

Project Saaman: Detailed DSA Analysis & Flow Report

Technical Breakdown of Data Structures used in the Inventory Management System

This report details the internal architecture of the Saaman system, focusing on **how** and **why** specific data structures were chosen to optimize logistical operations. It includes specific test case flows for Low Stock, Urgent Orders, and Standard Operations.

1. Data Structure Inventory & Rationale

Data Structure	Primary Use Case	Why this structure?	Complexity
Hash Map (Dictionary)	Product Lookup Storage of SKU details (Price, Name, Stock).	Real-world warehouses handle millions of SKUs. Scanning a barcode must be instant. A List would be $O(N)$ (too slow). A Hash Map is $O(1)$.	Access: $O(1)$
Min-Heap (Priority Queue)	Reorder Alerts Tracking items running out of stock.	We only care about the <i>lowest</i> stock items. Sorting a list is $O(N \log N)$. A Heap keeps the minimum element at the top automatically.	Peek: $O(1)$ Insert: $O(\log N)$
Max-Heap (Priority Queue)	Shipping Lane Dispatching orders.	First-In-First-Out (FIFO) is bad if a VIP orders or food expires. Max-Heap allows 'Urgent' items to jump the queue dynamically.	Pop Max: $O(\log N)$
Binary Search Tree (BST)	Stability Reporting Organizing inventory by 'Days Remaining'.	Allows efficient range queries (e.g., 'Show me all items with > 15 days stock'). Keeps data sorted for reports.	Search: $O(\log N)$
Circular Linked List	Audit Schedule Worker shelf assignments.	Audits never end. When the list ends, it must loop back to the start. A circular list models this infinite loop perfectly.	Next: $O(1)$

2. Detailed Scenario Flow Analysis

The following tables trace the exact path of data through the system for specific edge cases requested.

Test Case A: The 'Low Stock' Crisis

Scenario: A customer orders 5 units of 'Gaming Mouse', but the warehouse only has 2 units.

Step	System Action	Data Structure Interaction & Logic
1	Order Entry	Hash Map Lookup: System queries `products['MOUSE-001']`. Result: `{ 'stock': 2 }`.
2	Validation	Logic Check: `Request(5) > Stock(2)`. Result: Insufficient Stock . Order cannot proceed to Shipping Queue.
3	Handling	Shortage Queue (List/DB): Order is flagged as 'PENDING RESTOCK'. It does NOT enter the Shipment Max-Heap yet.
4	Reorder Trigger	Min-Heap Update: Since stock is effectively 0 (all reserved), the 'Days Remaining' score drops to 0. Heapify: This SKU 'bubbles up' to the root of the Min-Heap.
5	Alert	Dashboard Alert: The dashboard polls `heap[0]`. It sees 'Gaming Mouse' and triggers a visual Red Alert for the manager.

Test Case B: The 'Expiring / V.I.P' Order

Scenario: A VIP Customer orders 'Fresh Milk' which expires in 2 days. 50 other orders are already waiting.

Step	System Action	Data Structure Interaction & Logic
1	Scoring	Score Calculation: Base Score: 10 VIP Bonus: +50 Expiry (<7 days): +500 Total Priority Score: 560.
2	Queue Entry	Max-Heap Push: The order is pushed into the `ShippingQueue`. Standard orders have scores around 10-20.
3	Rebalancing	Heapify (Bubble Up): Because 560 is massively larger than 20, this node swaps with its parents until it reaches the Top (Root) of the heap.

4	Dispatch	Pop Max: The Warehouse Packer requests the next task. The Heap pops the Root (Score 560). Outcome: This order is processed immediately , skipping the 50 pending orders.
---	----------	--

Test Case C: The 'Normal' Order

Scenario: A Standard Customer orders a non-perishable item (e.g., 'USB Cable'). Plenty of stock.

Step	System Action	Data Structure Interaction & Logic
1	Scoring	Score Calculation: Base Score: 10 VIP Bonus: 0 Expiry Bonus: 0 Total Priority Score: 10.
2	Queue Entry	Max-Heap Push: Order pushed with Score 10.
3	Rebalancing	Heapify (Sink): Since 10 is lower than the VIP orders (Score 560) and Expiring orders (Score 500), it remains near the bottom or leaves of the tree.
4	Processing	Wait State: The system will not pop this order until all High Priority items are cleared. It ensures VIPs and Expiry risks are handled first.