What the Stock?!

Project Site: https://github.com/SkylerMalinowski/WBSF

Team Member Contribution Breakdown

Contribution Matrix

Section	Sub-Section	Pts	Jack Aquino	Raj Balaji	Gregory Leonberg	Jake Lewandowski	Skyler Malinowski	Vincent Taylor	Jonatan Yanovsky
Project Management [18]	Document Merge	11	0%	0%	50%	0%	50%	0%	0%
	Project Coordination & Progress Report	5	0%	0%	50%	0%	50%	0%	0%
	Plan of Work	2	4%	4%	40%	4%	40%	4%	4%
Interaction Diagrams [30]	UML Diagrams	15	20%	20%	0%	20%	0%	20%	20%
	Description of Diagram	15	20%	20%	0%	20%	0%	20%	20%
Classes & Specifications [10]	Class Diagram	4	50%	0%	0%	0%	0%	0%	50%
	Data Types & Operation Signatures	4	50%	0%	0%	0%	0%	0%	50%
	Traceability Matrix	2	0%	0%	0%	0%	100%	0%	0%
System Architecture & System Design [15]	Architectural Styles	4	0%	0%	0%	0%	0%	100%	0%
	Identify Subsystems	2	0%	100%	0%	0%	0%	0%	0%
	Mapping Subsystems to Hardware	2	0%	100%	0%	0%	0%	0%	0%
	Persistent Data Storage	4	0%	0%	0%	100%	0%	0%	0%
	Network Protocol	1	0%	0%	100%	0%	0%	0%	0%
	Global Control Flow	1	0%	0%	100%	0%	0%	0%	0%
	Hardware Requirements	1	0%	0%	100%	0%	0%	0%	0%
Algorithms & Data Structures [4]	Algorithms	2	0%	0%	0%	0%	0%	100%	0%
	Data Structures	2	0%	0%	0%	0%	0%	100%	0%
User Interface Design & Implementation [11]	What Has Changed	6	40%	0%	0%	0%	60%	0%	0%
	Ease of Use	5	20%	0%	0%	0%	0%	0%	80%
Design of Tests [12]	Test Cases	4	0%	100%	0%	0%	0%	0%	0%
	Coverage of Tests	4	0%	0%	0%	100%	0%	0%	0%
	Integration Testing Strategy	4	20%	0%	80%	0%	0%	0%	0%
	TOTAL	**100**	**14.28**	**14.08**	**15.00**	**14.08**	**14.40**	**14.08**	**14.08**

Figure 0.0 - Contribution Matrix

Contribution Pie-Chart

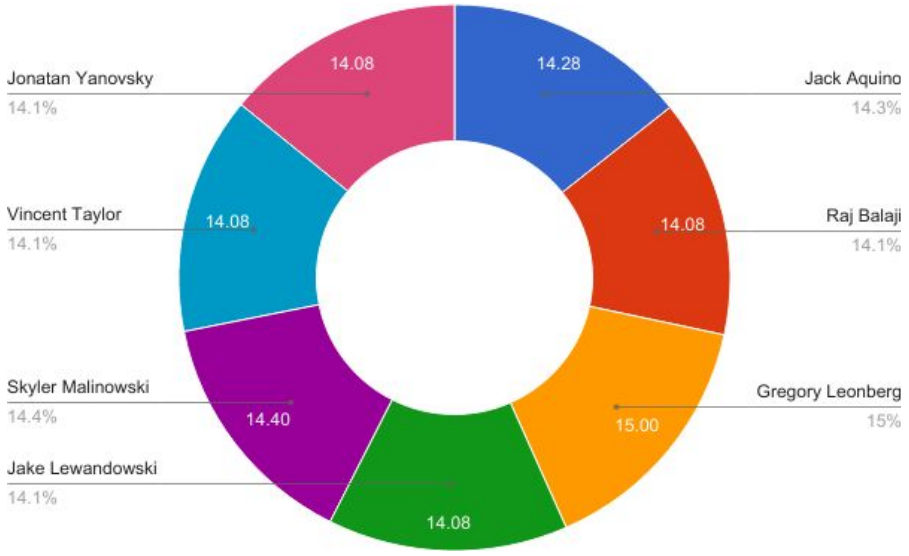


Figure 0.1 - Contribution Pie-Chart

Table of Contents

Interaction Diagrams

Use Case 1

Sequence Diagram

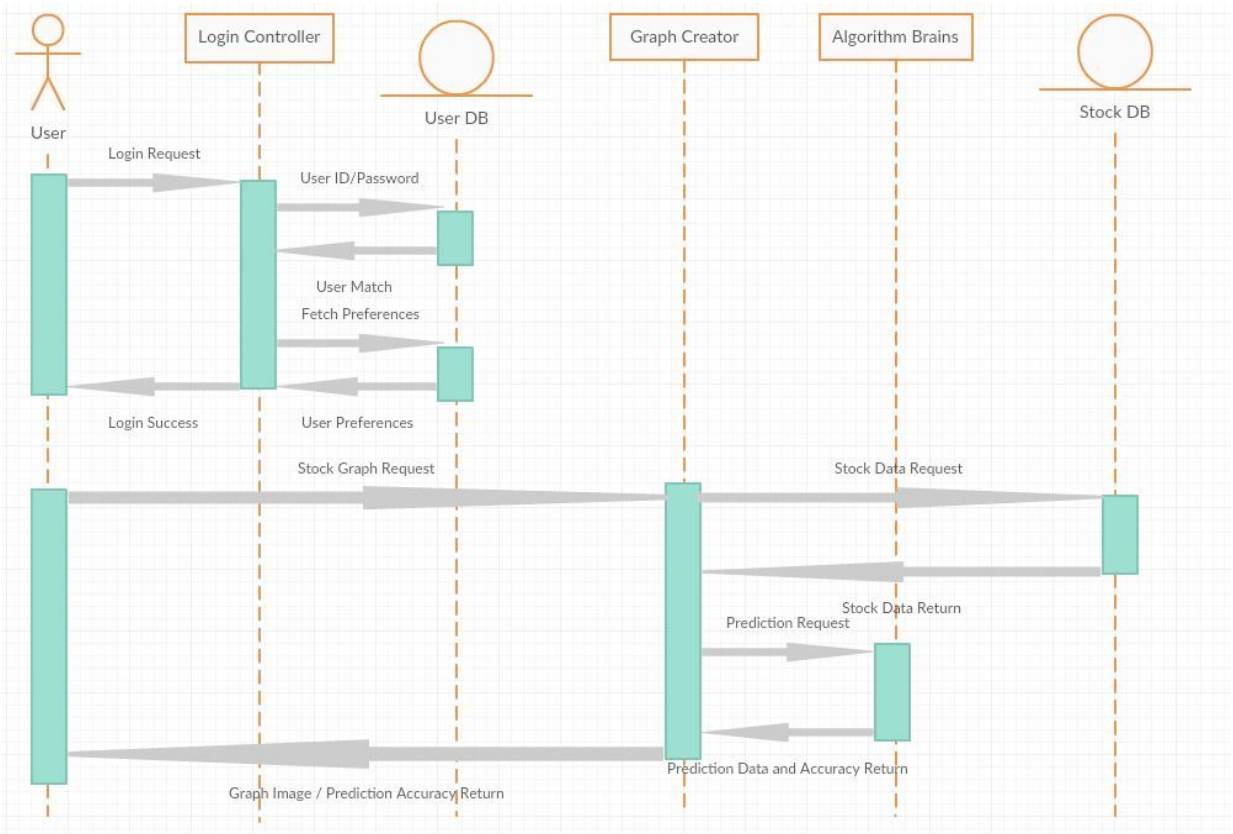


Figure 1.1 - UC1 Sequence Diagram

Explanation

For UC-1, the main modules are the graph creator, algorithm brains, and the stock database.

In this use case, we can assume that the user has already logged into the website, with the Login Controller and the User Database being utilized to fulfill that request. We can also assume that the graph creator pulls stock history and stock prediction data from the stock database, and uses those data points to create a graph. Thus, the graph creator “does things.” It can then pass the generated graph to the website interface to be displayed. The graph creator can also ask the algorithm brains to compute a certain stock. For example, a stock may have been recently added into the database and its future price was not predicted yet, then when the graph creator needs this data asap but the data does not exist yet, it can force the algorithm brains to give priority to this stock over the others in the prediction queue.

The algorithm brains compute future stock prices based on stock history. The accuracy of the prediction depends on how many price data points were stored in the stock history and the algorithm used. The algorithm brains module also has a priority queue to perform prediction on stocks one at a time, thus each stock can be assigned a priority so a more critical stock can be computed before others. Once the future stock price data has been computed, it will be stored into the stock database for future use and can be passed directly to the graph creator for displaying the newly calculated price values to the user. Clearly, the algorithm brains module “does things.”

The third module for this scenario is the stock database, which “knows things.” This module stores stock price history and predicted stock prices, and can be queried for stored data or have new data be saved into the database. The stock database interacts with the algorithm brains and the graph creator and can pass data between those modules.

Use Case 2

Sequence Diagram

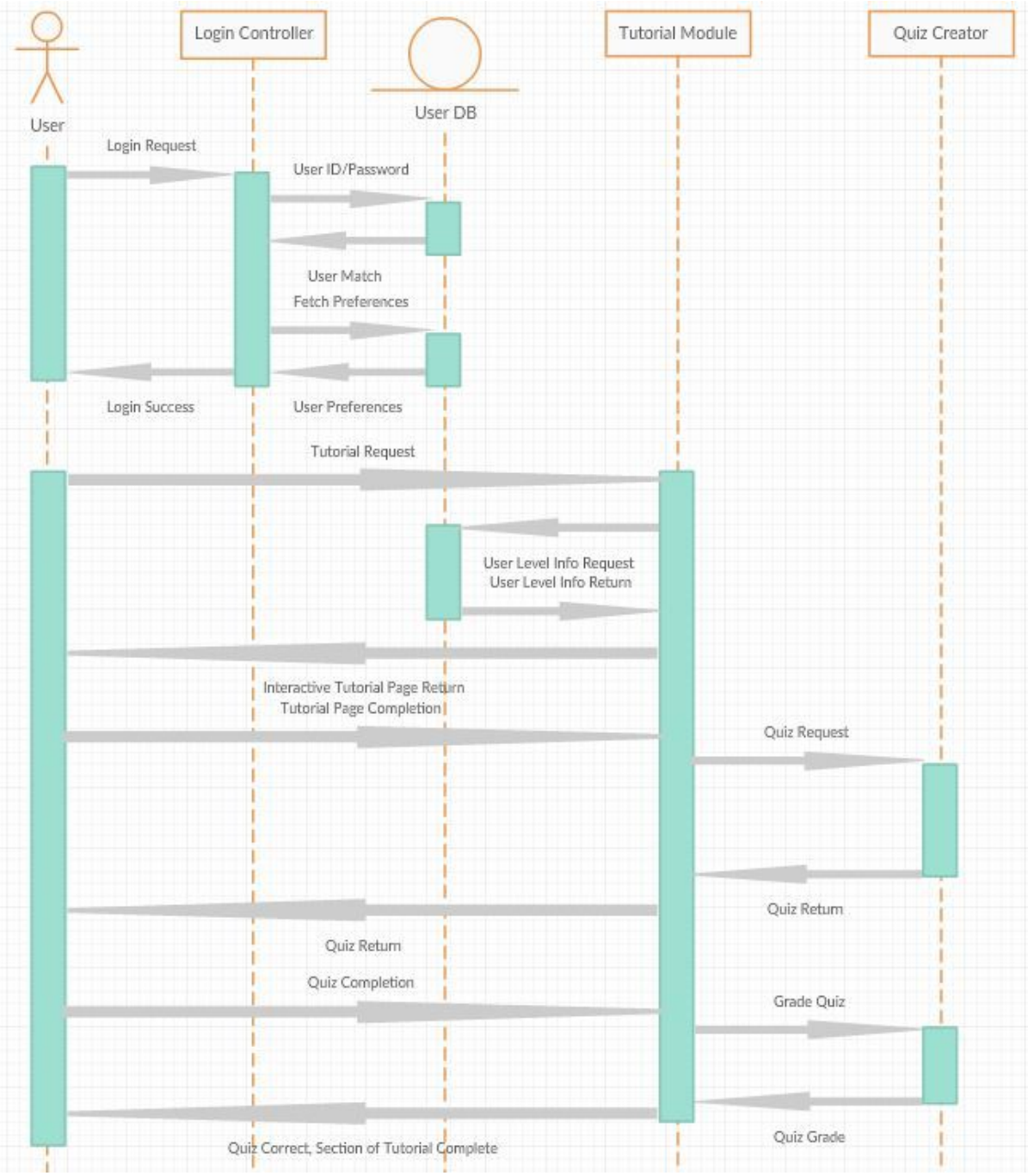


Figure 1.2 - UC2 Sequence Diagram

Explanation

For this UC, the main prevalent design principle is to separate responsibilities into “knowing” vs “doing.” When an entity has a responsibility in the “knowing” category, its job is to keep a record of data and make that data available when it is requested. For example, the User Database falls into the “knowing” category, as it stores the usernames, passwords, and preferences of each user. An entity in the “doing” category, on the other hand, involves taking data from the user input and from the databases and processing it, making changes and showing results depending on the data. The Login Controller is one such example of this, as it takes the user input (login credentials) and compares it to what is stored in the database, then loads the user’s preferences after verifying that the credentials are correct.

In this use case, the main players that are unique to this UC are the Tutorial Module and Quiz Creator. After the user has logged, the user may request to play through a tutorial. The User Database stores information regarding the User’s progress in the tutorial, which will affect what topics the user can have access to. For example, one cannot learn about short selling stocks before learning about buying stocks the normal way. If the User has not completed the “buying stocks” tutorial and quiz, the User Database knows that the User has not completed it, and the Tutorial Module will be responsible for not allowing the User to access the “short selling” tutorial. Thus, this ensures that the responsibilities of the Tutorial Module only include *doing* things with known data. Once completed, any section of the tutorial module may be repeated at the User’s will.

The Quiz Creator’s responsibilities involve *knowing* things more than they do *doing* things. The purpose of quizzes are to test the User’s knowledge of a certain subject after they have completed it’s tutorial. The Tutorial Module will ping the Quiz Creator, passing it a subject. The Quiz Creator will obtain a quiz about that subject, and return it to the Tutorial Module, which then gives it to the User to complete. Upon completion, the User passes back answers to the Tutorial Module, which gives the answers to the Quiz Creator. The Quiz Creator checks the User’s answers with the correct answers that the Quiz Creator knows, and then passes back a result (pass/fail, percentage correct) to the Tutorial Module. With this information, the Tutorial Module will decide whether or not to allow the user to access a new section.

Use Case 4

Sequence Diagram

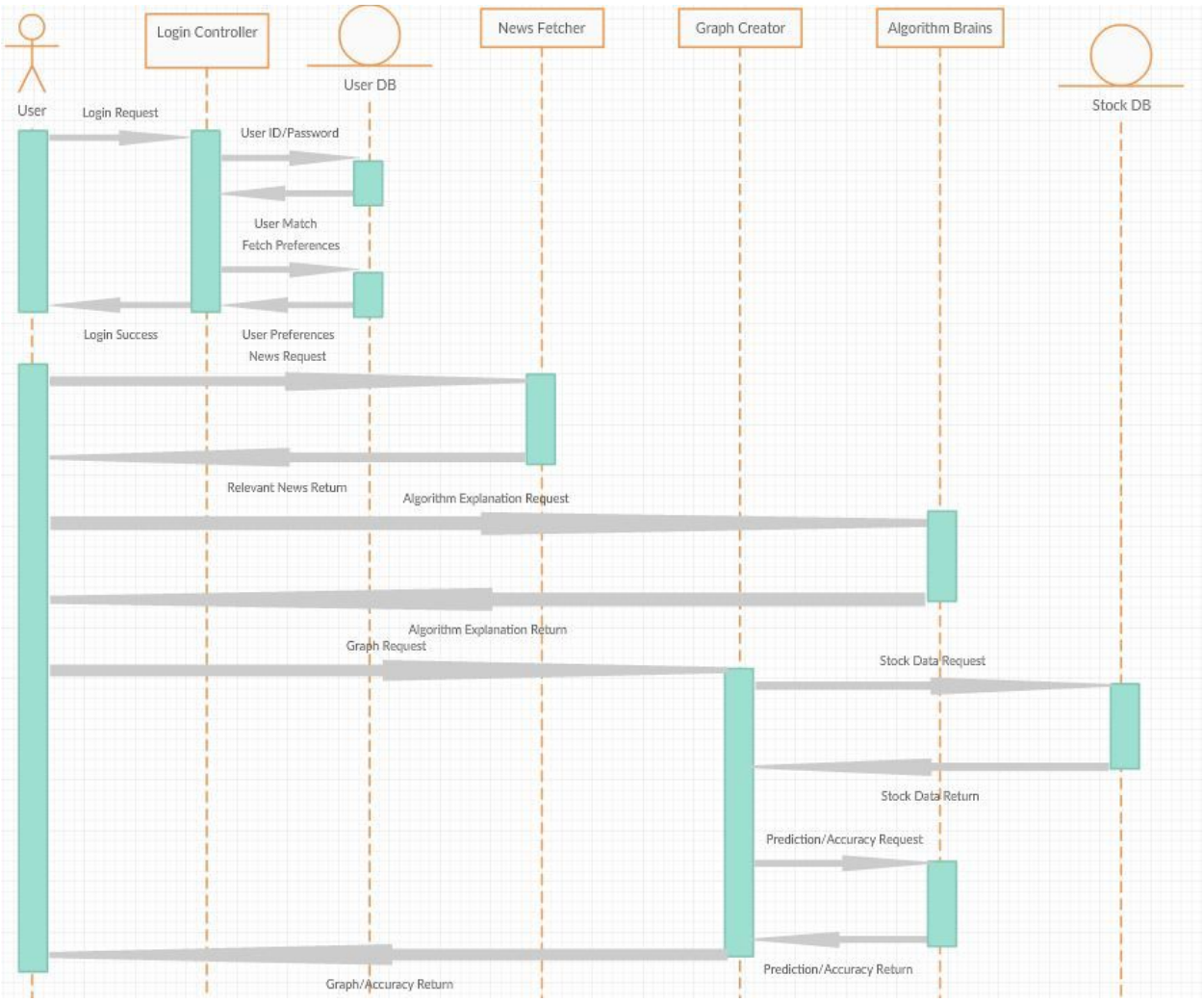


Figure 1.3 - UC3 Sequence Diagram

Explanation

This use case is accessed under the assumption that the user is logged in (therefore the sequence diagram does not need to show the login sequence). The user then requests information from one stock or a group of stocks as decided by the user. After the user requests the info the data about the company is fetched though google's newsfeed servers and displayed to the user.The system will display the information fetched by the news feed and display a brief amount of the article for the user to read to where the user can extend it if they please. The system will explain more about a selected stocks including the previous price information and the predicted action the stock's price will take. Additionally, the system will explain/ show the reason behind the algorithm's decision, this includes even a rudimentary answer. The system will most likely display the line of best fit to show the user the predicted path that the stock will take; thus, both explaining the reasoning and showing more information about the decision and stock. Finally, the Stock will update periodically (~15 minutes) this will allow for the google finance api to update and display new information. At the time of new information being displayed the algorithm will adjust; therefore, changing the prediction and the reasoning behind the change. With the polling of google finance the system will refresh the news feed and redisplay it for the user to see.

The design principle behind this is to make the responsibilities of the objects highly cohesive and specific to its own process. For example, the graph creator does not explain the process behind the making of the predicted graph, that's the algorithm Brain's work. Similarly the news fetcher doesn't tie into explaining either the algorithm or the graphs as the purpose of this education tool is to teach users how stock prices are affected by a company's financial decisions.

The Graph Creator has to communicate to the stock database to fetch the required data from the stock database. It then sends the fetched data to the algorithm controller to process the information. It then retrieves the processed information and displays it to the user. So the Graph Creator has the responsibility of that of a *communicator* as it interconnects 2 other processes.

The Stock database holds the stock values of various companies and updates itself with new data every 15 mins . Hence the stock database has the responsibility of *knowing* the data to quicken the process as retrieving the data from websites such as Google finances takes up time.

The Algorithm brains has to process the data that is sent to it from the Graph creator and then arrange the data such that it can be drawn in the form of a graph, and so the Algorithm brains is *doing* the computation for the prediction model of the stocks.

The news feed has 2 responsibilities in comparison in a relatively small scale, it *communicates* to the external news website a fetching request, it then *processes(doing)* the information by removing the advertisements and then displays the news to the user.

Class Diagram and Interface Specification

Class Diagram

Class Name -private attributes +public attributes (no variable types or parameters)

Back end classes (taken from report 1 concept definitions) (written in python and SQL):

LoginLogoutController
-username
-password
+login()
+logout()

GraphCreator
-stockQuote
-stockPredictionData
-stockHistoryData
-graphDuration
+generateGraphType()

PredictionAlgorithms
-stockQuote
-algoType
-stockHistoryData
-predictionDuration
+predictAlgoType()

PredictionAlgorithmController
-predictionQue
-userRequestedStock
+autoOrganizePredictionQue()
+forceStockPrediction()

QuizCreator
-previousUserHistory
-quizQuestions
-userFocusArea
+buildQuiz()

NewsFetcher
-stockQuote
-newsArticlePublicationDate
+fetchNews()

StockDatabase Controller
-dataToWrite
-stockQuote
-isPredictionData
+fetchStockData()
+writeStockData()

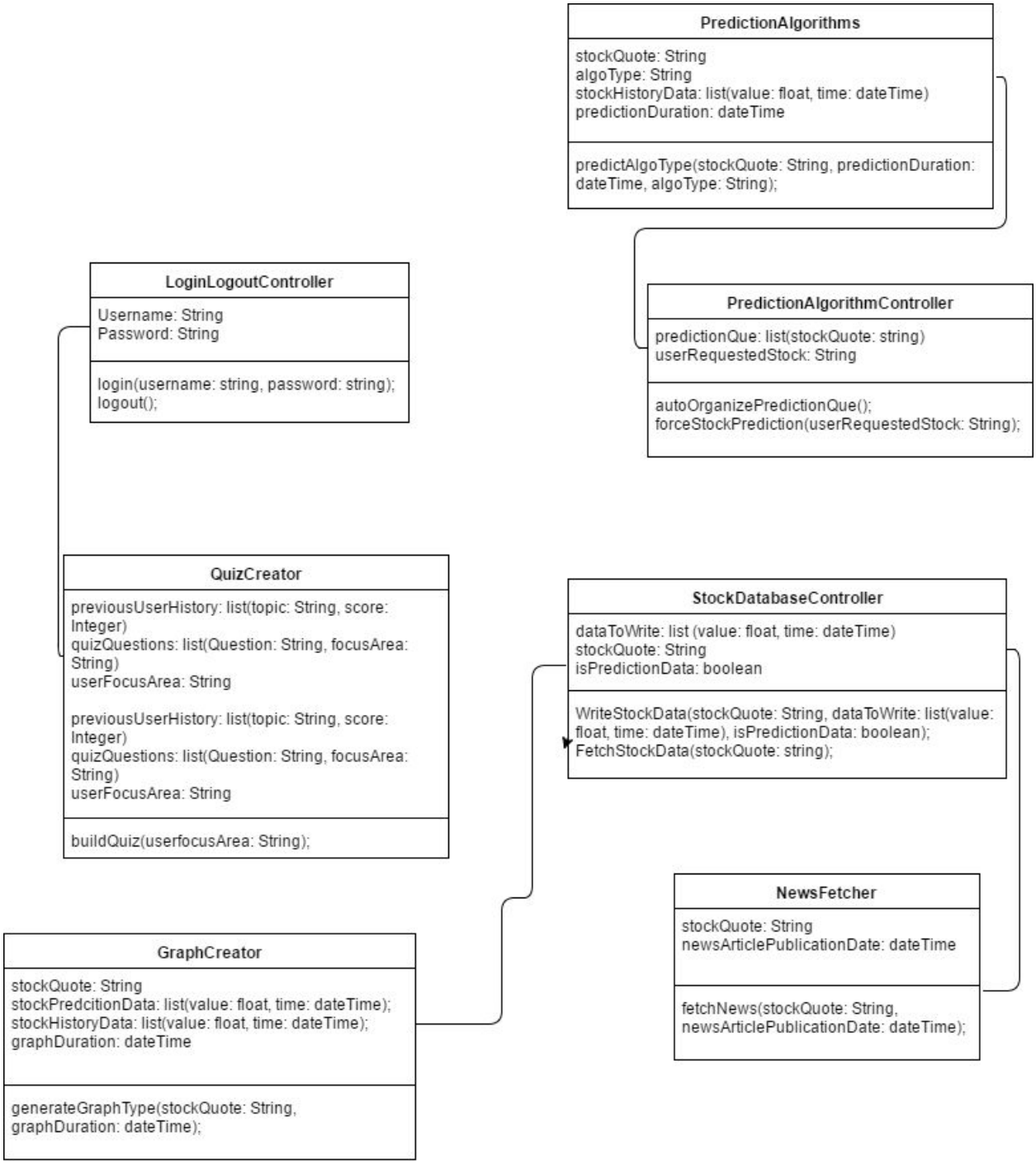


Figure 2.1 - Class Diagram

Data Types and Operation Signatures

Do only after listing the classes above.
For each (private) variable in the class diagram, list the data type and explain the variable.
For each (public) function, list return type, input parameters, and explain the function.

Variable types:
list(a, b, etc) = python list, containing list elements composed of the set of variables (a, b, etc)
normal types = string, bool, void, float, int
dateTime = python dateTime type, composed of date data concatenated with time data
image = data in an image format (jpeg, png, etc)

LoginLogoutController
-username : string - an identifier of a particular User Account.
-password : string - a verifier of a particular User Account.
+login(username : string, password : string) : bool - As input, the function will take a username and password pairing. The function will then check the User Database and see if it contains this particular pairing, and return true (success) if it is found, and false otherwise. Sets the particular User Account as logged on.
+logout() : void -sets the particular User Account as logged off.

GraphCreator
-stockQuote : string -A ticker symbol for the particular stock
-stockPredictionData : list(float value, dateTime time) - The future, predicted value of a stock over “time”
-stockHistoryData : list(float value, dateTime time) - The actual value of a stock over “time”. Will have undefined values for future time slots
-graphDuration : dateTime - A slice of time, the x axis of a graph.
+generateGraphType(stockQuote : string, graphDuration : dateTime) : image - takes the input ticker symbol and produces a historical or prediction graph over the time range specified by graphDuration.

PredictionAlgorithms
-stockQuote : string - Ticker Symbol
-algoType : string - which prediction algorithm to use
-stockHistoryData : list(float value, dateTime time) - the historical (actual stock price) value of a stock over a time specified by “time”
-predictionDuration : dateTime - a section of time over which to apply the prediction algorithm. Note that this input should not only include a future time period, but a past time period in order to get prediction accuracy
+predictAlgoType(stockQuote : string, predictionDuration : dateTime, algoType : string) : list(float value, dateTime time) -takes a stock ticker, a prediction algorithm to use, and a prediction duration and returns a value of predicted values of the stock over the time value “time.”

PredictionAlgorithmController
-predictionQue : list(stockQuote : string)
-userRequestedStock : string
+autoOrganizePredictionQue() : void
+forceStockPrediction(userRequestedStock : string) : dateTime // returns the expected time for stock prediction algorithms to predict all stocks in the queue ahead of the requested stock

QuizCreator
-previousUserHistory : list(topic : string, score : int) - stores the scores that the User obtained on the quizzes/tests for specific sections in (<topic>,<score>) pairs
-quizQuestions : list(question : string, focusArea : string) - each question in our quizzes will be a string. We will maintain a pool of questions that this QuizCreator can pull from to create a quiz for a user, and each question will have a focusArea associated with it (e.g. “What is a stock” is part of focusArea “stocks”).
-userFocusArea : string - a subject, what the User wants to study in a particular learning session
+buildQuiz(userFocusArea : string) : list(question : string) -takes the userFocusArea input and returns questions that pretain to that subject.

NewsFetcher
-stockQuote : string - ticker symbol
-newsArticlePublicationDate : dateTime -a valid date range for news articles (i.e. “I want news articles in the dates specified by newsArticlePublicationDate”);
+fetchNews(stockQuote : string, newsArticlePublicationDate : dateTime) : list(htmlLink : string)
-returns a list of links to news articles about the specified stock, published within the time window specified.

StockDatabaseController
-dataToWrite : list(float value, dateTime time)
-stockQuote : string
-isPredictionData : bool
+writeStockData(stockQuote : string, dataToWrite : list(float value, dateTime time), isPredictionData : bool) : void
+fetchStockData(stockQuote : string, isPredictionData : bool) : list(float value, dateTime time)

Traceability Matrix

Domain Concept: Login Controller; Logout Controller
Derived Class: LoginLogoutController

Explanation: Because of the implementation, the two listed domain concepts derived a single class to take care of both functions outlined. The name is a hybrid of the two domains the class was derived from.

Domain Concept: Graph Creator
Derived Class: GraphCreator
Explanation: A well defined task grouping, as outlined in the domain concept, are taken care of by this directly derived class. There was no need to alter its name as a result.

Domain Concept: Prediction Algorithms
Derived Class: PredictionAlgorithms
Explanation: A specific task grouping, as outlined in the domain concept, are taken care of by this directly derived class. There was no need to alter its name as a result.

Domain Concept: Quiz Creator
Derived Class: QuizCreator
Explanation: A specific task grouping, as outlined in the domain concept, are taken care of by this directly derived class. There was no need to alter its name as a result.

Domain Concept: News Fetcher
Derived Class: NewsFetcher
Explanation: A well defined task grouping, as outlined in the domain concept, are taken care of by this directly derived class. There was no need to alter its name as a result.

Domain Concept: Stock Database; Stock Database Controller
Derived Class: StockDatabaseController
Explanation: How we are implementing reading and writing to the stock database called for this derivation of a class that that draws from two similar domains.

Domain Concept: Algorithm Controller
Derived Class: PredictionAlgorithmController
Explanation: A well defined task grouping, as outlined in the domain concept, are taken care of by this directly derived class. There was no need to alter its name as a result.

System Architecture and System Design

Architectural Styles

We are going to use a multitier architecture design. This design will revolve around the user of a three tier system. Essentially this will separate the UI, Logic, and Data portions of our project. This mimics our intended team layout towards completing the project by breaking down teams to handle each of these modules separately. Each of these modules will be completed on separate platforms; thus ensuring both longevity and upgradability for the future. This approach incorporates the use of multiple platforms for development. This way in the future parts of the project are upgradable or even replaceable. In the case of language upgrades or OS changes.

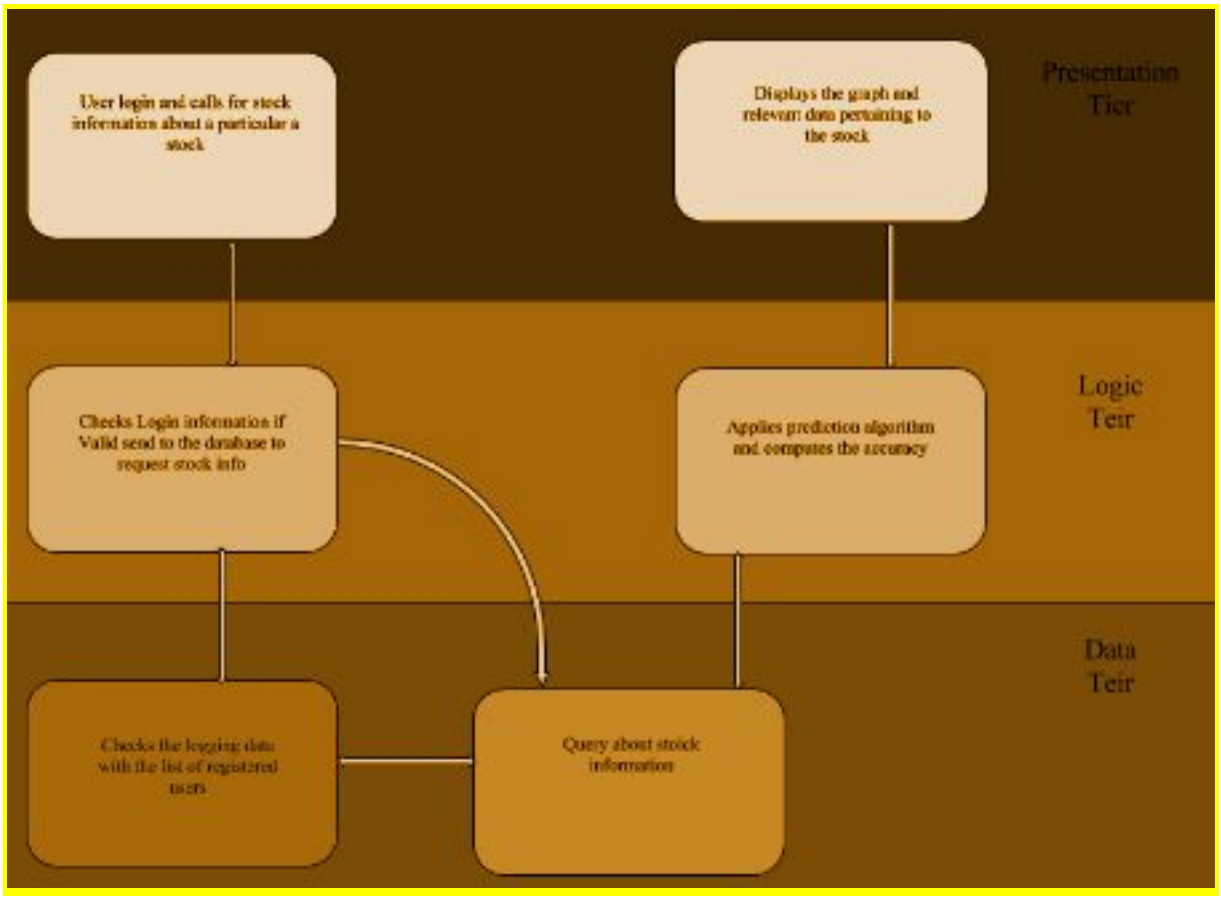


Figure 3.1 - Architectural Diagram

Presentation Tier:

This tier handles/represents the UI of the system, including graphs, weblayout, login screens, search bars and news feeds. This portion of system is the only part the user can access the program directly. Accordingly, the presentation tier communicates with the other tiers in order to exchange information between the data and logic tier for the user to manipulate indirectly. Finally, the data from both the database and prediction algorithms needs to be translated into understandable information; thus this tier must handle that transformation.

Logic Tier:

This tier pulls information from the presentation tier and does most of the computing and heavy lifting of the program. Mainly, this tier will focus on the prediction of stocks and the user login and registration. Along with prediction, the logic tier will act as a way to communicate quiz grades, tutorial points and user's request to the database. In other words, the logic tier ,in addition to doing heavy computing, will act as a middleman and exchange data between the database and the presentation tier. The logic tier will also call other apis from google to get news feed information to relay to the presentation tier

Data Tier:

Data consist of user records, grades, stock tables and information, as well as additionally data required for data processing. Users should not be able to directly access this information without going through the logic tier. In the case of our program the user's search regarding a stock will first be processed through the data tier. This will provide an api for the logic tier to user for the retrieval of stock information as well as the retrieval of account information. The data tier will store and manipulate data by itself to limit its dependence on the logic tier thus the creation and deletion of data will primarily be in this tier. This tier will primarily call the google and yahoo finance api's to organize and gather data for future use.

Identifying Subsystems

Our software solution uses the Model-View Controller approach, as well as the Client-Server approach. The Stock data and data manipulation are handled by the server machine, and the client machine deals with calls to the server as well as data visualization. The server acts as the model and controller and the client acts as the viewer.

Database System

Functions include

1. Database Management: Stores the hundreds of information about the stocks in relation to their companies.
2. Algorithm storage: Stores the logic behind the algorithms
3. User Database: Stores the user(s) information.

The server system is where Data is generally stored, and updates itself with new data from the internet/ application managers at regular intervals.

The server system communicates directly to both the Client and the Processing subsystems.

Client System

Functions include:

1. User Interface: Retrieving commands from the user and displaying the text in a GUI.
2. News Fetcher: Accessing news website for latest financial information.
3. Stock Ticker: Accesses current information for a single stock.
4. User Preferences: User settings for options such as color palletes.
5. User Favorites: A list of stocks a user wants to track.

Processing System

This handles the tasks of actually compiling the data into a more visual form for the user to understand.

Includes:

1. Graph Creation: The values of the stocks are sent to the system in a numeric form which is then processed as a graph and sent back to the client side
2. Quiz Creator: Creates a quiz to test the user's information on how stocks works based on their preferences/ area of interest.

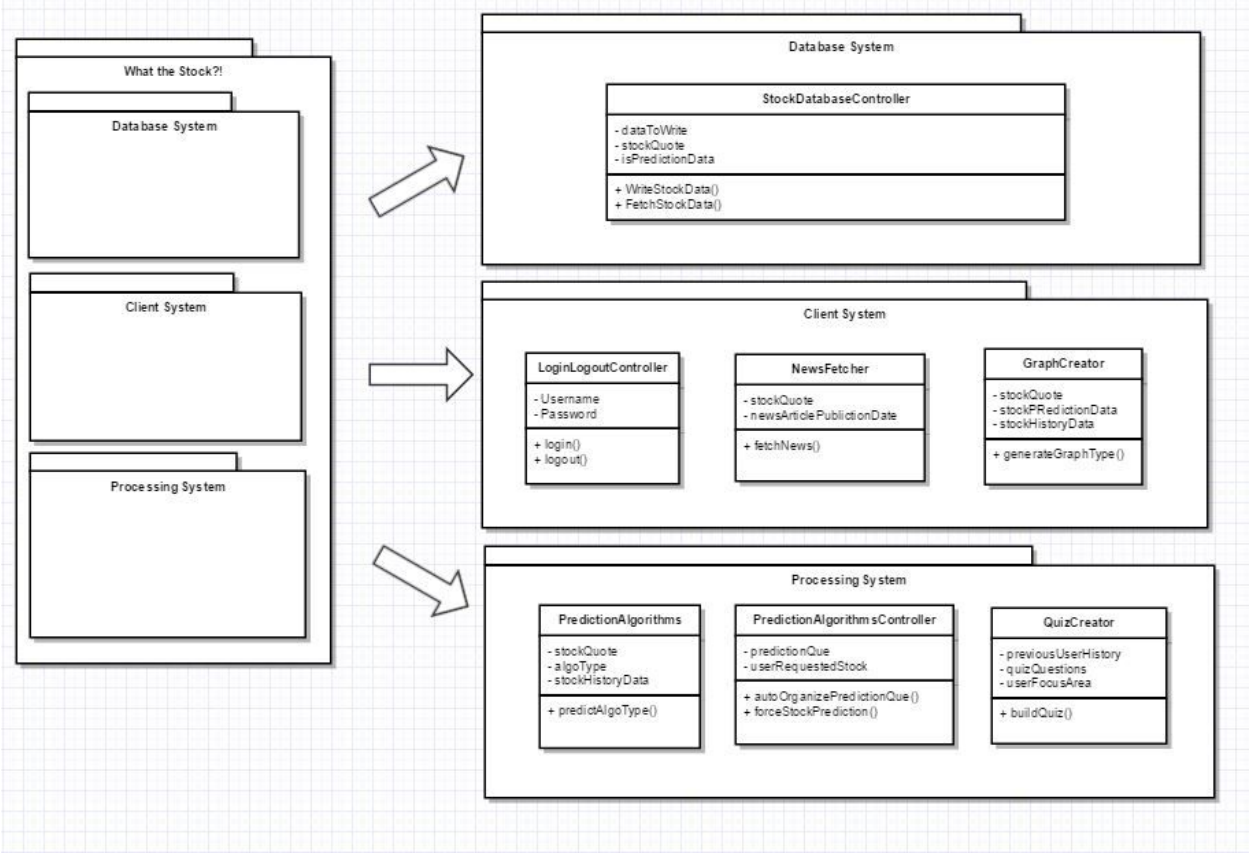


Figure 3.2 - UML Package Diagram

Mapping Subsystems to Hardware

Our system needs to run on at least 2 different machines, with a database to store and process the information from the user and another to host the website.

The client side machine will be either Local host and then at later stages it will be on a dedicated machine that responds to the user when they access our application through the internet.

The server side machine has more processing intensive tasks and as such will be run on a more powerful machine as opposed to the machine that handles client side.

The server side machine comprises of both the Database and Processing subsystems enclosed inside it.

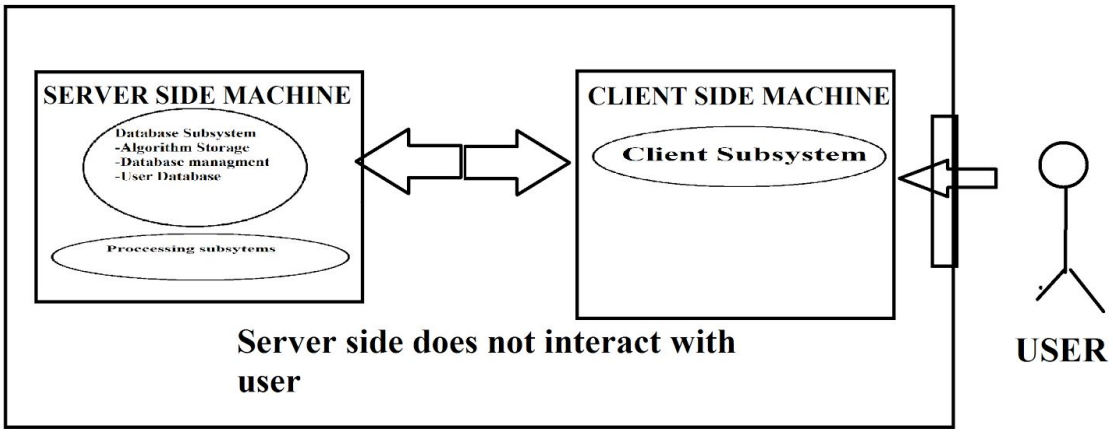


Figure 3.3 - Subsystem to Hardware Mapping, Visual Representation

Persistent Data Storage

Our system requires data to be saved past one execution.Persistent objects in the system include: user account information, past trends for all stocks, past news for all stocks, relative information for all stocks and user previous searches.

Considering the size of the data we are required to store, a relational database is the more logical option for our storage management. This will also be a simple way to store our data, as each instance can represent a single stock or a user's account info. Each instance has many attributes as well, so using flat file as storage would be a real hassle.

Most of the database schema will be pretty similar, since they will contain similar attributes. The attributes involved include: the stock ID, stock ticker, username, password, followed stocks, and many more which can be seen below in the relation models.

Stock ID	Company	Ticker	Current Value
XX-XXX	Apple	AAPL	139.78

Relation model for any given stock

User ID	Name	Username	Password	Followed Stocks
XX-XXX	John Doe	johndoe1	john1	Apple, Google, etc.

Relation model for user account information

Stock ID	Company	Company	Past Value	Date
XX-XXX	Apple	AAPL	130.21	2/5/2017

Relation model for stock history

Network Protocol

In order to interface between the JavaScript code in the end user's web browser and the data and processing engines on the server side, a series of AJAX requests using the jQuery library are made in order to pass along function calls and receive results. The communication is asynchronous and event driven.

Global Control Flow

The software system is event driven. It waits for users to trigger a change in status via clicking various links and buttons. When requests are made, they are processed and handled at that time.

There are no timers in the system. Even the updating of the stock data is event driven and triggered by user requests.

The system is an event-response system, though it has basic time constraints in order to maintain an illusion of real-time behavior. No task should take longer than 2-3 seconds, so as not to inconvenience users. Other than this minimum level of execution performance, no definite constraints exist.

Each core subsystem, such as graphing, prediction, and user login runs as a separate process on the same system. Each of these processes only uses a single thread of execution. Due to the lack of shared resources between them, there are no concerns in regards to race-conditions, synchronization, or resource locking.

Hardware Requirements

This software package has two sets of hardware requirements: one for the server device and one for client devices.

The server device must have at least a 300 MHz processor, 192 MB RAM, 1 GB of storage, and an internet connection with a speed of at least 1 MB/s. It will run a CLI only server distribution of Ubuntu Linux with Python and SQLite installed.

The client device must have a color display with a minimum screen resolution of 640x480, a connection speed of at least 256 KB/s, 512 MB of RAM, 5 GB of storage, and a processor capable of supporting the SSE2 instruction set. It must also be capable of running a full multi-tasking operating system with a graphical user interface.

Algorithms and Data Structures

Algorithms

LSA (Least Square Approximation)

About

Least Squares approximation fits both linear and nonlinear equations to plotted points/sets of data. The combination of different observations as being the best estimate of the true value; errors decrease with aggregation rather than increase. As more points are added to a model the predicted line becomes more accurate in predicting the behavior of the data as time increase (as long as the data is modeled within the same domain)

Method

The method utilizes regression analysis.

$$\hat{a} = \frac{n\left(\sum_{i=1}^{n} x_i y_i\right) - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{n\left(\sum_{i=1}^{n} x_i^2\right) - \left(\sum_{i=1}^{n} x_i\right)^2} \qquad \hat{b} = \frac{1}{n}\left(\sum_{i=1}^{n} y_i - \hat{a}\sum_{i=1}^{n} x_i\right)$$

Where X and Y are data plotted points with each “i” pertaining to a certain data
Point and the general equation of a line is y = ax+b.

For a more general approach the use of matrices to fit into cubic cor higher functions. Plotting the line of best fit through least square approximation. Depending on what part one observes our equation will give us the output of a matrix $[a_0;a_1;a_2]$. a_0 is c in aforementioned part one, a_1 is b, and a_2 is a. These values are computed by solving a transformation matic $(C^TC)^{-1}C^Ty$. Where y is the plotted points on the y axis that we choose to take depending on part 2. These points act as the “solution”. The next step is to create the C matrix or transformation matrix. That matrix consists of three vectors.

$V_1=[1;1;1;1\ldots1], V_2=[x_1,x_2,x_3,x_4 \ldots\ldots x_n], V_3=[x_1^2,x_2^2,x_3^2,x_4^2 \ldots\ldots x_n^2]$ …
$....V_m[x_1^m,x_2^m,x_3^m,x_4^m \ldots\ldots x_n^m]$

General Equation:

$$\sum_{j=1}^{n} X_{ij}\beta_j = y_i,\ (i = 1, 2, \ldots, m),$$

General Vector form for data input:

$$\mathbf{X} = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1n} \\ X_{21} & X_{22} & \cdots & X_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{m1} & X_{m2} & \cdots & X_{mn} \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}.$$

Such a system usually has no solution, so the goal is instead to find the coefficients which fit the equations "best," in the sense of solving the quadratic minimization problem

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\arg\min}\, S(\boldsymbol{\beta}),$$

The objective function of S is given below

Transformation Matrix is given below

The output of the upper transformation and the X matrix equates to the coefficients of the equation of best fit

Lines of best fit plotted over time:

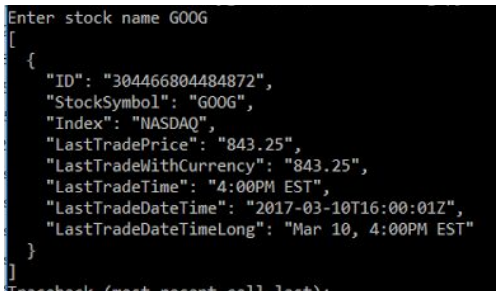
Figure 4.1 - LSA Sample Analysis

FIgure 4.1 shows the progression of the algorithm applied overtime as more data points are added the gray line becomes quickly outdated and incorrect. As seen above the gray line mimics the beginning have of the stocks progression from September of 2005 to about November 2006. As time progresses the grey the function still shows the general slopes and pivots in the graph estimating when a graph makes a drastic drop or increase. As more data points are presented the algorithm is reapplied;thus, increasing the accuracy of the dips and rises of the graph and overall averaging out the short term rises and fall presented by “sporadic” data.

Data Structures

The Data structures are going to be dictionary data structure the vector data structure provided by VPython and the python's numpy.linalg built in data structures. Many of the functions included in vector provide the necessary calculations to generate and multiply vectors. Numpy.linalg provides ample coverage of more complex matrix calculations such as inversion, solving linear matrices and approximation linear least square solutions. The dictionary function allows us to organize and split data received from JSON files. These JSON files are provided by the Yahoo Finance API and Google API. Having the freedom to return multiple inquiries and variables via functions allows us to send year's worth of data through various functions. The numpy vector data structures provide ease needed to solve least square approximation problems for prediction.

Example output for Google API:



Example Output for Yahoo API: (Section Taken From 2014-2015)

User Interface Design and Implementation

What Has Changed?

At the time of writing, we have currently implemented part of a Learning System that will contain educational material regarding the Stock Market. The Learning System would be accessed from a “Main Menu” part of the app, and would allow the User to select specific topics to study. The user can read a selection of material, as well as take a quiz on said material.

While we were always aware that the quizzes would be a multiple choice format, we had to decide on how to present the actual material of the topics. We debated between three different ways to present it: in a Slide format; in a Visual Novel format; or in a One-Page Scrolling format.

In a Slide format, the material would be placed in a form similar to PowerPoint slides, and the user would move from one slide to the next using mouse clicks. The benefits to this format is that since our group is familiar with how slides should be formatted, it would be the easiest to implement. However, the drawback is that it is hard to find a balance between having a lot of detail, and have it not be boring. If we wanted to include a lot of information (i.e. explanations and diagrams), we would either have to use walls of text (which are boring), or a large amount of slides (which would increase the user’s effort, especially if they wanted to look back at previous bits of information).

A Visual Novel is a type of Japanese interactive game using static graphics and small chunks of text to tell a story. One idea that we had was to implement our material in the form of a visual novel. We would create a mascot, such as a teacher character, and have this mascot “teach” the User the material. The amount of text that the User would read would be the same (if not more, to incorporate “realistic” dialogue), but the theory, the immersive experience created would made the User feel more involved with a character talking to them. The idea that we thought of was meant to invoke the feeling of having a teacher explain an example to the User, which was a better experience to having the User read a textbook — the User would retain more content in by not losing interest as fast by being enthralled and or immersed.

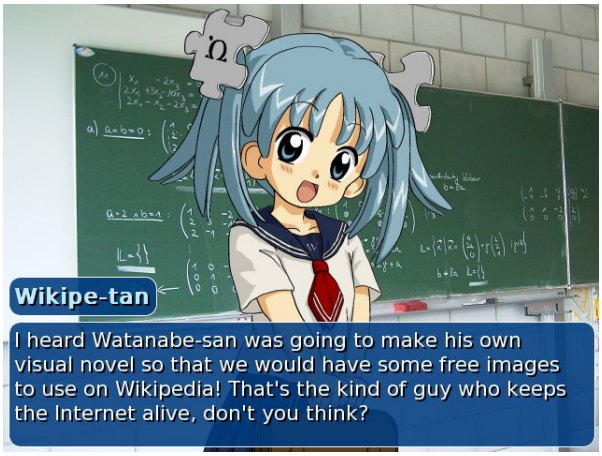


Figure 5.1 - Sample Visual Novel Screen

Above is an example of a Visual Novel's interface. The main points of interest are the human character (Wikipe-tan), the relatively bite-sized chunks of text, and the illusion that the character is speaking said text.

The One-Page Scrolling format is a new age style for webpages. It uses the design philosophy of maximizing information and minimizing page redirection. Because of this, the use of color is important to segment information for the user rather than physical segmentation by using a different pages. This in turn means that the user's attention can be kept longer and information retention is increased. Other benefits are: low programming overhead implement; walls of text are broken down into chunks; other visuals can help to enhance the user experience; and vertical scrolling is the easiest motion on any platform (mouse or touchscreen).

We explored the possibility of using a Visual Novel Engine; we were unable to find one that would easily allow us to integrate everything online. Rather than start developing our app in a VN format and risk being unable to integrate each sub-team's work into one cohesive project, we decided to instead opt for the One-Page Scrolling format for delivering information. The Slide format is effective but we believe that the One-Page Scroll format is superior on several fronts including low programming overhead and easiest controls for all platforms.

There has been some idea refinements made to the User Interface screens. Moreover, the main menu screen and the three subscreens that get directed to via buttons labeled "Learn", "Search", and "Portfolio".

Below (figures 4.2) is of the main menu screen: figure 4.2a is for desktop screens; and figure 4.2b is for phone screens. The Search box is not a button, rather an input box for the user.

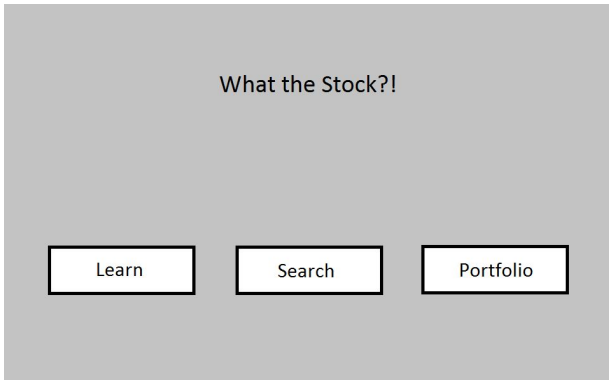


Figure 5.2a (Desktop)

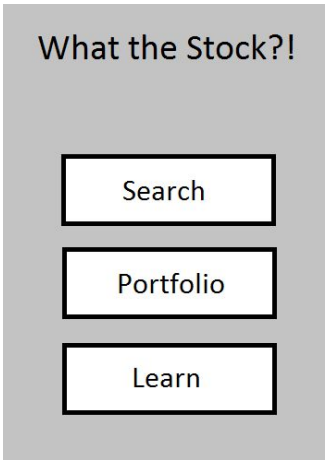


Figure 5.2b (Phone)

Below (figures 4.3) is of the search screen. The “Search” box is for user text input of stock symbol. The box under is the stock ticker. Below that is a graph of the stock, or portfolio, showing the historic data, the present day line, and the prediction given what prediction algorithm was chosen. There is an option button attached to the graph to alter what is being displayed.

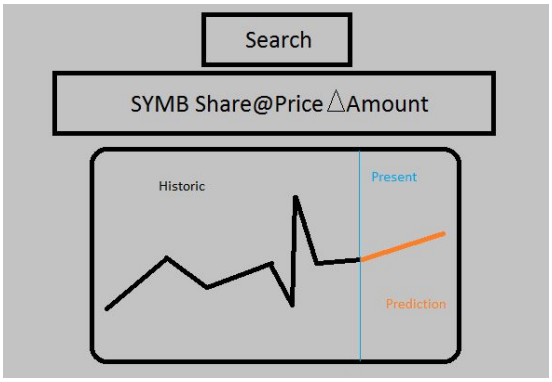


Figure 5.3

Below (figures 4.4) is of the portfolio screen. This shows both the current stocks that are in a temporary portfolio for that session and the stocks that the user has saved into their portfolio that will persist after the session. The boxes below the header displays the ticker symbol and the name of the company. Pardon not putting placeholder text in there.

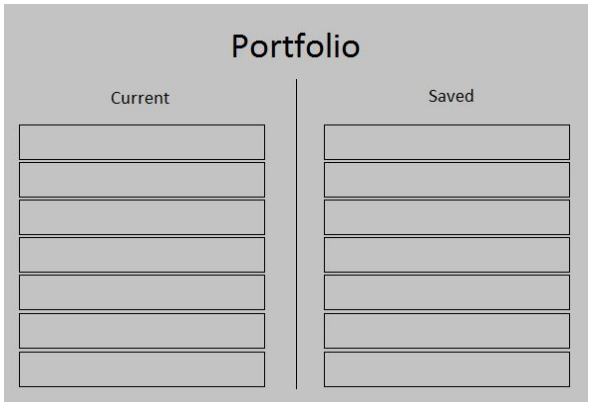


Figure 5.4

Below (figures 4.5) is of the learn screen. The “Take Placement Test” button leads directly into the placement test; until taken, the lessons are locked. The boxes below that are the lessons by name — here “Lesson 1”, “Lesson 2”, are placeholders.

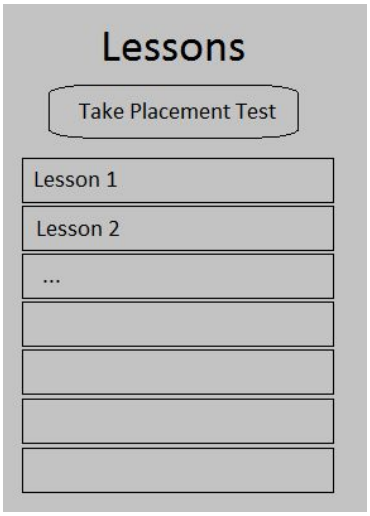


Figure 5.5

Ease of Use

By the time our app is finished, the end goal is for the user to be able to understand it without a large amount of effort. We have to ensure that our app is not mentally or physically taxing to use. We start our easy design by implementing a simple main menu page, consisting of three buttons. In order to progress to a different part of the app, all the User would need to do is one simple click. The buttons will have labels that say where the User will go after they've clicked on it, so it should not be that mentally taxing either.

When deciding on how to present our slide information for the Learning aspect of the app, we found that One-Page Scrolling would present the information in both a meaningful and non-taxing way. While Powerpoint included too many clicks, and the Visual Novel format was unfeasible to implement given our time frame, the One-Page Scrolling will allow the User to traverse through the slides by using the mouse scroll wheel, or simply the up and down arrow keys. Compared to navigating between material using "back" and "next" buttons, One Page Scrolling has the benefit of the User not needing to move the mouse cursor. From our personal experiences with reading online material, both online and from other classes, the information is easier to absorb if the controls needed to navigate through said material are minimal.

In cases where we need to obtain user input, it will be designed in such a way that the User should not have to think too hard to input data. Most aspects of the app can be reduced to a "point-and-click" model, where a majority of the user inputs will be in the form of mouse clicks. The only aspects which will require keyboard inputs will be for the login sequence (username and password), as well as inputting ticker symbols.

Design of Tests

Test Descriptions & Coverage of Tests

Logging On

For a user to be granted access to the system, the user must log in first. For a log on to occur the user must successfully enter a correct combination of username and password. Then, this username and password is sent to the user database to look for a match. If the combination is correct, the user database will allow the user access to the system.

Testing

For testing the logon system, the main concern is to see if the user's input is received and sent correctly. So naturally, the first aspect we will be testing is that the username and password that has been inputted by the used is received by the controller. Then, a test must be taken to see if both of these strings are properly sent to the user database. If both of these tests pass, the only aspect remaining is to see if the user is authorized accurately. If the credentials match, the user shall be allowed to access the system. If not, the user should be prompted to try again. To test that the user entered the correct information, our test case will enter a combination of multiple User names and Passwords through a function that generates random combinations. The test case has to achieve a satisfactory result of 100% accuracy, since there shouldn't be any room for error in login phase. Also, the login has to be safe against certain vulnerability attacks such as SQL Injections.

Stock Searching

The act of stock searching is easily the most integral part of the entire system. It is obvious that stock data is necessary for the system to work correctly and for predictions to occur. The user should be able to search any stock in the database and be taken to the specific page for said stock. The testing for this scenario includes making sure that the user's search request is received and sent to the database.

Testing

The testing for this case has two main parts. These parts include: testing if the user input is properly read and testing if the input is correctly sent to the stock database. First, it is important to see that when the user inputs a string, the user controller receives this string. Obviously, it is imperative that the user input is read correctly so that the search function can be used properly. The next step is to test whether or not, the user input is now sent as input to the database controller. From here, a test to see if the database controller communicates properly with the database will be conducted. If all these aspects work properly, the user should be taken to the inputted stock's page. Verify that the stock is fetched quick enough and multiple requests don't overload the system

News Fetching

Our system is set to include the latest news involving stocks. Thus, news fetching is a large aspect for our system. Up to date news is required for predictions to be accurately made and for users to be correctly informed. The news feed will be updated constantly and data will be taken from the RSS feed of a reliable source.

Testing

To test the news fetching system, the first thing to be checked should be that the correct news is being fetched. This means that certain company names must be given as input so relevant news is displayed. The main two sources of this input will be: the user's "followed" stocks (taken from the user database) and the string searched if the user is using the search function to go to a certain stock's page. So, testing must be implemented to see that when on the main page, user "followed" stock names are sent to the RSS feed and the news for these stocks is returned. Also, when a search for a certain company is implemented and the user is taken to that company's page, the name of the company is sent to the RSS feed and the correct news is displayed. The names of the editor/article is verified by confirming with the official source and cross mapping the sources to verify that the origin of the news is reliable

Prediction

One of the main attributes of our system is being able to accurately predict stocks. Thus, our prediction algorithms must be tested in order to prove that they actually work.

Testing

An effective way to test the algorithms is to use old stock data to predict trends that have already happened. By doing this, it is possible to tell if the algorithm works as planned. Through testing, we will also be able to verify a confidence rating for each algorithm in certain situations. It is imperative that the algorithms are tested thoroughly, to ensure that our users get the most accurate data. The algorithm is tested via mathematical analysis and breakdown on similar cases by explaining previous scenarios in history.

Page Testing

There will be a couple of different types of pages that will require testing to make sure that the pages are loaded properly and the correct information is displayed. The types of pages included in our system include: login, registration, tutorial, quiz, main and stock.

Testing

For each type of page, it is required to test that each loads in the correct fashion. Once it is determined that each page can be loaded, testing then needs to be done to confirm that the correct information is loaded and displayed.

- For the login page, all that needs to be displayed are two input elements, one input element that submits the two input and sends it to the user database and another element which takes the user to an account creation page. Testing must also be done to ensure that correct login info authorizes the user, while incorrect info does not and prompts the user to try again.
- The registration page must be tested to show that a new user can make a username and password that is put into the user database. Also, this page needs to prompt the user to enter a stronger password, if the user has previously entered one that is considered too weak.
- Testing the tutorial page means to verify that the proper messages are displayed sequentially to teach the user about the stock market.
- The quiz page is tested to show that questions are displayed with an attribute that accepts answers as input. This input from the user will be sent to a database to check if the answer is correct or not, thus prompting the quiz page to provide feedback to the user.
- On the main page, a test should prove that news, graphs and predictions are displayed for the user's "followed" stocks. Also, an search attribute that takes input and sends it to the stock database must be present.
- The stock page is to include a graph of the stock's trends, a prediction on the stock with a confidence rating and the latest contextual news for the given stock. Testing will be done to show that all of these attributes are present.
- We test the appearance of the page by checking the tags of the displayed output to ensure that all of the images/ text load reliably
- For the quiz, ensure that the questions are not repeated and redundant, and accurately matches user preferences with a survey breakdown review.

Test Coverage:

LoginLogoutcontroller: Verify that each of the if-branches are taken, and ensure that the function responds properly to the random inputs of the test case (such as alphabets, letters and expressions in the username/password inputs). Also to be resistant to multiple entries, and prompt user

Graph Creator : Check for validity of the graphs when random/ incorrect data (such as infinity, boundary conditions etc). Ensure that the graph that is formed does not represent rapid increase/decrease, and that the graph that is rendered is visually readable.

Prediction Algorithms: Ensure that the algorithm that is displayed is in a readable form (does not contain unnecessary technical jargon or bits of code that is involved

Quiz Creator: Ensure that the questions are not repeated and redundant, and accurately matches user preferences with a survey breakdown review.

News Fetcher: Verify that the sources are correct, and can be attested by other sources.

StockDatabaseController: Verify that the correct stock info is fetched by ensuring the right id matches with the company name. Ensure that the Database controller does not crash/overload despite multiple requests and processes quickly enough.

Integration Testing Strategy

Integration testing will be conducted via Big Bang testing, specifically using Usage Model Testing.

Each module will be unit tested for internal functionality as well as interface specification in order to make sure the communication links between modules are consistent in what they expect for input and output. Then the modules will be linked together.

Each team will test their group of modules as a cohesive unit following Usage Model Testing in order to verify the functionality of their software against a specific set of use cases that involve their sub-systems. During this process, defined interfaces will be thoroughly tested as well as communication protocols.

Once each team’s functionality has been successfully tested, the three systems will be combined into the final software system. This final system will undergo Usage Model Testing against all use cases specified, with the tester using the system as a normal end user would in

order to verify full functionality. At this time, additional small features can possibly be tweaked/introduced in order to enhance the user experience.

Project Management

Coordination and Progress Report

Implemented Use Cases

No use cases are yet completed. Use cases 1, 2, and 3 are currently being implemented via various modules.

In the Pipe

As per the Progress Matrix in a below subsection, the stock cache, user database, learning system, and placement test are effectively complete. Their skeletons are created and need to be implemented alongside another interdependent system. In the case of stock fetching and stock cache, they need to be used together in order to make certain they can populate the cache from user queries. In the case of placement test and user database, the placement test should update the user database with flags.

Other project components are currently being fabricated, specifically: stock fetching; the tutorial; placement test; quizzes; and stock ticker. The stock fetcher is halfway finished and might even be finished ahead of schedule. The stock ticker should be completed soon and tested alongside the stock fetcher. The tutorial has undergone revision after exploration of different implementation means. The placement test and quizzes and roughly halfway complete.

Takeaway: project component fabrication is following schedule.

Activities

In an effort to promote team communication, sense of progress and direction, we employ:

- Group Meetings
    - This provides a domain such that most (5/7) group members are present for full group and subgroup communication. Large scope questions can be discussed as to solidify the direction and conception of the project.
        - Those who cannot make it have to provide a verbal progress report to group members whom their part concerns (subgroup members, subgroup leader, et cetera).
    - Minimum: 1 per week (1 hour). Current Average: ~1.66 per week (~2.33 hours).

- Group Chat-Room
    - This provides a domain such that group members can ask questions, answer, or just inform the entire group about something of value regarding the project.
    - In the advent that a subset of group members need to discuss something that only pertains to them, a smaller chatroom can be created for the duration until resolution. After resolution, the sub chatroom will be dissolved.

- Direct Messaging
    - The provides a one-on-one discussion for a specific part of the project or direct question-and-answer regarding the a project part, hence a sub chatroom would be too large of a domain.

Plan of Work

Each of the three vertical stacks (stock graphing and prediction [Team 1], website with user login and news and stock ticker [Team 2], educational tutorial and quizzes [Team 3]) operate independently of the others.

The tasks for Team 1 start out with linear dependencies, and then split off a the last stage. All 3 members will work together to complete the Stock Cache first, which has an allotted time of 2 weeks to completion. They then work on the Stock Fetching for 2 weeks. Then they work on the prediction algorithms for 16 days. After this, they each work on one of the three remaining components (Confidence Rating, Interactive Graphs, Algorithm Explanations) for 2 weeks. At this point, Team 1 finishes its tasks, dissolves, and merges into the two teams for the Website and Android app.

Team 2's tasks are more disparate, but can be separated into two sets of dependencies. The first task that must be completed is the User Database, which is a dependency for multiple components. This occurs in parallel with the Stock Ticker and Contextual News in order to maximize throughput. These are all given a timeline of 3 weeks to complete. After this stage, there remain 3 tasks (Account Registration, User Login, User Favorites) that can all be completed in parallel in a timespan allotted of 24 days. At this point, Team 2 finishes its tasks, dissolves, and merges into the two teams for the Website and Android app.

Team 3's tasks occur in two parallel timelines (Tutorial, Color Preferences) (Placement Test, Quizzes, Spot-Checks) with mostly linear dependencies for each line. The Tutorial and Placement Test are worked on simultaneously for 4 and 3 weeks respectively. Then the Color Preferences and Quizzes are worked on simultaneously for 18 and 14 days, respectively. At this time, the first timeline is complete and that member joins the second timeline to assist with Spot-Checks. In total, 3 weeks are allocated for the Spot-Check component. At this point, Team 3 finishes its tasks, dissolves, and merges into the two teams for the Website and Android app.

Demo 1 will consist of 3 separate demos. Team 1 will demonstrate their stock fetching and prediction by producing graphs on demand via a simple website. Team 2 will demonstrate their completed website that allows user to login, register, and see news and stock tickers. Team 3 will demonstrate the Placement Test, Quizzes, and Interactive Tutorial via a simple website.

The website and Android app are ongoing teams that grow as members from other teams finish their tasks. They will be the key difference between Demo 1 and Demo 2.

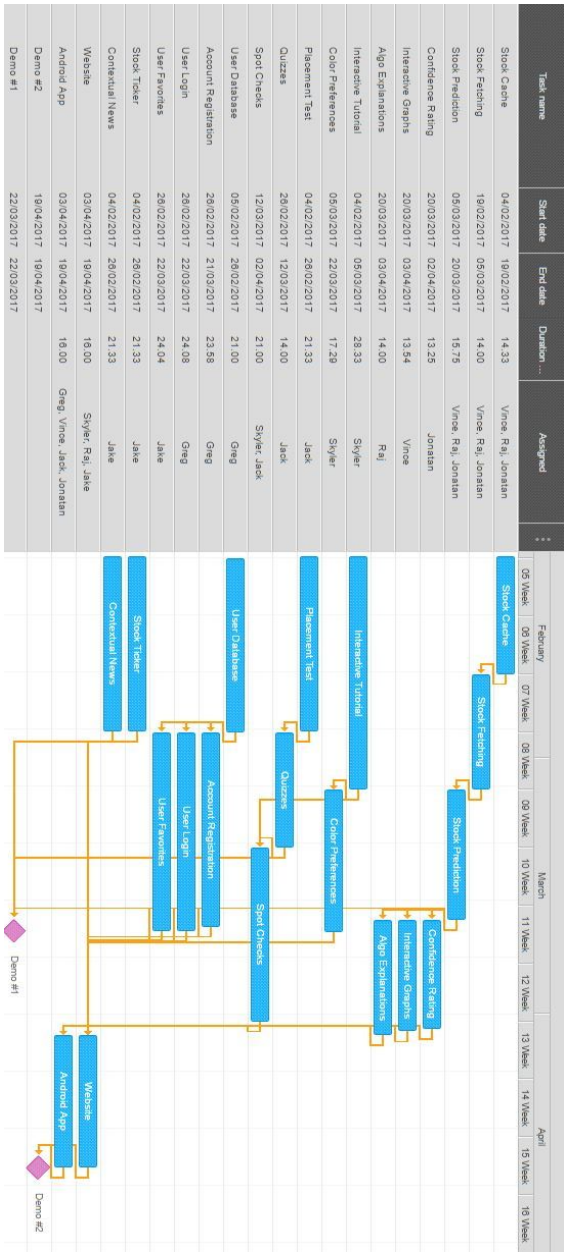


Figure 7.1 - Code Gantt Chart

Note: Interactive Tutorial is now called Learning System; Stock Database is now Stock Cache.

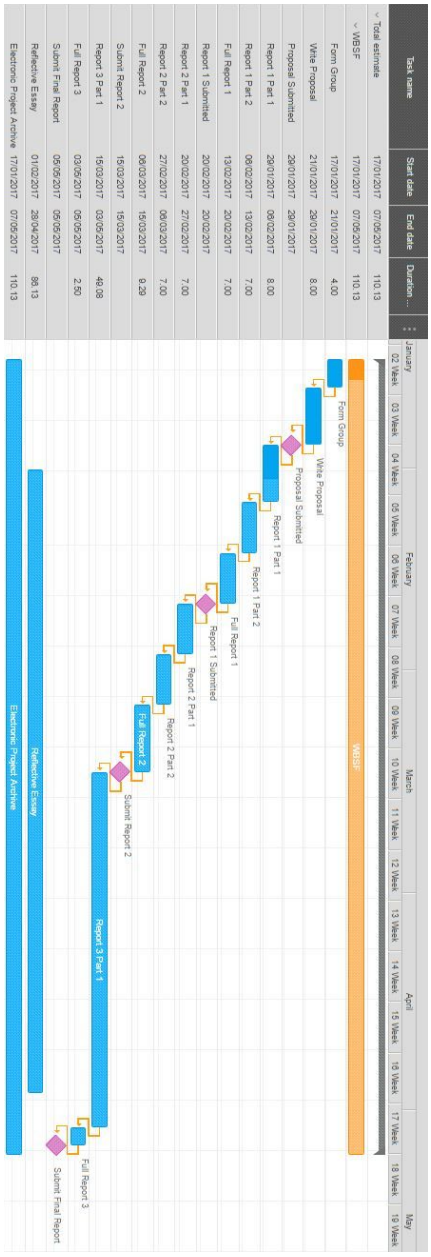


Figure 7.2 - Documentation Gantt Chart

Responsibility Breakdown

The project will be composed of three teams. Each team will operate completely independently of the others until it finishes its tasks, at which point it will dissolve and the members will join the two teams for stitching together the final website and producing the Android application.

The integration will be coordinated and performed by Vincent Taylor for Team 1, Gregory Leonberg for Team 2, Skyler Malinowski for Team 3. Each of these teams' integrated product will be their demo for demo 1.

Final integration will be coordinated and performed by Gregory Leonberg for the Android Application and Skyler Malinowski for the Web Application. These will be the products for demo 2.

	Team 1	**Team 2**	**Team 3**
Members:	Balaji, Raj Taylor, Vincent Yanovsky, Jonatan	Leonberg, Gregory Lewandowsky, Jake	Aquino, Jack Malinowski, Skyler
Features:	Stock Cache Stock Fetching Stock Prediction Interactive Graphs Algorithm Explanations Confidence Rating	Login System User Registration Account Database User Favorites Stock Ticker Contextual News Feed	Placement Test Interactive Tutorial Proficiency Quizzes Spot-Checks Color Options

Figure 7.3 - Team Breakdown

Progress Matrix

Member	Completed	Working-On	Queued
Leonberg, Gregory	User Database	Account Registration User Login	Android App
Lewandowski, Jake		Stock Ticker Contextual News	User Favorites Website
Malinowski, Skyler	Learning System	Color Preferences	Spot Checks Website
Aquino, Jack	Placement Test	Quizzes	Spot Checks Android App
Taylor, Vincent	Stock Cache	Stock Fetching	Stock Prediction Interactive Graphs Android App
Yanovsky, Jonatan	Stock Cache	Stock Fetching	Stock Prediction Confidence Rating Android App
Balaji, Raj	Stock Cache	Stock Fetching	Stock Prediction Algo Explanations Website

Figure 7.4 - Progress Matrix

References

https://creately.com/
[Tool used for creating UML Sequence diagrams]

https://ganttpro.com/
[Tool used for creating gantt charts]

https://help.ubuntu.com/community/Installation/SystemRequirements
[Minimum hardware requirements for Server]

https://www.mozilla.org/en-US/firefox/50.1.0/system-requirements/
[Minimum hardware requirements for Client]

https://upload.wikimedia.org/wikipedia/commons/6/62/Wikipe-tan_visual_novel_%28Ren%27Py%29.png
[Media Image of an example Visual Novel Interface]