Milestone 6

Ian M. McConihay

College of Science, Engineering and Technology, Grand Canyon University

CST-150: C# Programming I

Mark Smithers

September 29, 2024

Video Link:

https://www.loom.com/share/2c19c453df1140aa8925cb629b61bf72?sid=f2780488-6859-46d2-9f64-3a9c117d5538

Video Link cont.:

https://www.loom.com/share/dbda952a3d844ff6b485ad7fa3bea02a?sid=d026cb35-d85c-4d3b-b981-cdf659959491

Github: https://github.com/Ian-McConihay/CST-150

What was challenging?

Implementing the sort for columns was difficult at first but it ended up being a simple fix.

What did you learn?

CRUD functionality in WinForms.

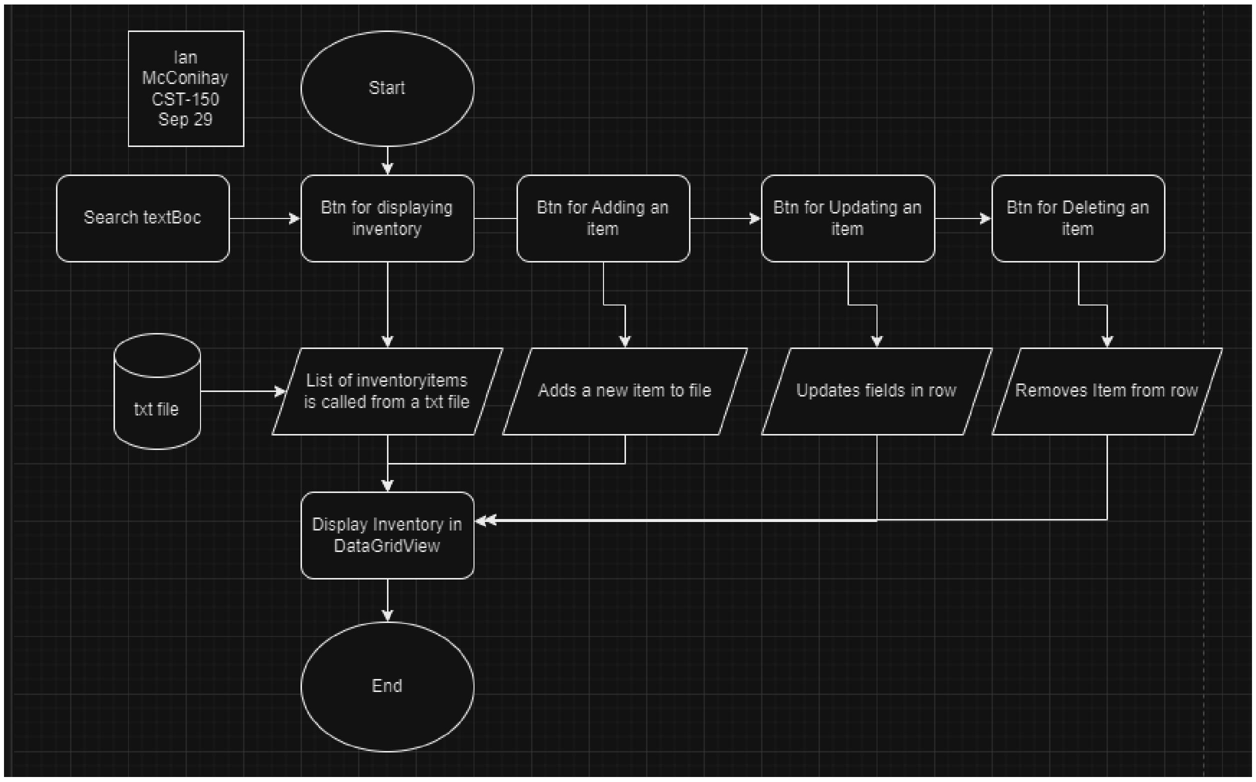
How would you improve on the project?

Create a menu or initial view to load CRUD operations

How can you use what you learned on the job?

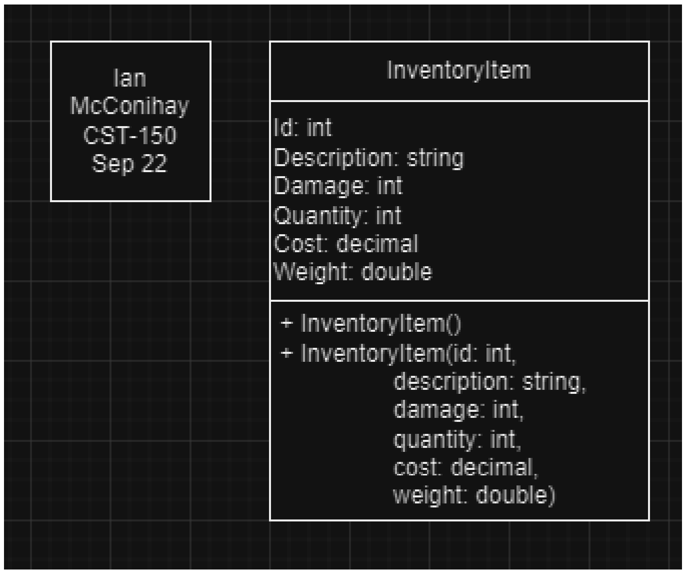
CRUD is the core for the bulk of application functionality.

Figure 1: FlowChart



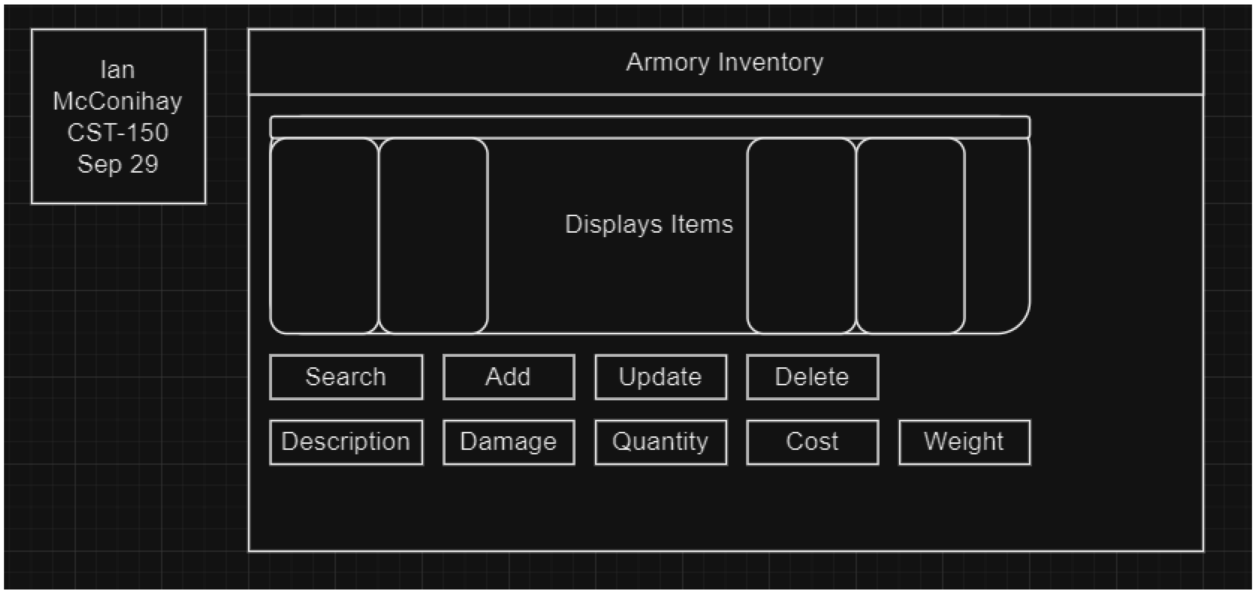
At the start of this application the text file data persists into a grid view of inventory items. There will be a series of buttons to perform Adding items, updating, deleting, and a search box. For the update button there will be text fields for the user to enter in their items information.

Figure 2: UML InventoryItem



No Changes.

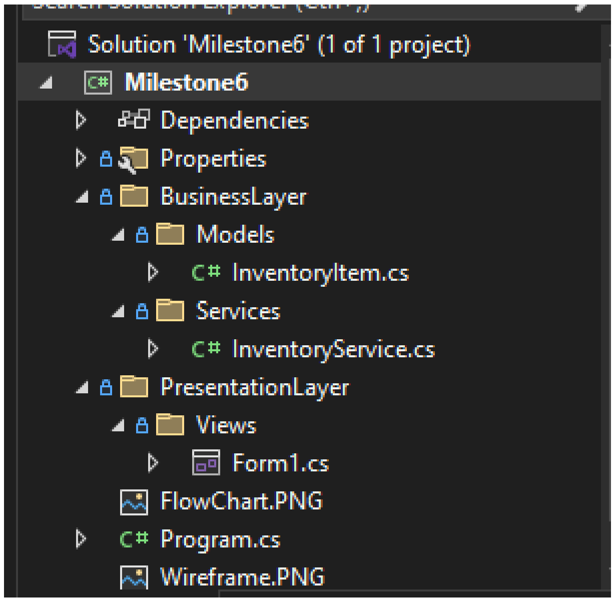
Figure 3: Wireframe



The updated wire frame has a handful of changes. The form has a name when displayed for the user. There are now a series of buttons to manage the inventory items. A few items and the table have been adjusted as well. The user will have columns to display the information.

N-Layer

Figure 4: N-Layer

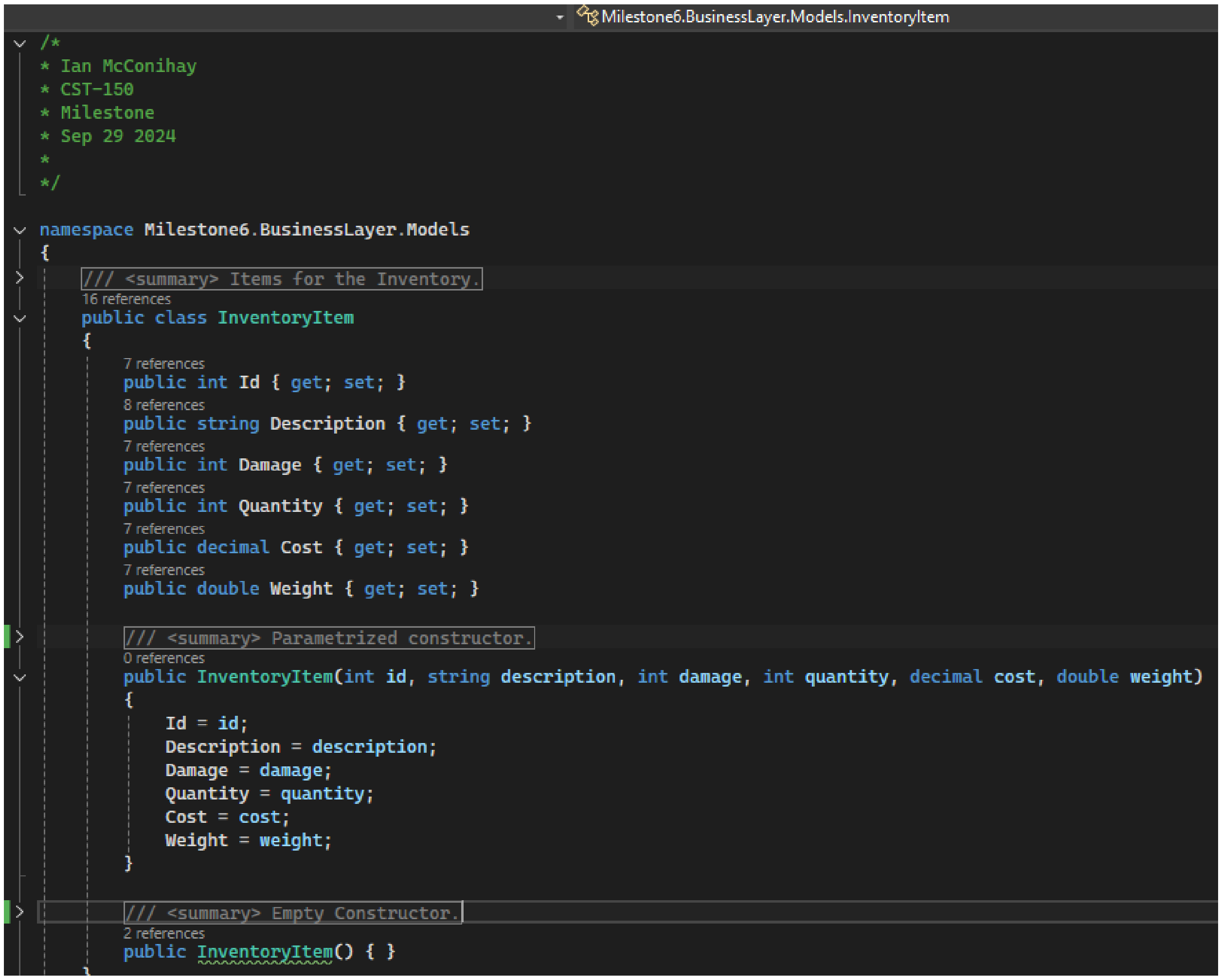


```
Solution 'Milestone6' (1 of 1 project)
  Milestone6
    Dependencies
    Properties
    BusinessLayer
      Models
        InventoryItem.cs
      Services
        InventoryService.cs
    PresentationLayer
      Views
        Form1.cs
    FlowChart.PNG
    Program.cs
    Wireframe.PNG
```

Here is a screenshot of the file structure for the application. N-layer was required for milestone. InventoryItem and InventoryService has been moved to the BusinessLayer and The PresentationLayer contains the Main form for design.

Application Screenshots

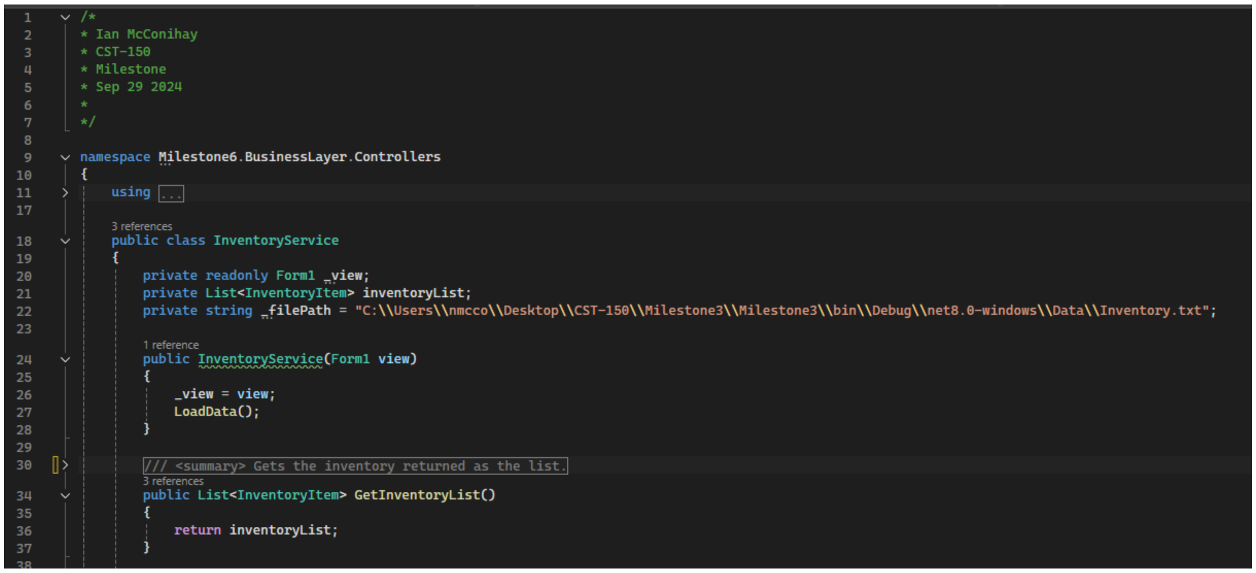
Figure 5: Code



```
Milestone6.BusinessLayer.Models.InventoryItem

/*
 * Ian McConihay
 * CST-150
 * Milestone
 * Sep 29 2024
 *
 */

namespace Milestone6.BusinessLayer.Models
{
    /// <summary> Items for the Inventory.
    16 references
    public class InventoryItem
    {
        7 references
        public int Id { get; set; }
        8 references
        public string Description { get; set; }
        7 references
        public int Damage { get; set; }
        7 references
        public int Quantity { get; set; }
        7 references
        public decimal Cost { get; set; }
        7 references
        public double Weight { get; set; }

        /// <summary> Parametrized constructor.
        0 references
        public InventoryItem(int id, string description, int damage, int quantity, decimal cost, double weight)
        {
            Id = id;
            Description = description;
            Damage = damage;
            Quantity = quantity;
            Cost = cost;
            Weight = weight;
        }

        /// <summary> Empty Constructor.
        2 references
        public InventoryItem() { }
```

Figure 5 shows that I have created a Models folder. In the model folder I now will store my InventoryItem and any other class objects to be added. The InventoryItem has only added an empty constructor.

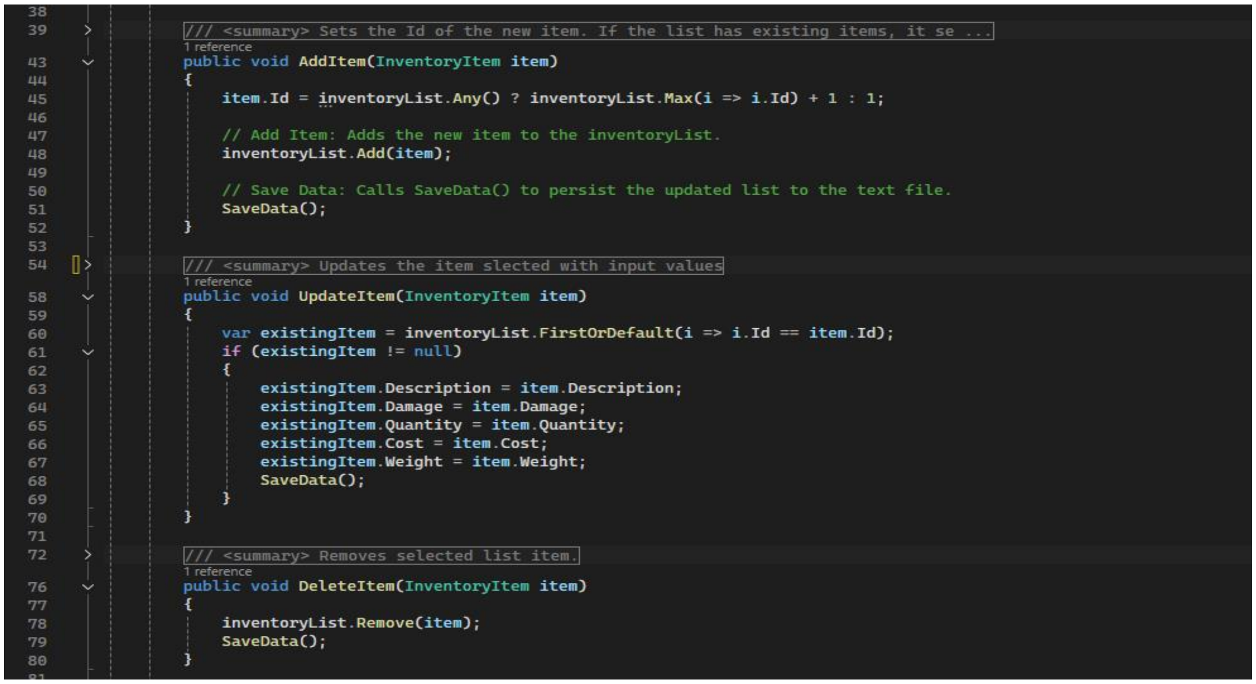
Figure 6: Code



```
/*
 * Ian McConihay
 * CST-150
 * Milestone
 * Sep 29 2024
 *
 */

namespace Milestone6.BusinessLayer.Controllers
{
    using ...

    3 references
    public class InventoryService
    {
        private readonly Form1 _view;
        private List<InventoryItem> inventoryList;
        private string _filePath = "C:\\Users\\nmcco\\Desktop\\CST-150\\Milestone3\\Milestone3\\bin\\Debug\\net8.0-windows\\Data\\Inventory.txt";

        1 reference
        public InventoryService(Form1 view)
        {
            _view = view;
            LoadData();
        }

        /// <summary> Gets the inventory returned as the list.
        3 references
        public List<InventoryItem> GetInventoryList()
        {
            return inventoryList;
        }
```

Figure 6 shows the Inventory service starting with the citation. We have the inventoryList and filePath being initialized at the top. Then the GetInventoryList method which is used continuously in the main form to pull the backed list.

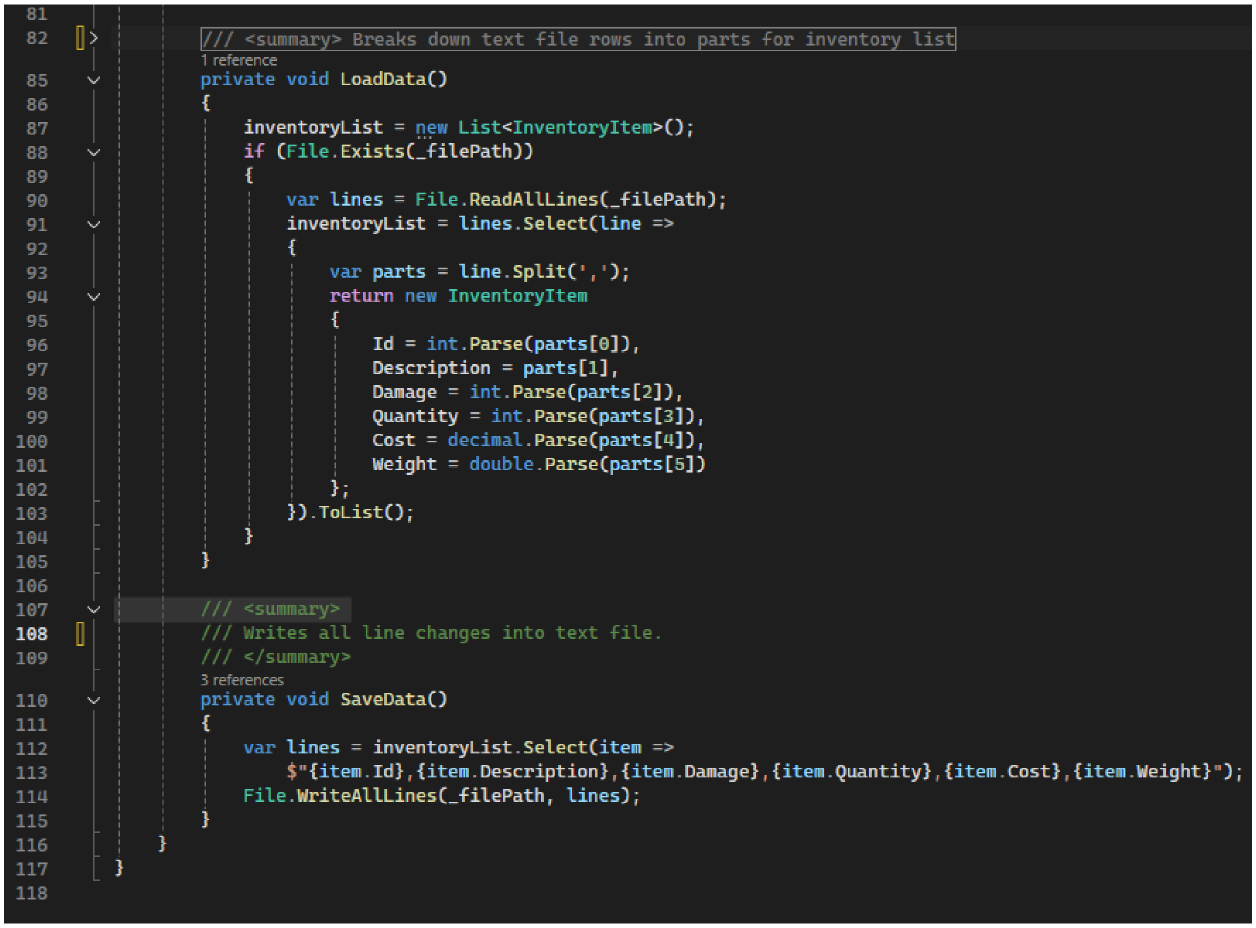
Figure 7: Code



```
        /// <summary> Sets the Id of the new item. If the list has existing items, it se ...
        1 reference
        public void AddItem(InventoryItem item)
        {
            item.Id = inventoryList.Any() ? inventoryList.Max(i => i.Id) + 1 : 1;

            // Add Item: Adds the new item to the inventoryList.
            inventoryList.Add(item);

            // Save Data: Calls SaveData() to persist the updated list to the text file.
            SaveData();
        }

        /// <summary> Updates the item slected with input values
        1 reference
        public void UpdateItem(InventoryItem item)
        {
            var existingItem = inventoryList.FirstOrDefault(i => i.Id == item.Id);
            if (existingItem != null)
            {
                existingItem.Description = item.Description;
                existingItem.Damage = item.Damage;
                existingItem.Quantity = item.Quantity;
                existingItem.Cost = item.Cost;
                existingItem.Weight = item.Weight;
                SaveData();
            }
        }

        /// <summary> Removes selected list item.
        1 reference
        public void DeleteItem(InventoryItem item)
        {
            inventoryList.Remove(item);
            SaveData();
        }
```

Figure 7 goes over the AddItem method that takes in information from the main form and then increments the id to whatever the max id is plus 1. The UpdateItem method replaces the items values with the main form method. DeleteItem removes the selected item from the inventoryList.

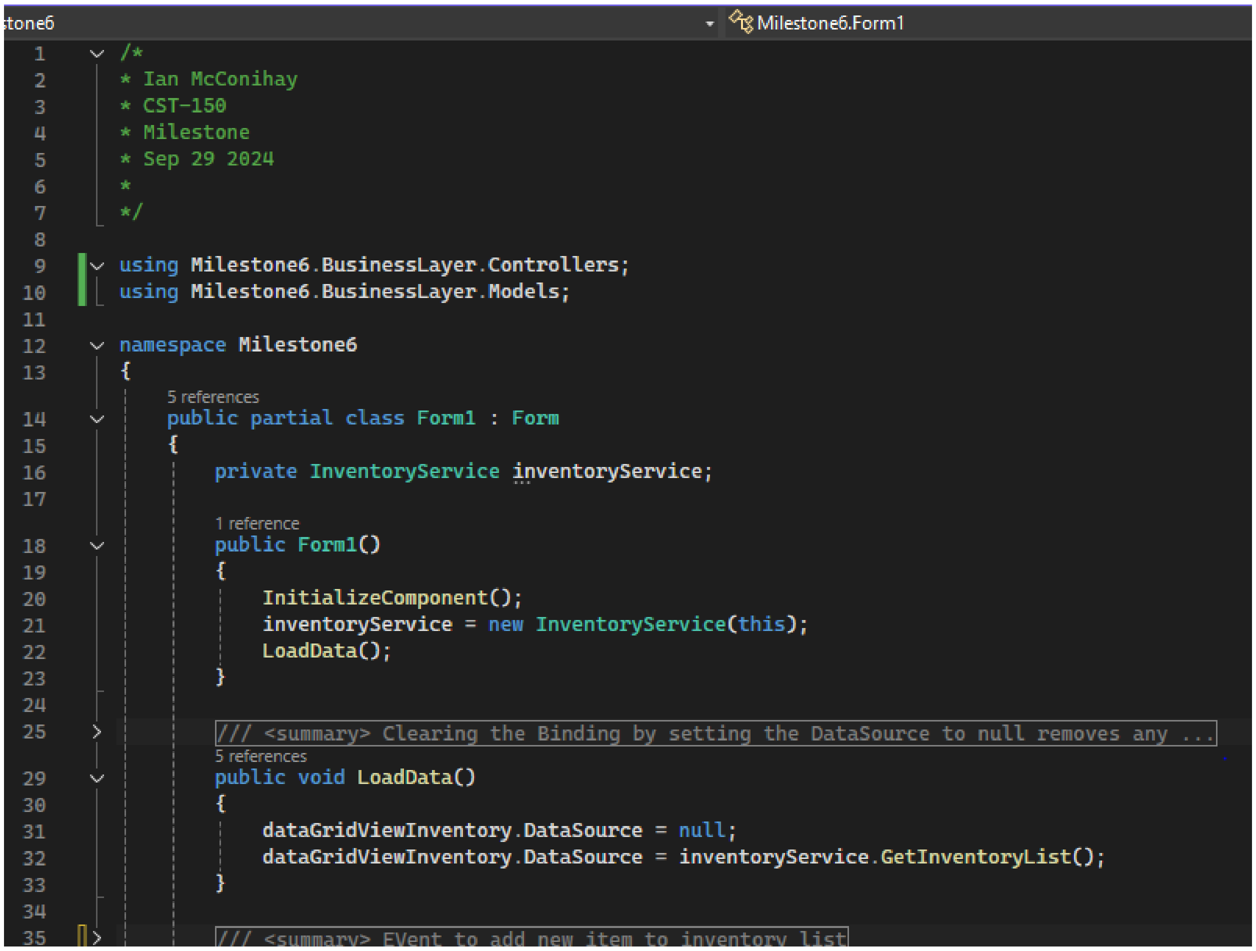
Figure 8: Code



```
        /// <summary> Breaks down text file rows into parts for inventory list
        1 reference
        private void LoadData()
        {
            inventoryList = new List<InventoryItem>();
            if (File.Exists(_filePath))
            {
                var lines = File.ReadAllLines(_filePath);
                inventoryList = lines.Select(line =>
                {
                    var parts = line.Split(',');
                    return new InventoryItem
                    {
                        Id = int.Parse(parts[0]),
                        Description = parts[1],
                        Damage = int.Parse(parts[2]),
                        Quantity = int.Parse(parts[3]),
                        Cost = decimal.Parse(parts[4]),
                        Weight = double.Parse(parts[5])
                    };
                }).ToList();
            }
        }

        /// <summary>
        /// Writes all line changes into text file.
        /// </summary>
        3 references
        private void SaveData()
        {
            var lines = inventoryList.Select(item =>
                $"{item.Id},{item.Description},{item.Damage},{item.Quantity},{item.Cost},{item.Weight}");
            File.WriteAllLines(_filePath, lines);
        }
    }
}
```

Figure 8 has LoadData method that creates a new inventoryList to parse and read through the lines. The SaveData method then will take in the file path and the new inventoryList and it will override the current text file and write over it.

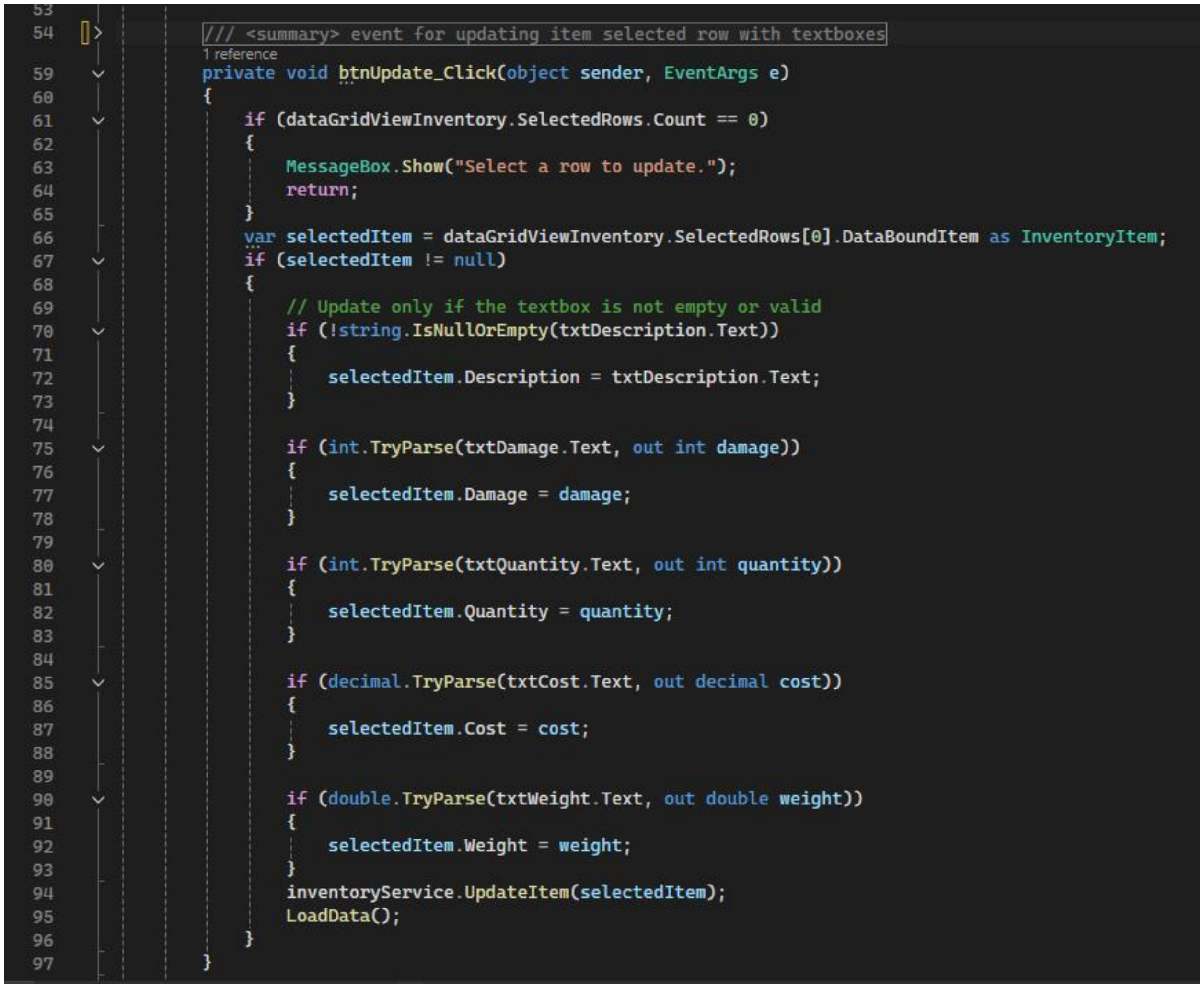
Figure 9: Code



```
stone6                                              Milestone6.Form1
/*
* Ian McConihay
* CST-150
* Milestone
* Sep 29 2024
*
*/

using Milestone6.BusinessLayer.Controllers;
using Milestone6.BusinessLayer.Models;

namespace Milestone6
{
    5 references
    public partial class Form1 : Form
    {
        private InventoryService inventoryService;

        1 reference
        public Form1()
        {
            InitializeComponent();
            inventoryService = new InventoryService(this);
            LoadData();
        }

        /// <summary> Clearing the Binding by setting the DataSource to null removes any ...
        5 references
        public void LoadData()
        {
            dataGridViewInventory.DataSource = null;
            dataGridViewInventory.DataSource = inventoryService.GetInventoryList();
        }

        /// <summary> EVent to add new item to inventory list
```

Figure 9 starts the main form off with the citation. We initialize our inventory service so the business layer can communicate with the presentation layer. Then the method LoadData for setting the Datasource for the dataGridInventory. We set it to null to clear and reset the binding when we have to call LoadData after changes.

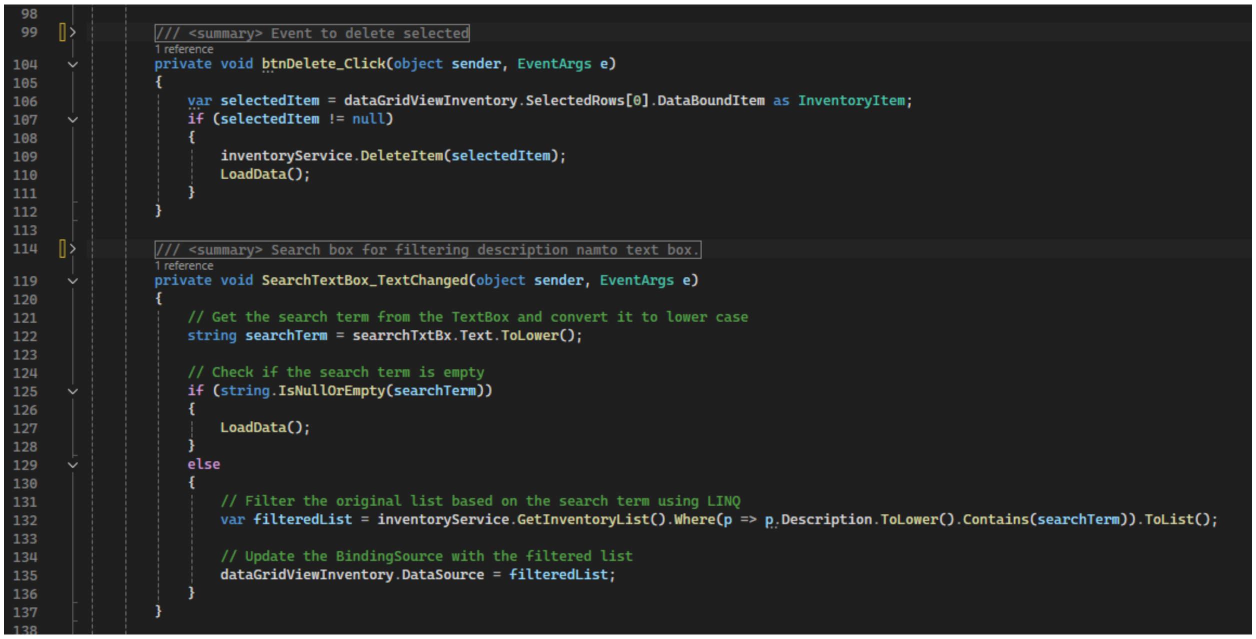
Figure 10: Code



```
        /// <summary> event for updating item selected row with textboxes
        1 reference
        private void btnUpdate_Click(object sender, EventArgs e)
        {
            if (dataGridViewInventory.SelectedRows.Count == 0)
            {
                MessageBox.Show("Select a row to update.");
                return;
            }
            var selectedItem = dataGridViewInventory.SelectedRows[0].DataBoundItem as InventoryItem;
            if (selectedItem != null)
            {
                // Update only if the textbox is not empty or valid
                if (!string.IsNullOrEmpty(txtDescription.Text))
                {
                    selectedItem.Description = txtDescription.Text;
                }

                if (int.TryParse(txtDamage.Text, out int damage))
                {
                    selectedItem.Damage = damage;
                }

                if (int.TryParse(txtQuantity.Text, out int quantity))
                {
                    selectedItem.Quantity = quantity;
                }

                if (decimal.TryParse(txtCost.Text, out decimal cost))
                {
                    selectedItem.Cost = cost;
                }

                if (double.TryParse(txtWeight.Text, out double weight))
                {
                    selectedItem.Weight = weight;
                }
                inventoryService.UpdateItem(selectedItem);
                LoadData();
            }
        }
```

Figure 10 screenshot only contains the update click event. This event has I added some logic so that you can update individual values for an inventory item. At the end we call our service and then reload the data.

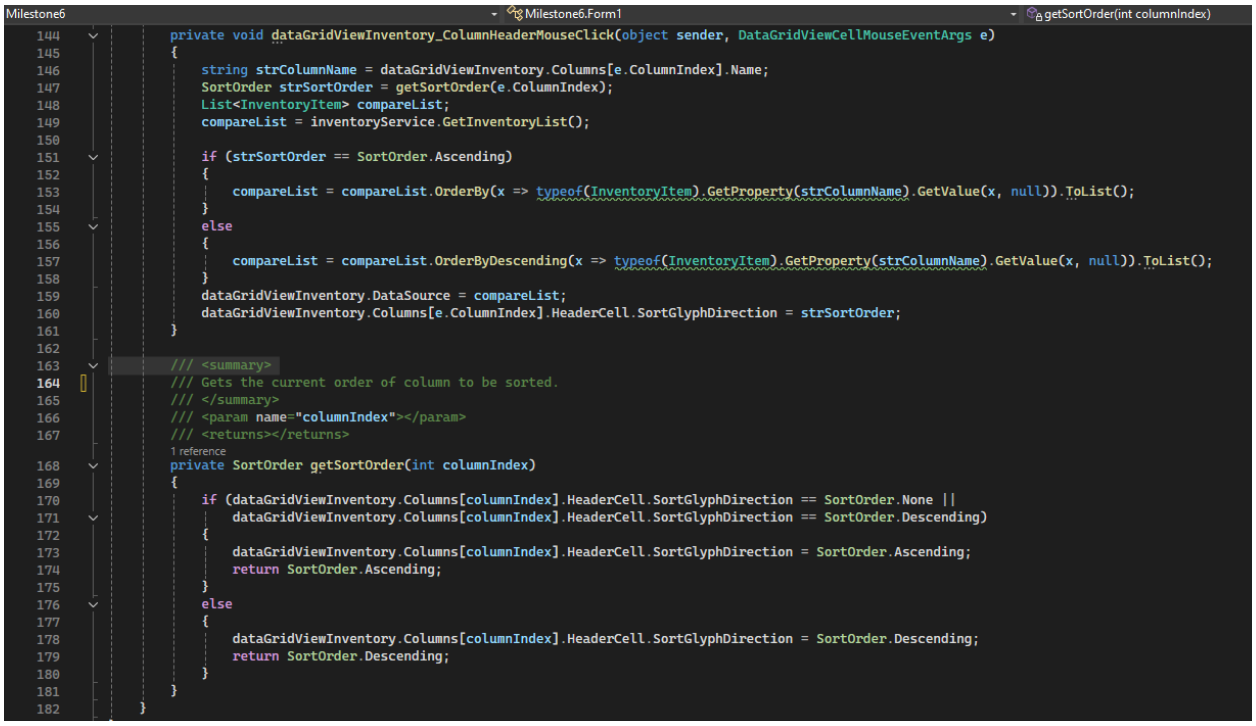
Figure 11: Code



```
/// <summary> Event to delete selected
1 reference
private void btnDelete_Click(object sender, EventArgs e)
{
    var selectedItem = dataGridViewInventory.SelectedRows[0].DataBoundItem as InventoryItem;
    if (selectedItem != null)
    {
        inventoryService.DeleteItem(selectedItem);
        LoadData();
    }
}

/// <summary> Search box for filtering description namto text box.
1 reference
private void SearchTextBox_TextChanged(object sender, EventArgs e)
{
    // Get the search term from the TextBox and convert it to lower case
    string searchTerm = searrchTxtBx.Text.ToLower();

    // Check if the search term is empty
    if (string.IsNullOrEmpty(searchTerm))
    {
        LoadData();
    }
    else
    {
        // Filter the original list based on the search term using LINQ
        var filteredList = inventoryService.GetInventoryList().Where(p => p.Description.ToLower().Contains(searchTerm)).ToList();

        // Update the BindingSource with the filtered list
        dataGridViewInventory.DataSource = filteredList;
    }
}
```

Figure 11 has the delete click event that requires the user to select a row to be deleted. Next is the search method using the text box the user can enter a description. The textbox then will filter the list using LINQ to see if the list contains the description.

Figure 12: Code

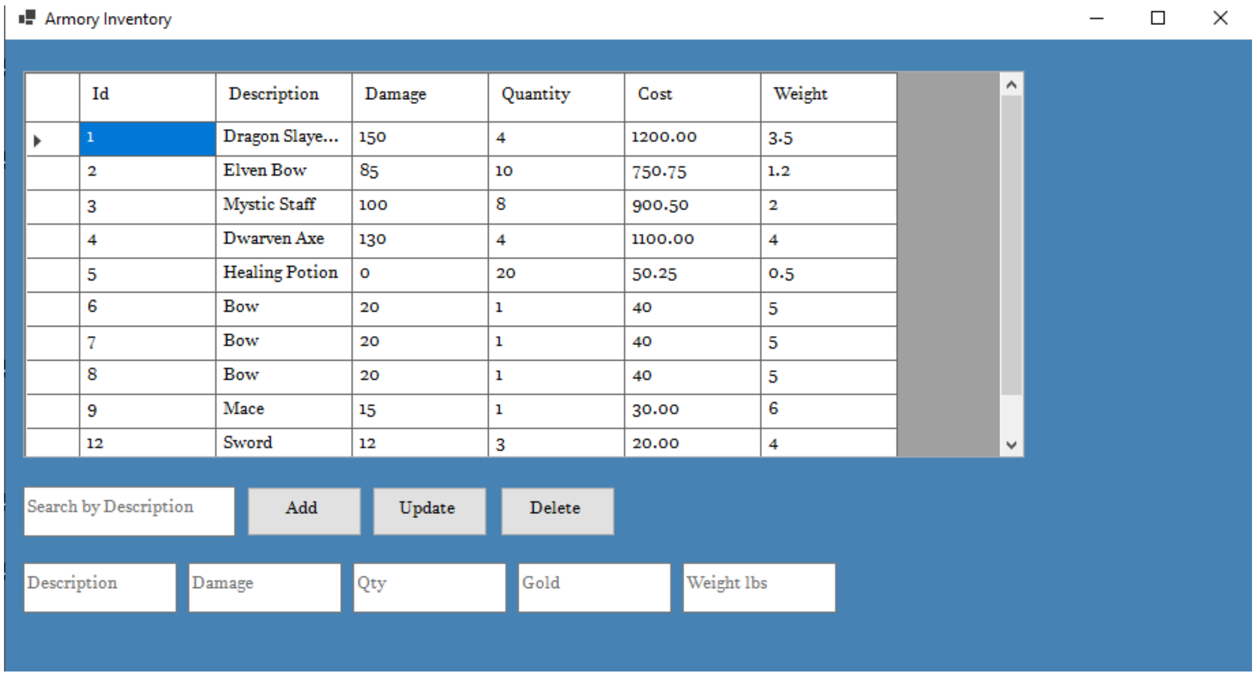


Milestone6 | Milestone6.Form1 | getSortOrder(int columnIndex)

```
        private void dataGridViewInventory_ColumnHeaderMouseClick(object sender, DataGridViewCellMouseEventArgs e)
        {
            string strColumnName = dataGridViewInventory.Columns[e.ColumnIndex].Name;
            SortOrder strSortOrder = getSortOrder(e.ColumnIndex);
            List<InventoryItem> compareList;
            compareList = inventoryService.GetInventoryList();

            if (strSortOrder == SortOrder.Ascending)
            {
                compareList = compareList.OrderBy(x => typeof(InventoryItem).GetProperty(strColumnName).GetValue(x, null)).ToList();
            }
            else
            {
                compareList = compareList.OrderByDescending(x => typeof(InventoryItem).GetProperty(strColumnName).GetValue(x, null)).ToList();
            }
            dataGridViewInventory.DataSource = compareList;
            dataGridViewInventory.Columns[e.ColumnIndex].HeaderCell.SortGlyphDirection = strSortOrder;
        }

        /// <summary>
        /// Gets the current order of column to be sorted.
        /// </summary>
        /// <param name="columnIndex"></param>
        /// <returns></returns>
        1 reference
        private SortOrder getSortOrder(int columnIndex)
        {
            if (dataGridViewInventory.Columns[columnIndex].HeaderCell.SortGlyphDirection == SortOrder.None ||
                dataGridViewInventory.Columns[columnIndex].HeaderCell.SortGlyphDirection == SortOrder.Descending)
            {
                dataGridViewInventory.Columns[columnIndex].HeaderCell.SortGlyphDirection = SortOrder.Ascending;
                return SortOrder.Ascending;
            }
            else
            {
                dataGridViewInventory.Columns[columnIndex].HeaderCell.SortGlyphDirection = SortOrder.Descending;
                return SortOrder.Descending;
            }
        }
    }
```

Figure 12 contains my biggest struggle that was to sort based on the click on the column's header. The automatic sort of function does not work with List. So, I had to grab the columns name and not the value in order to sort the list to ascending or descending based on the column. The method getSortOrder works with the top method to get the order its currently sorted.

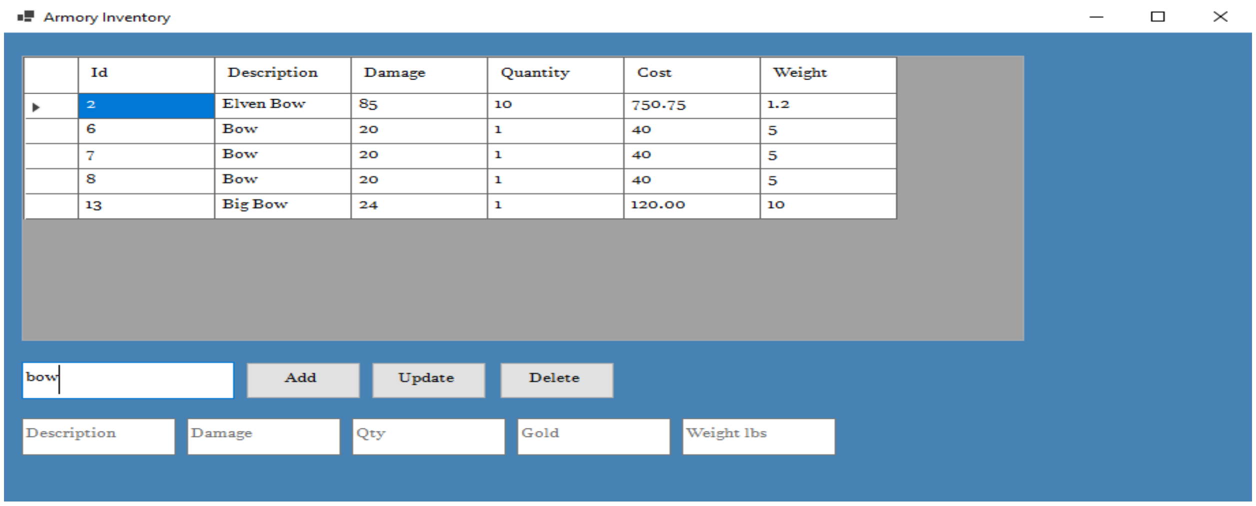
Figure 13: Application Start



Id	Description	Damage	Quantity	Cost	Weight
1	Dragon Slaye...	150	4	1200.00	3.5
2	Elven Bow	85	10	750.75	1.2
3	Mystic Staff	100	8	900.50	2
4	Dwarven Axe	130	4	1100.00	4
5	Healing Potion	0	20	50.25	0.5
6	Bow	20	1	40	5
7	Bow	20	1	40	5
8	Bow	20	1	40	5
9	Mace	15	1	30.00	6
12	Sword	12	3	20.00	4

The start of the application displays all of the controls now available to the user. Global styling has been added for better accessibility. As we can all so many items have been added and also removed to to the jump in Ids.

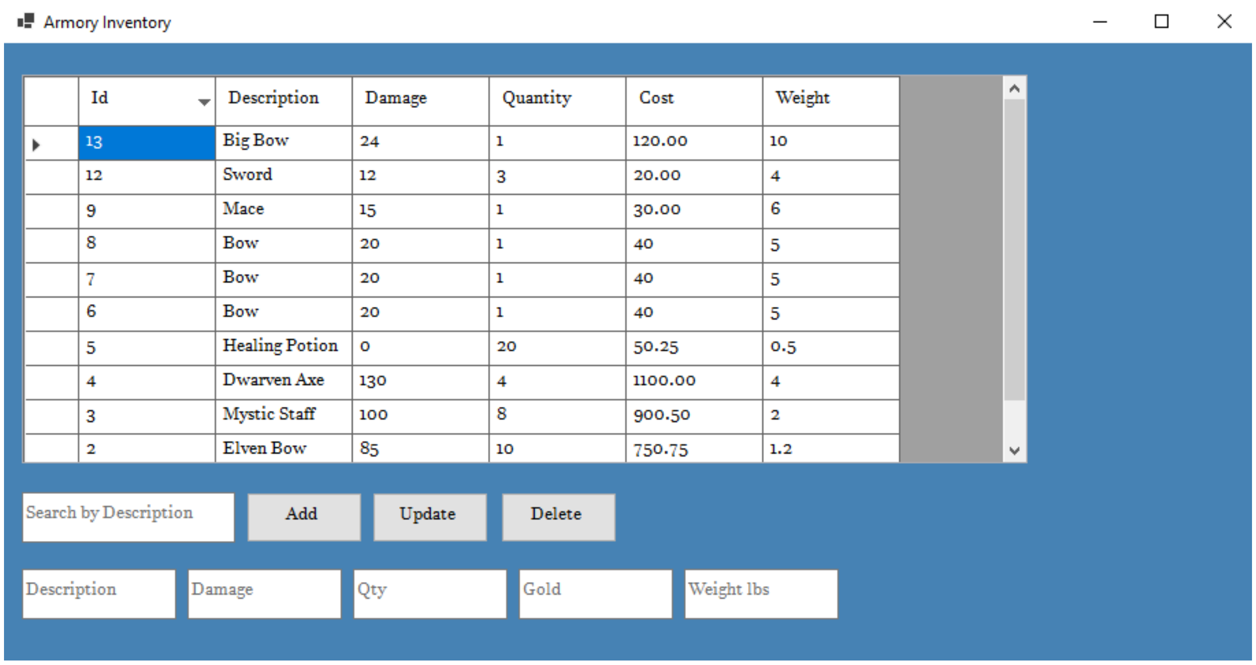
Figure 14: Application Search



Id	Description	Damage	Quantity	Cost	Weight
2	Elven Bow	85	10	750.75	1.2
6	Bow	20	1	40	5
7	Bow	20	1	40	5
8	Bow	20	1	40	5
13	Big Bow	24	1	120.00	10

Here we have searched using the word “bow”. The search brought up a new list of all items containing bow. The search in the video will demonstrate the procedural changes as the word is typed.

Figure 15: Application Sort



Armory Inventory

Id	Description	Damage	Quantity	Cost	Weight
13	Big Bow	24	1	120.00	10
12	Sword	12	3	20.00	4
9	Mace	15	1	30.00	6
8	Bow	20	1	40	5
7	Bow	20	1	40	5
6	Bow	20	1	40	5
5	Healing Potion	0	20	50.25	0.5
4	Dwarven Axe	130	4	1100.00	4
3	Mystic Staff	100	8	900.50	2
2	Elven Bow	85	10	750.75	1.2

This is the sort functionality being displayed. We can see I have selected the ID column. The rows are now sorted with the highest Id down to the lowest.

Bug Reports

Bug Report: NONE

Class name

Method name :

Steps to reproduce the bug:

Expected results

Actual results

details: N/A

Solution

1. List your computer specs (type of computer, OS, memory, etc)

Device name	DESKTOP-IAQ5CCD
Processor	Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz
Installed RAM	8.00 GB (7.88 GB usable)
Device ID	A0AC8D02-4885-4491-B27B-B40F0A0D2E35
Product ID	00356-02139-31547-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	Touch support with 10 touch points

2. Create 3 test cases

Valid File with Proper Data: The method should correctly populate the armoryInventory array with InventoryItem objects based on properly formatted data in the file.

File is Empty: The method should display a warning message indicating the file is empty and leave the armoryInventory array uninitialized.

File with Incorrect Data Format: The method should show an error message indicating an issue with loading data and only initialize valid InventoryItem objects in the armoryInventory array.

3. List 3 Programming conventions that will be used all milestones

Naming, Format, and Documentation Conventions

4. Create Use case diagram

System Boundary: Representing the WinForms application.
Use Case: "View Inventory" indicating the functionality provided by the application.
Actor: "User" who interacts with the system to view the inventory.

Monday
Start: 900pm End: 9:30pm Activity: Read announcements
Start: 930pm End: 1030 Activity: DQ1 and DQ 2
Start: 1030pm End: 1100pm Activity: Read Book
Tuesday
Start: 900pm End: 9:30pm Activity: Participation post
Start: 930pm End: 1030 Activity: Activity 6
Start: 1030pm End: 1100pm Activity: Read Book
Wednesday
Start: End: Activity: N/A
Start: End: Activity: N/A
Start: End: Activity: N/A
Thursday
Start: 900pm End: 9:30pm Activity: Participation post
Start: 930pm End: 1030 Activity: Activity 6
Start: 1030pm End: 1100pm Activity: Read Book
Friday
Start: 900pm End: 9:30pm Activity: Participation post
Start: 930pm End: 1030 Activity: Milestone
Start: 1030pm End: 1100pm Activity: Read Book
Saturday
Start: 900pm End: 9:30pm Activity: Activity 6
Start: 930pm End: 1030 Activity: Milestone
Start: 1030pm End: 1100pm Activity: Milestone
Sunday
Start: 900pm End: 9:30pm Activity: Activity 6
Start: 930pm End: 1030 Activity: Milestone
Start: 1030pm End: 1100pm Activity: Milestone