The ASE Avengers Present

SUPER Trade Wizard 7000

J.P. Morgan Project Proposal

9/28/2016

Brennan Wallace [bgw2119]

Stanislav Peceny [skp2140]

Akanksha Gupta [ag3749]

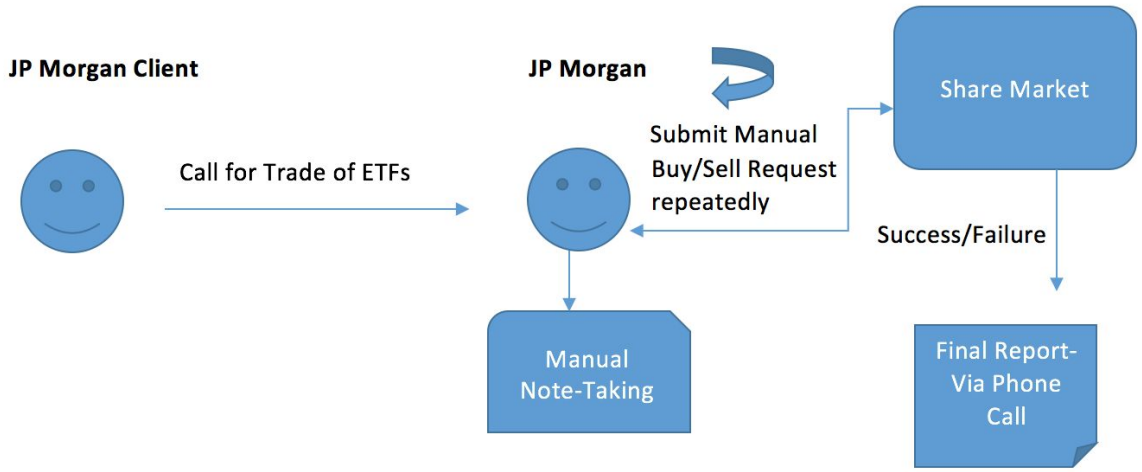
Ishan Guru [ig2333]

Github repo: https://github.com/nyletara/ASEavengers.git

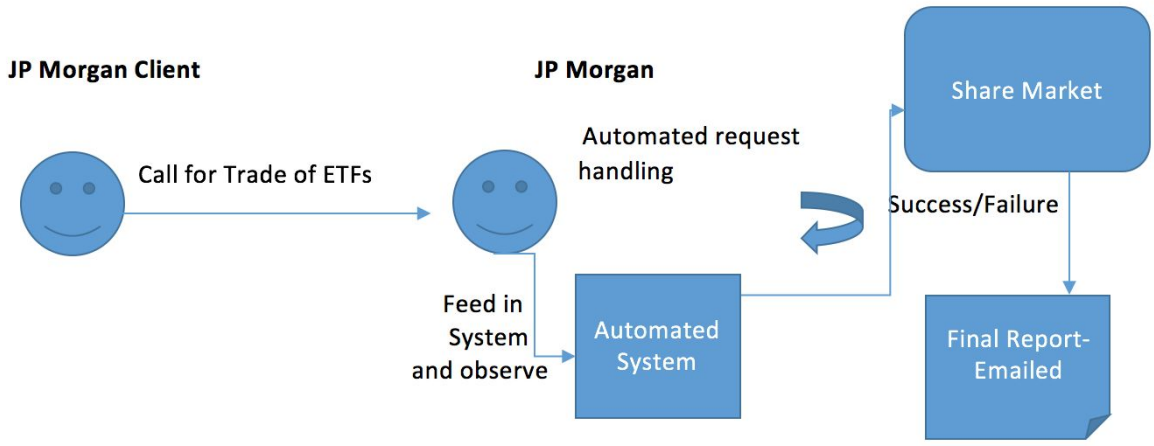
Trello: https://trello.com/b/RyfTSza6

Project Proposal

Currently, J.P. Morgan & Chase clients call the trader at J.P. Morgan to buy or sell certain ETFs. This process from a trader's perspective involves receiving an order of certain quantity, and buying or selling in “chunks” over a set period of time. For example, a client can make a request to sell 100K shares of a certain ETF over ten hours, which is noted down manually by the trader. He further responds to the request by selling equally-sized bundles of ETF shares throughout the time period of the trade at equidistant intervals. This process is conducted manually, in which a trader enters each one of these “child-trades” over the set period of time, and has a physically written “parent-order” that is required to be completed. In the example above, every five hours the trader is required to manually enter a sell order of 20K shares so that by the end of the day all shares are sold. As evident from the description above, this process is prone to errors as each trader may enter incorrect information that can lead to incorrectly placed orders. Furthermore, this manual method leads to increased time spent on performing trades and failure to fulfill orders within a desired time frame. Hence, the process requires repeatedly submitting trades for each client, which is error-prone and time consuming. The current, manual process, is shown in the diagram below.



Our customer, J.P. Morgan would like to replace the current manual method of servicing their clients with an automated system in the form of a web application. This solution would give time to traders to do less manual work, service higher number of clients, and allow for better practices within the firm. Hence, the system proposed is an automated trading system that allows the trader to simply input details including quantity, side (buy/sell), urgency of the parent-order, select a trading strategy, and place the order. Consequently, child-orders are created and sent when necessary by the algorithm. The trader is then able to monitor the progress of the parent order, as well as all pending child orders. The automated is represented by the diagram below.



Specifically, the client requires an algorithm that trades using the Time-Weighted Average Price (TWAP) methodology, and thus eliminates error-prone and time consuming manual order placement. TWAP is an industry term that breaks down a single trade into several smaller transactions to minimize the impact from fluctuations of market price. Although exchange side failures are uncontrollable by the system, it is essential to create a reliable, robust, and error-handling system to minimize the number of rejected orders.

The target users of our system will be J.P. Morgan employees trading for the firm. There are two types of users for our system traders and supervisors. Traders will use the system primarily to execute trades. Supervisors will perform various administrative tasks on the system (mirroring their job responsibilities). Both will use the system to view various data, such as progress of the trades placed with the system and various historical data related to activity on the system.

There are two main benefits provided by the system. First, by automatically breaking down and issuing orders to exchanges the system automates the time consuming and error-prone task that traders would otherwise have to be performed manually. Second, all commands by users and transactions with the exchange will be tracked and recorded, providing greater accountancy for JPMorgan's trading activities. The result of this will be that JPMorgan can identify better performance trends and consultant an accurate historical account when investigating an issues.

User Stories

1: Login to Application

Description: As an authorized JPMorgan trader, I need to be able to access the application and based on my role and privileges perform certain functions including placing trades or analyzing data.

2: Logout of Application

Description: As an authorized trader or administrator, I need to be capable of logging out of the application to ensure that unauthorized users do not gain access to confidential information, analytics, and cannot perform trades. I click the ‘Logout’ button and lose privileges to all functions except for ‘Login’.

3: Trader views history.

Description: As a trader I can click a button to have my entire history displayed including parent and child trades clearly organized. This data should further be sortable and filterable by date, ETF, and strategy used.

4: Buy ETFs

Description: As a trader I want to enter the details for each trade, click buy, and confirm my trade. The details include the ETF, the number of shares to purchase, time to complete the parent trade.

5: Sell ETFs

Description: As a trader I want to enter the details for each trade, click sell, and confirm my trade. The details include the ETF, the number of shares, time to complete the parent trade.

6: Select Trading Strategy

Description: As a trader, I want to be able to select the trading strategy that I want my order to follow. Specifically, I only want to select the Time Weighted Average Price (TWAP) strategy.

7: Tracking trading activity

Description: As a trader, I want to be able to view the current status of a parent order that was previously made, including viewing the child orders associated with the parent.

Error Cases

Likely error cases include various invalid input types and formats. On the client side, such errors shall be handled on the User Interface through preventative measures including careful validation and error messages. On the server side we shall rely on validation and error messages sent to the UI and consequently relayed to the user. An example would be an inappropriate input such as typing in a non-positive whole number into the amount of ETFs requested. We would ideally prevent such an action though client side technologies (there are Javascript libraries that block such behavior). However, we would also have validation that occurs when submitting data. The validation would stop the submission and ask for re-entering valid values, explaining why the submission had been cancelled. Lastly on the server we would have a similar response if a trade order fails the servers' validation from server to client to user. Another type of error is something that occurs due to an issue with exchange (i.e. it does not respond or refuses to process an order). In such a case the server would know first and then the message would be propagated to the user in the method described above. Finally, all of this type of activity would be carefully logged by the servers (through such logging would likely exclude issues the the client facing UI deals with on its own).

Technologies

We are planning on using the Meteor framework due to how fast it is to build user based web-applications. Meteor is a Javascript based full-stack framework built on Node.js and is opinionated making the decision for the persistent data store automatic as Meteor only works with MongoDB.

Architecture Update

We are considering a new architecture system that separates the concerns of user interactions and account with the actual trading. Doing so allows both choices to be optimal and if one needs to be updated we can do so without scraping the entire project (this could be done by the client at a future date as well). We will keep Meteor to handle user-accounts and user-interactions however trading requirements will be passed to a separate server (a node.js instance for its event style system) which will handling sending and receiving messages to the exchange. The twist from a more typical setup is that the trading node.js system will update the database meteor reads trade data from and in this way we can save a lot of back and forth between meteor and the node.js system (plus it avoids meteor's slowness by removing it from the trade execution flow).