ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA TOÁN - CƠ - TIN HỌC
---oooOoooo---

BÁO CÁO NHÓM 1

MÔN CÔNG NGHỆ PHẦN MỀM

Chủ đề: Mô hình hệ thống chứng khoán

Giảng viên : PGS.TS Bùi Việt Hà

Thành viên trong nhóm : Đặng Hữu Tấn - 20001972
Hoàng Đình Trung - 20001986
Phan Văn Tuấn - 20001987
Nguyễn Minh Vũ - 20001993

Hà Nội, 10 - 2023

MỤC LỤC

Lời nói đầu

Trong thế giới ngày càng phức tạp của công nghiệp tài chính, việc tích hợp và tự động hóa quy trình giao dịch trở nên quan trọng hơn bao giờ hết. Hệ thống chứng khoán không ngoại lệ cũng là một phần quan trọng của nền kinh tế toàn cầu. Để đáp ứng nhu cầu ngày càng tăng của thị trường, cần có một hệ thống mạnh mẽ, linh hoạt và có khả năng mở rộng. Đồng thời, sự tích hợp giữa các hệ thống cũng trở nên cần thiết, đòi hỏi việc áp dụng các mô hình và phương pháp tiên tiến. Mô hình hệ thống EIP (Enterprise Integration Patterns) chính là một trong những phương pháp đã được chứng minh hiệu quả trong việc giải quyết vấn đề này.

Báo cáo này nhằm mục tiêu giới thiệu và mô tả cách thức xây dựng mô hình hệ thống chứng khoán dựa trên mô hình hệ thống EIP. Bằng cách áp dụng EIP, chúng ta có thể nắm bắt, mô tả và giải quyết các vấn đề phức tạp trong việc tích hợp hệ thống, đồng thời tối ưu hóa quy trình giao dịch, nâng cao hiệu suất và đảm bảo tính bảo mật.

Trong phạm vi bài báo cáo, chúng em xin trình bày về việc ***thiết kế một hệ thống chứng khoán đơn giản***. Bài báo cáo chúng em bao gồm các nội dung chính sau:

Phần 1: Giới thiệu chung

Phần 2: Tổng quan dự án

Phần 3: Yêu cầu chức năng (Functional requirements)

Phần 4: Yêu cầu phi chức năng (Non-Functional requirements)

Phần 5: Kịch bản sử dụng (Use cases)

Phần 6: Giao diện người dùng

Trong quá trình thực hiện đề tài không tránh khỏi những sai sót, nhóm chúng em rất mong nhận được sự đánh giá, góp ý của thầy cô để bài báo cáo của chúng em được hoàn thiện hơn.

Chúng em xin chân thành cảm ơn!

I. Giới thiệu chung

1 Mục đích

Tài liệu nhằm đem đến một cái nhìn tổng quát về hệ thống giao dịch chứng khoán **StockTKL**, bằng cách mô tả mục tiêu và phạm vi của dự án, các tính năng quan trọng, yêu cầu chức năng và phi chức năng, cũng như một cái nhìn tổng quan về cách hệ thống hoạt động. Nó giúp định rõ những gì dự án sẽ làm và cung cấp một cơ sở cho việc phát triển và triển khai hệ thống giao dịch chứng khoán.

Tài liệu này đóng vai trò quan trọng trong việc định hình dự án và đảm bảo rằng mọi người trong dự án hiểu rõ mục tiêu và phạm vi của nó, giúp tạo điểm xuất phát rõ ràng cho quá trình phát triển hệ thống giao dịch chứng khoán.

2 Phạm vi

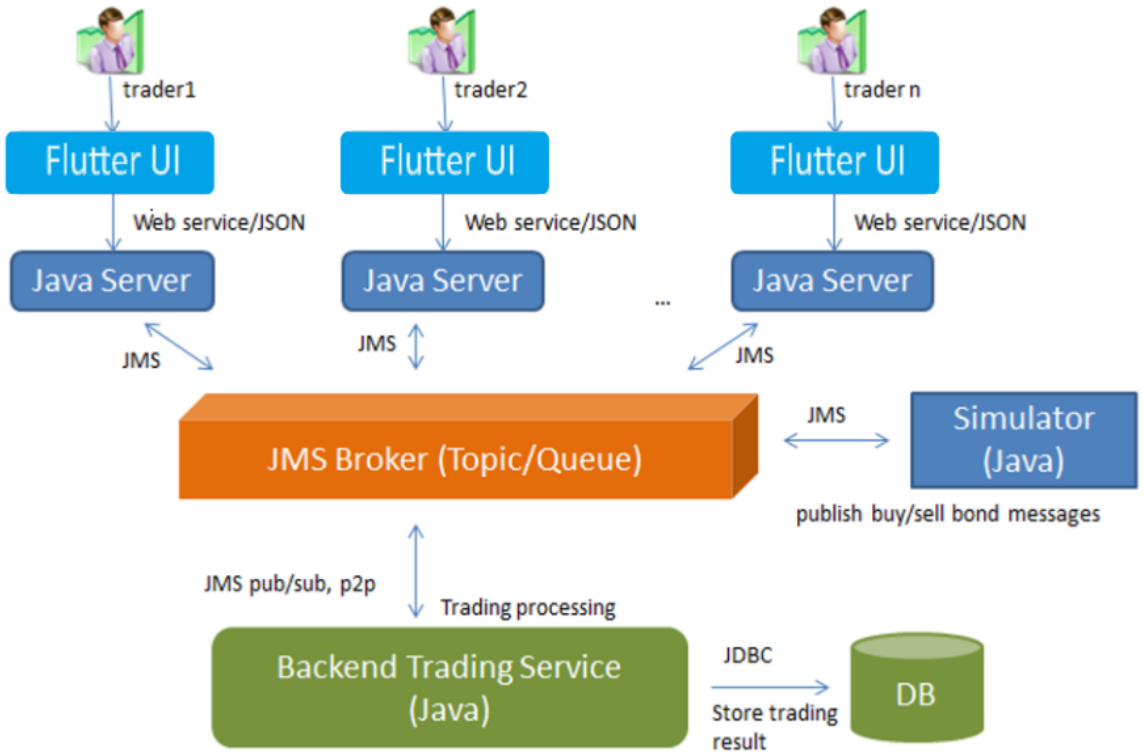
Hệ thống giao dịch chứng khoán StockTKL được thiết kế để phục vụ cho người dùng muốn thực hiện giao dịch chứng khoán một cách nhanh chóng, tiện lợi và minh bạch.

II. Tổng quan dự án

1. Tổng quan

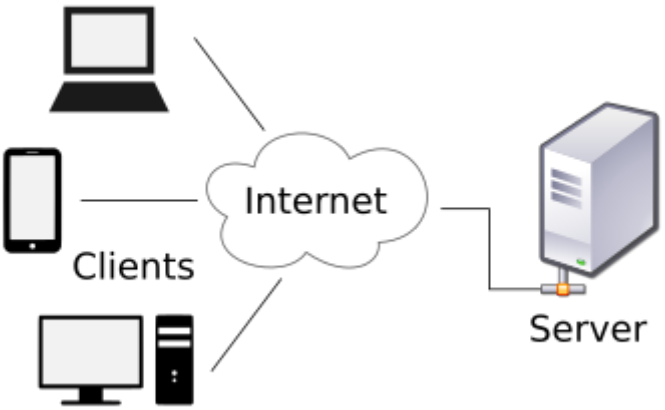
Hệ thống của nhóm em được thiết kế để mô phỏng quá trình giao dịch chứng khoán một cách đơn giản. Hệ thống chứng khoán là một mô hình khá phức tạp và tinh vi, do đó nhóm chúng em đã đơn giản hoá logic, quá trình này có thể không giống y hệt với quá trình giao dịch trong thực tế.

Hệ thống của chúng em chia thành 5 hệ thống con như hình mô tả dưới đây:



- Frontend Flutter UI System:
  + Gồm giao diện người dùng cho các nhà giao dịch.
  + Cung cấp khả năng tương tác trực tiếp với hệ thống, cho phép người dùng đặt lệnh giao dịch.
- Frontend Java Server:
  + Đóng vai trò như một lớp trung gian giữa giao diện người dùng và hệ thống backend.
  + Xử lý các yêu cầu từ UI Flutter và chuyển dữ liệu đến JMS Broker.
- Simulator (Java):
  + Mô phỏng quá trình giao dịch chứng khoán.
  + Phát đi các thông báo mua/bán cổ phiếu dựa trên yêu cầu từ người dùng.
- Backend Trading Service (Java):
  + Xử lý logic giao dịch thực sự, đảm bảo tất cả các giao dịch đều tuân theo quy định.
  + Lưu kết quả giao dịch vào cơ sở dữ liệu thông qua JDBC.
- Messaging Channel (JMS Broker - nhóm sử dụng: RabbitMQ):
  + Phục vụ việc truyền thông giữa các hệ thống con bằng cách sử dụng mô hình publish/subscribe và point-to-point.
  + Đảm bảo dữ liệu được truyền một cách hiệu quả và đáng tin cậy giữa các thành phần của hệ thống.

2. Mô hình Client/Server



Máy chủ cung cấp các API (Application Programming Interface) để cho phép các máy khách kết nối và thực hiện giao dịch, truy vấn dữ liệu và kiểm tra tình trạng tài khoản. Nó có tính bảo mật cao để đảm bảo an toàn của dữ liệu và giao dịch của khách hàng.

Máy khách là các ứng dụng hoặc phần mềm mà các nhà đầu tư sử dụng để truy cập và thực hiện các giao dịch trên thị trường chứng khoán. Các máy khách có thể là ứng dụng máy tính, trình duyệt web, hoặc ứng dụng di động. Người dùng có thể thực hiện các giao dịch mua bán cổ phiếu, truy vấn thông tin về cổ phiếu, kiểm tra tài khoản và xem các biểu đồ và dữ liệu thị trường.

Luồng cơ bản:

Người dùng mở ứng dụng máy khách và đăng nhập vào tài khoản của họ. Máy khách gửi yêu cầu giao dịch hoặc truy vấn đến máy chủ thông qua API. Máy chủ xử lý yêu cầu, kiểm tra thông tin và tài khoản của người dùng, và thực hiện giao dịch nếu hợp lệ. Kết quả của giao dịch được gửi từ máy chủ về máy khách, và người dùng nhận thông báo kết quả.

3. Công cụ, công nghệ sử dụng

3.1. Công nghệ sử dụng

3.1.1 Về phía Backend:

a. Java

- Ưu điểm:

+ Tính đáng tin cậy và ổn định: Java nổi tiếng về tính đáng tin cậy và ổn định của ứng dụng. Điều này quan trọng trong hệ thống chứng khoán để đảm bảo giao dịch được thực hiện một cách đáng tin cậy.

+ Hiệu suất cao: Java có khả năng xử lý hiệu suất cao và đa nhiệm, điều quan trọng trong việc xử lý hàng triệu lệnh giao dịch hàng ngày.
+ Bảo mật: Java có các thư viện và framework mạnh mẽ để thực hiện bảo mật trong ứng dụng. Điều này quan trọng để đảm bảo an toàn giao dịch và bảo mật dữ liệu.
+ Cộng đồng và tài liệu lớn: Java có một cộng đồng lớn và tài liệu phong phú, giúp giải quyết vấn đề và tìm hiểu cách sử dụng một cách dễ dàng.
+ Quản lý giao dịch: Java Transaction API (JTA) có thể được sử dụng để quản lý giao dịch trong hệ thống chứng khoán, đảm bảo tính toàn vẹn và đồng nhất của dữ liệu giao dịch.
+ Hệ thống phân phối và mở rộng: Java có các công cụ và framework cho phân phối và khả năng mở rộng. Điều này giúp xây dựng hệ thống chứng khoán phân phối và có khả năng mở rộng để đối phó với tải cao.
+ Xử lý thời gian thực: Hệ thống chứng khoán phải xử lý dữ liệu trong thời gian thực. Java hỗ trợ xử lý real-time thông qua framework như Spring boot,...

- Nhược điểm:

+ Phát triển mất thời gian: Phát triển bằng Java có thể tốn nhiều thời gian hơn so với việc sử dụng các ngôn ngữ khác, đặc biệt là trong việc xây dựng và cấu hình.
+ Khả năng mở rộng phức tạp: Việc mở rộng hệ thống Java có thể phức tạp hơn so với một số ngôn ngữ khác, và yêu cầu kiến thức chuyên sâu về quản lý tài nguyên và hiệu suất.

b. Java Spring Boot

- Framework Java phổ biến cho phát triển hệ thống.

- Ưu điểm:

+ Phát triển nhanh chóng: Spring Boot giúp giảm thiểu công đoạn cấu hình và tập trung vào việc phát triển ứng dụng. Điều này giúp tiết kiệm thời gian và giúp phát triển ứng dụng nhanh hơn.
+ Dependency Injection (DI): Spring Boot sử dụng DI để quản lý các phụ thuộc giữa các thành phần của ứng dụng. Điều này giúp tạo ra mã dễ đọc và dễ bảo trì.
+ Spring Security: Spring Boot tích hợp Spring Security, cho phép dễ dàng thực hiện xác thực và ủy quyền trong hệ thống chứng khoán, giúp đảm bảo tính bảo mật của ứng dụng.

+ Tích hợp dễ dàng: Spring Boot có nhiều tích hợp sẵn với các công nghệ và framework khác, giúp kết nối dễ dàng với cơ sở dữ liệu, giao tiếp với dịch vụ bên ngoài, và xây dựng API RESTful.
+ Quản lý cấu hình dễ dàng: Spring Boot cung cấp các cơ chế để quản lý cấu hình ứng dụng, bao gồm cấu hình bằng file properties hoặc YAML.
+ Tài liệu và cộng đồng lớn: Spring Boot có một cộng đồng lớn và tài liệu phong phú, giúp giải quyết vấn đề và tìm hiểu cách sử dụng.

- Nhược điểm:

+ Overhead: Spring Boot có thể có một chút overhead do việc sử dụng DI và các tính năng tự động cấu hình. Điều này có thể ảnh hưởng đến hiệu suất trong các tình huống đòi hỏi hiệu suất tối ưu cực đại.
+ Quá nhiều tính năng: Spring Boot đi kèm với nhiều tính năng và thư viện, điều này có thể làm cho ứng dụng trở nên phức tạp hơn cần thiết. Việc chỉnh sửa và cấu hình có thể đôi khi làm cho ứng dụng trở nên khó bảo trì.
+ Kích thước ứng dụng lớn: Do tích hợp nhiều tính năng và thư viện, ứng dụng Spring Boot có thể có kích thước lớn hơn so với các ứng dụng được xây dựng bằng các framework nhẹ hơn.
+ Không phù hợp cho ứng dụng nhỏ: Đối với các ứng dụng nhỏ, việc sử dụng Spring Boot có thể làm cho ứng dụng quá phức tạp và tốn thời gian phát triển.

c. Caching

- Để lưu trữ dữ liệu trung gian và giảm thời gian truy vấn cơ sở dữ liệu.

- Ưu điểm:

+ Tăng hiệu suất: Caching giúp cải thiện hiệu suất bằng cách lưu trữ dữ liệu trung gian ở các vị trí gần hơn và nhanh chóng truy cập. Điều này giúp giảm thời gian truy cập cơ sở dữ liệu, đặc biệt là khi xử lý giao dịch giao dịch nhanh.
+ Giảm tải cho cơ sở dữ liệu: Caching giúp giảm tải cho cơ sở dữ liệu bằng cách giảm số lần truy cập dữ liệu cơ sở dữ liệu thực tế. Điều này đảm bảo cơ sở dữ liệu hoạt động hiệu quả hơn và không bị quá tải.
+ Tích hợp hệ thống phân phối: Caching có thể được sử dụng để lưu trữ dữ liệu tạm thời trên các máy chủ phân phối, giúp cung cấp dữ liệu cho các nút trong mạng nhanh hơn và giảm tải cho cơ sở dữ liệu.

+ Tính nhất quán và tính toàn vẹn dữ liệu: Caching có thể được cấu hình để cung cấp tính nhất quán và tính toàn vẹn dữ liệu, đảm bảo rằng dữ liệu được lưu trữ và cập nhật một cách đúng đắn.

- Nhược điểm:

+ Bộ nhớ tốn kém: Caching có thể yêu cầu bộ nhớ tốn kém, đặc biệt là khi lưu trữ lượng lớn dữ liệu trung gian. Điều này có thể ảnh hưởng đến tài nguyên và khả năng mở rộng của hệ thống.
+ Hạn chế cho dữ liệu thay đổi thường xuyên: Caching thường phù hợp cho dữ liệu ít thay đổi hoặc không thay đổi trong một khoảng thời gian ngắn. Đối với dữ liệu thay đổi thường xuyên, cần quản lý cơ chế hết hạn của cache.
+ Cơ hội mất dữ liệu: Trong trường hợp lưu trữ cache trên bộ nhớ đệm tạm thời, có nguy cơ mất dữ liệu nếu xảy ra sự cố hoặc sự cố về bộ nhớ.

d. Java Transaction API (JTA)

- Dùng để quản lý và kiểm soát các giao dịch trong các ứng dụng.

- Ưu điểm:

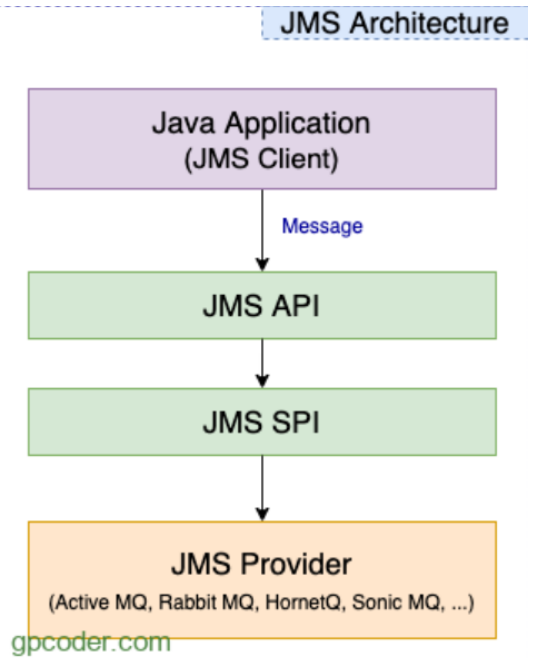
+ Tính toàn vẹn giao dịch: JTA đảm bảo tính toàn vẹn của giao dịch, nghĩa là nếu có lỗi xảy ra trong quá trình thực hiện giao dịch, thì toàn bộ giao dịch sẽ được hủy bỏ và không ảnh hưởng đến cơ sở dữ liệu. Điều này đặc biệt quan trọng trong hệ thống chứng khoán để đảm bảo tính nhất quán của dữ liệu giao dịch.
+ Tính nhất quán dữ liệu: JTA đảm bảo rằng dữ liệu trong cơ sở dữ liệu luôn ở trạng thái nhất quán sau mỗi giao dịch. Điều này đảm bảo rằng dữ liệu giao dịch không bị thiếu hoặc không chính xác.
+ Cách li giao dịch (Isolation): JTA hỗ trợ cách ly giao dịch, đảm bảo rằng một giao dịch không ảnh hưởng đến các giao dịch khác đang diễn ra đồng thời. Điều này đặc biệt quan trọng trong hệ thống chứng khoán để tránh xung đột dữ liệu.
+ Xử lý giao dịch phân phối: JTA cho phép xử lý giao dịch phân phối, tức là có thể thực hiện giao dịch trên nhiều hệ thống và cơ sở dữ liệu khác nhau. Điều này quan trọng khi cần tích hợp nhiều nguồn dữ liệu trong hệ thống chứng khoán.

- Nhược điểm:

+ Phức tạp: Việc sử dụng JTA có thể làm cho mã nguồn phức tạp hơn, và cần sử dụng các tùy chọn và cấu hình phức tạp để quản lý giao dịch.
+ Hiệu suất: Giao dịch có thể tạo ra một overhead nhất định và làm giảm hiệu suất của hệ thống. Điều này đặc biệt quan trọng trong các hệ thống chứng khoán nơi hiệu suất là một yếu tố quan trọng.
+ Khó triển khai: Triển khai JTA có thể phức tạp và đòi hỏi hiểu biết chuyên sâu về quản lý giao dịch.
+ Yêu cầu tài nguyên máy chủ: Quản lý giao dịch có thể đòi hỏi tài nguyên máy chủ cao, điều này có thể ảnh hưởng đến chi phí và khả năng mở rộng của hệ thống.
+ Thời gian phát triển: Việc triển khai JTA và quản lý giao dịch có thể tốn thời gian trong quá trình phát triển ứng dụng.

e. Java Message Service (JMS)

- JMS API là một phần của đặc tả kỹ thuật Java Enterprise Edition (Java EE), là một API trung gian hướng thông báo Java (MOM) để gửi tin nhắn giữa hai hoặc nhiều client.
- JMS mô tả các phương thức tạo bởi chương trình Java cho việc: tạo (Create), gửi (Send), nhận (Receive), đọc (Read) tin nhắn.
- JMS cho phép giao tiếp giữa các thành phần khác nhau của một ứng dụng phân tán được kết nối lỏng lẻo, đáng tin cậy và hỗ trợ bất đồng bộ.
- Giao tiếp giữa các client được tạo ra bởi **message broker** qua các tiêu chuẩn truyền tin bất đồng bộ như AMQP, MQTT.
- JMS bao gồm 2 thành phần:
  + API : hỗ trợ chức năng cho người phát triển phần mềm.
  + SPI (Service Provider Interface) : cho phép các Provider tạo ra tool JMS tích hợp, định hướng cho mọi người sử dụng theo hướng chuẩn hóa.



Message Queue :

- Là ứng dụng trung gian hỗ trợ gửi và nhận tin nhắn,đảm bảo toàn vẹn dữ liệu, transaction và cân bằng cho việc gửi nhận dữ liệu.

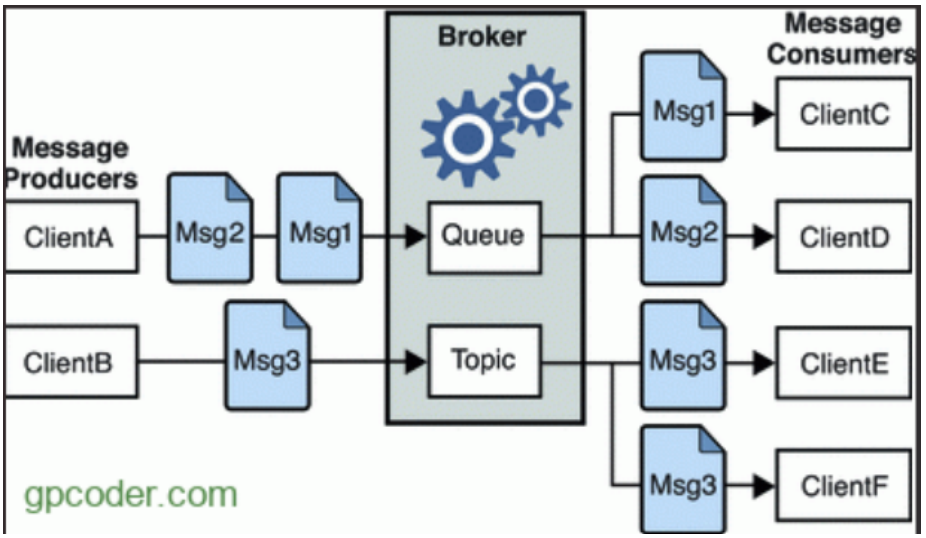
Một số MQ tiêu biểu :

JMS PROVIDER	ORGANIZATION
WebSphere MQ	IBM
Weblogic Messaging	Oracle Corporation
Rabbit MQ	Rabbit Technologies(acquired by Spring Source)
Active MQ	Apache Foundation
HornetQ	JBoss
Sonic MQ	Progress Software
TIBCO EMS	TIBCO
OpenMQ	Oracle Corporation
SonicMQ	Aurea Software

JMS Broker :

- Để gửi hoặc nhận tin nhắn, JMS Client phải kết nối với JMS message server (còn gọi là JMS Broker):

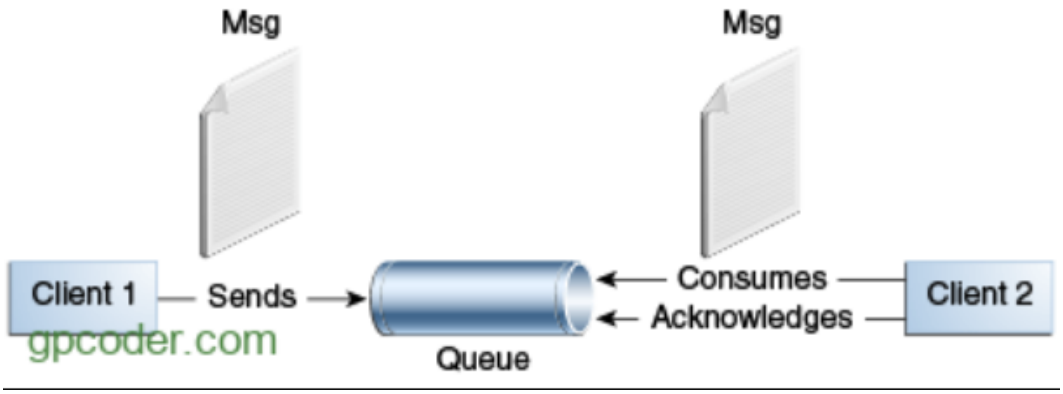
+ Một Connection mở ra một kênh liên lạc giữa client và broker.
+ Tiếp theo, client phải thiết lập một session để tạo, tạo và đọc tin nhắn. Bạn có thể nghĩ về session như một luồng thông báo xác định cho một cuộc trò chuyện cụ thể giữa client và broker. Bản thân client là producer hoặc consumer.
+ Transaction được tạo bởi client là transaction giữa producer và broker hoặc giữa broker và consumer, nhưng không bao giờ giữa producer và consumer.
+ Producer gửi tin nhắn đến destination (đích) mà broker quản lý. Consumer truy cập destination đó để lấy tin nhắn.
+ Tin nhắn bao gồm header (tiêu đề), optional properties (thuộc tính), và body (nội dung). Body chứa dữ liệu, header chứa thông tin broker cần định tuyến và quản lý tin nhắn và các thuộc tính có thể được xác định bởi các ứng dụng client hoặc bởi provider để phục vụ nhu cầu của chính họ trong việc xử lý tin nhắn.
+ Connection, Session, Destination, Message, Producer và Consumer là những đối tượng cơ bản tạo nên một ứng dụng JMS.



Các mô hình JMS

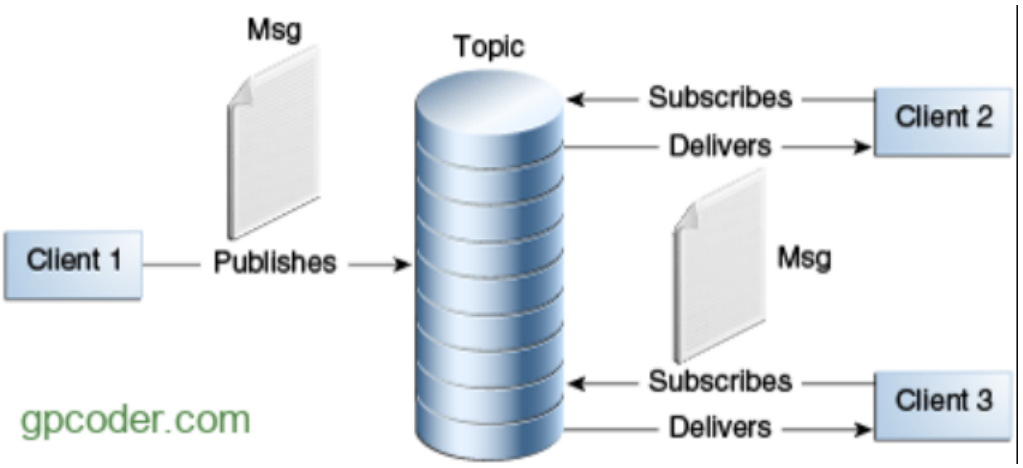
- JMS hỗ trợ 2 mô hình trao đổi dữ liệu là P2P (Point to Point) và Pub/Sub (Publisher/ Subscriber). Các mô hình trao đổi này gọi là JMS message domain.

+ **P2P (Point to Point) :**



- Có 3 thành phần chính là sender, queue và receiver
  + Sử dụng queue là nơi lưu trữ các message. Queue giữ lại các tin nhắn cho đến khi Client nhận hoặc bị timeout
  + Cho phép người nhận không cần active tại thời điểm gửi message
  + Sử dụng tín hiệu acknowledges để kích hoạt cho việc nhận message ở người nhận hoàn thành.

+ **Pub/Sub (Publisher/ Subscriber) :**



- Với Pub/Sub cũng có 3 thành phần chính là sender, Topic, và receiver :
    + Topic thực chất là 1 queue lớn, chứa nhiều queue con có mức độ ưu tiên khác nhau.
    + Cho phép 1 người gửi và nhiều người nhận
    + Topic sẽ lưu lại toàn bộ message mà không bị mất đi cho đến khi MOM được reset, hay xóa.
    + Mỗi Subscriber sẽ chỉ nhận được message từ topic sau khi đã subscription.
    + Không cần sử dụng thông tin acknowledge và message được chuyển đến các subscriber chỉ là bản copy.
    + Mô hình bảo mật không cao do gửi nhiều người nhận nhưng được ưu điểm đó là áp dụng thuận lợi cho hệ thống phân tán.
    + Mô hình Pub/Sub đòi hỏi đối tượng nhận phải active hay listener tại thời điểm gửi nhận message, nếu không message sẽ mất đi (non-durable). Do vậy, để message vẫn không mất đi nhưng được lưu trữ tạm thời trong MOM và cho phép người dùng truy cập vào MOM với tính xác thực thì chức năng durable phải được kích hoạt.

Cơ chế gửi nhận message trong JMS được chia thành 2 phần :

- Đồng bộ (Synchronous) : đối tượng nhận và gửi message thông qua thực hiện phương thức receive và phương thức này giữ ứng dụng chờ đợi cho đến khi message tới.
- Bất đồng bộ (Asynchronous) : đối tượng nhận bắt buộc phải đăng ký cơ chế lắng nghe MessageListener để đón nhận message và phương thức được kích hoạt là onMessage để đón nhận và xử lý message..

- Ưu điểm của JMS:

- Reliability (Độ tin cậy) : Tin nhắn trung gian (Messaging middleware) đảm bảo tin nhắn được phân phối đến người nhận (receiver). Nếu người nhận ngừng hoạt động vì một số lý do, hệ thống nhắn tin sẽ lưu trữ tin nhắn cho đến khi nó hoạt động trở lại. Đảm bảo một message chỉ được gửi một lần, tránh trường hợp bị mất message hoặc trùng lặp.
- Asynchronous (Bất đồng bộ) : JMS Provider, client có thể gửi và nhận message không đồng bộ. Nghĩa là người gửi và người nhận không cần chờ nhau.

- Ease of Integration (dễ tích hợp) : Nhiều ứng dụng có thể được viết bằng các ngôn ngữ lập trình khác nhau hoặc chạy trên các máy chủ khác nhau. Miễn là chúng sử dụng cùng một giao thức, một phần mềm trung gian nhắn tin có thể được sử dụng để thiết lập liên lạc giữa chúng.
- Scalability (Khả năng mở rộng): Hệ thống nhắn tin giúp quảng bá tin nhắn đến nhiều receiver, giúp mở rộng.
- Flexibility (Tính linh hoạt) : có thể gán các giao thức khác nhau cho các tin nhắn khác nhau, cho phép ta linh hoạt trong việc thiết kế kiến trúc của mình.
- Loosely coupled (Kết nối lỏng lẻo) :
    + JMS API là đặc tả kỹ thuật nên được triển khai bởi tất cả các JMS Provider. Vì vậy chúng ta có thể thay đổi Provider hiện tại sang Provider mới với ít thay đổi (có nghĩa là chỉ cấu hình) hoặc không thay đổi mã ứng dụng JMS.
    + Tách biệt người gửi và người nhận, và người truyền tin trung gian sẽ lưu tin nhắn đến khi người nhận có thể xử lý.
    + Interoperability (Khả năng tương tác) : JMS API cho phép khả năng tương tác giữa các ngôn ngữ nền tảng Java khác như Scala và Groovy.

- Nhược điểm JMS:

JMS sẽ không bao gồm các tính năng sau, bởi vì JMS chỉ là hệ thống gửi nhận tin:

- Load Balancing/Fault Tolerance
- Error/Advisory Notification
- Administration
- Security
- Wire Protocol
- Message Type Repository

Vì có nhiều hạn chế của JMS nên trong các hệ thống distributed sẽ sử dụng các libs được implement dựa trên JMS Kafka, Zookeeper,...

3.1.2 Về phía Frontend:

a. Flutter: framework phát triển ứng dụng mã nguồn mở của Google, cho phép phát triển ứng dụng di động đa nền tảng từ một mã nguồn duy nhất, giúp tiết kiệm thời gian và công sức cho nhà phát triển.

Ưu điểm: cho phép phát triển ứng dụng nhanh chóng và hiệu quả trên cả iOS và Android.

Nhược điểm: cần thời gian làm quen với Dart

b. Dart: ngôn ngữ được sử dụng để phát triển logic ứng dụng trong Flutter.

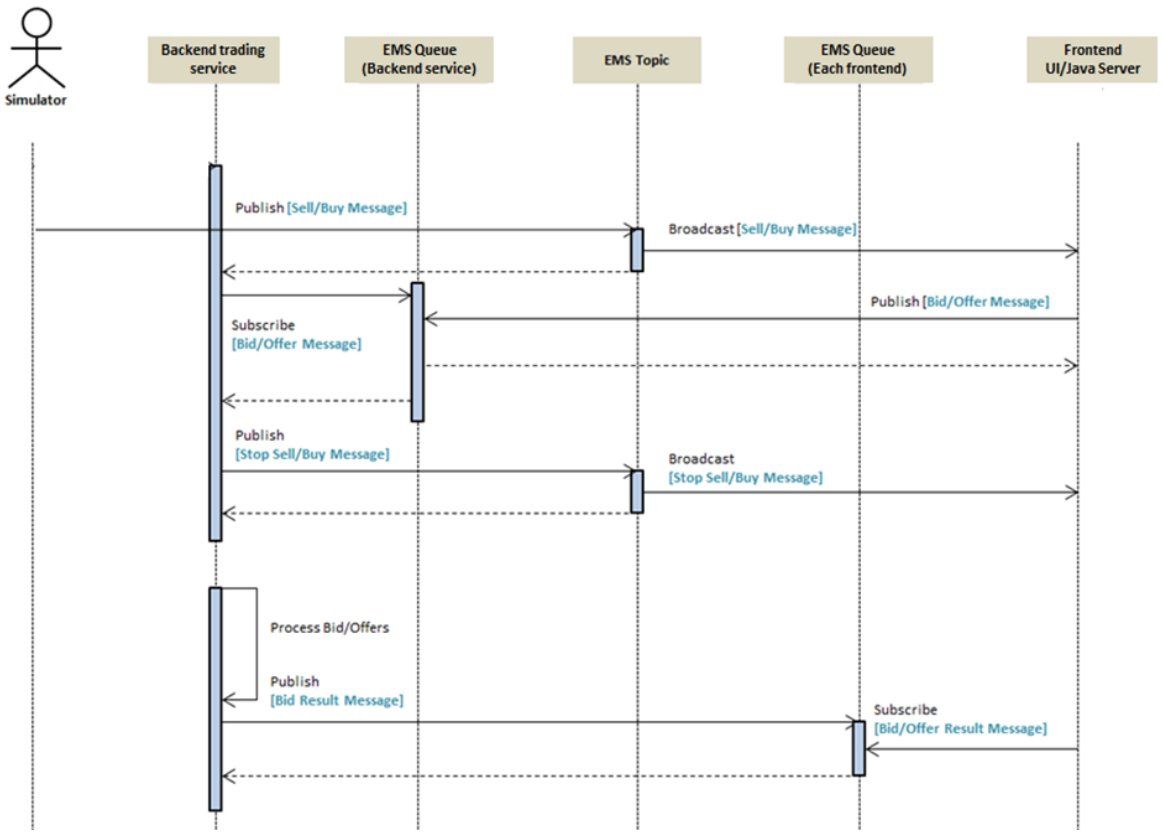
Ưu điểm: cú pháp dễ đọc, hiệu suất cao, hỗ trợ nhiều tính năng hiện đại của ngôn ngữ lập trình.

Nhược điểm: khả năng hỗ trợ thư viện có hạn so với một số ngôn ngữ lập trình khác.

3.2. Công cụ sử dụng

- **Visual Studio Code**: trình biên dịch và mã nguồn mở được sử dụng để phát triển mã nguồn mở Dart, Java.
- **Git, Github**: sử dụng để quản lý mã nguồn dự án, tạo điều kiện cho việc làm đồng thời và cộng tác giữa nhiều người.
- **MySQL** : là một hệ quản trị cơ sở dữ liệu, được sử dụng để lưu trữ và quản lý dữ liệu trong các ứng dụng.
- **Docker**: nền tảng ứng dụng mã nguồn mở giúp đóng gói và triển khai các ứng dụng vào các môi trường độc lập.

4. Trading Workflow



a. Các thành phần của biểu đồ:

- **Simulator:** giả lập việc tạo và gửi lệnh từ người dùng
- **Backend trading service:** xử lý lệnh từ Simulator, kiểm tra và khớp lệnh
- **EMS Queue (Backend service):** lưu trữ và quản lý thông tin giao dịch đã khớp từ Backend trading service
- **EMS Topic:** kênh truyền dẫn thông tin giữa Simulator và Backend trading service
- **EMS Queue (Each frontend):** lưu trữ và truyền thông tin giao dịch đã khớp đến Frontend UI/Java Server
- **Frontend UI/Java Server:** hiển thị thông tin giao dịch, cung cấp giao diện trực quan cho người dùng

b. Chuỗi tương tác:

- Đặt lệnh giao dịch:
  + ***Simulator*** tạo và gửi lệnh đặt giao dịch đến ***Backend trading service*** thông qua ***EMS Topic***.
  + ***Backend trading service*** xử lý và kiểm tra lệnh giao dịch.

+ ***Backend trading service*** sau khi khớp lệnh, gửi thông tin giao dịch đã khớp đến ***EMS Queue (Backend)***.
+ ***EMS Queue (Backend)*** lưu trữ thông tin và chuyển thông tin đến ***EMS Queue (Frontend)***.
+ ***EMS Queue (Frontend)*** truyền thông tin về ***Frontend UI/Java Server*** để hiển thị cho người dùng.

- Xem thông tin giao dịch:
+ ***Frontend UI/Java Server*** yêu cầu lấy thông tin giao dịch từ ***EMS Queue (Frontend)***.
+ ***EMS Queue (Frontend)*** gửi thông tin giao dịch đã khớp về ***Frontend UI/Java Server***.
+ Người dùng xem thông tin giao dịch trên giao diện của ***Frontend UI/Java Server***.

III. Yêu cầu chức năng (Functional requirements)

1. Đăng nhập

- Hệ thống hỗ trợ xác thực người dùng thông qua tên đăng nhập và mật khẩu
- Người dùng cần có khả năng thay đổi mật khẩu
- Hệ thống cần ghi lịch sử đăng nhập và các hoạt động liên quan đến tài khoản

2. Mua cổ phiếu

- Người dùng phải có khả năng tìm kiếm theo tên cổ phiếu hoặc ký hiệu mã chứng khoán
- Người dùng phải có khả năng xem thông tin chi tiết giá cổ phiếu bao gồm giá, biểu đồ thay đổi giá, và tin tức liên quan
- Người dùng phải có khả năng mua cổ phiếu với số lượng và giá cụ thể
- Hệ thống phải xử lý lệnh mua và báo lại khi lệnh được thực hiện thành công

3. Bán cổ phiếu

- Người dùng phải có khả năng xem danh sách cổ phiếu đang sở hữu
- Người dùng phải có khả năng bán cổ phiếu với số lượng và giá cụ thể
- Hệ thống phải xử lý lệnh bán và báo lại khi lệnh được thực hiện thành công

IV. Yêu cầu phi chức năng (Non-Functional requirements)

1. Hiệu năng

- Hệ thống phải có khả năng xử lý nhiều giao dịch cùng lúc và đảm bảo thời gian phản hồi nhanh chóng.
- Hệ thống phải có khả năng theo dõi và ghi nhật ký theo dõi hiệu suất hệ thống.

2. Bảo mật

- Tất cả các thông tin giao dịch, thông tin tài khoản phải được bảo mật bằng cách sử dụng các biện pháp mã hoá dữ liệu và xác thực 2 yếu tố.
- Hệ thống phải có khả năng phát hiện và ngăn chặn các hành vi không hợp lệ hoặc tấn công từ bên ngoài.

3. Ổn định hệ thống và xử lý sự cố

Trong quá trình hoạt động hệ thống chứng khoán, sự ổn định của hệ thống đóng vai trò quan trọng, giúp đảm bảo tính liên tục và hiệu quả trong việc cung cấp dịch vụ cho các nhà đầu tư. Một "Major Production Crash" có thể dẫn đến việc ngừng giao dịch, gây ra mất niềm tin từ phía nhà đầu tư và ảnh hưởng đến sự ổn định của thị trường.

Nguyên nhân: Việc crash xảy ra do MQSeries gặp sự cố, kéo theo nhiều thành phần khác. Điều này được xác định do hàng đợi Dead Letter trong MQSeries.

Giải pháp: Cần một mô hình giúp giảm tốc độ dòng tin nhắn, giúp ứng dụng thêm thời gian xử lý. Ta triển khai mô hình Message Dispatcher để quản lý chuyển tiếp tin nhắn đến các Performer dựa trên tình trạng của chúng. Mỗi Performer sẽ xử lý một tin nhắn và sau đó trở lại trạng thái không bận, sẵn sàng cho tin nhắn tiếp theo.

V. Kịch bản sử dụng (Use cases)

1. Use Case

Use Case 1: Hệ thống tự động khớp lệnh giao dịch

- *Mô tả:* Hệ thống tự động khớp lệnh mua và lệnh bán dựa trên các điều kiện đã được đặt trong lệnh mua và lệnh bán.
- *Hành động:*
  + Hệ thống nhận lệnh mua và lệnh bán từ người dùng hoặc bên ngoài.
  + Hệ thống kiểm tra sự hợp lệ của lệnh. (vd: đủ số lượng chứng khoán hoặc đủ số dư tiền)

+ Lệnh được đưa vào hàng đợi chờ khớp.
+ Hệ thống tìm kiếm lệnh phù hợp để khớp lệnh
+ Nếu có lệnh mua và lệnh bán phù hợp, hệ thống tự động khớp lệnh bằng cách cập nhật các giao dịch tương ứng.
  (Các điều kiện khớp lệnh cụ thể được trình bày bên dưới)
+ Hệ thống thông báo cho người dùng kết quả của giao dịch khớp lệnh.

Use Case 2: Xem danh sách các lệnh đang chờ khớp lệnh

- *Mô tả:* Người dùng có thể xem danh sách các lệnh mua và lệnh bán đang chờ để được khớp lệnh.
- *Hành động:*
  + Người dùng truy cập vào hệ thống chứng khoán.
  + Người dùng chọn xem danh sách các lệnh mua và lệnh bán đang chờ khớp lệnh.
  + Hệ thống hiển thị danh sách các lệnh đang chờ khớp lệnh, bao gồm các thông tin như mã cổ phiếu, giá, số lượng, …
  + Người dùng có thể lọc và sắp xếp danh sách theo tiêu chí mong muốn.

Use Case 3: Đặt lệnh mua

- *Mô tả:* Người dùng có thể đặt lệnh mua cho một số lượng cố định của chứng khoán ở một mức giá. Hệ thống sẽ tự động kiểm tra khớp lệnh với các lệnh bán có điều kiện tương tự.
- *Hành động:*
  + Người dùng truy cập vào hệ thống chứng khoán.
  + Người dùng chọn tạo lệnh mua và nhập các thông tin, bao gồm mã cổ phiếu, giá, số lượng, …
  + Hệ thống tiếp nhận lệnh mua từ người dùng, kiểm tra tính hợp lệ của lệnh.
  + Hệ thống kiểm tra khớp lệnh với các lệnh bán đang chờ khớp lệnh.
  + Nếu có khớp lệnh, hệ thống thông báo kết quả và cập nhật giao dịch mới.
  + Người dùng có thể huỷ hoặc chỉnh sửa lệnh trước khi nó được khớp.

Use Case 4: Đặt lệnh bán

- *Mô tả:* Người dùng có thể đặt lệnh bán cho một số lượng cố định của chứng khoán ở giá được chỉ định (giá này không được sai khác 10% so với giá tham chiếu). Hệ thống sẽ tự động kiểm tra khớp lệnh với các lệnh mua có điều kiện tương tự.

- *Hành động:*
    + Người dùng truy cập vào hệ thống chứng khoán.
    + Người dùng chọn tạo lệnh bán và nhập các thông tin, bao gồm mã cổ phiếu, giá, số lượng, …
    + Hệ thống tiếp nhận lệnh bán từ người dùng.
    + Hệ thống kiểm tra khớp lệnh với các lệnh mua đang chờ khớp lệnh.
    + Nếu có khớp lệnh, hệ thống thông báo kết quả và cập nhật giao dịch mới.
    + Người dùng có thể huỷ hoặc chỉnh sửa lệnh trước khi nó được khớp.

Use Case 5: Xem lịch sử giao dịch

- *Mô tả:* Người dùng có thể xem lịch sử các giao dịch đã được thực hiện bởi hệ thống, bao gồm cả các giao dịch đã khớp lệnh và các giao dịch chưa khớp lệnh.
- *Hành động:*
    + Người dùng truy cập vào hệ thống chứng khoán.
    + Người dùng chọn xem lịch sử giao dịch.
    + Hệ thống hiển thị danh sách các giao dịch đã thực hiện, bao gồm thông tin như mã cổ phiếu, giá, số lượng, loại giao dịch, …
    + Hệ thống cung cấp chức năng xuất thông tin giao dịch ra định dạng file (vd: excel, PDF) để người dùng dễ dàng lưu trữ và phân tích.

Use Case 6: Xem thống kê giao dịch của mình

- *Mô tả:* Người dùng có thể xem thống kê tổng quan về các giao dịch mà họ đã thực hiện, bao gồm các thông tin như tổng giá trị giao dịch, số lượng giao dịch, lợi nhuận, …
- *Hành động:*
    + Người dùng truy cập vào hệ thống chứng khoán.
    + Người dùng chọn xem thống kê giao dịch của mình.
    + Hệ thống hiển thị thống kê tổng quan về các giao dịch của người dùng, bao gồm tổng giá trị giao dịch, số lượng giao dịch, lợi nhuận, …

Use Case 7: Xử lý khi khớp lệnh không thành công

- *Mô tả:* Trong trường hợp lệnh không được khớp thành công và giữ lại quá lâu trong hệ thống, hệ thống sẽ có các biện pháp xử lý phù hợp.
(*Các điều kiện khớp lệnh không thành công sẽ được trình bày bên dưới*)
- *Hành động:*
    + Hệ thống kiểm tra danh sách lệnh chưa khớp sau mỗi phiên giao dịch.

+ Đối với các lệnh chưa được khớp trong một ngày, hệ thống sẽ tự động hủy lệnh.
+ Hệ thống gửi thông báo cho người dùng về việc huỷ lệnh và lý do.
+ Người dùng có quyền tái đặt lệnh hoặc điều chỉnh thông tin lệnh để tăng khả năng khớp lệnh trong lần đặt tiếp theo.

2. Điều kiện khớp lệnh

Hệ thống giao dịch thực hiện so khớp các lệnh mua và lệnh bán chứng khoán theo nguyên tắc ưu tiên về giá và thời gian, cụ thể như sau:

- *Ưu tiên về giá (Price Priority):* Điều quan trọng nhất trong nguyên tắc khớp lệnh là ưu tiên về giá. Theo nguyên tắc này, các lệnh được khớp dựa trên mức giá, với các lệnh mua đặt giá cao hơn sẽ được ưu tiên khớp với các lệnh bán đặt giá thấp hơn và ngược lại. Tuy nhiên hai giá được khớp nhau chỉ dao động trong khoảng 1%.

- *Ưu tiên về thời gian (Time Priority):* Trong trường hợp hai lệnh có cùng mức giá, thì nguyên tắc ưu tiên về thời gian sẽ được áp dụng. Lệnh nào được đặt trước thì sẽ được khớp trước. Điều này khẳng định rằng thời gian đặt lệnh là yếu tố quyết định khi giá bằng nhau.

- *Ưu tiên về khối lượng (Volume Priority):* Trong trường hợp cả mức giá và thời gian của lệnh mua hoặc lệnh bán đều giống nhau thì lệnh nào có khối lượng giao dịch nhiều hơn sẽ được ưu tiên thực hiện trước.

3. Quy luật update giá cổ phiếu dựa trên cung và cầu

Do sự phức tạp trong việc biến động giá cổ phiếu hàng ngày phụ thuộc vào nhiều yếu tố từ kinh tế, chính trị, quy luật cung - cầu, báo cáo tài chính doanh nghiệp, truyền thông, … Nhóm chúng em quyết định xây dựng một quy luật đơn giản hơn để demo việc cập nhật giá cổ phiếu dựa trên cung và cầu. Dưới đây là cách thức xây dựng quy luật của nhóm:

a. Khởi tạo:
- Giá cổ phiếu ban đầu: **P_initial**
- Số lượng cổ phiếu được mua: **Q_bought**
- Số lượng cổ phiếu được bán: **Q_sold**
- Khoảng thời gian xem xét: **T** (ngày)

b. Xác định số lượng cổ phiếu chênh lệch:

Q_difference = Q_bought - Q_sold

c. Tính phần trăm thay đổi giá

- Nếu **Q_difference** > 0 (nghĩa là có nhiều người mua hơn số người bán):

  Phần trăm thay đổi giá = (Q_difference / T) * K

- Nếu **Q_difference** < 0 (nghĩa là có nhiều người bán hơn số người mua):

  Phần trăm thay đổi giá = (Q_difference / T) * L

- Nếu **Q_difference** = 0 (nghĩa là có sức bán bằng sức mua):

  Phần trăm thay đổi giá = không thay đổi

trong đó: **K** và **L** là hệ số quyết định, có thể tự đặt giá trị cho **K**, **L** để điều chỉnh độ nhạy dựa trên thực nghiệm hoặc thông tin từ thị trường

d. Cập nhật giá cổ phiếu

P_update = P_initial + P_initial * (Phần trăm thay đổi giá)

Để dễ hình dung, ta lấy một ví dụ như sau. Giả sử ta có cổ phiếu của công ty A:

- P_initial = $100
- Q_bought = 1500 cổ phiếu
- Q_sold = 1000 cổ phiếu
- T = 1 ngày

=> Q_difference = Q_bought - Q_sold = 1500 - 1000 = 500

Do Q_difference > 0; giả sử K = 0.02:

Phần trăm thay đổi giá = 500 / 1 * 0.02 = 10%

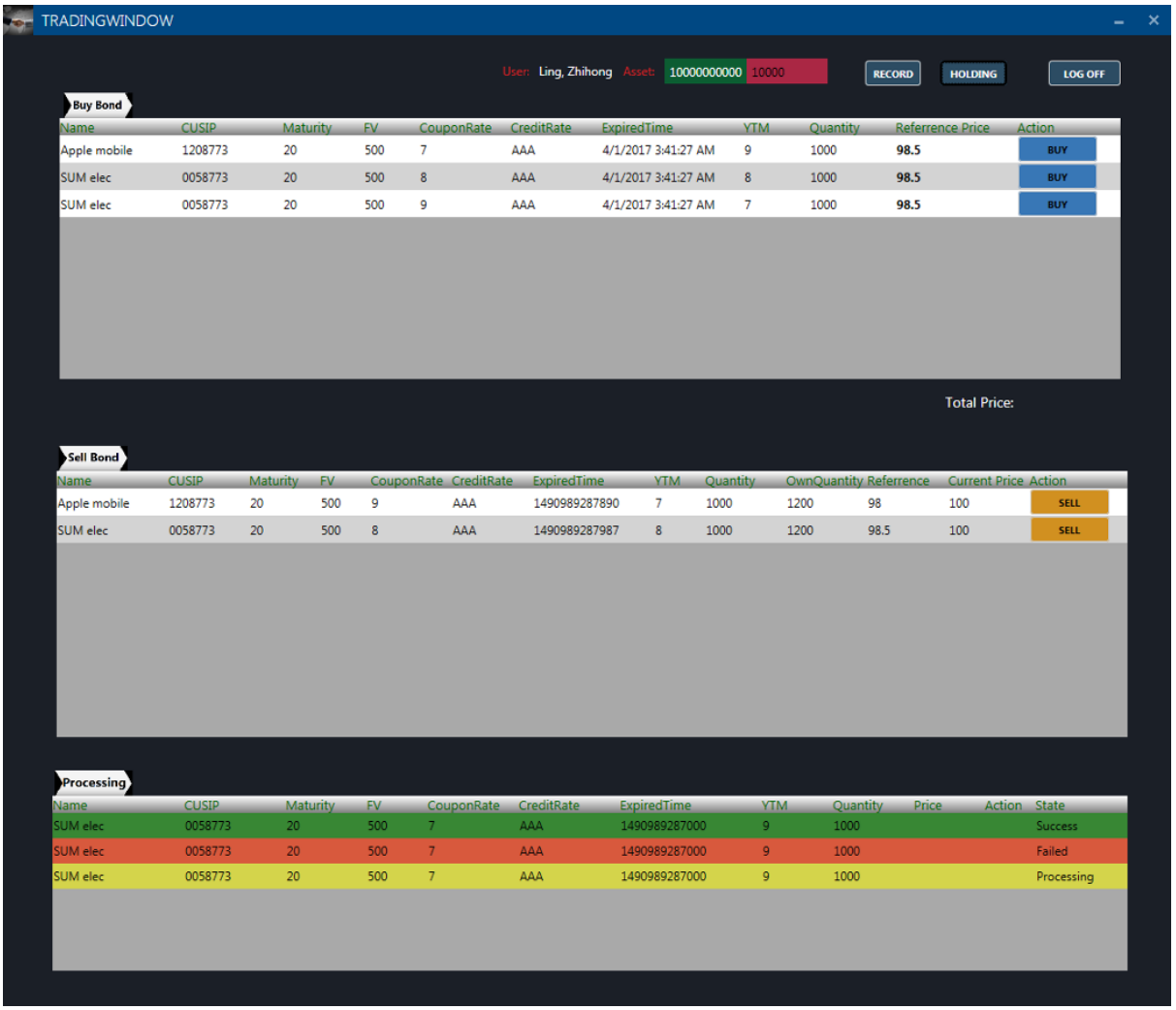
Do đó, P_update = 100 + 100 * 10% = $110

Như vậy, chỉ sau 1 ngày, dựa theo quy luật này, giá cổ phiếu công ty A tăng từ $100 lên $110.

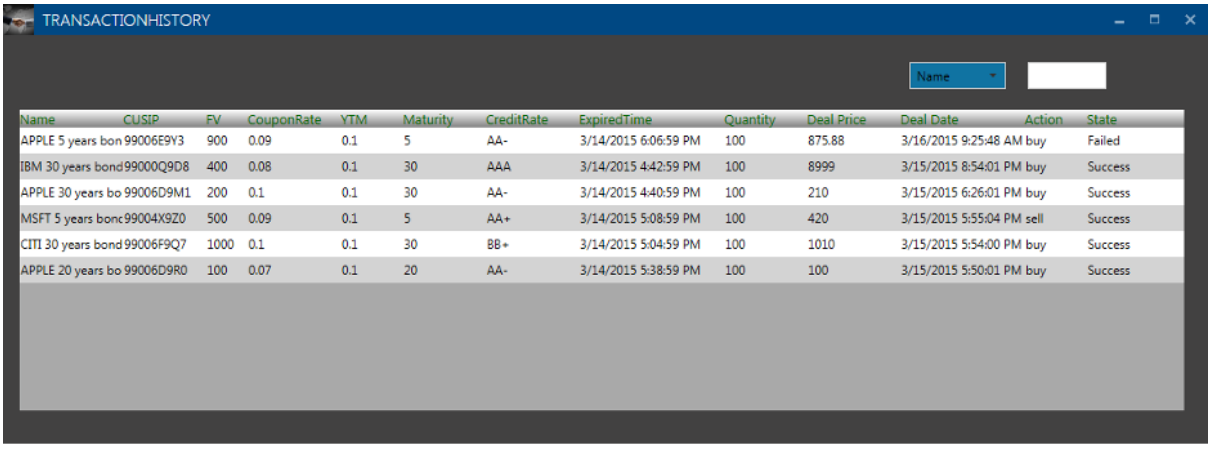
VI. Giao diện người dùng

Dưới đây là một số mô phỏng về các giao diện chính mà nhóm chúng em dự định làm:

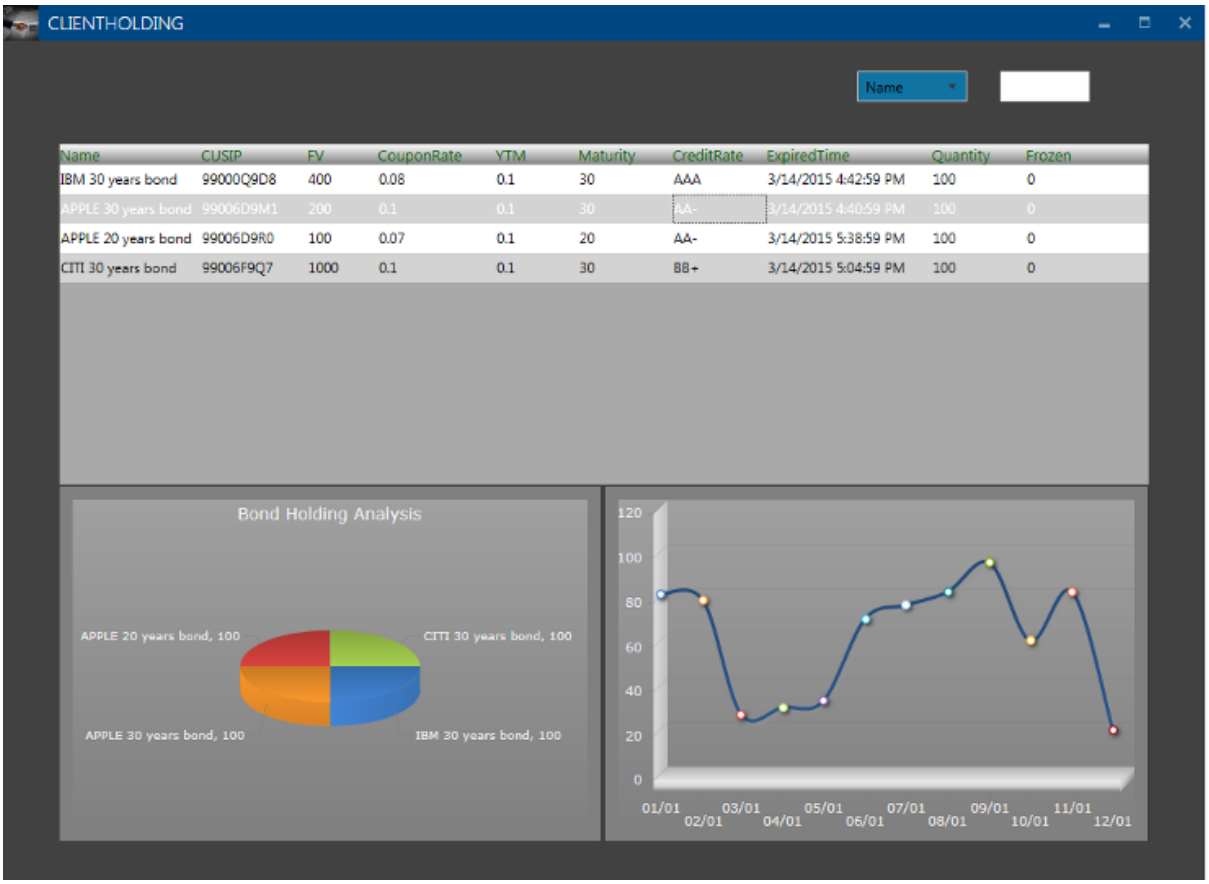
- Màn hình các quá trình mua/ bán được diễn ra:



- Màn hình lịch sử giao dịch:



- Màn hình thống kê giao dịch:



Kết luận

Trong quá trình thiết kế hệ thống chứng khoán, chúng em đã cân nhắc việc tối ưu hoá và đơn giản hoá một số quy trình để phù hợp với phạm vi và thời gian của dự án. Tuy nhiên, chúng em đã đảm bảo rằng những tính năng quan trọng và cốt lõi của một hệ thống chứng khoán thực thụ vẫn được thể hiện và phát triển một cách đúng đắn. Thông qua việc này, chúng em mong muốn phản ánh một cách chân thực bản chất và những hoạt động cơ bản của thị trường chứng khoán.

Chúng em hy vọng rằng sản phẩm cuối cùng sẽ đáp ứng được kỳ vọng của thầy cô và các bạn cùng học. Chúng em rất mong nhận được những góp ý, phản hồi và đánh giá từ mọi người để dự án của chúng em có thể được hoàn thiện hơn trong tương lai.

Tài liệu tham khảo

[1]https://topdev.vn/blog/gioi-thieu-jms-java-message-services/
[2]https://www.oracle.com/technical-resources/articles/java/intro-java-message-service.html
[3]https://docs.spring.io/spring-framework/docs/3.1.x/spring-framework-reference/html/jms.html
[4]https://docs.flutter.dev/
[5]https://dart.dev/guides
[6]https://www.progress.com/tutorials/jdbc/understanding-jta
[7]https://thuvienphapluat.vn/phap-luat/thoi-su-phap-luat/nguyen-tac-khop-lenh-trong-giao-dich-chung-khoan-niem-yet-la-gi-quy-dinh-ve-viec-sua-huy-lenh-giao--236045-50180.html
[8]https://www.ssi.com.vn/tin-tuc/tin-tuc-chung/cac-thuat-ngu-chung-khoan-co-ban-cho-nha-dau-tu-f0