

Eric Gathinji

Programming in C# CST-150-0500

Grand Canyon University

13th July 2025

Milestone 5

Github link: <https://github.com/Ericgathinji444/GCU>

video link: <https://youtu.be/Z5aaRohvW8w?si=lgZDFdR-3pf4rqTg>

UPDATED FLOWCHART

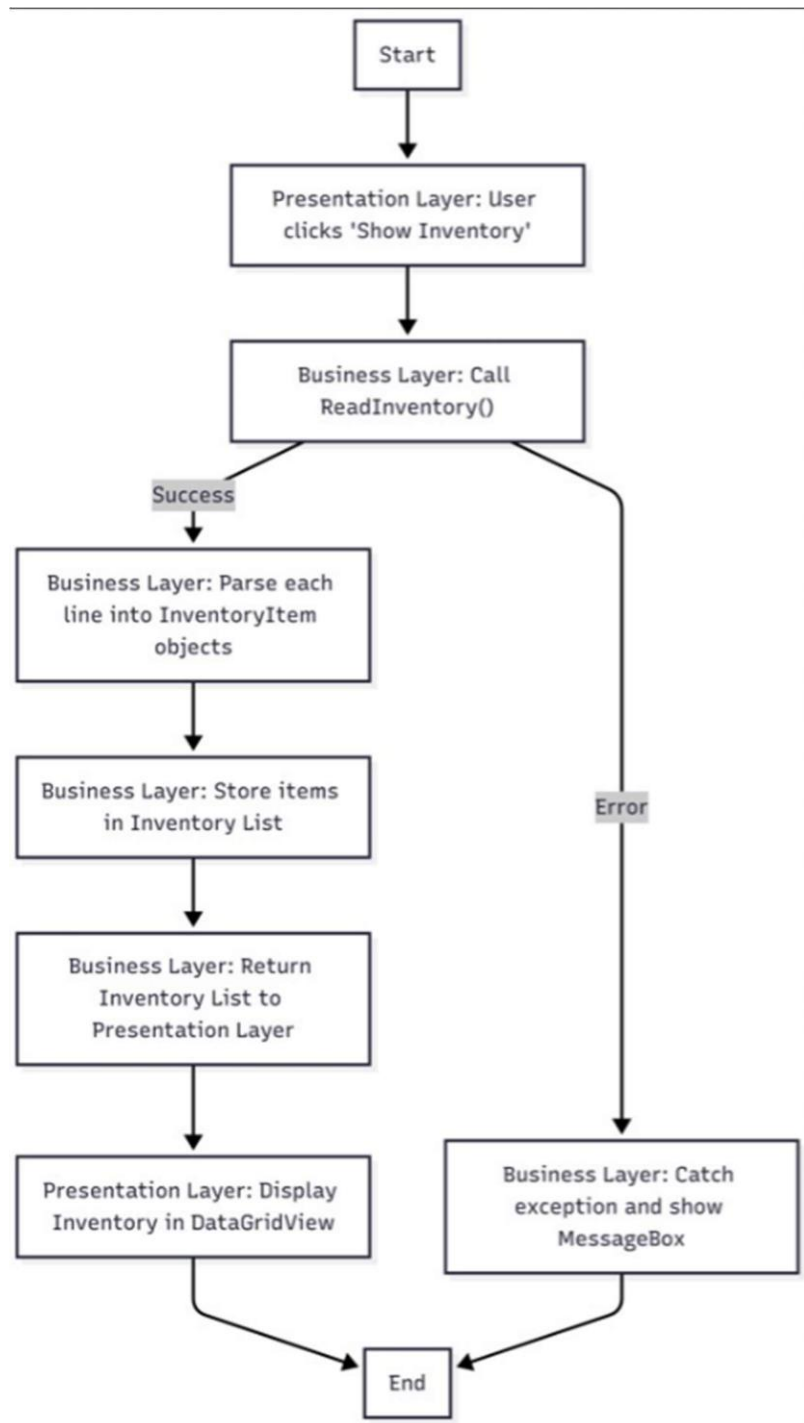


Figure 1:Updated flowchart for milestone 5.

Explanation:

This updated flowchart reflects Milestone 5's refactored logic.

The original milestone had logic directly in the Form1.cs class, including file reading and item parsing.

Now, the flow separates responsibilities:

The Form1 form only triggers the display.

The BusinessLayer.InventoryManager handles file reading and data parsing.

This represents a clean separation of concerns aligned with N-er architecture.

UPDATED WIREFRAME.

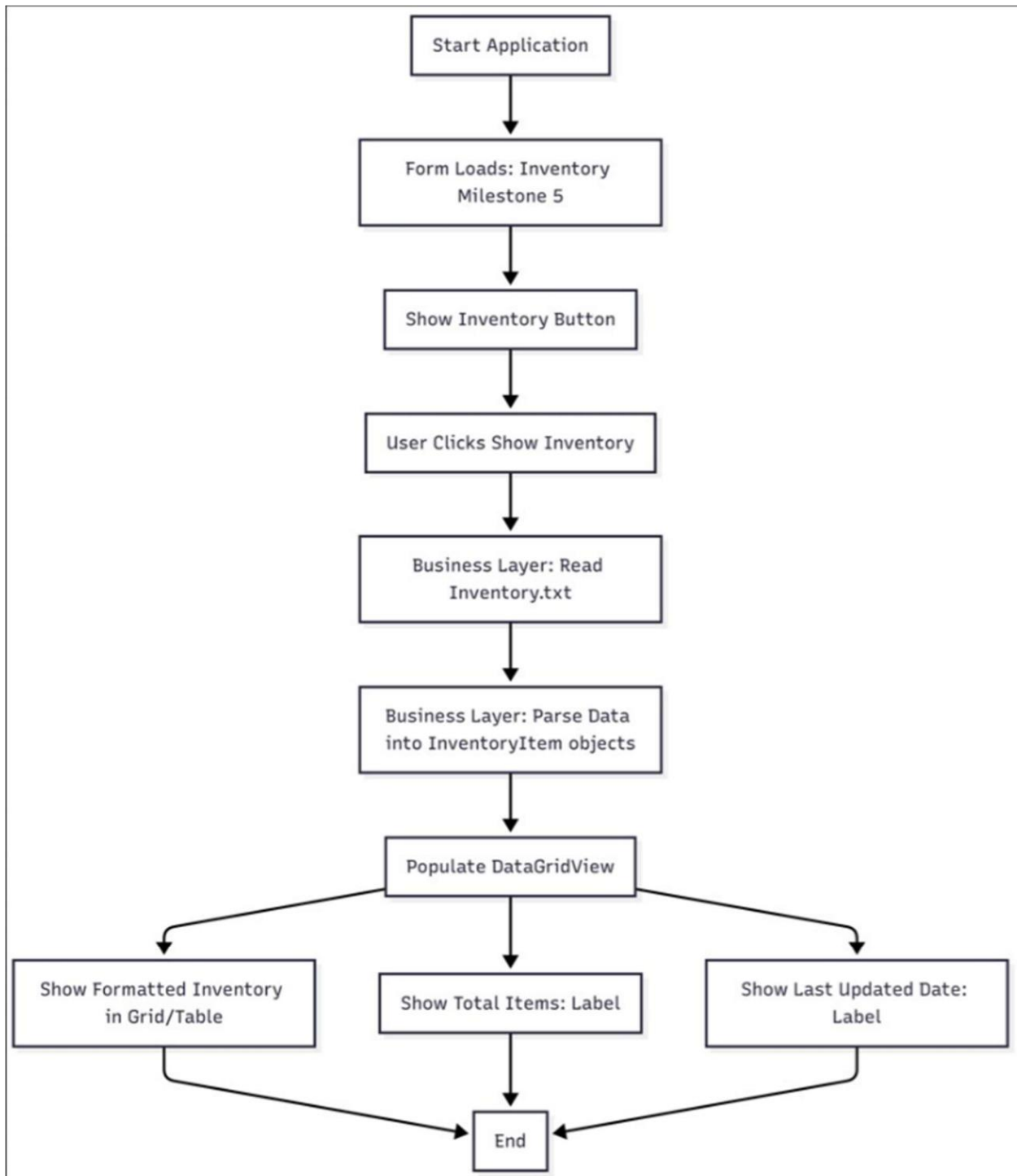


Figure 2:Updated wireframe for milestone 5

Figure 2 Explanation:

The updated wireframe now includes a DataGridView instead of a label for displaying inventory.

This visual change allows:

Proper alignment of columns for each attribute

Easier readability of large inventory sets

A professional user interface

The "Show Inventory" button remains, triggering data retrieval from the business logic.

UML FOR ALL CLASSES

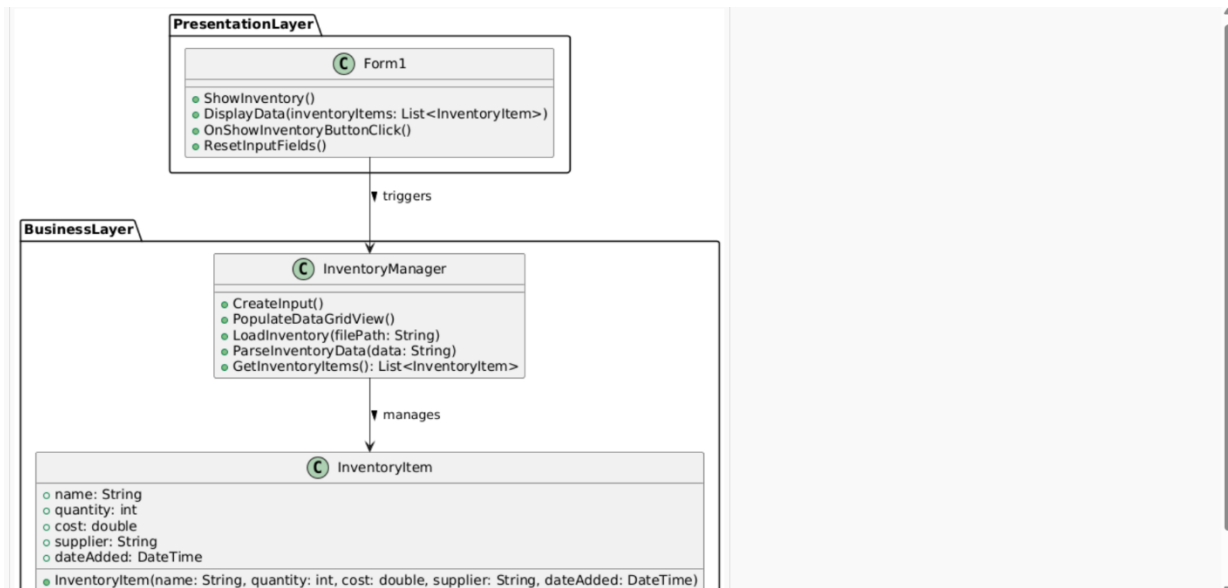


Figure 3: A UML class diagram illustrating how Form 1 and InventoryItem relate to one another in a file processing application.

Explanation: Form1 and InventoryItem are the two classes displayed in the UML diagram. Through its methods and interactions with InventoryItem objects, Form1 manages file selection, reading, and displaying inventory data. InventoryItem is the data structure for each item and contains attributes like Name, Price, and Quantity

UPDATED INVENTORY BEING DISPLAYED.

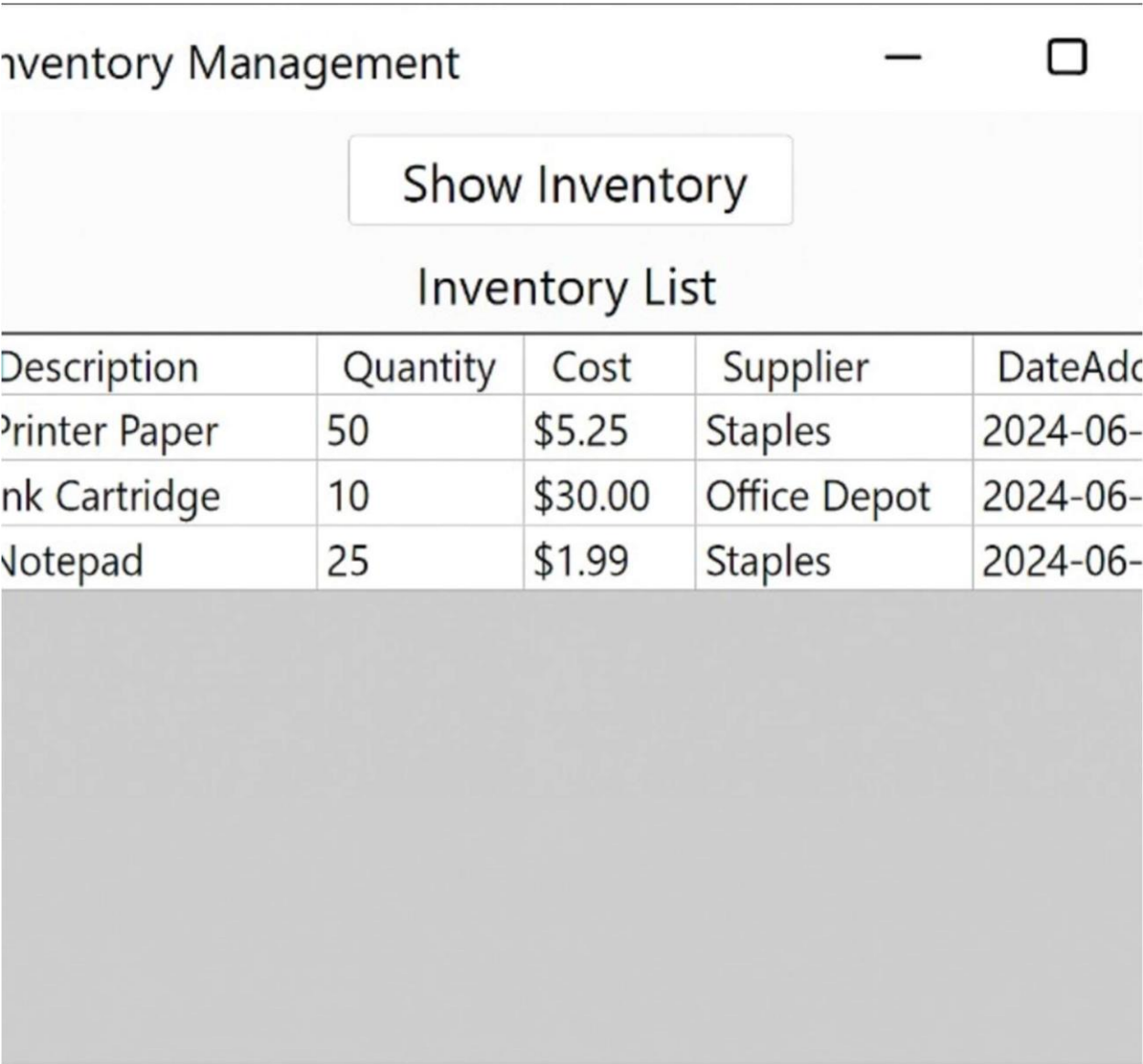


Figure 4:Display of inventory

Explanation:

The DataGridView control now presents all five inventory attributes as column headers:

Name

Quantity

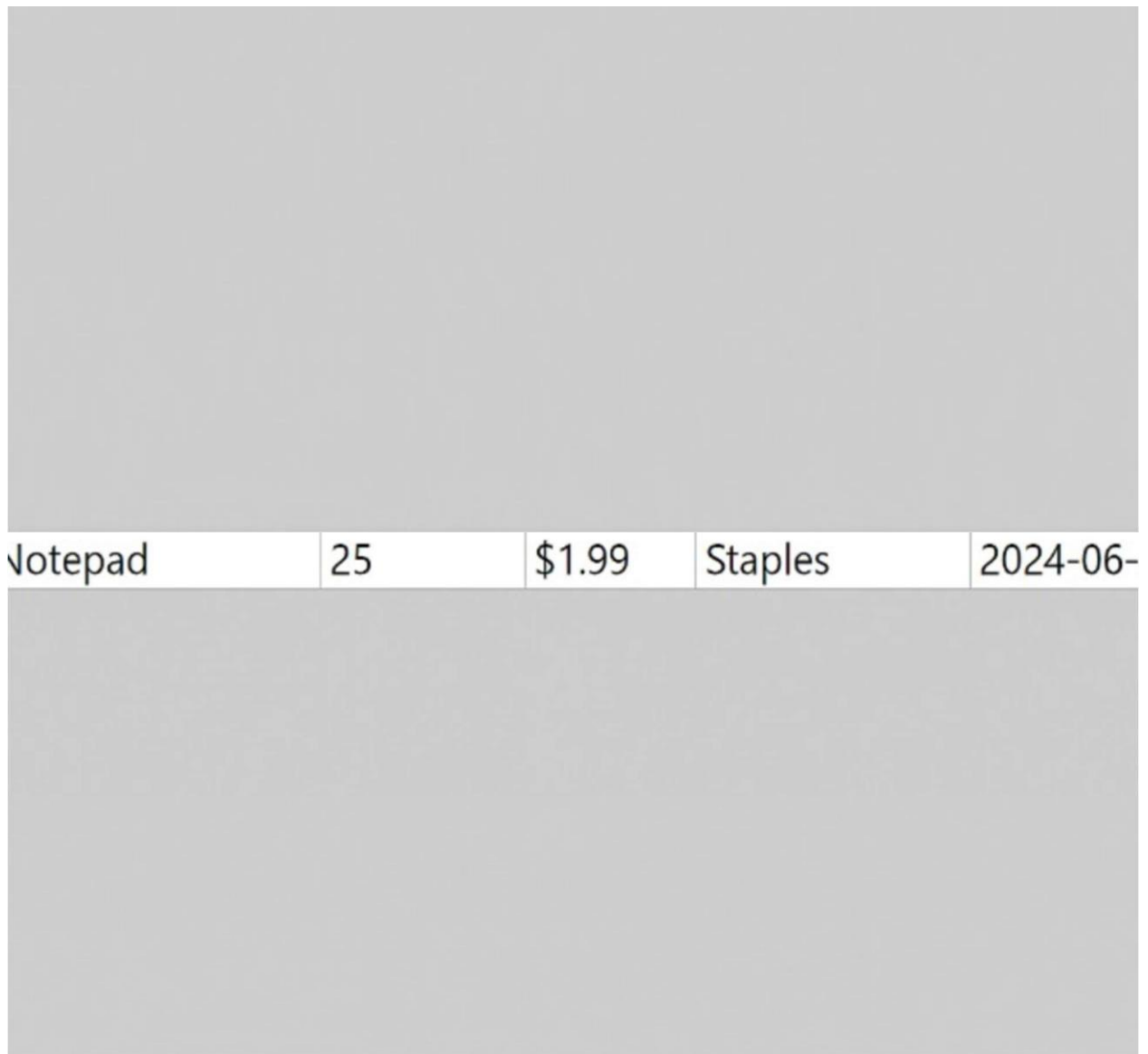
Cost

Supplier

Date Added

Using a grid improves layout, user experience, and aligns with professional data presentation.

SCREENSHOT OF FORM AFTER BEING POPULATED



The screenshot shows a form with a table containing one row of inventory data. The table has five columns. The first column contains the text 'Notepad', the second contains the number '25', the third contains the price '\$1.99', the fourth contains the item name 'Staples', and the fifth contains the date '2024-06-'. The table is set against a light gray background.

Notepad	25	\$1.99	Staples	2024-06-
---------	----	--------	---------	----------

Figure 5: Screenshot of the form after being populated

Explanation:

This shows the form after InventoryManager loads the data from Inventory.txt.

The inventory rows are listed in the grid, demonstrating that:

Data was read from the file

Proper formatting is applied

The n-layer separation is working as intended

Screenshot of the form behind the code

```
using InventoryApp.Business;
using System;
using System.Windows.Forms;

namespace InventoryApp
{
    public partial class MainForm : Form
    {
        private Inventory _inventory;

        public MainForm()
        {
            InitializeComponent();
            _inventory = new Inventory();
            dgvItems.DataSource = _inventory.Items;
        }
    }
}
```

Figure 6:screenshot of Maiform.cs

Explanation

Form1.cs contains no business logic. It delegates:

File operations to InventoryManager

Data structure for InventoryItem

This follows clean architecture, making the UI layer easier to maintain and test.

SCREENSHOT(S) OF CODE FOR ALL CLASSES

```
// CS1-150 Milestone 5 - Business Logic Layer
// Author: Eric Gathinji
// Description: This class manages reading and holding inventory data using N-layer architecture.
```

```
using System;
using System.Collections.Generic;
using System.IO;
```

```
namespace InventoryApp.Business
```

```
{
```

```
    1 reference
```

```
    public class Inventory
```

```
    {
```

```
        2 references
```

```
        public List<InventoryItem> Items { get; private set; }
```

```
        0 references
```

```
        public Inventory()
```

```
        {
```

```
            Items = new List<InventoryItem>();
```

```
            LoadInventoryFromFile();
```

```
        }
```

```
    1 reference
```

```
    private void LoadInventoryFromFile()
```

```
    {
```

```
        string path = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, @"Data\Inventory.txt");
```

```
        try
```

```
        {
```

```
            if (File.Exists(path))
```

```
            {
```

```
                string[] lines = File.ReadAllLines(path);
```

```
                foreach (string line in lines)
```

```
                {
```

```
                    string[] parts = line.Split(',');
```

```
                    if (parts.Length == 5)
```

```
                    {
```

```
                        string desc = parts[0];
```

```
                        int qty = int.Parse(parts[1]);
```

```
                        decimal cost = decimal.Parse(parts[2]);
```

```
                        string category = parts[3];
```

```
                        DateTime date = DateTime.Parse(parts[4]);
```

```
                        Items.Add(new InventoryItem
```

```
                        {
```

```
                            Description = desc,
```

```
                            Quantity = qty,
```

```
        decimal cost = decimal.Parse(parts[2]);
        string category = parts[3];
        DateTime date = DateTime.Parse(parts[4]);

        Items.Add(new InventoryItem
        {
            Description = desc,
            Quantity = qty,
            Cost = cost,
            Category = category,
            DateAdded = date
        });
    }
}
else
{
    throw new FileNotFoundException("Inventory.txt not found at " + path);
}
catch (Exception ex)
{
    throw new Exception("Error loading inventory file", ex);
}
```

(local variable) string[] parts
'parts' is not null here.

Figure 7: Screenshots of the Inventory.cs class explaining the logic behind the fetching of data.

```
// CST-150 Milestone 5 - Business Logic Layer
// Author: Eric Gathinji
// Description: This class manages reading and holding inventory data using N-layer architecture.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace InventoryApp.Business
{
    3 references
    public class InventoryItem
    {
        1 reference
        public string Description { get; set; }
        1 reference
        public int Quantity { get; set; }
        1 reference
        public decimal Cost { get; set; }
        1 reference
        public string Category { get; set; }
    }
}
```

Figure 8: Screenshot of the InventoryItem.cs showing how the code is laid out on display

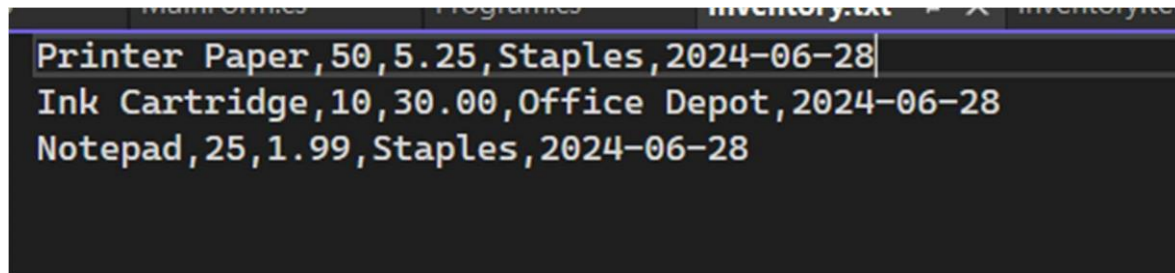
Explanation:

The InventoryItem class is a POCO (Plain Old CLR Object) with properties only — it's responsible for holding inventory values.

The InventoryManager class reads from the file and parses items into the array.

This separation is aligned with business logic layer practices where UI is isolated from core functionality.

SCREENSHOT OF ORIGINAL TEXTFILE



The screenshot shows a text editor window with a dark background. The title bar at the top reads 'Inventory.txt'. The text content is as follows:

Item description	Quantity	Cost	Supplier	Date added
Printer Paper	50	5.25	Staples	2024-06-28
Ink Cartridge	10	30.00	Office Depot	2024-06-28
Notepad	25	1.99	Staples	2024-06-28

Screenshot of the original text file

Explanation:

This is the source data (Inventory.txt) the application reads.

Each line is comma-separated with:

Item description

Quantity

Cost

Supplier

Date added

This file drives the entire application logic, simulating real-world data import workflows.

ADD ONS

Milestone 5 research

Follow-up questions

Milestone 5 Follow-up questions

Platform used to play the game- YouTube

Name of the Game: Escape Room

Category: Interactive

Gameplay description: Escape Room: The YouTube Adventure is an interactive puzzle game in which players solve riddles and escape virtual rooms by selecting options by clicking on video links. The experience blends rational problem-solving with storytelling. Although it's simple to understand, people accustomed to faster-paced games may find it monotonous and boring.

How can it be improved?

Including alternative storylines would increase the excitement and replay value.

Describe the color scheme

To create an interesting escape room environment, the game uses a color scheme that includes shades of gray, blue, and dim lighting.

How could it be improved?

Smoother and better transitions between options or more colorful visual effects could improve the experience.

The Key takeaway for the milestone project

Users may find a project more interesting if interactive features and choice-based navigation are included.

Bug report: None

Follow-up questions

What was challenging?

Refactoring the logic to provide a distinct division of duties between layers was the difficult part of Milestone 5.

What did you learn? I gained knowledge on how to successfully apply N-layer architecture to improve apps' maintainability.

How would you improve the project?

In order to guarantee that every component operates accurately and consistently, I would enhance the project by include thorough unit tests

How can you use what you learned on the job?

I can develop better structured and reliable software solutions at work by utilizing these testing techniques and architectural concepts.

ADD ON

Naming conventions- using the standard naming (like camelCase) makes the code easier to read.

Properties – using PascalCase. E.g Quantity Code

structure- using comments for clarification.

Consistency- The formatting is consistent throughout my project.

Computer specs- The OS: Windows 11

RAM: 8GB

Processor: Intel Core i5

Tutor discovery 5

I read the instructions given by my instructor, Mark Smithers.

Weekly Activity

Start Tuesday 8th July 2025 End

9:00 am 4:00 pm Milestone 5

Start Wednesday 9th July 2025 End

11:00 am 2:00 pm Milestone 5

Start Thursday 10th July, 2025 End

9:00 am 2:00 pm Milestone 5