API Documentation

REST API Endpoints

Health Check

Endpoint: `GET /api/v1/health`
Purpose: Verify system status and get performance metrics
Response: System health status, uptime, order statistics, and active symbols
Use Case: Monitoring and load balancer health checks

Submit Order

Endpoint: `POST /api/v1/orders`
Purpose: Create and process a new trading order
Parameters: symbol, order_type, side, quantity, price (for limit orders)
Order Types Supported: market, limit, ioc, fok
Response: Order status, fill information, and execution details
Use Case: Primary order entry point for traders and algorithms

Submit Advanced Order

Endpoint: `POST /api/v1/advanced-orders`
Purpose: Create conditional orders that trigger based on market conditions
Parameters: symbol, order_type, side, quantity, trigger_price, limit_price (for stop_limit)
Advanced Order Types: stop_loss, stop_limit, take_profit
Response: Order confirmation with trigger conditions
Use Case: Risk management and automated trading strategies

Cancel Order

Endpoint: `DELETE /api/v1/orders/{order_id}`
Purpose: Remove an existing order from the order book
Parameters: order_id (path parameter)
Response: Cancellation confirmation and fill summary
Use Case: Order management and position adjustment

Get Order Status

Endpoint: `GET /api/v1/orders/{order_id}`
Purpose: Retrieve current status and execution details of a specific order
Parameters: order_id (path parameter)
Response: Complete order details including fills and timestamps
Use Case: Order monitoring and reconciliation

Get Order Book

Endpoint: `GET /api/v1/orderbook/{symbol}`
Purpose: Retrieve current market depth and price levels
Parameters: symbol (path), depth (query parameter for levels count)
Response: Bid/ask ladder with quantities and best bid/offer
Use Case: Market data display and trading decision support

Get Performance Metrics

Endpoint: `GET /api/v1/performance`
Purpose: Monitor system performance and latency statistics
Response: Detailed latency metrics, throughput, and performance data
Use Case: System monitoring and performance optimization

WebSocket API Feeds

Trade Feed

Connection: `ws://localhost:8080/ws/trades`
Purpose: Real-time stream of all trade executions
Data: Trade details including price, quantity, participants, and fees
Use Case: Live trade monitoring, execution analysis, and position tracking

Order Book Feed

Connection: `ws://localhost:8080/ws/orderbook`
Purpose: Real-time updates of order book changes
Data: Complete bid/ask depth and quantity changes
Use Case: Market making, price discovery, and order book analysis

BBO Feed (Best Bid/Offer)

Connection: `ws://localhost:8080/ws/bbo`
Purpose: Real-time updates of best bid and ask prices
Data: Tightest spreads and market liquidity changes
Use Case: Spread monitoring, market liquidity assessment, and arbitrage

API Design Philosophy

Request-Response Pattern (REST)

- **Order Management:** All order operations use REST for reliability
- **Data Queries:** Snapshot data requests use REST for simplicity
- **Error Handling:** Structured error responses with clear codes

Publish-Subscribe Pattern (WebSocket)

- **Market Data:** Real-time feeds for low-latency data distribution
- **Trade Executions:** Immediate notification of order fills
- **System Events:** Live updates for order book changes

Data Consistency

- **Atomic Operations:** All order processing is atomic
- **Sequential Updates:** WebSocket messages maintain order sequence
- **State Synchronization:** Clients can detect and recover from missed messages

Error Handling

- **Validation Errors:** Invalid parameters return 400 with specific details
- **Business Logic Errors:** Order rejections include reason codes
- **System Errors:** Internal errors return 500 with request identifiers

Client Integration Guide

Order Lifecycle

1. Submit order via REST API
2. Receive immediate execution response
3. Monitor fills via WebSocket trade feed
4. Check final status via order status endpoint
5. Cancel if needed via cancellation endpoint

Market Data Integration

1. Connect to WebSocket feeds
2. Receive initial snapshot
3. Process real-time updates
4. Handle reconnection with sequence numbers
5. Verify data consistency with REST endpoints

Best Practices

- Use WebSocket for real-time data instead of polling
- Implement exponential backoff for reconnections
- Validate all responses and handle errors gracefully
- Monitor performance metrics for system health
- Use order status API for reconciliation

API Testing (Examples)

1. Basic Limit Order Matching

```
echo "=== TEST 1: Basic Limit Order Match ==="
# Bob sells 1 BTC @ $50,000
curl -X POST http://localhost:8000/api/v1/orders \
  -H "Content-Type: application/json" \
  -d '{
    "symbol": "BTC-USDT",
    "order_type": "limit",
    "side": "sell",
    "quantity": "1.0",
    "price": "50000"
  }'

# Alice buys 1 BTC @ $50,000 (should match instantly)
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{
    "symbol": "BTC-USDT",
    "order_type": "limit",
    "side": "buy",
    "quantity": "1.0",
    "price": "50000"
  }'
```

2. Price Priority (REG NMS Compliance)

```
echo "=== TEST 2: Price Priority Enforcement ==="
# Charlie sells @ $50,100 (worse price)
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"limit","side":"sell","quantity":"1.0","price":"50100"}'

# Bob sells @ $50,000 (better price)
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"limit","side":"sell","quantity":"1.0","price":"50000"}'

# Alice buys with market order
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"market","side":"buy","quantity":"1.0"}'
```

3. Time Priority (FIFO)

```
echo "=== TEST 3: Time Priority (First-In-First-Out) ==="
```

```
# David sells @ $50,000 (FIRST)
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"limit","side":"sell","quantity":"1.0","price":"50000"}'

# Eve sells @ $50,000 (SECOND - same price)
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"limit","side":"sell","quantity":"1.0","price":"50000"}'

# Frank buys 1 BTC
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"market","side":"buy","quantity":"1.0"}'
```

4. Market Order Walking Through Book

```
echo "=== TEST 4: Market Order Price Walking ==="
# Setup multiple price levels
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"limit","side":"sell","quantity":"1.0","price":"50000"}'
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"limit","side":"sell","quantity":"2.0","price":"50100"}'

# Large market buy (needs 2.5 BTC)
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"market","side":"buy","quantity":"2.5"}'
```

5. IOC (Immediate or Cancel) Order

```
echo "=== TEST 5: IOC Order - Fill What You Can ==="
# Setup liquidity
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"limit","side":"sell","quantity":"1.5","price":"50000"}'

# IOC buy for more than available
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"ioc","side":"buy","quantity":"3.0","price":"50000"}'
```

6. FOK (Fill or Kill) Order

```
echo "=== TEST 6: FOK Order - All or Nothing ==="
# Only 1 BTC available
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"limit","side":"sell","quantity":"1.0","price":"50000"}'

# FOK wants 2 BTC
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"fok","side":"buy","quantity":"2.0","price":"50000"}'
```

7. Stop-Loss Order (BONUS FEATURE)

```
echo "=== TEST 7: Stop-Loss Order Protection ==="
# Alice places stop-loss sell at $49,000 (if price drops)
curl -X POST http://localhost:8000/api/v1/advanced-orders \
  -H "Content-Type: application/json" \
  -d '{
    "symbol": "BTC-USDT",
    "order_type": "stop_loss",
    "side": "sell",
    "quantity": "1.0",
    "trigger_price": "49000.00"
  }'
```

8. Stop-Limit Order (BONUS FEATURE)

```
echo "=== TEST 8: Stop-Limit Order ==="
# Bob places stop-limit sell: stop at $49,500, limit at $49,000
curl -X POST http://localhost:8000/api/v1/advanced-orders \
  -d '{
    "symbol": "BTC-USDT",
    "order_type": "stop_limit",
    "side": "sell",
    "quantity": "1.0",
    "trigger_price": "49500.00",
    "limit_price": "49000.00"
  }'
```

9. Take-Profit Order (BONUS FEATURE)

```
echo "=== TEST 9: Take-Profit Order ==="
# Charlie places take-profit sell at $55,000
curl -X POST http://localhost:8000/api/v1/advanced-orders \
  -d '{
    "symbol": "BTC-USDT",
    "order_type": "take_profit",
    "side": "sell",
    "quantity": "1.0",
    "trigger_price": "55000.00"
  }'
```

10. Order Cancellation

```
echo "=== TEST 10: Order Cancellation ==="
# Place an order
ORDER_RESPONSE=$(curl -s -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"limit","side":"buy","quantity":"1.0","price":"49000"}')

# Extract order ID and cancel it
ORDER_ID=$(echo $ORDER_RESPONSE | python3 -c "import sys, json;
print(json.load(sys.stdin)['order_id'])")
```

```
curl -X DELETE http://localhost:8000/api/v1/orders/$ORDER_ID
```

11. Real-time WebSocket Trade Feed

```
// Open browser console and run:
const ws = new WebSocket('ws://localhost:8080/ws/trades');
ws.onmessage = (event) => {
  const data = JSON.parse(event.data);
  if (data.type === 'trade') {
    console.log(` TRADE: ${data.quantity} BTC @ $${data.price} | Fees: ${data.maker_fee}/${data.taker_fee}
USDT`);
  }
};
```

12. Real-time Order Book Feed

```
// Watch order book changes
const ws = new WebSocket('ws://localhost:8080/ws/orderbook');
ws.onmessage = (event) => {
  const data = JSON.parse(event.data);
  if (data.type === 'orderbook_update') {
    console.log(' ORDER BOOK UPDATED:', data.bids.length + ' bids, ' + data.asks.length + ' asks');
  }
};
```

13. Order Book Depth Query

```
echo "=== TEST 13: Order Book Snapshot ==="
# Get top 10 levels
curl "http://localhost:8000/api/v1/orderbook/BTC-USDT?depth=10"
```

14. System Health & Performance Metrics

```
echo "=== TEST 14: System Health Check ==="
curl http://localhost:8000/api/v1/health

echo "=== PERFORMANCE METRICS ==="
curl http://localhost:8000/api/v1/performance
```

15. Fee Calculation Verification

```
echo "=== TEST 15: Fee Calculation ==="
# Make a trade and check WebSocket for fees
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"limit","side":"sell","quantity":"1.0","price":"50000"}'
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"market","side":"buy","quantity":"1.0"}'
```

16. Persistence & Recovery Test

```
echo "=== TEST 16: Crash Recovery ==="
# Place some orders, then restart server and verify they're still there
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"limit","side":"buy","quantity":"5.0","price":"48000"}'

# Restart server...
docker-compose restart matching-engine

# Check if order survived restart
curl http://localhost:8000/api/v1/orderbook/BTC-USDT
```

REAL-WORLD TRADING SCENARIOS

Scenario A: Market Maker Bot

```
echo "=== REAL SCENARIO: Market Making ==="
# Market maker places tight spread
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"limit","side":"buy","quantity":"1.0","price":"49999"}'
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"limit","side":"sell","quantity":"1.0","price":"50001"}'
```

Scenario B: Risk Management

```
echo "=== REAL SCENARIO: Risk Management ==="
# Trader with long position sets stop-loss
curl -X POST http://localhost:8000/api/v1/advanced-orders \
  -d '{
    "symbol": "BTC-USDT",
    "order_type": "stop_loss",
    "side": "sell",
    "quantity": "2.0",
    "trigger_price": "48500.00"
  }'
```

Scenario C: Large Institutional Order

```
echo "=== REAL SCENARIO: Large Order Impact ==="
# Fill book with liquidity
for i in {1..10}; do
  curl -s -X POST http://localhost:8000/api/v1/orders \
    -d "{\"symbol\":\"BTC-USDT\",\"order_type\":\"limit\",\"side\":\"sell\",\"quantity\":\"2.0\",\"price\":\"$((50000 +
i*100))\"}" > /dev/null
done

# Institution buys 15 BTC with FOK
curl -X POST http://localhost:8000/api/v1/orders \
  -d '{"symbol":"BTC-USDT","order_type":"fok","side":"buy","quantity":"15.0","price":"51000"}'
```