Final Project	
Canteen Inventory Management System	
Course Code: CPE 201L	**Program:** BS CPE
Course Title: Data Structure and Algorithm	**Date Performed:** November 8, 2025
Section: CPE 2-A	**Date Submitted:** November 8, 2025
Leader: Palmes, Lewis Clark L. **Members:** Disomnong, Jalilah M. Nerio, Hannah Grace A.	**Instructor:** Engr. Maria Rizette Sayo

1. Objective(s):

- To design an efficient inventory system that organizes and manages product data using appropriate data structures such as stacks, queues, and linked lists.
- To demonstrate how different data structures can improve data handling, retrieval speed, and overall system performance.
- To apply stack, queue, and linked list operations in real-world inventory processes like adding, removing, and tracking items.
- To enhance problem-solving and programming skills by integrating multiple data structures into a functional and user-friendly inventory system.

2. Intended Learning Outcomes (ILOs):

- To understand the practical applications of stacks, queues, and linked lists in managing and organizing inventory data.
- To develop the ability to choose and implement the most suitable data structure for different system functions.
- To enhance logical thinking and coding skills through hands-on implementation of core data structure concepts.
- To gain experience in designing a structured and efficient program that solves real-world inventory management problems.

3. Discussion:

This project was developed to showcase how fundamental data structures like stack, queue, and linked list can be effectively used to manage product and transaction data in an organized and efficient way. These structures were chosen for their unique properties and advantages, each contributing to different aspects of the system's performance and functionality.

The stack was applied in the transaction module, specifically for the undo transaction feature. This allows users to revert the most recent operation in case of mistakes or changes in input. The stack operates under the Last In, First Out (LIFO) principle, which means the last data entered is the first to be removed (GeeksforGeeks, 2024). In the context of the system, this concept ensures that only the most recent transaction can be undone, providing both accuracy and control. This mirrors real-world applications of stacks in text editors and transaction systems where the latest action must be easily reversible.

The linked list was used as the foundation for managing product information. Each product is connected through a unique product ID, allowing seamless navigation and updates between nodes. The flexibility of a linked list makes it ideal for managing dynamic data, as it allows insertion and deletion of records without needing to reorganize the entire dataset (GeeksforGeeks, 2024). This results in better memory utilization and faster data manipulation compared to static arrays. The use of a linked list also ensures that data remains connected and easy to traverse, which is essential for searching and updating inventory details efficiently.

The queue was implemented in the report generation module, where it follows the First In, First Out (FIFO) concept. In this section, when the number of records reaches ten, the oldest record is automatically removed to make room for new entries. This approach helps maintain a manageable and updated report while preventing overflow or excessive data storage. The FIFO logic ensures fairness and maintains the correct order of transactions, just as queues are used in operating systems, printers, and customer service systems (GeeksforGeeks, 2024).

Integrating these three data structures into a single system highlighted how each structure plays a vital role in solving different types of problems. The stack handles recent actions, the queue manages orderly data flow, and the linked list efficiently connects and organizes information. This combination allowed the system to perform faster and more reliably while improving data organization.

Through this project, it became evident that choosing the right data structure greatly influences the overall system design, memory management, and speed of operations. Beyond just coding, the project emphasized the importance of algorithmic thinking and understanding how data moves within a system. By applying these principles, the developed Inventory System not only performs its intended functions effectively but also demonstrates how theoretical concepts in data structures can be directly applied to real-world software development.

4. Materials and Equipments:

Hardware:

- Links
- Computer Software:
- Python Programming Language
- PyCharm
- GitHub

5. Procedure:

1. **System Planning and Design**

   The project started with planning the overall structure and identifying how data structures would be applied in different parts of the system. The system was divided into four main components: Dashboard, Product, Transaction, and Report. Each part was mapped to a specific data structure, the linked list for products, the stack for transactions, and the queue for reports. A simple flowchart and data flow diagram were created to visualize how data would move through the system.

2. **Setting Up the Development Environment**

   The programming environment was prepared by choosing a language that supports dynamic data structure implementation which is python language. Basic libraries and classes were set up for handling linked lists, stacks, and queues. The system's input and output structure were also defined to allow users to add, edit, delete, and view data easily.

3. **Implementing the Linked List for Product**

   The linked list was created to store and manage product details such as product ID, name, quantity, and price. Each node represented one product and was connected through pointers or links. Functions were implemented to perform operations like addProduct(), updateProduct(), deleteProduct(), and displayProducts(), ensuring

efficient insertion and deletion without the need to shift data, unlike arrays.

4. **Implementing the Stack for Transaction**

The stack data structure was developed to handle transaction activities. Each time a user performed a transaction (such as adding or removing an item), the details were pushed onto the stack. This design allowed the undo function to pop the most recent transaction when needed, following the Last In, First Out (LIFO) approach. This helped maintain transaction accuracy and provided an easy way to correct recent mistakes.

5. **Implementing the Queue for Report**

The queue was used in the report section of the system. Each transaction processed was enqueued into the report list. When the number of transactions in the report reached ten, the first (oldest) transaction was automatically dequeued, following the First In, First Out (FIFO) rule. This method ensured that only the ten most recent transactions were displayed in the report, keeping it concise and relevant.

6. **Testing and Debugging**

After implementing all modules, the system was tested to ensure that all data structures worked correctly together. Test cases were performed to verify adding, deleting, undoing, and generating reports. Logical and runtime errors were debugged to ensure smooth execution and data consistency across all modules.

7. **Final Integration and Evaluation**

Once all parts were verified, the modules were integrated into one complete system. This program's functionality, efficiency, and user interface were evaluated. The project successfully demonstrated the importance of using appropriate data structures in solving real-world programming problems, particularly in inventory management systems.

8.

Pseudocode

```
Start
    Initialize Linked List for products
    Initialize Stack for transactions
    Initialize Queue for reports (maximum of 10 items allowed)
   Repeat
      Display main menu:
      1. Add product
      2. Edit product
      3. Delete product
      4. Add transaction
      5. Undo last transaction
      6. View report
      7. Exit program
     Input user choice
     If choice = 1 then
        Create a new product node
        Insert product into Linked List
        Display “Product added successfully”
     Else if choice = 2 then
        Input product ID
        Search Linked List for product ID
        If found then
           Update product details
           Display “Product updated successfully”
        Else
           Display “Product does not exist”
        End if
     Else if choice = 3 then
        Input product ID
        Search Linked List for product ID
```

```
            If found then
                Delete product node from Linked List
                Display “Product deleted successfully”
            Else
                Display “Product does not exist”
            End if
        Else if choice = 4 then
            Input product ID and quantity
            Update product quantity in Linked List
            Push transaction onto Stack
            If Queue is full then
                Remove oldest transaction (Dequeue)
            End if
            Insert new transaction into Queue
            Display “Transaction added successfully”
        Else if choice = 5 then
            If Stack is empty then
                Display “No transaction to undo”
            Else
                Pop last transaction from Stack
                Reverse its effect in Linked List
                Remove same transaction from Queue if it exists
                Display “Last transaction has been undone”
            End if
        Else if choice = 6 then
            Display all transactions in Queue
        Else if choice = 7 then
            Display “Exiting program...”
            Exit loop
        Else
            Display “Invalid choice. Please try again.”
         End if
    Until choice = 7
End
```

Algorithm

Main Program

Step 1: Start

Step 2: Initialize an empty Linked List for products

Step 3: Initialize an empty Stack for transactions

Step 4: Initialize an empty Queue for reports (maximum of 10 items allowed)

Step 5: Display the main menu with choices:

1. Add product
2. Edit product
3. Delete product
4. Add transaction
5. Undo last transaction
6. View report
7. Exit program

Step 6: If the user selects:

ADD PRODUCT (uses Linked List)

Step 7: Create a node for the product

Step 8: Insert the new product into the Linked List

Step 9: Show confirmation “Product added successfully”

Step 10: Go back to Step 5

EDIT PRODUCT (uses Linked List)

Step 11: Search the Linked List for the product ID entered by the user

Step 12: If product is found, update its details

Step 13: Show confirmation “Product updated successfully”

Step 14: If product not found, show “Product does not exist”

Step 15: Go back to Step 5

DELETE PRODUCT (uses Linked List)

Step 16: Search the Linked List for the product ID

Step 17: Remove the node from the Linked List

Step 18: Show confirmation “Product deleted successfully”

Step 19: If product not found, show “Product does not exist”

Step 20: Go back to Step 5

ADD TRANSACTION (uses Stack + Queue)

Step 21: Ask the user for product ID and quantity

Step 22: Update the Linked List product quantity based on the transaction

Step 23: Push the transaction into the Stack (LIFO)

Step 24: Show “Transaction added successfully”

Step 25: Go back to Step 5

UNDO TRANSACTION (uses Stack)

Step 26: Check if the Stack is empty

Step 27: If stack is empty, show “No transaction to undo”

Step 28: If not empty, pop the last transaction from the Stack

Step 29: Reverse the effect of the transaction in the Linked List

Step 30: Remove the same transaction from the Queue (if it exists)

Step 31: Show “Last transaction has been undone”

Step 32: Go back to Step 5

VIEW REPORT (uses Queue)

Step 33: Display all transactions currently stored in the Queue

Step 34: Check if the Report Queue already contains 10 items

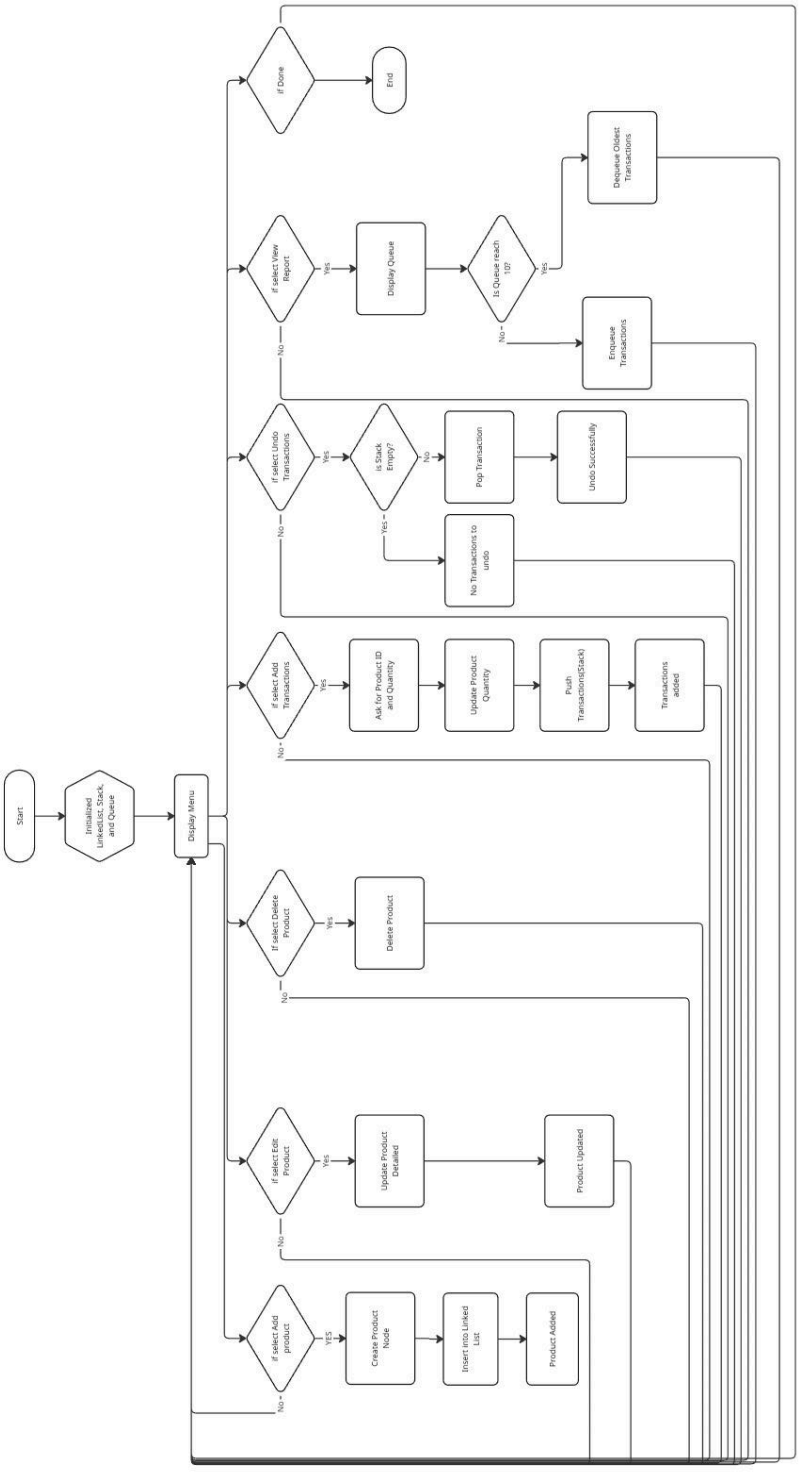
Step 35: If Queue is full, remove the oldest transaction (Dequeue)

Step 36: Insert the new transaction into the Queue

Step 37: Go back to Step 5

Step 38: End

Flowchart


Start
Initialized LinkedList, Stack, and Queue
Display Menu
If Done
End
If select View Report
Yes
Display Queue
Is Queue reach 10?
Yes
Dequeue Oldest Transactions
No
Enqueue Transactions
If select Undo Transactions
Yes
is Stack Empty?
No
Pop Transaction
Undo Successfully
Yes
No Transactions to undo
No
If select Add Transactions
Yes
Ask for Product ID and Quantity
Update Product Quantity
Push Transaction(Stack)
Transactions added
No
If select Delete Product
Yes
Delete Product
No
If select Edit Product
Yes
Update Product Detailed
Product Updated
No
If select Add product
YES
Create Product Node
Insert into Linked List
Product Added
No

Source Code

```
# Stack implementation using list
class Stack:
    def __init__(self):
        self.stack = []

    def push(self, item):
        self.stack.append(item)

    def pop(self):
        if not self.is_empty():
            return self.stack.pop()
        return None

    def peek(self):
        if not self.is_empty():
            return self.stack[-1]
        return None

    def is_empty(self):
        return len(self.stack) == 0

    def size(self):
        return len(self.stack)

# Node class for Linked List
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

# Singly Linked List implementation
class LinkedList:
```

```
        def __init__(self):
            self.head = None

        def append(self, data):
            new_node = Node(data)
            if not self.head:
                self.head = new_node
                return
            last = self.head
            while last.next:
                last = last.next
            last.next = new_node

        def delete(self, key):
            current = self.head
            prev = None
            while current and current.data["id"] != key:
                prev = current
                current = current.next
            if not current:
                return False
            if prev is None:
                self.head = current.next
            else:
                prev.next = current.next
            return True

        def find(self, key):
            current = self.head
            while current:
                if current.data["id"] == key:
                    return current.data
                current = current.next
            return None
```

```
    def to_list(self):
        result = []
        current = self.head
        while current:
            result.append(current.data)
            current = current.next
        return result

    def update(self, key, new_data):
        current = self.head
        while current:
            if current.data["id"] == key:
                current.data.update(new_data)
                return True
            current = current.next
        return False

# Queue implemented with LinkedList
class LinkedListQueue:
    def __init__(self, max_size=10):
        self.list = LinkedList()
        self.front = None
        self.rear = None
        self.max_size = max_size

    def enqueue(self, data):
        if len(self.to_list()) >= self.max_size:
            self.dequeue()

        self.list.append(data)
        if not self.front:
            self.front = self.list.head
        current = self.list.head
```

```
        while current.next:
            current = current.next
        self.rear = current

    def dequeue(self):
        if self.is_empty():
            return None
        data = self.front.data
        self.front = self.front.next
        self.list.head = self.front
        if not self.front:
            self.rear = None
        return data

    def peek(self):
        if self.is_empty():
            return None
        return self.front.data

    def is_empty(self):
        return self.front is None

    def to_list(self):
        return self.list.to_list()

    def remove_matching(self, match_func):
        prev = None
        current = self.list.head
        while current:
            if match_func(current.data):
                removed_data = current.data
                # Remove node from linked list
                if prev is None:
                    # removing head
```

```
                self.list.head = current.next
                self.front = self.list.head
            else:
                prev.next = current.next
                self.front = self.list.head
            # Update rear
            if self.list.head is None:
                self.rear = None
            else:
                cur = self.list.head
                while cur.next:
                    cur = cur.next
                self.rear = cur
            return removed_data
        prev = current
        current = current.next
    return None
```

6. Output

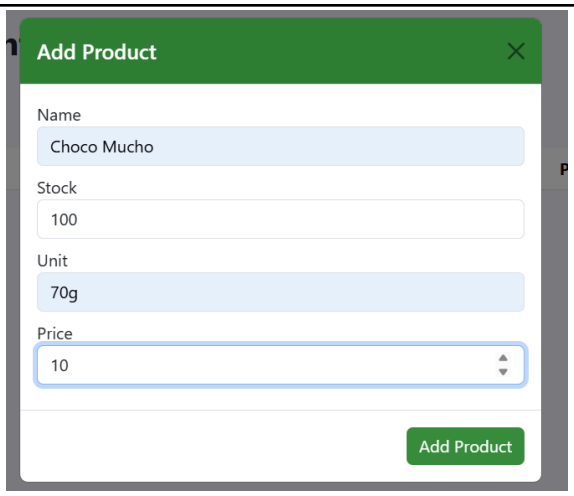


Figure 1: Add Product

In Figure 1, the Add Product feature is shown, allowing the user to input new products with details such as name, stock, unit, and price to keep the inventory updated.

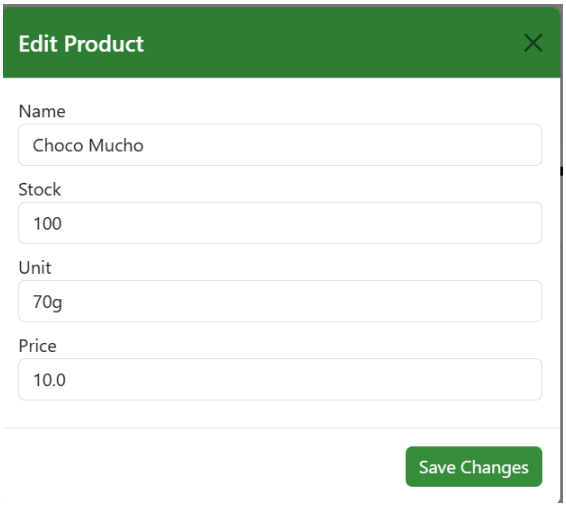


Figure 2: Edit Product

In Figure 2, the Edit Product feature is illustrated, enabling the user to modify existing product information to maintain accurate and current records.

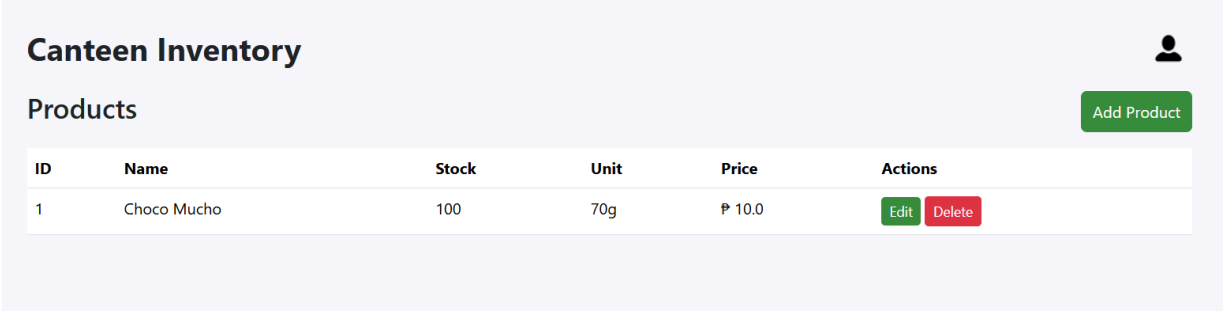


Figure 3: Delete Product

In Figure 3, the Delete Product feature is displayed, allowing the user to remove products that are no longer needed, keeping the inventory organized.

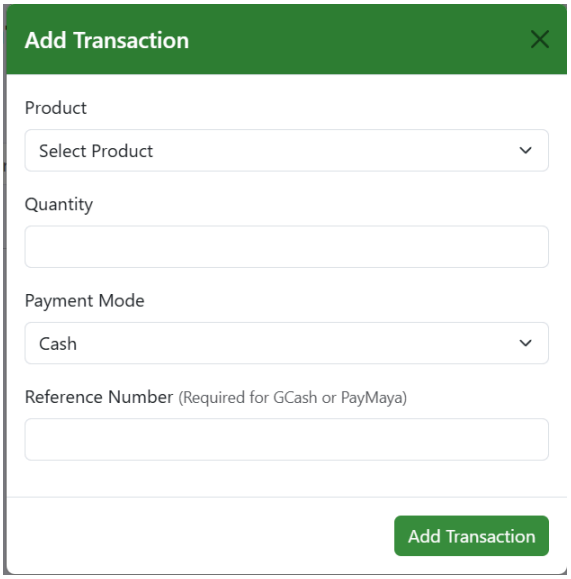


Figure 4: Add Transaction

In Figure 4, the Add Transactions feature is shown, which lets the user record new sales or stock movements linked to the corresponding products for real-time tracking.

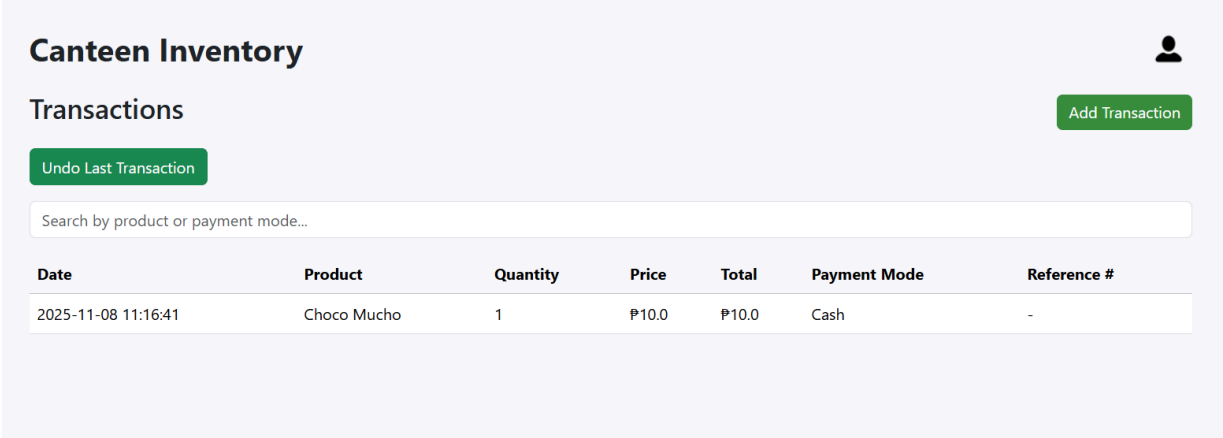


Figure 5: Undo Transaction

In Figure 5, the Undo Transactions feature is displayed, allowing the user to reverse the most recent transaction in case of errors, ensuring inventory accuracy.

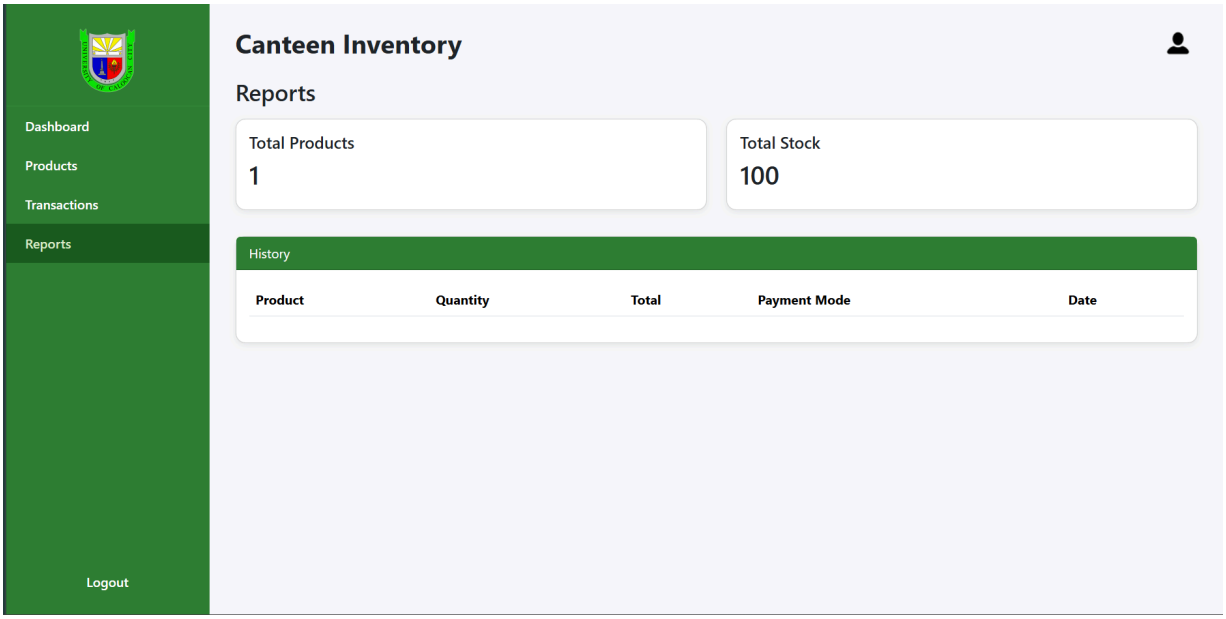


Figure 6: View Report

In Figure 6, the View Transaction feature is illustrated, providing a detailed history of all transactions, including product movements, dates, and amounts, for monitoring and analysis.

7. Conclusion:

In conclusion, developing this Inventory System provided a meaningful opportunity to apply the theoretical principles we learned in the Data Structure and Algorithm course to a real-world programming project. By integrating stack, queue, and linked list data structures, this system demonstrated how proper data organization leads to improved performance, flexibility, and functionality.

The stack feature successfully applied the Last-In, First-Out (LIFO) principle to manage transaction reversals, reflecting how stacks are used for undo operations in real applications [1]. The queue structure ensured fair and orderly report handling by using the First-In,

First-Out (FIFO) principle, maintaining only the most recent ten records to keep data relevant [2]. Meanwhile, the linked list provided dynamic and efficient product data management, supporting quick insertions and deletions without the limitations of fixed memory size [3].

More importantly, this project strengthened the understanding of algorithmic problem-solving and the importance of selecting appropriate data structures for specific system needs. It bridged the gap between classroom concepts and practical application, showing that theoretical knowledge of data structures is fundamental to developing efficient, maintainable, and scalable software solutions.

8. References

[1] GeeksforGeeks, "LIFO (LastInFirstOut) approach in Programming," GeeksforGeeks, Jul. 11, 2025. https://www.geeksforgeeks.org/dsa/lifo-last-in-first-out-approach-in-programming/

[2] GeeksforGeeks, "Queue data structure," GeeksforGeeks, Jul. 23, 2025. https://www.geeksforgeeks.org/dsa/queue-data-structure/

[3] GeeksforGeeks, "Advantages and disadvantages of linked list," GeeksforGeeks, Aug. 05, 2025. https://www.geeksforgeeks.org/dsa/advantages-and-disadvantages-of-linked-list/

[4] Hero Vired, "What is Stack in Data Structures? Types and Advantages," Hero Vired, 2024. [Online]. Available: https://herovired.com/learning-hub/blogs/what-is-stack-in-data-structures/

[5] CCBP, "Implementation of Queue Using Linked List: Program & its Advantages," CCBP, 2024. [Online]. Available: https://www.ccbp.in/blog/articles/queue-using-linked-list

[6] Youcademy, "Advantages and Disadvantages of Linked Lists," Youcademy, 2024. [Online]. Available: https://youcademy.org/advantages-disadvantages-of-linked-lists/