Document 1: Full System Documentation

Cryptocurrency Matching Engine - Full System Documentation

Overview

This system implements a full cryptocurrency order matching engine designed to simulate the core operations of a real-world electronic exchange. The system is developed entirely in **C++** with full support for REST API, WebSocket feeds, persistence, multiple order types, and price-time priority matching.

The system architecture is modular, highly extensible, and follows principles inspired by **Reg NMS (Regulation National Market System)** used in real-world financial markets.

Technology Stack

Component	Technology
Programming Language	C++17
Build System	CMake
Package Manager	vcpkg
REST API Server	Crow C++ Framework
JSON Parsing	nlohmann/json
WebSocket Server	Crow WebSocket
Persistence	Journal-based file logging
OS	Windows 10/11

Build Instructions

1. Install Dependencies

a. Install vcpkg

git clone https://github.com/microsoft/vcpkg.git

cd vcpkg

./bootstrap-vcpkg.bat

b. Install Required Packages

vcpkg install crow nlohmann-json asio gtest

2. Build Using CMake

cd <project-root>

mkdir build

cd build

cmake .. -DCMAKE_TOOLCHAIN_FILE=C:/path/to/vcpkg/scripts/buildsystems/vcpkg.cmake -DVCPKG_TARGET_TRIPLET=x64-windows

cmake --build . --config Release --parallel

3. Run the Matching Engine

cd src/Release

./engine_app.exe

System Architecture

Modules:

- **MatchingEngine**: Core matching logic, order book management.
- **PersistenceManager**: Order persistence, journaling, recovery.
- **OrderBook**: Bid/Ask book with price-time priority.
- **MarketDataServer**: REST API + WebSocket feeds.
- **TradeExecutionFeed**: Trade dissemination.
- **FeeCalculator**: Maker-taker fee calculation.

REST API Endpoints

Endpoint	Description
POST /orders	Submit new orders
GET /bbo/{symbol}	Best bid/offer

Endpoint	Description
GET /orderbook/{symbol}?depth=N	L2 order book snapshot
GET /health	Health check

WebSocket Feeds

Path	Description
/ws/trades	Live trade feed
/ws/orderbook	L2 order book updates

Data Structures

1. OrderBook

std::map<double, std::deque<Order*>, std::greater<double>> bids_;

std::map<double, std::deque<Order*>, std::less<double>> asks_;

- bids_ is sorted in descending order.
- asks_ is sorted in ascending order.
- Each price level holds a FIFO deque of orders preserving time priority.

2. Order

```
struct Order {
 std::string orderId;
 std::string symbol;
 Side side;
 OrderType type;
 double price;
 double quantity;
 std::chrono::timestamp;
};
```

3. TradeReport

Contains complete trade execution info (symbol, price, quantity, fees, maker/taker info).

Matching Logic Explained

Price-Time Priority

- Incoming order (taker) searches opposite book (bids/asks).
- Price match check based on taker side and order type.
- FIFO matching at each price level.

Matching Flow

for (auto it = sideBook.begin(); it != sideBook.end() && taker.remaining() > 0;)

- Determine matched quantity as min(taker.remaining(), maker.remaining()).
- Execute trade.
- Deduct quantities.
- Publish trade reports.
- Log persistence events (NEW, PARTIAL_FILL, FILLED).

Order Types Supported

- LIMIT
- MARKET
- IOC (Immediate-Or-Cancel)
- FOK (Fill-Or-Kill)

Persistence Layer

PersistenceManager writes every order event into an append-only journal file:

{"event": "NEW", "orderId": "o100", "symbol": "BTC-USDT", "side": "BUY", ...}

Events handled:

- NEW
- RESTED
- PARTIAL_FILL
- FILLED

- CANCELED

Trade Dissemination

- Live trades published via TradeExecutionFeed.
- WebSocket clients receive real-time trade reports.
- Level-2 book updates published on every change.

Reg NMS Inspired Features

Feature	Implementation
Price-Time Priority	Strictly enforced via map+deque
Order Protection	Prevents trade-through
Real-Time Feeds	WebSocket push of trades and book
Audit Trail	Persistent journal logging
