What the Stock?!

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

Team Member Contribution Breakdown

(Equal Contributions)

Contribution Matrix

Section	Sub-Section	Pts	Jack Aquino	Raj Balaji	Gregory Leonberg	Jake Lewandowski	Skyler Malinowski	Vincent Taylor	Jonatan Yanovsky
Project Management [13]	Report Coherance	5	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Team Coordination	8	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
Summary of Change [5]		5	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
Consumer Statement of Requirements [6]		6	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
Glossary of Terms [4]		4	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
System Requirements [6]	Enumerated Function Requirements	2	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Enumerated Non-Function Requirements	2	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	On-Screen Appearance Requirements	2	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
Functional Requirements Specification [30]	Stakeholder & Actor Goals	2	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	UC: Casual Description	7	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	UC: Case Diagrams	5	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Traceability Matrix	1	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	UC: Full Description	10	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	System Sequence Diagrams	5	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
Effort Estimation [4]		4	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
Domain Analysis [25]	Domain Model	15	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	System Operational Contract	5	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Mathematical Model	5	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
Interaction Diagrams [40]	UML Diagrams	10	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Description of Diagrams	10	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Alternate Solution Description	10	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Use Design Patterns	10	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
Class Diagram and Interface Specification [20]	Class Diagram Description	5	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Signature Structures	5	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	OCL Diagrams	10	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
System Architecture and System Design [15]	Architectual Styles	5	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Identifying Subsystems	3	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Mapping Subsystems to Hardware	3	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Persistant Data Storage	1	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Network Protocol	1	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Global Control Flow	1	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Hardware Requirements	1	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
Algorithm and Data Structures [4]	Algorithms	2	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Data Structures	2	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
User Interface Design and Implementation [11]	What Has Changed	4	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Ease of Use	7	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
Design of Tests [12]	Coverage Tests	6	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Integration Testing	6	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
History of Work [5]	Key Accomplishments	2	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Future Works	2	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
	Responsibility Matrix	1	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%	14.29%
TOTAL		**200**	**28.57**	**28.57**	**28.57**	**28.57**	**28.57**	**28.57**	**28.57**

Contribution Pie-Chart

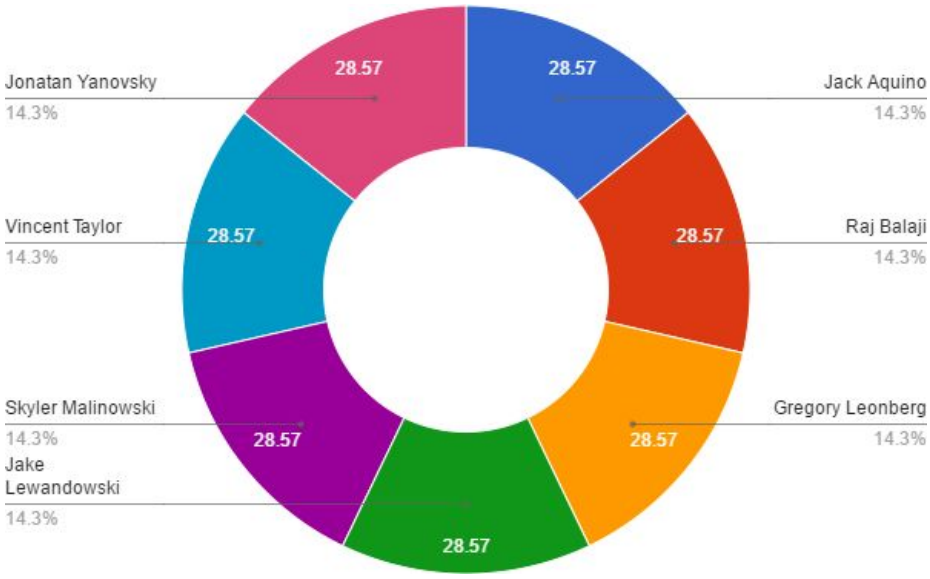

Table of Contents

0. Summary of Changes

Project Changes

- Name change: “WBSF (Web-Based Stock Forecasters)” to “What the Stock?!”.
- Goal changed to learning website.
- The database and database controllers are wrapped into single entities.
- Use Cases have been added for login, registration, and logout.
- Requirements have been added in relation to time based behaviors and the tutorial.
- All diagrams have been re-made to match the new class structure.
- The user interface design has been altered to match the final implementation.
- VN (Visual Novel) type lessons were reconsidered and added after completion of one-page type lessons.
- Implementing the use of Apache alongside Flask.
- Redacted idea of user custom color preferences in favor of color palettes.
- Stock database only holds prediction coefficients instead of historical data and coefficients.

Report Changes

- Customer Statement of Requirements was tweaked
- Glossary of terms was updated
- System Requirements was updated
    - Enumerated Functional Requirements was updated
    - Enumerated Nonfunctional Requirements was tweaked
    - On-Screen Appearance Requirements was overhauled
- Functional Requirements Specification was updated
    - Stakeholder was tweaked
    - Actors and Goals was updated
    - Use Cases was overhauled
    - System Sequence Diagram was updated
- Effort Estimation was added
- Domain Analysis was updated
    - Domain Model was updated
    - System Operation Contract was updated
    - Mathematical Model was updated

- Interaction Diagrams was updated
- Class Diagrams and Interface Specification was updated
    - Traceability Matrix was added
- System Architecture and System Design was unmodified
- Algorithms and Data Structures was tweaked
    - Algorithms was teaked
    - Data Structures was tweaked
- User Interface Design and Implementation was overhauled
- Design of Tests was tweaked
    - Test Descriptions and Coverage of Tests was tweaked
    - Integration Testing Strategy was tweaked
- History of Work was updated
    - Key Accomplishments was added
    - Future Work was added
    - Responsibility Breakdown was updated
- References was updated

Please Note: tweaked means minor alterations; updated means altered to reflect changes made elsewhere; overhaul means major revisions made.

Please Note: Because of the shear amount alterations made by multiple team members, it is difficult to give the precise changes made to each section.

1. Customer Statement of Requirements

Problem Statement

When someone thinks about the Stock Market, not many people can confidently say that they know how it works. Many people have the idea that the Stock Market like a high-risk, high-reward game where investors have the potential to make a large amount of money. While it is true that the rewards of the Stock Market are high, the risks are not as high as people believe them to be. With enough knowledge of how the stock market works, it is possible to accurately predict the market and consistently make money through stocks. Hence learning how to use the Stock Market properly is a profitable and worthwhile endeavor for future planning.

However, many people lack the necessary resources to learn about the Stock Market. Compulsory education systems merely gloss over the subject, and higher education does not explore the subject unless it's for a specific degree. The goal of the "What the Stock?!" app is to provide an educational tool with which users can learn more about the stock market. The users which would benefit the most from this app are people with close to zero knowledge of the stock market, from students in high-school level economics courses to individuals who want to learn how to invest their money wisely. Our app would gradually introduce the end users to important concepts of the stock market, while consistently quizzing users on previous material in order to ensure that the user properly learned it.

We wish to provide a useful tool that, among other uses, will provide a tutorial about basic investing. The tutorial will explain how to navigate through various stocks, as well as how to interpret various bits of data regarding investing in general. The tutorial will cover the basics, such as the definition of stocks, what a portfolio is, and the incentives to sell stocks of their business. The tutorial is impart all users –from ones with close to zero knowledge through veteran investors who may have forgotten a thing or two about stocks – with the basic knowledge of the stock market. By the time the user has completed it, they will have enough knowledge to explore various stock tickers through our app, as well as interpret the data that graphs represent.

Since the main idea of our app is to be a learning tool, we have also decided to implement a "review system" that will pop up while the user is on the app. The tutorial system is built to ensure that the user learns the material, and the review system will ensure that the user retains everything that they learned. At certain, randomly chosen intervals on the app, a review question will pop up that asks the user something that they have already learned from the tutorial. These periodic quizzes will vary in topics, so that the user will not have the same quiz question repeated to them (unless they have already done all the quiz questions available).

Motivation

The stock market has a steep learning curve, there is wording, knowledge, intuition, and a plethora of skills that require constant practice; there lacks a tool set that can gently, and steadily guide a user into the stock market. The number of people that do not have the ability to research information or endlessly search for answers is incredibly high. People range from full-time students looking to set up a secure future, to elderly adults who want to earn extra spending money. Full-time students who are burdened with class often find a little time to heavily research secure stocks. Additionally, they cannot constantly monitor changes in the stock market, since they have classes and other activities to attend to. The elderly on the other hand may have the time, but not the skill. Searching for information may seem rudimentary, but to a retiree looking to invest in multiple places, the sheer amount of information alone acts as a deterrent. Not to mention the fact that the energy spent on compiling all of this information distracts the trader from their original goal of spending more money.

The massive amount of history and fields each stock dabbles into require research. Each stock has past figures, that to an inexperienced trader, look like Rorschach blobs. Aforementioned, the time for a range of traders is inaccessible or inconvenient. To even begin to understand the various readings on the dow jones is a trifling task; however, there are more options and more figures to command. Even if someone was to compile all of the information, required to understand basic stock trading; As their portfolio grow so does the amount of research and the need for diversification. Since most research points to diversifying a portfolio. The next logical step is to branch out and explore other areas of investment. Thus, the amount of research and time will grow, which mentioned earlier may pose impossible or inconvenient. This is purely the factor of research, after expanding a portfolio the trader then has to monitor and compile the information. This level of organization proves difficult when combining multiple sources together, and constantly switching between stocks to watch their development.

Even more, experienced traders may find the amount of information daunting. Compiling and organizing thousands of numbers, while also monitoring real time requires dedication and complete devotion. Once again the dilemma of time becomes and issue. With the required skill and determination the compilation of information is often error prone since it reigns from a multitude of different sources. The constant checking and verification prove tedious and infuriating, no matter the scale. Without a proper organization method or tools, individuals may never be able to expand and meet their goal of investing. The constant need for information as well as real-time changes proves difficult and disheartening. Without progression, diversification shrinks, inversely, the chance of an individual losing their money increases. Without the money to expand or the know how the user could find themselves in the red and losing more money than they bargained for.

The necessity for a tool that informs the user on the stock market's current state while guiding them through the entire trading process is too high to ignore. With limited time, money,

and information, trading becomes difficult and stressful. The lack of organization leads to mismanagement and with enough data and human input, an error is bound to occur, such error may lead to loss, and overall deter the trader from engaging in trading. With an increasing amount of companies and data, the need for an organized, educational, quick, and straightforward system becomes necessary.

For a long time now, the stock market has been a successful tool in helping individuals earn money through investments in a variety of companies. The act of investing in stocks has made a lot of people a lot of money. Naturally, you would think that everybody would be involved in the stock market. Well, unfortunately, this is not the case.

According to CNN, only 35% households headed by someone with a high school diploma as the highest form of education actually have money in the stock market. This number goes up to 72% for households with members who have college degrees. Obviously, this is a concern since the stock market is a win-win for the investors and the companies, themselves. From the statistics, we can see that there is simply a lack of education in the act of investing and how the stock market works.

For newcomers, the act of purchasing a stock seems like a large and risky commitment. Without the proper research and basic knowledge, one would not know where to start. Most, if not all, people in this situation just turn their backs and stay away from the market as a whole. All of these people are missing out on potential big monetary benefits. A stock market is a scary place and not many people are willing to hold a newcomer's hand and teach them the ins and outs for free.

That is where we come in. We are creating a system for the average person, uneducated about the stock market. This system will be an easy way for these people to learn how the stock market works and eventually lead them to start purchasing their own stocks. This system will use an effective tutorial system in which the user is tough about the most basic aspects of the markets up to the more advanced strategies. Our vision is to be able to educate a person who has never even heard about the stock market, into one who can confidently invest and earn some easy cash.

The tutorial will not be the only feature of our system. Another attribute of the system will be real-time, effective stock prediction. We will include several stock prediction algorithms for more confidence in our predictions. Each user will be able to search any stock and view trends and predictions for said stock. The user will be able to pick stocks to track to prevent them from having to search each time. Along with our stock prediction algorithms, there will be a confidence rating for said algorithm. This will help our users be confident in their investment strategies.

As you can see, the main goal of this system will be to turn any person into an above average stock market investor and help them earn some extra cash. The tutorial combined with

the post tutorial stock prediction algorithms will be effective means of bringing people up to speed on the ways of the stock market. We hope to increase the stock market literacy of the population by a sizeable amount. With this system, we believe this goal is easily achievable.

2. Glossary of Terms

Apache: A free, widely used web server able to be run on Windows and Linux. This server supports languages like HTML, CSS, and Javascript and has many features useful to web developers.

Broker: A person who buys or sells an investment for you in exchange for a fee (a commission).

Capital: Financial assets or the financial value of assets, such as cash (Usually initial).

CSS (Cascading Style Sheet): A type of programming language (specifically, style sheet language) used to describe the appearance of web pages written in HTML. In other words, CSS is used to set the visual aspect of a web page.

Confidence Rating: Rating determined by the prediction algorithm on how “sure” it is of its own prediction.

DHTML (Dynamic Hypertext Markup Language): An umbrella term for a collection of technologies: HTML (Hypertext Markup Language); CSS (Cascading Style Sheet); and JS (JavaScript). These are used for creating interactive and animated websites. These languages compose the core three technologies of the World Wide Web.

Dividend: A payment made by a corporation to its shareholders.

Fall: Fall in stock price.

Flask: A micro web-framework used to allow backend Python code to communicate with a website’s frontend code.

HTML (Hypertext Markup Language): A language used to encode the structure of a web page. Contains webpage text and hyperlinks, and may also contain Javascript code and CSS layout schemes to define a web page.

JavaScript: Javascript enables a wide range of possibilities for a web developer who may want to add custom behavior to their website.

Jquery: A fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.

Margin: A margin account lets a person borrow money (take out a loan essentially) from a broker to purchase an investment. The difference between the amount of the loan, and the price of the securities, is called the margin.

Portfolio: A grouping of financial assets such as stocks, bonds and cash equivalents, as well as their funds counterparts, including mutual, exchange-traded and closed Funds: Are typically mainly in stock, with some amount of cash, which is generally quite small, as opposed to bonds, notes, or other securities.

Plotly: A web-based data visualization tool that allows one to generate online interactive graphs.

Rally: A rapid increase in the general price level of the market or of the price of a stock.

Real-Time: Relating to a system in which input data is processed within milliseconds so that it is available virtually immediately as feedback.

Rise: Increase in stock price.

Stock: A stock is an ownership share of a corporation/company.

Stock Symbol: A one-character to three-character, alphabetic root symbol, which represents a publically traded company on a stock exchange. Apple's stock symbol is AAPL.

Trend: The general direction of a market or of the price of an asset.

VN (Visual Novel): A japanese storytelling medium that is a hybrid between a video game and a choose-your-own adventure story.

Volume: The number of shares of stock traded during a particular time period, normally measured in average daily trading volume.

XHR (XMLHttpRequest): An API in the form of an object whose methods transfer data between a web browser and a web server. The object is provided by the browser's JavaScript environment. Particularly, retrieval of data from XHR for the purpose of continually modifying a loaded web page is the underlying concept of Ajax design.

Yield: The simple ratio of annual dividends divided by the share price.

YTD (Year to Date): Refers to the period beginning the first day of the current calendar or fiscal year up to the current date.

3. System Requirements

Enumerated Functional Requirements

Weights (higher = greater priority)
3 = primary feature - required for success.
2 = required for proper functioning of primary features.
1 = secondary feature - for convenience.

Identifier	Priority	Description
REQ-1	3	The system shall provide as accurate as it can prediction of the stocks for the selected company.
REQ-2	1	The system shall provide a response in a reasonable amount of time.
REQ-3	1	The system shall be navigable with a minimal number of redirections
REQ-4	3	The system shall provide a tutorial mode in which the user can learn about the stock market and stocks in general.
REQ-5	1	The system shall locally encrypt user passwords to protect the integrity of user information.
REQ-6	2	The system shall provide new users the opportunity to register an account with a unique username in order to keep track of various tutorial and portfolio information.
REQ-7	1	The system shall remember the user's preferences.
REQ-8	2	The system shall have a security login to prevent unauthorized access.
REQ-9	2	The system will provide both past and present stock history of the selected company using a graph.
REQ-10	1	The system will allow users to follow certain stocks in a portfolio.
REQ-11	1	The system will provide "feed" information from other news services.
REQ-12	2	The system will display and explain a confidence rating for the prediction.
REQ-13	2	The system will allow the user to control the graph axis.
REQ-14	1	The system will allow for the user to replay the tutorial.

REQ-15	2	The system should allow the user to search for certain stocks.
REQ-16	3	The system will quiz the user on the material learned through the tutorial upon login.
REQ-17	2	The system should refresh and apply the prediction algorithms periodically.
REQ-18	3	The system shall have a quiz at the end of a lesson in order to test the user's knowledge of the material.
REQ-19	1	The system should use multiple algorithms to provide stock predictions.
REQ-20	3	The system will have a mobile friendly user interface.
REQ-21	1	The system will provide auto-completion of the stock search field
REQ-22	3	The system will provide a visual novel tutorial as an alternative interface

Figure 3.1

Enumerated Nonfunctional Requirements

F.U.R.P.S. =

Functionality

The F in the FURPS+ acronym represents the main product features that are familiar within the business domain of the solution being developed. The functional requirements can also be very technically oriented. Functional requirements that you may consider to be also architecturally significant system-wide functional requirements may include auditing, licensing, localization, mail, online help, printing, reporting, security, system management, or workflow

For our education tool, the main features would be:

- A tutorial mode about the application
- Ability to fetch data from stock history websites
- High-level Algorithms to predict stock data
- Interactive graphs with stock and prediction data
- Easy interactive UI that helps the user understand the data presented to them
- Fetch specific stock information is inputted by the user

- Account creation, and a login system with which users can save their preferences between sessions.

Usability

Usability includes looking at, capturing, and stating requirements based around user interface issues — things such as accessibility, interface aesthetics, and consistency within the user interface.

User Interface Accessibility – Make an online website that can be accessed at any time, as well as a mobile app that allows the user to access the website from his phone, as opposed to a PC. The interface should fit to match the screens of the user's devices and make the most use of that device's interface.

Distinctive Features – Naturally guide the user toward important information on the website with side tabs that hide themselves when not moused over. The Tabs are colored in black to make the links easier to identify from one another

User Interface Consistency – The parts of the page that change when the User attempts to do different things should be minimized. That is, every page in our website should follow a general template, where certain menus are identical for each page.

Reliability

Reliability includes aspects such as availability, accuracy, and recoverability — for example, computations, or recoverability of the system from shutdown failure.

Availability – As our system will be run off of one of our own computers, availability will be only when the computer is on and functioning as a server for the website (in server mode). We will not be using a paid website / dedicated server provider. Any computers can execute the Python + Apache files using cmd as long as they follow the ReadME file in our github page

Accuracy – The accuracy of the stock charts will be limited by our information source, which is the very accurate Yahoo Finance API. Our algorithms should provide prediction results that are at least 80% accurate.

Recoverability – The program code is stored on Github, a Cloud-hosted Configuration Management Tool, so that it can be restored easily if our server crashes. The local server database for stock information can be dynamically reconstructed simply by running the program from a clean reset state. The local user information database will be routinely updated whenever

a new account is created. The database updates using a write-through technique that saves the information permanently. In case of a major crisis, the github can roll back. There are also local states stored on each of our computers.

Performance

Performance involves things such as throughput of information through the system, system response time (which also relates to usability), recovery time, and start-up time.

Speed – The front end of the website is written in Javascript, while the backend of the website is written in Python. Javascript is lightweight, and thus does not take long to execute, while the Python code is written to minimize server requests so that the code can execute faster. Flask is used to wrap up the code for the website

Efficiency – probably not maximum, since we are not using multithreading for the back end. Since we are not using the CPU too much multithreading is not necessary.

Resource Consumption – would use a lot of system RAM, probably less CPU, but a lot of the network to retrieve data from a database. The Plotly API key is limited and has to be reloaded after a certain number of tries due to using the free version of the Plotly module.

Scalability – not gonna scale well as the backend and database program will be designed for a limited number of users and stocks to reduce complexity. Multiple users can cause the website to lag and at worst case use up all the Plotly API keys, so limited users only.

Serviceability

Our code will be on Github so we can maintain, modify, adapt it easily by pushing new code to it, but then the most recent build must be downloaded onto the server and restarted, causing a crash to the existing build (not extensible well / poor installability). It is sustainable to last for a demo but probably really inefficient over a longer period of time. Testable immediately through GUIs being developed for website and algorithm views. It is modular, though, at each end of the program being relatively separate from each other. Javascript can be coded to allow localization of our program easily.

Repair Speed – We are running on GitHub so in the unfortunate case that an error does arise then we can always revert back to a state where the site worked. This allows us time to repair while decreasing the time in which the site is down. The speed of which the repair occurs also increases with the use of GitHub since we can see exactly what lines were changed

between the previous version and the damaged one. This allows for immediate analysis and repair.

Installability – Since we are running off of a web-based server the installability of this software is high. Almost every modern browser supports Javascript and most preinstalled/popular browsers support the most current Javascript version. Without the need for a dedicated application like .exe, or .jar. The user should not have trouble using the service across a multitude of devices.

Testability – Our code does not revolve around a particular stock so as long as it works with one stock it will work for all others. Also, the Stock market is continually changing so most scenarios our program handles are tested with the use of a single stock.

Configurability – Our code is configurable since it fits around most stock scenarios. The use of graphs and multiple prediction techniques allows for the user to configure their output and presentation of data. Since there are multiple prediction techniques the system is configurable to fit different scenarios. This also reflects the system's Adaptability.

On-Screen Appearance Requirements

Because this project has a web-interface, there are key visual requirements that have to be met to convey information to the user in an easy to read manner. Unfortunately they occur at different scales so to speak. The microscale refers to the individual elements on a page, and the macroscale refers to a page and how it is laid out with elements. These two scales have to be considered when talking about the user interface and its requirements. Additional details are given in each subsection regarding exact requirements with a picture as a reference point.

Elements

A module that has accomplishes a specific purpose and is self-contained.

Search Box

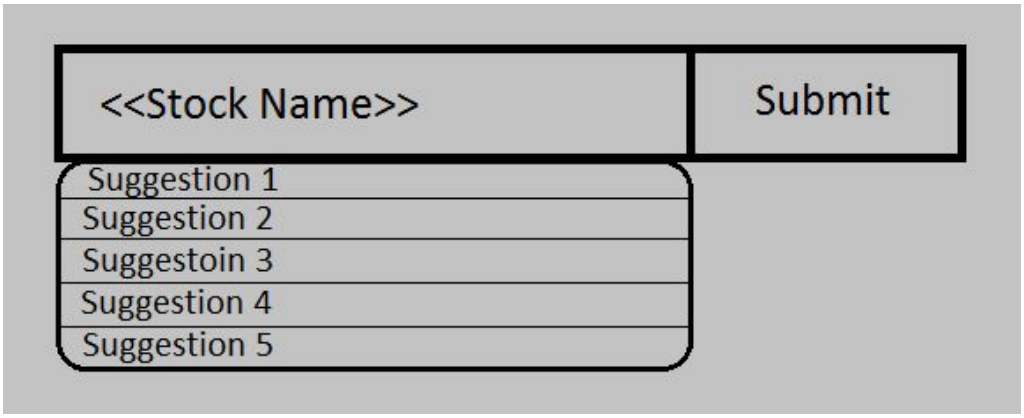

Figure 3.2

This element must contain a text entry box, a submission button, and a suggestion drop-down box. To have the user efficiently and effectively use the search box it should employ an autocomplete feature that acts on the drop-down box as a contextual dropdown — the dropdown suggestions populate dynamically with the user's entry into the text input box. The submit button takes the entered text and sends it to be processed out of the user's sight.

Interactive Stock Graph

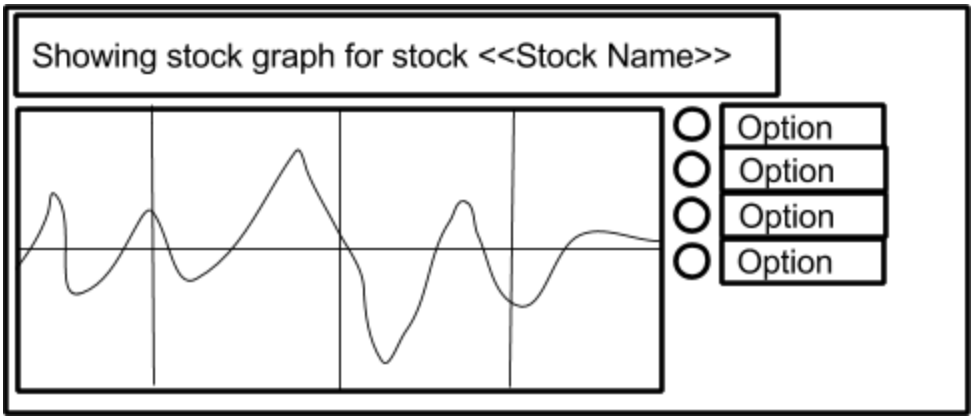

Figure 3.3

This element must display an interactive graph, the name of the stock being graphed, and interactive graph options. The interactive options are seen to the right of the graph; they are for showing or hiding the various algorithms that can be applied to the graph. This provides the user with control over the data they want to see regarding the graph. Because the graph is interactive, the user can manipulate the graph frame with zooming and panning around.

Contextual News

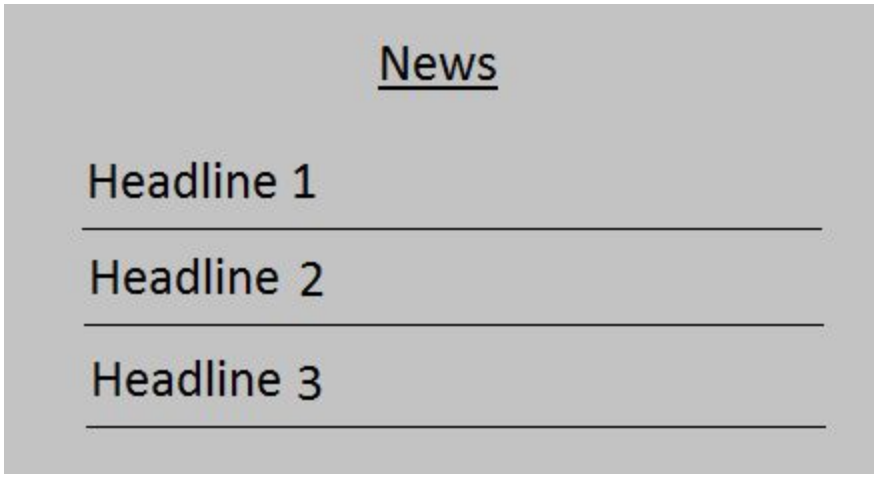

Figure 3.4

This element must display a list of relevant news headlines corresponding to the stock being searched. In the ideal case, this entire element will look like a list, where the title of the list contains the name or ticker symbol of the stock searched, and the elements of the list are links to the news articles. As a design choice, we opted to pull only four or five articles for this element.

Stock Ticker

Figure 3.5

.The Stock Ticker refers to the bits of information pertaining to the current price of a stock. It consists of three important pieces of information: the stock's ticker symbol, its price per share, and a symbol stating whether or not the stock's price went "up" or "down" for the day.

Normal Lessons

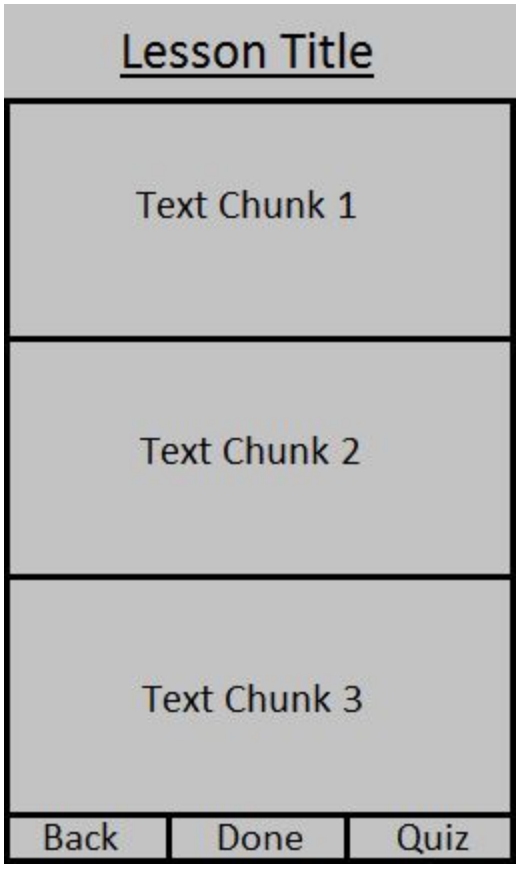

Figure 3.6

Our app is a learning tool, and Lessons are the primary way to convey information about the Stock Market to the user. Lesson pages will have a title conveying the main idea of the material, along with smaller titles that convey topics pertaining to the main idea, as well as divide the page. Each section of the page will contain paragraphs of text that elaborate on the ideas being presented.

Visual Novel Lessons

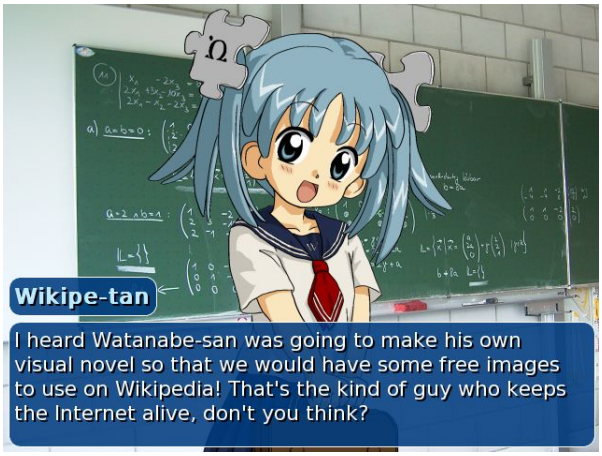

Figure 3.7

Under the basic premise that different people learn in different ways, we have opted to include Lessons in a Visual Novel format, instead of just plain text. The information conveyed in a Visual Novel Lesson is the same information conveyed in the standard Lesson pages. However, this element will include one or more characters, a background image, and names, in order to convey the feeling of being spoken to in a classroom by the character. In this example, the character “Wikipe-tan” is speaking to the user in a classroom setting.

Quizzes

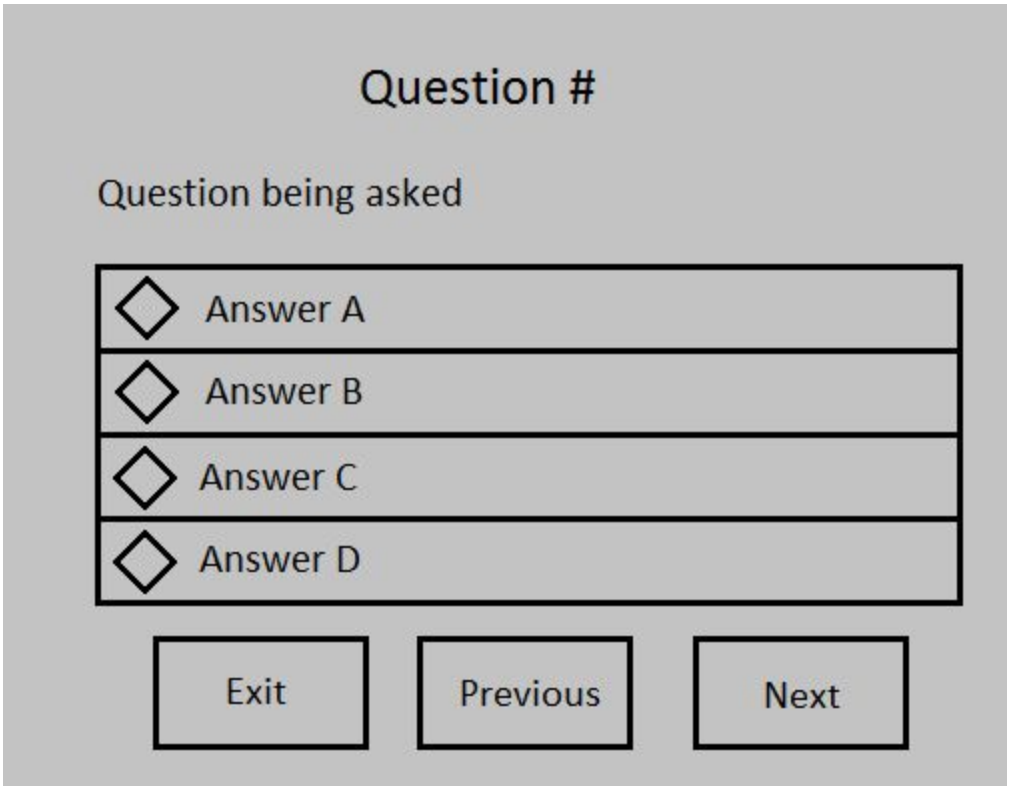

Figure 3.8

Quiz elements will consist of simple multiple-choice questions pertaining to the information from a particular lesson. A quiz consists of multiple questions: as such, the Quiz element should have a title that states the material of the questions, the question itself (and its question number), and a list of 3-5 answers to choose from. The user should be able to click on one of the answers to select it as their choice, and only one of those answers should be the correct one.

Panels

Panels can be considered a type of element. It is accessed from a page usually via a button but it causes another sub page to appear. Once the button is pressed, this panel element becomes visible over the page and other elements. This means that there is no redirecting and thus no additional loading for this panel element. It is as if a subpage appears.

Panel Buttons

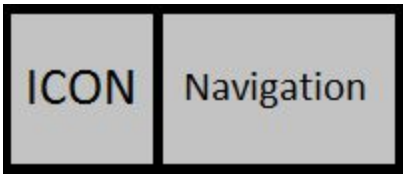

Figure 3.9(a)

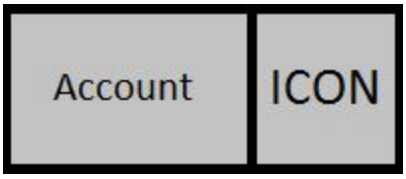

Figure 3.9(b)

These panel element must be be buttons with an icon and text to convey what the buttons do or lead to dealing with.

Account Login/Logout/Register Panel

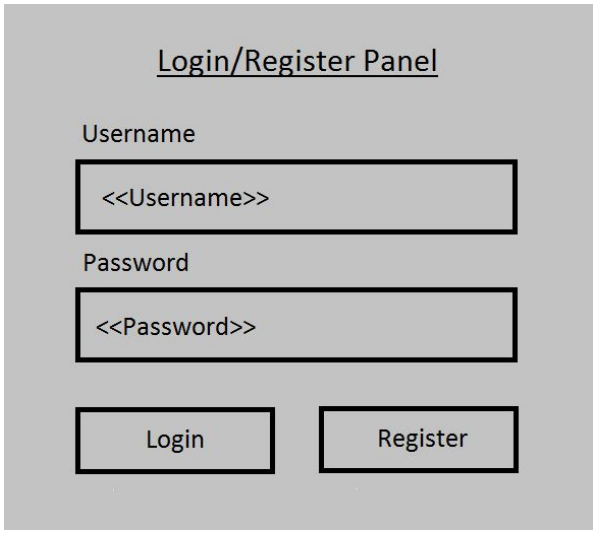

Figure 3.10(a)

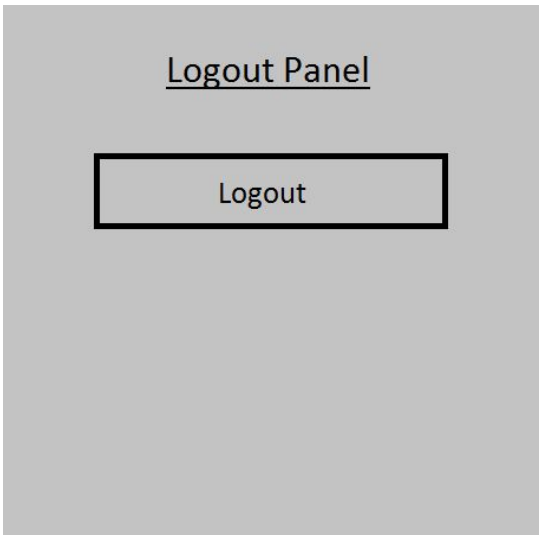

Figure 3.10(b)

This panel element must contain a username entry box, a password entry box, a login button, logout button, and a register button. The login and register button takes the credentials entered in the username and password boxes and processed them. The login button checks the database for the user as starts their session if they exist; while the register button add the account to the database if the username is not already taken.

To remove ambiguity of account status: if the user is not logged in, the logout button is hidden — as seen in Figure 3.1(a); if the user is logged in, then only the logout button is visible to the user — as seen in Figure 3.1(b). And the user is notified of what is going on after a button is pressed via browser (login success/failure, logout success/failure, register success/failure).

Navigation Panel

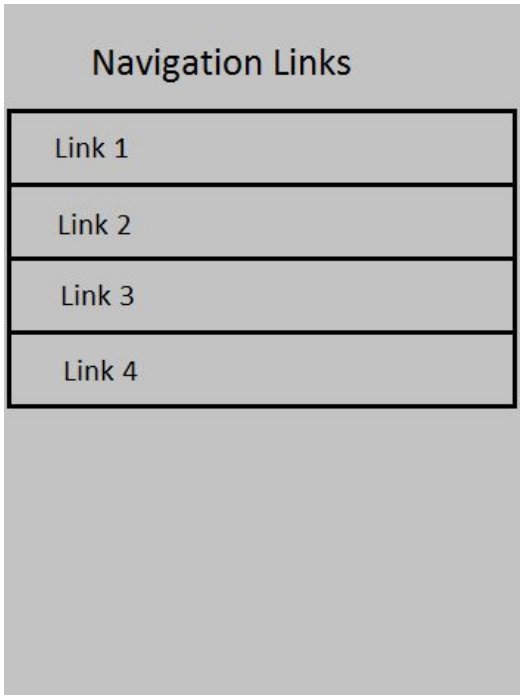

Figure 3.11

This panel element must have buttons that act as links. Said links go to other core pages — splash page, learning page, search page, and portfolio page — on the website. This allows the user to go anywhere on our site with ease.

Pages

Splash Page

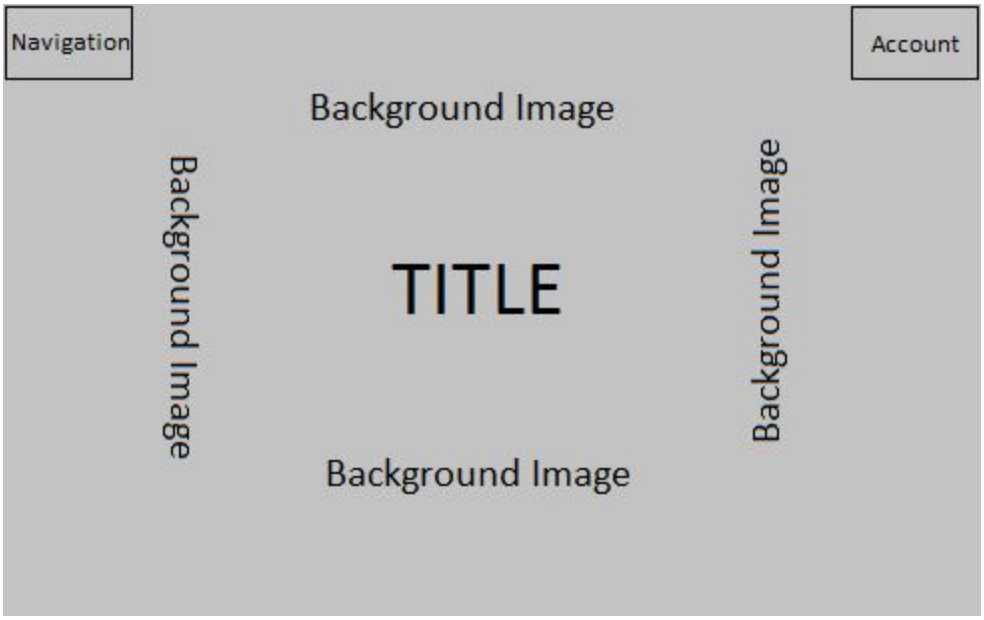

Figure 3.12

The Splash Page of our website is the face of our website. It serves little purpose, other than acting as a page where the User is redirected to upon first loading our website. The main reason for this page is to have a “neutral page” as our website’s face, rather than a page containing the work of one specific sub-team. The user should be able to freely choose which part of our site to go to from here, without being influenced by any flashy details about one subteam. The splash page itself should be plain: it will contain a logo for our site, as well as our site’s title, “What the Stock?”. Ergo this only requires a background image, small amounts of text, and the panel buttons.

Learning Page

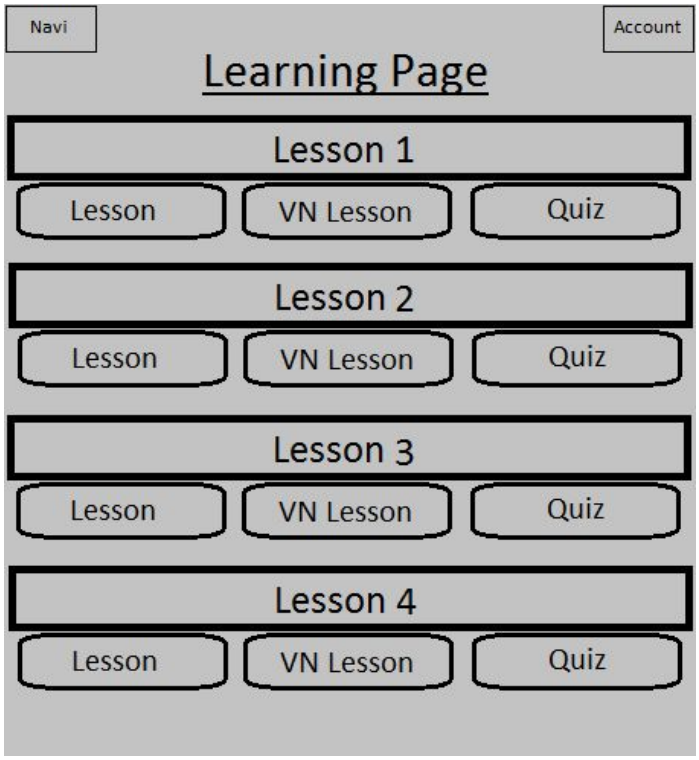

Figure 3.13

This page act as a hub to normal lessons, visual novel lessons, and their accompanying quiz. Thus this object must display lesson headers or titles, normal lesson buttons, visual novel lesson buttons, and quiz buttons. Visually, each topic cluster (header bar, normal lesson button, visual novel lesson button, and quiz button) must look separate from another topic cluster so the user can quickly scan the page and discern each cluster without much thought.

Search Page

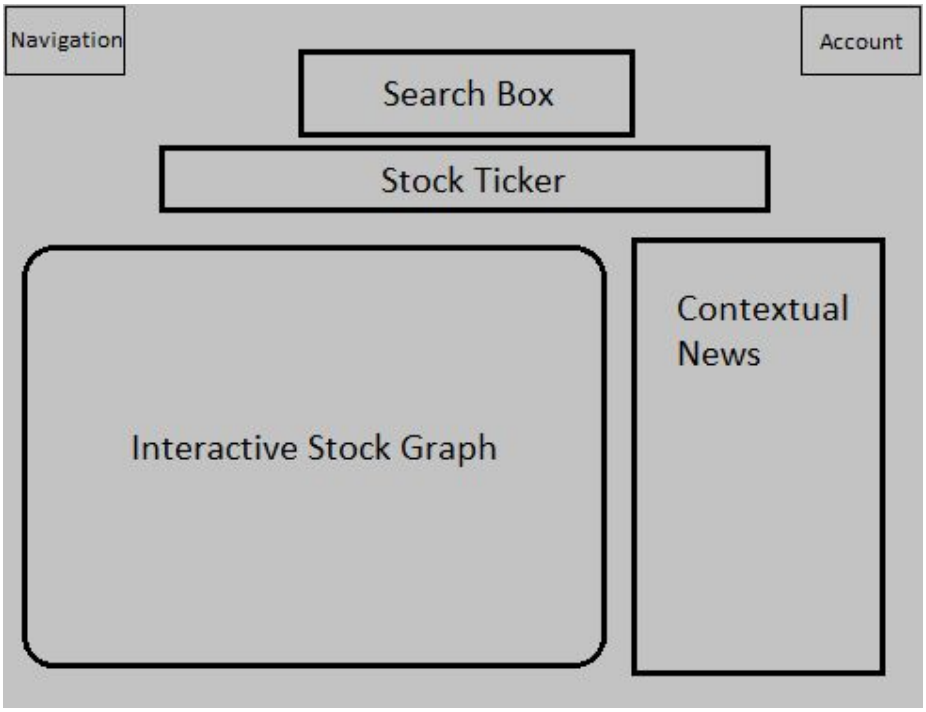

Figure 3.14

This page must display the search box element, the contextual news element, the stocker ticker element, the stock graph element, and the panel element buttons. On this page, the User will be able to enter a Stock ticker symbol and/or company name into the search box, and obtain an interactive stock graph, stock ticker info, and contextual news about that stock.

Portfolio Page

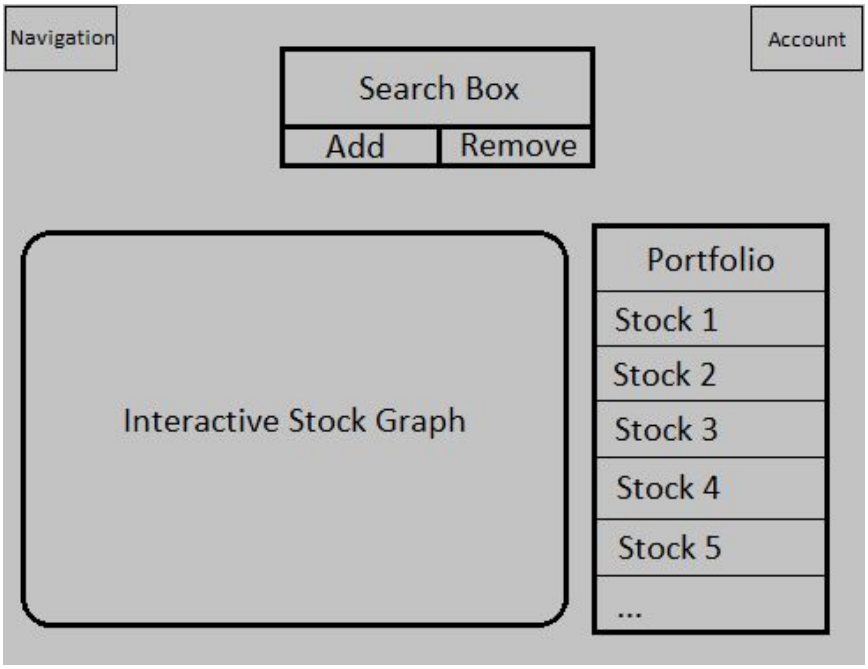

Figure 3.15

This page is intended to be a way for the User to save a series of Stock Graphs all in one place. These stock graphs would be saved in a database along with other pertinent user information; the user should be able to return to this page in between sessions, and have all the stocks listed still be there upon her return. The ideal portfolio will have a list of the stocks currently saved by the user, and will have an interactive Stock Graph for each stock saved. The Search Box will be used to add and remove stocks from the User's portfolio.

4. Functional Requirements Specification

Stakeholder

- Inexperienced
- Students
- Stock Advisors (Easily guide their clients through the process)
- Business Schools
- Business Classes
- Teachers/Professors (Demonstration Purposes)
- International Traders
- International Finance Workers

Actors and Goals

Actor	Actor's Goal	Use Case
User	To use the stock market learning tool to understand how the stock market works	UC-2, UC-6,UC-3, UC-4,UC-5, UC-4,UC-7, UC-9,UC-8
Plotly package (participating)	To plot information that is fetched from google & yahoo finances	UC-9
Yahoo/Google Finance (participating)	To provide information from past stock history to the database	UC-9
User Database (participating)	To display companies chosen by the user to be displayed on the home page.	UC-9, UC-1, UC-3

Figure 4.1

Use Cases

Casual Description

Use Case : Casual Text Description :: Corresponding requirements.

UC-1 : A user may want to login/logout from the site. Login to resume progress tracking and portfolio tracking; logout to end their session of progress tracking.
:: REQ-2, REQ-3, REQ-5, REQ-6, REQ-8

UC-2 : A user may want to use our learning service to be taught about the stock market in a traditional manner.
:: REQ-2, REQ-3, REQ-4, REQ-7, REQ-20

UC-3 : A user may want to have a specific list of companies kept track of so that each time they enter the site it loads that list for them.
:: REQ-2, REQ-3, REQ-7, REQ-9, REQ-10, REQ-15, REQ-21

UC-4 : A user may want all of the relevant info about how the prediction model works as well as news about the company so as to make a more informed decision.
:: REQ-11, REQ-12

UC-5 : A user may want their data presented with different options, such as not graphing predictions, for viewing aesthetics or to reduce information overload.
:: REQ-7, REQ-13, REQ-20

UC-6 : A user may want to register an account with our website so their individual progress and stock portfolio is saved across all sessions. (duplicate?)
:: REQ-2, REQ-3, REQ-5, REQ-6, REQ-8, REQ-20

UC-7 : A user may want to learn information in the learning system through alternative means that provides auditory and visual feedback.
:: REQ-2, REQ-3, REQ-20, REQ-22

UC-8 : A user may want to test their knowledge against what they have learned and be given feedback on their performance, either through the automated system or through retaking sections.
:: REQ-2, REQ-3, REQ-14, REQ-16, REQ-18, REQ-20

UC-9 : A user may want to easily search for a stock.
:: REQ-1, REQ-2, REQ-9, REQ-11, REQ-15, REQ-17, REQ-19, REQ-20, REQ-21

Use Case Diagram

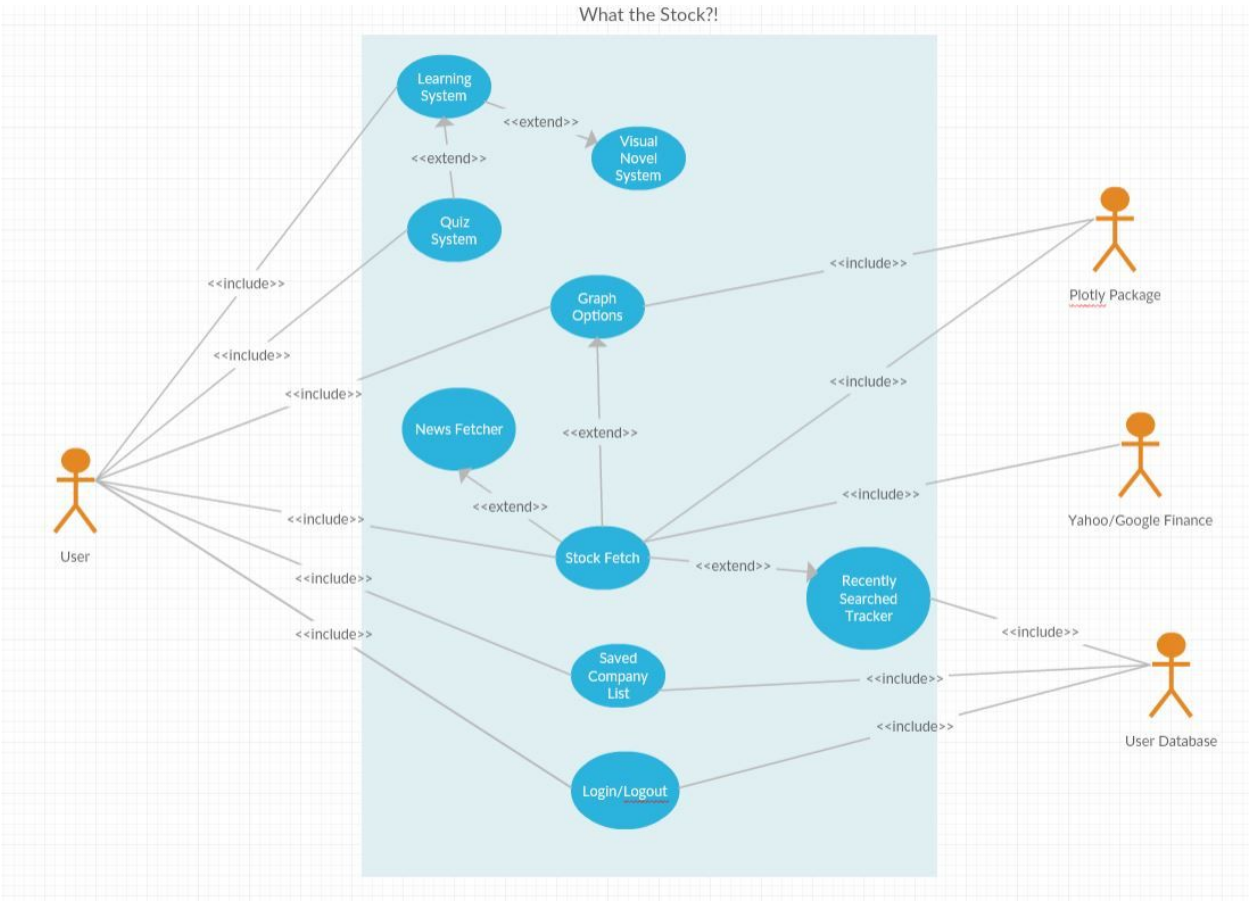

Figure 4.2

Traceability Matrix

REQ'T	Pts	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9
REQ-1	**3**									X
REQ-2	**1**	X	X	X			X	X	X	X
REQ-3	**1**	X	X	X			X	X	X	
REQ-4	**3**		X							
REQ-5	**1**	X					X			

REQ-6	**2**	X					X			
REQ-7	**2**		X	X		X				
REQ-8	**2**	X					X			
REQ-9	**2**			X						X
REQ-10	**1**			X						
REQ-11	**1**				X					X
REQ-12	**2**				X					
REQ-13	**2**					X				
REQ-14	**1**								X	
REQ-15	**2**			X						X
REQ-16	**3**								X	
REQ-17	**2**									X
REQ-18	**3**								X	
REQ-19	**1**									X
REQ-20	**3**		X			X	X	X	X	X
REQ-21	**1**			X						X
REQ-22	**1**							X		
Total Pts		***7***	***10***	***10***	***3***	***7***	***10***	***6***	***12***	***16***

Figure 4.3

Fully-Dressed Description

UC-9: Search

1) User Navigates to the menu bar
2) User Clicks on Search Page to get redirected to the stock search page
3) User clicks in box to type the company, the system will then try to autocomplete the company name and bring up company stock symbols relating to the user's input

4) After selecting a stock the page will update with the graph,prediction (LSA and RSI), accuracy, relative error, newsfeed and current stock ticker.
5) The user may then select another stock by repeating step 3) which will repeat the processes 3-4)
6) The user may also choose to navigate to a new page in which case the information will be cleared and searching for a stock will end

UC-8: Quiz

1) The user desires to test their acquired knowledge so they will navigate to the lessons tab in the menu bar
2) The user will read through the selected reading and/or take the visual novel 7)
3) The user may read the section over and click the finished reading button to proceed to the quiz. Note the user may leave but doing so is not covered in this use case
4) The user takes the quiz answering the questions leading to a score that will also give feedback on which sections needed clarification.
5) If the user passes the quiz they can retake the quiz; however, they may also proceed to the next section
6) If the user fails the quiz they will be given an option to reread the lesson, or the user may quit for the time being
7) If the user chooses the visual novel they will follow the onscreen dialogue
8) At the end they will select “finished reading”
9) This will take the user to the quiz which repeats 4)-6)
10) Once the user is satisfied with their score/performance they may quit or continue onto the next lesson

UC-3: Portfolio

1) The user will navigate to the registration tab and enter in the required credentials
2) Once the user has registered or logged in whichever is needed they can proceed to the portfolio page
3) The user will select the portfolio page from the menu on the left
4) User clicks in box to type the company, the system will then try to autocomplete the company name and bring up company stock symbols relating to the user's input
5) After selecting a stock the page will update with the graph, and prediction (LSA and RSI)
6) The user may continue to add to their portfolio by repeating steps 4-5 up to 5 times
7) The user may remove stocks from their portfolio by hitting remove after following step 4)
8) Afterward the user may leave the page and have their information stored
9) The user may return to their information by completing steps 1-3) or just step 3) if the user is already logged in

System Sequence Diagram

UC-9: Search

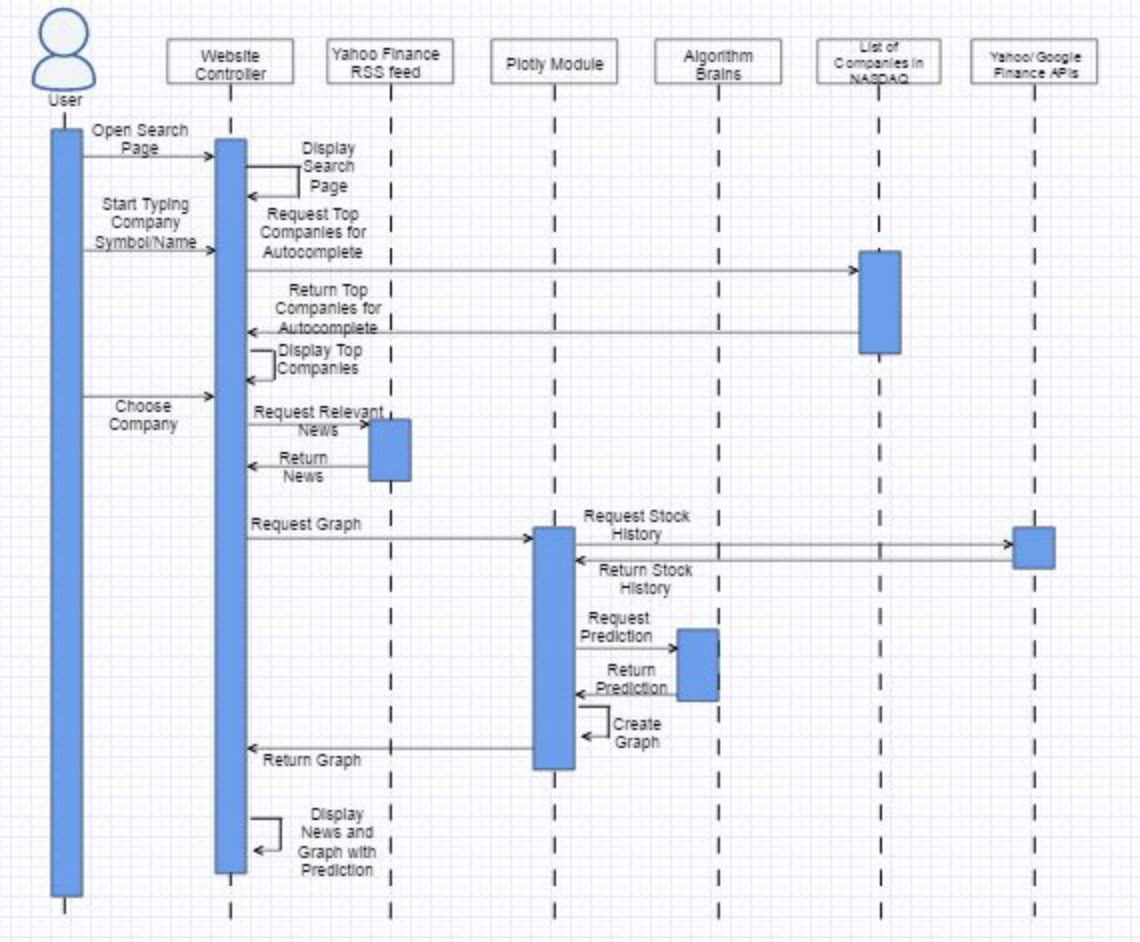

Figure 4.5 - UC-9 Sequence Diagram

Explanation

For UC-9, the main modules are the plotly module, algorithm brains, Yahoo! Finance RSS feed, Yahoo! Finance API, Google Finance API and the list of companies in the NASDAQ, which is in the form of a string..

In this use case, it is arbitrary whether or not the user is logged in to the system. After the user searches for a company, the website controller interacts with all the other modules to receive relevant data. The Plotly module uses historical data taken from the Yahoo! Finance API and current data from the Google Finance API. The Plotly module then interacts with the

algorithm brains to receive a prediction. The past data, present data and prediction information is all used by the Plotly module to create a graph that includes lines showing the company's stock history (dating back to 2010 the earliest), the current value, the algorithm brains past predictions and the algorithm brains prediction for the future of the stock. It can then pass the generated graph to the website interface to be displayed.

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.

Using the Yahoo! Finance RSS feed, the news fetcher receives the links and titles of the 3 most recent articles. These headlines are then sent and displayed on the web interface.

The last module for this case is the list of companies in the NASDAQ. As mentioned before this list is kept as a string. The list is used for autocomplete purposes. When entering a company name or stock symbol, the website controller interacts with the list and displays a list of companies that match the user's inputted string. NOTE: The autocomplete feature only represents companies that are part of the NASDAQ, however the user can search any company outside the NASDAQ as long as the exact stock symbol of the company is entered.

UC-8: Quiz

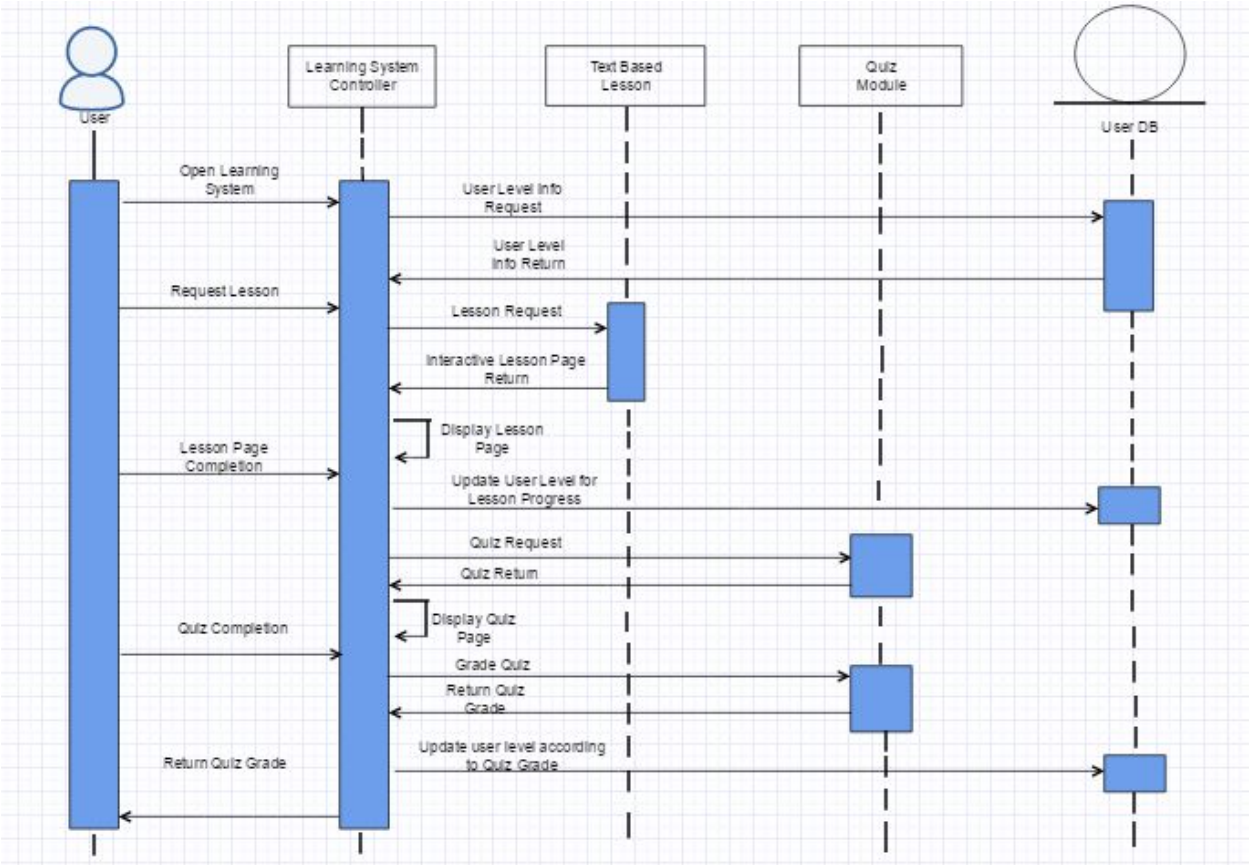

Figure 4.6(a) - UC-8 Sequence Diagram for Text Based Lesson Path

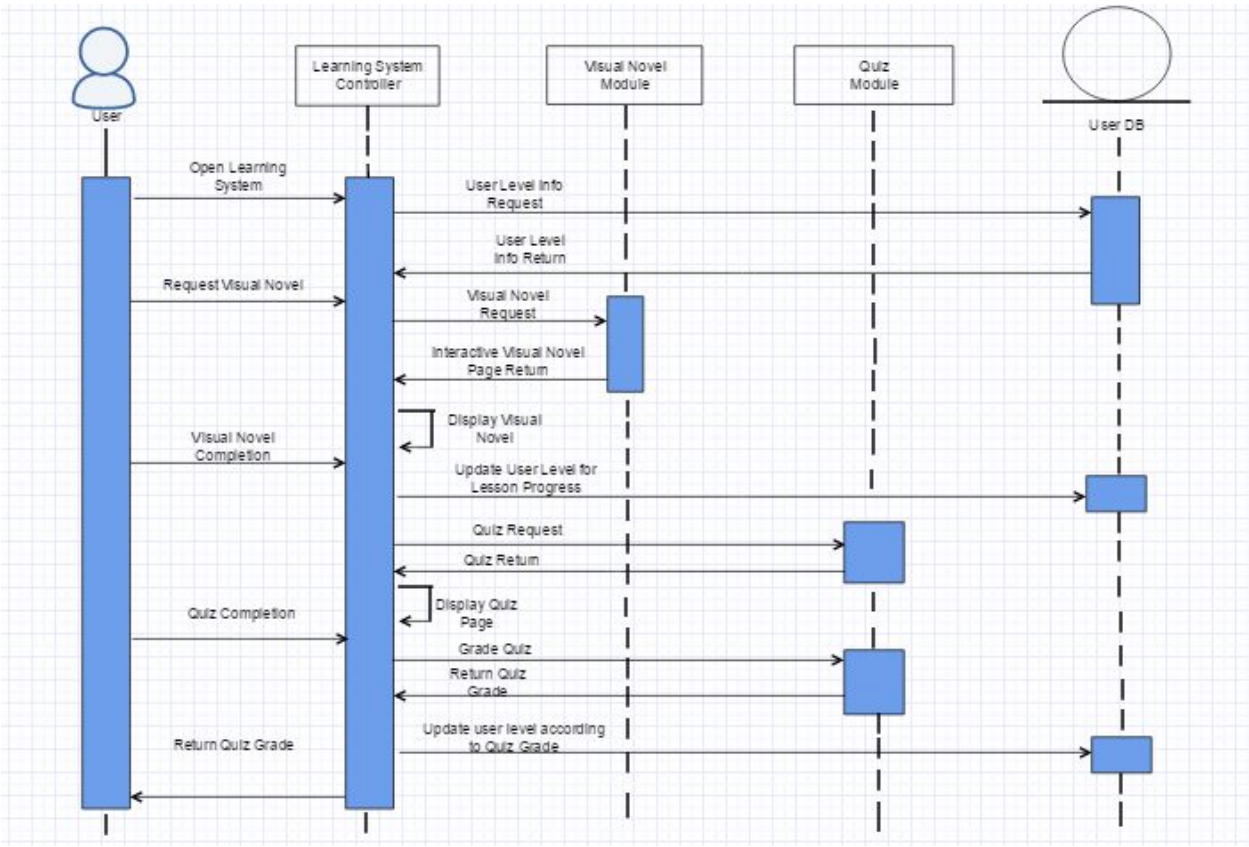

Figure 4.6(b) - UC-8 Sequence Diagram for Visual Novel Path

Explanation

The Learning System does not require a user to be logged in, but it is recommended. When not logged in, the user can access and “unlock” any of the tutorials, visual novels and quizzes (once either the tutorial or visual novel has been completed). However, the user’s progress will not be saved once the session has ended. Preferably, the user would be logged in to have access to this section of our system.

In this use case, the user may request to go through a text based lesson. The User Database stores information regarding the User’s progress in the learning system, 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. Once completed, any section of the tutorial module may be repeated at the User’s will.

Alternatively, the user may select to take a visual novel for the lesson.This action invokes the Visual Novel Module to display the visual novel for the given lesson. Again, the User Database stores information on the user’s progress in this Visual Novel. Once complete, the

user has access to the quiz for the lesson. Also any tutorial or visual novel can be repeated, endlessly, Only one of the entities needs to be complete for the given lesson's quiz to be available.

.

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. A key design choice for the system is that the user is not told which questions they got wrong on the quiz. This is in place so the user does not just immediately go back, retake the quiz and change their answers on their incorrect questions. However, the system does give hints to the user on which questions they got wrong. For example, "When re-reading this section, try to think about what benefits a company has to selling parts of itself to investors, and what benefits an investor has for giving his money to a company". This is so the user can go back and focus on that particular section before retaking the quiz.

UC-3: Portfolio

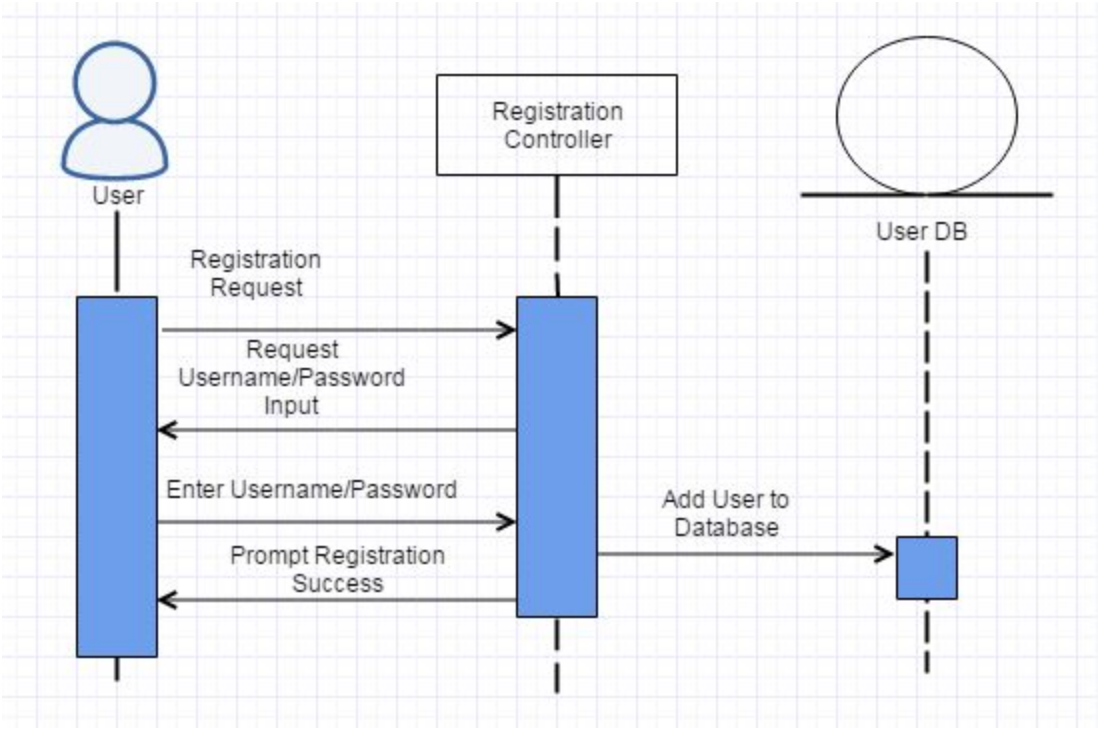

Figure 4.7(a) - UC-3 Sequence Diagram For Registration

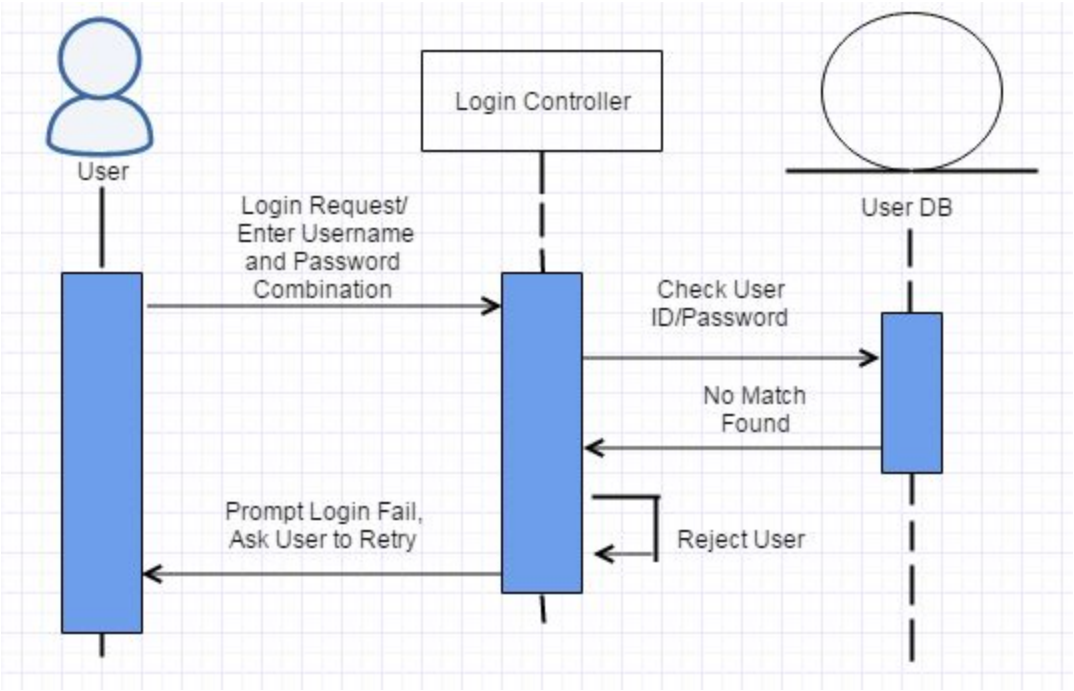

Figure 4.7(b) - UC-3 Sequence Diagram For Login Failure

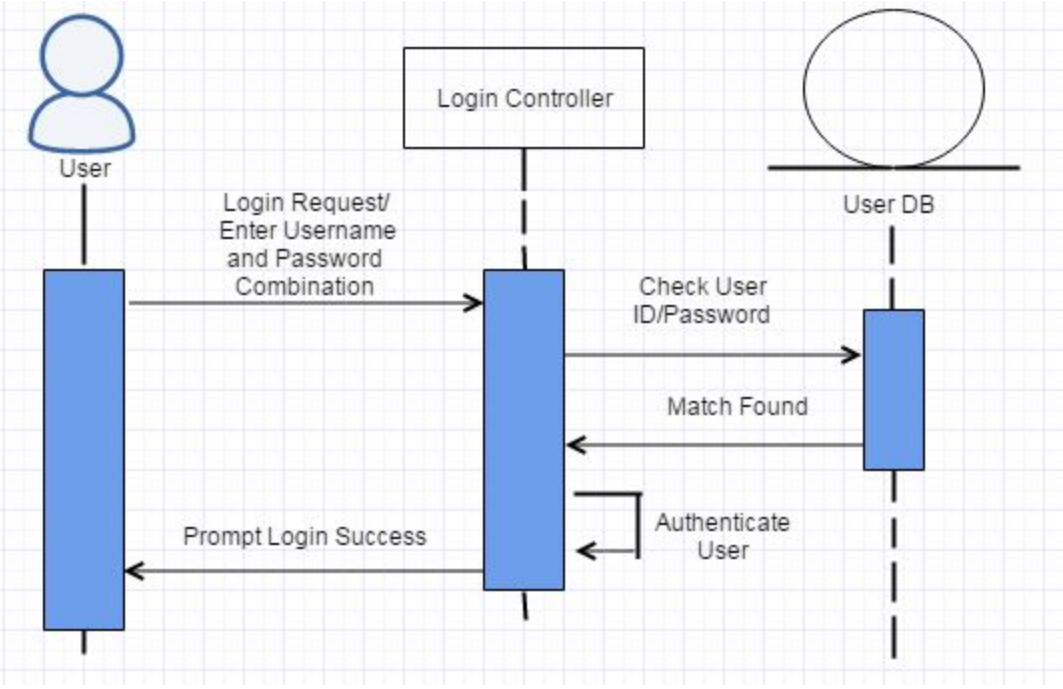

Figure 4.7(c) - UC-3 Sequence Diagram For Login Success

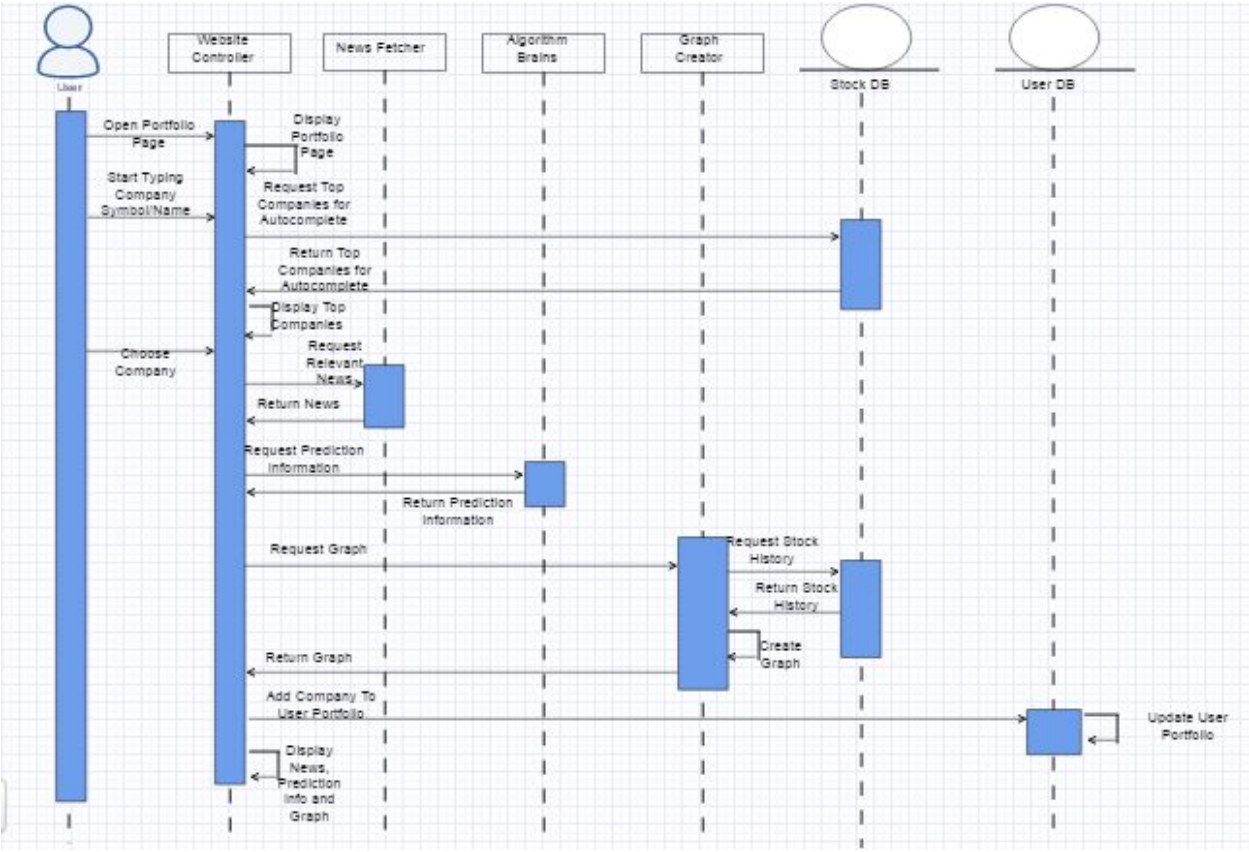

Figure 4.7(d) - UC-3 Sequence Diagram (This takes place after Figure 4.7(a) or Figure 4.7(c)

Explanation

The User Portfolio is the only part of the system that requires the user to be registered and logged in (*Figures 4.7(a) and 4.7(c)*). This is simply because there would be no portfolio to show without user authentication, since the user's portfolio is stored in the User Database.

The main modules of this UC are the News Fetcher, Graph Creator, Algorithm Brains, Stock Database and User Database. The Portfolio page is extremely similar to the search page. The user may search companies and symbols to get news, predictions and historical data displayed in graphs for the given company. The main difference is that the user can choose to add these companies to their Portfolio. This actions causes the user database to be updated, adding this company to their portfolio. Now, once the user opens the portfolio page, these companies that have been added are loaded immediately. This prevents the user from having to repeatedly search the same company every time the system is used. Of course, the user has the option to remove the company from their portfolio, as well.

5. Effort Estimation

Assumption: Let PF (Productivity Factor) be equal to 28 hours per UCP (Use Case Point).

Unadjusted Actor Weight

Actor	Actor's Goal	Category
User	Interacts with the system via GUI (Graphical User Interface).	Complex
Plotly package	A subsystem which interacts with system via an API.	Simple
Yahoo/Google Finance	A subsystem which interacts with system via an API.	Simple
User Database	A subsystem which interacts with system via network protocol (HTTP requests).	Average

Figure 5.1 - Actor Complexity Table

$$UAW\ (Unadjusted\ Actor\ Weight) = 3 \cdot Complex + 2 \cdot Average + 1 \cdot Simple$$

Calculations: $3(1) + 2(1) + 1(2) = 7$

$$UAW = 7$$

Unadjusted Use Case Weight

Use Case	Description	Category
Log (UC-1)	Average UI (User Interface). One participating actor (User Database). Four steps (login) or two steps (logout) for success scenario.	Average
Learn (UC-2)	Complex UI (User Interface). Three steps for success scenario.	Average
Track (UC-3)	Average and Complex UI (User Interface). One participating actor (User Database).	Complex

	Many steps for success scenario (depends on autocomplete).	
Information (UC-4)	Complex UI (User Interface). One step for success scenario.	Simple
InteractiveGraph (UC-5)	Complex UI (User Interface). Two participating actor (Plotly package, Yahoo/Google Finance). One to four steps for success scenario.	Average
Register (UC-6)	Average UI (User Interface). One participating actor (User Database). Four steps for success scenario.	Average
LearnVN (UC-7)	Complex UI (User Interface). Three steps for success scenario.	Average
Quiz (UC-8)	Complex UI (User Interface). One participating actor (User Database). Six to ten steps for success scenario.	Average
Search (UC-9)	Average and Complex UI (User Interface). Many steps for success scenario (depends on autocomplete).	Complex

Figure 5.2 - Use Case Table

$UUCW\ (Unadjusted\ Use\ Case\ Weight) = 15 \cdot Complex + 10 \cdot Average + 5 \cdot Simple$

Calculations: $15(2) + 10(6) + 5(1) = 95$

$UUCW = 95$

Adjusted Use Case Points

$UUCP\ (Adjusted\ Use\ Case\ Points) = UAW + UUCW$

Calculations: $7 + 95 = 102$

$UUCP = 102$

Technical Complexity Factor

Technical Factor	Description	Weight	Perceived Complexity	Product

T1	Distributed, Web-based system, because of Log (UC-1) and Track (UC-3).	2	5	$2(5) = 10$
T2	User expects exceptional response time, internet connectivity permitting.	2	4	$2(4) = 8$
T3	Cross screen dimensional portability is important to the user.	2	2	$2(2) = 4$
T4	Many concurrent users is required.	2	2	$2(2) = 4$
T5	Internal processing ranges from light to severe.	2	2	$2(2) = 4$
T6	User database easily expandable.	1	3	$1(3) = 3$
T7	Stock database easily expandable.	1	3	$1(3) = 3$
T8	No unique training required.	1	0	$1(0) = 0$
			$Total = \Sigma[W_i \cdot F_i]$:	36

Figure 5.3 - Technical Factor Table

$TCF\ (Technical\ Complexity\ Factor) = 0.6 + 0.01\Sigma[W_i \cdot F_i]$

Calculations: $0.6 + 0.01[36] = 0.96$

$TCF = 0.96$

Environmental Complexity Factor

Environmental Factor	Description	Weight	Perceived Impact	Product
E1	Team familiarity with most of the programming languages to be employed.	2	5	$2(5) = 10$
E2	Stable requirements expected.	1	5	$1(5) = 5$
E3	Most of the team is highly motivated, but some team members occasionally slack.	1	4	$1(4) = 4$
E4	Project not entirely object oriented (because of web-based components).	1	3	$1(3) = 3$
E5	Beginner familiarity with UML (Unified Modeling Language) based	1	2	$1(2) = 2$

	development.			
E6	Programming languages are of average difficulty are to be employed.	-1	3	$-1(3) = -3$
E7	Some familiarity with application problem.	2	3	$2(3) = 6$
			$Total = \Sigma[W_i \cdot F_i]$:	27

Figure 5.4 - Environmental Factor Table

$ECF\ (Environmental\ Complexity\ Factor) = 1.4 - 0.03\Sigma[W_i \cdot F_i]$

Calculations: $1.4 - 0.03[27] = 0.59$

$ECF = 0.59$

Use Case Points

$UCP\ (Use\ Case\ Points) = UUCP \cdot TCF \cdot ECF$

Calculations: $102 \cdot 0.96 \cdot 0.59 = 57.7728$

$UCP = 57.7728$

Duration Estimate

$Duration = UCP \cdot PF$

Calculations: $57.7728 \cdot 28 = 1617.6384$

$Duration = 1617.6384 \approx 1618$ hours

6. Domain Analysis

Domain Model

1. Components of Domain Model:

    Website Interface
    User Database
    Plotly Module
    Yahoo/Google Finance
    Prediction Algorithms
    Quiz System
    Learning System

Use Cases In Detail

UC-9: Search

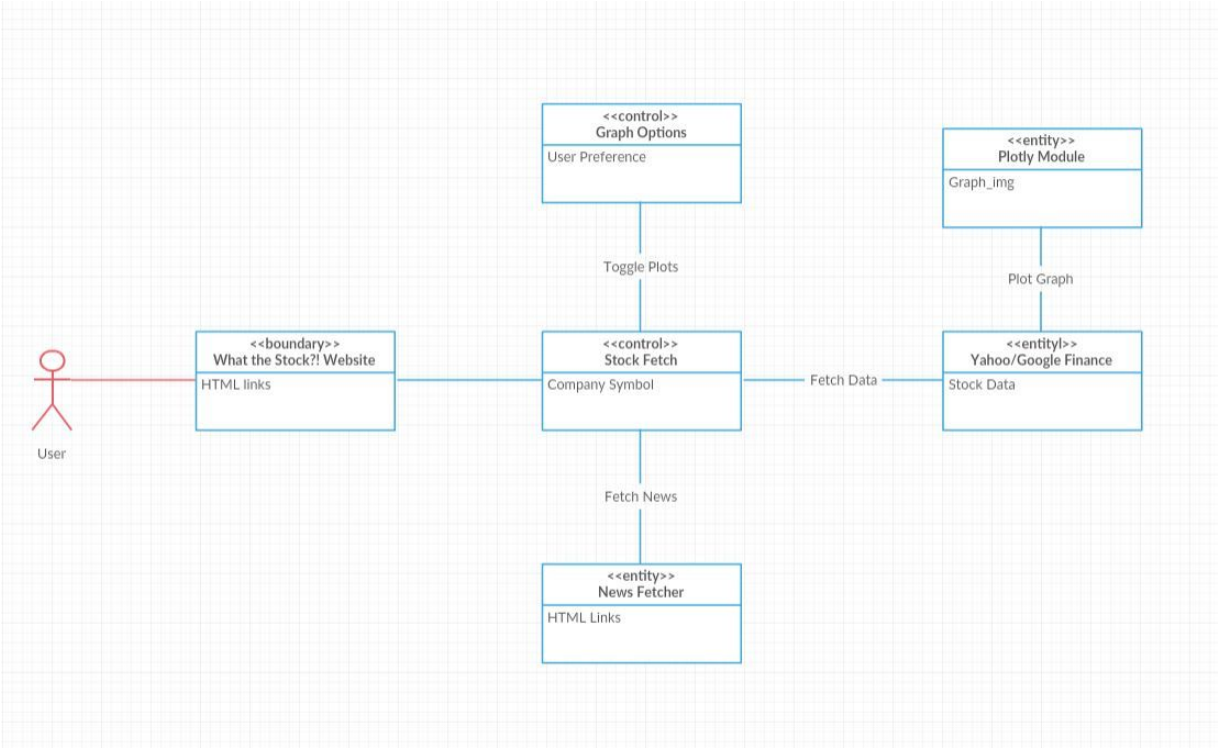

Figure 6.1

Explanation

The user interacts with the Stock Fetcher to lookup a company based on either their symbol or name through the autocomplete feature, which brings up a list of potential companies. Once a company is selected, the Yahoo/Google Finance module retrieves historical stock market value of the selected company and passes it to the Plotly module. Plotly then draws a graph and displays it to the user. Simultaneously, a request is sent to the news fetcher to retrieve the links to the latest articles of the selected company.

UC-8: Quiz

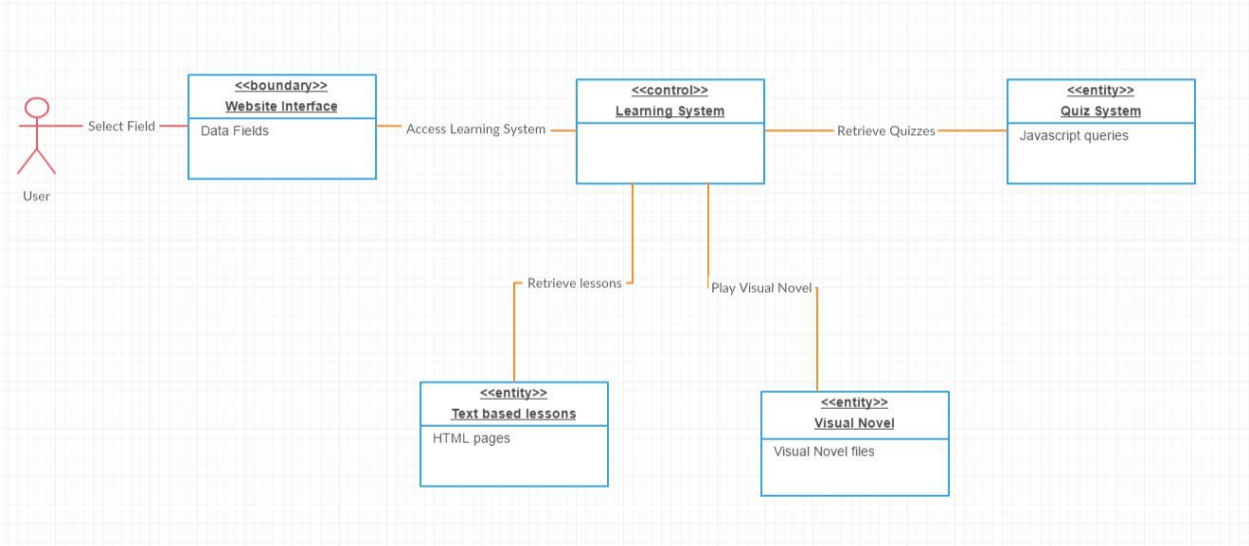

Figure 6.2

Explanation

User opens the learning system through the website interface. The Learning system has two methods to educate the user on the stock market, either through a visual novel system or through simple text based lessons. They can also take a quiz, or retake a quiz, to test their knowledge.

UC-3:

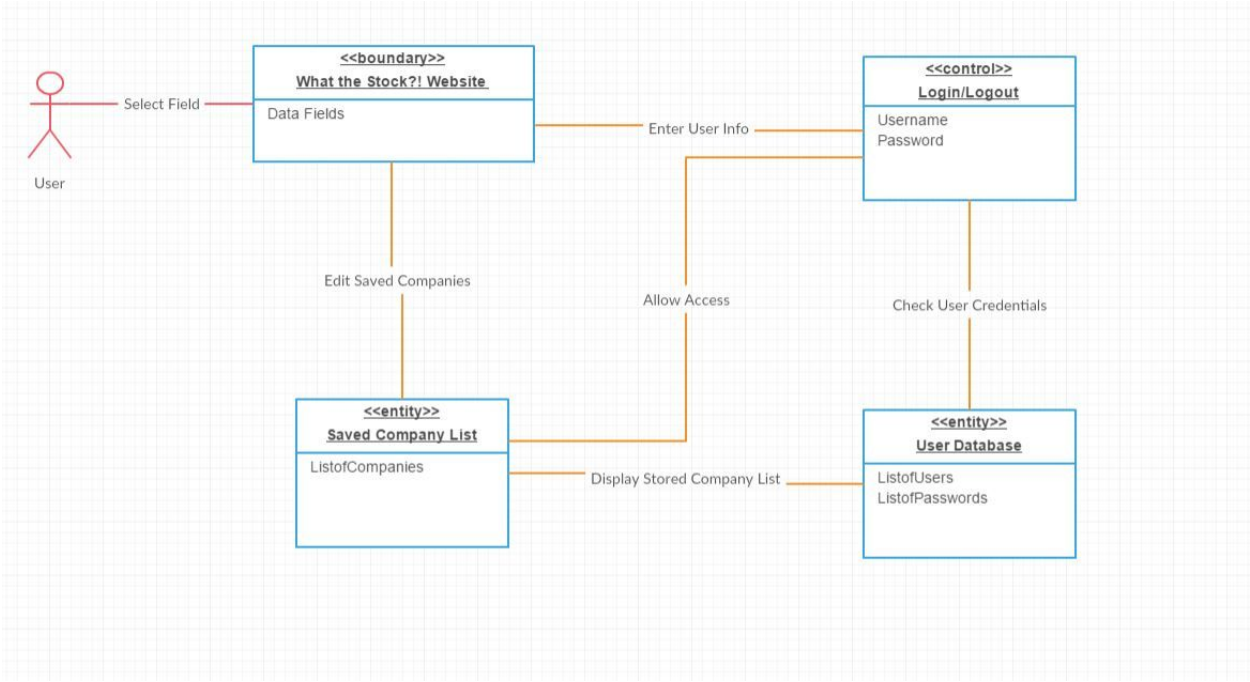

Figure 6.3

Explanation

The User enters his login information at the website interface. The entered info is checked with the stored data in user database. If he has not yet registered an account, he makes one. Then, once logged in, the system lets him access the saved company list that is tied individually to each user. The user can then add or remove companies to his portfolio (saved company list)

Concept Definitions

Login/Logout - <<control>>

Verifies the user's information

What the Stock Website Interface- <<boundary>>

Front page of our website with multiple fields/links to allow the user access to parts of our program

User - Interacts with <<boundary>>

Components to navigate the website or app and gain access to our services

News Fetcher - <<entity/worker>>

Fetches news article links from Google Finance news

Yahoo/Google Finance - <<entity/worker>>

Retrieves information about a stock from Google/Yahoo Finances records

User Database - <<entity/thing>>

Stores a list of users and their credentials in the database

Learning System - <<entity/thing>>

Contains both text based and Visual lessons about the stock market

Stock Fetcher - <<entity/control>>

Sends a command to Google/Yahoo finance APIs to retrieve data while simultaneously sending a command to News Fetcher to retrieve latest news articles. Toggles on Certain Plots for the user to interact with

Plotly Module - <<entity/thing>>

Plots a graph of the fetched stock information as an image

Graph Options - <<entity/thing>>

Allows the user to toggle certain plots on or off and allows them to zoom in the graph for better viewing consistency

Association Definitions

Concept Pair	Association Description	Association Name
User ↔ Website interface	User accesses the website interface which has various fields/links	Select Field
Website Interface ↔ Stock Fetcher	The website sends a request to the stock fetcher to retrieve information about a company	Retrieve Stock info
Website Interface ↔ Login/Logout	The website sends the information entered by the user to the control system to verify their credentials	Enter User info
Plotly module ↔ Google/Yahoo finance	The stock information retrieved from yahoo/google finance is sent to plotly	Plot Graph

Learning System ↔ Quiz System	Recommends the corresponding material to the user based on their progress	Retrieve Quiz
Login/Logout ↔ User Database	Checks the information entered by the user with that of the previously stored information from the database	Check User Credentials
Stock Fetcher ↔ Yahoo/Google Finances	Sends a command to Google/Yahoo finances to fetch stock information from a single company	Fetch Data
Stock Fetcher ↔ Graph options	Sends a command that modifies how the Stock information is represented	Toggle Plots
Website Interface ↔ Saved Company List	Makes changes to the Saved Company List (portfolio) based on the users edits	Edit Saved Companies
User Database ↔ Saved Company List	Retrieves information stored by the user in the previous session	Display Stored Company List

Figure 6.4

Attribute Definitions

Concept	Attributes	Attribute Description
Website Interface	Data Fields	Sends requests to different part of the website based on selection
Stock Fetch	Company Symbol	Contains the NASDAQ symbol of a company
Graph options	User Preference	Contains the choice to toggle certain plots of the stock graph of the selected company/ zoom in
News Fetcher	HTML links	Contains links that redirect the user to latest news articles of the selected company
Quiz System	Javascript queries	Contains queries that mimic multiple choice questions for the user to select

Text Based Lessons	HTML pages	Contains html pages with neatly formatted paragraphs that explain how the stock market works
Visual Novel system	Visual novel files	Each visual novel file contains an interactive classroom style lesson
User Database	ListofUsers	Contains a string that represents the username
	ListofPasswords	Contains a string that represents the password
Saved Company List	ListofCompanies	Contains a list of strings that each represent a company symbol

Figure 6.5

Traceability Matrix

Domain Concept	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9
User	x	x	x	x	x	x	x	x	x
Website Interface	x	x	x	x	x	x	x	x	x
Learning System		x					x	x	
User Database			x			x			
News Fetcher				x					
Plotly Module					x				
Login/Logout	x								
Stock Fetcher									x
Quiz System								x	
Visual Novel System							x		
Text Based Lessons		x							

Saved Company List (Portfolio)			x						

Figure 6.6

System Operation Contract

UC-9: Search

Operation	getCompanies()
Pre-conditions	- Stock DB is running
Post-conditions	- Company list returned

Figure 6.7

Operation	getNews(ticker)
Pre-conditions	- Company exists - Yahoo Finance API is working
Post-conditions	- News hrefs returned if company exists, otherwise “false” returned

Figure 6.8

Operation	getGraph(ticker)
Pre-conditions	- Plotly API is working - Yahoo Finance API is working - Company exists - Stock DB is running
Post-conditions	- Graph iframe returned if company exists, otherwise “false” returned

Figure 6.9

Operation	getAcc(ticker)
Pre-conditions	- Company exists - Yahoo Finance API is working

Post-conditions	- Accuracy returned if company exists, otherwise “false” returned

Figure 6.10

Operation	getRelAcc(ticker)
Pre-conditions	- Company exists - Yahoo Finance API is working
Post-conditions	- Relative Accuracy returned if company exists, otherwise “false” returned

Figure 6.11

UC-8: Quiz

Operation	getLevel(sessionID)
Pre-conditions	- The userDB is running
Post-conditions	- If a user with the sessionID exists, their level is set and “true” is returned - Else “false” is returned

Figure 6.12

Operation	getLesson(sessionID, index)
Pre-conditions	- The userDB is running
Post-conditions	- If a user with the sessionID exists, their lesson value at the index is returned

Figure 6.13

Operation	setLesson(sessionID, index, val)
Pre-conditions	- The userDB is running
Post-conditions	- If a user with the sessionID exists, their lesson flag at the index is set and “true” is returned - Else “false” is returned

Figure 6.14

Operation	getQuiz(sessionID, index)

Pre-conditions	- The userDB is running
Post-conditions	- If a user with the sessionID exists, their quiz value at the index is returned - Else “false” is returned

Figure 6.15

Operation	setQuiz(sessionID, index, val)
Pre-conditions	- The userDB is running
Post-conditions	- If a user with the sessionID exists, their quiz flag at the index is set and “true” is returned - Else “false” is returned

Figure 6.16

Operation	setLevel(sessionID, index, val)
Pre-conditions	- The userDB is running
Post-conditions	- If a user with the sessionID exists, their leve is set and “true” is returned - Else “false” is returned

Figure 6.17

UC-3: Portfolio

Operation	getPort(sessionID)
Pre-conditions	- The userDB is running
Post-conditions	- If a user with the sessionID exists, their ticker at the index in their portfolio is returned - Else “false” is returned

Figure 6.18

Operation	addPort(sessionID, ticker)
Pre-conditions	- The userDB is running
Post-conditions	- If a user with the sessionID exists, the ticker is added to the user’s portfolio so long as their portfolio has <5 tickers in it already and

	“true” is returned - Else “false” is returned

Figure 6.19

Operation	getGraph(ticker)
Pre-conditions	- Plotly API is working - Yahoo Finance API is working - Company exists - Stock DB is running
Post-conditions	- Graph iframe returned if company exists, otherwise “false” returned

Figure 6.20

Operation	remPort(sessionID, ticker)
Pre-conditions	- The userDB is running
Post-conditions	- If a user with the sessionID exist and they have a stock in their portfolio with the given ticker, it is removed and “true” is returned - Else “false” is returned

Figure 6.21

Mathematical Model

LSA (Least Square Approximation)

Our method of approach to predicting the value of a stock consists of using least square approximation to predict the incline or decline of a particular stock. The model works in the following steps:

1) Determining the form of the already plotted graph this consists of finding the shape of the current graph. Here shape consists of what degree polynomial fits best for the current graph. Most likely, the graphs will fit best about a polynomial of At^10+Bt^9+Ct^8…+G=H. The way the program will determine this is to simply use the average price of a stock as a makeshift x-axis. Then based on how many times it crosses this axis (as determined by the data points) we can estimate the degree of the figure.

2) Determine our data point set. Basically, depending on the time span chosen by our user (This comes from how far back the user wishes to see up to a certain value for computational purposes). The standard would follow along with the lines of:
    a) 45 Day Period ~ 30 Work Days +- 3 days (holidays/weekends)

3) Determining the current trend or most notably recent change. This method consists of looking at how the stock has performed most recently whether it has increased or decreased since last checked. This kind of process would occur every 15 minutes and compare the current change with the previous one 15 minutes ago. This way we can crudely determine whether or not the price has increased or decreased.

4) Pf-PO=ΔP if ΔP >0 we have and increase if Δ<0 we have a decrease.
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 [a0;a1;a2]. a0 is c in aforementioned part one, a1 is b, and a2 is a. These values are computed by solving a transformation matrix (CTC)-1CTy. 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. V1=[1;1;1;1…1],V2=[x1,x2,x3,x4 …….xn],V3=[x12,x22,x32,x42 ……xn2],Vn=[.....]

5) We then find the point matching that line at the point in time in the next day; Thus, we find an approximation that predicts the value of the stock in the next day. This graph will be updated every 15 minutes to determine the future price with increased accuracy. This process will continue and with a larger time span, the accuracy will decrease overall since the line is more general and random outliers that are caused by unpredictable events are unpredictable.

6) Accuracy or confidence. If the stock has been showing a decline then we assume that the future will hold the same depending on the value of ΔP. The system will compare its approximation with that of the delta P and based on how to the close are in both magnitudes and sign the system will reflect that as an accuracy (Preliminary).

An example of the output:

Time	Price
0	110
1	118
2	92
3	48

3.5	7

Figure 6.22

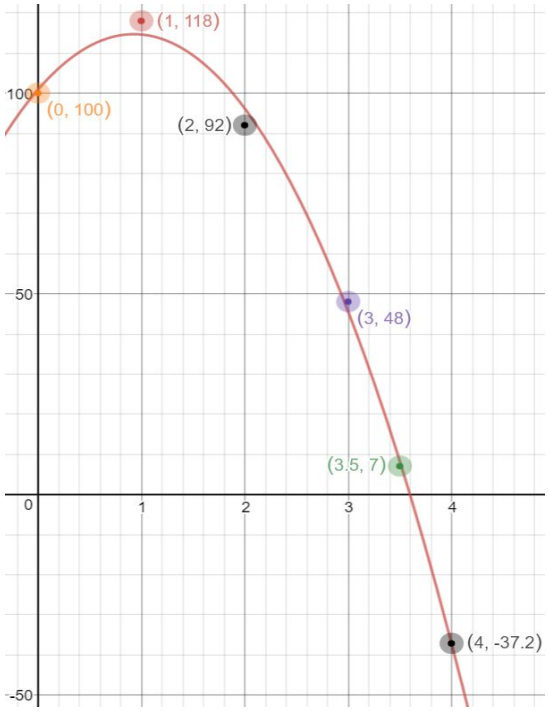

Figure 6.23

C=[1,0,0];[1,1,1];[1,2,4];[1,3,9];[1,3.5]
y=[100];[118];[92];[48];[7]
Solution: a_0= 101.00
a_1= 29.77
a_2=-16.11
Here the black dot represents the approximate value after time hits 4

Note: When designing this function the use of the 10th power for precision is derived from the fact that higher powers require more computing time and more storage space. After testing around with values over a ~30 day period the conclusion was drawn that on average the graph would pass an “x-axis” less than 10 times in a monthly period. This is also why this algorithm handles small changing stock values better. Sportatic data that fluctuates rapidly is not ideal for this type of prediction; however, most of our users are not looking for fast paced real time data on penny stock or stocks with high volatility, so this algorithm is ideal. Another design choice was to offset the data by about a day so that users would not get confused about overlapping prediction data and past data. This is not an issue for clarity since plotly can “hide” certain plots from the user; therefore. the shift does not affect the user's understanding.

RSI (Relative Strength Index)

Function receives a list Data of price-time stock data with 14 or more elements.

RSI will be applied on a time Period = (size(Data) - 14).

From this list, we create a list Price_Changes containing the differences in the price between each day.

We then create two more lists, Gains and Losses. Gains contains (positive values from Price_Changes), and zero values in the other elements. Losses contains absolute(negative values from Price_Changes), and zero values in the other elements.

For each day in the RSI period:

Calculate smoothed moving average (SMA) to smooth our Data with the formula: newval = (prevval * (period - 1) + newdata) / period, we do this in the next two steps for the data contained in our Gains and Losses lists:

Calculate avgUp = (avgUp * (Period - 1) + Gains[nextElement]) / Period
Calculate avgDown = (avgDown * (Period - 1) + Losses[nextElement]) / Period

Use the above two values to calculate Relative Strength (RS) = avgUp / avgDown

Calculate the RSI value (100 - 100(1 + RS)) and append into an RSI list

Return RSI list.

An example of the output is a 14-entry list containing floats (ranging from 0 to 100) such as:

50	40	45	67	70	80	90	45	34	34	46	6	20	56

Figure 6.24

Where each entry is treated as a percentage and is plotted onto a separate graph than the price plot. This creates a graph such as (where the yellow line at the bottom is the RSI data):


Past Data for AMD
RSI for AMD
Dollars ($)
14
12
10
Jan 8
2017
Jan 22
Feb 5
Feb 19
Mar 5
Mar 19
Apr 2
Apr 16
RSI percentage (%)
100
80
60
40
20
0
Jan 8
2017
Jan 22
Feb 5
Feb 19
Mar 5
Mar 19
Apr 2
Apr 16

Figure 6.25

7. Interaction Diagrams

Design Patterns Used: Command.

This Design Pattern means that one class will serve as a command class and will contain commands and can be called to execute the command on another class (send the command). One class that may want another class to perform an operation can create a command object and load it with an instruction, and the send it off to perform its requested operation. Commands may also return asynchronously when the desired request is fulfilled. This helps simplify code on the “manager” class by separating when to call a command from when to load it with parameters (such as instructions to execute). A “custodian” class may also prepare the command instead of the “manager” in order to minimize the workload on the “manager” class. However, because our code does not utilize a “custodian” class we will just include the “Manager” and “Command” classes in our interaction diagrams. In our frontend code, the Javascript code located in userHelpers.js and plotHelpers.js can create XMLHTTPRequests as commands and loads them with instructions.

Example: Website Requests Backend RSI Resource

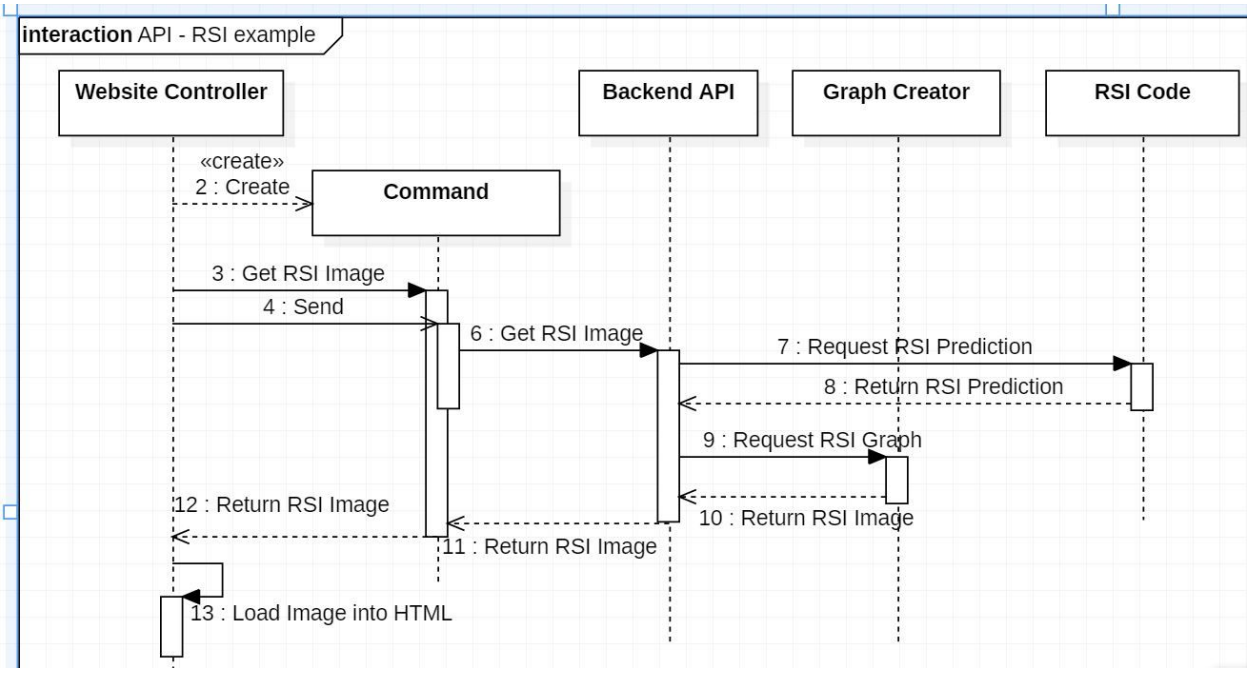

Figure 7.1

Explanation

Our website can directly access our backend API. Our API has direct control over prediction, graphing, fetching, and database functions. Our API can be accessed through an XMLHTTPRequest command from our javascript frontend. An onload function can be attached to this command, which can execute then asynchronously. Once this command is sent, our API can detect this command and execute various functions according to the message specified within the command.

Example: Website Wants to Store Data Into the User Database

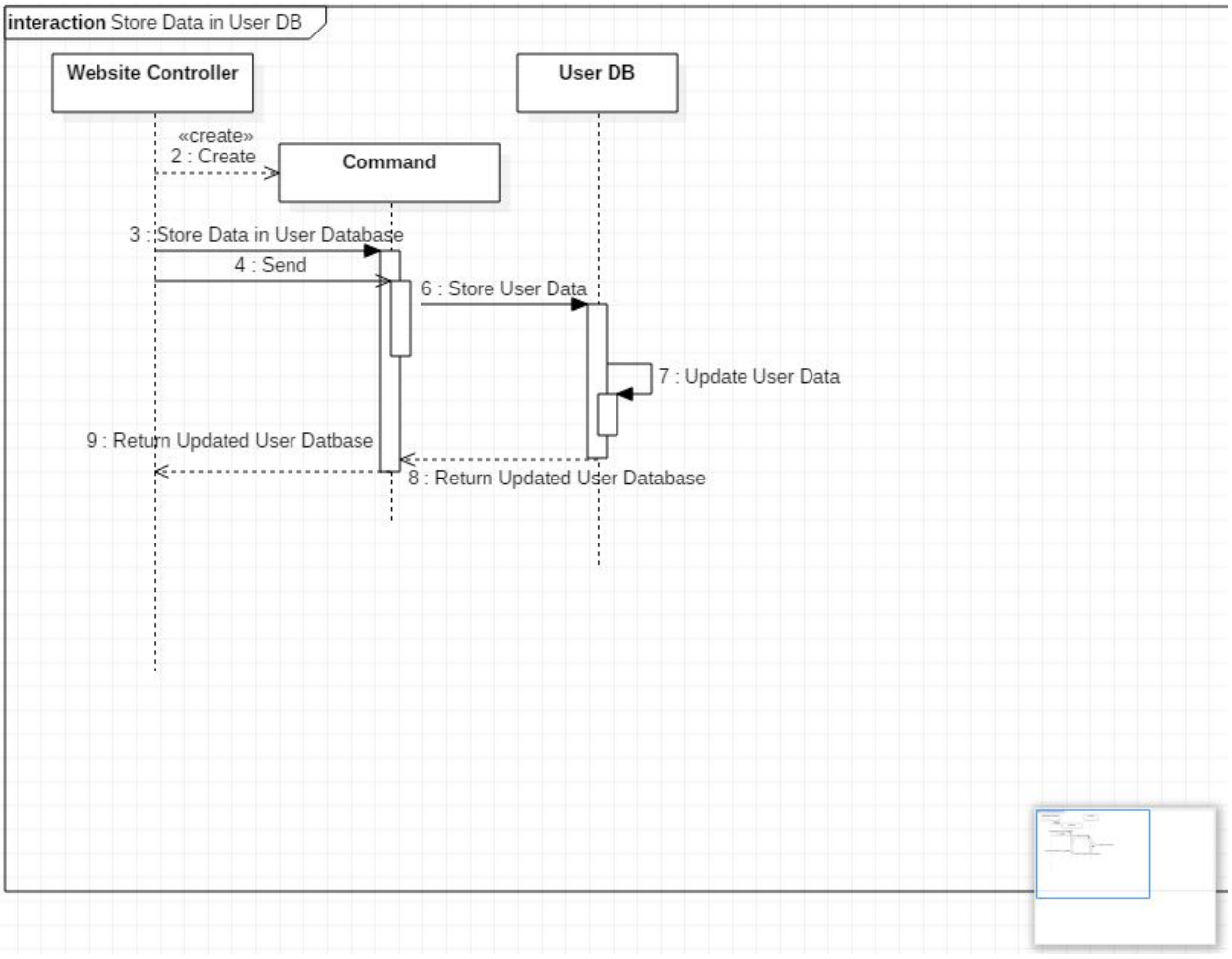

Figure 7.2

Explanation

The website controller can access the backend user database, as well. The user database stores data on usernames, passwords, completed lessons/quizzes and user portfolio.

The website controller stores data due to a variety of circumstances. These include: registration of a user, the registration of a user, tutorial completion, visual novel completion, the passing of a quiz and the addition of a stock to the user portfolio.

Example: Adding a new stock to the Stock Cache

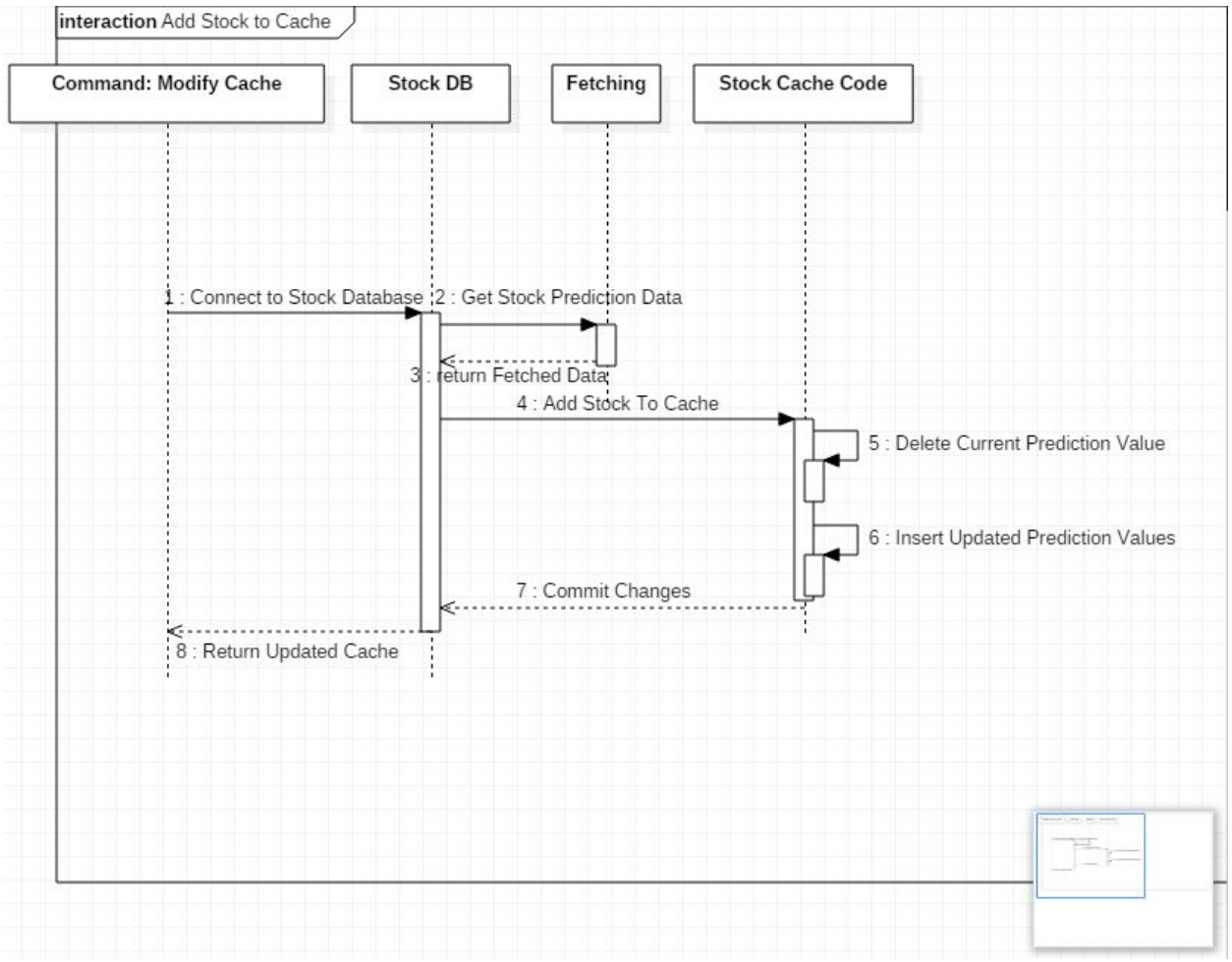

Figure 7.3

Explanation

This diagram displays how the stock cache can be updated. To do this, a connection to the Stock Database is made. From the stock database, prediction values are fetched. These prediction values are then sent to the stock cache code, where the old data is removed from the cache and the new data is put in the old data's place.

It is important to note that the stock cache can only contain one hundred stocks after it will start deleting old entries and replacing them with new entries according to Least Recently Used replacement.

Interaction Example: Prediction Algorithm

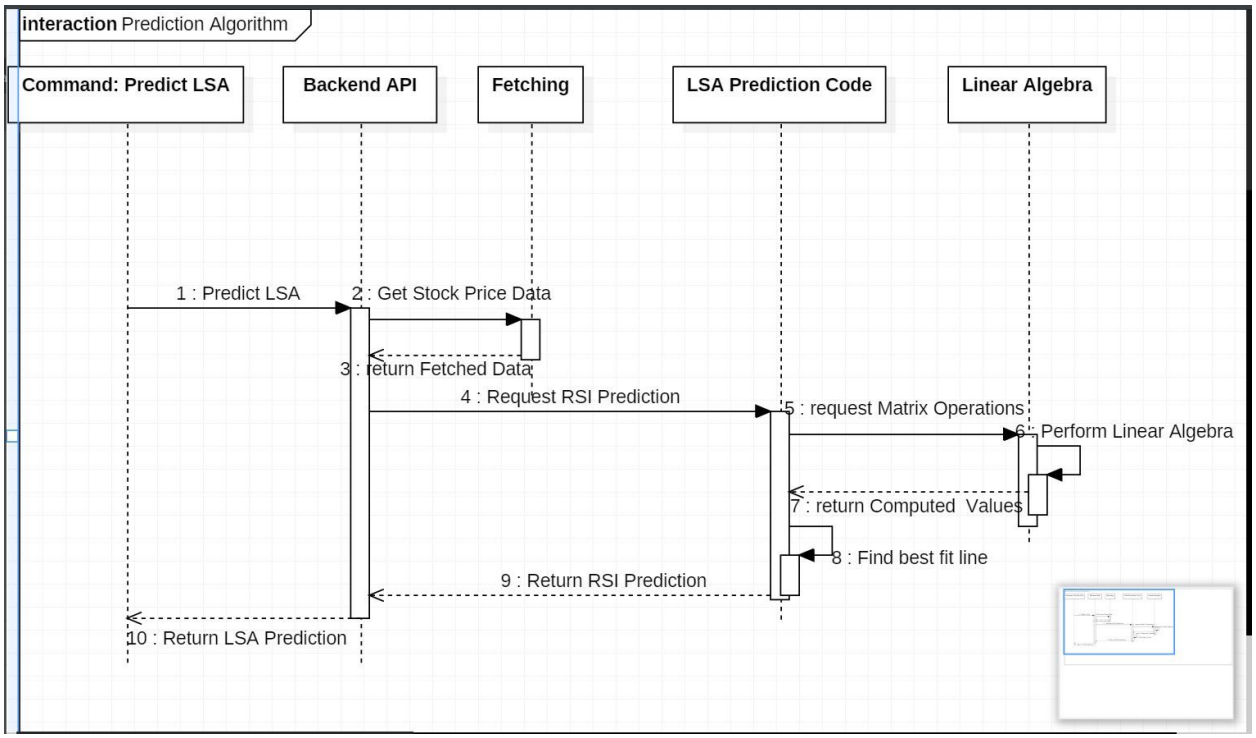

Figure 7.4

Explanation

This diagram focuses on the backend interaction and chooses to not show the frontend website controller to simplify the diagram. Thus, it assumes a command has already been created. The command is to get the LSA stock prediction algorithm to create a line of best fit for the stock being predicted.

Here we again use command architecture to enable communication between the separated frontend and backend. As this example is more focused on backend, it will include the fetching code to grab stock data and linear algebra code to show that the LSA algorithm is more than a single function, but instead multiple functions that depend on one another's output to create a line of best fit. In this case, linear algebra is used to manipulate matrices that contain stock price data to create a curve to the power of 10. To show the many functions that are executing, self calls are labeled on the diagram to show an unlisted function of one class calling another unlisted function. These functions were not listed to simplify the diagram.

This LSA data is then returned to the same backend API controller that called fetching and started the prediction. This controller then in turn returns the data to the command. From

here, the command would return to the front end, or, not shown for reasons of simplicity, call graphing to create a price prediction graph.

Use Case 1: Login/Logout

Sequence Diagram

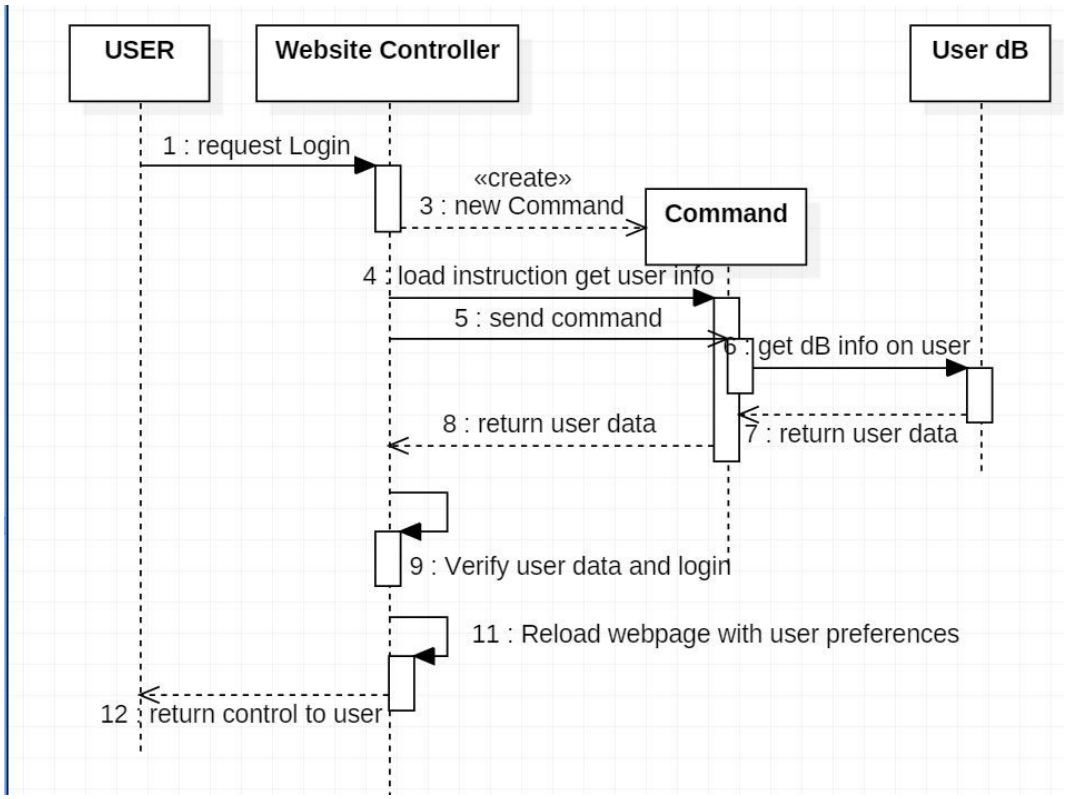

Figure 7.5

Explanation

Note: this diagram does not show the logout controller, although logging out would be a simple interaction between only the user and the website frontend (website controller).

For UC-1, the main modules are the website frontend, the command used to fetch user data, and the user database that contains a user's username and password and website preferences. When a user tries to log in, they input their username and a password into the website which can then check if these parameters match what is stored in the database. The command is loaded with an instruction to fetch a user's data, and it executes this instruction when it is subsequently sent. The user database returns the user's website preferences and

checks if the username and password are correct and then logs the user into the website and reloads the page.

Use Case 2: Scroll Lesson

Sequence Diagram

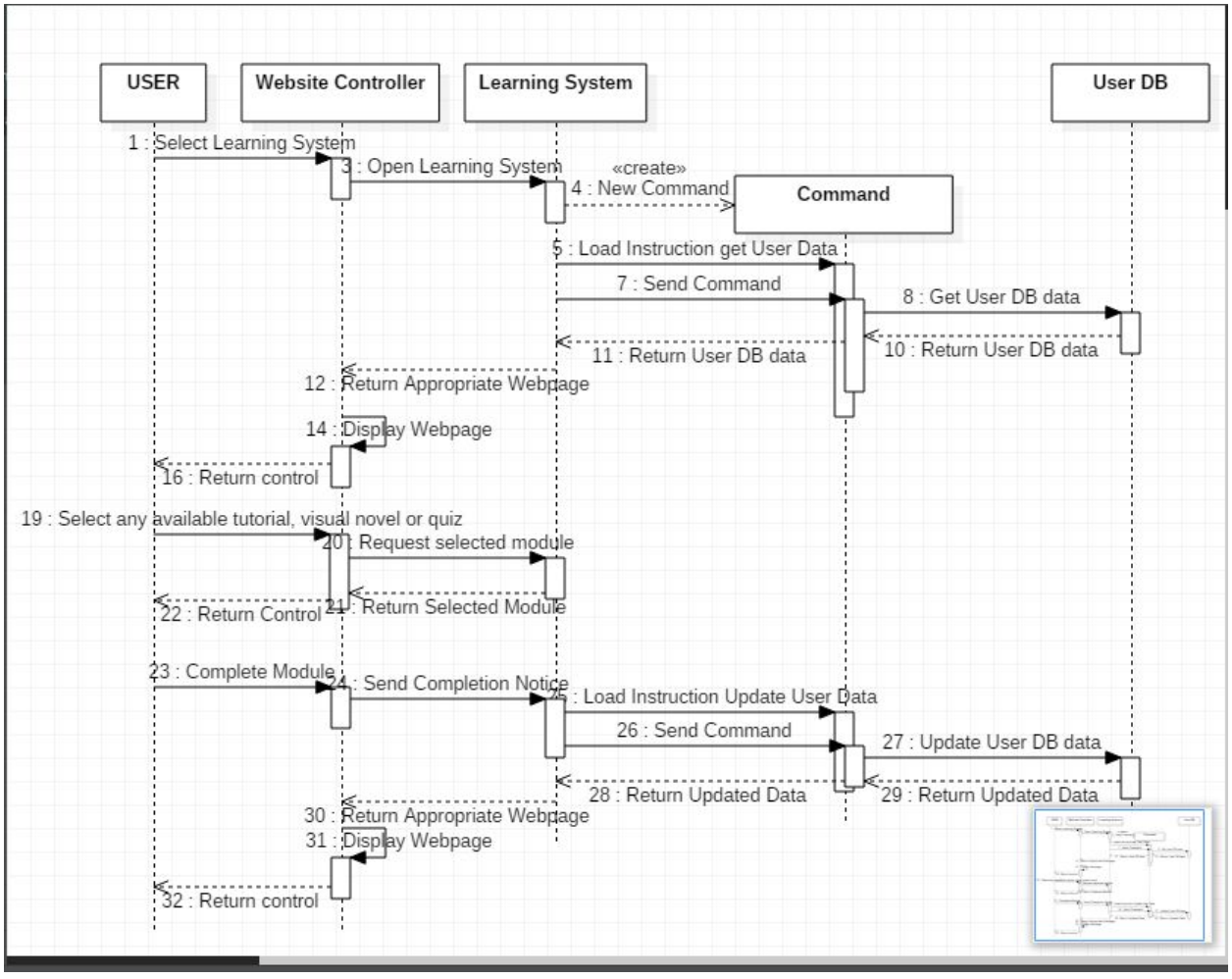

Figure 7.6

Explanation

NOTE: It is assumed that the user has logged on before accessing the learning system in the diagram. User authentication is required for access to the user database, which monitors individual progress in the learning system.

This UC the main feature of our system. The learning system is in place to educate users about the stock market. When the user chooses to open the learning system, the website controller processes this request and does so accordingly. The learning system module fetches data from the user database to examine the user's progress in the system. Each tutorial, visual novel and quiz will not be available until the required credentials are met. All of this information is stored within the user database. Once this information is fetched, the learning system can determine what lesson the user is on and which areas of the system the user has access to. The website controller loads the page accordingly and returns control to the user.

The user can now select any tutorial, visual novel or quiz that is available to them. The website controller processes this request, as well. The learning system returns the appropriate page to the controller, which loads the page. The user completes the given module requested and notifies the system when he or she finishes. This information is sent back to the learning system module. If the user has finished reading a quiz or a visual novel, the learning system module updates the user database accordingly and allows the user to take the quiz for the given lesson. If the user has completed a quiz, the learning system module grades the quiz, updates the user database accordingly and returns the user's score. If the user has passed the quiz, the next lesson becomes available. If not, the user is prompted to go over the lesson again and focus on certain aspects. The grading and returning of the quiz is not indicated in the diagram, but is simply just a function of the learning system module and a return prompt to the user.

Use Case 4: Information

Sequence Diagram

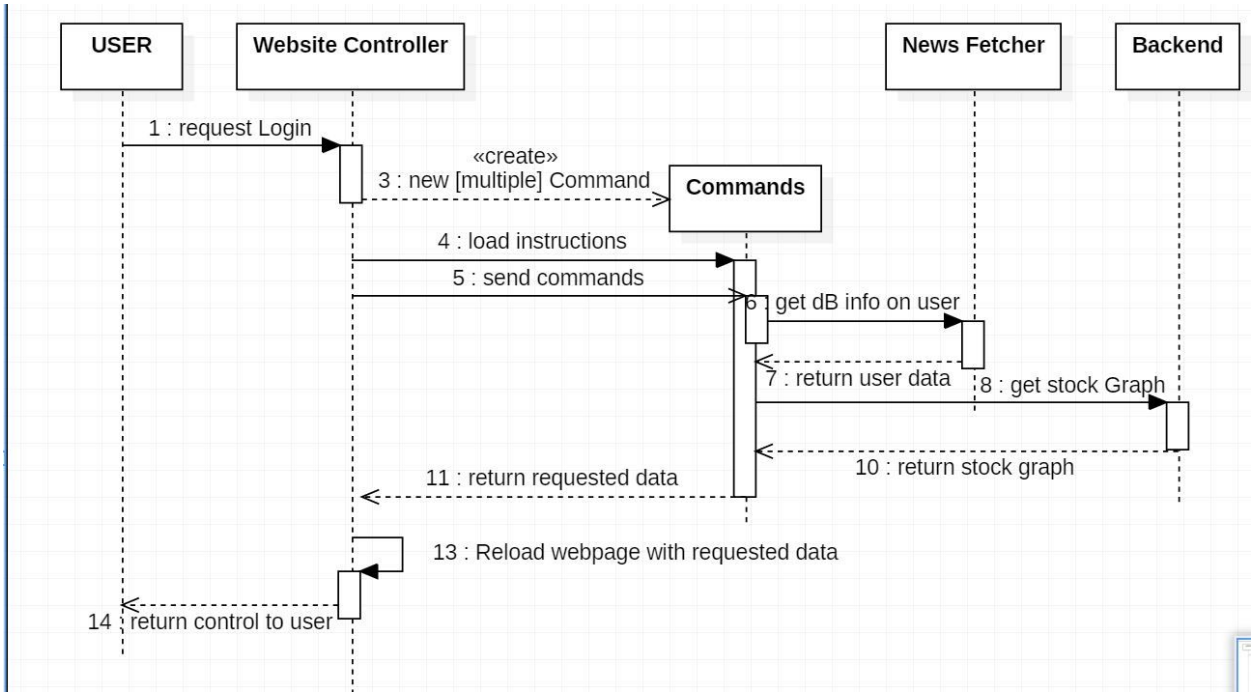

Figure 7.7

Explanation

Note: this interaction diagram has been simplified to include only a list of commands instead of going in-depth into the structure of each command. This is because there are multiple commands being created relatively quickly and it would create clutter in the diagram.

Here the backend class includes the stock prediction, graphing, stock database, and confidence algorithms. This would take multiple commands instead of just one as portrayed in the diagram, as it would return multiple datasets including the prediction confidence and a stock price graph.

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, in the form of a another web page explaining the algorithm as documented by the backend team.

8. 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()

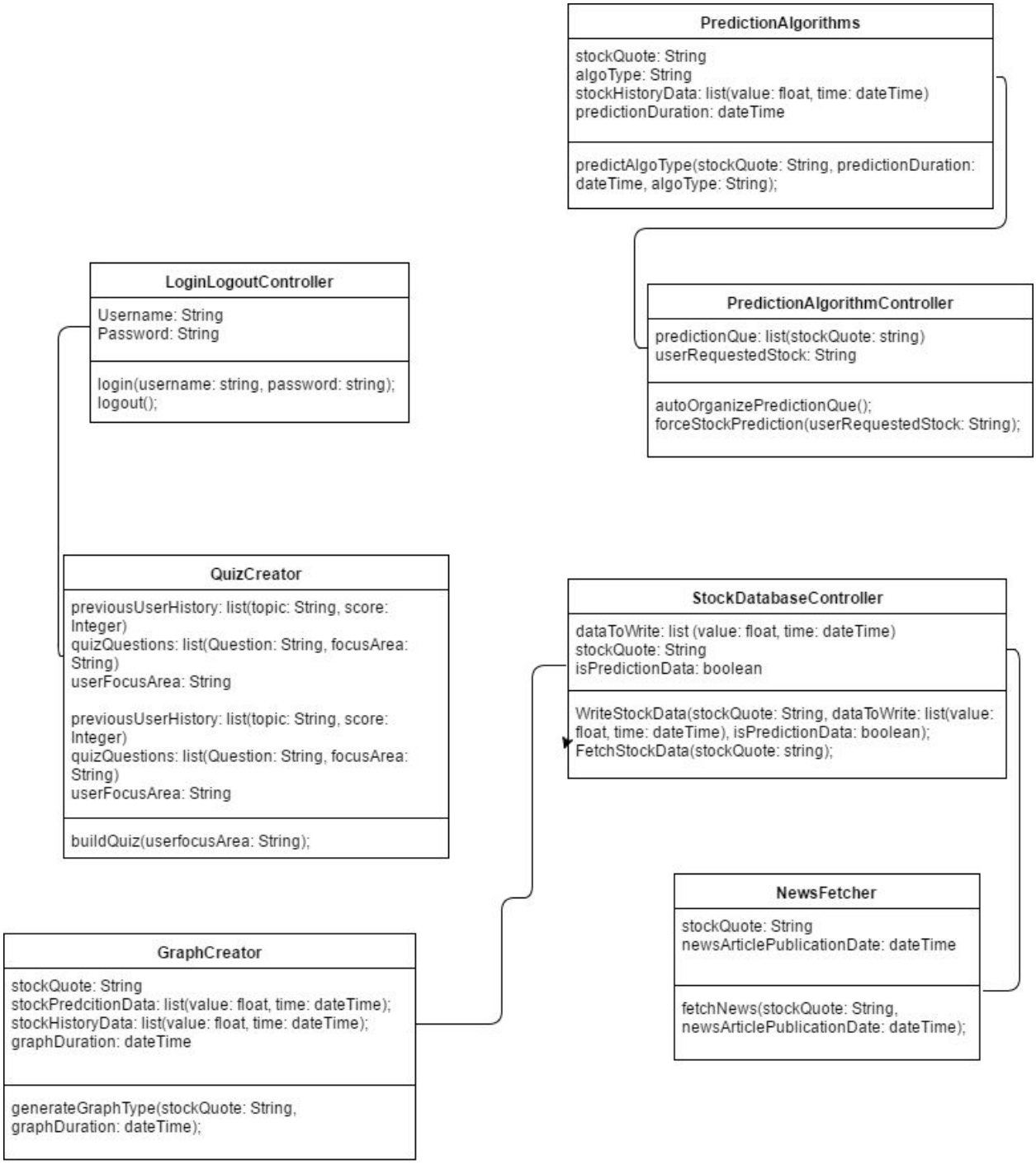

Figure 8.1

Class Diagram Explanation

This class diagram shows the interconnections between our classes. As you can see, our classes are organized into groups of two or three interdependent classes, including:

(LoginLogoutController - QuizCreator), (PredictionAlgorithms - PredictionAlgorithmController), and (GraphCreator - StockDatabaseController - NewsFetcher). These three groups are distinct in functionality, and do not need an overwhelming amount of communicate between themselves except to pass messages/commands from one group to another. Since we do not store stock predictions in stock database, the predictions group does not need to send or receive data from the stock database. Also, the stock database / graphing / news fetching do not require a user to be logged in, so the login-quiz, and stock database groups can be separated from one another. This is a result of our group's initial decision to implement a separation of concerns to simplify and reduce complexity.

Traceability Matrix

Classes/Use Case	1	2	3	4	5	6	7	8	9
LoginLogoutController	X	X	X			X		X	
GraphCreator			X		X				X
PredictionAlgorithms			X		X				X
PredictionAlgorithmController			X		X				X
QuizCreator		X						X	
NewsFetcher				X					X
StockDatabaseController			X			X			

Figure 8.2

Description/Explanation:

Notes:

- Whenever PredictionAlgorithms executes, we may assume that PredictionAlgorithmController and GraphCreator execute along with it.
- We may also assume that anytime the StockDatabase Controller or QuizCreator is used/called, logging into the site is necessary.
- There is no UserDatabase Controller as that functionality is built directly into the frontend Javascript code which dynamically stores and loads data from a separate user database whenever a quiz is being generated, or whenever a user logs in to the website.

UC-1: “A user may want to login/logout from the site. Login to resume progress tracking and portfolio tracking; logout to end their session of progress tracking.” The LoginLogoutController is critical for a user who may want to login/logout from the site.

UC-2: “A user may want to use our learning service to be taught about the stock market in a traditional manner.” A user who wants to utilize our learning service may want to take quizzes and reading our website’s tutorials and visual novels. The quizzes require logging in, thus the LoginLogoutController, and also require the QuizCreator to work properly.

UC-3: “A user may want to have a specific list of companies kept track of so that each time they enter the site it loads that list for them.” This requires a user to log in, stock prediction to create prediction lines on our graphs, which themselves require graphing, and a stock database to record which stocks the user is following. This does not include news fetching when multiple stocks are brought up at once.

UC-4 : “A user may want all of the relevant info about how the prediction model works as well as news about the company so as to make a more informed decision.” Knowing how the prediction model works is not directly associated with the PredictionAlgorithms running, as one is just a description, while the other is a piece of code. Furthermore, knowing relevant info about a company requires the use of our NewsFetcher.

UC-5: “A user may want their data presented with different options, such as not graphing predictions, for viewing aesthetics or to reduce information overload.” Options to select and/or toggle the data presented in our graphs is built into the Plotly functionality and does not require a custom class on our part, except to generate the graph and prediction data.

UC-6: “A user may want to register an account with our website so their individual progress and stock portfolio is saved across all sessions.” This requires logging in and also makes use of our StockDatabase Controller to save a user’s stock portfolio.

UC-7: “A user may want to learn information in the tutorial system through alternative means that provides auditory and visual feedback.” This use case requires our Visual Novel information format. Our Visual Novel is not a class of its own, but rather built into our website’s javascript frontend. Because of this, no classes are checked.

UC-8: “A user may want to test their knowledge against what they have learned and be given feedback on their performance, either through the automated system or through retaking sections.” This implies that the user will be taking quizzes, which requires a user to be logged in and the QuizCreator to be utilized.

UC-9: “A user may want to easily search for a stock.” Searching for a stock is built into the Javascript frontend. When a stock is found, stock prediction and graphing take over to generate a plot, along with relevant news articles being displayed

Note: Design patterns and interaction diagrams were done in the in the interaction diagram section

Object Constraint Language Contracts (including invariants, preconditions, postconditions) for classes and their operations:

Notes: Only describing functions, not private attributes.

LoginLogoutController

Invariant: This function integrates with the frontend Javascript code to allow other frontend components to see whether the user is logged in or not.

+login()

Preconditions: A user is not already logged in.

Postconditions: A user is logged out.

+logout()

Preconditions: A user is already logged in.

Postconditions: A user is logged out.

GraphCreator

+generateGraphType()

Invariant: This is an API function that calls other functions to fetch data and compute stock predictions.

Preconditions: Prediction and stock fetching have returned the required data.

Postconditions: A plotly graph is created.

PredictionAlgorithms

+predictAlgoType()

Invariant: There are two algorithm types - LSA and RSI. The type has implicitly been specified by the calling class, PredictionAlgorithmController.

Preconditions: Stock fetching has successfully returned a sufficient number of data points to apply the algorithms on.

Postconditions: The requested type of prediction algorithm is run on the input data set and returns a computed result for a line of best fit, future stock price prediction, or an RSI data-line for graphing.

PredictionAlgorithmController

Note: +autoOrganizePredictionQue() is unimplemented in our actual code and is unnecessary

+forceStockPrediction()

Invariant: This is built into the API and is not actually a separate class, but rather just a way to communicate with our stock prediction algorithms.

Preconditions: This assumes that only one stock prediction request is made at any point in time, as our code does not yet implement queues and could only process one request at a time.

Postconditions: The PredictionAlgorithms class is called to perform the heavy lifting of data computation and the result is returned to the graphing function for plotting the data.

QuizCreator
+buildQuiz()

Invariant: This class not only builds quizzes, but also accesses user information about previous quiz performance / lessons read.

Preconditions: A user is logged in. A user is able to take the quiz requested.

Postconditions: A quiz is generated and returned to the user for knowledge examination.

NewsFetcher
+fetchNews()

Invariant: This class does not require access to user information, but is called whenever a stock is searched through the website's Stock Search feature.

Preconditions: The requested headline provider is accessible and returns valid data.

Postconditions: A list of news headlines is returned to be placed into the webpage.

StockDatabase Controller

Invariant: This database controller is able to communicate with predictions in the case that data needs to be saved into the database for later access. This also integrates with stock fetching to be able to cache the fetched price data of the top 100 stocks into the database.
+fetchStockData()

Preconditions: The data requested from the database is specified. The database is accessible.

Postconditions: The requested data is returned to the calling class/function.
+writeStockData()

Preconditions: The type of data to be entered into the database is specified, and the data to be stored is not empty. The database is accessible.

Postconditions: Data is written into the database.

9. 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.

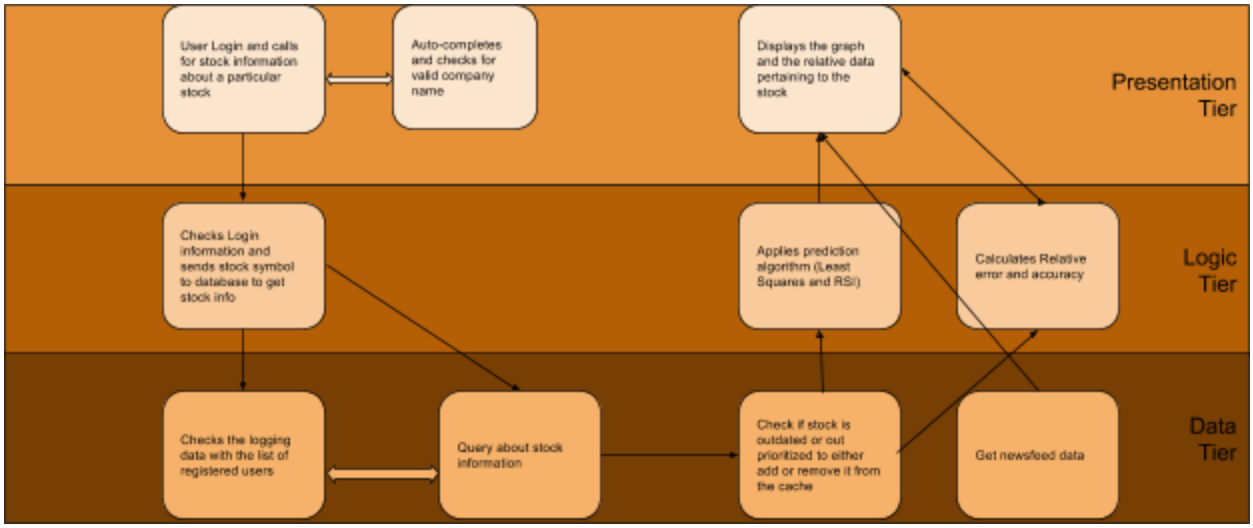

Figure 9.1 - Architectural Diagram

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. This tier will handle button presses and the auto-completion of company names as to communicate to the logic tier. The presentation tier also handles the user's reading text and visual novels. With visual novels this tier will communicate with the logic tier to call the api to generate the visual novels graphics and animations. 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 yahoo finance 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 and stock ticker information to relay to the presentation tier. Additionally, this tier communicates with plotly and RSI functions to combine both graphs, and calculating error and accuracy. This tier also handles the manipulation of the visual novels for each lesson by communicating with the visual novel generator's api.

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 after receiving the stock symbol. 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 handle the inclusion and expulsion of information from the cache for least square approximation prediction based on a priority system. This tier will primarily call the google and yahoo finance api's to organize and gather data for future use as well as the cache for cases that regard prediction.

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.

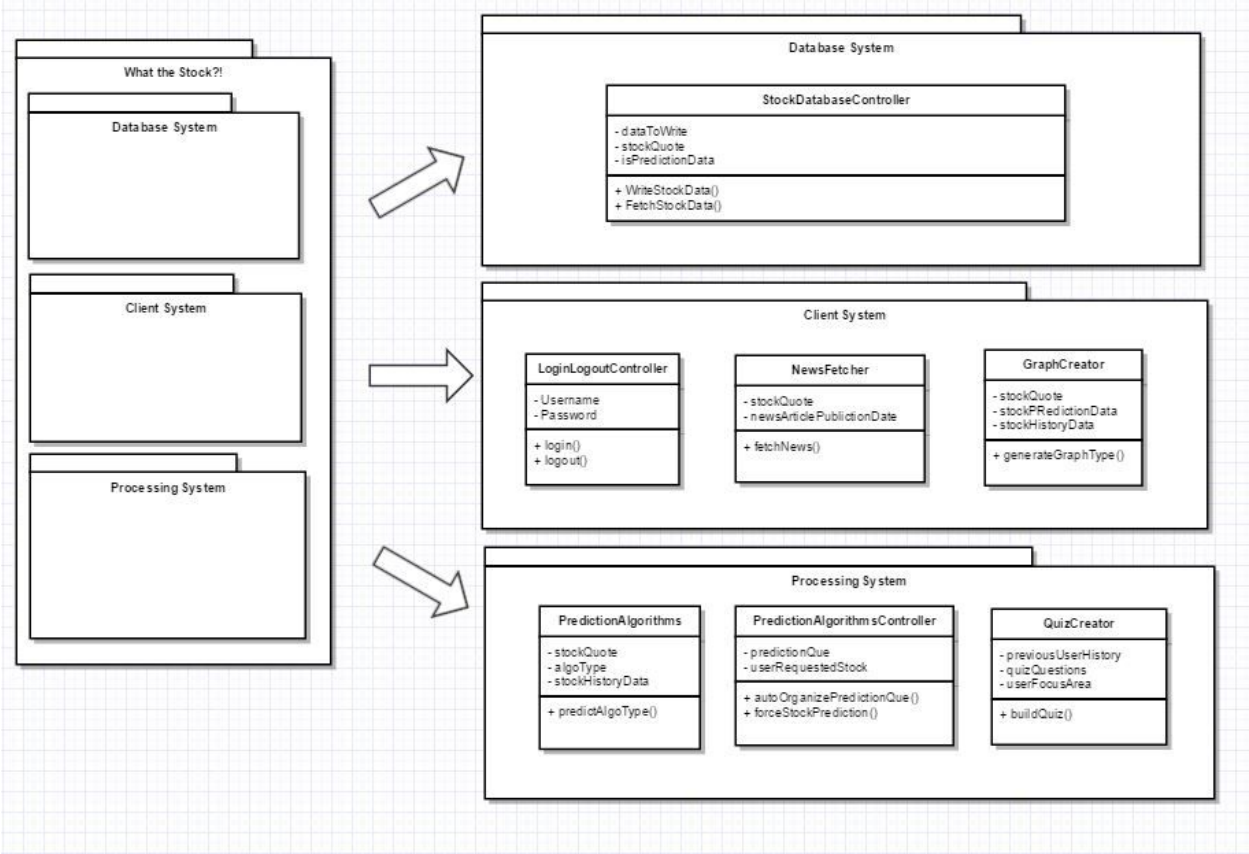

Figure 9.2 - UML Package Diagram

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.

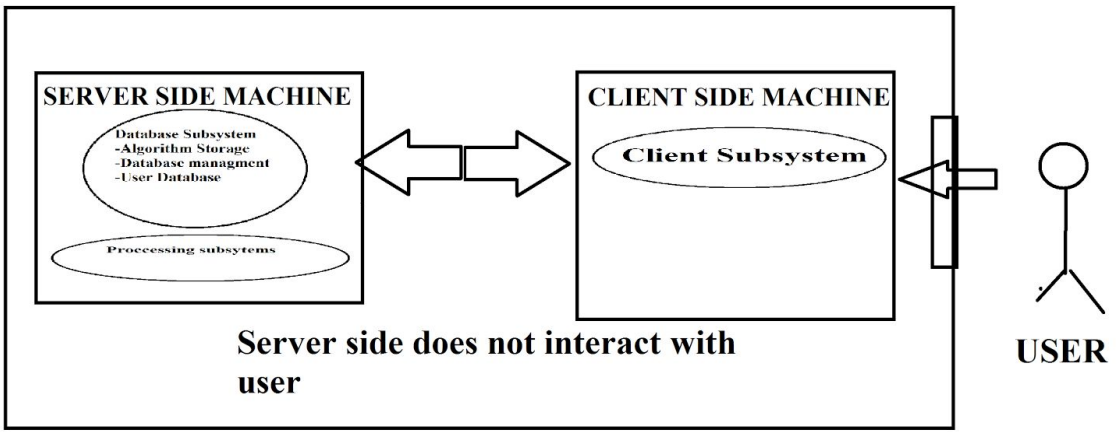

Figure 9.3 - Subsystem to Hardware Mapping, Visual Representation

Persistent Data Storage

Our system requires data to be saved past one execution.Persistent objects in the system include: user account information, their password, the session in terms of how long they're logged in, and the different types of lesson for their portfolio. We fetch from the SQL database entries as needed

```
cursor.execute("""CREATE TABLE IF NOT EXISTS
        user(
        username TEXT,
        password TEXT,
        session INTEGER,
        lessonStates TEXT,
        quizStates TEXT,
        portfolio TEXT,
        placementTaken INTEGER,
        modeSwitch INTEGER);""")
```

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.

And for the values of the prediction that we calculate, we store them in the cache and retrieve them as needed. We also add the date when they're added into the cache and replace the prediction values if it's been 3 days past the day they were added

```
cursor.execute("""CREATE TABLE IF NOT EXISTS
        Prediction(
        Company_name TEXT,
        Company_symbol TEXT,
        Value1 Float, (Each float represents a coefficient for the 10th Power function generated by
prediction, a0)
        Value2 Float,(a1)
        Value3 Float,(a2)
        Value4 Float,(a3)
        Value5 Float,(a4)
        Value6 Float,(a5)
        Value7 Float,(a6)
        Value8 Float,(a7)
        Value9 Float,(a8)
        Value10 Float,(a9)
        Value11 Float,(a10)
```

Network Protocol

Javascript calls XMLHTTPRequests using HTTP GET requests. We use Flask to make python accessible using HTTP GET requests. Javascript libraries such as Jquery are downloaded using HTTP GET requests at page load. Jquery handles animation, and other important features (such as auto complete, formatted zoom in for mobile users, etc)

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 stocks are updated by the introduction of new data to the system i.e after 3 days or by the most relevant data being changed.

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 300MHz 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.

10. 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.

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

Figure 10.1

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:

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

Figure 10.2

General Vector form for data input:

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

Figure 10.3

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

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

Figure 10.4

The objective function of S is given below

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{m} \left| y_i - \sum_{j=1}^{n} X_{ij}\beta_j \right|^2 = \left\| \mathbf{y} - \mathbf{X}\boldsymbol{\beta} \right\|^2.$$

Figure 10.5

Transformation Matrix is given below

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}.$$

Figure 10.6

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 10.7 - LSA Sample Analysis

FIgure 10.7 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.

RSI (Relative Strength Index)

About

Relative Strength Index generates a day-to-day indication of the momentum of bought and sold stocks. A list of past stock price data is passed into the function, on which the RSI algorithm analyzes the most recent 14 days. This time period can be changed for a more long term or short term analysis of stock strength. RSI computes the ratio of the closing prices of one day to the next day and treats this value as the momentum of a stock’s increase or decrease in price.

Method

First we take in an array of price data, and split this data into up and down days, according to the following formula:

Up Array - For Up days, when there is a increase in stock price, record the positive change in price:

$$U = \text{close}_{\text{now}} - \text{close}_{\text{previous}}$$
$$D = 0$$

Figure 10.8

Down Array - For Down Days, and decreases in stock price, record the absolute value of the negative change in price:

$$U = 0$$
$$D = \text{close}_{\text{previous}} - \text{close}_{\text{now}}$$

Figure 10.9

Now we perform the following steps in a for loop for (period)-times, or 14-times, for each day in the period:

Next we smooth these arrays using SMMA (smoothed or modified moving average). To be specific, we use an exponential smoothing function such as:

$$s_t = \alpha \cdot x_t + (1 - \alpha) \cdot s_{t-1}.$$

Figure 10.10

where α is the *smoothing factor*, and α = 1/period = 1/14.

The SMMA code in the github smooths on a per-array-element basis each time the for loop executes, each iteration pertaining to a particular day:

```
avg_up = (avg_up * (period-1) + gains[i + (period - 1)]) / period
avg_down = (avg_down * (period-1) + losses[i + (period - 1)]) / period
```

From this smoothed data, we can calculate the momentum of the strength of the stock, which we term Relative Strength (RS):

$$RS = \frac{\text{SMMA}(U, n)}{\text{SMMA}(D, n)}$$

Figure 10.11

Where SMMA(U,n) means the smoothed data from the up days that we calculated in the previous step. Likewise, SMMA(D,n) means the down days.

Finally, we can compute the RSI for a particular day based on the following formula:

$$RSI = 100 - \frac{100}{1 + RS}$$

Figure 10.12

We store this value in an RSI data array.

When the for loop terminates, we return the RSI array with the float values stored in it to be plotted.

This creates a data set such that we can assess the strength of a stock as its price fluctuates:

Figure 10.13 - Example of an RSI Plot

Data Structures

LSA (Least Square Approximation)

This algorithm takes into account the past 30 days the x value being the number of workdays 45 days ago and the y-axis is the high values of each day. The LSA uses two matrices: One is a Combination matrix with columns of $1, x, x^2, x^3, ..., x^x$, the other is a matrix of the y values. Well, call the Combination Matrix C. Matrix C is then manipulated $(C^T * C)^{-1} \cdot C^T * Y$. The resulting matrix is one column matrix that has the coefficients for the curve of best fit. Vince decided through testing to use a precision of x^10 power. A function with the coefficients from the output matrix is plotted alongside the current graph. The curve is then extended one day ahead up to three days before the algorithm recalculates the coefficients. This allows us to see the predicted closing price of the stock for the next day.

Along with prediction normalization needs to be taken into account to adjust our prediction line to fit our plotted stock information. Vince's Normalization algorithm calculates the difference between points then divides it by a minimum and maximum (-25% of the current value, +25% of the current value) and then adds it to the current day's value. This normalizes it within a 50% total error. Additionally, this is how we predict our next day price by calculating the difference between the current value and the predicted value as given by the prediction function and then normalizing it.

Numpy Array

Numpy arrays are easy to append. They are C++'s vectors but in python. They are accessed the same way as regular arrays and are easy to delete from and resize. The use of Numpy arrays allows me to create matrices from arrays (More information on that is below). For the bumpy array most of the time they held floats, but in addition to that, they held the data type DateTime. This made graphing with plotly incredibly easy and intuitive.

Numpy Matrix

As Stated earlier, the matrix for numpy comes from the resizing of a numpy array. This allowed me to merge multiple arrays into a matrix. Numpy's reshape allowed the creation of append-able 2-dimensional arrays, something the base array class for python couldn't easily do. Numpy Matrix provided methods for matrix multiplication, inversion, and transposition. These tasks are all hard to code in for nxn dimensional arrays. Numpy Matrix also allowed for us to conveniently store and transport information regarding the coefficients throughout the cache and other methods

Datetime

Datetime allows the transportation of the year, month, day and time in one data type. This conveniently allows us to display dates on the plotly graph as well as manipulate dates. Manipulation was made convenient by date time with the time delta function as well as the weekday function. Time delta allowed for us to advance days without having to worry about adding or subtracting months and years. The Weekday attribute of DateTime allowed us to easily determine work days, which is a crucial point in a stocks prediction. Another benefit of date time was that it allowed us to easily search a stock based on time since yahoo finance takes DateTime as an input.

Pandas_Datareader

This data structure is basically a multi-dimensional list that is meant to read directly from databases. It basically acts as a carbon copy of the database allowing us to access table-like information by using the “.” function and using brackets to access the element. This allowed us to send data through the functions as needed but also allowed us to split up the data without the need to implement our own yahoo finance database reading methods. Additionally, using the pandas_datareader plotting data is exceptionally easy.

General 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:

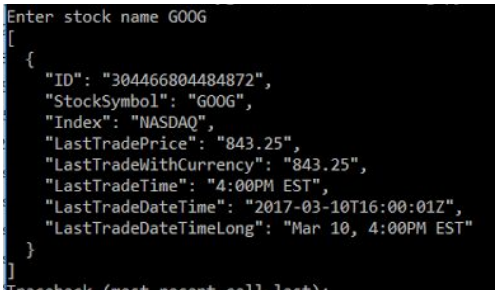

Figure 10.14

Example Output for Yahoo API:

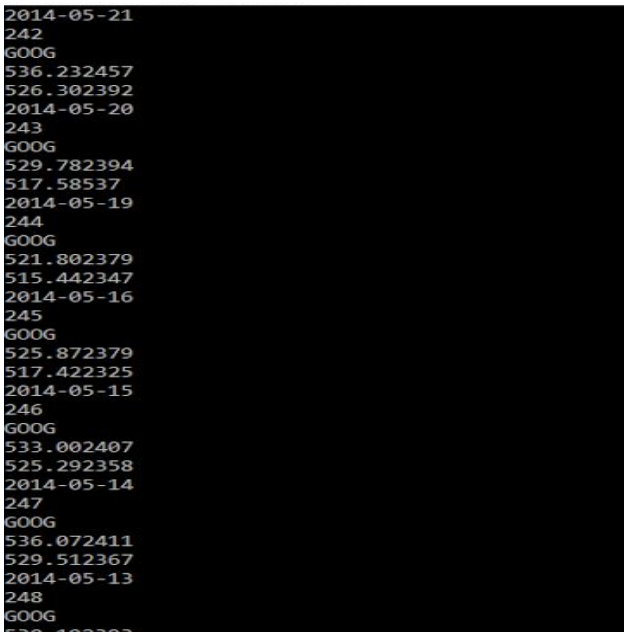

Figure 10.15 - Data Taken From 2014-2015

RSI (Relative Strength Index)

RSI creates several Numpy arrays and returns a Numpy array. It takes in a Pandas data object as a parameter and transfers the data stored in it into a Numpy array. From there, I declare several numpy arrays: price_change, gains, losses, and rsi.

Numpy Array

RSI uses Numpy just like LSA uses it for holding data in a convenient storage object. Numpy allows me to resize and cut off elements from the array without declaring a new array and copying all the old data to the new array. They can also hold almost any data type and are

simple to use for manipulating stock price data. Numpy also contains methods for performing computations on Numpy arrays.

Pandas

This is used as an import as the only RSI function parameter is a type of Pandas object that stores stock price data, and time/date. This object comes from the stock fetching code which downloads a multi-dimensional list containing stock price data, including the low, high, and closing prices for a stock. The closing prices list is separated from the Pandas object and passed into the RSI function. This list can essentially be treated like an array, which could create possible confusion as to the difference between Numpy and Pandas. Whenever an array or list needs to be used, I choose to declare it as a Numpy array, as they append easily and are in general more convenient for manipulating data, while Pandas is used only to transfer stock data into the function.

DateTime

DateTime is also used to work with the Pandas time entry for each data point. This allows us to manipulate the time and specify a two week period for which RSI Measures. Depending on the 2 week span we get 9-10 workdays (pending holidays) to which RSI actually measures. DateTime allows us to manipulate time to find workday or to access days in the past.

11. User Interface Design and Implementation

For the most part, every on-screen design that we made diagrams for earlier was able to be implemented with very little changes. While the style and format of the implemented designs varied from the original sketches, they did not alter the information presented, and thus had very little effect on how much effort a user has to exert while using the app. However, some aspects of the app require some explanation in order to understand the user's effort, and other aspects of the app have changed since its initial mock-ups.

Design 1: Site Navigation

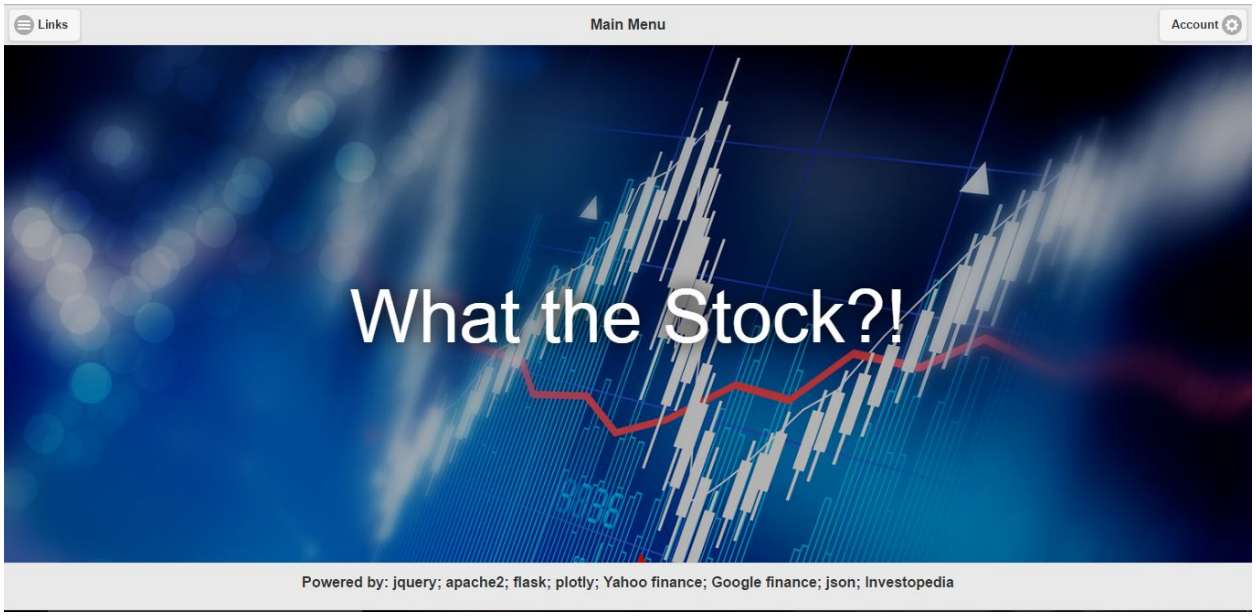

Figure 11.1 - Splash Screen / Main Menu

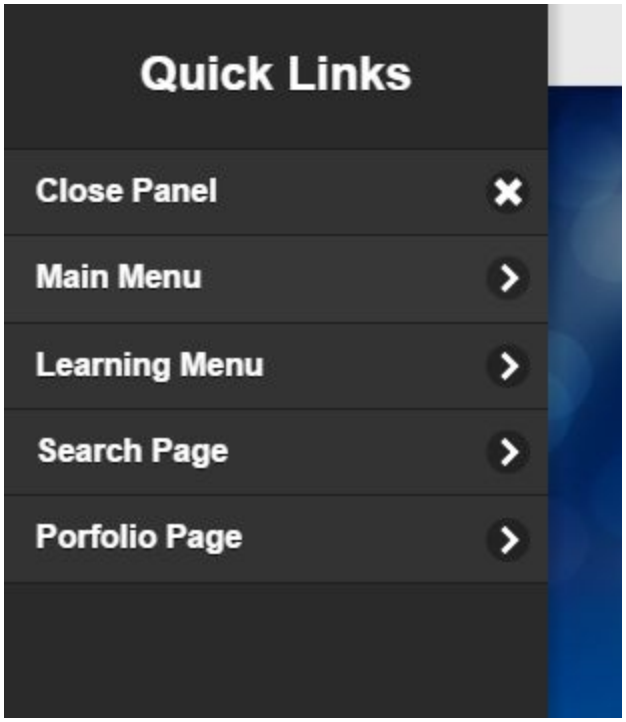

Figure 11.2 - The Quick Links Menu

Pictured above as figure 11.1 is our app's main menu page. The intent of this page is to act as a 'hub' for the user, from which the user can access any other part of our website with a minimal amount of clicks. If the user clicks on the "Links" button on the top-left of the site, the Quick Links menu of figure 11.2 pops up on the left side of the screen, providing buttons that take the user to any part of the website with one more click. Thus, with two clicks, a user can be redirected to any part of the website that she chooses.

Design 2: The Portfolio Page

Our Portfolio Page was designed with the idea that a user will want to keep track of certain stocks in between sessions. However, the portfolio page has undergone significant changes from its initial conception due to time constraints. The original portfolio page mock-up can be seen in figure 3.15. This iteration displays one graph per tracked stock instead of a composite graph like originally thought.

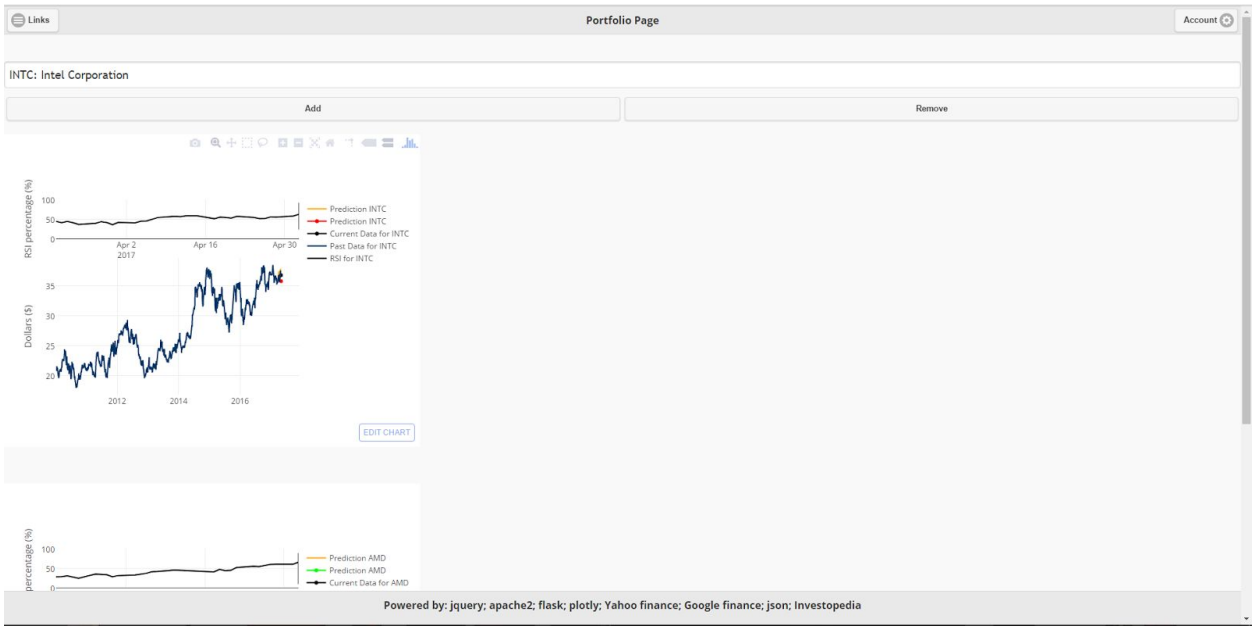

Figure 11.3 - Portfolio Page, Populated With Stocks

Shown above in figure 11.3 is our implemented portfolio page. This information exists as a list of Interactive Stock Graphs, rather than one Interactive Stock Graph and a list of stocks. Thus, the User Effort has to increase; instead of the user clicking once on a stock name to change which graph is being displayed, the user has to scroll up and down along the page to find the graph that she's looking for.

Design 3: The Interactive Stock Graph

The Interactive Stock Graph is an element that graphs the historical data of a stock's price, along with the data obtained about that stock from our prediction algorithm. For the most part, the layout of this element is the same as the layout of the mockup described in figure 3.3; the graph is displayed, along with some toggleable options on what information is displayed. The only real difference is the addition of the RSI graph on top of the stock graph as seen in figure 11.5. Both graphs are intractable in the same way.

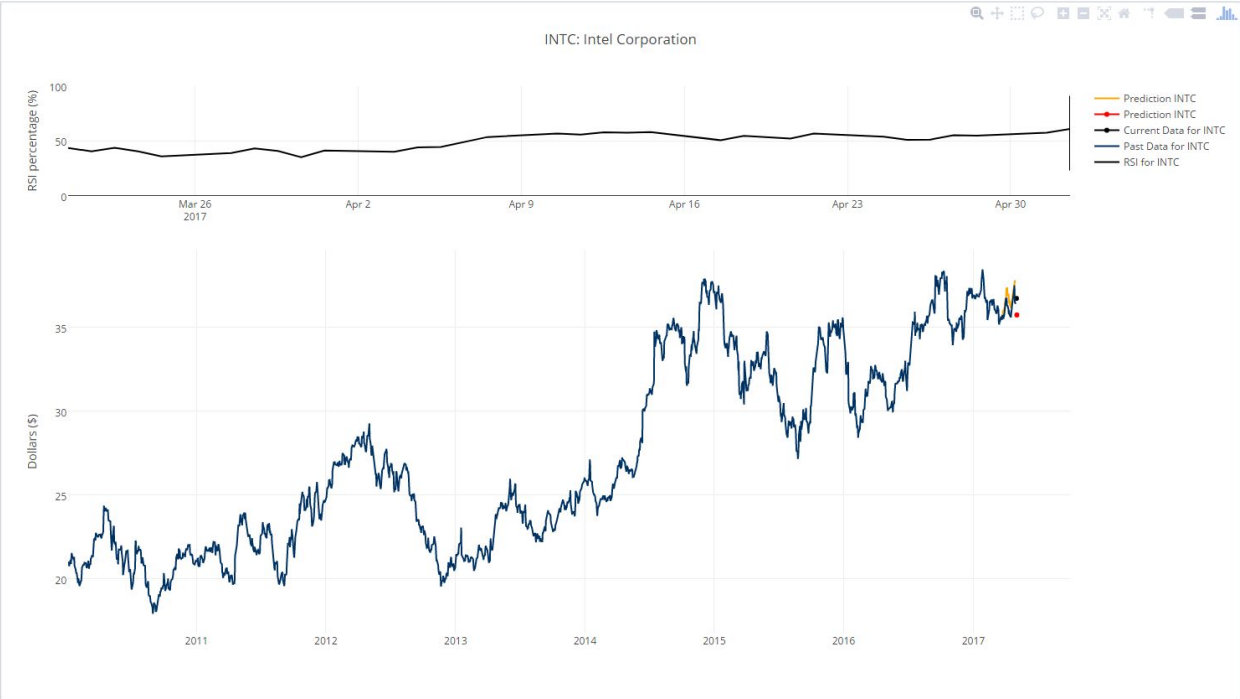

Figure 11.4 - Interactive Stock Graph

Displayed in figure 11.4 is one of our interactive stock graphs. While all the information that we wanted is presented, we have underestimated the user effort required to view the info. The app pulls historical data for the app from the past few years, and then plots prediction data from the past 20-30 days. Thus, if the user wants to view this prediction data, he will first have to first zoom into the most recent parts of the graph by clicking and dragging to create a region to zoom up on.

Design 4: The Search Page

The search page allows the user look up stocks and get information like the graph, prediction, and new. But since our mock up pictures, as seen in figure 3.14, the search page has undergone revision with the addition of buttons and text outputs with a modification to its layout as a result.

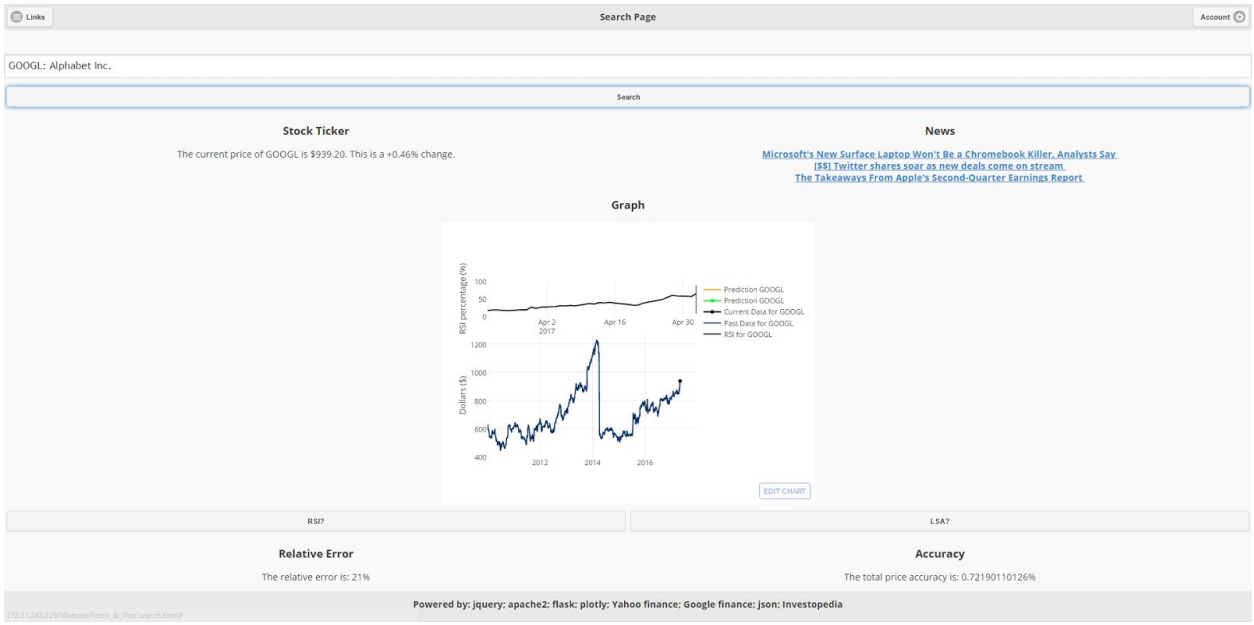

Figure 11.5 - Search Page

Displayed in figure 11.5 is the search page as implemented. The aforementioned revisions include: buttons to provide RSI and LSA descriptions for the user so they can understand what the functions mean in context of the stock as graphed; relative error of the prediction algorithm, for confidence rating; accuracy of the prediction algorithm, for another type of confidence rating; and a layout modification to accommodate the additional information and buttons.

12. Design of Tests

Test Descriptions and 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.

Graphing

Plotting data appropriately and effectively is an underlying attribute of the prediction and stock representation. Graphing gives the user a visual representation of prediction data and historical data.

Testing

An effective way to test graphing is to plot data the data ranges for several time periods of a stock and compare them to make sure that the graphing is consistent. Another way to test graphing is to add multiple graphs with different forms of data in different formats. This tests lines that cross over one another and tests the functionality of changing which graphs appear with buttons on the side. Additionally, this method allows us to test color configurations as to appeal to a demographic that is color blind or colour impaired.

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. An additional method is to plot a prediction for one day leave that prediction and check the predicted value vs the next day's data. This test is long because we have to wait for the stock to reopen; however, this also allows us to test accuracy and error

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.

13. History of Work

Key Accomplishments

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]) operated independently of the others.

The tasks for Team 1 were executed mostly independent of each other. Vince worked on the Stock Fetching for 2 weeks, then spent 2 weeks working on Interactive Graphs, then spent 1 week working on Confidence Rating. Raj worked on the Stock Cache for 2 weeks, then spent 1 week working on the autocomplete feature. Jon worked on the Stock Cache for 2 weeks, then spent 2 weeks working on Confidence Rating, and then 1 week working on RSI Prediction.

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.

For Team 2, Greg worked on the User Database for 4 weeks, and then Account Registration and User Login for 2 weeks, and then User Favorites for 1 week. Jake worked on Stock Ticker and Contextual news for 3 weeks.

Team 3's tasks occur in two parallel timelines (Tutorial) (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 Quizzes are worked on simultaneously for 18 days. 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.

For Team 3, Skyler worked on the Learning System control template code for 3 weeks, then worked on the Quizzes and Lessons for 2 weeks, then worked on Spot-Checks for 2 weeks. Jack worked on the Quizzes for 2 weeks, then the Placement Quiz for 1 week, then Lessons for 2 weeks, then the Visual Novel Lessons for 3 weeks.

For the website, Greg and Skyler worked on system integration for 2 weeks and then spent 2 weeks crafting the final website.

Future Work

In the future, we would like to add several more prediction algorithms, so that the user has a more varied set of inputs with which they can make an informed decision. We would also like to refactor the python code so that it all runs out of one flask wrapped application instead of two, to minimize network usage and decrease dependency confusion.

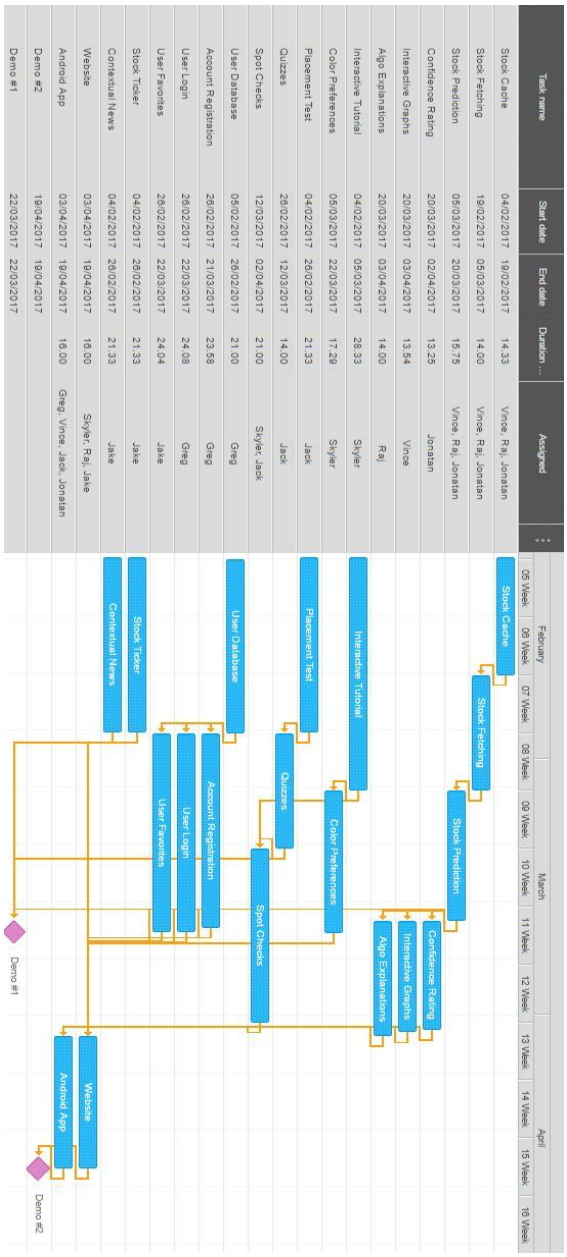

Figure 13.1 - Code Gantt Chart

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

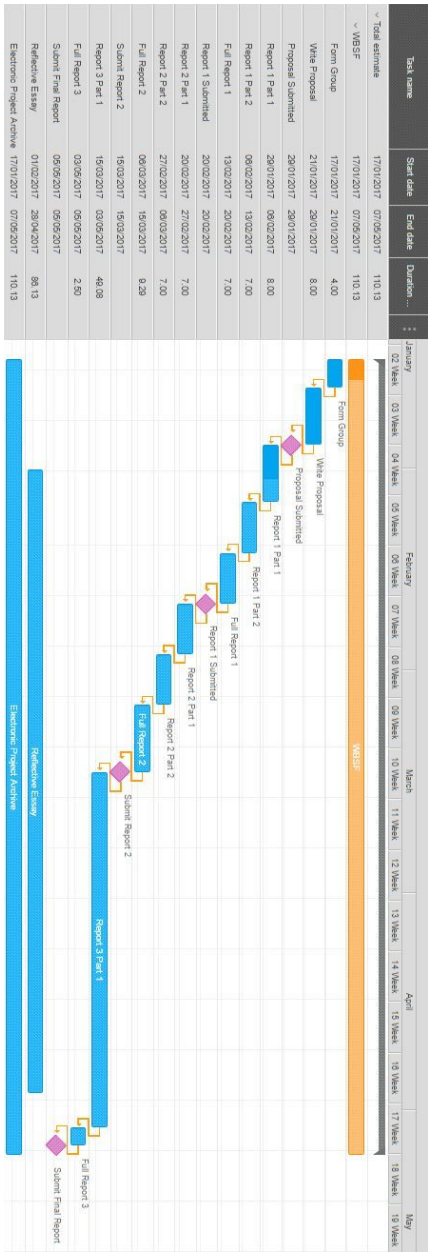

Figure 13.2 - Documentation Gantt Chart

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 and Skyler Malinowski.

	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 Quiz Interactive Tutorial Proficiency Quizzes Spot-Checks Visual Novel

Figure 13.3 - Team Breakdown

Progress Matrix

Member	**Completed**
Leonberg, Gregory	User Database Account Registration User Login User Favorites Autocomplete Website
Lewandowski, Jake	Stock Ticker Contextual News
Malinowski, Skyler	Learning System Lessons 1-6 Spot Checks

	Website
Aquino, Jack	Learning System Placement Quiz Lesson 7 Quizzes 1-7
Taylor, Vincent	Stock Fetching LSA Prediction Interactive Graphs Confidence Rating
Yanovsky, Jonatan	RSI Prediction Confidence Rating
Balaji, Raj	Stock Cache Autocomplete

Figure 13.4 - Progress Matrix

14. References

http://www.timothysykes.com/2013/06/trading-terms-you-need-to-know/
(Used for Customer Statement of Requirements, as well as Glossary of Terms)

http://www.wikinvest.com/wiki/What_is_a_stock%3F
(Used for Customer Statement of Requirements, as well as Glossary of Terms)

https://businessanalysttraininghyderabad.wordpress.com/2014/08/05/what-is-furps/
(Used for Enumerated Non-Functional Requirements)

http://math.mit.edu/~gs/linearalgebra/ila0403.pdf
(Used for least square approximation and method description)

https://www.desmos.com/
(Used to test the methodology and make figure in the mathematical model section)

http://www.investopedia.com/terms/l/line-of-best-fit.asp
(Used for figures in the mathematical model section and for description information)

http://money.cnn.com/2014/09/18/investing/stock-market-investors-get-rich/
(Used for Customer Statement of Requirements, as well as Glossary of Terms)

https://www.codecademy.com/
(Used for inspiration for On-Screen Appearance Requirements and User Interface Specification)

https://finance.yahoo.com/
(Used for inspiration for On-Screen Appearance Requirements and User Interface Specification)

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]

https://en.wikipedia.org/wiki/Dynamic_HTML
(Used for Glossary Term)

https://upload.wikimedia.org/wikipedia/commons/6/67/RSIwiki.gif
[*Media Image of an RSI-plot example]*

https://api.jquery.com/
(Used for Glossary Term)