

# **SDF PROJECT REPORT**

---

**JAYPEE INSTITUTE OF  
INFORMATION TECHNOLOGY**

**B. TECH 1<sup>ST</sup> SEMESTER**

**TOPIC: STOCK MANAGEMENT  
SYSYEM IN C**

**BATCH : A - 8**

---

**SUBMITTED BY:**

**SARTHAK SABHARWAL    2401020234**

**PRATYAKSH TYAGI        2401020232**

**KESHAV MAINI            2401020233**

**MAULSHREE SHARMA    2401020218**



---

# Table of Contents

## Topics

- Overview
- Objectives
- Scope
- Problem Statement
- Introduction
- System Design
- Implementation
- Limitations and Future Improvements
- Conclusion

---

# Overview

The **Stock Management System** is a software application created to help businesses manage their inventory efficiently. It keeps track of all products, their quantities, and related details like price and supplier. The system allows users to perform actions such as adding new items, updating stock levels, deleting items, and searching for specific products by name or ID.

Additionally, it helps with generating bills for purchases, logging activities for tracking changes in inventory, and ensuring accurate records. All of this is done through a simple console interface, making it easy for anyone to use. The system reduces manual work and the chances of errors while saving time and effort for businesses.

---

# Objectives

The primary goals of the Stock Management System are:

- **Organized Inventory:** Ensure that all products in stock are recorded accurately with details like name, price, quantity, and ID.
- **Simplified Stock Management:** Provide easy options to add, remove, or update product information as required.
- **Error Reduction:** Reduce the mistakes that can occur in manual stock management, such as incorrect records or missing items.
- **User-Friendly System:** Create a simple and straightforward interface for users with minimal technical knowledge.
- **Logging and Tracking:** Maintain logs of all activities (e.g., changes in stock, purchases) for better accountability and tracking.
- **Billing Integration:** Simplify the process of purchasing products and generating accurate bills.

---

# Scope

The system is designed for small and medium-sized businesses that need a simple way to manage their stock. It is useful in retail stores, warehouses, or any place where products need to be tracked.

The Stock Management System can:

- Keep records of all products in the inventory.
- Help in quickly finding products by their ID or name.
- Ensure businesses know when to reorder products based on stock levels.
- Record purchases and generate bills for customers.
- Provide a history of activities, making it easier to track changes or issues.
- This system focuses on being efficient, accurate, and easy to use. It is best suited for environments where advanced systems may not be affordable or necessary.

---

# Problem Statement

Managing stock manually comes with several challenges:

- It is time-consuming to maintain records in notebooks or spreadsheets.
- Manual processes are prone to errors, such as forgetting to update stock or recording incorrect quantities.
- Businesses can face difficulties in quickly locating product details when needed.
- Without proper logs, it's hard to track changes in inventory or identify who made the changes.
- Generating bills manually can be tedious and may lead to calculation errors.
- To address these issues, the Stock Management System was developed. It automates inventory tasks, ensures accuracy, and saves time. By keeping proper records and logs, it also provides transparency and accountability.

---

# Introduction

The Stock Management System is a simple yet powerful tool created using the C programming language. It leverages basic programming concepts like file handling, user-defined functions, and console management to provide an efficient way to manage inventory.

This system is ideal for businesses looking to organize their stock records without investing in expensive software. It includes functionalities such as adding new products, updating stock levels, searching for items, and generating bills. Each action is logged for better record-keeping and accountability.

Using this system, businesses can ensure their inventory is always up-to-date, reduce errors, and improve productivity. It is a practical solution for anyone who wants to manage their stock without needing advanced technical knowledge.

---

# System Design

The **System Design** section explains how the Stock Management System is built and how its different parts work together to achieve its goals. This includes the system's overall structure, how data is handled using files, and the main components that make everything function smoothly.

## System Architecture

The Stock Management System is designed to work in three main parts:

### 1. User Interface (UI):

- This is what the user interacts with on the screen. It includes menus, options, and prompts for entering details like product name, ID, or quantity.
- The UI is designed to be simple, using a console-based approach that anyone can navigate easily.

### 2. Business Logic:

- This is where the actual work happens. It includes all the functions that allow users to perform tasks like adding a new product, updating stock, or generating bills.
- These functions are carefully written to handle errors and ensure the data entered by the user is valid.



---

### 3.Data Storage:

- Instead of using a database, this system uses files to store all information.
- Files like NextFile.txt, activity\_log.txt, and purchase\_log.txt hold the stock details, user activities, and purchase records. This ensures that data is saved even when the program is closed.

---

## File Handling

File handling is a key part of this system. It allows the program to store data permanently so that it can be accessed later. Here's how it works:

### Files Used in the System:

- **NextFile.txt:** Stores all the product details like name, ID, quantity, and price. This is the main file for managing stock.
- **activity\_log.txt:** Keeps track of every action performed in the system, such as adding a product, deleting an item, or updating stock. This helps in monitoring and accountability.
- **purchase\_log.txt:** Records all the purchases made, including details like the product name, quantity, and total price.

### How File Handling Works:

- When you add, update, or delete an item, the program writes the new information into these files.
- To avoid data loss during updates, a temporary file (e.g., **TempFile.dat**) is created. The updated data is written to this file, and once everything is correct, the temporary file replaces the old file.

---

## Benefits of File Handling in This System:

- **Data Persistence:** The data remains available even if the program is closed.
- **Flexibility:** Users can easily view, modify, or remove records without worrying about overwriting important information.
- **Security:** Each change is logged in the activity file, making it easy to track what has been done.

This design ensures that the system is simple to use, efficient, and reliable, even for users who are not tech-savvy. By dividing the system into these parts, it becomes easier to understand how each section works and contributes to managing inventory effectively.

---

# Implementation

The implementation section breaks down each functionality of the Stock Management System, explaining how it works, its purpose, its significance, and how errors are handled with real-life examples.

---

# Console Management

## Purpose:

Console management makes the user interface more readable and visually appealing by adding colors and managing the text placement on the screen.

## How it Works:

- **setcolor(int ForgC):** Changes the text color in the console. For example, red is used for errors, green for success, and blue for headings.
- **resetcolor():** Resets the text color to default after a message is displayed.
- **gotoxy(short x, short y):** Positions the cursor at specific coordinates on the console screen.

## Significance:

It enhances the user experience, making it easier to differentiate between success, error messages, and general instructions.

## Error Handling:

If an invalid color code is provided to setcolor, the function ignores the request.

- **Example:** If the user tries to use a color code outside the range (0–15), the program will continue without crashing and will not change the text color.

---

## Welcome Screen (void wel\_come(void);)

### **Purpose:**

The welcome screen introduces the program, displaying the project title and contributors in an attractive format.

### **How it Works:**

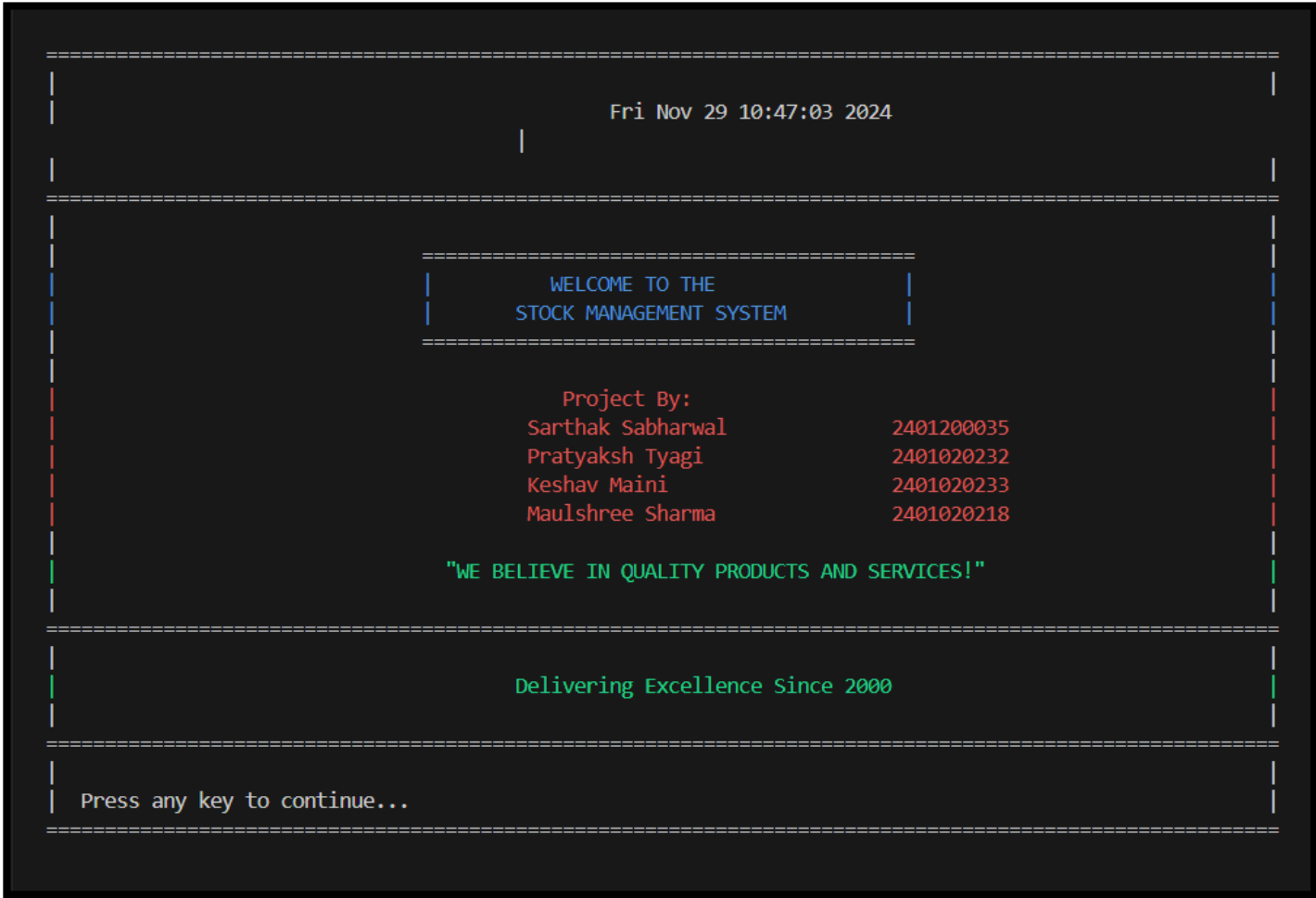
- It shows a banner with the project name, date, and a motivational tagline.
- Color changes highlight different parts of the message.

### **Significance:**

The welcome screen sets the tone for the program and creates a professional impression.

### **Error Handling:**

If a system failure occurs during screen display (e.g., unsupported console operations), the program skips the welcome screen and moves to the login screen.



---

## Login Screen (void login();)

### Purpose:

Ensures only authorized users can access the system.

### How it Works:

- Asks the user for a username and password.
- Compares the entered values with the stored credentials (user and pass).
- Allows up to three attempts before exiting the program.

### Significance:

Protects the system from unauthorized access.

```
===== LOGIN =====
      USERNAME: -user
      PASSWORD: -****
=====
| WELCOME TO STOCK MANAGEMENT SYSTEM !!!! LOGIN IS SUCCESSFUL |
=====
                        Press any key to continue...|
```



---

## Error Handling:

- **Case 1:** If the user enters incorrect credentials, they see an error message and are informed of the remaining attempts.
  - **Example:** Entering "wronguser" and "wrongpass" will display:  
*"Login unsuccessful. You have 2 attempts left."*
- **Case 2:** If the user fails all three attempts, the program terminates with a message: *"Access denied after 3 attempts."*

```
===== LOGIN =====  
      USERNAME:-user1  
  
      PASSWORD:_****  
      SORRY !!!! LOGIN IS UNSUCCESSFUL  
      You have 2 attempt(s) left
```

---

## Main Menu (void menu();)

### Purpose:

The main menu serves as the central navigation hub for the system.

### How it Works:

- Displays a list of options (e.g., Add Items, Read Products, Search Products).
- Accepts the user's choice and redirects them to the corresponding function.

### Significance:

Provides a structured way to access all features of the system.

```
=====
|                               Stock Management System                               |
=====

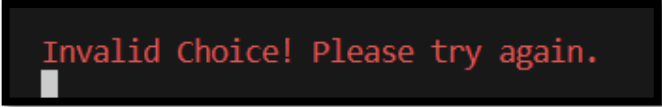
| Choice | Action                               |
|-----|-----|
| 1      | Add Items                           |
| 2      | Read Products                       |
| 3      | Search Products                     |
| 4      | Edit Items                          |
| 5      | Update Items                        |
| 6      | Purchase Items                      |
| 7      | Delete Items                        |
| 8      | View Activity Log                   |
| 9      | Exit                                |
|-----|-----|

Enter your choice [1-9]: █
```

---

## Error Handling:

- Case 1: If the user enters an invalid choice (e.g., 10), they see a message: *"Invalid choice! Please try again."* The menu is displayed again.
- Case 2: If the input is non-numeric (e.g., "abc"), the program clears the input buffer and asks for a valid choice.

A terminal window with a black background and a white border. It displays the text "Invalid Choice! Please try again." in a red, monospaced font. A white cursor is positioned at the end of the line, below the text.

```
Invalid Choice! Please try again.
```

---

## Add Items or Delete Items( void add item(); and void deleteproduct(void); )

### **Purpose:**

- Adds new products to the stock.
- Delete pre-existing products from the stock.

### **How it Works:**

#### **-To add:**

- Prompts the user for product details (name, company, price, ID, and quantity).
- Validates the inputs and stores the details in NextFile.txt.

#### **-To delete:**

- Prompts the user to enter the name of the product you want to delete.
- Deletes the products if it exists.

### **Significance:**

Keeps the inventory up-to-date with new products and deletes the product you want.

## Add Product:

```
> ===== Enter Product Detail =====  
  
+-----+  
| Product ID : 05  
+-----+  
| Product Name : Chips  
+-----+  
| Product Company : Lays  
+-----+  
| Price [10-5000] Rupees : 50  
+-----+  
| Quantity [1-500] : 150  
+-----+  
  
===== Item Added Successfully =====  
Press 'Enter' to add another item or any other key to go to the main menu
```

## Delete Product:

```
===== Delete Product =====  
  
+-----+  
| Enter product name to delete: Detergent  
+-----+  
  
+-----+  
| Product Deleted Successfully  
+-----+  
  
Press any key to go to Main Menu!  
█
```

## Error Handling:

- **Case 1:** If the product ID already exists, the user sees: "The product ID already exists."

```
===== Enter Product Detail =====  
  
+-----+  
| Product ID : 95  
  
THE PRODUCT ID ALREADY EXISTS.  
  
+-----+
```

- **Case 2:** If the price or quantity is out of range (e.g., negative price or quantity above 500), the program rejects the input and asks the user to re-enter valid values.
- **Case 3 :** If the product name does not match any name present in the inventory, it shows the message : "Product not found"

```
===== Delete Product =====  
+-----+  
| Enter product name to delete: SAMPLE  
  
+-----+  
| Product Not Found |  
+-----+  
  
Press any key to go to Main Menu!  
█
```

## Update Product Quantity ( void update\_stock(); )

### Purpose:

Allows users to add to or reduce the stock of an existing product.

### How it Works:

- Searches for the product by ID.
- Asks the user to choose between adding or reducing the stock.
- Updates the file with the new quantity.

### Significance:

Helps maintain accurate stock levels.

```
===== UPDATE STOCK =====
| Name | Price | Company | ID | Quantity |
+-----+-----+-----+---+
| Toothpaste | 60 | Colgate | 1 | 50 |
| Shampoo | 150 | Dove | 2 | 30 |
| Soap | 40 | Lux | 3 | 100 |
| Biscuit | 20 | ParleG | 4 | 200 |
| Chips | 50 | Lays | 5 | 150 |
| Notebool | 80 | Classmate | 6 | 60 |
| Pen | 10 | Reynolds | 7 | 500 |
| Detergent | 100 | SurfExcel | 8 | 80 |
+-----+-----+-----+---+

Enter the Product ID to update stock: 6

Product Found: Notebool (ID: 6)
Current Stock: 60

Choose an option:
+-----+
| 1. Increase Stock |
| 2. Decrease Stock |
+-----+

Enter your choice: 1
Enter the quantity to add: 10
+-----+
| Stock updated! New Stock: 70 |
+-----+
Press any key to return to the menu...|
```

---

## Error Handling:

- **Case 1:** If the product ID does not exist, the user sees: "Error: Product ID not found."
- **Case 2:** If the quantity to reduce exceeds the available stock, an error message is displayed: "Error: Insufficient stock."



---

## Edit Product Details ( void edit\_item(); )

### **Purpose:**

Enables users to modify existing product details like name, company, price, or quantity.

### **How it Works:**

- Finds the product by ID.
- Prompts the user to enter new values for the details.
- Updates the file with the new information.

### **Significance:**

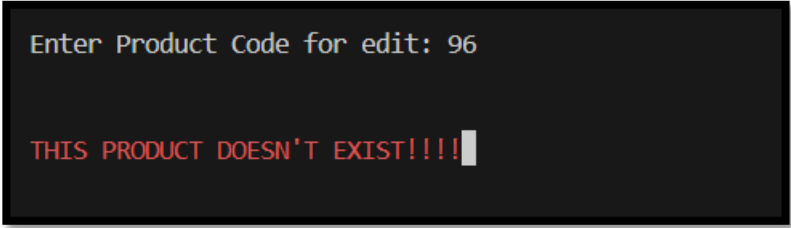
Allows corrections and updates to product information.

```
===== Edit Item =====  
  
Enter Product ID for edit: 8  
  
+-----+  
|           Record Found           |  
+-----+  
  
Product Name       : Detergent  
Product Company    : SurfExcel  
Price              : 90  
Product ID         : 8  
Product Quantity   : 80  
  
+-----+  
|           New Record             |  
+-----+  
  
New Product Name    : Detergent  
  
New Product Company : SurfExcel  
  
New Price [10-5000] Rupees : 100  
  
Enter New Product ID : 8  
  
New Quantity [1-500] : 80  
  
Press 'y' to edit the existing record or any key to cancel...y  
  
+-----+  
|           YOUR RECORD IS SUCCESSFULLY EDITED!!!           |  
+-----+  
█
```

---

## Error Handling:

- **Case 1:** If the product ID is not found, the user sees: "This product doesn't exist."
- **Case 2:** If invalid data is entered (e.g., non-alphabetic characters in the name), the program asks the user to re-enter valid data.

A screenshot of a terminal window with a dark background. The first line shows a prompt 'Enter Product Code for edit: 96' in a light green color. The second line shows an error message 'THIS PRODUCT DOESN'T EXIST!!!!' in a red color, followed by a white cursor block.

```
Enter Product Code for edit: 96
```

```
THIS PRODUCT DOESN'T EXIST!!!!
```

---

## Read Items in Stock ( void read\_item(); )

### Purpose:

Displays all the products currently available in the stock.

### How it Works:

- Reads data from NextFile.txt.
- Formats and displays the details in a tabular format.

### Significance:

Provides a quick overview of the inventory.

```
===== Item List =====
```

Product Name	Price (Rupees)	Company	Product ID	Quantity
Toothpaste	60	Colgate	1	50
Shampoo	150	Dove	2	30
Soap	40	Lux	3	100
Biscuit	20	ParleG	4	200
Chips	50	Lays	5	150
Notebook	80	Classmate	6	70
Pen	10	Reynolds	7	500
Juice	120	Real	8	40
Detergent	100	SurfExcel	9	80
Milk	60	Amul	10	70

### Error Handling:

If the file is empty or does not exist, the user sees: "No records available in the database."

---

## Search Item by ID or Name ( void search item(); )

### **Purpose:**

Finds a product based on its name or ID.

### **How it Works:**

- Lets the user choose between searching by name or ID.
- Scans the file and displays the product details if found.

### **Significance:**

Saves time in locating specific products.

===== Search Product =====

```
+-----+
| 1. Search by Product Name |
| 2. Search by Product ID   |
+-----+
```

| Enter your choice (1/2): 2

Enter product ID to search: 4

```
+-----+
|              Record Found              |
+-----+
| Product Name      : Biscuit             |
| Product Company   : ParleG              |
| Product Price     : 20                  |
| Product ID        : 4                   |
| Product Quantity  : 200                 |
+-----+
```

Press any key to go to Main Menu!█

---

## Error Handling:

If the product is not found, the user sees: "No record found with the entered name/ID."

```
===== Search Product =====
+-----+
| 1. Search by Product Name      |
| 2. Search by Product ID       |
+-----+
| Enter your choice (1/2): 1     |
+-----+

Enter product name to search: Colgate

No record found with the name 'Colgate'.

Press any key to go to Main Menu!
```

## Purchase Items & Bill Generation ( void purchase\_item(); )

### Purpose:

Allows the user to purchase products and generates a bill.

### How it Works:

- Deducts the purchased quantity from the stock.
- Creates a bill with product details and total price.

### Significance:

Simplifies the sales process and keeps stock updated.

```
Enter the Product ID to purchase: 5
Enter quantity to purchase: 10

=====
|                               Purchase Successfull                               |
=====
| Product ID   : 5
| Product Name : Chips
| Company      : Lays
| Price (Each) : 50
| Quantity     : 10
| Total Price  : 500
|
|
=====
|                               Bill Generated in 'bill.txt'                               |
=====

Press any key to go to Main Menu!
```



```
File Edit View
===== BILL =====
Product Name : Chips
Company      : Lays
Price (Each) : 50
Quantity     : 10
Total Price  : 500
=====
|
```

## Error Handling:

- **Case 1:** If the requested quantity exceeds available stock, the user sees: "Error: Insufficient stock. Available quantity: X."
- **Case 2:** If the product ID does not exist, the user is notified: "Product not found."

---

Logging Activities ( void log\_activity(const char \*message);  
 , void log\_purchase(const char \*message); and void  
 view\_activity\_log(); )

**Purpose:**

Records all changes and transactions in log files.

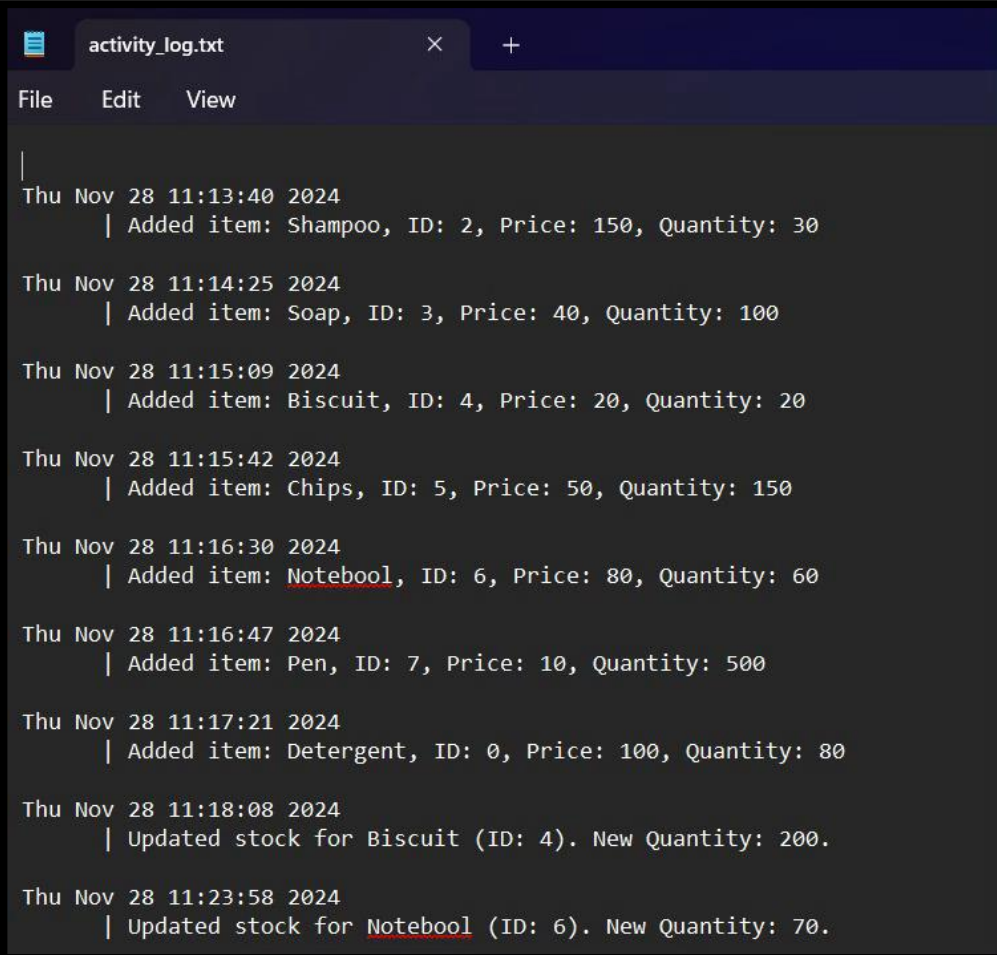
**How it Works:**

- Logs actions like adding, deleting, or updating items in activity\_log.txt.
- Logs purchases in purchase\_log.txt.

**Significance:**

Ensures accountability and helps track changes.

## Activity Log (in .txt form)



```
activity_log.txt
File Edit View
|
Thu Nov 28 11:13:40 2024
    | Added item: Shampoo, ID: 2, Price: 150, Quantity: 30

Thu Nov 28 11:14:25 2024
    | Added item: Soap, ID: 3, Price: 40, Quantity: 100

Thu Nov 28 11:15:09 2024
    | Added item: Biscuit, ID: 4, Price: 20, Quantity: 20

Thu Nov 28 11:15:42 2024
    | Added item: Chips, ID: 5, Price: 50, Quantity: 150

Thu Nov 28 11:16:30 2024
    | Added item: Notebool, ID: 6, Price: 80, Quantity: 60

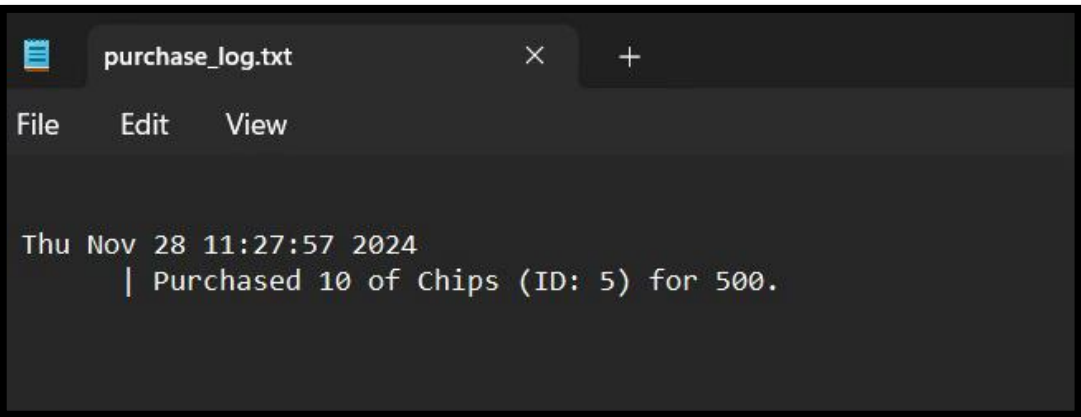
Thu Nov 28 11:16:47 2024
    | Added item: Pen, ID: 7, Price: 10, Quantity: 500

Thu Nov 28 11:17:21 2024
    | Added item: Detergent, ID: 0, Price: 100, Quantity: 80

Thu Nov 28 11:18:08 2024
    | Updated stock for Biscuit (ID: 4). New Quantity: 200.

Thu Nov 28 11:23:58 2024
    | Updated stock for Notebool (ID: 6). New Quantity: 70.
```

## Purchase Log (in .txt form)



```
purchase_log.txt
File Edit View

Thu Nov 28 11:27:57 2024
    | Purchased 10 of chips (ID: 5) for 500.
```

## Activity log (display in program)

```
===== Activity Log =====
Thu Nov 28 11:13:40 2024
  | Added item: Shampoo, ID: 2, Price: 150, Quantity: 30
Thu Nov 28 11:14:25 2024
  | Added item: Soap, ID: 3, Price: 40, Quantity: 100
Thu Nov 28 11:15:09 2024
  | Added item: Biscuit, ID: 4, Price: 20, Quantity: 20
Thu Nov 28 11:15:42 2024
  | Added item: Chips, ID: 5, Price: 50, Quantity: 150
Thu Nov 28 11:16:30 2024
  | Added item: Notebool, ID: 6, Price: 80, Quantity: 60
Thu Nov 28 11:16:47 2024
  | Added item: Pen, ID: 7, Price: 10, Quantity: 500
Thu Nov 28 11:17:21 2024
  | Added item: Detergent, ID: 0, Price: 100, Quantity: 80
Thu Nov 28 11:18:08 2024
  | Updated stock for Biscuit (ID: 4). New Quantity: 200.
Thu Nov 28 11:23:58 2024
  | Updated stock for Notebool (ID: 6). New Quantity: 70.
Thu Nov 28 11:24:39 2024
  | Deleted item: Detergent, ID: 8, Price: 100, Quantity: 80
Thu Nov 28 11:25:38 2024
  | Added item: Juice, ID: 8, Price: 120, Quantity: 40
Thu Nov 28 11:26:00 2024
  | Added item: Detergent, ID: 9, Price: 100, Quantity: 80
Thu Nov 28 11:26:56 2024
  | Added item: Milk, ID: 10, Price: 60, Quantity: 70
```

## Error Handling:

If the log file cannot be opened, the user sees: "Error: Could not log activity." The program continues without logging.

---

## Exit the Program

### **Purpose:**

Ends the program gracefully.

### **How it Works:**

- Displays an exit message.
- Ensures all files are closed before termination.

### **Significance:**

Prevents data loss and ensures smooth program closure.

### **Error Handling:**

If the program is terminated abruptly, unsaved data might be lost. Proper use of logs minimizes this risk.

---

# Limitations and Future Improvements

## Limitations:

- The system uses text files for storage, which can slow down performance with large datasets.
- The console-based interface lacks visual appeal and may not be intuitive for all users.
- Only a single user login is supported, limiting its use in multi-user environments.
- The system cannot handle real-time stock updates or automatic synchronization with other systems.
- It lacks advanced features like barcode scanning or integration with other business tools.

## Future Improvements:

- Transitioning to a database like MySQL or MongoDB for faster and more scalable data management.
- Developing a graphical user interface (GUI) to make the system more user-friendly and visually engaging.
- Adding multi-user support with role-based access to enhance security and collaboration.
- Implementing real-time stock tracking and low-stock alerts for dynamic inventory management.
- Integrating the system with external tools like barcode scanners, online ordering platforms, or accounting software.
- Migrating to a cloud-based system for better accessibility and multi-device usage.

---

## Conclusion

The **Stock Management System** is a well-structured solution designed to simplify inventory management for small and medium-sized businesses. It effectively automates key tasks such as adding, updating, and deleting stock, while ensuring data accuracy and consistency through robust file handling mechanisms. The system's features, like product search, billing, and activity logging, provide a comprehensive approach to managing inventory efficiently.

By leveraging simple programming concepts, the system is accessible to users with minimal technical expertise, making it an ideal choice for businesses looking for cost-effective tools. Its focus on user-friendly interaction, transparency through activity logs, and secure data handling highlights its practical application in real-world scenarios. However, the system does have some limitations, such as reliance on text-based storage and a basic console interface. Despite these challenges, it provides a solid foundation for inventory management and can be further enhanced with features like database integration, real-time updates, and a graphical user interface.

In conclusion, the Stock Management System meets its primary objectives by addressing common inventory challenges, reducing manual errors, and saving time. With planned future improvements, it has the potential to evolve into an even more versatile and scalable tool for businesses of all sizes.