**MINISTRY OF EDUCATION AND SCIENCE OF**

**THE KYRGYZ REPUBLIC**

**KYRGYZ STATE TECHNICAL UNIVERSITY named after I.Razzakov**

**Institute of Information Technology**

**Department of Computer Systems Software**

**Course work**

By discipline: Object-oriented programming on the topic:

"Grocery store management system"

Performed by: Shumkarbekov Sultan SE (English)-1-21

Checked by: Musabaev E. B.

Bishkek 2024

**Tables of contents**

[**Introduction** 3](#_Toc167420464)

[**Goal** 3](#_Toc167420465)

[**Analyzing and forming requirements** 3](#_Toc167420466)

[**User requirements** 3](#_Toc167420467)

[**Functional requirements:** 3](#_Toc167420468)

[**Non-functional requirements:** 4](#_Toc167420469)

[**Use case** 5](#_Toc167420470)

[**Class diagram** 5](#_Toc167420471)

[The system contains 10 classes: 5](#_Toc167420473)

[**Description of classes and their purpose** 6](#_Toc167420474)

[1. Class Unit 6](#_Toc167420475)

[2. Class Product 6](#_Toc167420476)

[3. Class ShoppingCart 6](#_Toc167420477)

[4. Class Inventory: 6](#_Toc167420478)

[5. Class User 7](#_Toc167420479)

[6. Class Admin 8](#_Toc167420480)

[7. Class SalesReport 8](#_Toc167420481)

[8. Class Salesperson 8](#_Toc167420482)

[9. Class ReportGenerator 8](#_Toc167420483)

[10. Class StockAlert 9](#_Toc167420484)

[11. Class Menu 9](#_Toc167420485)

[12. Class OptionProcessor: 9](#_Toc167420486)

[**Code** 10](#_Toc167420487)

[An example of the work 21](#_Toc167420488)

[Conclusion 26](#_Toc167420489)

# **Introduction**

The proposed system aims to streamline the management of a product store by providing an efficient and user-friendly platform for both administrators and customers. By automating key processes such as inventory management, sales tracking, and customer transactions, the system aims to enhance operational efficiency and improve the overall shopping experience.

# **Goal**

The goal of the system is to provide administrators with tools to manage product inventory, track sales performance, and make informed decisions about pricing and stock levels. For customers, the goal is to create a seamless shopping experience, allowing them to easily browse products, add items to their cart, and complete purchases with minimal effort.

# **Analyzing and forming requirements**

## **User requirements**

System users: administrator

The administrator must be able to perform the following functions:

1. Login

2. Add Quantity to a Product

3. Change Price of a Product

4. View Sales Report

5. Add a Product

6. Return Main Menu (log out)

7. View all products

The user (client) must be able to perform the following functions:

1. Select a product

2. Enter quantity

3. Cart view

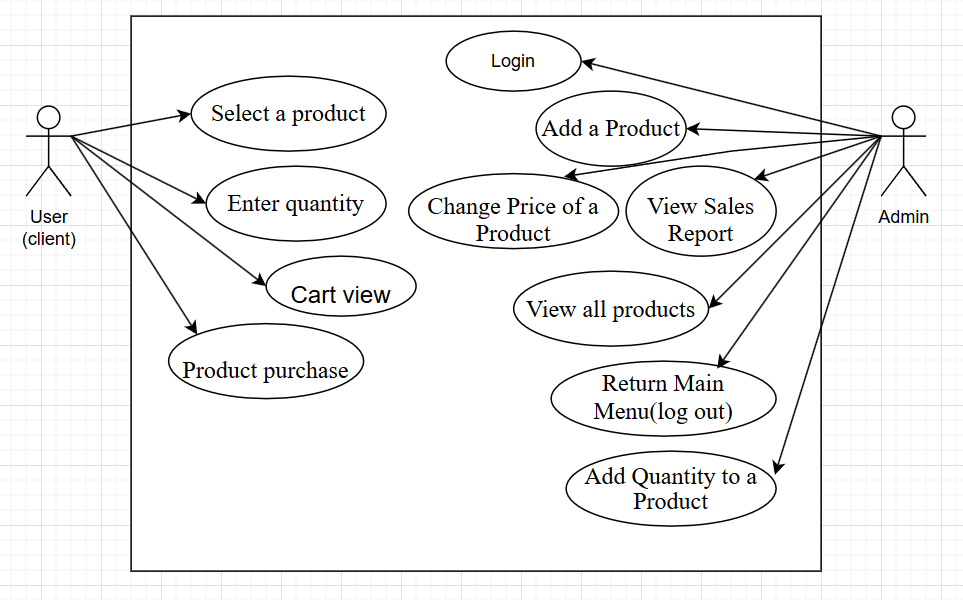
4. Product purchase

## **Functional requirements:**

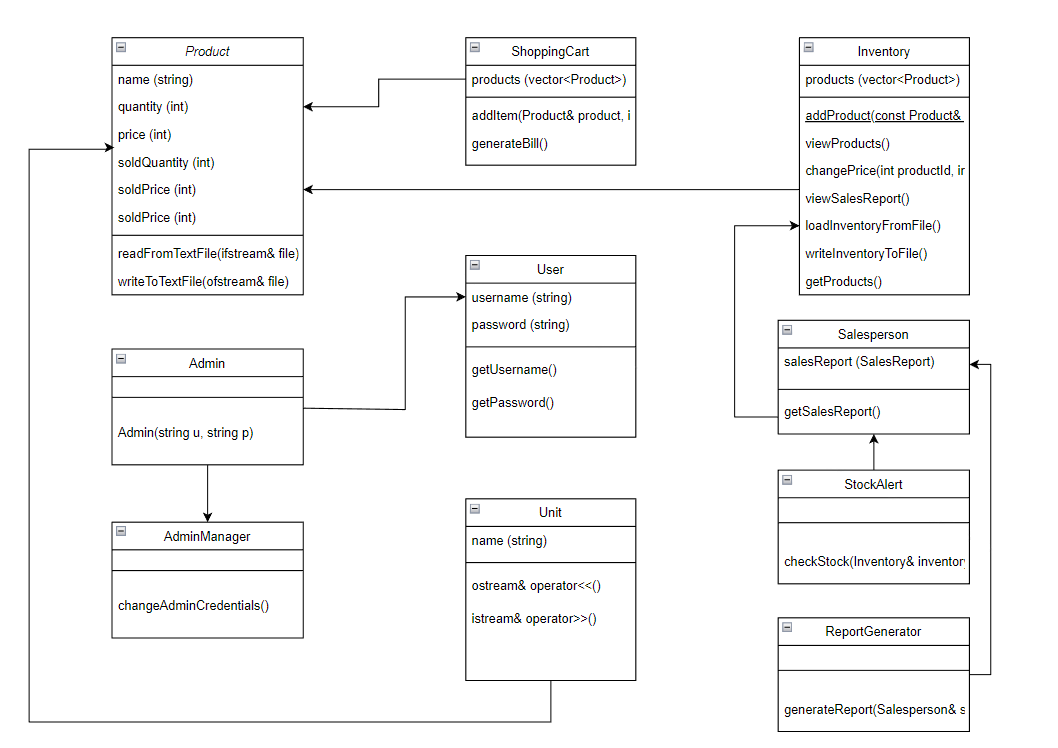
1. Authorization of the administrator:
2. The administrator must be able to log in using his username and password.
3. Adding the quantity of the product:
4. The administrator should be able to increase the quantity of a specific product in the warehouse.
5. Change in the price of the product:
6. The administrator must be able to change the price of a specific product.
7. Viewing the sales report:
8. The administrator should be able to view the sales report, which includes information about the number of items sold and the total amount of sales.
9. Adding a new product:
10. The administrator should be able to add new products to the system by specifying the name, quantity and price.
11. Return to the main menu:
12. The administrator should be able to exit the administrator mode and return to the main menu.
13. View all products:
14. The administrator should be able to view a list of all available products.
15. Product selection by the customer:
16. The customer must be able to select products from the list of available products.
17. Entering the quantity of the product by the customer:
18. The customer must be able to specify the quantity of the selected product.
19. Viewing the shopping cart:
20. The customer should be able to view the items added to the cart.
21. Purchase of goods:
22. The customer should be able to complete the purchase by paying for the selected items.

## **Non-functional requirements:**

1. Performance: The system must be fast and responsive to ensure user convenience when working with it.
2. The system must ensure the security of user data and goods stored in the system.
3. Reliability: The system must be reliable and stable to avoid data loss and malfunction.
4. Ease of use: The system interface should be intuitive and user-friendly for users of all categories.
5. Availability: The system should be available to users at any time of the day without interruptions.

**Use case**  


**Class diagram**

****

The system contains 10 classes: Unit, Product, ShoppingCart, Inventory, User, Admin, SalesReport, Salesperson, ReportGenerator, StockAlert, Menu, AdminManager, and OptionProcessor

# **Description of classes and their purpose**

## 1. Class Unit

Class Assignment: Unit is a class representing a basic unit. It has a single member variable name to store the name of the unit. The class provides a constructor to initialize the name, and includes overloaded << and >> operators for stream input and output.

## 2. Class Product

Class Assignment: The Product class represents a product in the inventory. It includes attributes such as name, quantity, price, soldQuantity, and soldPrice to manage product information and sales data. It provides a constructor to initialize these attributes for a product.

Description of the methods of the class:

Product(string n, int q, int p): Constructor to initialize the name, quantity, and price of the product. It also initializes soldQuantity and soldPrice to zero.

## 3. Class ShoppingCart

Class Assignment: The ShoppingCart class manages the items added by a customer for purchase. It allows adding items to the cart, generating a bill, and applying discounts based on the total amount.

Description of the methods of the class:

void addItem(Product& product, int quantity): Adds a product with the specified quantity to the shopping cart. It updates the product's quantity, soldQuantity, and soldPrice accordingly.

void generateBill(): Generates a bill for the items in the shopping cart. It calculates the total amount, applies discounts if applicable, and prompts for payment and change due.

## 4. Class Inventory:

Class Assignment: The Inventory class manages the list of products available for sale. It allows adding new products, viewing the product list, updating quantities and prices, viewing sales reports, and removing products from inventory.

Description of the methods of the class:

void addProduct(const Product& product): Adds a new product to the inventory.

void viewProducts(): Displays the list of products in the inventory with their IDs, names, quantities, and prices.

void addQuantity(int productId, int quantity): Increases the quantity of a product in the inventory.

void changePrice(int productId, int newPrice): Updates the price of a product in the inventory.

void viewSalesReport(): Displays the sales report showing the quantity sold and total sales for each product.

void removeProduct(int productId): Removes a product from the inventory based on its ID.

vector<Product>& getProducts(): Returns a reference to the vector of products in the inventory.

## 5. Class User

Class Assignment: The User class represents a user of the system. It includes attributes for username and password.

Description of the methods of the class:

User(string u, string p): Constructor to initialize the username and password of the user.

string getUsername() const: Returns the username of the user.

string getPassword() const: Returns the password of the user.

void setUsername(const string& u): Sets the username of the user.

void setPassword(const string& p): Sets the password of the user.

## 6. Class Admin

Class Assignment: The Admin class represents an administrator user of the system. It inherits from the User class and provides specific functionalities for administrative tasks.

Description of the methods of the class: Inherits methods from the User class.

## 7. Class SalesReport

Class Assignment: The SalesReport class manages the sales data of products. It allows adding sold products and viewing sales reports.

Description of the methods of the class:

void addSoldProduct(const Product& product): Adds a product to the list of sold products.

void viewSalesReport(): Displays the sales report showing the quantity sold and total sales for each product.

## 8. Class Salesperson

Class Assignment: The Salesperson class represents a salesperson user of the system. It inherits from the User class and includes a SalesReport object to manage their sales data.

Description of the methods of the class: Inherits methods from the User class.

## 9. Class ReportGenerator

Class Assignment: The ReportGenerator class generates reports for salespersons. It provides a static method to generate sales reports for a given salesperson.

Description of the methods of the class:

static void generateReport(Salesperson& salesperson): Generates a sales report for the specified salesperson.

## 10. Class StockAlert

Class Assignment: The StockAlert class checks the stock levels of products in the inventory and alerts if any product's quantity is low.

Description of the methods of the class:

static void checkStock(Inventory& inventory): Checks the stock levels of products in the inventory and prints a low stock alert if necessary.

## 11. Class Menu

Class Assignment: The Menu class provides static methods to display different menus for user interaction.

Description of the methods of the class:

static void showMainMenu(): Displays the main menu options.

static void showAdminMenu(): Displays the admin menu options.

## 12. Class OptionProcessor:

Class Assignment: The OptionProcessor class processes user options selected from the menus.

Description of the methods of the class:

static void processAdminOption(int option, Inventory& inventory, string& username, string& password): Processes the selected admin option and performs corresponding actions on the inventory.

**Code**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

class Product {

public:

string name;

int quantity;

int price;

int soldQuantity;

int soldPrice;

Product(string n, int q, int p) : name(n), quantity(q), price(p), soldQuantity(0), soldPrice(0) {}

};

class ShoppingCart {

private:

vector<Product> products;

public:

void addItem(Product& product, int quantity) {

product.quantity -= quantity;

product.soldQuantity += quantity;

product.soldPrice += quantity \* product.price;

products.push\_back(product);

}

void generateBill() {

int total = 0;

cout << "\nYour Cart:\n";

cout << "Item\t\tQuantity\tTotal\n";

for (Product& product : products) {

cout << product.name << "\t\t" << product.soldQuantity << "\t\t" << product.soldPrice << endl;

total += product.soldPrice;

}

// Discount calculation

if (total >= 100 && total < 200) {

cout << "\nTotal Amount: " << total << endl;

cout << "Discount 5%" << endl;

total \*= 0.95;

}

else if (total >= 200 && total < 300) {

cout << "\nTotal Amount: " << total << endl;

cout << "Discount 10%" << endl;

total \*= 0.9;

}

else if (total >= 300 && total < 400) {

cout << "\nTotal Amount: " << total << endl;

cout << "Discount 15%" << endl;

total \*= 0.85;

}

else if (total >= 400 && total < 500) {

cout << "\nTotal Amount: " << total << endl;

cout << "Discount 20%" << endl;

total \*= 0.8;

}

else if (total >= 500) {

cout << "\nTotal Amount: " << total << endl;

cout << "Discount 25%" << endl;

total \*= 0.75;

}

cout << "\nTotal Amount after discount: " << total << endl;

int collected;

cout << "Amount given by the customer: ";

cin >> collected;

cout << "Change due: " << collected - total << endl;

cout << "----------------------------------------------\nThanks for shopping with us! Hope to see you again!";

}

};

class Inventory {

private:

vector<Product> products;

public:

void addProduct(const Product& product) {

products.push\_back(product);

}

void viewProducts() {

cout << "\nProduct List:\n";

cout << "ID\tName\t\tQuantity\tPrice\n";

for (size\_t i = 0; i < products.size(); ++i) {

cout << i + 1 << "\t" << products[i].name << "\t\t" << products[i].quantity << "\t\t" << products[i].price << endl;

}

}

void addQuantity(int productId, int quantity) {

if (productId >= 1 && productId <= products.size()) {

products[productId - 1].quantity += quantity;

}

}

void changePrice(int productId, int newPrice) {

if (productId >= 1 && productId <= products.size()) {

products[productId - 1].price = newPrice;

}

}

void viewSalesReport() {

int totalSales = 0;

cout << "\nSales Report:\n";

cout << "Product\t\tQuantity\tPrice\n";

for (Product& product : products) {

product.soldPrice = product.soldQuantity \* product.price;

totalSales += product.soldPrice;

cout << product.name << "\t\t" << product.soldQuantity << "\t\t" << product.soldPrice << endl;

}

cout << "------------------------------------\nTotal\t\t\t\t" << totalSales << endl;

}

void removeProduct(int productId) {

if (productId >= 1 && productId <= products.size()) {

products.erase(products.begin() + productId - 1);

}

}

vector<Product>& getProducts() { return products; }

};

class User {

private:

string username;

string password;

public:

User(string u, string p) : username(u), password(p) {}

string getUsername() const { return username; }

string getPassword() const { return password; }

void setUsername(const string& u) { username = u; }

void setPassword(const string& p) { password = p; }

};

class Admin : public User {

public:

Admin(string u, string p) : User(u, p) {}

};

class SalesReport {

private:

vector<Product> soldProducts;

public:

void addSoldProduct(const Product& product) {

soldProducts.push\_back(product);

}

void viewSalesReport() {

int totalSales = 0;

cout << "\nSales Report:\n";

cout << "Product\t\tQuantity\tPrice\n";

for (const Product& product : soldProducts) {

int productPrice = product.price \* product.soldQuantity;

totalSales += productPrice;

cout << product.name << "\t\t" << product.soldQuantity << "\t\t" << productPrice << endl;

}

cout << "------------------------------------\nTotal\t\t\t\t" << totalSales << endl;

}

};

class Salesperson : public User {

private:

SalesReport salesReport;

public:

Salesperson(string u, string p) : User(u, p) {}

SalesReport& getSalesReport() { return salesReport; }

};

class ReportGenerator {

public:

static void generateReport(Salesperson& salesperson) {

salesperson.getSalesReport().viewSalesReport();

}

};

class StockAlert {

public:

static void checkStock(Inventory& inventory) {

bool isLowStock = false;

cout << "\n\nStock Alert!\n";

for (Product& product : inventory.getProducts()) {

if (product.quantity < 5) {

isLowStock = true;

cout << "Quantity of " << product.name << " is less than 5" << endl;

}

}

if (!isLowStock) {

cout << "No low stock items." << endl;

}

}

};

class Menu {

public:

static void showMainMenu() {

cout << "Main Menu\nFor admin Enter 1\nFor user Enter 2\nFor exiting software Enter 3\n: ";

}

static void showAdminMenu() {

cout << "\nAdmin Menu\n";

cout << "1. View Products\n";

cout << "2. Add Quantity to a Product\n";

cout << "3. Change Price of a Product\n";

cout << "4. View Sales Report\n";

cout << "5. Add a Product\n";

cout << "6. Main Menu\n";

cout << "Enter your choice: ";

}

};

class OptionProcessor {

public:

static void processAdminOption(int option, Inventory& inventory, string& username, string& password) {

switch (option) {

case 0:

case 6:

int productIdToDelete;

cout << "Enter product ID to delete: ";

cin >> productIdToDelete;

inventory.removeProduct(productIdToDelete);

break;

case 1:

inventory.viewProducts();

break;

case 2:

int productId, quantity;

cout << "Enter product ID: ";

cin >> productId;

cout << "Enter quantity to add: ";

cin >> quantity;

inventory.addQuantity(productId, quantity);

break;

case 3:

int newPrice;

cout << "Enter product ID: ";

cin >> productId;

cout << "Enter new price: ";

cin >> newPrice;

inventory.changePrice(productId, newPrice);

break;

case 4:

inventory.viewSalesReport();

break;

case 5:

string productName;

int productPrice, productQuantity;

cout << "Enter product name: ";

cin >> productName;

cout << "Enter product price: ";

cin >> productPrice;

cout << "Enter product quantity: ";

cin >> productQuantity;

inventory.addProduct(Product(productName, productQuantity, productPrice));

break;

}

}

};

int main() {

Inventory inventory;

inventory.addProduct(Product("Bread", 10, 10));

inventory.addProduct(Product("Chips", 10, 15));

inventory.addProduct(Product("Biscuit", 10, 5));

inventory.addProduct(Product("Juice", 10, 30));

inventory.addProduct(Product("Drinks", 10, 50));

inventory.addProduct(Product("Rice", 10, 200));

inventory.addProduct(Product("Grain", 10, 100));

inventory.addProduct(Product("Noodles", 10, 20));

inventory.addProduct(Product("Nuggets", 10, 125));

inventory.addProduct(Product("Bubble", 10, 2));

string username, password;

int option;

do {

Menu::showMainMenu();

cin >> option;

if (option == 2) {

ShoppingCart cart;

int productId, quantity;

inventory.viewProducts();

cout << "Enter product ID: ";

cin >> productId;

cout << "Enter quantity: ";

cin >> quantity;

if (productId > 0 && productId <= inventory.getProducts().size() && quantity > 0 &&

quantity <= inventory.getProducts()[productId - 1].quantity) {

Product& selectedProduct = inventory.getProducts()[productId - 1];

cart.addItem(selectedProduct, quantity);

}

else {

cout << "Invalid product ID or quantity. Please try again." << endl;

}

cart.generateBill();

}

else if (option == 1) {

cout << "Enter username: ";

cin >> username;

cout << "Enter password: ";

cin >> password;

if (username == "admin" && password == "123") {

int adminOption;

do {

Menu::showAdminMenu();

cin >> adminOption;

OptionProcessor::processAdminOption(adminOption, inventory, username, password);

} while (adminOption != 6);

}

else {

cout << "Invalid username or password. Please try again." << endl;

}

}

StockAlert::checkStock(inventory);

} while (option != 3);

return 0;

}

# **An example of the work**

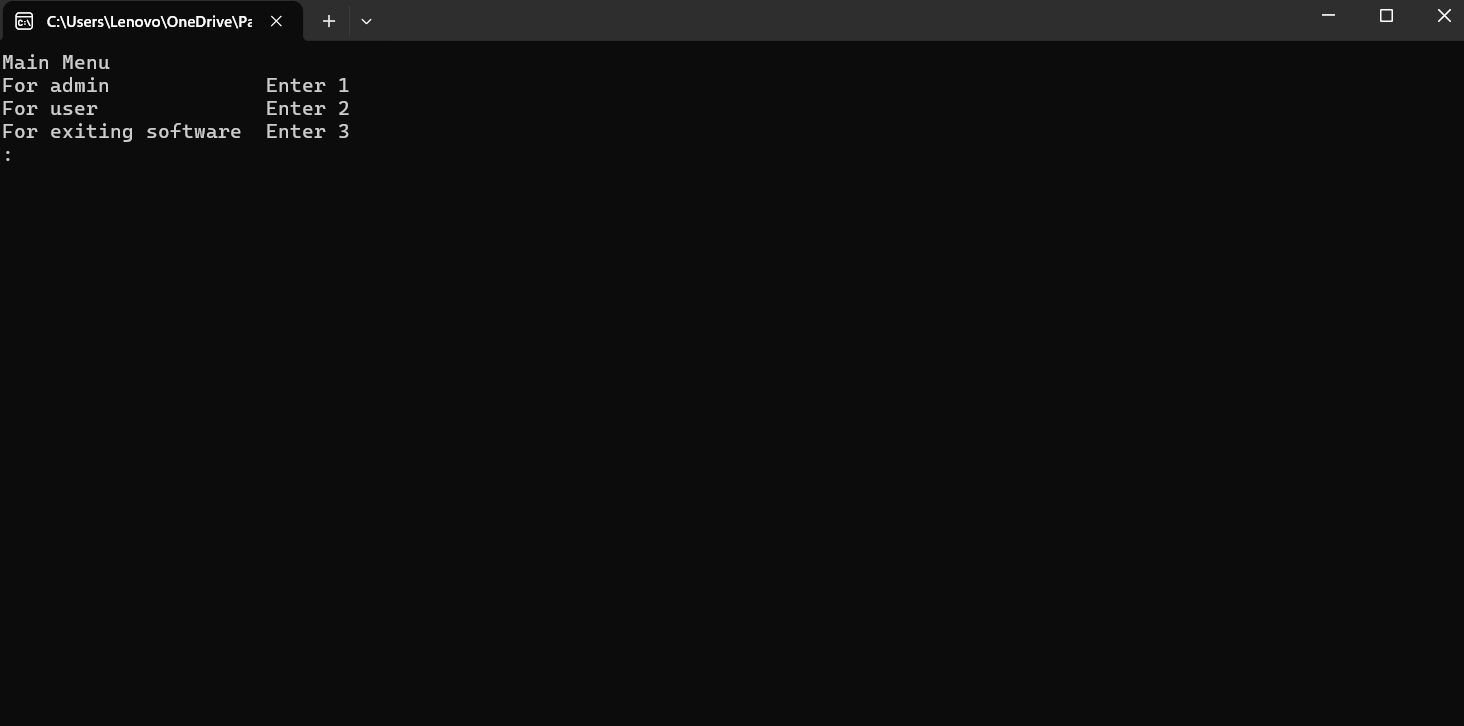


Figure No.1. Main menu

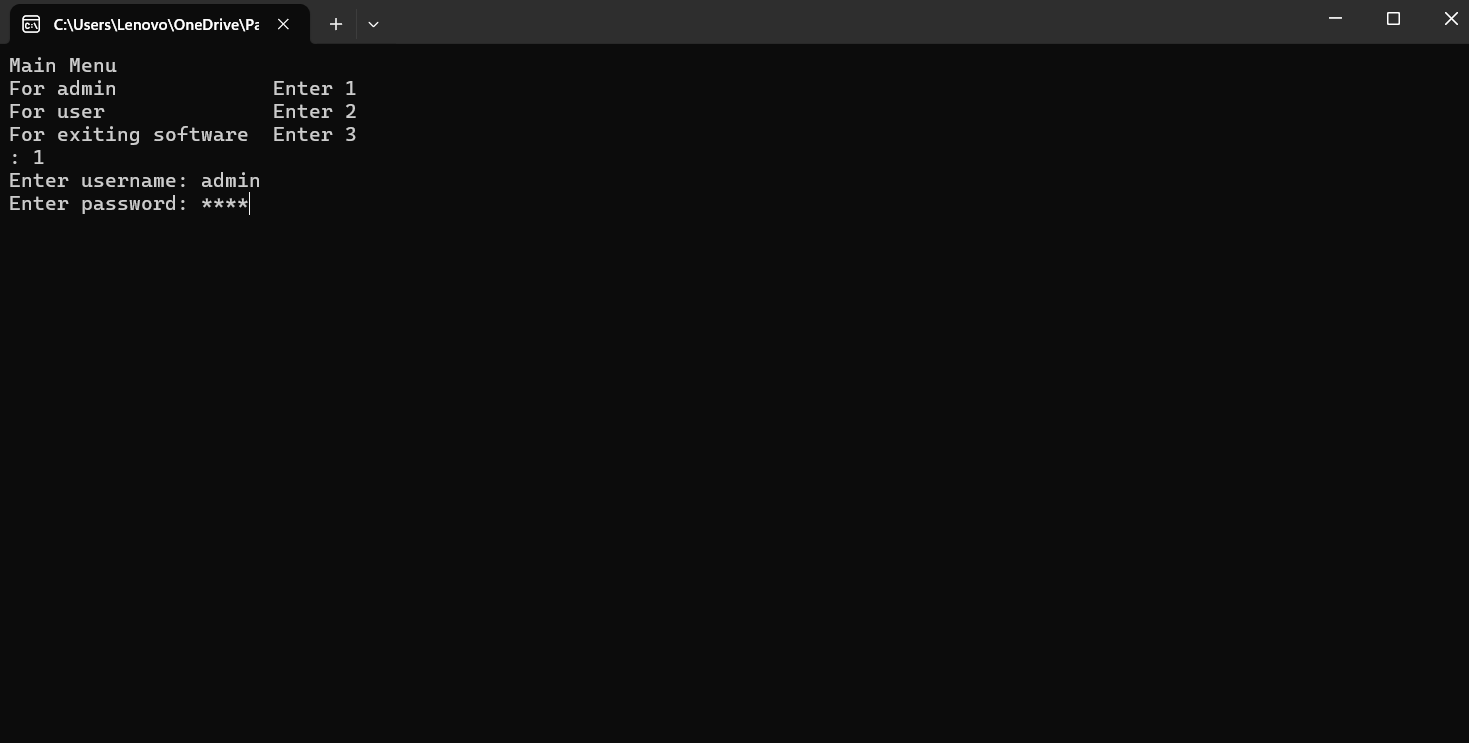


Figure No.2. We choose a role, for example, an administrator(1), and in order to have access to its functions, you need to enter a password and login to verify your identity.

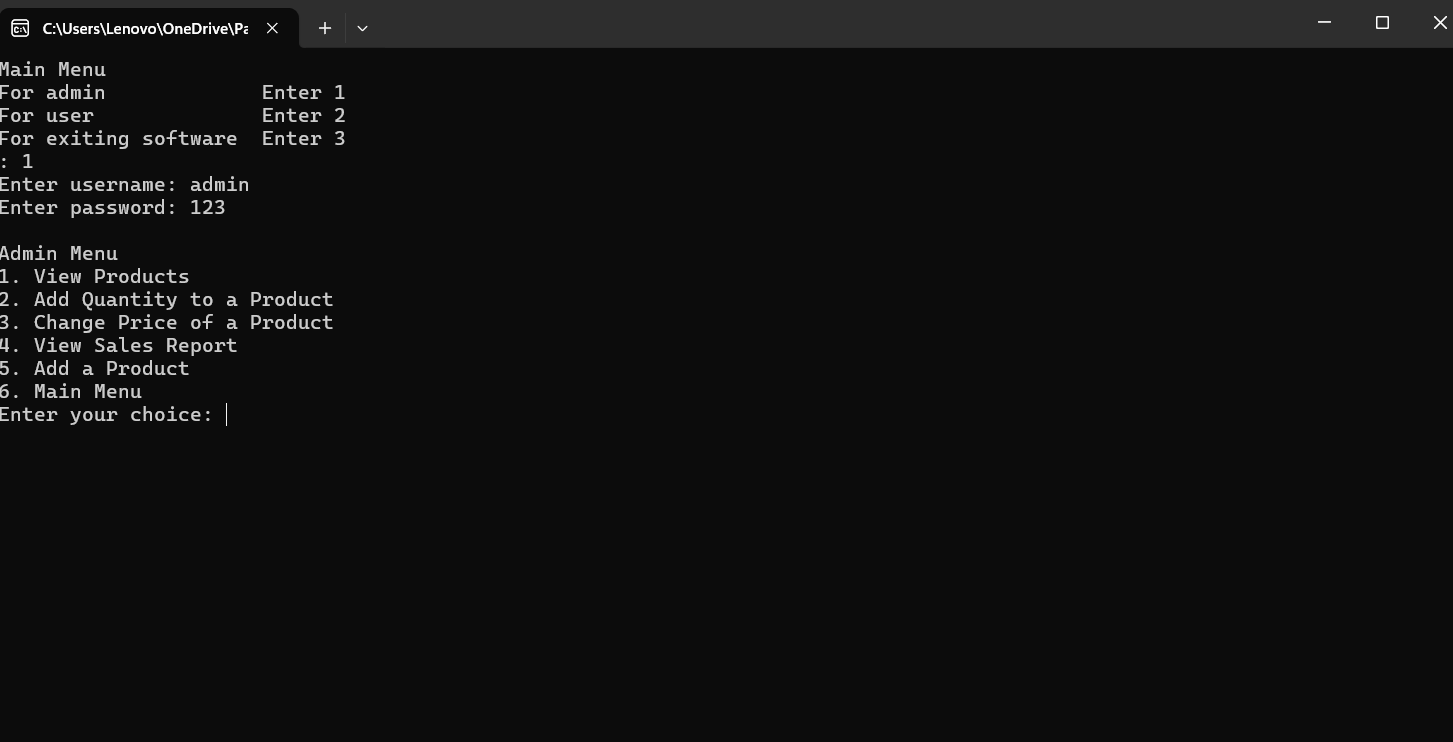


Figure No.3. After successfully entering the username and password, we have access to the admin functions

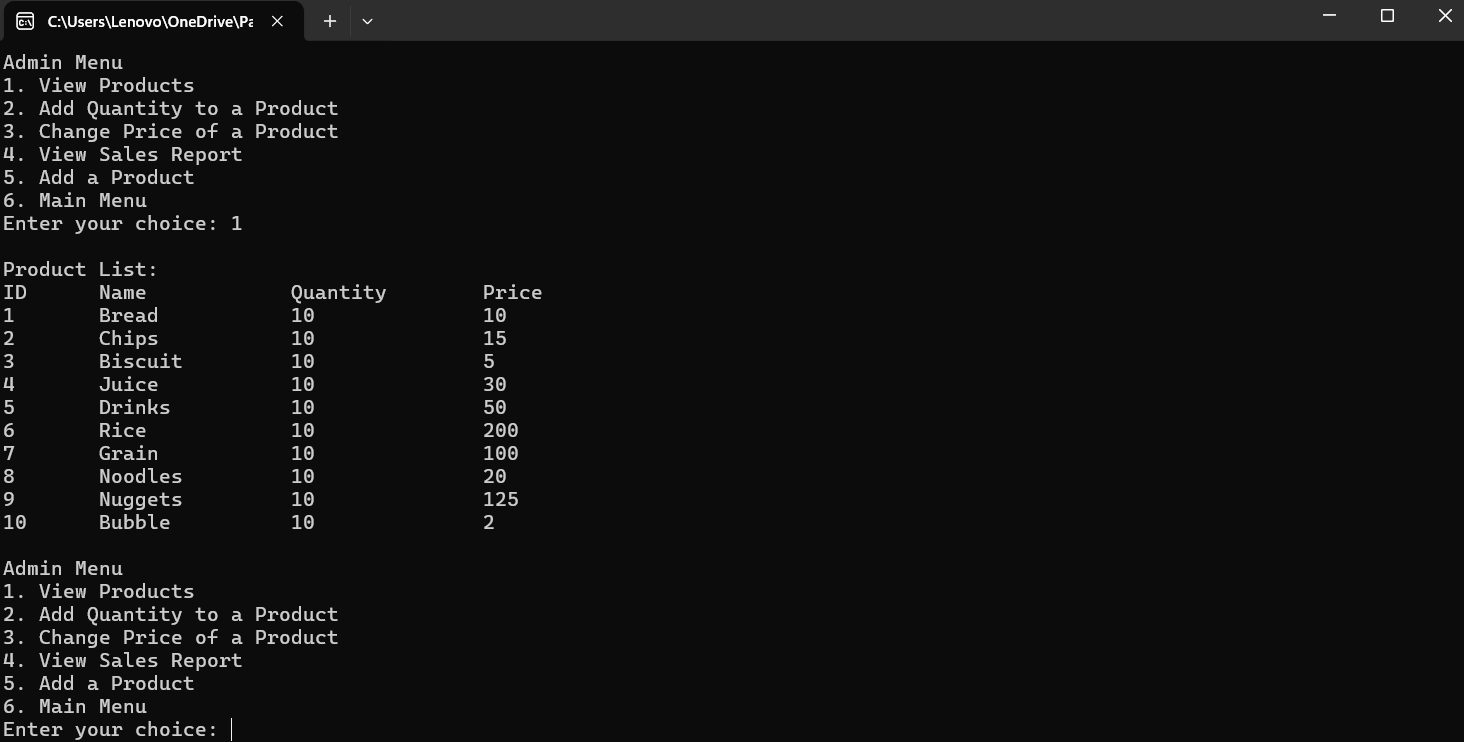


Figure No.4. View all products

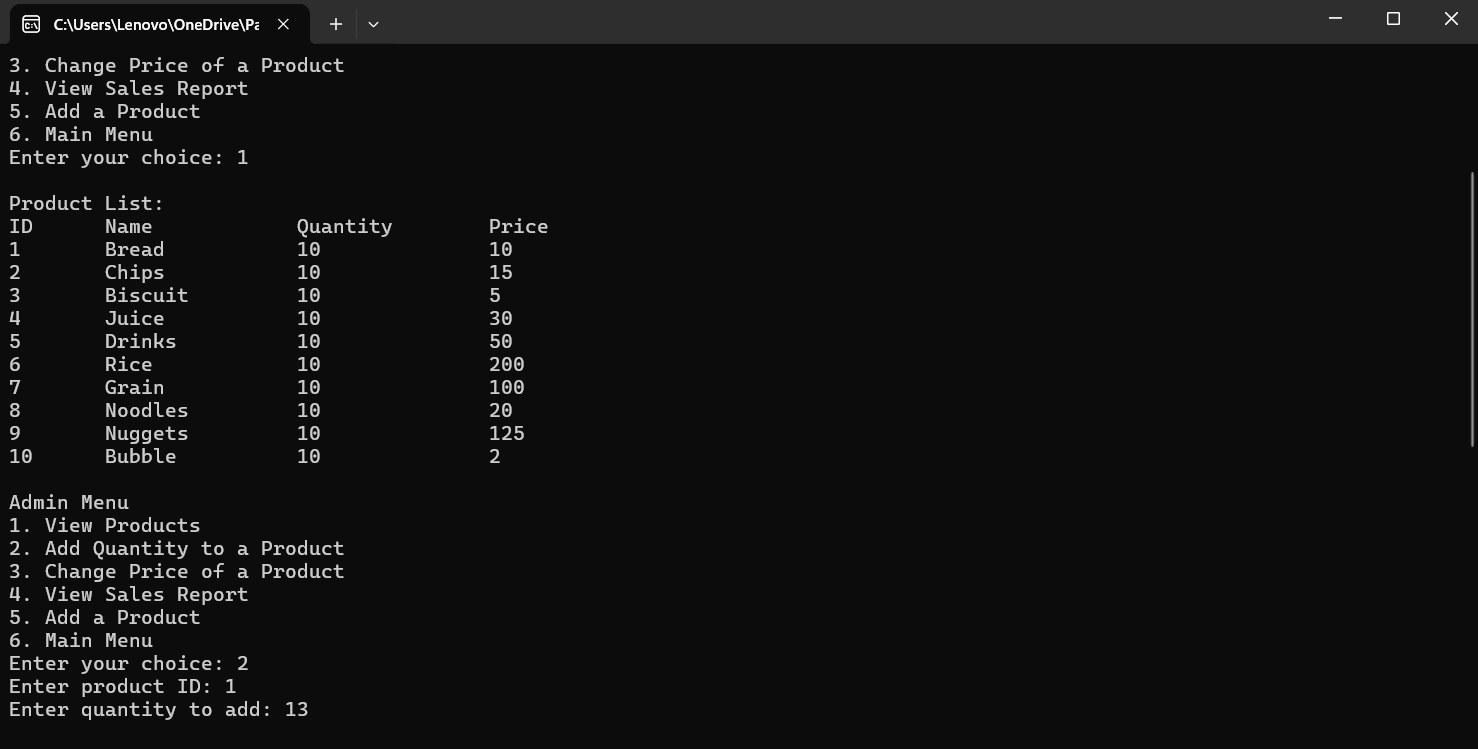


Figure No.5. Change quantity a product

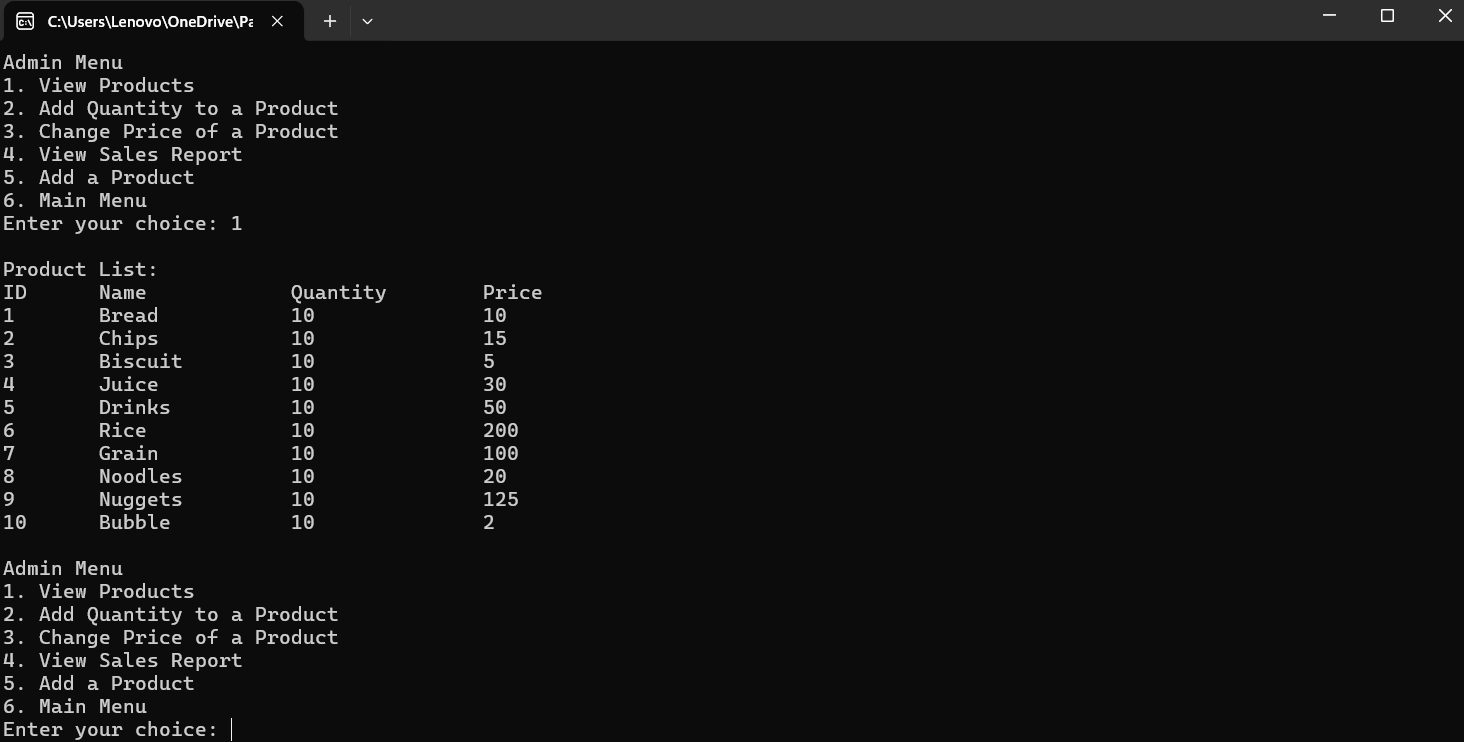


Figure No.6. Change price a product

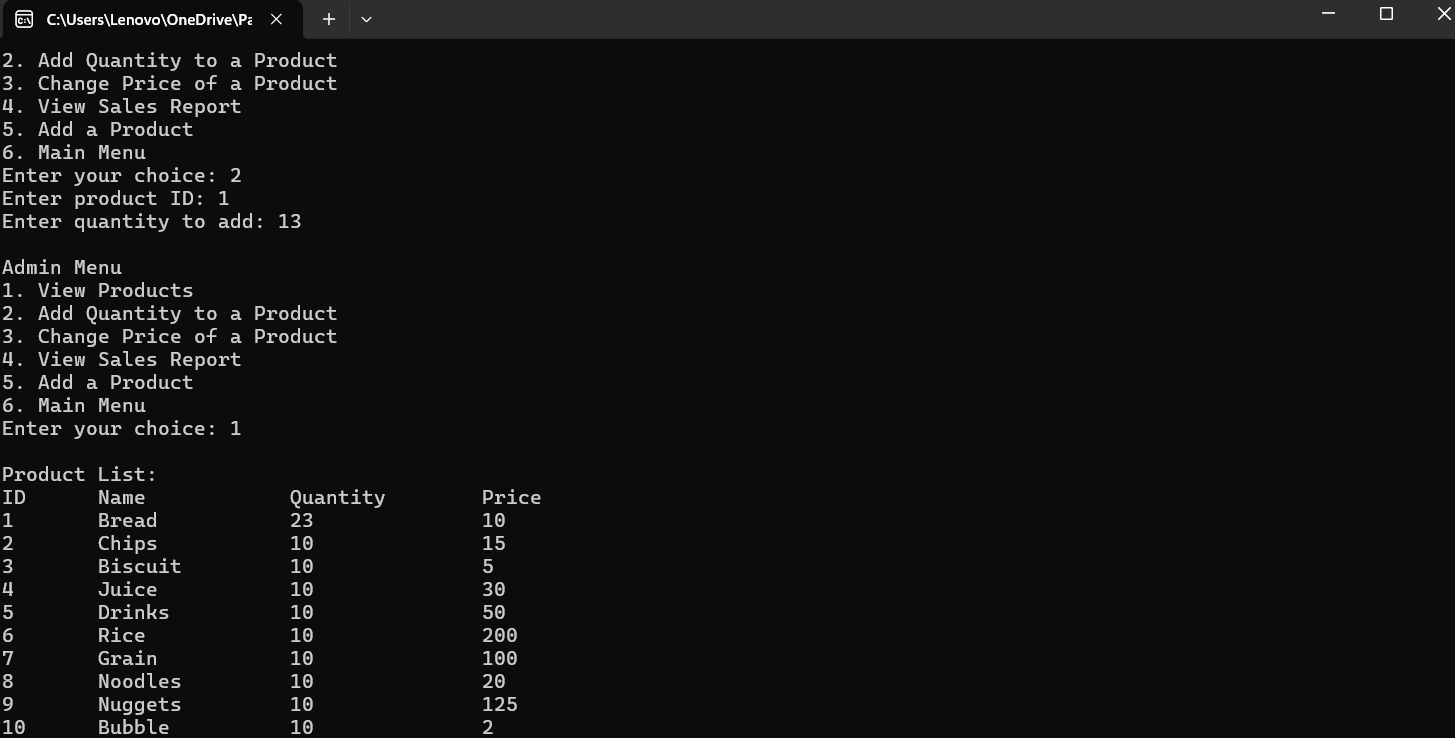
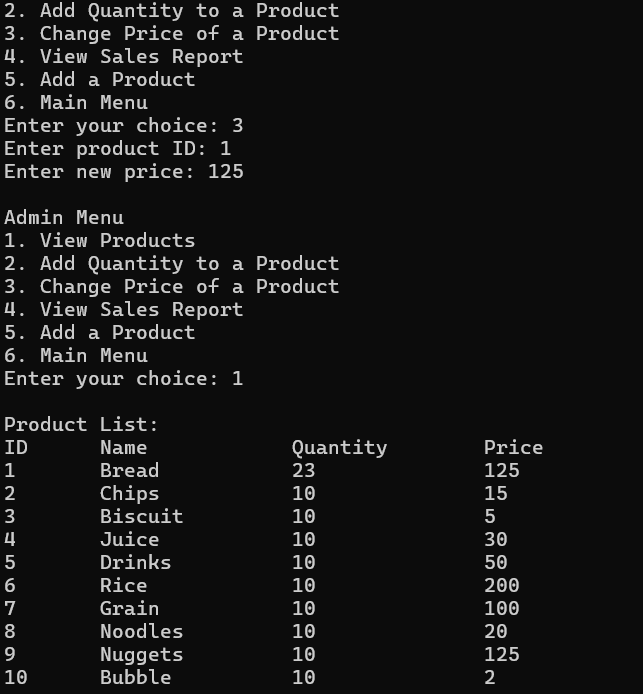
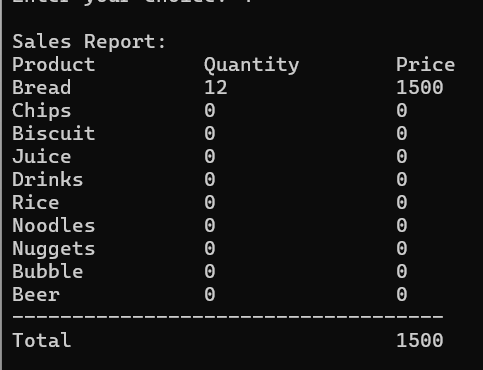


Figure No.7. Result after change quantity a product

  
Figure No.8. Result after change price a product

  
Figure No.9. View sales perort

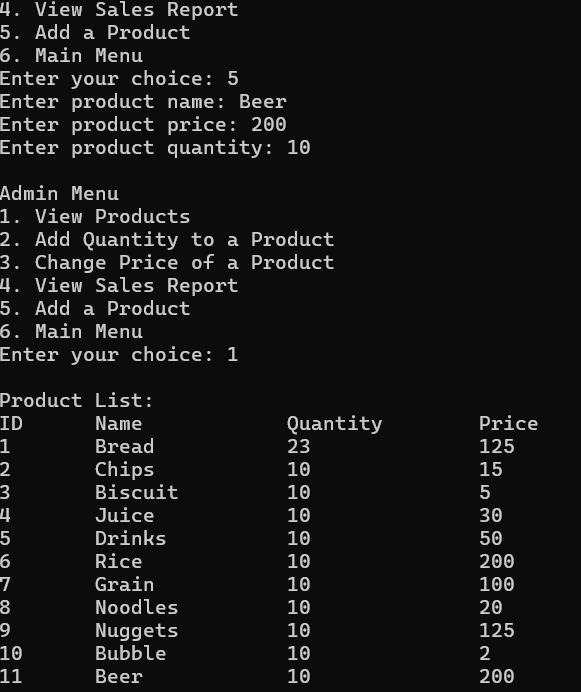


Figure No.10. Add new product



Figure No.11. Sale of product

**Conclusion**

In conclusion, this coursework has provided a comprehensive overview and practical implementation of a simple inventory management system in C++. Through the design and development of various classes such as Product, ShoppingCart, Inventory, User, Admin, SalesReport, Salesperson, ReportGenerator, StockAlert, Menu, and OptionProcessor, we have successfully created a modular and efficient system for managing products, sales, users, and administrative tasks.

The Product class encapsulates product attributes and functionalities, while the ShoppingCart class facilitates the management of customer purchases and bill generation. The Inventory class enables the administration of product listings, including addition, removal, and modification of product details. User management is handled through the User, Admin, and Salesperson classes, which provide authentication and authorization functionalities.

Sales data tracking and reporting are facilitated by the SalesReport and ReportGenerator classes, allowing for insights into product performance and salesperson productivity. Additionally, the StockAlert class enhances inventory management by providing timely notifications for low stock items.

The Menu and OptionProcessor classes offer a user-friendly interface for interacting with the system, simplifying navigation and operation. Overall, the coursework has demonstrated the practical application of object-oriented programming principles in building a functional inventory management system, laying a strong foundation for further learning and development in software engineering and related fields.