**CS3354 Software Engineering**

**Final Project Deliverable 2**

GrowFlow

Brendan Go, Nevin Gilday, Anish Gajera, Sarem Hussein, Israa Hanoun

[Repository Link](https://github.com/anishgajera2/3354-GrowFlow)

**1.** Delegation of tasks:

Final project draft description - Anish

Setting up github - Anish

Software Process model - Sarem

Software Requirements - Sarem

Use case diagram - Israa

Sequence diagram - Anish

Class diagram - Nevin

Architectural design - Brendan

Everything else - asynchronously through team

**2.**

**Project Deliverable 1 content**

**1.** The objective of GrowFlow is to design a comprehensive solution streamlining farm operations. GrowFlow is a software system designed to help local farmers manage their farmland, livestock, equipment, resources, and distribute products to their customers. GrowFlow aims to provide farmers with a centralized platform to manage all aspects of their farm. Some of these aspects include crop management, inventory control, crop rotation scheduling, customer engagement, etc. The software is equipped with advanced analytics tools that provide real-time insights into the farm’s performance, allowing farmers to make informed decisions about their operations. The software integrates with other tools, such as weather forecasting systems, to provide a comprehensive view of the farm's conditions and needs, allowing farmers to make informed decisions about planting, harvesting, and resource allocation. Additionally, the software includes features to manage the supply chain, from ordering seeds and equipment to delivering products to market.

* To comply with the proposed changes, our group has fairly delegated tasks, this can be seen on the third question of this document.
* On the final report, we will compare our software to a similar application and convey how they differ.

**2.** Github Repository already set up. [Repository Link](https://github.com/anishgajera2/3354-GrowFlow)

**4.** The incremental model is employed for the project GrowFlow. This is because the process model allows the development of each program to be divided into smaller and more manageable sections,which is important for meeting the complex requirements of the project . All four increments can participate in meeting the criteria of the function and non-functional requirements. Through each increment, the software can be comprehensively tested and evaluated before moving on to the subsequent increment.

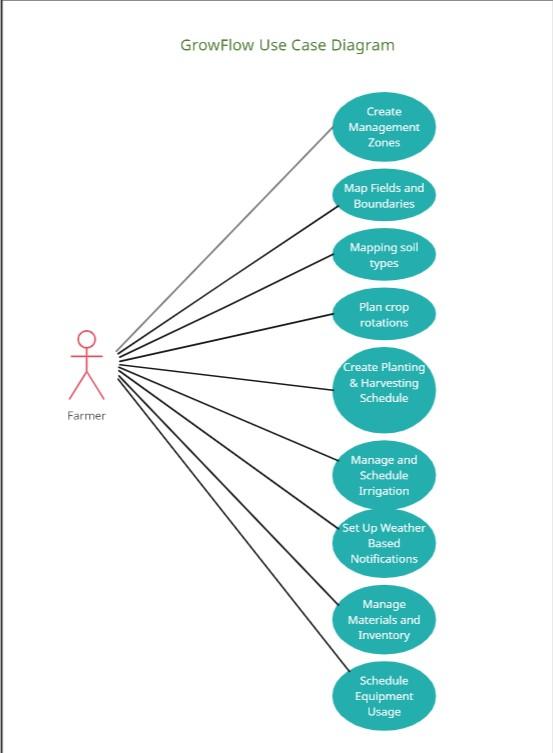
**5.a) Functional requirements**

1. **Farmland management**: The software should enable farmers to manage their farmland by offering the feature for crop rotation, soil mapping, and land-use analysis.
2. **Equipment management:** GrowFlow should incorporate a wide range of functions related to equipment management. Firstly, the software should be able to track and record the history of each piece of equipment. Secondly, the software should have a means of scheduling maintenance. Lastly, the software should alert farmers through notifications when maintenance is due.
3. **Crop Managemen**t: The software should alert users when it is the right time to plant each crop. It should also provide farmers the history of each crop, including ideal planting periods.
4. **Budget Management**: Farmers should be able to input bills into the system and the software should provide reports based on a specified criteria.
5. **Water/Sprinkler**: GrowFlow should allow users to monitor water usage and evaluate the effectiveness of the sprinkler system.
6. **Livestock management:** Farmers should be able to monitor and document information about all the animals in the farmland.General information such as height, weight, and age should be documented in the software.
7. **Weather monitoring**: GrowFlow should monitor weather patterns and predict severe weather conditions.Additionally, the software should be able to alert farmers about potential hazardous weather.

**5.b.) Non-functional requirements**

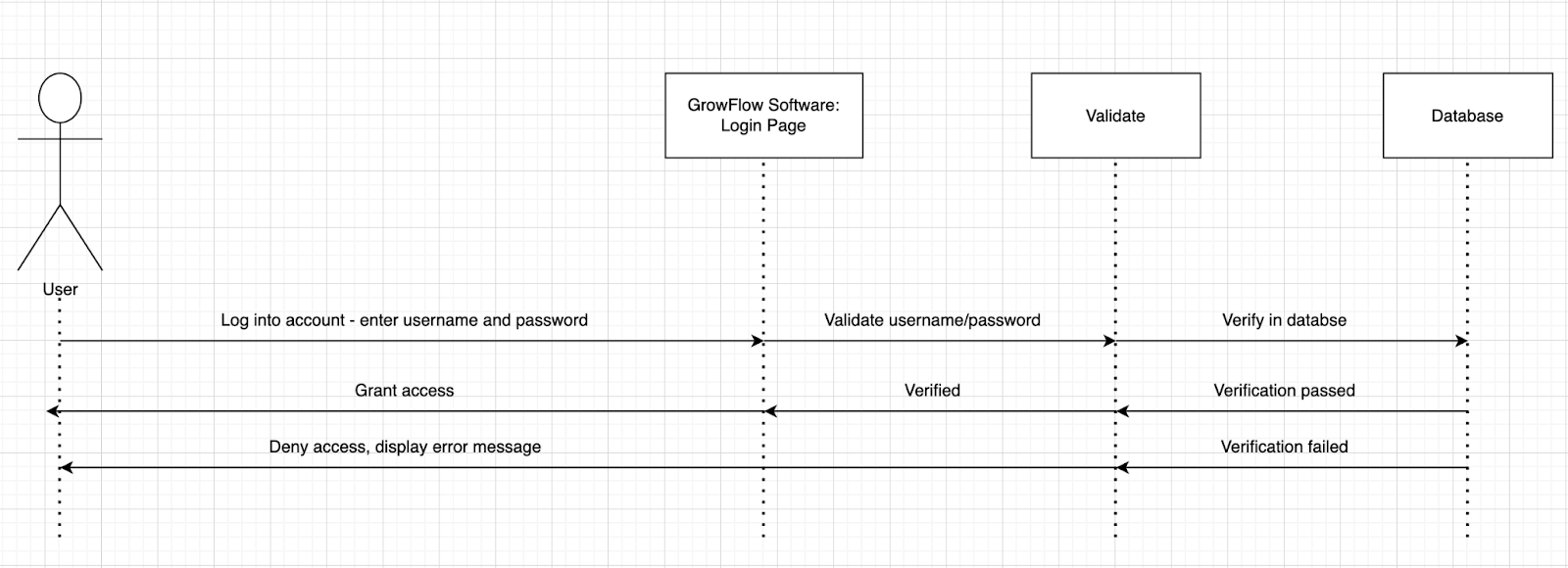
1. **Accounting requirements:** Financial transactions for the farm should be documented precisely so that there is an accurate financial report in the system.
2. **Safety/Security requirements:** Employee information should be kept confidential. Security measures such as encryption should take place to protect against security threats.
3. **Space requirements:** Storage space should be maximized for the various data generated from farm operations, such as livestock information, financial data and customer data.
4. **Performance requirements:** Users should be able to receive fast response for the entirety of the function the software provides.
5. **Usability requirements:**Farmers should easily learn and navigate all of the functionality the software provides.
6. **Dependability requirements:** It is essential to keep the software running even in circumstances where power outages and hazardous weather might occur.
7. **Regulatory requirements:** The software should adhere to relevant industry standards, essentially those set up by farmland organizations.
8. **Ethical requirements:** The software should be designed to ensure information or data acquired from the farm is not accessible to unauthorized parties.
9. **Environmental requirements:** Farmers should be able to access the software from various operating systems, web browsers, and mobile devices.
10. **Operational requirements:** The software should allow users to enter data, monitor farm operations, and collaborate with other farmers.
11. **Development requirements:** Development of the software should take place in a reliable platform such as Visual Studio Code.The programming language should also be chosen based on the reliability for the purpose of managing farmlands.

**6. Use-Case Diagram**

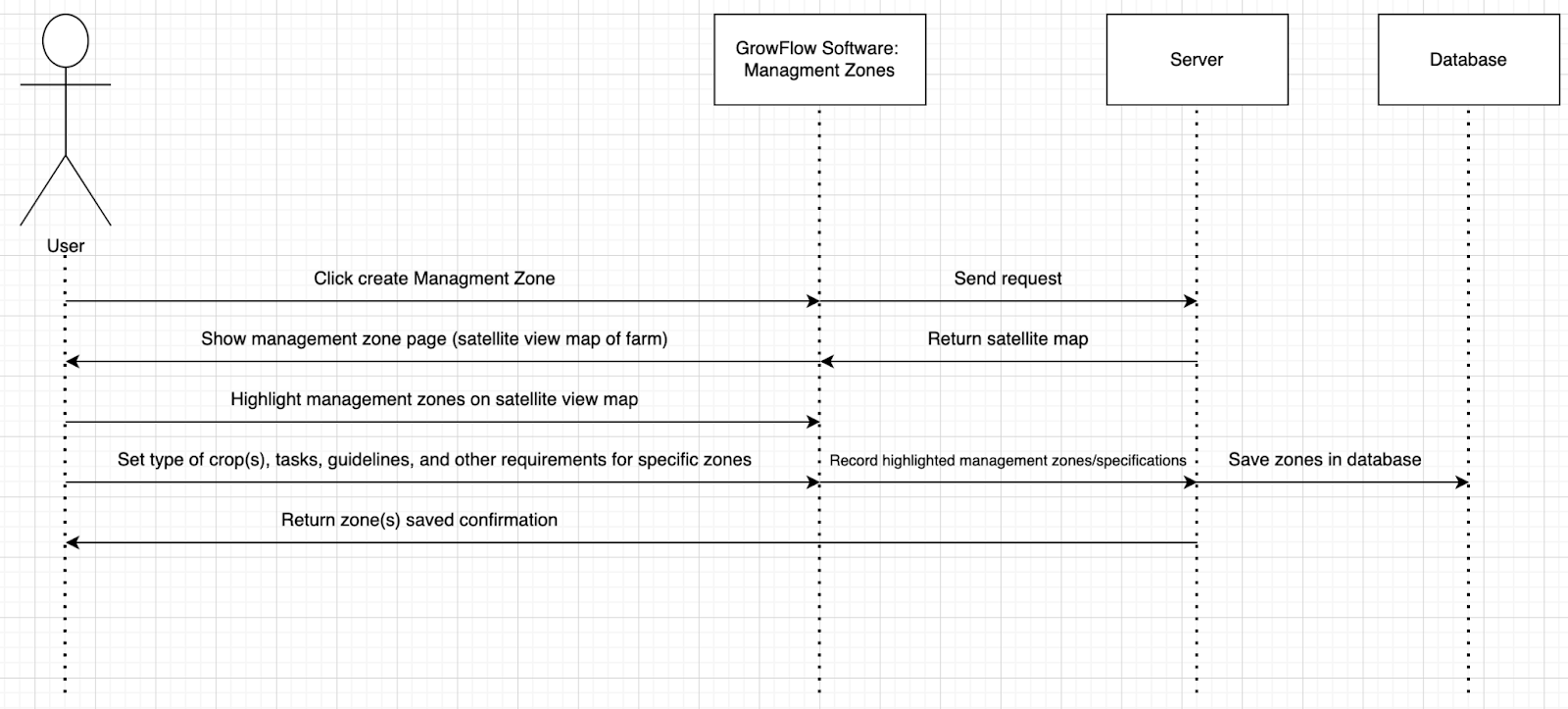
****

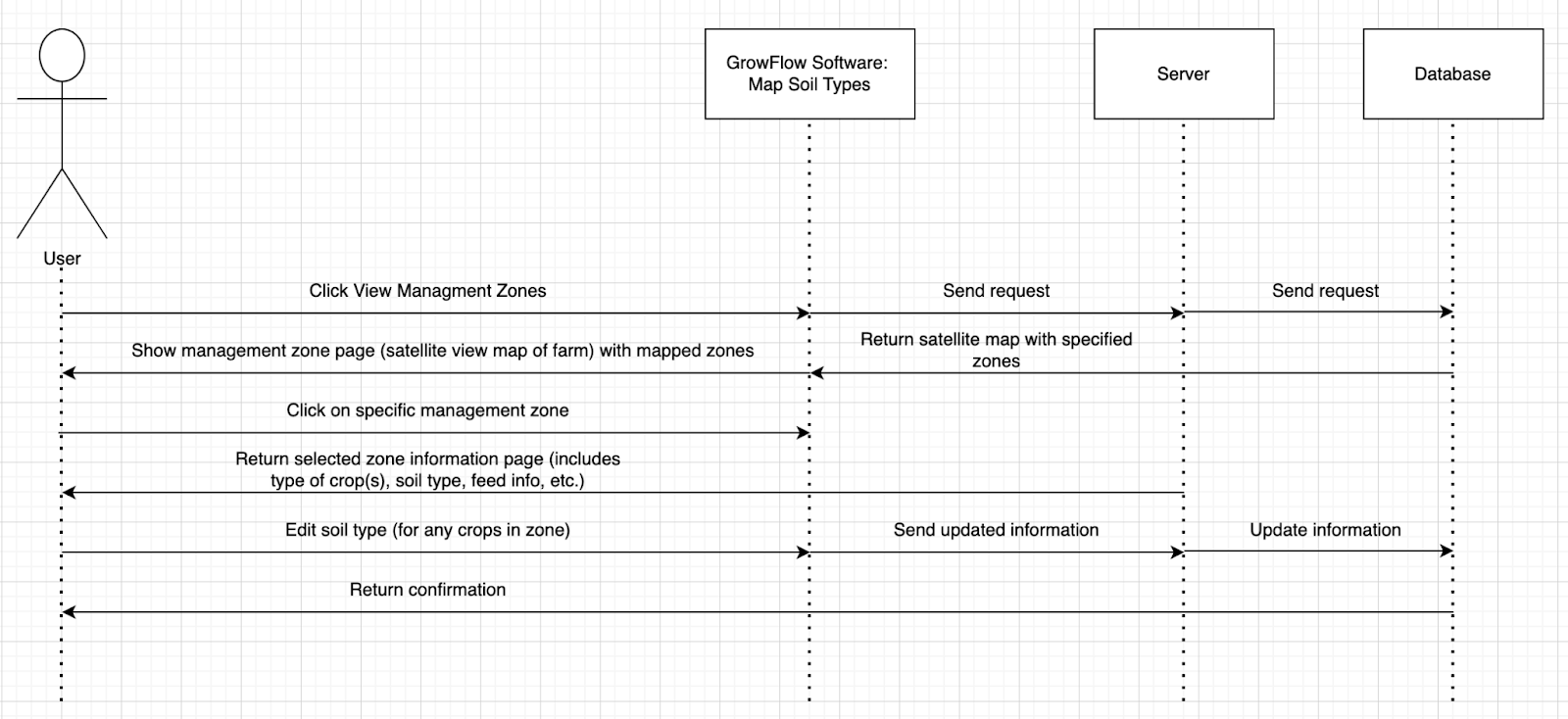
**7. Sequence Diagrams**

**User Login**

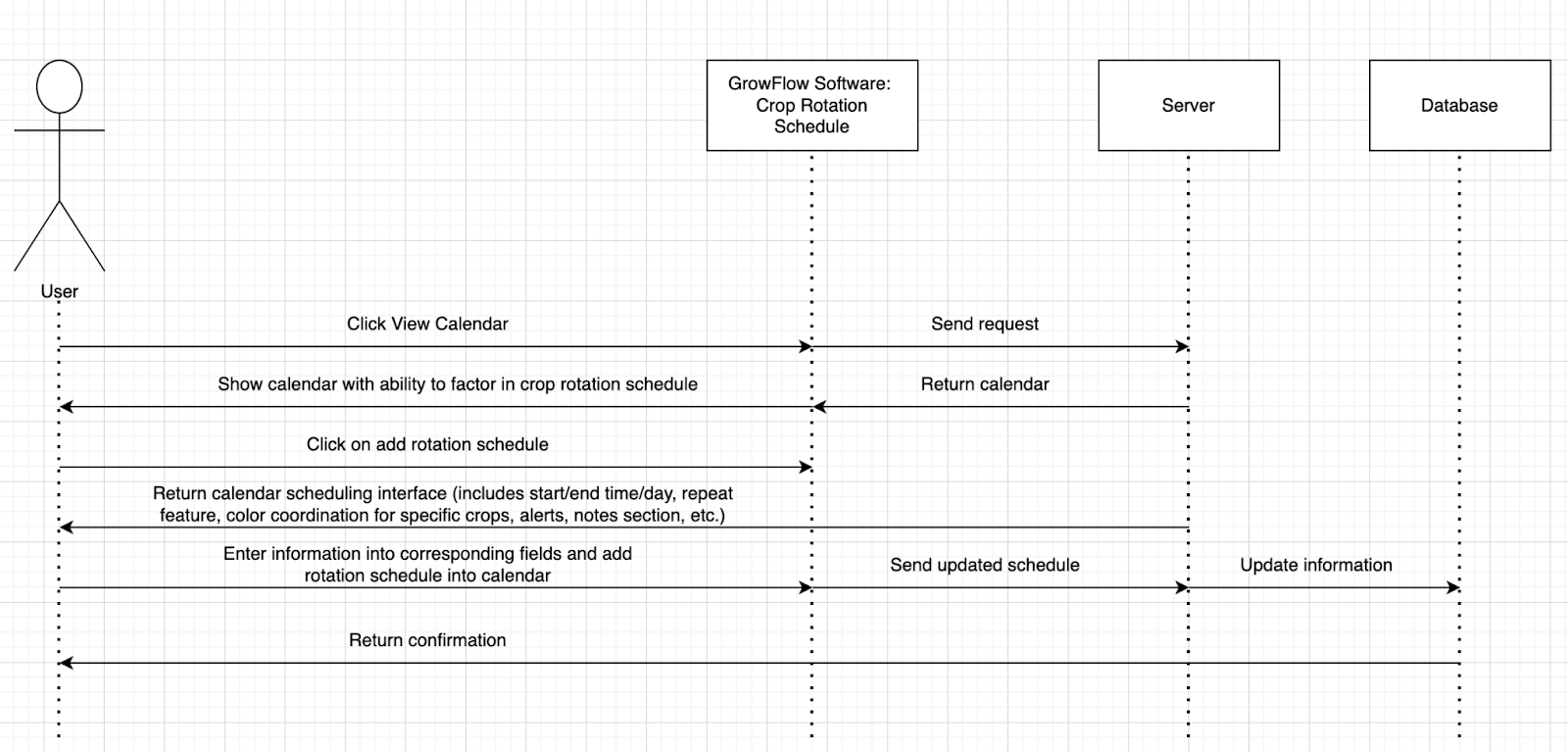
****

**Management Zones**

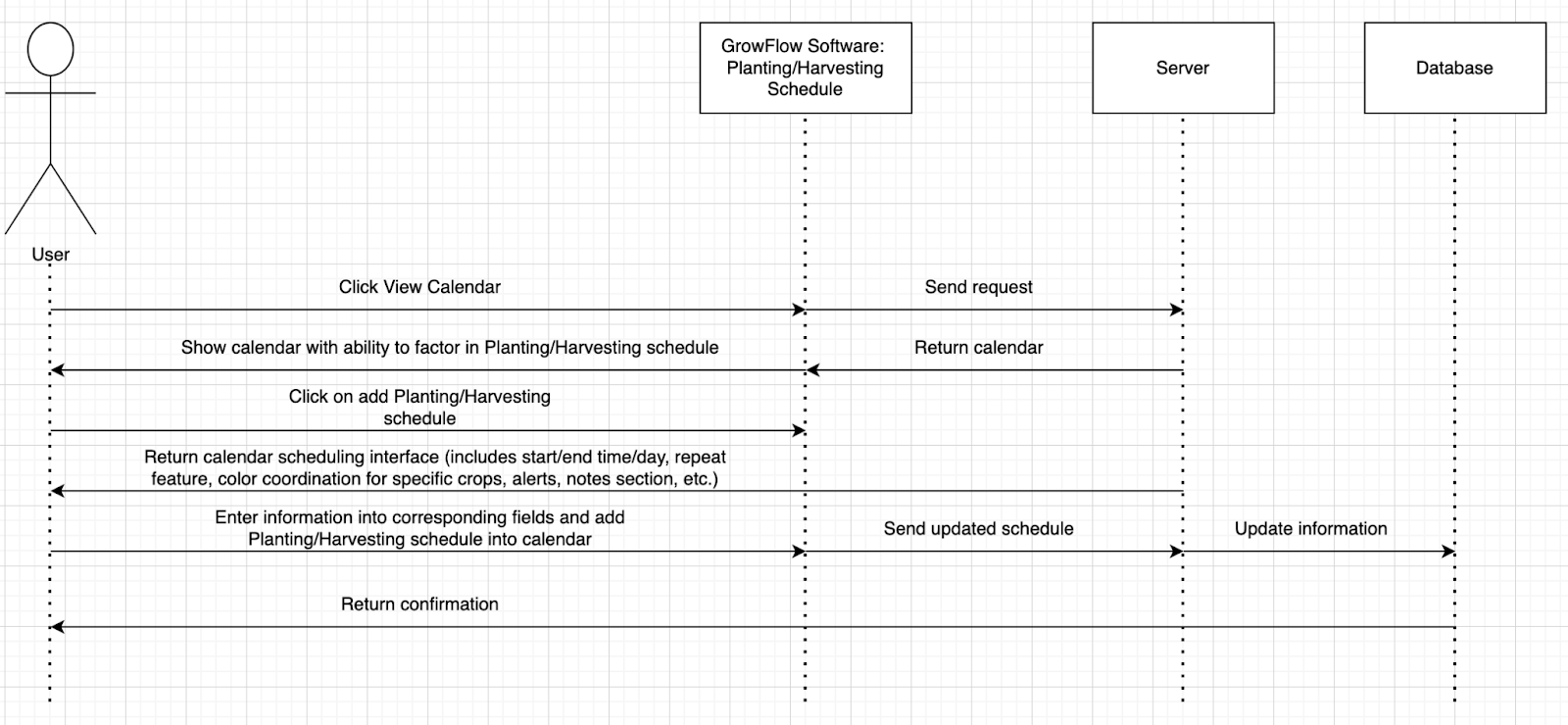
****

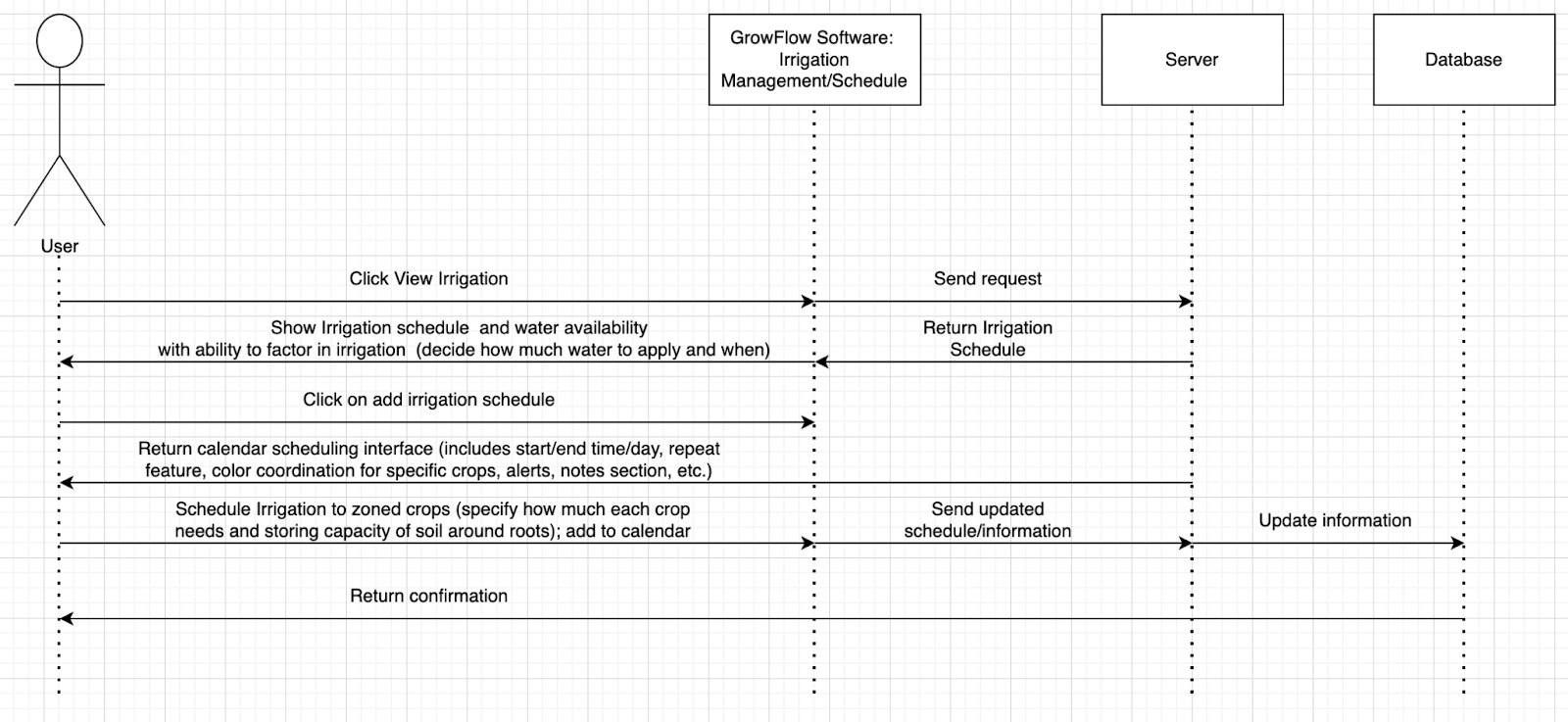
**Mapping Soil Types**

**Plan Crop Rotation**

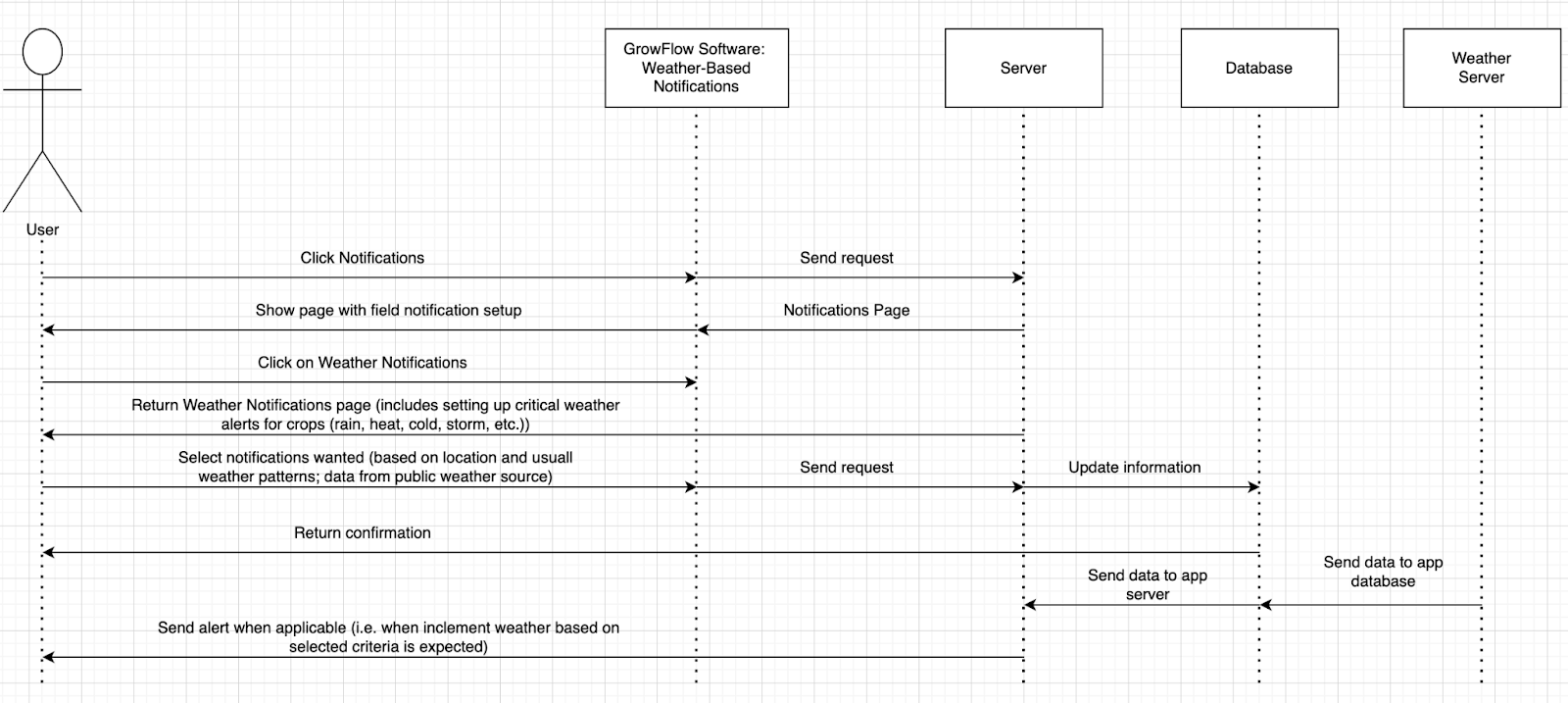
****

**Create Planting/Harvesting Schedule**

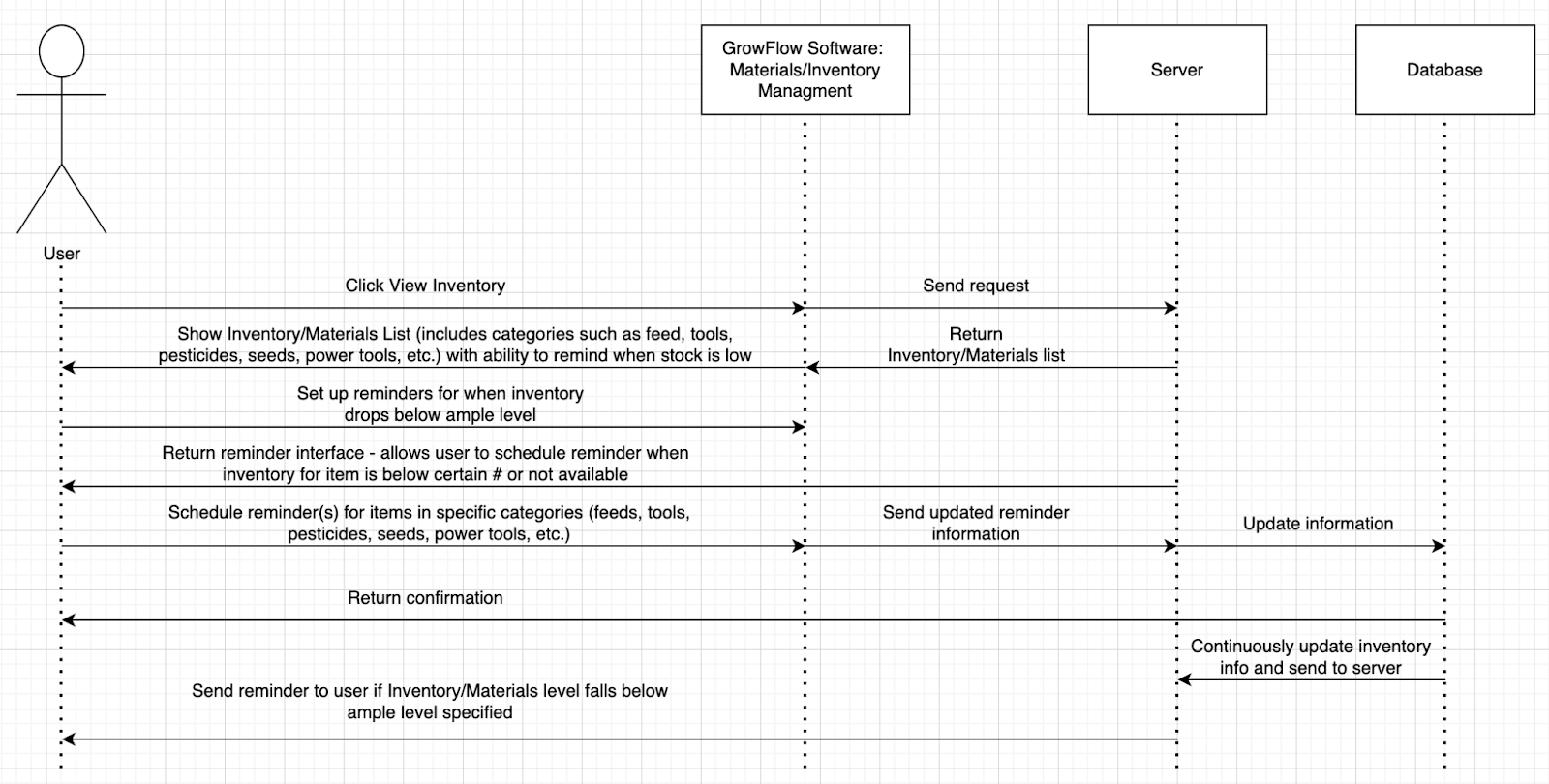
****

**Manage and Schedule Irrigation**

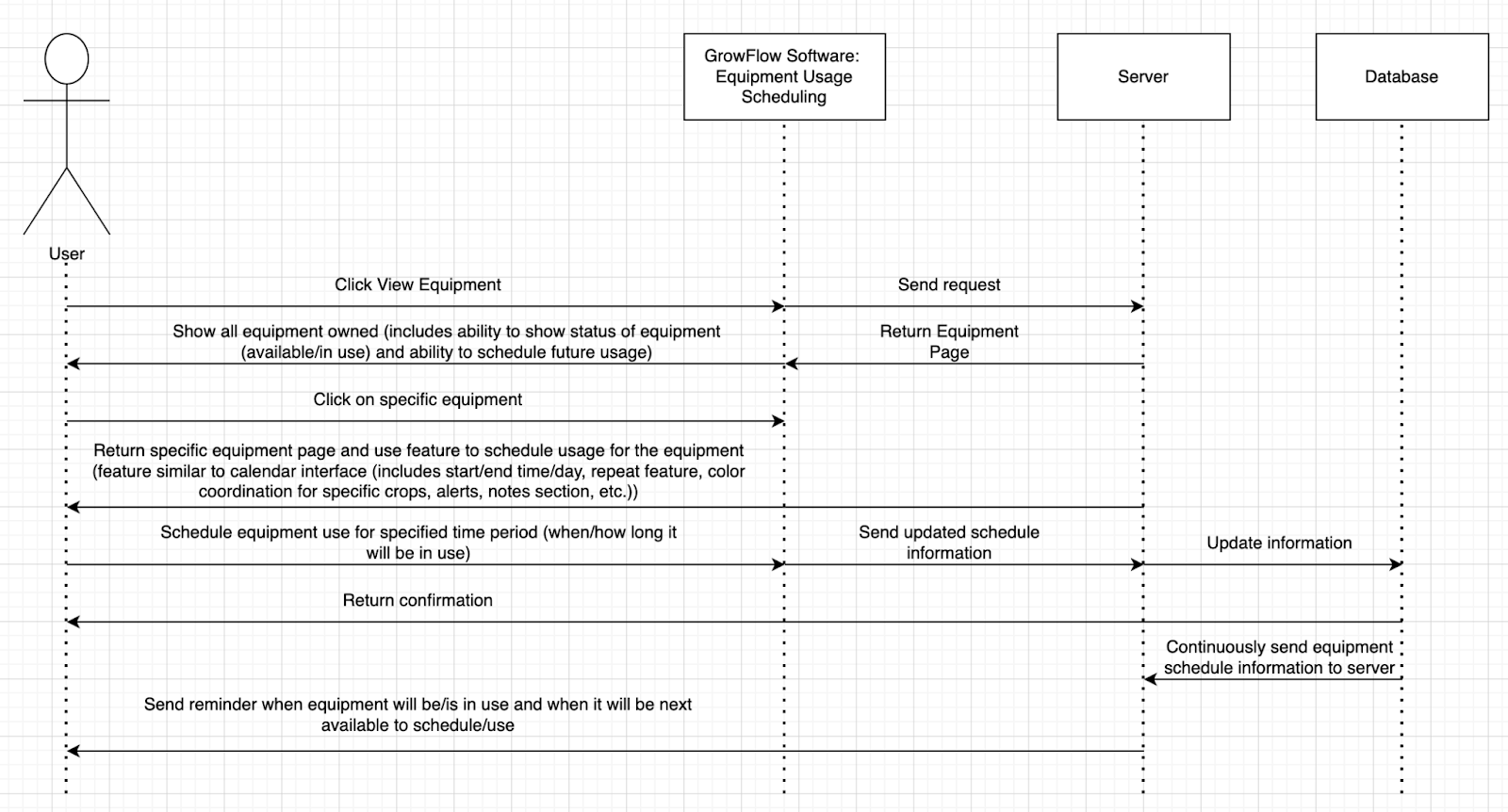
**Set Up Weather Based Notifications**

****

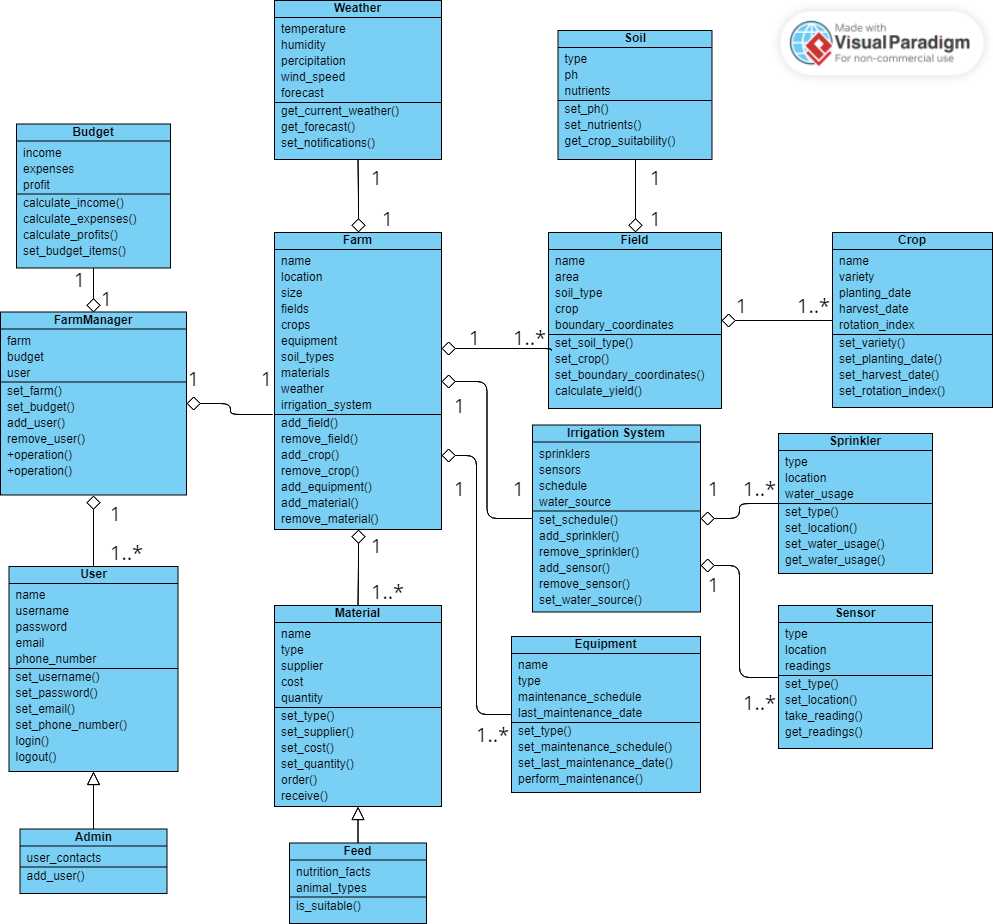
**Manage Materials and Inventory**

****

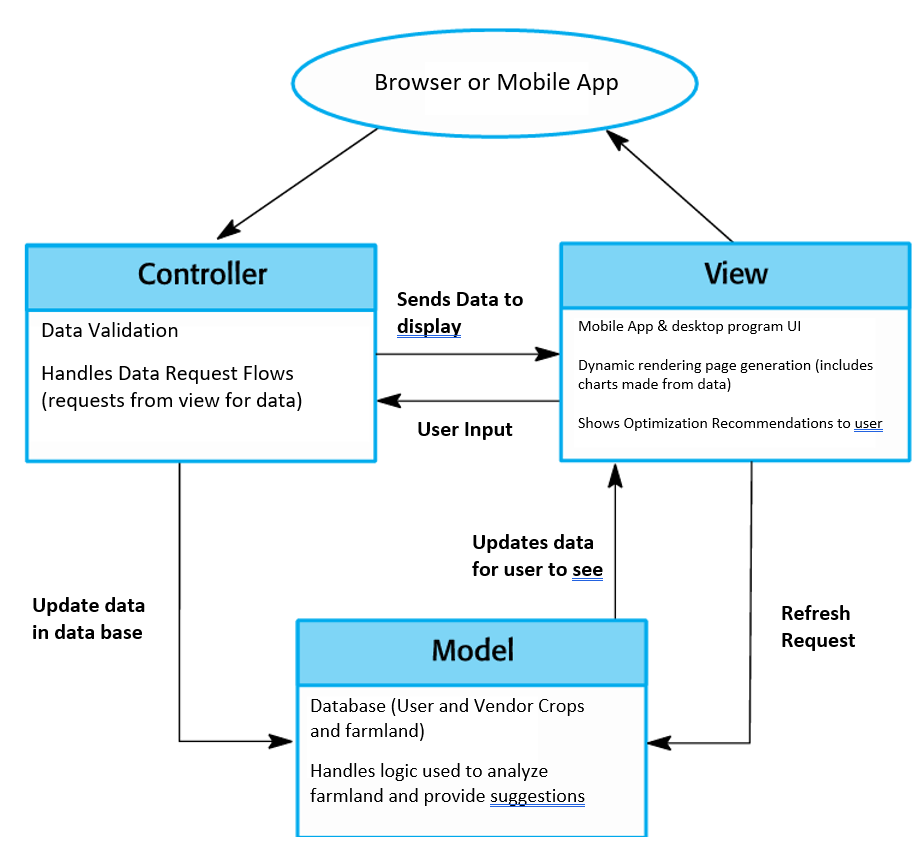
**Schedule Equipment Usage**

****

**8. Class Diagram**

****

**9.**

For our project, we chose to represent GrowFlow with the MVC architecture pattern. MVC is one of the more popular designs for applications, which is what GrowFlow’s main form of customer interaction is striving to be. I believe it would also work very well, as the plan is very simple to follow, and we don’t need multiple levels like layers or steps. The only thing needed is a division between what the customer sees vs the brain and database behind it.

Our group all agreed on MVC as being the most appropriate for GrowFlow.

Reasons:

* Easy to understand and implement
* Appropriate as GrowFlow is meant to be a software Application
* We can split up the work for efficiency
* The software may need to be updated fairly consistently to make sure that the user
* interface is appropriate for the customer, and you don’t want it to affect the whole system
* It promotes a good initial planning process
* System would be easy to maintain after implementation, if we were going to implement it

**End of Project Deliverable 1 content**

**3.1 -** Project start date: October 1, 2023

**Justification:** Starting in October would provide enough time to complete the project, including any potential adjustments or iterations that may be needed, before farmers begin their preparations for the upcoming farming season.

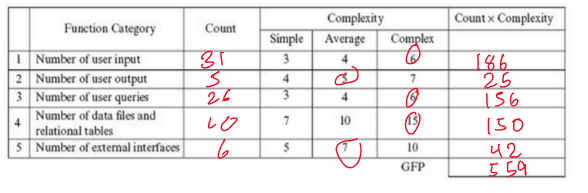
- Project end date: March 1, 2024

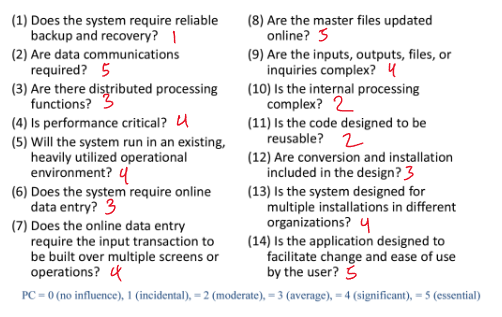
**Justification:** The project being completed by March will allow farmers to extensively test the software and provide feedback to the developers for improvement. The feedback cycle is essential to ensure that any problems or improvements found during testing may be dealt with before the start of the farming season in April.

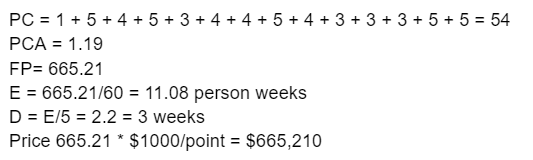
* Weekends will not be counted in the schedule because complex software such as GrowFlow requires various resources, including the software development team and stakeholders testing the program. The weekends give these resources a chance to relax, recharge, and be available for the project the following week. Weekend work isn’t taken into account in the project plan to prevent resource overload, which can result in burnout and lower productivity.
* Work hours: eight hours a day.

**Justification:** The project is going to require a typical workday length. The total duration includes breaks, meetings, and other related factors.

**3.2** -







Our group had a pretty unanimous decision for the Function point method. The factors we considered for choosing our algorithmic estimation technique were the simplicity of calculation as well as how easy it would be to calculate based on the data we already had. We came to the function point method as we realized it was very easy to get the values off of our class diagram from the 1st deliverable.

To find each of our numbers we went off of the slides and Gross function point chart and then referenced what in our class diagrams might count for each of the categories. We looked at setter and getter functions and how they would translate to each category like the user input as well as what portions would require its own database from user input for number 5. Regarding the pricing complexity, we got together as a group to decide on the correct value for each of the 14 points and then added them together to find the PC.

Once the chart and PC had been filled out, the rest of the calculations were fairly straightforward. The PCA which turned out to be 1.19 was computed from the equation: .65 +.01(PC sum), The FP was then calculated by multiplying the GFP from the chart by the PCA to get 665.21. The effort was then found by dividing the function point by productivity in which we used a generic 60 function points per person-week and since our team size is 5 to find the total estimated time, we divided the effort by the size and got that the project should take a little over 2 weeks.

For the cost of the project, since we do not have any prior experience to go off of to find the cost per function point, research was performed to find that the average cost per function point is anywhere from $500-$1500 per function point. For this model we will use the mean of $1000 per function point, putting our cost at just around $665,000.

**3.3**

Development Workstations: Each developer working on the project will need a high-performance workstation with enough processing power, memory, and storage capacity to run the development tools and build the app.

The cost of a high-end development workstation can range from $1,500 to $3,000 or more.

So for this cost, we average about 1500 per developer, though this cost can potentially be mitigated/eliminated as the dev team utilized for the project would probably have this requirement covered.

Servers: The cost of a server can vary widely depending on its specifications, with a basic server costing around $1,000 to $2,000, and more powerful servers costing tens of thousands of dollars.

The app server realistically does not need a super powerful server, so this cost can be boiled down to the $1000-2000 range.

Storage: You will need sufficient storage capacity to store the app's source code, development assets, and any data generated during the development process. The cost of storage can vary depending on the capacity and type of storage used, with a basic external hard drive costing around $100 to $200, and enterprise-grade storage solutions costing thousands of dollars.

Since this app will have multiple databases and storing large amounts of user data, Growflow should probably look into multiple external hard drives making this cost around $100-$200 \* 3.

**3.4**

Integrated Development Environment (IDE)

Most software elements are open-source and free for users making many of the software requirement costs free/ uncertain as it depends on what level is being used for the development.

Version Control System: A version control system is used to manage and track changes to the source code and development assets.

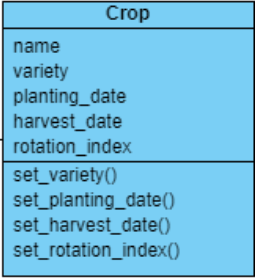
Libraries and Frameworks: Libraries and frameworks are pre-built code components that can be used to speed up development and reduce coding errors. Many popular libraries and frameworks such as jQuery, React, and Angular are free to use, but some may require a license fee for commercial use.

Database Management Systems: A database management system is used to manage the application's data storage and retrieval. The cost of a database management system can vary depending on the vendor and the licensing model, with popular systems such as MySQL and PostgreSQL ranging from free to several thousand dollars per license.

3.5

For the Cost of Personnel, it would seem to be reasonable to assume the development team would consist of at least 4 people. This being a project manager and at least 3 developers. And looking at the average cost being around $100 an hour for developers, we can assume the average hourly cost of $400-500 an hour for the development team. Past development and looking at training, this cost can realistically be covered by either a current employee or a small-time hiree which can be priced around $30/hr.

4.

The unit of the software I want to test is the crop class. The crop class looks like this:

The plan for testing the crop class is to create a crop object and assign it values for all of its attributes using the getters and setters. I will then write tests for all the getters and setters to make sure they are working properly. Next I want to validate the constructor by assigning attribute values and checking that they return the correct values. Lastly I want to validate a is\_ready\_to\_harvest() method. This method wasn’t included in the original class diagram for “crop” but it should return if the crop is ready to harvest based on the current date.

The code for the Crop class is the following:

import java.time.LocalDate;

public class Crop {

private String name;

private String variety;

private LocalDate plantingDate;

private LocalDate harvestDate;

private int rotationIndex;

public Crop() {

}

public Crop(String name, String variety, String plantingDate, String harvestDate, int rotationIndex) {

this.name = name;

this.variety = variety;

this.plantingDate = LocalDate.parse(plantingDate);

this.harvestDate = LocalDate.parse(harvestDate);

this.rotationIndex = rotationIndex;

}

public boolean isReadyToHarvest(String currentDate) {

LocalDate date = LocalDate.parse(currentDate);

return date.isEqual(harvestDate) || date.isAfter(harvestDate);

}

// getters and setters for all attributes

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getVariety() {

return variety;

}

public LocalDate getPlantingDate() {

return plantingDate;

}

public LocalDate getHarvestDate() {

return harvestDate;

}

public int getRotationIndex() {

return rotationIndex;

}

public void setVariety(String variety) {

this.variety = variety;

}

public void setPlantingDate(String plantingDate) {

this.plantingDate = LocalDate.parse(plantingDate);

}

public void setHarvestDate(String harvestDate) {

this.harvestDate = LocalDate.parse(harvestDate);

}

public void setRotationIndex(int rotationIndex) {

this.rotationIndex = rotationIndex;

}

}

The code for the CropTest class is the following:

import org.junit.Test;

import static org.junit.Assert.\*;

import java.time.LocalDate;

public class CropTest {

@Test

public void testConstructor() {

Crop crop = new Crop("Tomato", "Roma", "2023-04-01", "2023-06-30", 1);

assertNotNull(crop);

assertEquals("Tomato", crop.getName());

assertEquals("Roma", crop.getVariety());

assertEquals(LocalDate.parse("2023-04-01"), crop.getPlantingDate());

assertEquals(LocalDate.parse("2023-06-30"), crop.getHarvestDate());

assertEquals(1, crop.getRotationIndex());

}

@Test

public void testGetHarvestDate() {

Crop crop = new Crop("Tomato", "Roma", "2023-03-01", "2023-05-30", 2);

assertTrue(crop.isReadyToHarvest("2023-05-30"));

assertFalse(crop.isReadyToHarvest("2023-05-01"));

assertTrue(crop.isReadyToHarvest("2023-06-01"));

}

@Test

public void testGetName() {

Crop crop = new Crop();

crop.setName("Tomato");

assertEquals("Tomato", crop.getName());

}

@Test

public void testGetPlantingDate() {

Crop crop = new Crop();

crop.setPlantingDate("2022-04-20");

assertEquals(LocalDate.parse("2022-04-20"), crop.getPlantingDate());

}

@Test

public void testGetRotationIndex() {

Crop crop = new Crop();

crop.setRotationIndex(1);

assertEquals(1, crop.getRotationIndex());

}

@Test

public void testGetVariety() {

Crop crop = new Crop();

crop.setVariety("Beefsteak");

assertEquals("Beefsteak", crop.getVariety());

}

@Test

public void testIsReadyToHarvest() {

Crop crop = new Crop("Tomato", "Beefsteak", "2022-04-20", "2022-07-20", 1);

assertTrue(crop.isReadyToHarvest("2022-07-20"));

assertFalse(crop.isReadyToHarvest("2022-07-10"));

}

@Test

public void testSetHarvestDate() {

Crop crop = new Crop();

crop.setHarvestDate("2022-07-20");

assertEquals(LocalDate.parse("2022-07-20"), crop.getHarvestDate());

}

@Test

public void testSetName() {

Crop crop = new Crop();

crop.setName("Tomato");

assertEquals("Tomato", crop.getName());

}

@Test

public void testSetPlantingDate() {

Crop crop = new Crop();

crop.setPlantingDate("2022-04-20");

assertEquals(LocalDate.parse("2022-04-20"), crop.getPlantingDate());

}

@Test

public void testSetRotationIndex() {

Crop crop = new Crop();

crop.setRotationIndex(1);

assertEquals(1, crop.getRotationIndex());

}

@Test

public void testSetVariety() {

Crop crop = new Crop();

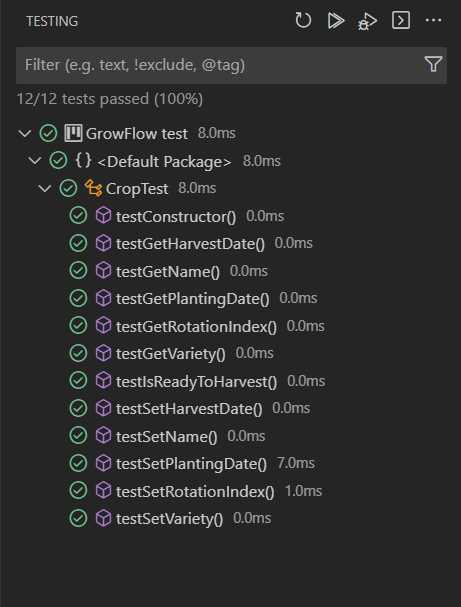
crop.setVariety("Beefsteak");

assertEquals("Beefsteak", crop.getVariety());

}

}

The following are the results from testing the class:



**5.** The rise of technology has revolutionized farming, and the use of farm management software has made it easier for farmers to manage their operations. GrowFlow is a farm management app that aims to streamline farm operations by providing farmers with a centralized platform to manage their farmland, livestock, equipment, resources, and distribute products to their customers. Our app is unique in that there isn't really any other software that has all of the functions we are offering in just one app.

AGRIVI is a software with farm management, crop management, farm planning, equipment management, and budgeting. AGRIVI also allows users to create field maps, track soil quality through IOT SOIL products, and track weather conditions with AGRIVI IOT METEO [1]. AGRIVI has much in common with GrowFLow, but it still falls short in some aspects.

AgSquared Farm Management Software is another sophisticated product with many functionalities, such as planning, equipment and machinery management, harvest forecasting, and field mapping. AgSquared provides these services through a monthly per-user subscription and has 3 different plans based on the users’ needs [2].

Other farm management softwares include Farmbite and Conservis Farm Management. Farmbite offers livestock management, farm planning, and an eCommerce software fully integrated in with crop plans [3]. Conservis Farm Management Software tracks field activity, manages inventory, and provides yield analysis [4]. There are some other farm management softwares, but they all have limited functionalities and are not as expansive.

GrowFlow has taken all of the different functionalities that might be used on a farm and combined them all into one comprehensive and conclusive app, and that is not something available on the market right now. Not only that, but GrowFlow also offers unique features that are not available in other farm management apps.

**6.** In summary, our project idea came forth through group brainstorming sessions. After considering various options, we ultimately decided to focus on a farming-related topic and named our project GrowFlow. GrowFlow is a comprehensive software program designed to streamline agricultural operations and provide farmers with a centralized platform to manage different aspects of their farm, including equipment, crops, budget, and livestock. We identified functional requirements such as equipment management, crop management, budget management, and livestock management, as well as non-functional requirements such as safety, security, and ethical considerations. For the software development process, we opted for the incremental model, which allows for manageable sections of development. We also conducted JUnit testing for one of the classes based on our class diagram. To estimate the total cost of GrowFlow, our team utilized the Function Point cost modeling technique. Ultimately, by meeting both functional and non-functional requirements, GrowFlow aims to provide end-users with a solution that maximizes their farm operations.

**7.**

[1] “Farm management software for Digital Agriculture,” *AGRIVI*, 31-Mar-2023. [Online]. Available: https://www.agrivi.com/. [Accessed: 20-Apr-2023].

[2] “Farm management software,” *AgSquared*. [Online]. Available: https://www.agsquared.com/. [Accessed: 21-Apr-2023].

[3] “Farm management software for your whole farm,” *Farmbrite*. [Online]. Available: https://www.farmbrite.com/. [Accessed: 21-Apr-2023].

[4] “Conservis: Farm Management Software: Agriculture Apps: Farming Apps,” *| Conservis farm management software simplifies farm management and empowers farmers to make smarter data-driven decisions.*, 05-Jan-2023. [Online]. Available: https://conservis.ag/. [Accessed: 21-Apr-2023].

[5] E. A. Caburao, “Top 8 farm management software for 2023,” *SafetyCulture*, 21-Mar-2023. [Online]. Available: https://safetyculture.com/app/farm-management-software/. [Accessed: 20-Apr-2023].

**8.** [GrowFlow Slides](https://docs.google.com/presentation/d/19UIY0IjV-XzjN9jN1iud0Aa2sCh_LBEVawCY66JzFSI/edit?usp=sharing)

**10.** Github Repository already set up and updated. [Repository Link](https://github.com/anishgajera2/3354-GrowFlow)