A picture containing plant, green, leaf, vegetable

Description automatically generated

**Garden App**

**Report 2**

**CSCI 441**

**GitHub URL:** [**https://github.com/Kirapants07/CSCI441\_VA\_Group1\_Spring2023\_GardenApp**](https://github.com/Kirapants07/CSCI441_VA_Group1_Spring2023_GardenApp)

**March 15, 2023**

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

# Individual Contributions Breakdown

## Responsibility Matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Task** | **Daniel** | **Gavin** | **Kira** | **Richard** | **Todd** |
| Section 4 Interaction Diagrams, Classes: (User & UserAuth), Test Case Designs 1, 2, 3, 4, 5 | 100% |  |  |  |  |
| Section 1b. System Operation Contracts, Classes: Zipcode, PlantingZone, Test Coverage, Integration Testing |  | 100% |  |  |  |
| Section 3 (parts a, b, and d), User Interface Design and Implementation, Project Management and Team Leader |  |  | 100% |  |  |
| Classes: All API Classes and diagram, Section 2 Data Model and Persistent Data Storage |  |  |  | 100% |  |
| Section 1a. Conceptual Model (parts i,ii,iii, and iv), Classes: Plants, PlantCalcs, Traceability Matrix, Class diagram, Test Cases 6,7,8,9,10 |  |  |  |  | 100% |

# Table of Contents

[Individual Contributions Breakdown 2](#_Toc1957386803)

[Responsibility Matrix 3](#_Toc390572220)

[Table of Contents 3](#_Toc934986032)

[Work Assignment 4](#_Toc1173261047)

[a. Team Profile (Members & Strengths) 5](#_Toc2046632082)

[b. Sub-Teams 5](#_Toc830470681)

[Part 1: 5](#_Toc1136238039)

[1. Analysis and Domain Modeling 6](#_Toc228998773)

[a. Conceptual Model 6](#_Toc365042135)

[I. CONCEPT DEFINITIONS 7](#_Toc944900423)

[II. ASSOCIATION DEFINITIONS 8](#_Toc1263857697)

[III. ATTRIBUTE DEFINITIONS 9](#_Toc2038298789)

[IV. TRACEABILITY MATRIX 9](#_Toc1954352203)

[b. System Operation Contracts 10](#_Toc1339101154)

[2. Data Model and Persistent Data Storage 13](#_Toc1177967346)

[3. Mathematical Model 14](#_Toc489025927)

[Part 2: 14](#_Toc1630405390)

[1. Interaction Diagrams 15](#_Toc163034075)

[2. Class Diagram and Interface Specification 16](#_Toc144538884)

[a. Class Diagram 16](#_Toc1442888990)

[b. Data Types and Operation Signatures 16](#_Toc1579528682)

[c. Traceability Matrix 16](#_Toc196443119)

[3. System Architecture 16](#_Toc1875009081)

[a. Identifying Subsystems 16](#_Toc106245522)

[b. Architecture Styles 16](#_Toc1957380441)

[c. Mapping Subsystems to Hardware 16](#_Toc1265373089)

[d. Connectors and Network Protocols 16](#_Toc898807114)

[e. Global Control Flow 16](#_Toc1492025675)

[i. Execution Orderness 16](#_Toc97354512)

[ii. Time Dependency 16](#_Toc749668173)

[f. Hardware Requirements 16](#_Toc584023780)

[Part 3: 16](#_Toc436453931)

[3. Algorithms and Data Structures 16](#_Toc909287270)

[a. Algorithms 16](#_Toc1230644970)

[b. Data Structures 16](#_Toc985415955)

[c. Concurrency: 16](#_Toc1616126619)

[4. User Interface Design and Implementation 16](#_Toc257053686)

[5. Design of Tests 16](#_Toc1034612807)

[a. Test cases 16](#_Toc1893853324)

[b. Test Coverage 16](#_Toc55910635)

[c. Integration Testing 16](#_Toc708507372)

[4. Project Management and Plan of Work 16](#_Toc1781005858)

[a. Merging the Contributions from Individual Team Members 16](#_Toc1653170827)

[b. Project Coordination and Progress Report 16](#_Toc1811628742)

[d. Plan of Work 16](#_Toc1525919893)

[e. Breakdown of Responsibilities 16](#_Toc1750873796)

[5. References 16](#_Toc935022100)

# Work Assignment

### Team Profile (Members & Strengths)

Daniel Dietrich has 3.5 years of collegiate education in the field of computer science. His programming experience includes PHP, SQL, JavaScript, HTML, CSS, Java, and C++. Past projects include construction of Device Inventory forms and an API using PHP, JavaScript, SQL and HTML.

Gavin Cyr has 3 years of coding experience in multiple coding languages through classes for a bachelor’s degree in computer science. He has coded multiple different projects as a student using the languages HTML, JavaScript, CSS, PHP, SQL, Java, and C++.

Kira Luethe has some experience coding in several languages, including JavaScript, PHP, Java, C++, CSS, SQL, and Python. She has built several web apps as a student, and she especially enjoys working on the graphic interface design of projects.

Richard Williams has 12 years of experience at a medical software company as a front-end development manager. Have 2 years of experience in designing and deploying a custom API for internal use. Bringing skills in HTML, CSS, JavaScript, PHP, MySQL, C#, AWS deployment, Java, and Agile Development Lifecycle.

Todd Wood has over 14 years' experience in management. Including over 5 years at the Director level. Additionally, he has nearly completed his BS in Computer Science from Fort Hays State University. Todd’s skills include programming and problem solving in C++, C, Java, Python, JavaScript, HTML and SQL. He has built several applications including an author’s webpage and a text-based memory game. Todd’s strengths are data calculation and manipulation.

### Sub-Teams

Work on this project has been divided into two teams, one for the front-end development of the website, and one for the back-end development of the API, database, and user authentication. The team lead, Kira Luethe, will organize communication between group members and the instructor, and ensure timely completion of project stages.

The first team, Daniel Dietrich and Richard Williams, will build the backend components, including the API, database design and implementation, and user authentication.

The second team, Gavin Cyr, Kira Luethe, and Todd Wood, will design and implement the website. This encompasses user interface and graphic design, and features, which include planting zone selection, and date trackers.

# Part 1:

## Analysis and Domain Modeling

### ***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: | * 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: | * 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. |

## Data Model and Persistent Data Storage

The backend system is comprised of two separate databases. The first is plantdata and the second is useradmin. Both databases are relational and created using MySQL and hosted through AWS EC2 Linux.  
  
The first is meant to be read-only data taken from sources provided containing relevant planting and zone data. An API will feed this data to the application in a read-only format. The API will also be available for public use, documentation to be provided on webpage.  
  
Graphical user interface, text, application

Description automatically generated  
  
The second table, useradmin, is meant for persistent storage of user data. This includes login credentials, user information and saved user parameters. Examples of user parameters are planting dates, search history, etc. Saved user parameters will be saved in a document style format in a XAML format. This will allow the application be flexible on what it stores while the application is further developed. Once initial development is complete this style may be fleshed out to be more specific to user parameters being utilized by the application. Application will be able to read and wrtite to this table via the API, user authorization will be included to ensure that this table is only editable by the application. This endpoint will not be included in the public API mentioned above for the plantdata table.  
  
Graphical user interface, text, application

Description automatically generated

## Mathematical Model

Not Applicable.

# Part 2:

## Interaction Diagrams

Table

Description automatically generated

Diagram

Description automatically generated with medium confidenceTable

Description automatically generated with medium confidenceA picture containing text

Description automatically generatedText

Description automatically generated with low confidenceA picture containing graphical user interface

Description automatically generatedA picture containing table

Description automatically generatedTable

Description automatically generated with medium confidenceA picture containing diagram

Description automatically generatedGraphical user interface

Description automatically generated

## Class Diagram and Interface Specification

* 1. Class Diagram

Diagram, schematic

Description automatically generated

* 1. 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.
  1. 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

## 3. System Architecture

### 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.

# Part 3:

## Algorithms and Data Structures

* 1. Algorithms  
     Not Applicable.
  2. Data Structures  
     This project will use a JS arrayList <PLANT> in order 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

The user interface implementation has not yet been completed. It is scheduled for completion between the submission of Report 2 Final (March 15, 2023) and the first Demo (April 5, 2023).

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.

## 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 an account is created using invalid values.  **Input Data:** userName, email, password, zipCode | |
| **Test Procedure:** | **Expected Result:** |
| Step 1: User enters username, email, password and 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, followed by the account creation page. |
| Step 2: User enters username, email, password and valid zipCode. | The API reports back with the proper Hardiness zone determined from the entered zipCode, 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 Garden Management page is displayed for the newly created user class. |

|  |  |
| --- | --- |
| **Test Case Identifier:** TC-2  **Use Case Tested:** UC-2  **Pass/Fail Criteria:** The test will pass if the user is able to add a note to their personal garden data using the Add Note feature. The test will fail if the user is unable to create a new note to their personal garden after entering valid text.  **Input Data:** Text data | |
| **Test Procedure:** | **Expected Result:** |
| Step 1: The user selects the “Add Note” option without entering any text. | The System notifies the user that text is required, and no changes are made to the user data. |
| Step 2: The user enters text in the provided text field and selects “Add Note” option. | The System stores the new note in the data for the User class in the Database. The System displays an updated Garden Management page that includes the newly added note. |

|  |  |
| --- | --- |
| **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 in 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 reports 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-6  **Use Case Tested:** UC-6  **Pass/Fail Criteria:** The test will pass if the Gardenbox is successfully drawn with all plants in the desired location. The test will fail in any case where the Gardenbox is not drawn, or the required elements are not in their desired location  **Input Data:** From Class User – User plant information. From Class Plant – Plant Pictures | |
| **Test Procedure:** | **Expected Result:** |
| Step 1: User logs into their account    Step 2: The user selects the Gardenbox button (future feature) | Will be tested in another test    Button will send the request to the function that draws box (future feature) |
| Step 3: Gardenbox is drawn | Box is drawn with plantings in appropriate spaces |

|  |  |
| --- | --- |
| **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 enters the wrong username or PW. The test will fail if the login works with bad input. The test will fail if the login does not work with good input.  **Input Data:** Username and Password | |
| **Test Procedure:** | **Expected Result:** |
| User Enters Username and Password | If username is not valid, message returned USERNAME or PASSWORD INVALID  If Password is not valid message returned USERNAME or PASSWORD INVALID  If both are valid, display user account with plants and user information |

|  |  |
| --- | --- |
| **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 passwords. 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 update does not store. 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 but can still be logged into. The test will fail if the account still functions. The test will fail if the account is deleted. The test will fail if the account can no longer be logged into. 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.

## Project Management and Plan of Work

### ***Merging the Contributions from Individual Team Members***

To ensure uniform formatting and appearance, this project uses Microsoft Word’s sharing feature. This enables each group member to simultaneously access the same file, and it updates changes in real-time. This file sharing allows 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 allows users to make comments outside of the body of the document, which further fosters communication between members, and allows the group to clearly track ownership of each section.

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

### ***Project Coordination and Progress Report***

At the time of this report, the group has not yet implemented the front-end capabilities for the Garden App, as the planning and creation of Reports 1 and 2 have not allowed time. This coding is scheduled to begin following the completion of Report 2, before Demo 1. At that time, the front-end sub team (Gavin, Kira, and Todd) will focus on implementing the following use cases: UC-3 Filter by Zip Code, UC-4: Search by Plant, and UC-5: Enter Planting Date. The following use case is not required for minimum functionality and will be implemented after demo 1: UC-2: Take Note.

Use cases involving user authentication will be implemented by the front-end team in conjunction with the back-end sub team (Dan and Richard). These use cases include UC-1 Create a User Account, UC-7 Log-In, UC-8 Password Reset. Two related use cases will be implemented after the first demo, as they are not required for minimum functionality: UC-9: Update User Information and UC-10: Disable User Account.

The following use case will be reserved for possible future functionality of the Garden App: UC-6 Design Garden Box.

The back-end capabilities created by Richard are already functional. These include a database that stores plant data populated by freely available datasets, and an API with endpoints that include “/zipcode”, “/plantingzone”, “/plant”, “/plantgrowingrelationship”, and “/plantinginstructions”. Although it was not developed into a use case, the API provides future functionality for providing companion planting guides to users.

### ***Plan of Work***

Graphical user interface

Description automatically generated

### Breakdown of Responsibilities

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

Modules, Class diagrams, and Design of Tests are yet to be completed. This planning phase is scheduled to occur between the completion of Report 2 (Part 1) and as part of the submission of Report 2 (Part 2).

* + 1. Who will coordinate the integration?

The integration will be coordinated by team leader Kira Luethe, as the project’s GitHub is hosted on her account. All members will 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 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 Report 2 (Part 2).

## 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/>