**Citi NAM Tech Analyst Induction 2014**

**Group B**

**Tesseract Trading Platform**



Tenzing Passang, Junde Lin, Kewin Li

August 2014

**Table of Content**

Introduction………………………………………………………………………….…………………………………………………..1

DB Schema…………………..…………………………………………………………………….………………………………………2

Architecture Diagram…………………………………………………………………………………………………………………3

Two-Moving Averages………………………..……………………………………………………………………………………..4

Runbook…………………………………………………………………………………………………………………………………….5

Challenges Faced……………………………………………………………………………………………………………………….6

Future………………………………………………………………………………………………………………………………………..7

1. **Introduction**

We were required to construct a multi-tier platform for automated stock trading. The user is allowed to view different stock options, monitor their performance and use different algorithms to select strategies that they will be used to trade. For the strategy, they are allowed to customize the long average period, short average period, number of shares, and exit strategy percentage. The platform pull the current market price that are given from Yahoo Finance Services and set the logistic that will perform trade for them. After the strategies are running, the user will be able to view the analysis of the strategies and choose if they want the strategy to stop or continue to run.

**Team Members:**

Tenzing Passang:

Tenzing is responsible for designing the architecture of the Tesseract Trading Platform, the database design, JPA/Persistence Entity, and construct Yahoo Client which pulls data from the Yahoo Finance Services.

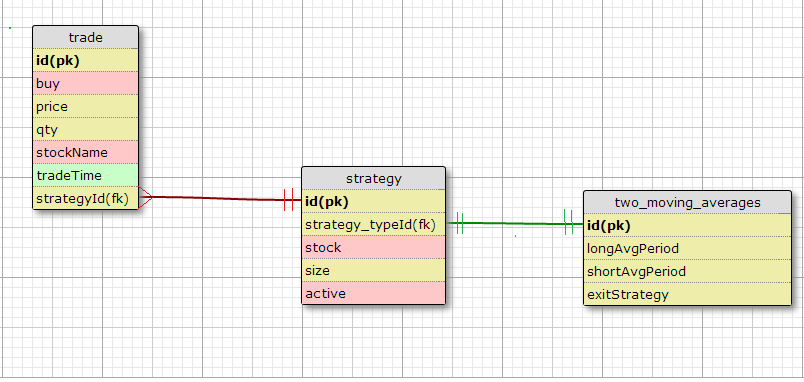
Junde Lin:

Junde is responsible for the front-end of the Tesseract Trading Platform and adjustment of how things look in the front end and make sure that all features are correctly performed on the User Interface. He is also responsible for REST service that links the data from the client to the server.

Kewin Li:

Kewin is responsible for constructing the logistics of the trading algorithms and reading and writing to and from the QueueSender using JMS.

1. **Database Schema**



The database schema for the project was designed to accommodate the different trading strategies. At the center of the design is a “strategy” table which specifies which strategy is implemented.

Originally, various different algorithms, namely: two moving averages, Bollinger bands, and price breakout, were to be implemented. However due to time constraints only the ‘two moving averages’ algorithm was implemented. Hence, the schema was simplified.

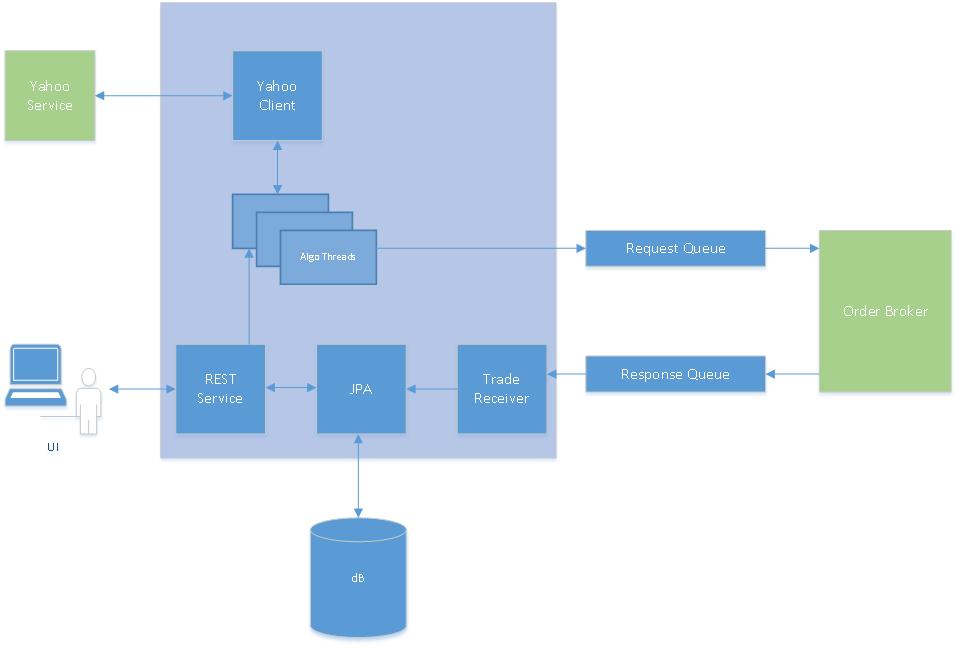
The strategy table is connected to the trade table such that:

* For every strategy, there is zero to n trades.
* Every trade is associated with a strategy

The details about the strategy (in this case, two moving averages) are stored in another table called the two\_moving\_averages.

NB: Originally, the details about the other algorithms were stored in their own tables which were later omitted to simplify the schema.

1. **Architecture/Components**



The diagram above shows the overall architecture of the project. The elements in blue were all created throughout this project while the elements in green were provided to us.

A brief description on each of the elements is given below:

1. User Interface (UI) – HTML, JavaScript, JQuery, AJAX

The UI is the front end of the application. It is a window to the whole project and gives the user various options. In order to start trading, the user can select a strategy and then put in all the parameters for that particular strategy. After the user runs the strategy, real time information about the particular stock is displayed. Whenever the algorithm makes a trade, buy or sell, the trade information gets displayed in the UI. Along with that, every trade cycle (buy-sell, sell-buy) will give a profit/loss percentage which is also updated to the UI.

1. REST Service – JAX-RS

All communications with the UI is first processed by this service. There are various services for various options that the user is presented in the UI, e.g, start strategy, stop strategy, display profit/lost, etc. This in-turn works with the Persistence Entity which persists/retrieves data from the database. The REST service also acts as an starting point for the algorithm to start as once it gets the command from the UI, it opens up a thread to run the algorithm.

1. Persistence Entity – JPA

The persistence entity deals with all the database reads and writes. Java entities through JPA were created to make the operations very smooth. Various elements of JPA were used to carry out the CRUD – create, retrieve, update and delete operations. JPQL was used as the query language to query the database. The JPA is the only part which connects to the database directly, all other units talk to the database through this persistence entity.

1. Algorithm – Multiple Java Threads

The algorithm is the main logic of the whole project. Based on the parameters provided by the user, the algorithm keeps track of historical data (in this case, the two moving averages) and decides whether to short/long a trade. The algorithm gets the data from the yahoo client, which in-turn talks to the yahoo finance API.

Based on that, if the algorithm decides to make a trade, it sends the trade to the Request Queue.

1. Yahoo Client – Java HttpURLConnection

This client makes an Http connection with the Yahoo Finance API and retrieves the required information for the algorithm to process. It has logic to choose the price (last traded price or average of ask/bid) based on the stock type.

1. Request Queue – JMS ActiveMQ

This is a JMS queue which takes trade requests from the algorithm and then sends it to the Order Broker for processing.

1. Order Broker

This is a simulator which simulates a stock broker. It reads trades from the Request Queue and then sends it to the Stock Exchange simulator called QuickFixJ. Once the order is processed, it is then sent to the Response Queue.

1. Trade Receiver – Java ActiveMQ Listener

This part listens to the trades on the Response Queue and uses the JPA to persist the trades into the database.

1. **Two-Moving Averages**

Every 5 seconds, the Two-Moving Averages would call the Yahoo Client Service which get the latest traded price from Yahoo Finance Service with the stock name and adds in the price tree-map. There are two price tree-maps: one that is used to calculate the longMovingAverages and the other use to calculate the shortMovingAverages of the strategy. After that, the two price tree-maps are checked to see if all the entries fit into the timestamp period between the timestamp now and the timestamp subtract the longMovingPeriod or the timestamp subtract the shortMovingPeriod. The sum of the entire price in either of the tree-map is recorded by sumLongPrice or sumShortPrice. If an entry is deleted from either the tree-map, then the price of that entry is deleted from that of the sumLongPrice or sumShortPrice. The price is checked to see if it exceed the boundary of the exit condition, and if it is, it will be bought or sold immediately. After that, shortMovingAverage and the longMovingAverage are calculated with the sumLongPrice and sumShortPrice and the size of the entries in both of the price tree-maps. The shortMovingAverage and longMovingAverage are compared to the previous shortMovingAverage and longMovingAverage and are sold if the shortMovingAverage passes the longMovingAverage or are bought if the shortMovingAverage goes below the longMovingAverage. The trades are bought, sold, or exited; it is passed to the OrderBroker on the Order\_Request Queue via JMS.

**5. Run Book**

* Run ActiveMQ for the queues
* Start the derby database
* Run the order broker as a java application from the project OrderBroker: src/com.citi.trading.order/OrderBroker.java
* Start the quickfixJ exchange
* Start the Listener to Response queue as a java application from the project StockRestService:

Java Resources/src/cc.service/ReceivingTrade.java

* Finally, start the WebService and run the ServiceResource.java on the tomcat server. The Service Resource is on the project StockRestService/ Java Resources/src/cc.service/ServiceResource.java
* Open <http://localhost:8080/StockRestService/login.html> for the UI.

1. **Challenges Faced**

* It took some time to fix all of the bugs that occurred when we ran our initial version of the Two Moving Averages Algorithm. Sometimes, the moving averages weren’t calculating correctly and other times, the algorithms were not exiting correctly. We had to dissect our code and then debug it in order to pinpoint where the bug was occurring.
* Trying to agree on a specific architectural logic.
* Agreeing on the same variable name.
* Handling the various exceptions in our code.
* Unifying a specific User-Interface color scheme.
* Errors in our codes
* Since our code base was spread across three different machines, it was difficult to maintain a working copy of the code. Hence, connecting the different elements of the project was a bit tedious.

1. **Future Changes/Improvements:**

* Implement all the three strategies/algorithms.
* Create unit test for each and every logic in the project
* Calculate profit/loss for each and every closed position so as to isolate each and every trade.
* Make the UI more fluid and show additional information to the user, for example, statistics based on day, time, number of shares, particular stocks, various parameters, various strategies, long/short, etc.