E-COMMERCE SHOPPING CART

SUBMITTED BY

MAHESH KARREVULA – AP23110011446

NAGA MALLESWARA RAO NAGULAPATI – AP23110011411

MANOHAR SURAJ GADIPARTHI – AP23110011452

VISHNU ROHITH GOGINENI – AP23110011464

SUBMITTED TO

Mrs. Kavitha Rani Karnena

OOPS WITH C++ (CSE 202)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SRM UNIVERSITY–AP

NEERUKONDA, MANGALAGIRI, GUNTUR

ANDHRA PRADESH – 522 240

[NOVEMBER, 2024]

1. INTRODUCTION

In today's digital era, e-commerce platforms are vital for simplifying shopping experiences. This project simulates a miniature e-commerce system, combining user registration, login functionality, product catalog management, and a shopping cart feature.

The system is designed using C++ and object-oriented programming (OOP) principles, ensuring scalability and maintainability. It demonstrates the real-world application of classes and objects, encapsulation, and data abstraction.

This report discusses the design, implementation, functionality, and future prospects of the e-commerce system.

2. OBJECTIVES

The project's objectives include:

- Providing a secure, modular platform for shopping.
- Demonstrating practical applications of OOP concepts.
- Allowing seamless navigation for product browsing, cart operations, and checkout.

Through these objectives, the system aims to simplify online shopping workflows while emphasizing robust and reusable code.

3. SYSTEM OVERVIEW

The system operates through the following major components:

1. **User Management**: Handles user registration and login.
2. **Product Catalog**: Displays available products.
3. **Shopping Cart**: Manages product additions, removals, and total calculation.
4. **Checkout**: Processes payments and clears the cart post-purchase.

4. IMPLEMENTATION DETAILS

4.1. Product Class

The Product class defines the basic structure for storing product details.
Attributes:

- id: Stores the unique identifier for a product.
- name: Stores the name of the product.
- price: Stores the cost of the product.

Constructor: The constructor initializes a product with its id, name, and price. This ensures every product is uniquely identifiable with predefined properties.

Example:

```
Product laptop(1, "Laptop", 999.99);
cout << "Product: " << laptop.name << ", Price: $" << laptop.price << endl;
```

Here, a Product object named laptop is created with the ID 1, name "Laptop", and price 999.99. The object data can then be accessed using dot notation.

4.2. User Class

The User class manages login and registration functionalities.
Attributes:

- username: Stores the user's login name.
- password: Stores the user's password.

Methods:

- checkPassword(string inputPassword): Compares the input password with the stored password to verify credentials.

Code Example:

```
User user1("admin", "password123");
if (user1.checkPassword("password123")) {
    cout << "Login successful!" << endl;
} else {
    cout << "Invalid credentials!" << endl;
}
```

This block creates a User object with the username "admin" and password "password123". The checkPassword() method is invoked to validate login.

4.3. CartItem Class

This class is responsible for holding individual items in the shopping cart.
Attributes:

- **product**: A Product object representing the item.
- **quantity**: The number of units of the product.

Code Example:

```
Product phone(2, "Smartphone", 499.99);

CartItem item(phone, 3);

cout << "Product: " << item.product.name << ", Quantity: " << item.quantity << endl;
```

This ensures that each cart item is linked to a specific product with its quantity stored.

4.4. ShoppingCart Class

The ShoppingCart class manages the overall cart functionality. It maintains a list of CartItem objects and provides methods for cart operations.

Attributes:

- vector<CartItem> items: A dynamic list to hold cart items.

Methods:

1. **Add Item**: Adds a product to the cart.

```
void addItem(Product product, int quantity) {
    items.push_back(CartItem(product, quantity));
    cout << "Added " << quantity << " x " << product.name << " to the cart." << endl;
}
```

2. **Remove Item**: Removes a product from the cart by ID.

```
void removeItem(int productId) {
    for (int i = 0; i < items.size(); i++) {
        if (items[i].product.id == productId) {
            items.erase(items.begin() + i);
```

```
            cout << "Item removed from the cart!" << endl;
            return;
        }
    }
    cout << "Item not found in the cart!" << endl;
}
```

3. **View Cart**: Displays all cart items and calculates the total cost.

```
void viewCart() {
    double total = 0.0;
    cout << "Your cart:" << endl;
    for (auto &item : items) {
        double cost = item.product.price * item.quantity;
        cout << item.quantity << " x " << item.product.name << " = $" << cost << endl;
        total += cost;
    }
    cout << "Total: $" << total << endl;
}
```

4. **Checkout**: Computes the final cost and clears the cart.

```
void checkout() {
    double total = 0.0;
    for (auto &item : items) {
        total += item.product.price * item.quantity;
    }
    cout << "Final total: $" << total << endl;
    items.clear();
    cout << "Thank you for shopping!" << endl;
}
```

5. USER INTERACTION FLOW

The overall user flow is controlled by a menu system. A simple loop ensures continuous interaction until the user decides to exit.

KEY STEPS IN THE FLOW

1. Displaying the Menu

The program begins by presenting a menu of options to the user. This menu serves as the control center for navigating the application. Each option corresponds to a specific action within the system.

Menu Example:

```
cout << "1. View Products" << endl;
cout << "2. Add to Cart" << endl;
cout << "3. Remove from Cart" << endl;
cout << "4. View Cart" << endl;
cout << "5. Checkout" << endl;
cout << "6. Exit" << endl;
```

This menu allows users to:

1. Browse available products.
2. Add items to their shopping cart.
3. Remove items from their cart.
4. View the contents of their cart, along with the total cost.
5. Proceed to checkout to finalize the purchase.
6. Exit the program.

2. Handling User Input

The program reads the user's choice using a `do-while` loop. This ensures that the menu is repeatedly displayed until the user decides to exit (choice `6`).

```
int choice;
do {
    cout << "Enter your choice: ";
```

```
    cin >> choice;
    switch (choice) {
        // Case statements handle individual choices
    }
} while (choice != 6);
```

The program uses a switch statement to map each menu option to its corresponding functionality.

3. Viewing Products

When the user selects **"View Products"**, the program displays a catalog of all available products. This catalog is stored in a vector of Product objects.

Code:

```
for (auto &product : catalog) {
    cout << product.id << ". " << product.name << " - $" << product.price << endl;
}
```

Output Example:

1. Laptop - $999.99

2. Smartphone - $499.99

3. Headphones - $49.99

This step helps users identify the products they want to purchase.

4. Adding Items to the Cart

When the user chooses **"Add to Cart"**, the program asks for:

- The product ID (to identify the item).
- The quantity (number of units to purchase).

Code:

```
int id, qty;
```

```
cout << "Enter product ID and quantity: ";

cin >> id >> qty;

if (id > 0 && id <= catalog.size()) {

    cart.addItem(catalog[id - 1], qty); // Adding to the cart

    cout << "Added " << qty << " x " << catalog[id - 1].name << " to the cart." << endl;

} else {

    cout << "Invalid product ID!" << endl;

}
```

This ensures:

1. The product ID is valid.
2. The selected product and quantity are correctly added to the cart.

Example Interaction:

Enter product ID and quantity: 2 3

Added 3 x Smartphone to the cart.

5. Removing Items from the Cart

If the user selects **"Remove from Cart"**, the program prompts for the product ID of the item to be removed. It then searches for this product in the cart and removes it.

Code:

```
int id;

cout << "enter product id to remove: ";

cin >> id;

cart.removeitem(id);
```

Output:

- If the product is found:

Item removed from the cart!

- If not found:

Item not found in the cart!

This provides flexibility to modify the cart contents before finalizing the purchase.

6. Viewing the Cart

The **"View Cart"** option displays all items currently in the cart, including:

- Product name
- Quantity
- Subtotal (price × quantity)

Additionally, it calculates and displays the total cost of all items in the cart.

Code:

cart.viewCart();

Example Output:

Your cart:

3 x Smartphone = $.1499.97

2 x Headphones = $99.98

Total: $1599.95

This feature helps users review their selections before proceeding to checkout.

7. Checkout

The **"Checkout"** option finalizes the purchase by:

1. Calculating the total cost.
2. Displaying the total amount to the user.
3. Clearing the cart for a fresh start.

Code:

cart.checkout();

Example Output:

Final total: $1599.95

Thank you for shopping!

This marks the completion of the shopping session.

8. Exiting the Program

If the user selects **"Exit"**, the program terminates the loop and ends gracefully.

Code:

```
case 6:
   cout << "Exiting..." << endl;
   break;
```

Flow Summary

The interaction flow is built to:

1. Be intuitive and user-friendly.
2. Cover all essential shopping cart functionalities (viewing, adding, removing, checking out).
3. Allow iterative refinement of cart contents before finalizing.

This menu-driven approach makes the program interactive, modular, and easy to use.

6.CONCLUSION

The E-Commerce Shopping Cart project demonstrates how object-oriented programming (OOP) can be used to build a simple online shopping system. The system includes features like user login, product catalog, cart management, and checkout.

By using OOP concepts such as classes and objects, the project is organized and easy to understand. The different classes (Product, User, CartItem, ShoppingCart, and ECommerce) help manage specific tasks, making the code modular and maintainable.

Through this project, I learned how to integrate basic e-commerce functionalities like browsing products, adding them to a cart, and placing an order. It also helped me practice problem-solving and writing clear, efficient code.

Future Enhancements

This project can be expanded by adding features like:

- A search or filter option for products.
- A secure payment system.
- A user-friendly interface.

Overall, the project successfully demonstrates how to build a simple e-commerce system using C++, and provides a solid foundation for further development.