TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

Report

On

Smartshop: An Online Shopping System using C++

Submitted by:

Agrima Shahi(THA080BEI004)

Kripa Timalsena (THA080BEI023)

Samiksha Dhakal (THA080BEI038)

Shristi Mallik (THA080BEI044)

Submitted to:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

6th December, 2024

COPYRIGHT

ACKNOWLEDGEMENT

We would like to express our sincere gratitude and appreciation to everyone who contributed, directly or indirectly, to the initiation and completion of our project. We are thankful to Thapathali Campus, IOE for providing us with the opportunity to put our knowledge and skills in C++ programming to the test. This challenge allowed us to come up with new ideas and methods for designing, problem-analyzing and solving.This project has led us to be more enthusiastic and motivated in our pursuit of research goals. We would like to express our gratitude for the exceptional dedication and contributions made by each team member, and ensure that they are all equally appreciated. We express our gratitude to Er. Mukunda Raj Joshi, our supervisor, for his valuable assistance and support throughout the project. Finally, upon the completion of our project, we would like to express our gratitude to the authors and programming libraries we referenced.

Agrima Shahi(THA080BEI004)
Kripa Timalsena (THA080BEI023)
Samiksha Dhakal (THA080BEI038)
Shristi Mallik (THA080BEI044)

ABSTRACT

In the rapidly evolving digital marketplace, the need for efficient, user-friendly, and secure online shopping platforms has never been more critical. This proposal outlines the development of an advanced Online Shopping System using C++, designed to enhance user experience and streamline transaction processes. The system aims to provide a comprehensive solution for managing product listings, handling customer orders, and facilitating secure payment transactions.

The proposed system will incorporate a robust back-end architecture to ensure high performance and reliability. Key features will include an intuitive user interface for seamless navigation, real-time inventory management, personalized customer profiles, and secure authentication mechanisms to protect user data. The C++ programming language, known for its efficiency and control over system resources, will be employed to develop core functionalities and optimize system performance.

This project will address common challenges in online shopping, such as scalability, data integrity, and system security, by leveraging C++'s advanced capabilities. The end goal is to deliver a scalable and maintainable online shopping system that meets the needs of both end-users and administrators, ultimately contributing to a more streamlined and satisfying online shopping experience.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

EDI	Electronic Data Interchange
e-Commerce	Electronic Commerce
VS code	Visual Studio Code
IDE	Integrated Development Environment
API	Application Programming Interface
RAM	Random Access Memory
VRAM	Video Random Access Memory
GCC	GNU Compiler Collection
OS	Operating System

1. INTRODUCTION

In today's era, electronic commerce has transformed retailing, and its history reflects changes in technology and consumer behavior. The first online transaction took place in the 1970s using electronic data interchange (EDI). It wasn't until the 1990s that online shopping began to take shape, with the advent of secure online transaction technologies.

Key companies like Amazon and eBay played pivotal roles in the momentum of online shopping. The late 1990s and early 2000s saw significant growth due to technological advancements and increased internet usage.[1]

Today, companies like Amazon and Alibaba have reshaped consumer expectations by offering a wide range of products and simplified shopping experiences. The expansion of online shopping has been driven by factors such as convenience, wider product selection, and the integration of cutting-edge technologies like AI and machine learning.

To capitalize on the growing opportunities, we propose developing an innovative online shopping application using the C++ programming language, aiming to provide a user-centric platform that combines ease of use, personalization, and advanced features.

1.2 Motivation

The motivation for developing an online shopping system in C++ comes from the challenge of creating a practical, real-world e-commerce platform. We aim to build a secure, scalable, and user-friendly application that integrates essential features like product management, shopping cart functionality, and order processing.

This project offers an opportunity to apply C++ meaningfully and deepen our understanding of advanced concepts such as socket programming and data persistence. By undertaking this venture, we seek to enhance our technical skills while emphasizing the importance of e-commerce in today’s digital landscape. The aim of delivering a seamless user experience, combined with the intricacies of network communication and data management, makes this project both challenging and rewarding.

1.3 Objectives

The main objectives of our project are listed below:

- To enhance our programming skills in C++ by developing a robust practical application that will focus on object-oriented programming and socket programming.

- To design and implement a highly functional online shopping platform that will effectively showcase the power of C++ in network-based systems.

- To create a user-friendly interface that empowers customers to easily search for products, view detailed information, and manage their shopping carts seamlessly. By integrating client-server communication using Winsock, we are set to gain invaluable hands-on experience in network programming.

- To ensure secure user authentication and reliable data management through effective file-handling techniques.

2. LITERATURE REVIEW

In recent years, online shopping has become an essential part of modern society. As technology advances, so does the convenience and accessibility of e-commerce platforms, allowing consumers to purchase products from the comfort of their homes. The demand for online shopping has grown exponentially, driven by the desire for efficiency, variety, and competitive pricing. Alongside global giants, local and niche e-commerce platforms have also gained traction, catering to specific markets and offering unique products. With the rise of mobile shopping apps and websites, the landscape of retail has been transformed, creating new opportunities and challenges for businesses and consumers alike. This literature review examines various online shopping applications, focusing on their features, strengths, and limitations, to understand the dynamics of this rapidly evolving industry. [2]

2.1 Amazo

In recent years, online shopping has become a key component of modern society. Advances in technology have enhanced the convenience and accessibility of e-commerce, allowing consumers to shop from home. The demand has skyrocketed, driven by a preference for efficiency, variety, and competitive pricing. Alongside global giants, local and niche e-commerce platforms have emerged, serving specific markets with unique offerings. The rise of mobile shopping apps and websites has transformed the retail landscape, creating both opportunities and challenges for businesses and consumers. This literature review examines various online shopping applications to understand their features, strengths, and limitations in this rapidly evolving industry.

2.2 eBay

eBay is a well-known online marketplace that allows users to purchase and sell items through auctions or direct transactions. It features a vast array of products, ranging from brand-new to pre-owned items, focusing on user-determined pricing and bidding. The platform's auction feature introduces an exciting element, but it may also result in unpredictable prices and longer wait times for auction completions. Furthermore, eBay has encountered challenges regarding seller legitimacy and conflicts between buyers and sellers, which can adversely affect user experience.

2.3 Alibaba

Alibaba is a massive online marketplace primarily serving businesses, though it also caters to individual consumers. It is especially popular for sourcing products from manufacturers, often at wholesale prices. Alibaba's strength lies in its vast network of suppliers, making it a go-to platform for bulk purchases. However, the platform can be challenging for new users due to language barriers, complex logistics, and the need for careful vetting of suppliers to avoid scams and low-quality products.

2.4 Walmart

Walmart's online shopping platform extends the reach of its physical stores, offering a wide range of products from groceries to electronics. It is particularly known for its competitive pricing and in-store pickup options. Walmart's integration with its brick-and-mortar stores provides a seamless experience for customers who want the flexibility of online shopping with the convenience of local pickup. However, the platform can suffer from inventory discrepancies and occasional delivery delays, which may frustrate users.

2.5 Daraz

Daraz is one of the most popular online shopping platforms in Nepal, offering a wide range of products from electronics to fashion and household items. It is known for its extensive product catalog, frequent sales events, and user-friendly app interface. Daraz also provides various payment options, including cash on delivery, making it accessible to a broader audience in Nepal. However, delivery delays and occasional issues with product quality or authenticity can detract from the overall shopping experience.

2.6 SastoDeal

SastoDeal is another prominent e-commerce platform in Nepal that specializes in offering a wide range of affordable products, including electronics, fashion, and home appliances.

The platform is highly regarded for its customer service and often features attractive discounts. One of SastoDeal's strengths lies in its localized content, which is specifically tailored to meet the needs and preferences of the Nepali market. Through its partnerships with local sellers, SastoDeal can present unique and culturally relevant products to its customers. However, like many e-commerce platforms, it does encounter challenges such as delivery times and occasional discrepancies in product descriptions.

2.7 Fun

Online shopping applications have transformed the shopping experience by providing convenience and a range of engaging features. Shoppers can easily explore various products from the comfort of their homes, with personalized recommendations enhancing their experience by introducing them to new items. Many platforms also employ gamification strategies, such as loyalty programs and flash sales, which add excitement and urgency to the process. Additionally, users benefit from the ability to compare prices, read reviews, and even utilize augmented reality to try on clothes virtually. The integration of social media further enriches the experience, allowing users to share their discoveries and reviews with friends, and fostering a sense of community around shopping.

2.8 Challenges

Despite the many advantages, online shopping applications also present several challenges. Ensuring a seamless and secure transaction process is critical, but it can be difficult to maintain, especially in regions with inconsistent internet connectivity or limited access to reliable payment methods. User trust can be compromised by issues such as counterfeit products, delayed deliveries, or inaccurate product descriptions, leading to dissatisfaction and negative reviews. Additionally, the sheer volume of available products can overwhelm users, making it challenging to navigate the platform efficiently and find the desired items. For developers, creating a responsive and user-friendly interface that works well across different devices and screen sizes requires careful design and testing. Moreover, the competitive nature of the e-commerce industry means that businesses must continually innovate to stand out, which can be resource-intensive and complex to manage. Lastly, addressing the diverse needs and preferences of a global audience while maintaining local relevance adds another layer of complexity to developing and maintaining an online shopping platform.

3. PROGRAM ALGORITHM

3.1 Initialization

a. Set up interface

- Set up the basic user interface elements, including login prompts, main menu options, and error messages.

b. Render Initial State:

- Display the initial interface to the user.
- Initialize the Networking Environment.
- Ensure the network environment is ready for client-server interactions.

c. Create server Instance

- Start the server to listen on a designated port.
- Prepare the server to accept incoming client connections.

d. Load Configuration Files

- Load necessary configuration files that contain essential settings such as server credentials, product information, and user data.
- Ensure that all configuration parameters are correctly initialized and accessible.

e. Initialize user interface

- Initialize the interface such as the login screen or main menu.
- Ensure that all interactive elements are functioning and ready for user input.

f. Handle Initialization Errors:

- Implement error handling for connection failures or network issues.
- Provide feedback if the initialization process encounters any problems.

3.2 Authentication

a. User Sign-Up:

- Buyers can log in or sign up using credentials stored in a file (credentials.txt).
- Display a sign-up form to the user, requesting essential details such as username and password.
- Validate the provided information on the client side, ensuring that fields are properly filled out and meet the necessary criteria.

b. Seller-Side Validation:

- Sellers log in with predefined credentials (e.g., admin, admin123).
- On receiving the sign-up data, the server checks for the uniqueness of the username and email against the existing records in the database.

c. Error Handling:

- Provide clear and immediate feedback if the sign-up or login process fails, such as displaying error messages for incorrect credentials or already existing usernames.

3.3 Main Menu Operations

a. Item Management

- Items are categorized (e.g., FoodAndBeverages, Clothes, Books etc.) and stored in a map.
- Each category maps item names to their respective prices and stock for easy retrieval and organization.

b. View Products

- Allows the user to browse the available products in the store.
- Users can scroll through the products and select items of interest.

c. Add to Cart:

- Enables users to select products and add them to their virtual shopping cart.
- The system updates the cart dynamically, reflecting the total items and cost.
- The total cost is calculated and saved to a file (customerrecords.txt).

d. View Cart:

- Provides an overview of all the items the user has added to their shopping cart.

e. Checkout:

- Initiates the process of finalizing the purchase.
- Once confirmed, the system processes the transaction and generates an order confirmation.

f. Feedback/Complaint

- The system may prompt the user to provide details and may issue a reference number for tracking the complaint.
- Offers a way for users to report any issues or concerns they may have regarding the products or services.

g. Logout

- Logs the user out of their account this option is essential for protecting user privacy, especially on shared devices.
- After logging out, the user is typically redirected to the login screen or the main landing page.

3.4 Shopping Cart Management

a. Add Products to Cart:

- The customer captures user input through a console interface.
- This input is processed by the client application, which directly manipulates data structures in C++ (such as arrays or vectors) to add the selected items to the cart.

- The cart is updated in the client's local memory, reflecting the user's choices immediately.

b. View Cart:

- Users can view their current cart by selecting an option in the console interface. The client iterates through the cart's data structures, displaying the product names, quantities, and prices.

c. Calculate Total Cost:

- The client calculates the total cost by iterating through the cart and summing the product prices, considering quantities.
- The total cost is displayed to the user in the console, along with options to proceed with the checkout process or make further changes to the cart.

3.5 Checkout Process

a. User Confirmation:

- Before finalizing the purchase, the system presents the user with a detailed summary of their order.
- This includes a list of items in the cart, their respective quantities, individual prices, and the total cost including taxes and shipping fees.

3.6 Main Menu Operations

a. View or Manage Stock

- Verify if the product already exists in the catalog. If it does, update the stock quantity.
- If the product does not exist, add the new product to the catalog. If it exists, update the stock count.

b. Address Feedbacks

- Accept Feedback or complaints from customer.

- Address and reply back to the customer.

c. View Order History

- Fetch the order history from the records, including details of each order.
- Transmit the order history to the client, displaying all relevant order details if s/he asks for it.

3.7 Logout

a. Redirect to Main Menu:

- The server should redirect the user or admin to the main menu or login screen.
- This ensures that the user is aware of their new state and can initiate a new session if desired.
- If redirected to the login screen, the system should prompt the user to log in again if they wish to continue using the system.

b. Error Handling:

- In case of any issues during the logout process (e.g., failure to clear session data), the server should log the error and inform the client with a message to retry or contact support.

3.8 Exit the console

a. Terminate Program:

- The server sends a final message to the client. This provides a clear indication that the application is closing.

b. Save Current State:

- Ensure that all current data, including user information, product catalogs, order histories, and complaint records, are saved to persistent storage.

- This prevents data loss and ensures that the system can be resumed seamlessly in future sessions.

4. PROPOSED SYSTEM ARCHITECTURE

4.1 System Architecture

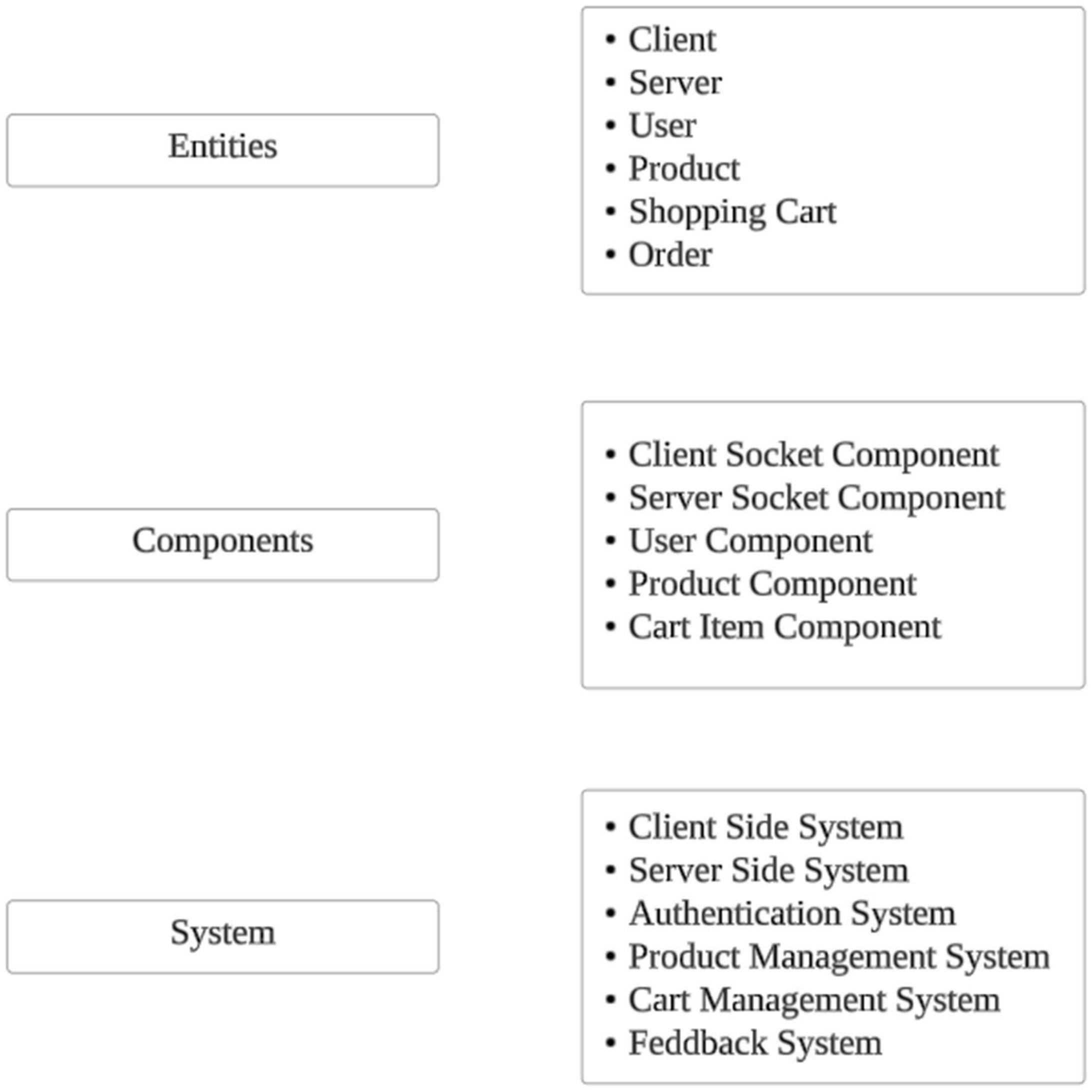


Figure 4- 1 System Architecture

Here the Entity-Component-System Architecture is shown:

4.1.1 Entities

In the application, the main entities include:

- Client: Represents the user-side application that interacts with the server.
- Server: Manages the application's core functionalities, including user authentication, product management, and order processing.
- User: Represents the customers interacting with the application.
- Product: Represents the items available for purchase.
- Shopping Cart: Manages the items selected by the user for potential purchase.
- Order: Represents completed transactions by users.

4.1.2 Components

Each entity is composed of various components that define its behavior and properties:

- Client Socket Component: Manages communication between the client and server, including sending requests and receiving responses.
- Server Socket Component: Handles incoming connections and processes requests from clients.
- User Component: Stores user details, including username, password, and account information.
- Product Component: Contains product details such as name, price, category, and stock quantity.
- Cart Item Component: Manages individual items within the shopping cart, including product reference and quantity.

4.1.3 Systems

Systems process entities by iterating over their components and performing specific tasks:

- Client-Side System: Handles user interactions on the client side, including login, browsing products, and managing the shopping cart. It communicates with the server using the Client Socket Component.
- Server-Side System: Manages the core functionalities on the server side, using the Server Socket Component.
- Authentication System: Handles user login and registration on the server side, validating credentials using the Authentication Component.
- Product Management System: Manages the creation, modification, and removal of products on the server side using the Product Component.
- Cart Management System: Allows users to add, update, or remove items from their shopping cart, managed on the client side and synchronized with the server.
- Communication System: Ensures smooth communication between the client and server, handling requests and responses through the socket components.[3]

4.2 Flowchart

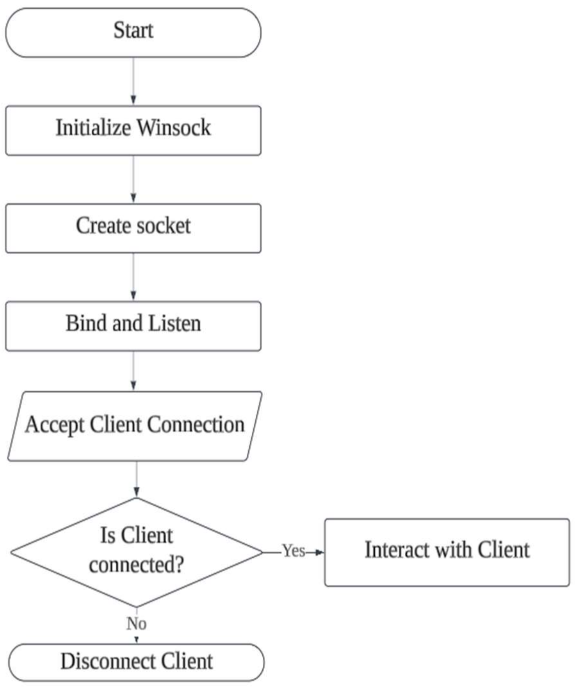


Figure 4- 2 Networking

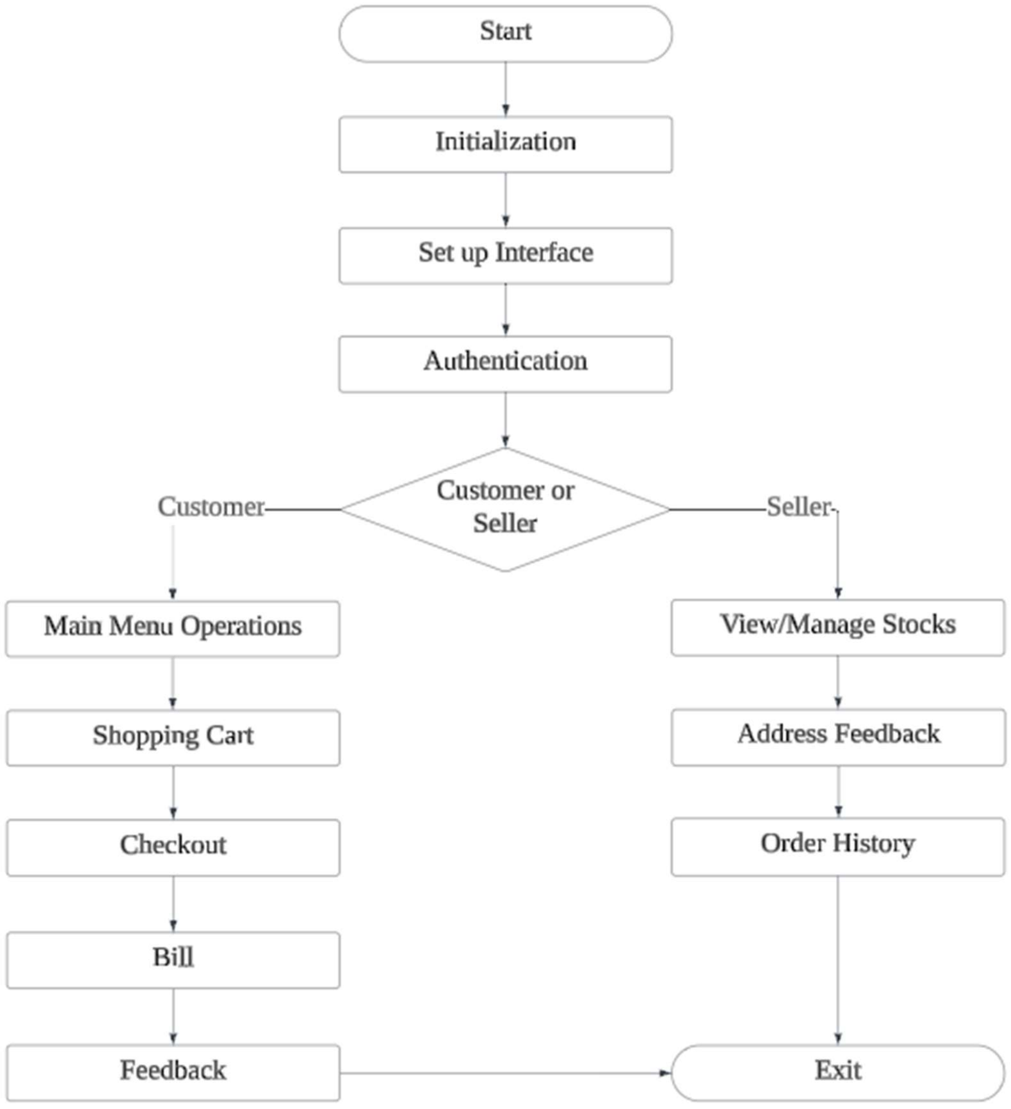


Figure 4- 3 Flowchart for SmartShop

4.3 Tools and environment

We will utilize a combination of IDEs, libraries, and tools to facilitate efficient development, debugging, and testing.

4.3.1 Language

To code this game, only the C++ programming language is being used.

4.3.2 IDEs

a. Visual Studio 2022

Visual Studio 2022 is a powerful integrated development environment (IDE) developed by Microsoft. It supports multiple programming languages like C++, C#, Python, and more, enabling developers to build, test, and deploy applications for various platforms. With features like IntelliCode, advanced debugging tools, and enhanced collaboration capabilities, it streamlines the software development process.

4.3.3 Compiler

a. G++

G++ is a popular compiler specifically designed for C++ programming, building on the foundation of the GCC suite. Renowned for its efficiency, reliability, and strong support for modern C++ standards, G++ is widely used in both academic and professional environments. It allows seamless compilation and building of C++ source code across different platforms, including Windows, Linux, and macOS, making it an ideal choice for developing cross-platform applications like SmartShop.

5. METHODOLOGY

5.1 Header Files

- <iostream>: Standard input/output stream objects, used for basic I/O operations like cin and cout.
- <Winsock2.h>: Windows Sockets API, used for network communication, including socket creation and management.
- <WS2tcpip.h>: Provides advanced Winsock functionalities like support for IPv6 and helper functions for address translation.
- <tchar.h>: Defines macros and types for handling Unicode and ANSI character strings, enabling portable code for text processing.
- <thread>: Standard library for multi-threading, used to create and manage threads for parallel execution.
- <string>: Provides the std::string class for handling and manipulating text data conveniently.
- <map>: Standard template library (STL) associative container, used for storing key-value pairs in a sorted manner.
- <sstream>: String stream classes, used for performing formatted I/O operations on string objects.
- <fstream>: File stream classes, used for file handling operations such as reading from and writing to files.
- <vector>: Standard template library (STL) dynamic array container, used for efficiently managing collections of elements.
- <iomanip>: Used for manipulating I/O stream formats, such as setting precision and formatting output with manipulators like std::setw.

These header files are essential for the functionality of the online shopping system, handling everything from I/O operations to networking.

5.2 Functions

Functions are essential in this system because they modularize the code, leading to improved organization, readability, and maintenance. Each function manages a particular component, such as initialization, user input processing, and system logic updates. This modular design promotes code reuse, simplifies debugging, and makes it easier to modify or add features. Additionally, functions help abstract complex processes, which can improve code clarity and understanding, particularly in larger projects like game development. Some functions used are:[4]

- fillItems(): Initializes the items and names data structures with predefined values.
- addToCart(): Takes two arguments: the category of the item and the item name and increments the quantity of the specified item in the "selected_items" map.
- printBill(): Calculates and displays the total bill for the items selected by the customer. Prints a summary of items in the cart, including their category, name, quantity, and cost, Computes and shows the total amount due.
- show(): Displays a list of items available for selection, including their costs.
- select():This is a virtual function in the base 'shop' class and overridden in each derived class (mobile, laptop, courses) to handle the selection of items specific to that category.
- showMenu(): Displays the main menu options for the user to choose from (e.g. Mobile, Laptop, Courses, Checkout).
- sellermenu(): Displays the main menu options for seller to choose from (e.g. check/manage stocks, see feedbacks/complaints, etc.).
- main(): Entry point of the program. This function calls 'fillItems()' to initialize data manage interaction loop where user can select items or proceed to checkout.
- is_open(): Check if the file opened successfully.
- close(): Close the files and clean up resources.
- loadCredentials(): Function to load buyer credentials from file.
- viewTransactionHistory(): Shows transaction history with buyer's name.
- updateStockToFile(const string& category): Updates changes stock into the file.
- viewCart(): Shows selected items along with their price and total cost.
- loadItemsFromFile("File_name"): Loads previously saved items from file to console.
- clearScreen(): Clears console screen.

- Initialize(): Sets up socket to send messages.
- complaint(): Connects to server and sends complaints/feedback from buyer.
- viewFeedbackHistory(): Shows buyer's feedback to seller and seller can give responses to the feedbacks.
- seeResponses(): Shows buyer's feedback along with responses from seller to buyer.
- checkManageStocks(): Can check and edit stocks for different items.
- selectCategory(): Displays the main menu options for the user to choose from e.g. Foods, Clothes, etc.

5.3 Conditional Statements

Conditional statements are integral to the functionality of the "SmartShop: An Online Shopping System," as they dictate the flow of the application based on user inputs and system states. These statements ensure that the system responds correctly to different scenarios, such as user authentication, product management, and checkout processes. They are crucial for validating user credentials during login, verifying stock availability before adding products to the cart, and handling errors like incorrect input or insufficient inventory. By using conditional logic, the system maintains a smooth and user-friendly experience, ensuring that each action is processed according to the defined business rules.

Some specific conditional statements used in this project include:

- **If-Else Statements:** These are used to manage simple decision-making processes, such as checking if a user is logged in before allowing access to the shopping cart or admin features. For example, when a user tries to log in, an if statement checks whether the entered username and password match the stored credentials. If they do, the user is granted access; otherwise, an error message is displayed.

- **If-Else Ladder:** This structure is used in scenarios where multiple conditions need to be checked sequentially, such as in the main menu where the system decides whether to display options for browsing products, viewing the cart, or managing products (for admins). Depending on the user's input, the appropriate function is called. For example, in the checkout process, the system might use an if-else ladder to determine the payment method selected by the user and proceed accordingly.

6. SCOPE AND APPLICATIONS

Online shopping has transformed into an engaging and convenient activity for individuals across all age groups, becoming a vital part of daily life for many. An online shopping system is a dynamic digital platform that empowers users to explore, select, and purchase a wide array of products or services from the comfort of their homes. Its extensive reach and diverse uses have profoundly impacted business operations and consumer relationships. With the increasing desire for doorstep delivery, many innovative digital platforms have emerged to meet this growing demand. To better understand the motivations and strategies driving these online retailers, we proudly developed a cutting-edge platform called Smartshop.

- This method builds an online shopping system using an open graphics library instead of a common shopping engine.
- This offers a practical option for developing systems in C++.
- It helps both developers and users learn more about system development with C++.
- C++ is a good choice because it can store and work with large amounts of data effectively.

7. TIME ESTIMATION

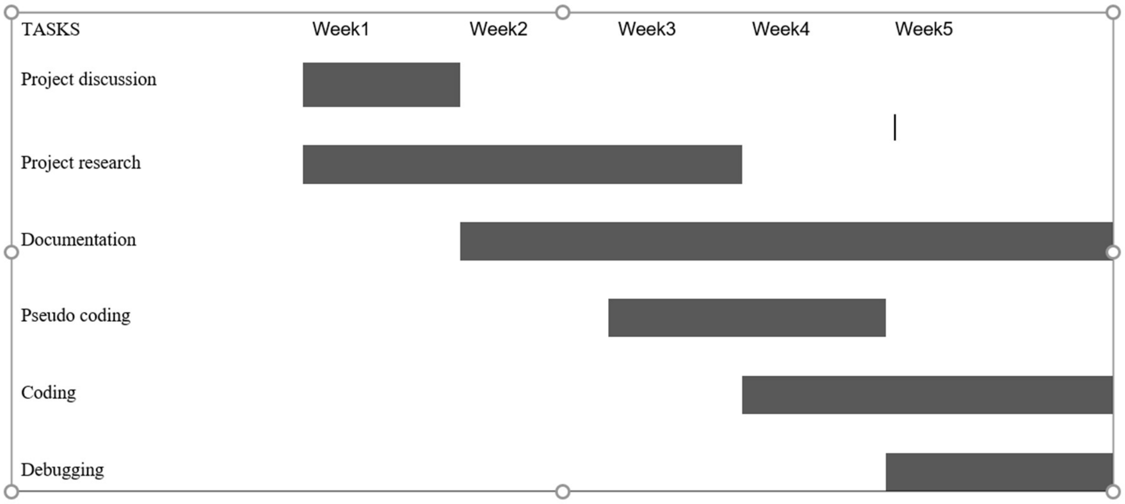


Table 7 - 1 Time Estimation

8. FEASIBILITY ANALYSIS

8.1 Economic Feasibility

Since this project is made using C++, there's no financial cost required for building this project. Along with that, all the software being used in the development of the project is easy to access and learn so, there is not any financial support needed.

8.2 Technical Feasibility

The technical feasibility of the online shopping system project is promising due to the availability of well-established technologies and tools. The project utilizes C++ with the Entity-Component-System (ECS) architecture, ensuring modularity and scalability in design. Socket programming enables efficient communication between the client and server. Libraries such as <iostream>, <map>, and <string> support data handling and management, while networking headers like <Winsock2.h> and <WS2tcpip.h> provide robust connectivity solutions. With C++ offering high performance and flexibility, coupled with easily accessible development environments like Visual Studio 2022, the project is technically feasible to implement and extend for future enhancements.

8.3 Operational Feasibility

It runs on any device (Windows/Linux/mac OS) provided that the device meets the system requirement.

8.3.1 System Requirement

- 32bit / 64bit Operating system
- 8 GB or more RAM
- 256 MB or more VRAM
- WINDOWS OS (8 or higher) or Linux or Mac OS

9. SOURCE CODE

9.1 Server Code

```
#include<iostream>
#include<WinSock2.h>
#include<Ws2tcpip.h>
#include<tchar.h>
#include<thread>
#include<vector>
#include<fstream>
#include <string>

using namespace std;

#pragma comment(lib,"ws2_32.lib")

bool Initialize() {
    WSADATA data;
    return WSAStartup(MAKEWORD(2, 2),
&data) == 0;
}

// Function to retrieve the last serial number
from the first part of the last line of
customer_messages.txt
int getLastSerialNumber() {
    ifstream infile("customer_messages.txt",
ios::in);

    // Check if the file opened successfully
    if (!infile.is_open()) {
        return 0; // Return 0 if the file doesn't exist
or is empty
    }

    string lastLine;
    string line;

    // Read the file line by line and keep only the
last line
    while (getline(infile, line)) {
        lastLine = line;
    }

    infile.close();

    // Extract the serial number from the last line
    if (!lastLine.empty()) {
        size_t pos = lastLine.find(". ");
        if (pos != string::npos) {
            try {
                // Convert the part before ". " into an
integer
                return stoi(lastLine.substr(0, pos));
            }
            catch (...) {
                return 0; // Return 0 if parsing fails
            }
        }
    }

    return 0; // Return 0 if the file is empty or
doesn't have valid entries
}



void InteractWithClient(SOCKET clientSocket,
vector<SOCKET>& clients) {
    cout << "Client connected\n";
    char buffer[4096];

    // Open separate files for messages and
responses
    ofstream
messageLog("customer_messages.txt",
ios::app);
    ofstream responseLog("seller_responses.txt",
ios::app);
    if (!messageLog.is_open() ||
!responseLog.is_open()) {
        cerr << "Error: Unable to open or create
log files.\n";
        closesocket(clientSocket);
        return;
    }

    while (true) {
        int bytesrecvd = recv(clientSocket, buffer,
sizeof(buffer), 0);

        if (bytesrecvd <= 0) {
            cout << "Client disconnected\n";
            break; // Exit the loop if client
disconnects or an error occurs
        }

        string message(buffer, bytesrecvd);

        // If the message is empty, continue to the
next iteration
        if (message.empty()) {
            continue; // Ignore empty messages
        }

        cout << message << endl;

        // Check the prefix of the message to classify
it
```

```
        if (message.rfind("Message from the
customer:", 0) == 0) {
            // Log regular customer messages
            messageLog << getLastSerialNumber()
+ 1 << ". " << message << endl;
        }
        else if (message.rfind("Reply to Feedback",
0) == 0) {
            // Log replies to feedback
            responseLog << message << endl;
        }

        // Broadcast the message to other clients
        for (auto client : clients) {
            if (client != clientSocket) { // Do not send
to the sender
                int bytessent = send(client,
message.c_str(), message.length(), 0);
                if (bytessent == SOCKET_ERROR) {
                    cout << "Error sending message to
a client.\n";
                }
            }
        }
    }

    // Close the files and clean up resources
    messageLog.close();
    responseLog.close();

    auto it = find(clients.begin(), clients.end(),
clientSocket);
    if (it != clients.end()) {
        clients.erase(it);
    }
    closesocket(clientSocket);
}


int main() {
    if (!Initialize()) {
        cout << "Winsock initialization failed " <<
endl;
        return 1;
    }

    cout << "Server program started\n";

    SOCKET listenSocket = socket(AF_INET,
SOCK_STREAM, 0);
    if (listenSocket == INVALID_SOCKET) {
        cout << "Socket creation failed\n";
        return 1;
    }

    // Create address structure
    int port = 12345;
    sockaddr_in serveraddr;
    serveraddr.sin_family = AF_INET;
    serveraddr.sin_port = htons(port);

    // Convert the IP address (0.0.0.0) to binary
form and set it
    if (InetPton(AF_INET, _T("0.0.0.0"),
&serveraddr.sin_addr) != 1) {
        cout << "Setting address structure
failed\n";
        closesocket(listenSocket);
        WSACleanup();
        return 1;
    }

    // Bind the socket to the address and port
    if (bind(listenSocket,
reinterpret_cast<sockaddr*>(&serveraddr),
sizeof(serveraddr)) == SOCKET_ERROR) {
        cout << "Bind failed\n";
        closesocket(listenSocket);
        WSACleanup();
        return 1;
    }

    // Listen for incoming connections
    if (listen(listenSocket, SOMAXCONN) ==
SOCKET_ERROR) {
        cout << "Listen failed\n";
        closesocket(listenSocket);
        WSACleanup();
        return 1;
    }

    cout << "Server is listening on port: " <<
port << endl;

    vector<SOCKET> clients;

    while (1) {
        // Accept a client connection
        SOCKET clientSocket =
accept(listenSocket, nullptr, nullptr);
        if (clientSocket == INVALID_SOCKET) {
            cout << "Invalid client socket\n";
            continue;
        }

        clients.push_back(clientSocket);
        thread t1(InteractWithClient, clientSocket,
ref(clients));
        t1.detach();
    }

    closesocket(listenSocket);
    WSACleanup();
    return 0;
}
```

9.2 Client Side Code

```
#include <iostream>
#include<Winsock2.h>
#include<WS2tcpip.h>
#include<tchar.h>
#include<thread>
#include<string>
#include <map>
#include <sstream>
#include <fstream>
#include <vector>
#include <iomanip>

using namespace std;

#pragma comment(lib, "ws2_32.lib")

// Data structures
map<string, map<string, double>> items;  //
Category -> Item -> Price
map<string, map<string, int>> stock;     //
Category -> Item -> Stock
map<string, map<string, int>> selected_items;
// Customer's Cart (Category -> Item ->
Quantity)
map<string, string> buyerCredentials = {
{"default_user", "default_pass"} }; // Predefined
buyer credentials
const string sellerUsername = "admin";
const string sellerPassword = "admin123";
string username = "Guest";

// File to store buyer credentials
const string credentialsFile = "credentials.txt";

// Function to load buyer credentials from file
void loadCredentials() {
    ifstream infile(credentialsFile);
    if (infile.is_open()) {
        string username, password;
        while (infile >> username >> password) {
            buyerCredentials[username] =
password;
        }
        infile.close();
    }
    else {
        cout << "No previous credentials found.
Starting fresh.\n";
    }
}

void clearScreen() {
    system("cls");
    cout << "SMARTSHOP" << endl;
}
// Function to save a new buyer credential to file
void saveCredential(const string& username,
const string& password) {
    ofstream outfile(credentialsFile, ios::app); //
Open in append mode
    if (outfile.is_open()) {
        outfile << username << " " << password
<< "\n";
        outfile.close();
    }
    else {
        cout << "Error: Unable to save credentials
to file.\n";
    }
}

// Function to handle buyer sign-up
void buyerSignup() {
    string  password;
    cout << "Enter a new username: ";
    cin >> username;
    cout << "Enter a new password: ";
    cin >> password;

    if (buyerCredentials.find(username) !=
buyerCredentials.end()) {
        cout << "Username already exists. Please
try a different one.\n";
    }
    else {
        buyerCredentials[username] = password;
        saveCredential(username, password); //
Save to file
        cout << "Signup successful! You can now
login.\n";

    }
}

// Function to handle buyer login
bool buyerLogin(string& username) {
    cout << "Enter username: ";
    cin >> username;
    cout << "Enter password: ";
    string password;
    cin >> password;

    if (buyerCredentials.find(username) !=
buyerCredentials.end() &&
buyerCredentials[username] == password) {
        cout << "Login successful!\n";
        clearScreen();
        return true;
    }
    else {
        cout << "Invalid credentials. Please try
again or sign up.\n";
        return false;
    }
}

// Function to handle seller login
bool sellerLogin() {
    cout << "Enter seller username: ";
    string username;
    cin >> username;
    cout << "Enter seller password: ";
    string password;
    cin >> password;

    if (username == sellerUsername &&
password == sellerPassword) {
        cout << "Seller login successful!\n";
        clearScreen();
        return true;
    }
    else {
        cout << "Invalid seller credentials.\n";
        return false;
    }
}


// Function to load items and stock from a file
void loadItemsFromFile(const string& category)
{
    ifstream file(category + ".txt");
    string item;
    double price;
    int quantity;
    while (file >> item >> price >> quantity) {
        items[category][item] = price;
        stock[category][item] = quantity;
    }
}

void updateStockToFile(const string& category)
{
    ofstream file(category + ".txt");
    if (!file.is_open()) {
        cout << "Failed to open the file for
writing!" << endl;
        return;
    }

    for (const auto& entry : stock[category]) {
        file << entry.first << " " <<
items[category][entry.first] << " " <<
entry.second << endl;
    }

    file.close();
}

void viewTransactionHistory() {
    ifstream infile("customerrecords.txt");

    if (!infile.is_open()) {
        cout << "Error: Unable to open the
transaction history file.\n";
        return;
    }

    cout << "\n--- Transaction History ---\n";
    string line;
    bool hasContent = false;

    while (getline(infile, line)) {
        hasContent = true;
        cout << line << "\n";
    }

    infile.close();

    if (!hasContent) {
        cout << "No transactions found in the
history.\n";
    }
    else {
        cout << "\n--- End of Transaction History -
--\n";
    }

    // Offer the user a choice to return to the menu
or exit
    int choice;
    while (true) {
        cout << "\nOptions:\n";
        cout << "1. Return to Menu\n";
        cout << "2. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "Returning to menu...\n";
            clearScreen();
            return; // Exit the loop and return to the
menu
        case 2:
            cout << "Exiting program. Goodbye!\n";
            clearScreen();
        exit(0); // Terminate the program
      default:
        cout << "Invalid choice. Please try
again.\n";
      }
      return;
  }

}

// Class customer
class customer {
public:
  string name;
  map<string, map<string, pair<int,
double>>> cart;  // category -> item ->
{quantity, price}


  customer() {
      cout << "--WELCOME TO SMARTSHOP--
\n\n";
  }

  void addToCart(string category, string item,
int quantity) {
      selected_items[category][item] +=
quantity;  // Correct quantity update
  }

  void displayBill(ostream& out, bool
showHeader = true) {
      const int widthCategory = 20; // Width for
Category column
      const int widthItem = 20;     // Width for
Item column
      const int widthQuantity = 10; // Width for
Quantity column
      const int widthCost = 10;     // Width for
Cost column

      int total_amount = 0;
      cout << endl<<"Customer name :" <<
username << endl;

      if (showHeader) {
        out << std::left // Align text to the left
          << std::setw(widthCategory) <<
"Category"
          << std::setw(widthItem) << "Item"
          << std::setw(widthQuantity) <<
"Quantity"
          << std::setw(widthCost) << "Cost"
          << "\n";
        out << std::string(widthCategory +
widthItem + widthQuantity + widthCost, '-') <<
"\n"; // Separator line
      }

      for (auto& categoryPair : selected_items) {
        for (auto& itemPair :
categoryPair.second) {
          string item = itemPair.first;
          int quantity = itemPair.second;

          if (items[categoryPair.first].find(item)
!= items[categoryPair.first].end()) {
            double cost = quantity *
items[categoryPair.first][item];
            out << std::left
              << std::setw(widthCategory) <<
categoryPair.first
              << std::setw(widthItem) << item
              << std::setw(widthQuantity) <<
quantity
              << std::setw(widthCost) << cost
              << "\n";
            total_amount += cost;
          }
          else {
            out << std::left
              << std::setw(widthCategory) <<
categoryPair.first
              << std::setw(widthItem) << item
              << std::setw(widthQuantity) <<
quantity
              << std::setw(widthCost) <<
"ERROR"
              << "\n";
          }
        }
      }

      out << std::string(widthCategory +
widthItem + widthQuantity + widthCost, '-') <<
"\n";
      out << std::left
        << std::setw(widthCategory + widthItem
+ widthQuantity) << "Total Amount:"
        << std::setw(widthCost) <<
total_amount
        << "\n";
      out << std::string(widthCategory +
widthItem + widthQuantity + widthCost, '-') <<
"\n";
  }
  void viewCart() {
      cout << "\n--- Your Cart ---\n";
      displayBill(cout);
  }
```

```
   void printBill() {
      string name;

      cout << "\n--- Final Bill ---\n";

      displayBill(cout);

      // Write the bill to the file
      ofstream outfile("customerrecords.txt",
ios::app);
      if (outfile.is_open()) {
         displayBill(outfile, false); // Don't repeat
the header for the file
         outfile << "*****THANK YOU &&
HAPPY ONLINE SHOPPING*****\n\n";
         outfile.close();
      }
      else {
         cout << "Error: Unable to open the file
for writing.\n";
      }

      cout << "*****THANK YOU && HAPPY
ONLINE SHOPPING*****\n";
   }


};



// Class shop and its subclasses
class shop {
public:

   virtual void show() {}
   virtual void select(customer& obj) {}

   void showMenu() {
      cout << "                Menu\n";
      cout << " ----------------------------------------
-\n";
      cout <<
"1.Shop\n2.Checkout\n3.Feedback/Complaint\n4
.See Responses\n5.Return to login page\n6.Exit
the program\n";
      cout << " ----------------------------------------
-\n";
   }
};

class FoodAndBeverages : public shop {
public:
   void show() override {
      loadItemsFromFile("FoodAndBeverages");
      const int widthItem = 25;  // Adjusted width
for longer item names
      const int widthRate = 15; // Width for Rate
column
      const int widthStock = 10; // Width for
Stock column

      cout << "Food and Beverages:\n";
      cout << std::left
         << std::setw(widthItem) << "Item"
         << std::setw(widthRate) << "Rate (Rs.)"
         << std::setw(widthStock) << "Stock"
         << "\n";
      cout << std::string(widthItem + widthRate
+ widthStock, '-') << "\n";

      for (const auto& item :
items["FoodAndBeverages"]) {
         cout << std::left
            << std::setw(widthItem) << item.first
            << std::setw(widthRate) <<
item.second
            << std::setw(widthStock) <<
stock["FoodAndBeverages"][item.first]
            << "\n";
      }


   }

   void select(customer& customer) override {
      string item;
      int quantity;
      cout << "Type 'return' to go back.\n";
      cout << "Enter item name to add to your
cart: ";
      cin >> item;

      if (item == "return") {
         clearScreen();
         return;
      }


      if (items["FoodAndBeverages"].find(item)
!= items["FoodAndBeverages"].end()) {
         cout << "Enter quantity: ";
         cin >> quantity;


         if (stock["FoodAndBeverages"][item] >=
quantity) {

customer.addToCart("FoodAndBeverages",
item, quantity);
```

```
            stock["FoodAndBeverages"][item] -=
quantity;

updateStockToFile("FoodAndBeverages");
            cout << "Item added to your cart.\n";
          }
          else {

            cout << "Error: Not enough stock
available. Only "
              <<
stock["FoodAndBeverages"][item]
              << " item(s) are available.\n";
          }
        }
        else {
          cout << "Item not found.\n";
        }
      }

};


class Clothes : public shop {
public:
   void show() override {
      loadItemsFromFile("Clothes");
      const int widthItem = 20;  // Width for Item
column
      const int widthRate = 15;  // Width for Rate
column
      const int widthStock = 10; // Width for
Stock column

      cout << "Clothes:\n";
      cout << std::left
         << std::setw(widthItem) << "Item"
         << std::setw(widthRate) << "Rate (Rs.)"
         << std::setw(widthStock) << "Stock"
         << "\n";
      cout << std::string(widthItem + widthRate
+ widthStock, '-') << "\n";

      for (const auto& item : items["Clothes"]) {
         cout << std::left
            << std::setw(widthItem) << item.first
            << std::setw(widthRate) <<
item.second
            << std::setw(widthStock) <<
stock["Clothes"][item.first]
            << "\n";
      }
   }

   void select(customer& customer) override {
      string item;
      int quantity;
      cout << "Type 'return' to go back.\n";
      cout << "Enter item name to add to your
cart: ";
      cin >> item;

      if (item == "return") {
         clearScreen();
         return;  // Exit the select function and
return to the previous menu
      }

      // Check if the item exists in the "Clothes"
category
      if (items["Clothes"].find(item) !=
items["Clothes"].end()) {
         cout << "Enter quantity: ";
         cin >> quantity;

         // Check if requested quantity is available
in stock
         if (stock["Clothes"][item] >= quantity) {
            customer.addToCart("Clothes", item,
quantity);
            // Decrease the stock after purchase
            stock["Clothes"][item] -= quantity;
            updateStockToFile("Clothes");
            cout << "Item added to your cart.\n";
         }
         else {
            // If requested quantity exceeds
available stock
            cout << "Error: Not enough stock
available. Only "
               << stock["Clothes"][item]
               << " item(s) are available.\n";
         }
      }
      else {
         cout << "Item not found.\n";
      }
   }

};


class Stationery : public shop {
public:
   void show() override {
      loadItemsFromFile("Stationery");
      const int widthItem = 20;
      const int widthRate = 15;
      const int widthStock = 10;

      cout << "Stationery:\n";
      cout << std::left
```

```
            << std::setw(widthItem) << "Item"
            << std::setw(widthRate) << "Rate (Rs.)"
            << std::setw(widthStock) << "Stock"
            << "\n";
        cout << std::string(widthItem + widthRate
+ widthStock, '-') << "\n";

        for (const auto& item : items["Stationery"])
{
            cout << std::left
                << std::setw(widthItem) << item.first
                << std::setw(widthRate) <<
item.second
                << std::setw(widthStock) <<
stock["Stationery"][item.first]
                << "\n";
        }
    }

    void select(customer& customer) override {
        string item;
        int quantity;
        cout << "Type 'return' to go back.\n";
        cout << "Enter item name to add to your
cart: ";
        cin >> item;

        if (item == "return") {
            clearScreen();
            return;  // Exit the select function and
return to the previous menu
        }

        // Check if the item exists in the "Stationery"
category
        if (items["Stationery"].find(item) !=
items["Stationery"].end()) {
            cout << "Enter quantity: ";
            cin >> quantity;

            // Check if requested quantity is available
in stock
            if (stock["Stationery"][item] >=
quantity) {
                customer.addToCart("Stationery",
item, quantity);
                // Decrease the stock after purchase
                stock["Stationery"][item] -= quantity;
                updateStockToFile("Stationery");
                cout << "Item added to your cart.\n";
            }
            else {
                // If requested quantity exceeds
available stock
                cout << "Error: Not enough stock
available. Only "
                    << stock["Stationery"][item]
                    << " item(s) are available.\n";
            }
        }
        else {
            cout << "Item not found.\n";
        }
    }



};

class Electronics : public shop {
public:
    void show() override {
        loadItemsFromFile("Electronics");
        const int widthItem = 20;
        const int widthRate = 15;
        const int widthStock = 10;

        cout << "Electronics:\n";
        cout << std::left
            << std::setw(widthItem) << "Item"
            << std::setw(widthRate) << "Rate (Rs.)"
            << std::setw(widthStock) << "Stock"
            << "\n";
        cout << std::string(widthItem + widthRate
+ widthStock, '-') << "\n";

        for (const auto& item :
items["Electronics"]) {
            cout << std::left
                << std::setw(widthItem) << item.first
                << std::setw(widthRate) <<
item.second
                << std::setw(widthStock) <<
stock["Electronics"][item.first]
                << "\n";
        }
    }

    void select(customer& customer) override {
        string item;
        int quantity;
        cout << "Type 'return' to go back.\n";
        cout << "Enter item name to add to your
cart: ";
        cin >> item;

        if (item == "return") {
            clearScreen();
            return;  // Exit the select function and
return to the previous menu
        }
        // Check if the item exists in the
"Electronics" category
        if (items["Electronics"].find(item) !=
items["Electronics"].end()) {
            cout << "Enter quantity: ";
            cin >> quantity;

            // Check if requested quantity is available
in stock
            if (stock["Electronics"][item] >=
quantity) {
                customer.addToCart("Electronics",
item, quantity);
                // Decrease the stock after purchase
                stock["Electronics"][item] -=
quantity;
                updateStockToFile("Electronics");
                cout << "Item added to your cart.\n";
            }
            else {
                // If requested quantity exceeds
available stock
                cout << "Error: Not enough stock
available. Only "
                    << stock["Electronics"][item]
                    << " item(s) are available.\n";
            }
        }
        else {
            cout << "Item not found.\n";
        }
    }

};

class Books : public shop {
public:
    void show() override {
        loadItemsFromFile("Books");
        const int widthItem = 20;
        const int widthRate = 15;
        const int widthStock = 10;

        cout << "Books:\n";
        cout << std::left
            << std::setw(widthItem) << "Item"
            << std::setw(widthRate) << "Rate (Rs.)"
            << std::setw(widthStock) << "Stock"
            << "\n";
        cout << std::string(widthItem + widthRate
+ widthStock, '-') << "\n";

        for (const auto& item : items["Books"]) {
            cout << std::left
                << std::setw(widthItem) << item.first
                << std::setw(widthRate) <<
item.second
                << std::setw(widthStock) <<
stock["Books"][item.first]
                << "\n";
        }
    }

    void select(customer& customer) override {
        string item;
        int quantity;

        // Allow user to type "return" to go back
        cout << "Type 'return' to go back.\n";
        cout << "Enter item name to add to your
cart: ";
        cin >> item;

        // If user types "return", exit the function
        if (item == "return") {
            clearScreen();
            return;  // Exit the select function and
return to the previous menu
        }

        if (items["Books"].find(item) !=
items["Books"].end()) {
            cout << "Enter quantity: ";
            cin >> quantity;

            // Check if requested quantity is available
in stock
            if (stock["Books"][item] >= quantity) {
                customer.addToCart("Books", item,
quantity);
                // Decrease the stock after purchase
                stock["Books"][item] -= quantity;
                updateStockToFile("Books");  //
Update the stock in the file
                cout << "Item added to your cart.\n";
            }
            else {
                // If requested quantity exceeds
available stock
                cout << "Error: Not enough stock
available. Only "
                    << stock["Books"][item]
                    << " item(s) are available.\n";
            }
        }
        else {
            cout << "Item not found.\n";
        }
    }
```

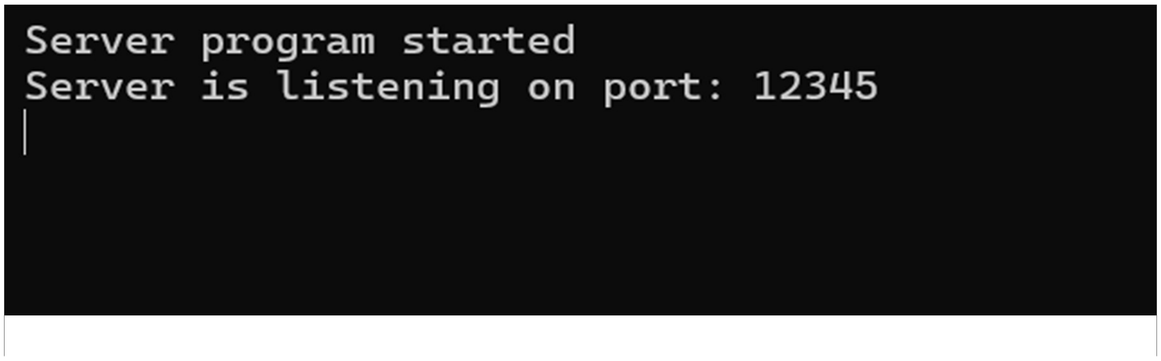
```
};

bool Initialize() {
   WSADATA data;
   return WSAStartup(MAKEWORD(2, 2),
&data) == 0;
}

void SendMsg(SOCKET s) {
   string message;
   cout << "Write your feedback or complaint
below and press enter to send.\n";
   cout << "(Type 'menu' to return to the main
menu or 'exit' to quit.)\n";

   while (true) {
      getline(cin, message);

      if (message == "menu") {
         cout << "Returning to the main
menu...\n";
         break;  // Exit the loop and return to the
menu
      }
      else if (message == "exit") {
         cout << "Exiting the program. Thank you
for your feedback!\n";
         exit(0);  // Exit the program
      }

      string msgToSend = "Message from the
customer: " + message;
      if (message.empty()) {
         cout << "Empty message entered. Not
sending.\n";
         continue;
      }

      int bytesent = send(s, msgToSend.c_str(),
msgToSend.length(), 0);
      if (bytesent == SOCKET_ERROR) {
         cout << "Error sending message.\n";
         break;
      }
      cout << "Message sent successfully.\n";
   }

   closesocket(s);
   WSACleanup();

}



void ReceiveMsg(SOCKET s) {
   char buffer[4096];
   int recvlength;
   string msg = "";
   while (1) {
      recvlength = recv(s, buffer, sizeof(buffer),
0);
      if (recvlength <= 0) {
         cout << "disconnected from the server"
<< endl;
         break;
      }
      else {
         msg = string(buffer, recvlength);
         cout << msg << endl;
      }
   }
}

void complaint() {
   if (!Initialize()) {
      cout << "Initialize winsock failed" <<
endl;

      return ;
   }
   SOCKET s;
   s = socket(AF_INET, SOCK_STREAM, 0);
   if (s == INVALID_SOCKET) {
      cout << "invalid socket created" << endl;
      return ;
   }

   int port = 12345;
   string serveraddress = "127.0.0.1";
   sockaddr_in serveraddr;
   serveraddr.sin_family = AF_INET;
   serveraddr.sin_port = htons(port);
   inet_pton(AF_INET, serveraddress.c_str(),
&(serveraddr.sin_addr));

   if (connect(s, reinterpret_cast
<sockaddr*>(&serveraddr), sizeof(serveraddr))
== SOCKET_ERROR) {
      cout << "not able to connect to server" <<
endl;
      closesocket(s);
      WSACleanup();
      return ;
   }

   cout << "Welcome to feedback/complaint
section." << endl;

   thread senderthread(SendMsg, s);
   thread receiver(ReceiveMsg, s);
```

```
    senderthread.join();
    receiver.join();


}



void viewFeedbackHistory() {
    const string chatHistoryFile =
"C:\\Users\\dsami\\source\\repos\\server\\server\
\customer_messages.txt";

    while (true) {
        ifstream infile(chatHistoryFile);
        if (!infile.is_open()) {
            cout << "No feedback history found.\n";
            return;
        }

        // Display all feedback
        string line;
        cout << "\nFeedback History:\n";
        while (getline(infile, line)) {
            cout << line << endl;
        }
        infile.close();

        cout << "\nEnter the Serial Number (S.N)
of the feedback you want to reply to, or type
'menu' to return to the menu: ";
        string input;
        cin >> input;

        if (input == "menu") {
            cout << "Returning to the menu...\n";
            clearScreen();
            return;
        }

        int serialNumber;
        try {
            serialNumber = stoi(input);
        }
        catch (...) {
            cout << "Invalid input. Please enter a
valid serial number or type 'menu'.\n";
            continue;
        }

        infile.open(chatHistoryFile);
        if (!infile.is_open()) {
            cout << "Error reopening feedback
history file.\n";
            return;
        }

        bool found = false;
        string selectedFeedback;
        while (getline(infile, line)) {
            if (line.find(to_string(serialNumber) +
".") != string::npos) {
                selectedFeedback = line;
                found = true;
                break;
            }
        }
        infile.close();

        if (!found) {
            cout << "No feedback found with Serial
Number " << serialNumber << ".\n";
            continue;
        }

        cout << "\nSelected Feedback:\n" <<
selectedFeedback << endl;
        cout << "Enter your reply: ";
        cin.ignore();
        string reply;
        getline(cin, reply);

        // Sending the reply to the server
        if (!Initialize()) {
            cout << "Failed to initialize Winsock.
Reply not sent.\n";
            return;
        }

        SOCKET s = socket(AF_INET,
SOCK_STREAM, 0);
        if (s == INVALID_SOCKET) {
            cout << "Failed to create socket. Reply
not sent.\n";
            WSACleanup();
            return;
        }

        const int port = 12345;
        const string serverAddress = "127.0.0.1";

        sockaddr_in serverAddr = {};
        serverAddr.sin_family = AF_INET;
        serverAddr.sin_port = htons(port);
        inet_pton(AF_INET, serverAddress.c_str(),
&serverAddr.sin_addr);

        if (connect(s, (sockaddr*)&serverAddr,
sizeof(serverAddr)) == SOCKET_ERROR) {
            cout << "Failed to connect to server.
Reply not sent.\n";
            closesocket(s);
            WSACleanup();
            clearScreen();
            return;
        }

        string fullReply = "Reply to Feedback " +
to_string(serialNumber) + ": " + reply;
        if (send(s, fullReply.c_str(), fullReply.size(),
0) == SOCKET_ERROR) {
            cout << "Failed to send reply to the
server.\n";
            clearScreen();
        }
        else {
            cout << "Reply sent successfully.\n";
        }

        closesocket(s);
        WSACleanup();
    }
}


void checkFeedbackComplaints() {
    cout << "\nFeedback/Complaints:\n";
    viewFeedbackHistory();
}



void seeResponses() {
    ifstream
feedbackFile("C:\\Users\\dsami\\source\\repos\\s
erver\\server\\customer_messages.txt");
    ifstream
responseFile("C:\\Users\\dsami\\source\\repos\\s
erver\\server\\seller_responses.txt");

    if (!feedbackFile.is_open()) {
        cerr << "Error: Unable to open
customer_messages.txt at the specified path." <<
endl;
    }
    if (!responseFile.is_open()) {
        cerr << "Error: Unable to open
seller_responses.txt at the specified path." <<
endl;
    }


    string feedbackLine, responseLine;
    vector<string> responses;

    // Read all responses into a vector for easy
searching
    while (getline(responseFile, responseLine)) {
        responses.push_back(responseLine);
    }

    // Process each feedback line by line
    while (getline(feedbackFile, feedbackLine)) {
        // Extract the serial number from the
feedback
        size_t pos = feedbackLine.find(". ");
        if (pos != string::npos) {
            try {
                int serialNumber =
stoi(feedbackLine.substr(0, pos));

                // Display the feedback
                cout << "Feedback #" <<
serialNumber << ": " <<
feedbackLine.substr(pos + 2) << endl;

                // Search for a corresponding response
                bool responseFound = false;
                for (const auto& response : responses)
{
                    string searchString = "Reply to
Feedback " + to_string(serialNumber) + ":";
                    if (response.find(searchString) !=
string::npos) {
                        // Extract and display the actual
response message
                        cout << "Response: " <<
response.substr(response.find(":") + 2) << endl;
                        responseFound = true;
                        break;
                    }
                }

                // If no response is found, indicate it
                if (!responseFound) {
                    cout << "Response: This
feedback/complaint is not responded." << endl;
                }

            }
            catch (...) {
                cout << "Error processing feedback
line: " << feedbackLine << endl;
                continue;
            }
        }
    }

    feedbackFile.close();
    responseFile.close();
}

void checkManageStocks() {
    vector<string> categories = {
        "FoodAndBeverages",
        "Clothes",
        "Stationery",
        "Electronics",
        "Books"
    };

    for (const auto& category : categories) {
        string fileName = category + ".txt";

        // Read items from the file
        ifstream infile(fileName);
        if (!infile.is_open()) {
            cout << "Error: Could not open " <<
fileName << "\n";
            continue;
        }

        map<string, pair<double, int>>
categoryItems; // Store item name, price, and
stock
        string itemName;
        double price;
        int stock;

        cout << "\nManaging stocks for category: "
<< category << "\n";
        cout << "----------------------------------------
--\n";
        cout << left << setw(20) << "Item Name"
<< setw(15) << "Price (Rs.)" << setw(10) <<
"Stock" << "\n";
        cout << "----------------------------------------
--\n";

        while (infile >> itemName >> price >>
stock) {
            categoryItems[itemName] = { price,
stock };
            cout << left << setw(20) << itemName
                << setw(15) << price
                << setw(10) << stock << "\n";
        }
        infile.close();

        // Menu for managing stocks and prices
        while (true) {
            cout << "\nOptions:\n";
            cout << "1. Edit stock\n";
            cout << "2. Edit price\n";
            cout << "3. Add new item\n";
            cout << "4. Go to next category\n";
            cout << "5. Return to the menu\n";
            cout << "Enter your choice: ";
            int choice;
            cin >> choice;

            if (choice == 5) {
                clearScreen();
                cout << "Returning to the main
menu...\n";
                return;
            }

            if (choice == 4) break;



            cout << "Enter item name: ";
            cin >> ws;
            getline(cin, itemName);

            if (categoryItems.find(itemName) ==
categoryItems.end()) {
                cout << "Error: Item \"" <<
itemName << "\" not found.\n";
                continue;
            }

            switch (choice) {
            case 1: { // Edit stock
                cout << "Current stock: " <<
categoryItems[itemName].second << "\n";
                cout << "Enter new stock: ";
                cin >> stock;
                categoryItems[itemName].second =
stock;
                break;
            }
            case 2: { // Edit price
                cout << "Current price: Rs." <<
categoryItems[itemName].first << "\n";
                cout << "Enter new price: Rs.";
                cin >> price;
                categoryItems[itemName].first =
price;
                break;
            }


            case 3: {
                cout << "Enter new item name: ";
                cin >> ws;
                getline(cin, itemName);

                if (categoryItems.find(itemName) !=
categoryItems.end()) {
                    cout << "Error: Item \"" <<
itemName << "\" already exists.\n";
                    continue;
                }
            cout << "Enter price (Rs.): ";
            while (!(cin >> price) || price < 0) {
                clearScreen();
                cout << "Invalid input. Please enter
a valid price: ";
                cin >> price;
            }

            cout << "Enter stock: ";
            while (!(cin >> stock) || stock < 0) {
                clearScreen();
                cout << "Invalid input. Please enter
a valid stock quantity: ";
                cin >> stock;
            }

            categoryItems[itemName] = { price,
stock };

            ofstream outfile(fileName, ios::app); //
Append mode
            if (!outfile.is_open()) {
                cout << "Error: Could not write to
" << fileName << "\n";
                continue;
            }
            outfile << itemName << " " << price
<< " " << stock << "\n";
            outfile.close();

            cout << "Item \"" << itemName <<
"\" added successfully.\n";
            continue;
        }
        default:
            cout << "Invalid choice. Try
again.\n";
        }

        ofstream outfile(fileName);
        if (!outfile.is_open()) {
            cout << "Error: Could not write to "
<< fileName << "\n";
            continue;
        }

        for (const auto& entry : categoryItems) {
            outfile << entry.first << " " <<
entry.second.first << " " <<
entry.second.second << "\n";
        }
        outfile.close();
        cout << "Changes saved for " <<
category << ".\n";
    }
  }
```

```
    cout << "\nStock management complete.\n";
}

void sellermenu() {
    while (true) {
        cout << "\nSeller Menu:\n";
        cout << "1. Check/Manage Stocks\n";
        cout << "2. Check
Feedback/Complaints\n";
        cout << "3. View Transaction History\n";
        cout << "4. Return Login page\n";
        cout << "5. Exit the program\n";
        cout << "Enter your choice: ";
        int choice;
        cin >> choice;

        switch (choice) {
        case 1:
            clearScreen();
            checkManageStocks();
            break;
        case 2:
            clearScreen();
            checkFeedbackComplaints();
            break;
        case 3:
            clearScreen();
            cout << "Transaction History" << endl;
            viewTransactionHistory();
            break;
        case 4:
            clearScreen();
            cout << "Returning to login page..\n";
            return;
        case 5:
            clearScreen();
            cout << "Exiting.. ";
            exit(0);
        default:
            cout << "Invalid choice. Please try
again.\n";
        }
    }
}

void selectCategory() {
    customer myCustomer;

    shop* shop = nullptr;
```

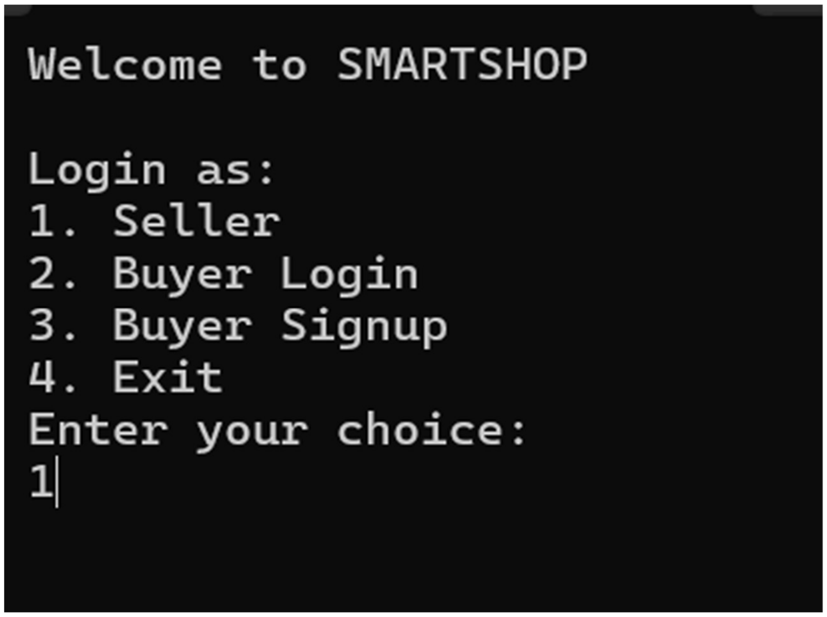
```
while (true) {
    cout << "\nSelect Category:\n";
    cout << "1. Food and Beverages\n";
    cout << "2. Clothes\n";
    cout << "3. Stationery\n";
    cout << "4. Electronics\n";
    cout << "5. Books\n";
    cout << "6. View Cart\n";
    cout << "7. Return to Menu\n";
    cout << "Enter your choice:";
    int choice;
    cin >> choice;

    switch (choice) {
    case 1: {
        clearScreen();

        FoodAndBeverages foodShop;
        shop = &foodShop;
        shop->show();
        shop->select(myCustomer);
        break;
    }
    case 2: {
        clearScreen();
        Clothes clothesShop;
        shop = &clothesShop;
        shop->show();
        shop->select(myCustomer);
        break;
    }
    case 3: {
        clearScreen();
        Stationery stationeryShop;
        shop = &stationeryShop;
        shop->show();
        shop->select(myCustomer);
        break;
    }
    case 4: {
        clearScreen();
        Electronics electronicsShop;
        shop = &electronicsShop;
        shop->show();
        shop->select(myCustomer);
        break;
    }
    case 5: {
        clearScreen();
        Books booksShop;
        shop = &booksShop;
        shop->show();
        shop->select(myCustomer);
        break;
    }
        case 6:
            clearScreen();
            myCustomer.viewCart();
            break;
        case 7:
            clearScreen();
            cout << "Returning to menu...\n";
            return ;
        default:
            clearScreen();
            cout << "Invalid option. Please try
again.\n";
        }


    }
}

void login() {
    while (true) {
        cout << "Login as:\n";
        cout << "1. Seller\n2. Buyer Login\n3.
Buyer Signup\n4. Exit\n";
        cout << "Enter your choice:" << endl;
        int choice;
        cin >> choice;

        if (choice == 1) {
            clearScreen();
            if (sellerLogin()) {
                sellermenu();
            }
        }
        else if (choice == 2) {
            clearScreen();
            string username;
            if (buyerLogin(username)) {

                customer c;

                shop* s = nullptr;
                while (true) {
                    if (!s) {
                        shop temp;
                        temp.showMenu();
                    }

                    cout << "Select an option (1-6): ";
                    int val;
                    cin >> val;

                    if (val == 1) {
                        clearScreen();
                        selectCategory();
                    }
                else if (val == 3) {
                    clearScreen();
                    complaint();
                }
                else if (val == 4) {
                    clearScreen();
                    seeResponses();
                }
                else if (val == 5) {
                    clearScreen();
                    cout << "Returning to Login
Page...\n";
                    login();

                }
                else if (val == 6) {
                    return;
                }
                else if (val == 2) {
                    clearScreen();
                    c.printBill();

                    // Offer a choice after billing
                    cout << "What would you like to
do next?\n";
                    cout << "1. Return to Menu\n2.
Exit Program\n";
                    int postBillChoice;
                    cin >> postBillChoice;

                    if (postBillChoice == 1) {
                        clearScreen();
                        continue; // Return to menu
                    }
                    else if (postBillChoice == 2) {
                        clearScreen();
                        cout << "Exiting the program.
Thank you for shopping!\n";
                        return; // Exit the program
                    }
                    else {
                        clearScreen();
                        cout << "Invalid choice.
Exiting by default.\n";
                        return;
                    }
                }
                else {
                    cout << "Invalid choice. Please
try again.\n";
                }
            }
            return;
        }
    }
    else if (choice == 3) {
            clearScreen();
            buyerSignup();
        }
        else if (choice == 4) {
            clearScreen();
            cout << "Exiting...\n";
            return;
        }
        else {
            cout << "Invalid choice. Please try
again.\n";
        }
    }
};


int main() {
    cout << "Welcome to SMARTSHOP" << endl
<< endl;

    loadCredentials();
    login();


    WSACleanup();


            return 0;
}
```

10. OUTPUT



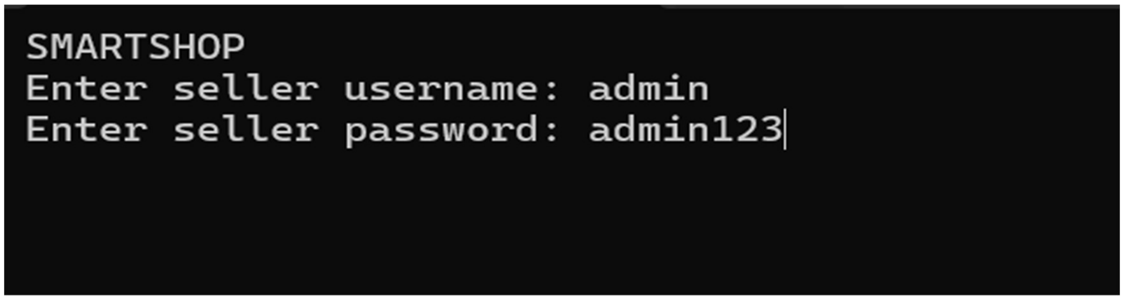
```
Server program started
Server is listening on port: 12345
```

Figure 10- 1 Server Side program output



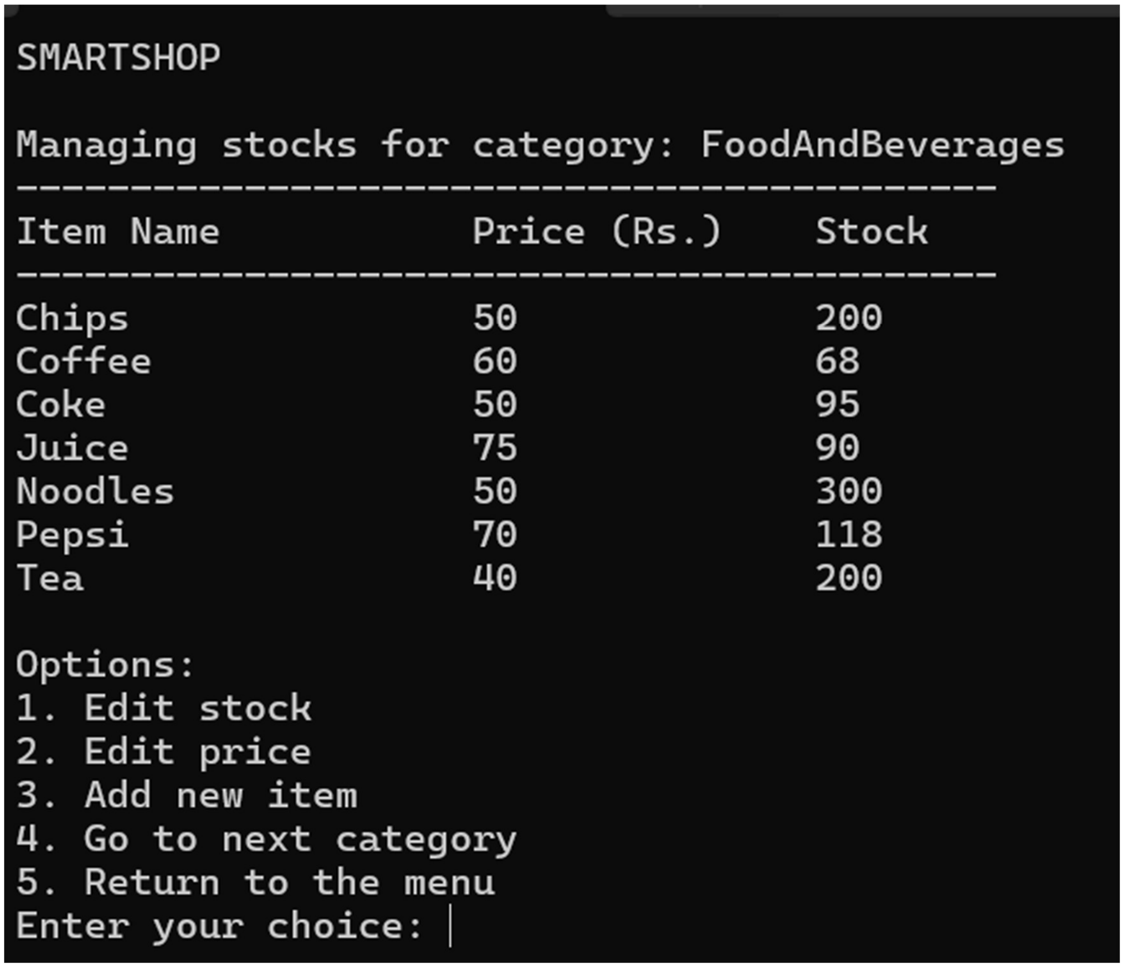
```
Welcome to SMARTSHOP

Login as:
1. Seller
2. Buyer Login
3. Buyer Signup
4. Exit
Enter your choice:
1
```

Figure 10- 2 Login Page



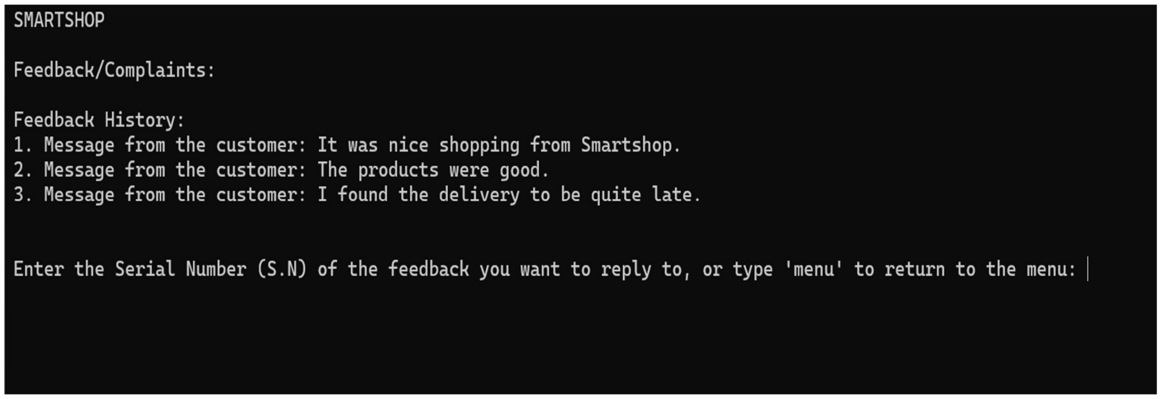
```
SMARTSHOP
Enter seller username: admin
Enter seller password: admin123
```

Figure 10- 31 Seller username and password



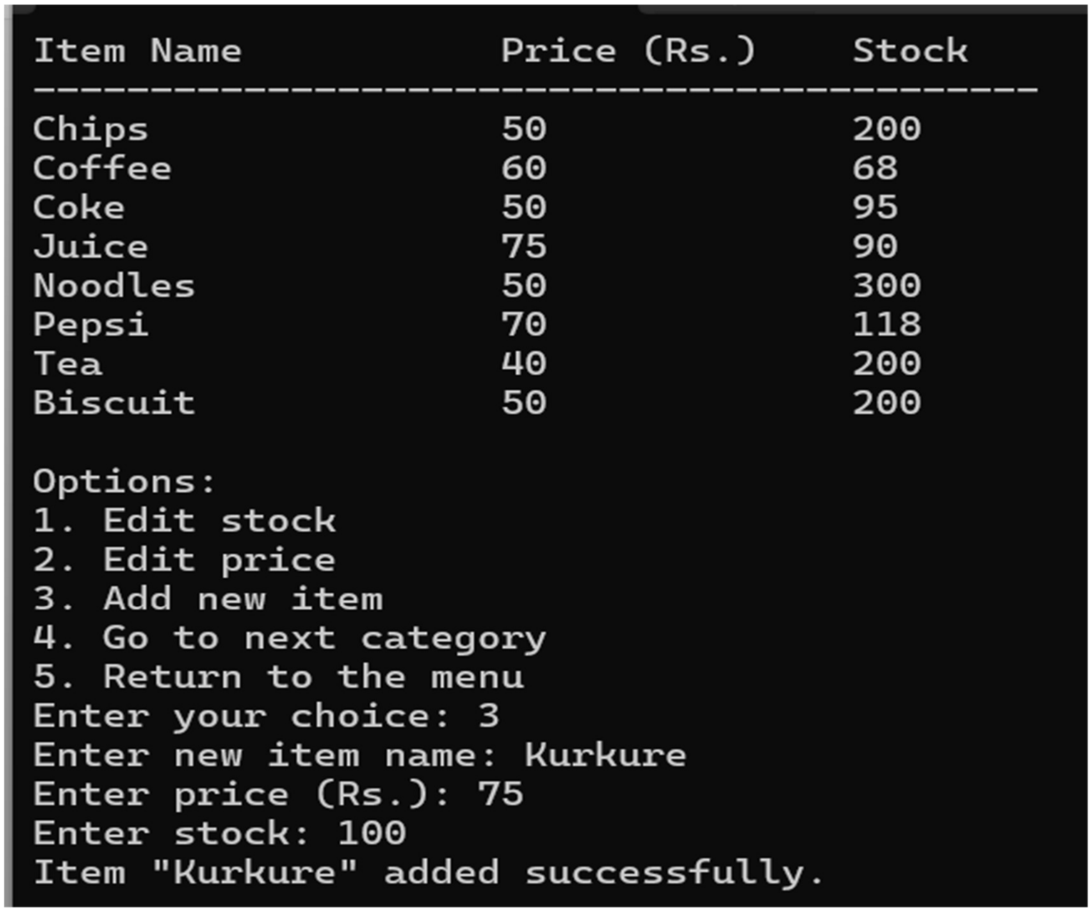
```
SMARTSHOP

Managing stocks for category: FoodAndBeverages
-------------------------------------------
Item Name           Price (Rs.)    Stock
-------------------------------------------
Chips               50             200
Coffee              60             68
Coke                50             95
Juice               75             90
Noodles             50             300
Pepsi               70             118
Tea                 40             200

Options:
1. Edit stock
2. Edit price
3. Add new item
4. Go to next category
5. Return to the menu
Enter your choice:
```

Figure 10- 4Stock Management



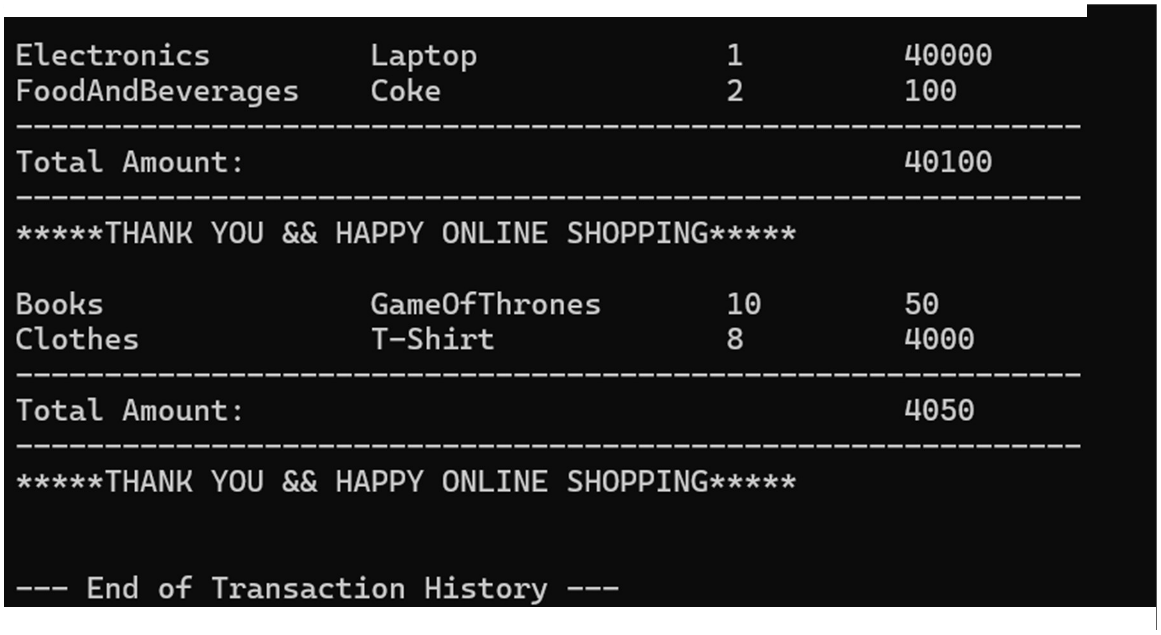
```
SMARTSHOP

Feedback/Complaints:

Feedback History:
1. Message from the customer: It was nice shopping from Smartshop.
2. Message from the customer: The products were good.
3. Message from the customer: I found the delivery to be quite late.


Enter the Serial Number (S.N) of the feedback you want to reply to, or type 'menu' to return to the menu:
```

Figure 10- 5 View and Reply Feedbacks/Complaints



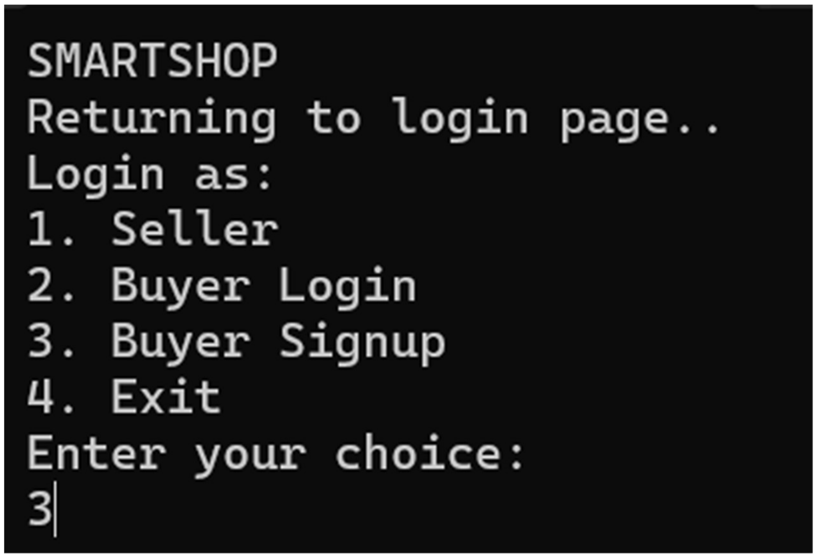
```
Item Name                   Price (Rs.)     Stock
-------------------------------------------------
Chips                       50              200
Coffee                      60              68
Coke                        50              95
Juice                       75              90
Noodles                     50              300
Pepsi                       70              118
Tea                         40              200
Biscuit                     50              200

Options:
1. Edit stock
2. Edit price
3. Add new item
4. Go to next category
5. Return to the menu
Enter your choice: 3
Enter new item name: Kurkure
Enter price (Rs.): 75
Enter stock: 100
Item "Kurkure" added successfully.
```

Figure 10- 6 Adding new stock



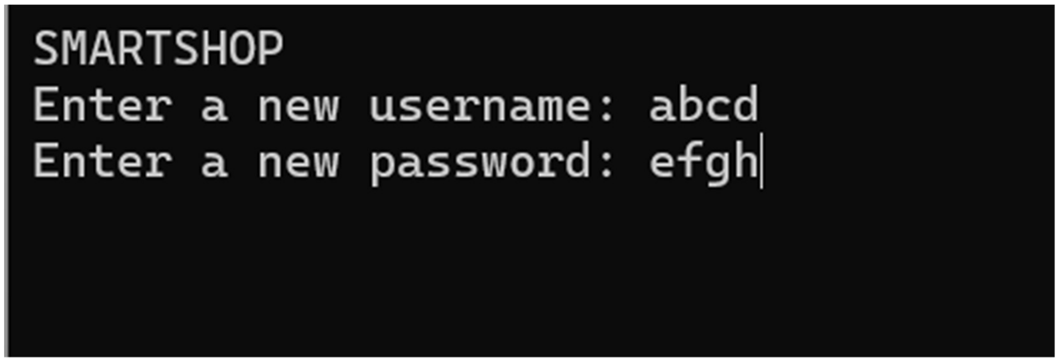
```
Electronics         Laptop              1         40000
FoodAndBeverages    Coke                2         100
------------------------------------------------------------
Total Amount:                                     40100
------------------------------------------------------------
*****THANK YOU && HAPPY ONLINE SHOPPING*****

Books               GameOfThrones       10        50
Clothes             T-Shirt             8         4000
------------------------------------------------------------
Total Amount:                                     4050
------------------------------------------------------------
*****THANK YOU && HAPPY ONLINE SHOPPING*****


--- End of Transaction History ---
```

Figure 10- 7 Transaction History



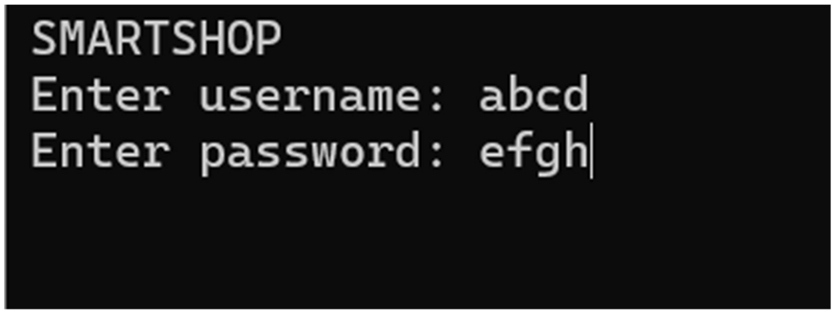
```
SMARTSHOP
Returning to login page..
Login as:
1. Seller
2. Buyer Login
3. Buyer Signup
4. Exit
Enter your choice:
3
```

Figure 10- 8 Buyer Signup



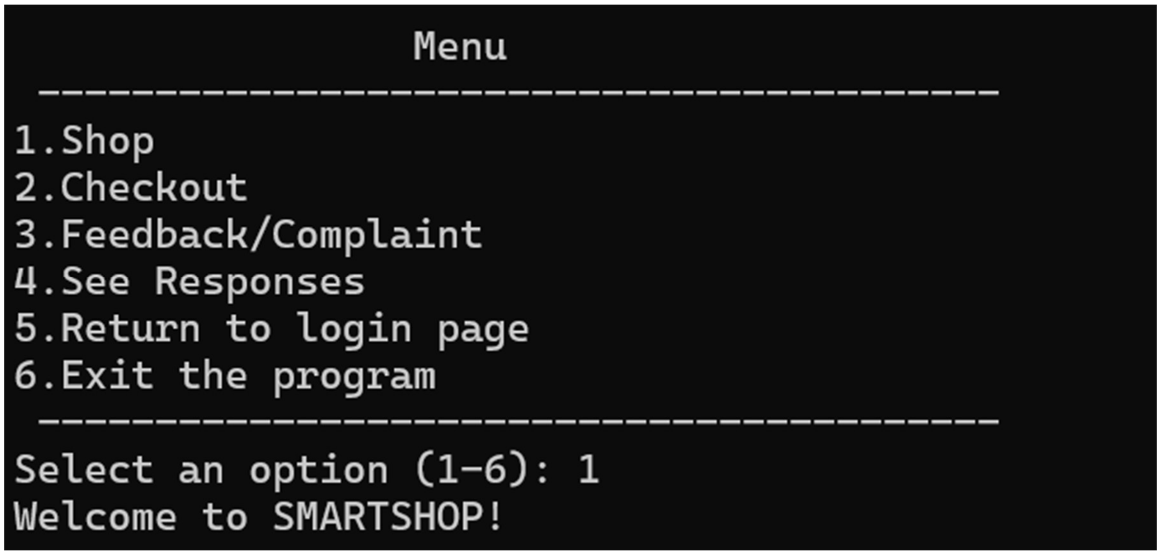
```
SMARTSHOP
Enter a new username: abcd
Enter a new password: efgh
```

Figure 10- 9 Signup Page



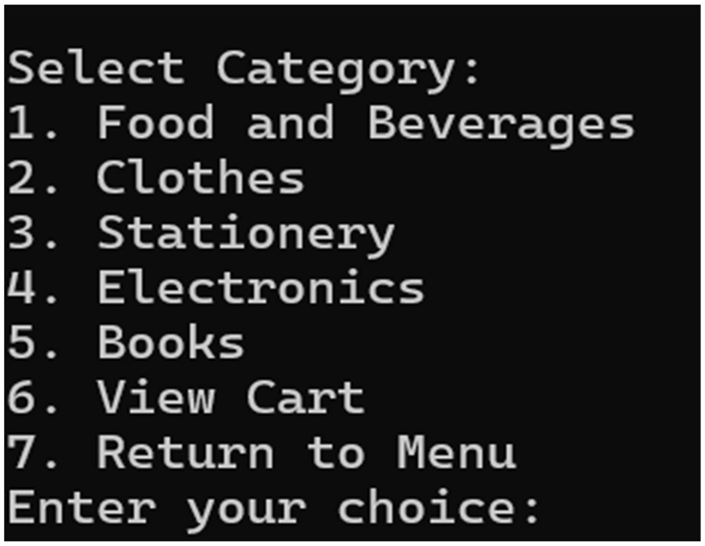
```
SMARTSHOP
Enter username: abcd
Enter password: efgh
```

Figure 10- 10 Buyer Login



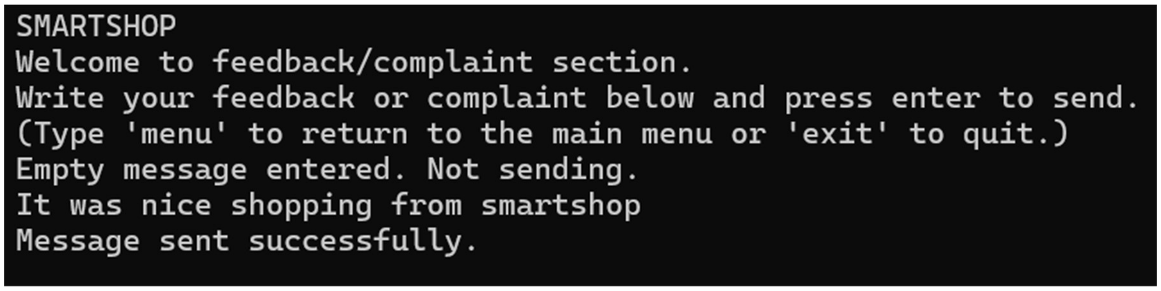
```
                 Menu
 ----------------------------------------
1.Shop
2.Checkout
3.Feedback/Complaint
4.See Responses
5.Return to login page
6.Exit the program
 ----------------------------------------
Select an option (1-6): 1
Welcome to SMARTSHOP!
```

Figure 10- 11 Buyer Menu



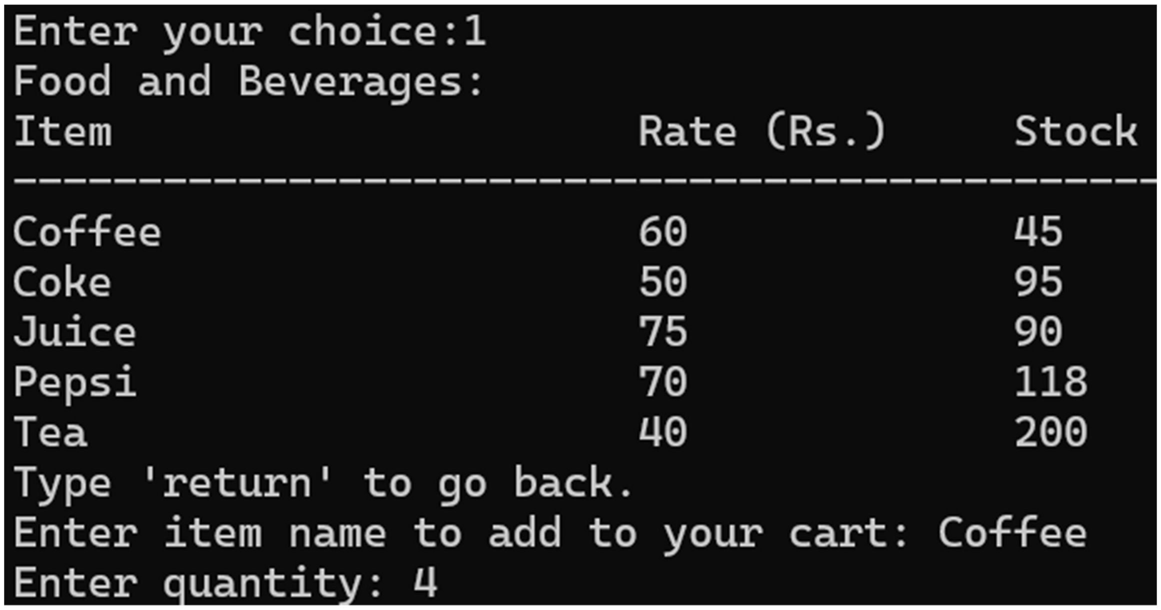
```
Select Category:
1. Food and Beverages
2. Clothes
3. Stationery
4. Electronics
5. Books
6. View Cart
7. Return to Menu
Enter your choice:
```

Figure 10- 12 Items in the Shop



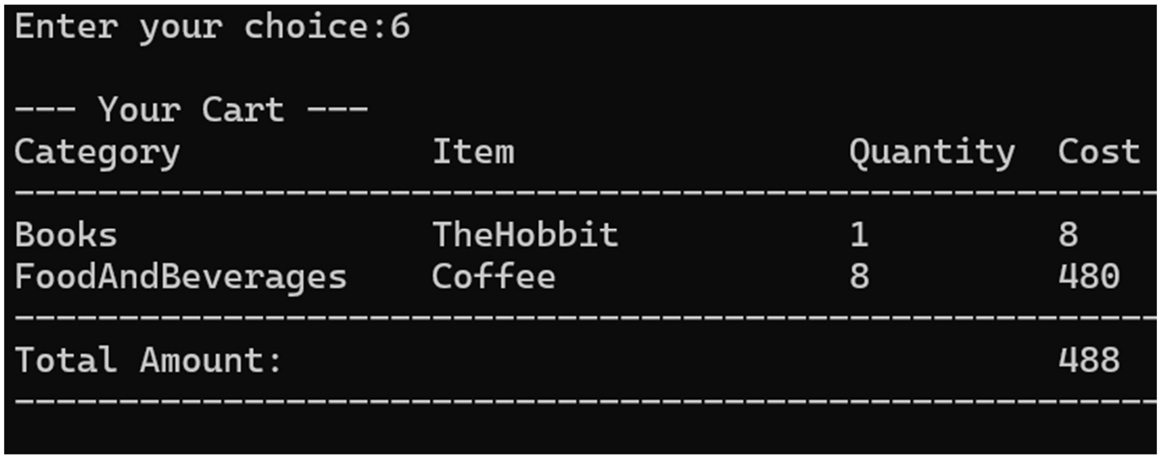
```
SMARTSHOP
Welcome to feedback/complaint section.
Write your feedback or complaint below and press enter to send.
(Type 'menu' to return to the main menu or 'exit' to quit.)
Empty message entered. Not sending.
It was nice shopping from smartshop
Message sent successfully.
```

Figure 10- 13 Feedback/Complaint



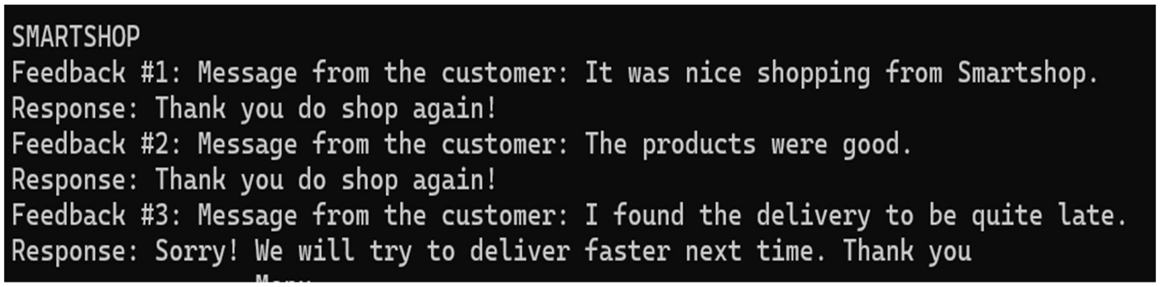
```
Enter your choice:1
Food and Beverages:
Item                     Rate (Rs.)     Stock
-----------------------------------------------
Coffee                   60             45
Coke                     50             95
Juice                    75             90
Pepsi                    70             118
Tea                      40             200
Type 'return' to go back.
Enter item name to add to your cart: Coffee
Enter quantity: 4
```

Figure 10- 14 Adding the items in the cart



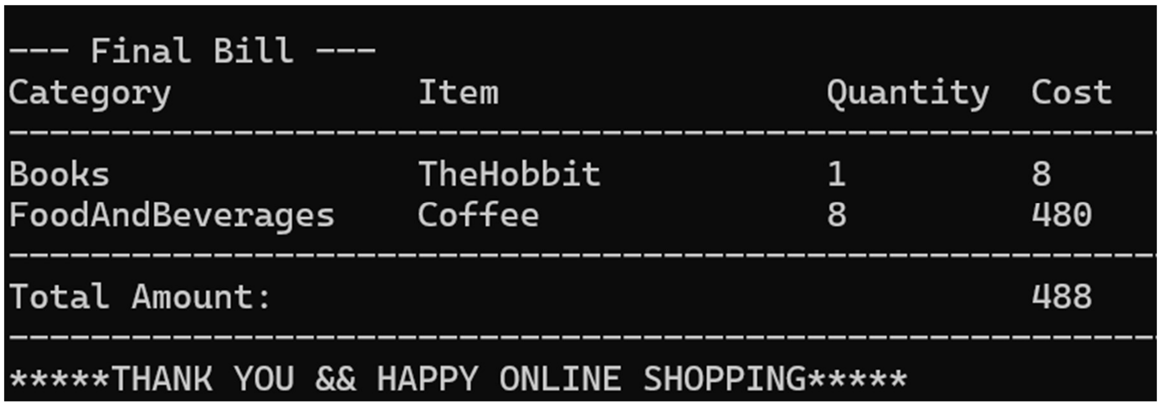
```
Enter your choice:6

--- Your Cart ---
Category            Item                Quantity  Cost
-------------------------------------------------------
Books               TheHobbit           1         8
FoodAndBeverages    Coffee              8         480
-------------------------------------------------------
Total Amount:                                     488
-------------------------------------------------------
```

Figure 10- 15 Cart



```
SMARTSHOP
Feedback #1: Message from the customer: It was nice shopping from Smartshop.
Response: Thank you do shop again!
Feedback #2: Message from the customer: The products were good.
Response: Thank you do shop again!
Feedback #3: Message from the customer: I found the delivery to be quite late.
Response: Sorry! We will try to deliver faster next time. Thank you
```

Figure 10- 16 View Responses



```
--- Final Bill ---
Category            Item                Quantity  Cost
-------------------------------------------------------
Books               TheHobbit           1         8
FoodAndBeverages    Coffee              8         480
-------------------------------------------------------
Total Amount:                                     488
-------------------------------------------------------
*****THANK YOU && HAPPY ONLINE SHOPPING*****
```

Figure 10- 17 Final Bill

9. REFERENCES

[1] Sunila Gorey, “Scope of eCommerce: The Future of Online Shopping” [Online]. Available: https://webandcrafts.com/blog/scope-of-ecommerce [Accessed :21 August 2024]

[2] Ramesh Vaidhya, “Online Shopping in Nepal: Prefereces and Problems” [Online]. Available:https://www.researchgate.net/publication/340072267_Online_Shopping_in_Nepal_Preferences_and_Problems [Accessed: 18 August 2024]

[3] saad0510, “Online Shopping System” [Online]. Available: https://github.com/saad0510/online-shopping-system/tree/main/Server/Headers. [Accessed: 15 August 2024]

[4] Geeksforgeeks, “Implementing Interactive Online Shopping in C++” [Online]. Available: https://www.geeksforgeeks.org/implementing-interactive-online-shopping-in-c/ [Accessed: 15 August 2024]