StakeHub

Problem-Order Matching System and enchancement

In today's fast-evolving tech landscape, the ability to think creatively, leverage all available resources, and deliver solutions with both speed and quality is more valuable than ever.

As a developer, you're not just expected to write clean code — you're expected to solve real-world problems, adapt quickly to changing requirements, and make systems work securely and efficiently across environments. It's not just about what you know, but how you apply it.

This assessment is designed to test your adaptability, practical thinking, and creative problem-solving in a realistic development scenario. You are encouraged to use your judgment, explore tools and frameworks (including AI tools), and demonstrate how you would approach a unique technical challenge in a production-like setup.

Below is a practical query where you're required to enhance and integrate code written by your colleagues, which would not be perfect nor easy to understand — a situation that is extremely common in modern product-based companies. Your ability to work with partially built systems, secure data exchange, and deliver a polished, usable solution is what this task aims to evaluate.

Application Purpose of Order Matching System.

You are given the problem of company for matching of orders -similar to stock exchanges.

You would be having two Table to work on namely-"Pending Order Table" and "Completed Order Table".

In which the given table below is the list of pending orders in the system.

Pending Order Table			
Buyer Qty	Buyer Price	Seller Price	Seller Qty
10	99	100	20
50	98	101	20
70	97	102	130
80	96	103	150
10	96	104	70

In which this data represents is called as 'Pending Order Table'. Wherein When the price of buyer and seller matches , The data is Moved from Pending Order Table to "Completed Order Table''.

Completed Order Table	
Price	Qty
100	20

To Further Understand the Order matching Certain examples are given.

Example 1

Existing Pending Order table

Pending Order Table			
Buyer Qty	Buyer Price	Seller Price	Seller Qty
10	99	100	20
50	98	101	20
70	97	102	130
80	96	103	150
10	96	104	70

Existing Completed Order Table

Completed Order Table	
Price	Qty
100.5	50

A new data is inserted New Input =Buyer, Qty=20, Price=100

Results:

Then the resulted Pending Order table would be

Pending Order Table			
Buyer Qty	Buyer Price	Seller Price	Seller Qty
10	99	101	20
50	98	102	130
70	97	103	150
80	96	104	70
10	96	105	10

Completed Order Table	
Price	Qty
100	20
100.5	50

If clearly noticed the pending order table has seller data of 100 price and 20 qty has been moved to Completed order table.

Example 2

Existing Pending Order table

Pending Order Table			
Buyer Qty	Buyer Price	Seller Price	Seller Qty
10	99	100	20
50	98	101	20

70	97	102	130
80	96	103	150
10	96	104	70

Existing Completed Order Table

Completed Order Table	
Price	Qty
100.5	50

New Input Buyer, Qty=100, Price=100

Results:

Then the resulted Pending Order table would be

Buyer Qty	Buyer Price	Seller Price	Seller Qty
80	100	101	20
10	99	102	130
50	98	103	150
70	97	104	70
80	96	105	10

Completed Order Table	
Price	Qty
100	20
100.5	50

In Completed order table a data of price and qty is shown whenever the price of seller and buyer is matched. Further the qty is shown of the [Minimum of (buyer and seller)]

If clearly noticed the pending order table has seller data of 100 price and 20 qty has been moved to Completed order table. And remaining buyer qty of 80 is shown in pending table.

You will be provided with:
- Frontend UI (React/Next.js): Order form, live tables, and price-time chart
- Backend order matching logic (Node.js)

Task

Your task is to build the backend APIs, implement encrypted communication, secure the transactions, and create a mobile app wrapper for the frontend.

Backend Service (Port 3000)

- Use Node.js
- Expose APIs to:
  - Place buyer/seller orders
  - Return pending & completed orders
- Ensure row-level locking using transactions

Secure API Communication

- Use RSA encryption (PKCS#1 v1.5) with Node.js decryption (primary).
- Implement OpenSSL fallback if Node decryption fails
- Encrypt request data on frontend, decrypt on backend
- Use Node.js version 23.3.0 or above

Mobile App Wrapper

- Create a basic Android or iOS app using WebView to load the frontend
- Add splash screen or demo branding

Database

Setup a Mysql or Postgres database with locking implemented in transactions so that a transactions could not update a same data twice in a single timeframe.

Evaluation Criteria

Your solution will be evaluated based on the following criteria:

- Security: RSA + fallback to OpenSSL implemented properly
- Backend: Node.js APIs, DB integration, locking
- Database: Transaction safety and accurate matching
- Mobile: Functional wrapper, loads frontend UI
- Bonus: Performance optimization using logs, fallback handling

Submission

You are required to host the file at your GitHub repository and host the application and send the URL to the Indeed Chat and to mail id.

*Disclaimer-This is a test is for skill assessment purpose only and your response wouldn't be used by the company in their commercial use.