Psychic Waffle Project Proposal

A.J. Feather
af2849@columbia.edu

Andrew Grant
amg2215@columbia.edu

Jacob Graff
jag2302@columbia.edu

Jake Weissman
jdw2159@columbia.edu

October 6, 2016

1 Synopsis

1.1 Motivation

Large buy/sell orders have the potential to significantly move markets if not executed properly. Big orders that are executed in a single trade can adversely push the market price against the trader. For example, if a trader shows he or she has a large sell order, the relevant asset's price will almost inevitably move lower; this will result in the trader selling at an average lower price. The goal of this project is to build a system that accounts for these market dynamics and executes client orders as optimally as possible for them. By filling orders on their behalf *and* at the optimal price, our clients will save time and money. Our system will also provide the customer with visibility in terms of how an order is actually executed. The customer currently has no way of systematically monitoring an order; our system will provide the customer with immediate feedback on executed orders, as well as a history of past orders.

1.2 Functionality

Our system will receive a customer order, figure out the most optimal method of execution (it will usually split the order up into increments), and fill the order. The heart of the system will be an algorithm that figures out the most optimal method of execution: given X shares left to sell at time Y, the system will determine the next sell order. This requires the system to calculate when to execute the next sale of shares, as well as how many shares to sell at the current price.

Furthermore, the system will provide the user with immediate feedback detailing each execution attempt. More specifically, after each execution attempt, the system will respond with whether the order was filled or not, and if the order was filled how many shares were traded and at what average price. We also plan to provide the ability for the trader who placed the transaction to monitor the progress of the transaction as it is being executed throughout the day. However as of now the user will not be able to stop or cancel an order mid-day; that is, once an order is requested, the system will complete it.

As now, we will implement a TWAP algorithm; TWAP stands for Time-Weighted Average Price. This is a fancy way of saying we will break up the order evenly throughout the day. Nonetheless, the customer has indicated they may want to change this in the future.

1.3 Users

Since the system was requested by a Pension Fund, this will be our primary user for now. Furthermore, our system is designed to handle large sell orders for customers who are happy to get their orders done over the course of the day (as opposed to instantaneously). Nonetheless, anyone trading Acme on this exchange, who needs similar types of orders filled should be able to use our system. This would include Hedge Funds, Banks, Individual Traders, Brokers and other financial professionals/firms. While the platform will accommodate

those entities, the specific user interfacing with our platform will be a trader who will be executing the transaction on behalf of their client. The system will initially support a single user to place and monitor trades and will eventually scale up to accommodate more traders to use the system concurrently. Furthermore, this single user will only be able to execute a single order.

1.4 Usage

As of now the client's main use of our system will be to execute large sell orders. The additional features we plan to provide are better monitoring tools and potentially an improved visualization platform as a stretch goal.

1.5 User Stories

1.5.1 User Login

The user should be able to log in to the platform with unique user credentials.

1.5.2 Request Number of Shares

Users will need to be able to retrieve the number of total shares available for sale.

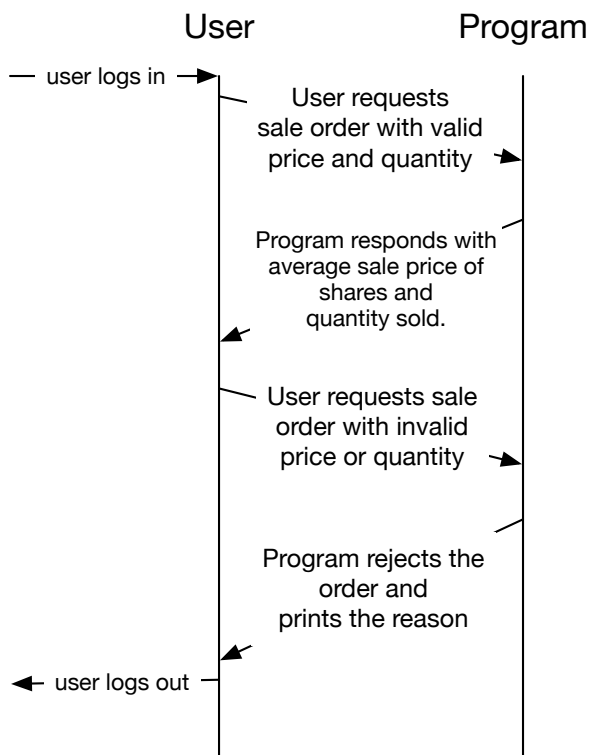### 1.5.3 Request Best/Worst Sell Price Available

In case the user wants to buy at a particular point in time, we will give them the ability to request the best and worst available buy prices.

1.5.4 Submit Sell Order

The user should be able to send a sell order to the system, giving a sale price and the number of shares thaneed to be soldld.

1.6 Sell Order Work Flow

The largest back end challenge we face at this time is attempting to match the user's specified price. To handle this, we plan on using a form of the VWAP algorithm. The following diagram shows that workflow (on the next page).

User   Program

— user logs in →

User requests
sale order with valid
price and quantity

Program responds with
average sale price of
shares and
quantity sold.

User requests sale
order with invalid
price or quantity

Program rejects the
order and
prints the reason

← user logs out

Note: Users may submit any price, but the program will only allow the user to purchase shares within the range in which it expects it can execute at the correct price with a reasonable amount of certainty.

1.7 Error Handling

We will do our best to provide the user with the ability to report unexpected behavior at any point. We will also write tests that cover most essential features that will log errors while the program is running in addition to the full testing suite.

These logs will include issues such as failed connections and unexpected return values.

We also plan to cleanse all text as it is entered into the program.

2 Technology

2.1 Programming Language

The bulk of our code will be written in python - python has many easily accessible dev tools, and can for many applications substantially decrease development time.

2.2 Development Framework

Our development framework will be flask (http://flask.pocoo.org/), a web development framework, which allows webservers to be created quite easily, and gives a lot of flexibility to the developers.

In addition, we will use pip, the python package manager, to install the packages we need, and virtualenv in order to properly maintain our python environment.

2.3 Build Automation Tool

Our preliminary build automation tool will be PyBuilder (https://pybuilder.github.io/).

2.4 Static Analysis Tool

Our preliminary static analysis tool will be Pylint (https://www.pylint.org/), a popular linter for python.

2.5 Unit testing tool

Our preliminary unit testing tool will be PyUnit, the unit testing tool that is built in to python. However, we recently stumbled upon unittest, another unit-testing library, which we may use.

2.6 Persistent Data Store

Our persistant data store will be a PostgresQL database, in which we will store (at least initially) user data and password hashes. We will also store order history in some capacity.