 |
| E8 | Programming language of average difficulty will be used | -1 | 3 | -3 |
| Environmental Factor Total: | | | | 11 |

Environment Complexity Factor (ECF) = 1.07

Use Case Points (UCP) = 89

Duration = (UCP) 89 \* (PF) 28 = 2492 person-hours

### User Effort Estimation

#### UC-1: Create a User Account

1. NAVIGATION: total 2 clicks, as follows:
   1. Click “Sign Up” button

--- *after completing data entry as shown below ---*

* 1. Click “Sign Up” button to submit

1. DATA ENTRY: total 1 clicks and 17 keystrokes, as follows:
   1. Click cursor to “Name” text field
   2. Press any alphanumerical key(s)
   3. Press “Tab” key to move to next text field (“Email”)
   4. Press any alphanumerical key(s), followed by “@”, followed by any alphanumerical key(s), followed by “.com”
   5. Press “Tab” key to move to next text field (“Password”)
   6. Press any alphanumerical key(s)
   7. Press “Tab” key to move to next text field (“Zipcode”)
   8. Press any five numeric keys

#### UC-3: Filter by Zip Code

1. NAVIGATION: total 2 clicks, as follows:
   1. Click cursor to “Zipcode” search box

--- *after completing data entry as shown below ---*

* 1. Click “Search” button

1. DATA ENTRY: total 0 clicks and 5 keystrokes, as follows:
   1. Press any five numeric keys

#### UC-4: Search by Plant

1. NAVIGATION: total 2 clicks, as follows:
   1. Click cursor to “Plant Name” search box

--- *after completing data entry as shown below ---*

* 1. Click “Search” button

1. DATA ENTRY: total 0 clicks and 1 keystrokes, as follows:
   1. Press any alphanumerical key(s)

#### UC-5: Enter Planting Date

1. NAVIGATION: total \_ clicks, as follows:
   1. Click on “Add” button next to plant name on search results page

--- *after completing data entry as shown below ---*

* 1. Click “Add Planting” button

1. DATA ENTRY: total \_ clicks and \_\_ keystrokes, as follows:
   1. Click in “Planting date” field
   2. Type ten alphanumeric keys in the format: “mm/dd/yyyy”
   3. Click outside of Planting Date box to close popout calendar

# Domain Analysis

### a. Conceptual Model

The domain was derived from the various use cases of the system. The system is initially divided by a guest user vs a registered users, both of which interact with the Main website. Then divided further by specific tasks needed to accomplish use cases.

**Main Website:**

This concept is where the user will enter all information and then visually see all the requested information back. It will be the main portal for the user.

**User Account:**

This concept will handle the specific user account information for registered. It will retrieve the user information from the website, and then request the data from the user storage. It will then feed that information back to the user Main website for display and processing by the plant retrieval system.

**Plant Retrieval:**

This concept takes the info from either the guest user portion of the website or the user account and grabs the plant data from the storage. It then processes that information, performs calculations and then sends the data to the website

**Zone System:**

This concept will take a zip code entered onto the website from either a guest or user, send it to the to the plant retrieval system for specific plants. It also sends specific zone information back to the website for display.

**User Storage:**

This concept takes requests for registered user data and sends the information back to the user account. It also stores user data, creates new users and stores their data, updates and deletes users upon request from the User Account system.

**Plant Storage:**

This concept stores all plant data. It will take requests for specific plants, ranges of plants, or plants by zone. It takes no other requests and is not editable by users.

Diagram

Description automatically generated

#### CONCEPT DEFINITIONS

|  |  |  |
| --- | --- | --- |
| **Responsibility** | **Type** | **Concept Name** |
| R1: Display the information to the User | D | Website |
| R2: Retrieve Information from the user | D | Website |
| R3: Create a new User Account | D | User Account |
| R4: Updating user information | K | User Storage |
| R5: Deleting User | D | User Account |
| R6: Store the new User Account | K | User Storage |
| R7: Retrieve user Account information | D | User Account |
| R8: Sending Information about the user | K | User Storage |
| R9: Retrieving zone information | D | Zone System |
| R10: Retrieving plant list | K | Plant Storage |
| R11: Requesting a list of plants for Guest | D | Plant Retrieval |
| R12: Requesting list of stored user plants based on ID | D | Plant Retrieval |
| R13: Calculating Plant information | D | Plant Retrieval |

#### ASSOCIATION DEFINITIONS

|  |  |  |
| --- | --- | --- |
| **Concept Pair** | **Associate Description** | **Association Name** |
| Main Website ó User Account | Sends and receives information regarding registered users | Display / retrieve Information |
| Main Website ó Plant Retrieval | Sends and receives information regarding plants | Display / retrieve Information |
| Main Website ó Zone System | Sends and receives information regarding zones | Display / retrieve Information |
| User Account ó User Storage | Sends information to store, add, delete, update. Retrieves all user information | Update / Retrieve User information |
| User Account ó Plant Retrieval | Sends Specific user plant information | Sends / Receives information |
| Plant Retrieval ó Zone System | Sends Specific user zone information | Sends / Receives information |
| Plant Retrieval ó Plant Storage | Sends requests and retrieves information | Send / Retrieve information |
| Registered Users ó Main Website | Enters information | Retrieve information |
| Guest User ó Main Website | Enters information | Retrieve information |

#### ATTRIBUTE DEFINITIONS

|  |  |  |
| --- | --- | --- |
| **Concept** | **Attribute** | **Definition** |
| Main Website | getUserInformation | Gets information from the user |
| displayUserInformation | Displays the information about the user |
| getUserRequest | Gets information the user is requesting |
| displayUserRequest | Displays the requested information |
| User Account | getUserAccount | Gets information for the user account |
| updateUserAccount | Updates or Deletes User account |
| createNewUser | Creates a new user account |
| Plant Retrieval | getPlantData | Gets plant data from user request |
| getPlantsinZone | Gets plants from a specific zone |
| getUserPlants | Gets the plants that belong to the user |
| Zone System | getZoneInformation | Gets information about the zone |
| User Storage | createUser | Creates a new user |
| updateUser | Updates a user |
| deleteUser | Deletes a user |
| Plant Storage | sendRequestedInfo | Sends requested plant data |

#### TRACEABILITY MATRIX

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Domain Concepts** | | | | | |
| **USE**  **CASE** | **PW** | Main Website | User Account | User Storage | Plant Retrieval | Plant Storage | Zone System |
| UC-1 | 3 | X | X | X |  |  |  |
| UC-2 | 2 | X |  | X |  |  |  |
| UC-3 | 3 | X |  |  | X |  | X |
| UC-4 | 5 | X | X | X | X | X |  |
| UC-5 | 4 | X |  | X | X | X |  |
| UC-6 | 4 | X | X | X | X |  |  |
| UC-7 | 3 | X | X | X |  |  |  |
| UC-8 | 3 | X | X | X |  |  |  |
| UC-9 | 3 | X | X | X |  |  |  |
| UC-10 | 3 | X | X | X |  |  |  |
|  |  |  |  |  |  |  |  |

### System Operation Contracts

|  |  |
| --- | --- |
| Use Case: | UC-1 |
| Preconditions: | * User has accessed the Sign In page for the website via a device with internet connectivity, such as a computer or smartphone. |
| Postconditions: | * New user account is created and stored in the database. |

|  |  |
| --- | --- |
| Operation | Take Note |
| Use Case: | UC-2 |
| Preconditions: | * User is currently logged in and is currently on the page showing their saved garden data. |
| Postconditions: | * User’s custom note is saved to the database and displayed on their garden information page. |

|  |  |
| --- | --- |
| Operation | Filter by Zip Code |
| Use Case: | UC-3 |
| Preconditions: | * User has accessed the Plant data search page using a device with internet connectivity, such as a computer or smartphone. |
| Postconditions: | * The data page is updated with data filtered by the valid zip code entered by the user. |

|  |  |
| --- | --- |
| Operation | Search by Plant |
| Use Case: | UC-4 |
| Preconditions: | * User has accessed the Plant data search page using a device with internet connectivity, such as a computer or smartphone. |
| Postconditions: | * The data page is updated with data on the plant specified in the search criteria. |

|  |  |
| --- | --- |
| Operation | Enter Planting Date |
| Use Case: | UC-5 |
| Preconditions: | * User has logged in with valid user credentials. * User has accessed the Garden Management page using a device with internet connectivity, such as a computer or smartphone. * User has entered a valid zip code and a Hardiness Zone has been associated with the User’s account. |
| Postconditions: | * Planting date, Germination date and Harvest date for the specified plant have been saved to the database for the user’s account and are displayed on the Garden Management page. |

|  |  |
| --- | --- |
| Operation | Design Garden Box |
| Use Case: | UC-6 |
| Preconditions: | * User has logged in with valid user credentials. * User has accessed the Garden Management page using a device with internet connectivity, such as a computer or smartphone. |
| Postconditions: | * A graphic displaying the Garden Box design has been stored to the database for the user account and is displayed when the user accesses the Garden Management page. |

|  |  |
| --- | --- |
| Operation | Log In |
| Use Case: | UC-7 |
| Preconditions: | * User has accessed the Main page using a device with internet connectivity, such as a computer or smartphone. * User has previously created an account using an email and password combination. |
| Postconditions: | * User is signed into system and the main page is updated and the user’s stored information is accessible via the Garden Management Page. |

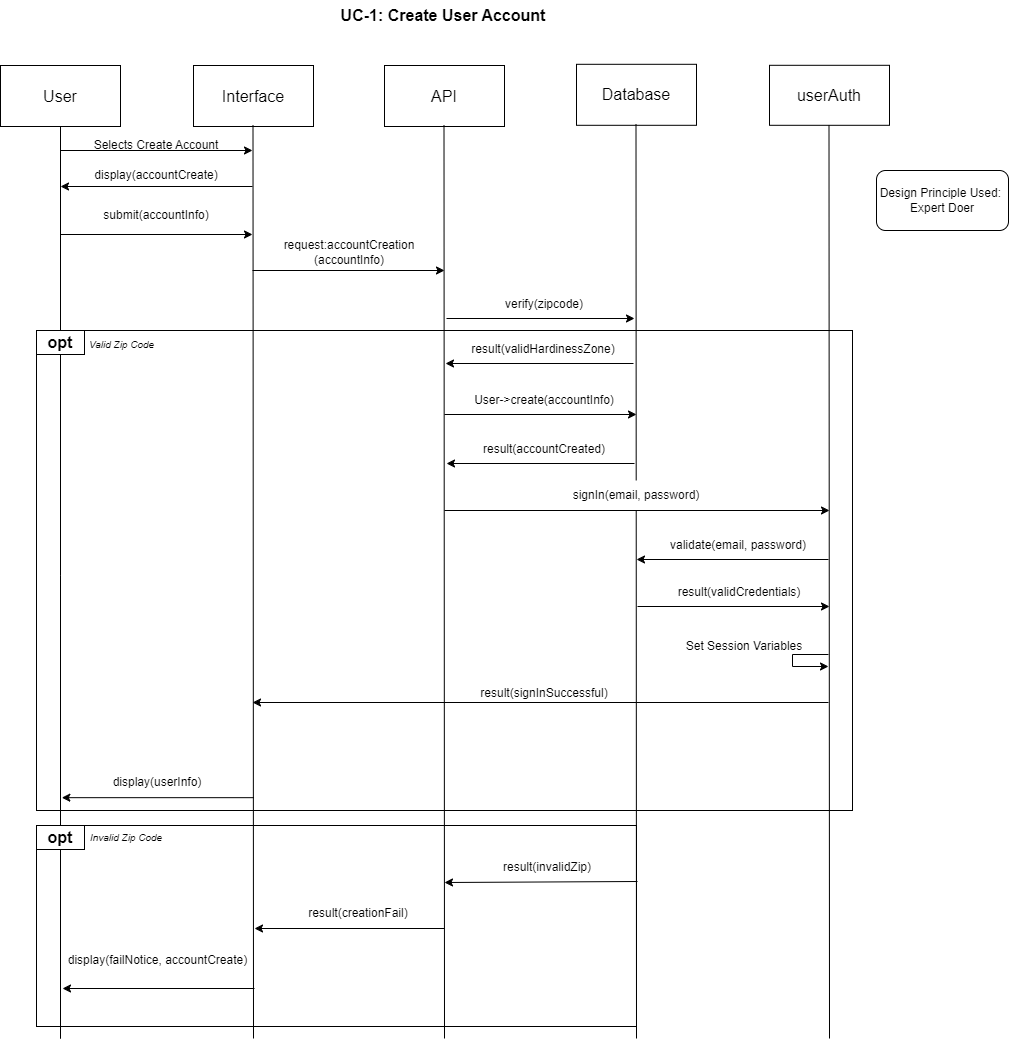
|  |  |
| --- | --- |
| Operation | Password Reset |
| Use Case: | UC-8 |
| Preconditions: | * User has accessed the Sign In page using a device with internet connectivity, such as a computer or smartphone. * User has previously created an account using an email and password combination. |
| Postconditions: | * The password associated with the given account has been changed and updated in the database. |

|  |  |
| --- | --- |
| Operation | Update User Information |
| Use Case: | UC-9 |
| Preconditions: | * User has logged in with valid credentials. * User has accessed the User Profile page using a device with internet connectivity, such as a computer or smartphone. |
| Postconditions: | 1. The edited user information field is updated on the User Profile page and in the database. |

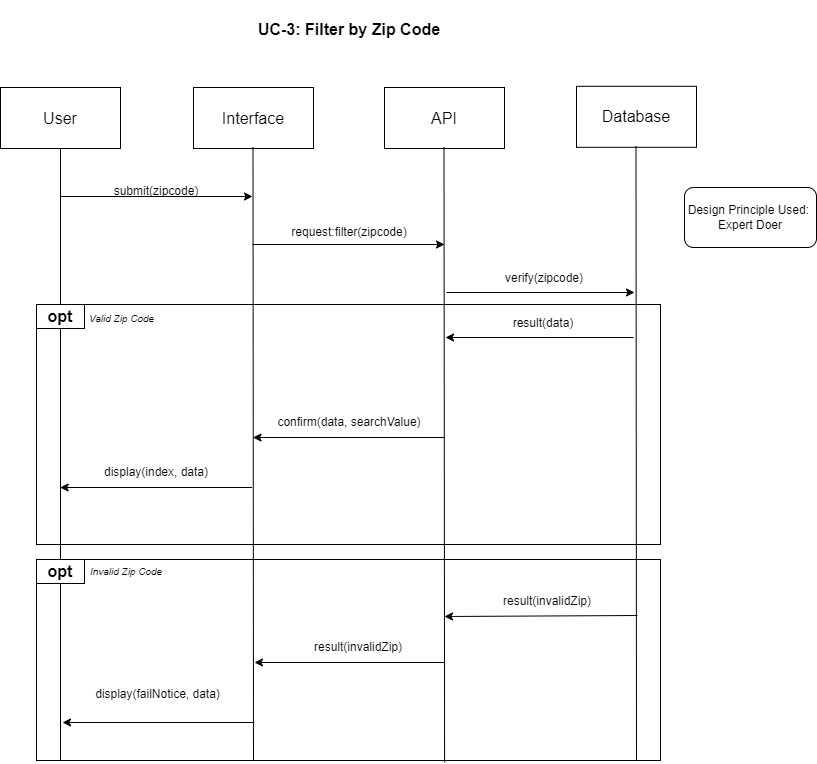
|  |  |
| --- | --- |
| Operation | Disable User Account |
| Use Case: | UC-10 |
| Preconditions: | 1. User has logged in with valid credentials. 2. User has accessed the User Profile page using a device with internet connectivity, such as a computer or smartphone. |
| Postconditions: | 1. The User account credentials and all data related to the account have been removed from the database. |

# Interaction Diagrams

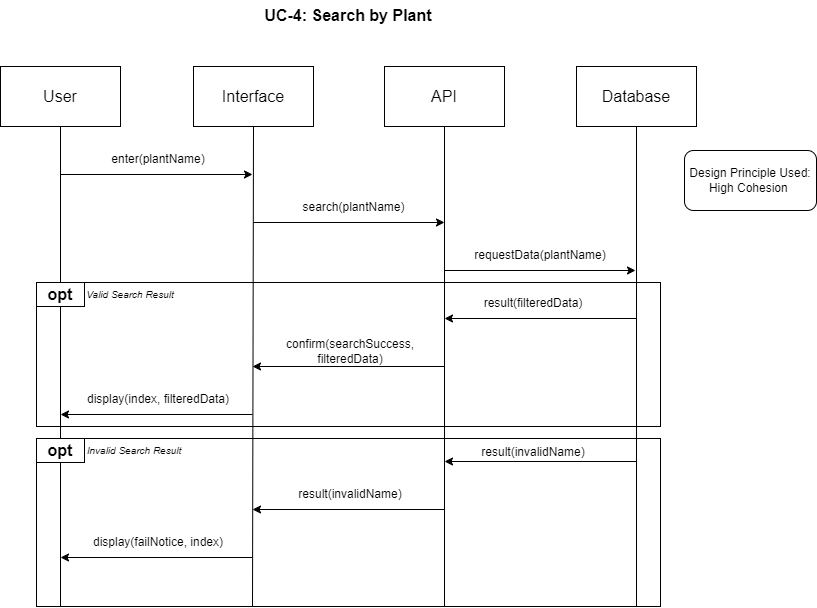
### Interaction Diagrams



UC-1: The design pattern used in this revised Interaction Diagram is the Controller Design Pattern. This was chosen as it enables more secure data-sharing between the Front End and the Database. All Data is processed and then handed off to the API to create the account in the database, preventing unwanted access to the database.



UC-3: The design pattern used in this revised Interaction Diagram is the Publisher-Subscriber Pattern. This was chosen due to the nature of requesting data from the database. The Interface acts as our Subscribed and the API acts as our Publisher. This prevents users from having direct access to the database from the interface.

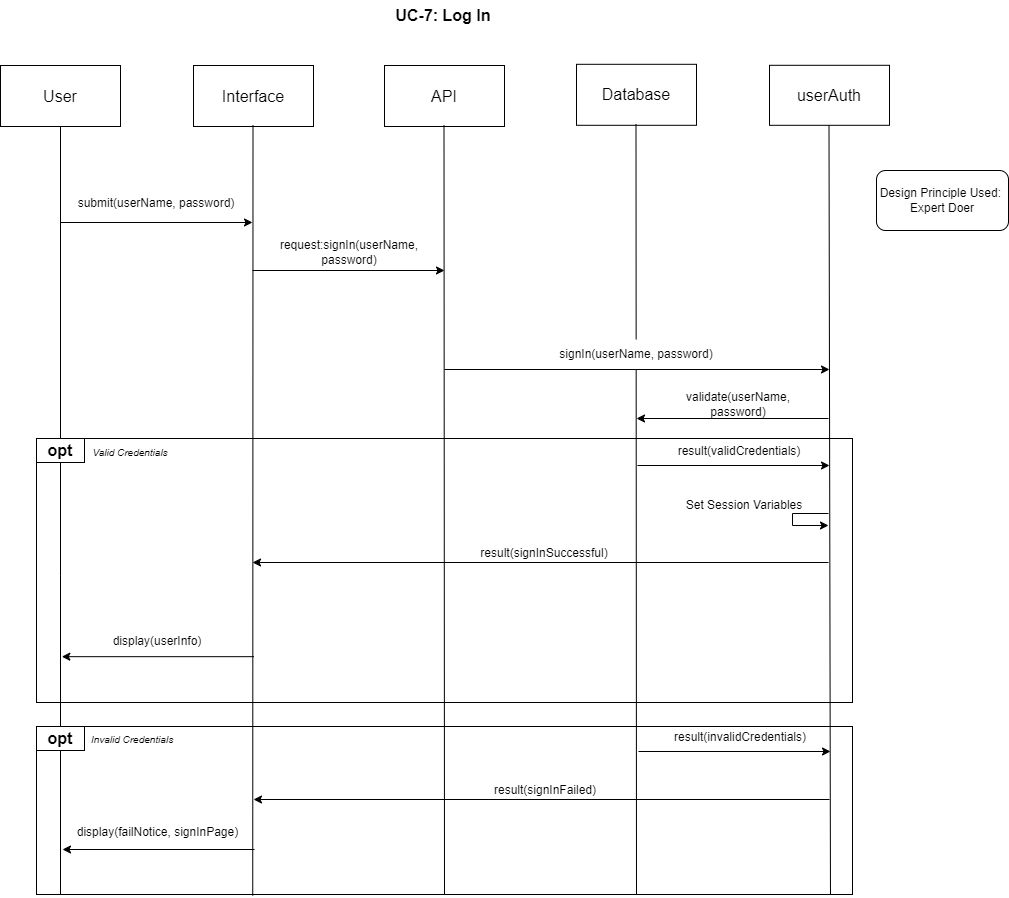


UC-4: The design pattern used in this revised Interaction Diagram is the Published-Subscriber Pattern. Much like in UC-3, the Interface acts as our Subscribed and the API acts as our Publisher, preventing direct access to the database from the Interface.

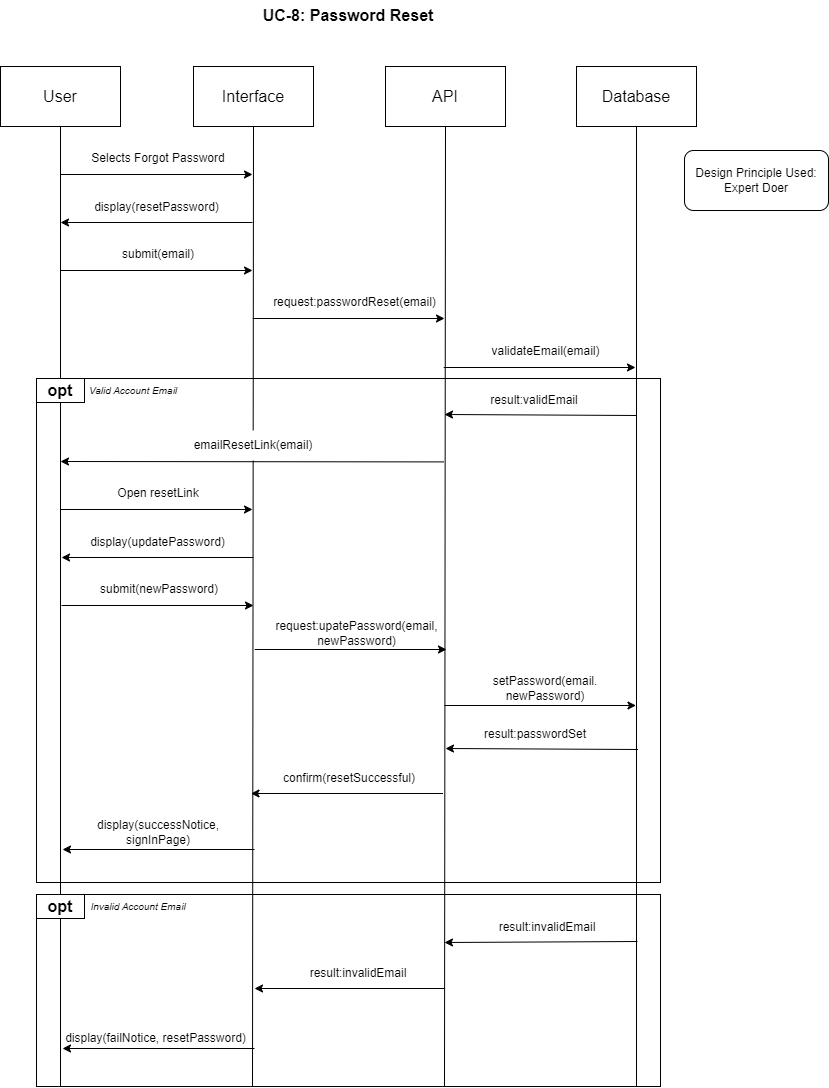
Text

Description automatically generated with low confidence

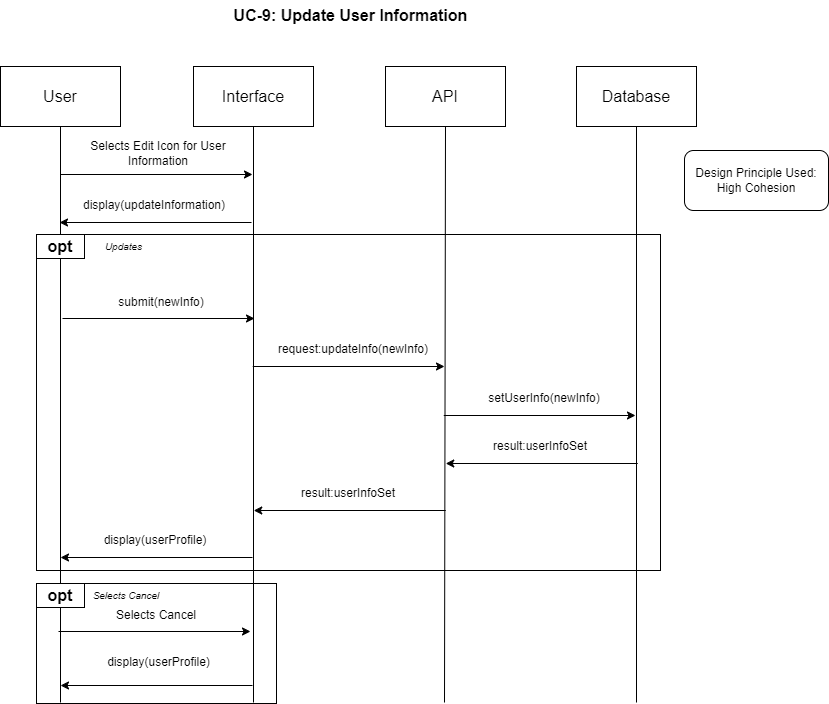
UC-5: The design pattern used in this revised Interaction Diagram is the Controller Design Pattern. This was chosen to improve security when updating user information in the Database. All Data is processed and then handed off to the API. This prevents the Interface from having direct access to the database and allows additional layers of protection to be implemented.



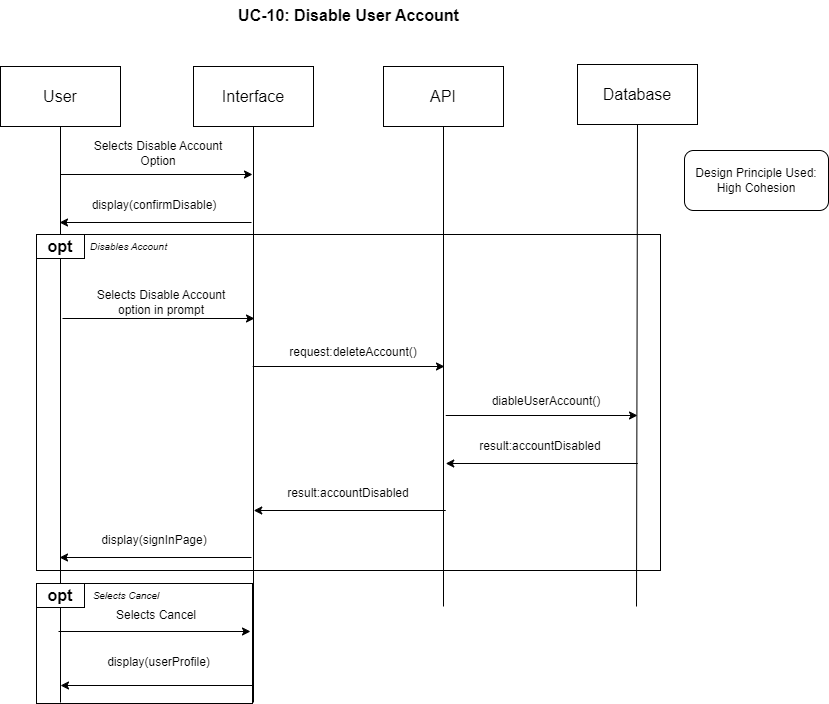
UC-7: The design pattern used in this revised Interaction Diagram is the Controller Design Pattern. This was chosen as it enables more secure data sharing between the Front End and the Database. All Credential information is processed and then handed off to the API to validate the data in the database, preventing unwanted access to the database.



UC-8: The design pattern used in this revised Interaction Diagram is the Controller Design Pattern. This was chosen as it enables more secure processing when a password reset is requested. Additionally, as the database and interface are modified, the API can be more easily maintained, allowing for better software reliability and longevity.



UC-9: The design pattern used in this revised Interaction Diagram is the Controller Design Pattern. Rather than the Interface modifying the database directly, the new user information is passed to the API, which functions as our controller. This aids in database security. Additionally, as the database and interface are modified, the API can be more easily maintained, allowing for better software reliability and longevity.



UC-10: The design pattern used in this revised Interaction Diagram is the Controller Design Pattern. This new design helps keep the database secure and prevents the improper usage of UC-10 to mass disable accounts. Additionally, as the database and interface are modified, the API can be more easily maintained, allowing for better software reliability and longevity.

# Class Diagrams and Interface Specification

### Class Diagram

Diagram, schematic

Description automatically generated

### Data Types and Operation Signatures

|  |
| --- |
| **User** |
| -username: String |
| -zipCode: int |
| -plantData: text |
| -email: String |
| -password: String |
| +getUsername(): String |
| +setUsername(): void |
| +getZipCode(): int |
| +setZipCode(): void |
| +getPlantData(): text |
| +addPlantingData(): void |
| +getEmail(): String |
| +setEmail(): void |
| -getPassword(): String |
| -setPassword(): void |

**Class: User**

Attributes:

* username: String – The username of the User.
* zipCode: int – The zip code that the User entered for plant data.
* plantData: text – The User’s saved plant data.
* email: String – The User’s email address.
* password: String – The encrypted User password.

Methods:

* getUsername(): String - Returns the value of username.
* setUsername(): void - Sets username to a provided value.
* getZipCode(): int - Returns the value of zipCode.
* setZipCode(): void - Sets zipCode to a provided value.
* getPlantData(): text - Returns the value of plantData.
* addPlantingData(): void - Adds provided data to the User’s plantData.
* getEmail(): String - Returns the value of email.
* setEmail(): void - Sets email to a provided value.
* getPassword(): String - Returns the value of zipCode – This is only used during the User Authentcation process.
* setPassword(): void – Sets password to a provided value that has been encrypted.

|  |
| --- |
| **userAuth** |
| +validateEmail(): boolean |
| +validatePassword(): boolean |

**Class: userAuth**

Methods:

* validateEmail(): boolean - Returns True or False based on if the email is a valid email.
* validatePassword(): boolean – Returns True or False based of if the email and password combination are valid for a user account.

|  |
| --- |
| **Plant** |
| -id: int |
| -name: String |
| -type: String |
| -planting instructions: String |
| -germinationDate: String |
| -harvestDate: String |
| +getPlantID(): int  +setPlantId(): void |
| +getName(): String |
| +setName(): void |
| +getType(): String |
| +setType(): void |
| +getPlantingInstructrions(): String |
| +setPlantingInstructions(): void |
| +getGerminationDate(): String |
| +setGerminationDate(): void |
| +getHarvestDate(): String |
| +setHarvestDate(): void |

**Class: Plant**

Attributes:

* + id: int – the plant identification number
  + name: String – Name of the plant
  + type: String – Type of plant
  + planting instructions: String – instructions on how to plant
  + germinationDate: String – germination date range as string value
  + harvestDate: String – havest date range as a string value
* zipCode: int – The zip code that the User entered for plant data.
* plantData: text – The User’s saved plant data.
* email: String – The User’s email address.
* password: String – The encrypted User password.

Methods:

* +getPlantID(): int – Returns the ID of the plant as an int value
* +setPlantId(): void – sets the ID of the plant
* +getName(): String – Returns the name of the plant as a string value
* +setName(): void – Sets the name of the plant
* +getType(): String – Gets the type of plant as a string value
* +setType(): void – Set the type of plant
* +getPlantingInstructions(): String – Gets the planting instructions as a string value
* +setPlantingInstructions(): void- Sets the planting instructions
* +getGerminationDate(): String – Gets the germination date range as a string
* +setGerminationDate(): void – Sets the germination date range
* +getHarvestDate(): String – Gets the harvest date range as a string
* +setHarvestDate(): void – Sets the harvest date range

|  |
| --- |
| **PlantCalc** |
| -plantId: int |
| -plantingDate: Date |
| -germinationDate: Date |
| -harvestDate: Date |
| +getPlantID(): int |
| +setPlantId(): void |
| +setPlantingDate(): void |
| +getPlantingDate(): Date |
| +setGerminationDate(): void |
| +getGerminationDate(): Date |
| +setHarvestDate(): void |
| +getHarvestDate(): Date |
| +extractDate(): int |
| +calculateDate(): Date |

**Class: PlantCalc**

Attributes:

* -plantId: int – Id of the plant
* -plantingDate: date – Date the plant was planted (comes from user data)
* -germinationDate: date – Estimated date the plant will germinate calculated
* -harvestDate: date - Estimated date the plant will reap harvest calculated

Methods:

* +getPlantID(): int – returns plant ID
* +setPlantId(): void – sets the ID of the plant
* +setPlantingDate(): void – Sets the date of planting from user data
* +getPlantingDate(): Date – returns a Date object that is the day the plant was planted
* +setGerminationDate(): void – sets the germination date calls calculateDate()
* +getGerminationDate(): Date – returns a Date object that is an estimate of germination
* +setHarvestDate(): void – sets the date of potential harvest calls calculateDate()
* +getHarvestDate(): Date – returns a Date object that is an estimate of harvest
* +extractDate(): int – takes a string date range and extracts day value as an int
* +calculateDate(): Date – calls extractDate() returns a Date value based on calling functions Date

|  |
| --- |
| **PlantGrowingRelationship** |
| -id: int  -plantIdOne: int  -plantNameOne: String |
| -plantIdTwo: int  -plantNameTwo: String |
| -relationship: String |
|  |
|  |
| + getPlantIDOne (): int |
| + setPlantIDOne (): void |
| + getPlantIdTwo(): int |
| + setPlantIdTwo(): void |
| + checkRelationship(): String |
|  |

**Class: PlantGrowingRelationship**

Attributes:

* -id: int – id of search
* - plantIdOne: int – Id of the plant
* - plantNameOne: String – name of plant one
* - plantIdTwo: int – ID of the plant to check relationship with
* - plantNameTwo: String – name of plant two
* - relationship: String – relationship between the plants Eg. (“helps, neutral, or avoid”)

Methods:

* + getPlantIDOne(): int – returns ID of plant one
* + setPlantIDOne (): void – sets the ID of plant one
* + getPlantIDTwo(): int – returns ID of plant one
* + setPlantIDTwo(): void – sets the ID of plant one
* +checkRelationship(): String – retrieves and sets relationship between plant one and plant two

|  |
| --- |
| **PlantingInstructions** |
| -id: String |
| -plantId: String |
| -plantName: String |
| -zoneID: String  -zoneNumber: String  -plantingZoneSub: String  -plantingType: String  -startDate: String  -endDate: String |
|  |
| + getID(): String |
| +getPlantId(): String |
| +getPlantName(): String |
| +getZoneID(): String  +getZoneNumber(): String  +getPlantingZoneSub(): String  +getPlantingType(): String  +getStartDate(): String  +getEndDate(): String  +checkInstructions(): String |
|  |

**Class: Planting Instructions**

Attributes:

* -id: int – id of search
* -plantID: String – unique id of plant
* -plantName: String – name of plant
* -zoneID: String – unique ID of planting zone
* -zoneNumber: String – plant hardiness zone number (eg. “8”)
* -plantingZoneSub: String – sub planting zone (eg. “A”)
* -plantingType: String – season that the plant should be started (eg. “Spring”)
* -startDate: String –start of recommended date range for planting
* -endDate: String – end of recommended date range for planting

Methods:

* +getPlantID(): String – returns plantID
* +getPlantName(): String – returns plantName
* +getZoneID(): String – returns zoneID
* +getZoneNumber(): String – returns zoneNumber
* +getPlantingZoneSub(): String – returns plantingZoneSub
* +getPlantingType(): String – returns plantingType
* +getStartDate(): String – returns startDate
* +getEndDate(): String – returns endDate
* +checkInstructions(): String – sets all variable data linked to given search id

|  |
| --- |
| **ZipCode** |
| -zipCode: int |
| +setZipCode(): void |
| +getZipCode(): int |

**Class: ZipCode**

Attributes:

* -zipCode: int - The zip code that the User entered for plant data.

Methods:

* +setZipCode(): void - Sets zipCode to a provided value.
* +getZipCode(): int - Returns the value of zipCode.

|  |
| --- |
| **PlantingZone** |
| -zoneID: String  -zipCode: int  -zoneNumber: String |
| +getZoneID(): String  +getZoneNumber(): String |
| +getPlantingZoneSub(): String |

**Class: PlantingZone**

Attributes:

* -zoneID: String
* -zipCode: int
* -zoneNumber: String

Methods:

* +getZoneID(): String
* +getZoneNumber(): String
* +getPlantingZoneSub(): String

**API Classes**

Diagram

Description automatically generated

|  |
| --- |
| **auth** |
| -conn: Database |
| -suppliedUserName: String |
| -suppliedPassword: String |
|  |
|  |
| +getConn(): Database |
| -getUser(): user |
| -authenticate(): boolean |
|  |

**Class: auth**

Attributes:

* -conn: Database – PDO Connection from PHP to mySQL Database
* -suppliedUserName: String – Username supplied to api for authentication
* -suppliedPassword: String – Password supplied to api for authentication
* -pepper: String

Methods:

* +getConn(): Database – Returns active instance of Database class provided during constructor
* -getUser(): user – Database query for all user information, used for comparing provided to actual in authentication
* -authenticate(): Boolean – Compare contents of getUser() with attributes above to verify user can log in

|  |
| --- |
| **database** |
| -conn: Database |
| -dbName: String |
|  |
| +connect(): Database |
| +createDatabase(): Database |
|  |

**Class: database**

Attributes:

* -conn: Database – PDO Connection from PHP to mySQL Database
* -dbName: String – Name of system database to make connection with

Methods:

* +connect(): Database – Connects to provided dbName with credentials provided inside function. This connection allows for reading and writing to tables that were previously created. Returns a successful PDO connection for use in php.
* +createDatabase(): Database – Function used for rapid deployment of plant database wipes and recreates data in case of emergency or for testing purposes.

|  |
| --- |
| plant |
| -conn: Database |
| #tableName: String |
|  |
| +getConn(): Database }  +read(): json  -growingrelationshipAssoc(): array  -instructionsAssoc(): array  -instructionsQuery(): string  -zoneidlookup(): array |
|  |

**Class: plant**

Attributes:

* -conn: Database – PDO Connection from PHP to mySQL Database
* #tableName: String – Name of table accessed by this class

Methods:

* +getConn(): Database - Returns currently available DBO database connection
* +read(): json - Returns a array mutated into JSON to be returned by the API to the requestor
* -growingrelationshipAssoc(): array – Internal function used to specialize return data when URL paramaterss are provided
* -instructionsAssoc(): array – Internal function used to specialize return data when URL paramaterss are provided
* -instructionsQuery(): string – Internal function used to construct a specialized PDO query to be used by the above functions
* -zoneidlookup(): array – Internal function that returns an array object that represents a planting zone when a zip code is provided.

|  |
| --- |
| plantgrowingrelationship |
| -conn: Database |
| #tableName: String |
|  |
| +getConn(): Database }  +read(): json |
|  |

**Class: plantgrowingrelationship**

Attributes:

* -conn: Database – PDO Connection from PHP to mySQL Database
* #tableName: String – Name of table accessed by this class

Methods:

* +getConn(): Database - Returns currently available DBO database connection
* +read(): json - Returns a array mutated into JSON to be returned by the API to the requestor

|  |
| --- |
| plantinginstructions |
| -conn: Database |
| #tableName: String |
|  |
| +getConn(): Database }  +read(): json |
|  |

**Class: plantinginstructions**

Attributes:

* -conn: Database – PDO Connection from PHP to mySQL Database
* #tableName: String – Name of table accessed by this class

Methods:

* +getConn(): Database - Returns currently available DBO database connection
* +read(): json - Returns a array mutated into JSON to be returned by the API to the requestor

|  |
| --- |
| plantingzone |
| -conn: Database |
| #tableName: String |
|  |
| +getConn(): Database }  +read(): json  -zipCodeAssoc(): Array |
|  |

**Class: plantingzone**

Attributes:

* -conn: Database – PDO Connection from PHP to mySQL Database
* #tableName: String – Name of table accessed by this class

Methods:

* +getConn(): Database - Returns currently available DBO database connection
* +read(): json - Returns a array mutated into JSON to be returned by the API to the requestor
* -zipCodeAssoc(): Array – Internal function used with read() to customize output to include a list of all zipcodes associated with a provided zone

|  |
| --- |
| users |
| -conn: Database |
| #tableName: String |
|  |
| +getConn(): Database }  +read(): json  +create(): json  +update(): json  +delete(): json |
|  |

**Class: users**

Attributes:

* -conn: Database – PDO Connection from PHP to mySQL Database
* #tableName: String – Name of primary table accessed by this class

Methods:

* +getConn(): Database - Returns currently available DBO database connection
* +read(): json - Returns a array mutated into JSON to be returned by the API to the requestor
* +create(): json – Receives a POST call through API with a verifiable JSON object, creates the new user and returns a copy of the information provided with a success code.
* +update(): json - receives a PUT call through API with a verifiable JSON object, updates the existing user and returns a copy of the information provided with a success code. This requires a user UUID be provided to match user.
* +delete(): json – receives a DELETE call through API with one or many user UUID and deletes the provided user from the database.

Note: All of the above endpoints REQUIRE authentication to function. Additionally, singular users table allows access to userData table since users and userData are always one to one.

|  |
| --- |
| zipcode |
| -conn: Database |
| #tableName: String |
|  |
| +getConn(): Database }  +read(): json  -zoneAssoc(): json |
|  |

**Class: plantinginstructions**

Attributes:

* -conn: Database – PDO Connection from PHP to mySQL Database
* #tableName: String – Name of table accessed by this class

Methods:

* +getConn(): Database - Returns currently available DBO database connection
* +read(): json - Returns a array mutated into JSON to be returned by the API to the requestor. Due to the amount of zip codes in the continental United States, this function requires that one specific zip code be provided via URL parameter. Otherwise, this function returns nothing.
* -zoneAssoc(): json – Internal function utilized by read() to return specific zone information when a specific zipcode URL param is provided.

### Traceability Matrix

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Classes** | | | | | | | | |
| **Domain Concepts** | **PW** | API Classes | PlantCalc | Plants | Plant  Growing  RelationShip | Planting Instructions | PlantingZone | User | UserAuth | ZipCode |
| Main Website | 3 |  |  | X |  |  |  | X |  | X |
| User Account | 4 |  |  | X |  |  |  | X | X | X |
| User Storage | 2 | X |  | X |  |  |  |  |  |  |
| Plant Retrieval | 6 |  | X | X | X | X | X | X |  |  |
| Plant Storage | 4 | X |  | X |  | X | X |  |  |  |
| Zone System | 3 |  |  | X |  | X | X |  |  |  |

Main Website:

* Gets and Displays plants from the Plant class
* Gets and Displays User Data from the user class, and userID class
* Gets and Displays Zip Codes from the ZipCode Class

User Account:

* Retrieves User Account info From User Account Class
* Authenticates User from User Auth Class
* Gets user zipcode from UserClass
* Get Plant IDs from UserClass

User Storage:

* Uses API Classes to get User Data from Storage
* Uses API to get user stored plants for plants class

Plant Retrieval:

* Gets Plants from Plants Class
* Gets Calculations from PlantCalc Class
* Gets Plant growing RelationShips
* Gets Planting instructions from PlantingInstructionsClass
* Gets Zone data from zone class
* Gets User Plants from the User Class

Plant Storage

* Stores Plant Data used in plant class using API class
* Stores Planting Data used in PlantingInstructionsClass using API class
* Stores Zone data In Zone class using API class

Zone System

* Used to determine planting with plantinginstructions class
* Determines types of plants using plants class
* Determines plant zone information using zone class

### Design Patterns

The best design pattern for improving the class diagram and Interface specification is the publisher subscriber pattern. Decentralizing control and allowing control to pass object to object will improve the overall design and performance. It will also allow for the reusing of our objects for purposes other than just a plant database. For example, the plant search could easily be adapted to a book search, or other similar search.

In our use cases the objects are calling other objects or servers objects and do not know what is happening and do not need to know what is happening. There is also a sharing of data across several different objects. Such as the user object will share data with the plant object and visa a versa. The publisher subscriber model allows the separation of concerns.

The publisher subscriber model will decentralize the plant search which currently handles a larger number of tasks and provides data to a number of other objects. Further, since our plant search pulls data from two different areas, it will do well as a subscriber with the publisher sending the data instead of requiring completely separate calls.

### OCL Contracts

**MainScreen:**

Context MainScreen::clickLogin

Invariant: loginButton

Post-conditional: login.html

Context MainScreen::clickSignUp

Invariant: signUpButton

Post-conditional: sign-up.html

Context MainScreen::searchZipCode

Pre-conditional: zipcode = zipcode.getZipcode()

Post-conditional: return getZipcodes(zipcode)

Context MainScreen::searchPlant

Pre-conditional: plantName = plant.getName()

Post-conditional: return getPlants(plantName)

Pre-conditional: plantType = plant.getType();

Post-conditional: return getPlants(plantType)

Context MainScreen::clickAccountInfo

Invariant: accountInfoButton

Post-conditional: userInfo.html

**LoginActivity:**

Context LoginActivity::passwordChecked:boolean

Invariant: auth, loginButton, signupLink

Pre-conditional: username = enteredUsername

Pre-conditional: password = enteredPass

Post-conditional: return validateUser

**SignupActivity:**

Context: SignupActivity::signup

Pre-conditional: firstname = firstname

Pre-conditional: lastname = lastname

Pre-conditional: username = enteredUsername

Pre-conditional: password = enteredPass

Post-conditional: onSignUpSuccess()

Post-conditional: onSignUpFailed()

Context: SignupActivity::onSignUpSuccess

Invariant: signupButton

Pre-conditional: If sign up success

Post-conditional: setResult(RESULT\_OK, null)

Context: SignupActivity::onSignUp

Invariant: signupButton

Pre-conditional: If sign up failed

Post-conditional: Alert.alert(‘Sign up failed’)

Context: SignupActivity::checked:boolean

Invariant: firstname, lastname, username, password

Pre-conditional: firstname.isEmpty()

Pre-conditional: lastname.isEmpty()

Pre-conditional: username.isEmpty() || username == existingUsername

Pre-conditional: password.isEmpty()

Post-conditional: return validateUser

# System Architecture and System Design

### Identifying Subsystems

Our application will consist of several subsystems:

**User Interface Layer:**

User Data Enter, update – This system will take in user data that allows them to create, delete or update information.

New Account Entry – This system will be used for creating a new account.

Plant Search, Display – This system will allow the user to search for and will be in charge of displaying the plant info.

User Plant add/ delete – This system will allow the user to add plants to their profile, and delete plants from their profile.

Zone system search / Display – This system will get allow the user to enter a zone and display zone information

**Application Layer:**

Data Validation – This system will validate the user's entered information is correct, and will make sure the data returned from the database is correct

Data Calculation – This system will perform the calculations needed for the users displayed information such as germination dates.

Data Verification – This system will verify the user data against the database.

REST API – This system will communicate with requested information with the database, and receive the information from the database.

User Authentication – This will be responsible for password and user ID verification

**Database Layer:**

Plant DB – This database will contain all the searchable plant and zone information. It will be retrieve only.

User Information DB – This database will store all the user information. It will allow the creation of user accounts, the retrieval of user information, the updating of user accounts and the deletion of user accounts.

Graphical user interface, application

Description automatically generated

### Architecture Styles

For our application, we are using the layered architectural style along with client-server, separating the components into three main layers. We will have the user interface layer, the application layers, and the database layer.

The user interface layer will include all the user interface functionality. The layer will be responsible for collecting and displaying all information required to and from the user. It will allow the user to enter their login information, create an account, look at the list of plants, add plants to their planner, and view the data related to their plantings.

The application layer will be responsible for handling all data sending and requests using the Representational state transfer (REST) system. It will handle all requests to and from the database. The application will request user, or plant information through the REST API and the Database will send the required information back through the REST API. The application layer will be responsible for checking for appropriate data, authenticating the user info, and sending the appropriate data to the interface layer.

The database layer will handle all the data storage. We will follow the CRUD system for the database. The data backend will use MySQL along with PHP. There will be two separate storage systems. One will be static and will not allow user updates. It will be for retrieval only. This system will work whether the user has an account or not.

The other database system will store user data such as passwords, user chosen plant data, and user information and user history. The second system will allow for creation, updates and deletion of user information and will only be accessible with an account.

The main system will run in a client server pattern, with the client sending and receive the data while the server receives and processes the data. Multiple clients may connect to the server and use the database at once in a many to one relationship.

### Mapping Subsystems to Hardware

The database system will be mapped server-side and will be hosted on Amazon Web Services and will require SQL and PHP support. Since AWS is scalable, we will use the minimum DB storage allowable, but with possibility of expanding further into the future. Both Databases will be stored in the same server space. However, the DB system can be hosted on any server that allows for SQL and PHP.

The application layer will be hosted partially server-side on the webserver and partially on the client-side. The REST API will be hosted on the webserver as will the user authentication system. Both will be written in PHP. The data verification, data validation and data calculation will be mapped to the client-side browser using JavaScript

The User interface will be mapped to the client-side browser and will be supported by all browsers, both desktop and mobile. It will use CSS along with HTML to create the interface. All services will require a data connection between the client and the server.

### d. Connectors and Network Protocols

Not applicable.

### e. Global Control Flow

#### Execution Orderness

This program runs in a linear execution style. The order of the events will differ depending on the user. First, the user can either create an account, sign in, or continue as a guest. If the user is creating an account, they will enter their information, create a valid login, and enter their zip code to save to their account. The options after this if they are only a guest is they can search plants by their name and see the information on the plants, filter plants by a zip code to see hardiness zones and information on the plants in those zones and enter planting dates for plants to see the expected germination and harvest dates. Users who are signed in can also save their planting dates, design a garden box, take notes on their personal garden already created, update user information, and disable user account.

#### Time Dependency

The program will not have any timers in the application and does not need any physical time down to the hour. It will use a calendar to select the date that it was planted and to calculate the germination dates and harvest dates. It will need to take in the current date to use when selecting the planting date.

### f. Hardware Requirements

This application will require connection to either Wi-Fi or mobile data to run. This program will be able to be run on any device that can use a browser of some kind. It will require a display of some kind as well.

# Algorithms and Data Structures

* 1. Algorithms  
     Not Applicable.
  2. Data Structures  
     This project will use a JS arrayList <PLANT> to store a list of plant objects that will be returned from the database to the user interface.
  3. Concurrency:

Not Applicable.

# User Interface Design and Implementation

To maintain an easy-to-use interface, the colors, text, and buttons have been kept simple and streamlined. Additionally, all core functionality are easily available for users.

At this time, only minimal changes have been made to the initial screen mock-ups from Report 1. The most significant change here was the development of the navigation bar, which will include search bars that filter results from the plant database by zipcode or plant name. Making the search bars available on the navigation bars ensures that they are easily accessible on all pages, therefore reducing the user effort to use this core functionality.

Additionally, the user account information is available on the navigation bar to ensure users can log in/out on every page, as well as easily access user-specific information while logged in.

## Initial User Interface Requirements Diagrams

REQ-21

Graphical user interface

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generated

REQ-22

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

REQ-23

A picture containing graphical user interface

Description automatically generatedA picture containing text

Description automatically generated

REQ-24

Graphical user interface, text, application, chat or text message

Description automatically generatedGraphical user interface, text, application

Description automatically generated

REQ-25

Graphical user interface, text, application, chat or text message

Description automatically generatedA screenshot of a phone

Description automatically generated with medium confidence

## User Interface Specification

### Preliminary/Final Designs

#### UC-1: Create a User Account

Step 1: Select “Sign Up” from home page.

Graphical user interface, text, application

Description automatically generated

Final Design:

Graphical user interface, text, application, email

Description automatically generated

Step 2: Type in First Name, Last Name, User Name, and Password. (Changed from: Type in Name, Email, Password, and Zip Code.)

Graphical user interface

Description automatically generated with medium confidence

Final Design:

Graphical user interface, text, application, email

Description automatically generated

Step 3: Click “Sign Up”.

Graphical user interface, text, application, chat or text message

Description automatically generated

Final Design:

Graphical user interface, text, application

Description automatically generated

Step 4 (option zip code is valid- System displays Account creation Confirmation.)

Graphical user interface, text, application

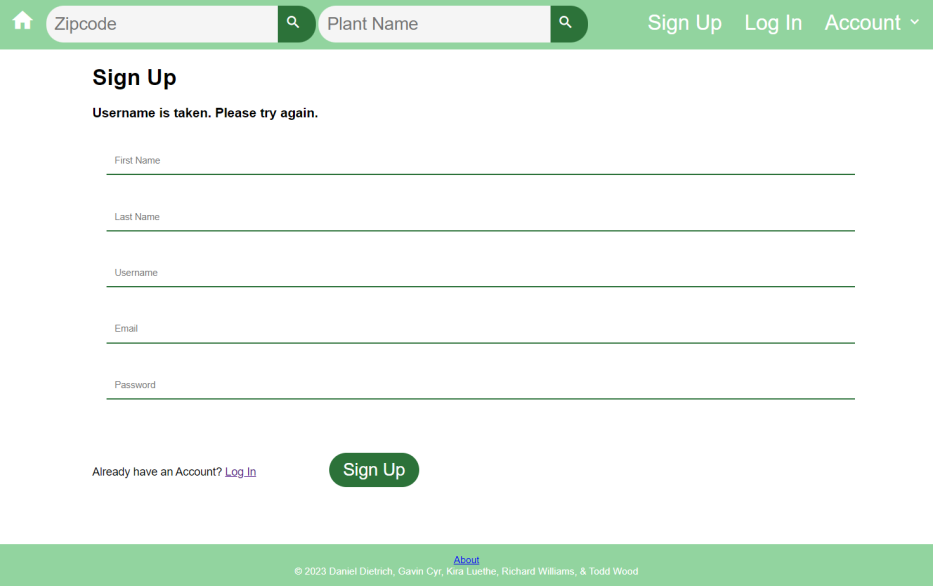
Description automatically generated

Step 4 (opt: invalid input): System warns user that input is invalid. Go back to Step 2.

Graphical user interface, text, application, chat or text message

Description automatically generated

Final Design:



#### UC-3: Filter by Zip Code

Step 1: User types zip code into zip code search box

Graphical user interface, text, application

Description automatically generated

Final Design:

Graphical user interface, text, application, email

Description automatically generated

Step 2: User Clicks Search button or presses “Enter”

Graphical user interface, text, application, email

Description automatically generated

Final Design:

Graphical user interface, text, application

Description automatically generated

Step 3 (opt: valid zip code): System displays Search results

Graphical user interface, text, application

Description automatically generated

Final Design:

Graphical user interface, text, application

Description automatically generated

Step 3 (opt: invalid zip code): System warns user that Zip Code is invalid. Go back to Step 1.

Graphical user interface, text, application, chat or text message

Description automatically generated

Final Design:

Graphical user interface, text, application, email

Description automatically generated

#### 

#### UC-4: Search by Plant

Step 1: User types query into Plant name search box.

Graphical user interface, text, application, chat or text message

Description automatically generated

Final Design:

Graphical user interface, text, application, email

Description automatically generated

Step 2: User Clicks Search button or presses “Enter”.

Graphical user interface, text, application, chat or text message

Description automatically generated

Final Design:

Graphical user interface, text, application, email

Description automatically generated

Step 3 (opt: query returns results): System displays Search results.

Graphical user interface, text, application

Description automatically generated

Final Design:

Graphical user interface, text, application

Description automatically generated

Step 3 (opt: query does not return results: System notifies User that no results were found.)

Graphical user interface, text, application

Description automatically generated

Final Design:

Graphical user interface, text, application, email

Description automatically generated

#### 

#### UC-5: Enter Planting Date

(Final Design: Not yet implemented)

Step 1: On a search result page, select “add” button next to desired plant.

Graphical user interface, text, application

Description automatically generated

Step 2: Enter correct planting date for selected plant.

Graphical user interface, application

Description automatically generated

Step 3: Save Planting Date

Graphical user interface, text, application, chat or text message

Description automatically generated

Step 4: New planting has been added to “Garden Management” Page

Graphical user interface, text, application, chat or text message

Description automatically generated

# Design of Tests

### Test cases

|  |  |
| --- | --- |
| **Test Case Identifier:** TC-1  **Use Case Tested:** UC-1  **Pass/Fail Criteria:** The test will pass if the user is able to successfully create an account if all entered values are valid. If the values are not valid, the system will notify the user and allow them to fix these values. The test will fail if the user is unable to create and account using valid credentials or if the account is created using invalid values.  **Input Data:** userName, email, password, firstName, lastName | |
| **Test Procedure:** | **Expected Result:** |
| Step 1: User enters username of preexisting account. | The API reports back that an account with that Username already exists and propmts the user to enter different user information and try again. |
| Step 2: User enters username, email, password and valid zipCode. | The API reports back that a new User class entry is created using the provided data and is stored in the database. The system then logs the user in using the provided credentials and the User Information page is displayed for the newly created user class. |

|  |  |
| --- | --- |
| **Test Case Identifier:** TC-3  **Use Case Tested:** UC-3  **Pass/Fail Criteria:** The test will pass if the user can successfully filter the data by entering a valid zipCode value. If the value is invalid, the system notifies the user of the invalid entry and allows the user to enter a different value. The test will fail if the user is unable to filter the data by entering a valid ZipCode.  **Input Data:** zipCode | |
| **Test Procedure:** | **Expected Result:** |
| Step 1: The user enters an invalid zipCode. | The API reports back that the Hardiness zone could not be determined due to an invalid zipCode entry and the System displays a message stating that the zipCode value is invalid. |
| Step 2: The user enters a valid zipCode. | The API reports back with the proper Hardiness zone determined from the entered zipCode. The entered zipCode is stored in the user data in the Database if the user is logged in the session cookies if the user is not logged in. |

|  |  |
| --- | --- |
| **Test Case Identifier:** TC-4  **Use Case Tested:** UC-4  **Pass/Fail Criteria:** The test will pass if the user can successfully search for data by entering a valid plantName value. If the value is invalid, the system notifies the user of the invalid entry and allows the user to enter a different value. The test will fail if the user is unable to search for data by entering a valid plantName value.  **Input Data:** plantName | |
| **Test Procedure:** | **Expected Result:** |
| Step 1: The user enters an invalid  plantName. | The API reports back that data could not be retrieved for the entered plantName. The System notifies the user that data could not be retrieved for the given plantName and to ensure that a valid plantName was entered. |
| Step2: The user enters a valid plantName. | The API retrieves the relevant plant data from the Database using the entered plantName and returns the data. The System then displays this data to the user. |

|  |  |
| --- | --- |
| **Test Case Identifier:** TC-5  **Use Case Tested:** UC-5  **Pass/Fail Criteria:** The test will pass if the user can successfully add a plant entry to their plantData by entering a valid plantName and plantingDate. If either value is invalid, the system notifies the user and allows them to enter different values.  **Input Data:** plantName, plantingDate | |
| **Test Procedure:** | **Expected Result:** |
| Step 1: The user enters an invalid plantName and an invalid plantingDate. | The API reports back that data could not be retrieved for the entered plantName. The System notifies the user that data could not be retrieved for the given plantName and to ensure that a valid plantName was entered. |
| Step 2: The user enters an invalid plantName and a valid planting Date. | The API reports back that data could not be retrieved for the entered plantName. The System notifies the user that data could not be retrieved for the given plantName and to ensure that a valid plantName was entered. |
| Step 3: The user enters a valid plantName and an invalid plantingDate. | The System notifies the user that the plantingDate entered is not valid and check the date entered. |
| Step 4: The users enters a valid plantName and a valid plantingDate. | The API searches the database for information based on the entered plantName and returns the data. The Calc class calculates the Germination date and Harvest date based on the entered plantingDate. The System adds the relevant plant data to the appropriate User plantData in the Database. The System then displays the Garden Management page with all added plants for the current user along with accompanying dates. |

|  |  |
| --- | --- |
| **Test Case Identifier:** TC-7  **Use Case Tested:** UC-7  **Pass/Fail Criteria:** The test will pass if the user is able to log in and their information is retrieved. The test will pass if the user logs in using valid credentials. If the credentials are invalid, the user is not signed in. The test will fail if the user is unable to log in using valid credentials or is able to access data using invalid credentials.  **Input Data:** Username and Password | |
| **Test Procedure:** | **Expected Result:** |
| Step 1: The user enters an invalid username/password combination. | The API reports back that the user credential combination is invalid. The login screen is displayed along with a message to the user that the credentials were invalid and to please try again. |
| Step 2: The user enters a valid username/password combination. | The API reports that user authentication was successful. Session variables for the user’s Username and User ID are set and the user is directed to the User Information page. |

|  |  |
| --- | --- |
| **Test Case Identifier:** TC-8  **Use Case Tested:** UC-8  **Pass/Fail Criteria:** The test will pass if the user successfully resets their password. The test will pass if the password does not change due to being the same as the previous password. The test will pass if the user cancels the reset password. The test will fail if the user is able to change to the same password. The test will fail if the password is not reset despite valid entry.  **Input Data:** Email Address, New Password | |
| **Test Procedure:** | **Expected Result:** |
| Step 1: User clicks on forgot password link.    Step 2: User enters a new password          Step 4: Password is changed in the DB.    Step 4: Password changed displayed to user | Users will be re-directed to a new page to enter their email and a reset password dialog appears.    User enters a new password. If the password is the same, a dialog telling the user to choose a different password shows. If the password does not meet criteria a dialog showing criteria shows. If the password is valid the system will accept.    The new password is stored in the db.    Password Changed! Message will show to the user. |

|  |  |
| --- | --- |
| **Test Case Identifier:** TC-9  **Use Case Tested:** UC-9  **Pass/Fail Criteria:** Test will pass if user does or doesn’t update information. Test will fail if user updates and the updated information is not stored. Test will fail if the area to update information does not appear.  **Input Data:** User Information to be changed | |
| **Test Procedure:** | **Expected Result:** |
| Step 1: User Login  Step 2: User information retrieved  Step 3: User chooses Update information button    Step 4: User enters updated information    Step 5: User clicks save    Step 6: Information is written to db and shown to user. | Tested in another test  Tested in another test    Button will take user to a page to edit information      Boxes allowing the editing of information will allow the user to make changes    Save button locks in the user changes    DB writes the changes, and sends the new information back. |

|  |  |
| --- | --- |
| **Test Case Identifier:** TC-10  **Use Case Tested:** UC-10  **Pass/Fail Criteria:** Test will pass if user account no longer functions. The test will fail if the account still functions. The test will fail if the account is deleted. Test will fail if the account is disabled even if the phrase is not exact.  **Input Data:** Confirmation message for disabling account | |
| **Test Procedure:** | **Expected Result:** |
| Step 1: User Login  Step 2: User information retrieved  Step 3: User chooses Disable Account Button    Step 4: Dialog appears requiring the user to type in “Disable My Account”      Step 5: User Clicks Accept after typing phrase exactly.    Step 6: DB sets disabled flag.    Step 7: System no longer displays user information | Tested in another test  Tested in another test    Button will take user to a page to disable account      Box appears requiring exact phrase to be typed.        If Phrase exact, then move to disable. If not typed exactly as shown. warn user the phrase is not correct, please try again.    System sets db flag disabled.    User information is no longer visible and account usage is no longer possible. |

### Test Coverage

All of our test cases cover the essential classes that are necessary to the operation of the Garden App. We will adjust and add test cases as we need to as we develop more of our classes and methods. Testing will be done for as many possible cases that a class could go through. Test procedures will have a format of both a pass procedure and a fail procedure in response to a user input. If tests fail because of a invalid user input, the application will ask the user to enter a valid input. One example of testing would be the login screen, where if the user inputs an invalid username and/or password, the application will tell them the username or password is invalid and ask them to enter their information again. The fail case will repeat until the user inputs their correct information, at which point the user can login and access their information in their account. There are test cases for entering zip codes, writing notes, searching plant names, and entering plant dates as well. Our test cases will be specific so that it can cover all possible cases. We want our application to run as smoothly as possible with no errors which is why we have test cases to show that our Garden App works properly and has been tested thoroughly.

### Integration Testing

We are using the strategy of bottom-up testing for our application. This is an approach to integrate testing that tests the lowest level components first and then use those components to help and test the higher-level components. It will ensure that the building blocks of code operate as needed before we use them in other sections of our code. This will also allow us to test what we are working on as we complete parts of the application. This approach fits our project and makes it easier to find the bugs of the project as we test to find whether it is coming from the integration of the code components or if the bug is coming from how the classes are designed and coded. By understanding the relationships between the objects in the system, the bottom-up testing approach is more efficient and straightforward in that you can quickly narrow down where the problem lies. An example of this would be testing the garden box feature in the application. First you must pass the test that allows you to login and save your information, then you can test whether you can draw and create your own personal garden box in the application. After going through the individual functions, we test the features which call for interactions between more than one object or class.

# History of Work, Current Status, and Future Work

### Merging the Contributions from Individual Team Members

To ensure uniform formatting and appearance, this project used Microsoft Word’s sharing feature. This enabled each group member to simultaneously access the same file, and it updated changes in real-time. This file sharing allowed members to instantly view all work completed by other members and to format their own work accordingly, as well as ensuring consistency in ideas and wording.

Additionally, this file sharing allowed users to make comments outside of the body of the document, which further fostered communication between members, and allowed the group to clearly track ownership of each section.

The text formatting presented the greatest challenge, as Microsoft Word’s heading styles can be finnicky to work with. To ensure uniformity, all text formatting was handled by the group leader, who hosted the report file.

### History of Work

Graphical user interface

Description automatically generated

The developer team completed documentation for this project according to the course due dates: Project Proposal on Feb 6, Report 1, Part 1 on Feb 17, Report 1 Final on Feb 23, Report 2, Part 1 on March 1, and Report 2 Final on March 15. During this time, Richard “Matt” developed a usable database and API with endpoints.

After the completion of Report 2, the rest of the team worked on creating basic app functionality in preparation for the Demo 1 due date of April 5. Gavin worked on data retrieval related to the zip code endpoint (UC-3 Filter by Zip Code), Todd worked on data retrieval related to the plants and planting instructions endpoints (UC-4: Search by Plant), Dan implemented user authentication features (UC-1 Create a User Account, UC-7 Log-In), and Kira created an attractive user interface.

Key Accomplishments:

1. Constructed Database
2. Created API with functional endpoints
3. Set up User Authentication
4. Pull and display zip code data
5. Pull and display plant data
6. Implemented a simple and clean user interface

Currently, the team is revising and completing additional documentation for Report 3.

After Report 3, the team will add additional functionality to the app by Demo 2 on May 3. Gavin and Todd will work together to implement use case UC-5: Enter Planting Date, which involves allowing users to store data about which plants they have planted, and the date planted, as well as calculating date ranges for harvest. Dan and Richard “Matt” will work on use cases UC-9: Update User Information, UC-10: Disable User Account, and UC-8 Password Reset. Kira will improve the user interface by adding mobile responsiveness and creating interfaces for account data and for adding user’s plant data.

The following use cases will be reserved for possible future functionality of the Garden App: UC-6 Design Garden Box, UC-2: Take Note.

### Breakdown of Responsibilities

* + 1. List the names of modules and classes that each team member is currently responsible for developing, coding, and testing

Design of Tests are yet to be completed. This planning phase is scheduled to occur between the completion of Report 3 Final and as part of the submission of Demo 2.

* + 1. Who will coordinate the integration?

As additional features are added, the integration will be coordinated by team leader Kira Luethe, as the project’s GitHub is hosted on her account. All members will continue to utilize git version control to ensure smooth branching and merging.

* + 1. Who will perform and integration testing? (The assumption is that the unit testing will be done for each unit by the student who developed that unit.)

Each group member will continue to be responsible for their own unit testing on the code that they develop. Responsibilities for integration testing will be assigned when Design of Tests has been completed, at the submission of Demo 2.

# 14. References

**Fellow student report links:**

<https://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2019-g13-report3.pdf>

<https://www.ece.rutgers.edu/~marsic/books/SE/projects/ParkingLot/2019f-g4-report3.pdf>

**API/Datasets Used to populate Database:**

<https://rapidapi.com/aptitudeapps/api/usda-plant-hardiness-zones/details>

<https://sage.nelson.wisc.edu/data-and-models/datasets/crop-calendar-dataset/>