CREDIT SUISSE

MODULE 2: GLOBAL TECHNICAL ANALYST PROGRAM

CASE STUDY

ROITRAINING
MAXIMIZE YOUR TRAINING INVESTMENT

This page intentionally left blank.

Table of Contents

Overview

This case study focuses on building a robust trading system component utilizing cutting edge technologies for processing big data. Test-driven development will be used in all software development. This will include both unit and integration testing. Processing of both archive data files and streaming data will be incorporated in the system.

Case Study Presentation

On day seven of this program, each case study group will make a presentation of their project. This 10- to 15-minute presentation should include a discussion of the technical aspects of the solution that the team designed and developed over the course of the training.

Each team member is responsible to showcase a work product and describe the role played in developing the solution. In particular, the design of the solution should be presented, as well as details on how that design was implemented. Any problems, challenges, difficulties, or "features" should be acknowledged and discussed. For those issues that were resolved, a discussion of the solution should be included. For any issues that are still unresolved, a discussion of what was attempted, and how the issue could be approached should be provided.

The presentation will be made in the classroom. Those in attendance will include the instructor, the other teams in the class, and additional invited guests.

After a team has made their presentation, there will be a period for any questions that the audience may have for the team, or for specific individuals, about the end deliverable or the role played to develop the deliverable. Expect to receive several interesting questions from the audience. In the unlikely event that no questions are asked, the instructor may assume the role of *Interrogator* and pose a few questions of his/her own.

The Instructor's Role

During the case study sessions, your instructor will play a variety of roles depending upon what the specific deliverable or activity calls for. During the case study periods, the instructor will wear many different hats.

One primary role is simply an extension of the traditional instructor role—that of being a **Technical Resource/Coach/Mentor**. When a team encounters problems in determining how to approach a particular problem, the instructor may facilitate discussions to reach a solution, or may play the role of **Product Owner** to provide input to requirements and expected deliverables.

The instructor will regularly check in with each group to discuss how the group is progressing and proactively deal with any issues they are facing.

Stock Trading Platform

Introduction

Trading systems are comprised of many loosely coupled services that facilitate trading activity via Electronic Communication Networks (ECNs). There are many different ECNs, and each will offer slightly different services to the traders, depending on the requirements of that market. For example, a foreign exchange market ECN will behave differently to a fixed income market. For this case study, we will focus on fixed income (e.g., bond) trading. Examples of fixed income ECNs include Bloomberg, Tradeweb, Euronext, and Eurex Bonds.

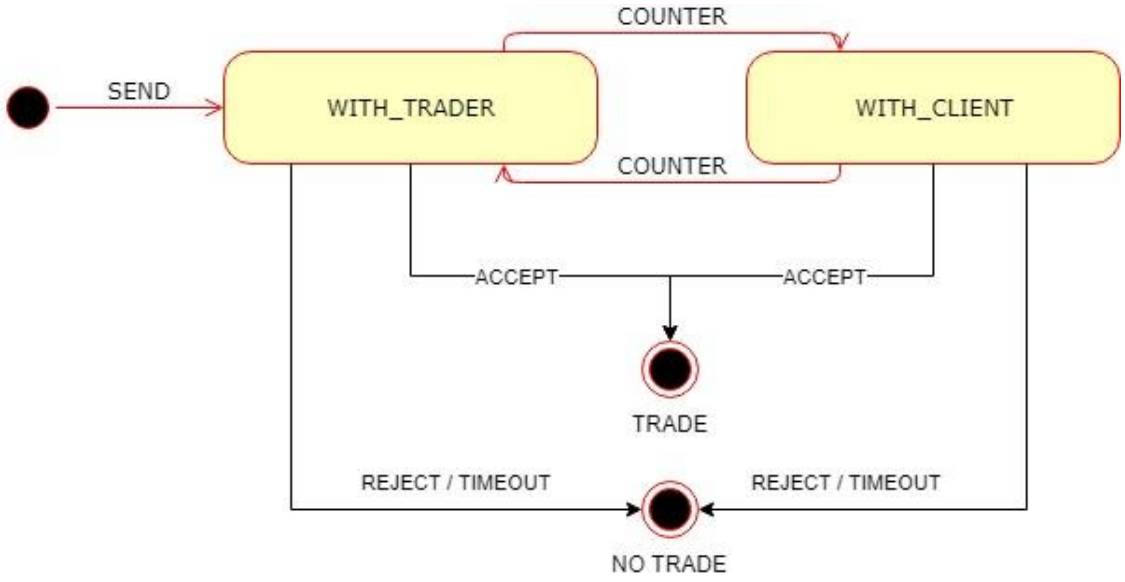
In a fixed income market, traders buy and sell bonds (generally referred to as instruments) on behalf of the bank or other institution they are working for. The institution they work for is responsible for settling the trade: paying for whatever the trader bought and collecting payment for whatever was sold. These institutions are legally bound by the terms of the market and are sometimes referred to as the "legal entity" for which the trader is working. Within a market, there will be a separate identity for each trader, and another for each legal entity they are trading for.

The purpose of this case study is to extend the existing functionality of a trading system within the bank, and to provide data insights to our traders to help them make more informed decisions. The trading system we will be extending is implemented internally as a suite of components which communicate by passing Kafka messages across the network to one another.

Request for Quote (RFQ)

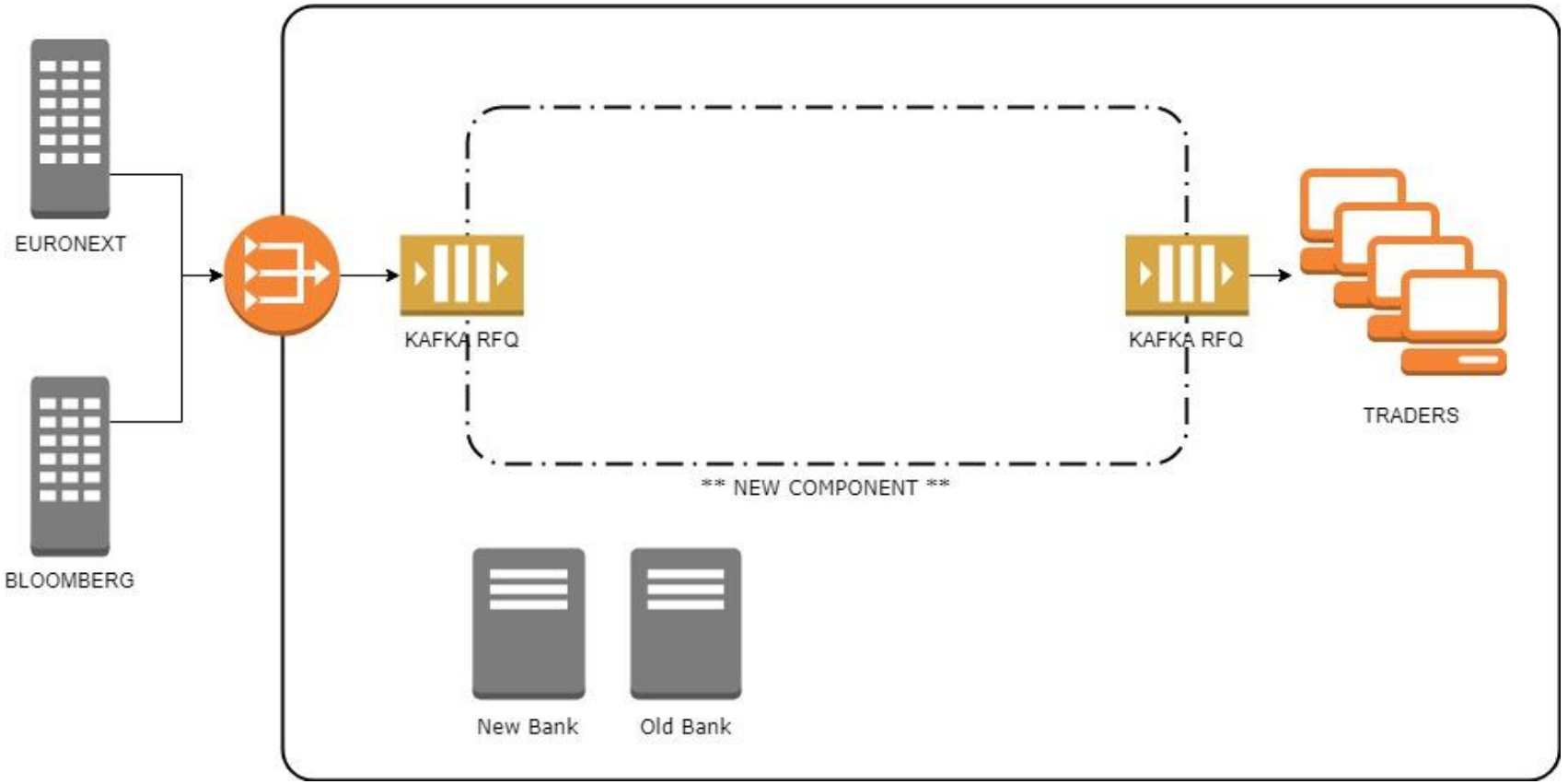
When a customer wants to trade, they send a request for quote message (RFQ) to our traders. The RFQ will indicate the instrument the customer is interested in, a trade side (whether they want to buy/sell it), a quantity they want to trade, and an opening price they are willing to trade at.

Upon receipt of an RFQ, the trader has a limited time in which to respond and either: accepting the price and trading, rejecting it outright, or sending a counter-offer back to the sender.



A simple finite state diagram for an RFQ negotiation

Below is a high-level context diagram of key stakeholders and systems involved in our fixed income trading system. The diagram shows two ECNs: Euronext and Bloomberg. Indicative prices are published for all the bonds the bank is actively trading to these markets (not shown). The diagram shows (in orange) the message flow from the ECN to the traders' desktop systems, via our proposed new component, when an RFQ is received.



For added complexity (and realism), we might also consider the case where the bank has recently merged with another bank: shown as new/old bank subsystems in the diagram.

The Project

Your task this week is to build a new component which 'decorates' any incoming RFQs with additional metadata that will provide data insights to help our traders make more informed decisions when responding to RFQs.

Important Note on Expectations

Completing all user stories is not a requirement. What is a requirement is that each story is delivered completed and with good quality code supported by relevant tests. It is better to have one solid story completed than say, three partially or not fully tested.

Implementation Requirements

The following are requirements for the system:

- The new component will be built using Apache Spark
- You are free to use whichever Spark modules you prefer
- UNIT tests are mandatory
- Try to implement stories which would support the epic, but ultimately, it's the team's decision whether to implement separate/discrete User Stories or target set of stories which contribute to a single epic. In any case, epic can be used for context setting and for better understanding of the end-to-end flow.

Dictionary of Terms

Term	Description
Trader	A trader is an individual who engages in the buying and selling of financial instrument in any financial market, either for himself or on behalf of another person or institution.
Instrument	A generic term for a financial commodity that can be traded on a financial market.
Bond	A bond is a fixed income instrument that represents a loan made by an investor to a borrower (typically corporate or governmental). A bond could be thought of as an I.O.U. between the lender and borrower that includes the details of the loan and its payments. A bond has an end date when the principal of the loan is due to be paid to the bond owner and usually includes the terms for variable or fixed interest payments that will be made by the borrower. Bonds are used by companies, municipalities, states, and sovereign governments to finance projects and operations. Owners of bonds are debtholders, or creditors, of the issuer.
ISIN	International Securities Identification Number (or ISIN) is a standard identifier for a financial commodity, used widely throughout various trading systems.
ECN	An electronic communication network (ECN) is a computerized system that automatically matches buy and sell orders for securities in the market. It connects major brokerages and individual traders so they can trade directly between themselves without going through a middleman and make it possible for investors in different geographic locations to quickly and easily trade with each other.
Legal entity	Any individual, corporation, association, or other organization that has, in the eyes of the law, the capacity to make a contract or an agreement, and the abilities to assume an obligation and to discharge indebtedness.
RFQ	An electronic message passing protocol used to negotiate a deal between two (or more) parties. If an agreement is NOT met using the protocol, then no trade is executed. If an agreement IS made, then a legally binding trade is executed, and the legal entities involved are committed to that trade.
Trade	A contract to either buy or sell a given quantity of some financial instrument at an agreed price.
Liquidity	A measure of whether an instrument is actively trading on a market. If no one can agree on a trade, then the instrument is said to be *illiquid*. Certain instruments, such as government bonds, generally trade well and are said to be *liquid*.

Reference Data and Examples

There are many types of negotiation messages used in our trading system. They are based upon the open Financial Information eXchange (FIX) messaging standard[1]. For any queries relating to the messages, a good place to check is the reference section on the ONIXS website[2]. Between banks, the messages are transmitted using an optimal binary format to maximize speed. Within the bank, and for the purpose of this project, our on-the-wire transport will be over TCP/IP[3]. All messages will be formatted using JSON object literal syntax[4]. It is worth noting that Google provide a simple but powerful Java library that you might find useful for working with JSON[5].

Your component should generate a single outgoing message for each RFQ using JSON object literal format. This message will contain the RFQ's ID and any other fields you deem necessary. The messages you need to focus on primarily for inputs are the RFQ (incoming from Kafka) and the Trade Capture Report (in trade history log files).

Incoming RFQ messages and historical trade capture reports are both recorded in JSON, the details of which follow:

Incoming RFQ Message Fields (custom format used within the bank via Kafka streams)

Field		Description
id	String	A unique identifier for this RFQ.
traderId	Long	The bank's ID for this trader.
entityId	Long	The bank's ID for this legal entity.
instrumentId	String	An ID for the asset being trader. Expressed as an ISIN[6].
qty	Long	The quantity to trade.
price	Double	The price at which to trade.
side	Char	'B' for buy, 'S' for sell.

An example of a valid RFQ message:

```
{
  'id': 9315444593154445,
  'traderId': 3351266293154445953,
  'entityId': 5561279226039690843,
  'instrumentId': 'AT0000383864',
  'qty': 250000,
  'price': 1.58,
  'side':'B'
}
```

Trade Capture Reports (FIX formatted)

There are many fields in a trade capture report; significant fields for this component have been given below:

Field		Description
TraderId	Long	The bank's ID for this trader.
EntityId	Long	The bank's ID for this legal entity.
SecurityId	String	An ID for the asset being trader. Expressed as an ISIN[6].
LastQty	Long	The quantity to trade.
LastPx	Double	The price at which to trade.
Side	Integer	1 for buy, 2 for sell

An example of a valid trade capture report message:

```
{
  'TraderId': 7514623710987345033,
  'EntityId': 5561279226039690843,
  'MsgType': 35,
  'TradeReportId': 4812853437600684994,
  'PreviouslyReported': 'N',
  'SecurityID': 'AT0000A0N9A0',
  'SecurityIdSource': 4,
  'LastQty': 350000,
  'LastPx': 115.064,
  'TradeDate': '2018-06-12',
  'TransactTime': '20180612-02:06:15',
  'NoSides': 1,
  'Side': 1,
  'OrderID': 1134246321951090765,
  'Currency': 'EUR'
}
```

References

[1] Wikipedia: https://en.wikipedia.org/wiki/Financial_Information_eXchange
[2] Onixs: https://www.onixs.biz/fix-dictionary/4.4/index.html
[3] Wikipedia: https://en.wikipedia.org/wiki/Internet_protocol_suite
[4] JSON.org: https://www.json.org/
[5] Google: https://sites.google.com/site/gson/gson-user-guide
[6] ISIN database: https://www.isin.org/isin-database/

Epic Descriptions

Notes: The epics described below constitute a completed system. The initial delivery need not include all the epics. Epics required for an initial go-live are: 1, 2, 3, and 8. Consideration should be given to the remaining epics as they may influence your design decisions.

Epic 1: Gather Trade Metadata in Response to Incoming RFQ

When an RFQ message is received, our new component should be able to process big data sets for various data. The component should be extensible and support various different data mining operations. Initially, it is sufficient to simply print these messages to a log file for manual verification. For development purposes, a test program is provided that can be used to simulate incoming RFQ messages in JSON format. Sample trade data files, in JSON format, will be supplied by the team working on trade bookings.

Epic 2: Consume RFQ Messages from a Kafka Stream

The component should receive the incoming RFQ messages in JSON format from a Kafka stream.

Epic 3: Publish Metadata to a Kafka Stream

The component must publish the metadata to a Kafka queue for downstream systems to consume.

Epic 4: Merge Data from Old Bank's Archives

Our bank (new bank) is in the process of merging with another bank (old bank). The legal entities for the old bank will soon be redundant as all existing contracts and portfolios are migrated to the new bank's legal entity. The legacy system contains massive amounts of trade data that should be used alongside the new bank's data.

Epic 5: Intraday Statistics

Initially, it is acceptable for batch jobs to be used to generate historic trade archives. A downside to this is that the trades executed during trading hours are not included in the metadata calculations. Intraday trades should eventually be included in these calculations.

Epic 6: RFQ-Based Statistics

There are other trading system components that manage the RFQ negotiation lifecycle. It has been noted that these systems' log files contain data indicating the outcome of the RFQ, and whether a resulting trade was executed. Our component should be able to process these log files and produce RFQ (as well as trade) insights.

Epic 7: Intelligent Insights

As a technology expert, you should offer suggestions for future enhancements that you think might be of use to the traders. Can machine learning be utilized to provide more intelligent data insights in our component? What recommendations might you make?

Epic 8: Deploying Our Component

Our component will need to be deployed to both our test and production environments. What challenges should be considered when doing this? Think about: cluster/cloud services, how to share trade archive files between HDFS nodes, Spark servers, component orchestration, component configurations (dev/test/prod), Kafka stream integration.

Epic 9: Error Handling

The component should be robust and be able to handle incorrect message inputs and missing data without requiring a restart. Any metadata records published should provide sensible default values for missing fields; e.g., volume traded might be -1 if there were errors loading the data.

User Story Descriptions

Story 1: Total Volume Traded for Instrument

Requirement
As a trader, I want to be able to gauge the level of a customer's interest in a given instrument when I receive their RFQ. A good indicator of this is the volume that they have traded over the past week/month/year for that instrument.

Acceptance Criteria
Unit and integration tests that clearly demonstrate the operation of the required functionality, including handling errors.

Story 2: Total Volume Traded by Legal Entity

Requirement
As a trader, I want to know the health of our trading relationship with a customer when an RFQ is received. One indicator of this is the overall volume traded (with their legal entity) over the past week/month/year. Our component should publish this information.

Acceptance Criteria
Unit and integration tests that clearly demonstrate the operation of the required functionality, including handling errors.

Implementation Note
Our component should only focus on the volume traded. Downstream systems will interpret the quality of our relationship based upon these numbers.

Story 3: RFQ Strike Rates

Requirement
As a trader, I want to know the health of our trading relationship with a customer when an RFQ is received from them. If they like trading with us, then the percentage of RFQs that result in a trade will be higher than if they don't. We should be able to show the percentage of RFQs that resulted in a trade (i.e., the customer's RFQ strike rate) when the RFQ is received.

Acceptance Criteria
Unit and integration tests that clearly demonstrate the operation of the required functionality, including handling errors.

Story 4: Instrument Liquidity

Requirement
As a market maker, I am obliged to trade certain quantities of an instrument to maintain their liquidity and to retain my status in the market. When I receive an RFQ, I should be notified if the instrument being requested is illiquid. I can then decide whether to trade at a poor price to help its liquidity, or to avoid trading if I feel that the instrument poses a high investment risk.

Acceptance Criteria
Unit and integration tests that clearly demonstrate the operation of the required functionality, including handling errors.

Implementation Note
Our component should report the volume traded by our bank with all counterparties for this instrument in the last month. Downstream systems will interpret this data and present it accordingly.

Story 5: Average Traded Price

Requirement
As a trader, I need to make quick decisions regarding a fair price. In a volatile market, where prices are changing frequently, this can be difficult but very profitable when done right. In order to help, I'd like to see the average price traded by the bank over the past week for all instruments included in an RFQ.

Acceptance Criteria
Unit and integration tests that clearly demonstrate the operation of the required functionality, including handling errors.

Story 6: Development Utilities

Requirement
As a developer, I have simple test tools to simulate RFQ submission. While adequate, they could be improved to include a simple history mechanism to allow test RFQs to be resent.

Implementation Note
The ChatServer can currently be used to send String lines to our component. Use this as a starting point for an improved too.

Acceptance Criteria
Happy development team.

Story 7: Trade Side Bias

Requirement
As a trader, I would like to be able to detect whether a customer is trying to build or offload their position for an instrument they are attempting to trade with us. This will help me determine whether they are genuinely interested in making a trade against an incoming RFQ. By seeing the amount I have successfully traded this instrument with them over the past month and week, I can get a feel for whether they are keener to buy or sell and judge my response accordingly.

Acceptance Criteria
Unit and integration tests that clearly demonstrate the operation of the required functionality, including handling errors.

Implementation Note

Our component should report the ratio (as a single figure) of buy/sell traded for downstream systems to display. Figures for weekly and monthly trading should be sent. A negative number will indicate that there was no trading for this instrument in the given period.