Farming Program

Software Requirements Specification

Version 0.4

18 December 2014

1312754 Program Manager

1307211 Lead Designer

1333187 Lead Programmer

Thomas Tester

1311808 Tester

Prepared for

MOD003263—Software Engineering

Lecturer: Cristina Luca, Ph.D.

Fall 2014

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Description** | **Author** | **Comments** |
| 12/10/14 | Version 0.1 | 1312754 | Initial documentation |
| 18/12/14 | Version 0.2 | 1312754 | First Draft : Completed Section One of the SRS. |
| 18/12/14 | Version 0.3 | 1312754 | Second Draft: Completed Section 2 of the SRS and renamed the Software Name from Farm Management System to Farming Program. |
| 18/12/14 | Version 0.4 | 1312754 | Integrated the sections provided by 1307211 |

# Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

|  |  |  |  |
| --- | --- | --- | --- |
| **Signature** | **SID** | **Title** | **Date** |
|  | 1307211 | Lead Designer | 18/12/2014 |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

Revision History ii

Document Approval ii

1. Introduction 1

1.1 Purpose 1

1.2 Scope 1

1.3 Definitions, Acronyms, and Abbreviations 1

1.4 References 1

1.5 Overview 1

2. General Description 2

2.1 Product Perspective 2

2.2 Product Functions 2

2.3 User Characteristics 2

2.4 General Constraints 2

2.5 Assumptions and Dependencies 3

3. Specific Requirements 3

3.1 External Interface Requirements 4

3.1.1 User Interfaces 4

3.1.2 Hardware Interfaces 4

3.1.3 Software Interfaces 4

3.1.4 Communications Interfaces 4

3.2 Functional Requirements 4

3.2.1 <Functional Requirement or Feature #1> 4

3.2.2 <Functional Requirement or Feature #2> 4

3.3 Use Cases <1307211> 5

3.3.1 Use Cases 6

3.3.2 Use Case Extended Description 1 7

3.3.3 Use Case Extended Description 2 8

3.3.4 Use Case Extended Description 3 8

3.4 Classes 9

3.4.1 Conatiner Class 10

3.4.2 Constants Class 11

3.4.2.1 Attributes 13

3.4.2.2 Functions 13

3.4.3 Crop Class 13

3.4.4 FarmingDataSet Class 15

3.4.5 Fertilizer Class 29

3.4.6 Field Class 30

3.4.7 Harvest Class 31

3.4.8 LogIn Class 32

3.4.9 PasswordEncryption Class 33

3.4.10 Program Class 34

} 35

3.4.11 SqlConnector Class 35

} 36

3.4.12 Staff Class 36

3.4.13 Storage Class 37

3.4.14 Vehicle Class 38

3.5 Code Not Implemented 39

3.6 Non-Functional Requirements 41

3.6.1 Maintainability 41

3.6.2 Portability 41

3.7 Design Constraints 41

3.8 Logical Database Requirements 41

3.9 Other Requirements 41

4. Analysis Models 41

4.1 Sequence Diagrams <1307211> 41

5. Change Management Process 44

8. Testing 44

8.1 Testing Log <1307211> 44

7. Program Management 55

# 1. Introduction

The Farming Program will aid the day-to-day activities of a farm manager and labourer. The program provides farm information such as the crop detail, harvest times, fertilizer doses and employment patterns, vehicles required for a harvest, storage status, container descriptions, field status, staff asignments and potential buyers. The program has been written in C# with a MS SQL database (T-SQL) on the back end.

## 1.1 Purpose

This is intended to be a live document which is constantly improved to provide the best possible developmental guidelines for the Farm Program. As it stands, the document will guide development teams in developing the Farming Program. Provide an in depth software awareness in order to better support updates to existing code or the addition of new modules to support future expansions.

## 1.2 Scope

The Farming Program is a MS SQL based program which has been designed to aid the farm personel catalog different activities. These activities are all based around the Crop class and the Harvest class. Each class clearly specifies its intended purpose by their miningful names. The current software will not provide dynamic querying capabilities and will not handle well extensive amounts of information or records. However these functionalities can be achieved due to the robust database employed and its architectural modularity.

## 1.3 Definitions, Acronyms, and Abbreviations

This document has five contributors. Each contribution will end with the contributing party’s Student Identification number or SID. Where they have contributed to large portions, the SID will appear at the beginning of the section as opposed to the end. Where no SID appears it should be assumed that all it is a contribution by 1312754 who is integrating this document.

## 1.4 References

The Program Manager <1312754> used:

Dennis, A., Wixom, B., Tegarden, D. and Dennis, A. (2009). *Systems analysis design, UML*

*version 2.0*. Hoboken, NJ: J. Wiley.

## 1.5 Overview

The software was developed to satidfy some of the needs of a farm. The program thrives to provide the farm with a way to catalog their major assets. When reviewing the code and understanding the audience is very impotant since the requirements gathered from them is the engine driving current and future development.

The following sections contain the specific requirements for the Farming Program as well as relevant planning requirements, design, implementation and testing for the current implementation. The sections will detail the different challenges or limitations a development team may encounter while executing the developmental plans outlined below and how to mitigate them with testing.

# 2. General Description

This section is not intended to provide specific requrirements, but a general overview of the basic requirements needed during the requirements gathering phase and other phases leading up to development, testing and deployment.

## 2.1 Product Perspective

As started in previous sections, the product tracks day-to-day administrative tasks done by the farm manager and the labourer of a farm. The program provides a database where both the manager and labourers can update or get updates for their daily activities. Especilfically the manager can check what crops are at what field and who is the staff assigned to take care of such crop. Additionally the labourer can see their field assignment and fertilizer requirements for the crop planted in such field.

## 2.2 Product Functions

The program is designed to store the following information in its database:

* The different types of crops cultivated with its field assignment.
* Harvest times and equipment required.
* Fertilisers to be used depending on the crop's requirements and the periods when the crop has to be treated.
* Provide a note section for more information about special one off treatments.
* Storage type and min/max temperature needed.
* The labour for sawing/planting, treating the crops and harvesting
* The Vehicles necessary for harvesting
* The Containers for storing the harvest

## 2.3 User Characteristics

Taking under consideration that the product is going to be used by farmers with little to no technologic experience, the program uses miningful classes with an easily understandable and intuitive design. There are no complicated queries and all of the information is presented as soon as the application loads.

## 2.4 General Constraints

The main constraint in this application will be the dependency implementation. The application is built with an embedded database as opposed to a dedicated SQL connection to a local/external server. This is to minimize the application footprint and maintenance for the customer.

This is based on the assumption that most of the farmers using the system will not have a dedicated Information Technology (IT) team. With this deployment method, the application becomes more portable and does not require the need of additional software installations (in this case MS SQL Server). During the implementation phase the developer team is not to use the built in wizards for creating SQL connections or DataSets. This is to keep the application’s code dynamic, reusable and undertandible to other team members.

## 2.5 Assumptions and Dependencies

There are to main assumptions for the dependencies of this software that must be addressed in this section.

* The system which will be running the application has .Net Framework 4.0 and above and is running a windows Oprationg System compatible with Visual Studio 2012 as a minimum.
* The system used for the development of this application is running Visual Studio 2012 or above with a combination of the following especifications:
  + Operating systems:
    - Windows 7 SP1 (x86 and x64)
    - Windows 8 (x86 and x64)
    - Windows Server 2008 R2 SP1 (x64)
    - Windows Server 2012 (x64)
  + Depending on the version Visual Studio, the version will install .Net 4.0 or 4.5 on the developer’s machine.
  + Architectures
    - 32-bit (x86)
    - 64-bit (x64)
  + Hardware Requirements
    - 1.6 GHz or faster processor.
    - 1 GB of RAM (1.5 GB if running on a virtual machine)
    - 10 GB (NTFS) of available hard disk space.
    - 5400 RPM hard drive.
    - DirectX 9-capable video card running at 1024 x 768 or higher display resolution.

# 3. Specific Requirements

This will be the largest and most important section of the SRS. The customer requirements will be embodied within Section 2, but this section will give the D-requirements that are used to guide the project’s software design, implementation, and testing.

Each requirement in this section should be:

* Correct
* Traceable (both forward and backward to prior/future artifacts)
* Unambiguous
* Verifiable (i.e., testable)
* Prioritized (with respect to importance and/or stability)
* Complete
* Consistent
* Uniquely identifiable (usually via numbering like 3.4.5.6)

Attention should be paid to the carefuly organize the requirements presented in this section so that they may easily accessed and understood. Furthermore, this SRS is not the software design document, therefore one should avoid the tendency to over-constrain (and therefore design) the software project within this SRS.

## 3.1 External Interface Requirements

### 3.1.1 User Interfaces

### 3.1.2 Hardware Interfaces

### 3.1.3 Software Interfaces

### 3.1.4 Communications Interfaces

## 3.2 Functional Requirements

This section describes specific features of the software project. If desired, some requirements may be specified in the use-case format and listed in the Use Cases Section.

### 3.2.1 <Functional Requirement or Feature #1>

3.2.1.1 Introduction

3.2.1.2 Inputs

3.2.1.3 Processing

3.2.1.4 Outputs

3.2.1.5 Error Handling

### 3.2.2 <Functional Requirement or Feature #2>

## 3.3 Use Cases <1307211>

The user case diagram has the manager and labourer as the actors because they are the ones who will be using the software. The manager finds the crop he looks for. He then checks the crop levels. He then has the options to check the harvest timetable, check the vehicles and check the storage level. By checking the timetable the manager could then decide to schedule a harvest.

The labour checks the harvest timetable to see if he/she has been assigned to a harvest. When doing that they check the storage space to see if there is enough room for the harvested crops to go on. The fertilizer schedule is also checked by the labourer to see if any fertilizer is assigned.

There are more than one extended descriptions as there is more than one goal by the manager. The labourer has a goal too but the descriptions only allow one actor per extended description so, extra ones have been created to accommodate it.

### 3.3.1 Use Cases



## 3.3.2 Use Case Extended Description 1

Goal: Add a new harvest

Actors: Manager

Overview: The manager of FindAName farm logs into the database to assign a harvest date. The manager sets the date of the harvest, picks which crop to be harvested. Fertilizer with the dosage is assigned as well as a vehicle, storage, container and a labourer.

|  |  |
| --- | --- |
| Actor Action | System Responce |
| 1. Manager adds a new harvest. |  |
|  | 1. System adds new harvest. |
| 1. Manager adds the crop to be harvested. |  |
|  | 1. System adds the crop. |
| 1. Manager assigns the fertilizer to be used. |  |
|  | 1. System adds the fertilizer. |
| 1. Manager assigns dosage to used. |  |
|  | 1. Systems adds dosage |
| 1. Manager assigns a vehicle. |  |
|  | 1. System adds the vehicle. |
| 1. Manager assigns the labourer. |  |
|  | 1. System adds the labourer. |
| 1. Manager assigns storage. |  |
|  | 1. Adds the storage. |
| 1. Manager assigns container |  |
|  | 1. Adds the container |

Alternative course

|  |  |
| --- | --- |
| Actor Action | System Responce |
| 1. Manager checks the harvests. |  |
|  | 1. Displays harvests. |
| 1. Manager removes a harvest. |  |
|  | 1. Removes the harvest from the database. |

## 3.3.3 Use Case Extended Description 2

Goal: Search for crops

Actors: Manager

Overview: The manager of FindAName farm checks for crops that are currently in cultivation. This will then allow the manager to see what field they are in, the stock levels and allow him to assign the harvest date.

|  |  |
| --- | --- |
| Actor Action | System Responce |
| 1. Manager looks for crops. |  |
|  | 1. System displays the details of the crops. |
| 1. Manager checks the stock |  |
|  | 1. System displays the amount of crop in stock |

## 3.3.4 Use Case Extended Description 3

Goal: Check harvest timetable

Actors: Labourer

Overview: The labourer checks the timetable to see what crops have been assigned to him, he also checks the fertilizer that has been assigned along with dosage. The storage available and container is also checked so the labourer is aware of it.

|  |  |
| --- | --- |
| Actor Action | System Responce |
| 1. Labour checks harvest date |  |
|  | 1. System displays harvest date. |
| 1. Checks crop to be harvested. |  |
|  | 1. System displays crop. |
| 1. Labourer checks fertilizer assigned. |  |
|  | 1. System displays the fertilizer. |
| 1. Labour checks storage assigned. |  |
|  | 1. Displays assigned storage. |
| 1. Labour checks assigned container |  |
|  | 1. Displays assigned container |

## 

## 3.4 Classes

The class diagram has all the classes on with the relationships on. The staff class has a 1 to many relationships with the harvest because when a harvest is assigned, a member of staff is assigned to the harvest. The labour and manager class inherits the attributes from the staff class due to them being staff.

The crop has a very strong 1 to many relationships with the harvest due to the fact that without a crop there wouldn’t be a harvest. The crop has a relationship with the field because without a field assigned, the crop won’t grow. It has a relationship with the fertilizer class because of the fertilizer that is assigned to the crop. However some crops don’t need a fertilizer to grow. The crop has a 1 to many relationship with the container because 1 crop can be stored in many containers, even though the container can store many crops, each crop needs a different type of container to be stored in.

The harvest has a relationship with the vehicle class because when a harvest is assigned, a vehicle is assigned to the harvest like all the other classes.



### 3.4.1 Conatiner Class

//MADE BY 1312754

//Integrated by 1312754

class Container

{

//Attributes

private int containerId;

private string containerName, containerType;

private int containerSize;

//Cosntructor

public Container(int containerId, string containerName, string containerType, int containerSize)

{

this.containerId = containerId;

this.containerName = containerName;

this.containerType = containerType;

this.containerSize = containerSize;

}

//properties

public string contName

{

get { return containerName; }

set { containerName = value; }

}

public string contType

{

get { return containerType; }

set { containerType = value; }

}

public int contSize

{

get { return containerSize; }

set { containerSize = value; }

}

public int ContainerID

{

get { return containerId; }

set { containerId = value; }

}

}

3.4.1.1 Attributes

3.4.1.2 Functions

<Reference to functional requirements and/or use cases>

### 

### 3.4.2 Constants Class

//CREATED BY: 1312754

//UPDATED BY:

//Integrated by: 1333187

class Constants

{

public static Boolean TESTING\_MODE = false;

//SQL Statements for the Tables

//Crops

public static String CROP\_DELETE\_QUERY = "DELETE FROM Crop WHERE CropID = @CropId";

public static String CROP\_INSERT\_QUERY = "INSERT INTO Crop (CropName, DatePanted, EstimatedHarvestDate, CropNotes, FertilizerID, CropStatus, LastDose, NextDose, DosedByStaff, CropStorageType, CropMinMax, FieldID) VALUES (@CropName, @DatePanted, @EstimatedHarvestDate, @CropNotes, @FertilizerID, @CropStatus, @LastDose, @NextDose, @DosedByStaff, @CropStorageType, @CropMinMax, @FieldID)";

public static String CROP\_SELECTALL\_QUERY = "SELECT \* FROM Crop";

public static String CROP\_UPDATE\_QUERY = @"UPDATE [dbo].[Crop] SET [CropName] = @CropName, [DatePlanted] = @DatePlanted, [EstimatedHarvestDate] = @EstimatedHarvestDate, [CropNotes] = @CropNotes, [FertilizerID] = @FertilizerID, [CropStatus] = @CropStatus, [LastDose] = @LastDose, [NextDose] = @NextDose, [DosedByStaff] = @DosedByStaff, [CropStorageType] = @CropStorageType, [CropMinMax] = @CropMinMax, [FieldID] = @FieldID WHERE (([CropID] = @Original\_CropID) AND ([CropName] = @Original\_CropName) AND ([DatePlanted] = @Original\_DatePlanted) AND ([EstimatedHarvestDate] = @Original\_EstimatedHarvestDate) AND ((@IsNull\_CropNotes = 1 AND [CropNotes] IS NULL) OR ([CropNotes] = @Original\_CropNotes)) AND ((@IsNull\_FertilizerID = 1 AND [FertilizerID] IS NULL) OR ([FertilizerID] = @Original\_FertilizerID)) AND ([CropStatus] = @Original\_CropStatus) AND ((@IsNull\_LastDose = 1 AND [LastDose] IS NULL) OR ([LastDose] = @Original\_LastDose)) AND ((@IsNull\_NextDose = 1 AND [NextDose] IS NULL) OR ([NextDose] = @Original\_NextDose)) AND ((@IsNull\_DosedByStaff = 1 AND [DosedByStaff] IS NULL) OR ([DosedByStaff] = @Original\_DosedByStaff)) AND ([CropStorageType] = @Original\_CropStorageType) AND ([CropMinMax] = @Original\_CropMinMax) AND ([FieldID] = @Original\_FieldID));

SELECT CropID, CropName, DatePlanted, EstimatedHarvestDate, CropNotes, FertilizerID, CropStatus, LastDose, NextDose, DosedByStaff, CropStorageType, CropMinMax, FieldID FROM Crop WHERE (CropID = @CropID)";

//Storage (Don't need same as container)

public static String STORAGE\_DELETE\_QUERY = "DELETE FROM Storage WHERE StorageID = @StorageID";

public static String STORAGE\_INSERT\_QUERY = "INSERT INTO Storage (StorageName, StorageCapacity, StorageAvailable, StorageNote) VALUES (@StorageName, @StorageCapacity, @StorageAvailable, @StorageNote)";

public static String STORAGE\_SELECTALL\_QUERY = "SELECT \* FROM Storage";

//Vehicles

public static String VEHICLE\_DELETE\_QUERY = "DELETE FROM Vehicle WHERE VehicleID = @VehicleID";

public static String VEHICLE\_INSERT\_QUERY = "INSERT INTO Vehicle (Type, Make, Model, Description, Capacity) VALUES (@Type, @Make, @Model, @Description, @Capacity)";

public static String VEHICLE\_SELECTALL\_QUERY = "SELECT \* FROM Vehicle";

public static String VEHICLE\_UPDATE\_QUERY = @"UPDATE [dbo].[Vehicle] SET [Type] = @Type, [Make] = @Make, [Model] = @Model, [Description] = @Description, [Capacity] = @Capacity WHERE (([VehicleID] = @Original\_VehicleID) AND ([Type] = @Original\_Type) AND ([Make] = @Original\_Make) AND ([Model] = @Original\_Model) AND ((@IsNull\_Description = 1 AND [Description] IS NULL) OR ([Description] = @Original\_Description)) AND ([Capacity] = @Original\_Capacity));

SELECT VehicleID, Type, Make, Model, Description, Capacity FROM Vehicle WHERE (VehicleID = @VehicleID)";

//Staff

public static String STAFF\_DELETE\_QUERY = "DELETE FROM Staff WHERE StaffID = @StaffID";

public static String STAFF\_INSERT\_QUERY = "INSERT INTO Staff (FirstName, LastName, Title, Rights) VALUES (@FirstName, @LastName, @Title, @Rights)";

public static String STAFF\_SELECTALL\_QUERY = "SELECT \* FROM Staff";

//Container

public static String CONTAINER\_DELETE\_QUERY = "DELETE FROM Container WHERE ContainerID = @ContainerID";

public static String CONTAINER\_INSERT\_QUERY = "INSERT INTO Container (contName, contType, contSize) VALUES (@contName, @contType, @contSize)";

public static String CONTAINER\_SELECTALL\_QUERY = "SELECT \* FROM Container";

public static String CONTAINER\_UPDATE\_QUERY = @"UPDATE [dbo].[Container] SET [contName] = @contName, [contType] = @contType, [contSize] = @contSize WHERE (([ContainerID] = @Original\_ContainerID) AND ([contName] = @Original\_contName) AND ([contType] = @Original\_contType) AND ([contSize] = @Original\_contSize));

SELECT ContainerID, contName, contType, contSize FROM Container WHERE (ContainerID = @ContainerID)";

//Harvest

public static String HARVEST\_DELETE\_QUERY = "DELETE FROM Harvest WHERE HarvestID = @HarvestID)";

public static String HARVEST\_INSERT\_QUERY = "INSERT INTO Harvest (HarvestStartDate, HarvestEndDate, StaffRequired, ContainerID, CropID, FieldID, VehicleID) VALUES (@HarvestStartDate, @HarvestEndDate, @StaffRequired, @ContainerID, @CropID, @FieldID, @VehicleID)";

public static String HARVEST\_SELECTALL\_QUERY = "SELECT \* FROM Harvest";

public static String HARVEST\_UPDATE\_QUERY = @"UPDATE [dbo].[Harvest] SET [HarvestStartDate] = @HarvestStartDate, [HarvestEndDate] = @HarvestEndDate, [StaffRequired] = @StaffRequired, [ContainerID] = @ContainerID, [CropID] = @CropID, [FieldID] = @FieldID, [VehicleID] = @VehicleID WHERE (([HarvestID] = @Original\_HarvestID) AND ([HarvestStartDate] = @Original\_HarvestStartDate) AND ([HarvestEndDate] = @Original\_HarvestEndDate) AND ([StaffRequired] = @Original\_StaffRequired) AND ([ContainerID] = @Original\_ContainerID) AND ([CropID] = @Original\_CropID) AND ([FieldID] = @Original\_FieldID) AND ([VehicleID] = @Original\_VehicleID));

SELECT HarvestID, HarvestStartDate, HarvestEndDate, StaffRequired, ContainerID, CropID, FieldID, VehicleID FROM Harvest WHERE (HarvestID = @HarvestID)";

//Fertilizer

public static String FERTILIZER\_DELETE\_QUERY = "DELETE FROM Fertilizer WHERE FertilizerID = @FertilizerID";

public static String FERTILIZER\_INSERT\_QUERY = "INSERT INTO Fertilizer (fertName, fertDose, fertNote) VALUES (@fertName, @fertDose, @fertNote)";

public static String FERTILIZER\_SELECTALL\_QUERY = "SELECT \* FROM Fertilizer";

public static String FERTILIZER\_UPDATE\_QUERY = @"UPDATE [dbo].[Fertilizer] SET [fertName] = @fertName, [fertDose] = @fertDose, [fertNote] = @fertNote WHERE (([FertilizerID] = @Original\_FertilizerID) AND ([fertName] = @Original\_fertName) AND ([fertDose] = @Original\_fertDose) AND ([fertNote] = @Original\_fertNote));

SELECT FertilizerID, fertName, fertDose, fertNote FROM Fertilizer WHERE (FertilizerID = @FertilizerID)";

//Field

public static String FIELD\_DELETE\_QUERY = "DELETE FROM Field WHERE FieldID = @FieldID";

public static String FIELD\_INSERT\_QUERY = "INSERT INTO Field (FieldName, FieldStatus, FieldNotes) VALUES (@FieldName, @FieldStatus, @FieldNotes)";

public static String FIELD\_SELECTALL\_QUERY = "SELECT \* FROM Field";

public static String FIELD\_UPDATE\_QUERY = @"UPDATE [dbo].[Harvest] SET [HarvestStartDate] = @HarvestStartDate, [HarvestEndDate] = @HarvestEndDate, [StaffRequired] = @StaffRequired, [ContainerID] = @ContainerID, [CropID] = @CropID, [FieldID] = @FieldID, [VehicleID] = @VehicleID WHERE (([HarvestID] = @Original\_HarvestID) AND ([HarvestStartDate] = @Original\_HarvestStartDate) AND ([HarvestEndDate] = @Original\_HarvestEndDate) AND ([StaffRequired] = @Original\_StaffRequired) AND ([ContainerID] = @Original\_ContainerID) AND ([CropID] = @Original\_CropID) AND ([FieldID] = @Original\_FieldID) AND ([StaffID] = @Original\_StaffID) AND ([VehicleID] = @Original\_VehicleID) AND ([StorageID] = @Original\_StorageID));

SELECT HarvestID, HarvestStartDate, HarvestEndDate, StaffRequired, ContainerID, CropID, FieldID, StaffID, VehicleID, StorageID FROM Harvest WHERE (HarvestID = @HarvestID)";

}

### 3.4.2.1 Attributes

### 3.4.2.2 Functions

<Reference to functional requirements and/or use cases>

### 3.4.3 Crop Class

//made by student 1311808

//integrated by 1312754 & 1333187

class Crop

{

//attributes

private string CropName, CropNotes, CropStatus, CropMinMax;

private DateTime DatePlanted, EstimatedHarvestDate, LastDose, NextDose;

private int FertilizerID, FieldID, CropID, DosedByStaffID, CropStorageType;

//constructors

public Crop (int cropID, string cropName, DateTime datePlanted, DateTime estimatedHarvestDate, string cropNotes, int fertilizerID, string cropStatus, DateTime lastDose, DateTime nextDose, int dosedByStaff, int cropStorageType, string cropMinMax, int fieldID)

{

this.CropName = cropName;

this.DatePlanted = datePlanted;

this.EstimatedHarvestDate = estimatedHarvestDate;

this.CropNotes = cropNotes;

this.FertilizerID = fertilizerID;

this.CropStatus = cropStatus;

this.LastDose = lastDose;

this.NextDose = nextDose;

this.DosedByStaffID = dosedByStaff;

this.CropStorageType = cropStorageType;

this.CropMinMax = cropMinMax;

this.FieldID = fieldID;

this.CropID = cropID;

}

//properties

public string cropName

{

get { return CropName;}

set { CropName = value; }

}

public int cropId

{

get { return CropID; }

set { CropID = value; }

}

public DateTime datePlanted

{

get { return DatePlanted; }

set { DatePlanted = value; }

}

public DateTime estimatedHarvestDate

{

get { return EstimatedHarvestDate; }

set { EstimatedHarvestDate = value; }

}

public string cropNotes

{

get { return CropNotes; }

set { CropNotes = value; }

}

public int fertilizerID

{

get { return FertilizerID; }

set { FertilizerID = value; }

}

public string cropStatus

{

get { return CropStatus; }

set { CropStatus = value; }

}

public DateTime lastDose

{

get { return LastDose; }

set { LastDose = value; }

}

public DateTime nextDose

{

get { return NextDose; }

set { NextDose = value; }

}

public int dosedByStaff

{

get { return DosedByStaffID; }

set { DosedByStaffID = value; }

}

public int cropStorageType

{

get { return CropStorageType; }

set { CropStorageType = value; }

}

public string cropMinMax

{

get { return CropMinMax; }

set { CropMinMax = value; }

}

public int fieldID

{

get { return FieldID; }

set { FieldID = value; }

}

}

### 3.4.4 FarmingDataSet Class

//Created by and modified by 1333187

namespace farmingprogram

{

class FarmingDataSet

{

//Sql adapters which hold different update,insert,delete and select queries

public static SqlDataAdapter cropDataAdapter;

public static SqlDataAdapter fertilizerAdapter;

public static SqlDataAdapter fieldAdapter;

public static SqlDataAdapter containerAdapter;

public static SqlDataAdapter staffAdapter;

public static SqlDataAdapter harvestAdapter;

public static SqlDataAdapter vehicleAdapter;

//Data tables when the select query is read they are saved on these tables

public static DataTable cropDataTable;

public static DataTable fertilizerDataTable;

public static DataTable fieldDataTable;

public static DataTable containerDataTable;

public static DataTable staffDataTable;

public static DataTable harvestDataTable;

public static DataTable vehicleDataTable;

#region Crop Set

public static void initializeCropSet() //Initializes the crop data set

{

SqlConnector.getConnection().Open(); //Open connection

cropDataAdapter = new SqlDataAdapter(); //Attach adapter to cropDataAdapter

//Declare Adapter queries

cropDataAdapter.SelectCommand = new SqlCommand(Constants.CROP\_SELECTALL\_QUERY, SqlConnector.getConnection());

cropDataAdapter.DeleteCommand = new SqlCommand(Constants.CROP\_DELETE\_QUERY, SqlConnector.getConnection());

cropDataAdapter.InsertCommand = new SqlCommand(Constants.CROP\_INSERT\_QUERY, SqlConnector.getConnection());

//If the update command is just being added

if (cropDataAdapter.UpdateCommand == null)

{

cropDataAdapter.UpdateCommand = new SqlCommand(Constants.CROP\_UPDATE\_QUERY, SqlConnector.getConnection()); //Add the update query

setCropUpdateParams(); //Add the parameters for the update query

}

cropDataTable = new DataTable(); //Create a new crop data table

cropDataAdapter.Fill(cropDataTable); //Populate the data table

MainProgram.getSingleton().cropBindingSource.DataSource = cropDataTable; //Send the cropGridView to view the crop data table

SqlConnector.getConnection().Close(); //Close connection

}

public static void addCrop(Crop crop) //Add a crop

{

//Parameters which are set when query is execute "@varName", value

cropDataAdapter.InsertCommand.Parameters.Add(new SqlParameter("@CropName", crop.cropName));

cropDataAdapter.InsertCommand.Parameters.Add(new SqlParameter("@DatePlanted", crop.datePlanted));

cropDataAdapter.InsertCommand.Parameters.Add(new SqlParameter("@EstimatedHarvestDate", crop.estimatedHarvestDate));

cropDataAdapter.InsertCommand.Parameters.Add(new SqlParameter("@CropNotes", crop.cropNotes));

cropDataAdapter.InsertCommand.Parameters.Add(new SqlParameter("@FertilizerID", crop.fertilizerID));

cropDataAdapter.InsertCommand.Parameters.Add(new SqlParameter("@CropStatus", crop.cropStatus));

cropDataAdapter.InsertCommand.Parameters.Add(new SqlParameter("@LastDose", crop.lastDose));

cropDataAdapter.InsertCommand.Parameters.Add(new SqlParameter("@NextDose", crop.nextDose));

cropDataAdapter.InsertCommand.Parameters.Add(new SqlParameter("@DosedByStaff", crop.dosedByStaff));

cropDataAdapter.InsertCommand.Parameters.Add(new SqlParameter("@CropStorageType", crop.cropStorageType));

cropDataAdapter.InsertCommand.Parameters.Add(new SqlParameter("@CropMinMax", crop.cropMinMax));

cropDataAdapter.InsertCommand.Parameters.Add(new SqlParameter("@FieldID", crop.fieldID));

try

{

SqlConnector.getConnection().Open(); //Open connection

cropDataAdapter.InsertCommand.ExecuteNonQuery(); //Execute query

}

finally

{

cropDataAdapter.InsertCommand.Parameters.Clear(); //Clear params to avoid error

SqlConnector.getConnection().Close(); //Close connection

}

}

//Sets update params so when row is edited there is no need to input hardcoded values

public static void setCropUpdateParams()

{

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@CropName", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "CropName", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@DatePlanted", global::System.Data.SqlDbType.Date, 0, global::System.Data.ParameterDirection.Input, 0, 0, "DatePlanted", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@EstimatedHarvestDate", global::System.Data.SqlDbType.Date, 0, global::System.Data.ParameterDirection.Input, 0, 0, "EstimatedHarvestDate", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@CropNotes", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "CropNotes", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@FertilizerID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "FertilizerID", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@CropStatus", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "CropStatus", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@LastDose", global::System.Data.SqlDbType.Date, 0, global::System.Data.ParameterDirection.Input, 0, 0, "LastDose", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@NextDose", global::System.Data.SqlDbType.Date, 0, global::System.Data.ParameterDirection.Input, 0, 0, "NextDose", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@DosedByStaff", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "DosedByStaff", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@CropStorageType", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "CropStorageType", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@CropMinMax", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "CropMinMax", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@FieldID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "FieldID", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_CropID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "CropID", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_CropName", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "CropName", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_DatePlanted", global::System.Data.SqlDbType.Date, 0, global::System.Data.ParameterDirection.Input, 0, 0, "DatePlanted", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_EstimatedHarvestDate", global::System.Data.SqlDbType.Date, 0, global::System.Data.ParameterDirection.Input, 0, 0, "EstimatedHarvestDate", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@IsNull\_CropNotes", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "CropNotes", global::System.Data.DataRowVersion.Original, true, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_CropNotes", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "CropNotes", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@IsNull\_FertilizerID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "FertilizerID", global::System.Data.DataRowVersion.Original, true, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_FertilizerID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "FertilizerID", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_CropStatus", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "CropStatus", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@IsNull\_LastDose", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "LastDose", global::System.Data.DataRowVersion.Original, true, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_LastDose", global::System.Data.SqlDbType.Date, 0, global::System.Data.ParameterDirection.Input, 0, 0, "LastDose", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@IsNull\_NextDose", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "NextDose", global::System.Data.DataRowVersion.Original, true, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_NextDose", global::System.Data.SqlDbType.Date, 0, global::System.Data.ParameterDirection.Input, 0, 0, "NextDose", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@IsNull\_DosedByStaff", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "DosedByStaff", global::System.Data.DataRowVersion.Original, true, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_DosedByStaff", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "DosedByStaff", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_CropStorageType", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "CropStorageType", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_CropMinMax", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "CropMinMax", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_FieldID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "FieldID", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

cropDataAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@CropID", global::System.Data.SqlDbType.Int, 4, global::System.Data.ParameterDirection.Input, 0, 0, "CropID", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

}

#endregion

#region Harvest set

public static void initializeHarvestSet() //Initializes the harvest data set

{

SqlConnector.getConnection().Open(); //Open connection

harvestAdapter = new SqlDataAdapter(); //Attach adapter to harvestAdapter

//Declare Adapter queries

harvestAdapter.SelectCommand = new SqlCommand(Constants.HARVEST\_SELECTALL\_QUERY, SqlConnector.getConnection());

harvestAdapter.DeleteCommand = new SqlCommand(Constants.HARVEST\_DELETE\_QUERY, SqlConnector.getConnection());

harvestAdapter.InsertCommand = new SqlCommand(Constants.HARVEST\_INSERT\_QUERY, SqlConnector.getConnection());

//If the update command is just being added

if (harvestAdapter.UpdateCommand == null)

{

harvestAdapter.UpdateCommand = new SqlCommand(Constants.HARVEST\_UPDATE\_QUERY, SqlConnector.getConnection()); //Add the update query

setHarvestUpdateParams(); //Add the parameters for the update query

}

harvestDataTable = new DataTable(); //Create a new datatable

harvestAdapter.Fill(harvestDataTable); //File harvest data table

MainProgram.getSingleton().harvestBindingSource.DataSource = harvestDataTable; //The harvestGridView will now display the datatable

SqlConnector.getConnection().Close(); //Close connection

}

public static void addHarvest(Harvest harvest) //Adding a harvest

{

//@HarvestStartDate, @HarvestEndDate, @StaffRequired, @ContainerID, @CropID, @FieldID, @StaffID, @VehicleID, @StorageID

harvestAdapter.InsertCommand.Parameters.Add(new SqlParameter("@HarvestStartDate", harvest.harvestStartDate));

harvestAdapter.InsertCommand.Parameters.Add(new SqlParameter("@HarvestEndDate", harvest.harvestEndDate));

harvestAdapter.InsertCommand.Parameters.Add(new SqlParameter("@StaffRequired", harvest.staffRequired));

harvestAdapter.InsertCommand.Parameters.Add(new SqlParameter("@ContainerID", harvest.containerID));

harvestAdapter.InsertCommand.Parameters.Add(new SqlParameter("@CropID", harvest.cropID));

harvestAdapter.InsertCommand.Parameters.Add(new SqlParameter("@FieldID", harvest.fieldID));

harvestAdapter.InsertCommand.Parameters.Add(new SqlParameter("@VehicleID", harvest.vehicle));//This should be the vehicle id of the vehicle assigned to the harvest.

try

{

SqlConnector.getConnection().Open(); //Open sql connection

harvestAdapter.InsertCommand.ExecuteNonQuery(); //Execute query

}

finally

{

harvestAdapter.InsertCommand.Parameters.Clear(); //Clear params to avoid error

SqlConnector.getConnection().Close(); //Close connection

}

}

//Sets update params so when row is edited there is no need to input hardcoded values

public static void setHarvestUpdateParams()

{

harvestAdapter.UpdateCommand.CommandType = global::System.Data.CommandType.Text;

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@HarvestStartDate", global::System.Data.SqlDbType.Date, 0, global::System.Data.ParameterDirection.Input, 0, 0, "HarvestStartDate", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@HarvestEndDate", global::System.Data.SqlDbType.Date, 0, global::System.Data.ParameterDirection.Input, 0, 0, "HarvestEndDate", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@StaffRequired", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "StaffRequired", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@ContainerID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "ContainerID", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@CropID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "CropID", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@FieldID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "FieldID", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@StaffID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "StaffID", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@VehicleID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "VehicleID", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@StorageID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "StorageID", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_HarvestID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "HarvestID", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_HarvestStartDate", global::System.Data.SqlDbType.Date, 0, global::System.Data.ParameterDirection.Input, 0, 0, "HarvestStartDate", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_HarvestEndDate", global::System.Data.SqlDbType.Date, 0, global::System.Data.ParameterDirection.Input, 0, 0, "HarvestEndDate", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_StaffRequired", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "StaffRequired", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_ContainerID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "ContainerID", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_CropID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "CropID", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_FieldID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "FieldID", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_StaffID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "StaffID", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_VehicleID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "VehicleID", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_StorageID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "StorageID", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

harvestAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@HarvestID", global::System.Data.SqlDbType.Int, 4, global::System.Data.ParameterDirection.Input, 0, 0, "HarvestID", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

}

#endregion

#region Fertilizer set

public static void initializeFertilizerSet() //Initializes the fertilizer dataset

{

SqlConnector.getConnection().Open(); //Open connection

fertilizerAdapter = new SqlDataAdapter(); //Create a new adapter for fertilizer

//Insert queries into adapter

fertilizerAdapter.SelectCommand = new SqlCommand(Constants.FERTILIZER\_SELECTALL\_QUERY, SqlConnector.getConnection());

fertilizerAdapter.DeleteCommand = new SqlCommand(Constants.FERTILIZER\_DELETE\_QUERY, SqlConnector.getConnection());

fertilizerAdapter.InsertCommand = new SqlCommand(Constants.FERTILIZER\_INSERT\_QUERY, SqlConnector.getConnection());

//Update command

if (fertilizerAdapter.UpdateCommand == null)

{

fertilizerAdapter.UpdateCommand = new SqlCommand(Constants.FERTILIZER\_UPDATE\_QUERY, SqlConnector.getConnection()); //Add the update command

setFertilizerUpdateParams(); //Set the update command params

}

fertilizerDataTable = new DataTable(); //New data table for fertilizer

fertilizerAdapter.Fill(fertilizerDataTable); //populate data table

MainProgram.getSingleton().fertilizerBindingSource.DataSource = fertilizerDataTable; //Display data table on the grid

SqlConnector.getConnection().Close();

}

public static void addFertilizer(Fertilizer fertilizer)

{

fertilizerAdapter.InsertCommand.Parameters.Add(new SqlParameter("@fertName", fertilizer.fertName));

fertilizerAdapter.InsertCommand.Parameters.Add(new SqlParameter("@fertDose", fertilizer.fertDose));

fertilizerAdapter.InsertCommand.Parameters.Add(new SqlParameter("@fertNote", fertilizer.fertNote));

try

{

SqlConnector.getConnection().Open();

fertilizerAdapter.InsertCommand.ExecuteNonQuery();

}

finally

{

fertilizerAdapter.InsertCommand.Parameters.Clear();

SqlConnector.getConnection().Close();

}

}

//Sets update params so when row is edited there is no need to input hardcoded values

private static void setFertilizerUpdateParams()

{

fertilizerAdapter.UpdateCommand.CommandType = global::System.Data.CommandType.Text;

fertilizerAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@fertName", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "fertName", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

fertilizerAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@fertDose", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "fertDose", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

fertilizerAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@fertNote", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "fertNote", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

fertilizerAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_FertilizerID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "FertilizerID", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

fertilizerAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_fertName", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "fertName", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

fertilizerAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_fertDose", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "fertDose", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

fertilizerAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_fertNote", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "fertNote", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

fertilizerAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@FertilizerID", global::System.Data.SqlDbType.Int, 4, global::System.Data.ParameterDirection.Input, 0, 0, "FertilizerID", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

}

#endregion

#region Vehicle set

//Initializes vehicle set

public static void initializeVehicleSet()

{

SqlConnector.getConnection().Open(); //Opens connection

vehicleAdapter = new SqlDataAdapter(); //Binds vehicleAdapter to a new data adapter

//Sets adapter commands

vehicleAdapter.SelectCommand = new SqlCommand(Constants.VEHICLE\_SELECTALL\_QUERY, SqlConnector.getConnection());

vehicleAdapter.DeleteCommand = new SqlCommand(Constants.VEHICLE\_DELETE\_QUERY, SqlConnector.getConnection());

vehicleAdapter.InsertCommand = new SqlCommand(Constants.VEHICLE\_INSERT\_QUERY, SqlConnector.getConnection());

//Sets update command and parameters

if (vehicleAdapter.UpdateCommand != null)

{

vehicleAdapter.UpdateCommand = new SqlCommand(Constants.VEHICLE\_UPDATE\_QUERY, SqlConnector.getConnection());

setVehicleUpdateParams();

}

//populates table with database data

vehicleDataTable = new DataTable();

vehicleAdapter.Fill(vehicleDataTable);

//View the data on a grid

MainProgram.getSingleton().vehicleBindingSource.DataSource = vehicleDataTable;

SqlConnector.getConnection().Close(); //Close connection

}

//Add a new vehicle

public static void addVehicle(Vehicle vehicle)

{

//Parameters which are set when query is execute

vehicleAdapter.InsertCommand.Parameters.Add(new SqlParameter("@Type", vehicle.type));

vehicleAdapter.InsertCommand.Parameters.Add(new SqlParameter("@Make", vehicle.make));

vehicleAdapter.InsertCommand.Parameters.Add(new SqlParameter("@Model", vehicle.model));

vehicleAdapter.InsertCommand.Parameters.Add(new SqlParameter("@Description", vehicle.description));

vehicleAdapter.InsertCommand.Parameters.Add(new SqlParameter("@Capacity", vehicle.capacity));

try

{

SqlConnector.getConnection().Open(); //Open connection

vehicleAdapter.InsertCommand.ExecuteNonQuery(); //Execute query

}

finally

{

vehicleAdapter.InsertCommand.Parameters.Clear(); //Clear params to avoid error

SqlConnector.getConnection().Close(); //Close connection

}

}

//Adds variable parameters to update command

static void setVehicleUpdateParams()

{

vehicleAdapter.UpdateCommand.CommandType = global::System.Data.CommandType.Text;

vehicleAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Type", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "Type", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

vehicleAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Make", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "Make", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

vehicleAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Model", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "Model", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

vehicleAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Description", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "Description", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

vehicleAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Capacity", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "Capacity", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

vehicleAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_VehicleID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "VehicleID", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

vehicleAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_Type", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "Type", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

vehicleAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_Make", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "Make", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

vehicleAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_Model", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "Model", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

vehicleAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@IsNull\_Description", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "Description", global::System.Data.DataRowVersion.Original, true, null, "", "", ""));

vehicleAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_Description", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "Description", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

vehicleAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_Capacity", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "Capacity", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

vehicleAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@VehicleID", global::System.Data.SqlDbType.Int, 4, global::System.Data.ParameterDirection.Input, 0, 0, "VehicleID", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

}

#endregion

//Deletes from the actual database where ID is the primary key and id param is the param string

public static int deleteFromTable(SqlDataAdapter adapter, String idParam, int id)

{

int returnCode = 0; //Return code for how many rows deleted

try

{

SqlConnector.getConnection().Open(); //Open connection

adapter.DeleteCommand.Parameters.Add(new SqlParameter(idParam, SqlDbType.Int)).Value = id; //Add the Param Value

returnCode = adapter.DeleteCommand.ExecuteNonQuery(); //execute the query and set return code

}

catch (Exception exception) //exceptions in foreign keys hence we handle then.

{

System.Windows.Forms.MessageBox.Show("Cannot delete the data, it is used on another table. Please delete this data first.");

}

finally //Clear params and close connection

{

adapter.DeleteCommand.Parameters.Clear();

SqlConnector.getConnection().Close();

}

return returnCode;

}

#region Field set

//Sets update params so when row is edited there is no need to input hardcoded values

private static void setFieldUpdateParams()

{

fieldAdapter.UpdateCommand.CommandType = global::System.Data.CommandType.Text;

fieldAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@FieldName", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "FieldName", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

fieldAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@FieldStatus", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "FieldStatus", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

fieldAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@FieldNotes", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "FieldNotes", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

fieldAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_FieldID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "FieldID", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

fieldAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_FieldName", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "FieldName", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

fieldAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_FieldStatus", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "FieldStatus", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

fieldAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@IsNull\_FieldNotes", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "FieldNotes", global::System.Data.DataRowVersion.Original, true, null, "", "", ""));

fieldAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_FieldNotes", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "FieldNotes", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

fieldAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@FieldID", global::System.Data.SqlDbType.Int, 4, global::System.Data.ParameterDirection.Input, 0, 0, "FieldID", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

}

public static void initializeFieldSet()

{

SqlConnector.getConnection().Open();

fieldAdapter = new SqlDataAdapter();

fieldAdapter.SelectCommand = new SqlCommand(Constants.FIELD\_SELECTALL\_QUERY, SqlConnector.getConnection());

fieldAdapter.DeleteCommand = new SqlCommand(Constants.FIELD\_DELETE\_QUERY, SqlConnector.getConnection());

fieldAdapter.InsertCommand = new SqlCommand(Constants.FIELD\_INSERT\_QUERY, SqlConnector.getConnection());

if (fieldAdapter.UpdateCommand == null)

{

fieldAdapter.UpdateCommand = new SqlCommand(Constants.FIELD\_UPDATE\_QUERY, SqlConnector.getConnection());

setFieldUpdateParams();

}

fieldDataTable = new DataTable();

fieldAdapter.Fill(fieldDataTable);

MainProgram.getSingleton().fieldBindingSource.DataSource = fieldDataTable;

SqlConnector.getConnection().Close();

}

public static void addField(Field field) //Add a field

{

//Set the param values for the query

fieldAdapter.InsertCommand.Parameters.Add(new SqlParameter("@FieldName", field.fieldName));

fieldAdapter.InsertCommand.Parameters.Add(new SqlParameter("@FieldStatus", field.fieldStatus));

fieldAdapter.InsertCommand.Parameters.Add(new SqlParameter("@FieldNotes", field.fieldNotes));

try

{

//Open a connection then execute the query

SqlConnector.getConnection().Open();

fieldAdapter.InsertCommand.ExecuteNonQuery();

}

finally

{

//Clear params and close connection

fieldAdapter.InsertCommand.Parameters.Clear();

SqlConnector.getConnection().Close();

}

}

#endregion

#region Container set

public static void initializeContainerSet()

{

SqlConnector.getConnection().Open();

containerAdapter = new SqlDataAdapter();

containerAdapter.SelectCommand = new SqlCommand(Constants.CONTAINER\_SELECTALL\_QUERY, SqlConnector.getConnection());

containerAdapter.DeleteCommand = new SqlCommand(Constants.CONTAINER\_DELETE\_QUERY, SqlConnector.getConnection());

containerAdapter.InsertCommand = new SqlCommand(Constants.CONTAINER\_INSERT\_QUERY, SqlConnector.getConnection());

if (containerAdapter.UpdateCommand == null)

{

containerAdapter.UpdateCommand = new SqlCommand(Constants.CONTAINER\_UPDATE\_QUERY);

setContainerUpdateParams();

}

containerDataTable = new DataTable();

containerAdapter.Fill(containerDataTable);

MainProgram.getSingleton().containerBindingSource.DataSource = containerDataTable;

SqlConnector.getConnection().Close();

}

public static void addContainer(Container container)

{

containerAdapter.InsertCommand.Parameters.Add(new SqlParameter("@contName", container.contName));

containerAdapter.InsertCommand.Parameters.Add(new SqlParameter("@contSize", container.contSize));

containerAdapter.InsertCommand.Parameters.Add(new SqlParameter("@contType", container.contType));

try

{

SqlConnector.getConnection().Open();

containerAdapter.InsertCommand.ExecuteNonQuery();

}

finally

{

containerAdapter.InsertCommand.Parameters.Clear();

SqlConnector.getConnection().Close();

}

}

//Sets update params so when row is edited there is no need to input hardcoded values

private static void setContainerUpdateParams()

{

containerAdapter.UpdateCommand.CommandType = global::System.Data.CommandType.Text;

containerAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@contName", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "contName", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

containerAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@contType", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "contType", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

containerAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@contSize", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "contSize", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

containerAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_ContainerID", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "ContainerID", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

containerAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_contName", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "contName", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

containerAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_contType", global::System.Data.SqlDbType.NVarChar, 0, global::System.Data.ParameterDirection.Input, 0, 0, "contType", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

containerAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@Original\_contSize", global::System.Data.SqlDbType.Int, 0, global::System.Data.ParameterDirection.Input, 0, 0, "contSize", global::System.Data.DataRowVersion.Original, false, null, "", "", ""));

containerAdapter.UpdateCommand.Parameters.Add(new global::System.Data.SqlClient.SqlParameter("@ContainerID", global::System.Data.SqlDbType.Int, 4, global::System.Data.ParameterDirection.Input, 0, 0, "ContainerID", global::System.Data.DataRowVersion.Current, false, null, "", "", ""));

}

#endregion

public static void initializeStaffSet()

{

SqlConnector.getConnection().Open();

staffAdapter = new SqlDataAdapter();

staffAdapter.SelectCommand = new SqlCommand(Constants.STAFF\_SELECTALL\_QUERY, SqlConnector.getConnection());

staffAdapter.DeleteCommand = new SqlCommand(Constants.STAFF\_DELETE\_QUERY, SqlConnector.getConnection());

staffAdapter.InsertCommand = new SqlCommand(Constants.STAFF\_INSERT\_QUERY, SqlConnector.getConnection());

staffDataTable = new DataTable();

staffAdapter.Fill(staffDataTable);

MainProgram.getSingleton().staffBindingSource.DataSource = staffDataTable;

SqlConnector.getConnection().Close();

}

}

### 3.4.5 Fertilizer Class

//MADE BY: pls add

//Integrated by 1312754

class Fertilizer

{

//Attributes

private int FertilizerID;

private string FertName;

private string FertDose;

private string FertNote;

//constructors

public Fertilizer(int fertilizerId, string fertname, string fertdose, string fertnote)

{

this.FertName = fertname;

this.FertDose = fertdose;

this.FertNote = fertnote;

this.FertilizerID = fertilizerId;

}

//properties

public string fertName

{

get { return FertName; }

set { FertName = value; }

}

public int fertilizerId

{

get { return fertilizerId; }

set { fertilizerId = value; }

}

public string fertDose

{

get { return FertDose; }

set { FertDose = value; }

}

public string fertNote

{

get { return FertNote; }

set { FertNote = value; }

}

}

### 3.4.6 Field Class

//CREATED BY pls add

//integrated by 1312754

class Field

{

//Attributes

private int FieldId;

private string FieldName, FieldStatus, FieldNotes;

//constructors

public Field(int fieldId, string fieldname, string fieldstatus, string fieldnotes)

{

this.FieldName = fieldname;

this.FieldNotes = fieldnotes;

this.FieldStatus = fieldstatus;

this.FieldId = fieldId;

}

//Properties

public string fieldName

{

get { return FieldName; }

set { FieldName = value; }

}

public string fieldNotes

{

get { return FieldNotes; }

set { FieldNotes = value; }

}

public string fieldStatus

{

get { return FieldStatus; }

set { FieldStatus = value; }

}

public int fieldID

{

get { return FieldId; }

set { FieldId = value; }

}

}

### 3.4.7 Harvest Class

//CREATED BY 1312754

class Harvest

{

//Attributes

private DateTime HarvestStartDate, HarvestEndDate;

private int harvestID, StaffRequired, ContainerID, CropID, FieldID, VehicleID;

//Constructors

public Harvest(int HarvestID, DateTime HarvestStartDate, DateTime HarvestEndDate, int StaffRequired, int ContainerID, int CropID, int FieldID, int VehicleID)

{

this.harvestID = HarvestID;

this.HarvestStartDate = HarvestStartDate;

this.HarvestEndDate = HarvestEndDate;

this.StaffRequired = StaffRequired;

this.ContainerID = ContainerID;

this.CropID = CropID;

this.FieldID = FieldID;

this.VehicleID = VehicleID;

}

//Properties

public DateTime harvestStartDate

{

get { return HarvestStartDate; }

set { HarvestStartDate = value; }

}

public DateTime harvestEndDate

{

get { return HarvestEndDate; }

set { HarvestEndDate = value; }

}

public int staffRequired//How many of the Staff will be required for the harvest.

{

get { return StaffRequired; }

set { StaffRequired = value; }

}

public int containerID

{

get { return ContainerID; }

set { ContainerID = value; }

}

public int cropID

{

get { return CropID; }

set { CropID = value; }

}

public int HarvestID

{

get { return harvestID; }

set { harvestID = value; }

}

public int fieldID

{

get { return FieldID; }

set { FieldID = value; }

}

public int vehicle

{

get { return VehicleID; }

set { VehicleID = value; }

}

### 3.4.8 LogIn Class

//CREATED BY: SP1062

//LoginDetail represents a class with the username and password and rights.

class LoginDetail

{

//Login username and password

private string username, password;

//Rights 0=Normal 1=Manager

private int rights;

//Constructor

public LoginDetail(string username, string password, int rights)

{

this.username = username;

this.password = password;

this.rights = rights;

}

//Properties

public int getRights()

{

return rights;

}

public string getUsername()

{

return username;

}

public string getPassword()

{

return password;

}

}

### 3.4.9 PasswordEncryption Class

//Made by 1333187

class PasswordEncryption

{

//Properties of password encryption

const string HASH = "SECORQWP$(OS";

const string SALT = "R£$%F%£E";

const string VI = "@1AFLVJEIQERJFCOO";

//Encryption method used 'http://www.goprogrammers.com/2013/10/understanding-encryption-and-decryption.html'

public static string Encrypt(string plainText)

{

byte[] plainTextBytes = Encoding.UTF8.GetBytes(plainText);

byte[] keyBytes = new Rfc2898DeriveBytes(HASH, Encoding.ASCII.GetBytes(SALT)).GetBytes(256 / 8);

var symmetricKey = new RijndaelManaged() { Mode = CipherMode.CBC, Padding = PaddingMode.Zeros };

var encryptor = symmetricKey.CreateEncryptor(keyBytes, Encoding.ASCII.GetBytes(VI));

byte[] cipherTextBytes;

using (var memoryStream = new MemoryStream())

{

using (var cryptoStream = new CryptoStream(memoryStream, encryptor, CryptoStreamMode.Write))

{

cryptoStream.Write(plainTextBytes, 0, plainTextBytes.Length);

cryptoStream.FlushFinalBlock();

cipherTextBytes = memoryStream.ToArray();

cryptoStream.Close();

}

memoryStream.Close();

}

return Convert.ToBase64String(cipherTextBytes);

}

//Decryption method used 'http://www.goprogrammers.com/2013/10/understanding-encryption-and-decryption.html'

public static string Decrypt(string encryptedText)

{

byte[] cipherTextBytes = Convert.FromBase64String(encryptedText);

byte[] keyBytes = new Rfc2898DeriveBytes(HASH, Encoding.ASCII.GetBytes(SALT)).GetBytes(256 / 8);

var symmetricKey = new RijndaelManaged() { Mode = CipherMode.CBC, Padding = PaddingMode.None };

var decryptor = symmetricKey.CreateDecryptor(keyBytes, Encoding.ASCII.GetBytes(VI));

var memoryStream = new MemoryStream(cipherTextBytes);

var cryptoStream = new CryptoStream(memoryStream, decryptor, CryptoStreamMode.Read);

byte[] plainTextBytes = new byte[cipherTextBytes.Length];

int decryptedByteCount = cryptoStream.Read(plainTextBytes, 0, plainTextBytes.Length);

memoryStream.Close();

cryptoStream.Close();

return Encoding.UTF8.GetString(plainTextBytes, 0, decryptedByteCount).TrimEnd("\0".ToCharArray());

}

}

### 3.4.10 Program Class

//Made by 1333187

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

SqlConnector.startConnection();

if (!Constants.TESTING\_MODE) //If not testing then prompt login

{

Application.Run(new LoginInterface()); //Prompts login

}

else //If testing

{

loginDetail = new LoginDetail("admin", "admin", 1); //Default user and password = admin, admin

SqlConnector.getConnection().Close(); //Close connection left open for login

}

if (loginDetail != null) //If login detail is not null then run the program

{

Application.Run(new MainProgram()); //Run main program

}

}

private static LoginDetail loginDetail; //Login detail field

public static void setLogin(LoginDetail ld) //Login detail setter

{

loginDetail = ld;

}

public static LoginDetail getLoginDetail() //Login detail getter

{

return loginDetail;

}

### }

### 3.4.11 SqlConnector Class

//Made by 1333187

class SqlConnector

{

private static SqlConnection connection = null; //SqlConnection which is used throughout program

//Start the sql connnection

public static void startConnection()

{

string connectionString = ConfigurationManager.ConnectionStrings["farmingprogram.Properties.Settings.DbFarmProgramConnectionString"].ConnectionString; //Gets the Connection String from resources ('App.Config')

connection = new SqlConnection(connectionString); //Creates a new connection

try

{

connection.Open(); //Opens connection

}

catch (Exception ex)

{

//Any errors will leave program useless

//So it shuts down

MessageBox.Show("Error opening connection. Closing program.");

Environment.Exit(0);

}

}

//Getter for the sql connection

public static SqlConnection getConnection()

{

return connection;

}

//Checks if password is valid with the arguments username, password they are both inputs from the user

public static Boolean validPassword(String username, String password)

{

if (String.IsNullOrEmpty(username) || String.IsNullOrEmpty(password)) //If the username and password is empty then Invalid credentials would be displayed

{

MessageBox.Show("Invalid credentials.");

return false;

}

string queryString = "SELECT \* FROM Login"; //Selects all data from login

SqlDataReader command = new SqlCommand(queryString, getConnection()).ExecuteReader(); //Executes command

try

{

while (command.Read()) //While the command still has a output

{

String name = command.GetString(0).ToLower(); //Get username from database

String pass = command.GetString(1).ToLower(); //Get password from database

int rights = command.GetInt32(2); //Get rights from database

if (name.Equals(username.ToLower()) && pass.Equals(PasswordEncryption.Encrypt(password).ToLower())) //See if user and password match

{

Program.setLogin(new LoginDetail(name, pass, rights)); //Set login detail

getConnection().Close();//Close connection

return true; //Return true if match

}

}

MessageBox.Show("Invalid username or password try again.");

}

finally

{

command.Close(); //Close command

}

return false; //Return false if no matches

}

### }

### 3.4.12 Staff Class

//CREATED BY 1312754

class Staff

{

//attribute

private int staffId, rights;

private string firstName, lastName, title;

//Constructor

public Staff(int staffId, string firstName, string lastName, string title, int rights)

{

this.staffId = staffId;

this.firstName = firstName;

this.lastName = lastName;

this.title = title;

this.rights = rights;

}

//Properties

public string FirstName

{

get { return firstName; }

set { firstName = value; }

}

public string LastName

{

get { return lastName; }

set { lastName = value; }

}

public string Title

{

get { return title; }

set { title = value; }

}

public int Rights

{

get { return rights; }

set { rights = value; }

}

public int StaffID

{

get { return staffId; }

set { staffId = value; }

}

}

### 3.4.13 Storage Class

//MADE BY 1307211

//Integrated by 1312754

class Storage

{

//attribute

private int StorageCapacity, StorageAvailable;

private string StorageName, StorageNote;

//Constructor

public Storage(string storageName, int storageCapacity, int storageAvailable, string storageNote)

{

this.StorageName = storageName;

this.StorageCapacity = storageCapacity;

this.StorageAvailable = storageAvailable;

this.StorageNote = storageNote;

}

//Properties

public string storageName

{

get { return StorageName; }

set { StorageName = value; }

}

public int storageCapacity

{

get { return StorageCapacity; }

set { StorageCapacity = value; }

}

public int storageAvailable

{

get { return StorageAvailable; }

set { StorageAvailable = value; }

}

public string storageNote

{

get { return StorageNote; }

set { StorageNote = value; }

}

}

### 3.4.14 Vehicle Class

//MADE BY 1307211

//Integrated by 1312754

class Vehicle

{

//Attributes

private int VehicleID, Capacity;

private string Type, Make, Model, Description;

//Constructor

public Vehicle(int vehicleID, string type, string make, string model, string description, int capacity)

{

this.Type = type; //Type constructor

this.Make = make; //Make constructor

this.Model = model; //Model constructor

this.Description = description; //Description constructor

this.Capacity = capacity; //Capacity constructor

this.VehicleID = vehicleID;

}

//Properties

public string type //Type properties

{

get { return Type; }

set { Type = value; }

}

public string make //Make properties

{

get { return Make; }

set { Make = value; }

}

public string model //Model properties

{

get { return Model;}

set { Model = value; }

}

public string description //Description properties

{

get { return Description; }

set { Description = value; }

}

public int capacity //Capacity Properties

{

get { return Capacity; }

set { Capacity = value; }

}

public int vehicleID { //Vehicle Properties

get { return vehicleID; }

set { vehicleID = value; }

}

}

## 3.5 Code Not Implemented

Due to some synchronization isues and miscommunication, there were classes implemented several times and or left out of implementation. The code below was submitted by 1307211

This code for the vehicle and storage is used to build a sql adapter for the database and to then populate the database with data. It first opens up the connection to the database when the program runs, it pulls all the data and populates the appropriate tables with it. It calls the constant class when the data is to be deleted for the command.

Note this code was used in a back-up program as the rest of the group couldn’t understand the code the programmer did due to him being absent for a couple of weeks. However this code was shown to the programmer who then corrected it to how it was meant to work meaning most of this code wasn’t integrated into the final project.

public static SqlDataAdapter vehicleDataAdapter;

public static DataSet initializeVehicleSet()

{

//start the connection

SqlConnector.startConnection();

vehicleDataAdapter = new SqlDataAdapter();

vehicleDataAdapter.SelectCommand = new SqlCommand(Costants.VEHICLE\_SELECTALL\_QUERY, SqlConnector.getConnection());

vehicleDataAdapter.DeleteCommand = new SqlCommand(Costants.VEHICLE\_DELETE\_QUERY, SqlConnector.getConnection());

vehicleDataAdapter.InsertCommand = new SqlCommand(Costants.VEHICLE\_INSERT\_QUERY, SqlConnector.getConnection());

// crate the dataset

DataSet dsVehicle = new DataSet();

//fill in the dataset with the data coming from the dataAdapter

vehicleDataAdapter.Fill(dsVehicle);

//close the connection

SqlConnector.getConnection().Close();

return dsVehicle;

}

public static int deleteVehicle(int vehicleId)

{

int returnCode = 0;

try

{

SqlConnector.getConnection().Open();

vehicleDataAdapter.DeleteCommand.Parameters.AddWithValue("@VehicleID", vehicleId);

returnCode = vehicleDataAdapter.DeleteCommand.ExecuteNonQuery();

}

finally

{

SqlConnector.getConnection().Close();

}

return returnCode;

}

public static SqlDataAdapter storageDataAdapter

public static DataSet initializeStorageSet()

{

//start the connection

SqlConnector.startConnection();

storageDataAdapter = new SqlDataAdapter();

storageDataAdapter.SelectCommand = new SqlCommand(Costants.STORAGE\_SELECTALL\_QUERY, SqlConnector.getConnection());

storageDataAdapter.DeleteCommand = new SqlCommand(Costants.STORAGE\_DELETE\_QUERY, SqlConnector.getConnection());

storageDataAdapter.InsertCommand = new SqlCommand(Costants.STORAGE\_INSERT\_QUERY, SqlConnector.getConnection());

// crate the dataset

DataSet dsStorage = new DataSet();

//fill in the dataset with the data coming from the dataAdapter

storageDataAdapter.Fill(dsStorage);

//close the connection

SqlConnector.getConnection().Close();

return dsStorage;

}

public static int deleteStorage(int storageId)

{

int returnCode = 0;

try

{

SqlConnector.getConnection().Open();

storageDataAdapter.DeleteCommand.Parameters.AddWithValue("@storageID", storageId);

returnCode = storageDataAdapter.DeleteCommand.ExecuteNonQuery();

}

finally

{

SqlConnector.getConnection().Close();

}

return returnCode;

}

## 3.6 Non-Functional Requirements

Non-functional requirements may exist for the following attributes. Often these requirements must be achieved at a system-wide level rather than at a unit level. State the requirements in the following sections in measurable terms (e.g., 95% of transaction shall be processed in less than a second, system downtime may not exceed 1 minute per day, > 30 day MTBF value, etc).

### 3.6.1 Maintainability

### 3.6.2 Portability

## 3.7 Design Constraints

Specify design constrains imposed by other standards, company policies, hardware limitation, etc. that will impact this software project.

## 3.8 Logical Database Requirements

Will a database be used? If so, what logical requirements exist for data formats, storage capabilities, data retention, data integrity, etc.

## 3.9 Other Requirements

Catchall section for any additional requirements.

# 4. Analysis Models

List all analysis models used in developing specific requirements previously given in this SRS. Each model should include an introduction and a narrative description. Furthermore, each model should be traceable the SRS’s requirements.

## 4.1 Sequence Diagrams <1307211>

The sequence diagram for the manager shows him adding a harvest. He logs into the database and assigns a new harvest. The system then adds the date the manager assigned with the cropID, name, date it was planted, the notes and status of the crop. The manager then picks what fertilizer is going to be used with the dosage which, the system then adds in the harvest. A vehicle, storage space and a labourer is then assigned and like before the system updates the harvest containing all the details.

The sequence diagram for the labour shows them checking the harvest timetable for any assigned harvests. The labourer starts off with checking the assigned date of the harvest assigned, the system then displays the date. The labourer then checks the fertilizer, fertilizer dosage, vehicle and storage space that has been assigned which again, the system will display to the labourer.



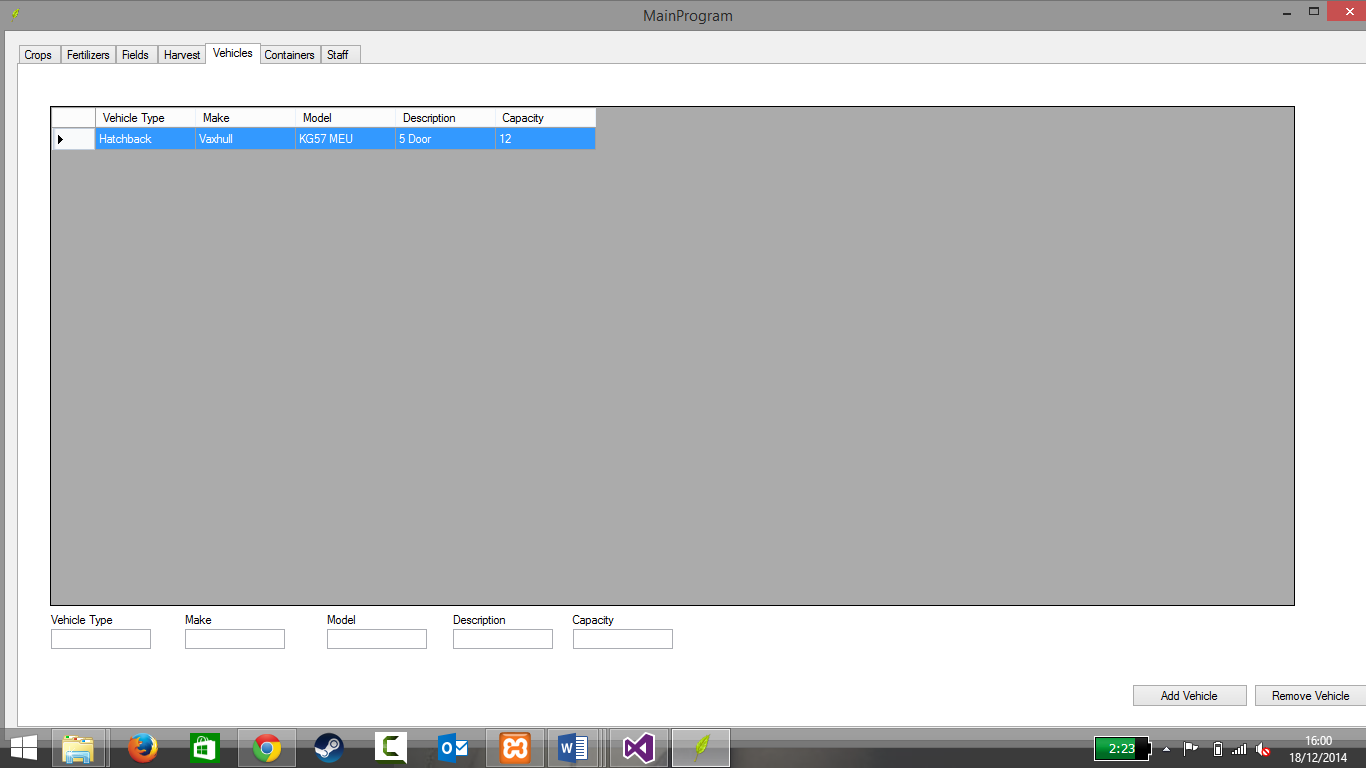
# 5. Change Management Process

Identify and describe the process that will be used to update the SRS, as needed, when project scope or requirements change. Who can submit changes and by what means, and how will these changes be approved.

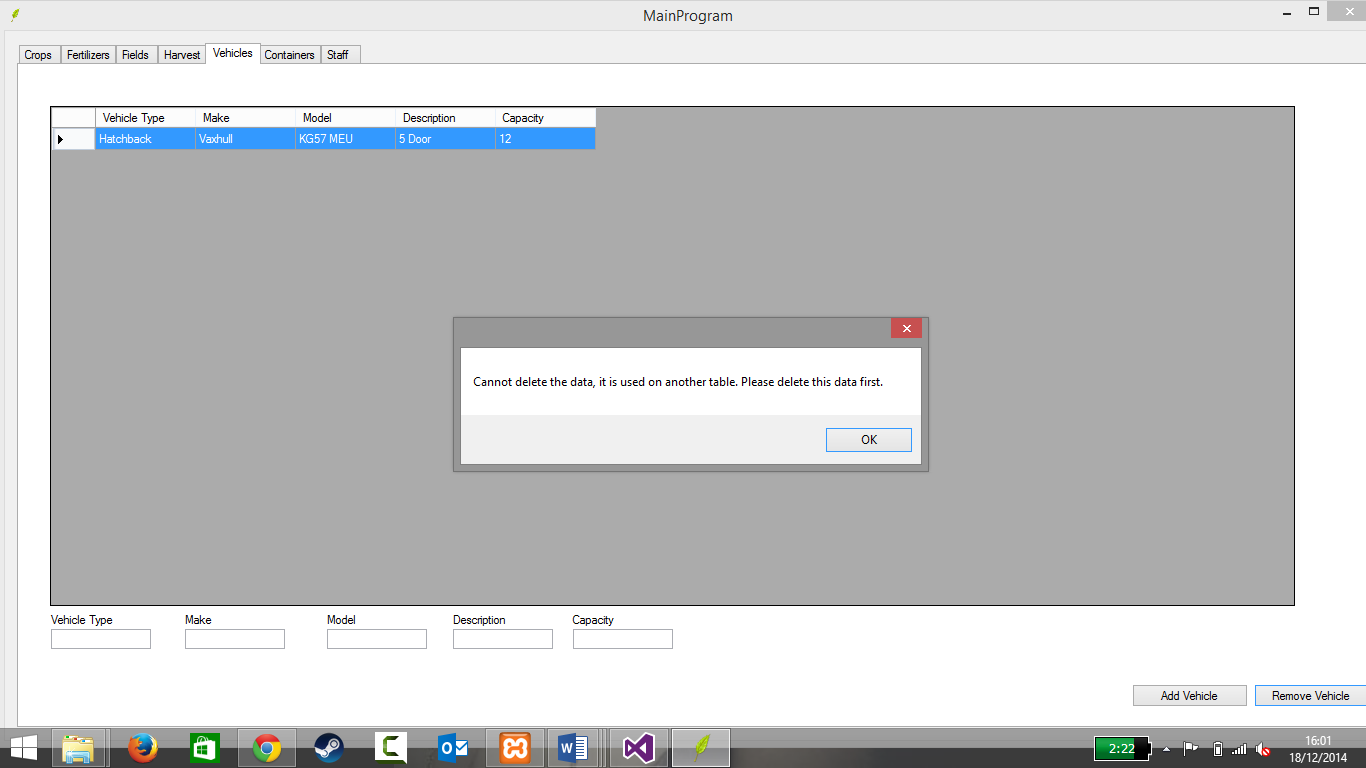
# 8. Testing

## 8.1 Testing Log <1307211>

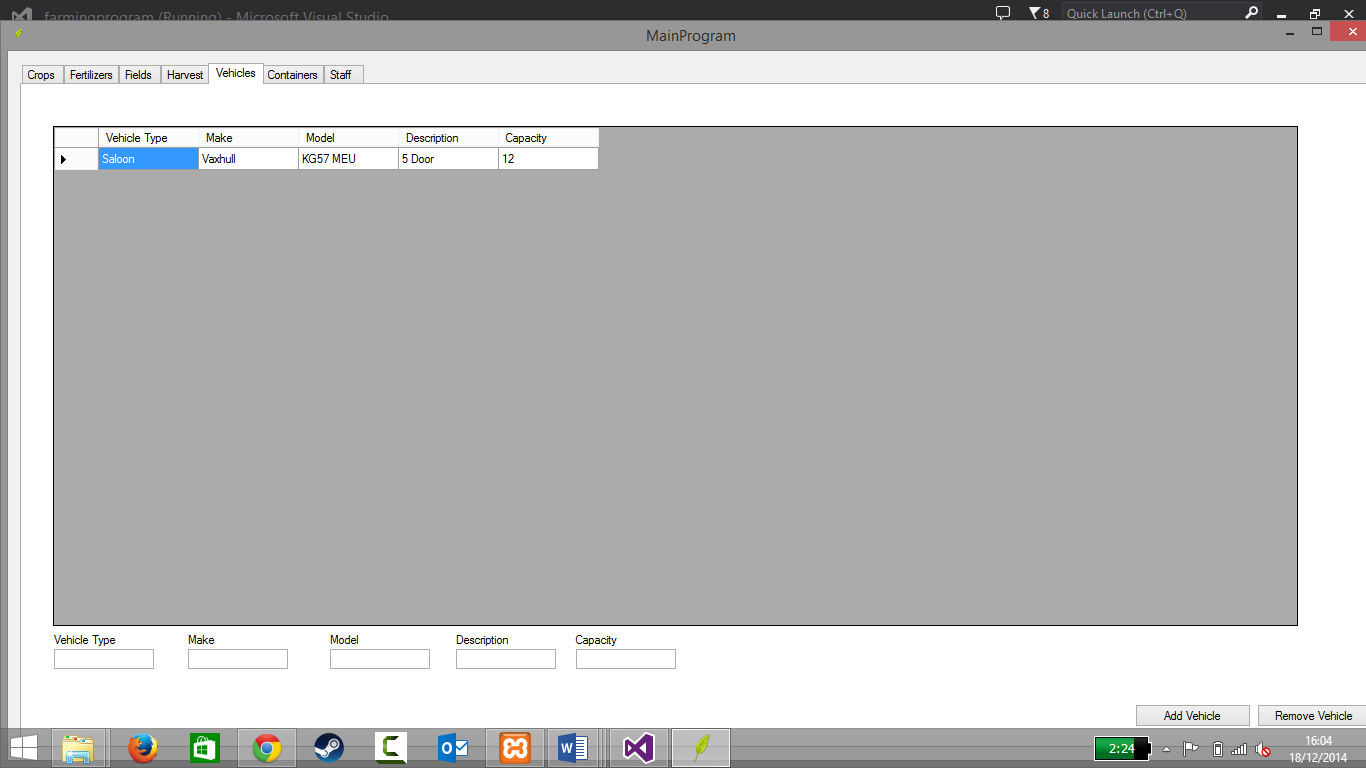
**Vehicle**



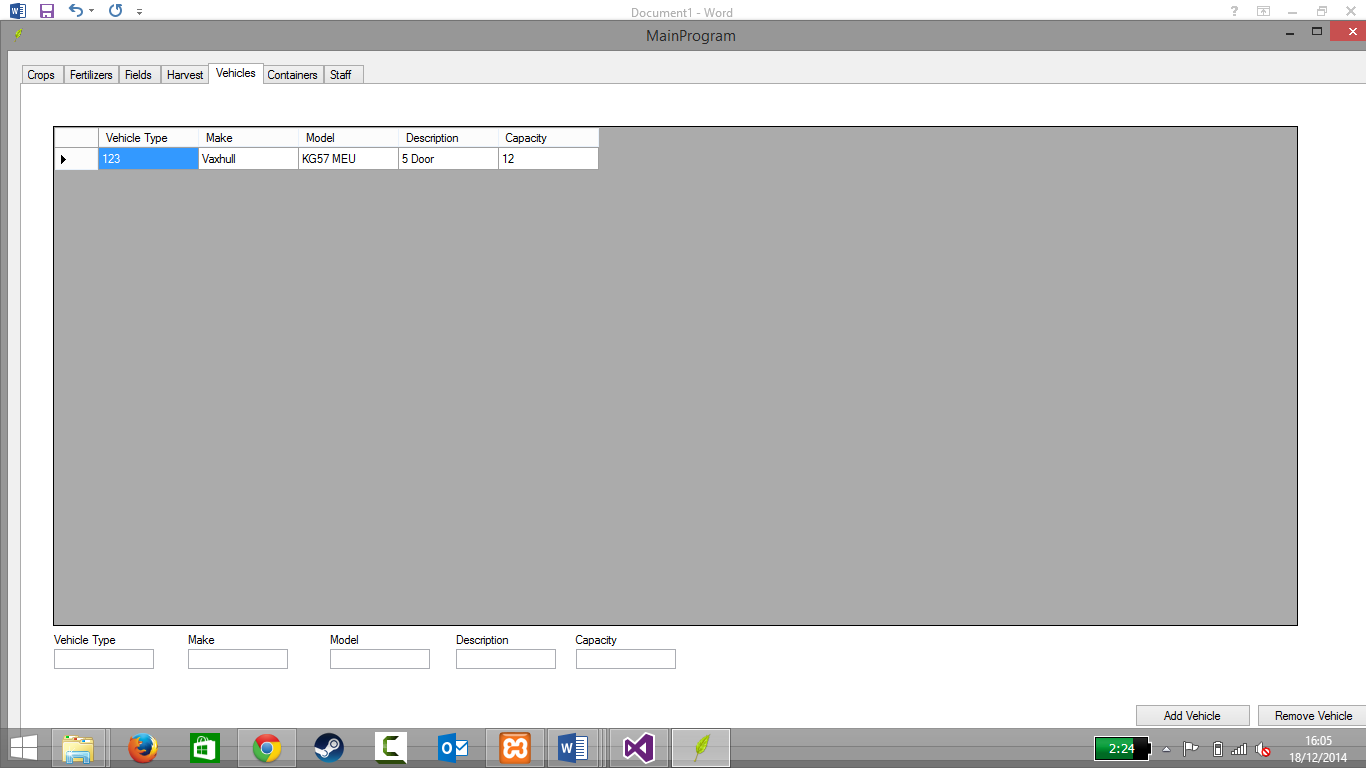
This shows adding a row, but as there isn’t the test fails.



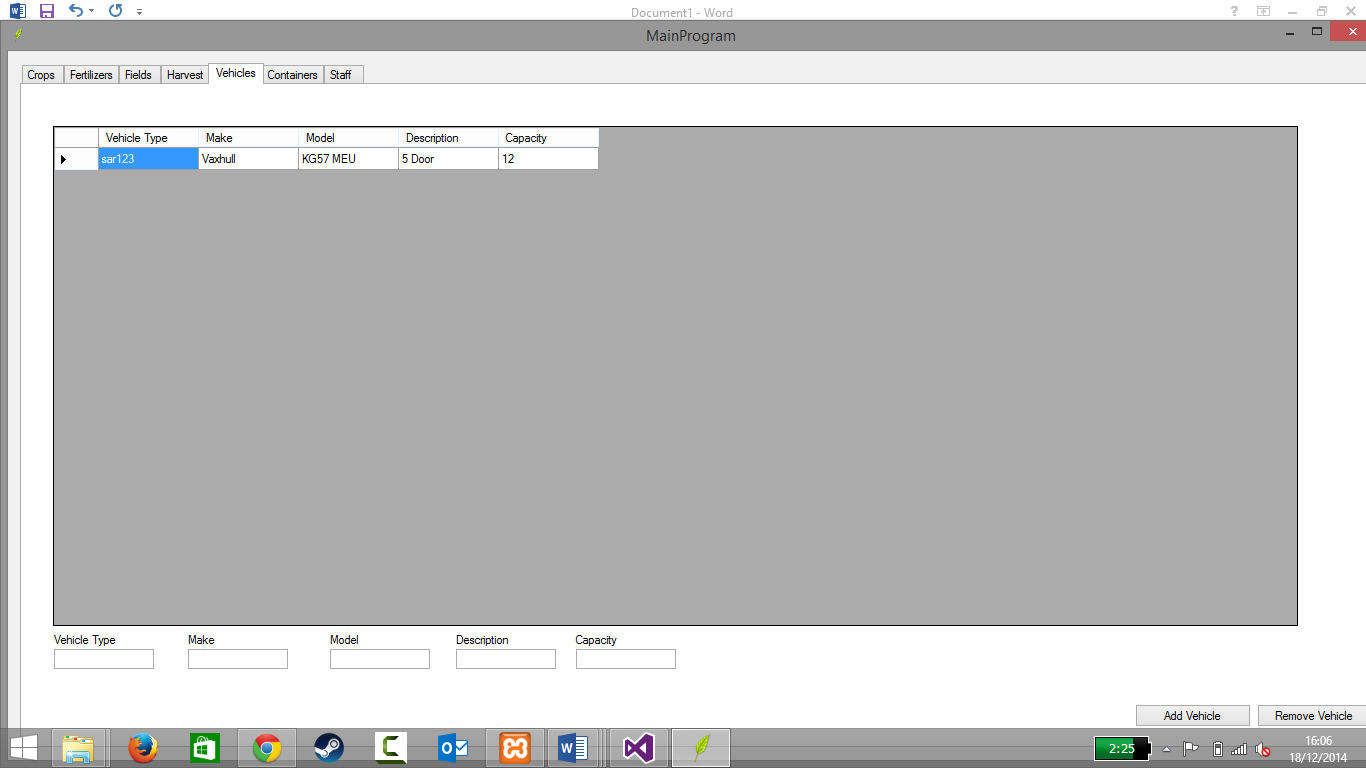
This show the data being deleted, but as no new row this means the data can’t be deleted



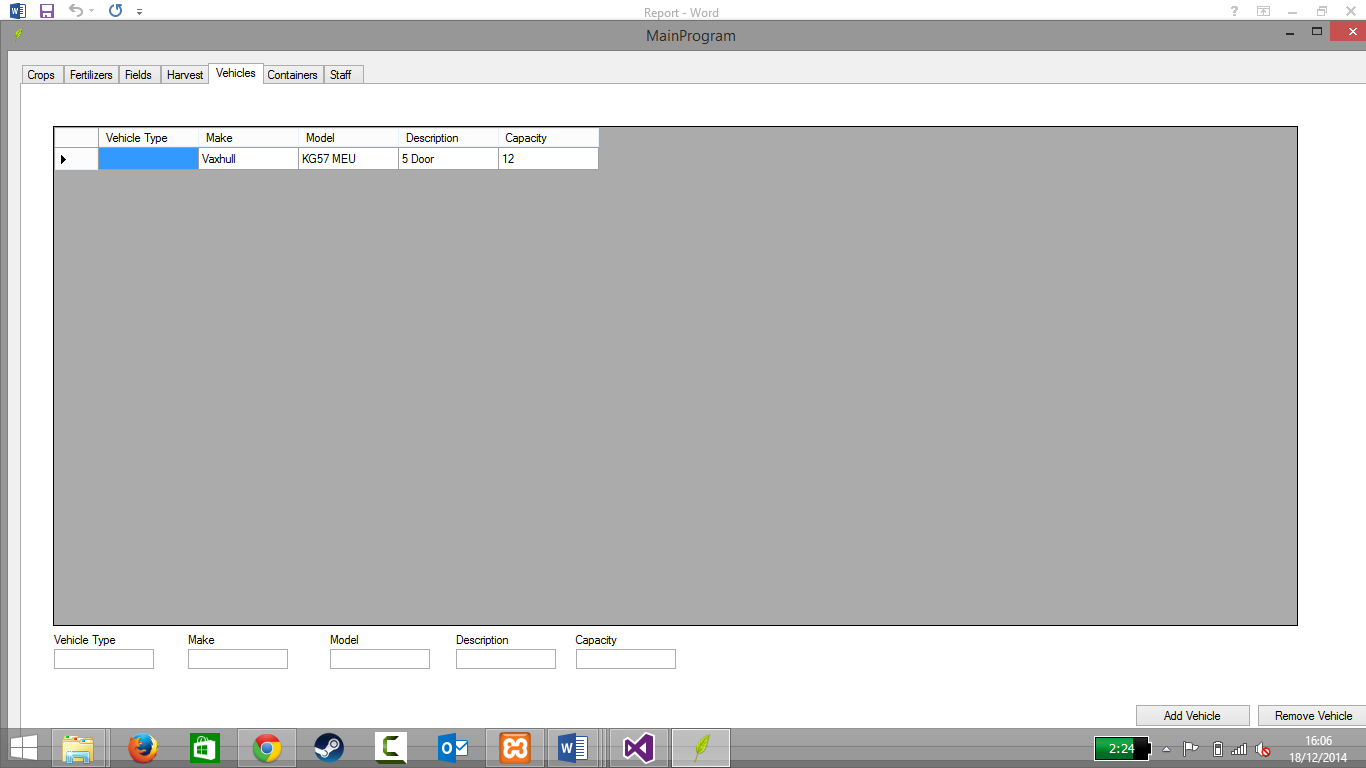
This screenshot shows the type being changed with letters



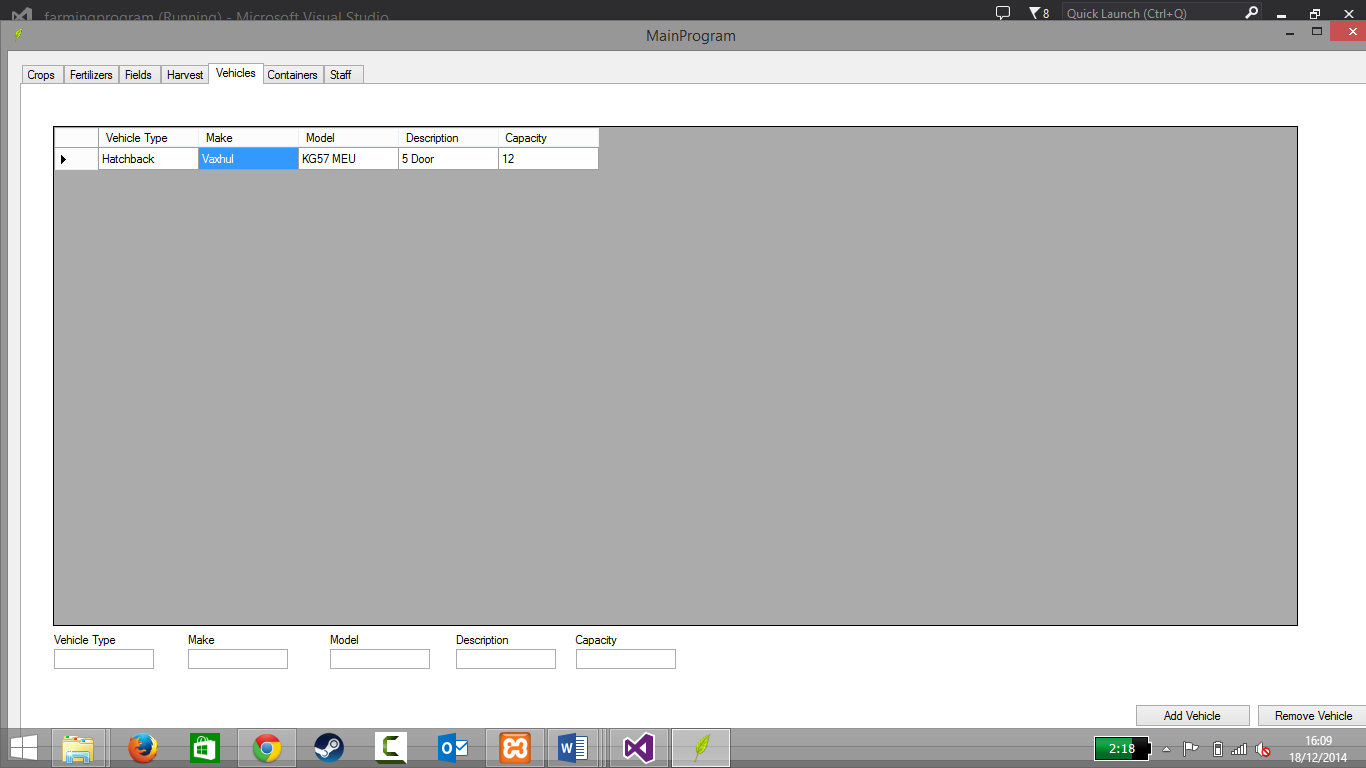
This screenshot shows the type being changed with numbers



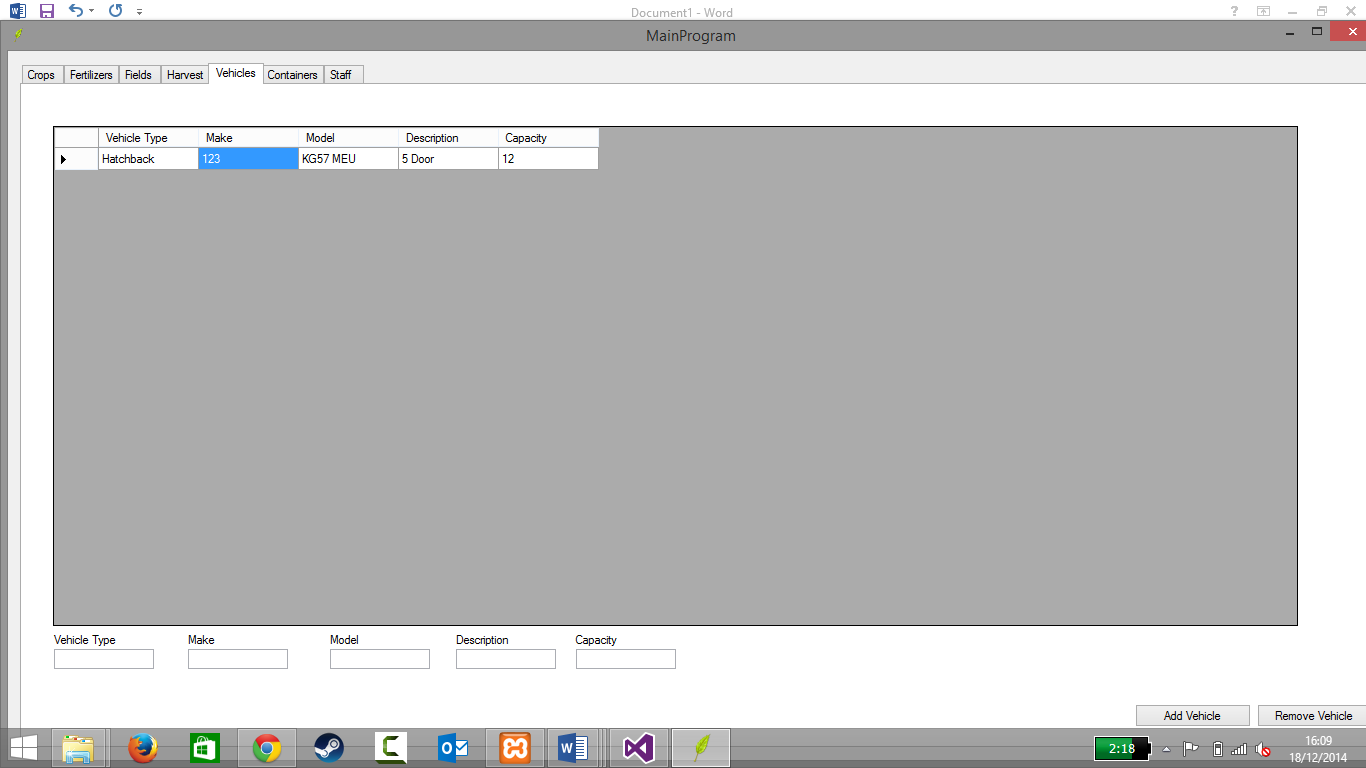
This screenshot shows the data being changed with letters and numbers



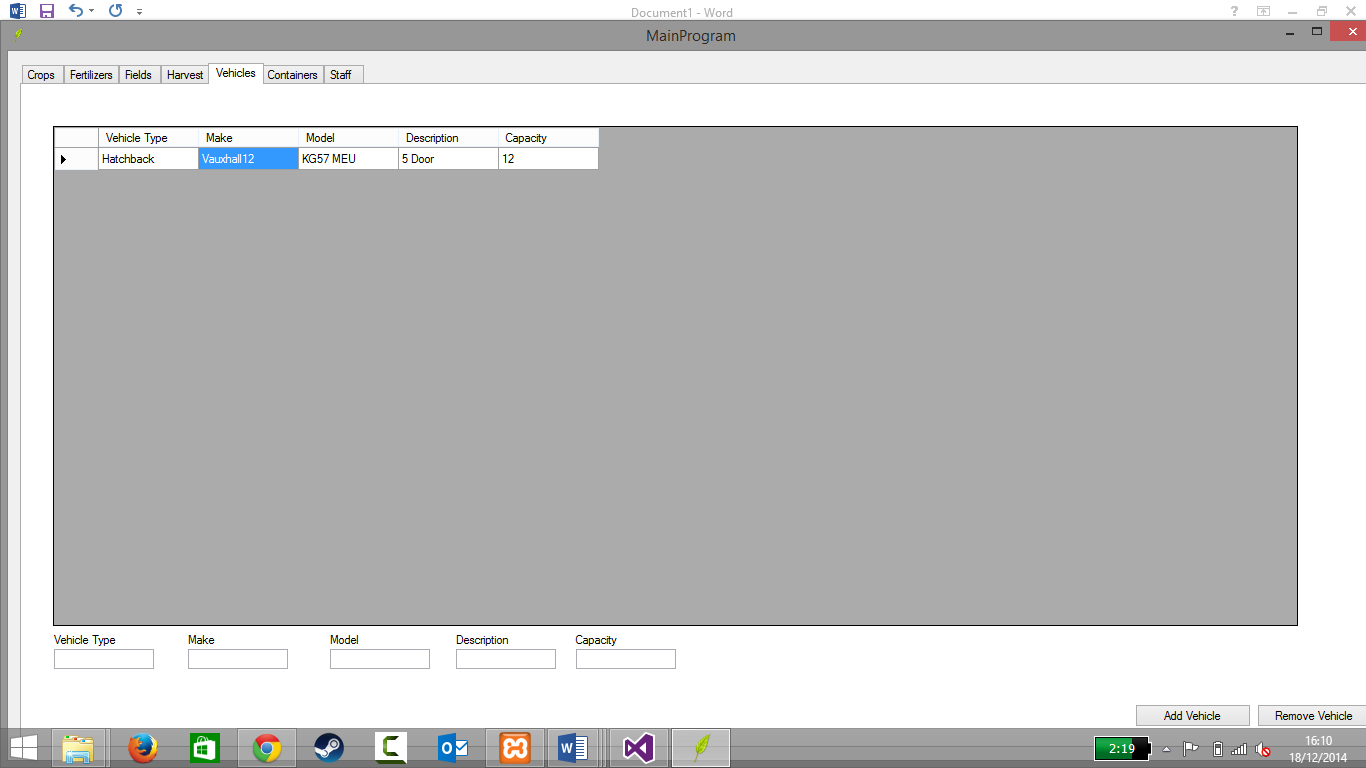
This screenshot shows the data being left blank



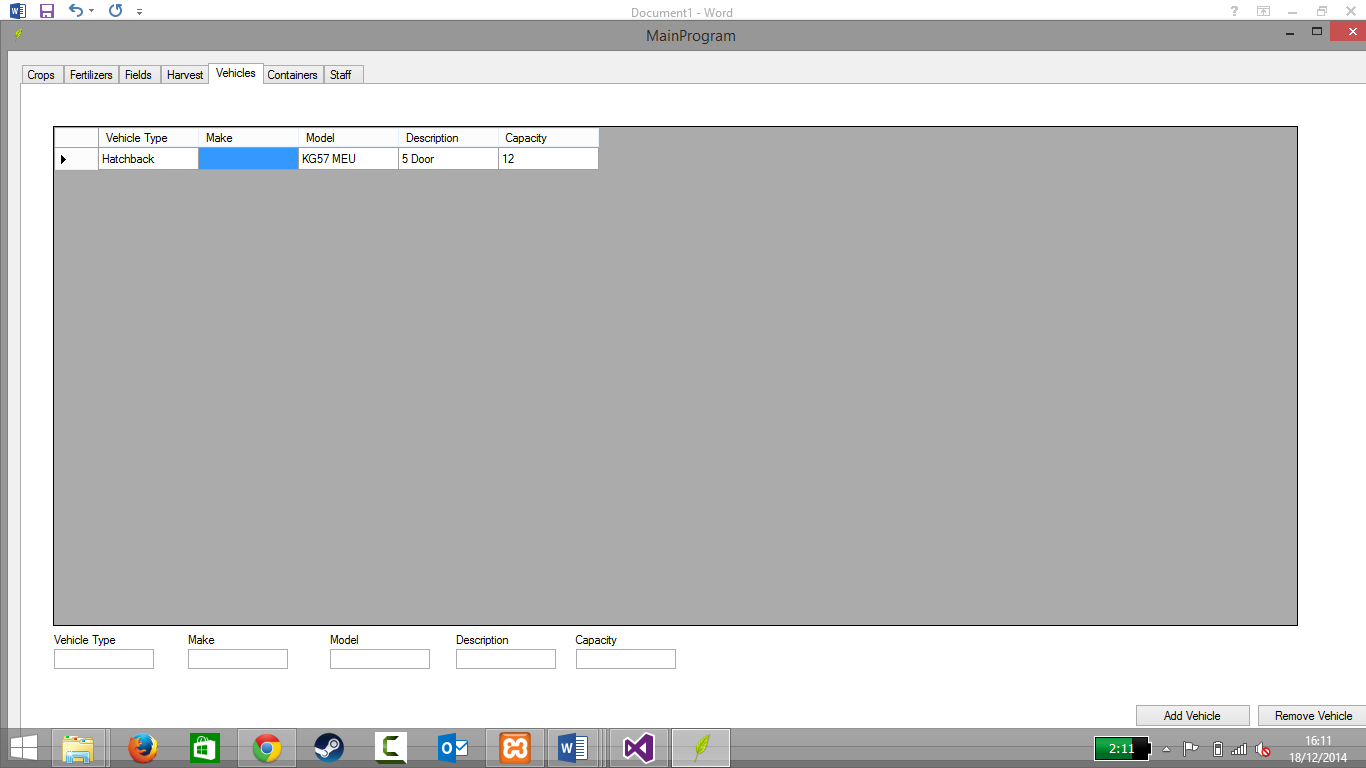
This shows the data being changed with letters



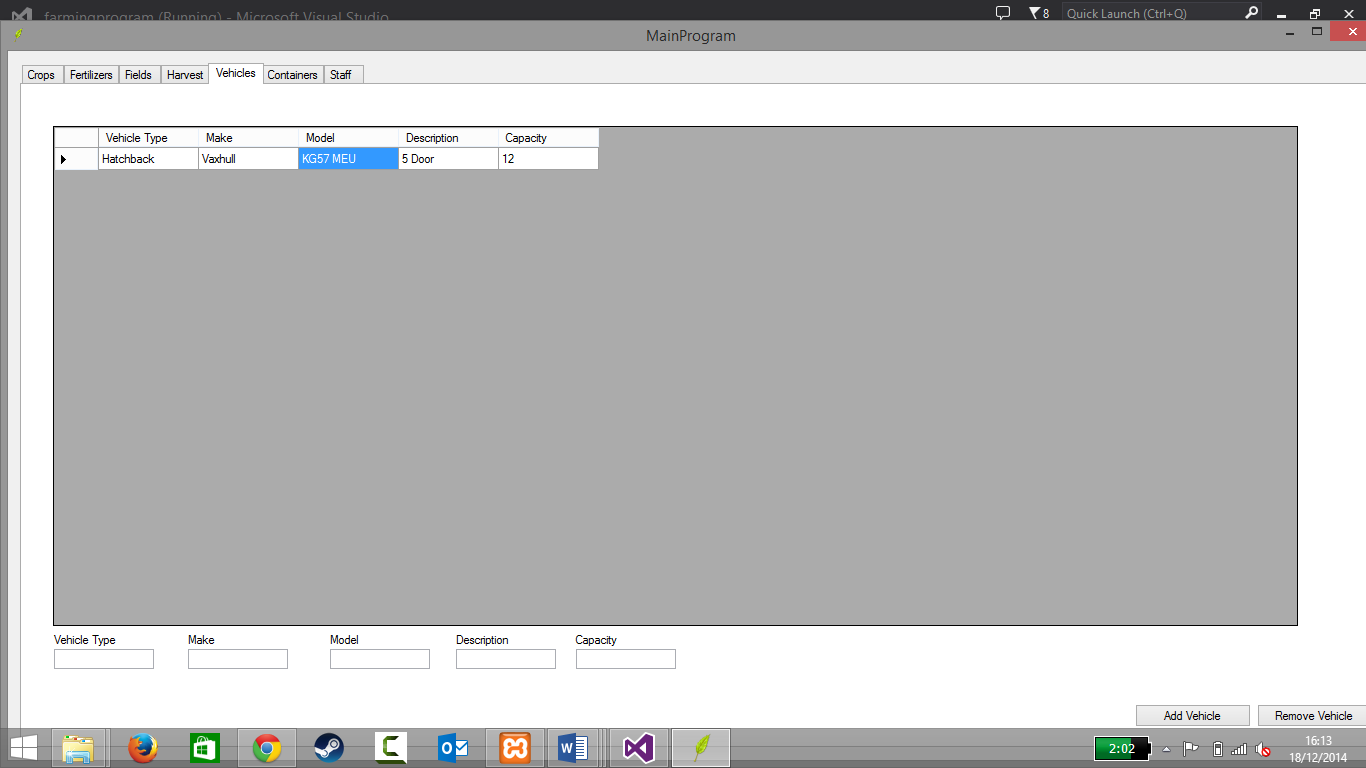
This shows the data being changed with only numbers



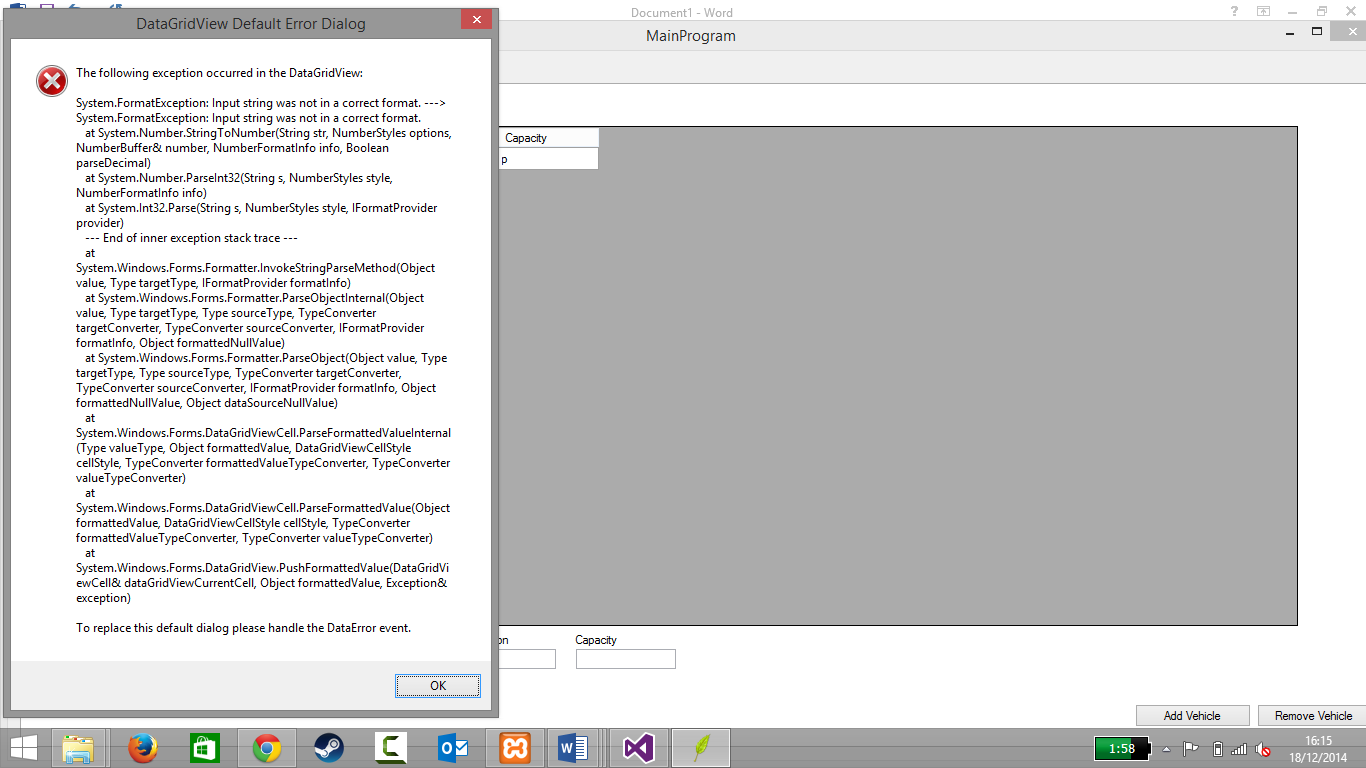
This shows letters and numbers changing the data



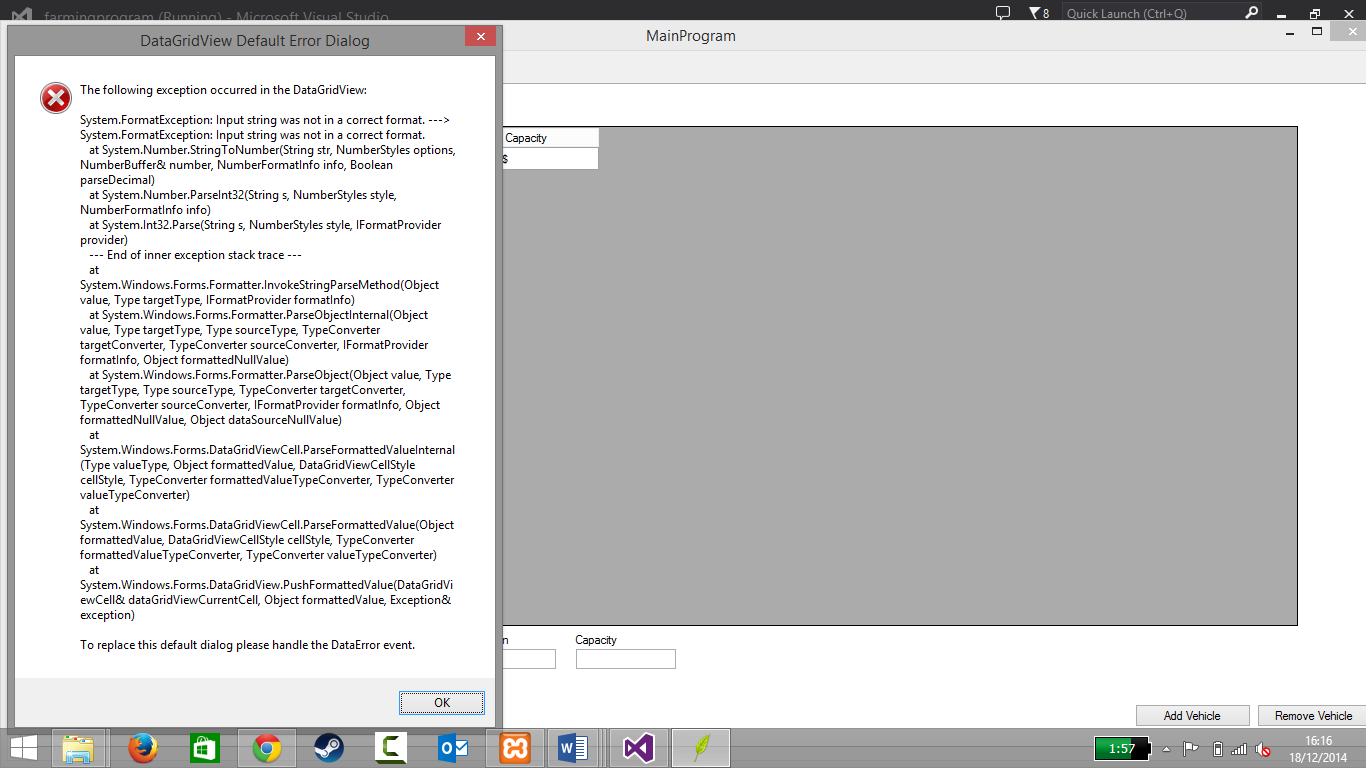
This shows the data being left blank



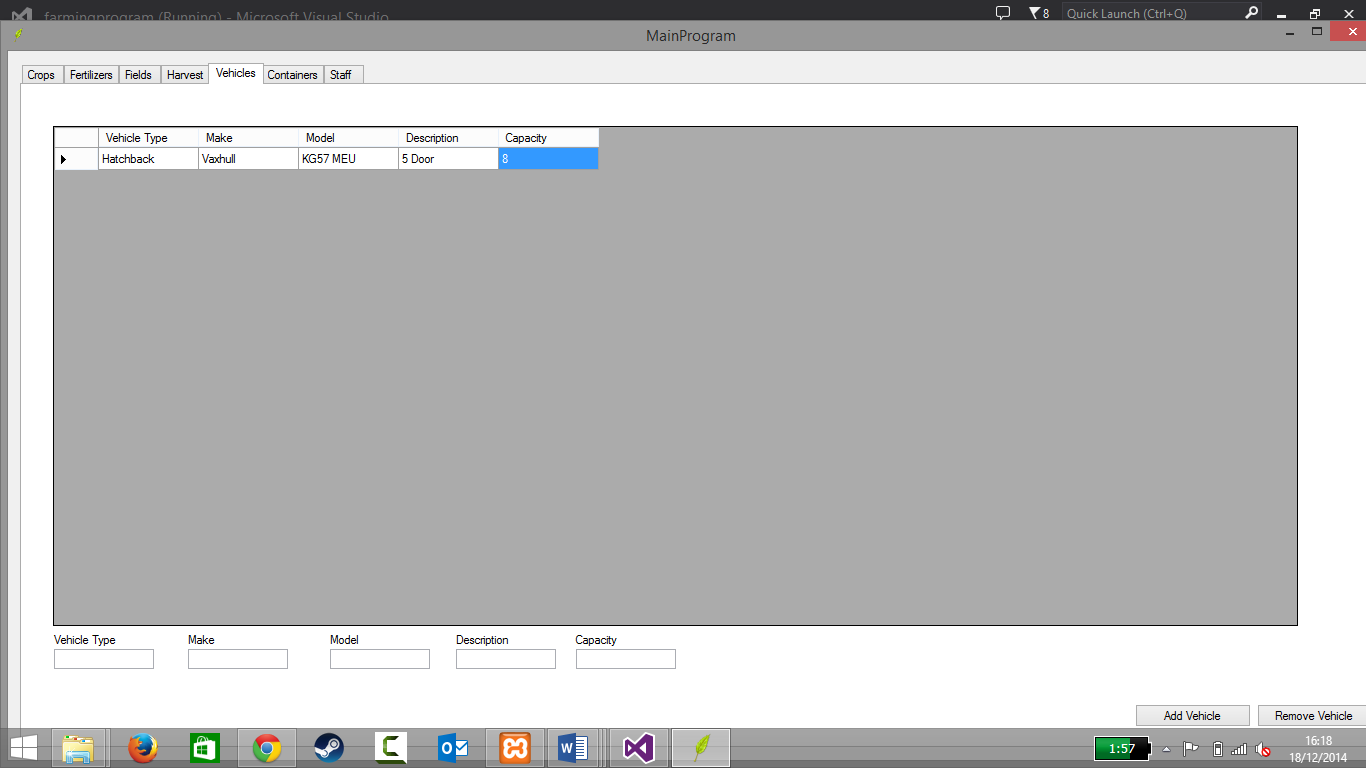
This shows that because there are letters and numbers in that both can be used to change it



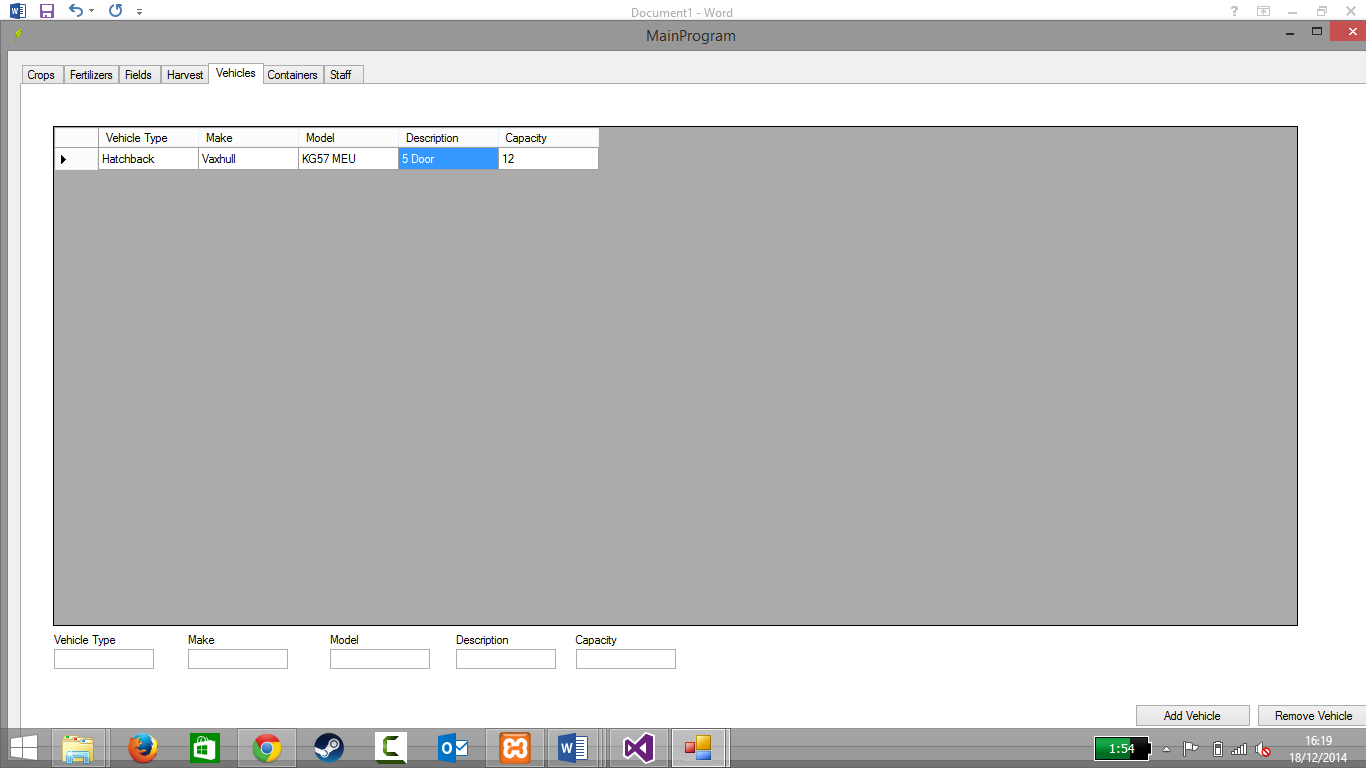
This shows the error thrown if you put a letter in the capacity



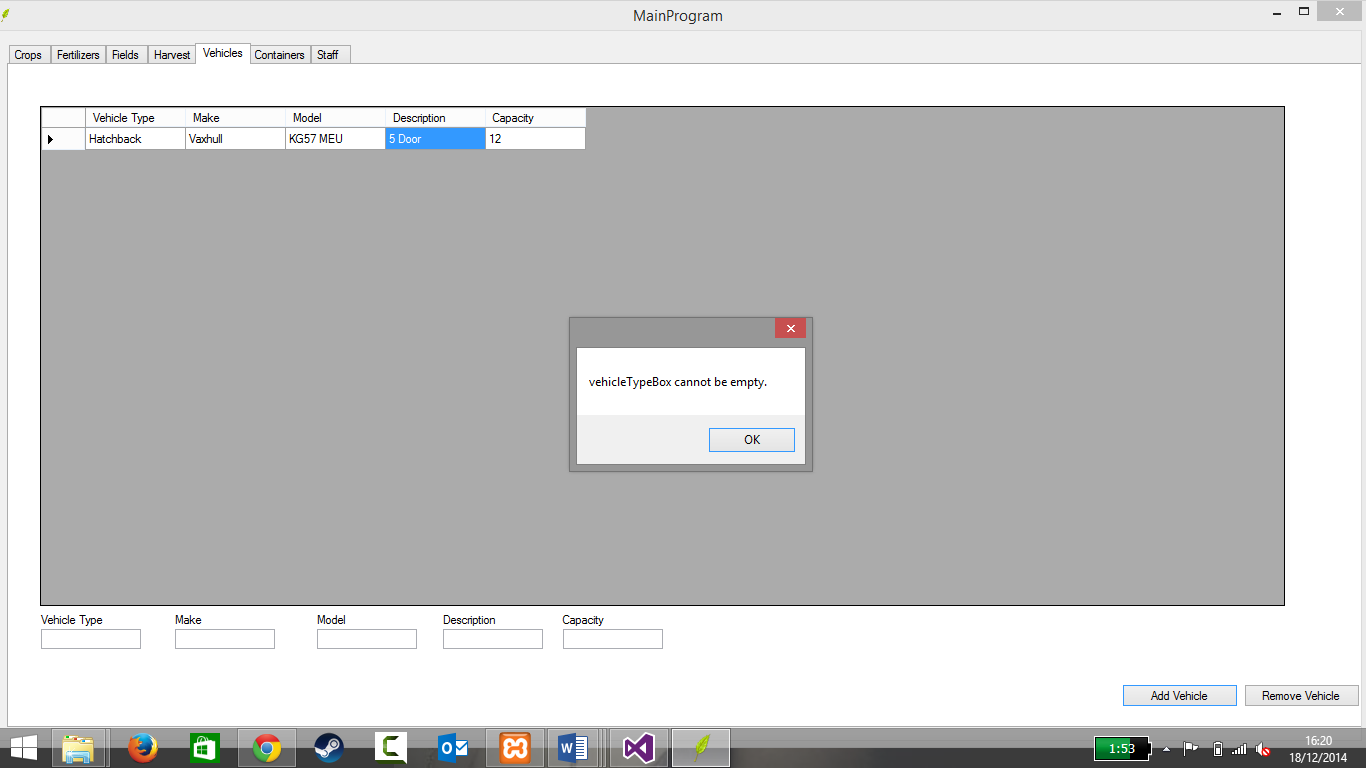
This shows the error thrown if a symbol is used



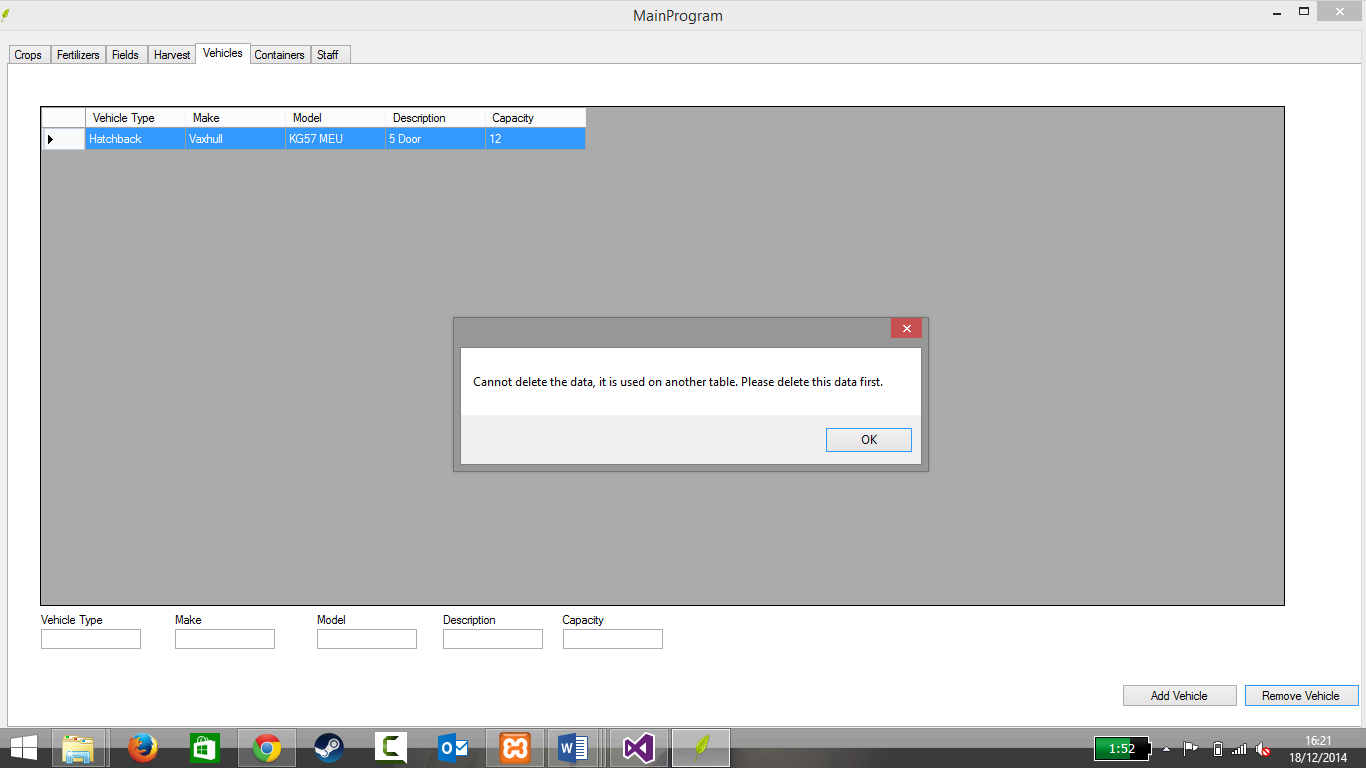
This screenshot shows there is no error is a number is entered



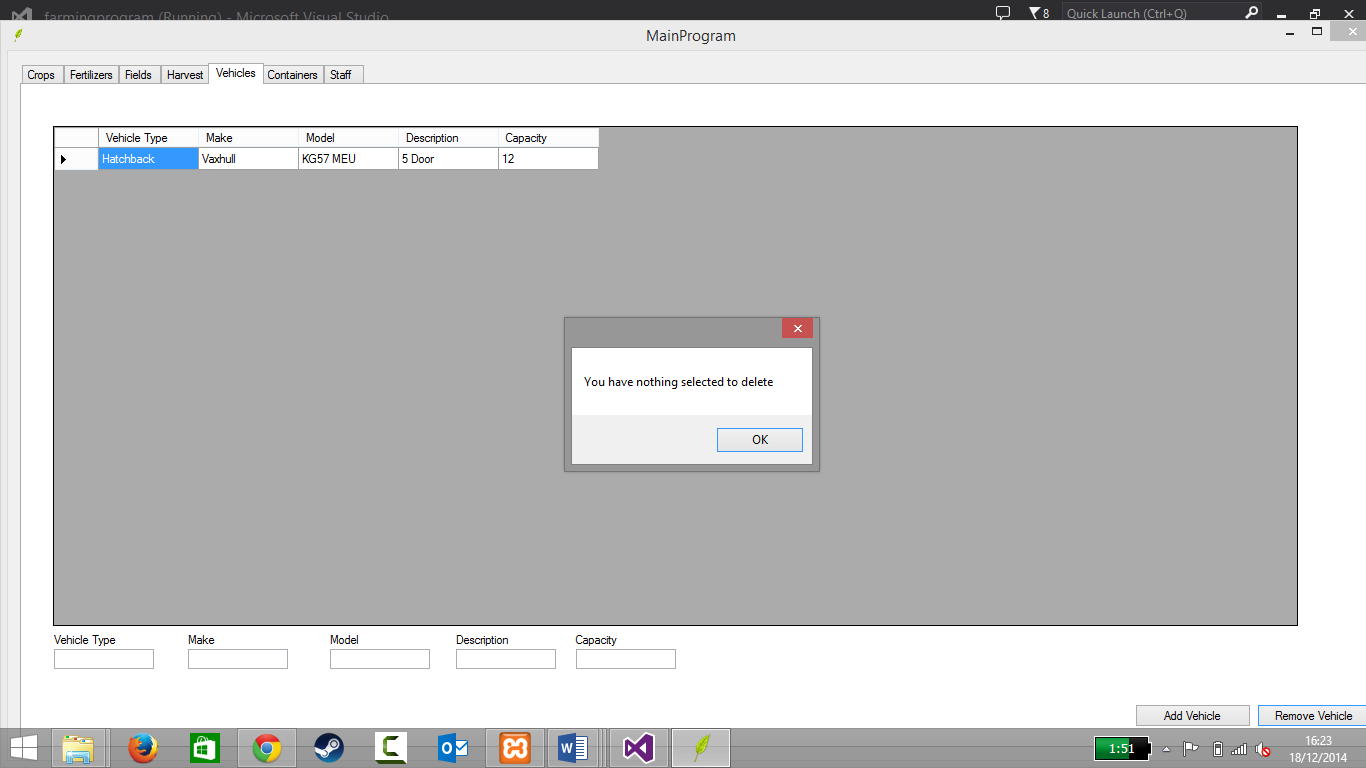
The screenshot shows that with a mixture of letter and numbers that it is assumed that because it’s a string it will work with only letters or numbers?



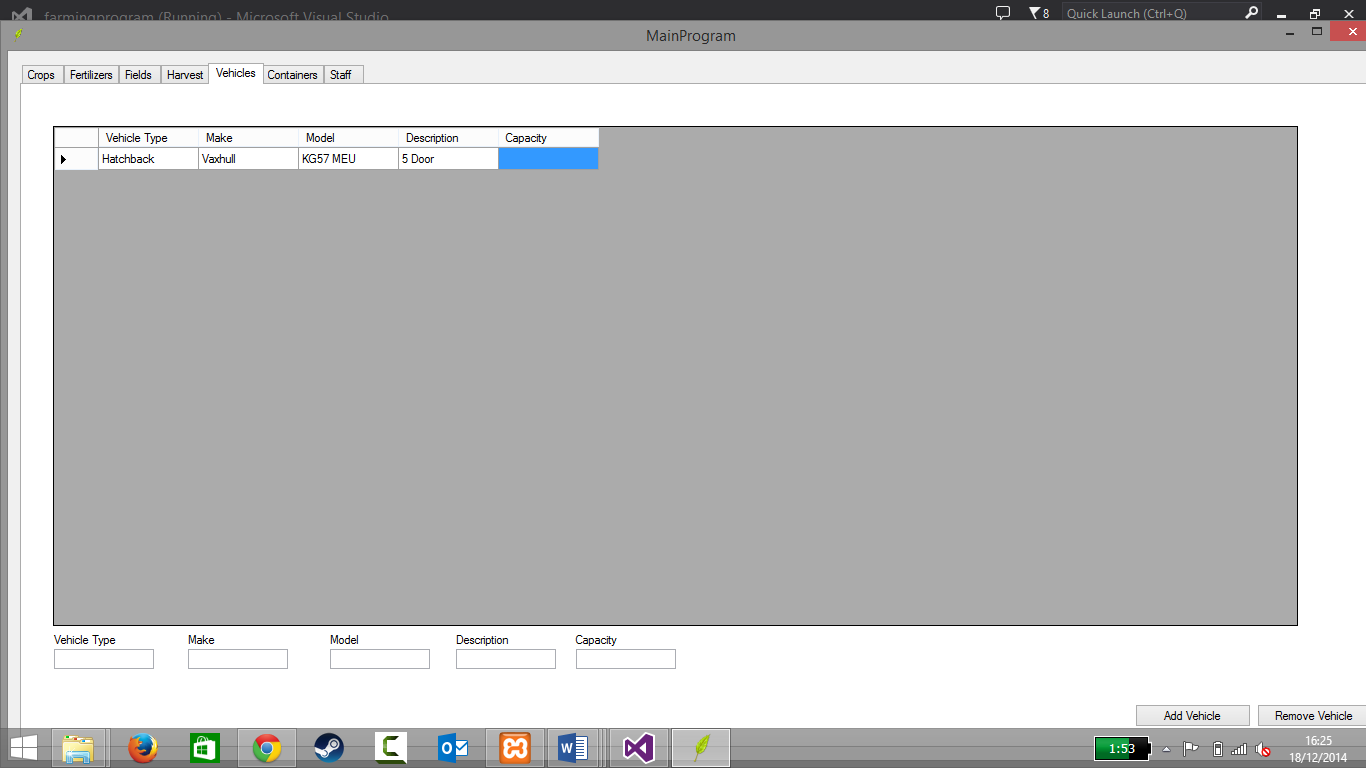
This screenshot shows the message box that comes up when the add vehicle is clicked without a row selected



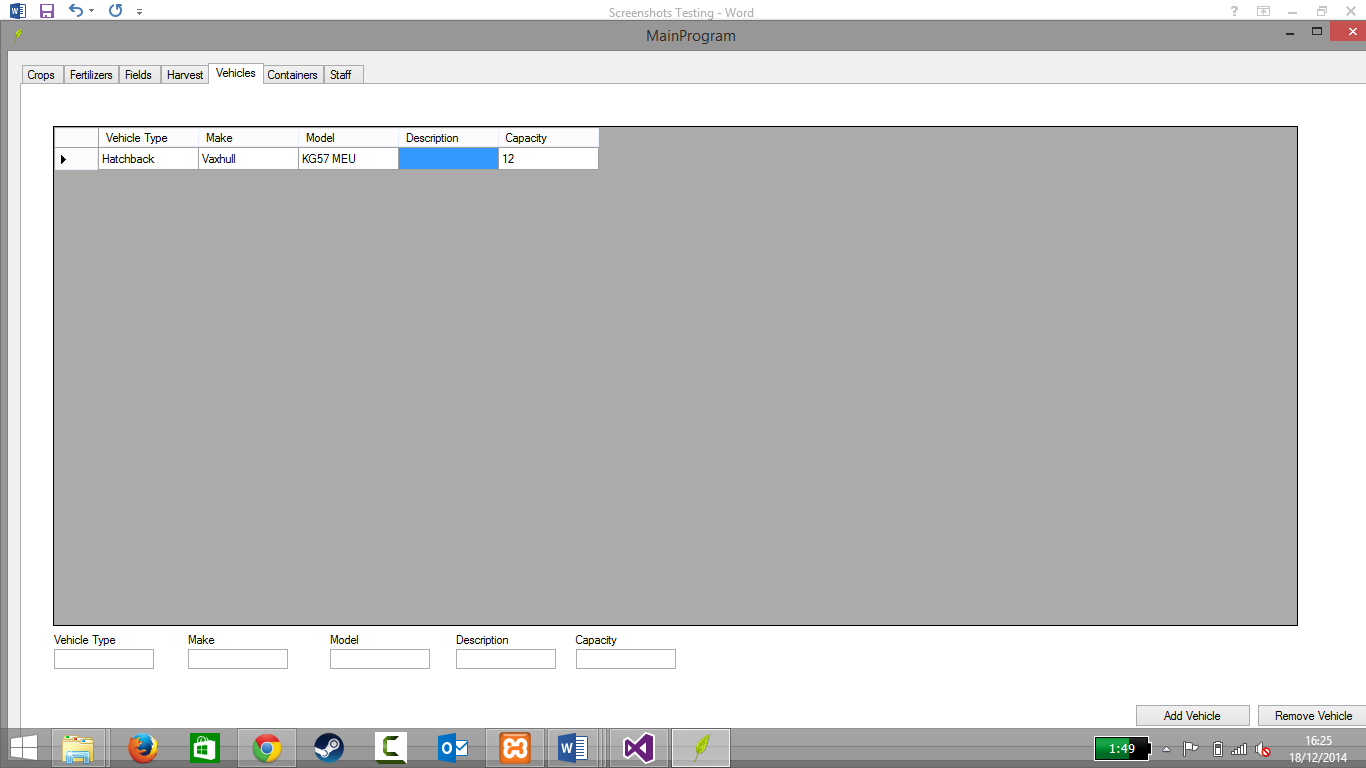
This screenshot shows the message that pops up when the remove vehicle is clicked and when the data is being used by another tabled such as harvest.



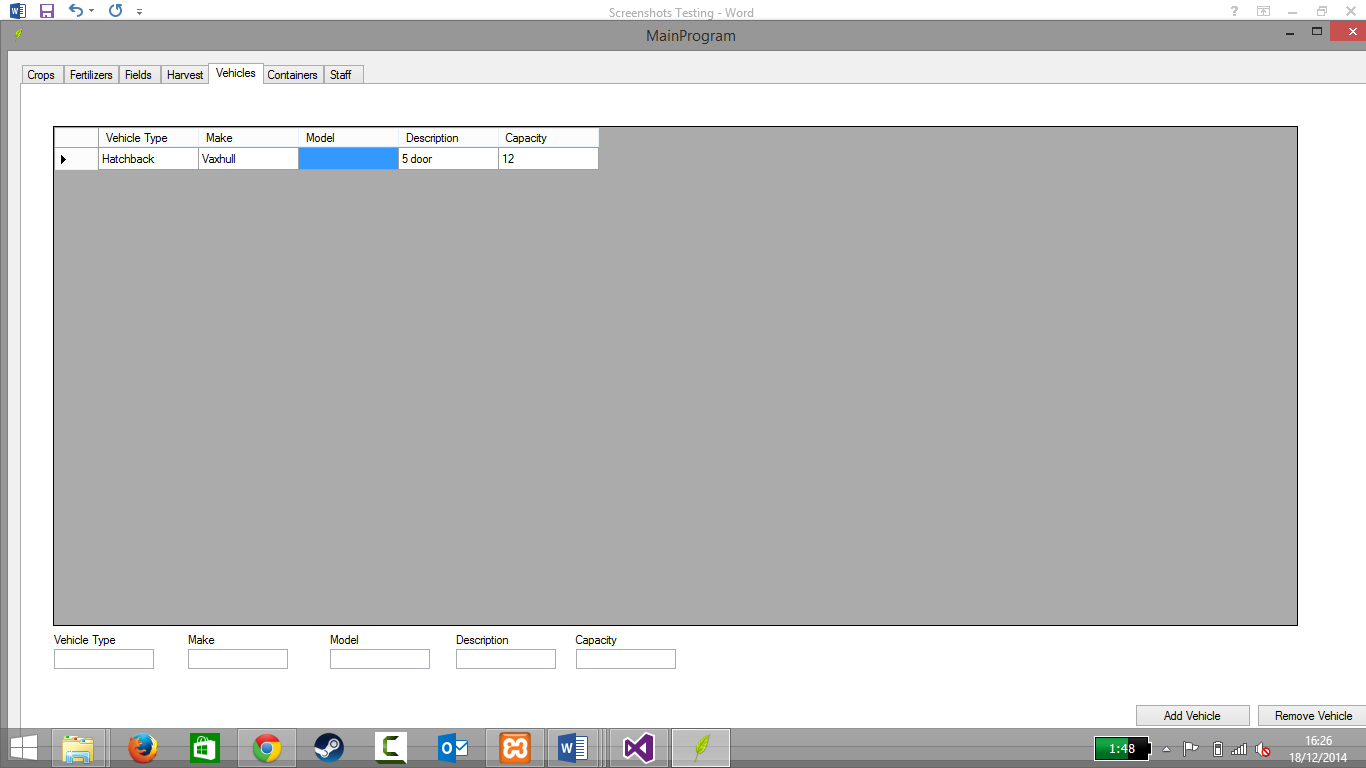
This screenshot shows the message box that comes up when the remove vehicle is deleted without the row being selected.



This screenshot shows the capacity left blank



This screenshot shows the description blank



This screenshot shows the model left blank

8.2 Testing Log <>

8.3 Testing Log<>

8.4 Testing Log<>

8.5 testing Log<>

# 7. Program Management