

**R04**

FYP Final Report

# **Stock Exchange Simulator**

By

AGARWAL Akshat and RAI Sukruti

**RO4**

Advised by

Prof. Wentao XIE

Submitted in partial fulfilment of the requirements for CPEG 4901

in the

Department of Computer Engineering

The Hong Kong University of Science and Technology

2024-2025

Date of submission: April 18, 2025

# Abstract

This project presents the development of a Stock Exchange Simulator designed to provide a robust, cost-effective testing platform for trading systems used by financial institutions. Motivated by the limitations of live exchange test sessions, such as high costs, limited availability, and rigid time windows, this simulator offers a versatile environment for firms to validate their trading logic and compliance with exchange-specific rules.

The development process was structured into multiple phases. Initially, a core matching engine was implemented to support various order types and enforce price-time priority logic. This engine was subsequently integrated with modular services, such as trade generation and validation, within a service-oriented architecture. A graphical user interface (GUI) test tool was also developed to enable users to place, amend, and cancel orders via the FIX protocol in real time.

Additionally, a web-based frontend was designed to display a live order book, providing real-time visualization of buy and sell orders by price level and enhancing usability for developers and traders. Together, these components deliver a scalable and realistic simulation framework for pre-deployment testing across diverse market scenarios.

# Table of Contents

<b>1 Introduction.....</b>	<b>4</b>
1.1 Overview .....	4
1.2 Objective .....	6
<b>2 Methodology .....</b>	<b>9</b>
2.1 Design .....	9
2.2 Implementation .....	16
2.3 Testing .....	27
2.4 Evaluation .....	45
<b>3 Discussion.....</b>	<b>47</b>
<b>4 Conclusion .....</b>	<b>49</b>
<b>5 Project Planning.....</b>	<b>51</b>
5.1 Distribution of Work .....	51
5.2 GANTT Chart .....	53
<b>6 Recommended Hardware &amp; Software .....</b>	<b>54</b>
<b>7 References.....</b>	<b>54</b>
<b>8 Appendix A: Glossary.....</b>	<b>55</b>
<b>9 Appendix B: Literature Survey .....</b>	<b>57</b>
<b>10 Appendix C: Meeting Minutes.....</b>	<b>90</b>
Minutes of the 1 <sup>st</sup> Project Meeting.....	60
10.2 Minutes of the 2 <sup>nd</sup> Project Meeting.....	60
10.3 Minutes of the 3 <sup>rd</sup> Project Meeting.....	61
10.4 Minutes of the 4 <sup>th</sup> Project Meeting .....	62
10.5 Minutes of the 5 <sup>th</sup> Project Meeting .....	62
10.6 Minutes of the 6 <sup>th</sup> Project Meeting .....	63
10.7 Minutes of the 7 <sup>th</sup> Project Meeting .....	64
10.8 Minutes of the 8 <sup>th</sup> Project Meeting .....	64
10.9 Minutes of the 9 <sup>th</sup> Project Meeting .....	65
10.10 Minutes of the 10 <sup>th</sup> Project Meeting .....	66

# 1 Introduction

## 1.1 Overview

Stock markets have historically played a pivotal role in economic growth by enabling companies to raise capital through the sale of shares, which investors purchase in hopes of receiving dividends or capital gains. The evolution of stock markets can be traced back to 13th-century Europe, with formal stock markets being established in the 18th-century to facilitate trade [1].

In today's fast-paced financial markets, electronic trading dominates the landscape, with a vast majority of trades being conducted through highly automated trading systems. These systems, namely Order Management Systems (OMS) and Execution Management Systems (EMS), are critical for buy-side and sell-side institutions, allowing them to execute trades efficiently and in compliance with exchange regulations. However, the testing of these systems remains a significant challenge. Testing on live exchanges is both expensive and restricted by time constraints, as only specific time windows are provided by the exchanges for running their test sessions. For firms operating across multiple exchanges, such as those in the Asia-Pacific region (APAC), this creates additional hurdles, as they must navigate a variety of exchange-specific rules and order types.

This project proposes the development of a Stock Exchange Simulator that replicates the core functionalities of real-world exchanges, including order matching, regulatory rule enforcement, and market behaviour. It aims to provide a flexible, cost-effective environment for firms to test their trading systems without relying on live exchange sessions. The simulator is designed to emulate the distinctive features of APAC markets, such as tick-size regulations, order types (e.g., market and limit orders), and price-time priority-based matching, thus supporting the validation of algorithmic trading systems and cross-market operations.

Beyond its simulation capabilities, the platform serves as a valuable tool for various buy-side organizations, quantitative trading teams, and fintech developers. It is helpful in integration testing of EMS/OMS components, stress testing under high-frequency or large-volume scenarios, and performance benchmarking in controlled environments. Additionally, a GUI-based test tool allows users to interact with the simulator through the FIX protocol by sending, amending, and

cancelling orders in real time.

To complement this, a web-based front end visualizes a live order book, a real-time, electronic list of all buy and sell orders for a specific asset, organized by price level. The order book provides a dynamic snapshot of current market demand and supply, making it an essential tool for understanding trading activity, anticipating price movements, and making informed decisions.

Together, these features make the Stock Exchange Simulator a comprehensive testing platform that is modular, regulation-aware, and designed to reflect the operational complexity of modern financial exchanges.

## 1.2 Objective

The primary objective of this project is to develop a robust and realistic Stock Exchange Simulator that facilitates comprehensive testing of trading systems including Execution Management Systems (EMS), Order Management Systems (OMS), and algorithmic trading platforms used by brokers and financial institutions. The simulator is designed to replicate the complexities of real-world exchanges while offering a flexible, cost-effective, and always-available alternative to traditional live exchange test environments.

- **To Develop a FIX-Based Matching Engine to Support Different Order Types:** At the heart of the simulator lies a custom-built matching engine capable of processing various order types such as market and limit orders. The matching logic is designed to enforce exchange-specific rules like price-time priority and tick-size constraints. By incorporating the Financial Information Exchange (FIX) protocol, the simulator facilitates standardized, real-time communication between clients and the exchange, enabling compatibility with industry-standard EMS and OMS platforms.
- **To Support Different Functions Such as New Order, Amend, and Cancel from Multiple Clients:** The simulator allows multiple concurrent clients emulating multiple traders or systems to interact with the exchange through distinct FIX sessions. These clients can place new orders, amend existing ones, or cancel orders in real time. The system is designed to process these actions efficiently while ensuring that each request complies with the trading rules and protocols of the simulated market, replicating the dynamics of live financial exchanges.
- **To Integrate Exchange-Specific Market Rules:** In order to provide a credible simulation environment, the platform incorporates exchange-specific regulations and behaviour models. The project has focused on simulating the rules and mechanisms of Hong Kong and Japan stock exchanges, including tick-size rules and execution algorithms. The architecture is modular to allow future extension to other global markets, thereby supporting multi-market algorithm testing.

- **To Maintain Order Book Persistence and Updation:** A central feature of the simulator is its persistent and dynamic order book, which maintains a structured record of active buy and sell orders. The system is capable of real-time updates and reflects market changes as orders are placed, matched, or cancelled. This enables a realistic and accurate simulation of trading environments, helping users visualize market depth and order flow dynamics.
- **To Develop a GUI-Based Test Tool for Sending New Orders, Amendments, and Cancellations via FIX:** To enhance usability and reduce the barrier to entry for non-technical users, a graphical user interface (GUI) test tool has been developed. This tool allows users to send new orders, execute amendments, and perform cancellations through the FIX protocol without needing to write manual FIX messages.
- **To Develop a User Interface:** Complementing the backend simulator is a secure, web-based frontend that visualizes the live order book. The order book interface offers real-time insights into market depth by showing active bids and asks across price levels. This snapshot helps simulate the conditions traders would experience in real markets. The frontend is particularly useful for testing algorithmic strategies that rely on market microstructure, and it includes access controls to ensure secure usage by authorized testers.

By combining FIX-based order routing, exchange-specific rule handling, and a modular architecture, the simulator provides a complete environment for testing and refining trading systems. One of its most impactful features is the real-time order book display, which offers a live, visual representation of market depth by bringing together all components of the system into a dynamic interface.

By implementing these objectives, the Stock Exchange Simulator will provide a robust, cost-effective, and flexible environment for firms to test and optimize their trading systems without relying on expensive live exchange test sessions.

Throughout the development of the Stock Exchange Simulator, several challenges emerged that tested both our technical skills and system design understanding. One of the most significant



hurdles was integrating the FIX protocol, which was entirely new to us. Understanding its complex message formats, session control mechanisms, and real-time communication flow required extensive research and experimentation. Additionally, implementing accurate order matching logic in accordance with exchange-specific rules proved to be more intricate than anticipated. Designing a queueing system that respected price-time priority, especially for overlapping order prices and quantities, demanded precision and careful handling of edge cases. Another major challenge was ensuring that the order book remained consistently updated in real time. Because the simulator needed to reflect every market event instantly, we had to build a robust update mechanism that maintained data consistency while handling multiple concurrent requests from different clients. Balancing real-time performance with reliability and correctness was particularly demanding. Despite these obstacles, continuous iteration, modular design, and targeted testing helped us refine the system and move closer to our objectives.

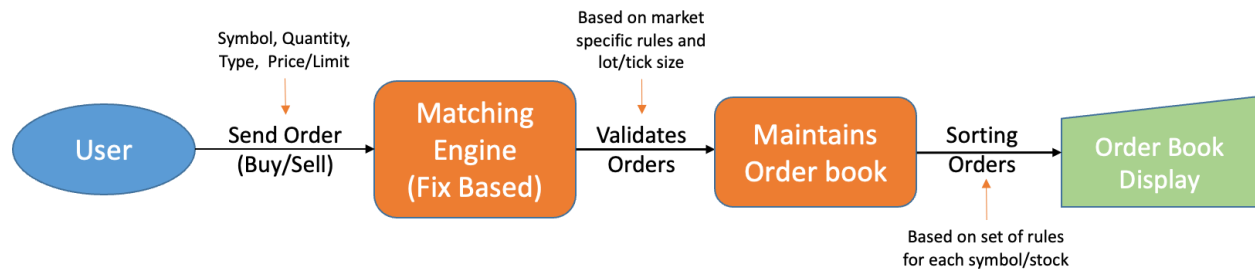
## 2 Methodology

### 2.1 Design

The Stock Exchange Simulator is built following a **Service-Oriented Architecture (SOA)**, enabling modularity, scalability, and high cohesion between components. Each major function of the exchange is encapsulated within a dedicated service, allowing the system to be extended or modified independently without affecting unrelated components. This design choice mirrors the architecture adopted in institutional trading systems and ensures realistic simulation behavior.

The system architecture comprises six major backend services and is structured to receive trading instructions via the FIX protocol, validate them, process them through a matching engine, and reflect the changes in a live order book display.

#### 2.1.1 Core Services and Workflow



**Fig 2.1.1: Pipeline Design**

- **FIXOrderReceiverService**

This service is the system's gateway, implemented using **QuickFIX/J**, which handles the reception of FIX messages from client systems (e.g., EMS/OMS or the Swing GUI). It establishes FIX sessions over TCP/IP, validates session state, and extracts trading instructions such as NewOrderSingle (D), Cancel (F), and Replace (G). These are then forwarded to the internal exchange simulator pipeline.

- **ExchangeRuleValidatorService**

This service enforces market-specific constraints, such as tick size restrictions, order

types (e.g., limit, market), and time-in-force conditions. Acting as the first line of validation, it ensures that orders comply with the exchange's rulebook before being accepted into the market.

- **MatchingService**

The heart of the simulator, the `MatchingService` implements a **price-time priority algorithm** to match buy and sell orders efficiently. Orders are stored in a per-symbol `Market` object, each maintaining separate bid and ask priority queues.

- **TradeGeneratorService**

Upon successful matches, this service creates `Execution` records capturing the matched price, quantity, and timestamps. These trades are persisted and made available for display on both frontend and testing interfaces.

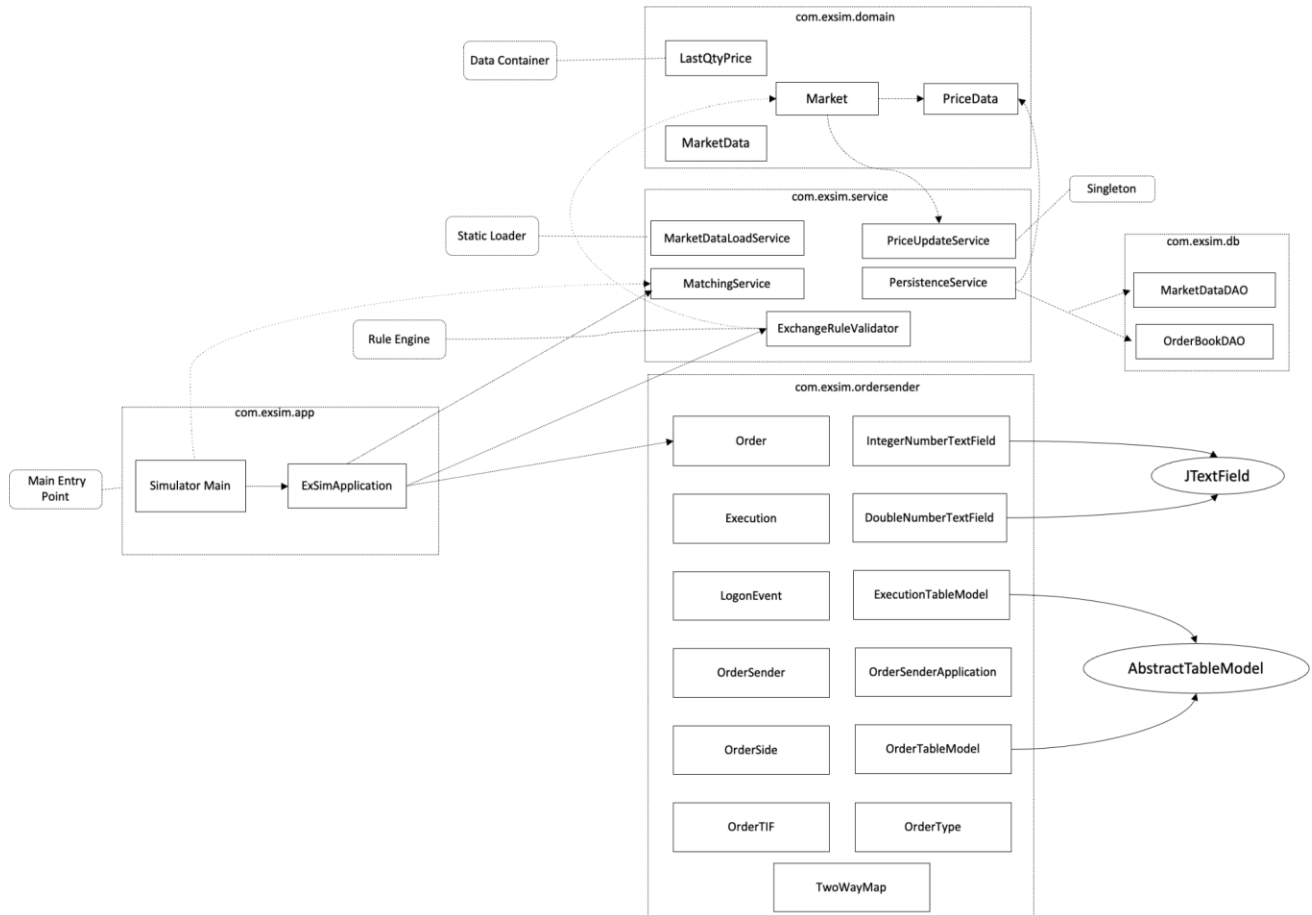
- **OrderbookPersistenceService**

To enable both state durability and real-time display, the simulator stores all order and execution data in a **PostgreSQL** database. The service uses DAO abstractions—`OrderBookDAO` and `MarketDataDAO`—to perform CRUD operations on order book snapshots, execution logs, and price statistics. Additionally, a JSON snapshot (`orderbook.json`) is generated to power the live frontend display.

- **OrderSenderService**

Though currently stubbed, this service is designed to simulate outbound order routing to external exchanges or dark pools in future iterations. This will support testing multi-market routing and hybrid venue simulations.

## 2.1.2 Domain Model and Class Design



**Fig 2.1.2: Class Diagram**

The Stock Exchange Simulator's codebase is structured around a clearly defined object-oriented model, with each component logically grouped into packages. These packages, as illustrated in the UML class diagram, follow domain-driven design principles and promote modularity, encapsulation, and maintainability.

### Key Domain Classes:

- **Order**

The Order object represents a client's trading instruction and is the fundamental input

processed by the exchange simulator. It contains attributes such as the trading symbol, price, quantity, order type (e.g., market or limit), side (buy/sell), time-in-force (TIF), and a client identifier. Orders are used across the system to drive matching, execution, and reporting logic. Once submitted, orders are validated, queued into the symbol-specific Market, and either matched or held in the book based on market conditions.

- **Market**

The Market class manages the live order book for a specific trading symbol. Internally, it maintains two separate priority queues—one for buy (bid) orders and another for sell (ask) orders. Buy orders are sorted from highest to lowest price, while sell orders are sorted from lowest to highest price, both using timestamp as a secondary key for price-time priority. The Market class encapsulates matching logic, order queueing, and the generation of execution events.

- **PriceData**

This object tracks real-time price information at the micro level for a particular symbol. It stores the current best bid and ask prices, the last traded price, and can be extended to include spread and mid-price. PriceData is continuously updated by the matching engine as orders are matched or canceled, and it forms the basis of the live market depth display on the frontend.

- **MarketData**

MarketData provides a macro view of trading activity. It aggregates metrics such as highest and lowest prices over a session, and average execution price. This data is persisted and exposed to the frontend to enable analytical insights and market summaries. It complements PriceData by offering historical and statistical perspectives.

## **DAO Layer (Database Access):**

To maintain a clear separation between application logic and data persistence, the simulator employs a DAO (Data Access Object) layer:

- **OrderBookDAO:** Manages CRUD operations related to order records in the PostgreSQL database. It handles inserts, updates, deletions, and filtered queries to support order book reconstruction and historical data access.
- **MarketDataDAO:** Responsible for persisting aggregated market statistics, including volume and price changes, which are used for frontend rendering and analytics.

These DAOs are leveraged by the PersistenceService and OrderbookPersistenceService to ensure that all in-memory market states are accurately reflected in the persistent store.

### Utility Services:

- **DBConfigService:** A singleton utility responsible for loading and maintaining PostgreSQL database configuration settings. This service ensures consistent access to connection details and manages retry logic in case of connection failures.

### Persistence Layer:

- The simulator uses **PostgreSQL** for all data persistence requirements. Initially, an H2 in-memory database was used during prototyping for its lightweight and transient nature, but it has since been replaced with PostgreSQL to support realism and scalability.
- Market state is also serialized into a JSON file (orderbook.json) after each change, allowing frontend applications to visualize the latest order book in real-time.

## 2.1.3 User Interface Design

There are two primary interfaces that interact with the backend:

- **Swing-based GUI Test Tool:** Built in Java, it allows manual testing of FIX order flows. Users can create, cancel, and amend orders via a simple form, and receive FIX-level execution reports.

- **React + Node.js Web Frontend:** This web-based UI displays real-time order book data, executions, and market statistics by fetching data from the backend via REST APIs.

## 2.1.4 Design Patterns and Object-Oriented Design Principles

The simulator applies a variety of design patterns and object-oriented programming (OOP) principles to promote robustness, scalability, and extensibility:

### Design Patterns:

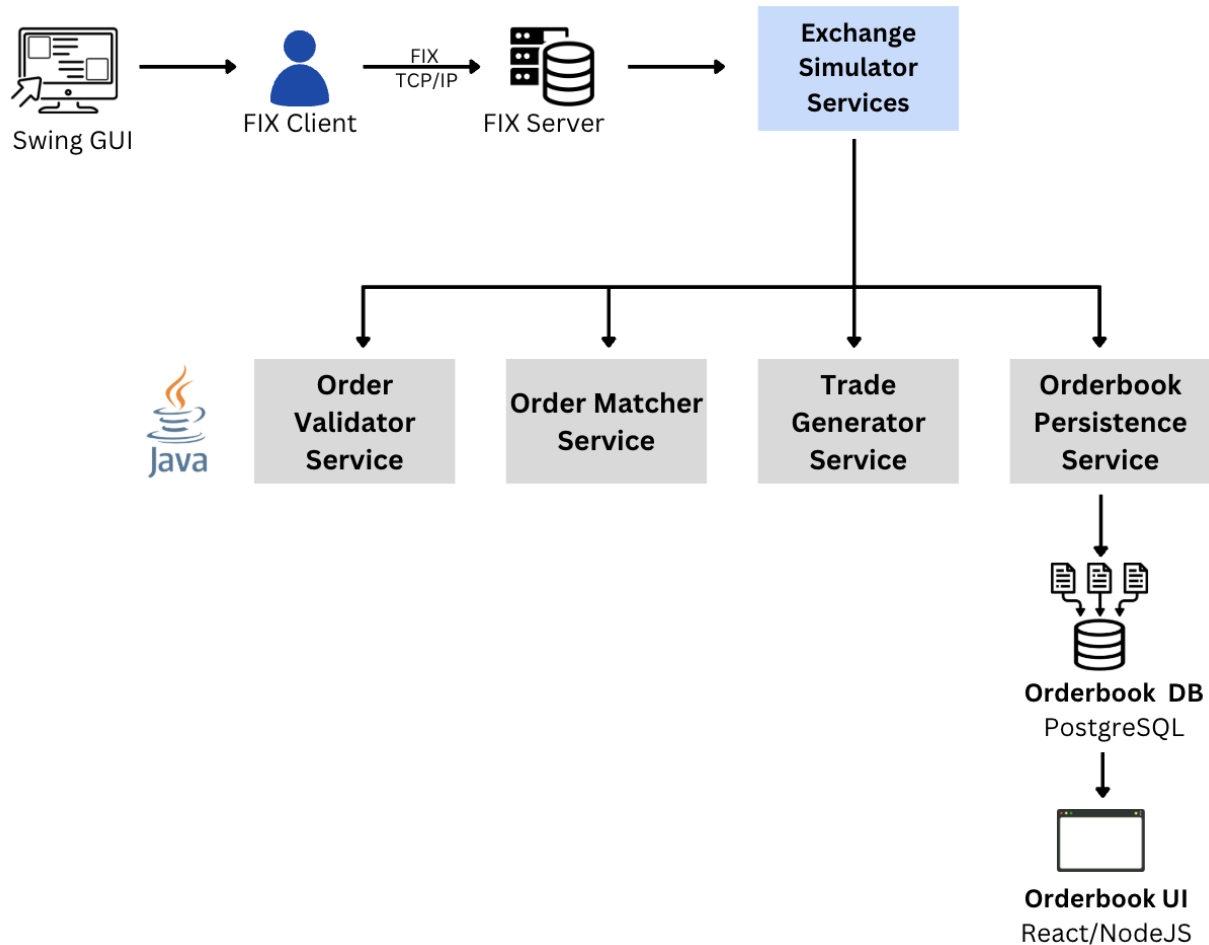
- **Singleton Pattern:** Implemented in DBConfigService to ensure a consistent database configuration is used system-wide.
- **Factory Pattern:** Used in OrderFactory to dynamically create orders based on FIX message fields. This allows support for multiple order types and ensures modular object creation logic.
- **Observer Pattern:** PriceUpdateService listens to trade events and pushes updates to the frontend (via orderbook.json or WebSocket). This ensures real-time UI updates without coupling UI logic into the core backend.
- **Strategy Pattern:** The MatchingService leverages this pattern to switch between different matching algorithms allowing future integration of new market models with minimal change.
- **Concurrent Data Structures:** Thread-safe structures like ConcurrentHashMap and synchronized PriorityQueues are used to manage concurrent order access and updates within the matching engine.

### Object-Oriented Design Principles:

- **Encapsulation:** Each core functionality such as validation, matching, trade logging, and persistence is encapsulated in a standalone service. This separation allows easier testing, debugging, and future enhancements.
- **Polymorphism:** The system supports multiple implementations of behavior, such as matching logic and FIX message handling. For example, different types of order matching strategies implement a shared interface and can be swapped at runtime.
- **Concurrency Handling:** Given the multi-client environment, thread safety is essential. The use of thread-safe collections and synchronized execution ensures consistent state updates, especially in high-frequency simulation scenarios.
- **Scalability:** The microservices-inspired modular architecture allows each component to scale independently. Services like `FIXOrderReceiverService` or `MatchingService` can be replicated or optimized individually as demand increases.



## 2.2 Implementation



**Fig 2.2a: Workflow Diagram**

The implementation of the Exchange Simulator is composed of multiple services and modules that together simulate a real-world trading environment. Each component is modularly designed and collaboratively contributes to processing and managing client orders using the FIX protocol, executing trades, updating the order book, and exposing results for external interfaces.

### Matching Engine Development:

At the core of the Exchange Simulator is the **MatchingService**, which maintains a mapping of stock symbols to their corresponding **Market** objects. Each **Market** maintains separate priority queues for buy and sell orders. Buy orders are arranged from highest to lowest price, and sell

orders from lowest to highest price, enabling efficient execution using a price-time priority model.

Key operations implemented in the matching engine include:

- **Order Insertion:** Orders received via FIX or other interfaces are inserted into the respective Market queue using the insert() method.
- **Matching Logic:** The match() method continuously compares the top buy and sell orders to identify executable pairs based on price and quantity.
- **Order Lookup and Deletion:** Orders can be queried using find() and removed using erase() for modifications or cancellations.
- **Thread-Based Execution:** The engine runs in a continuous thread initiated from SimulatorMain.java, enabling real-time matching and ensuring that market conditions are continuously monitored and updated.

### **Thread-Based Matching:**

The matching engine runs continuously in a background thread. This architecture ensures that order matching and execution happen in real-time, closely mimicking real exchange behavior. All updates to the order book are thread-safe and synchronized to maintain system consistency. A diagram depicting this continuous process using a dedicated matching thread and concurrent queues should be inserted here

## **FIX Interface Integration:**

The simulator uses **QuickFIX/J**, a widely adopted open-source FIX engine, to establish FIX sessions for external EMS/OMS clients. The core of this integration is managed by the `ExSimApplication` class, which implements the `quickfix.Application` interface. QuickFIX is used for implementing the FIX server logic, enabling stable and protocol-compliant communication.

Features of the FIX integration include:

- **Session Handling:** Supports multiple concurrent client sessions, with real-time heartbeat validation and message sequencing.
- **Message Processing:** Handles standard FIX 4.2 message types:
  - "D" → `NewOrderSingle` (New Order)
  - "G" → `OrderCancelReplace` (Amend Order)
  - "F" → `OrderCancelRequest` (Cancel Order)
- **Order Routing:** Based on message content, the order is validated, inserted into the order book, matched, and updated with execution status.
- **Execution Reporting:** Responses such as order acknowledgment, partial/full fills, replacements, and rejections are sent back via FIX using `ExecutionReport` or `OrderCancelReject`.

The application logs all executions and displays the current order book state after every transaction using `orderMatcher.display()` and `orderMatcher.displayOrderBook()`.

# RO4 FYP – Stock Exchange Simulator

```
INFO: Socket option: SocketSynchronousWrite=false
4/H 18, 2025 2:29:56 F4F quickfix.aina.NetworkInitiatorOption logOption
INFO: Socket option: SocketSynchronousWriteTimeout=5000
4/H 18, 2025 2:29:56 F4F quickfix.aina.NetworkInitiatorOption logOption
INFO: FIX.4.2.CLI-INIT-EX3IM (1/27.0.0.1:5249)
4/H 18, 2025 2:29:56 F4F quickfix.aina.SessionConnector startSessionTimer
INFO: SessionTimer start
4/H 18, 2025 2:29:56 F4F quickfix.aina.SingleThreadedEventHandlingStrategyY1 run
INFO: Started QF Message Processor
4/H 18, 2025 2:29:56 F4F quickfix.aina.InitiatorHandlerSessionCreated
INFO: MIN session created for FIX.4.2.CLI-INIT-EX3IM: locals:127.0.0.1:5249, class org.apache.aina.transport.socket.alo.NioSocketSession, remote:127.0.0.1:5249
-20250418-06:29:57, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=67 35=A 34=1 49=EX3IM 52=20250418-06:29:57.91.56=EX3IM 98=0 108=30 10=133 )
-20250418-06:29:57, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=67 35=A 34=1 49=EX3IM 52=20250418-06:29:57.91.56=EX3IM 98=0 108=30 10=133 )
-20250418-06:29:57, FIX.4.2.CLI-INIT-EX3IM, events, (Initiated login request)
-20250418-06:29:57, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=140 35=0 34=2 49=EX3IM 52=20250418-06:30:16.470 56=EX3IM 60=11744957816440 14=0 17=0 20=0 31=0 32=0 37=1744957816440 38=100 39=0 40=2 54=1 55=001.HK 59=0 60=20250418-06:30:16.439 10=075 )
-20250418-06:30:16, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=140 35=0 34=2 49=EX3IM 52=20250418-06:30:16.470 56=EX3IM 60=11744957816440 14=0 17=0 20=0 31=0 32=0 37=1744957816440 38=100 39=0 40=2 54=1 55=001.HK 150=0 151=100 10=011 )
-20250418-06:30:24, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=142 35=0 34=3 49=EX3IM 52=20250418-06:30:24.118 56=EX3IM 60=11744957824120 14=0 17=0 20=0 31=0 32=0 37=1744957824120 38=100 39=0 40=2 54=1 55=001.HK 59=0 60=20250418-06:30:24.118 10=157 )
-20250418-06:30:24, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=142 35=0 34=3 49=EX3IM 52=20250418-06:30:24.118 56=EX3IM 60=11744957824120 14=0 17=0 20=0 31=0 32=0 37=1744957824120 38=100 39=0 40=2 54=1 55=001.HK 150=0 151=100 10=025 )
-20250418-06:30:32, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=140 35=0 34=4 49=EX3IM 52=20250418-06:30:32.494 56=EX3IM 60=11744957832497 14=0 17=0 20=0 31=0 32=0 37=1744957832497 38=100 39=0 40=2 54=1 55=001.HK 59=0 60=20250418-06:30:32.494 10=086 )
-20250418-06:30:32, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=140 35=0 34=4 49=EX3IM 52=20250418-06:30:32.494 56=EX3IM 60=11744957832497 14=0 17=0 20=0 31=0 32=0 37=1744957832497 38=100 39=0 40=2 54=1 55=001.HK 150=0 151=100 10=039 )
-20250418-06:30:53, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=139 35=0 34=5 49=EX3IM 52=20250418-06:30:53.122 56=EX3IM 60=11744957853127 14=0 17=0 20=0 31=0 32=0 37=1744957853127 38=50 39=0 40=2 54=2 55=001.HK 59=0 60=20250418-06:30:53.122 10=029 )
-20250418-06:30:53, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=162 35=8 34=5 49=EX3IM 52=20250418-06:30:53.138 56=EX3IM 60=11744957853127 14=0 17=3 20=0 31=0 32=0 37=1744957853127 38=50 39=0 40=2 54=2 55=001.HK 150=0 151=50 10=192 )
-20250418-06:30:53, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=165 35=8 34=6 49=EX3IM 52=20250418-06:30:53.138 56=EX3IM 60=11744957853127 14=50 17=4 20=0 31=21 32=50 37=1744957853127 38=50 39=2 40=2 54=2 55=001.HK 150=2 151=50 10=104 )
-20250418-06:30:53, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=167 35=8 34=7 49=EX3IM 52=20250418-06:30:53.138 56=EX3IM 60=11744957853127 14=50 17=5 20=0 31=21 32=50 37=1744957853127 38=100 39=1 40=2 54=1 55=001.HK 150=1 151=50 10=104 )
-20250418-06:31:22, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=139 35=0 34=6 49=EX3IM 52=20250418-06:31:22.097 56=EX3IM 60=11744957882102 14=0 17=0 20=0 31=0 32=0 37=1744957882102 38=50 39=0 40=2 54=2 55=001.HK 150=0 151=50 10=089 )
-20250418-06:31:22, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=162 35=8 34=8 49=EX3IM 52=20250418-06:31:22.097 56=EX3IM 60=11744957882102 14=0 17=3 20=0 31=0 32=0 37=1744957882102 38=50 39=0 40=2 54=2 55=001.HK 150=0 151=50 10=089 )
-20250418-06:31:40, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=139 35=0 34=7 49=EX3IM 52=20250418-06:31:40.862 56=EX3IM 60=11744957900853 14=0 17=0 20=0 31=0 32=0 37=1744957900853 38=50 39=0 40=2 54=2 55=001.HK 150=0 151=50 10=099 )
-20250418-06:31:40, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=162 35=8 34=9 49=EX3IM 52=20250418-06:31:40.862 56=EX3IM 60=11744957900853 14=0 17=7 20=0 31=0 32=0 37=1744957900853 38=50 39=0 40=2 54=2 55=001.HK 150=0 151=50 10=099 )
-20250418-06:32:10, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=55 35=0 34=8 49=EX3IM 52=20250418-06:32:10.900 56=EX3IM 10=076 )
-20250418-06:32:10, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=56 35=0 34=10 49=EX3IM 52=20250418-06:32:10.900 56=EX3IM 10=118 )
-20250418-06:32:41, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=56 35=0 34=11 49=EX3IM 52=20250418-06:32:41.500 56=EX3IM 10=119 )
-20250418-06:32:41, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=55 35=0 34=9 49=EX3IM 52=20250418-06:32:41.500 56=EX3IM 10=077 )
-20250418-06:33:11, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=56 35=0 34=12 49=EX3IM 52=20250418-06:33:11.491 56=EX3IM 10=127 )
-20250418-06:33:11, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=56 35=0 34=10 49=EX3IM 52=20250418-06:33:11.491 56=EX3IM 10=125 )
-20250418-06:33:41, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=56 35=0 34=13 49=EX3IM 52=20250418-06:33:41.497 56=EX3IM 10=137 )
-20250418-06:33:41, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=56 35=0 34=11 49=EX3IM 52=20250418-06:33:41.497 56=EX3IM 10=135 )
-20250418-06:34:11, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=56 35=0 34=14 49=EX3IM 52=20250418-06:34:11.503 56=EX3IM 10=144 )
-20250418-06:34:11, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=56 35=0 34=12 49=EX3IM 52=20250418-06:34:11.503 56=EX3IM 10=122 )
-20250418-06:34:41, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=56 35=0 34=15 49=EX3IM 52=20250418-06:34:41.896 56=EX3IM 10=146 )
-20250418-06:34:41, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=56 35=0 34=13 49=EX3IM 52=20250418-06:34:41.896 56=EX3IM 10=134 )
-20250418-06:35:11, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=56 35=0 34=16 49=EX3IM 52=20250418-06:35:11.901 56=EX3IM 10=130 )
-20250418-06:35:11, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=56 35=0 34=14 49=EX3IM 52=20250418-06:35:11.901 56=EX3IM 10=132 )
-20250418-06:35:41, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=56 35=0 34=17 49=EX3IM 52=20250418-06:35:41.896 56=EX3IM 10=146 )
-20250418-06:35:41, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=56 35=0 34=15 49=EX3IM 52=20250418-06:35:41.896 56=EX3IM 10=144 )
-20250418-06:36:11, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=56 35=0 34=18 49=EX3IM 52=20250418-06:36:11.901 56=EX3IM 10=132 )
-20250418-06:36:11, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=56 35=0 34=16 49=EX3IM 52=20250418-06:36:11.901 56=EX3IM 10=147 )
-20250418-06:36:42, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=56 35=0 34=19 49=EX3IM 52=20250418-06:36:42.497 56=EX3IM 10=145 )
-20250418-06:37:12, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=56 35=0 34=18 49=EX3IM 52=20250418-06:37:12.289 56=EX3IM 10=141 )
-20250418-06:37:12, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=56 35=0 34=20 49=EX3IM 52=20250418-06:37:12.289 56=EX3IM 10=141 )
-20250418-06:37:12, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=56 35=0 34=20 49=EX3IM 52=20250418-06:37:12.289 56=EX3IM 10=141 )
-20250418-06:37:42, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=56 35=0 34=21 49=EX3IM 52=20250418-06:37:42.497 56=EX3IM 10=141 )
-20250418-06:37:42, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=56 35=0 34=19 49=EX3IM 52=20250418-06:37:42.497 56=EX3IM 10=148 )
-20250418-06:38:12, FIX.4.2.CLI-INIT-EX3IM, outgoing, (8=FIX.4.2.9=56 35=0 34=22 49=EX3IM 52=20250418-06:38:12.505 56=EX3IM 10=130 )
-20250418-06:38:12, FIX.4.2.CLI-INIT-EX3IM, incoming, (8=FIX.4.2.9=56 35=0 34=22 49=EX3IM 52=20250418-06:38:12.505 56=EX3IM 10=130 )
```

Fig 2.2b: Heartbeat FIX Engine

```
59899929", "entryTime": "174495999015", "closed": false, "filled": false, ("price": 30.0, "openQuantity": 100, "orderId": "1744959978508", "entryTime": "1744959978499", "closed": false, "filled": false), "askOrders": [{"price": 32.0, "openQuantity": 150, "orderId": "174495996287", "entryTime": "174495996287", "closed": false, "filled": false}]]
-20250418-07:06:31, FIX.4.2.EX3IM->CLIENT1, incoming, (8=FIX.4.2.9=55 35=0 34=8 49=EX3IM 52=20250418-07:06:31.076 56=EX3IM 60=1174495999109 10=133 11=150 12=0 13=0 14=0 15=0 16=0 17=0 18=0 19=0 20=0 21=0 22=0 23=0 24=0 25=0 26=0 27=0 28=0 29=0 30=0 31=0 32=0 33=0 34=0 35=0 36=0 37=0 38=0 39=0 40=0 41=0 42=0 43=0 44=0 45=0 46=0 47=0 48=0 49=0 50=0 51=0 52=0 53=0 54=0 55=0 56=0 57=0 58=0 59=0 60=0 61=0 62=0 63=0 64=0 65=0 66=0 67=0 68=0 69=0 70=0 71=0 72=0 73=0 74=0 75=0 76=0 77=0 78=0 79=0 80=0 81=0 82=0 83=0 84=0 85=0 86=0 87=0 88=0 89=0 90=0 91=0 92=0 93=0 94=0 95=0 96=0 97=0 98=0 99=0 100=0 101=0 102=0 103=0 104=0 105=0 106=0 107=0 108=0 109=0 110=0 111=0 112=0 113=0 114=0 115=0 116=0 117=0 118=0 119=0 120=0 121=0 122=0 123=0 124=0 125=0 126=0 127=0 128=0 129=0 130=0 131=0 132=0 133=0 134=0 135=0 136=0 137=0 138=0 139=0 140=0 141=0 142=0 143=0 144=0 145=0 146=0 147=0 148=0 149=0 150=0 151=0 152=0 153=0 154=0 155=0 156=0 157=0 158=0 159=0 160=0 161=0 162=0 163=0 164=0 165=0 166=0 167=0 168=0 169=0 170=0 171=0 172=0 173=0 174=0 175=0 176=0 177=0 178=0 179=0 180=0 181=0 182=0 183=0 184=0 185=0 186=0 187=0 188=0 189=0 190=0 191=0 192=0 193=0 194=0 195=0 196=0 197=0 198=0 199=0 200=0 201=0 202=0 203=0 204=0 205=0 206=0 207=0 208=0 209=0 210=0 211=0 212=0 213=0 214=0 215=0 216=0 217=0 218=0 219=0 220=0 221=0 222=0 223=0 224=0 225=0 226=0 227=0 228=0 229=0 230=0 231=0 232=0 233=0 234=0 235=0 236=0 237=0 238=0 239=0 240=0 241=0 242=0 243=0 244=0 245=0 246=0 247=0 248=0 249=0 250=0 251=0 252=0 253=0 254=0 255=0 256=0 257=0 258=0 259=0 260=0 261=0 262=0 263=0 264=0 265=0 266=0 267=0 268=0 269=0 270=0 271=0 272=0 273=0 274=0 275=0 276=0 277=0 278=0 279=0 280=0 281=0 282=0 283=0 284=0 285=0 286=0 287=0 288=0 289=0 290=0 291=0 292=0 293=0 294=0 295=0 296=0 297=0 298=0 299=0 300=0 301=0 302=0 303=0 304=0 305=0 306=0 307=0 308=0 309=0 310=0 311=0 312=0 313=0 314=0 315=0 316=0 317=0 318=0 319=0 320=0 321=0 322=0 323=0 324=0 325=0 326=0 327=0 328=0 329=0 330=0 331=0 332=0 333=0 334=0 335=0 336=0 337=0 338=0 339=0 340=0 341=0 342=0 343=0 344=0 345=0 346=0 347=0 348=0 349=0 350=0 351=0 352=0 353=0 354=0 355=0 356=0 357=0 358=0 359=0 360=0 361=0 362=0 363=0 364=0 365=0 366=0 367=0 368=0 369=0 370=0 371=0 372=0 373=0 374=0 375=0 376=0 377=0 378=0 379=0 380=0 381=0 382=0 383=0 384=0 385=0 386=0 387=0 388=0 389=0 390=0 391=0 392=0 393=0 394=0 395=0 396=0 397=0 398=0 399=0 400=0 401=0 402=0 403=0 404=0 405=0 406=0 407=0 408=0 409=0 410=0 411=0 412=0 413=0 414=0 415=0 416=0 417=0 418=0 419=0 420=0 421=0 422=0 423=0 424=0 425=0 426=0 427=0 428=0 429=0 430=0 431=0 432=0 433=0 434=0 435=0 436=0 437=0 438=0 439=0 440=0 441=0 442=0 443=0 444=0 445=0 446=0 447=0 448=0 449=0 450=0 451=0 452=0 453=0 454=0 455=0 456=0 457=0 458=0 459=0 460=0 461=0 462=0 463=0 464=0 465=0 466=0 467=0 468=0 469=0 470=0 471=0 472=0 473=0 474=0 475=0 476=0 477=0 478=0 479=0 480=0 481=0 482=0 483=0 484=0 485=0 486=0 487=0 488=0 489=0 490=0 491=0 492=0 493=0 494=0 495=0 496=0 497=0 498=0 499=0 500=0 501=0 502=0 503=0 504=0 505=0 506=0 507=0 508=0 509=0 510=0 511=0 512=0 513=0 514=0 515=0 516=0 517=0 518=0 519=0 520=0 521=0 522=0 523=0 524=0 525=0 526=0 527=0 528=0 529=0 530=0 531=0 532=0 533=0 534=0 535=0 536=0 537=0 538=0 539=0 540=0 541=0 542=0 543=0 544=0 545=0 546=0 547=0 548=0 549=0 550=0 551=0 552=0 553=0 554=0 555=0 556=0 557=0 558=0 559=0 560=0 561=0 562=0 563=0 564=0 565=0 566=0 567=0 568=0 569=0 570=0 571=0 572=0 573=0 574=0 575=0 576=0 577=0 578=0 579=0 580=0 581=0 582=0 583=0 584=0 585=0 586=0 587=0 588=0 589=0 590=0 591=0 592=0 593=0 594=0 595=0 596=0 597=0 598=0 599=0 600=0 601=0 602=0 603=0 604=0 605=0 606=0 607=0 608=0 609=0 610=0 611=0 612=0 613=0 614=0 615=0 616=0 617=0 618=0 619=0 620=0 621=0 622=0 623=0 624=0 625=0 626=0 627=0 628=0 629=0 630=0 631=0 632=0 633=0 634=0 635=0 636=0 637=0 638=0 639=0 640=0 641=0 642=0 643=0 644=0 645=0 646=0 647=0 648=0 649=0 650=0 651=0 652=0 653=0 654=0 655=0 656=0 657=0 658=0 659=0 660=0 661=0 662=0 663=0 664=0 665=0 666=0 667=0 668=0 669=0 670=0 671=0 672=0 673=0 674=0 675=0 676=0 677=0 678=0 679=0 680=0 681=0 682=0 683=0 684=0 685=0 686=0 687=0 688=0 689=0 690=0 691=0 692=0 693=0 694=0 695=0 696=0 697=0 698=0 699=0 700=0 701=0 702=0 703=0 704=0 705=0 706=0 707=0 708=0 709=0 710=0 711=0 712=0 713=0 714=0 715=0 716=0 717=0 718=0 719=0 720=0 721=0 722=0 723=0 724=0 725=0 726=0 727=0 728=0 729=0 730=0 731=0 732=0 733=0 734=0 735=0 736=0 737=0 738=0 739=0 740=0 741=0 742=0 743=0 744=0 745=0 746=0 747=0 748=0 749=0 750=0 751=0 752=0 753=0 754=0 755=0 756=0 757=0 758=0 759=0 760=0 761=0 762=0 763=0 764=0 765=0 766=0 767=0 768=0 769=0 770=0 771=0 772=0 773=0 774=0 775=0 776=0 777=0 778=0 779=0 780=0 781=0 782=0 783=0 784=0 785=0 786=0 787=0 788=0 789=0 790=0 791=0 792=0 793=0 794=0 795=0 796=0 797=0 798=0 799=0 800=0 801=0 802=0 803=0 804=0 805=0 806=0 807=0 808=0 809=0 810=0 811=0 812=0 813=0 814=0 815=0 816=0 817=0 818=0 819=0 820=0 821=0 822=0 823=0 824=0 825=0 826=0 827=0 828=0 829=0 830=0 831=0 832=0 833=0 834=0 835=0 836=0 837=0 838=0 839=0 840=0 841=0 842=0 843=0 844=0 845=0 846=0 847=0 848=0 849=0 850=0 851=0 852=0 853=0 854=0 855=0 856=0 857=0 858=0 859=0 860=0 861=0 862=0 863=0 864=0 865=0 866=0 867=0 868=0 869=0 870=0 871=0 872=0 873=0 874=0 875=0 876=0 877=0 878=0 879=0 880=0 881=0 882=0 883=0 884=0 885=0 886=0 887=0 888=0 889=0 890=0 891=0 892=0 893=0 894=0 895=0 896=0 897=0 898=0 899=0 900=0 901=0 902=0 903=0 904=0 905=0 906=0 907=0 908=0 909=0 910=0 911=0 912=0 913=0 914=0 915=0 916=0 917=0 918=0 919=0 920=0 921=0 922=0 923=0 924=0 925=0 926=0 927=0 928=0 929=0 930=0 931=0 932=0 933=0 934=0 935=0 936=0 937=0 938=0 939=0 940=0 941=0 942=0 943=0 944=0 945=0 946=0 947=0 948=0 949=0 950=0 951=0 
```

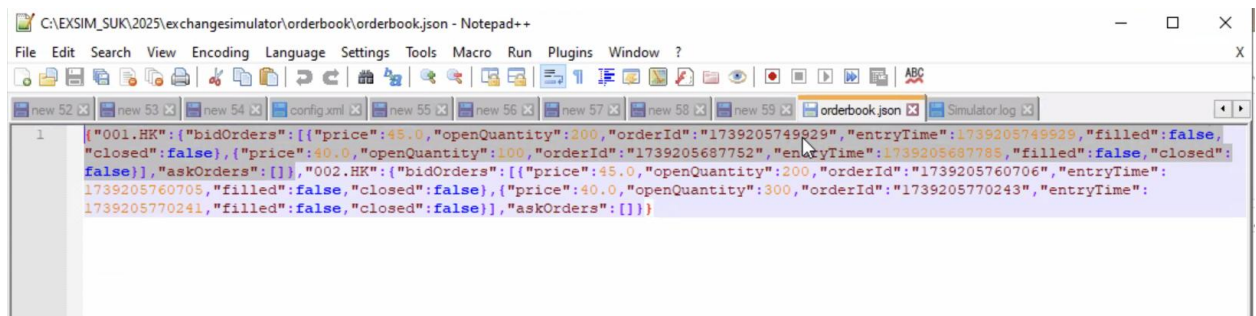
## Order Book Management and Persistence:

To maintain market state and support backend/frontend integration, the simulator uses both in-memory data structures and a PostgreSQL database.

- **In-Memory Storage:** The order book is represented using ConcurrentHashMap objects, providing fast access and real-time updates for each stock symbol.
- **PostgreSQL Database:** PostgreSQL is used for durable storage of:
  - Order and trade data
  - Execution records
  - Market statistics

Persistence is handled via DAO classes such as OrderBookDAO and MarketDataDAO, which abstract SQL logic and facilitate CRUD operations. PostgreSQL provides a reliable and scalable backend for storing and querying live and historical trading data.

- **JSON Serialization:** The PersistenceService serializes the current market state into orderbook.json using Jackson's ObjectMapper. This file contains per-symbol bid and ask queues, trade history, and timestamped updates. The orderbook.json file is continuously refreshed after every order placement, match, amendment, or cancellation—ensuring live synchronization with the frontend.



**Fig 2.2d: Bid Ask Order Queue (json)**

### **REST API for Frontend Integration:**

A dedicated REST API layer connects the PostgreSQL database to the frontend. This API retrieves the latest market data, order book snapshots, and trade statistics.

### **User Authentication (JWT):**

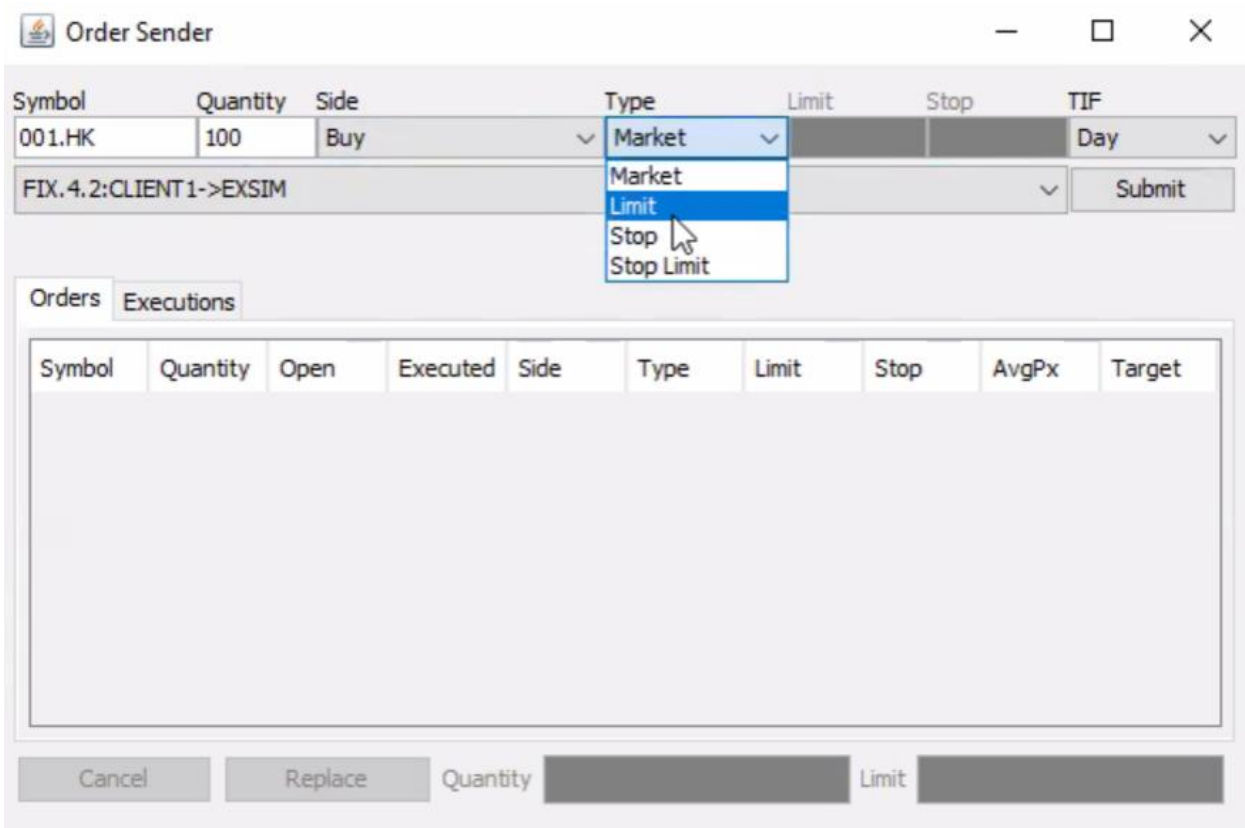
User authentication is handled at the beginning of each session using JWT (JSON Web Tokens), ensuring secure login and access to the frontend. All subsequent client requests are authorized through the JWT token, enabling user authentication management.

### **Graphical Test Tool for FIX Order Management (Swing GUI):**

A dedicated Java Swing-based GUI test tool was developed to simulate order interaction without requiring full integration with external EMS/OMS platforms. The Swing GUI is designed as a standalone desktop application, functioning as an order sender that connects to the simulator via FIX. This tool enables:

- Manual creation and submission of market and limit orders using FIX
- Real-time interaction with the matching engine, including Amend and Cancel operations
- Visualization of status updates and server responses directly in the UI
- Multiple client testing through configurable FIX session setups

This desktop utility provides traders and developers with a convenient interface for testing FIX behavior and debugging order flow logic.



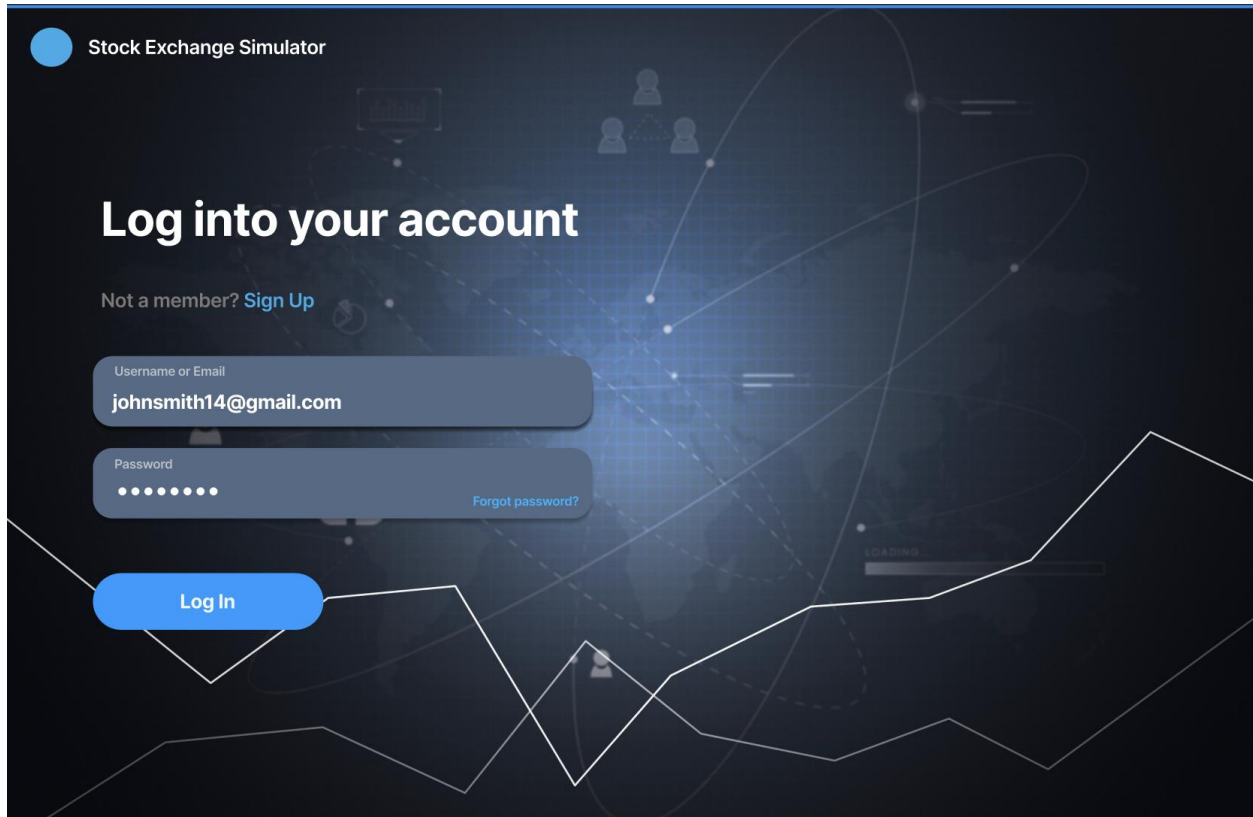
**Fig 2.2e: GUI Tool Supporting various Order Types**

### **Web Front-End for Real-Time Market Visualization:**

A modern web interface, built using React.js and Node.js, provides users with full visibility into the exchange's live state. User authentication is managed at login using JWT (JSON Web Tokens), ensuring secure access control across the frontend application.

## 1. User Authentication (Log In Dashboard)

- Displays a general login page, asking for user credentials such as email and password
- Key functions include:
  - Sign Up function
  - Forgot Password function



**Fig 2.2f: GUI- User Authentication**



## 2. Home Tab (Welcome Dashboard)

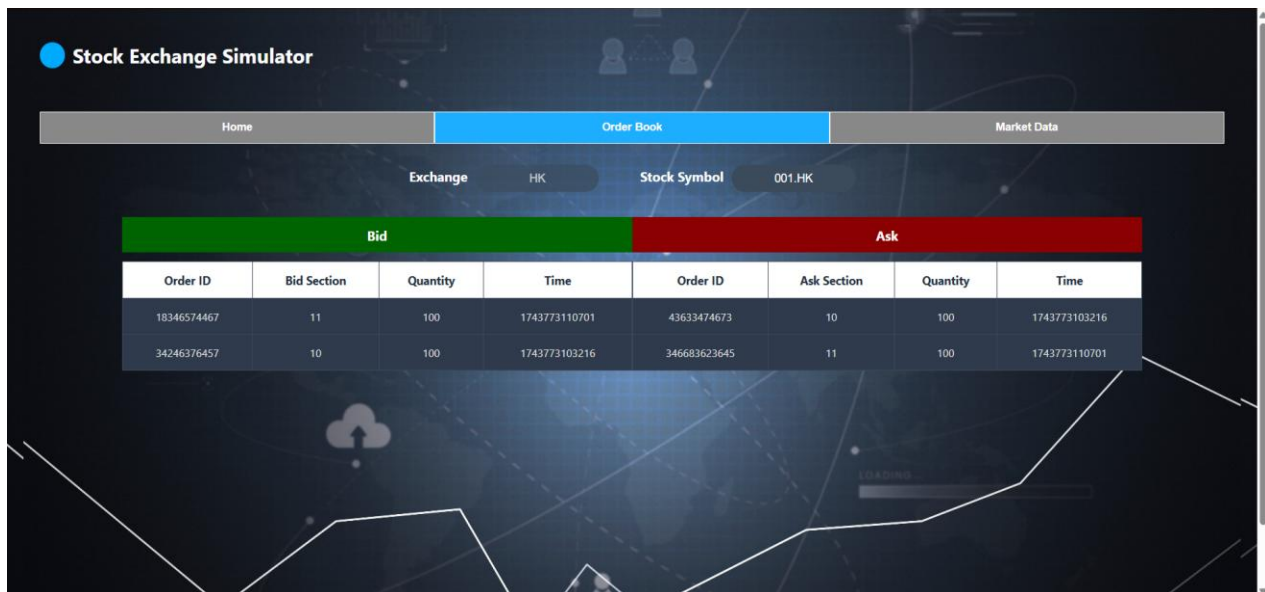
- Displays a personalized greeting and a table of the most active stocks.
- Key data points include:
  - Exchange
  - Stock Symbol
  - Total Executed Quantity
  - Average Price
  - Order Status (Open/Closed)



**Fig 2.2g: GUI- Home Page**

### 3. Order Book Tab

- Offers detailed visibility into bid and ask queues for any selected stock symbol.
- Data includes:
  - Order ID
  - Bid/Ask Price
  - Quantity
  - Timestamp
- This live representation reflects the exact state of the PostgreSQL-backed order book.
- Data is fetched in real-time through REST API calls.



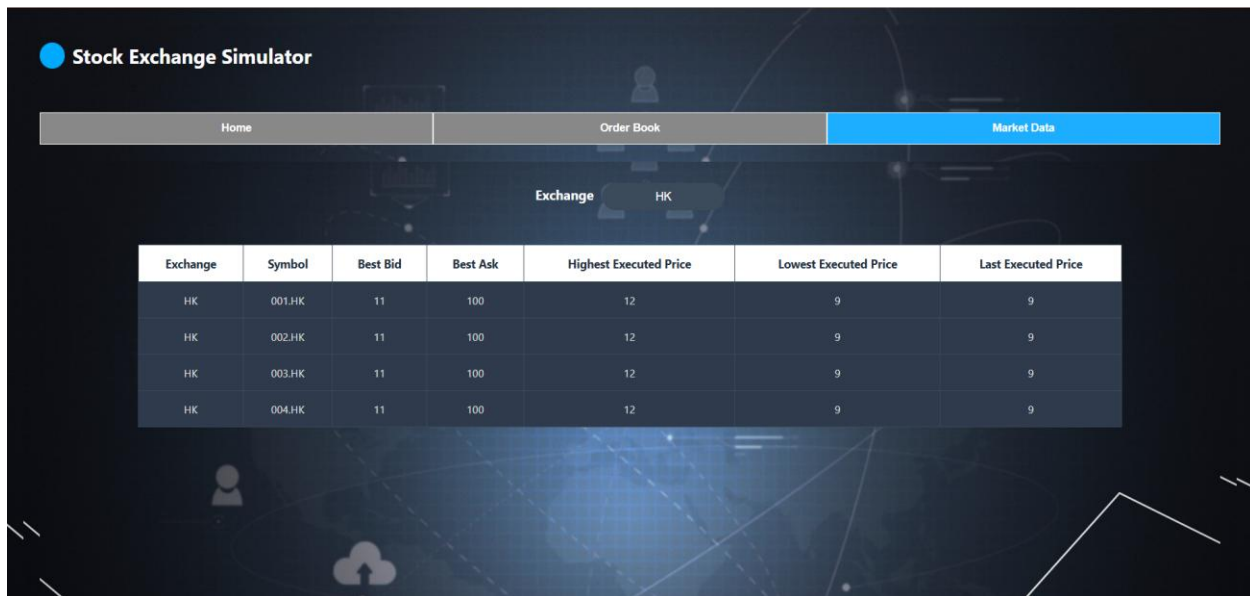
Bid				Ask			
Order ID	Bid Section	Quantity	Time	Order ID	Ask Section	Quantity	Time
18346574467	11	100	1743773110701	43633474673	10	100	1743773103216
34246376457	10	100	1743773103216	346683623645	11	100	1743773110701

**Fig 2.2h: GUI- Order Book**

#### 4. Market Data Tab

- Presents live trading statistics for each stock symbol on the exchange:
  - Best Bid / Best Ask
  - Highest / Lowest Executed Price
  - Last Executed Price
- The page is auto-updated using data pulled from the PostgreSQL database via the REST API.

This comprehensive full-stack implementation bridges backend trading logic with interactive, real-time visualization tools—creating a simulation platform that is intuitive, robust, and closely aligned with real-world trading systems.



Exchange	Symbol	Best Bid	Best Ask	Highest Executed Price	Lowest Executed Price	Last Executed Price
HK	001.HK	11	100	12	9	9
HK	002.HK	11	100	12	9	9
HK	003.HK	11	100	12	9	9
HK	004.HK	11	100	12	9	9

**Fig 2.2i: GUI- Market Data**

## 2.3 Testing

Testing was a crucial phase in ensuring that the Exchange Simulator accurately replicated real-world trading behavior and met both functional and performance requirements. We followed a multi-stage testing strategy that included unit testing, integration testing, and scenario-based testing using simulated market conditions.

### 2.3.1 Unit Testing:

Unit testing was conducted for all key backend services and domain logic classes, including:

- MatchingService
- FIXOrderReceiverService
- PersistenceService
- ExchangeRuleValidatorService
- PriceUpdateService

Each class was tested in isolation using custom-built test harnesses. These unit tests verified:

- **Order Insertion and Matching Logic:** Orders were correctly inserted into the appropriate bid/ask queues, and matching occurred according to the price-time priority mechanism.
- **Validation Rules:** The ExchangeRuleValidatorService correctly enforced exchange-specific constraints like tick size granularity and Time-in-Force (TIF) restrictions.
- **Execution Generation:** The TradeGeneratorService correctly logged executions with accurate price, quantity, and timestamp.
- **Market Statistics Update:** PriceUpdateService updated market statistics such as best bid/ask and last traded price after every match.

- **Persistence Layer Functions:** The DAO classes (OrderBookDAO and MarketDataDAO) performed CRUD operations correctly with PostgreSQL.

Assertions and logging statements were used to confirm expected behavior, detect anomalies, and ensure data consistency across states.

### 2.3.2 Integration Testing:

After validating the individual components, end-to-end integration testing was performed to verify seamless system functionality. This testing followed a manual simulation approach, mimicking real trading workflows through FIX messages.

Key integration tests included:

- **FIX Protocol Flow:** Using the Swing GUI and QuickFIX/J, FIX messages (types D, F, G) were sent to the simulator. The response lifecycle was traced in ExSimApplication, ensuring that orders were accepted, modified, cancelled, or rejected appropriately.
- **Service Interoperability:** Orders were routed from the FIX engine to the MatchingService, processed, and the resulting trades were persisted by the PersistenceService and serialized to orderbook.json.
- **Frontend-Backend Sync:** We verified that every trade execution or order update on the backend was correctly reflected in the web frontend's Order Book and Market Data tabs using live data fetched via REST API from PostgreSQL.

### 2.3.3 Walkthrough Testing Scenarios:

To further demonstrate the completeness of integration testing and provide readers with a clearer understanding of how various modules work together, we conducted a comprehensive walkthrough of several realistic trading scenarios. These step-by-step cases trace the full lifecycle of orders—starting from the receipt of FIX messages, passing through validation,

insertion into symbol-specific queues within the matching engine, and finally persisting the updated market state into PostgreSQL and orderbook.json.

Each scenario highlights not just the interaction between services (such as the FIXOrderReceiver, ExchangeRuleValidatorService, MatchingService, and PersistenceService), but also showcases how these updates propagate to the frontend via REST APIs. This allows the user to monitor real-time changes through the live Order Book and Market Data interfaces.

The following section documents a selection of these test cases—ranging from simple limit order insertions to partial and full matches, validation failures, and multi-symbol maintenance—illustrating the robustness of the system under a variety of operational conditions.

### **Scenario 1: Placing Multiple Buy Limit Orders for 001.HK**

**Description:** Client sends three limit buy orders for 001.HK at prices 20.0, 20.5, and 21.0 for 100 quantities each.

Step 1: Buy @ 20.0, Qty: 100

#### **FIX Message (New Order Received):**

```
8=FIX.4.2|9=140|35=D|34=349|49=CLIENT1|52=20250417-
19:31:27.746|56=EXSIM|11=1744918287731|21=1|38=100|40=2|44=20.0|54=1|55=001.HK|59=
0|10=092|
```

#### **FIX Message (Ack Response Sent):**

```
<20250417-19:31:27, FIX.4.2:EXSIM->CLIENT1, outgoing>
8=FIX.4.2|9=164|35=8|34=349|49=EXSIM|52=20250417-
19:31:27.762|56=CLIENT1|11=1744918287731|17=020|20=0|31=0|32=0|37=1744918287731|38
=100|39=0|40=2|54=1|55=001.HK|150=0|151=100|10=026|
```

#### **JSON:**

```
"001.HK": {
  "bidOrders": [
    { "price": 20.0, "openQuantity": 100, "orderId": "1744918287731", "entryTime":
1744918287746, "closed": false, "filled": false }
  ],
  "askOrders": []
}
```

**001.HKBid-Ask Queue:**

**BIDS:**

*\$21.00 x 100 - CLIENT1 - Fri Apr 18 03:31:45 HKT 2025*

**ASKS: -**

Similarly for Buy @ 21.0, Qty: 100 and Buy @ 22.0, Qty: 100

**JSON:**

```
"001.HK": {
  "bidOrders": [
    { "price": 21.0, "openQuantity": 100, "orderId": "1744918305597", "entryTime":
1744918305594, "closed": false, "filled": false },
    { "price": 20.5, "openQuantity": 100, "orderId": "1744918298641", "entryTime":
1744918298639, "closed": false, "filled": false },
    { "price": 20.0, "openQuantity": 100, "orderId": "1744918287731", "entryTime":
1744918287746, "closed": false, "filled": false }
  ],
  "askOrders": []
}
```

**001.HKBid-Ask Queue:**

**BIDS:**

*\$21.00 x 100 - CLIENT1 - Fri Apr 18 03:31:45 HKT 2025*

*\$20.50 x 100 - CLIENT1 - Fri Apr 18 03:31:38 HKT 2025*

*\$20.00 x 100 - CLIENT1 - Fri Apr 18 03:31:27 HKT 2025*

**ASKS: -**

**Scenario 2: Sell Side Limit Order Fully Matches with Best Bid**

**Description:** A sell limit order for 001.HK is submitted at a price of 21.0 (same as current best bid) for quantity 50. The order fully matches with the best buy order (also 100 @ 21.0), and both clients receive execution reports.

**FIX Message (New Order Received):**

8=FIX.4.2|9=140|35=D|34=234|49=CLIENT1|52=20250417-19:38:36.010|56=EXSIM|11=1744918716020|21=1|38=50|40=2|44=21.0|54=2|55=001.HK|59=0|10=070|

**FIX Message (Ack Response Sent):**

<20250417-19:38:36, FIX.4.2:EXSIM->CLIENT1, outgoing>  
8=FIX.4.2|9=164|35=8|34=254|49=EXSIM|52=20250417-19:38:36.010|56=CLIENT1|11=1744918716020|17=112|20=0|31=0|32=0|37=1744918716020|38=50|39=0|40=2|54=2|55=001.HK|150=0|151=50|10=010|

**FIX Message (Execution Report – Seller):**

8=FIX.4.2|9=167|35=8|34=264|49=EXSIM|52=20250417-19:38:36.010|56=CLIENT1|11=1744918716020|17=122|20=0|31=21.0|32=50|37=1744918716020|38=50|39=2|40=2|54=2|55=001.HK|150=2|151=0|10=174|

**FIX Message (Execution Report – Buyer Partial Fill):**

8=FIX.4.2|9=169|35=8|34=274|49=EXSIM|52=20250417-19:38:36.010|56=CLIENT1|11=1744918305597|17=132|20=0|31=21.0|32=50|37=1744918305597|38=100|39=1|40=2|54=1|55=001.HK|150=1|151=50|10=042|

**Outcome:**

The sell-side order is **fully executed** at \$21.00 against the existing best bid.

The buy-side order for 100 at \$21.00 is **partially filled** (50/100), and 50 remains in the order book.

**001.HKBid-Ask Queue:**

**BIDS:**

*\$21.00 x 50 - CLIENT1 - Fri Apr 18 03:31:45 HKT 2025*

*\$20.50 x 100 - CLIENT1 - Fri Apr 18 03:31:38 HKT 2025*

*\$20.00 x 100 - CLIENT1 - Fri Apr 18 03:31:27 HKT 2025*

**ASKS: -**

**JSON:**

"001.HK": {

"bidOrders": [



```
{ "price": 21.0, "openQuantity": 50, "orderId": "1744918305597", "entryTime":
1744918305594, "closed": false, "filled": false },

{ "price": 20.5, "openQuantity": 100, "orderId": "1744918298641", "entryTime":
1744918298639, "closed": false, "filled": false },

{ "price": 20.0, "openQuantity": 100, "orderId": "1744918287731", "entryTime":
1744918287746, "closed": false, "filled": false }

],

"askOrders": []

}
```

### **Scenario 3: Sell Limit Orders Received with price greater than best bid for 001.HK**

**Description:** A sell order for 001.HK is placed at 22.0, higher than the best bid (21.0). It is queued on the ask side.

- FIX Acknowledgment Sent: Order acknowledged and queued.

#### **001.HKBid-Ask Queue:**

##### ***BIDS:***

*\$21.00 x 50 - CLIENT1 - Fri Apr 18 03:31:45 HKT 2025*

*\$20.50 x 100 - CLIENT1 - Fri Apr 18 03:31:38 HKT 2025*

*\$20.00 x 100 - CLIENT1 - Fri Apr 18 03:31:27 HKT 2025*

##### ***ASKS:***

*\$22.0 x 50- CLINT1 - Fri Apr 18 07:51:41 HKT 2025*

#### **JSON:**

```
"001.HK": {

  "bidOrders": [

    { "price": 21.0, "openQuantity": 50, "orderId": "1744918305597", "entryTime":
1744918305594, "closed": false, "filled": false },

    { "price": 20.5, "openQuantity": 100, "orderId": "1744918298641", "entryTime":
```

```
1744918298639, "closed": false, "filled": false},
  { "price": 20.0, "openQuantity": 100, "orderId": "1744918287731", "entryTime":
1744918287746, "closed": false, "filled": false }
],
"askOrders": [
  { "price": 22.0, "openQuantity": 50, "orderId": "1744918736078", "entryTime":
1744918736078, "closed": false, "filled": false }
]
}
```

#### **Scenario 4: Sell Limit Orders Received with price greater than best bid for 001.HK**

**Description:** A sell order for 001.HK is placed at 23.0, higher than the best bid (21.0). It is queued on the ask side.

#### **001.HKBid-Ask Queue:**

##### ***BIDS:***

*\$21.00 x 50 - CLIENT1 - Fri Apr 18 03:31:45 HKT 2025*

*\$20.50 x 100 - CLIENT1 - Fri Apr 18 03:31:38 HKT 2025*

*\$20.00 x 100 - CLIENT1 - Fri Apr 18 03:31:27 HKT 2025*

##### ***ASKS:***

*\$22.0 x 50- CLINT1 - Fri Apr 18 07:51:41 HKT 2025*

*\$23.0 x 50- CLINT1 - Fri Apr 18 12:06:58 HKT 2025*

#### **JSON:**

```
"001.HK": {
  "bidOrders": [
    { "price": 21.0, "openQuantity": 50, "orderId": "1744957832497", "entryTime":
1744957832494, "closed": false, "filled": false },
    { "price": 20.5, "openQuantity": 100, "orderId": "1744957824120", "entryTime":
1744957824118, "closed": false, "filled": false },
```

```
{ "price": 20.0, "openQuantity": 100, "orderId": "1744957816440", "entryTime":
1744957816454, "closed": false, "filled": false }

],

"askOrders": [

  { "price": 22.0, "openQuantity": 50, "orderId": "1744957882102", "entryTime":
1744957882097, "closed": false, "filled": false },

  { "price": 23.0, "openQuantity": 50, "orderId": "1744957900853", "entryTime":
1744957900862, "closed": false, "filled": false }

]

}
```

**Scenario 5: A sell order is placed for 001.HK at 20.0, which is lower than the best bid (21.0).**

**Validation Result:** FIX Order Rejected – Price below best bid.

*Sell Side Order reject sent back to sender client:*

**001.HKBid-Ask Queue (unchanged):**

**BIDS:**

*\$21.00 x 50 - CLIENT1 - Fri Apr 18 03:31:45 HKT 2025*

*\$20.50 x 100 - CLIENT1 - Fri Apr 18 03:31:38 HKT 2025*

*\$20.00 x 100 - CLIENT1 - Fri Apr 18 03:31:27 HKT 2025*

**ASKS:**

*\$22.0 x 50- CLINT1 - Fri Apr 18 07:51:41 HKT 2025*

*\$23.0 x 50- CLINT1 - Fri Apr 18 12:06:58 HKT 2025*

**Scenario 6: Sell Order at 21.0 (Partial Fill, Rest Queued)**

**Description:** A large sell order for 150 shares at 21.0 is placed. The best bid has only 50 available, so the order is partially filled and the remaining 100 is queued.

**Execution Outcome:**

- Sell order: Partially filled for 50
- Buy order at 21.0: Fully filled and removed
- Remaining sell order of 100 is queued at ask = 21.0

**001.HKBid-Ask Queue:**

***BIDS:***

*\$20.50 x 100 - CLIENT1 - Fri Apr 18 03:31:38 HKT 2025*

*\$20.00 x 100 - CLIENT1 - Fri Apr 18 03:31:27 HKT 2025*

***ASKS:***

*\$21.00 x 50 - CLIENT1 - Fri Apr 18 13:31:45 HKT 2025*

*\$22.0 x 50- CLINT1 - Fri Apr 18 07:51:41 HKT 2025*

*\$23.0 x 50- CLINT1 - Fri Apr 18 12:06:58 HKT 2025*

**JSON:**

"001.HK": {

"bidOrders": [

{ "price": 20.5, "openQuantity": 100, "orderId": "1744957824120", "entryTime":  
1744957824118, "closed": false, "filled": false },

{ "price": 20.0, "openQuantity": 100, "orderId": "1744957816440", "entryTime":  
1744957816454, "closed": false, "filled": false }

],

"askOrders": [

{ "price": 21.0, "openQuantity": 100, "orderId": "1744958513491", "entryTime":  
1744958513483, "closed": false, "filled": false },

{ "price": 22.0, "openQuantity": 50, "orderId": "1744957882102", "entryTime":  
1744957882097, "closed": false, "filled": false },

```
{ "price": 23.0, "openQuantity": 50, "orderId": "1744957900853", "entryTime":  
1744957900862, "closed": false, "filled": false }  
  
]  
  
}
```

### **Scenario 7: Buy Order Rejected – Price Above Best**

**Description:** A buy order is placed for 001.HK at 22.0, while the best ask is 21.0. As per the exchange validation rules, a buy order price exceeding the best ask is invalid and hence rejected.

**Validation Result:** FIX Order Rejected – Price exceeds best ask.

**001.HKBid-Ask Queue (unchanged):**

**BIDS:**

**\$20.50 x 100 - CLIENT1 - Fri Apr 18 03:31:38 HKT 2025**

**\$20.00 x 100 - CLIENT1 - Fri Apr 18 03:31:27 HKT 2025**

**ASKS:**

**\$21.00 x 50 - CLIENT1 - Fri Apr 18 13:31:45 HKT 2025**

**\$22.0 x 50- CLINT2 - Fri Apr 18 07:51:41 HKT 2025**

**\$23.0 x 50- CLINT3 - Fri Apr 18 12:06:58 HKT 2025**

### **Scenario 8: Introducing a New Symbol – 002.HK Buy Orders**

**Description:** A client places a limit buy order for a new stock symbol, 002.HK, at 30.0 for 100 shares.

**Execution Result:** Order is accepted and added to a fresh order book for 002.HK.

**002.HKBid-Ask Queue:**

**BIDS:**

**\$30 x 100 - CLIENT1 - Fri Apr 18 04:22:47 HKT 2025**

**ASKS: -**

**JSON:**

```
"002.HK": {  
  
  "bidOrders": [  
  
    { "price": 30.0, "openQuantity": 100, "orderId": "1744959878508", "entryTime":  
1744959878498, "closed": false, "filled": false }  
  
  ],  
  
  "askOrders": []  
  
}
```

**Scenario 9: Second Buy Order @31.0 for 002.HK**

**Description:** A new buy order is placed for 002.HK at 31.0. It is inserted above the previous order due to higher price priority.

**002.HKBid-Ask Queue:**

***BIDS:***

***\$31 x 100 - CLIENT1 - Fri Apr 18 05:51:06 HKT 2025***

***\$30 x 100 - CLIENT1 - Fri Apr 18 04:22:47 HKT 2025***

**ASKS: -**

**JSON:**

```
"002.HK": {  
  
  "bidOrders": [  
  
    { "price": 31.0, "openQuantity": 100, "orderId": "1744959890929", "entryTime":
```

```
1744959890918, "closed": false, "filled": false},

  {"price": 30.0, "openQuantity": 100, "orderId": "1744959878508", "entryTime":
1744959878498, "closed": false, "filled": false}

],

"askOrders": []

}
```

### **Scenario 10: Third Buy Order @32.0 for 002.HK**

**Description:** A third limit buy order is submitted at 32.0, now forming the new best bid.

**002.HKBid-Ask Queue:**

***BIDS:***

*\$32 x 100 - CLIENT1 - Fri Apr 18 10:26:17 HKT 2025*

*\$31 x 100 - CLIENT1 - Fri Apr 18 05:51:06 HKT 2025*

*\$30 x 100 - CLIENT1 - Fri Apr 18 04:22:47 HKT 2025*

***ASKS: -***

**JSON:**

```
"002.HK": {

  "bidOrders": [

    {"price": 32.0, "openQuantity": 100, "orderId": "1744959895588", "entryTime":
1744959895576, "closed": false, "filled": false},

    {"price": 31.0, "openQuantity": 100, "orderId": "1744959890929", "entryTime":
1744959890918, "closed": false, "filled": false},
```

```
{ "price": 30.0, "openQuantity": 100, "orderId": "1744959878508", "entryTime":  
1744959878498, "closed": false, "filled": false }  
  
],  
  
"askOrders": []  
  
}
```

### **Scenario 11: Sell Order Rejected @30.0 – Price Below Best Bid**

**Description:** A sell order for 002.HK at 30.0 is rejected since it is priced below the best bid (32.0), violating the exchange's order validation rule.

**Validation Result:** FIX Order Rejected – Price below best bid.

**002.HKBid-Ask Queue (unchanged):**

**BIDS:**

*\$32 x 100 - CLIENT1 - Fri Apr 18 10:26:17 HKT 2025*

*\$31 x 100 - CLIENT1 - Fri Apr 18 05:51:06 HKT 2025*

*\$30 x 100 - CLIENT1 - Fri Apr 18 04:22:47 HKT 2025*

**ASKS:** -

### **Scenario 12: Valid Sell Order @32.0 Fully Matches Best Bid**

**Description:** A valid sell order for 002.HK is placed at 32.0 for quantity 150. It fully matches with the best bid (buy) of 32.0 x 100, leaving 50 in the sell queue.

**Execution Result:**

- Buy order @32.0: Fully filled
- Sell order @32.0: Partially filled (50 remaining)

**002.HKBid-Ask Queue:**



**BIDS:**

*\$31 x 100 - CLIENT1 - Fri Apr 18 05:51:06 HKT 2025*

*\$30 x 100 - CLIENT1 - Fri Apr 18 04:22:47 HKT 2025*

**ASKS:**

*\$32 x 100 - CLIENT1 - Fri Apr 18 14:09:33 HKT 2025*

**JSON:**

```
"002.HK": {  
  
  "bidOrders": [  
  
    { "price": 31.0, "openQuantity": 100, "orderId": "1744959890929", "entryTime":  
1744959890918, "closed": false, "filled": false },  
  
    { "price": 30.0, "openQuantity": 100, "orderId": "1744959878508", "entryTime":  
1744959878498, "closed": false, "filled": false }  
  
  ],  
  
  "askOrders": [  
  
    { "price": 32.0, "openQuantity": 50, "orderId": "1744959966257", "entryTime":  
1744959966243, "closed": false, "filled": false }  
  
  ]  
  
}
```

**Scenario 13: Rejected Sell Order @30.0 (Still Below Best Bid)**

**Description:** Another attempt to place a sell order at 30.0 for 002.HK fails due to price being below the current best bid (31.0).

**Validation Result:** FIX Order Rejected – Price below best bid.

**002.HKBid-Ask Queue (unchanged):**

***BIDS:***

*\$31 x 100 - CLIENT1 - Fri Apr 18 05:51:06 HKT 2025*

*\$30 x 100 - CLIENT1 - Fri Apr 18 04:22:47 HKT 2025*

***ASKS:***

*\$32 x 100 - CLIENT1 - Fri Apr 18 14:09:33 HKT 2025*

**Scenario 14: Sell Order @31.0 Matches Best Bid Fully**

**Description:** A sell order for 002.HK at 31.0 for 150 is submitted. It fully matches the top bid (buy) of 31.0 x 100, leaving 50 queued at the top of the ask side.

**Execution Result:**

- Buy order @31.0: Fully filled
- Sell order @31.0: Partially filled, 50 queued

**002.HKBid-Ask Queue (unchanged):**

***BIDS:***

*\$30 x 100 - CLIENT1 - Fri Apr 18 04:22:47 HKT 2025*

***ASKS:***

*\$31 x 100 - CLIENT1 - Fri Apr 18 15:51:06 HKT 2025*

*\$32 x 100 - CLIENT1 - Fri Apr 18 14:09:33 HKT 2025*

**JSON:**

"002.HK": {

"bidOrders": [

{ "price": 30.0, "openQuantity": 100, "orderId": "1744959878508", "entryTime":

```

1744959878498, "closed": false, "filled": false}

],

"askOrders": [

  {"price": 31.0, "openQuantity": 50, "orderId": "1744959998015", "entryTime":
1744959998014, "closed": false, "filled": false},

  {"price": 32.0, "openQuantity": 50, "orderId": "1744959966257", "entryTime":
1744959966243, "closed": false, "filled": false}

]

}

```

Order Sender

Symbol

Quantity

Side

Type

Limit

Stop

TIF

002.HK

150

Sell

Limit

31

Day

FIX.4.2:CLIENT1->EXSIM

Submit

Orders

Executions

Symbol	Quantity	Open	Executed	Side	Type	Limit	Stop	AvgPx	Target
001.HK	100	100	0	Buy	Limit	20.0		0.0	EXSIM
001.HK	100	100	0	Buy	Limit	20.5		0.0	EXSIM
001.HK	100	0	100	Buy	Limit	21.0		21.0	EXSIM
001.HK	50	0	50	Sell	Limit	21.0		21.0	EXSIM
001.HK	50	50	0	Sell	Limit	22.0		0.0	EXSIM
001.HK	50	50	0	Sell	Limit	23.0		0.0	EXSIM
001.HK	50	0	0	Sell	Limit	20.0		0.0	EXSIM
001.HK	150	100	50	Sell	Limit	21.0		21.0	EXSIM
001.HK	100	0	0	Buy	Limit	22.0		0.0	EXSIM
002.HK	100	100	0	Buy	Limit	30.0		0.0	EXSIM
002.HK	100	0	100	Buy	Limit	31.0		31.0	EXSIM
002.HK	100	0	100	Buy	Limit	32.0		32.0	EXSIM
002.HK	150	0	0	Sell	Limit	30.0		0.0	EXSIM
002.HK	150	50	100	Sell	Limit	32.0		32.0	EXSIM
002.HK	150	0	0	Sell	Limit	30.0		0.0	EXSIM
002.HK	150	50	100	Sell	Limit	31.0		31.0	EXSIM

Cancel

Replace

Quantity

Limit

**Fig 2.3a: Order List**

**Order Sender**

Symbol	Quantity	Side	Type	Limit	Stop	TIF
002.HK	150	Sell	Limit	31		Day

FIX. 4.2:CLIENT 1->EXSIM Submit

**Orders** **Executions**

Symbol	Quantity	Side	Price
001.HK	50	Sell	21.0
001.HK	50	Buy	21.0
001.HK	50	Sell	0.0
001.HK	50	Sell	21.0
001.HK	50	Buy	21.0
001.HK	100	Buy	0.0
002.HK	150	Sell	0.0
002.HK	100	Sell	32.0
002.HK	100	Buy	32.0
002.HK	150	Sell	0.0
002.HK	100	Sell	31.0
002.HK	100	Buy	31.0

Cancel Replace Quantity Limit

**Fig 2.3b: Execution List**

This interface displays a series of submitted buy and sell limit orders for multiple stocks (001.HK and 002.HK) along with their execution status. The Order List in Fig 2.3a shows the order placement panel and the live state of each order—highlighted in green for successful executions and red for rejections. The latter figure provides a breakdown of execution details, including price and quantity, verifying accurate order processing and matching behaviour of the exchange simulator.

## 2.4 Evaluation

Following the comprehensive unit and integration testing phases, the exchange simulator underwent a structured evaluation to validate its real-world applicability, robustness, and compliance with exchange-specific requirements. The evaluation focused on accuracy, performance, scalability, and overall system integrity, and was carried out using realistic trading scenarios via both GUI interaction and FIX protocol-based message flow.

### **Accuracy of Order Book:**

We evaluated the accuracy of the order book by simulating various market scenarios including multiple buy/sell limit orders, partial and full executions, and invalid order submissions. The system consistently maintained price-time priority across all trades, correctly prioritized orders, and updated the bid-ask queue dynamically. Each order was persisted and reflected in the JSON structure and PostgreSQL database without error. Additionally, scenarios involving multiple stock symbols (e.g., 001.HK and 002.HK) confirmed the simulator's ability to isolate and manage distinct order books accurately.

### **Order Validity and Rule Compliance:**

A critical component of evaluation was testing order validation logic. The simulator enforced exchange-specific rules such as:

- Rejecting orders with prices outside valid tick ranges.
- Rejecting orders that violated price-time priority (e.g., limit buy orders above best ask or sell orders below best bid).
- Accepting and partially filling orders when matched conditions were met. All validation checks were logged and traced via the console, and the corresponding rejection acknowledgments were properly routed back to the client via FIX messages. This validated the correctness of rule enforcement and system response.

### **Scalability and Efficiency:**

We stress-tested the system by placing a large number of concurrent buy and sell orders across multiple instruments. The simulator efficiently handled the queueing, matching, and serialization of these orders into both the live JSON market data file and the PostgreSQL database. There were no memory leaks, dropped packets, or backlog in the message queue, indicating the simulator's scalability and high-throughput capability.

### **Integration and Synchronization:**

The full stack, ranging from FIX communication to order matching, persistence, and frontend visualization, was tested in integration. Executed trades and book updates on the backend were instantly reflected in the frontend Order Book and Market Data tabs, confirming seamless synchronization between services. RESTful APIs provided accurate and real-time data, ensuring that the web interface remained aligned with the simulator's backend state throughout the tests.

### **GUI and Usability:**

The Swing-based GUI for order submission and execution monitoring functioned as an effective tool for manual testing and interaction. Users were able to place complex trade orders, observe color-coded execution outcomes, and verify trade results in real time. This interface made the testing process intuitive and provided clear visual feedback, supporting manual validation and debugging.

Through targeted scenario testing, rule validation, and performance evaluation, we have demonstrated that the simulator faithfully replicates the behavior of a real exchange under diverse market conditions. It met its core objectives of:

- Accurate trade matching and prioritization.
- Rigorous enforcement of exchange rules.
- Scalable performance under load.
- End-to-end service interoperability from order entry to frontend rendering.

## 3 Discussion

### **Persistence Layer Migration:**

One of the significant challenges faced during the development was related to the persistence layer. Initially, the system used an H2 in-memory database due to its simplicity and speed, especially for early-stage testing and integration. However, during frontend integration with the ReactJS-based UI, we encountered limitations with H2's compatibility and REST-based data handling, particularly when it came to data consistency and concurrent access across services.

To resolve this, we transitioned our persistence layer to PostgreSQL in the later stages of the project. Thanks to the modular segregation provided by our DAO (Data Access Object) architecture, the migration process was streamlined. The abstraction of persistence logic from the business logic ensured that changes to the underlying database had minimal impact on the overall system design, significantly reducing development time and avoiding widespread refactoring.

### **Market Data Replay and Order Book Initialization:**

The project initially included a Market Data Replay Service intended to bootstrap the order book upon server start-up by replaying historical orders. However, deeper integration with live order management and dynamic book reconstruction required enhancements to the service. As full integration was not feasible within the project timeline, we chose to disable the feature in the final build.

### **Order Validation Rule Scope:**

The current validation logic supports core rules for both the Hong Kong and Japan exchanges, including price boundaries, order side checks, and best bid/ask validations. However, not all regulatory and structural rules from these markets are fully implemented—particularly edge cases like short-sell bans, tick-size constraints under volatility control mechanisms, and trading halts. The ValidationService is modular and extensible, allowing for incremental implementation of additional rules as required.

### **Frontend Integration with Java Backend:**

Integrating the standalone Java backend with the ReactJS frontend posed initial challenges, especially in exposing trade and market data via RESTful services. While Java based server handled FIX sessions and internal services, React required structured and performant endpoints to fetch live order data. We resolved this by establishing PostgreSQL as the common integration point, with the frontend polling the database through REST APIs.

### **Scalability and Deployment Considerations:**

The Exchange Simulator is inherently scalable due to its service-oriented architecture. Each exchange instance is encapsulated and can run independently, allowing multiple exchanges (e.g., HK, JP) to be simulated concurrently by launching parallel simulator engines. While high availability and fault tolerance were not priorities for this testing tool, the architecture can support enhancements such as:

- Failover support using container orchestration (e.g., Kubernetes)
- State restoration from persisted order books upon service restart
- Sharding or load-balancing across trading instruments for performance



## 4 Conclusion

The successful development of the Stock Exchange Simulator represents a significant stride toward democratizing access to sophisticated, exchange-grade testing environments for EMS and OMS systems. In an industry where real-world exchange testing is constrained by cost, time windows, and limited access, our simulator offers a high-fidelity alternative that is accessible, scalable, and deeply aligned with real exchange protocols and behaviors.

Throughout the course of the project, we adopted a modular, service-oriented design that not only streamlined the development process but also laid a robust foundation for future scalability and cross-exchange adaptability. Our FIX-based matching engine replicates the intricacies of price-time priority and tick-size enforcement, while the validator enforces primary exchange rules for multiple markets. With the integration of PostgreSQL as a persistent storage layer and RESTful APIs to communicate with the ReactJS frontend, we established a full-stack system that is both technically resilient and user-friendly.

The GUI Swing application and ReactJS frontend both played vital roles in enhancing usability. The former enabled low-barrier testing of FIX order flows, while the latter provided traders and developers with a clean, responsive, and authenticated view of live order books and market data thereby, bringing the simulation one step closer to a production-grade testing interface.

Perhaps the most impactful aspect of the simulator is its extensibility. By encapsulating the simulator engine per exchange instance, the system can support multi-exchange environments concurrently. This, combined with its open-ended validation framework and order book architecture, positions the simulator for future enhancements such as short-sell validation and volatility interruption logic. With containerization and orchestration, it could easily scale to meet the demands of institutional-grade testing.

In summary, this project not only fulfils the stated objectives but also lays the groundwork for continued innovation in the realm of algorithmic trading validation. It bridges the gap between academic experimentation and industry application, offering a tangible, production-adjacent tool that can serve start-ups, financial institutions, and individual researchers alike. The Stock

Exchange Simulator stands as a testament to thoughtful design, resilient engineering, and a deep understanding of the financial ecosystem.

# 5 Project Planning

## 5.1 Distribution of Work

Task	Akshat	Sukruti
Initial Proposal of Idea	○	●
Analysing the feasibility	●	○
Expectation Discovery	●	○
Literature Survey	○	●
Analyse the existing technology	○	●
Writing Proposal Report	●	●
Develop order matching algorithm	○	●
Develop trade generation service	●	○
Develop market database	●	○
Develop trade validator	○	●
Develop Market data replay service	○	●
Incorporate market database	○	●
Developing UI for user interface	●	○
Develop the user authentication system	●	○
Test the user authentication system	●	○
Test order matching algorithm	○	●
Test trade generation service	○	●
Test market database	●	○
Test trade validator	○	●
Test market data replay service	○	●

# RO4 FYP – Stock Exchange Simulator

Write the monthly reports	●	●
Write the progress report	●	●
Write the final report	●	●
Prepare for the presentation	●	●
Prepare for video trailer	●	●

● Leader ○ Assistant

## 5.2 GANTT Chart

Task	Jun	July	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr
Initial Proposal of Idea	Completed										
Analysing the feasibility	Completed	Completed									
Expectation Discovery		Completed									
Literature Survey			Completed								
Analyse the existing technology			Completed								
Writing Proposal Report			Completed	Completed							
Develop order matching algorithm				Completed	Completed						
Develop trade generation service					Completed						
Develop market database					Completed						
Develop trade validator					Completed	Completed					
Develop Market data replay service						Completed	Completed				
Incorporate market database							Completed				
Developing UI for user interface								Completed			
Develop the user authentication system									Completed		
Test the user authentication system									Completed	Completed	Completed
Test order matching algorithm									Completed	Completed	Completed
Test trade generation service									Completed	Completed	Completed
Test market database									Completed	Completed	Completed
Test trade validator									Completed	Completed	Completed
Test market data replay service									Completed	Completed	Completed
Write the monthly reports					Completed	Completed	Completed	Completed			
Write the progress report								Completed	Completed		
Write the final report										Completed	Completed
Prepare for the presentation											Completed
Prepare for video trailer											Completed

 Completed

 Planned

## 6 Recommended Hardware & Software

### Hardware:

Processor:	Intel i5 or higher
Minimum Display Resolution:	1024 * 768 with 16-bit color
Internal Memory:	512 GB, 80 GB free
RAM:	8 GB

### Software

Operating System:	Windows 10, LINUX 7, Mac OS
Database:	MYSQL
Front end:	Java, HTML
Programming Language:	Java 11

## 7 References

- [1] I. Hwang. “A Brief History of the Stock Market and Stock Exchanges”. Available: <https://www.sofi.com/learn/content/history-of-the-stock-market/>
- [2] FIX Global Technical Committee. Introduction. Available: [https://www.fixtrading.org/online-specification/introduction/#:~:text=The%20Financial%20Information%20Exchange%20\(FIX%C2%AE\)%20Protocol%20is%20a%20message,partners%20wishing%20to%20automate%20communications.](https://www.fixtrading.org/online-specification/introduction/#:~:text=The%20Financial%20Information%20Exchange%20(FIX%C2%AE)%20Protocol%20is%20a%20message,partners%20wishing%20to%20automate%20communications.)
- [3] Wikipedia. Financial Information eXchange. Available: [https://en.wikipedia.org/wiki/Financial\\_Information\\_eXchange#:~:text=The%20Financial%20Information%20eXchange%20\(FIX,to%20securities%20transactions%20and%20markets.](https://en.wikipedia.org/wiki/Financial_Information_eXchange#:~:text=The%20Financial%20Information%20eXchange%20(FIX,to%20securities%20transactions%20and%20markets.)
- [4] A. Johnson. “3 Best Stock Market Simulators of 2022”. Available: <https://www.nasdaq.com/articles/3-best-stock-market-simulators-of-2022>

## 8 Appendix A: Glossary

- Buy Side - refers to institutional investors, such as pension funds, mutual funds, insurance companies, and hedge funds that buy securities for investment purposes. These entities make investment decisions on behalf of their clients or themselves, aiming to generate returns on the capital invested.
- Derivatives - are financial instruments whose value is derived from an underlying asset or group of assets. They can be used for hedging against risks, speculating on price movements, or gaining exposure to assets without owning them directly. Common types of derivatives include futures, options, swaps, and forwards.
- ECNs - or Electronic Communication Networks, are automated systems that match buy and sell orders for securities in financial markets. They allow traders to directly access liquidity and execute trades without the need for a traditional broker. ECNs facilitate faster and more transparent trading by displaying real-time order book information and enabling order matching between various market participants.
- EMS - Execution Management System is a software platform used by institutional traders to manage the execution of their trades across multiple brokers and trading venues. They provide functionalities such as order creation, routing, execution monitoring, and post-trade analysis, helping traders optimize trade execution and achieve best execution practices, and often integrate with market data feeds and trading algorithms to enhance decision-making and streamline the trading process.
- FIX Protocol - is a standardized messaging format used for the electronic communication of trade-related information in the financial industry. It enables the transmission of orders, executions, and other trading-related messages between different parties, such as buy-side institutions, sell-side firms, and trading venues. FIX Protocol helps streamline and automate trading processes, improving efficiency, reducing errors, and increasing the speed of trade execution in financial markets.
- FOK- "Fill or Kill," which is a type of time-in-force order used in trading. When a trader submits a Fill or Kill order, it instructs the broker to either execute the entire order immediately at the specified price or cancel the order entirely. If the order cannot be filled immediately and completely, it is cancelled instead of being partially executed. FOK orders are often used when traders want to ensure that their orders are either fully executed at the desired price or not executed at all.
- OMS – Order Management System is a software platform used by investment managers and institutional traders to manage and control the lifecycle of orders for financial securities. They facilitate order entry, routing, execution, and allocation across various asset classes and trading venues and help streamline trading operations, improve efficiency, and provide comprehensive order tracking and reporting capabilities to optimize the investment process.

- Paper Trading- is a practice where individual simulate the process of buying and selling securities without actually investing real money, users generally use virtual accounts with funds to place trades based on similar to real market conditions.
- Sell Side - refers to the segment of the financial industry that involves entities such as investment banks, brokers, and market makers that facilitate trading and investment. These entities typically provide services such as market research, underwriting, trading, and advisory services to institutional and retail clients.
- Tick size - refers to the minimum price movement or increment by which the price of a financial instrument. It is set by the exchange or regulatory body and can vary based on the price of the security. For example, a tick size of \$0.01 means that the price can change in increments of one cent. It plays a crucial role in determining the liquidity and price dynamics of a market.



## 9 Appendix B: Literature Survey

In our research for the project, we delved into the necessary technologies and existing simulation platforms and identified the key insights and niche for our ideas.

### The Role of FIX Protocol in Trading

The Financial Information Exchange (FIX®) Protocol is a message standard developed to facilitate the electronic exchange of information related to securities' transactions and has become the industry standard for the electronic communication of trade-related information [2]. Originally developed to standardize messaging between institutional clients and broker-dealers, the FIX protocol now supports a broad range of communication between financial entities. The flexibility and wide adoption of FIX make it indispensable in electronic trading, reducing redundancy and ensuring smooth communication across various asset classes such as equities, derivatives, and bonds.

FIX is utilized by buy-side institutions, sell-side brokers, stock exchanges, and Electronic Communication Networks (ECNs) to ensure accurate and timely trade execution [3]. As financial markets continue to evolve, FIX remains a critical component of modern trading infrastructures by facilitating fast, reliable, and secure trade communication.

### Stock Market Simulators

Stock simulators are platforms that mimic real-time stock market conditions, allowing users to place trades with virtual money. These platforms serve as educational tools that teach investors about stock trading, risk management, and market analysis without the risk of financial loss [4].

We studied three leading stock market simulators, analysed their advantages and shortcomings and identified gaps that our project addresses.

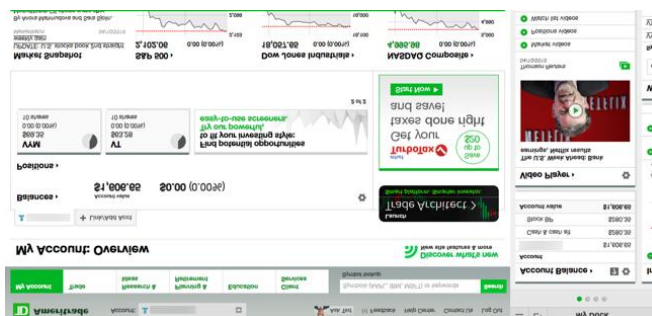
- **paperMoney by TD Ameritrade:** This simulator stands out for providing a robust learning environment, allowing users to explore complex investment strategies like

## RO4 FYP – Stock Exchange Simulator

futures and forex. It is tied to a real brokerage platform, making it ideal for those intending to use TD Ameritrade for real-world trading.

**Pros:** Free, extensive toolset for advanced trading strategies, chart indicators, and studies.

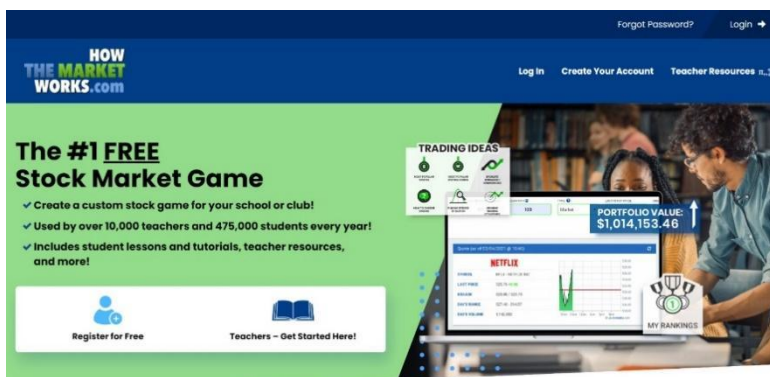
**Cons:** Lacks a dynamic user interface and no index funds trading.



- **HowTheMarketWorks:** This platform offers paper trading with a focus on education. It provides users with \$100,000 in virtual money to trade real-time in a fun, engaging game-like environment. It is ideal for beginners and features educational resources covering mutual funds, ETFs, and commodity futures.

**Pros:** Excellent for novices, with a library of educational content and contests with friends.

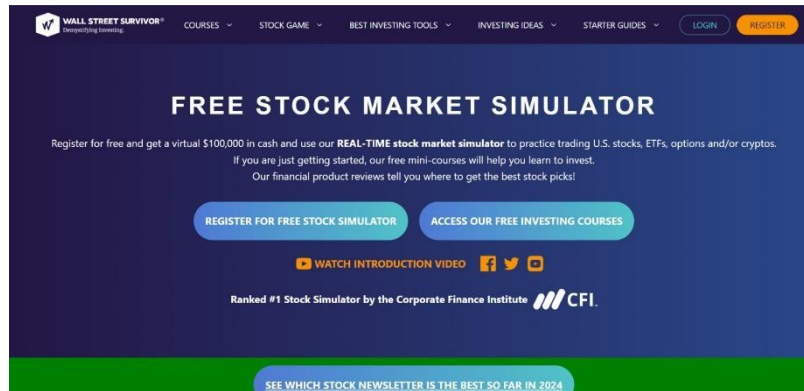
**Cons:** Not connected to real brokerage accounts and limited to paper trading.



- **WallStreetSurvivor:** Known for its longevity and game-like structure, this simulator allows users to buy and sell virtual investments while earning prizes like e-books and real cash as they improve.

**Pros:** Gamified learning with achievement badges and prizes; useful for learning personal finance.

**Cons:** Users should keep in mind that the Wall Street Survivor is not linked to a real brokering firm, unlike some other platforms such as paperMoney [4].



While these platforms focus on the range of options available for traders to practice their skills and improve their understanding of the stock market, the Stock Exchange Simulator closely replicates real stock exchange behaviour, including order matching, regulatory rules, and market sessions across various exchanges across the APAC region. Additionally, it integrates the FIX Protocol for seamless connectivity a feature absent in the others, as they focus on retail trading. The project also supports multi-market operations, includes algorithmic trading and advanced order types like FOK (Fill or Kill), and offers Market Data Replay for back testing using real historical data from sources like Bloomberg and Reuters.

# 10 Appendix C: Meeting Minutes

## 10.1 Minutes of the 1<sup>st</sup> Project Meeting

Date: July 2, 2024

Time: 12:00 pm

Place: Zoom

Present: Akshat Agarwal, Sukruti Rai, Prof. David Rossiter

Absent: None

### Approval of minutes:

- The minutes of the last meeting were approved without amendment.

### Report on progress:

- All team members have read the instructions of the Final Year Project online and have done research for the topic.

### Discussion items:

- Prof. Rossiter expressed his opinion about the project mainly being positive and explained a bit more about his expectations.
- Prof. Rossiter showed us a few examples of reports and suggested a timeline to work to ensure on time submissions.
- He also explained his concern about how it is hard to understand the value this project generates and suggested that we should also focus on building a story to make sure these concerns could be addressed.

### Goals for the coming month:

- Make sure we are able to demonstrate the value of the project before locking.
- Conduct research to start working on the proposal report.

## 10.2 Minutes of the 2<sup>nd</sup> Project Meeting

Date: September 9, 2024

Time: 1:30 pm

Place: Zoom

Present: Akshat Agarwal, Sukruti Rai, Prof. David Rossiter

Absent: None

### Approval of minutes:

- The minutes of the last meeting were approved without amendment

### Report on progress:

- All team members have prepared a draft of proposal report.

### Discussion items:

- Professor Rossiter shared his feedback on the report.
- He was satisfied with the content of the report and had some concerns about our goals.
- He provided suggestions on formatting to improve the readability of the report.
- He shared the experience of groups from the past to help us avoid common mistakes.

### Goals for the coming month:

- Incorporate the necessary changes based on professor's feedback.
- Complete and submit the report before the deadline and start working on the coding part of the project in coming weeks.

## 10.3 Minutes of the 3<sup>rd</sup> Project Meeting

Date: October 22, 2024

Time: 08:30 pm

Place: Zoom

Present: Akshat Agarwal, Sukruti Rai, Prof. David Rossiter

Absent: None

### Approval of minutes:

- The minutes of the last meeting were approved without amendment

### Report on progress:

- All team members have started working on initial development of the project.

### Discussion items:

- Group members demonstrated the initial developments.
- Group members shared the challenges faced- difficulty in understating and implementing exchange specific rules.
- Group members received feedback from supervisors.

### Goals for the coming month:

- Simulating the Hong Kong Exchange Behaviour for order matching.
- Enhancing the matching criteria and expanding support for different market-specific rules.
- Developing the core engine user interface for seeing the order book.

## **10.4 Minutes of the 4<sup>th</sup> Project Meeting**

Date: November 29, 2024

Time: 6:00 pm

Place: Zoom

Present: Akshat Agarwal, Sukruti Rai, Prof. David Rossiter

Absent: None

### Approval of minutes:

- The minutes of the last meeting were approved without amendment

### Report on progress:

- All team members have been developing the core backend functionalities of the system

### Discussion items:

- Group members demonstrated the developments – order book, matching engine and thread based matching.
- Group members shared the challenges faced- difficulty in managing the complexity of matching engine.
- Group members received feedback from supervisors.

### Goals for the coming month:

- Develop and implement FIX protocol
- Enhancing the matching engine to support various exchanges and asset types.

## 10.5 Minutes of the 5<sup>th</sup> Project Meeting

Date: February 13, 2025

Time: 10:30 am

Place: Zoom

Present: Akshat Agarwal, Sukruti Rai, Prof. Wentao Xie

Absent: None

### Approval of minutes:

- The minutes of the last meeting were approved without amendment

### Report on progress:

- All team members have worked on the backend systems and progress report

### Discussion items:

- Final testing of backend functions
- User Interface for the system
- Division of figma designs

### Goals for the coming month:

- Complete the figma designs and begin the front end coding

## 10.6 Minutes of the 6<sup>th</sup> Project Meeting

Date: February 27, 2025

Time: 10:30 am

Place: Starbucks HKUST

Present: Akshat Agarwal, Sukruti Rai

Absent: None

### Approval of minutes:

- The minutes of the last meeting were approved without amendment

### Report on progress:

- All team members have worked on the backend systems and testing.

### Discussion items:

- Testing of the backend system
- Areas to be fixed and issues to be addressed

### Goals for the coming month:

- Complete the testing of the backend
- Complete the front end design

## 10.7 Minutes of the 7<sup>th</sup> Project Meeting

Date: March 17, 2025

Time: 1:30 pm

Place: Learning Commons HKUST

Present: Akshat Agarwal, Sukruti Rai

Absent: None

### Approval of minutes:

- The minutes of the last meeting were approved without amendment

### Report on progress:

- All team members have worked on the backend systems and testing.

### Discussion items:

- Prof Wentao and group members introduced themselves
- Team members introduced the topic to new supervisor and gave a brief about the progress
- Prof Wentao provided suggestions for progress report

### Goals for the coming month:

- Complete the submit the progress report before the deadline
- Prepare a demonstration by mid-March.

## 10.8 Minutes of the 8<sup>th</sup> Project Meeting

Date: April 4, 2025

Time: 08:30 pm

Place: Zoom

Present: Akshat Agarwal, Sukruti Rai

Absent: None

### Approval of minutes:

- The minutes of the last meeting were approved without amendment

### Report on progress:

- All team members have worked on the backend and frontend systems

### Discussion items:

- Testing the backend system and finalizing the UI
- Finalizing the figma designs for the front end

### Goals for the coming month:

- Complete the integration of the system



## 10.9 Minutes of the 9<sup>th</sup> Project Meeting

Date: April 15, 2025

Time: 11:00 pm

Place: Zoom

Present: Akshat Agarwal, Sukruti Rai

Absent: None

### Approval of minutes:

- The minutes of the last meeting were approved without amendment

### Report on progress:

- All team members have worked on the backend and frontend systems

### Discussion items:

- Reviewing the finished backend and front end design
- Discussing final modification to the UI
- Planning the integration of the system

### Goals for the coming month:

- Complete the submit the final report before the deadline
- Completion of the code before the deadline

## **10.10 Minutes of the 10<sup>th</sup> Project Meeting**

Date: April 17, 2025

Time: 05:00 pm

Place: Zoom

Present: Akshat Agarwal, Sukruti Rai, Prof. Wentao Xie

Absent: None

### Approval of minutes:

- The minutes of the last meeting were approved without amendment

### Report on progress:

- All team members have worked on the project and final report

### Discussion items:

- Prof Wentao has gone through an initial draft of the report
- Team members discussed their concerns for material to be included in the report
- Prof Wentao provided suggestions for final report

### Goals for the coming month:

- Complete the submit the final report before the deadline
- Prepare the final demonstration by next week