Eric Gathinji

Programming in C# CST-150-0500

Grand Canyon University

24th JUNE 2025

Milestone 1

Github link: https://github.com/Ericgathinji444/inventoryApp

Video link: https://youtu.be/zzzuTH2Czxc?si=xNaNrKHcrLW2jTmt

Data Flow Chart

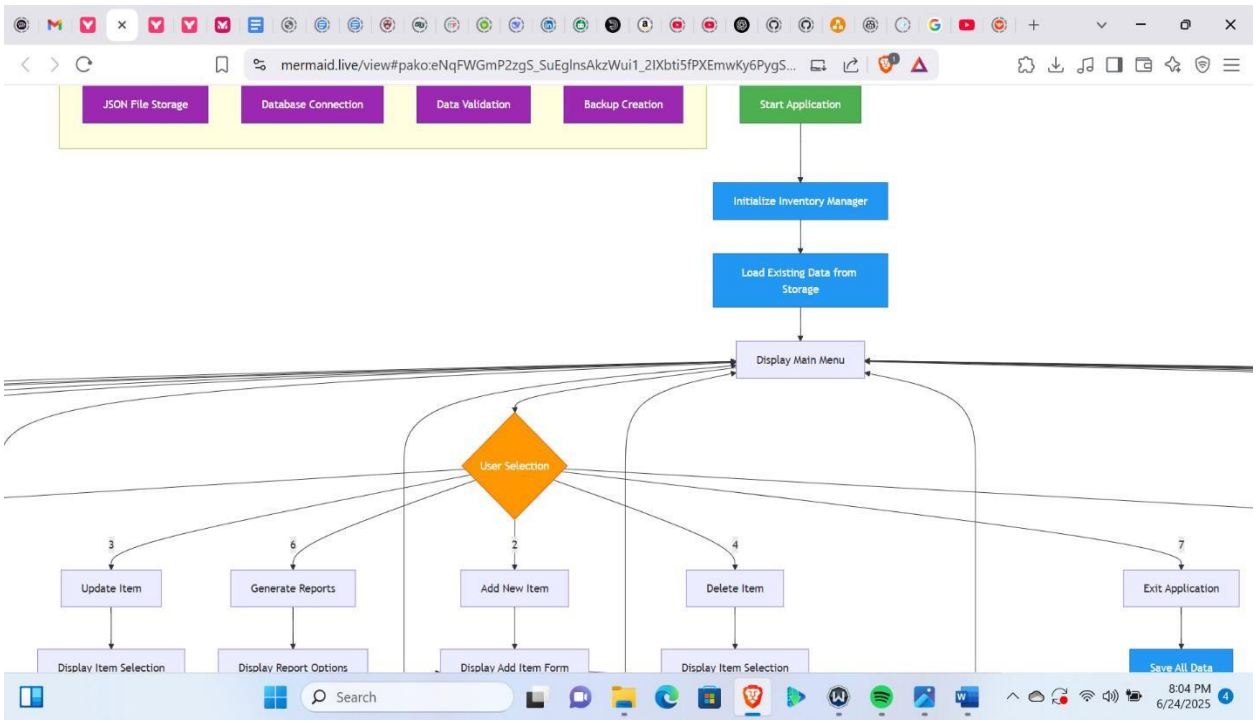


Figure 1

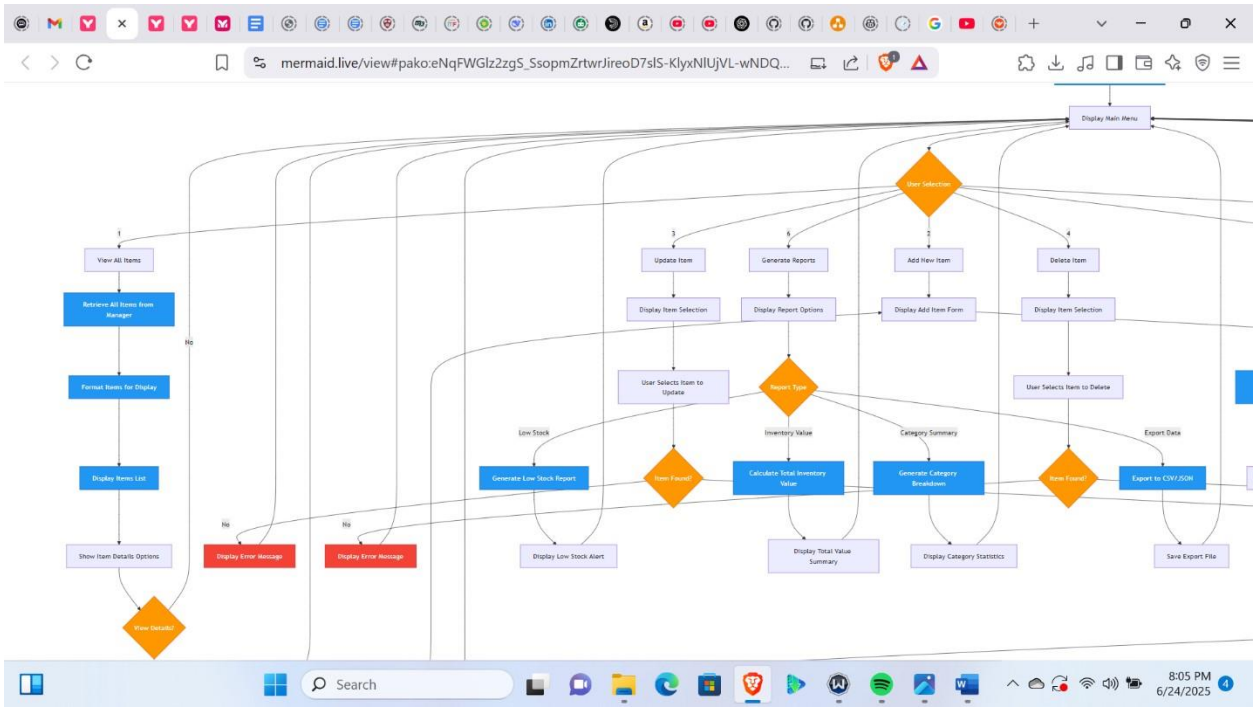


Figure 1.2

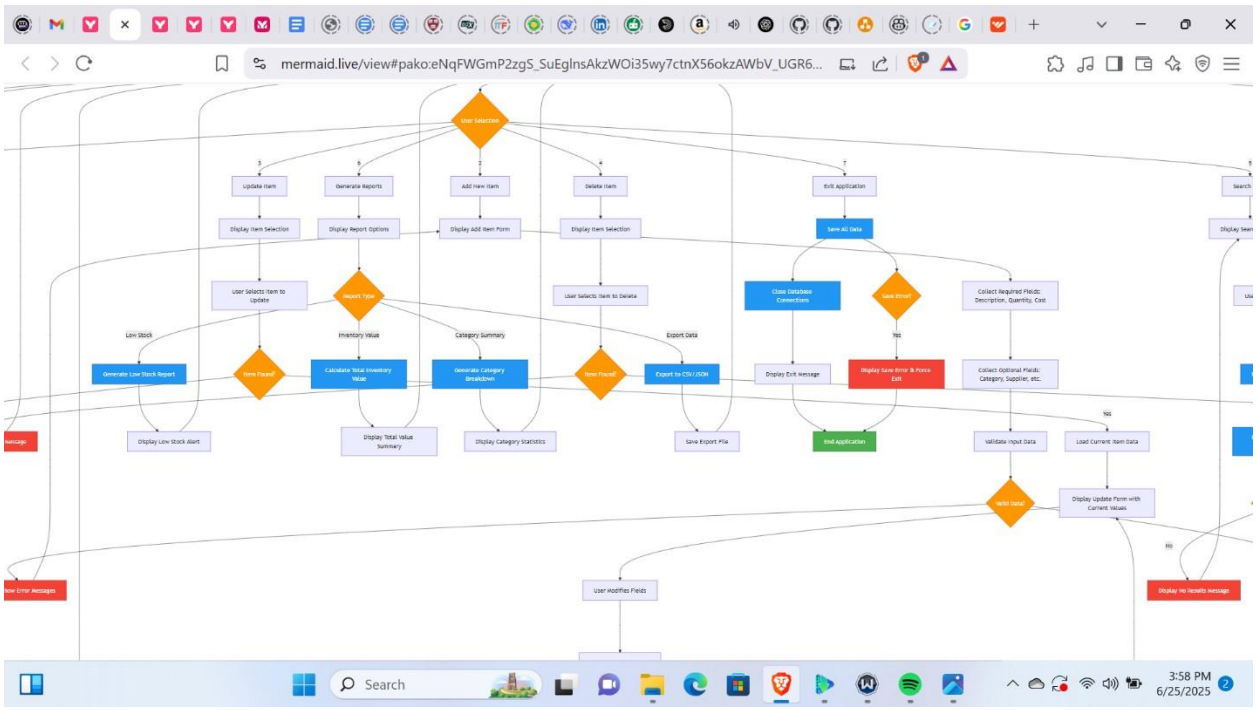


Figure 1.3

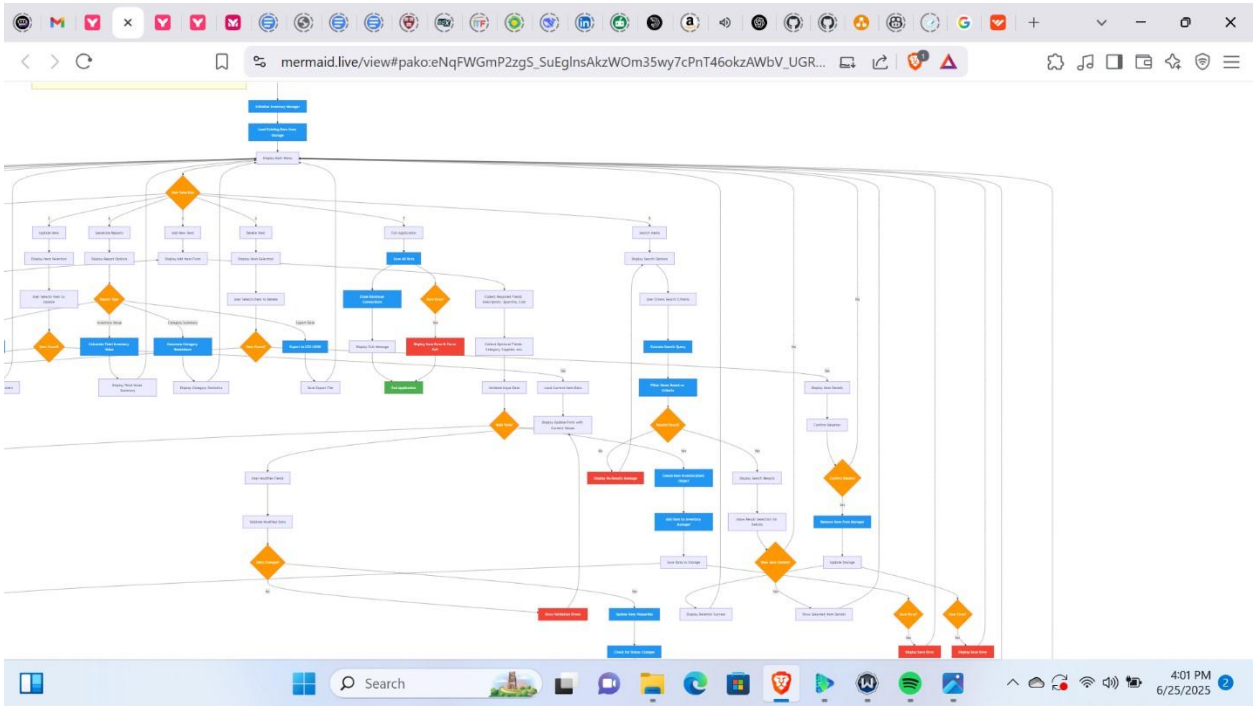


Figure 1.4

Figure 1, 1.1 ,1.2 ,1.3and 1.4:Flowchart defining the data flow through the application

In Figure 1, 1.1, 1.2 1.3 1nd 1.4 there's an illustration of the flow of data through the inventory application, showing how user inputs, processes and user output. The application begins and initializes hard-coded data at the start of the flowchart, after which important elements include the inventory table. Users can perform activities such as removing an item once the dashboard is visible.

UML CLASS DIAGRAM

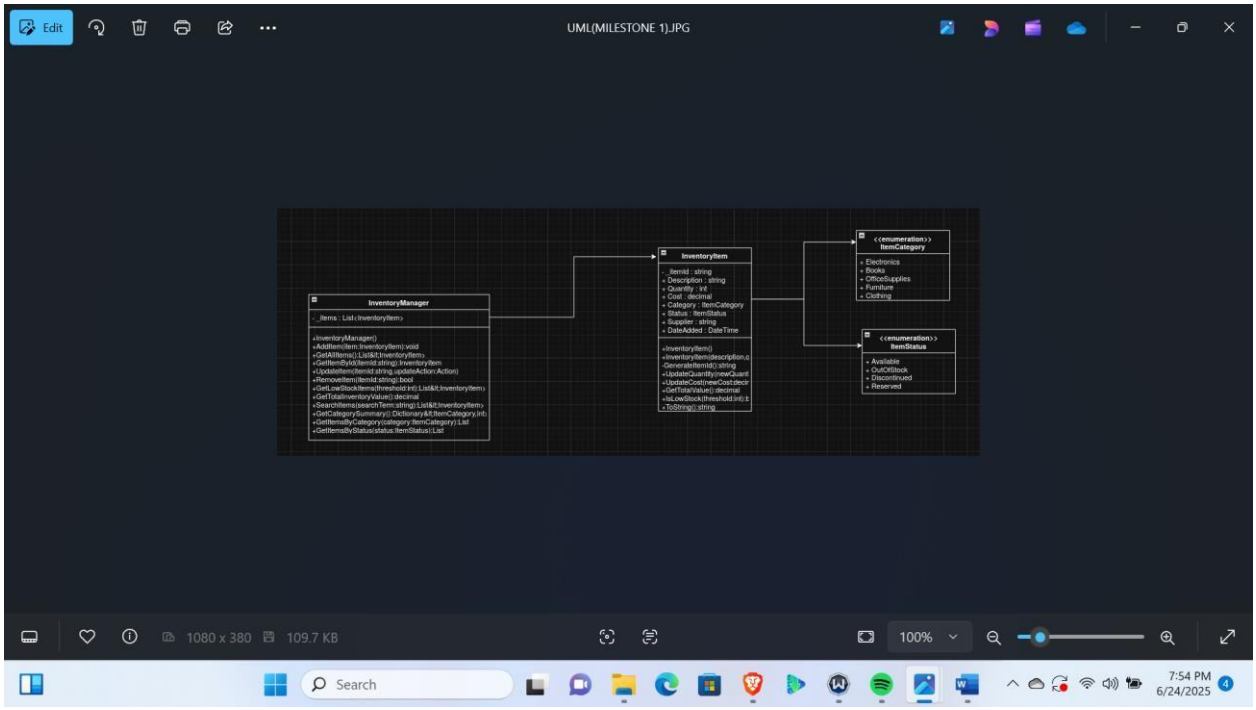


Figure 2: This UML class diagram shows an inventory management system. With classes for inventory manager, inventory item and enumerations for item categories and item statuses

Initial Data Model

```
public class InventoryItem
{
    [Required(ErrorMessage = "Description is required")]
    [StringLength(200, ErrorMessage = "Description cannot exceed 200 characters")]
    public string Description { get; set; }

    [Required(ErrorMessage = "Quantity is required")]
    [Range(0, int.MaxValue, ErrorMessage = "Quantity must be non-negative")]
    public int Quantity { get; set; }

    [Required(ErrorMessage = "Cost is required")]
    [Range(0.01, double.MaxValue, ErrorMessage = "Cost must be greater than 0")]
    public decimal Cost { get; set; }

    // Additional properties
    public string ItemId { get; set; }

    public ItemCategory Category { get; set; } = ItemCategory.OfficeSupplies;

    public ItemStatus Status { get; set; } = ItemStatus.Available;

    [StringLength(100, ErrorMessage = "Supplier name cannot exceed 100 characters")]
    public string Supplier { get; set; }

    public DateTime DateAdded { get; set; } = DateTime.Now;

    // Constructor
    public InventoryItem()
    {
        ItemId = GenerateItemId();
    }

    public InventoryItem(string description, int quantity, decimal cost)
    {
        Description = description;
        Quantity = quantity;
        Cost = cost;
        ItemId = GenerateItemId();
    }

    // Methods
    private string GenerateItemId()
    {
        return $"INV_{Guid.NewGuid().ToString("N")[..8].ToUpper()}";
    }

    public void UpdateQuantity(int newQuantity)
    {
        if (newQuantity < 0)
            throw new ArgumentException("Quantity cannot be negative");

        Quantity = newQuantity;

        if (newQuantity == 0)
            Status = ItemStatus.OutOfStock;
        else if (Status == ItemStatus.OutOfStock && newQuantity > 0)
            Status = ItemStatus.Available;
    }

    public void UpdateCost(decimal newCost)
    {
        if (newCost < 0)
            throw new ArgumentException("Cost cannot be negative");

        Cost = newCost;
    }

    public decimal GetTotalValue()
    {
        return Quantity * Cost;
    }

    public bool IsLowStock(int threshold = 5)
    {
        return Quantity <= threshold && Status == ItemStatus.Available;
    }

    public override string ToString()
    {
        return $"{ItemId} - {Description} (Qty: {Quantity}, Cost: {Cost:C})";
    }
}
```

Figure 3:Data Model for the application

The data model has the following: itemId, quantity, cost and, description. These explain each inventory item.

Low-Fidelity Wireframes / Prototypes

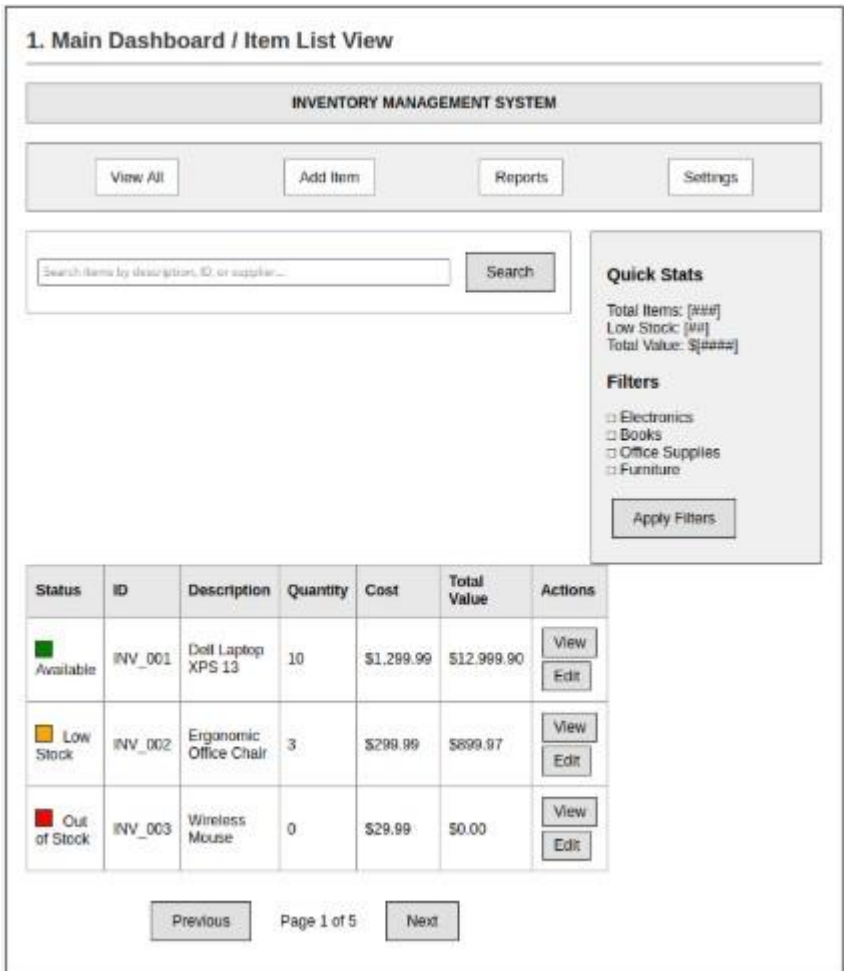


Figure 4:Main Dashboard / Item List View

- This wireframe represents the primary entry point in our application flowchart, corresponding to the "Display Main Menu" and "View All Items" flow paths.
- The low-fidelity design emphasizes functional layout over visual polish, using simple boxes and basic typography to focus on information hierarchy and user workflow.

-
- The status indicators (green, orange, red squares) provide immediate visual feedback about inventory health, while the tabular layout efficiently displays all required properties from our data model (ItemId, Description, Quantity, Cost, etc.).
  The sidebar Quick Stats and Filters sections demonstrate how the wireframe anticipates user needs for summary information and data filtering, directly supporting the search and reporting flows outlined in our flowchart.

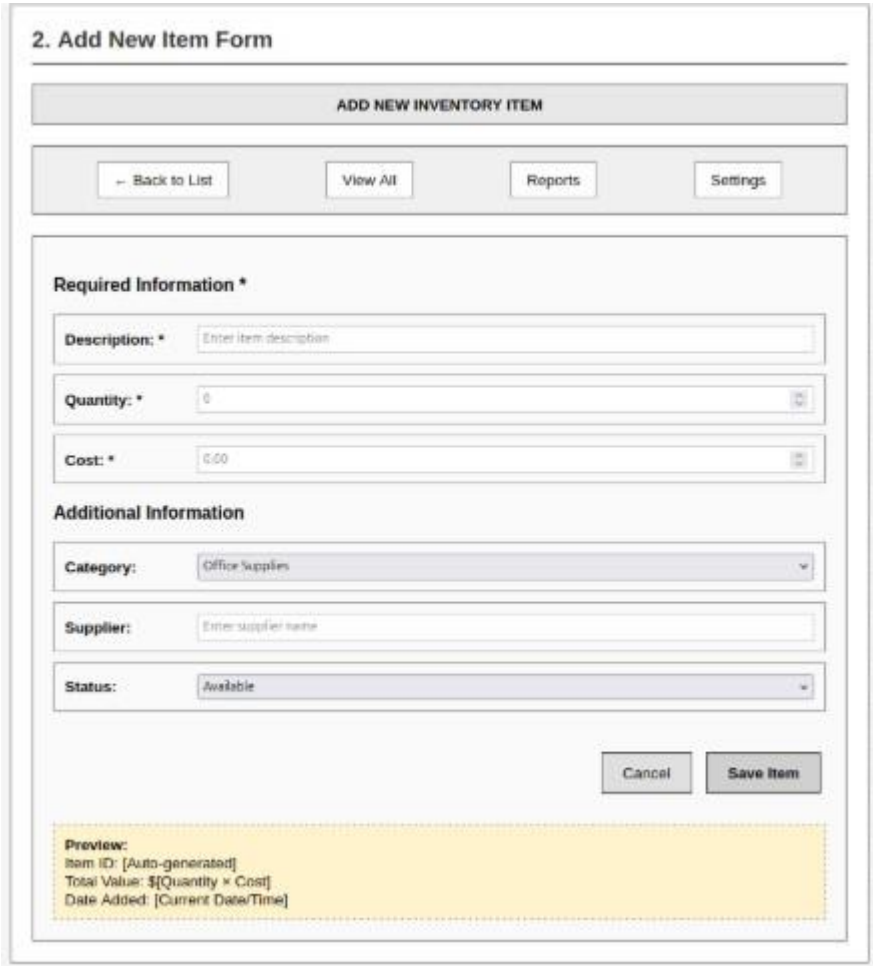
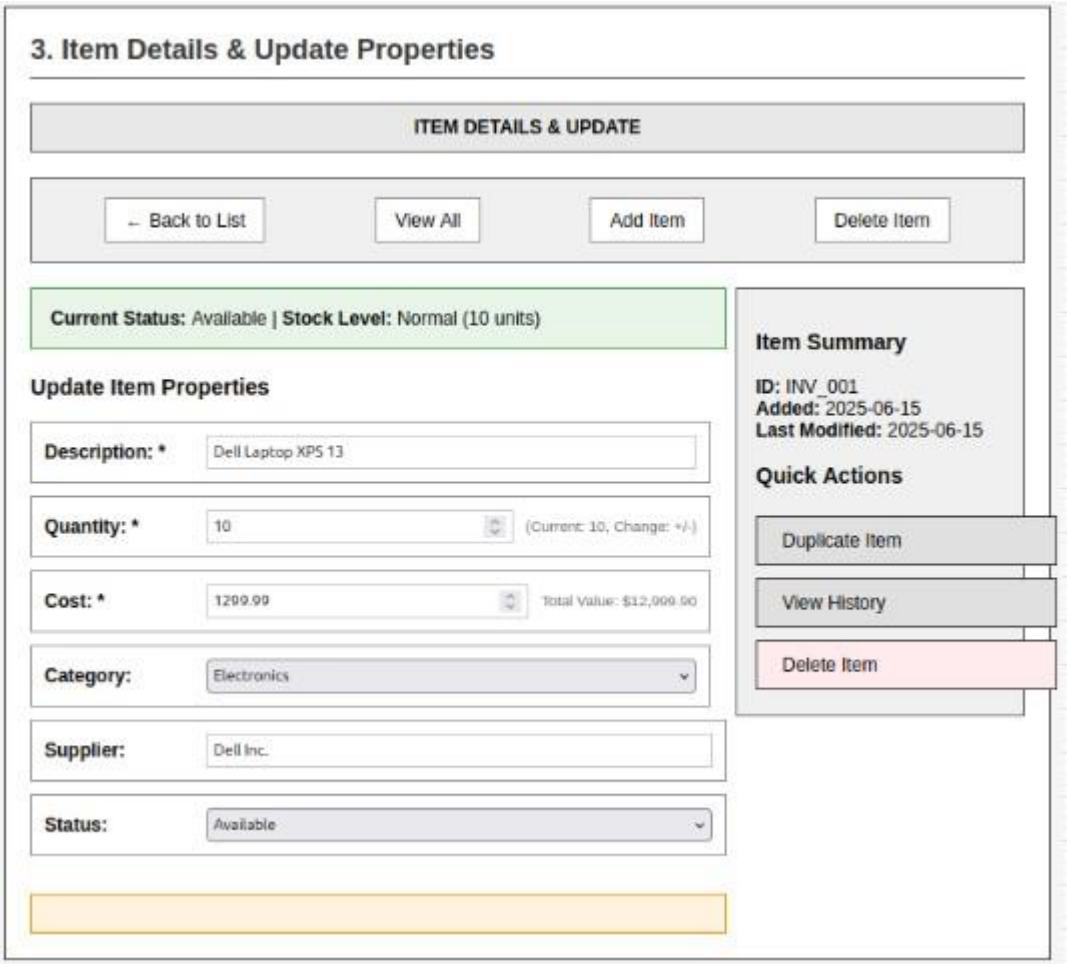


Figure 5: A form to accept input for a new item

- This form wireframe directly implements the "Add New Item" flow from our flowchart, showing the clear separation between required fields (Description, Quantity, Cost) and optional additional properties.

-
-
- The low-fidelity approach uses simple input fields and dropdown menus to maintain focus on data collection workflow rather than visual design elements.
  The preview section at the bottom demonstrates real-time feedback to users, showing how their input will be processed before final submission, which aligns with our flowchart's validation steps.
  The form structure directly maps to our InventoryItem class properties, ensuring consistency between the data model and user interface design.
-



-
-

Figure 6:A form to show all the properties of a single item and update its properties

- This wireframe combines both the "View Item Details" and "Update Item" flows from our application flowchart, creating an efficient single-screen experience for item management.
  The low-fidelity design clearly separates read-only summary information (Item Summary sidebar) from editable fields (main form area), helping users understand what can be modified.
  The current status display and change tracking features (showing current vs. new values) provide important feedback during the update process, supporting the validation and confirmation steps outlined in our flowchart.
- The Quick Actions sidebar anticipates common user workflows like duplication and deletion, demonstrating how wireframes can identify additional features beyond the core requirements while maintaining the simple, functional aesthetic of low-fidelity prototyping.

What is an inventory?

Inventory in C# refers to a collection used to store and manage a list of products mostly is applications such as games. Arrays, lists (list<T>), (Dictionary<TKey, TValue>), and custom classes that provide attributes like ItemName, Quantity, Price, and ID can all be used to implement it. An inventory system allows for adding, removing, searching, and updating items. An inventory could also be used to maintain stock levels and product descriptions in a store application or recover items in a game. Developers can create flexible and adaptable systems that work well with databases and UI by employing object-oriented concepts and containing inventory logic in classes.

How is it useful

In C# an inventory is helpful because it facilitates the efficient and orderly organization of item data. An inventory system enables developers to monitor important information such as item names, amounts, and price availability in any kind of application, including games, retail systems, and warehouse management systems. Key functions include adding new products, maintaining levels of stock, and creating reports are all made possible by it. This increases accuracy and lowers errors caused by humans. It also improves the overall usability of the application. C# developers may design flexible and scalable inventory systems that adapt to business requirements by utilizing object-oriented programming and collections.

-
-

What is the most important information stored in an inventory? why?

The most important details kept in an inventory are the item name, availability status or ID quantity. These have a direct effect on how the system maintains and monitors stock. Each product is individually identified by its item name or ID ensuring handling and ensuring accuracy. Quantity shows how much of each item is in stock. In order to assist users in making well-informed decisions availability status shows whether an item is in stock or out of stock. An

inventory system cannot efficiently support company operations or avoid problems like overstocking without this essential data.

What was challenging

The most challenging part of creating this milestone 1 was designing the flowchart.It was difficult to link the UI design with the flowchart like submission and feedback loops while maintaining visual clarity.

ADD ONS:

Programming Conventions

Naming conventions: using ALL_CAPS with underscores

For example; MAX_ITEMS

Properties- Using Pascal Code

For example: Quantity

Code structure: using comments for clarification and clear identification.

Consistency: The formatting is consistent throughout the project**.**

Computer specs

The OS – Windows 11

RAM: 8GB

Processor: Intel core i5

Tutor Discovery

I reached out to my instructor Mark Smithers after being informed my work was unsatisfactory. This comment helped me keep the flowcharts and everything else as required in the instructions.

Weekly activity:

Start Monday: June 23, 2025: 8:00 am End: 12:00 pm Milestone 1

Start Tuesday: June 10, 2025: 8:00 am End: 12:00 pm Milestone 1

Start Wednesday: June 11, 2025:8:00 am End:12:00 pm Activity 1

Milestone 1 research

List 3 inventory applications:

Zoho Inventory

Square for Retail

Fishbowl Inventory

List 5 features of the inventory application

Input items- Stock tracking

Modify or update the existing inventory Remove

or delete an item from the inventory.

Show all the items in the inventory.

Helps users to quickly find items and speed up data entry(Desai,2025).

Reference

Desai, A. (2025). *Enhancing Inventory Management with Progressive Web Applications (PWAs): A Scalable Solution for Small and Large Enterprises*.

Bug report; None **Follow-up**

questions

What was challenging?

Designing the flowchart

What did you learn?

How to create a user-friendly system

How would you improve on the project?

Use a better error-handling system.

How can you use what you learned on the job?

To design a clear and well-structured code.