Test Assignment: “Simplified Matching Engine”

Important Notice

Completing this test assignment does not guarantee an offer of employment; it only serves as a demonstration of your skills and knowledge.

Overview

You are asked to implement a basic system for matching trading orders. No need for a running web service or any formal API – a console application or a library is enough. The focus is on the core business logic of order matching and on producing clear, maintainable code.

Matching Logic

1. Placing an Order (PlaceOrder)
    a. When a new limit order arrives, attempt to match it with opposite-side orders:
        i. For a Buy order, look for Sell orders with sellPrice <= buyPrice.
        ii. For a Sell order, look for Buy orders with buyPrice >= sellPrice.
    b. Match from the best price first (highest buy or lowest sell), and break ties via FIFO.
    c. Each successful match creates a Trade record and reduces the RemainingQuantity of both orders.
    d. An order becomes Filled when RemainingQuantity = 0. If partially matched, it is PartiallyFilled.
    e. Fully filled orders are removed from the order book.
2. Canceling an Order (CancelOrder)
    a. If the order is still New or PartiallyFilled, you may cancel it.
    b. The order’s status becomes Cancelled and it is removed from the order book.
3. Order Statuses
    a. New: just created, yet to be filled.
    b. PartiallyFilled: partially matched, some volume remains.
    c. Filled: fully matched, no remaining quantity.
    d. Cancelled: removed from the book before full execution.
4. Concurrency
    a. Please, demonstrate thread safety, show how you handle simultaneous PlaceOrder or CancelOrder calls (e.g., locks or concurrent collections).

What to Implement

1. Project Setup
    a. A console application or class library is sufficient.
    b. A core matching service (e.g., OrderMatchingService) with methods like:
        i. PlaceOrder(...)
        ii. CancelOrder(orderId)
        iii. A way to view active orders or a trade history.
2. Data Storage
    a. In-memory is enough (using dictionaries, lists, or custom data structures).
    b. Buy and Sell lists should be sorted correctly (price priority, FIFO).
3. Tests or Demonstrations
    a. Single order placement (no match).
    b. Two opposing orders (one Buy, one Sell) that match fully or partially.
    c. Checking status transitions (PartiallyFilled, Filled).
    d. Canceling an order before it is fully matched.
    e. More complex scenarios with multiple orders, showing correct matching sequence.

Submission & Evaluation

1. Code: Provide a Git repo link or a compressed archive.
2. Documentation: Include a short README describing:
    a. Project structure (domain classes, services).
    b. How to run your solution (if it’s a console app, detail any input or usage).
    c. How to run and interpret tests or demonstrations.
3. Evaluation Criteria:
    a. Domain Logic & Architecture: clarity in the design, correct matching approach.
    b. Correctness: accurate price/time matching, proper status updates, trade generation.
    c. Code Quality: SOLID principles, appropriate use of C# features (LINQ, async/await if relevant).
    d. Tests: coverage of the main scenarios, correctness of results.

Remember: Completing this assignment does not guarantee a job offer. We appreciate the effort and look forward to seeing your approach!