Software Requirements Document (SRD)

1. Introduction

This document outlines the software requirements for a C++ Object-Oriented Programming (OOP) project designed to manage a simple e-commerce system. The system enables users to interact with products, place orders, and store data via file I/O. The system is based on fundamental OOP concepts such as encapsulation, inheritance, polymorphism, abstraction, and exception handling.

2. Functional Requirements

The system should fulfill the following functional requirements:

2.1 User Management

Login: The system allows users (Admin and Customers) to log in using their credentials.

Admin Role: Admin users can add, remove, and list products.

Customer Role: Customers can view products, place orders, and view their order history.

2.2 Product Management

Add Products: Admins can add new products to the inventory.

Remove Products: Admins can remove products from the inventory.

Display Products: Customers and admins can view the product list.

2.3 Order Management

Create Orders: Customers can create orders by selecting products.

Save Orders: Orders should be saved to a file for persistence.

Display Orders: Admins can view all orders placed by customers.

2.4 File I/O

The system should support storing and retrieving order details from a file.

2.5 Error Handling

The system should handle exceptions such as invalid input and file errors gracefully using try-catch blocks.

3. Non-Functional Requirements

The system should meet the following non-functional requirements:

3.1 Performance

The system should respond promptly to user input, with minimal delays when adding products, creating orders, and displaying data.

3.2 Usability

The system should provide a user-friendly command-line interface (CLI) for interaction.

Proper prompts and error messages should guide users in case of incorrect input.

3.3 Reliability

The system should be stable and robust, with minimal crashes or errors.

3.4 Maintainability

The code should be modular, with proper use of functions and classes for easy maintenance and extension.

3.5 Scalability

The system should handle an increasing number of products and orders without significant degradation in performance.

Class Diagram

Below is the class diagram that models the system:

plaintext

Copy code

+---------------------+ +--------------------+ +------------------+

| User |<------>| Admin | | Customer |

|---------------------| |--------------------| |------------------|

| - userID: int | | - adminLevel: int | | - customerID: int|

| - name: string | |--------------------| |------------------|

| - password: string | | + addProduct() | | + placeOrder() |

|---------------------| | + removeProduct() | | + viewProducts() |

| + displayDetails() | | + listProducts() | | + viewOrders() |

+---------------------+ +--------------------+ +------------------+

^ ^ ^

| | |

+----------------------------+------------------------+

|

+--------------------------+

| Product |

|--------------------------|

| - productID: int |

| - name: string |

| - price: float |

|--------------------------|

| + displayProductDetails()|

+--------------------------+

^

|

+--------------------------+

| Order |

|--------------------------|

| - orderID: int |

| - productList: vector<Product>|

|--------------------------|

| + addProductToOrder() |

| + saveToFile() |

| + displayOrderDetails() |

+--------------------------+

Interaction Diagram

The interaction diagram below shows the sequence of actions in the system for creating an order:

plaintext

Copy code

Admin Customer Product Order

| | | |

| 1. Login | | |

| ------------>| | |

| | 2. View Product List |

| |-------------------------------->|

| | | |

| | 3. Select Product to Order |

| |<--------------------------------|

| | | 4. Create Order|

| | |<----------------|

| | 5. Add Product to Order |

| |-------------------------------->|

| | | |

| | 6. Save Order to File |

| |-------------------------------->|

| | | |

| | 7. Display Order Details |

| |<--------------------------------|

Test Cases

Below are the key test cases for your system:

Test Case 1: Test Admin Product Management

Description: Verify that the admin can add and remove products.

Steps:

Admin logs in.

Admin adds a new product.

Admin removes the added product.

Expected Result: Product is added and removed successfully.

Test Case 2: Test Customer Order Creation

Description: Verify that a customer can view products and place an order.

Steps:

Customer logs in.

Customer views the list of products.

Customer selects a product and adds it to the order.

Customer places the order.

Expected Result: The order is created and saved to the file.

Test Case 3: Test File I/O Functionality

Description: Verify that order details are saved and retrieved correctly from a file.

Steps:

Create an order.

Save the order to a file.

Close and reopen the system.

Load and display the saved orders.

Expected Result: The order should be retrieved correctly from the file.

Test Case 4: Test Exception Handling for Invalid Input

Description: Ensure the system gracefully handles invalid inputs.

Steps:

Customer tries to place an order with an invalid product ID.

Admin tries to remove a non-existing product.

Expected Result: System displays an appropriate error message and doesn’t crash.

Results

Test Case 1: Product added and removed successfully.

Test Case 2: Order created and saved successfully.

Test Case 3: File I/O works correctly; orders are saved and retrieved successfully.

Test Case 4: Invalid inputs are handled gracefully with error messages.