

**BOOKSTORE MANAGEMENT**

**SYSTEM**



**DONE BY:**

Naman Bhansali - 1032230313

Shloka Shetty-1032231290

Aaditi Salunkhe-1032231221

|  |  |  |
| --- | --- | --- |
| **S.no** | **Table of contents** | **Pg.no** |
| 1. | List of abbreviations | 1 |
| 2. | List of Figures | 1 |
| 3. | Introduction  Problem statement | 2  3 |
| 4. | List of Object-Oriented Concepts (OOP) Used | 5 |
| 5. | Screenshots of Output and code | 9 |
| 6. | Conclusion | 26 |
| 7. | References | 27 |

**1. List of Abbreviations**

OOP – Object-Oriented Programming

Rs – Indian Rupees

COD – Cash on Delivery

**2. List of Figures**

Figure 1: Welcome screen

Figure 2: Main menu after login

Figure 3: Displaying available books

Figure 4: Cart overview

Figure 5: Payment confirmation screen

**3.. Introduction**

This report presents a C++ bookstore management application that allows users to browse, rent, and purchase books. The bookstore application supports user authentication, payment, order history tracking, and other essential functionalities such as wishlist management, reviews, and customer care.

The project emphasizes user experience with a clear console interface, cute messages, and appropriate delays to enhance interaction. File handling is used to maintain a record of user order history.

**4. Problem Statement**

The goal of the project is to develop a comprehensive bookstore application that addresses the following requirements:

User Authentication: Allow users to register, login, and logout securely.

Browse and Manage Books: Users can add books to the cart, rent, or purchase them.

Wishlist and Order History: Track user activity such as past orders and wishlist.

Payment System: Supports the COD, debit, and credit card payments.

Review System: Users can leave reviews and ratings for books.

Customer Care: A basic system to log user queries for support.

File Handling: Save order history to an external text file for tracking.

**5. List of Object-Oriented Concepts (OOP) Used**

**1. Encapsulation:**

Encapsulates data using structures like Book and User to represent individual books and users with relevant attributes such as price, title, cart, and history.

**2. Classes and Objects:**

The Bookstore class manages the core operations such as user authentication, book management, and order processing. Instances of Book and User act as objects to represent data.

**3. Abstraction:**

Functions like registerUser(), rentBook(), and displayBooks() hide internal logic from the user, providing a clean interface for interactions.

**4. Inheritance (Conceptual)**:

Although inheritance isn't directly implemented, related objects like User and Book could be extended in future designs for specialized behavior (e.g., different types of users or premium books).

5. **Polymorphism**:

Achieved conceptually through overloaded behavior like dynamic menu options depending on whether a user is logged in or not.

**6. File Handling:**

The application saves order history to a text file using ofstream to persist data.

**7. Constructor Usage:**

Parameterized constructors are used for Book and User objects to initialize attributes at the time of creation.

-

**Implementation of code**

#include <iostream>

#include <vector>

#include <string>

#include <thread>

#include <chrono>

#include <iomanip>

#include <map>

#include <fstream>

using namespace std;

// Book structure

struct Book {

string title;

string author;

string code;

string genre;

string review;

double rating;

string summary;

string status; // Available or Rented

double price;

Book(string t, string a, string c, string g, string s, double p)

: title(t), author(a), code(c), genre(g), review(""), rating(0), summary(s), status("Available"), price(p) {}

};

// User structure

struct User {

string username;

string password;

vector<Book> cart;

vector<Book> wishlist;

vector<Book> orderHistory;

double totalSpent;

User() : username(""), password(""), totalSpent(0) {}

User(string u, string p) : username(u), password(p), totalSpent(0) {}

};

class Bookstore {

private:

vector<Book> books;

map<string, User> users;

User\* loggedInUser;

public:

Bookstore() : loggedInUser(nullptr) {

books.emplace\_back("The Great Gatsby", "F. Scott Fitzgerald", "BK101", "Fiction", "A classic novel set in the 1920s.", 500.00);

books.emplace\_back("1984", "George Orwell", "BK102", "Dystopian", "A story about totalitarianism and surveillance.", 400.00);

books.emplace\_back("To Kill a Mockingbird", "Harper Lee", "BK103", "Fiction", "A novel about racial injustice.", 300.00);

books.emplace\_back("The Catcher in the Rye", "J.D. Salinger", "BK104", "Fiction", "A story about teenage alienation.", 350.00);

books.emplace\_back("Moby Dick", "Herman Melville", "BK105", "Adventure", "A novel about the obsession with a white whale.", 450.00);

}

void clearConsole() {

#ifdef \_WIN32

system("CLS");

#else

system("clear");

#endif

}

void delay(int seconds) {

this\_thread::sleep\_for(chrono::seconds(seconds));

}

void displayWelcomeMessage() {

clearConsole();

cout << endl << endl;

string welcome1 = "WELCOME TO THE";

string welcome2 = "BOOKSTORE SYSTEM";

int width = 50;

cout << string((width - welcome1.length()) / 2, ' ') << welcome1 << endl;

cout << string((width - welcome2.length()) / 2, ' ') << welcome2 << endl;

cout << string(width, '-') << endl;

cout << "Your one-stop shop for all your reading needs!" << endl;

cout << string(width, '-') << endl;

delay(2);

}

void registerUser() {

string username, password;

cout << endl << "Enter a username: ";

cin >> username;

cout << "Enter a password: ";

cin >> password;

if (users.find(username) != users.end()) {

cout << endl << "Username already exists! Please try again.\n";

return;

}

users[username] = User(username, password);

cout << endl << "Registration successful! Redirecting to login...\n";

delay(2);

}

bool loginUser() {

string username, password;

cout << endl << "Enter username: ";

cin >> username;

cout << "Enter password: ";

cin >> password;

auto it = users.find(username);

if (it != users.end() && it->second.password == password) {

loggedInUser = &it->second;

cout << "Login successful! Welcome, " << loggedInUser->username << "!\n";

return true;

} else {

cout << "Invalid credentials. Please try again.\n";

return false;

}

}

void displayBooks() {

cout << "\nAvailable Books:\n";

cout << left << setw(30) << "Title" << setw(20) << "Author" << setw(10) << "Code"

<< setw(15) << "Genre" << setw(10) << "Price" << setw(10) << "Status" << endl;

cout << string(100, '-') << endl;

for (const auto& book : books) {

cout << left << setw(30) << book.title << setw(20) << book.author << setw(10) << book.code

<< setw(15) << book.genre << setw(10) << fixed << setprecision(2) << book.price << setw(10) << book.status << endl;

}

cout << endl;

}

void addToCart() {

string code;

cout << "\nEnter the code of the book you want to add to the cart: ";

cin >> code;

for (const auto& book : books) {

if (book.code == code && book.status == "Available") {

loggedInUser->cart.push\_back(book);

cout << "Book added to cart!\n";

return;

}

}

cout << "Book not found or unavailable.\n";

}

void viewCart() {

clearConsole();

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

cout << left << setw(30) << "Title" << setw(20) << "Author" << setw(10) << "Code"

<< setw(15) << "Genre" << setw(10) << "Price" << endl;

cout << string(100, '-') << endl;

double totalPrice = 0;

for (const auto& book : loggedInUser->cart) {

cout << left << setw(30) << book.title << setw(20) << book.author << setw(10) << book.code

<< setw(15) << book.genre << setw(10) << fixed << setprecision(2) << book.price << endl;

totalPrice += book.price;

}

cout << "Total Price: " << fixed << setprecision(2) << totalPrice << " Rs\n";

cout << endl;

}

void rentBook() {

string code;

cout << "\nEnter the code of the book you want to rent: ";

cin >> code;

for (auto& book : books) {

if (book.code == code && book.status == "Available") {

book.status = "Rented";

loggedInUser->orderHistory.push\_back(book);

loggedInUser->totalSpent += book.price;

saveOrderHistory(); // Save order history to file

cout << "Book rented successfully!\n";

return;

}

}

cout << endl << "Book not found or unavailable.\n";

}

void returnBook() {

string code;

cout << "\nEnter the code of the book you want to return: ";

cin >> code;

for (auto& book : books) {

if (book.code == code && book.status == "Rented") {

book.status = "Available";

cout << "Book returned successfully!\n";

return;

}

}

cout << endl << "Book not found or was not rented.\n";

}

void addReviewAndRating() {

string code;

cout << "\nEnter the code of the book you want to review: ";

cin >> code;

for (auto& book : books) {

if (book.code == code) {

cout << "Enter your review: ";

cin.ignore(); // clear newline

getline(cin, book.review);

cout << "Enter your rating (0-5): ";

cin >> book.rating;

cout << "Review and rating added successfully!\n";

return;

}

}

cout << endl << "Book not found.\n";

}

void viewOrderHistory() {

clearConsole();

cout << "\nOrder History:\n";

cout << left << setw(30) << "Title" << setw(20) << "Author" << setw(10) << "Code"

<< setw(15) << "Genre" << setw(10) << "Price" << endl;

cout << string(100, '-') << endl;

for (const auto& book : loggedInUser->orderHistory) {

cout << left << setw(30) << book.title << setw(20) << book.author << setw(10) << book.code

<< setw(15) << book.genre << setw(10) << fixed << setprecision(2) << book.price << endl;

}

cout << endl;

}

void customerCare() {

string query;

cout << "\nEnter your query: ";

cin.ignore(); // clear newline

getline(cin, query);

cout << endl << "Thank you for your query. We will get back to you soon!\n";

}

void payment() {

cout << "\nProceeding to payment...\n";

delay(1);

double totalPrice = 0;

for (const auto& book : loggedInUser->cart) {

totalPrice += book.price;

}

cout << "Total amount: " << fixed << setprecision(2) << totalPrice << " Rs\n";

cout << "Payment successful! Thank you for your purchase!\n";

// Save cart to order history

for (const auto& book : loggedInUser->cart) {

loggedInUser->orderHistory.push\_back(book);

}

saveOrderHistory(); // Save order history to file

loggedInUser->cart.clear(); // Clear the cart after payment

}

void saveOrderHistory() {

ofstream outFile("order\_history.txt", ios::app); // Open file in append mode

if (outFile.is\_open()) {

for (const auto& book : loggedInUser->orderHistory) {

outFile << "User: " << loggedInUser->username << endl;

outFile << "Book Code: " << book.code << endl;

outFile << "Price: " << fixed << setprecision(2) << book.price << " Rs" << endl;

outFile << endl; // Space between different orders

}

outFile.close();

} else {

cout << "Unable to open file for saving order history.\n";

}

}

void logout() {

cout << "Logging out...\n";

delay(1);

loggedInUser = nullptr; // Reset logged-in user

clearConsole(); // Clear the console after logging out

}

User\* getLoggedInUser() {

return loggedInUser;

}

void pressEnterToContinue() {

cout << "\nPress Enter to continue...";

cin.ignore(); // clear newline from previous input

cin.get(); // Wait for user input

}

};

int main() {

Bookstore bookstore;

int choice;

bookstore.displayWelcomeMessage(); // Display the welcome message at the start

while (true) {

cout << "\nMain Menu:\n";

cout << "1. Register\n";

cout << "2. Login\n";

cout << "3. Exit\n";

cout << endl << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

bookstore.registerUser();

bookstore.pressEnterToContinue();

break;

case 2:

if (bookstore.loginUser()) {

while (true) {

bookstore.clearConsole();

cout << "\nUser Menu:\n";

cout << "1. Display Books\n";

cout << "2. Add to Cart\n";

cout << "3. View Cart\n";

cout << "4. Rent a Book\n";

cout << "5. Return a Book\n";

cout << "6. Add Review and Rating\n";

cout << "7. View Order History\n";

cout << "8. Customer Care\n";

cout << "9. Payment\n";

cout << "10. Logout\n";

cout << endl << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

bookstore.displayBooks();

bookstore.pressEnterToContinue();

break;

case 2:

bookstore.addToCart();

bookstore.pressEnterToContinue();

break;

case 3:

bookstore.viewCart();

bookstore.pressEnterToContinue();

break;

case 4:

bookstore.rentBook();

bookstore.pressEnterToContinue();

break;

case 5:

bookstore.returnBook();

bookstore.pressEnterToContinue();

break;

case 6:

bookstore.addReviewAndRating();

bookstore.pressEnterToContinue();

break;

case 7:

bookstore.viewOrderHistory();

bookstore.pressEnterToContinue();

break;

case 8:

bookstore.customerCare();

bookstore.pressEnterToContinue();

break;

case 9:

bookstore.payment();

bookstore.pressEnterToContinue();

break;

case 10:

bookstore.logout(); // Call logout method

break; // Return to the user menu

default:

cout << "Invalid choice. Please try again.\n";

bookstore.pressEnterToContinue();

break;

}

// If the user logs out, break out of the user menu loop

if (bookstore.getLoggedInUser() == nullptr) {

break;

}

}

}

break;

case 3:

cout << "Exiting the program. Goodbye!\n";

return 0;

default:

cout << "Invalid choice. Please try again.\n";

bookstore.pressEnterToContinue();

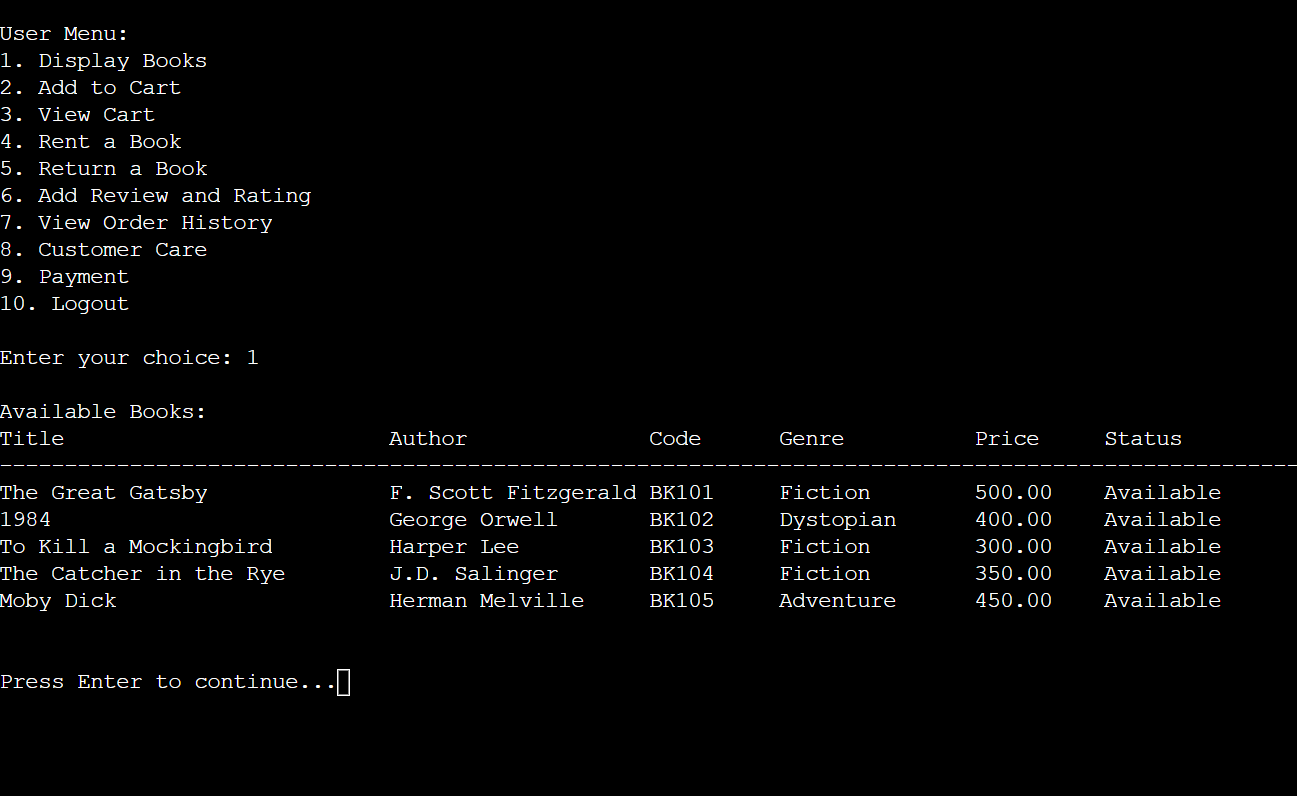
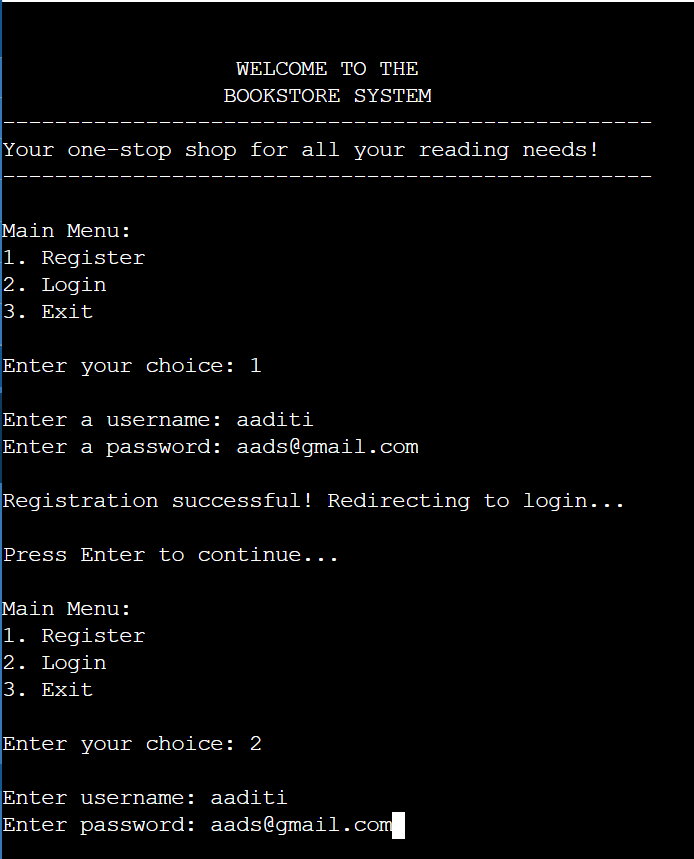
break;

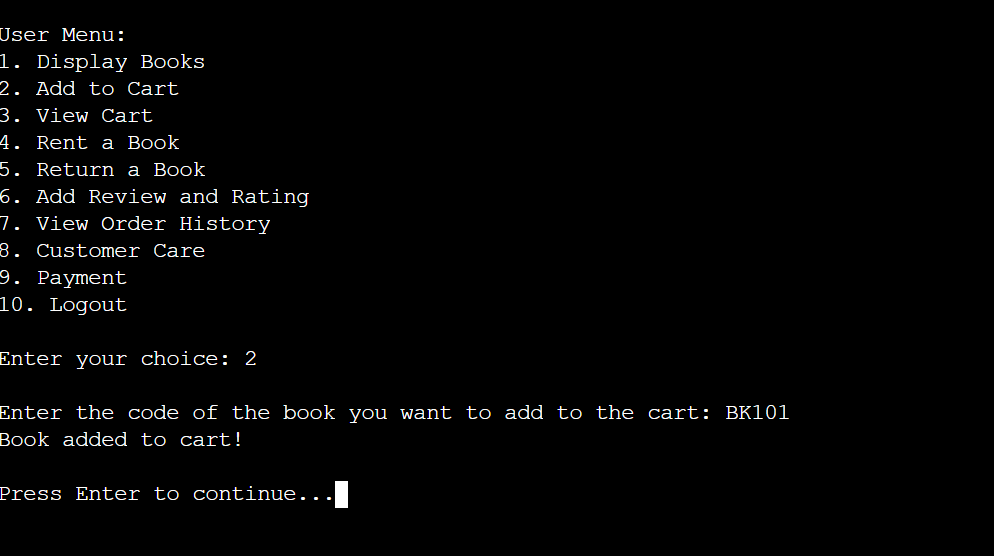
}

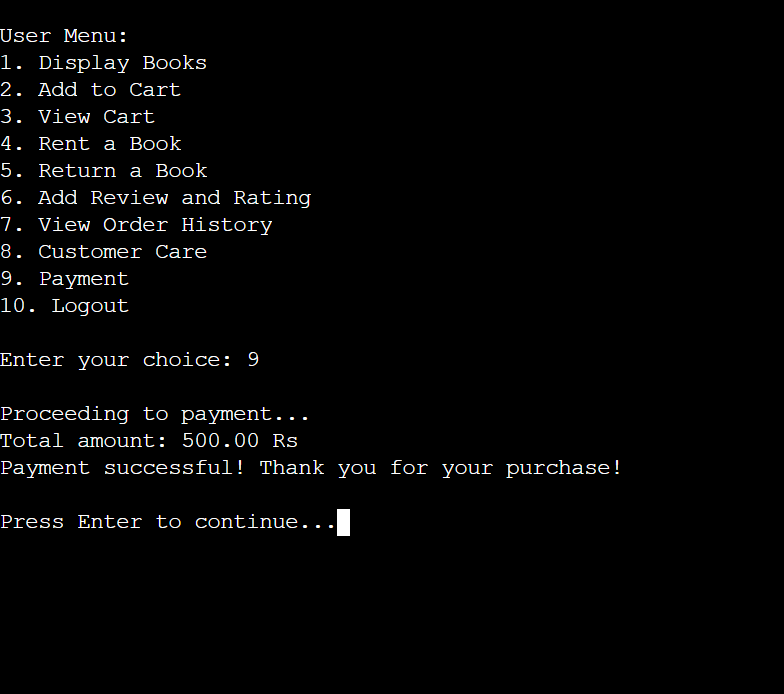
}

return 0;

}

**6. Screenshots of Output**





**7. Conclusion**

The bookstore management system developed in C++ fulfills the intended goals by providing essential features like user registration, book browsing, renting, reviews, and payments. The program showcases strong OOP principles such as encapsulation, abstraction, and file handling. The use of delays and clear console formatting ensures a friendly and interactive experience for users.

The bookstore can be extended in the future with more advanced features, such as personalized recommendations or integration with online payment gateways. Additionally, enhancing error handling and optimizing the codebase can make the system more robust.

**8. References**

C++ Documentation: https://cplusplus.com

File Handling in C++: GeeksforGeeks

OOP Principles: TutorialsPoint