Project Plan

Callum Thomas – Max Gunton – James Ryan – Shawn Dhaliwal

University of Victoria

SENG 468

# Project Plan

## Timeline

Feb 1st

Documentation (Project Plan, Initial Requirements, Architecture)

* Hard copy submitted to the professor in class on day of deadline.

Verified execution of 1 user workload file

* Logfile submitted to the professor via **email** by midnight.

Initial build of the system components (Transaction server, database, workload generator, web server)

**Build on workstation using the quote server if possible, or a mock quote server if unable.**

Feb 8th

Verified execution of 10 user workload file.

**Build on workstation or VM and must use the course quote server.**

Feb 15th

Verified execution of 45 user workload file

**Build system using docker containers.**

Mar 1st

Verified execution of 100 user workload file

Mar 8th

Verified execution of 1000 user workload file

Mar 22nd

Verified execution of final 2019 workload file by 11:59:59 PM.

Web Interface built. Start testing.

April 3rd

Group project presentations during regular class time

Presentation schedule will be announced in class, end of March

Students must attend all presentation days

April 5th

Demonstration of web interface

Note: one hour for the demo must be booked with your TA by March 24th

April 9th

Submission of final project reports, both printed and in PDF format, by 11:59:59 PM

# Project Administration

## Source Code Repository

The project source code is being hosted on a private GitHub repository. If access is necessary to the repository, please email [ryan\_james@hotmail.ca](mailto:ryan_james@hotmail.ca) with your Github user name so that you can be added as a contributor so that you can view it.

# Initial Requirements

## Transaction Server

Receive a request from a client (workload tester or webserver) and process it. Communicating with the quote server and database as necessary during request processing, then providing a response back to the client to use as feedback.

## Database

Need a Database to store users, transactions, pending buys and sells, and log entries. Transaction server will send queries to the database modifying it.

## Workload Generator

Needs to generate random workloads to test the performance of our architecture. Workloads need to be .txt files to be read by the webserver and sent to the transaction server.

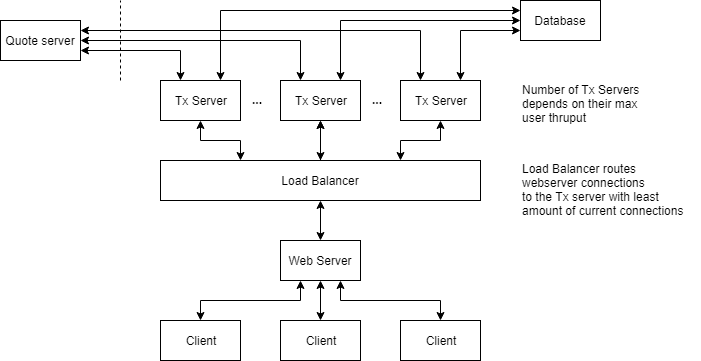
## Workload Tester

The tester needs to be able to correctly read each user command specified in a .txt workload file, and then send it to the transaction server. The workload files can be found on the course website. This will be used to read verification workload files provided by the professor to hand in for various milestones

## Front End User Client and Web Server

Web server needs to serve html and JavaScript of the front-end user client to the customer. Web server should be a server that runs in a Docker container responding to HTTP requests. Front-end user client needs to be an easy to use interface for the user to create and edit an account, buy and sell stocks, look up quotes, set buy and sell triggers, and add money to their balance.

# Architecture



## Transaction Server

The transaction server has been planned to be a multi-threaded application developed in Python 3.7 which will receive and process requests from clients. It will have direct communication with the quote server, database, and web server (client connections). Depending on how many clients each server handles concurrently, the multiple instances of the server may be run with a load balancer between it and the clients such that the number of users on each server is below its maximum users. We are using a MySQL Python 3.7 driver for interfacing with databases. Connections between the client and the transaction server will be handled using TCP socket connections, using Python’s Socket API.

### On Initialization

On startup of a transaction server, the server will open a socket and bind it to the user specified port. If no port is specified in the arguments of the server, the default of 8080 will be used. The server will then listen and wait for a connection from a client and accept the connection.

### On Connection from a Client

On a connection from a client, the server will start a new thread for that connection to use, and then wait for another connection. Each connection will have its own thread. This thread will proceed to a function designated as the start point for a new connection.

This thread will then wait for data from the client it is connected to. Upon receiving data, it will open a connection to the database and parse through the data and forward the parsed data to a corresponding function to process that specific request type. Once the request has been processed a response will be sent to the client and the socket will then wait for more data to be sent.

### On Connection Close

On the closing of a connection to the transaction server from a client, the thread for this connection will perform cleanup and then close.

## Load Balancer

The load balancer will act as a mediator between the web server and the transaction server(s). On a new connection, it will route the connection to the transaction server which has the least current active connections.

If the load of the current number of servers goes above a specified threshold (to be determined after testing), a new transaction server is to be created, and its connection information can be added on the fly without any downtime.

The load balancer will likely be developed using Python 3.7 for ease of development and to ensure compatibility with the transaction server and the web server.

## Database

The database is implemented using the MySQL 8.0 DBMS.

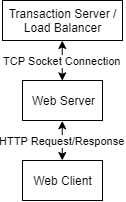
### Tables

* Users
  + User id
  + Accessible money
  + Reserved money
* User\_stocks
  + Id
  + User id
  + Stock name
  + Accessible stocks
  + Reserved stocks
  + Buy trigger
  + Buy amount
  + Sell trigger
  + Sell amount
* Pending\_buys
  + Id
  + User id
  + Stock name
  + Expiry
  + Num
  + Req amount
  + Transaction number
* Pending\_sells
  + Id
  + User id
  + Stock name
  + Expiry
  + Num
  + Req amount
  + Transaction number
* Log\_entries
  + Id
  + Type
  + Timestamp
  + Server
  + Transaction number
  + Command
  + Price
  + Action
  + User id
  + Stock symbol
  + Funds
  + Quote server time
  + Crypto key
  + Filename
  + Error message
  + Debug message

## Web Server/Client

### Web Server

The purpose of the web server is to serve webpages to a client connecting through a web browser, and to translate HTTPS requests from the client into a request that the transaction server can read and then pass the request to the transaction server (or load balancer) using a TCP socket connection. When the webserver receives a response from the transaction server (or load balancer), it will then send a response to the client’s HTTPS request back to the client.



### Web Client

This will be implemented purely using HTML webpages with minimal CSS and JavaScript. The reason for this is to focus on functionality of the prototype and put the development of the look and feel of the web client for a future version.

## Security Considerations

The Web Client and Web Server will need to transfer data using HTTPS. The support for HTTPS will prevent man in the middle attacks and eavesdropping attacks. Python provides support for this in it’s HTTP module in the Python Standard Library.

Communication between the Web Client and Transaction server will need for the socket connections to use Transport Layer Security encryption. The Python library “ssl” in the Python Standard Library, allows for this functionality. This will provide proper authentication and end to end encryption between the source and destination.