DSA Project: Online Marketplace

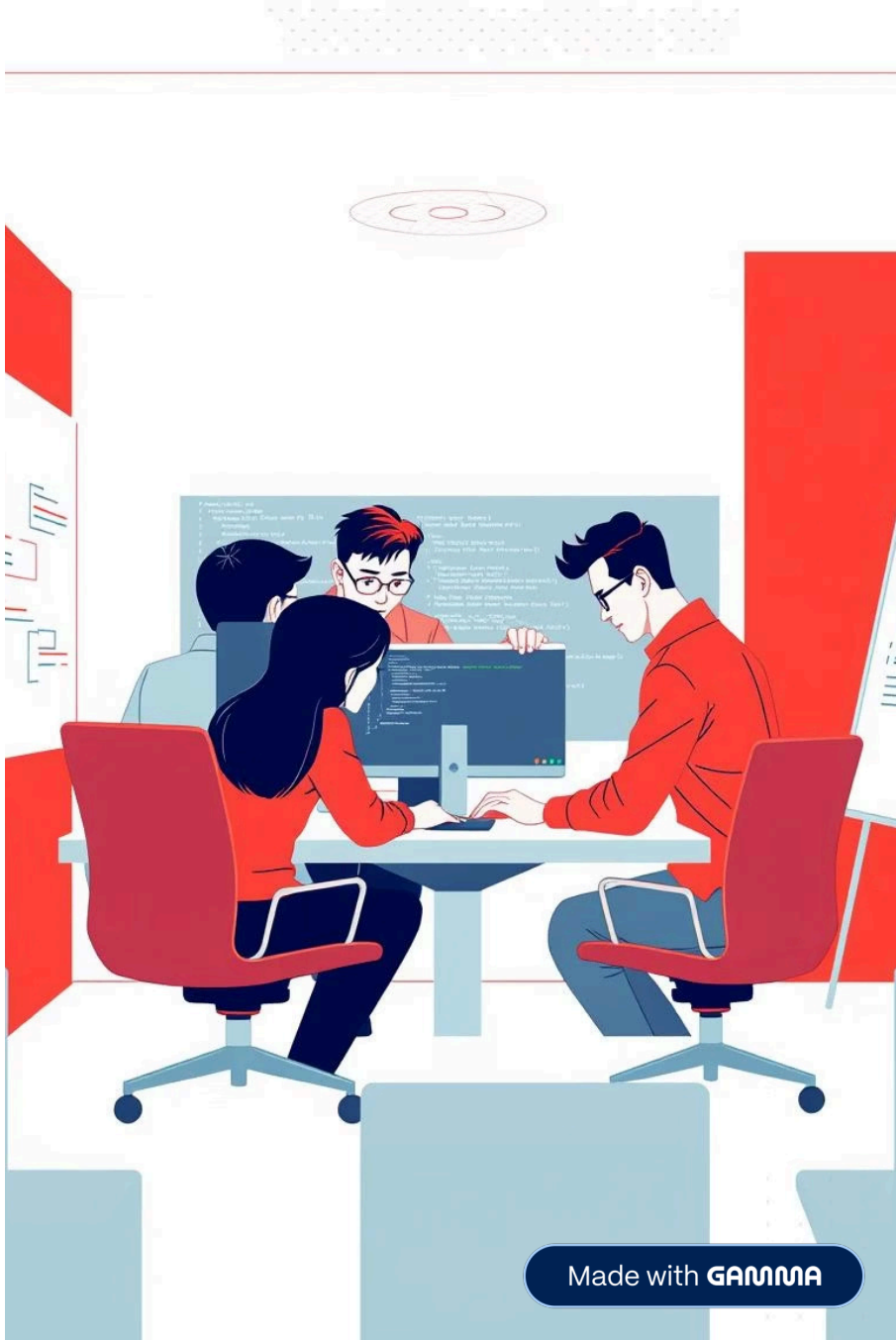
Week 10:
Domain Study, Modules, & Data Structures

Made with GAMMA

Team Members

Meet the dedicated individuals driving this project forward:

- **Bilal Azfar** (Lead)
- Musab Abbasi
- Huzaifa Soomar



Made with GAMMA

Understanding the Online Marketplace Domain

Our project simulates a dynamic online marketplace where sellers list items and buyers place orders. This platform requires robust systems to store product information, manage inventory in real-time, efficiently match buy and sell orders, and intelligently suggest product bundles.

Core Actions

- Add/Search Product
- Place Buy/Sell Order

Advanced Operations

- Match Buy-Sell Orders
- Generate Recommendations

Reporting

- Generate Inventory Reports

Made with GAMMA

Functional Requirements: Building Core Capabilities

To achieve our marketplace vision, we've defined several key functional requirements. These outline the essential features users will interact with and the underlying processes that ensure smooth operation.

- **Product Management:** Add, update, and delete products within the catalog, allowing for multi-level categorization.
- **Order Processing:** Place buy and sell orders, maintaining distinct queues for incoming requests.
- **Order Matching:** Implement priority rules (price/time) to match buy and sell orders effectively.
- **Recommendation Engine:** Provide product suggestions by analyzing co-purchase relationships and identifying top related items.
- **Reporting:** Generate inventory summaries, order-book overviews, and logs of completed matches for system insights.

Made with GAMMA

Non-Functional Requirements: Ensuring Performance

Beyond basic functionality, our marketplace must meet critical non-functional requirements to guarantee a reliable, efficient, and user-friendly experience. These aspects dictate the quality and robustness of our system.

Efficiency

Targeting O(log n) complexity for core matching operations to ensure swift transactions.

Scalability

Designed to handle a high volume of orders and concurrent users without performance degradation.

Reliability

Robust matching logic to prevent incorrect trades and maintain data integrity.

Maintainability

Modular architecture for easy understanding, updates, and future enhancements.

Made with GAMMA

High-Level Architecture: Modular Design

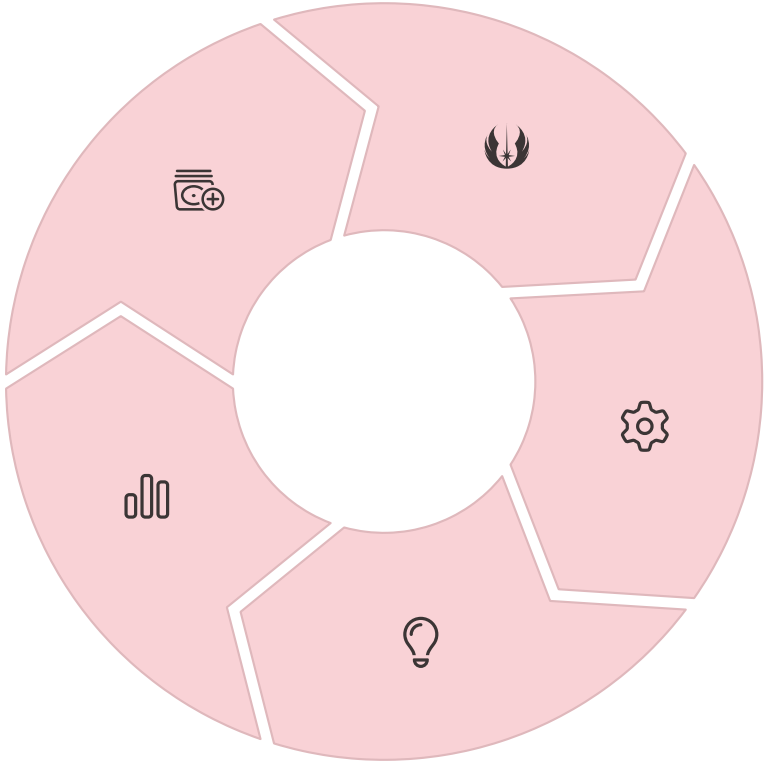
Our system is structured into distinct modules, each responsible for a specific set of functionalities. This modular approach enhances maintainability, scalability, and clarity of design, enabling parallel development and easier debugging.

Catalog Module

Stores product data, categorizes items, and manages inventory.

Reports Module

Provides summaries of inventory, orders, and transactions.



OrderBook Module

Manages buy/sell orders using priority queues (heaps).

Matching Engine

Executes trades based on priority rules, updates inventory.

Recommendation Module

Generates bundle suggestions using graph-based algorithms.

Made with GAMMA

Data Structures: The Foundation of Efficiency

The selection of appropriate data structures is paramount for optimizing performance and ensuring the smooth operation of our online marketplace. Each module leverages specific structures tailored to its functional demands.

Module	Data Structure	Purpose
Catalog	Hash Table	Fast product lookup
Category Hierarchy	Tree	Organize categories
OrderBook	Max-Heap / Min-Heap	Buyer/Seller priority
Order Pipeline	Queue	Process incoming orders
Recommendation	Graph (Adj List)	Co-purchase relationships
Reports	Arrays/Linked Lists	Summaries & logs

Made with GAMMA

Matching Logic: Precision Trading

The heart of our marketplace is the matching engine, designed for rapid and accurate order fulfillment. It prioritizes orders based on price and time, ensuring fair and efficient transactions, with a complexity of O(log n).

Prioritize Orders

Buy orders are stored in a max-heap (highest price first), and sell orders in a min-heap (lowest price first).

Evaluate Conditions

A match occurs if the highest buy price is greater than or equal to the lowest sell price (BuyPrice ≥ SellPrice).

Execute Trade

Upon a successful match, orders are removed from their respective heaps, and the trade is processed.

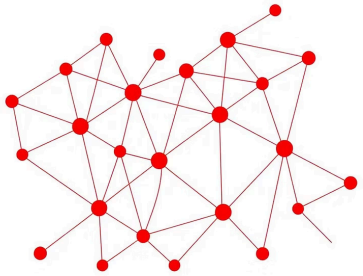
Update System

Inventory quantities are adjusted, and the transaction is logged for audit and reporting purposes.

Made with GAMMA

Recommendation Logic: Smart Suggestions

Our recommendation engine leverages graph theory to provide intelligent product suggestions. By analyzing co-purchase patterns, we can enhance user experience and drive sales through personalized bundles.

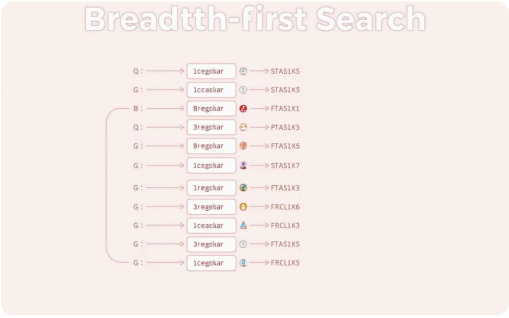


Graph Construction

Nodes represent products, and edges represent the frequency of products bought together. Stronger edges indicate higher correlation.

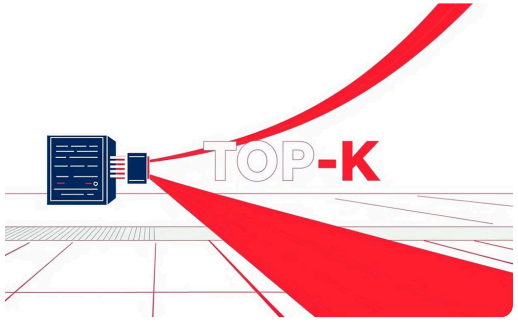
Product Viewing

When a user views a specific product, the system identifies it as a node within the co-purchase graph.



Traversal & Sorting

The graph is traversed (using BFS/DFS) from the viewed product node, and connected neighbors are sorted by edge weight.



Top-K Suggestions

The system then returns the top-k most relevant product suggestions, forming effective bundles for the user.

Made with GAMMA

Project Milestones & Risk Mitigation

Our project roadmap outlines key development phases, from initial design to final integration. Concurrently, we've identified potential risks and formulated strategies to mitigate them, ensuring project stability and success.

Milestones

1

Week 9: Foundation

Domain study, module breakdown, sample dataset created.

2

Week 10: Design

Architecture finalized, data structures chosen, matching logic drafted.

3

Week 11: Implementation I

Catalog + OrderBook implementation, basic heaps, simple matching engine.

4

Week 12: Implementation II

Full matching engine, recommendation graph, demo with test data.

5

Week 13: Finalization

Reports module, final integration, bug fixing, and presentation.

Risks & Mitigation

Risk: Incorrect Heap Ordering

Mitigation: Rigorous unit testing and dry runs to validate heap integrity.

Risk: Large Graph Size

Mitigation: Utilize adjacency list for graph representation and limit recommendation scope.

Risk: Hash Collisions

Mitigation: Employ chaining or well-distributed hash functions to minimize collisions.

Risk: Tight Coupling

Mitigation: Enforce clear, modular interfaces between components to maintain independence.

Made with GAMMA

Technology & Tools

Our project will be developed using C++, leveraging its powerful features to build a robust and efficient marketplace system. While Standard Template Libraries (STLs) are permitted, we emphasize custom data structure implementation to deepen understanding and control.

Language: C++

Chosen for performance, low-level control, and extensive libraries, ideal for data structure implementation.

Key Features

Utilizing classes for modularity, templates for generic programming, and file I/O for data persistence.

Data Structures

Custom implementations are preferred for core data structures, though STLs can be used for utility tasks.

Made with GAMMA