## CS 242 Final Project Proposal

**Trade Simulator**

**Christian Cygnus (cygnus2), Daniel Tian**

1. **Abstract**
   1. **Project Purpose**

Provide a locally served stock trading/arbitrage system that would allow aspiring traders to learn how to write trading algorithms and test them out on a mock exchange.

## Background/Motivation

When I worked for IMC during a Winter Internship, the interns all participated in a mock exchange that served the same general purpose as my project proposal. I had a lot of fun testing out my trader against bots, and I learned a lot from the process. However, the simulator wasn’t perfect, and I think I could make it more stable and usable if I made my own version. Additionally, building such a simulator would give me additional experience that might be valuable before I start my full-time position at IMC.

# Technical Specifications

* 1. **Platform: ​** Backend + Local Clients, Web based status checker
  2. **Programming Languages: ​** Java
  3. **Testing Framework:** Junit, along with Mockito and PowerMock
  4. **Stylistic Conventions:** Standard Java Convention (Checkstyle) + Method Javadoc
  5. **SDK: ​**Dropwizard for REST
  6. **IDE: ​**IntelliJ (WebStorm for Web Dev)
  7. **Tools/Interfaces: ​**Google Chrome for Web
  8. **Target Audience: ​**Computer Scientists wanting to learn the basics of trading/arbitrage

# Functional Specifications

## Features

* Backend that would support subscribing clients and automatic bots to different exchanges
* Client SDK that would handle serving book updates to client code
* Website for monitoring bot status and other exchange information

## Scope of project

Currently my goals would be limited to a service that runs locally on network, not anything that allows users to connect from outside the LAN. Additionally, there may be limitation on the types of activities that could happen on this exchange (most likely just bid/asks that can be immediate execute or long-standing trade). There most likely won’t be support for advanced limit purchases or specific infrastructure to simulate options trading.

# Timeline:

## Week 1

* + 1. Build backend API on a server
       - Basic POST request for BUY/SELL
       - Basic GET for info such as current price
    2. Build barebones Java library for interacting with server
       - Can POST to server
       - Can GET current price

## Week 2

## Build basic bots for the market

## Will buy and sell to simulate the ups and downs of the market

## Statistical distributions to vary from a mean price

## Add functionality to the client library for interacting with server

## Immediate buy/sell, along with good ‘til canceled

## Add endpoints for user creation

## Can subscribe handlers to handle updates

## For “extra credit”, have client code subscribe to updates pushed from server

* 1. **Week 3**
     1. Build website for basic monitoring
        + Bid/Ask spread for single market
        + Current price
        + User list
        + Book depth
     2. Expose API endpoint to get access to server data
        + Get the current bid/ask from the server
        + Get a list of users
        + Get a list of the current symbols
        + Get a list of the transactions for a specific symbol
        + Any additional data that is being stored could be retrieved
  2. **Week 4**
     1. Enhance Website
        + Live graphs of the trades on any exchange
        + Live graph for user performance
     2. Add more advanced bots to the exchange
        + At least one additional bot with a different technique than the simple bot
     3. Server Enhancements
        + Add tracking of funds to the server
        + Ensure consistency with multiple clients on the same exchange

1. **Future Enhancements**

This project has a large number of possible extensions. They include:

* The ability to pull in actual stock pricing data to practice writing algorithms on normal stock data
* Additional types of bots/algorithms that play on the arbitrage exchange
* Enhanced control/monitoring features for the web interface that would show a real time graph of the stock price and the book depth
* Arbitrage + Other techniques for trading supported

The only question that remains is if IMC would let me make it an open source project. While I am sure for my own private learning and in an academic setting, they would not have a problem with the software. But I may only be able to release it internally for new interns if IMC deems it as a competition threat.

Week 1 Rubric:

| **Category** | **Weight** | **Scoring** | **Requirement** |
| --- | --- | --- | --- |
| Basic Preparation | 2 | 0-1 | Ready to go at the start of section |
| Cleverness | 2 | 0-2 | The hardest points on the rubric |
| Code Submission | 4 | 0-2 | Submitted correct content on time and to the correct location in the repository |
| Decomposition | 4 | 0-2 | Project is adequately decomposed to different classes and methods |
| Documentation | 4 | 0-2 | Comments for each class and each function are clear and are following style guides |
| Effort | 2 | 0-2 | Perform considerable amount of work |
| Naming | 2 | 0-2 | Variable names and method names are readable and are following the naming conventions of the language you chose |
| Overall Design | 5 | 0-2.5 | Have nice approaches and structures in overall |
| Participation | 5 | 0-2.5 | Interact with the group 2 times (ask a question, make a comment, help answer a question, etc.) |
| Presentation | 4 | 0-2 | Present the code clearly |
| Requirements – Backend API | 5 | 0-2.5 | * 0 points: Lacks any Backend API progress * 1 point: Implements GET or POST requests for transactions, but not both * 2 points: Implements both GET and POST requests for transactions * 2.5 points: Implements both GET and POST requests for transactions, along with endpoints for getting users |
| Requirements - Client | 5 | 0-2.5 | * 0 points: Lacks any form of Client code * 1 point: Implements GET or POST requests for transactions, but not both * 2 points: Implements both GET and POST requests for transactions * 2.5 points: Implements both GET and POST requests for transactions, along with a demo use of the client code |
| Requirements – Backend Unit Testing | 4 | 0-2 | * 0 points: No unit tests covering Client code * 1 point: Minimal testing, does not cover all endpoints * 2 points: Testing covers all endpoints, offline |
| Requirements – Client Unit Testing | 4 | 0-2 | * 0 points: No unit tests covering Client code * 1 point: Minimal testing, does not cover all endpoints * 2 points: Testing covers all endpoints, offline |
| **Total** | **52** |  |  |

Week 2 Rubric

| **Category** | **Weight** | **Scoring** | **Requirement** |
| --- | --- | --- | --- |
| Basic Preparation | 2 | 0-1 | Ready to go at the start of section |
| Cleverness | 2 | 0-2 | The hardest points on the rubric |
| Code Submission | 4 | 0-2 | Submitted correct content on time and to the correct location in the repository |
| Decomposition | 4 | 0-2 | Project is adequately decomposed to different classes and methods |
| Documentation | 4 | 0-2 | Comments for each class and each function are clear and are following style guides |
| Effort | 2 | 0-2 | Perform considerable amount of work |
| Naming | 2 | 0-2 | Variable names and method names are readable and are following the naming conventions of the language you chose |
| Overall Design | 5 | 0-2.5 | Have nice approaches and structures in overall |
| Participation | 5 | 0-2.5 | Interact with the group 2 times (ask a question, make a comment, help answer a question, etc.) |
| Presentation | 4 | 0-2 | Present the code clearly |
| Requirements – User Creation | 4 | 0-2 | * 0 points: No endpoint for user creation * 1 point: Only Client-side or Server-side support for user creation * 2 points: Both Client- and Server-side support for user creation |
| Requirements – Server Push Updates | 5 | 0-2.5 | * 0 points: No messages can be pushed from the server * 1 point: Messages can be sent, but not in response to transactions * 2 points: Messages can be sent in response to transactions, but only to one user * 2.5 points: Messages can be sent asynchronously to all subscribed users in response to transactions |
| Requirements – Client Subscribe Update Handlers | 5 | 0-2.5 | * 0 points: No messages can be received in the client * 1 point: Messages can be received, but no action can be taken on them * 2 points: Messages can be received and processed statically in method * 2.5 points: Messages can be asynchronously received and can be passed on to subscribed handlers |
| Requirements – Immediate Transactions | 4 | 0-2 | * 0 points: No immediate transactions can be sent * 1 point: Immediate transactions can be sent, but only support matching with a single other transaction * 2 points: Immediate transactions can be sent and filled if they can match with any number of corresponding transactions in the database |
| Requirements – Client bot | 4 | 0-2.5 | * 0 points: No bot exists * 1 point: Bot exists and can send a single transaction * 2 points: Bot exists and can send transactions in a loop * 2.5 points: Bot exists and can intelligently sent transactions with prices based on the historic price momentum. |
| Requirements – Unit Testing | 4 | 0-2 | * 0 points: No unit tests covering new features * 1 point: Limited testing, does not cover all new endpoints * 2 points: Testing covers all endpoints (offline) |
| **Total** | **60** |  |  |

Week 3 Rubric

| **Category** | **Weight** | **Scoring** | **Requirement** |
| --- | --- | --- | --- |
| Basic Preparation | 2 | 0-1 | Ready to go at the start of section |
| Cleverness | 2 | 0-2 | The hardest points on the rubric |
| Code Submission | 4 | 0-2 | Submitted correct content on time and to the correct location in the repository |
| Rubric | 2 | 0-1 | Rubric was completed on time |
| Decomposition | 4 | 0-2 | Project is adequately decomposed to different classes and methods |
| Documentation | 4 | 0-2 | Comments for each class and each function are clear and are following style guides |
| Effort | 2 | 0-2 | Perform considerable amount of work |
| Naming | 2 | 0-2 | Variable names and method names are readable and are following the naming conventions of the language you chose |
| Overall Design | 5 | 0-2.5 | Have nice approaches and structures in overall |
| Participation | 5 | 0-2.5 | Interact with the group 2 times (ask a question, make a comment, help answer a question, etc.) |
| Presentation | 4 | 0-2 | Present the code clearly |
| Requirements – Website Current Prices | 5 | 0-2 | * 0 points: Website is non-functional and/or non-existent * 1 point: Shows price, but only for one market * 2 points: Shows price, bid, and ask for the selected market |
| Requirements – Website Users List | 5 | 0-2 | * 0 points: Website is non-functional and/or non-existent * 1 point: Can only fetch a single user * 2 points: Shows a full list of registered users, updates automatically |
| Requirements – Website Book Depth | 5 | 0-2 | * 0 points: Website is non-functional and/or non-existent * 1 point: Can only see bids or asks, or not switchable between markets * 2 points: Can view live updating book depth with clear divide between bids and asks |
| Requirements – Backend endpoints | 5 | 0-2 | * 0 points: No new endpoints were added * 1 point: Added only one endpoint for symbol summaries, but did front-end parsing on transactions * 2 points: Endpoints added for symbols and for getting a specific symbol’s transactions |
| Requirements – Unit Testing | 5 | 0-2 | * 0 points: No unit tests covering new features * 1 point: Limited testing, does not cover all new endpoints * 2 points: Testing covers all endpoints, with manual test plan for web UI |
| Schedule | 2 | 0-1 | * Revised schedule if necessary |
| **Total** | **63** |  |  |

Week 4 Rubric

| **Category** | **Weight** | **Scoring** | **Requirement** |
| --- | --- | --- | --- |
| Basic Preparation | 2 | 0-1 | Ready to go at the start of section |
| Cleverness | 2 | 0-2 | The hardest points on the rubric |
| Code Submission | 4 | 0-2 | Submitted correct content on time and to the correct location in the repository |
| Rubric | 2 | 0-1 | Rubric was completed on time |
| Decomposition | 4 | 0-2 | Project is adequately decomposed to different classes and methods |
| Documentation | 4 | 0-2 | Comments for each class and each function are clear and are following style guides |
| Effort | 2 | 0-2 | Perform considerable amount of work |
| Naming | 2 | 0-2 | Variable names and method names are readable and are following the naming conventions of the language you chose |
| Overall Design | 5 | 0-2.5 | Have nice approaches and structures in overall |
| Participation | 5 | 0-2.5 | Interact with the group 2 times (ask a question, make a comment, help answer a question, etc.) |
| Presentation | 4 | 0-2 | Present the code clearly |
| Requirements – Live Updating Graph | 5 | 0-2.5 | * 0 points: No graphs displayed on the website * 1 point: Graph only displays a static status of the exchange * 2 points: Graph is live-updating with information from the selected exchange * 2.5 points: Same as above, but user performance can also be tracked by the live updating graph |
| Requirements – Fund tracking | 5 | 0-2 | * 0 points: User funds are entirely ignored * 1 point: User funds shown, but not updated in all cases * 2 points: User funds are updated when any bid/ask they submitted gets matched and executed. Funds can be positive or negative |
| Requirements – Additional Bot | 5 | 0-2.5 | * 0 points: No addition bots were added to the exchange * 1 point: Bot added follows technique very similar to the Simple Bot * 2 points: New bot has unique behavior and can trade on the exchange * 2.5 points: New bot + additional ways to launch bots that doesn’t require extensive setup each time |
| Requirements – Multiple Connection Stability | 5 | 0-2 | * 0 points: Crashes happen frequently when multiple users/bots connect * 1 point: Server mostly stable, occasional data inconsistencies may arise over time * 2 points: Server is fully able to handle multiple client threads without race conditions on data access |
| Requirements – Unit Testing | 5 | 0-2 | * 0 points: No unit tests covering new features * 1 point: Limited testing, does not cover all new endpoints * 2 points: Testing covers all endpoints and bots, with manual test plan for web UI |
| Schedule | 2 | 0-1 | * Revised schedule if necessary |
| **Total** | **63** |  |  |