SAT PART B

Unit 4 Software Development

Sophie MacRaild

Part B – Unit 4 Outcome 1
Development and Evaluation

Table of Contents

Component 1: Using a programming language to develop the software solution

Nursery Organiser was created using the **Visual Basic** coding language, using a **Windows Forms** document on **Visual Studio**. This was chosen due to the easy **Graphic User Interface** (GUI) layout, and time restraints.

To access a digital copy of the code from GitHub click here or go to https://github.com/macra08842/SAT.git .

1.1 Processing features

Processing features used within code include:

- Classes (public and private)
  - These are used for different objects in the GUI and in the form in general
- Control structures (selection and iteration)
  - Loops, statements and other functions are used
- Functions
  - Enter data
  - Sort the database
  - Search the database
  - Modify/update a single field in the database
    - *This is not fully working upon completion*

These all worked together to create a comprehensive code with a variety of different functions.

See code in *Appendix A: Code* or GitHub link for specific use

1.2 Internal documentation

See code in *Appendix A: Code* or GitHub link for specific use

1.3 Validation techniques

Validation techniques used within code include:

- Existence checking
  - Boolean values are used to make sure a value exists
  - If-then-else statements and loops require for certain variables
- Range checking
  - For each function to work, a parameter was used, and then the variable was compared against it to ensure it is valid.
- Type checking
  - Checks the assigned variables and the data type that is given to it

See code in *Appendix A: Code* or GitHub link for specific use

1.4 Critical and creative thinking in modification and evaluation

Throughout the coding process various challenges appeared. Creative thinking needed to be used to produce solution alternatives and then critical thinking was applied to these alternatives to select a solution or modification to move forward. Details about three main challenges are below.

Challenge #1:
Having a separate form for a user login system was not possible, since the GUI displayed will appear different on different screen dimensions, and also makes it difficult to connect it to the home page.
Solution alternatives:

- Modify the order of all the forms to make the login page appear first. Get the user to keep trying the password until correct, then homepage will appear
- Use input boxes appear before the homepage is loaded to be a username and password. If the password is incorrect, the program will close.

Solution selected and reason why:
The input box solution was chosen to be used in this iteration, as it fulfills the password requirements and is user friendly. The homepage will not open if the login is incorrect – it will shut down the program.

Challenge #2:
Choosing the best way to code the functions was a challenge, as Python was initially trialled before the programmer's illness, which resulted in a change to using Visual Basic. Then transferring these sample codes made for a console to a GUI brought a second challenge.
Solution alternatives

- Find sample codes on how to do the functions of reading, writing, searching and sorting the code, with different ways on how to actually do the functions.
- Use the GUI code from one and place it into the other code.

Solution selected and reason why
The selected solution was a result of a combination of several sample codes together, in order to create a functioning code that works with the GUI.

Challenge #3:
Modifying, updating and deleting data were proven to be a challenge throughout this development process.
Solution alternatives

- Use function which does the whole modification process at once.
- Read, search, delete, write and save the updated data to the CSV

Solution selected and reason why
The selected solution for updating and deleting a specific field, was a combination of both solution alternatives, however, it more closely followed that of the second, simpler read-search-delete-write-save to CSV function. Deleting the whole CSV, however, was completed in one clean function.

Component 2: Managing Data and files and testing the software solution

2.1 Organisation and manipulation of data structures to manage data and files

The central database is stored in the same file as the software solution. This is through the file path on a USB stick, being 'E:\Back-upSAT\SAT\SATPartB'. In this file, all the necessary files and documents for the software to run will be found.

Data structures used within code include:

- One-dimensional arrays
  - This is how the data is stored on the CSV
- Records
  - This is the titles at the top of the CSV and how the arrays are categorised

See code in *Appendix A: Code* or GitHub link for specific use

2.2 Management of security of data and files

Proposed and implemented: There is a login system as the software opens. This is in the form of input boxes for 'username' and 'password' in this iteration of the software.

The software currently has various measures to protect the data stored. These include:

- Upon starting the software, a username and password are required to open the homepage.
- The software is stored on an encrypted USB stick, which requires a username and password when attempted to be opened.

Due to the low number of staff and varying internet availability due to the remote location of the nursery, there is low security risk and therefore there are lower security levels in this iteration of the software. The varying level of staff computer skills also limits the type of security that can be used initially. To fit these situations, a simple login system has been used, but can be changed in later iterations to add further security to the data.

2.3 Testing techniques and test data

The following Testing Table was used to test data and detect all errors. In includes ensuring that:

- actual outputs match expected outputs
- the solution performs reliably
- navigation and/or menu items correctly select the appropriate page
- screen elements appear where and when they should.

Testing Table

Item to be tested	Data Type	Data inputs	Expected outputs	Actual Outputs	Changes required
Username	string	"Admin"	Open password inputBox	Opened password inputBox	No changes required
Password	string	"Admin"	Open homePage	Message box indicating login was successful. Opened the homePage	No changes required
Username	string	"Steven" (or any other string)	Error message, followed by the program closing	Error message, followed by the program closing	No changes required
Password	string	"Oh dear" (or any other string)	Error message, followed by the program closing	Error message, followed by the program closing	No changes required
Search CSV	string	Name of client/plant pre-existing in the CSV	Label will display the details regarding the searched item. If the item is not there, an error message will come up on the screen	Label will display the details regarding the searched item. If the item is not there, an error message will come up on the screen	No changes required – except stopping the label from moving
Sort CSV	string	Click button	CSV sorts alphabetically by the name of the client or plant (depending on the CSV file sorted), displaying it in a list box	CSV sorts alphabetically by the name of the client or plant (depending on the CSV file sorted), displaying it in a list box	Aesthetic formatting may still need to occur – stop things from moving around
Read CSV	N/A	CSV file from path	Labels display how many items are in the CSV	Labels display how many items are in the CSV	Stop the label from moving when the program begins

			Corresponding list boxes will display the CSV	Corresponding list boxes will display the CSV	
Input plant name	string	Text box/input box	Moves the function to the next step, when adding a new item to the CSV	Moves the function to the next step, when adding a new item to the CSV	No changes required
Input plant type	string	Input box	Moves the function to the next step, when adding a new item to the CSV. If it is the last stage of the loop, it submits the data to the CSV	Moves the function to the next step, when adding a new item to the CSV. If it is the last stage of the loop, it submits the data to the CSV	No changes required
Input plant quantity	integer	Input box	Moves the function to the next step, when adding a new item to the CSV. If it is the last stage of the loop, it submits the data to the CSV	Moves the function to the next step, when adding a new item to the CSV. If it is the last stage of the loop, it submits the data to the CSV	No changes required
Input client name	string	Text box	Moves the function to the next step, when adding a new item to the CSV	Moves the function to the next step, when adding a new item to the CSV	No changes required
Running the program on a new computer	all	forms	The program consistently opens in the same way, no matter the computer or software	The program opened successfully, however, the GUI shifted, causing elements	Changes required for consistent operation as the software is opened

				to be left off the screen	- Completed, no further changes required
Updating / modifying records	String and integer	Textbox, message box and input boxes	The entered plant name is searched for in the CSV, if it is found a message box appears saying it is found and confirming that it is to be modified.	Message box said the plant could not be found.	Changed the coding of the location/locator.
Updating / modifying records	String and integer	Textbox, message box and input boxes	The entered plant name is searched for in the CSV, if it is found a message box appears saying it is found and confirming that it is to be modified.	No message box appeared.	Modified the search loop code – Completed, no further changes required.
Updating / modifying records	String and Integer	Input box	Input boxes appear where user can enter modified plant name, plant group and number	Input boxes appear where user can enter modified plant name, plant group and number	No changes required
Modifying plant data	String and Integer	Input box	Modified record appears in database list	Records after the plant that was modified were deleted	Separate Delete and Modify Function – Completed, no further changes required
Updating / modifying records	String and Integer	Input box	Modified record writes to database or deletes	Record not modified/deleted and message saying that couldn't write to file because file name was in use by another	Ensure all files were closed, error still occurs. Give temp file for the Delete and the Modify

				process – sometimes the plant data file, sometimes the temp file	different names, error occurs now with plantdata data file unable to be used Was unsuccessful in resolving this issue in this iteration of the code
Modifying plant record in csv			Uninterrupted movement between input boxes	'Index outside the bounds of the array' comes up after clicking 'yes' to wanting to modify data – clicking on enter goes to the modifying data input boxes	Identified the line of code that produced this error and altered various elements. Was unsuccessful in resolving this issue in this iteration of the code

Component 3: Usability Testing

3.1 Preparation and conduction of usability tests

3.1.1 What will be tested

To test the code, the following will be tested to ensure it functions correctly for the users and fulfills the functional requirements within the code.

- Logging onto the program
- Inputting/adding new data into the CSV
- Deleting the entire CSV (deleting a dummy CSV so real data is not lost)
- Searching the CSV
- Sorting the CSV
- Modifying/Updating the CSV[1]

The Graphic User Interface (GUI) will also have to be tested through usability testing. This will assess the affordance, usability and aesthetic properties of the GUI, such as the objects moving around on the screen unintentionally, but also test the user's ability to successfully complete the functionality.

Please refer to ***Appendix C: Tester Usability Table*** and **Appendix D: Tester Questionnaire** for what test subjects thought about the GUI and software solution from a user's point of view.

3.1.2 Test requirements

The test will be run over the weekend of the 13th-14th of August. The test should take no more than 15 minutes per person, as they will follow the *Tester Usability Table* and *Tester Questionnaire* and work through the software and then evaluate it after testing. The equipment needed is a computer that can run *Windows 11* and *Visual Studio 2022*. It would need to be in a quiet environment so proper assessments can be made.

3.1.3 Test participants

Test participants aged between, 10-21, 40-49 and 50-59 with a range of technical abilities were used because this is a representation of the ages and technical abilities of the people who work at the nursery. The participants volunteered to test the software and are active participators in citizen science.

[1] Note that this was not successful in this iteration after testing and was not able to function.

3.1.4 Determine test metrics

Success/Failure Metrics

- Success = tester completes task without asking questions or making mistakes (ie. having to 'go back')
- Partial success = tester needs to ask questions and/or makes a mistake but still completes task
- Failure = tester is unable to complete task, even after asking questions or making mistakes

Quantitative Metrics

Test subject will be asked to answer the following questions on the scale seen below

1: Very Hard 2: Hard 3: Neither hard nor easy 4: Easy 5: Very Easy

1. How easy is the program to read?
2. How easy is the program to use?
3. How easy is it to add an item to the database?
4. How easy is it to delete the database?
5. How easy is it to login?
6. How easy is it to search the database?
7. How easy is it to sort the database alphabetically?
8. How easy is it to navigate the software?

The following will also be measured:

- Average time taken to complete the test
- Average number of errors made
- Average number of questions asked to developer while testing
- Success/Failure of Test
- Developer observations:

Qualitative Metrics

Additional qualitative feedback will also be gathered and implemented for further testing. This will be completed through asking four open-ended questions. The results will be used in the evaluation stages.

Questions used:

1. In general, how easy is the software to navigate?
2. Describe how well the functions worked.
3. What would you like to see in a future update to easy usability?
4. Any additional comments?

3.1.5 Usability Test and Tester Feedback

See ***Appendix C: Tester Usability Table*** and ***Appendix D: Tester Questionnaire*** for actual test and feedback sheets used by testers.

3.2 Document results of usability tests

See ***Appendix C: Tester Usability Table*** and ***Appendix D: Tester Questionnaire*** for the combined results of the testers.

3.3 Modifications based on the results of usability tests

The usability testing resulted in varying feedback due to age and technical abilities of the testers. The usability tests resulted in some propsed modifications for the next iteration of the software, but also created a list for evaluation for future iterations. Things found from the tester usability tests were that the younger users were more technologically minded and found it easier to navigate than the older ones, leading to less critical comments.

Based on these results, the following modifications could be made in future iterations of the code:

- Clearing up the GUI to make it easier to read
- Moving the search function results to make them easier to read
- Having more functions on the one page to ease useability
- Have more graphics to allow for easier navigation
- Have a menu page on the side or up the top which allows for easier navigation
- More instructional values on how things work and what to do would be helpful
- More parameters for what you can input, or more descriptive errors so the user does not panic if something goes wrong
- Have the username and password system more secure and more professional

Component 4: Evaluation of Software Solution

4.1 Strategies for evaluating the efficiency and effectiveness of the software solution

Evaluation Criteria from SAT Part A

1) *Affordance:*

Affordance is what a user can do with an object based on the users' capabilities. It is not the physical properties of an object. The following criteria would be asked based around this concept:

- Can the user access the manager functions if they don't have the pin?
- Is a user has logged in incorrectly, does the software open?
- Is it clear what each button is meant to do?
- If a user is not logged in, is it clear they cannot use the features on that page?
- How effective are the graphics at indicating how they should be used?

Strategy: The usability test results can be used to evaluate the affordance of the code. The questions on the Tester Questionnaire are specifically written to evaluate the effectiveness of the software in terms of the level of affordance, so the test participants answers will account for this.

2) *Usability:*

Usability is the degree to which a software can be used by specified users to achieve quantified objectives with effectiveness, efficiency and satisfaction. The following criteria would be asked based around this concept:

- How easy is it to use for a non-technical user?
- Do all the functions work and save as programmed?
- Is it clear how to use the program?
- How efficient and effective is the software for a user?
- Is it quick for the user to perform the desired function within the software?

Strategy Step 1: The results of the Tester Usability Tests can be analysed to evaluate the usability of the code.
Strategy Step 2: The average times taken to do a function or use the software will also measure and quantify the efficiency of the code.
Strategy Step 3: The qualitative questions in the Tester Questionnaire also evaluate the usability.

3) *Cost:*

Cost is the financial cost that the software would have for the user or client. The following criteria would be asked based around this concept:

- Does it cost the client a lot to use?
- Do the client need to buy anything new for it to work?

Strategy Step 1: The cost is evaluated by calculating any software or hardware needed for the client to run it.

Strategy Step 2: The cost then can be paired with other criteria to evaluate the effectiveness of the solution, when compared to what the client already has.

4) Security:

Security is an ongoing process involving people and practices and ensures application confidentiality, integrity and availability with both the user's information and the database as a whole. The following criteria would be asked based around this concept:

- Does it protect the company's data?
- Does it maintain the user's privacy?
- Can you accidently modify data?

Strategy Step 1: The security is evaluated by comparing other security measures on other software, to the one currently in Nursery Manager.
Strategy Step 2: Getting somebody to hack in without the username or password, would find the weak points and allow for a full effectiveness evaluation and determine what to do better in future iterations.

5) Interoperability:

Interoperability refers to the functionality of different programs to exchange information, share files and use the same protocols. In this context, it includes opening the database. The following criteria would be asked based around this concept:

- Does the software easily connect to the database?
- Is the database easily able to be transformed to other forms once downloaded?

Strategy: To evaluate the interoperability of this software, try to open the database, and utilise the database in different ways. This would be, inside the code, see how the database is integrated into the GUI and how the functions relate to it, and consequently, the effectiveness of the database, and how efficiently it is used.

6) Marketability:

Marketability is based around how well developed the software is to fit an existing, external market and/or need for this software. The following criteria would be asked based around this concept:

- Is the software able to be adapted to meet a more general cliental?
- Is the software a new addition to the current market?

Strategy Step 1: To evaluate the marketability of the software, it will need to be evaluated on how usable it is, through the usability testing, and then improvements made based on that.
Strategy Step 2: Next, the software will need to be trialled on different devices, with different processors to see if it can be brought out to the general public. This would require evaluating it based on the efficiency on different processors, and the effectiveness of getting the task done concisely for the general public's users.

7) Client's Requirements:

- Does it fit the client's needs?

Strategy: The client's requirements within the SRS are listed in the following table. To evaluate if it fit the client's needs, compare them to the table, and see how much corresponds.

4.2 Evaluation of the efficiency and effectiveness of the software solution in meeting requirements

Functional Requirements from Part A SRS:

Function	Evaluation
A central, readable database stored either within the program or on the hard disk/external hard drive	The database was stored on a USB stick and is in the same file as the program. This was effective, but inefficient if it is going to be used in future iterations. This partially met the requirements.
Accessible offline	This was unable to occur in this iteration of the code and is therefore defunct in this iteration of the code. In future iterations, this can be tested by submerging the code to different conditions and modifying the code so it works accordingly. In this iteration, it did not meet the requirements.
Input/Write data	This was successful, effective and met the requirements. In future iterations, clearing the GUI could increase efficiency and usability
Read data	This was successful, effective and met the requirements. In future iterations, this could be done more efficiently in the code, instead of using lots of lines.
Search data	This was successful, effective and met the requirements. In future iterations, this could be done more efficiently for the user, as it could accept more values and search for other values, not just the main names.
Sort data	This was successful, effective and met the requirements. This could be improved in future iterations by allowing for it to sort it in other ways, not just by the names alphabetically. This will increase effectiveness.
Modify data (including deleting individual data records)	This function was written in the code but was unable to fully work for individual records. The only way to delete data with it working, was to delete contents of the entire database and then add in all the data again. This met the requirement of deleting data, but not modifying. This was unsuccessful and neither efficient nor effective.
Have an accessible database	Defunct in this iteration of the code. Therefore it did not meet the requirements.
See the history and visual data e.g., Graphs on how many sales, how	Defunct in this iteration of the code. It is out of the scope for this school SAT, due to time and base level client requirements. This was not pursued in this iteration and therefore did not meet the requirements.

the plants' produce are each year.	

Non-functional requirements from Part A SRS

Function	Evaluation
The software should be easily downloaded off either the internet, an app store or from an external drive like a USB (potability)	The software is able to be opened from a USB stick and, thus far, through Visual Studio. This was due to time and knowledge constraints. This was effective for this early iteration, but not efficient. In future iterations, this can be downloaded directly from a USB stick, the internet (website) or an app store. This would improve efficiency, but also marketability. This did not meet the requirements in this iteration.
Compatible with both MacOS and Windows computers (portability)	Defunct in this iteration of the code. Visual basic had to be used due to time restraints, and it resulted in the program only being able to be used on Windows based computers. This was ineffective, but it was efficient for this iteration although need not fully meet the requirements.
Fixed easily through updating the system or a quick system maintenance (maintainability)	This is able to occur, but only if the source code is returned to the developer again. Therefore, system maintenance is unable to be completed remotely. This is effective for this early iteration, but inefficient. This only partially met the requirements in this iteration.
Good security, so the client's information is not jeopardised or at risk, but a simple security procedure (robustness)	The security in this iteration has a very, very simple username and password system. This provided a basic security measure for the software, which will minimally prevent people from entering the software. This, and the USB stick being encrypted provides a basic security for the software, however, more will need to be done in further iterations. This current solution is efficient, but not effective for high security measures. Despite this, the base level requirements were met in this iteration.
The users should be able to choose to save or access their data at any given point in time (usability)	This was mostly successful, but whilst the data **can be saved** after being entered, they **cannot access the database** (download it) at any given time. This is semi-effective, as it does technically save the work, but it is not effective or efficient enough. This partially met the requirements in this iteration.
Australian Plant Nursery also expressed that they want a user-friendly graphic user interface (GUI) to prevent human error and to simplify	A GUI was successfully built in this iteration of the code. The usability testing proved that it was easy to navigate, and the functions were reasonably successful. While this GUI system was proven to be effective by the test participants, it was not fully efficient, as there were sending the user to different forms to perform a function instead of performing it within the form. This could be improved in future iterations by improving or implementing some of the comments made in the usability table (**Appendix C: Tester Usability Table**) and having a single page for

the data addition process and increase usability	many more functions, this could increase the efficiency as well as the effectiveness of the GUI in the solution. However, the GUI did met the requirements for this iteration.

4.3 Evaluation of how the development model assisted in the development of the software solution

The development model chosen – the agile model – allowed for contacting the client after smaller changes were made. This allowed for a more customer-focused approach, which was able to be reflected in the code. An example of this more frequent contact was when the security measures were put into place. In this initial iteration of the code, the client discussed that a very minimal security measure was needed, which then resulted in the universal username and password in the initial iteration. Other contacts like this supported the ongoing development of the solution and refined the functionality and abilities of the software solution.

This felt like the most natural development to use, since it is frequently contacting them for reviewing the software. However, due to different issues that came up, the development model may have been forgotten at times, and the development of the software continued without as many interactions as hoped. But when these hurdles were jumped, the agile model was returned to and contact with the client was made more frequently. The agile model should have been followed more strictly, as it would have helped to further meet the client's needs.

The waterfall model would not have worked in this situation, since it is based around minimal contact, with the main one being during the analysis stage – much earlier on. The spiral model may have worked for this software solution; however, it does not allow for as many revisits and problem-solving stages as the agile model does. Hence, the agile model was chosen over these other options, and it was the best choice.

4.4 Critical and Creative thinking through the evaluation of the analysis design and development stage and the identification and description of improvements to the software solution

Creative and critical thinking was used at each stage of the software development and can be found throughout all components of Part B's documentation.

- Creative thinking was used to generate different ways to evaluate and solve issues. For example:
    - Issues with time, regarding developer illness required the plan to change.
    - Other issues with the code, and coming up with a solution for them after the debugging process
- Critical thinking was used to decide which of the solutions to actually use. For example:
    - From the creative solutions, critical judgement had to be used to see whether they were feasible, and what options were most efficient and effective.

Component 5: Assessing the Project Plan

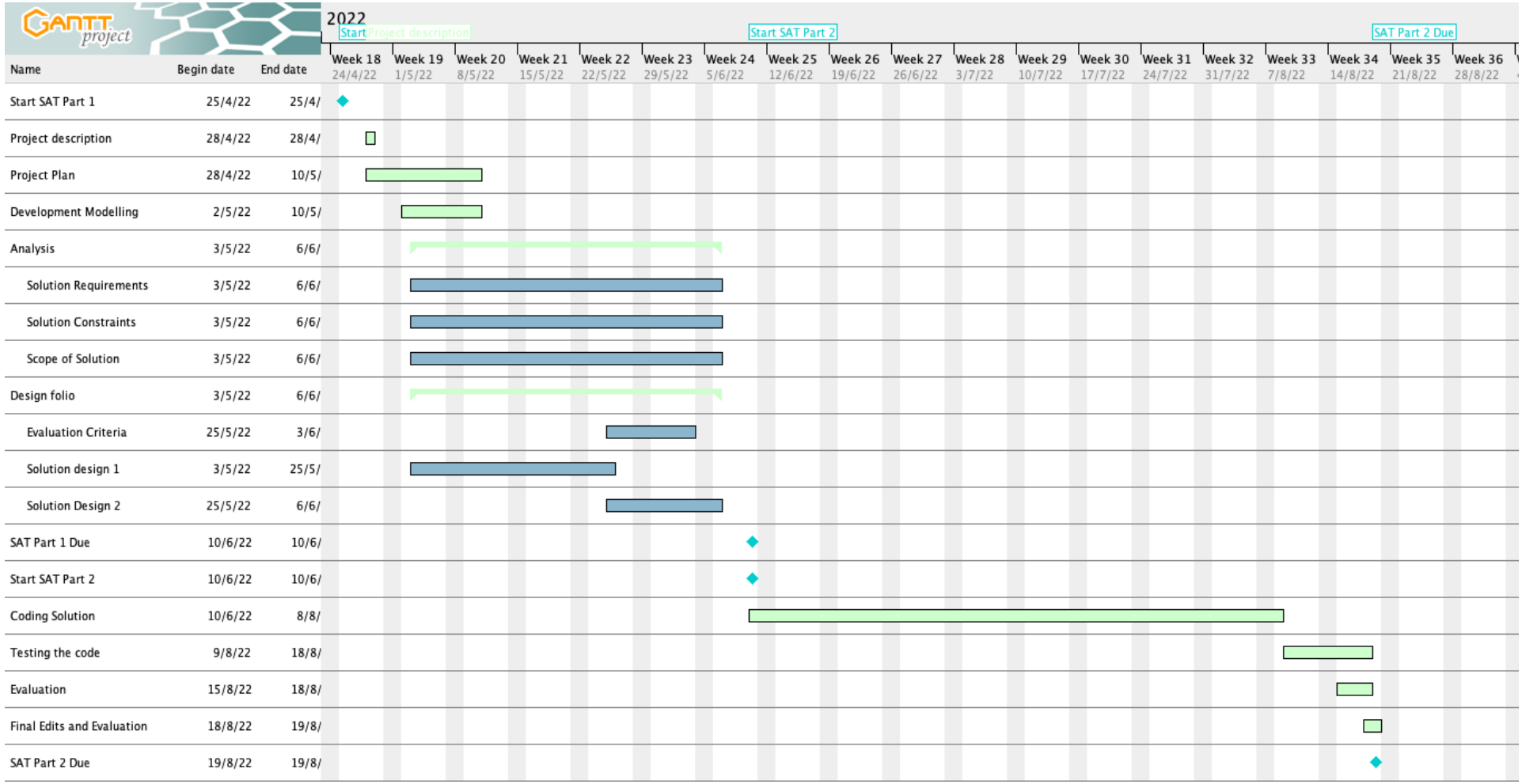
5.1 Modifications made to the initial plan

Original Timeline Table showing modifications

Task Description	WBS Number	Duration	Start Date	Due Date	Modified Start Date	Modified Due Date	Reason for change & impact on timeline	Critical Path or Dependency
Start Part 1	1.0	1 day	28/4/22	28/4/22				Critical Path
Project Description	1.1	1 day	28/4/22	28/4/22				Critical Path
Project Plan	1.2	12 days	28/4/22	10/5/22		2/5/22	Task was completed faster than anticipated	Critical Path
Development Modelling	1.3	8 days	2/5/22	10/5/22				Critical Path
Start Analysis	**2.0**	**1 day**	**3/5/22**	**3/5/22**				**Critical Path**
Analysis	2.1	35 days	3/5/22	6/6/22				Critical Path
Solution Requirements	2.2	35 days	3/5/22	6/6/22				Critical Path
Solution Constraints	2.3	35 days	3/5/22	6/6/22				Critical Path
Scope of Solution	2.4	35 days	3/5/22	6/6/22				Critical Path
Start Design Folio	**3.0**	**1 day**	**3/5/22**	**3/5/22**				**Critical Path**
Design Folio	3.1	35 days	3/5/22	6/6/22				Critical Path
Evaluation Criteria	3.2	10 days	25/5/22	3/6/22				Critical Path
Solution Design 1	3.3	22 days	3/5/22	25/5/22				Dependency
Solution Design 2	3.4	13 days	25/5/22	6/6/22				Dependency
SAT Part 1 Due	**4.0**	**1 day**	**10/6/22**	**10/6/22**				**Critical Path**
Start Part 2	5.0	1 day	13/6/22	13/6/22				Critical Path
Coding Solution	5.1	49 days	10/6/22	29/7/22		**6/8/22**	Developer illness – major impact on timeline	Critical Path

Testing the Code	5.2	3 days	29/7/22	1/8/22	**6/8/22**	**8/8/22**	Result of developer illness – major impact on timeline	Dependency
Begin Evaluation	**6.0**	**1 day**	**1/8/22**	**1/8/22**	**7/8/22**			**Dependency**
Evaluation	6.1	7 days	1/8/22	8/8/22				Dependency
Final Edits and Evaluation	6.2	14 days	8/8/22	22/8/22		**19/8/22**	**Due date was modified by teacher – Major impact on timeline**	Dependency
SAT Part 2 Due	**7.0**	**1 day**	**26/8/22**	**26/8/22**		**19/8/22**	**Due date was modified by teacher – Major impact on timeline**	**Critical Path**

5.2 Modified Gantt Chart



5.3 Effectiveness of the project plan

The project plan's modifications are reflected in this Gantt Chart shown. This project plan worked reasonably successfully, however, due to unforeseen circumstances, significant changes had to be made during Part B of the SAT. Developer illness and a changed due date greatly impacted the way the project plan was executed; however, the solution was able to be adapted to meet these changes. Switching from Python to Visual Basic aided the quick construction of the software's GUI in Visual Studio and allowed for the reuse and remodelling of older code, which was essential after time lost due to illness. Despite these changes, the project plan was followed with proportional times to the original and the project was still able to be completed in the time given.

Appendix A: Code

Code for software
GitHub link: https://github.com/macra08842/SAT.git

Form 1/Home Page:

```
Public Class HomePage

  'opens up the relevent pages
  Private Sub btnPlants_Click(sender As Object, e As EventArgs) Handles btnPlants.Click

    Me.Hide()
    PlantData.Show()

  End Sub

  Private Sub btnClient_Click(sender As Object, e As EventArgs) Handles btnClient.Click

    Me.Hide()
    Client_Data.Show()

  End Sub

  Private Sub btnManager_Click(sender As Object, e As EventArgs) Handles btnManager.Click

    Me.Hide()
    Manage.Show()

  End Sub

  Private Sub HomePage_Load(sender As Object, e As EventArgs) Handles MyBase.Load

    'username and login system for when the software loads - simple password

    Try
      Dim username = InputBox("Please enter your USERNAME", "Login")
      If username = "admin" Then
        Dim password = InputBox("Please enter your PASSWORD", "Login")
        If password = "admin" Then
          MessageBox.Show("Login successful - Click OK to continue", "information",
MessageBoxButtons.OK, MessageBoxIcon.Information)
        Else
          MessageBox.Show("Incorrect Password", "error", MessageBoxButtons.OK,
MessageBoxIcon.Error)
          Me.Close()
        End If
      Else
```

```
            MessageBox.Show("Incorrect Username", "error", MessageBoxButtons.OK,
MessageBoxIcon.Error)
            Me.Close()
          End If
        Catch ex As Exception
          MsgBox("An error has occurred: " & ex.Message & vbNewLine & "Please try again")
        End Try


    End Sub
End Class
```

Form 2/Plant Data:

```
Public Class PlantData

    'Define any variables that will be used
    Dim nameList(0) As String
    Dim plantType(0) As String
    Dim currentEntry As String
    Dim plantNumber(0)
    Dim NumOfChange As Integer
    Dim path As String = "E:\Back-upSAT\SAT\SATPartB\PlantData.csv"

    'Read the CSV function cell by cell and line by line
    Function ReadCSV()
        Try
            Using MyReader As New Microsoft.VisualBasic.FileIO.TextFieldParser(path)

                'Define the Necessary Variables:
                Dim cellTicker As Integer = 0
                Dim rowTicker As Integer = 0
                MyReader.TextFieldType = FileIO.FieldType.Delimited
                MyReader.SetDelimiters(",")
                Dim row As String()
                Dim listBoxEntry As String
                lbxDataBase.Items.Clear()

                'Loop Through Each Row:
                While Not MyReader.EndOfData
                    row = MyReader.ReadFields()
                    'Loop Through Each Cell:
                    For Each cell In row
                        cellTicker += 1
                        Select Case cellTicker
                            Case 1
                                If rowTicker = nameList.Length Then
                                    ReDim Preserve nameList(UBound(nameList) + 1)
                                End If
                                nameList(rowTicker) = cell
```

```
                    listBoxEntry += cell
                Case 2
                    If rowTicker = plantType.Length Then
                        ReDim Preserve plantType(UBound(plantType) + 1)
                        plantType(UBound(plantType)) = cell
                    Else
                        plantType(rowTicker) = cell
                    End If
                    listBoxEntry += ": " & cell
                Case 3
                    If rowTicker = plantNumber.Length Then
                        ReDim Preserve plantNumber(UBound(plantNumber) + 1)
                        plantNumber(UBound(plantNumber)) = cell
                    Else
                        plantNumber(rowTicker) = cell
                    End If
                    listBoxEntry += ", " & cell
                    lbxDataBase.Items.Add(listBoxEntry)
                    listBoxEntry = ""
                End Select
            Next
            cellTicker = 0
            rowTicker += 1
        End While
        If rowTicker = 0 Then
            lblResults.Text = "There are no current plant entries." & vbNewLine & "Add entries to the
CSV to get started."
        Else
            lblResults.Text = "There are " & (rowTicker - 1) & " plants logged in this CSV."
        End If
    End Using
    Catch ex As Exception
        Dim errorMessage = MsgBox("An error occurred while reading the CSV: " & ex.Message &
vbNewLine & "Please try again.", vbCritical)
    End Try
End Function

Function WriteToCSV(name, type, number)
    ReadCSV()
    'Add mark details to file
    currentEntry = vbNewLine & name & "," & type & "," & number
    My.Computer.FileSystem.WriteAllText(path, currentEntry, True)
    ReadCSV()
End Function

Private Sub btnBackP_Click(sender As Object, e As EventArgs) Handles btnBackP.Click
    'When clicked, this button sends the user back to the home page
    Me.Hide()
    HomePage.Show()

End Sub
```

```
    Private Sub Form2_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        lblSearchPlant.Text = "Loading File..."
        Using MyReader As New Microsoft.VisualBasic.FileIO.TextFieldParser(path)
            'DEFINE THE NECESSARY VARIABLES:
            Dim cellTicker As Integer = 0
            Dim rowTicker As Integer = 0
            Dim listBoxEntry As String
            MyReader.TextFieldType = FileIO.FieldType.Delimited
            MyReader.SetDelimiters(",")
            Dim row As String()

            'LOOP THROUGH EACH ROW
            While Not MyReader.EndOfData
                row = MyReader.ReadFields()
                'LOOP THROUGH EACH CELL
                If rowTicker > 0 Then
                    For Each cell In row
                        cellTicker += 1
                        Select Case cellTicker
                            Case 1
                                ReDim Preserve nameList(UBound(nameList) + 1)
                                nameList(UBound(nameList)) = cell
                                listBoxEntry += cell
                            Case 2
                                ReDim Preserve plantNumber(UBound(plantNumber) + 1)
                                plantNumber(UBound(plantNumber)) = cell
                                listBoxEntry += ": " & cell
                            Case 3
                                ReDim Preserve plantType(UBound(plantType) + 1)
                                plantType(UBound(plantType)) = cell
                                listBoxEntry += ", " & cell

                        End Select
                    Next
                    cellTicker = 0
                End If
                rowTicker += 1
            End While
        End Using
        lblSearchPlant.Text = "CSV File Loaded Into Array"
    End Sub

    'when the software loads, read the CSV

    Private Sub PlantData_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        ReadCSV()
    End Sub

    Private Sub btnSearchPlants_Click(sender As Object, e As EventArgs) Handles
btnSearchPlants.Click
```

```
    Dim nameToSearch = InputBox("Let's Find Your Entry." & vbNewLine & "Enter Plant Name: ")
    Dim locator As Integer = 0
    Dim found As Boolean = False
    'goes through each line of the code until a match is found
    For Each plant In nameList
      lblSearchPlant.Text = "Searching..."
      locator += 1
      If plant = nameToSearch Then
        lblSearchPlant.Text = "Plant " & nameToSearch & " Found!"
        Dim location As Integer = locator - 1
        lblResults.Text = nameToSearch & "  Results:" & vbNewLine & "Number Avaliable: " &
plantType(location) & vbNewLine & "Type of plant: " & plantNumber(location)
        found = True
        Exit For
      End If
    Next
    If found = False Then
      lblSearchPlant.Text = "Plant not found" & vbNewLine & "Please try again"
    End If
  End Sub
  'writes to the bottom of the CSV in an empty field
  Private Sub btnSaveNew_Click(sender As Object, e As EventArgs) Handles btnSaveNew.Click
    Dim name As String = tbxNewPlant.Text
    'Check for Empty Field:
    If name = "" Then
      MsgBox(" Please enter a PLANT NAME in the text box above", vbCritical)
    Else
      Try
        Dim plantType = InputBox("Please enter the TYPE of plant that " + name + " is: ")
        If plantType = "" Then
          MsgBox("Please enter a PLANT TYPE in the text box ", vbCritical)
        Else
          Dim plantNumber As String = InputBox("Please enter the QUANTITY of " + name + "
currently in stock ")
          If IsNumeric(plantNumber) And plantNumber >= 0 And plantNumber <= 100000 Then
            WriteToCSV(name, plantNumber, plantType)
          Else
            MsgBox(" Please enter the PLANT QUANTITY as a positive number 100000 or less ",
vbCritical)
          End If
        End If
      Catch ex As Exception
        MsgBox("An error has occurred: " & ex.Message & vbNewLine & "Please try again")
      End Try
    End If

  End Sub


  'Clears the text box
  Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
```

```
    lblResults.Text = "Results"
  End Sub

  Private Sub btnCleartbx_Click(sender As Object, e As EventArgs) Handles btnCleartbx.Click
    tbxNewPlant.Clear()
  End Sub

  Private Sub btnClearModify_Click(sender As Object, e As EventArgs) Handles btnClearModify.Click
    tbxNameChange.Clear()
  End Sub

  'Delete Record within CSV
  Private Sub btnDelete_Click(sender As Object, e As EventArgs) Handles btnDelete.Click


    Dim nameToModify As String = tbxNameChange.Text
    Dim locator As Integer = 0
    Dim found As Boolean = False
    Dim tempDFile As String = "tempD.csv"
    Dim deleteRecord As Boolean = False
    Dim delimiter As String = ","
    ' Dim newRecord As String = nameList + delimiter + plantType + delimiter + plantNumber

    For Each plant In nameList
      locator += 1
      If plant = nameToModify Then
        'Dim location As Integer = locator - 1
        found = True
        Exit For
      End If
    Next
    Dim dYes As MsgBoxResult = MsgBox(nameToModify & " found! Would you like to delete this
entry?", vbYesNo)
    If found = False Then
      MsgBox("Searched Item: " & nameToModify & " was not found. Please Try again")
    End If

    If dYes = MsgBoxResult.Yes And found = True Then
      Try
        Dim fileReader As New System.IO.StreamReader(path)
        Dim fileWriter As New System.IO.StreamWriter(tempDFile, True)

        Do While fileReader.Peek() <> -1
          currentEntry = fileReader.ReadLine()
          Dim currentRecord() As String = Split(currentEntry, ",")
          'compare the read position with the name to modify, and if it equals tue, write it to the
temp file
          If (Not String.Compare(currentRecord(locator), nameToModify) = 0) Then
            fileWriter.WriteLine(currentEntry)

          Else
```

```
                deleteRecord = True

            End If
        Loop

        fileWriter.Close()
        fileReader.Close()
        ' delete original file, and replace the temp file details to path's
        My.Computer.FileSystem.DeleteFile(path)
        My.Computer.FileSystem.RenameFile(tempDFile, path)


      Catch ex As Exception
        MsgBox("An error has occured: " & ex.Message & vbNewLine & "Please Try again")
      End Try
    End If

    ' Return deleteRecord

  End Sub
  'Modify Record
  Private Sub btnModify_Click(sender As Object, e As EventArgs) Handles btnModify.Click
    Dim nameToModify As String = tbxNameChange.Text
    Dim locator As Integer = 0
    Dim found As Boolean = False
    Dim tempMFile As String = "tempM.csv"

    Dim editedRecord As Boolean = False
    Dim deleteRecord As Boolean = False
    Dim delimiter As String = ","
    ' Dim newRecord As String = nameList + delimiter + plantType + delimiter + plantNumber

    For Each plant In nameList
      locator += 1
      If plant = nameToModify Then
        Dim location As Integer = locator - 1
        found = True
        Exit For
      End If
    Next
    Dim dYes As MsgBoxResult = MsgBox(nameToModify & " found! Would you like to modify this
entry?", vbYesNo)
    If found = False Then
      MsgBox("Searched Item: " & nameToModify & " was not found. Please Try again")
    End If
    If dYes = MsgBoxResult.Yes And found = True Then
      Try
        'deletes the selected record
        Dim fileReader As New System.IO.StreamReader(path)
        Dim fileWriter As New System.IO.StreamWriter(tempMFile, True)
```

```
            Do While fileReader.Peek() <> -1
                currentEntry = fileReader.ReadLine()
                Dim currentRecord() As String = Split(currentEntry, ",")
                If (Not String.Compare(currentRecord(locator), nameToModify) = 0) Then
                    fileWriter.WriteLine(currentEntry)

                Else
                    deleteRecord = True

                End If
            Loop

            fileWriter.Close()
            fileReader.Close()

            My.Computer.FileSystem.DeleteFile(path)
            My.Computer.FileSystem.RenameFile(tempMFile, path)

        Catch ex As Exception
            MsgBox("An error has occured: " & ex.Message & vbNewLine & "Please Try again")
        End Try
        'writes new record to replace the old one
        Dim name = InputBox("Please enter the PLANT NAME that you want to edit")
        If name = "" Then
            MsgBox(" Please enter a PLANT NAME in the text box above", vbCritical)
        Else
            Try
                Dim plantType = InputBox("Please enter the TYPE of plant that " + name + " is: ")
                If plantType = "" Then
                    MsgBox("Please enter a PLANT TYPE in the text box ", vbCritical)
                Else
                    Dim plantNumber As String = InputBox("Please enter the QUANTITY of " + name + "
currently in stock ")
                    If IsNumeric(plantNumber) And plantNumber >= 0 And plantNumber <= 100000 Then
                        WriteToCSV(name, plantNumber, plantType)
                    Else
                        MsgBox(" Please enter the PLANT QUANTITY as a positive number 100000 or less ",
vbCritical)
                    End If
                End If
            Catch ex As Exception
                MsgBox("An error has occurred: " & ex.Message & vbNewLine & "Please try again")
            End Try
        End If
    End If
  End Sub
End Class
```

Client Data:

```
Public Class Client_Data
  'note that a lot of the functions here are simular to that in plantData

  Dim clientName(0) As String
  Dim plantType(0) As String
  Dim plantName(0) As String
  Dim currentEntry As String
  Dim plantNumber(0)
  Dim NumOfChange As Integer
  Dim path As String = "E:\Back-upSAT\SAT\SATPartB\ClientData.csv"

  Function ReadCSV()
    Try
      Using MyReader As New Microsoft.VisualBasic.FileIO.TextFieldParser(path)

        'Define the Necessary Variables:
        Dim cellTicker As Integer = 0
        Dim rowTicker As Integer = 0
        MyReader.TextFieldType = FileIO.FieldType.Delimited
        MyReader.SetDelimiters(",")
        Dim row As String()
        Dim listBoxEntry As String
        lbxClientSearch.Items.Clear()

        'Loop Through Each Row:
        While Not MyReader.EndOfData
          row = MyReader.ReadFields()
          'Loop Through Each Cell:
          For Each cell In row
            cellTicker += 1
            Select Case cellTicker
              Case 1
                If rowTicker = clientName.Length Then
                  ReDim Preserve clientName(UBound(clientName) + 1)
                End If
                clientName(rowTicker) = cell
                listBoxEntry += cell
              Case 2
                If rowTicker = plantType.Length Then
                  ReDim Preserve plantType(UBound(plantType) + 1)
                  plantType(UBound(plantType)) = cell
                Else
                  plantType(rowTicker) = cell
                End If
                listBoxEntry += ": " & cell
              Case 3
                If rowTicker = plantNumber.Length Then
                  ReDim Preserve plantNumber(UBound(plantNumber) + 1)
                  plantNumber(UBound(plantNumber)) = cell
                Else
                  plantNumber(rowTicker) = cell
```

```
                End If
                listBoxEntry += ", " & cell
            Case 4
                If rowTicker = plantName.Length Then
                    ReDim Preserve plantName(UBound(plantName) + 1)
                    plantName(UBound(plantName)) = cell
                Else
                    plantNumber(rowTicker) = cell
                End If
                listBoxEntry += ", " & cell
                lbxClientSearch.Items.Add(listBoxEntry)
                listBoxEntry = ""
            End Select
        Next
        cellTicker = 0
        rowTicker += 1
    End While
    If rowTicker = 0 Then
        lblSearchClient.Text = "There are no current plant entries." & vbNewLine & "Add entries
to the CSV to get started."
    Else
        lblSearchClient.Text = "There are " & (rowTicker - 1) & " plants logged in this CSV."
    End If
  End Using
Catch ex As Exception
  Dim errorMessage = MsgBox("An error occurred while reading the CSV: " & ex.Message &
vbNewLine & "Please try again.", vbCritical)
End Try
End Function

Private Sub btnBackC_Click(sender As Object, e As EventArgs) Handles btnBackC.Click

  Me.Hide()
  HomePage.Show()

End Sub

Private Sub Form2_Load(sender As Object, e As EventArgs) Handles MyBase.Load
  lblSearchClient.Text = "Loading File..."
  Using MyReader As New Microsoft.VisualBasic.FileIO.TextFieldParser(path)
    'DEFINE THE NECESSARY VARIABLES:
    Dim cellTicker As Integer = 0
    Dim rowTicker As Integer = 0
    Dim listBoxEntry As String
    MyReader.TextFieldType = FileIO.FieldType.Delimited
    MyReader.SetDelimiters(",")
    Dim row As String()

    'LOOP THROUGH EACH ROW
    While Not MyReader.EndOfData
      row = MyReader.ReadFields()
```

```
            'LOOP THROUGH EACH CELL
            If rowTicker > 0 Then
                For Each cell In row
                    cellTicker += 1
                    Select Case cellTicker
                        Case 1
                            ReDim Preserve clientName(UBound(clientName) + 1)
                            clientName(UBound(clientName)) = cell
                            listBoxEntry += cell
                        Case 2
                            ReDim Preserve plantName(UBound(plantName) + 1)
                            plantName(UBound(plantName)) = cell
                            listBoxEntry += ": " & cell
                        Case 3
                            ReDim Preserve plantType(UBound(plantType) + 1)
                            plantType(UBound(plantType)) = cell
                            listBoxEntry += ", " & cell
                        Case 4
                            ReDim Preserve plantNumber(UBound(plantNumber) + 1)
                            plantNumber(UBound(plantNumber)) = cell
                            listBoxEntry += ", " & cell
                            lbxClientSearch.Items.Add(listBoxEntry)
                            listBoxEntry = ""
                    End Select
                Next
                cellTicker = 0
            End If
            rowTicker += 1
        End While
    End Using
    lblSearchClient.Text = "CSV File Loaded Into Array"
End Sub

'writes to the CSV (function only)
Function WriteToCSV(name, plantName, plantType, plantNumber)
    ReadCSV()
    currentEntry = vbNewLine & name & "," & plantName & "," & plantType & "," & plantNumber
    My.Computer.FileSystem.WriteAllText(path, currentEntry, True)
    ReadCSV()
End Function

'enter new data
Private Sub btnCDatabase_Click(sender As Object, e As EventArgs) Handles btnCDatabase.Click

    Dim name As String = tbxClientName.Text
    'Check for Empty Field:
    If name = "" Then
        MsgBox(" Please enter a CLIENT NAME in the text box above", vbCritical)
    Else
        Try
```

```
            Dim plantName = InputBox(" Please enter the NAME of the plant that " + name + " has
ordered ")
            If plantName = "" Then
               MsgBox("Please enter a PLANT NAME into the text box", vbCritical)
            Else
               Dim plantType = InputBox("Please enter the TYPE of plant that " + plantName + " is ")
               If plantType = "" Then
                  MsgBox(" Please enter a PLANT TYPE in the text box", vbCritical)
               Else
                  Dim plantNumber As String = InputBox("Please enter the QUANTITY of " + plantName +
" ordered ")
                  If IsNumeric(plantNumber) And plantNumber >= 0 And plantNumber <= 100000 Then
                     WriteToCSV(name, plantName, plantType, plantNumber)
                  End If
               End If
            End If
         Catch ex As Exception
            MsgBox("An error has occured: " & ex.Message & vbNewLine & "Please try again")
         End Try
      End If

   End Sub

   'searches the function - same as for plant data
   Private Sub btnEnterSearch_Click(sender As Object, e As EventArgs) Handles btnEnterSearch.Click
      Dim nameToSearch = InputBox("Let's Find Your Entry." & vbNewLine & "Enter Client Name: ")
      Dim locator As Integer = 0
      Dim found As Boolean = False
      For Each client In clientName
         lblSearchClient.Text = "Searching..."
         locator += 1
         If client = nameToSearch Then
            lblSearchClient.Text = "Client: " & nameToSearch & " Found!"
            Dim location As Integer = locator - 1
            lblResults.Text = nameToSearch & "'s Orders are:" & vbNewLine & "Plant Type: " &
plantType(location) & vbNewLine & "Plant Name: " & plantName(location) & vbNewLine & "Plant
Quantity: " & plantNumber(location)
            found = True
            Exit For
         End If
      Next
      If found = False Then
         lblSearchClient.Text = "Client Not Found" & vbNewLine & "Please try again"
      End If
   End Sub

   Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
      tbxClientName.Clear()
   End Sub
End Class
```

Management:

```
Public Class Manage

    Dim nameListP(0) As String
    Dim plantTypeP(0) As String
    Dim currentEntryP As String
    Dim plantNumberP(0)
    Dim NumOfChangeP As Integer
    Dim clientName(0) As String
    Dim plantTypeC(0) As String
    Dim plantNameC(0) As String
    Dim currentEntryC As String
    Dim plantNumberC(0)
    Dim NumOfChangeC As Integer
    Dim pathPlants As String = "E:\Back-upSAT\SAT\SATPartB\PlantData.csv"
    Dim pathClients As String = "E:\Back-upSAT\SAT\SATPartB\ClientData.csv"

    'reading the plants CSV
    Function ReadPCSV()
        Try
            Using MyReader As New Microsoft.VisualBasic.FileIO.TextFieldParser(pathPlants)

                'Define the Necessary Variables:
                Dim cellTicker As Integer = 0
                Dim rowTicker As Integer = 0
                MyReader.TextFieldType = FileIO.FieldType.Delimited
                MyReader.SetDelimiters(",")
                Dim row As String()
                Dim listBoxEntry As String
                lbxPlantD.Items.Clear()

                'Loop Through Each Row:
                While Not MyReader.EndOfData
                    row = MyReader.ReadFields()
                    'Loop Through Each Cell:
                    For Each cell In row
                        cellTicker += 1
                        Select Case cellTicker
                            Case 1
                                If rowTicker = nameListP.Length Then
                                    ReDim Preserve nameListP(UBound(nameListP) + 1)
                                End If
                                nameListP(rowTicker) = cell
                                listBoxEntry += cell
                            Case 2
                                If rowTicker = plantTypeP.Length Then
```

```
                    ReDim Preserve plantTypeP(UBound(plantTypeP) + 1)
                    plantTypeP(UBound(plantTypeP)) = cell
                  Else
                    plantTypeP(rowTicker) = cell
                  End If
                  listBoxEntry += ": " & cell
                Case 3
                  If rowTicker = plantNumberP.Length Then
                    ReDim Preserve plantNumberP(UBound(plantNumberP) + 1)
                    plantNumberP(UBound(plantNumberP)) = cell
                  Else
                    plantNumberP(rowTicker) = cell
                  End If
                  listBoxEntry += ", " & cell
                  lbxPlantD.Items.Add(listBoxEntry)
                  listBoxEntry = ""
              End Select
            Next
            cellTicker = 0
            rowTicker += 1
          End While
          If rowTicker = 0 Then
            lbxPlantD.Text = "There are no current plant entries." & vbNewLine & "Add entries to the
CSV to get started."
          Else
            lbxPlantD.Text = "There are" & (rowTicker - 1) & "plants logged in this CSV."
          End If
        End Using
      Catch ex As Exception
        Dim errorMessage = MsgBox("An error occurred while reading the CSV: " & ex.Message &
vbNewLine & "Please try again.", vbCritical)
      End Try
    End Function

    Function ReadCCSV()
      Try
        Using MyReader As New Microsoft.VisualBasic.FileIO.TextFieldParser(pathClients)

          'Define the Necessary Variables:
          Dim cellTicker As Integer = 0
          Dim rowTicker As Integer = 0
          MyReader.TextFieldType = FileIO.FieldType.Delimited
          MyReader.SetDelimiters(",")
          Dim row As String()
          Dim listBoxEntry As String
          lbxClientD.Items.Clear()

          'Loop Through Each Row:
          While Not MyReader.EndOfData
            row = MyReader.ReadFields()
            'Loop Through Each Cell:
```

```
                For Each cell In row
                    cellTicker += 1
                    Select Case cellTicker
                        Case 1
                            If rowTicker = clientName.Length Then
                                ReDim Preserve clientName(UBound(clientName) + 1)
                            End If
                            clientName(rowTicker) = cell
                            listBoxEntry += cell
                        Case 2
                            If rowTicker = plantTypeC.Length Then
                                ReDim Preserve plantTypeC(UBound(plantTypeC) + 1)
                                plantTypeC(UBound(plantTypeC)) = cell
                            Else
                                plantTypeC(rowTicker) = cell
                            End If
                            listBoxEntry += ": " & cell
                        Case 3
                            If rowTicker = plantNumberC.Length Then
                                ReDim Preserve plantNumberC(UBound(plantNumberC) + 1)
                                plantNumberC(UBound(plantNumberC)) = cell
                            Else
                                plantNumberC(rowTicker) = cell
                            End If
                            listBoxEntry += ", " & cell
                        Case 4
                            If rowTicker = plantNameC.Length Then
                                ReDim Preserve plantNameC(UBound(plantNameC) + 1)
                                plantNameC(UBound(plantNameC)) = cell
                            Else
                                plantNumberC(rowTicker) = cell
                            End If
                            listBoxEntry += ", " & cell
                            lbxClientD.Items.Add(listBoxEntry)
                            listBoxEntry = ""
                    End Select
                Next
                cellTicker = 0
                rowTicker += 1
            End While
            If rowTicker = 0 Then
                lbxClientD.Text = "There are no current plant entries." & vbNewLine & "Add entries to
the CSV to get started."
            Else
                lbxClientD.Text = "There are" & (rowTicker - 1) & "plants logged in this CSV."
            End If
        End Using
    Catch ex As Exception
        Dim errorMessage = MsgBox("An error occurred while reading the CSV: " & ex.Message &
vbNewLine & "Please try again.", vbCritical)
    End Try
```

```
End Function

'send back to homepage
Private Sub btnBackM_Click(sender As Object, e As EventArgs) Handles btnBackM.Click

    Me.Hide()
    HomePage.Show()

End Sub

'send to plantdata page to search
Private Sub lblSearchPlant_Click(sender As Object, e As EventArgs) Handles lblSearchPlant.Click

    Me.Hide()
    PlantData.Show()

End Sub

Private Sub btnPSort_Click(sender As Object, e As EventArgs) Handles btnPSort.Click

    'SORT plants CSV BUTTON

    ReadPCSV()
    Dim tempSortArray(1)
    Dim rowTicker As Integer = 0
    For Each plant In nameListP
        If rowTicker > 0 Then
            ReDim Preserve tempSortArray(tempSortArray.Length)
        End If
        rowTicker += 1
        tempSortArray(rowTicker) = plant
    Next
    rowTicker = 0
    For Each element In plantTypeP
        rowTicker += 1
        tempSortArray(rowTicker) += "," & element
    Next
    rowTicker = 0
    For Each element In plantNumberP
        rowTicker += 1
        tempSortArray(rowTicker) += "," & element
    Next
    Array.Sort(tempSortArray, 2, tempSortArray.Length - 2)
    My.Computer.FileSystem.WriteAllText(pathPlants, Join(tempSortArray, vbNewLine), False)
    ReadPCSV()


End Sub

Private Sub btnSearchC_Click(sender As Object, e As EventArgs) Handles btnSearchC.Click
```

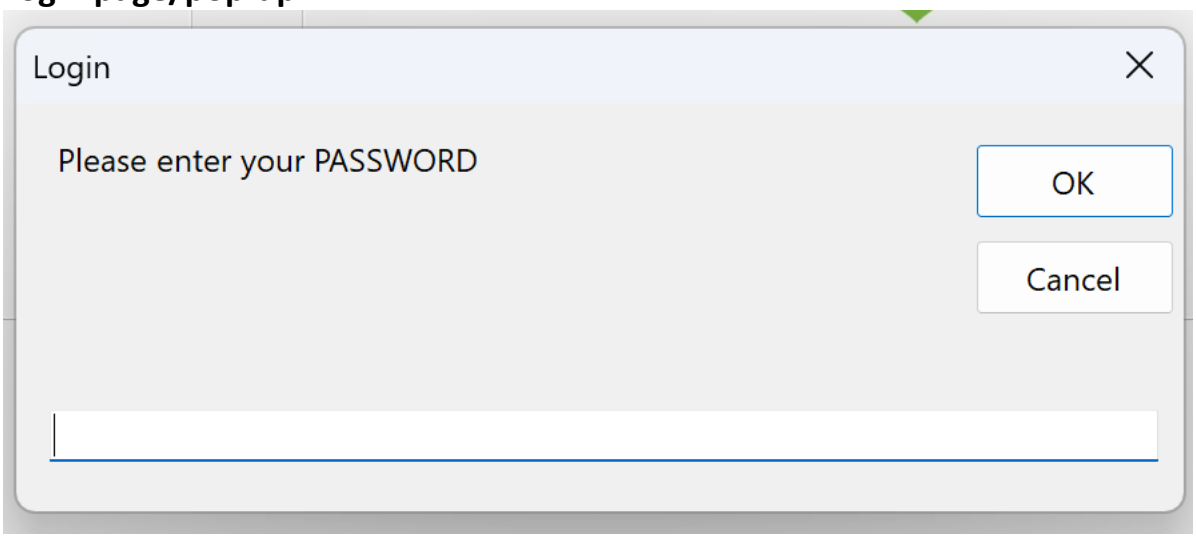
```
        Me.Hide()
        Client_Data.Show()

    End Sub

    Private Sub btnSortC_Click(sender As Object, e As EventArgs) Handles btnSortC.Click

        'SORT clients CSV BUTTON
        ReadCCSV()
        Dim tempSortArray(1)
        Dim rowTicker As Integer = 0
        For Each client In clientName
            If rowTicker > 0 Then
                ReDim Preserve tempSortArray(tempSortArray.Length)
            End If
            rowTicker += 1
            tempSortArray(rowTicker) = client
        Next
        rowTicker = 0
        For Each order In plantNameC
            rowTicker += 1
            tempSortArray(rowTicker) += "," & order
        Next
        rowTicker = 0
        For Each order In plantTypeC
            rowTicker += 1
            tempSortArray(rowTicker) += "," & order
        Next
        rowTicker = 0
        For Each order In plantNumberC
            rowTicker += 1
            tempSortArray(rowTicker) += "," & order
        Next
        Array.Sort(tempSortArray, 2, tempSortArray.Length - 2)
        My.Computer.FileSystem.WriteAllText(pathClients, Join(tempSortArray, vbNewLine), False)
        ReadCCSV()

    End Sub

    Function clearPCSV()
        'clears/deletes the WHOLE CSV of information
        Try
            My.Computer.FileSystem.WriteAllText(pathPlants, "Plant Name, Plant Quantity, Plant Type",
False)
        Catch ex As Exception
            MsgBox("ERROR: " & ex.Message, vbCritical, "ERROR")
        End Try
        lbxPlantD.Items.Clear()
        lbxPlantD.Text = "There's nothing here yet! Add entries to the CSV to get started."
        Array.Clear(nameListP, 1, nameListP.Length - 1)
        Array.Clear(plantNumberP, 1, plantNumberP.Length - 1)
```

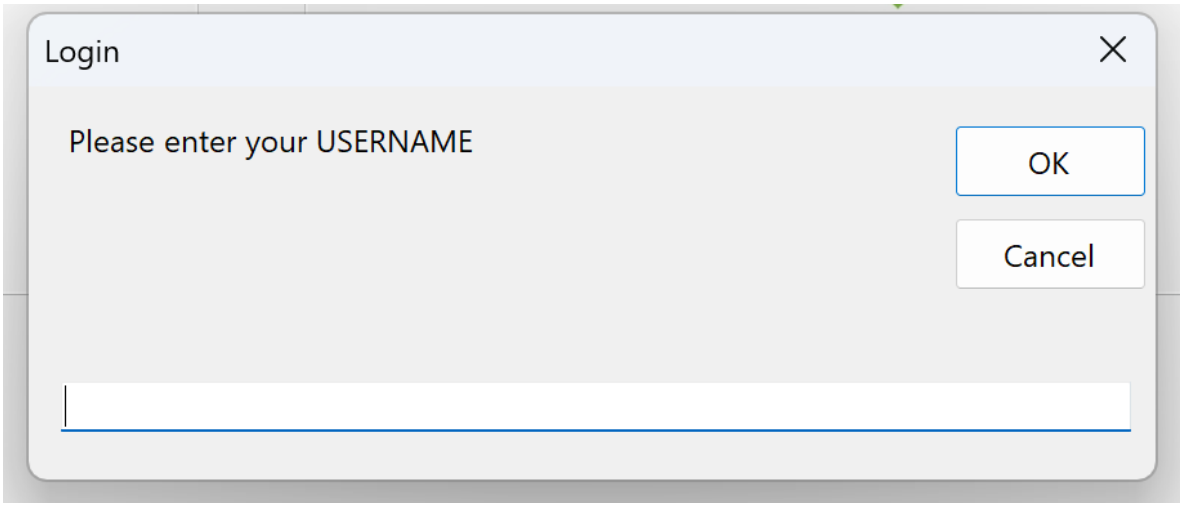
```
        Array.Clear(plantTypeP, 1, plantTypeP.Length - 1)
        ReadPCSV()
    End Function

    Function clearCCSV()
        'clears/deletes the WHOLE CSV of information
        Try
            My.Computer.FileSystem.WriteAllText(pathClients, "Client Name, Plant Name, Plant Type,
Plant Quantity", False)
        Catch ex As Exception
            MsgBox("ERROR: " & ex.Message, vbCritical, "ERROR")
        End Try
        lbxClientD.Items.Clear()
        lbxClientD.Text = "There's nothing here yet! Add entries to the CSV to get started."
        Array.Clear(clientName, 1, clientName.Length - 1)
        Array.Clear(plantNameC, 1, plantNameC.Length - 1)
        Array.Clear(plantTypeC, 1, plantTypeC.Length - 1)
        Array.Clear(plantNumberC, 1, plantNumberC.Length - 1)
        ReadCCSV()
    End Function

    Private Sub btnClearPData_Click(sender As Object, e As EventArgs) Handles btnClearPData.Click
        'connects the clear function to a button
        Dim sure As MsgBoxResult = MsgBox($"Are you sure you want to clear the
CSV?{vbNewLine}This will permanently erase all data!", vbYesNo)
        If sure = MsgBoxResult.Yes Then
            clearPCSV()
            MsgBox("CSV data cleared")
        Else
            MsgBox("Oops - let's get you back to that normal screen")
        End If
    End Sub

    Private Sub btnClearCData_Click(sender As Object, e As EventArgs) Handles btnClearCData.Click
        'connects the clear function to the button
        Dim sure As MsgBoxResult = MsgBox($"Are you sure you want to clear the
CSV?{vbNewLine}This will permanently erase all data!", vbYesNo)
        If sure = MsgBoxResult.Yes Then
            clearCCSV()
            MsgBox("CSV data cleared")
        Else
            MsgBox("Oops - let's get you back to that normal screen")
        End If
    End Sub
End Class
```

Appendix B: GUI Screenshots

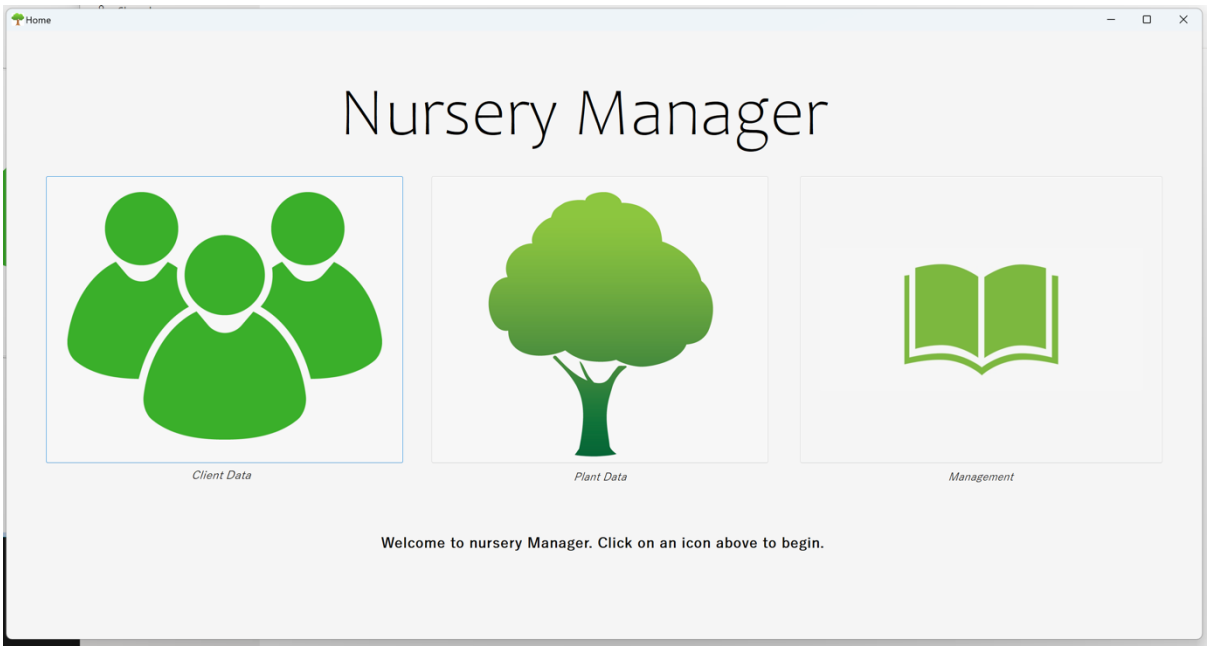
Photos of GUI in the software

Login page/pop-up:

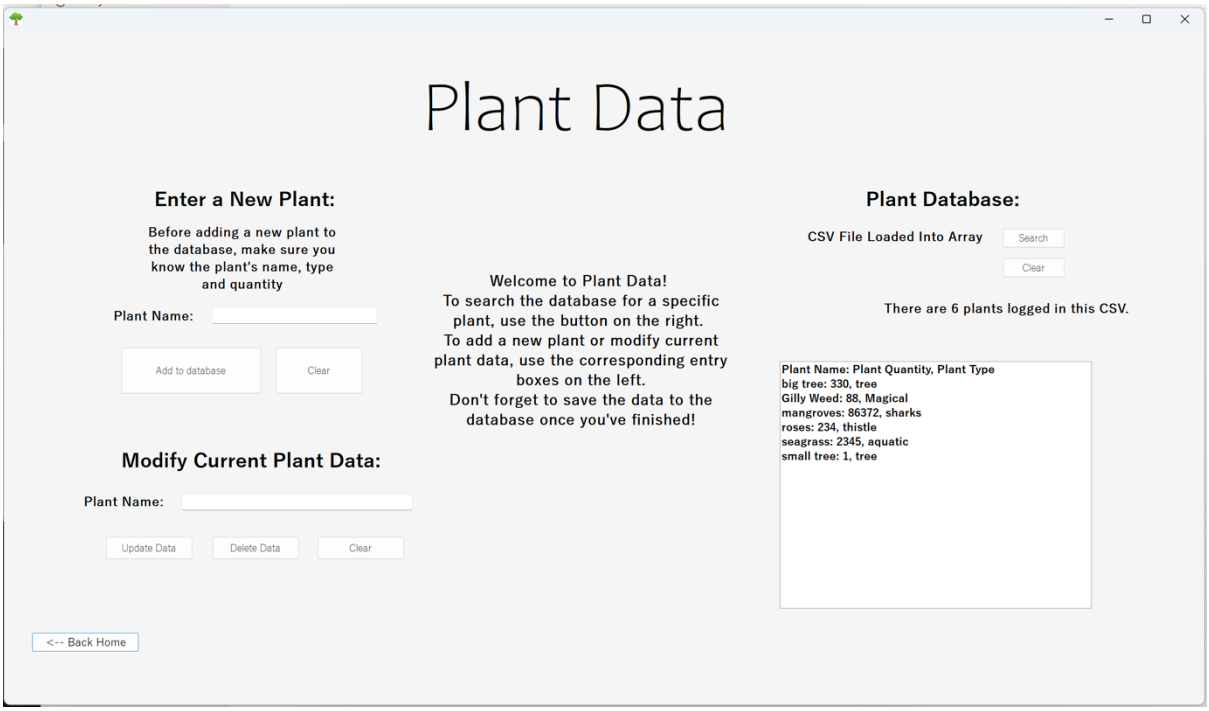




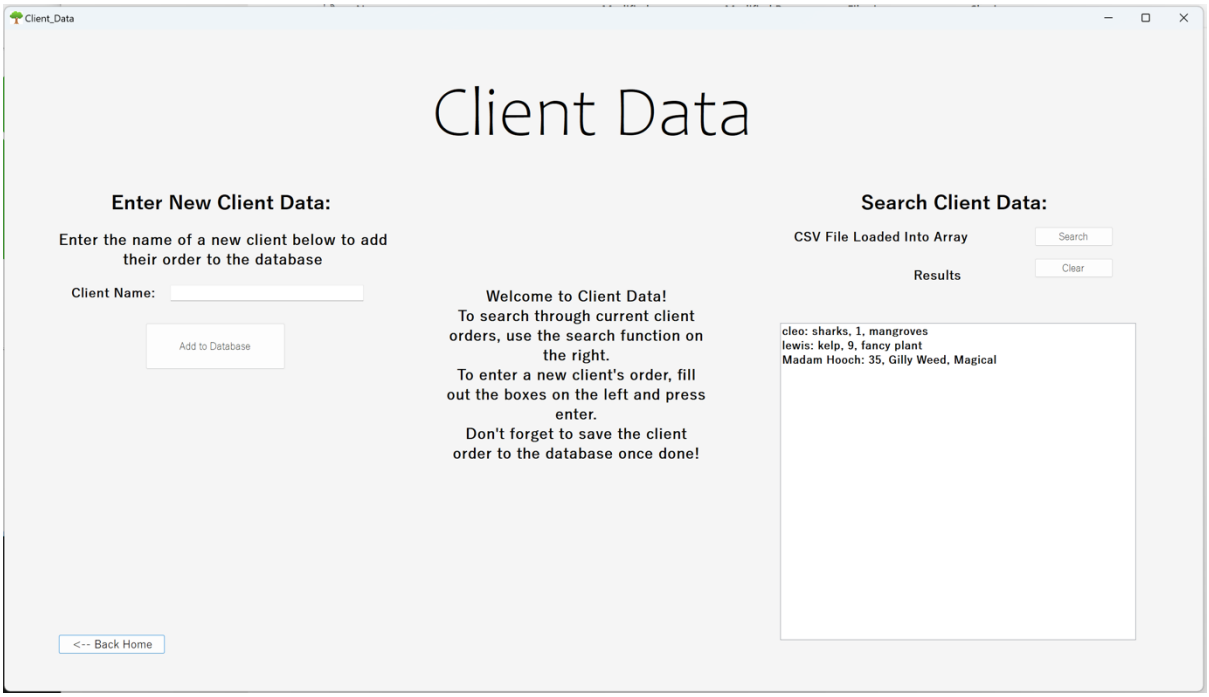
Home page:



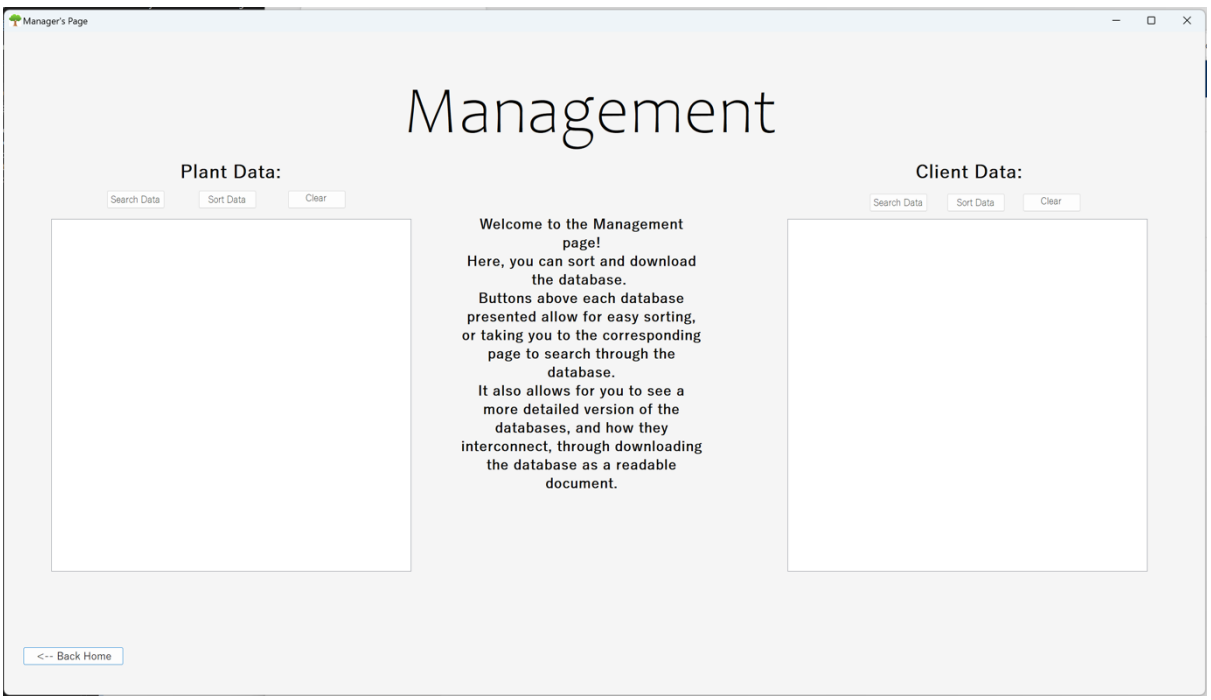
Plant data page:



Client data page:



Management page:



Appendix C: Tester Usability Table

Tester Usability Table

Item Tested	Procedure/Instructions	Success?	Comments
Logging into the software	Enter the set username and password (admin), and click enter/ok to open the homepage	yes	Process was straight forward. It was great and easy to use. Hide the password in the future with dots.
Button to plant data page	Click plant data button on the homepage	yes	Easy to find – pleasing graphics. Found the button using the picture.
Button to client data page	Click client data button on the home screen	yes	Easy to find – pleasing graphics. Found the button using the picture
Button to management page	Click management button on the homepage	yes	Easy to find – pleasing graphics. Found the button using the picture.
Button to home page	Click back to homepage button to return to the homepage	yes	Easy. The button was read easily.
Add client button	Click button after entering name into the text box and fill in the corresponding input boxes until data is submitted to database. Otherwise, an error will appear on the screen	yes	The headings/ columns for the CSV only appeared after a new input. It's easy. Questioned where everything was.
Enter new plant button	Click button **after** entering name into the text box and fill in the corresponding input boxes until data is submitted to database. Otherwise, an error will appear on the screen	yes	Error was confusing when words were put in instead of numbers. Instructions about clear button for input box would be helpful. Maybe have all the input boxes on the same page.
Search button (client data or plant data pages)	Click the button after entering the data that is wanted to be searched	yes	Took a moment to find the search button. Relatively

			easy to use. Sometimes the objects on screen moved in the wrong way.
Sort button (management page)	Click button to sort the CSV in alphabetical order, displayed on the screen	yes	Did what it was meant to do – perhaps display CSV prior to sorting as well. Consider sorting for other values in future iterations. You can see the names
Search button (management page)	Click the button to go to the corresponding screen to search	yes	Worked, as it brought it back to the corresponding search page, however, in future iterations maybe have it on the same screen.
Clear CSV (the same procedure and comments for client and plants)	Click the 'clear' button on management page, click yes for the warning, and then the CSV will be cleared of all data.	yes	Easy to do with good warnings before
Modification (currently not in use)	To be included in a future iteration	NO	If it is not working, consider having it greyed out or not there to avoid user confusion. (affordance)

Appendix D: Tester Questionnaire

Tester Questionnaire – Combined Answers

Please answer the questions below using the following scale:

1: Very Hard 2: Hard 3: Neither hard nor easy 4: Easy 5: Very Easy

1. How easy is the program to read? Average: 4
2. How easy is the program to use? Average: 4
3. How easy is it to add an item to the database? Average: 4.75
4. How easy is it to delete the database? Average: 5
5. How easy is it to login? Average: 5
6. How easy is it to search the database? Average: 4.25
7. How easy is it to sort the database alphabetically? Average: 5
8. How easy is it to navigate the software? Average: 4.25

Please answer the following questions below:

1. In general, how easy is the software to navigate?

Reasonably easy, although at times the buttons or text boxes could have been easier to locate. It was easy to navigate. Finding the 'back home' button was difficult

2. Describe how well the functions worked.

The functions worked as expected, and the inclusion of warnings in the message boxes (eg. The warning before deleting the CSV) were extra helpful as they ensured that I wasn't about to do something that wasn't what I intended. The error that popped up when words (a string) was put into a place where numbers are put (integer value), a confusing error popped up.

3. What would you like to see in a future update to easy usability?

Some of the fonts were quite thin and therefore difficult to read, so making them larger would be helpful. Add photos of the plants when the search results come up.

4. Any additional comments?

The program was easy to navigate, and the home page had pleasing graphics that represented the forms well. The user loved plants.

Developer Use Only:

- Average time taken to complete the test: 10.5 minutes
- Average number of errors made: 0-1 – but some testers asked have clarifying questions to avoid making errors, eg. 'Do I click here next?'
- Average questions asked to developer while testing: 3 (one tester had significantly more questions than all the others)

    - Example of types of questions asked: After client name entered: ‘Can I add a new plant, or does it already have to be in the database?’
- Success/Failure of Test: Partial success = tester needs to ask questions and/or makes a mistake but still completes task
- Developer observations:
    - The teenage tester was able to navigate the software’s GUI much more quickly, and without asking questions, compared to the adult testers