Team #01

1. Dante Pasionek → Dpasi314
2. Preston Kelley → pkelley
3. Walker Schmidt → walkyd12

Project

PaperTrader - An alternative to traditional stock market simulators
Creating a user-friendly trading environment to provide real-time information to gain valuable trading suggestions

Project Summary

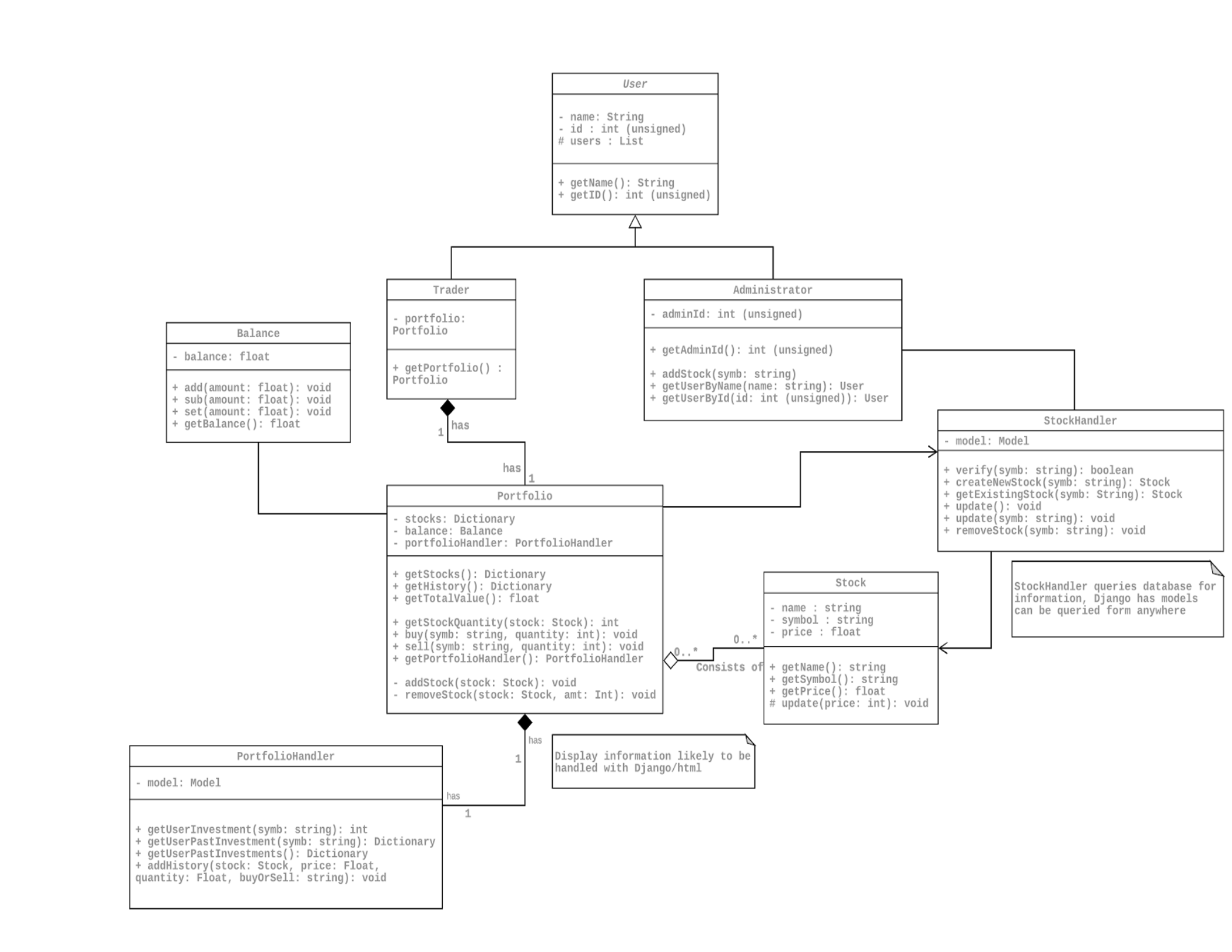
A python based stock market simulator, web application that allows users to trade in real time with fake currency. Traders are able track their investments and find optimal trading strategies.

Index

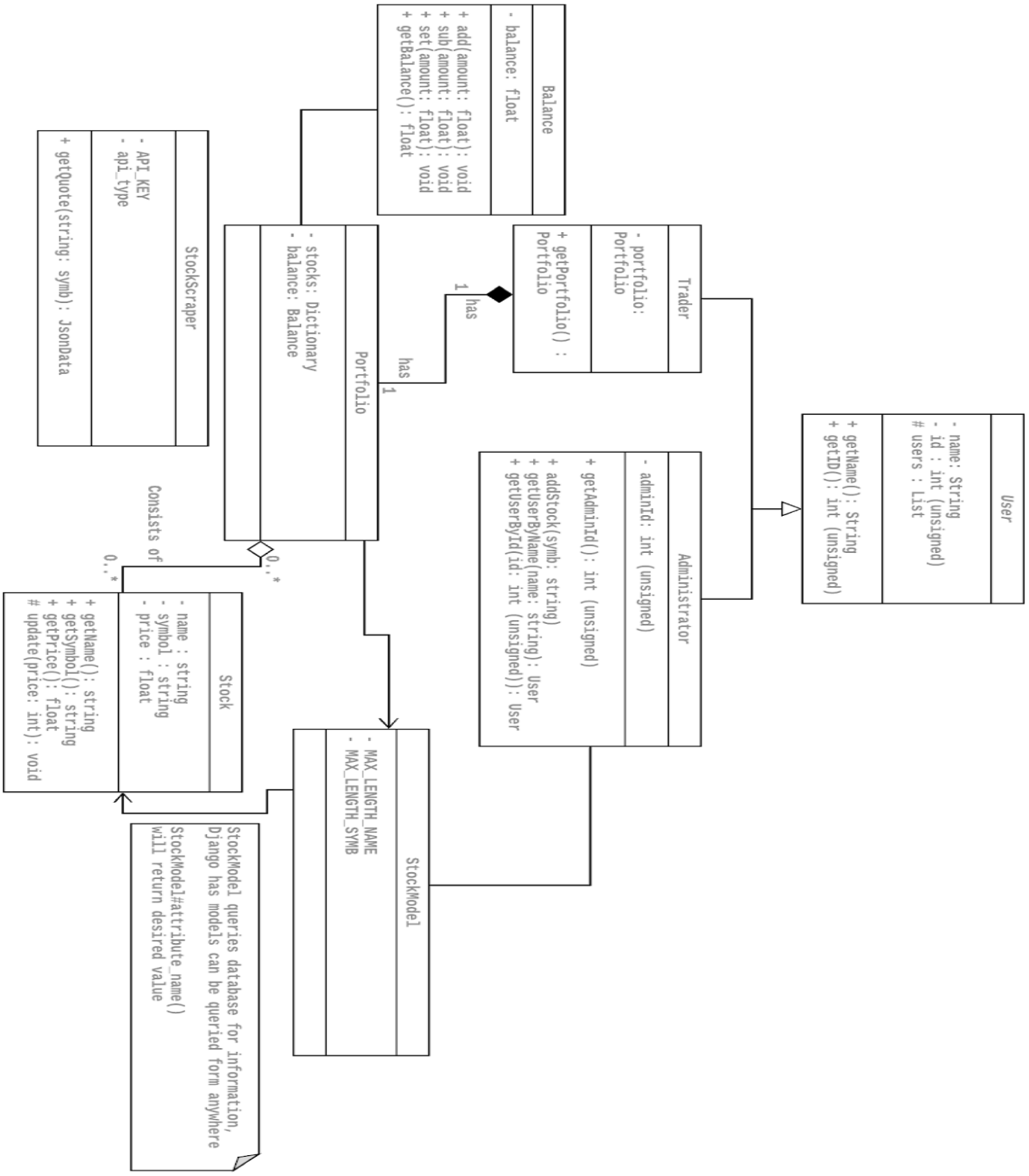
Original Class Diagram

We did not receive any feedback for our class diagram, however since starting to create this project, there have been many changes.

Most notably, understanding how Django models work has made the purpose of StockHandler change into a StockModel, which would essentially have the same purpose with slightly different functionalities. Additionally, this iteration was meant to create the smaller classes that are less dependent on the larger classes such as the Model classes or information scraper.


User
- name: String
- id : int (unsigned)
users : List
+ getName(): String
+ getID(): int (unsigned)
Trader
- portfolio: Portfolio
+ getPortfolio() : Portfolio
Administrator
- adminId: int (unsigned)
+ getAdminId(): int (unsigned)
+ addStock(symb: string)
+ getUserByName(name: string): User
+ getUserById(id: int (unsigned)): User
Balance
- balance: float
+ add(amount: float): void
+ sub(amount: float): void
+ set(amount: float): void
+ getBalance(): float
1
has
has
1
Portfolio
- stocks: Dictionary
- balance: Balance
- portfolioHandler: PortfolioHandler
+ getStocks(): Dictionary
+ getHistory(): Dictionary
+ getTotalValue(): float
+ getStockQuantity(stock: Stock): int
+ buy(symb: string, quantity: int): void
+ sell(symb: string, quantity: int): void
+ getPortfolioHandler(): PortfolioHandler
- addStock(stock: Stock): void
- removeStock(stock: Stock, amt: Int): void
StockHandler
- model: Model
+ verify(symb: string): boolean
+ createNewStock(symb: string): Stock
+ getExistingStock(symb: String): Stock
+ update(): void
+ update(symb: string): void
+ removeStock(symb: string): void
StockHandler queries database for information, Django has models can be queried form anywhere
Stock
- name : string
- symbol : string
- price : float
+ getName(): string
+ getSymbol(): string
+ getPrice(): float
# update(price: int): void
0..*
0..*
Consists of
PortfolioHandler
- model: Model
+ getUserInvestment(symb: string): int
+ getUserPastInvestment(symb: string): Dictionary
+ getUserPastInvestments(): Dictionary
+ addHistory(stock: Stock, price: Float, quantity: Float, buyOrSell: string): void
has
1
has
1
Display information likely to be handled with Django/html

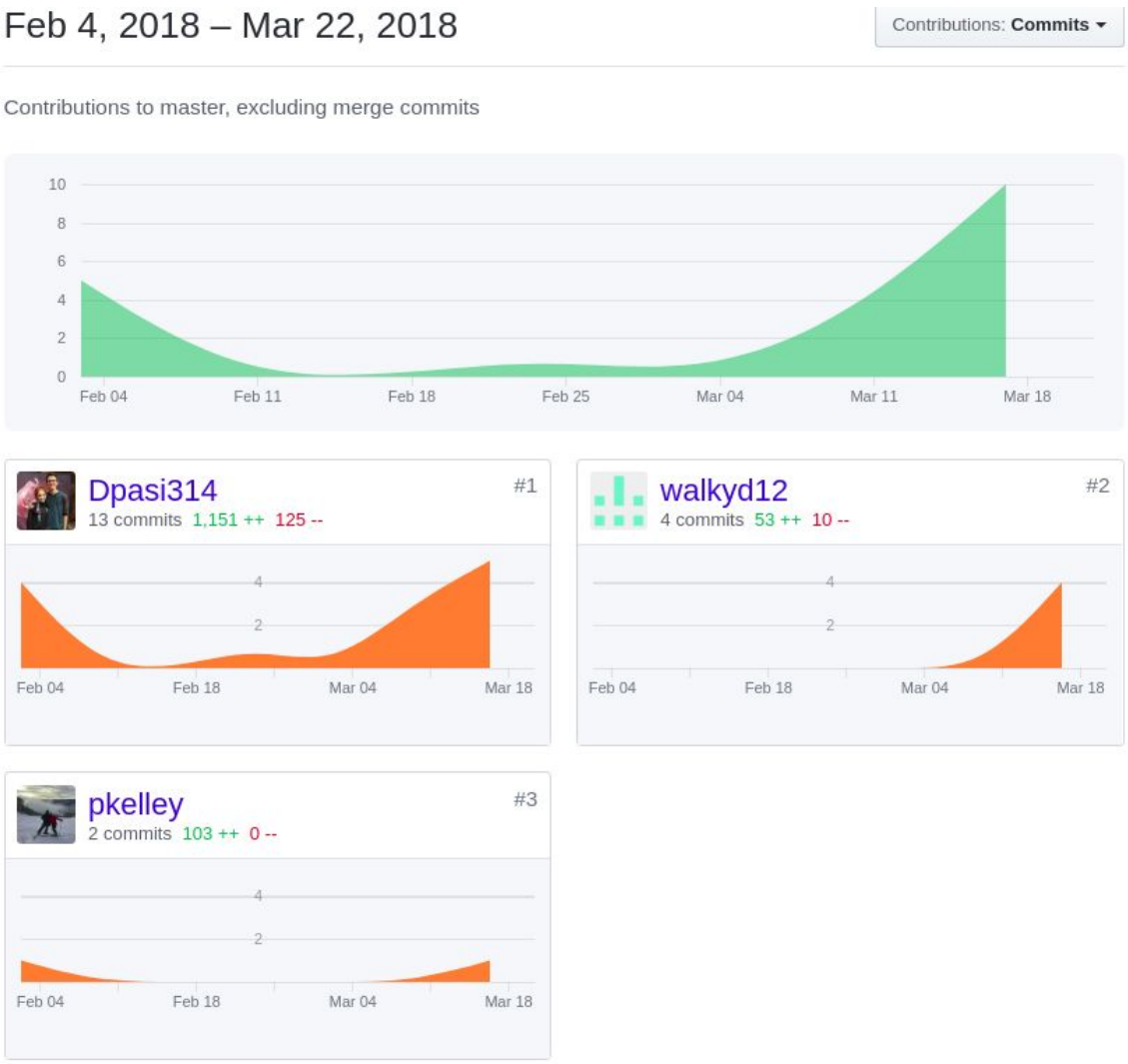
Updated Class Diagram



The Portfolio class has been created as a skeleton while the PortfolioModel is created. Additionally, the aforementioned StockModel class exists in the current working version of PaperTrader. The objects that the system will directly interact with (Stock, User, Trader, Administrator) have also been created. Finally, as shown in the class diagram the StockScraper class has been created; a rudimentary information gathering class that will be used to collect information about a given stock.

An important note is that there **is a floating class** (StockScraper) which has been created, and will be integrated in the next sprint. It was created now to help make integration into the system easier once StockModel is complete.

Current Contributions



Work Summary

The past few weeks have been dedicated to research the Django framework. Django Models provide us with essential functionality to easily query an SQLite database for information storage. Unlike normal objects, with a Model we're using object creation as a sort of query to get information returned to the client. Thus far, this project has implemented several key features. Firstly, the User abstraction was created, and the sub-classes Administrator and Trader that inherit it. Secondly, a generic StockModel was created for testing purposes (although has proved to me more difficult than originally thought. The Stock object was also created, similar to StockModel in the sense it will be involved in stock information, however the Stock object is what the system will interact with. Finally, the last major implementation was the rudimentary code for the StockScraper which uses the site Alpha Vantage to obtain stock information through an API request which returns JSON values to parse. Skeletons for most of the classes have been created at this point.

Current Contributions - By Member

Dante

- Created abstract User Class, designed Trader and Administrator subclasses. Pushed site scaffolding, created Balance, Stock, and StockScraper classes.

Walker

- Implemented Trader and Administrator classes using User abstraction. Created StockModel, designing PortfolioModel, created temporary filler while Models are being built

Preston

- Created TraderApp scaffolding, designing PortfolioModel

Future Contributions - By Member

Preston

- Troubleshoot current Django MVC problems

Walker

- Create PortfolioModel and integrate into system and Trader class

Dante

- Implement design patterns (Observer, Composite). Observer pattern will be used by the Portfolio and Stock object.

Estimated Remaining Work

Primarily, the rest of this semester will be spent implementing the StockModel, and PortfolioModels with Django. These are the cornerstones of functionality for this project, and will respectively handle Stock and Portfolio queries. Integrating the StockScraper with the PortfolioModel is another critical step that will need to be achieved. Once basis is complete, a rudimentary front-end will be created to help make the client interaction with the system less jarring than using a command line.

Test cases for the User subclasses has been created to ensure that each class is inheriting properly. Similarly, test cases for the StockScraper have been created to check accuracy of the data being returned to the program.

Next Iteration

By the time of next iteration, this project will ideally have the following modules completed:

- PortfolioModel
- StockModel
- Model integration in system
- Finalized version of StockScraper

With most of the small classes out of the way, these three objectives will take up the majority of time efficiently implementing .