Exchange Matching Engine — Submission Report

Date: 2025-11-05

Executive Summary

- A real-time cryptocurrency exchange matching engine with price-time priority, REST API, and WebSocket streaming.
- Backend: Node.js/Express + MongoDB. Frontend: React/Vite/Tailwind.
- Key features: Live orderbook/trades, idempotency, rate limiting, metrics, and recovery.
- Security: No secrets in repo; .env used for configuration; idempotency prevents duplicates.

Architecture Overview

- API Layer: Express routes for orders, orderbook, trades, health, metrics.
- Matching Engine: Single-threaded per instrument; broadcasts deltas and trades.
- Services: Order validation, persistence, idempotency, and WebSocket broadcaster.
- Persistence: MongoDB with transactions and TTL indexes for idempotency keys.

Data Model (MongoDB)

- orders: { order_id, client_id, instrument, side, type, price, quantity, filled_quantity, status, timestamps } with matching indexes.
- trades: { trade_id, buy_order_id, sell_order_id, price, quantity, timestamp } indexed by trade_id/timestamp.
- snapshots: periodic orderbook snapshots (optional).
- idempotency_keys: { key, order_id, response, expires_at } with TTL.

APIs

- POST /orders: Submit order (limit/market). Body includes optional idempotency_key.
- POST /orders/:id/cancel: Cancel a resting order.
- GET /orders/:id: Retrieve order details.
- GET /orderbook?levels=N: Orderbook snapshot.
- GET /trades?limit=N: Recent trades.
- GET /healthz and /metrics: Health and Prometheus metrics.

Real-time & Client Integration

- WebSocket ws://localhost:3000/ws with subscribe to [orderbook, trades, orders].
- Server pushes: orderbook_delta, trade, order_update with idempotency_key when present.
- Frontend React UI updates live without refresh; improved formatting for readability.

Idempotency & Security

- Idempotency keys enforce one-result-per-key; duplicate use returns 409 conflict guidance.
- .env used for secrets; .gitignore excludes env files; .env.example uses placeholders.
- No hardcoded credentials — repository sanitized (checked README/DESIGN/code).

Setup & Run (Local)

- Backend: cd backend && npm install && cp .env.example .env (set MONGODB_URI) && npm run dev (http://localhost:3000).
-

Frontend: cd frontend && npm install && npm run dev (http://localhost:5173).
- Optional: backend/tests/load-test.js for load; fixtures/gen_orders.js to generate sample orders.

Testing & Metrics

- Unit tests via Jest (matching engine, validation, idempotency).
- Load test script targets ~2k orders/sec sustained; observe latency percentiles.
- Prometheus metrics include orders_received_total, trades_total, latency histograms.

Troubleshooting

- MongoDB connection failures: ensure mongod running; verify MONGODB_URI in backend/.env.
- WebSocket not updating: backend must be running; check VITE_WS_URL in frontend/.env.
- Port in use: change PORT in backend/.env; restart process.

Conclusion

- Project delivers a functional, observable, and secure real-time exchange core ready for demos and evaluation.
- Clear separation of concerns and recovery strategy ensure resiliency.
- Next steps: multi-instrument support, external queue, snapshot-based recovery.

Generated automatically from project sources.