# Web-Based Stock Forecaster


## Course : 16:332:568 Software Engineering Web Application

**Group 6**

Group Member:
Ze Liu(zl265)
Tong Wu(tw445)
Xinwei Zhao(xz245)
Jay borkar(jb1545)
Xinyu Lyu(xl422)

Instructor : Prof. Shiyu Zhou

# Table of Contents

1.Customer Statement of Requirements....4
1.1. Problem Statement....4
2. Glossary of Terms....5
3. System Requirements....7
3.1. Enumerated Functional Requirements....7
3.2. Enumerated Non-Functional Requirements....8
3.3. On Screen Appearance Requirements....9
4. Functional Requirements Specification....13
4.1. Stakeholders....13
4.2. Actors and Goals....13
4.3. Use Cases....13
4.3.1. Casual Description....14
4.3.2. Use Case Diagram....14
4.3.3. Fully Dressed Description....14
4.3.4. System Sequence Diagrams....19
5. Effort Estimation....20
5.1. Unadjusted Actor Weight....20
5.2. Unadjusted Use Case Weight (UUCW)....21
5.3. Technical Complexity Factor (TCF) — Nonfunctional Requirements....22
5.4. Environment Complexity Factor (ECF)....24
6. Domain Analysis....24
6.1. Concept Definitions....24
6.2. System Operation Contracts:....28
6.3. Mathematical Models....30
7. System Architecture and System Design....31
7.1. Architectural Styles....31
7.3. Mapping Subsystems to Hardware....32
7.4. Persistant Data Storage....32
7.5. Network Protocol....33
7.6. Global Control Flow....33
7.7. Hardware Requirements....34
8. Algorithms and Data Structures....34
8.1. Algorithms....34
Artificial Neural Network (ANN)....34
Long Short Term Model Networks (LSTM)....35

Support Vector Machine (SVM).....36
Bayesian Curve Fitting.....36
On Balance Volume (OBV) .....38
Rate of Change (ROC) .....38
Moving Average Convergence Divergence (MACD).....38
8.2. Data Structures.....38
9. User Interface Design and Implementation.....39
10. Design of Tests .....40
11. History of Work.....41
11.1. Future Work .....42
12. References .....43

## CONTRIBUTION BREAKDOWN

All members contributed equally.

# 1.Customer Statement of Requirements

## 1.1 Problem Statement

The stock market has always been a place where businessmen could invest successfully and earn a profit. But investing successfully and earning a profit can be a challenge and most investors lose money every year. The stock market is constantly rising and falling on a daily bases, stocks can go up a certain amount one day and then the next day their prices may have gone down significantly. As the stocks are always changing and without proper research and analysis, it is difficult to make smart investments.

For daily trading, these risks are a major factor in keeping away novel, eager yet unaccustomed, traders from the market. They lack the appropriate tools needed for the decision making process when compared to existing professional traders. These investors are not able to predict a stock's future performance due to various factors. It's very common for individual investors to not know the financial stability of the company in which they are trying to invest. There are many reasons for these but the primary reason behind this is the fact that most investors just do not have the time and resources to do research and implement in-depth analysis of the company in which they are investing and the stock market. Although professional investors have an advantage as they rely on a research heavy way to invest, their methods are not only time consuming and stressful but also prone to human error.

The stock market is constantly rising and falling. The problem here lies with the fact that the same volatility in the market that can be exploited for profit, can be the cause for loss as well. As both amateur and knowledgeable investors, this is what worries as.

With this realization, our project will attempt to benefit a broad range of investors by predicting future stock prices which can be used by them to make their own decision on whether to buy, sell, or hold the stock. There are many prediction methodologies that fall into two general categories : fundamental analysis and technical analysis. Fundamental analysis involves evaluating a company's past performance as well as the credibility of its accounts. Technical analysis involves determining the future price of a stock based solely on the potential patterns discovered from observing historical stock process and volumes of trades.

Our project focuses on technical analysis. The first and foremost purpose of our system is to predict future stock prices.

# 2. Glossary of Terms

**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.

**Back-Propagation Network:** A feed forward multilayered neural network that is a commonly used neural network paradigm.

**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.

**Epoch:** A step in the training process of an artificial neural network

**Forecast:** A prediction of the future based on special knowledge

**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

**LSTM:** Long Short-term memory units, a special kind of Recurrent neural network

**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.

**Opening Price:** The price a stock starts off at a particular trading day.

**Range:** A stock's low price and high price for a particular trading period, such as the close of a day's trading, the opening of a day's trading, a day, a month, or a year.

**Regression:** A mathematical way of stating the statistical linear relationship between one independent and one dependent variable

**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.

**Stock Market:** The marketplace for buyers and sellers of stocks.

**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.

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

# 3. System Requirements

## 3.1 Functional Requirements

| Identifier | Priority | Requirement |
|---|---|---|
| REQ 1 | 5 | The system will acquire past stock prices and data from all stocks listed on Yahoo! Finance on a daily basis. The stock data will then be placed into a stock database. |
| REQ 2 | 5 | The system shall allow the administrator to add and remove stocks from the list of stocks that predictions are made on. |
| REQ 3 | 5 | The system shall periodically apply prediction algorithms or models on the obtained data and store the results to a central database. |
| REQ 4 | 5 | The system shall allow for users to select for a particular company and display the predicted trade decision associated with that stock. |
| REQ 5 | 4 | The stock data stored in the database will be used by the prediction algorithm to calculate the stock's prediction |
| REQ 6 | 4 | The website should display the graph of the stock prices. |
| REQ 7 | 3 | 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. |
| REQ 8 | 2 | The system should log and display to the administrator the time taken for previous prediction sessions. |
| REQ 9 | 1 | The system should track what stocks users are searching for and log and display these analytics to the administrator. |

**Analysis: Functional Requirements**

Our main requirement is to be able to obtain the historical data for all or most stocks (REQ-1) in the Yahoo! Finance API and then from this data be able to generate a prediction (REQ-3). For REQ-1, our end goal is to have all the stocks listed in Yahoo! Finance to be stored in the database. However, for demo and test purposes, we will only have 40 or so of the top fortune 500 companies in the database. However, if a user does search for a stock that is not in the database, the system will query the information from Yahoo! Finance and the user will still be able to obtain the information.

REQ-5 are straight to the point, the prediction algorithms that the Prediction Team will create uses the historical data stored in the database to generate a prediction model to be graphed alongside the stock's historical values. Finally, requirement REQ-8, was

suggested by the customer to further allow the administrator to satisfy user needs based on their searches.

## 3.2 Non-Functional Requirements

| Identifier | Requirement |
|---|---|
| REQ 10 | The system shall ensure that if a copy of the database were to be acquired unlawfully, all user passwords shall remain secure. |
| REQ 11 | The system shall be able to run through all prediction models for each and every listed stock within 1 hour. |
| REQ 12 | The system shall allow for users to get a prediction for a stock within at most 2 mouse click. |
| REQ 13 | The database will be updated daily to account for the daily changes to stock values. |
| REQ 14 | Prediction models will be updated daily. |

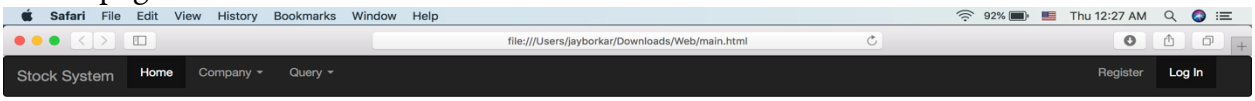
**Analysis: Non-Functional Requirements**

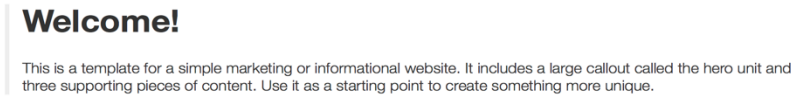
The key requirements are REQ-13 and REQ-14. In order for all the stock data to be up to date, the database will be updated daily and the prediction algorithms as well. Updating both the database and the prediction algorithm on a daily basis, is just a place holder value.

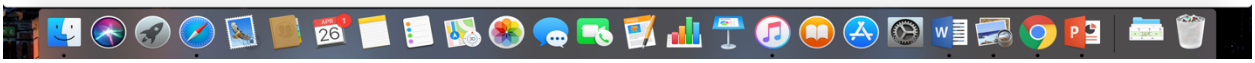
Since the system will be storing user accounts, REQ-10 needs to be specified for security purposes. REQ-11 insure that latency are kept to a low since the system will be a web service where a fraction of a second in loading can be the difference between user satisfaction and disappointment. REQ-12 insures user efforts for the most prominent feature (searching for predictions) is kept to a low.

## 3.3 On Screen Appearance Requirements

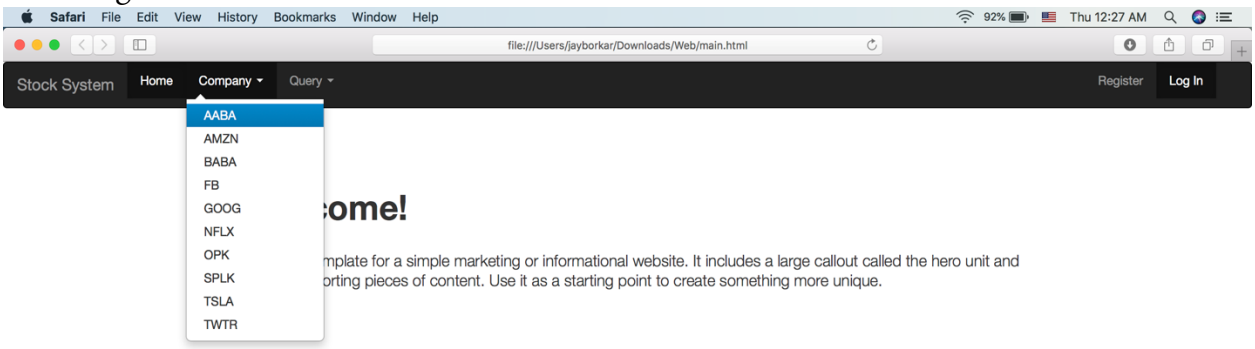
Home page

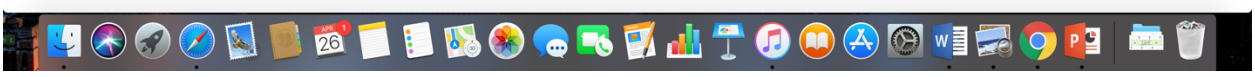


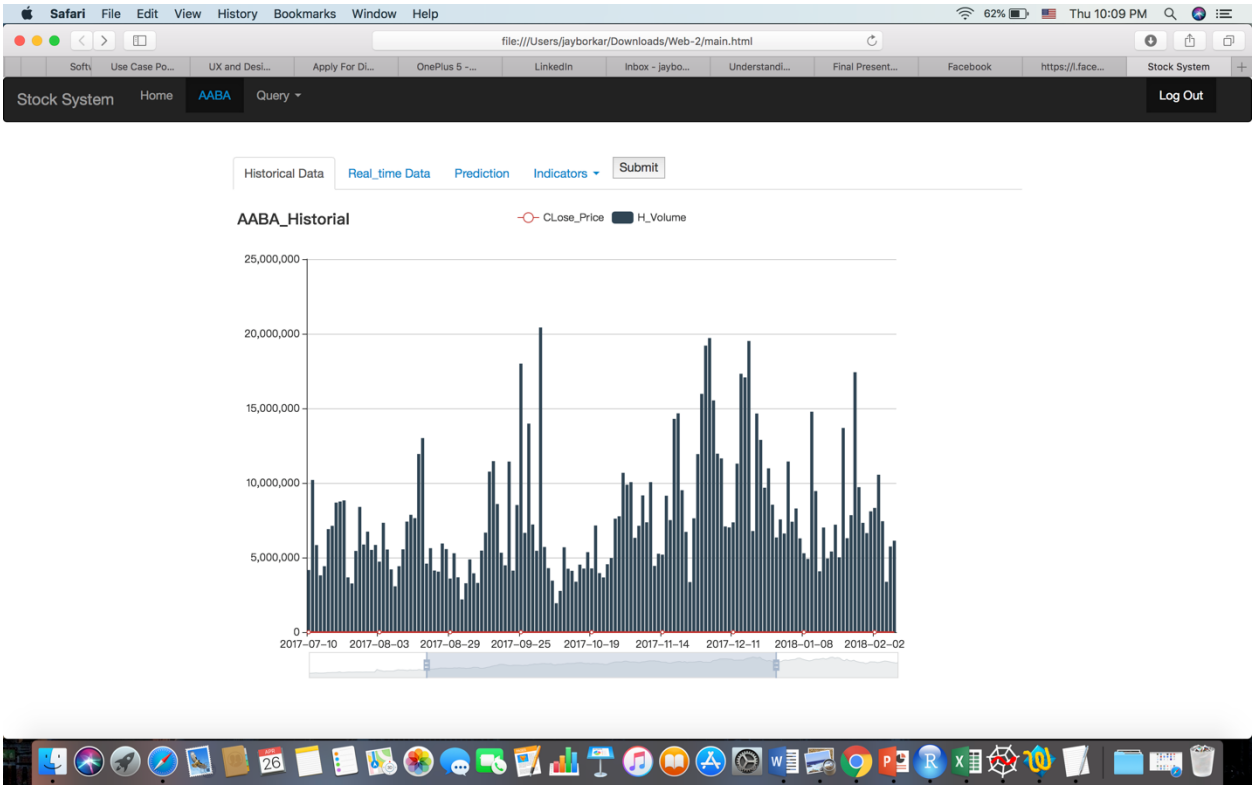



Selecting a stock



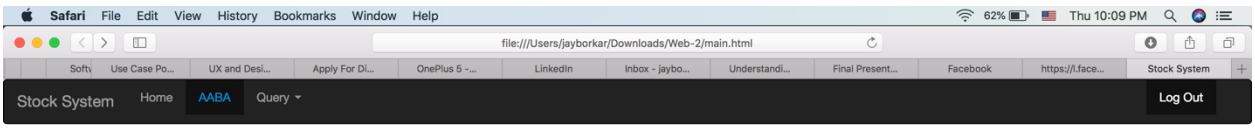



Four sections – Historical data, real-time data, Predictions and Indicators




Real time data

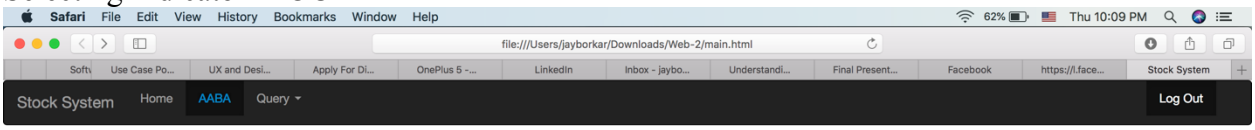
Prediction



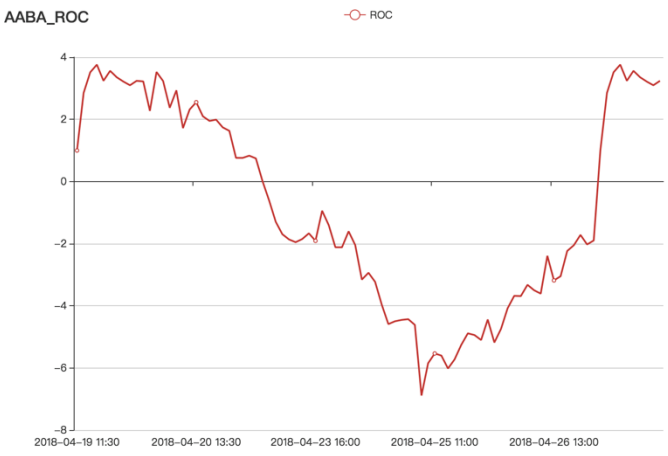

Historical Data | Real_time Data | Prediction | Indicators | Submit

| Method | Prediction |
|---|---|
| SVM | 71.63617582628244 |
| ANN | 72.57140350341797 |
| LSTM | 68.38434600830078 |
| BYS | 75 |

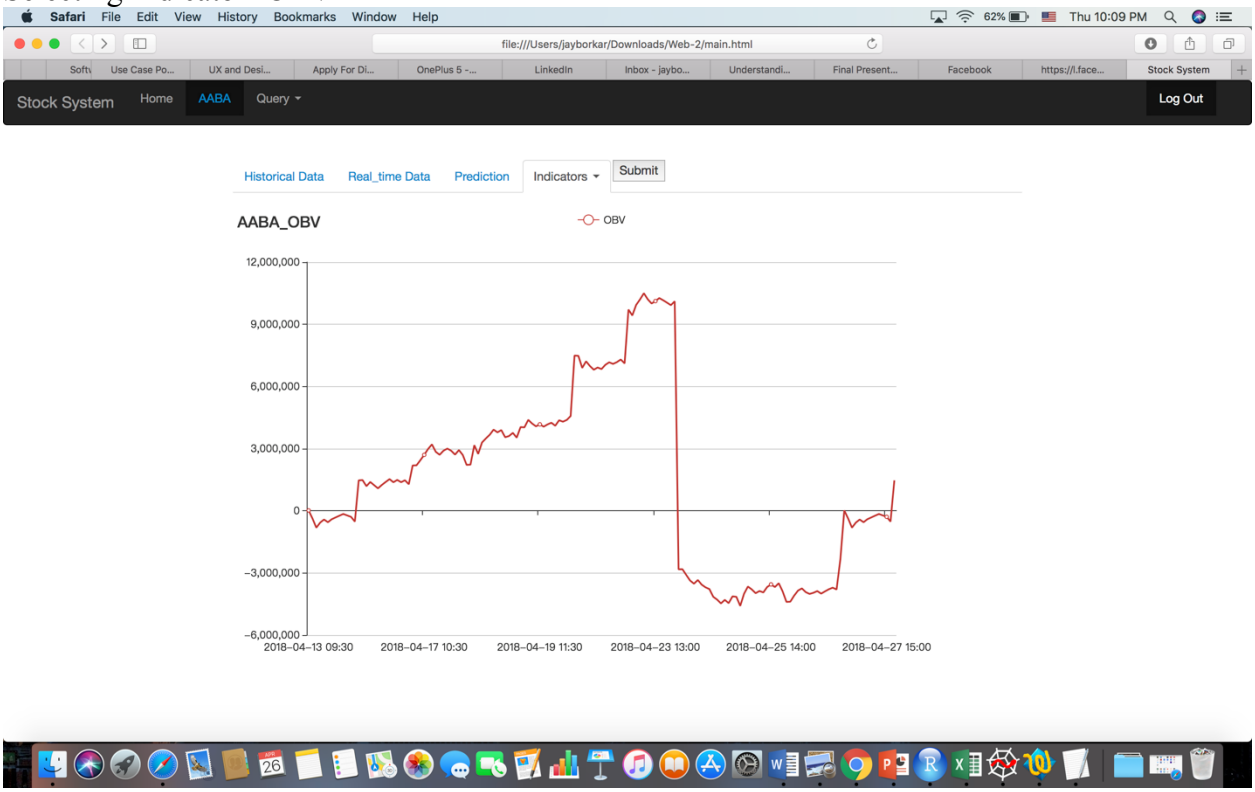
Selecting Indicator- ROC




Historical Data | Real_time Data | Prediction | Indicators | Submit




Selecting Indicator- OBV




Selecting Indicator- MACD

# 4. Functional Requirements Specification

## 4.1 Stakeholders

Three Stakeholders can be identified:

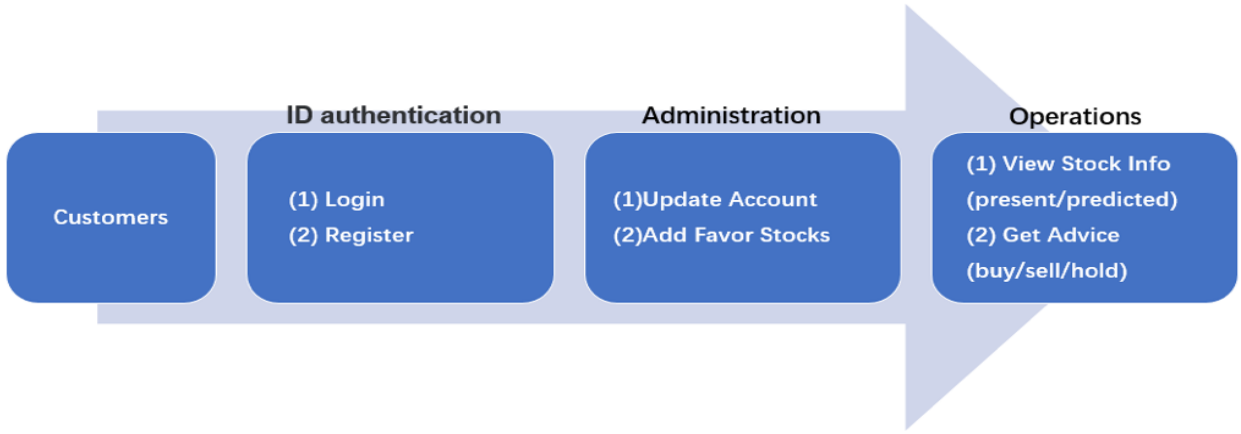
1. Users: any user of the web service registered or not.
2. Advertisers: provider of advertisements for the web service.
3. Administrator: maintains and updates website services.

## 4.2 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.

## 4.3 Use Cases



### 4.3.1 Casual Descritption

The summary use cases are as follows :

UC: Login — Allows user to access their account and view information for their tracked stocks.

UC: Register — Allow a visitors to fill out the a registration form and become a user.

UC: Select — Allows users to select the company to view the results.

UC: AddStock — Allows the admin to add a stock to the list of stocks predictions are made on. («include» login).

UC: RemoveStock — Allows the admin to remove a stock from the list of stocks predictions are made on («include» login).

UC: Update - Allows timer to initiate the loading of current stock prices from the price-provider into the database.

UC: PredictAndNotify — Allows timer to initiate the prediction of future stock prices to be later stored into the database. Registered Users are then notified if a change in prediction is made for a stock he/she has chosen to track.

UC: Logout — allow users to log out of the system.

### 4.3.4 Fully Dressed Description

Use case UC: 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: Add Stock**

**Related Requirements:** REQ2
**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
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.
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.
3. System notifies Administrator of all stocks that have failed to be added.

Use case UC: Select, allows the user the company to view the results

| Use Case : Select |
| --- |
| **Related Requirements:** REQ4, REQ9<br>**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<br>**Preconditions:** none<br>**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.<br>**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.<br>**Flow of events for the main success scenario:**<br>1. User types in the name or ticker symbol of a stock he/she would like to search for inside the "search" text field.<br>2. System verifies that the given stock exists within the Database.<br>3. System increments the "search" counter for that stock within the Database.<br>4. System retrieves historical prices of that stock from the Database.<br>5. Systems sends historical prices to the Grapher.<br>6. Grapher returns a line graph of the change of prices over time to the System.<br>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.<br>8. System displays the graph, prediction results, confidence values, and current stock price to the User.<br><br>**Flow of Events for Alternate Scenarios:**<br>2a. The given stock does not exist within the Database<br>1. System checks if the failed keyword exists within the Database.<br>1a. Exists: Systems increments the "search" counter for the failed keyword<br>1b. Doesn't Exist: System adds the failed keyword to the Database.<br>2. System displays a "no results found" message to the User and he/she is prompted to try another search. |

Use case: Update, deals with updating prices within the database: 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 : 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.
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.
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.

The final elaborated use case is also the most important. UC: 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 : Predict and Notify** |
| --- |
| **Related Requirements:** REQ3, REQ4, REQ5, REQ8<br>**Initiating Actors:** Timer<br>**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.<br>Participating Actors: Database<br>**Preconditions:** At least one stock exists within the database.<br>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.<br>**Failure End Conditions:** no meaningful failure end conditions.<br><br>**Flow of Events for Main Success Scenario:**<br>1. Timer notifies the System to start making predictions.<br>2. System resets and starts an internal timer.<br>3. System retrieves list of all stocks from the Database. (loop: for all stocks)<br>(loop: for all prediction models)<br>4. System predicts the future price of the stock and calculates a confidence value.<br>5. System stores the prediction and confidence value within the Database.<br>(end loop)<br>6. System retrieves results for all prediction models from within the Database.<br>7. System calculates overall prediction and gain/loss. (end loop)<br>8. System stops internal timer.<br>9. System stores the time taken to make all predictions in the Database.<br>10. System retrieves list of all Registered Users from the Database. (loop: for all Registered Users)<br>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)<br>12.System retrieves current and previous predictions for the stock from the Database.<br>13. System verifies that the current and previous predictions are different. |

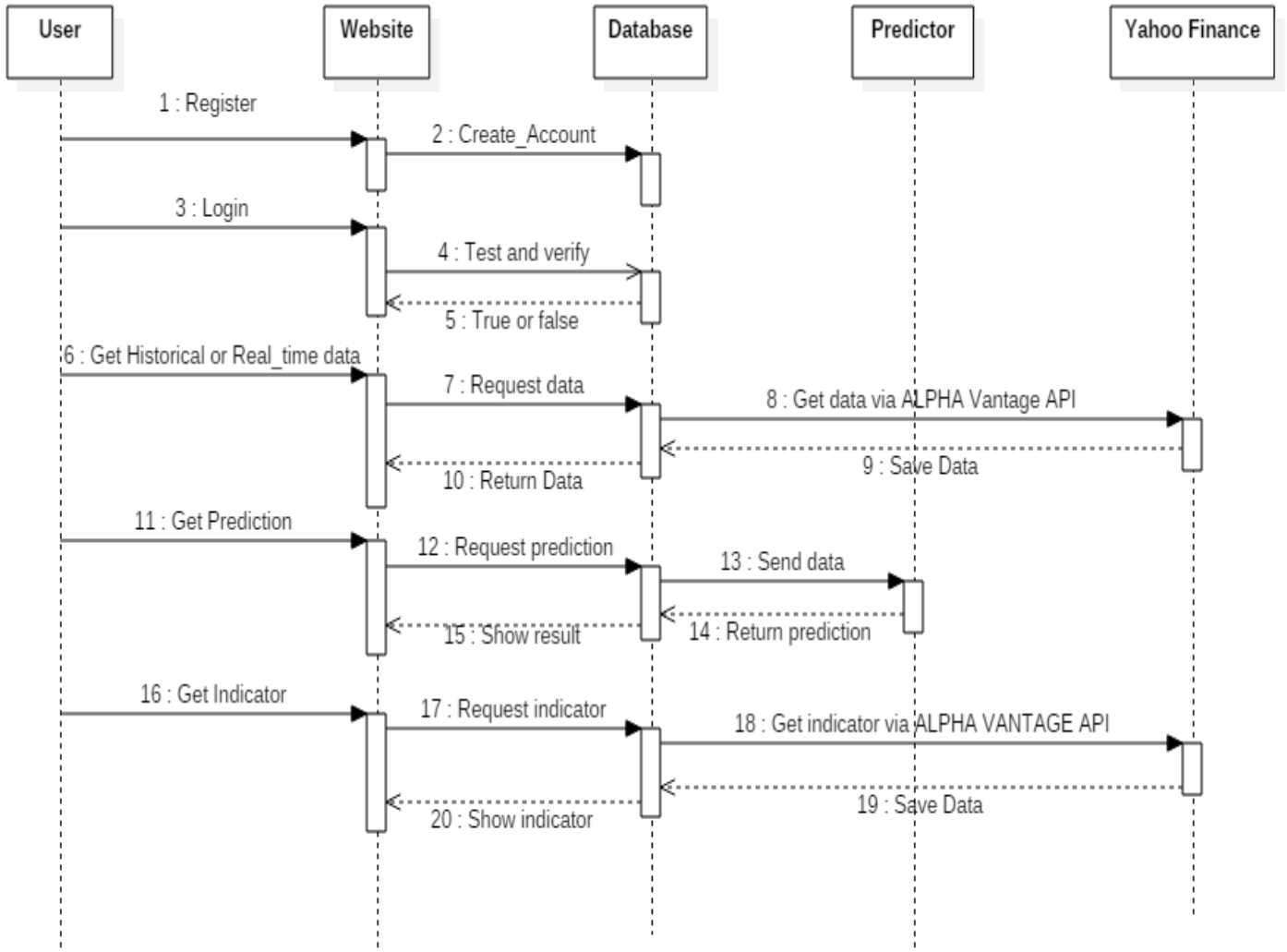
14. System alerts Registered User through his/her suggested means of alert. (end loop)

**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)

### 4.3.5 System Sequence Diagram



# 5. Effort Estimation

## 5.1Unadjusted Use Case Weight (UUCW)

The UUCW is one of the factors that contribute to the size of the software being developed. It is calculated based on the number and complexity of the use cases for the system. To find the UUCW for a system, each of the use cases must be identified and classified as Simple, Average or Complex based on the number of transactions the use case contains. Each classification has a predefined weight assigned. Once all use cases have been classified as simple, average or complex, the total weight (UUCW) is determined by summing the corresponding weights for each use case. The following chart shows the different classifications of use cases based on the number of transactions and the weight value assigned for each use case within the classification.

| 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 |
| 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 |
| 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 + 5 × average + 4 × complex = (5 × 5) + (5 × 10) + (4 × 15) = 135

## 5.2 Unadjusted Actor Weight (UAW)

The UAW is another factor that contributes to the size of the software being developed. It is calculated based on the number and complexity of the actors for the system. Similar to finding the UUCW, each of the actors must be identified and classified as Simple, Average or Complex based on the type of actor. Each classification also has a predefined weight assigned. The UAW is the total of the weights for each of the actors. The following chart shows the different classifications of actors and the weight value assigned.

| 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 |

UAW $=2\times\text{simple}+2\times\text{average}+4\times\text{complex} = (2 \times 1) + (2 \times 2) + (4 \times 3) = 18$

UUCP = UAW + UUCW = 18 + 135 = 153

## 5.3 Technical Complexity Factor (TCF)

The TCF is one of the factors applied to the estimated size of the software in order to account for technical considerations of the system. It is determined by assigning a score between 0 (factor is irrelevant) and 5 (factor is essential) to each of the 13 technical factors listed in the table below. This score is then multiplied by the defined weighted value for each factor. The total of all calculated values is the technical factor (TF). The TF is then used to compute the TCF with the following formula: TCF = 0.6 + (TF/100)

| Technical Factor | Description | Perceived Complexity | Weight | Calculated Factor |
|---|---|---|---|---|
| T1 | Distributed system (running on | 3 | 2 | $2\times3=6$ |

| | | | | |
|---|---|---|---|---|
| | multiple machines) | | | |
| T2 | Performance objectives (are response time and throughput performance critical?) | 3 | 1 | 1x3=3 |
| T3 | End-user efficiency | 3 | 1 | 1x3=3 |
| T4 | Complex internal processing | 3 | 1 | 1x3=3 |
| T5 | Reusable design or code | 0 | 1 | 1x0=0 |
| T6 | Easy to install (are automated conversion and installation included in the system?) | 3 | 0.5 | 0.5x3=1.5 |
| T7 | Easy to use (including operations such as backup, startup, and recovery) | 5 | 0.5 | 0.5x5=2.5 |
| T8 | Portable | 2 | 2 | 2x2=4 |
| T9 | Easy to change (to add new features or modify existing ones) | 1 | 1 | 1x1=1 |
| T10 | Concurrent use (by multiple users) | 4 | 1 | 1x4=4 |

| | | | | |
|---|---|---|---|---|
| T11 | Special security features | 5 | 1 | 1x5=5 |
| T12 | Provides direct access for third parties (the system will be used from multiple sites in different organizations) | 1 | 1 | 1x1=1 |
| T13 | Special user training facilities are required | 0 | 1 | 1x0=0 |

TFC = C1+C2*TFT
= 0.6 + 0.01 × 34 = 0.94

## 5.4 Environment Complexity Factor (ECF)

The ECF is another factor applied to the estimated size of the software in order to account for environmental considerations of the system. It is determined by assigning a score between 0 (no experience) and 5 (expert) to each of the 8 environmental factors listed in the table below. This score is then multiplied by the defined weighted value for each factor. The total of all calculated values is the environment factor (EF). The EF is then used to compute the ECF with the following formula:

ECF = 1.4 + (-0.03 x EF)

| 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 =153 × 0.94 × 0.83

=119.37 or 120 UseCasePoints

Duration =UCP × PF =120*28 = 3360 hours

The software system is estimated to take about 3,360 hours to complete.

# 6. Domain Analysis

## 6.1 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)
**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)

## 6.2 System Operation Contracts

**Operation Contract – Log On Function**

Name: Log On
Responsibilities: Allows user to log into his or her account.

Cross References: Use Cases: 1 (Search Stock), 2 (Check News Feed), 4 (Visit Friend's Profile)

Exceptions: If user does not have account; allows user to create an account.

**Operation Contract – Search Function**

Name: Search

Responsibilities: Takes users inputted stock name, symbol, or user name and match it to the database, and retrieve the data for that stock or user.

Cross References: Use Cases: 1 (Search Stock) and 4 (Visit Friend's Profile)
Exceptions: If the stock name, symbol, or user name does not exist.

Preconditions: You must be connected to the Internet to search stocks or users.

Post conditions:
- A stocks prediction was created and displayed.
- A user profile was displayed

**Operation Contract – Changing Tabs**

Name: Changing Tabs

Responsibilities: When users tap a tab the application switches to that page.

Cross References: Use Case: 4 (Visit Friend's Profile)

Exceptions: None.

Preconditions: If you go to the Social tab without being logged in then it will redirect you.

Post conditions:
- The screen was changed to another one.

**Operation Contract – Log Off Function**

Name: Log Off

Responsibilities: To disconnect the user from his or her account.

Cross Reference: Use Cases: 1 (Search Stock), 2 (Check News Feed), 4 (Visit Friend's Profile)

Exceptions: None.
Preconditions: Logging out will disconnect you from most of the features of the application.

Postconditions:
- The user's connection was ended.

## 6.3 Mathematical Models

**On Balance Volume (OBV)**

- Definition: It measures buying and selling pressure as a cumulative indicator that adds volume on up days and subtracts volume on down days. Chartists can look for divergences between OBV and price to predict price movements or use OBV to confirm price trends.
- Calculation:
    - If the closing price is above the prior close price then: Current OBV = Previous OBV + Current Volume
    - If the closing price is below the prior close price then: Current OBV = Previous OBV - Current Volume
    - If the closing prices equals the prior close price then: Current OBV = Previous OBV (no change)
- Recommendation: Research that OBV would often move before price.
    - Expect prices to move higher if OBV is rising while prices are either flat or moving down (Buy)
    - Expect prices to move lower if OBV is falling while prices are either flat or moving up (Sell or Hold)

**Rate of Change (ROC)**

- Definition: It is also referred to as simply Momentum that measures the percentage increase or decrease in price over a given period of time
- Calculation: ROC = [(Close - Close n periods ago) / (Close n periods ago)] * 100%
- Recommendation:
    - Prices are expected rising as long as the Rate-of-Change remains positive (Buy)
    - Prices are expected falling when the Rate-of-Change is negative (Sell or Hold)

**Moving Average Convergence Divergence (MACD)**

- Definition: It turns two trend-following indicators, moving averages, into a momentum oscillator by subtracting the longer moving average from the shorter moving average. As a result, the MACD offers the best of both worlds: trend following and momentum.
- Calculation: Standard MACD Line: (12-day exponential moving average (EMA) minus a 26-day EMA)
- Recommendation: Potential buy signals occur when the MACD moves above zero and potential sell signals when it crosses below zero

# 7. System Architecture and System Design

## 7.1 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

## 7.2 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.

## 7.3 Persistent Data Storage

For the stock forecaster the system does need to save data that will outlive a single execution of the system. A sample of the data that should be saved is stock tickers and their corresponding company names, user information, and stock data. The data will be stored using a relational database, specifically MySQL. MySQL was chosen for the job due to its open source nature and has all the functions required for the job. Also, several team members have experience with this database and feel that it is the best option. The data will be stored in three separate tables. The first table will hold the user data which includes username, password, and portfolio. The second table will have stock ticker, the

corresponding company name, and information about the company such as industry and a small description. The third and final table will have the stock ticker and the moving average data computed for the stock.

## 7.4 Network Protocol

Simply, our software will communicate with a single main database. This database will use Python 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 Python will output HTML to user's systems.

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

## 7.5 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.

## 7.6 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.

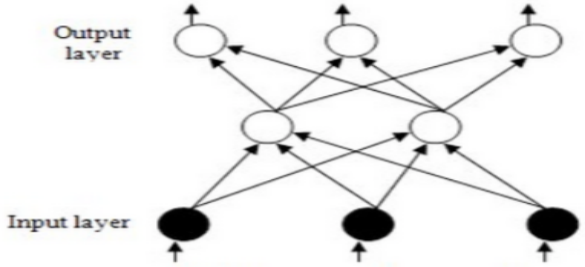
# 8. Algorithms and Data Structures

## 8.1 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

## LSTM networks

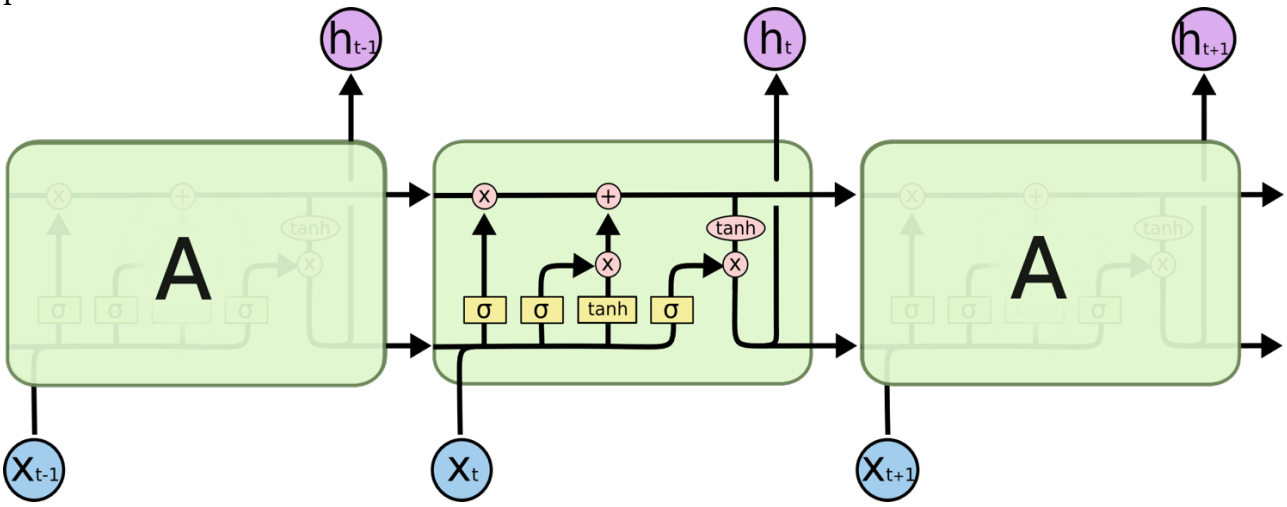
Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. A common architecture of LSTM is composed of a memory cell, an input gate, an output gate and a forget gate. An LSTM (memory) cell stores a value (or state), for either long or short time periods. This is achieved by using an identity (or no) activation function for the memory cell. In this way, when an LSTM network (that is an RNN composed of LSTM units) is trained with backpropagation through time, the gradient does not tend to vanish.

The LSTM gates compute an activation, often using the logistic function. Intuitively, the input gate controls the extent to which a new value flows into the cell, the forget gate controls the extent to which a value remains in the cell and the output gate controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit.

There are connections into and out of these gates. A few connections are recurrent. The weights of these connections, which need to be learned during training, of an LSTM unit are used to direct the operation of the gates. Each of the gates has its own parameters, that is weights and biases, from possibly other units outside the LSTM unit.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn. The key to LSTMs is the cell state. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.

Gates are composed out of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!" An LSTM has three of these gates, to protect and control the cell state.




**The repeating module in an LSTM contains four interacting layers.**

## Support Vector Machine

Support vector machines (SVMs, also support vector networks[1]) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

The method of Support Vector Classification can be extended to solve regression problems. This method is called Support Vector Regression. The model produced by support vector classification (as described above) depends only on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by Support Vector Regression depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction.

There are three different implementations of Support Vector Regression: SVR, NuSVR and LinearSVR. LinearSVR provides a faster implementation than SVR but only considers linear kernels, while NuSVR implements a slightly different formulation than SVR and LinearSVR.

## Bayesian Curve Fitting

In the curve fitting problem, we are given the training data x and t, along with a new test point x, and our goal is to predict the value of t. We therefore wish to evaluate the predictive distribution $p(t|x, x, t)$. Here we shall assume that the parameters $\alpha$ and $\beta$ are fixed and known in advance

A Bayesian treatment simply corresponds to a consistent application of the sum and product rules of probability, which allow the predictive distribution to be written in the form

$p(t|x, x, t) = \int p(t|x, w)p(w|x, t)\, dw$. Here $p(w|x, t)$ is the posterior distribution over parameters.

The integration in the above equation can also be performed analytically with the result that the predictive distribution is given by a Gaussian of the form

$$p(t|x, x, t) = N(t|m(x), s^2(x))$$

where the mean and variance are given by

$$m(x) = \beta\varphi(x)^T S \Sigma_{n=1}^{N} \varphi(x_n) t_n$$

$$s^2(x) = \beta^{-1} + \varphi(x)^T S \varphi(x)$$

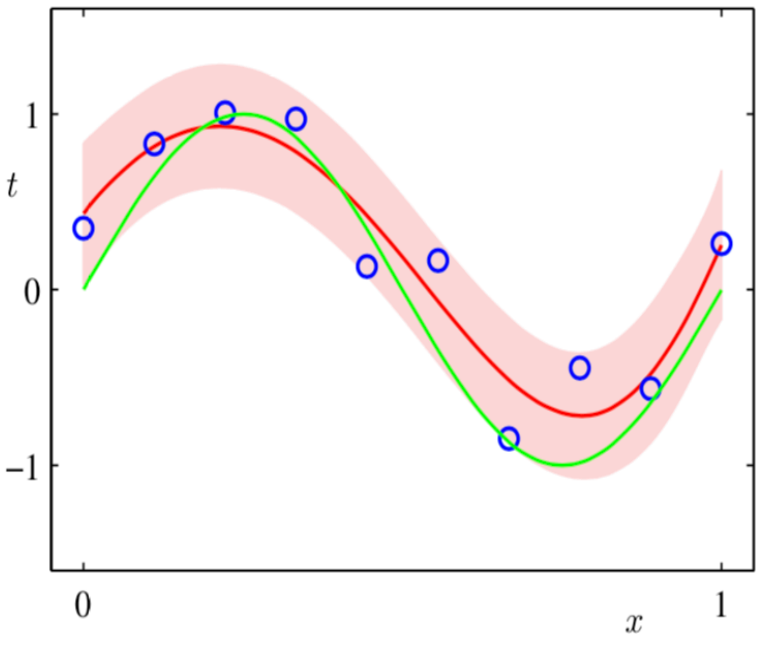
Here the matrix S is given by

$$S^{-1} = \alpha I + \beta \Sigma_{n=1}^{N} \varphi(x_n)\varphi(x)^T$$

where I is the unit matrix, and we have defined the vector $\varphi(x)$ with elements $\varphi_i(x) = x^i$ for $i = 0,...,M$.

We see that the variance, as well as the mean, of the predictive distribution in $p(t|x, x, t) = N(t|m(x), s^2(x))$ is dependent on x.

The predictive distribution resulting from a Bayesian treatment of polynomial curve fitting using an $M = 9$ polynomial, with the fixed parameters $\alpha = 5 \times 10^{-3}$ and $\beta = 11.1$ (corresponding to the known noise variance), in which the red curve denotes the mean of the predictive distribution and the red region corresponds to $\pm 1$ standard deviation around the mean.



**On Balance Volume (OBV)**

- Definition: It measures buying and selling pressure as a cumulative indicator that adds volume on up days and subtracts volume on down days. Chartists can look for divergences between OBV and price to predict price movements or use OBV to confirm price trends.
- Calculation:
    - If the closing price is above the prior close price then: Current OBV = Previous OBV + Current Volume
    - If the closing price is below the prior close price then: Current OBV = Previous OBV - Current Volume
    - If the closing prices equals the prior close price then: Current OBV = Previous OBV (no change)
- Recommendation: Research that OBV would often move before price.
    - Expect prices to move higher if OBV is rising while prices are either flat or moving down (Buy)
    - Expect prices to move lower if OBV is falling while prices are either flat or moving up (Sell or Hold)

**Rate of Change (ROC)**

- Definition: It is also referred to as simply Momentum that measures the percentage increase or decrease in price over a given period of time
- Calculation: ROC = [(Close - Close n periods ago) / (Close n periods ago)] * 100%
- Recommendation:
    - Prices are expected rising as long as the Rate-of-Change remains positive (Buy)
    - Prices are expected falling when the Rate-of-Change is negative (Sell or Hold)

**Moving Average Convergence Divergence (MACD)**

- Definition: It turns two trend-following indicators, moving averages, into a momentum oscillator by subtracting the longer moving average from the shorter moving average. As a result, the MACD offers the best of both worlds: trend following and momentum.
- Calculation: Standard MACD Line: (12-day exponential moving average (EMA) minus a 26-day EMA)
- Recommendation: Potential buy signals occur when the MACD moves above zero and potential sell signals when it crosses below zero

## 8.2 Data Structures

The only data structure used in this system is the built-in data structure provided by the python package Pandas called DataFrame. The DataFrame is a tabular, spreadsheet-like data structure. It contains an ordered collection of columns, each of which can be a different type. This data structure was chosen since it has many built-in methods such as join and sort so; we did not to implement these functions. This saves us time and energy as well as allowing us to focus on creating the other parts of the system. Furthermore, this data structure is very flexible to work with since it can take multiple types and columns can be added as needed. For instance, the data structure internally handles missing data points and automatically creates an index to work with. Individual rows and columns can be accessed similar to accessing arrays in other programming languages. This allows us to easily work with the data we get from Yahoo's Finance API.

# 9. User Interface Design and Implementation

Upon entering the site, the user is presented with the following interface. 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.

Each stock in our database will have their own individual pages. The page itself will be divided into four distinct section. The first section will show the historical data, second section will show real-time data , the third section will show predicted data and the fourth section shows indicators according to the selected indicators, it displays the graph. The graph will have an indicator of the current data and which points on the graph represent the stock prediction. The third section will be a column next to the graph. This is where you will see the stock ticker that identifies which stock you are currently viewing as well as the company name and a link to their website underneath it. Next to that ticker title will be a button that enables you to add that specific stock to your portfolio. Underneath this will be links to news articles about the stock. This way you can see relevant news information about the stock.

Upon selecting a stock, the user will be presented historical data, real-time data, predicted data. 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.

# 10. Design of Tests

### User Interface Testing

#### Search for Stock
This is an important concept in our domain analysis and it enables the user to search for a stock in our database. Once the stock is found the system will request the stock's prediction and load the stock page. The test for this use case is to primarily ensure that the search function is working and the input entered by the user is read by the system properly

#### Database Connection
This test will make sure that the database controller sends the request to the database properly. It will also make sure that the information that the database sends back is the same information that the controller requested.

### Stock Page

#### Load Page
This test will make sure that after the stock has been searched that the correct page is loaded. It will also make sure that the various components of the page is also loaded successfully. This will include the graphs.

#### Graph
This ensures that the stock's graph on the page is correct and loads properly. There will be two graphs, one will have a dependency on yahoo finance data to populate and the other will have data that will be populated from our database. We will be able to identify issues due to the fact that we have two graphs pulling data from different sources. This could be useful because if one graph's data isn't loading properly we can pinpoint it on that graphs data source which will aid in debugging.

### Web Pages

#### Load Page
Like the stock page, we will run a test for all the other pages on the website and make sure they load properly. These pages are Portfolio, Login, Registration, Logout, Home.

#### Prediction Algorithm Testing
The prediction algorithms will be tested using previous data. This has not been implemented yet in terms of actual testing, however manual testing has begun that

compares the result of the moving average based off of historical data. This testing can be automated using programs that will test for validity of both the algorithm itself as well as the validity of the code.

**Integration Testing**

Our integration testing will be done using big bang integration. Prior to compiling each individual sub part, we will conduct unit testing on individual pieces of software in order to validate methods and strategies so that we can eliminate logical errors and concentrate on technical errors that may occur as a result of integration. There aren't too many external dependencies (the only is yahoo! finance api) and everything is relatively modular since everything has its own purpose and is independent of other aspects of the website so pinpointing issues shouldn't be too much of an issue. We will design larger test cases once the program is complete that will allow the programmer to quickly identify any areas of concern, whether it be connection issues, algorithm issues or even API issues just as a few examples.

# 11. History of Work

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.

**Database and Data Collection Progress (Ze Liu and Tong Wu )**

At the start of the project, it was imperative to import the stock data from Alpha Vantage API to collect raw data from yahoo finance and put it in a database. Yahoo Finance was used because it held all the stock information that was needed. Ze Liu and Tong Wu worked together on importing the data and using SQL, they were able to pull the information and keep it in different tables. The goal for this to be accomplished by was February 25.

**Prediction Research and Algorithm Progress (Jay Borkar, Ze Liu and Xinyu Lyu)**

Another vital component of developing our product was having an accurate prediction model to forecast the stocks. Jay Borkar, Ze Liu and Xinyu Lyu studied various prediction models The algorithms were implemented by March 5 and testing began shortly thereafter. The results of these algorithms were displayed on the webpage for the stock selected. We implemented the Artificial Neural Network, Long short-term model (RNN), Support Vector Machine, and Bayesian Curve fitting. They also had to integrate the database with the prediction models to store the results. The algorithms were implemented by March 30 and testing began shortly thereafter. This was all worked on from March 15 to April 1.

**Front-End Web Development Progress(Xinwei Zhao,Tong Wu, Ze Liu, Xinyu Lyu)**

The actual product that a user would see, is the website, was developed by Xinwei Zhao, Tong Wu, Ze Liu, Xinyu Lyu created the Git repository. They also came up with the overall design of the website for the on-screen requirements and the wireframe by March 20. Then they began to start coding the website. For the first demo, Vinay and Krisha created mockups of what the website would look like, and created charts that would show what our website is supposed to look like. The final step was integrating the website with the database. They also integrated the prediction models into the website. This was all worked on from March 25 to April 7.

This software project helped us research and implement various prediction models such as ANN, LSTM, SVM and work on web development languages as well as learn knowledge within stock trading. Besides technical factor's such as these, we also have benefited from the experience of being in a team oriented environment.

## 11.1 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. Final details may include finishing touches such as FAQ page, terms of service, contact page, and various other low priority (yet necessary) web pages.

# 12. 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
   https://en.wikipedia.org/wiki/Stock_market_prediction

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

4. Understanding LSTM Networks
   http://colah.github.io/posts/2015-08-Understanding-LSTMs/

5. LSTM
   https://en.wikipedia.org/wiki/Long_short-term_memory

6. Support Vector Machines
   https://en.wikipedia.org/wiki/Support_vector_machine

7. Artificial Neural Network
   https://en.wikipedia.org/wiki/Artificial_neural_network

8. Pattern Recognition and Machine Learning by Christopher M. Bishop

9. Cover page image : http://faganasset.com/2015/04/16/if-not-the-stock-market-then-where/