TEAM 3

WEB BASED STOCK FORECASTERS

https://sites.google.com/site/group3stockforecasting/

REPORT 3

Peter Zhang

Vincent Chen

Robert Adrion

Syedur Rahman

Robin Karmakar

Mohammed Latif

Manoj Velagaleti

Responsibility Matrix

RESPONSIBILITIES	TEAM MEMBERS: Robert Adrion	Vincent Chen	Robin Karmakar	Mohammed Latif	Syedur Rahman	Manoj Velagaleti	Peter Zhang
Summary of Changes (5)			100%				
Customer Statement of Requirements (6)						100%	
Glossary of Terms (4)					100%		
System Requirements (6)							100%
Functional Requirements Specification (30)	14.3%	14.3%	14.3%	14.3%	14.3%	14.3%	14.3%
Effort Estimation (4)	60%				20%		20%
Domain Analysis (25)	25%	15%	15%	15%	15%	15%	15%
Interaction Diagrams (40)	25.6%	25.8%	14.3%	14.3%	10%	10%	
Class Diagram and Interface Spec. (20)			25%	25%	25%	25%	25%
System Architecture and Design (15)		33%		34%			33%
Algorithms and Data Structures (4)	5%		5%	5%	20%	5%	60%
User Interface Design and Implement. (11)					50%		50%
Design of Tests (12)	50%		50%				
History of Work, Current Status, and Future Work (5)						100%	
Project Management (13)		50%		50%			
TOTAL POINT ALLOCATION	29.38	29.81	29.96	30.56	28.14	28.24	28.94

Allocation of Points

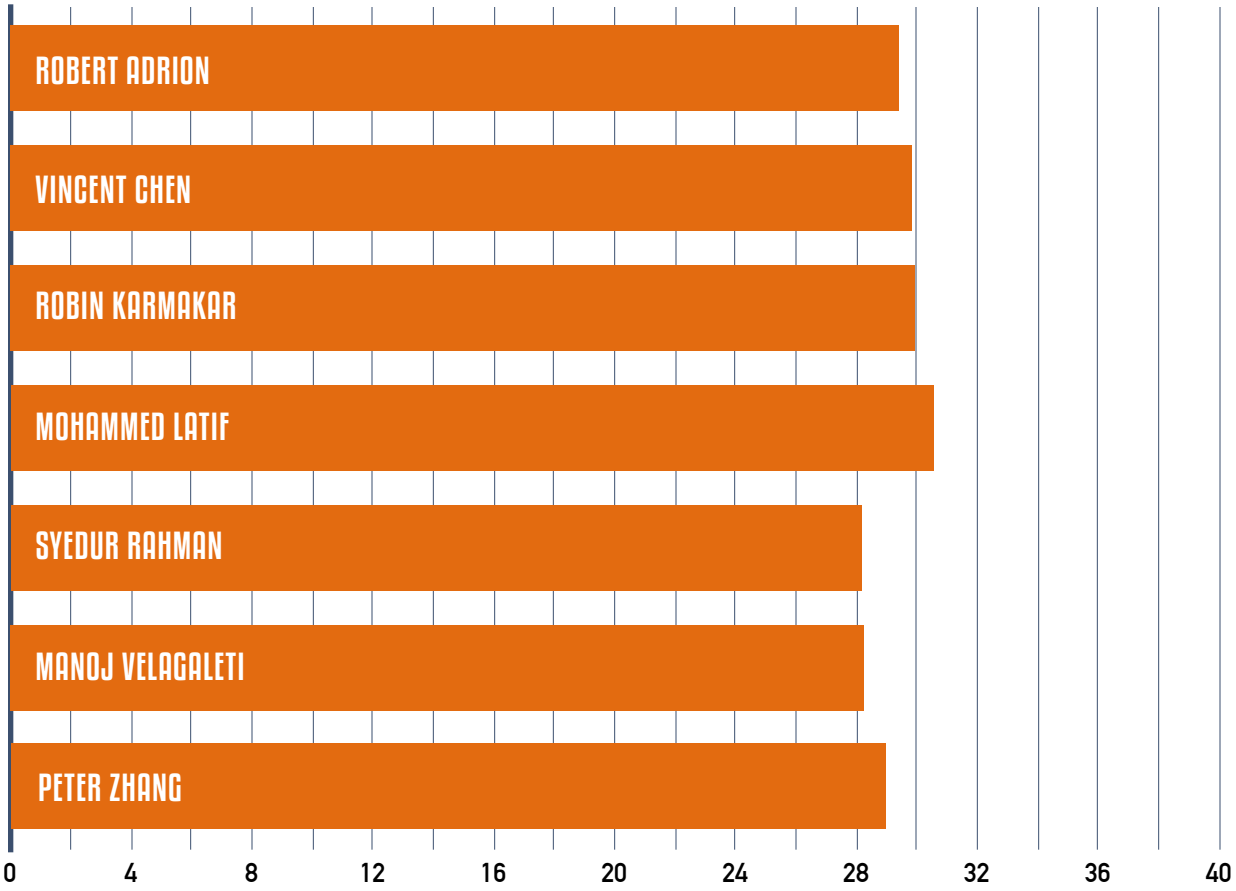


Table of Contents

Summary of Changes **6**
1. Customer Statement of Requirements **7**
1.1. Problem Statement 7
2. Glossary of Terms **10**
3. System Requirements **11**
3.1. Enumerated Functional Requirements 11
3.2. Enumerated Non-Functional Requirements 15
3.3. On Screen Appearance Requirements 16
3.4. Acceptance Tests 17
4. Functional Requirements Specification **19**
4.1. Stakeholders 19
4.2. Actors and Goals 19
4.3. Use Cases 20
4.3.1. Casual Description *20*
4.3.2. Use Case Diagram *22*
4.3.3. Traceability Matrix *23*
4.3.4. Fully Dressed Description *23*
4.3.5. Acceptance Tests for Use Cases *30*
4.3.6. System Sequence Diagrams *32*
5. Effort Estimation **37**
5.1. Unadjusted Actor Weight 37
5.2. Unadjusted Use Case Weight (UUCW) 37
5.3. Technical Complexity Factor (TCF) — Nonfunctional Requirements 39
5.4. Environment Complexity Factor (ECF) 40
6. Domain Analysis **41**
6.1. Concept Definitions 41
6.2. Traceability Matrix 44
6.3. Associations 45
6.4.Attributes 46
6.5. System Operation Contracts: 47
6.6. Mathematical Models 49
7. Interaction Diagrams **50**
8. Class Diagram and Interface Specification **59**
8.1. Class Diagram 59
8.2. Data Types and Operation Signatures 60
8.3.Traceability Matrix 65
9. System Architecture and System Design **66**

9.1. Architectural Styles 66
9.2. Identifying Subsystems 67
9.3. Mapping Subsystems to Hardware 68
9.4. Persistant Data Storage 68
9.5. Network Protocol 72
9.6. Global Control Flow 73
9.7. Hardware Requirements 73
10. Algorithms and Data Structures 74
10.1. Algorithms 74
Artificial Neural Network (ANN) 74
Autoregressive integrated moving average (ARIMA) 75
Simple Moving Average (SMA) 76
Exponential Moving Average (EMA) 77
Rate of Change (ROC) 77
Relative Strength Index (RSI) 77
Average True Range (ATR) 78
Average Directional Index (ADX) 78
Accumulative Swing Index (ASI) 79
10.2. Data Structures 80
11. User Interface Design and Implementation 81
12. Design of Tests 83
13. History of Work 86
13.1. Key Accomplishments 86
13.2. Future Work 87
14. References 88

Summary of Changes

1. Domain Model was overhauled completely to provide more suitable and logical concepts.
2. User Interface was overhauled completely to provide a more pleasant and fluid user experience.
3. Further mathematical algorithms were added to provide a more precise prediction for future stock prices.

4. Customer Statement of Requirements

4.1. Problem Statement

From its birth, the stock market has always been a place where businessmen could gain capital buy selling shares of their companies and where investors can purchase these shares in hope to make a profit when those companies prosper. Although the market still serves both these purposes, today it is judged less by what it does for businessmen seeking capital but rather for what it can do for investors seeking gain. In a time of high taxes and low wages, trading stocks seems like a viable and almost effortless way of becoming very rich very quickly. Although this scenario has proven true for many, there are just as many, if not more, who can proclaim the exact opposite. The problem here lies with the fact that the same volatility in the market that can be exploited for gain, can be the cause for loss as well. As both amateur and knowledgable investors, this is what worries as.

Motivation:

For day to day trading, these risks are a major factor in keeping away new, eager yet unaccustomed, traders from the market. They lack the appropriate tools needed for the decision making process when compared to existing professional traders. This scenario is witnessed way to often and can range from the young wanting to have a steady stream of capital without the need to invest in a part time job whilst also being a full time student or even the old who are well established into life attempting to save up for retirement. In both of these cases, these are individuals who would gladly consider trading stocks but many a times shift away from the idea due to the lack of time, recourses, and knowledge in the field.

To the college students amidst us, the idea of “gambling” in the stock market is fear inducing. Today, to turn this “gamble” into at most an informed estimate we would have to go through hours upon hours of research to first acquire the knowledge needed to even begin to understand how trading decisions are made. Understanding is only the first hurdle, however. Once one knows how to read charts, number crunch historical prices, or acquire and comprehend a companies balance sheets amongst many other details, the act of actually performing these tasks remains, something easier said than done. As students, the time and energy needed for such undertakings simply does not exist. This dilemma is one of which anyone foreign to the field of stocks can relate to.

The long-term investors among us, on the other hand, rely on a research heavy way to invest. There methods entail analyzing the financial performance of many companies and choosing the ones that appear to have the best growth potential. Since their prospects are long term, it may seem at first that the volatility of the market can be of no help here. Even these types of investors, however, are looking for as much profit as attainable and therefore do not overlook purchasing a stock at the lowest price possible. Thus, they too rely on the technical analysis techniques of day to day traders. Even with their acumen of the field, however, number crunching and attempting to eyeball trends off of charts for these split second decisions can not

only be time consuming and stressful but are also prone to human error.

Most investment advisors suggest maintaining a diverse investment portfolio in order to reduce the risk of investment loss. It is said we should not “put all of their eggs in one basket” and thus should increase the quantity of our investments over various industries. With ten times the stocks, however, arises the ten times the difficulties stated previously. Added on are the problem of organizing and keeping track of each and every investment. For the new and eager day trader among us, if the decision making of one investment seemed fearful how would he/she orchestrate his/her choices when suggested to acquire multiple shares? Without a proper way to declutter and systemize these decisions, these individuals may never move forward from owning one specific type of share at a time and thus be vulnerable to losses due to sudden and unforeseeable events even if their initial investment decision was deemed to be wise.

To summarize, the leading complications that exists with using the stock market as a viable stream of capital is the fact that many newcomers are not knowledgable with the financial concepts needed to make an educated trade decision. Even then, the knowledgeable as well, may find that their techniques are time consuming and stress inducing. When deciding to swallow a set of data and, based on multiple patterns and trends, spit out a conscious commitment that can be the difference between a great profit and a terrible loss, human error is bound to occur. When a myriad of companies are further introduced into the equation, these obstacles accumulate exponentially and the need of quick, clear , and organized information is paramount.

Vision:

What we, as both novel traders as well as long term investors, require is speed and simplicity. Envision being able to search for a particular stock and get a reasonable trading decision within the blink of an eye; a place which not only makes making these decisions effortless, but also provides the information necessary to learn the process behind the decision if we should ever choose to. Computer technology throughout the years has greatly introduced the idea of automation as well as being a learning tool for those that seek it. Everyone nowadays has access to the internet in some way shape or form whether it be a desktop computer, tablet, or even smartphone. Thus, to allow for effortless accessibility to the common user, such as you and I, a web based software service is required.

The first and foremost purpose of the system should be to predict future stock prices. There are many prediction methodologies that fall into two general categories that can and often will overlap: fundamental analysis and technical analysis. Fundamental analysts are concerned with the company that underlies the stock itself. They evaluate a company’s past performance as well as the credibility of its accounts. Technical analysts are not concerned with any of the company’s fundamentals. They generally seek to determine the future price of a stock based solely on the potential trends discovered from observing historical pricing of that stock. This set of past prices can range from dating back anywhere from a week to a couple years. Data used in fundamental analysis for a stock are generated much more slowly than the price and volume data used by technical analysts. Financial statements, for example, are filed quarterly and changes in earnings per share don't emerge on a daily basis. Some even argue

there is less reason to analyze a company's fundamentals because these are believe to be accounted for in the stock's prices. Therefore, considering that we want quick gains without the necessities of research and market knowledge, technical analysis should be primarily used by the proposed system to predict trade decisions for the user.

The use of technical analysis, however, requires the crunching of a myriad of numbers as well as inspecting detailed charts; tasks that we do not want to waste the time even attempting to perform. The use of software should not only automate this process but also produce prediction results with the accuracy of that greater than a human. Each of these results should also come with a reasonable confidence value asserting to the user how probable the given decision of "buy", "sell", or "hold out/sit in" is to be correct. There are various methods to gauge stock trends, and as many as possible should be used to increase prediction accuracy. The result of each model should be displayed to the user with up to date information regarding the current stock, as well as an overall decision based on a weighted average of all models. All of this should be easily accessible at 1-2 mouse clicks away. According to Akamai's *State of the Internet: Third Quarter, 2013 Report*[1], in the United States a resident, on average, has 9.8Mbps broadband connection speed and an average web page takes 2.8 seconds to load. Therefore, following these standards, loading times of this service should be no more than 2.8 seconds on at least a 9.8Mbps connection.

So far, the use of software to aid those of us looking for quick and effortless stock forecasts has been conceptually realized, but what about those that not only want a decision but also aspire to learn the process behind the decision? Some might argue that this will defeat the purpose of not having to be knowledgeable to get reasonable stock predictions. The beauty of this feature, however, is that it is completely optional and thus will serve both sides of the crowd: those that just want quick, easy, and well tested advice to base their decisions on and those that are looking for a little more reasoning behind those decisions. Thus, differing from other designs, each prediction result should have a way to reveal an explanation to the user of the technical analysis used to get to that specific result; an environment for users to learn similar processes used by professionals if he/she chooses to.

With the ability to gain quick prediction for a single stock, what happens if a user would like to broaden his/her investment portfolio with multiple stocks? Again, software can alleviate the issue. With an account based system, the web service should allow for us to track and view multiple predictions all at the same time based on stocks we both own or are simply interested in purchasing. When the system predicts and notices an impending prominent decision, we should be immediately alerted and informed of that decision. With the addition of personal accounts, however, security always becomes a concern. Using statistics gathered from the bank login protection software that runs on 4 million PCs over the last year, security vendor *Trusteer* found that 73 percent of users were using the password for their online bank sites to access at least one other website[2]. Thus, although no real personal data about users would be stored (besides their full name), passwords should always be kept secure. If a

---

[1]http://www.akamai.com/dl/akamai/akamai-soti-q313.pdf

[2]http://landing2.trusteer.com/sites/default/files/cross-logins-advisory.pdf

particular user is not logged into the service, however, he/she should still be able to search for a stock and view the predictive decision associated with it. Furthermore, if no stocks come to mind, we should be able to ask the system to recommend a stock to purchase based on the highest predicted gain for that day. All of these features should be organized into an easy to use modern user interface not only accessible to desktop browsers, but the mobile workspace as well. Through these methods, we will gain the organization and clutter free environment needed to mange multiple stocks. The current method of tracking both owned and interesting stocks is simply by remembering and memorizing. An automated method would ultimately relieve us from this task. Furthermore, making the system mobile friendly, will allow these services to be portable, especially in an era where people are always on the move and attached to smartphones and tablets.

Our business plan is to offer the service free and support the operations from commercial advertisement proceeds. Obviously, due to budget and technical limitations, not all existing stocks can be accounted for and thus, only a limited amount can be predicted by this service. These concerns also restrict the frequency of predictions as well as the rate of updating current stock prices. Thus, an administrative account should be created to allow changes to these limits should the service prove to be popular and budget allows for upgrades to superior hardware. The system should log and display to the administrator how long a prediction session has taken so that he/she may make adjustments to timings and/or the list of predicted stocks to compensate. The system should also keep track of what stocks users are searching for display these analytics to the administrator.

5. Glossary of Terms

Algorithms: a step-by step procedure for calculations.

Artificial intelligence: the intelligence exhibited by a machine or software, the branch of computer science that develops machines and software intelligence. This intelligence can perceive its environment and take actions that maximize its chances for success.

Closing price: the final price at which a security is traded on a given trading day. It represents the most up-to-date valuation of a security until the next trading day.

Database: an organized collection of data that are typically organized to model relevant aspects of reality in a way that supports process rewiring this information.

Data mining: The computational process of discovering patterns in large data sets involving methods at the intersection of artificial intelligence, machine learning statistics, and data base systems.

Fundamental analysis: a method of evaluating stocks. It tries to measure the value relating to economic, financial and other qualitative and quantitative factors to produce a value that an

investor can compare with current stock prices. With this they can decide what action to take with the security.

Long term investment: an account on the asset side of a company's balance sheet that represents the investments that a company intends to hold for more than a year. They may include stocks bonds, real estate and cash

Machine learning: A branch of artificial intelligence, concerned with the construction and study of systems that can learn from data.

Neural network: conceptually based off the central nervous system, it interconnects systems of neurons that can calculate values for inputs by feeding information through the network.

Short term investment: An account in the current assets section of a company's balance sheet. This account contains any investments that a company has made that will expire within one year. For the most part, these accounts contain stocks and bonds that can be liquidated fairly quickly.

Stocks: A type of security that signifies ownership in a corporation and represents a claim on part of the corporation's assets and earnings.

Technical analysis: a method of evaluating securities by analyzing the statistics generated by the market today. It uses charts and other tools to identify patterns (such as past price and volume) that can suggest future activity of that security

Technical indicators: look to predict the future price levels or simply the general price direction of a security by looking at past patterns.

Time Horizon: the length of time over which an investment is made or held before it is liquidated. It can range from seconds-decades.

User interface: the space where interaction between humans and machines occur.

"Watch List": a list of tracked stocks stored within the system for a particular registered user.

Web service: is a method of communicating between two electronic devices over the World Wide Web

6. System Requirements

6.1. Enumerated Functional Requirements

IDENTIFIER	PW	REQUIREMENT
REQ1	5	The system shall allow the administrator to add and remove stocks from the list of stocks that predictions are made on.
REQ2	5	The system shall continuously gather the time series of the current market data (stock prices, trading volumes, etc) for a set number of companies.
REQ3	5	The system shall periodically apply prediction algorithms or models on the obtained data and store the results to a central database.
REQ4	5	The system shall allow for users to search for a particular company and display the predicted trade decision associated with that stock.
REQ5	4	The system shall allow users to request a stock that is predicted to have a large gain in the near future.
REQ6	4	The system shall obtain and display a confidence value for each prediction given to the user.
REQ7	3	The system shall support registering new investors by providing a real-world email, which shall be external to our website. Required information shall include a unique login ID and a password that conforms to guidelines, as well as users first and last name. Upon successful registration, the system shall set up an account for the investor.
REQ8	3	The system shall allow for registered users to track stocks that he/she owns or is interested in purchasing. Registered users shall be alerted via various methods chosen by that user, if an impending prominent prediction is made for a stock he/she has chosen to track
REQ9	2	The system shall allow a method for users to learn the professional analysis used for each prediction method if he/she chooses to.
REQ10	2	The system shall allow for the administrator to alter the rate at which current stock prices are updated and the rate at which predictions are made.
REQ11	2	The system should log and display to the administrator the time taken for previous prediction sessions.
REQ12	1	The system should track what stocks users are searching for and log and display these analytics to the administrator.

IDENTIFIER	PW	REQUIREMENT
REQ13	1	The system should provide the user with options to troubleshoot various issues such as, but not limited to, a FAQ, tutorial system, or methods to contact technical support (i.e send an email to the admin) and show solutions to similar issues other users had.

Table 1: Enumerated functional requirements for a web based stock forecasting system.

In the customer statement of requirements, our customer explained the priority weights (PW) as follows. Obviously, the main solution this system is going to be proposing is to have an automated process of predicting future stock prices. Naturally, not all listed companies can be accounted for. The World Federation of Exchanges has a total of 42,417 listed companies as of January 2014[1] and without sophisticated hardware, even approaching this number of daily predictions is unfeasible. Thus, before predictions can even be made, someone must select the range of stocks that the system will handle. Therefore, REQ1 is given highest priority. Furthermore, the core function of the system is to forecast stock prices, regularly update current prices so accurate predictions can be made, and to allow users to be able to search for those predictions. Thus, REQ2, REQ3, and REQ4 respectively are also given highest priority.

Requirement REQ5 allows users to suggest for stock that is predicted to have a very high gain and REQ6 allows users to be reassured of predictions prescribed to them by means of a confidence value. These are added features and not essential (although highly desirable) to the core functionality of the system as stated before, and thus REQ5 and REQ6 were given a lower priority.

Requirement REQ7 deals with allowing users to register for an account with the system and REQ8 allows those registered users to be able to track stocks and add them to his/her "watch list". These are given an even lower priority because users will not need to be registered to be able to use essential features of the system. Furthermore, users can track certain stocks by other means such as remembering ticker names and performing individual searches of each stock. A "watch list" serves as a means of convenience for users.

Requirement REQ9 allows users to learn the decision making behind each prediction. This is given an even lower priority because not all users will be interested in this feature. For those that are, however, other means of "learning" are available such as researching through books or the internet. This feature again only serves as a convenience for the user by giving them quick and easy information all in one place. REQ10 and REQ11 are given similar priorities. REQ12 and REQ13 are desirable but are given lowest priority and will only be implemented if development time allows.

The requirement REQ5 does not specify the number of stocks that will be suggested to users. We may introduce an option for the user to choose an arbitrary number or, for ease of

[1] http://www.world-exchanges.org/statistics/monthly-reports

use, will make that decision for them. Furthermore, REQ8 is the compound of two similar requirements:

> REQ8a: The system shall allow for registered users to track stocks that he/she owns or is interested in purchasing.
>
> REQ8b: Registered users shall be alerted via various methods chosen by that user, if an impending prominent prediction is made for a stock he/she has chosen to track.

Since the second depends heavily on the first (and the only reason why an account is needed in the first place), it was thought to be more useful to combine these into one requirement. For testing purposes, however, it may be useful to separate the requirement into its two sub parts.

Finally, requirement REQ12, was suggested by the customer to further allow the administrator to satisfy user needs based on their searches. This is better said than done, however. The combination of available characters to input for a search keyword are limitless and thus one must take into consideration how the system will handle storing and keeping track of these almost infinite combinations. One solution may be to somehow validate a keyword to being meaningful or not (i.e. does it relate to a World Federation of Exchanges listed company?) and only track of those that are.

Some of the above requirements are vague and may need to be further detailed into sub requirements. Note that these will not be listed as requirements on their own as fragmentation can cause clutter and a lack of simplicity.

IDENTIFIER	REQUIREMENT
REQ3a	(As REQ3 in Table 1)
REQ3b	The system shall only predict prices for stocks up to a week ahead.
REQ4a	(As REQ4 in Table 1)
REQ4b	The system shall only load and display at most 5 results at a time.
REQ8a	(As REQ4 in Table 1)
REQ8b	The system shall allow for registered users to remove stocks from their “watch lit”
REQ8c	The system shall allow for registered users to set their method of notification (i.e. email, text, or none).

Table 2: Extended list of functional requirements from Table 1.

Requirement REQ3b introduces a forecast “length” business policy:

> **BP01**: The system shall only predict prices for stocks up to a week ahead.

There are many options when it comes to stock forecasting as time frames can range from weeks to months and even years. A real world implementation can take advantage of varying prediction models for each time frame but for simplicity we will only deal with a week where, because of the volatility of the market, we feel prediction is most needed. This will also leave room for further upgrades to the system in the future where the policy allows for predictions to span a greater number of time frames.

REQ4b is intended to keep loading times low (since charts are needed to be created for each result) for the user and introduces another business policy:

BP-02: The system shall only load and display at most 10 results at a time.

The number of displayed results for a search may differ depending on the acceptable amount of time needed for loading those results and thus may be altered in the future.

Furthermore, REQ8b gives the option to registered users to remove tracked stocks from their "watch list" as well as set their method of notification. We have limited these methods to two (i.e text and email) but a real world implementation might choose to enact more such as a telephone call or Facebook message.

6.2. Enumerated Non-Functional Requirements

IDENTIFIER	REQUIREMENT
REQ14	The system shall ensure that if a copy of the database were to be acquired unlawfully, all user passwords shall remain secure.
REQ15	The system shall be able to run through all prediction models for each and every listed stock within 1 hour.
REQ16	The system shall allow for users to get a prediction for a stock within at most 2 mouse click.
REQ17	The system should insure that any features that do not require a user to be logged in are not hidden to unregistered users.
REQ18	The system shall have loading times no greater than 2.8 seconds with at least a broadband speed 9.8Mbps.
REQ19	The system should be able to run with core functionality intact from a smartphone/tablet workspace.

Table 3: Enumerated non-functional requirements for a web based stock forecasting system.

Since the system will be storing user accounts, REQ14 needs to be specified for security purposes. REQ15 and REQ18 insure that latency are kept to a low since the system will be a

web service where a fraction of a second in loading can be the difference between user satisfaction and disappointment. Furthermore, the customer intends for the service to be as easily accessible as possible and thus REQ17 insures visitors to the service are not locked out of key features and REQ16 insures user efforts for the most prominent feature (searching for predictions) is kept to a low. REQ19 finally insures that the system is able to be used for not only desktop browsers but also smartphones and tablets as well. This requirement is of the least priority because we will be working on the functionality for modern desktop web browsers first and then optimize for mobile if development time allows.

6.3. On Screen Appearance Requirements

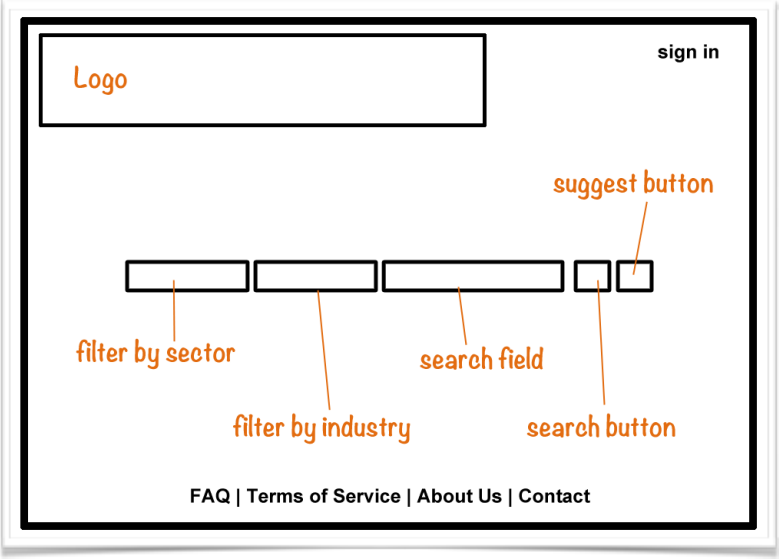


Figure 1: Customer's sketch of homepage

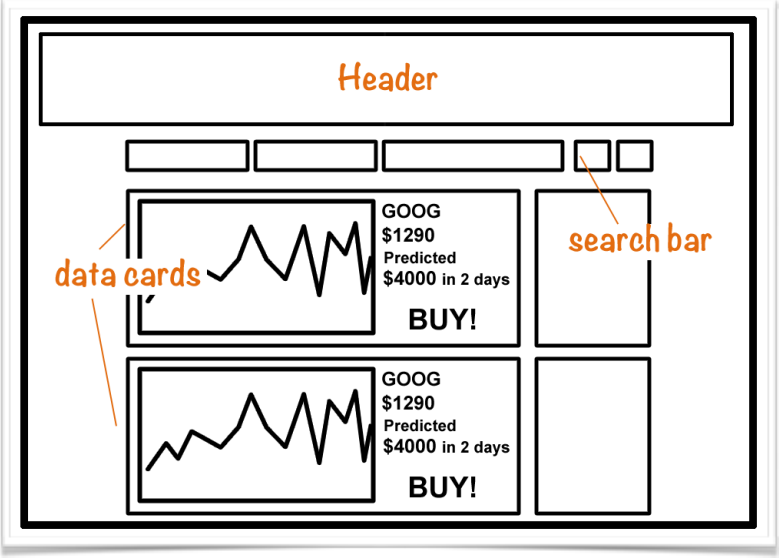


Figure 2: Customer's sketch of "results" page showing the organization of data.

IDENTIFIER	REQUIREMENT
REQ20	Figure 1 shows the design is for the first page a user comes across when accessing the web page. A search bar should be displayed with filtering options as well as a both a “search” and “suggest” button. This is also where a user will be allowed to sign in and visitors to register for an account. Various links should be displayed on the bottom of the page such as FAQ, About Us, Terms of Service, Contact, etc.
REQ21	Figure 2 shows the design for the page after logging in, after making a search, or after requesting a suggestion. All three of these pages will share similar layouts since they all will display similar information (where logging in will show tracked stocks). A persistent search bar is displayed at the top while stock data are given in cards filled with a chart of historical prices, current price, predicted price, confidence value, and predicted decision (buy, sell, hold/ sit-out). Most pages will use this “card” format to display information and thus there isn't any need to provide requirements for secondary pages (i.e. contact, registration, about us, etc).

Table 4: On screen Appearance requirements for a web based stock forecasting system.

6.4. Acceptance Tests

Acceptance tests that the customer will run to check that the system meets the requirements are as follows (not all requirements will be elaborated on due to limited time):

Acceptance test cases for REQ1:

ATC1.01 Use the administrative account to add a stock that does not already exist within the system. (pass: stock shows up on the list of stocks currently being tracked by the system)

ATC1.02 Use the administrative account to add a stock already being tracked by the system. (pass: stock does not show up twice on the list of stocks currently being tracked by the system)

ATC1.03 Use the administrative account to add an invalid stock (random letters). (pass: invalid stock does not show up on the list of stocks currently being tracked by the system)

ATC1.04 Use the administrative account to remove a stock from within the system. (pass: stock does not show up on the list of stocks currently being tracked by the system)

Acceptance tests for REQ2:

ATC2.01 View the current price of any stock within the system. Wait a time equivalent to the update timer and view the current price of the same stock. (pass: the two viewed prices should be different)

It should be noted there is a possibility that the price will not change after a successful update has been made but this possibility is very low considering that prices change continuously every time a trade goes through in real time and thus will be ignored.

Acceptance tests for REQ3:

ATC3.01 View the predicted price for a given stock and the day the stock is predicted to get to that price. Wait until stated day. Compare predicted price and actual price. (pass: predicted price and actual price have a small difference)

Since the system is working with predictions there really is no simple way to test that these predictions are working besides actually comparing predicted prices to actual prices. Using the given confidence value's associated with a prediction should give the customer an idea of how close a predicted price should be with its actual counterpart.

Acceptance tests for REQ4:

ATC4.01 Search for a stock known to be within the system using its ticker symbol. (pass: known stock's related data are shown)

ATC4.02 Search for a stock known to be within the system using a company name or part of a company name. (pass: known stock's related data are shown)

ATC4.03 Filter a search by a sector or industry and search without a keyword. (pass: all stocks within the given sector and industry are shown)

ATC4.04 Search by entering a random sequence of keys. (pass: no significant results should be found)

For test case 4.04 the customer should make sure that his/her random sequence of keys does not relate to any known company name or ticker symbol as that might trigger a valid search.

Also for test case 4.02, more than one result might be shown if the customer enters a vague search term such as “micro” in which the system might interpret the keyword to mean both “Microsoft Corp.” and “Micron Technology Inc.” It should be noted that for test case 4.03 the number of results loaded at a time will be limited by BP-02.

Acceptance tests for REQ5:

ATC5.01 Click the suggest button. (pass: stocks with high predicted gains are shown as a result)

ATC5.02 Select a sector and industry and click the suggest button. (pass: stock from the given sector and industry with high predicted gains are shown as a result)

Acceptance tests for REQ6:

ATC6.01 Search for a particular stock or get a suggestion from the system. (pass: a confidence value should be shown for every prediction)

Due to limited time, only the highest priority requirements were given acceptance tests. The customer can formulate his/her own tests for the remaining lower priority requirements.

7. Functional Requirements Specification

7.1. Stakeholders

Three Stakeholders can be identified:

1. Users: any user of the web service registered or not.
2. Advertisers: provider of advertisements for the web service.
3. Administrator: maintains and updates website services.

7.2. Actors and Goals

Eight actors can be identified:

1. Registered User: a registered user.
2. Visitors: any unregistered user
3. User: both a registered user and visitor (will be used for diagrams and descriptions where both a registered user and a visitor can interact with the system)
4. Database: records of stock information (i.e historical prices, prediction results, confidence value, etc), user data (i.e. username, password, tracked stocks, email, etc), and system data (timers, search logs, prediction time logs).

5. Price Provider (i.e. Yahoo! Finance): Provides the current pricing of a stock of interest
6. Timer: Tells the system when predictions should be made and when current stock prices should be updated.
7. Grapher (i.e. Google Charts): Provide visual charts from raw data.
8. Administrator: a special case User that maintains and updates website services.

7.3. Use Cases

7.3.1. Casual Description

The summary use cases are as follows:

UC-1: Login — Allows user to access their account and view information for their tracked stocks. [Derived from REQ7]

UC-2: AddStock — Allows the admin to add a stock to the list of stocks predictions are made on. («include» login). [Derived from REQ1]

UC-3: EditPredictionTimer — Allows for the admin to alter the rate at which prediction are made. («include» login). [Derived from REQ10]

UC-4: EditUpdateTimer — allows for the admin to alter the rate at which current stock prices are gathered («include» login). [Derived from REQ10]

UC-5: RemoveStock — allows the admin to remove a stock from the list of stocks predictions are made on («include» login). [Derived from REQ1]

UC-6: Search — allows a user or visitor to search for a particular stock through keywords and filter by sector or industry and view its corresponding prediction . [Derived from REQ4]

UC-7: Suggest — allows a user to request for the system to suggest a stock that is predicted to have the highest gain. [Derived from REQ5]

UC-8: Track — allow users to add a certain stock to their "watch list". («include» login, «include» search or «include» suggest). [Derived from REQ8]

UC-9: RemoveTrackedStock — allow users to remove a certain stock from their "watch list" [Derived from REQ8]

UC-10: Register — allow a visitors to fill out the a registration form and become a user. [Derived from REQ7]

UC-11: Logout — allow users to log out of the system. [Derived from REQ7]

UC-12: Update - allows timer to initiate the loading of current stock prices from the price-provider into the database. [Derived from REQ2]

UC-13: PredictAndNotify — Allows timer to initiate the prediction of future stock prices to be later stored into the database. Registered Users are then notified if an change in prediction is made for a stock he/she has chosen to track. [Derived from REQ3, REQ6, REQ8, REQ11]

UC-14: Learn — allows user and visitors to learn the decision making process behind a prediction by clicking on that prediction. [Derived from REQ9]

UC-15: Support — allows user to access technical support and send emails to the admin. [Derived from REQ13]

Some alternative use cases may be considered. For example, Use Case UC-13: PredictAndNotify can be broken up into two separate use cases (one for the prediction process and one for the notification process). This would require for the Timer actor to keep track of yet another time period. Since the customer stated that alerts should be immediately sent out as soon as a change in prediction for a user's tracked stock is detected, we concluded it would be best to have the notification process right after predictions are made. Thus, only one initiating actor is needed for both.

Furthermore, Use Case UC-6: Search, further elaborates on searching methods such as having the option to filter that were not strictly implied by REQ4. This is to give users added convenience as well as let them be able to "play" with the systems search functionality without having to name specific companies.

It should be noted that the different prediction models are not stated as extension use cases. This is because during the analysis process we still do not know all of the models we will be using and are still researching viable prediction methods. In addition, going into these models will lead into talk about implementation which is not desired in the analysis phase.

7.3.2. Use Case Diagram

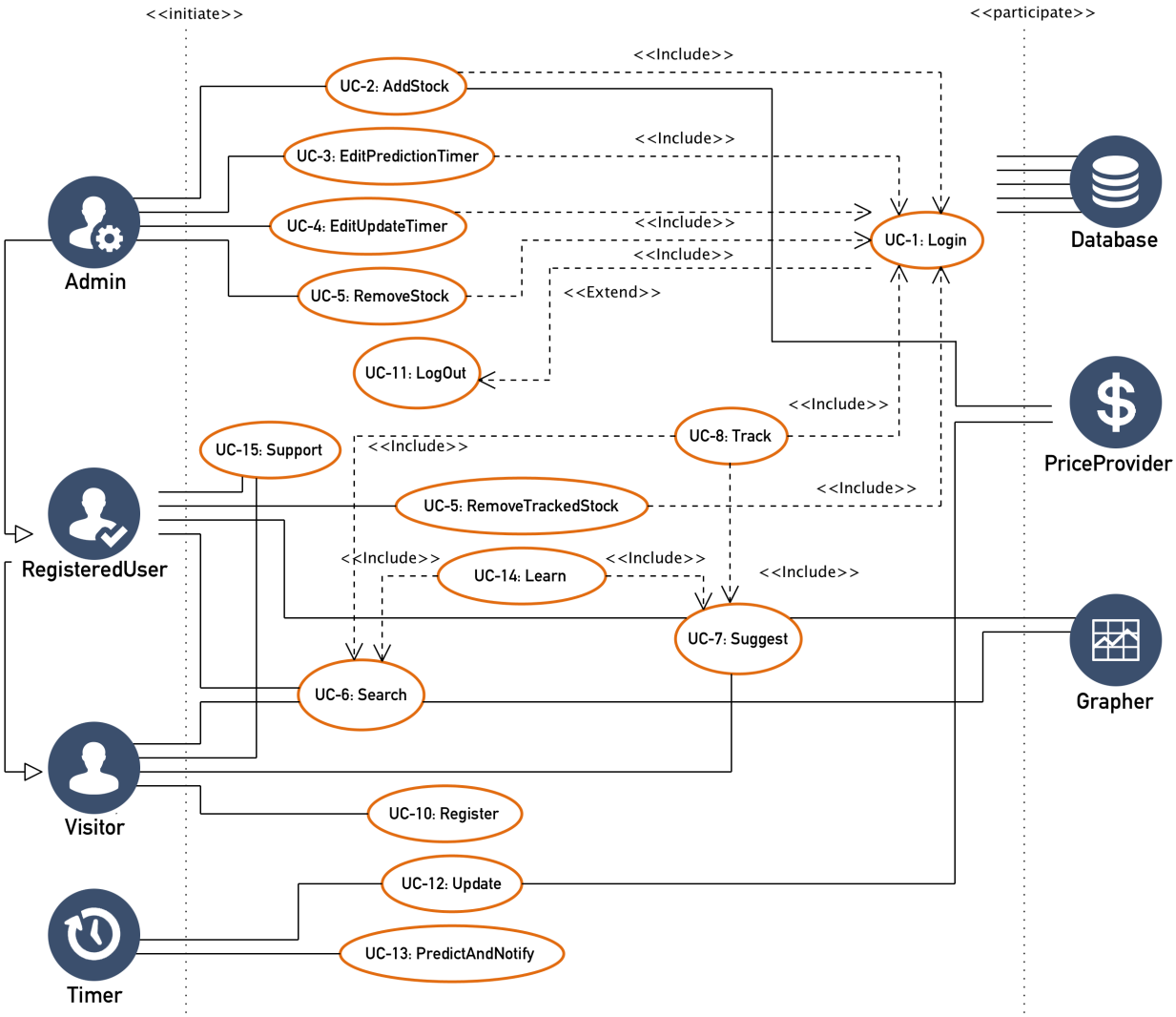


Figure 3: Use case diagram for our proposed web based stock forecasting system.

The use case diagram is shown in Figure 3. As shown by the diagram a registered user is a specialization of a visitor and an administrator is a specialization of a registered user. A visitor can search or get a suggestion from the system. A registered user can accomplish all that of a visitor but also is allowed to track stocks. Tracking a stock is an extension of a search or suggestion because the registered user must first retrieve the stock before he/she can track it. Similarly for UC-14: Learn, a user must first get a prediction before he/she can learn the involved analysis behind it and thus it must be an extension of a search or suggestion. It should be noted that although an administrator has similar rights to that of a registered user (search, track, get a suggestion), the diagram does not show these connections because these are not ultimately the administrators goals. The database is a participant in all use cases, and thus its connections are not shown to reduce clutter.

7.3.3. Traceability Matrix

REQ	PW	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11	UC-12	UC-13	UC-14	UC-15
REQ1	5		X			X										
REQ2	5												X			
REQ3	5													X		
REQ4	5						X									
REQ5	4							X								
REQ6	4													X		
REQ7	3	X									X	X				
REQ8	3								X	X				X		
REQ9	2														X	
REQ10	2			X	X											
REQ11	2													X		
REQ12	1						X									
REQ13	1															X
MAX PW		3	5	2	2	5	5	4	3	3	3	3	5	5	2	1
TOTAL PW		3	5	2	2	5	6	4	3	3	3	3	5	14	2	1

Figure 4: Traceability matrix mapping the use cases to system requirements. Priority weights (PW) are given in Table 1.

The above traceability matrix shows how use cases map to system requirements. With calculated priority weights the use cases can be prioritized by the following:

UC-13 > UC-6 > UC-2, UC-5, UC-12 > UC-7 >…
… UC-1, UC-8, UC-9, UC-10, UC-11 > UC-3, UC-4, UC-14 > UC-15

We will select the 5 use cases with highest priority for further elaboration. UC-5: RemoveStock, although a high priority, only deals with clearing the database for given stocks and thus will not be elaborated on. In its place we will further elaborate on UC-8: Track.

7.3.4. Fully Dressed Description

Use case UC-2: AddStock, allows the administrator to add stocks into the system. This is the first logical step necessary to instantiate the system since out of the box it will contain no stocks for predictions to be made on. It is up to the administrator to pick and choose companies he/she views customers will want to get forecasts for. As stated before, there are over 42,000 companies that the administrator can choose from and having a simple check box for each is unfeasible. He/She should rather enter desired stocks into a text box and have the system verify that the company exists. Since each and every stock already has its own unique identifier, known as a ticker symbol, the symbol will used for searches rather than terms susceptible to vagueness such as company names.

USE CASE UC-2: ADDSTOCK

Related Requirements: REQ1
Initiating Actor: Administrator
Actor's Goal: to add stocks to the list of stocks within the database (can be empty) that predictions are made on.
Participating Actors: Price Provider, Database
Preconditions: Administrator is logged in.
Success End Condition: All of the chosen stocks are added to the list of stocks within the database that predictions are made on. Historical prices for the given stocks are retrieved from the Price Provider and stored within the database.
Failed End Condition: The administrator is notified of all stocks out of the entered stocks that were not found in the Price Provider's database.

Flow of Events for Main Success Scenario:

Include::Login(UC-1)

1. **Administrator** enters ticker symbols for the stocks he/she wishes to add into an "Add Stocks" text field. If multiple ticker symbols are inserted, they should be separated by commas.
2. **Administrator** clicks the "Add Stocks" button.

(loop: for each ticker symbol entered)

1. **System** verifies the stock currently does not already exist within the **Database**.
2. **System** creates a request URL for historical prices using the ticker symbol.
3. **System** requests historical prices from the **Price Provider**.
4. **Price Provider** returns historical prices for the given stock.
5. **System** creates a request URL for current price using the ticker symbol.
6. **System** requests current price from the **Price Provider**.
7. **Price Provider** returns current price for the given stock.
8. **System** stores the historical prices for the given stock within the **Database**.

(end loop)

9. **System** notifies **Administrator** that all stocks were successfully added.

Flow of Events for Alternate Scenarios:

5a. Price Provider returns an invalid URL.

1. **System** records the ticker symbol.
2. **System** gets the next ticker symbol entered by the **Administrator** and goes to step (3) of the main success scenario. Once the loop has ended step (8) of the main success scenario is not executed.
3. **System** notifies **Administrator** of all stocks that have failed to be added.

Note that in alternate scenario 5a, an invalid URL is detected which in turn corresponds to an invalid ticker symbol (most likely caused by a typo). The system simply records the invalid symbol, and moves on to the next stock to be added. At the end, the admin is notified of all stocks that have failed to be added. Another version of this alternate scenario can have the admin be notified immediately as a stock fails to be added rather than waiting until the end of the sequence. We see no significance in choosing one over the other as both scenarios will ultimately return the same information.

In use case UC-6, it is assumed that a user can enter any combination of characters into the search field. Since searches (valid or invalid) need to be recorded for analytics, the system must first verify that an invalid search for a stock, although not located within our database is still a stock and not a random set of characters. This process of connecting keywords to known ticker symbols proves to be very difficult and thus will be ignored by our system, introducing another business policy:

BP-03: All search keywords (valid or invalid) will be logged and tracked

We believe that searches of gibberish will not be a frequent matter and thus will weed themselves out by having very low reoccurrences within the database (searches for "Microsoft" on the other hand may occur 10-100 times and thus will be at the top of analytics charts). Therefore, frequently clearing the database of any of keywords with low occurrences can be a solution to this problem. In the future, we may choose to invest in the time to create a way to connect certain keywords to ticker symbols. This will reduce the amount of "junk" stored in the database at a given time but in itself may be considered another software project.

USE CASE UC-6: SEARCH

Related Requirements: REQ4, REQ12
Initiating Actor: Visitor, Registered User (the word "User" will be used to represent both)
Actor's Goal: to find related information for a particular stock he/she has in mind.
Participating Actors: Database, Grapher
Preconditions: none
Success End Conditions: For the searched stock the visitor is provided with prediction results, confidence values, current price, and a line graph of the changing prices over time.

Failure End Condition: Search does not result in any stock stored within the database. A message indicating that system is currently not tracking his/her specified stock is displayed to the user and he/she is prompted to try another search.

Flow of events for the main success scenario:

1. **User** types in the name or ticker symbol of a stock he/she would like to search for inside the "search" text field.
2. **System** verifies that the given stock exists within the **Database**.
3. **System** increments the "search" counter for that stock within the **Database**.
4. **System** retrieves historical prices of that stock from the **Database**.
5. **Systems** sends historical prices to the **Grapher**.
6. **Grapher** returns a line graph of the change of prices over time to the **System**.
7. **System** retrieves prediction results for the given stock along with the confidence value for each prediction and the current price of the stock from the **Database**.
8. **System** displays the graph, prediction results, confidence values, and current stock price to the **User**.

Flow of Events for Alternate Scenarios:

2a. The given stock does not exist within the **Database**

1. **System** checks if the failed keyword exists within the **Database**.
   1a. Exists: **Systems** increments the "search" counter for the failed keyword
   1b. Doesn't Exist: **System** adds the failed keyword to the **Database**.
2. **System** displays a "no results found" message to the **User** and he/she is prompted to try another search.

Use case UC-8: Track, requires that a user be logged in and initiate a search or request a suggestion from the system.

USE CASE UC-8: TRACK

Related Requirements: REQ8
Initiating Actors: Registered User
Actor's Goal: Add a specific stock to a "wish list" so that the stock information is provided on the user's profile for easy access.
Participating Actors: Database
Preconditions: Registered User must be logged in and initiated a successful search (UC-5) or suggest (UC-6)
Success End Conditions: The selected stock the Registered User want to track will be added to his/her "watch list".
Failure End Conditions: no meaningful failure end conditions.

Flow of Events for Main Success Scenario:

Include::Login(UC-1)

Include::Search(UC-6) or Include::Suggest(UC-7)

1. **Registered User** clicks the "track" button for the stock he/she wishes to track.
2. **System** verifies stock does not already exist within the "watch list" stored for the Registered User within the **Database**.
3. **System** stores the stock within the "watch list" for the user in the Database.
4. **System** notifies **Registered User** the stock of interest has been added to his/her "watch list".

Flow of Events for Alternate Scenario:

2a. System detects the stock already exists within the "watch list" stored for the Registered User within the Database.

1. System notifies Registered User the stock of interest already exists within his/her "watch list"

An alternate scenario 2a is given where the stock happens to be already added to the registered user's "watch list". This scenario can be taken out altogether since the user's goal (to add a stock to his/her "watch list") is already accomplished. We thought it would be important to give the user feedback to his/her actions and thus decided to keep this alternate scenario.

Use case UC-12: Update deals with updating prices within the database and is similar to UC-2: AddStock in that it also requests current and historical pricing from the Price Provider. In this case, however, the possibility of requesting invalid stock information is removed since the system is specifying the requests rather than a human. This use case must not only deal with updating current day stock prices but also any day in-between the last stored price and the current day (If the system were to be shut down for a period of time longer than 24 hours).

USE CASE UC-12: UPDATE

Related Requirements: REQ2
Initiating Actors: Timer
Actor's Goal: To update pricing for the current day and any missing days in the Database for all stocks.
Participating Actors: Database, Price Provider
Preconditions: There is at least one stock within the database.
Success End Conditions: Prices stored in the Database are up to date for all stocks.
Failure End Conditions: no meaningful failure end conditions.

Flow of Events for Main Success Scenario:

1. **Timer** notifies the **System** to start updating.
2. **System** retrieves list of all stocks from the **Database**.
3. **System** retrieves current date.

(loop: for all stocks)

4. **System** retrieves date of last stored price within the **Database**.
5. **System** verifies the last retrieved date and current date match.
6. **System** creates a request URL using the ticker symbol and current date for the stock.
7. **System** requests current price from **Price Provider** using the request URL
8. **Price Provider** returns an up to date price for the stock.
9. **System** updates the pricing (corresponding to the current day) in the **Database**.
10. **System** waits two seconds.

(end loop)

11. **System** resets the **Timer**.

Flow of Events for Alternate Scenario:

5a. **System** notices last retrieved date and current date do not match.

1. **System** creates a request URL using the ticker symbol, current date, and last retrieved date for the stock.
2. **System** requests all prices in-between the current and last retrieved date from the **Price Provider** using the request URL.
3. **Price Provider** returns all prices in-between the current and last retrieved date.
4. **System** adds the missing historical prices in the **Database**.
5. Flow is restored to step 7 of Main success scenario.

It should be noted that in alternate scenario 5a, the request for both historical and current prices can possibly be combined together. We have separated these steps because by researching ahead, we've discovered that our Price Provider does not include a current price when returning historical prices. Also, to prevent the Price Provider from blocking requests we have decided to add a delay of two seconds between each request.

The final elaborated use case is also the most important. UC-13: PredictAndNotify deals with running prediction models for every existing stock within the database, calculating confidence values, calculating gain/loss, as well as notifying registered users of prediction changes. An internal timer is also used to log the amount of time a prediction session takes for analytics to be later displayed to the administrator. Since this process is all automated without any human interaction, no significant failure conditions exist.

USE CASE UC-13: PREDICT AND NOTIFY

Related Requirements: REQ3, REQ6, REQ8, REQ11
Initiating Actors: Timer
Actor's Goal: To initiate the prediction of future stock prices, store the predictions within the Database, and notify Registered Users of any prediction changes.
Participating Actors: Database
Preconditions: At least one stock exists within the database.
Success End Conditions: Predictions are made for all existing stocks and stored within the Database. Registered Users are alerted if any prediction changes are detected.
Failure End Conditions: no meaningful failure end conditions.

Flow of Events for Main Success Scenario:

1. **Timer** notifies the **System** to start making predictions.
2. **System** resets and starts an internal timer.
3. **System** retrieves list of all stocks from the **Database**.

(loop: for all stocks)
(loop: for all prediction models)

4. **System** predicts the future price of the stock and calculates a confidence value.
5. **System** stores the prediction and confidence value within the **Database**.

(end loop)

6. **System** retrieves results for all prediction models from within the **Database**.
7. **System** calculates overall prediction and gain/loss.

(end loop)

8. **System** stops internal timer.
9. **System** stores the time taken to make all predictions in the **Database**.
10. **System** retrieves list of all **Registered Users** from the **Database**.

(loop: for all Registered Users)

11. **System** retrieves list of all stocks tracked by the Registered User and his/her suggested means of alert (i.e. text, email) from the **Database**.

(loop: for all tracked stocks)

12. **System** retrieves current and previous predictions for the stock from the **Database**.
13. **System** verifies that the current and previous predictions are different.
14. **System** alerts Registered User through his/her suggested means of alert.

(end loop)
(end loop)

Flow of Events for Alternate Scenario:

10a. **System** notices current and previous predictions are the same

1. **System** retrieves the next stock and the flow is returned to step (9)

7.3.5. Acceptance Tests for Use Cases

Test cases for UC-2: AddStock include, but are not limited to:

TEST CASE IDENTIFIER: TC-2.01	
Use Case Tested: UC-2: AddStock — Alternate scenario 5a **Pass/Fail Criteria:** The test passes if the invalid stock is not added to the database and the administrator is notified of the error.	
TEST PROCEDURE:	**EXPECTED RESULT:**
Setup: The administrator logs into the system. Step 1: The administrator enters ticker symbols separated by commas into the “add stock” text box. At least one of the ticker symbols does not relate to a stock and is simply an input of random characters. Step 2: The administrator clicks the “add” button	 System notifies the administrator: [ticker symbol] was/were failed to be added.

Test cases for UC-6: Search include, but are not limited to:

TEST CASE IDENTIFIER: TC-6.01	
Use Case Tested: U6-2: Search — Alternate scenario 2a **Pass/Fail Criteria:** The test passes if the invalid search is added to the database if it doesn't already exist, or its counter incremented if it does it exist, otherwise it fails.	
TEST PROCEDURE:	**EXPECTED RESULT:**
Setup: User enters a keyword relating to a stock not being tracked by our system.	

Step 1: User hits enter or clicks the search button.	System checks to see if the searched keyword relates to any existing stocks within the database. No relation found: System checks to see if the keyword exists within the “searched for” database If keyword exists: System increments the counter associated with the keyword. If keyword does not exist: System adds the keyword to The user is given a “No results found” message and is prompted to try another search.

Test cases for UC-8: Track include, but are not limited to:

TEST CASE IDENTIFIER: TC-8.01	
Use Case Tested: UC-8: Track — Main success scenario **Pass/Fail Criteria:** The test passes if the selected stock is added to the registered user’s “watch list”, otherwise it fails.	
TEST PROCEDURE:	**EXPECTED RESULT:**
Setup: A registered user logs into the system and conducts a search or requests a suggestion from the system. Step 1: The registered user clicks the “track” button on the card displayed for the stock he/she wishes to track.	 System checks to see if the stock already exists within the registered user’s “watch list”. Stock does not exist: Registered user is given notification that the stock has been successfully added to his/her “watch list”

The above test cases are only a sample of the many that can be derived from all use cases success scenarios as well as the infinite amount of alternate scenarios.

7.3.6. System Sequence Diagrams

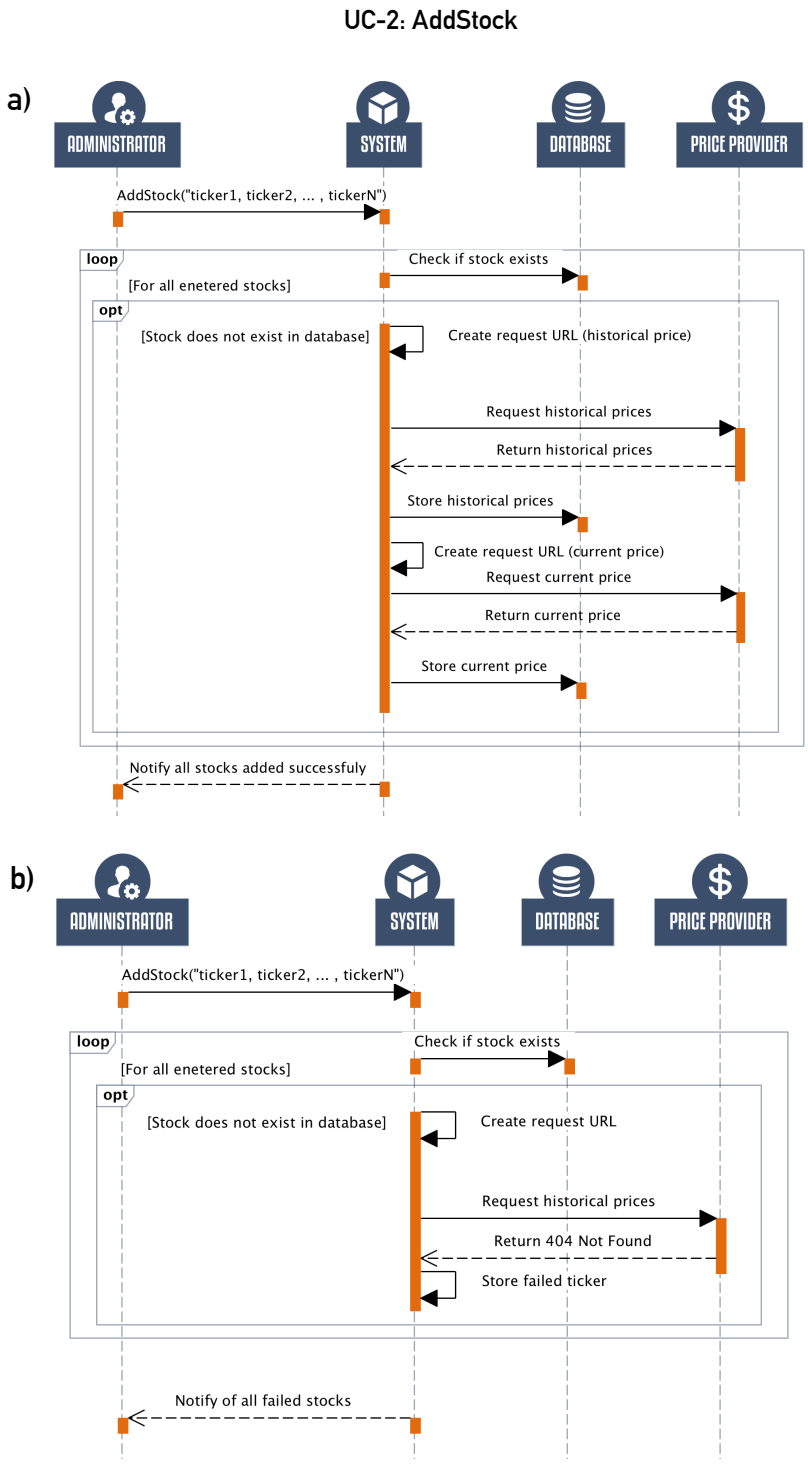


Figure 5: System sequence diagram from UC-2: AddStock. a) Main success scenario. b) An alternate scenario where the request URL does not exist.

Figure 5a shows the system sequence diagram for the main success scenario of UC-2: AddStock. A string of ticker symbols separated by commas is inserted by the administrator which initiates a loop within the system for each ticker symbol. The system first makes sure that the stock does not already exist within the database (to prevent from creating duplicates). If the stock does not exist, the system creates two request URL's using the given ticker symbol: one to request historical prices from the Price Provider and the other to request a current price. The system receives this data and stores it within the database. Once this process is ran for all stocks needed to be added, the administrator is notified of a success. Figure 5b, on the other hand, shows an alternate scenario where the administrator might have made a typo when entering a a specific ticker. Upon creating a request URL and then using that URL to request prices, the system is returned a 404 Page Not Found error. The failed ticker is remembered by the system and the loop moves on to the next stock. Upon looping through all entered tickers, the administrator is notified of the failed ticker.

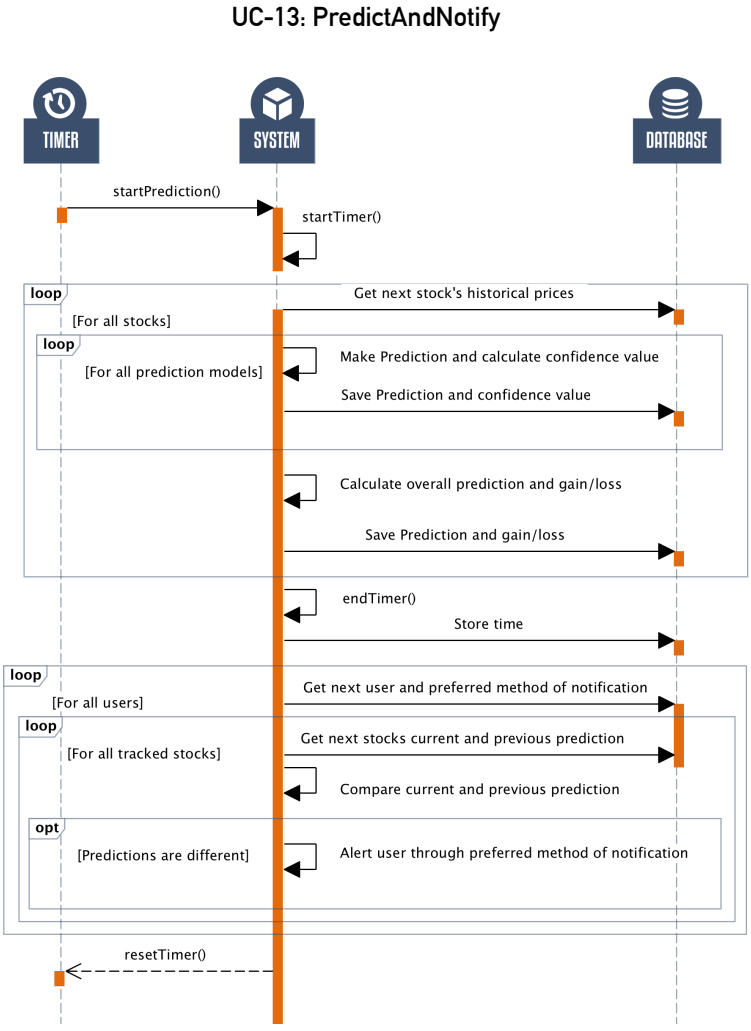


Figure 6: System sequence diagram for UC-13: PredictAndNotify

Figure 6 shows the system sequence diagram for UC-13: PredictAndNotify. The timer initiates this use case and a second internal timer within the system is also started. A loop begins for all stocks being tracked by the system. For each stock, prediction models are run on it's historical prices and confidence values are generated by testing previous predictions to current prices. Once all prediction models are ran, an overall prediction is generated along with the predicted gain/loss for that particular stock. All this information is stored within the database and the internal timer is stopped and the time is logged to be later displayed to the administrator. A second loop is then ran for all registered users. For each of their tracked stocks, a comparison is made between the previous prediction and the current prediction. If the two decisions differ, then a notification is sent out to that user. Finally, the initiating timer is reset.

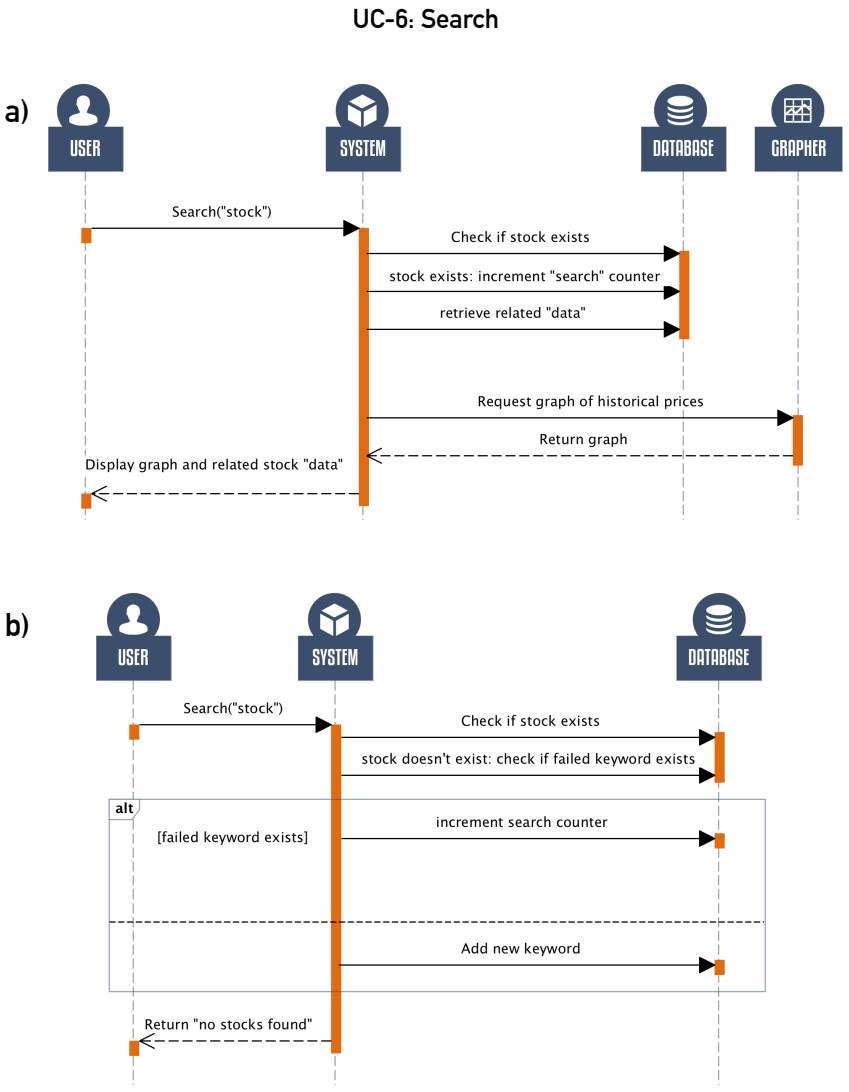


Figure 7: System sequence diagram for UC-6: Search. a) Main success scenario. b) An alternate scenario where a search fails to find any results.

Figure 7a shows the system sequence digram for the main success scenario of UC-6: Search. A user enters a keyword into a search bar and clicks the search button, thus initiating this use case. The system first checks if the keyword can be related to any existing stock within the database. If it exists a search counter is incremented for that stock for analytics and all of its related data is retrieved. This data includes historical prices, prediction results, confidence values, company name, ticker symbol, and current price. The historical data is sent to the grapher which then returns a visual chart from the raw data. The chart along with the other relevant data are then displayed to the user. Figure 7b shows an alternate scenario where the keyword does not relate to any stock being tracked by the system. The system will then check if the keyword exists within the database. If so, the keywords search counter is incremented. If not, the keyword is added to the database. The user is then returned a “no results found” message and is prompted to try another search.

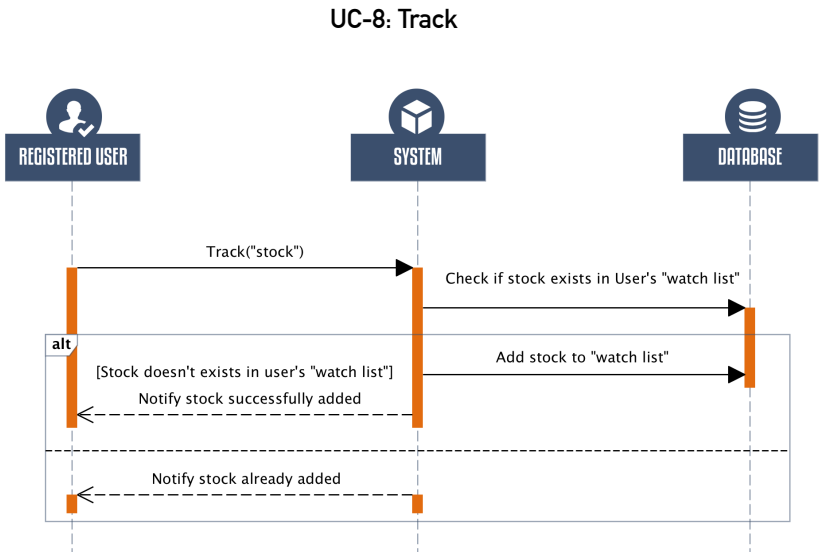


Figure 8: System sequence diagram for UC-8: Track

Figure 8 shows the system sequence diagram for UC-8 Track. A registered user must first trigger a search or request a suggestion from the system. Once data is shown for a given stock, the registered user clicks on the “track” button associated with that stock. The use case is then initiated. The system checks to see if the stock is already being tracked by the registered user. If so, he/she is notified that the given stock has already been added. If not, the stock is added to his/her “watch list” and he/she is notified of a success.

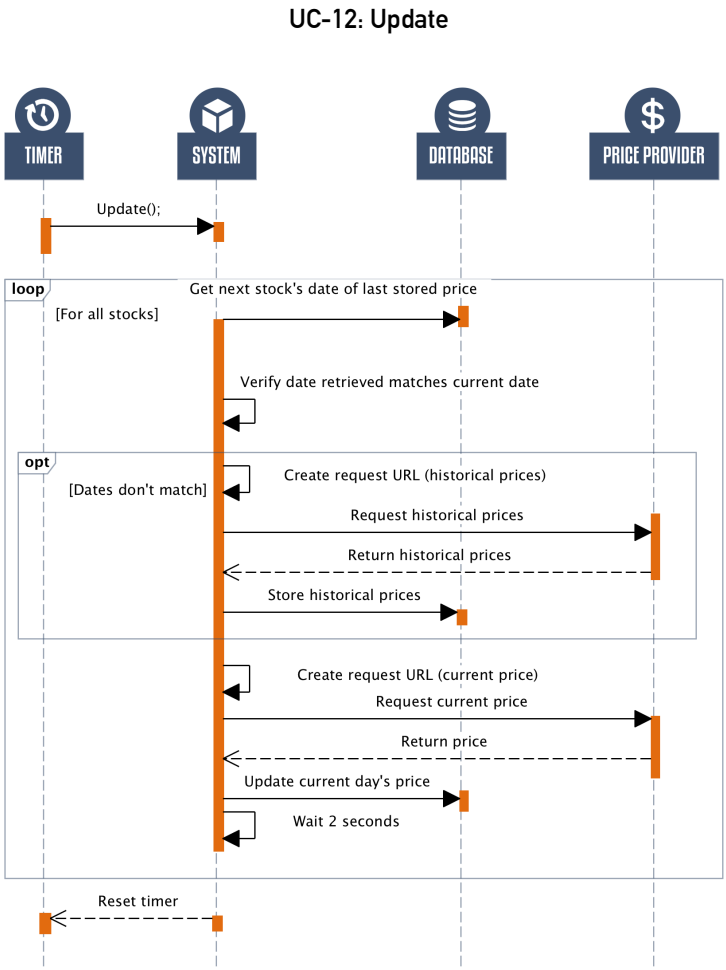


Figure 9: System sequence diagram for UC-12: Update

Figure 9 shows the system sequence diagram for UC-12: Update. This use case is initiated by the timer. The system loops through each and every stock being tracked by the system. For each stock, the date of the last stored price is retrieved and compared with the current date. If the dates don't match, the system creates a request URL for all historical prices in-between the two dates. Using the request URL, historical prices are requested from the Price Provider. Once returned, the data is saved to the database. For any case, a request URL for the current price is made by the system. The current price of a stock is requested from the Price Provider using the request URL. Once returned, the data is stored into the database. The system waits two seconds to prevent the Price Provider from blocking future request due to too frequent accesses. Once all stocks are intreated through, the initiating timer is rest.

8. Effort Estimation

8.1. Unadjusted Actor Weight

ACTOR NAME	DESCRIPTION	COMPLEXITY	WEIGHT
Registered User	a registered user.	Complex	3
Visitor	any unregistered user	Complex	3
User	both a registered user and a visitor	Complex	3
Database	records of stock information, user data, and system data.	Average	2
Price Provider	Provides the current pricing of a stock of interest	Average	2
Timer	Tells the system when predictions should be made and when current stock prices should be updated.	Simple	1
Grapher	Provide visual charts from raw data.	Simple	1
Administrator	special case User that maintains and updates website services.	Complex	3

$$\begin{aligned} \text{UAW} &= 2 \times \text{simple} + 2 \times \text{average} + 4 \times \text{complex} \\ &= (2 \times 1) + (2 \times 2) + (4 \times 3) \\ &= \mathbf{18} \end{aligned}$$

8.2. Unadjusted Use Case Weight (UUCW)

USE CASE	DESCRIPTION	COMPLEXITY	WEIGHT
Login	Allows user to access their account and view information for their tracked stocks.	Simple	5
AddStock	Allows the admin to add a stock to the list of stocks predictions are made on.	Complex	15
EditPredictionTimer	Allows for the admin to alter the rate at which prediction are made.	Average	10
EditUpdateTimer	allows for the admin to alter the rate at which current stock prices are gathered.	Average	10
RemoveStock	allows the admin to remove a stock from the list of stocks predictions are made on.	Average	10

USE CASE	DESCRIPTION	COMPLEXITY	WEIGHT
Search	allows a user or visitor to search for a particular stock through keywords	Complex	15
Suggest	allows a user to request for the system to suggest a stock that is predicted to have the highest gain.	Average	10
Track	Allows user to access their account and view information for their tracked stocks.	Average	10
RemoveTrackedStock	allow users to remove a certain stock from their “watch list”.	Average	10
Register	allow a visitors to fill out the a registration form and become a user.	Simple	5
Logout	allow users to log out of the system.	Simple	5
Update	allows timer to initiate the loading of current stock prices from the price provider into the database.	Complex	15
PredictandNotify	Allows timer to initiate the prediction of future stock prices to be later stored into the database. Registered Users are then notified if an change in prediction is made for a stock he/she has chosen to track.	Complex	15
Learn	allows user and visitors to learn the decision making process behind a prediction by clicking on that prediction.	Simple	5
Support	allows user to access technical support and send emails to the admin.	Simple	5

UUCW = 5 × simple + 6 × average + 4 × complex
= (5 × 5) + (6 × 10) + (4 × 15)
= **145**

UUCP = UAW + UUCW
= 18 + 145
= **163**

8.3. Technical Complexity Factor (TCF) — Nonfunctional Requirements

TECHNICAL FACTOR	DESCRIPTION	PERCEIVED COMPLEXITY	WEIGHT	CALCULATED FACTOR
T1	Distributed system (running on multiple machines)	3	2	$2 \times 3 = 6$
T2	Performance objectives (are response time and throughput performance critical?)	3	1	$1 \times 3 = 3$
T3	End-user efficiency	3	1	$1 \times 3 = 3$
T4	Complex internal processing	3	1	$1 \times 3 = 3$
T5	Reusable design or code	0	1	$1 \times 0 = 0$
T6	Easy to install (are automated conversion and installation included in the system?)	3	0.5	$0.5 \times 3 = 1.5$
T7	Easy to use (including operations such as backup, startup, and recovery)	5	0.5	$0.5 \times 5 = 2.5$
T8	Portable	2	2	$2 \times 2 = 4$
T9	Easy to change (to add new features or modify existing ones)	1	1	$1 \times 1 = 1$
T10	Concurrent use (by multiple users)	4	1	$1 \times 4 = 4$
T11	Special security features	5	1	$1 \times 5 = 5$
T12	Provides direct access for third parties (the system will be used from multiple sites in different organizations)	1	1	$1 \times 1 = 1$
T13	Special user training facilities are required	0	1	$1 \times 0 = 0$

TFC $= C1 + C2 * TFT$
$= 0.6 + 0.01 \times 34$
$= \mathbf{0.94}$

8.4. Environment Complexity Factor (ECF)

ENVIRONMENTAL FACTOR	DESCRIPTION	PERCEIVED IMPACT	WEIGHT	CALCULATED FACTOR
E1	Beginner familiarity with the UML- based development	3	1.5	1.5 × 3 = 4.5
E2	Some familiarity with application problem	2	0.5	0.5 × 2 = 1
E3	Some knowledge of object-oriented approach	2	1	1 × 2 = 2
E4	Beginner lead analyst	1	1.5	1.5 × 3 = 4.5
E5	Highly motivated, but some team members occasionally slacking	4	1	1 × 4 = 4
E6	Stable requirements expected	5	2	2 × 5 = 10
E7	No part-time staff will be involved	0	-1	-1 × 0 = 0
E8	Programming language of average difficulty will be used	3	-1	-1 × 3 = -3

ECF = C1+C2 × EFT
= 1.4 + -0.03 × 19
= **0.83**

UCP = UUCP × TCF × ECF
=163 × 0.94 × 0.83
=127.17 or **128** UseCasePoints

Duration =UCP × PF
=128*28
= **3584 hours**

The software system is estimated to take about 3,584 hours to complete.

9. Domain Analysis

9.1. Concept Definitions

The concepts and their definitions are discussed below.

Website:
Definition: A hypertext document connected to the World Wide Web.
Responsibilities:
· Display HTML document that shows the actor the current context(K)
. Shows what actions can be taken through buttons(K)

Credentials:
Definition: Specific user's username and associated password.
Responsibilities:
· hold a user's username and password (K)

Query:
Definition: search query.
Responsibilities:
· hold a specific search query (K)

Timekeeper:
Definition: keeps track of internal time of the system.
Responsibilities:
· Knows when to update current stock prices(K)
. Knows when to predict future stock prices(K)

Controller:
Definition: Directs or regulates the requests made from user or another concept.
Responsibilities:
· Access account creation (D)
· Retrieves information from Data Renderer and passes to Website (K)
· Coordinate decisions based on the specific use case (D)

PageMaker:
Definition: Generates display inputs ultimately for website
Responsibilities:
· Must be able to display text, numbers and graphics for website environment(D)
Motivation: Data and images simply cannot come to website in a quick and easy fashion. There must be a transformation or parsing of "raw" local data that can be manipulated to fit the

website environment. The reason for doing this dates to the fact that a web-site is necessary to accommodate a large spectrum of users.

Account:
Definition: holds account information for a specific user.
Responsibilities:
- holds account information for a specific user(K)

Tracker:
Definition: tracks stocks based on user's watchlist.
Responsibilities:
- holds tracking information for specific users(K)

Predictor:
Definition: Generate stock predictions.
Responsibilities:
- Apply prediction algorithms to data(D)

StockRetriever:
Definition: Collects data from Internet Fetch stock prices and re by querying the PriceProvider.
Responsibilities:
- Retains momentary stock data from external websites and passes to Data Handler(D)

StockExtractor:
Definition: Extracts stock data to be stored within the database.
Responsibilities:
- Extracts stock data from a given file and stores it within the database(D)

DB:Connection:
Definition: An organized collection of stock data, user data, and system data.
Responsibilities:
- Store timers (K)
- Store invalid searches (K)
- Store user data (K)
- Store stock data (K)

Notifier:
Definition: Handles Administrator communication with the system for a diverse range of messages
Responsibilities:
- Notify of user's comments and questions(D)
. Notify of any system problems or errors(D)

AccountHandler:
Definition: A way to prove to a computer system that you really are who you are.
Responsibilities:
- Determine the validity of a logon request (K)
- Provide help or tips to a user attempting a logon (D)
- Terminate malicious threats (D)

StockHistoricalPriceDoc:
Definition: Holds historical prices for a given stock.
Responsibilities:
- Holds historical prices for a given stock (K)

StockCurrentPriceDoc:
Definition: Holds current price for a given stock.
Responsibilities:
- Holds current price for a given stock (K)

StockInformationDoc:
Definition: Holds current stock information (Ticker symbol, company name, etc.) for a given stock.
Responsibilities:
- Holds current stock information for a given stock (K)

StockInfo:
Definition: Holds current stock information.
Responsibilities:
- Holds current stock information(K)

Chart:Connection
Definition: Retains a connection to the grapher.
Responsibilities:
- Retains a connection to the grapher(K)
- Graph a given set of data(D)

Searcher
Definition: Queries database for stocks.
Responsibilities:
- Get stocks based on a keyword(D)
- Get stocks based on predicted gain(D)

9.2. Traceability Matrix

	USE CASES														
DOMAIN CONCEPTS	Login	AddStock	EditPredictionTimer	EditUpdateTimer	RemoveStock	Search	Suggest	Track	RemoveTrackedStock	Register	Logout	Update	PredictAndNotify	Learn	Support
Controller	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Query						x	x								
Credentials	x														
Website	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
PageMaker	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Account	x							x	x		x				
DB: Connection	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Searcher						x	x								
Logger			x	x											
Account Handler	x									x	x				
Securer	x									x					
Tracker								x	x						
Graph: Connection						x	x								
Time Keeper												x	x		
Stock Retriever		x													
Stock Extractor		x													
Current Information Doc		x													
Historical Prices Doc		x													
Notifier								x					x		
Predictor													x		
Stock Info						x	x								

9.3. Associations

CONCEPT PAIR	ASSOCAITION DESCRIPTION	ASSOCIATION NAME
Controller<->AccountHandler	Controller obtains key information from AccountHandler Motivation: Controller needs to verify user before he is given certain privileges	Obtain
Data Handler <-> Stock Database	Data Handler retrieves valid stock information	Conveys requests
AccountDatabase <-> Account	Account database stores user Account data	Conveys requests
Data Handler <->System Database	Data Handler retrieves valid system information	Conveys requests
Watcher<-> Authenticator	Watcher will request information to Authenticator of which user's to watch Motivation: Watcher needs to know which user it should pay attention to	Requests notify
Website<->Controller	Website passes user's inputs to Controller Motivation: Data inputs need to be processed by other concepts otherwise there is no interaction	Pass Information
Renderer<->Display Page	Motivation: Display context is crucial to Website otherwise user cannot receive important feedbac	Prepare
Controller<->Addresse	Motivation: User administrator also needs feedback	Provides dat
AccountHandler<->DB:Connection	Motivation: AccountHandler needs data from database therefore DB:Connection is contacted.	Conveys request
Creator<->Data Handle	Motivation: New user information must be placed in Database therefore memory must be allocate	Requests memor
Controller<->Watche	Motivation: Watcher must know what information current user has inputted however it must first receive validation from gatekeeper	Passe
Watcher<->Gatekeepe	Motivation: Not all of user inputs may be valid therefore gatekeeper keeps validates user inputs sends information to watcher	Provides dat
Gatekeeper<->DataHandle	Motivation: System information needs to be updated if desire	Provides dat

CONCEPT PAIR	ASSOCAITION DESCRIPTION	ASSOCIATION NAME
Gatekeeper<->Fishe	Motivation: Information must be queried at certain desired times	Notifier
Fisher<->Data Handle	Motivation: Data must be updated at certain times for new analysi	Requests & update
FortuneTeller<->Data Handle	Motivation: Analysis is required of data otherwise there is no user feedbac	requests & obtain
Controller<->Rendere	Motivation: Controller provides new data that need to be processed for website upload	Provide

9.4.Attributes

StockInfo holds attributes related to specific stock data such as ticker symbol, company name, current price, and etc.

Query holds attributes related to a specific search query such as industry, sector and keyword. Unfortunately, industry and sector could not be implemented and thus the only remaining attribute is keyword.

StockRetriever has attributes holding the URLs required to retrieve stock information from the price provider.

DB:Connection holds attributes to connect to the local database including username, password, and database name.

Timekeeper holds attributes of the times when to update current stock prices as well as the time when to predict future stock prices.

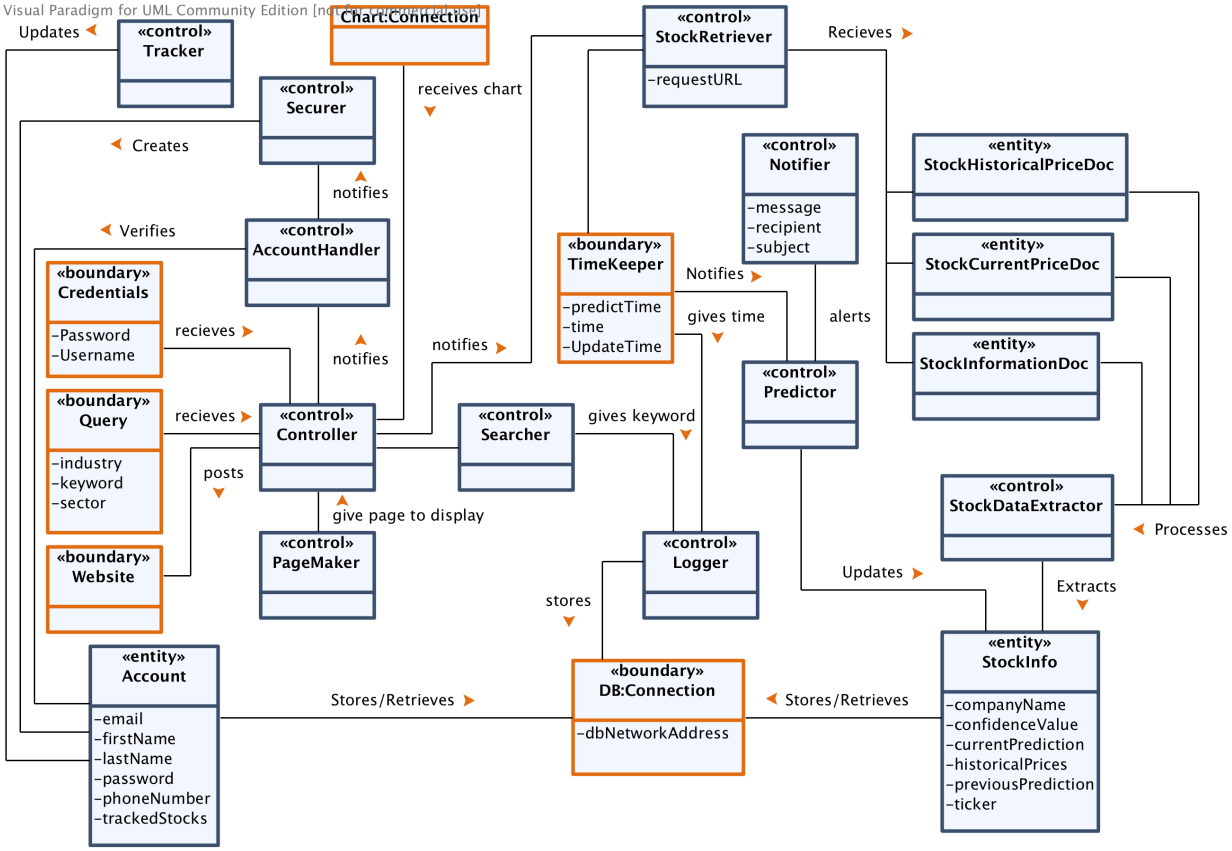


Figure 17: Domain model diagram.

9.5. System Operation Contracts:

1) Name: **AddStock**
Responsibilities: Allows user to add a stock to database that doesn't already exist and then stock predictions for this specific stock will be notified to the user's account.

Cross References: User Cases(1,2,10,13). Login, Search, Register, Predict and Notify

Expectations: If user adds stock then user will get predictions about stock, user must initially have an account for them to add a stock to view predictions.

Preconditions: Must be connected to the internet, must have a profile.

Postconditions: Stock predictions are displayed and notified to the specific user account.

2) Name: **Search**

Responsibilities: Takes users inputted stock name, user name or other means of filtering to a specific stock and match it to the database and retrieve the data and predictions for that stock.

Cross references: User cases: (1,2,10)Login, AddStock, Register

Expectations: Stock name must be found in the database or added to database if it doesnt exist.

Preconditions: Access to internet is required to search for stocks online, User must also have an account to have access to search for a stock.

Postconditions: A stock that the user searches for is found and its previous data as well as future predictions are given to the user.

3) Name: **Track**
Responsibilities: Add a specific stock to a "wish list" so that the stock information is provided on the user's profile for easy access.

Cross references: User cases: (1,2,6,10)Login, AddStock, Search, Register

Expectations: For access to a wish list, the user must first have an account..

Preconditions: The stock must first be existent in the database, if not then user must add stock to the database.

Postconditions: The stock that the user wants to track will be displayed on his wish list and with easy access to the stock information such as data and predictions.

4) Name: **Update**
Responsibilities: Allows timer to initiate the loading of current stock prices from the price-provider into the database.

Cross references: User cases: (3,4)EditPredictionTimer, EditUpdateTimer

Expectations: The stock data from the price-provider must be of easy access so that the website is updated in a timely manner so that the users can conveniently view and edit their stock information.

Preconditions: There needs to be a link from the price-provider to the website. Also algorithms that are used to predict the stock changes in the future must also be updated constantly so that accurate information is given to the users.

Postconditions: When a user goes to website to view their stocks data or information about the future predictions, the information will be accurate and will be displayed with the data refreshing and updating.

5) Name: **Predict and Notify**
Responsibilities: Allows timer to initiate the prediction of future stock prices
to be later stored into the database. Registered Users are then notified if an change in prediction is made for a stock he/she has chosen to track.

Cross references: User cases: (8,12)Track, Update

Expectations:Predictions for stocks are made in a certain time period, and everytime the new stock predictions are made through algorithms and other ways, then it is displayed to user. So user must have internet access, for constant updates as well as an account for the information to be displayed into.

Preconditions: The algorithms and link between the price-provider and websites database must be accurate. The users also must have selected which stocks they want to get predictions from by using the wish list.

Postconditions: The predictions are made using the required material and then displayed to the user. So the stocks that the user selects in the wish list will be updated and provided for the user to view.

9.6. Mathematical Models

ARIMA time series analysis - auto regressive integrated moving average. The basic principle of this is that with each collected data point the average changes. The average is what predicts the future.

Exponential Moving Average (EMA) : More weight is given to recent values though not all older data sets are not discarded.

Rate of Change (ROC) : Current prices are ratioed with the price n days aways. n is usually 5 to 10 days.

Relative Strength Index (RSI): Check the magnitude of the upward trends against the downward trends within a specific time interval (usually 9 – 14 days).

10. Interaction Diagrams

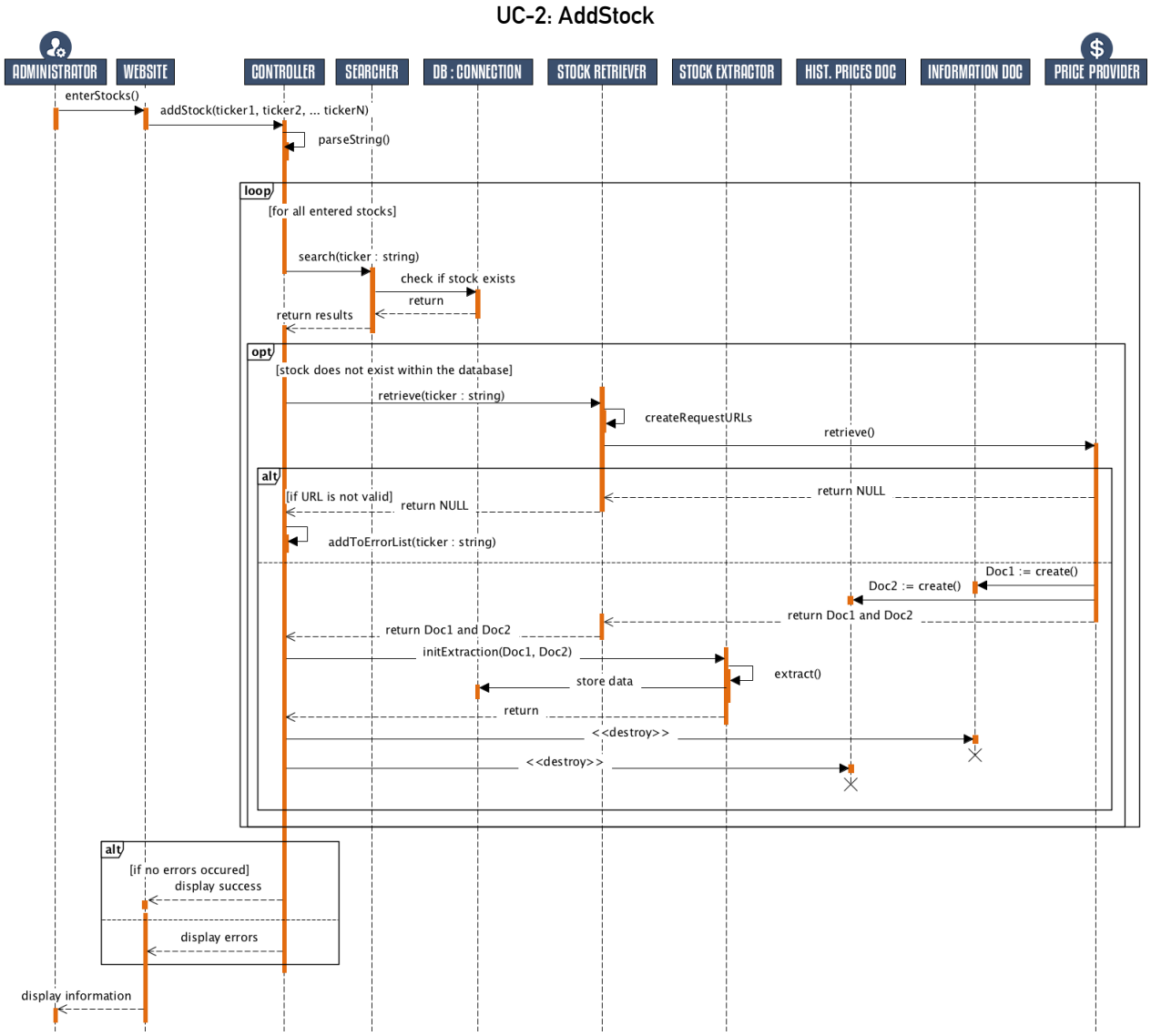


Figure-18: Design sequence diagram for UC-2: AddStock

Figure-18 shows the design sequence diagram for Use Case 2: AddStock. Once the administrator enters a sequence of comma separated ticker symbols the controller parses that string into an array of ticker symbols. A main loop is then entered for each ticker. To prevent duplicates, the searcher verifies that the stock does not already exist within the database. Once the stocks uniqueness is verified, two request URLs are created by the Stock Retriever to retrieve the necessary stock data. If the administrator made a typo or for some reason the URL is invalid, the failed stock is added to an error list and the loop continues onto the next stored ticker. Otherwise, the price provider will create and return two documents: a file containing historical prices and a file containing current stock information such as industry, sector, and

current prices. The relevant data is then extracted from the documents and stored within the database by the Stock Extractor. Once the loop has completed, the administrator is notified of errors that occurred if any.

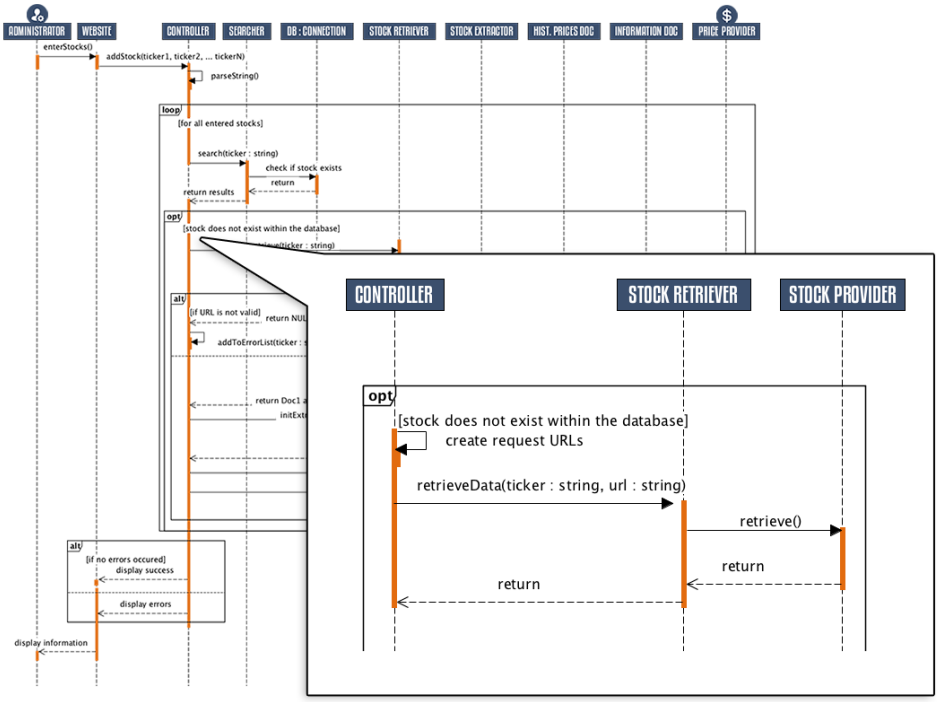


Figure-19: Alternate design sequence diagram for UC-2: AddStock

The question of who should create the request URLs can lead to many solutions. The expert doer principle suggest that this should be the controller since it first gets the knowledge of whether a stock already exists within the database or not (and thus whether a request URL must be created for it). Figure-19 demonstrates this alternate design where the controller first constructs the URLs and passes them to the stock retriever. It should be noted that to simplify this diagram, unrelated intermediate function calls have been removed. The high cohesion principle, on the other hand, would dictate that the stock retriever should be responsible for creating the request URL's. Since the controller's sole responsibility is to manage communication between concepts and nothing more, the initial design in Figure-18 is chosen.

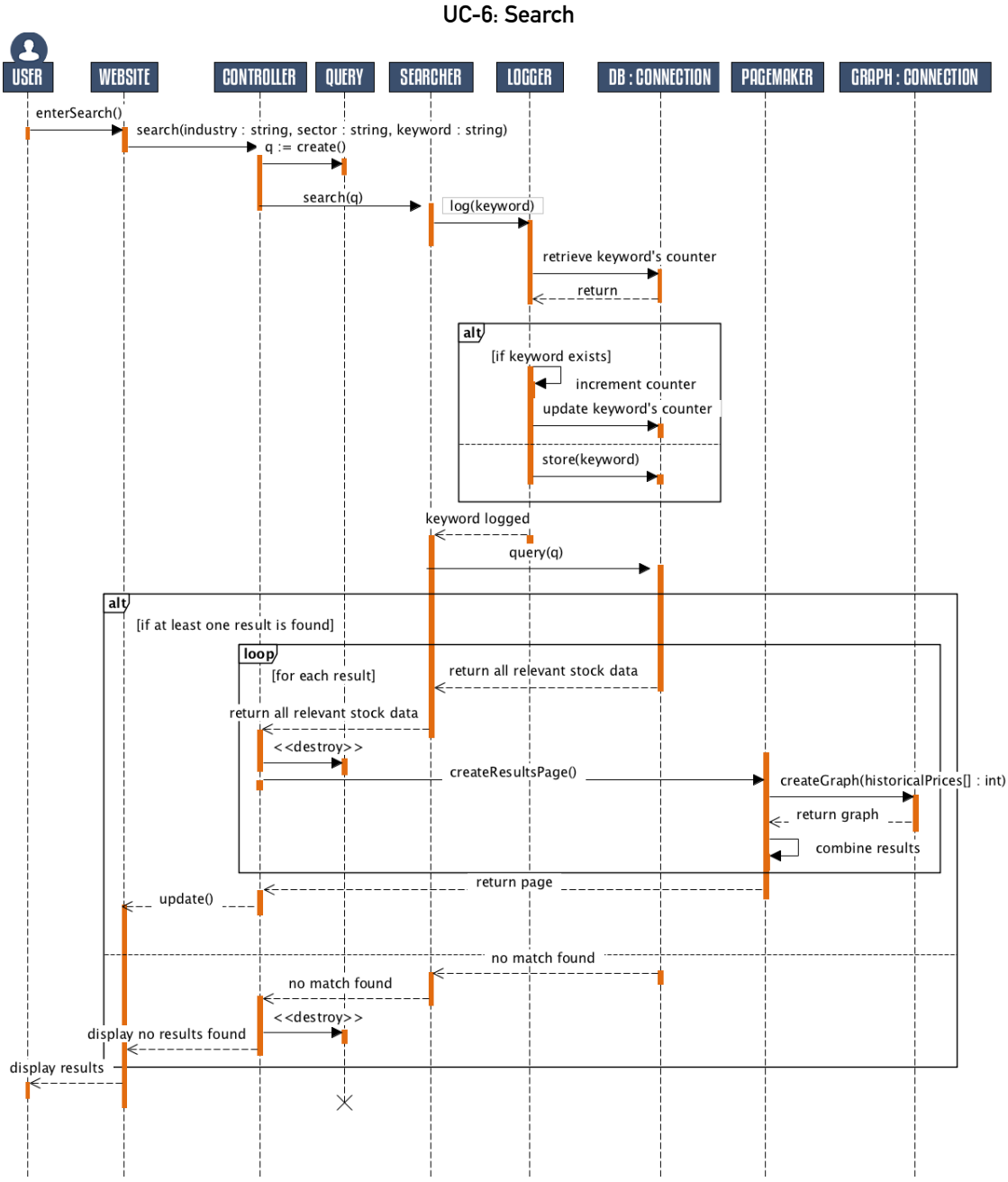


Figure-20: Design sequence diagram for UC-6: Search

Figure-20, shows the design sequence diagram for Use Case 6: Search. The controller creates a query using a user's search parameters which includes a keyword, industry, and sector. If a keyword is entered it is logged by the Logger for analytics and then the database is queried by the Searcher using the previously constructed query. For every result, a chart of historical prices is created. The pagemaker combines all relevant information for every search result into a page which is returned to the controller. The controller will then update the website. If no matches are found, the website is simply updated to show this.

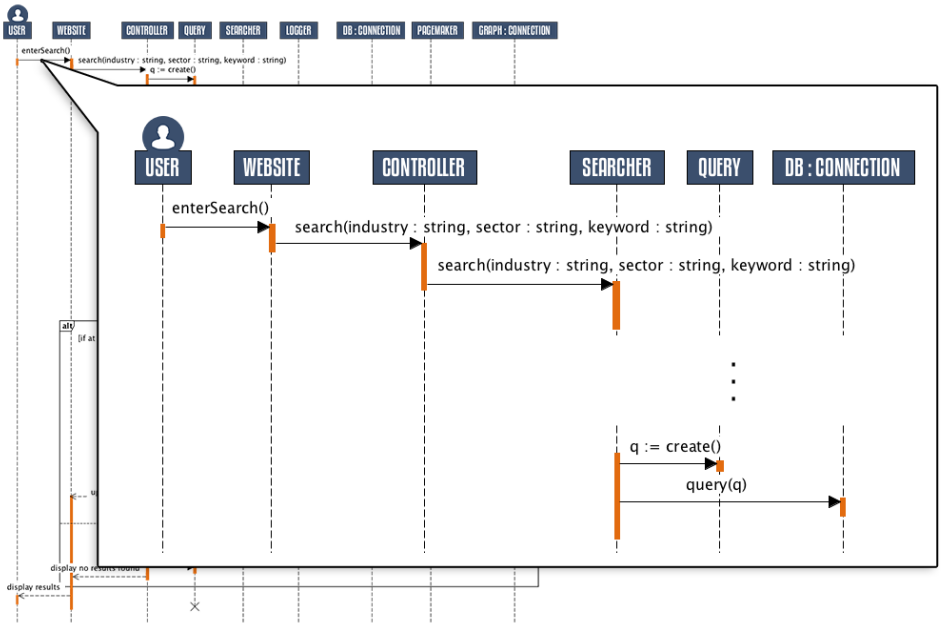


Figure-21: Alternate design for UC-6: Search

Figure-21 displays an alternate design to this sequence where the controller, instead of creating the query itself, passes on the relevant information to the searcher which would then be responsible for creating the query. It should be noted that unrelated intermediate function calls have been removed to further simplify this diagram. This design, again follows the high cohesion principle. The controller's responsibilities are limited to that of just coordinating tasks between concepts and nothing more. Furthermore, this method allows for the query to be created right before its use unlike in Figure-20 where the keyword must first be logged before the query is even utilized. Thus, the data is less prone to corruption. Therefore the method in Figure-21 will be followed.

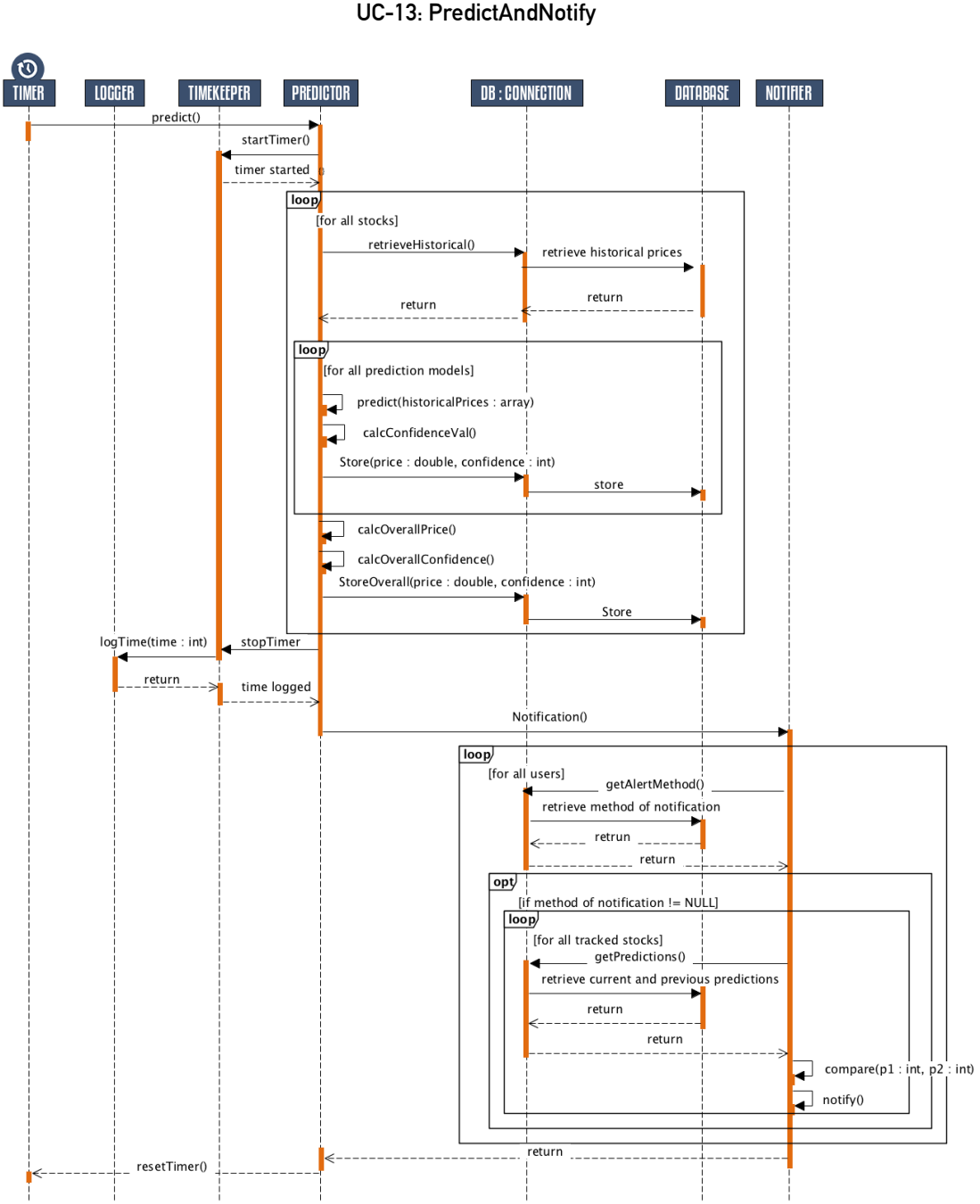


Figure-22: Design sequence diagram for UC-13: PredictAndNotify

Figure-22 shows the design sequence diagram for Use Case 13: PredictAndNotify. A Cronjob will initiate the predictor to start an internal timer within TimeKeeper. The predictor then retrieves historical prices for each stock from the database. For each prediction model, a prediction is made based on those historical prices and a confidence value is calculated, these are then stored into the database. An overall prediction is then made and stored within the database along with its confidence value. The timer is then stopped and the time sent to the logger to be logged. The predictor then calls the Notifier. For each user and for each tracked

stock for that user, a comparison is then made between the current prediction and the previous prediction. If a change is detected, the notifier sends out a notification through the users preferred method of notification.

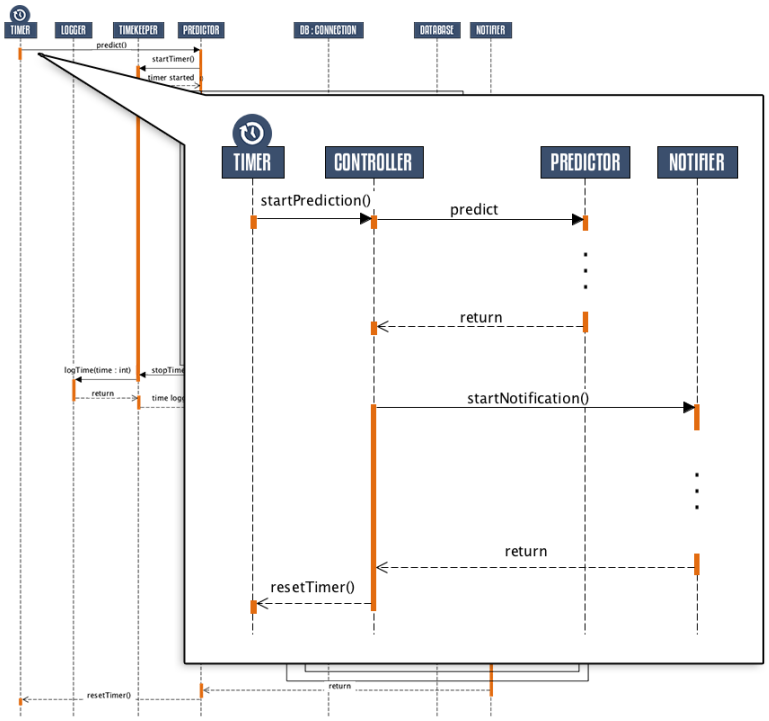


Figure-23: Alternate design sequence diagram for UC-13: PredictAndNotify

In an alternate design, as shown in Figure-23, the controller can be used as a buffer between the initiating actor and the predictor. Although this design appears more eloquent, where the controller serves as the sole communicator (thus not requiring that the predictor call the notifier as before) it would also transform the controller into a boundary concept and therefore leave the entire system to be prone to security faults. Since the controller serves as a way of coordination between all concepts, it should be paramount that no initiating actors have direct control over it. Therefore, this design will not be taken into consideration.

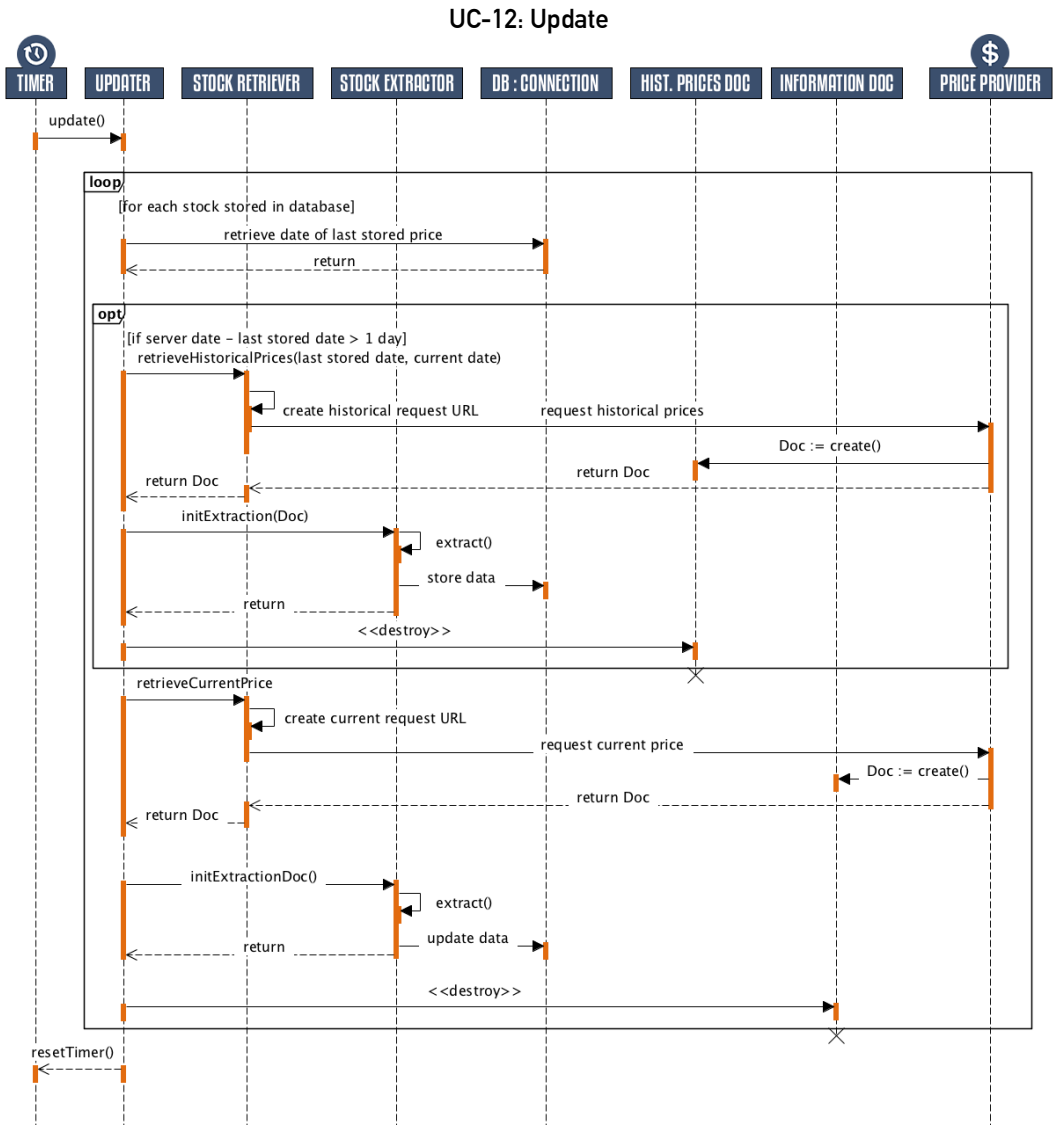


Figure-24: Design sequence diagram for UC-12: Update

Figure-24 shows the design sequence diagram for Use Case 12: Update. A main loop is first entered for each stock being stored within the database. The Updater compares the date of the last stored price for a stock with the server data and if the two differ, historical prices for the missing days are retrieved by the Stock Retriever and then extracted and stored by the Stock Extractor. The current price for each stock, in a similar fashion, is then retrieved by the Stock Retriever and then extracted and stored by the Stock Extractor.

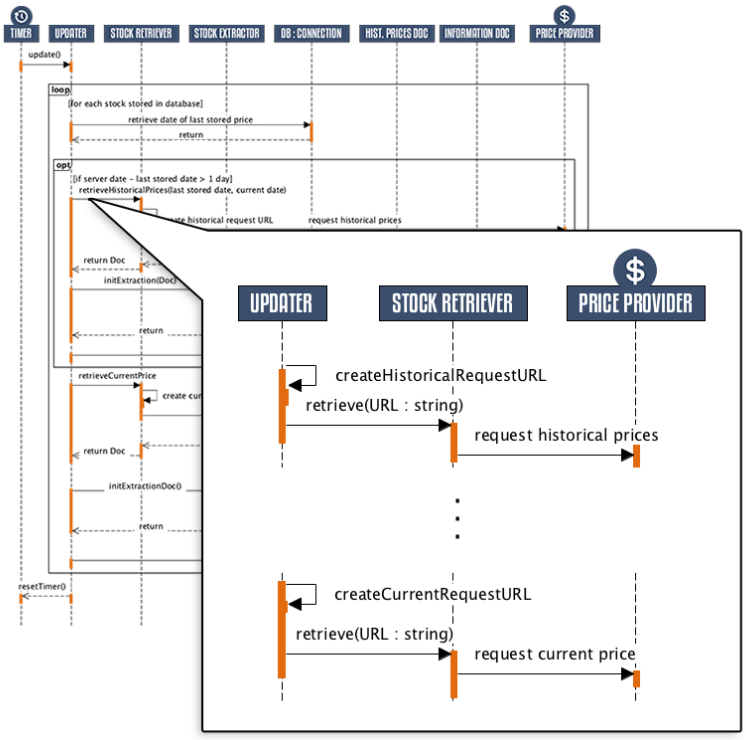


Figure-25: Alternate design sequence diagram for UC-12: Update

An alternate design is shown in Figure-25 where the request URL creation is handled by the Updater and then passed onto the Stock Retriever. This method follows the expert doer principle since the Updater first receives the information necessary to create a request URL. The high cohesion principle, on the other hand, would suggest that URL creation should remain a responsibility of the Stock Retriever. If you recall, we have already chosen that the responsibility of creating URLs remain a process within the Stock Retriever in the design of UC-2: AddStock, and thus for consistency will also choose the same here.

UC-8: Track

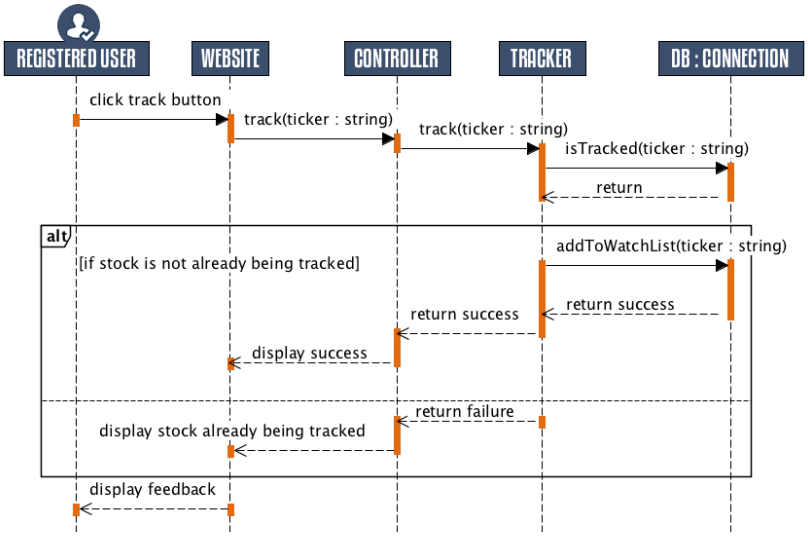


Figure-26: Design Sequence Diagram for UC-8: Track

Figure-26 shows the design sequence diagram for Use Case 8: Track. Once a user clicks the "track" button, the website notifies the controller of the event. The controller passes the associated stock to the Tracker which then will verify if the stock already exists within the user's "watch list". If not, the stock is added and the user is displayed the appropriate feedback through the Website.

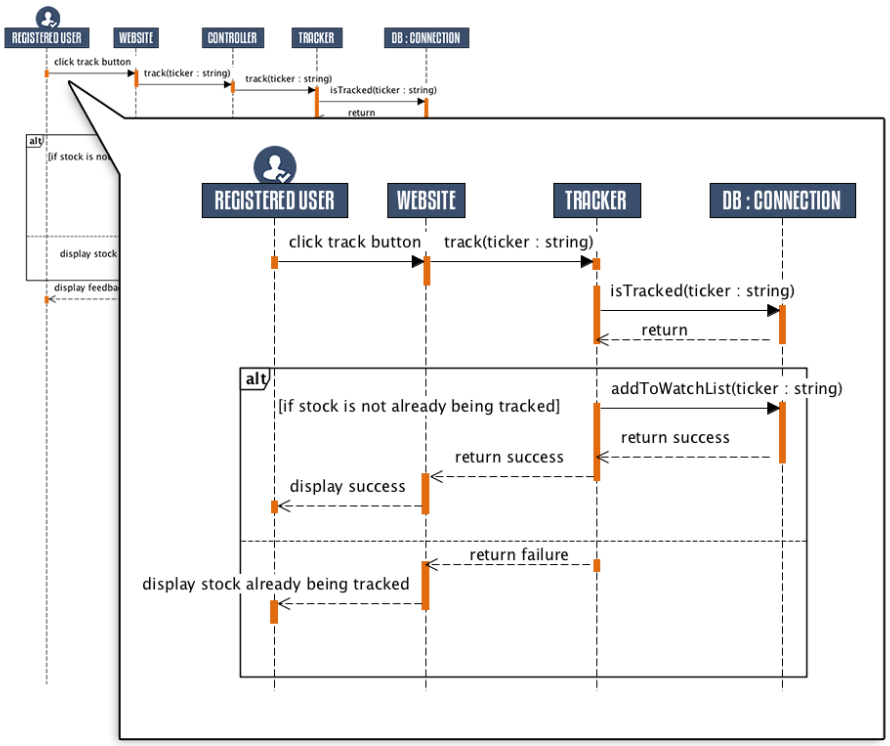


Figure-27: Alternate design sequence diagram for UC-8: Track

It is questionable whether the controller is even needed in this sequence. Figure-27 shows just that case. The controller is removed, and the website communicates directly with the tracker. This method is favored by the low coupling principle and the diagram seems to be greatly simplified. However, as stated before, the controller should be responsible for all communication between concepts (except for special cases) and thus for consistency as well as high cohesion we will refrain from using this design.

11. Class Diagram and Interface Specification

11.1. Class Diagram

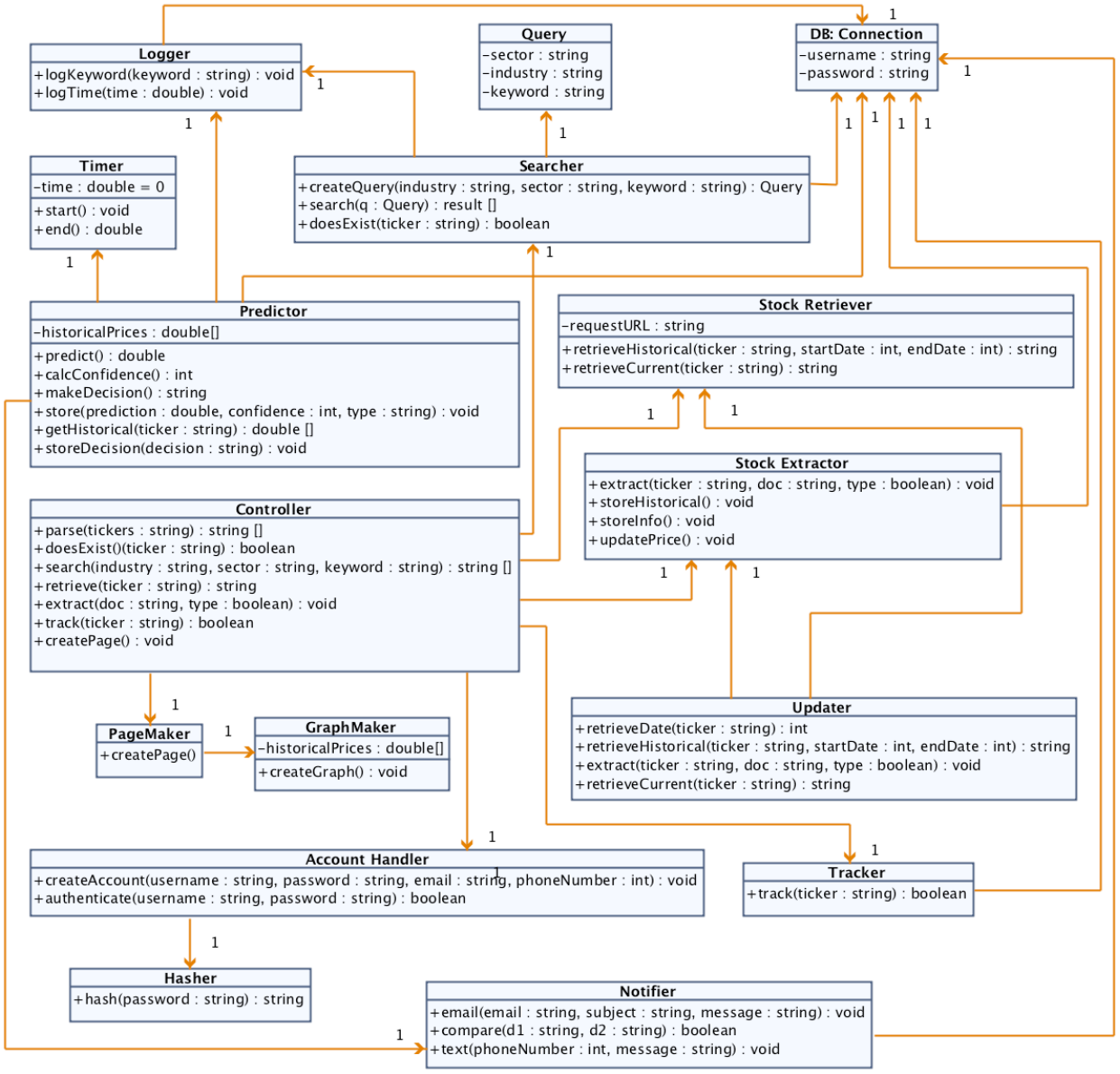


Figure-28: Class diagram for the web based stock forecasting system.

Figure-28 shows the derived class diagram including all relevant attributes and operations for our system.

11.2. Data Types and Operation Signatures

Controller

Coordinates the communication and transfer of data between classes.

+parse(tickers : string) : string []

Parses the input string of comma separated ticker symbols by converting it into an array of ticker symbols.

+doesExist()(ticker : string) : boolean

Calls the searcher to verify if a ticker exists within the database.

+search(industry : string, sector : string, keyword : string) : string []

Calls the searcher to search for a stocks matching the given industry, sector, and keyword.

+retrieve()(ticker : string) : string

Calls the Stock Retriever to retrieve necessary stock data and information and returns the respective CSV files.

+extract(doc : string, type : boolean) : void

Calls the Stock Extractor to extract to begin extracting the necessary data from doc (CSV file). “Type” alerts the method of whether the file contains historical prices or current stock information.

+track(ticker : string) : boolean

Calls the tracker to add the given stock to a user’s “watch list”. Returns true, if successful and false otherwise.

+createPage() : void

Calls the Page Maker to create the results page after a search is initiated.

Predictor

Makes predictions based on a variety of models and calculates confidence values for each prediction. Also calculates an overall predicted price, confidence value, and trade decision based on each individual model.

+predict() : double

Calculates and returns predicted stock price.

+calcConfidence() : int

Calculates and returns confidence value associated with predictions.

+makeDecision() : string
Returns a trading decision based on prediction results from all models.

+store(prediction : double, confidence : int, type : string) : void
Stores predicted price and confidence value into the database. “Type” alerts the method of which prediction model was used and thus which table the information should be saved under.

+getHistorical(ticker : string) : double []
Returns an array of historical prices for a given stock.

+storeDecision(decision : string) : void
Stores the trading decision associated with the overall prediction into the database.

Timer
Keeps track of the elapsed time for a given prediction session.

-time : double = 0
The running time since the timer was started. Initial value is set to 0.

+start() : void
Starts the timer.

+end() : double
Ends the timer and returns the elapsed time.

Logger
Logs keywords from user searches and times calculated by the timer.

+logKeyword(keyword : string) : void
Logs the keyword in a given user search into the database. If the keyword already exists, the counter associated with the keyword is incremented and then updated within the database.

+logTime(time : double) : void
Logs the elapsed time for a given prediction session into the database.

Query
Used to query the database.

-sector : string
User’s selected sector.

-industry : string
User's selected industry.

-keyword : string
User's entered search keyword.

Searcher
Queries the database based on a users unique search.

+createQuery(industry : string, sector : string, keyword : string) : Query
Creates a query object with the given sector, industry, and keyword.

+search(q : Query) : result []
Queries the database with the given query and returns the results.

+doesExist(ticker : string) : boolean
Returns true if the given stock exists within the database and false otherwise.

DB: Connection
Establishes a connection to the database.

-username : string
Username required to access the database.

-password : string
Password required to access the database.

Stock Retriever
Retrieves specific stock information from the Price Provider.

-requestURL : string
The initial part of the URL necessary to request a CSV file of stock information from the Price Provider.

+retrieveHistorical(ticker : string, startDate : int, endDate : int) : string
Retrieves and returns a CSV file of historical prices for the given stock within the given time interval.

+retrieveCurrent(ticker : string) : string
Retrieves and returns a CSV file of current stock information for the given stock, including industry, sector, current price, etc.

Stock Extractor
Extracts and stores relevant data from the CSV files returned from the Stock Retriever

+extract(ticker : string, doc : string, type : boolean) : void
    Extracts and stores the necessary information from the input CSV file. “Type” alerts the method of whether the file contains historical prices or current stock information.

+storeHistorical() : void
    Stores the extracted historical prices into the database.

+storeInfo() : void
    Stores the extracted current stock information into the database.

+updatePrice() : void
    Updates the current price of a stock within the database.

Updater
Makes sure that prices stored within the database are kept up to date.

+retrieveDate(ticker : string) : int
    Retrieves and returns the date of the last stored price for a given stock.

+retrieveHistorical(ticker : string, startDate : int, endDate : int) : string
    Calls the Stock Retriever to retrieve and return a CSV file of historical prices for the given stock within the given time interval.

+retrieveCurrent(ticker : string) : string
    Calls the Stock Retriever to retrieve and return a CSV file of current stock information for the given stock, including industry, sector, current price, etc.

+extract(ticker : string, doc : string, type : boolean) : void
    Calls the extractor to extract the relevant information from the given input file. “Type” specifies whether the file contains historical prices or current stock information.

PageMaker
Creates and organizes a page of results after a user initiates a search.

+createPage()
    Creates the results page after a search is initiated.

GraphMaker
Retrieves a graph of historical prices from the Grapher.

-historicalPrices : double[]
An array of historical prices for a given stock.

+createGraph() : void
Graphs the information within the array of historical prices.

AccountHandler
Handles user accounts, specifically the creation of new accounts and the authentication of existing accounts.

+createAccount(username : string, password : string, email : string, phoneNumber : int) : void
Creates an account with the given username, password, email, and phone number.

+authenticate(username : string, password : string) : boolean
Checks if a user is a registered user stored within the system. Returns true if so, and false otherwise.

Hasher
Hashes passwords when creating new accounts to ensure security.

+hash(password : string) : string
Hashes a given password.

Tracker
Handles the addition of new stocks into a user's "watch list".

+track(ticker : string) : boolean
Verifies whether a stock is stored within a user's "watch list" and if not, adds the given stock.

Notifier
Handles the notification of users when a trade decision is predicted

+email(email : string, subject : string, message : string) : void
Sends an email to the given email address with the given subject and message.

+text(phoneNumber : int, message : string) : void
Sends a text message to the given phone number with the given subject.

+compare(d1 : string, d2 : string) : boolean
Compares two trade decisions and returns true if they are the same and false otherwise.

11.3.Traceability Matrix

DOMAIN CONCEPTS	SOFTWARE CLASSES: Controller	Query	Predictor	Searcher	Timer	Logger	Stock Retriever	Stock Extractor	Updater	Grapher Maker	Account Handler	Hasher	Tracker	DB: Connection	Page Maker	Notifier
Controller	x															
Query		x														
Credentials											x					
Website																
PageMaker															x	
Account																
DB: Connection														x		
Searcher				x												
Logger						x										
Account Handler											x					
Securer												x				
Tracker													x			
Graph: Connection										x						
Time Keeper					x											
Stock Retriever							x									
Stock Extractor								x								
Current Information Doc																
Historical Prices Doc																
Notifier																x
Predictor			x													
Stock Info																

Figure-29: Domain model to software classes traceability matrix

Figure-29 shows how the software classes map to the domain model. As the matrix shows, a few concepts could not be mapped to. The website is simply the html page a user views on his/her browser. Account and StockInfo are concepts that are storing specific data which will already be stored within the relational database. The concepts of Current Information Doc and Historical Prices Doc will be provided to the system by the Price Provider as CSV files. All of these concepts do not require to be mapped to a specific class since they are all of the "knowing" type and thus perform no important tasks.

12. System Architecture and System Design

12.1. Architectural Styles

Our software uses several architectural styles. They follow:
1) Client/Server
2) Event-driven
3) Rule-based system
4) Database-centric

Client/Server Architecture

Client/Server is our main architectural style; it separates our system requirements into two easily programmable systems. First, the client, which acts as the User Interface, requests data from the server, and waits for the server's response. Secondly, the server, which authorizes users and processes stock data into information the user can use. It then sends this processed information to the client to display to the user.

Event-driven Architecture

Our system will only need to execute its functions after some major state change. It has no real time components like a video game. Instead, we'll have two event emitters, the user and the timer. Both will drive the application to execute relevant operations though the execution of different events. These events include login, adding new stocks, deleting stocks, and requesting an updated lists of stocks for the user; and a time-based update for the timer.

Rule-Based System

Our application will be rule-based. In other words, the system will use a set of rules that we determine it to analyze the stock information it gathers. These rules comprise a semantic reasoned which makes decisions for the application and the user. It uses a match cycle act cycle to deduct which stocks will be best to buy and which stocks would be best to sell. Then, it outputs these results to the user-interface.

Database-centric Architecture

Our system relies heavily on its database, both the store relevant stock data and to analyze the data we give it. The database-centric architecture offers:

1) A standard relational database management system. This means the data will be stored away from the client side application.
2) Dynamic table-driven logic. We need to update the tables every time stock prices change.
3) Stored procedures running on database servers to analyze our data.
4) A shared database for communication between parallel processes

In short, it's a good way of managing a large amount of data

12.2. Identifying Subsystems

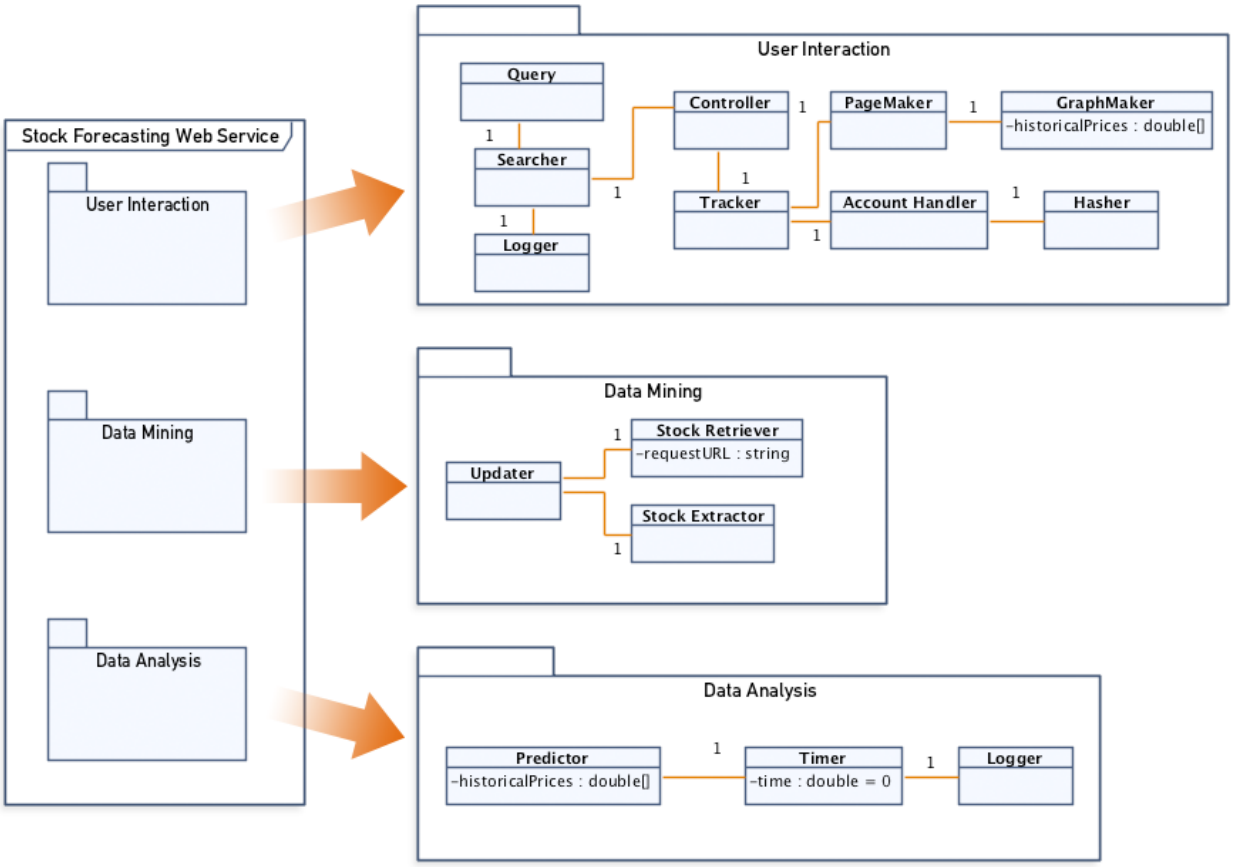


Figure-30: Package diagram showcasing the three subsystems within the main system.

Digging deeper into the system we observe three subsystems emerge as shown by Figure-30 as well as the classes relating to each subsystem. The user interaction subsystem deals with the web interface and basic input/outputs and encompasses any class that assists in such matters. The data mining subsystem deals with gathering relevant stock information at a fairly frequent intervals and storing that data locally, incorporating the classes of Updater, Stock Retriever, and Stock Extractor. The data analysis subsystem holds the responsibility of

implementing prediction models on the previously mined data and storing those predictions for later use.

12.3. Mapping Subsystems to Hardware

Our software employs the use of a client/server system. The client–server model of computing is a dispersed application structure that divides tasks between the provider of a resource or service, called the server, and an entity making a request, called the client. This establishment can be made with a network connection between host (the server) and client or it is possible for this relationship to exist on the same system, sharing hardware.

The web-based stock forecasting design we presume to execute will need to be accessible anywhere that web access exists. This means the client in our client/server relationship will be a web client. A web browser is an example of a web client, and can remotely access requested documentation from the server via HTTP. This web client/server model will need to be run across multiple computers, or subsystems.

A web browser will be used to request the various data from our server, as well as retrieve stock information and updates that can be sent and retrieved via communication with that server. The GUI, which will run on the web browser, will be executed client side while the process itself is handled by the server's web service. The web service will ensure for proper transmission of user data between the client and the server.

12.4. Persistant Data Storage

The system-to-be requires data to be saved in order to outlive a single execution of the system. This data includes historical stock prices for all relevant stocks, related information for all relevant stocks, user account information, user search queries, and the timers dictating when the system should update current stock prices as well as begin calculating predictions. In order to organize this data, one must first elaborate on the different methods of storage.

A flat file database typically consists of multiple text files storing one record per line. Text files are simple and portable and can, in most cases, be used without requiring special software. They make it, however, difficult to search for specific information or to create reports that include only certain fields from each record. Thus, when creating new records, numerous redundancies occur as a portion of information in one file must be rewritten to all others.

A relational database, on the other hand, consists of multiple tables linked by "keys" — certain pieces of information shared by two or more tables. This model takes advantage of the uniformity to build completely new tables out of required information from existing tables. In other words, it uses the relationship of similar data to increase the speed and versatility of the database.

The implementation of the timers previously stated will be through a Cronjob and thus a flat file is ultimately the best case for storing them since they require a simple line of code each such as:

```
0 */2 * * *  /home/username/test.php
```

Which makes the user script test.php run every two hours, at midnight, 2am, 4am, 6am, 8am, and so on.

For all other information the system-to-be relies heavily on large amounts of data storage, organization, analysis, and security. Thus, a relational database will be used for its efficiency and power. The tables are organized as follows:

STOCKS					
Field	Type	Null	Key	Default	Extra
StockID	int(11)	NO	PRI	NULL	auto_increment
Company	varchar(20)	NO		NULL	
Ticker	varchar(5)	NO		NULL	
Industry	varchar(20)	NO		NULL	
Sector	varchar(20)	NO		NULL	

Table-1: Organization of the table *Stocks* within the database.

The *Stocks* table, as shown by Table-1, holds all stock related information including the company name, ticker symbol, industry, and sector. An auto incremented ID is used to insure that each entry is unique.

HISTORICAL PRICES					
Field	Type	Null	Key	Default	Extra
StockID	int(11)	NO	PRI	NULL	
Date	date	NO		NULL	
Open	decimal(10,0)	NO		NULL	
High	decimal(10,0)	NO		NULL	
Low	decimal(10,0)	NO		NULL	
Close	decimal(10,0)	NO		NULL	
Volume	int(11)	NO		NULL	

Table-2: Organization of the table *Historical Price* within the database.

The *Historical Prices* table, as shown by Table-2, holds all historical data for all stocks. This historical data includes the data, opening price, daily high price, daily low price, closing price,

and volume. Stocks are differentiated from each other using the stockID established in the *Stocks* table.

USERS					
Field	Type	Null	Key	Default	Extra
UserID	int(11)	NO	PRI	NULL	auto_increment
Email	varchar(20)	NO		NULL	
Password	varchar(20)	NO		NULL	
SecurityKey	varchar(30)	NO		NULL	
Phone	int(10)	NO		NULL	

Table-3: Organization of the table *Users* within the database.

The *Users* table, as shown by Table-3, holds all user account related information including userID, email address, password (hashed), security key provided by the hasher, and phone number. The userID field is an auto incrementing integer, ensuring that every stored user is unique.

OVERALL PREDICTION					
Field	Type	Null	Key	Default	Extra
StockID	int(11)	NO		NULL	
Date	date	NO		NULL	
PredictedPrice	int(11)	NO		NULL	
ConfidenceValue	int(11)	NO		NULL	
PredictedDecision	varchar(10)	NO		NULL	

Table-4: Organization of the *Overall Prediction* table within the database.

The Overall Prediction table, as shown by Table-4, holds the overall prediction results and related information. This includes the prediction date, predicted price, calculated overall confidence value, and predicted trade decision. The stockID from the *Stocks* table is used to differentiate between different stocks.

PREDICTION1					
Field	Type	Null	Key	Default	Extra
StockID	int(11)	NO		NULL	
Date	date	NO		NULL	
PredictedPrice	int(11)	NO		NULL	
ConfidenceValue	int(11)	NO		NULL	

.
.
.

PREDICTION8					
Field	Type	Null	Key	Default	Extra
StockID	int(11)	NO		NULL	
Date	date	NO		NULL	
PredictedPrice	int(11)	NO		NULL	
ConfidenceValue	int(11)	NO		NULL	

Table-5: Organization of the intermediate prediction tables within the database.

The system will utilize a total of eight intermediate prediction models and a table will exist to store predictions results and related data for each, ash shown by Table-5. This includes the current data, the predicted price, and the calculated confidence value. The stockID from the *Stocks* table is used to differentiate between stocks.

TRACKEDSTOCKS					
Field	Type	Null	Key	Default	Extra
UserID	int(11)	NO		NULL	
StockID	int(11)	NO		NULL	

Table-6: Organization of the *Tracked Stocks* table within the database.

A table will be used to keep track of user's "watch lists". The *Tracked Stocks* table, as shown by Table-6, will associate a userID from the *Users* table to a stockID in the *Stocks* table.

KEYWORDS					
Field	Type	Null	Key	Default	Extra
KeywordID	int(11)	NO	PRI	NULL	auto_increment
keyword	varchar(20)	NO		NULL	
counter	int(11)	NO		NULL	

Table-7: Organization of the *Keywords* table within the database.

The Keyword table, as shown by Table-7, will store all keywords used for user search queries. Each entry gets a unique identifier, the keyword, and counter specifying the number of times the keyword was queried.

TIMES					
Field	Type	Null	Key	Default	Extra

Date	date	NO	PRI	NULL	
time	double(10)	NO		NULL	

Table-7: Organization of the *Times* table within the database.

The Times table, as shown by Table-7, holds the amount of time a given prediction session takes. Each time is associated with the date the prediction session had taken place. This will serve useful for future analytics.

Figure-31 below shows the relationship between all the aforementioned tables. Tables *Keywords* and *Times* share no relation and thus were omitted from the diagram.

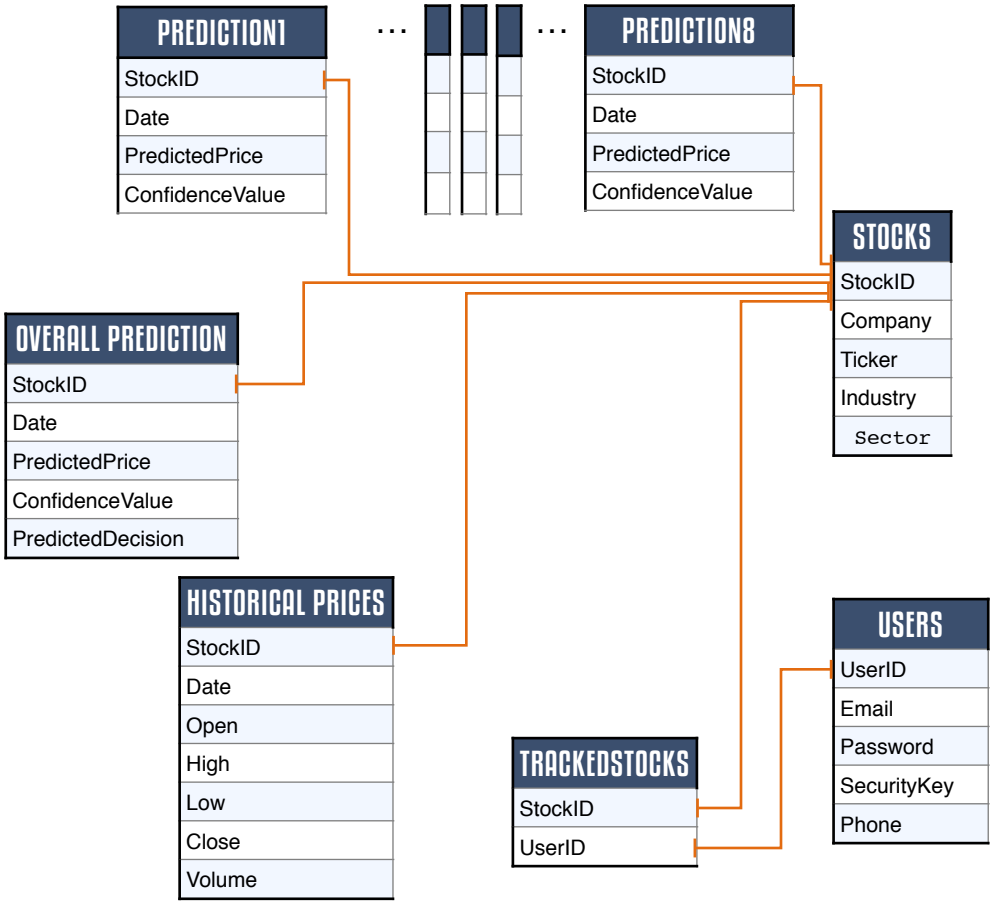


Figure-31: Relationship between tables within the database

12.5. Network Protocol

Simply, our software will communicate with a single main database. This database will use PHP scripts to both send data to our user's localized systems and call data from Yahoo stocks for analysis. The data itself will be managed by SQL software, and the PHP will output HTML to user's systems.

The components will be connected in the following way:
1) PHP requests for raw stock data from Yahoo
2) The data is stored by using SQL
3) Using SQL, we will apply our stock analysis algorithms
4) PHP waits for prompt from user's systems.
5) When prompted, PHP converts analyzed data into HTML
6) User's system converts HTML to working UI

We decided to use PHP because it is standard in creating dynamic web pages. Furthermore, it works well with relational database management systems. We chose SQL because it is the standard RDMS used to manage and manipulate large amounts of data. Lastly, we use HTML because, with the release of HTML 5, HTML has become one of the most powerful and simple markup languages for developing web pages.

12.6. Global Control Flow

Execution Orderliness:
The system can generally be defined as event-driven; it will wait for a user to make an action before processing data. The user's interaction will characterize their visit and the control structure will wait for the user's request, remaining idle until it receives such information. This allows for a user to sequence their actions upon a visit in different patterns without confusing the system. Some actions may require multithreading in order for updating to be accomplished thoroughly.

Time dependency:
The software will make use of timers to keep current, up-to-date, information regarding stocks in our database at all times. This is a real-time system that will update the database at exact defined times throughout a given day.

Concurrency:
Our system has been implemented to function on the Web; this means that multithreading must be supported. We expect that at some instance there will be concurrent users accessing either the website or the database, so that will be accounted for using multiple threads.

12.7. Hardware Requirements

There are really three instances that need to be addressed when looking at hardware specification; the hardware to run the server, the hardware to access the website, and what type of hardware is needed to use any supplemental mobile technology.

The server

The server requires a processor that has at least one 1.4 GHz single core (64-bit) or a 1.3 GHz dual-core, 2 Gigabytes of RAM, and has at least 10 gigabytes of hard drive available. The server computer must have networking capability allowing access to a router either via network cable or wirelessly. The router should be an UPnP-certified device; however this is not a requirement.

To access the website (via Computer or Mobile)
The website is accessible through any web-enabled device. Although having a display resolution of 1024x768 or greater is highly recommended.

Mobile App
In the event that a mobile app is developed, a smartphone will be required (iOS or Android).

These requirements are based off the use of a smaller database that if expanded may require more resources. Also, note that these not necessarily the minimal requirements, but are recommended for optimal performance.

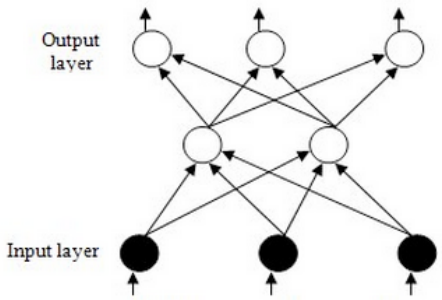
13. Algorithms and Data Structures

13.1. Algorithms

Artificial Neural Network (ANN)

An Artificial Neural Network is a system built upon the observation of how neural networks behave naturally. The practice of how artificial neurons relate to neural network training, or learning, is what the task at hand demands. This essentially means how we can adapt a mathematical algorithm to simulate the human brain.

These networks are excellent for designing the behavior of more complicated structures because of there ability to learn. A major advantage in using this method for what we seek to accomplish is that they can be used to model a system of events, stock trends, that are totally unknown and how it will react to distortion and interference or in our case the imperfections of the stock market.

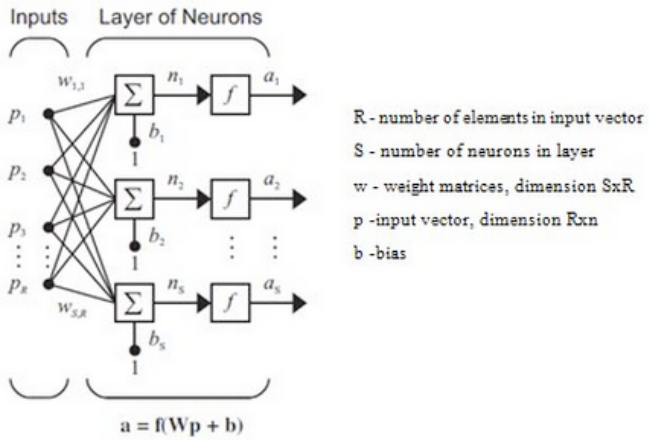


Multilayer neural network

Neuron networks can have up to three layers; the use of more layers will not improve the quality but will extend the learning process.

MATLAB features a special add-on application as one of their toolbox features known as the neural network toolbox. This means through the use of this software we can adapt the information we know about stocks to their system that is already set up to handle neural networks. Also, the code for all the activation and training algorithms has already been developed. It has all the information regarding structure that is required; structure of layers, and connectivity between them.

In order to do this MATLAB must first be "trained" with data. The data will be imported via a file that is created from historical data from a database, such as our database or a database such as Yahoo Finance. This data will have to be defined into a fitting problem for the toolbox. A set of input vectors will be arranged into columns in a matrix, while target vectors will be arranged in a second matrix.



Once the data is ready, the neutral network can be created. Through the use of various properties such as network type, layer number, train, adapt, performance, properties for, and the actual training itself a created network will be visualized graphically and made available to the user.

At this point our data will be pre-trained. Meaning our software team will have to manually upload historical data into MATLAB and analyze it to create graphical data. Our software proposes to give the user access to view these models versus actual graphical data, and for insight into the future. This is subject to change to a more dynamic and automated method that we seek to implement.

Autoregressive integrated moving average (ARIMA)

ARIMA models are said to be one of the most generic models for forecasting a time series of events that can be predict future trends. This technique is a regressive analysis that is an

established version of the random-walk and random-trend models used in stock prediction. The random walk model is a mathematical and financial theory approach to moving randomly forward, while random trend looks at the random walk model over a period of time to compare differences. This model will build upon that by adding lags of the differenced series and lags of forecasting errors.

The mathematical model below is the theoretical approach to the "ARIMA(p,d,q)" model;

The **p**, **q**, and **d** are defined as follows:

p is the number referring to order of autoregressive terms
d is the number of differences
q is the moving average

Given a time series of data X_t and index t an ARIMA model is:

$$\left(1-\sum_{i=1}^{p'}\alpha_i L^i\right)X_t=\left(1+\sum_{i=1}^{q}\theta_i L^i\right)\varepsilon_t$$

L is the lag, α_i is a parameter of auto regression, θ_i is a parameter of moving average and ε_t is the error.

Now, given that the term $\left(1-\sum_{i=1}^{p'}\alpha_i L^i\right)$ has a root of d, it can be rewritten as:

$$\left(1-\sum_{i=1}^{p'}\alpha_i L^i\right)=\left(1-\sum_{i=1}^{p'-d}\phi_i L^i\right)(1-L)^d$$

To get this model into ARIMA(p,d,q) form, p=p'−d:

$$\left(1-\sum_{i=1}^{p}\phi_i L^i\right)(1-L)^d X_t=\left(1+\sum_{i=1}^{q}\theta_i L^i\right)\varepsilon_t$$

Simple Moving Average (SMA)

A Simple Moving Average can be calculated by adding closing prices over a time period and then dividing that total by the time periods. This means that slow reactions are formed from long-term averages while faster reactions can be seen in a shorter term. These are averages over a given time-frame, traders can watch how short-term and long-term averages relate to one another and seen beginnings to an uptrend.

A generic mathematical model can be seen below.

The *n day simple moving average* for day d is computed by:

$$A_d = \frac{\sum_{i=1}^{n} M_{(d-i)+1}}{n} \quad n \leq d$$

For ten measurements, M_1 to M_{10}, we can calculate a four day moving average using the moving averages from these consecutive days:

$$\begin{aligned} A_4 &= (M_4 + M_3 + M_2 + M_1)/4 \\ A_5 &= (M_5 + M_4 + M_3 + M_2)/4 \\ &\vdots \\ A_{10} &= (M_{10} + M_9 + M_8 + M_7)/4 \end{aligned}$$

Four days of data are required before a four day moving average can be calculated, hence why the first term is A4.

Exponential Moving Average (EMA)

The Exponential Moving Average is very similar to the aforementioned SMA, but in this case more weight is given to the latest data. This model will change even more rapidly than the SMA to recent changes in price. This approach is generally more commonly used than the SMA.

The mathematical approach would be:

$$EMA = (P * \alpha) + [Previous\ EMA * (1 - \alpha)]$$

Where P = Current Price, α = Smoothing factor = $2/(1+N)$, and N = Number of time periods.

Rate of Change (ROC)

Rate of Change is a technical indicator that is a measurement of percent changes between recent prices and the price over a period of the past. It is an indication of the momentum of trends. Divergence in the ROC is generally a great indication that a drastic decline (or incline) may be in the near future by comparison of it with the price of its asset.

Mathematically, ROC can be simply looked at as:

$$ROC = \frac{(Current\ Closing\ Price - Closing\ Price"\ N")}{Closing\ Price\ "N"}$$

Where N = number of elapsed periods.

Relative Strength Index (RSI)

Relative Strength Index is another technical momentum indicator where the magnitude of recent changes in gain or loss determines conditions of assets that are bought and sold over their intended limit. This is generally done on a scale of 100 where overbought is 70 and above(overvalued asset) and when the RSI goes to 30 or below it is an indication that it is being sold too frequently and is under its intended value.

A simple mathematical approach:

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

Where RS is = (average gain)/(average loss)

Average True Range (ATR)

Average True Range is a moving average of true ranges. A true range indicator is an indication of the greatest factor when considering whether a current high is less than a current low, the absolute value of the current low are less than the previous close, and the absolute value of the current high is less than the previous close. Generally a stock with a high level of volatility will have a higher ATR while a low volatility stock has a lower ATR.

ATR calculation is as follows:

The range is simply defined as high minus low.

$$\text{true range} = \max[(\text{high} - \text{low}), \text{abs}(\text{high} - \text{close}_{\text{prev}}), \text{abs}(\text{low} - \text{close}_{\text{prev}})]$$

The **true range** is the largest of the following:
- Recent period's high minus recent period's low
- Absolute value of recent high minus the previous close
- Absolute value of recent low minus the previous close

The ATR at the moment of time t is calculated using the following formula:

$$ATR_t = \frac{ATR_{t-1} \times (n-1) + TR_t}{n}$$

Where the first ATR value is calculated using the arithmetic mean formula:

$$ATR = \frac{1}{n}\sum_{i=1}^{n} TR_i$$

Average Directional Index (ADX)

Average Directional Index is an indicator of trend strengths based off of an objective value. ADX will show the strength of trend regardless of whether it is up or down. This is a bit different from the other methods in that it does not indicate direction or momentum. This

indicator is usually observed verses the DMI, (Directional Movement Indicators) since ADX is a derivation of the relationship it has among these DMI lines in graphical terms.

The ADX is a combination of two other indicators, the positive directional indicator (+DI) and negative directional indicator (-DI). The ADX combines them and smooths the result with an exponential moving average.

Calculation of +DI and –DI needs price data consisting of high, low, and closing prices for every time period. First calculate the directional movement (+DM and −DM):

UpMove = today's high − yesterday's high
DownMove = yesterday's low − today's low
if UpMove > DownMove and UpMove > 0, then +DM = UpMove, else +DM = 0
if DownMove > UpMove and DownMove > 0, then −DM = DownMove, else −DM = 0

After selecting the number of periods, +DI and −DI are:

$$+\text{DI} = 100 * \frac{EMA\ of +DM}{ATR}$$

$$-\text{DI} = 100 * \frac{EMA\ of -DM}{ATR}$$

The EMA is calculated over the number of time periods, and the ATR is an exponential average of the true ranges. Thus:

$$ADX = 100 * \frac{EMA\ of\ |(+\text{DI}) - (-\text{DI})|}{[(+\text{DI}) + (-\text{DI})]}$$

Accumulative Swing Index (ASI)

Accumulative Swing Index is a variation of Welles Wilder's swing index. It is a way of comparison of bars that contain high, low, opening, and closing prices in a given time period. These bars can be defined as a value from 0 to 100 is an up bar and 0 to -100 is a down bar. When this index is an up bar it conveys that the long-term trend will be higher, and when it is a down bar it suggests that this trend may be lower.

The mathematical approach:

$$Swing\ Index = 50\ \left(\frac{CC - PC + 0.5(CC - CO) + 0.25(PC - PO)}{R}\right)\frac{K}{L}$$

The variables are defined as;

PO = Prior day's Open

CO = Current day's Open
PH = Prior day's High
CH = Current day's Open
PL = Prior day's Low
CL Current day's Low
PC = Prior day's Close
CC = Current day's Close
K = (the larger of the two) (1) $CH - PC$ or (2) $CL - PC$
L = Limit move
Choosing R is a multi-step process.

R = the largest of the three choices
(1) $CH - PC$
(2) $CL - PC$
(3) $CH - CL$

If (1), $R = (CH - PC) - 0.5(CL - PC) + 0.25(PC - PO)$
If (2), $R = (CL - PC) - 0.5(CH - PC) + 0.25(PC - PO)$
If (3), $R = (CH - CL) + 0.25(PC - PO)$

13.2. Data Structures

Our system, that is to be developed, will ultimately use a database to store all of its data. Making use of SQL, our database will contain tables holding the information being stored and/or retrieved. The timers, implemented through a CronJob, will be stored in a flat-file. We thought this approach was the most simple and accurate way of storing data thoroughly. The benefits of using both the flat file as well as the relational database have been extensively discussed in section 3.4 and can be read through again for further clarification. A brief summary is given below:

Text files are simple and portable and can, in most cases, be used without requiring special software. Since, we only need to store two timers, a relational database will offer no benefits in speed or organization. The implementation of the timers previously stated will be through a Cronjob and thus a flat file is ultimately the best case for storing them since they require a simple line of code each such as:

```
0 */2 * * *  /home/username/test.php
```

Which makes the user script test.php run every two hours, at midnight, 2am, 4am, 6am, 8am, and so on.

A relational database consists of multiple tables linked by "keys" — certain pieces of information shared by two or more tables. This model takes advantage of the uniformity to

build completely new tables out of required information from existing tables. In other words, it uses the relationship of similar data to increase the speed and versatility of the database. For most of its information the system-to-be relies heavily on large amounts of data storage, organization, analysis, and security. Thus, a relational database will be used for its efficiency and power when dealing with these large amounts of data.

14. User Interface Design and Implementation

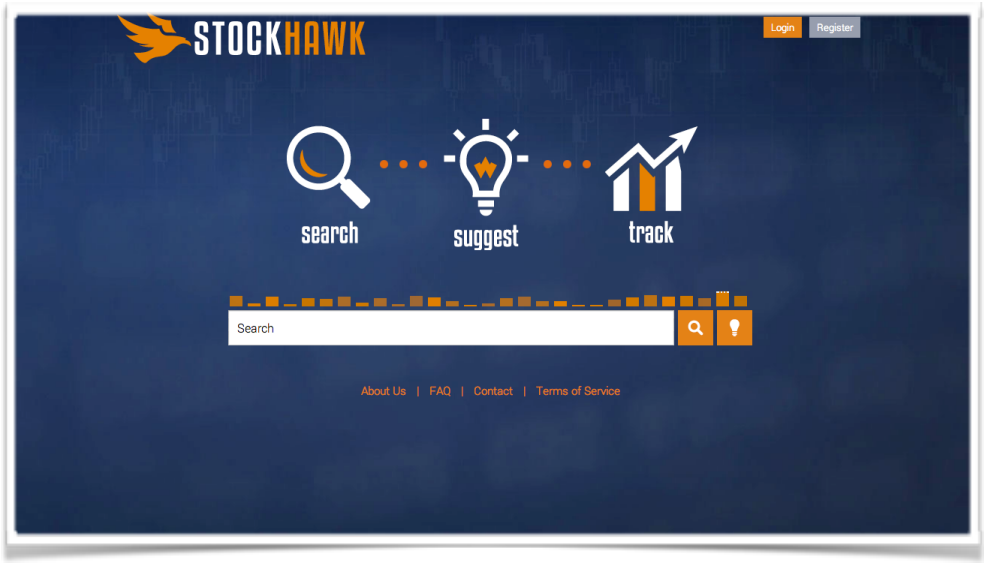


Figure 32: Home Page

Upon entering the site, the user is presented with the following interface. A search function is included on the home page for ease-of-use as the user can now quick search any stock without the hassle of logging in. The login button has been moved to the top right corner so as to separate it from the search button — this saves the user from having to differentiate between buttons.

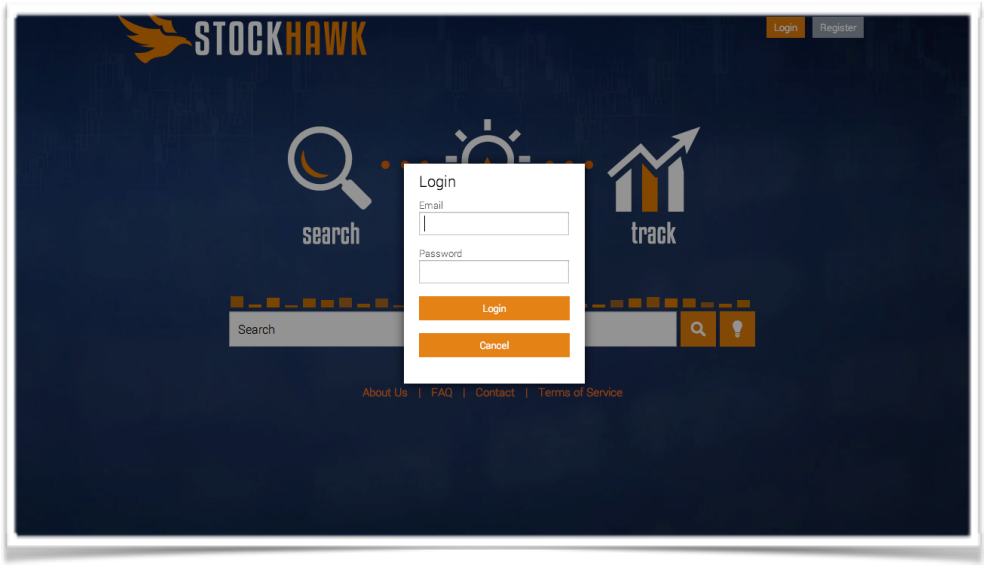


Figure-33: Login

Clicking the Login button calls the above interface. It is a standard login query which requires the user input two pieces of data: a username and a password. An account registration link is also included on this interface so the user does not have to close the login interface to register an account.

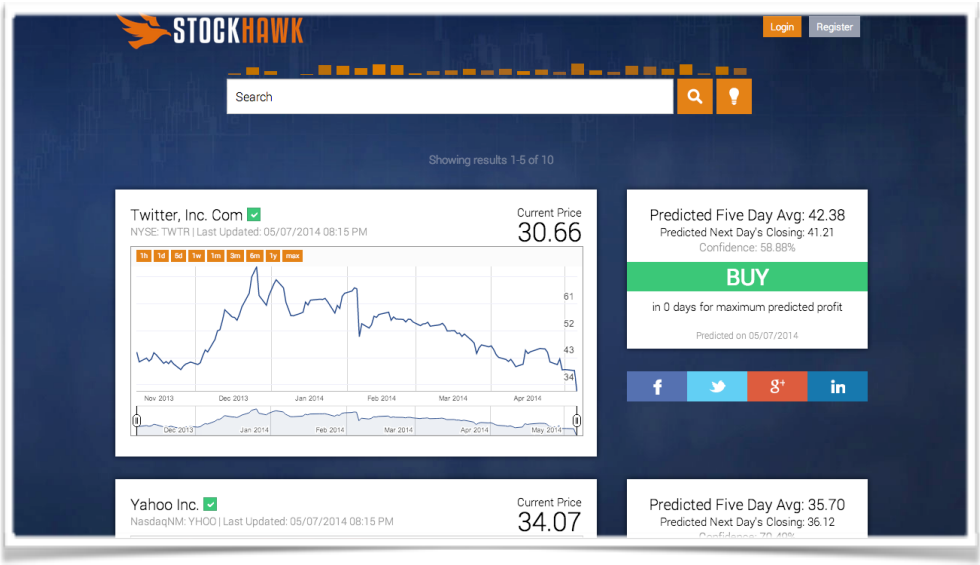


Figure-34: Dashboard

Upon logging in, the user is sent to the dashboard where users' stocks are saved and presented in the above card form. The dashboard's search module has been changed. The dashboard's original search module was unintuitive — it contained an arrow which "pulled" the search text field down. The arrow contained no clear indication it was connected to the search query, so we replaced it with three search fields: Sector, Industry, and General Search. All three are clearly labeled, easily accessible by the user, and intuitive. This will cut user effort by at least one click per search.

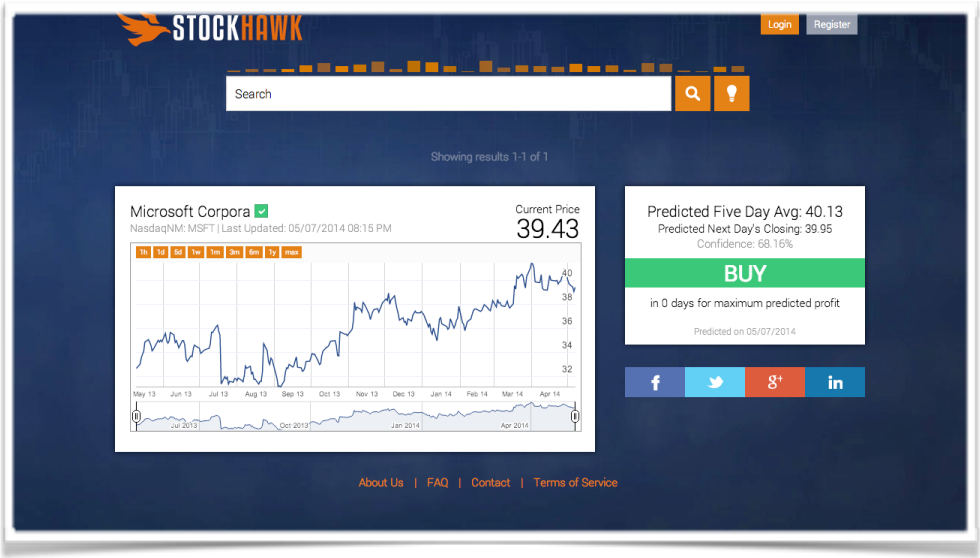


Figure-35: Search result.

Upon searching a stock and hitting enter, the user will be presented with either a match or a list of relevant stocks. All stocks will be shown with company name, price, change in price, and price charts clearly displayed because they are the most relevant data for the user. Furthermore, the user will be able to add stocks to their list of stock cards by clicking on the stock's company name or its details and traverse back to the dashboard by clicking anywhere else, keeping user effort low.

15. Design of Tests

A good controller is efficient for the stability and functionality of the proposed model and application. One can compare it to the engine of a car. Therefore, an important aspect of the controller is to keep track of certain stocks. This test can also check many other smaller issues along its path like an odometer that keeps track of the speed of a car. If an incorrect output is received then we can trace it to the cause(s) (engine, tire-rotation, pressure on gas-pedal,other calibrations) of the fault. Similarly, if the tracking of a stock is incorrect, perhaps it was a search of the stocks in the database or the current external website(YahooFinance). This can be checked through creating more boolean values. This test case is focused on the search function and uses the track function to display its findings (Function requirement 4). It will use state testing, to determine what the history of the grabbed stocks was so that it did grab the right ones. This is because in history there might be similar data such as industry, sector, and keyword however there will also be a timestamp.

```
public class ControllerTest{

       void checktracking(){

       //1. Set up
       Controller one = new Controller(/*set parameters*/);
       bool value = false;
       //make sure stocks are in the repository

       //2. act
       string[] tickers = one.search(/*parameters*/);

       /*it should be greater than one at least*/
       if(length of tickers >= 1){
              string retrievedone = tickers[1];
              if(
              /*Compared the retrieved stock with its history(using the stock name)
and
              preform certain analysis to make sure search generated the current of
true
             stock(s) (Can be done through checking timestamps.*/ ){
              value = track[retrievedone];}
              }
              else
```

```
		value = false;

	//3.verify
	if(value == 0)
		cout<<"Could not track";
	else
		cout<<"Tracking was fine";

}
}
```

For the predictor, not only must the confidence level be accurate but also the decisions that are made. Therefore, based on the comparison of what each prediction models and confidence level provided it makes only logical sense that the two should be directly related. Also the decisions made should be related to what the predicted price of a stock is based on the current price. Therefore certain events and scenarios need to be checked. This is probably the most important test case. This is because it is in the closest proximity of the customer. This covers important functional requirements 4 & 5.

```
public class predictorTest{

	void validconfidencemaking{

	//1.SetUp
	bool isFine;
Predictor oracle = new Predictor();

	//2.Act
	if(no stocks are in database)
		if(oracle.calcConfidence !=0)
			isFine = false;
	else{
	//required that some stocks are already in database
	predict();
	if(oracle.calcConfidence ==0)
		//check various models to see if that does make sense if so set isFine
to true
	else if(oracle.calcConfidence>0)
		//check various models to see if that does make sense  if so set isFine
to true

	else if(oracle.calcConfidence<0)
		//check various models to see if that does make sense  if so set isFine
to true

	else
		isFine = false;
}
//3. Verify
if(isFine==false)
	cout<<"Something is wrong";
else
	cout<<"Everything is fine";
```

```
}

bool validDecisionMaking{
//1.Setup
Predictor oracle = new Predictor();
double value = oracle.predict();
bool isFine;

//2.Act
if(oracle.predict > current price && oracle.makePredicton is buy)
       isFine = true;
else if(oracle.predict < current price && oracle.makePredicton is sell){
       isFine = true;
else if(oracle.predict > current price && oracle.makePredicton is do nothing){
       isFine = true;
else
       isFine = false;

//Verify
return isFine;
}
}
```

Boundary Testing can be performed for this too. Pass the confidence value thresholds as inputs to another test case and check what happens to the state and confidence as each prediction is sequentially entered.

Another important functional requirement - Req2 is tested by the case below. The regulation here is to make sure everything is up to do since we do not wish to derive any incorrect or out of date decisions for the customer. This test case is the first gatekeeper in that regulatory process.

```
class UpdaterChecker{

bool isUpdaterworking{

       1. SetUp
       Updater process = new Updater();
       //collect date of last known update from direct contact with Yahoo Finance
       int truedate = /*from external.
       bool isUptodate;

       2. Act
       if(truedate == process.retrievedate(/*specifiedticker*/)
              isUptodate == true;
       else
              isUptodate == false;
       3.Verify
       return isUptodate;
}
}
```

Integration Testing Strategy

The best one to utilize for this application is vertical integration. The most important reason is because it delivers a working testing platform quickly and time is of the essence. Time is precious here since time is money. Every second wasted could potentially cause customers to miss lucrative opportunities. Therefore when customers or programmers log tickets it should be answered quickly.

Testing Algorithms

To test algorithms that predict future stock prices we will simply wait and compare the predicted price to the actual price. We can further test these methods on past prices as well. The confidence value's will be calculated using a percent error between the predicted price and actual price and will be used to gauge an algorithms accuracy and effectiveness.

Testing Non-functional Requirements

Non-functional requirements that requires keeping track of time, can be tested easily by issuing a dummy variable and sleeping the program for that specified time, and checking the state of the program afterwards. Furthermore, third party browser extensions will be used to measure website loading times. Other tests such as gauging the ability of a user to intuitively utilize the system will be conducted through observing random users' inputs when told to perform a certain task. For example, a certain user might be told to perform a search and the number of correct/incorrect mouse clicks will be recorded.

16. History of Work

16.1. Key Accomplishments

Our team has decided that the most efficient way to handle the functionality of our proposed software is to split into sub groups. Since most of the team shares similar expertise and knowledge, the skills of each member will be the determining factor of when and how work may be allocated to another team member. This is to say that there will be adherence of product capabilities and awareness among multiple members of the team in the event that the customer seeks information about a specific task. That does not mean that the team in entirety will not know details about certain functionality of the software. As our software's capabilities will be tied to one another and work as a whole based on the correlative nature of the elements. Nevertheless, there will be ownership of the different aspects of design to a particular subgroup of the team.

(Vincent Chen and Mohammed Latif) were be responsible for the data mining and storage aspect of this project. Data mining is a term used in computer science for seeking patterns in large sets of data using different techniques. This team used historical data from a 3rd party supplier such as Yahoo Finance in our database, among other techniques (e.g. collected data will be received and overwritten into our database at frequent intervals). This team was solely responsible for the integration of the data collection and storage.

(Robert Adrion, Robin Karmakar, and Manoj Velagaleti) took responsibility for the prediction algorithms given the completed database of historical values and any other information attained in the database. This team was responsible for researching the most viable machine learning technique that can be implemented via software. In these regards, this generally refers to A.I. (Artificial Intelligence) algorithms that can and will learn from the data in our predefined database. This team also handled the research and implementation of as many secondary technical indicators (algorithms for trend and pattern detection) as possible.

(Peter Zhang and Syedur Rahman) have designed and implemented the user interface and graphical functions. The display provided a very organized, user-friendly and modern experience for the user. This team handled queries to the database for relevant information and showed them to the user. User accounts were also be handled by this team as well as any specific user related storage such as the “watch list”. This includes all the normal management functions an account has associated with it, such as account creation, deletion and password implementation. A tutorial system for new users can also be used.

This software project helped us learn new web development languages such as php and SQL as well as knowledge within stock trading. Key accomplishments include being able to set up a proper database and store as well as retrieve required data on demand. Furthermore, various prediction models were able to be researched and implemented to a fair degree of error. Besides technical factor’s such as these, we also have benefited from research in web design as well as the experience of being in a team oriented environment.

16.2. Future Work

In the future, we plan on adding further models to our overall predictions as well optimizing the current ones. We would also like to be able to send out notifications both via email and text to users of their tracked stocks as this could not be implemented due to a time constraint. Small other details can be added such as differentiating colors depending on whether a stock is within the system or not for the “keyword” graph shown to the administrator. Final details may include finishing touches such as FAQ page, terms of service, contact page, and various other low priority (yet necessary) web pages.

17. References

1. Software Engineering by Ivan Marsic, Rutgers University
http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf

2. Background knowledge of stock market prediction:
http://en.wikipedia.org/wiki/Stock_market_prediction

3. Useful definitions:
http://www.investopedia.com

4. Information on internet speeds and page loading times:
http://www.akamai.com/dl/akamai/akamai-soti-q313.pdf

5. Information on password reuse statistics:
http://landing2.trusteer.com/sites/default/files/cross-logins-advisory.pdf

6. Listed companies within the World Federation of Exchanges as of January 2014:
http://www.world-exchanges.org/statistics/monthly-reports

7. Mathematical Models:
http://www.vatsals.com/Essays/MachineLearningTechniquesforStockPrediction.pdf
http://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average

8. Software Engineering by Ivan Marsic, Rutgers University
http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf

9. Background knowledge of stock market prediction:
http://en.wikipedia.org/wiki/Stock_market_prediction

10. Useful definitions:
http://www.investopedia.com

11 Information on internet speeds and page loading times:
http://www.akamai.com/dl/akamai/akamai-soti-q313.pdf

12. Information on password reuse statistics:
http://landing2.trusteer.com/sites/default/files/cross-logins-advisory.pdf

13. Listed companies within the World Federation of Exchanges as of January 2014:
http://www.world-exchanges.org/statistics/monthly-reports

14 Mathematical Models:

http://www.vatsals.com/Essays/MachineLearningTechniquesforStockPrediction.pdf

http://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average