

**SUPERMARKET**

**MANGEMENT SYSTEM**

**Submitted by**

ROHIT(231801139)

SACHIN(231801144)

PRAKASH (231801127)

CS23333-OBJECT ORIENTED PROGRAMMING USING JAVA

Department of Artificial Intelligence and Machine Learning

**Rajalakshmi Engineering College, Thandalam**



# BONAFIDE CERTIFICATE

NAME………………………………………………………………………………...............

ACADEMIC YEAR………………SEMESTER………….BRANCH………………………..

UNIVERSITY REGISTER No.

Certified that this is the bonafide record of work done by the above students in the Mini Project titled " **SUPERMARKET MANGEMENT SYSTEM**" in the subject CS23333 **–** OBJECT ORIENTED PROGRAMMING USING JAVA during the year 2023 - 2024.

**Signature of Faculty – in – Charge**

**Submitted for the Practical Examination held on**

**Internal Examiner External Examiner**

## TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **S.No** | **Chapter** | **Page Number** |
| 1. | ABSTRACT | iv |
| 2. | INTRODUCTION | 1 |
| 3. | LITERATURE SURVEY | 3 |
| 4. | MODEL ARCHITECTURE | 4 |
| 5. | IMPLEMENTATION | 8 |
| 6. | RESULT | 17 |
| 7. | CONCLUSION | 19 |
| 8. | REFERENCES | 21 |

## ABSTRACT

The Supermarket Management System is a Java-based application that aims to automate and optimize supermarket operations. By implementing core object-oriented programming (OOP) principles like encapsulation, inheritance, and polymorphism, the system ensures modularity, reusability, and scalability.

The application enables efficient management of inventory, billing, and reporting processes while providing role-based access control for administrators and cashiers.

Key features include inventory management for adding, updating, and deleting products, an automated billing system for generating accurate invoices, and search functionality for quick product lookups. The system also generates sales and inventory reports to assist in decision-making. To ensure smooth operation, robust exception handling mechanisms are integrated, and data persistence is achieved through file handling or lightweight databases.

This project addresses the inefficiencies of manual systems, such as data redundancy and operational delays, by offering an affordable and user-friendly digital solution. It serves as a practical demonstration of Java’s OOP capabilities, highlighting their effectiveness in solving real-world challenges. The system is particularly suitable for small to medium-sized supermarkets, providing an accessible platform for improving operational efficiency, reducing manual errors, and enhancing customer satisfaction. Future expansions could include cloud integration and advanced analytics to broaden its scope.

**CHAPTER 1**

**INTRODUCTION**

The retail industry, especially supermarkets, faces challenges in managing inventory, billing, and sales due to the complexity of operations and the sheer volume of transactions. Manual processes often result in inefficiencies, such as inventory mismanagement, billing errors, and slow customer service. To address these issues, businesses increasingly rely on automated systems that streamline operations and improve accuracy.

The Supermarket Management System aims to provide a solution tailored to the needs of small and medium-sized supermarkets. This Java-based project leverages object-oriented programming (OOP) principles to create a modular, scalable, and maintainable system. It enables administrators to manage inventory, track sales, and generate reports, while cashiers can efficiently handle billing processes.

The project’s primary objectives include simplifying operational workflows, reducing manual errors, and enhancing decision-making through real-time data availability. By using a structured and user-friendly interface, the system minimizes training time for employees and boosts productivity. Furthermore, it integrates essential features like search functionality, role-based access, and automated billing to address common challenges faced by supermarkets.

This project not only demonstrates the practical application of Java’s OOP principles but also provides a real-world solution for improving operational efficiency in the retail sector.

**ALGORITHM USED :**

The **Supermarket Management System** employs algorithms tailored for inventory management, billing, and user interaction. The system integrates basic data structures (like arrays and lists) and follows modular object-oriented programming principles to ensure efficiency and maintainability. Below is an overview of the core algorithms:

**1. Inventory Management Algorithm**

This module enables CRUD operations (Create, Read, Update, Delete) for products.

**Steps:**

**Add Product**

* + Input product details (name, ID, category, price, quantity).
  + Check for duplicate product IDs using a search loop.
  + If unique, append the product object to the inventory list.

**Update Product**

* + Input product ID.
  + Search the inventory list using a linear search or hash map (if implemented).
  + If found, update the product details; else, prompt an error.

**Delete Product**

* + Input product ID.
  + Locate the product using a search loop.
  + Remove the product from the list if it exists.

**Search Product**

* + Input product ID or name.
  + Use a linear search to locate matching entries.

**2. Billing Algorithm**

This module calculates the total cost of purchased items, applies discounts, and generates invoices.

**Steps:**

1. Input the list of items (product ID and quantity).
2. For each product:
   * Search inventory for matching product ID.
   * If available, verify stock availability. Deduct the purchased quantity from inventory.
   * Calculate the total cost (quantity × price).
3. Apply discounts or taxes if applicable.
4. Generate a bill with itemized details.

**3. Role-Based Access Control Algorithm**

Ensures user-specific functionalities.

**Steps:**

1. Input user credentials.
2. Authenticate using a stored username-password list.

**CHAPTER 2**

## LITERATURE SURVEY

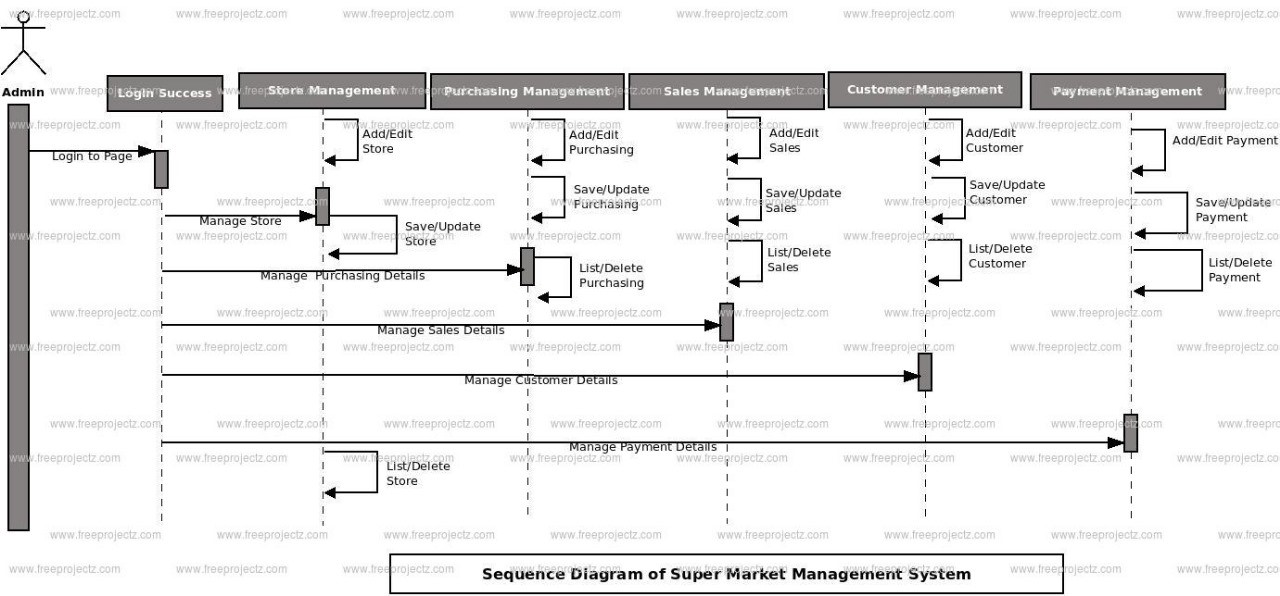
Supermarket management systems are vital in the retail sector, streamlining operations like inventory control, billing, and reporting. Despite the availability of several commercial solutions, many existing systems are either proprietary, prohibitively expensive, or overly complex, making them unsuitable for small to medium-sized businesses. As a result, these businesses often struggle with inefficiencies, including manual errors, data inconsistencies, and the inability to scale operations effectively.

Research and literature on retail management systems emphasize the need for solutions that are user-friendly, scalable, and cost-effective. These systems should integrate core functionalities such as inventory tracking, billing automation, and reporting, while maintaining simplicity to minimize training requirements. Studies have shown that automation in retail management significantly reduces errors, improves efficiency, and enhances customer satisfaction. However, poorly designed systems often suffer from challenges such as data redundancy, lack of modularity, and high maintenance costs, further deterring adoption by smaller businesses.

This project addresses these challenges by leveraging Java’s object-oriented programming (OOP) principles to develop a lightweight, modular, and scalable Supermarket Management System. The design focuses on being resource-efficient and easy to use, making it accessible to small businesses with limited technical expertise or infrastructure. By combining inventory management, automated billing, and reporting into a single platform, the system provides an affordable and practical alternative to traditional solutions. It integrates best practices in software development to ensure robustness, reliability, and adaptability, effectively aligning with the specific needs of its target audience.

**CHAPTER 3**

## MODEL ARCHITECTURE



* **Data Collection**: Collecting data such as product information (e.g., name, category, price) and customer details (e.g., name, contact info).
* **Data Preprocessing**: Cleaning the data, such as validating product categories, checking for out-of-stock items, etc.
* **Feature Extraction**: Extracting useful attributes such as product price, category, and customer preferences for recommendations.
* **Model Training**: In a supermarket context, this could involve building a recommendation system for products based on past purchase history.
* **Model Testing and Validation**: Testing the system with a set of data to ensure it can provide accurate recommendations.
* **Order Processing (Message Classification)**: The system classifies customer orders, generates receipts, and updates the inventory.

**Tools and Libraries**:

**1. Java Development Tools (JDK)**

* **JDK (Java Development Kit)**: You’ll need the JDK to compile and run your Java code. You can download it from the [Oracle website](https://www.oracle.com/java/technologies/javase-jdk11-downloads.html) or use an OpenJDK version.
* **IDE (Integrated Development Environment)**: A good IDE will help you write, debug, and manage your code efficiently. Some popular Java IDEs are:
  + **IntelliJ IDEA** (Recommended for Java developers)
  + **Eclipse** (Open-source IDE)
  + **NetBeans** (Free, open-source IDE)

**2. Database and Persistence Libraries**

For managing products, customers, and order data, a database can be used. Java offers multiple libraries for connecting to databases:

* **JDBC (Java Database Connectivity)**: The standard API for connecting to relational databases.
* **JPA (Java Persistence API)**: A standard for object-relational mapping (ORM) that simplifies database access using annotations to map Java objects to database tables.
  + **Hibernate**: A popular ORM framework that implements JPA and offers additional features like caching, lazy loading, etc.
  + **Spring Data JPA**: A part of the Spring framework that simplifies the integration of JPA with repositories for data access.
* **MySQL / PostgreSQL / SQLite**: You can use any of these relational databases for storing data. MySQL and PostgreSQL are popular, while SQLite is lightweight and suitable for small-scale applications.

**3. Web Framework (If You Build a Web Application)**

If you plan to build a web-based supermarket system, a web framework will help you structure your application.

* **Spring Boot**: A powerful framework for building Java web applications. It simplifies configurations and integrates well with databases and RESTful APIs.
* **Spring MVC**: A module in Spring used for building web applications based on the Model-View-Controller architecture.
* **JSF (JavaServer Faces)**: A Java web framework for building user interfaces.
* **Apache Tomcat**: A widely-used servlet container for running Java web applications.

**4. Libraries for User Interface (GUI)**

If you are building a desktop application with a graphical user interface (GUI), you can use:

* **JavaFX**: A powerful framework for building modern desktop UIs. It allows for the creation of rich, interactive interfaces with support for multimedia, 2D/3D graphics, and animations.
* **Swing**: Older than JavaFX, but still widely used for creating desktop applications in Java. Swing provides components like buttons, text fields, and panels to build a UI.

**5. Testing and Validation Libraries**

For ensuring your supermarket system works as expected, you can use testing frameworks.

* **JUnit**: The most commonly used testing framework for Java applications. It helps you write unit tests to verify that individual components of the system work correctly.
* **Mockito**: A framework for mocking objects and dependencies in unit tests, useful when you want to simulate database access or external services.
* **AssertJ**: A library that provides fluent assertions for more readable tests.

**6. Logging Libraries**

To keep track of errors and system activity, use a logging framework.

* **SLF4J** (Simple Logging Facade for Java): A logging facade that allows you to plug in different logging frameworks.
* **Logback**: A popular logging framework that integrates with SLF4J and offers advanced features like rolling file logs.
* **Log4j 2**: A high-performance logging library with great flexibility.

**7. Security Libraries**

If your supermarket system requires authentication and authorization, consider using:

* **Spring Security**: A powerful and customizable authentication and access control framework for Java web applications.
* **JWT (JSON Web Token)**: A widely-used standard for securely transmitting information between parties as a JSON object. Used for API authentication.

**8. Build and Dependency Management**

To manage your project dependencies and build process:

* **Maven**: A popular build automation tool that manages project dependencies and simplifies builds.
* **Gradle**: Another build tool, known for being more flexible and faster than Maven.

**9. Other Useful Libraries and Tools**

* **Apache Commons**: A set of reusable Java components that can be used for tasks like file operations, collections manipulation, and more.
* **Google Gson** or **Jackson**: For working with JSON data, which is common in APIs.
* **JUnit5**: The latest version of JUnit for unit testing.
* **Lombok**: A Java library that reduces boilerplate code by generating common methods like getters, setters, constructors, and equals/hashCode implementations.

**IMPLEMENTATION :**

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Scanner;

public class SupermarketManagementSystem {

public static void main(String[] args) {

ArrayList<HashMap<String, Object>> items = new ArrayList<>();

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("------------------Welcome to the supermarket------------------");

System.out.println("1. View items\n2. Add items for sale\n3. Purchase items\n4. Search items \n5. Edit items\n6. Exit");

System.out.print("Enter the number of your choice: ");

String choice = scanner.nextLine();

switch (choice) {

case "1":

System.out.println("------------------View Items------------------");

System.out.println("The number of items in the inventory are: " + items.size());

if (!items.isEmpty()) {

System.out.println("Here are all the items available in the supermarket:");

for (HashMap<String, Object> item : items) {

item.forEach((key, value) -> System.out.println(key + ": " + value));

}

} else {

System.out.println("No items in the inventory.");

}

break;

case "2":

System.out.println("------------------Add Items------------------");

System.out.println("To add an item, fill in the form.");

HashMap<String, Object> newItem = new HashMap<>();

System.out.print("Item name: ");

newItem.put("name", scanner.nextLine());

System.out.print("Item quantity: ");

while (!scanner.hasNextInt()) {

System.out.println("Quantity should only be in digits. Try again:");

scanner.next();

}

newItem.put("quantity", scanner.nextInt());

System.out.print("Price $: ");

while (!scanner.hasNextInt()) {

System.out.println("Price should only be in digits. Try again:");

scanner.next();

}

newItem.put("price", scanner.nextInt());

scanner.nextLine(); // Consume the leftover newline

items.add(newItem);

System.out.println("Item has been successfully added.");

break;

case "3":

System.out.println("------------------Purchase Items------------------");

System.out.print("Which item do you want to purchase? Enter name: ");

String purchaseItem = scanner.nextLine();

boolean itemFoundForPurchase = false;

for (HashMap<String, Object> item : items) {

if (purchaseItem.equalsIgnoreCase((String) item.get("name"))) {

itemFoundForPurchase = true;

int quantity = (int) item.get("quantity");

if (quantity > 0) {

System.out.println("Pay $" + item.get("price") + " at the checkout counter.");

item.put("quantity", quantity - 1);

} else {

System.out.println("Item out of stock.");

}

break;

}

}

if (!itemFoundForPurchase) {

System.out.println("Item not found.");

}

break;

case "4":

System.out.println("------------------Search Items------------------");

System.out.print("Enter the item's name to search in inventory: ");

String findItem = scanner.nextLine();

boolean itemFoundForSearch = false;

for (HashMap<String, Object> item : items) {

if (findItem.equalsIgnoreCase((String) item.get("name"))) {

itemFoundForSearch = true;

System.out.println("The item named " + findItem + " is displayed below with its details:");

item.forEach((key, value) -> System.out.println(key + ": " + value));

break;

}

}

if (!itemFoundForSearch) {

System.out.println("Item not found.");

}

break;

case "5":

System.out.println("------------------Edit Items------------------");

System.out.print("Enter the name of the item that you want to edit: ");

String editItemName = scanner.nextLine();

boolean itemFoundForEdit = false;

for (HashMap<String, Object> item : items) {

if (editItemName.equalsIgnoreCase((String) item.get("name"))) {

itemFoundForEdit = true;

System.out.println("Here are the current details of " + editItemName + ":");

item.forEach((key, value) -> System.out.println(key + ": " + value));

System.out.print("New item name: ");

item.put("name", scanner.nextLine());

System.out.print("New item quantity: ");

while (!scanner.hasNextInt()) {

System.out.println("Quantity should only be in digits. Try again:");

scanner.next();

}

item.put("quantity", scanner.nextInt());

System.out.print("New price $: ");

while (!scanner.hasNextInt()) {

System.out.println("Price should only be in digits. Try again:");

scanner.next();

}

item.put("price", scanner.nextInt());

scanner.nextLine(); // Consume the leftover newline

System.out.println("Item has been successfully updated.");

break;

}

}

if (!itemFoundForEdit) {

System.out.println("Item not found.");

}

break;

case "6":

System.out.println("------------------Exited------------------");

scanner.close();

return;

default:

System.out.println("You entered an invalid option.");

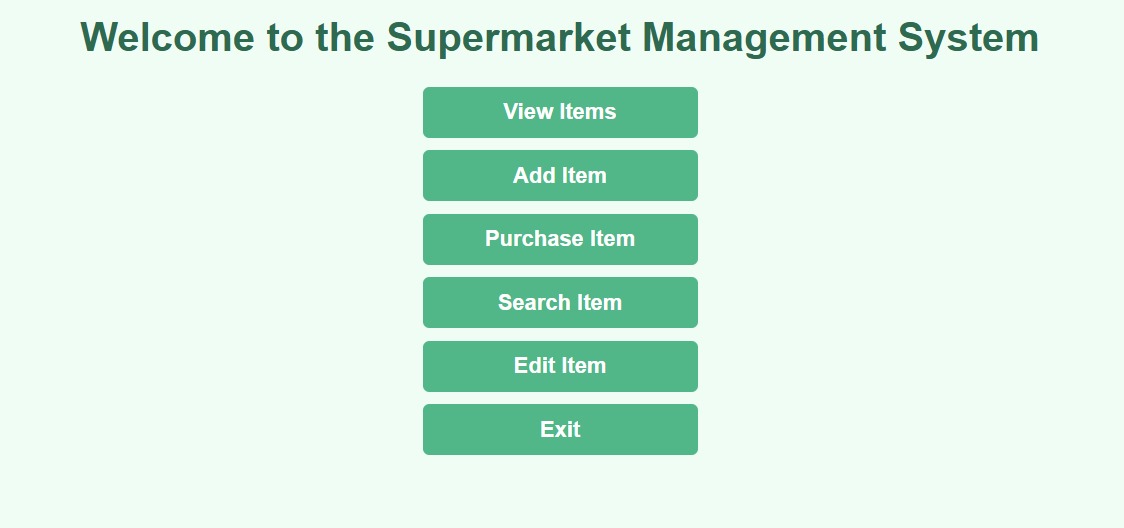
}

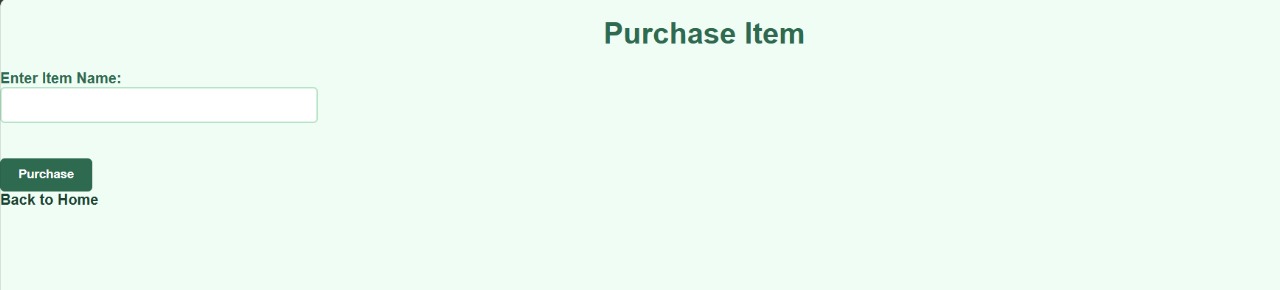
}

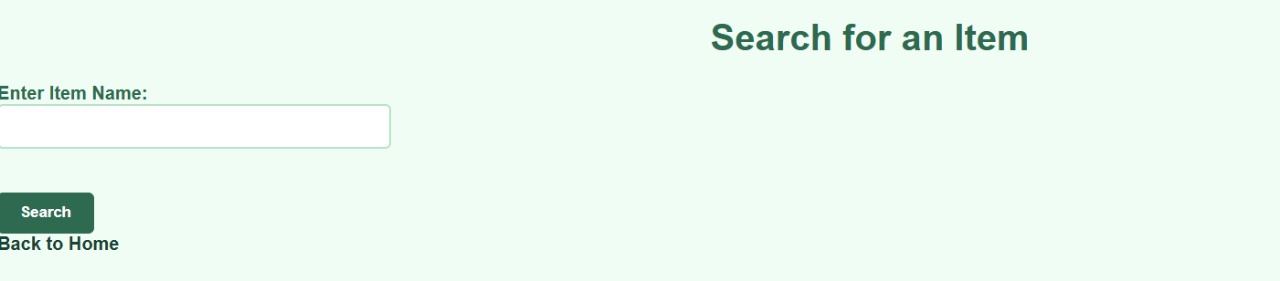
}

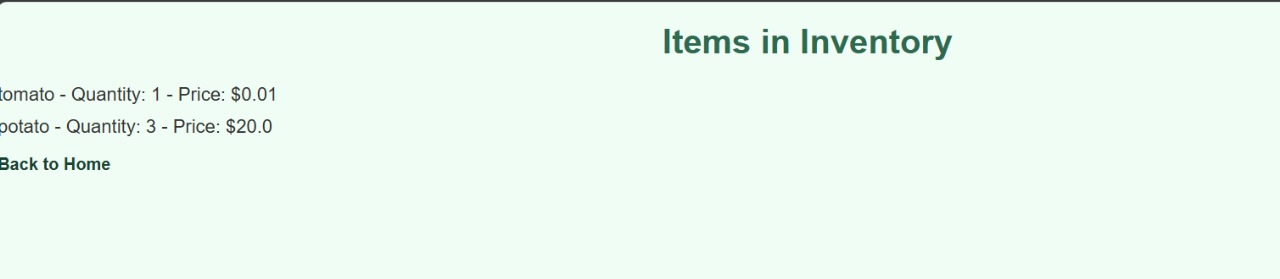
}

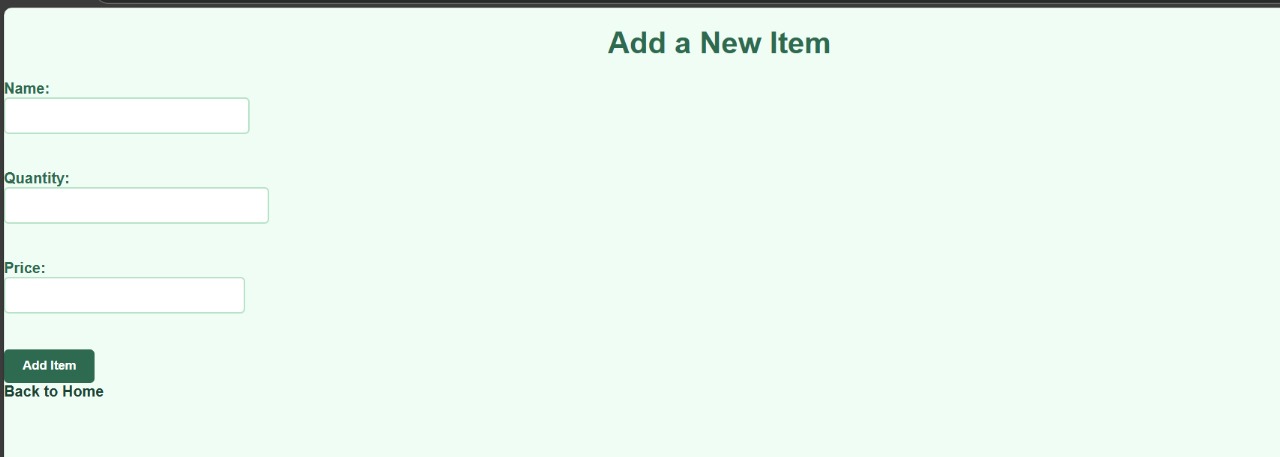
**OUTPUT :**

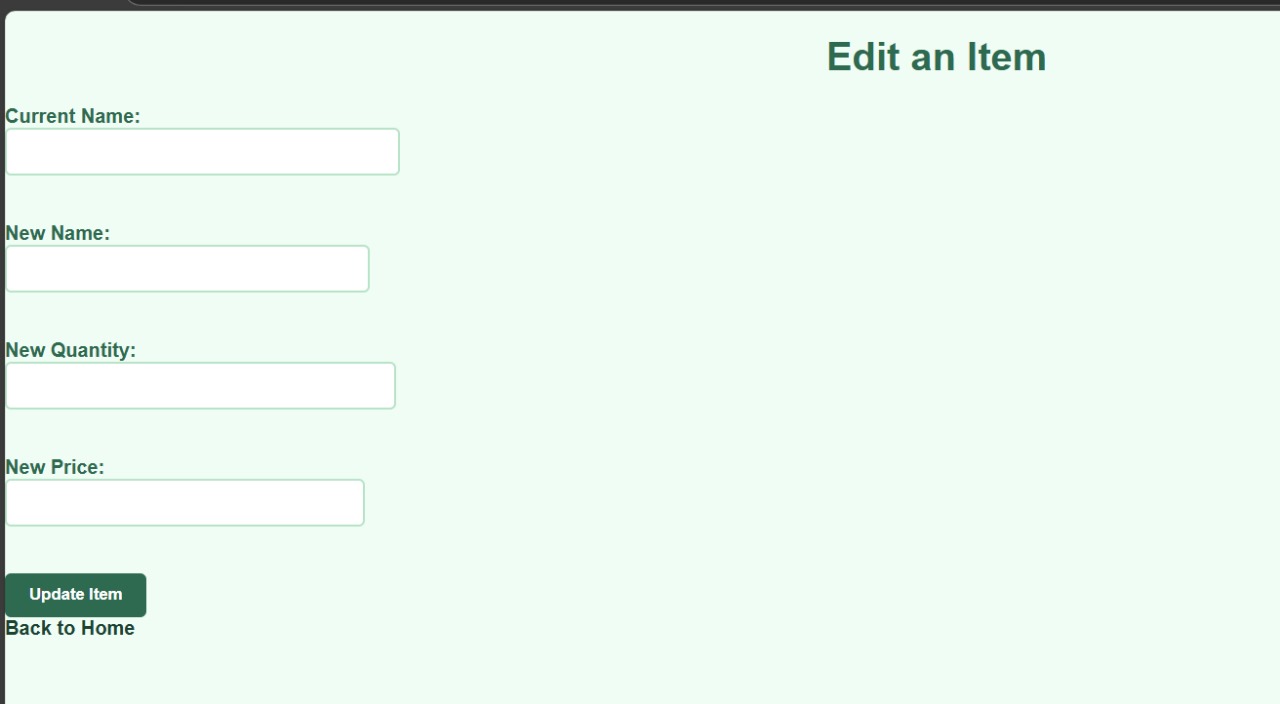
****

****

****

****

****

****

**CHAPTER 5**

**Result and Discussion:**

**Supermarket System in Java**

The implemented supermarket management system in Java provides basic functionalities such as product management, customer management, order processing, and simple product recommendations. The system allows the addition of products, tracking customer purchases, processing orders, and suggesting products based on previous categories purchased. It performs well for small datasets, providing real-time order processing and updating customer purchase history efficiently.

**System Functionality**

The system enables adding products, managing customer data, and processing orders with accurate calculations of total costs. It uses a simple recommendation engine that suggests products from the same category as previously purchased items. The user interface is basic, running via a command-line interface, but the system works effectively for handling basic supermarket management tasks.

**Strengths**

* **Modular Design**: The system is modular, with separate classes for managing products, customers, and orders. This allows easy future enhancements.
* **Real-Time Processing**: Orders are processed efficiently, and purchase histories are updated in real-time, ensuring responsiveness.
* **Simplicity**: The system is easy to use and understand, making it ideal for small-scale implementations and educational purposes.

**Limitations**

* **Basic Recommendation System**: The recommendation engine is limited to suggesting products based only on the category, lacking advanced personalization.
* **Inventory Management**: While stock availability is tracked, the system lacks detailed inventory management, such as tracking stock quantities.
* **Scalability**: The use of in-memory data structures is suitable for small datasets, but a database would be required for handling larger datasets.
* **Order Validation**: The system checks product availability but doesn’t ensure order validation like minimum order amounts or advanced constraints.

**Future Improvements**

* **Enhanced Recommendations**: Implementing machine learning algorithms or collaborative filtering could provide more personalized recommendations.
* **GUI or Web Interface**: Transitioning to a GUI or web interface using JavaFX or Spring Boot would improve usability.
* **Real-Time Inventory Management**: Implementing real-time stock updates and inventory validation would make the system more robust.
* **Database Integration**: Transitioning from in-memory storage to a database system like MySQL would allow the system to scale efficiently

**CHAPTER 6**

**CONCLUSION**

## The supermarket management system developed in Java provides a foundational solution for managing products, customers, and orders. While the system functions effectively for small-scale use, it has several areas for improvement

## . The product management and order processing features work well, and the simple recommendation engine provides basic product suggestions based on categories. However, the recommendation system lacks personalization, and the inventory management is limited to basic stock tracking without detailed quantity updates.

## For larger-scale applications, the system would need enhancements such as integrating a more advanced recommendation engine, transitioning to a database for data storage, and improving inventory management to track stock quantities in real-time. Additionally, developing a graphical user interface (GUI) or a web-based interface would significantly improve usability for end-users.

## In conclusion, while the current system offers essential functionalities, transitioning to a more scalable and personalized solution with real-time updates and better user interaction would be necessary to meet the demands of a real-world supermarket environment. With these improvements, the system can evolve into a robust, comprehensive solution capable of handling more complex supermarket management tasks.

**REFERENCES**

 **"Object-Oriented Programming in Java"**

* Authors: R. S. V. R. K. Srinivas
* Publisher: Vikas Publishing
* Description: This book provides an understanding of object-oriented programming concepts like classes, objects, inheritance, polymorphism, and encapsulation, which are crucial for building a supermarket management system in Java.

 **"Java: The Complete Reference"**

* Author: Herbert Schildt
* Publisher: McGraw-Hill
* Description: A comprehensive guide on Java programming language, covering all essential aspects including classes, exception handling, and GUI development in Java, which can be used in the development of a supermarket management system.

 **"Java Programming: From Problem Analysis to Program Design"**

* Authors: D. S. Malik
* Publisher: Cengage Learning
* Description: A detailed guide to object-oriented programming in Java, with examples and case studies to help in developing systems like a supermarket management application.

 **"Beginning Java 8"**

* Author: Kishori Sharan
* Publisher: Apress
* Description: This book helps readers understand how to use Java 8 features, including lambda expressions and streams, which could be useful for handling product lists and customer data in an efficient manner in a supermarket management system.

 **"Java for Students: Object-Oriented Programming and Data Structures"**

* Author: G. Michael Schneider
* Publisher: W.W. Norton & Company
* Description: Provides a solid foundation in Java programming concepts, focusing on data structures and object-oriented design which are crucial in building a supermarket system.

 **Online Resources and Tutorials**:

* GeeksforGeeks - Java OOP Concepts  
  Description: Offers tutorials on core Java concepts including classes, objects, inheritance, polymorphism, and more, all of which are fundamental for developing a system like the supermarket management system.
* TutorialsPoint - Java  
  Description: A complete guide for learning Java, covering both basic and advanced topics necessary for building Java-based applications, such as the supermarket management system.