C++ Shopping System with DSA - Final Report

1. Project Overview

This C++ project is a command-line based Shopping System designed with a Data Structures and Algorithms (DSA) perspective. The system simulates functionalities of a real-world e-commerce platform, such as browsing products, managing a shopping cart, administering inventory, applying discounts, tracking sales history, and generating UPI QR code payments. The entire project is modularized and emphasizes file-based persistence.

2. Major Modules & Data Structures Used

Admin Module:

• Manages inventory by adding, updating, and removing items.

• Can view purchase history and top-selling products.

Customer Module:

• Allows customer signup/login.

• Browse sections and add items to cart.

• Apply discount codes and pay via UPI.

Inventory Management:

• Uses AVL Tree for each section to maintain balanced and sorted inventory data.

Cart System:

• Priority Queue is used to track low-stock alerts and top-selling products.

Search System:

• KMP algorithm is used for efficient item substring searching.

Discount Codes:

• Implemented using Hash Maps for O(1) retrieval.

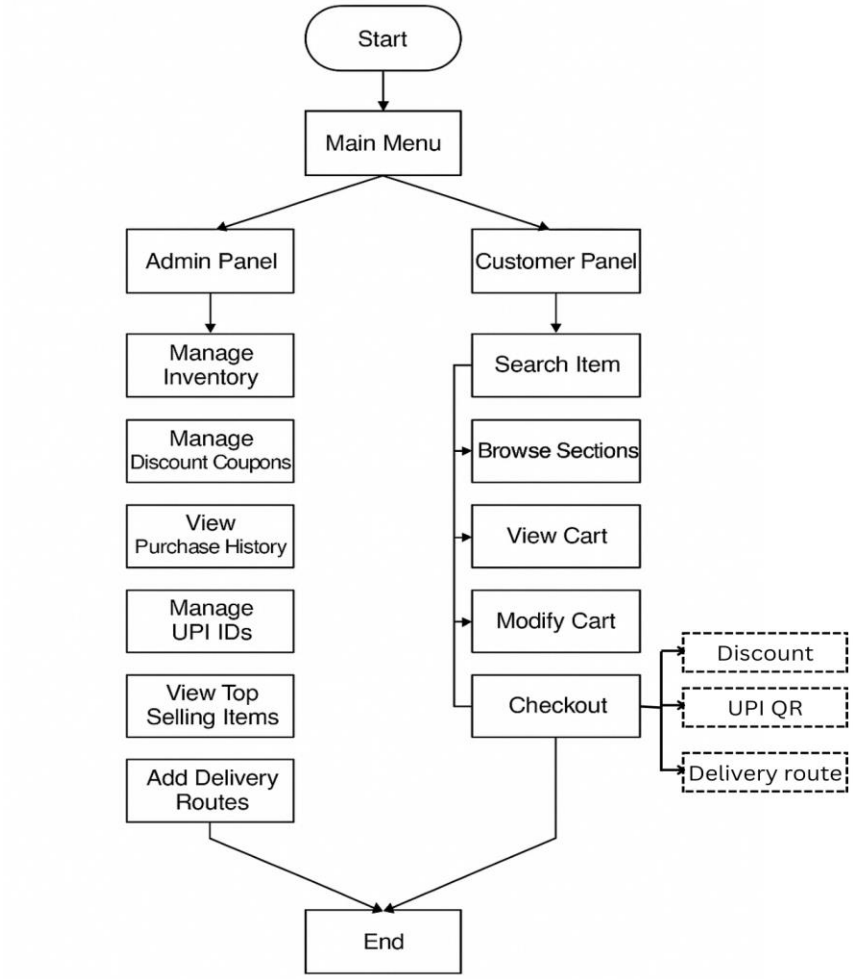
Delivery Routes:

• Implemented using Graph with Dijkstra's algorithm for shortest delivery path simulation.

Billing and QR Code:

• QR codes are generated for UPI payment using the vendor's UPI ID.

Placeholder diagrams:



3. Feature Summary

✓ Admin authentication (username: sandeep, password: 1234)
✓ Modular code with separate files for each component
✓ Inventory stored using AVL Trees and persistent storage in files
✓ Cart operations with editing and checkout
✓ Discounts applied via coupon codes
✓ Purchase history and bill generation with timestamp
✓ Search across inventory using KMP algorithm
✓ Delivery simulation using Graph and Dijkstra's
✓ QR-based UPI payment system with dynamic UPI selection based on amount.

➔ System Architecture & Main Module.

Module/File	Key Responsibilities
`main.cpp`	Entry point, main menu, role selection (Admin/Customer).
`admin.cpp`	Admin dashboard: inventory management, discounts, delivery routes, analytics.
`customer.cpp`	Customer signup/login, search, cart, checkout, delivery.
`inventory.cpp`	Inventory per section using AVL trees, CRUD, file persistence.
`cart.cpp`	Cart operations, sales tracking, top-seller analytics.
`utils.cpp`	Utility: unique IDs, datetime, purchase history.
`discount.cpp`	Coupon code management (add/view).
`delivery_graph.cpp`	Delivery routes as weighted graphs, Dijkstra's algorithm.
`generate_qr.cpp`	UPI payment management, QR code generation.
`search_utils.cpp`	KMP-based item search, global search.

4. Sample Interfaces

Below are typical user interface views for customer interaction:

[CUSTOMER MENU]

0. Search item
1. Browse Sections
2. View Cart
3. Modify Cart
4. Checkout
5. Logout

========= 👨‍💼 Admin PANEL =========

1. Manage Fruits
2. Manage Stationery
3. Manage Snacks
4. Manage Clothes

5. Manage Shop Items
6. Manage More Items
7. Manage Discount Coupons
8. View Purchase History
9. Manage UPI IDs
10. View Top Selling Items
11. Add Delivery Routes
0. Go to Main Menu

==========

5. Data Flow Overview and Core functionality.

Feature	Optimization Approach	Impact
Inventory	AVL Tree	Fast, balanced, update, add, remove
Search	KMP Algorithm	Efficient substring search.
Delivery	Dijkstra's Algorithm	Optimal delivery path. (shortest path)
Analytics	Min-heap	Fast top 5 selection for sales.
File I/O	Sectioned, append-only	Reduces overhead, improves recovery.
Payment	UPI range-matching, QR	Reduces payment errors, seamless UX.

Key Point & How they Work.

Why AVL Trees?

For scalable inventory management, AVL trees guarantee O(logn) operation avoiding performance bottlenecks as data grows.

Why KMP for Search?

KMP efficiently handles substring search, which is ideal for user-driven item lookups with partial or misspelled queries.

Delivery Route Optimization?

Dijkstra's algorithm ensures customers and admins get the shortest, most cost-effective delivery path, a real-world logistics requirement.

Sales Analytics?

Min-heap enables fast extraction of top-selling items, even as the number of products grows.

Payment Robustness?

UPI IDs are matched to payment amounts, reducing transaction errors; QR codes streamline digital payments.

Conclusion

This project demonstrates a robust implementation of a DSA-focused shopping system in C++. It integrates real-world features with academic data structures like AVL Trees, Graphs, Hash Maps, and Priority Queues. The separation of modules, use of file persistence, and extensibility via modern algorithms make it both educational and practical. Future work can include integration with GUI frameworks or a web-based frontend.