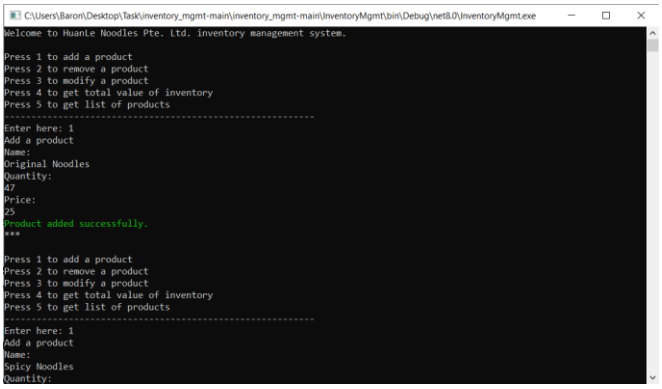
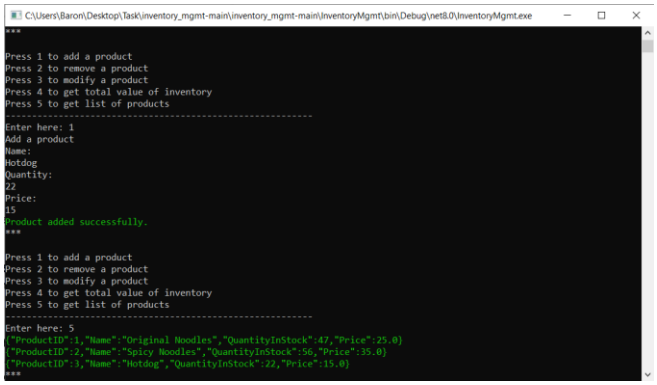
HuanLe Noodles Pte. Ltd. inventory management system

## I. Functionality

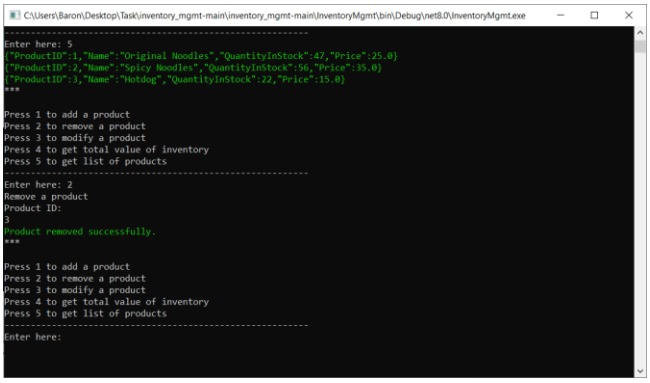
The **Inventory Management System** allows users to perform various operations to manage products, such as adding, removing, updating, checking total value, and listing all products in the inventory.

When a user selects **Option 1 (Add a Product)**, they are prompted to enter the product's name, quantity, and price. The system validates these inputs to ensure the name is not empty, and both the quantity and price are non-negative. If all validations pass, the product is added successfully, displaying a confirmation message; otherwise, relevant error messages appear.

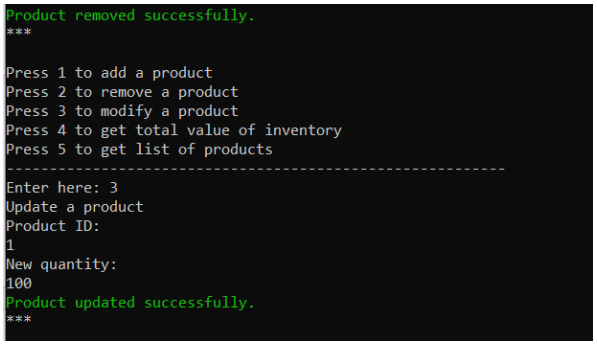


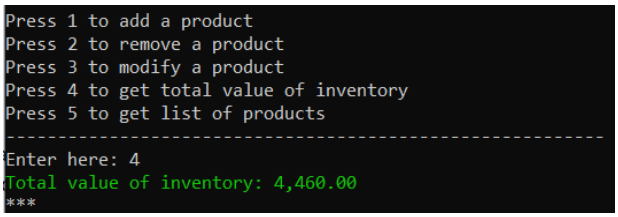

For **Option 2 (Remove a Product)**, the user enters a Product ID to delete a specific product. If the product exists, it is removed, and a success message is shown; if not, an error message notifies the user that the product was not found.



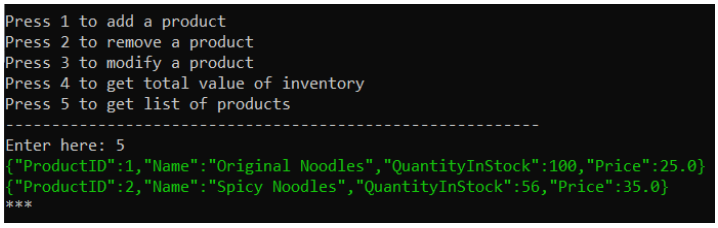
In **Option 3 (Modify a Product)**, the user enters the Product ID and a new quantity to update the inventory. The system checks if the product exists and if the quantity is non-negative. If valid, the product's quantity is updated, and a confirmation message is displayed; otherwise, error messages indicate whether the product ID was invalid, or the quantity was negative.




**Option 4 (Get Total Value of Inventory)** calculates the total value of all products by multiplying each product's quantity by its price and summing the results. This total is then displayed; if no products are in inventory, the total value is shown as zero.




Finally, **Option 5 (List Products)** displays all products in JSON format, including each product's ID, name, quantity, and price. If no products are available, a message indicates that the inventory is empty. This comprehensive functionality enables efficient inventory management while providing immediate feedback for errors and successful actions.



## II. Manual Test Cases

For Manual Test cases, I used the test plan given which outlines the approaches for HuanLe Noodles Pte. Ltd. Inventory Management System manual testing. The objective of this test plan is to ensure that all functionalities work as expected. Testing will cover various functions, including product addition, removal, modification, inventory value, and display of products.

In the excel sheet, I created 3 test plan sheets specifically:

- Add Product Test Plan
- Remove Product Test Plan
- Modify Product Test Plan

**Scope**

- Add Product - Functionality for adding products to inventory, including validation of input fields for product name, quantity, and price.
- Modify Product - Functionality for updating product quantity based on Product ID.
- Get Total Inventory Value - Calculates and displays the total value of inventory.
- Get List of Products - Displays all products in inventory with relevant details in JSON format.
- Remove Product - Functionality to remove products from inventory based on Product ID.
- Validation Scenarios - Error handling for invalid inputs, including non-existent Product IDs, special characters, and negative numbers.

## III. Automated Test Cases

For Automated Test Cases, I create new test methods based on the existing test methods and based on my output in manual test plan. The testing strategy validates the functionality of HuanLe Noodles Pte. Ltd.'s Inventory Management System through automated unit tests. The objective of this automated test case is to ensure that each function within the system behaves as expected.

### a. InventoryManagerTest.cs

The InventoryManagerTest.cs file contains a suite of unit tests for the InventoryManager class within the Inventory Management System. These tests validate the core functionalities of inventory operations, including adding, removing, updating products, and calculating total inventory value. The tests are structured to

capture and verify console outputs, ensuring that the program's output messages align with expected results.

Here’s an overview of each test method I created:

1. **TestRemoveProduct**

- Purpose: Tests removing a product by ID, including cases for both existing and non-existent products.
- Functionality: Adds and removes a product, then verifies successful removal. It also attempts to remove a non-existent product, ensuring the appropriate error message ("Product not found") is displayed.

2. **TestAddProductQuantity**

- Purpose: Ensures quantity validation during product updates.
- Functionality: Checks for errors when updating a product with a negative quantity, alphabetic, or special characters as input. For each case, the console output should confirm invalid input.

3. **TestGetTotalValueEmptyInventory**

- Purpose: Validates the total value calculation when no products are present.
- Functionality: Attempts to get the total inventory value in an empty inventory, confirming that it displays "Total value of inventory: 0".

4. **TestUpdateProductNonExistentID**

- Purpose: Ensures appropriate handling when attempting to update a non-existent product.
- Functionality: Tries to update a product with an invalid ID and verifies that the console output reflects "Product not found."

5. **CalculateTotalValueMultipleProducts**

- Purpose: Confirms the correct calculation of total value for multiple products.
- Functionality: Adds several products with varying quantities and prices, retrieves the total value, and checks for the expected sum (95.0) in the console output.

b. ProductTest.cs

The ProductTest.cs file contains unit tests for validating the properties of the Product class within the Inventory Management System. Each test method verifies different scenarios for creating a Product instance and checks if the values of attributes (like name, quantity, and price) adhere to defined validation rules.

Here's an overview of each test method I created:

**1. TestAddProductEmptyName**

Purpose: Ensures that a product with an empty name is invalid.

Functionality: Creates a product with an empty Name attribute. The validation results confirm that a product name cannot be empty, marking the product as invalid.

**2. TestAddProductZeroQuantity**

Purpose: Validates that a product with zero quantity is still valid.

Functionality: Creates a product with QuantityInStock set to zero. Since zero is an acceptable boundary value, the product is expected to be valid according to business rules.

**3. TestAddProductMaxQuantity**

Purpose: Confirms that a product with the maximum integer quantity is valid.

Functionality: Creates a product with QuantityInStock set to int.MaxValue. The validator confirms that the maximum allowable integer is acceptable, marking the product as valid.

**4. TestAddProductHighPrice**

Purpose: Verifies that a product with an extremely high price within the allowable decimal range is valid.

Functionality: Creates a product with a price near the upper limit for decimal values. The test confirms that the system accepts high prices within this range, and the product is considered valid.

**5. TestAddProductWhitespaceName**

Purpose: Ensures that a product with a whitespace-only name is invalid.

Functionality: Creates a product with a name consisting only of whitespace. Validation results confirm that names cannot be only whitespace, marking the product as invalid.

## 6. TestAddProductBoundaryQuantityAndPrice

Purpose: Confirms that a product with both minimum valid values for quantity and price is valid.

Functionality: Creates a product with QuantityInStock set to zero and Price set to zero. Since both values meet minimum requirements, the test expects the product to be valid.

## Test Explorer