STOCK GURU

Group #10

By

Aneesh Abhyankar – aneesh.abhyankar@rutgers.edu

Aaditya Shukla – aaditya.shukla@rutgers.edu

Megha Katwala – megha.katwala@rutgers.edu

Shreyas Bhandare – shreyas.bhandare@rutgers.edu

Vishalsingh Hajeri – vishalsingh.hajeri@rutgers.edu

Rutgers, The State University of New Jersey
Department of Electrical and Computer Engineering

TABLE OF CONTENTS

Breakdown of Individual Contribution

All the members contributed equally.

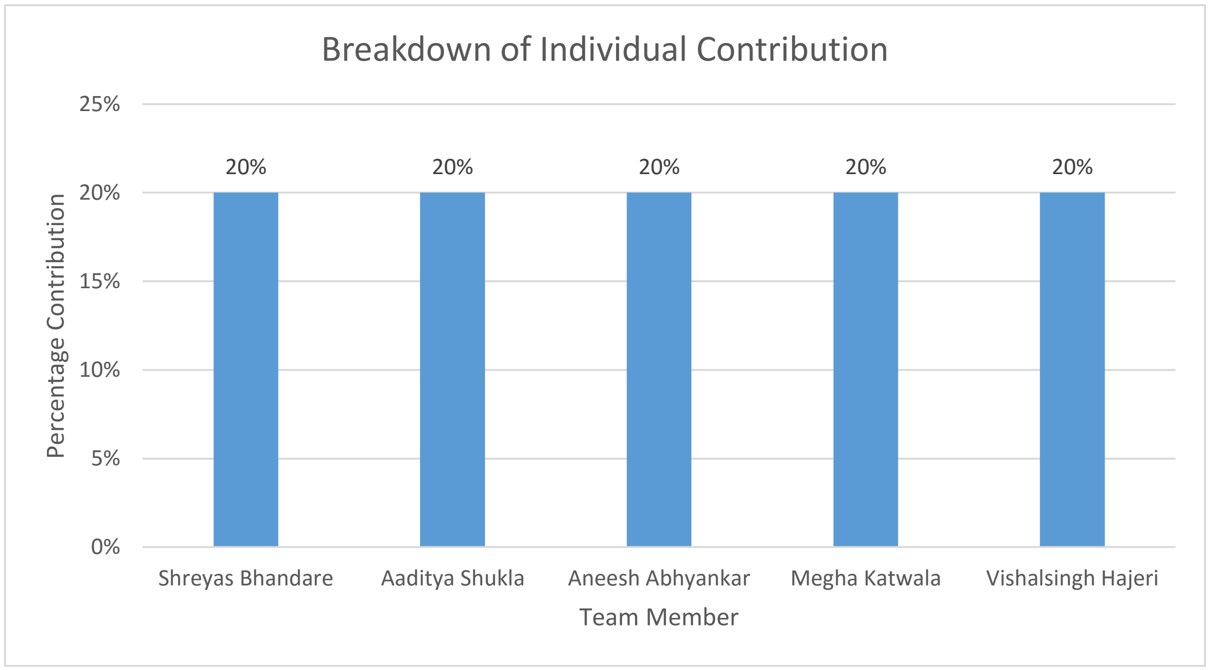


Figure 1 – Breakdown of Contribution

Summary of Changes

Changes made with respect to Phase-I proposal

- Adding more ticker symbols in database as per requirements.
- Prediction method addition – Support Vector Machine and Bayesian Curve fitting methods added. Earlier only Artificial Neural Network was being considered.
- Trends are shown easier to understand and navigation panel added.
- Web – service with API key are implemented.

Customer Statement Requirement

Stock Investment may seem to be a lucrative way of making quick money but without proper analysis of the stock being purchased, one might incur huge losses. Especially because of the erratic nature of stock price fluctuations it becomes extremely challenging to invest successfully and eventually earn profits from those investments. Most investors just do not have the time and resources to implement the lengthy analysis that takes place by full-time investors or employees of large investment organizations.

Professional investors and firms have an advantage; they have a team of investors to help with the research and spend their entire careers studying the markets. On the other hand, normal investors do not have such a luxury, many of whom work far away from the field of investments.

With this realization, we aim at building a Web API that will attempt to benefit a broad range of investors by providing them with stock predictions based on a sophisticated mathematical algorithm which uses the historical stock prices (up to 1 year) along with the most recent ones to forecast the future prices. We believe that such a tool can help them to make an informed decision on whether to buy, sell, or hold the stock.

Glossary of Terms

1. **Database**: A database is a means of storing information in such a way that information can be retrieved from it at a later stage.
2. **Bid**: A bid is an offer made by an investor, a trader or a dealer to buy a security. The bid will stipulate both the price at which the buyer is willing to purchase the security and the quantity to be purchased.
3. **Volume**: Volume is the number of shares or contracts traded in a security or an entire market during a given period of time. It is simply the amount of shares that trade hands from sellers to buyers as a measure of activity.
4. **Market capitalization**: It is the total dollar market value of all of a company's outstanding shares. Market capitalization is calculated by multiplying a company's shares outstanding by the current market price of one share. The investment community uses this figure to determine a company's size, as opposed to sales or total asset figures.
5. **Trading day**: It is the time span that a particular stock exchange is open. For example, the New York Stock Exchange is, as of 2015, open from 9:30 AM Eastern Time to 4:00 PM Eastern Time. Trading days are usually Monday to Friday.
6. **Closing price**: It is the final price at which a security is traded on a given trading day. The closing price represents the most up-to-date valuation of a security until trading commences again on the next trading day.
7. **SVM:** Support Vector Machine algorithm for prediction.
8. **ANN**: Artificial Neural Network Algorithm
9. **Bayesian Curve Fitting**: A prediction algorithm.
10. **YAHOO API**: an API provided by Yahoo Finance API to have access to yahoo finance's database of stock quotes.
11. **Admin** – Personnel who initiates data collection and prediction events on the request from user.
12. **Bollinger Band**: They are a volatility indicator similar to the Keltner channel. Bollinger Bands consist of: an N-period moving average (MA) an upper band at K times an N-period standard deviation above the moving average (MA + Kσ) a lower band at K times an N-period standard deviation below the moving average (MA − Kσ).
13. **SMA: Simple Moving average -** A simple moving average (SMA) is a simple, or arithmetic, moving average that is calculated by adding the closing price of the security for a number of time periods and then dividing this total by the number of time periods.

Functional Requirement

Identifier	Requirement	Priority
REQ1	The system should allow a user to acquire stock information by providing the company symbol.	4
REQ2	The system should continuously acquire most recent stock data (Stock value, trading volume etc.) for a set number of companies.	5
REQ3	Stock data for a company of up to 1 year should be retrieved from a Finance API and saved in the database.	5
REQ4	Real Time data should be retrieved from a Finance API and stored in a database.	5
REQ5	The time slice between two real-time points should be no more than 1 minute and the database should be continuously updated.	4
REQ6	The system should be able to analyze the data stored in the database in an attempt to accurately predict the stock price in the near future using machine learning algorithms.	5
REQ7	The system should be able to spot patterns in the graphs and suggest user whether to hold, sell or buy stock	4

Table 1 – Functional Requirements

Non-Functional Requirements

Identifier	Requirement	Priority
REQ1	System will display past stock value trends	4
REQ2	System will display market news from different web sources	3
REQ3	The system will be well documented via reports and possibly and API Documentation.	4
REQ4	The system will be implemented in Java, and will connect to a database that will also be implemented in Java.	5
REQ5	The database will be updated daily to account for the daily changes to stock values.	4
REQ6	Site will be simple and clean so that the customer can view everything easily.	4

Table 2 – Non-Functional Requirements

On-Screen Requirements

Identifier	Description	Priority
REQ1	Home Screen will display selected ticker current information	4
REQ2	There will be very less clicks for user to find necessary information	3
REQ3	Website will be aesthetically pleasant to watch and operate	4
REQ4	On-screen Graphs will have value navigator	2

Table 3 – On-Screen Requirements

Use Case - Casual Description

No.	Title	Description
1	View Ticker Information	Allows a user to select a ticker from drop down menu and displays the corresponding information on screen.
2	Edit/Update Time	Allows the admin to alter the rate at which the current stock prices are retrieved from the yahoo API
3	Data Collection	In order to generate a prediction, we need to retrieve stock and market data from the Yahoo! Finance API and store it into the database. This is done in the background.
4	Obtain and Display Prediction	Predicts the stock value based on the results of prediction algorithm which gets its input parameters from the database.
5	Suggest	Allows the user to request the system to suggest a stock that is predicted to have the highest gain
6	View Trends and Indicators	Display historical trends and Indicators
7	View Market Quotes	Display latest market news and feeds

Table 4 – Use Case Casual Description

Use Case – Fully Dressed Description

Fully dressed description is given for important use cases.

1) Use Case – 3 Data Collection

Related Requirements: REQ2, REQ3, REQ4, REQ5

Actor's Goal: To retrieve data from YAHOO API and store into Database

Participating Actors: Database, System, Admin, YAHOO API

Preconditions: Schemas are created in the database and internet connection is active, Eclipse IDE is set-up.

Successful End Conditions: Data is stored in the database.

Failed End Conditions: Data is not stored in the database.

Flow of Events for Main Scenario:

- Java program is set-up in Eclipse IDE
- Schemas are created in the database
- Admin initiates data request procedure to Yahoo API and parses the specific data
- Parsed data is stored in the database.

Flow of Events for Alternate Scenario:

- Java program is set-up in Eclipse IDE
- Schemas are created in the database
- Admin initiates data request procedure to Yahoo API. But data is not received
- No data is stored in the database

2) Use Case – 4 Obtain and Display Prediction

Related Requirements: REQ2, REQ3, REQ4, REQ5, REQ6

Actor's Goal: To predict data stored into Database for particular symbol for particular time interval

Participating Actors: Database, System, Admin, User

Preconditions: Database is populated and internet connection is active, Eclipse IDE is set-up.

Successful End Conditions: Data is predicted and stored in the database and displayed on screen

Failed End Conditions: Data is not predicted and/or not stored in the database.

Flow of Events for Main Scenario:

- Java program is set-up in Eclipse IDE
- Database is populated
- Admin initiates prediction procedure based on user's request

- Data is predicted and stored in database and displayed on screen

Flow of Events for Alternate Scenario:

- Java program is set-up in Eclipse IDE
- Database is populated
- Admin initiates prediction procedure based on user's request
- Data is not predicted and/or not stored in database and displayed on screen

Use Case – Traceability Matrix

Requirements	Priority	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7
REQ1	4	X						X
REQ2	5	X		X	X		X	
REQ3	5		X	X	X		X	
REQ4	5			X	X		X	
REQ5	4		X	X	X		X	X
REQ6	5				X	X		
REQ7	4					X		X

Table 5 – Traceability Matrix

System Sequence Diagram: Implemented Use case: Data Collection

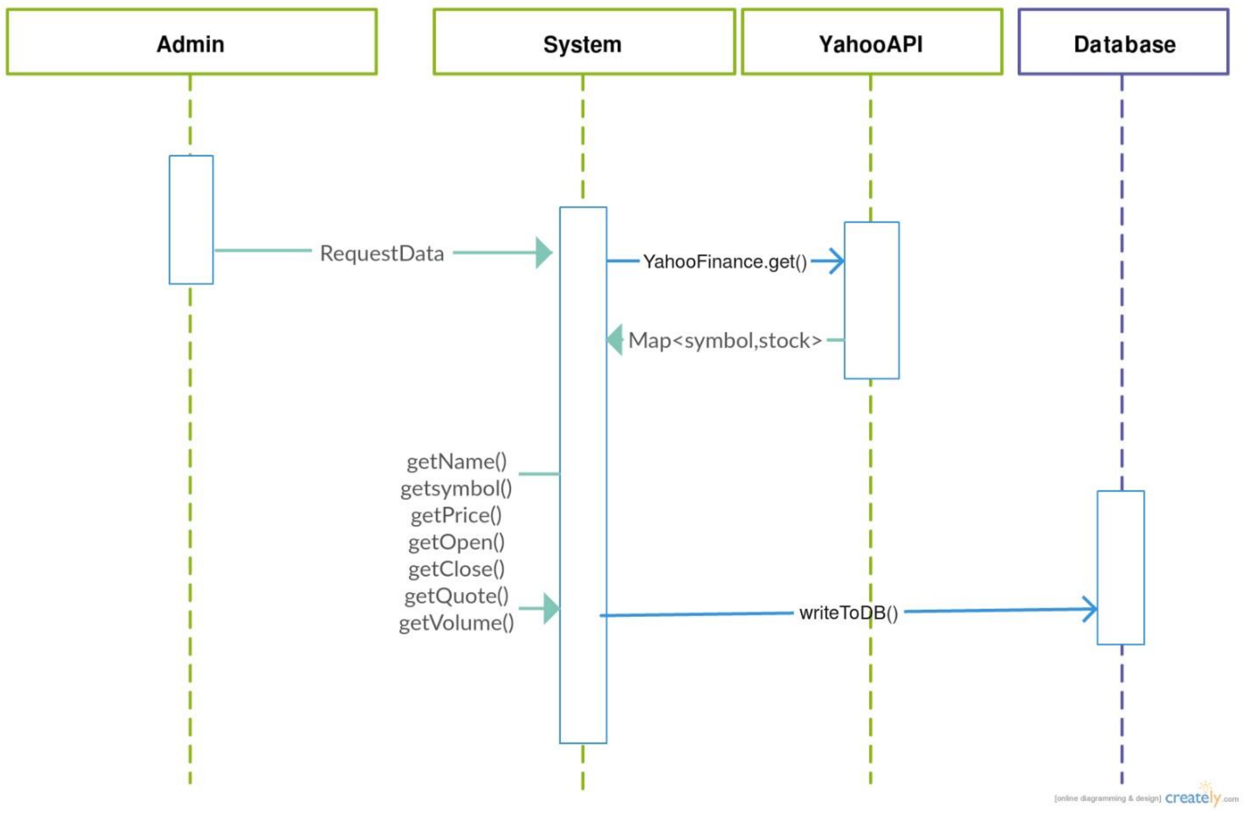


Figure 2 – System Sequence Diagram for Data Collection

Three type of Stock data is collected from YAHOO finance API.

1) Historical daily data – daily closing values for stock symbol for last one year.
2) Real time data – The script that runs on current day and stores real time current stock value data for that day.
3) Historical daily minute wise data – The script runs every day after data collection started and stores daily minute wise current stock value data so that user can see the daily trend for particular stock.

Admin will initiate the process of data collection by running the data collection script either manually or with automated timer. The script declares object of YahooFinance class and sends data request to the system. The system will call the YahooAPI and YahooAPI will return Map with keys and values of stock symbols and its attributes. This data can be accessed from the system using getName(), getSymbol(), getPrice(), getOpen() etc. function. After retrieving this data, it is stored in database using SQL query.

System Sequence Diagram: Implemented Use case: Obtain & Display Prediction

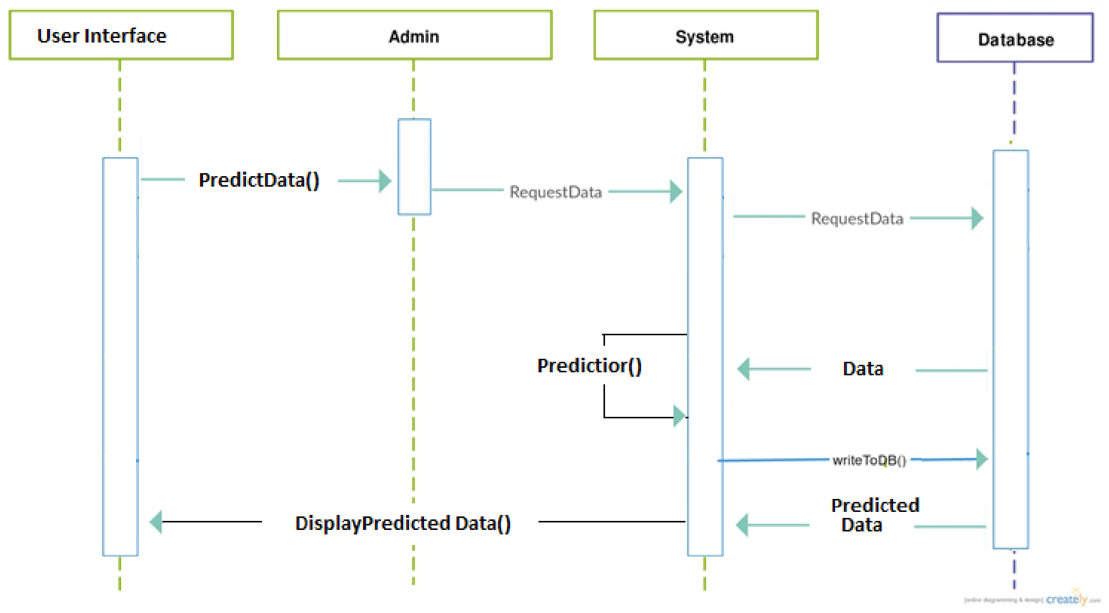


Figure 3 – System Sequence Diagram for Obtaining prediction

1) User request for short term/long term prediction from the system.
2) Admin initiates the process by collecting data from database.
3) Short term and long term predictions (1 day, 5 day and 30 day) predictions are calculated using prediction algorithms.
4) Predicted data is again stored in the database and displayed on the screen.

User Interface Specification

Preliminary Design Mock-Ups are shown below.

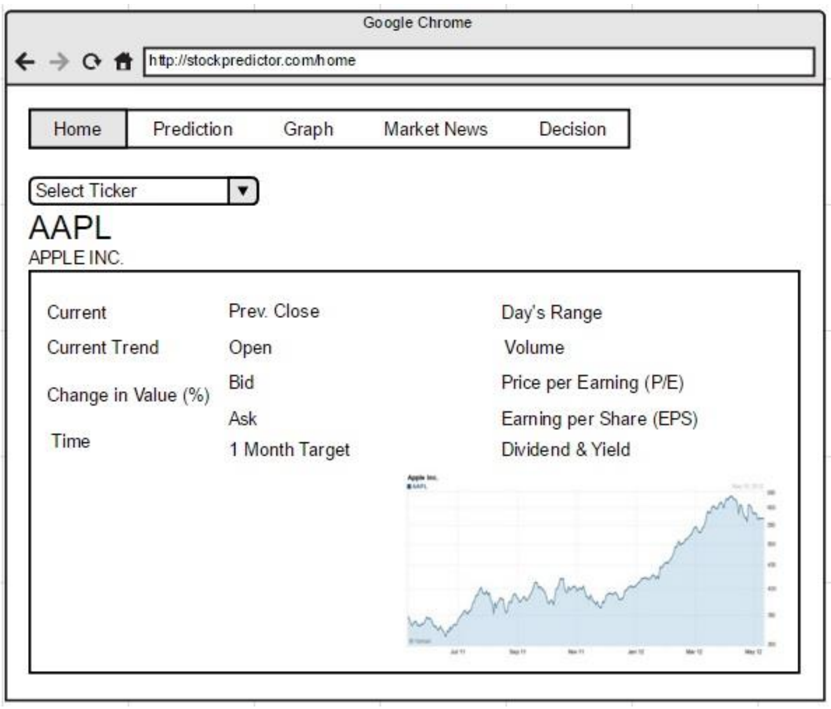


Figure – 4

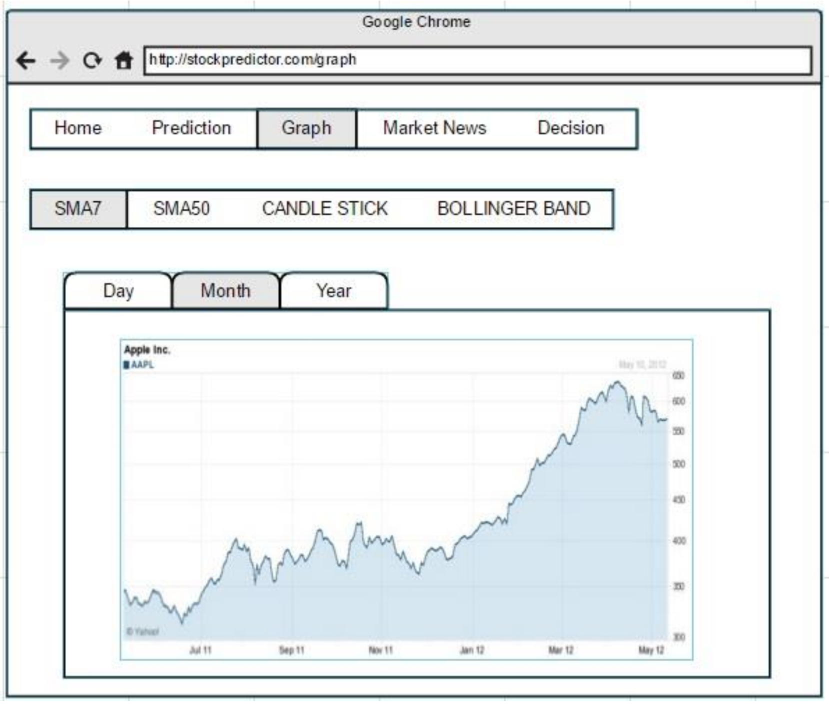
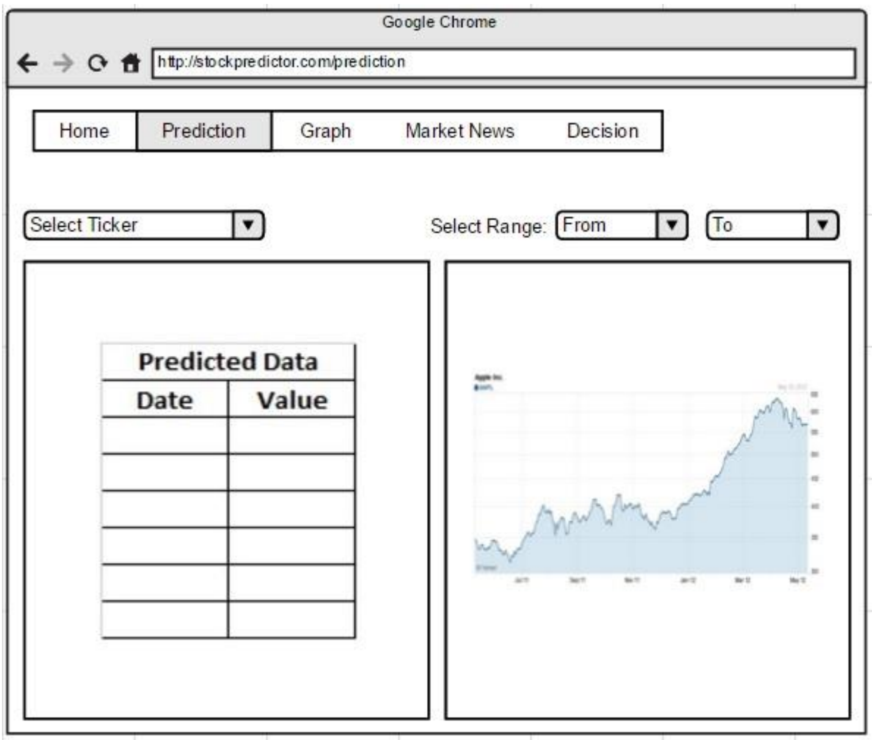
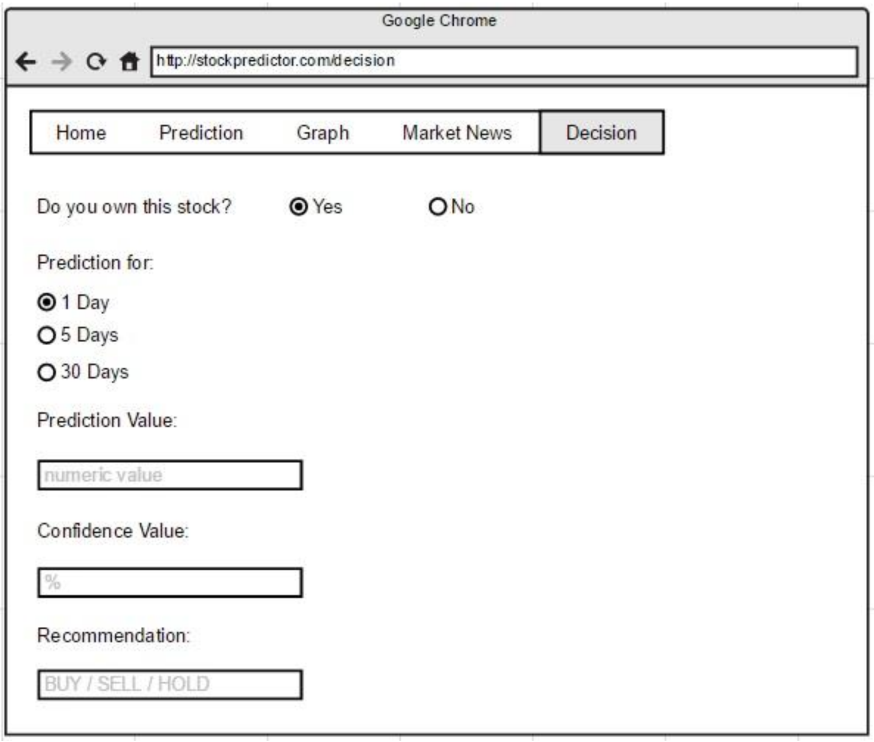


Figure - 5



Predicted Data	
Date	**Value**

Figure – 6



Do you own this stock? [✓ Yes] No

Prediction for:

[✓ 1 Day]
5 Days
30 Days

Prediction Value: numeric value

Confidence Value: %

Recommendation: BUY / SELL / HOLD

Figure - 7

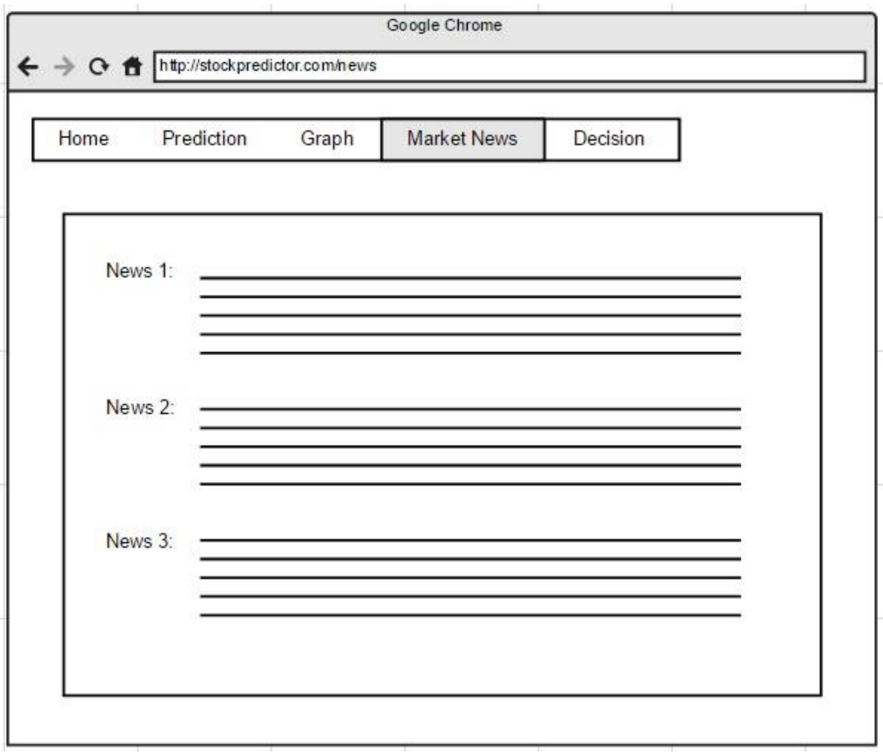

Google Chrome
http://stockpredictor.com/news
Home
Prediction
Graph
Market News
Decision
News 1:
News 2:
News 3:

Figure - 8

Effort Estimation

Actor name	Description of relevant characteristics	Complexity	Weight
User	User is interacting with the system via a graphical user interface	Complex	3
Admin	The actor is a person interacting via a graphical user interface.	Complex	3
System	System performs operations on the getting data from database, processing it and hosting it on a network.	Simple	1
Database	Database is another system interacting through a protocol.	Average	2
YAHOO API	API is another system which interacts with our system through a defined application programming interface	Simple	1

UAW (home access) = 2 X simple + 2 X average + 2 X complex = 2 X 1 + 1 X 2 + 2 X 3 = 10

Use case	Description	Category	Weight
View Ticker Information	Simple user interface. 3 steps for the main success scenario. 3 participating actors (User, System, Database)	Average	10
Edit/Update Time	No user interface. 3 participating actors (Admin, Database, YAHOO API)	Simple	5
Data Collection	No user interface. 3 participating actors (Admin, Database, YAHOO API)	Average	10
Obtain and Display Prediction	Good user interface. 4 participating actors (Admin, Database, YAHOO API, System)	Complex	15
Suggest	Moderate User Interface. 3 participating actors (Admin, Database, System)	Complex	15
View Trends and Indicators	Good User Interface. 4 participating actors (User, Admin, Database, System)	Average	10
View Market Quotes	Good User Interface. 4 participating actors (User, Admin, Database, System)	Simple	5

UUCW (home access) = 2 X simple + 5 X average + 2 X complex = 2 X 5 + 3 X 10 + 2 X 15 = 70

UUCP = UAW + UUCW = 10 + 70 = 80

Technical factor	**Description**	**Weight**	**Perceived Complexity**	**Calculated Factor (Weight * Perceived Complexity)**
T1	Distributed system	2	3	2*3 = 6
T2	Users expect good performance	1	3	1*3 = 3
T3	End-user expects efficiency	1	3	1*3 = 3
T4	Internal processing is relatively complex	1	3	1*3 = 3
T5	No requirement for reusability	1	0	1*0 = 0
T6	Extremely easy to install	0.5	1	0.5*1 = 0.5
T7	Ease of use is very important	0.5	5	0.5*5 = 2.5
T8	Extremely Portable	2	5	2*5 = 10
T9	Easy to change minimally required	1	1	1*1 = 1
T10	Concurrent use is required	1	4	1*5 = 5
T11	Security is not a very significant concern	1	1	1*1 = 1
T12	No direct access for third parties	1	0	1*0 = 0
T13	No unique training needs	1	0	1*0 = 0
Technical Factor Total:				35

TFC = C1 + C2 * TFT = 0.6 + 0.01 * 35 = 0.95

Environmental factor	**Description**	**Weight**	**Perceived Impact**	**Calculated Factor (Weight * Perceived Impact)**
E1	Beginner familiarity with the UML-based development	1.5	1	1.5*1 = 1.5
E2	Some familiarity with application problem	0.5	2	0.5*2 = 1
E3	Decent knowledge of object-oriented approach	1	3	1*3 = 3
E4	Beginner lead analyst	0.5	1	0.5*1 = 0.5
E5	Highly motivated	1	5	1*4 = 5
E6	Stable requirements expected	2	5	2*5 = 5
E7	No part-time staff will be involved	-1	0	-1*0 = 0
E8	Programming language of average difficulty will be used	-1	3	-1*3 = -3
Environmental Factor Total:				13

ECF = C1 + C2 * EFT = 1.4 + -0.03 * 13 = 1.01

UCP = UUCP * TCF * ECF = 80 * 0.95 * 1.01 = 77.76 or 78

Use Case Points Duration = UCP * PF = 78 * 28 = 2184 hours.

Domain Analysis

Concept Definitions:

Responsibility Description	**Type**	**Concept Name**
Searches the database for the requested stock the user inputted.	D	StockSearcher
Pulls relevant information for the stock after a search is conducted.	D	DataPuller
Uses the neural networking, SVM and Bayesian Prediction models to determine an accurate prediction for a stock.	D	StockPredictor
Updates the user interface based upon user input or database updates.	D	InterfaceUpdater
Grabs market news from different websites and RSS feeds and displays to user	D	RSSPoster
Grabs current stock data from the API.	D	CurrentDataGrabber

Association Descriptions:

Concept Pair	**Association Description**	**Association Name**
StockSearcher↔DataPuller	StockSearcher conveys stock symbol to DataPuller	conveys-symbol conveys-data
StockPredictor↔InterfaceUpdater	StockPredictor conveys predicted stock price to InterfaceUpdater and InterfaceUpdater updates the UI	updates
CurrentDataGrabber↔DataPuller	CurrentDataGrabber updates DataPuller with the new stock data and DataPuller stores that information for access later	conveys-stockinfo, stores
RSSPoster↔ InterfaceUpdater	RSSPoster grabs the market news from websites and InterfaceUpdater updates the UI	Diplays-news
DataPuller↔ InterfaceUpdater	DataPuller pulls the data from database and InterfaceUpdater shows Ticker Information and Trends as well as indicators on screen	Display-Trends/Information

Attribution Definitions

Responsibility	**Attribute**	**Concept**
String of the given stock ticker symbol	tickerSymbol	StockSearcher
Check if the given ticker symbol is in the database	isValidTicker	DataPuller
Boolean value to choose whether to display top/bottom 5 by percentage or magnitude	dispType	InterfaceUpdater
List that contains all the stock tickers to query the API with to get current/historical stock data.	stockList	CurrentDataGrabber

Traceability Matrix

	UC1	UC2	UC3	UC4	UC5	UC6	UC7
StockSearcher				X		X	
DataPuller	X			X		X	
StockPredictor				X	X		
InterfaceUpdater	X			X	X	X	
RSSPoster							X
CurrentDataGrabber	X	X	X				X
Max Priority	3	4	5	5	4	4	3
Priority Weight	9	4	5	20	8	12	6

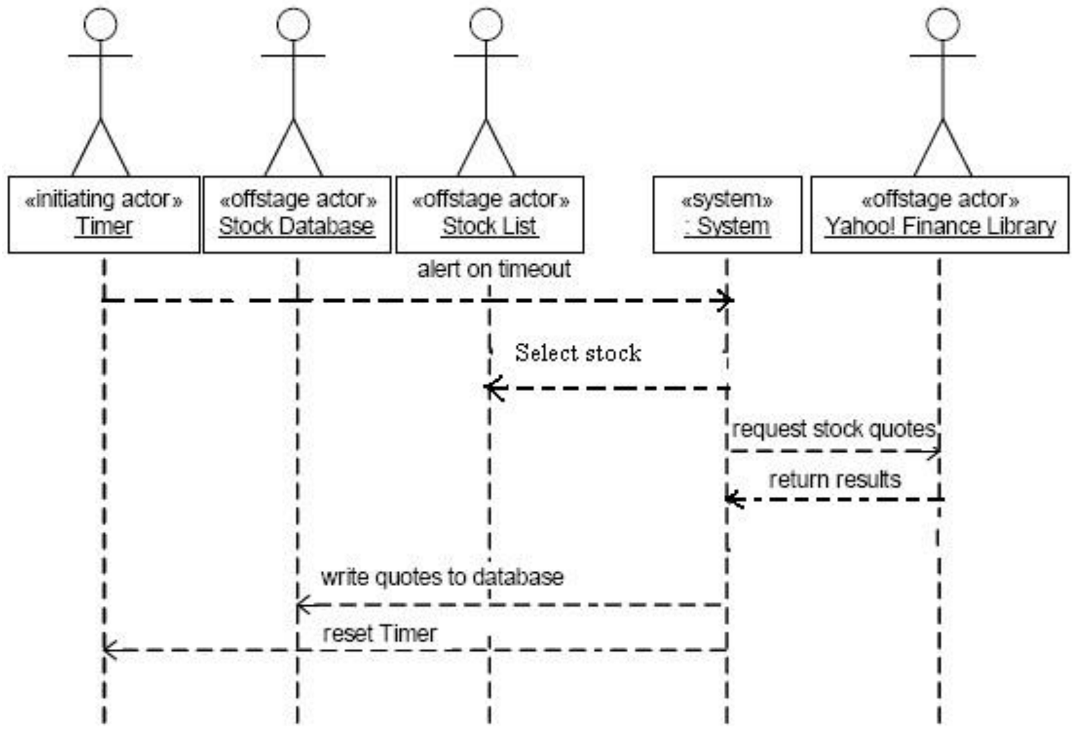
System Operation Contracts

1) **Operation Contract – Search Function**
Name: Search
Responsibilities: Takes users inputted symbol and match it to the database, and retrieve the data for that stock or user.
Exceptions: If the data does not exist in the database
Preconditions: Server has not stored data for the particular symbol
Post conditions: A stocks prediction was created and displayed.

2) **Operation Contract – Changing Tabs**
Name: Changing Tabs
Responsibilities: When users tap a tab the application switches to that page. Cross
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.

Interaction Diagrams

Use Case 3: Data Collection



- **Initiating Actor**: The initiating actor is the Timer, which is set in UC2 to initiate UC3 on regular basis.
- **Actor's Goal**: To gather information from YAHOO API and store the results in Stock Database on the local machine.
- **Participating actors**: Stock List, Stock Database, Yahoo! Finance Library
- **Preconditions**: Timer Interval has been set, and the Stock List has been defined.

  The Stock Database exists and implements the expected schema.
- **Trigger**: The Timer times out.
- **Post conditions**: The Stock Database is updated. The Timer is reset. The System Menu is shown.
- **Main Success Scenario:**

→ 1. Timer initiates Stock Database update. ← 2. System reads Stock List.

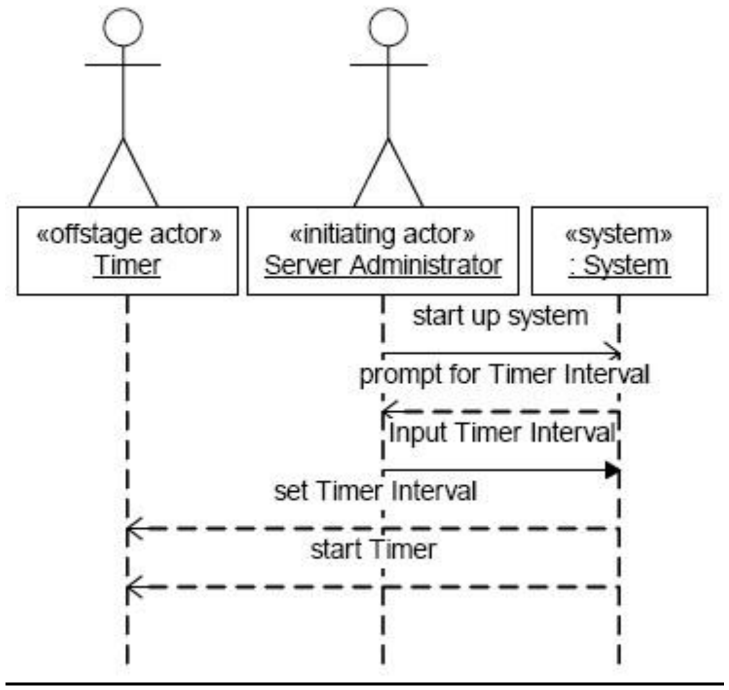
← 3. System calls Yahoo! Finance Library.

← 4. System copies the information returned by the Yahoo! Finance Library into the Stock Database.

← 5. System resets Timer.

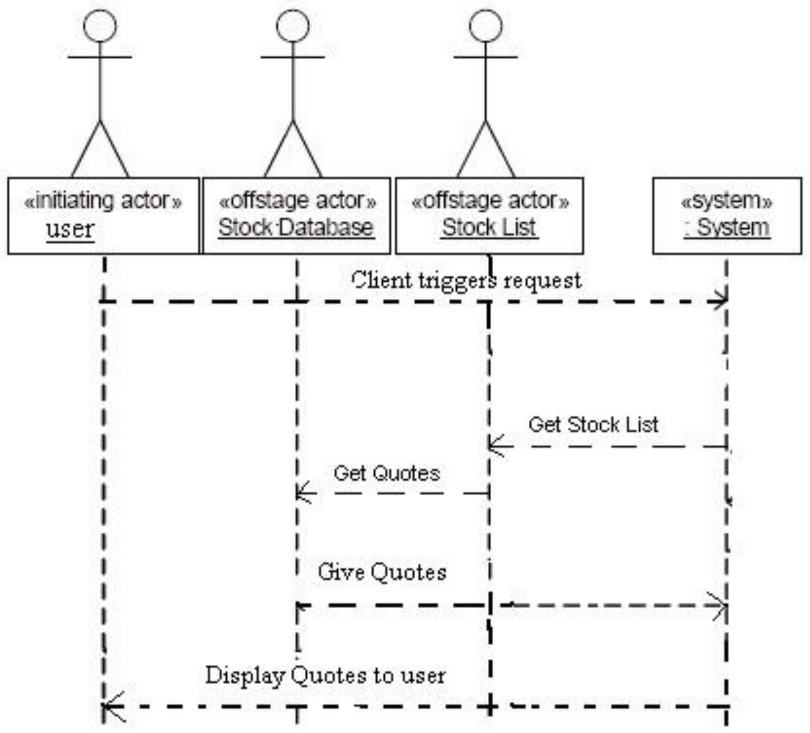
← 6. System returns to System Menu.

Use Case 2: Set Timer Interval



- **Initiating Actor**: Server Administrator
- **Actor's Goal**: Get the system to regularly update the Stock Database •
- **Participating Actors**: Timer
- **Preconditions**: None.
- **Trigger**: The Server Administrator is started.
- **Post conditions**: The Timer Interval has been set and the Timer has started counting down to the next Database update. The System Menu is shown.

Use Case 1: Display stock Information to client



- **Initiating Actor**: The initiating actor is the User, the Web Client.
- **Actor's Goal**: To gather stock quote information from stock database and display the results at the client end on the local machine.
- **Participating actors**: Client, Stock List, Stock Database.
- **Preconditions**: Stock List has been defined. The Stock Database exists and implements the expected schema.
- **Trigger**: The Client requests stock quotes.
- **Post conditions**: The Stock quote is displayed on the client's browser window.

  The System Menu is shown.
- **Main Success Scenario:**

  → 1. Client initiates Stock quotes request.

  ← 2. System reads Stock List.

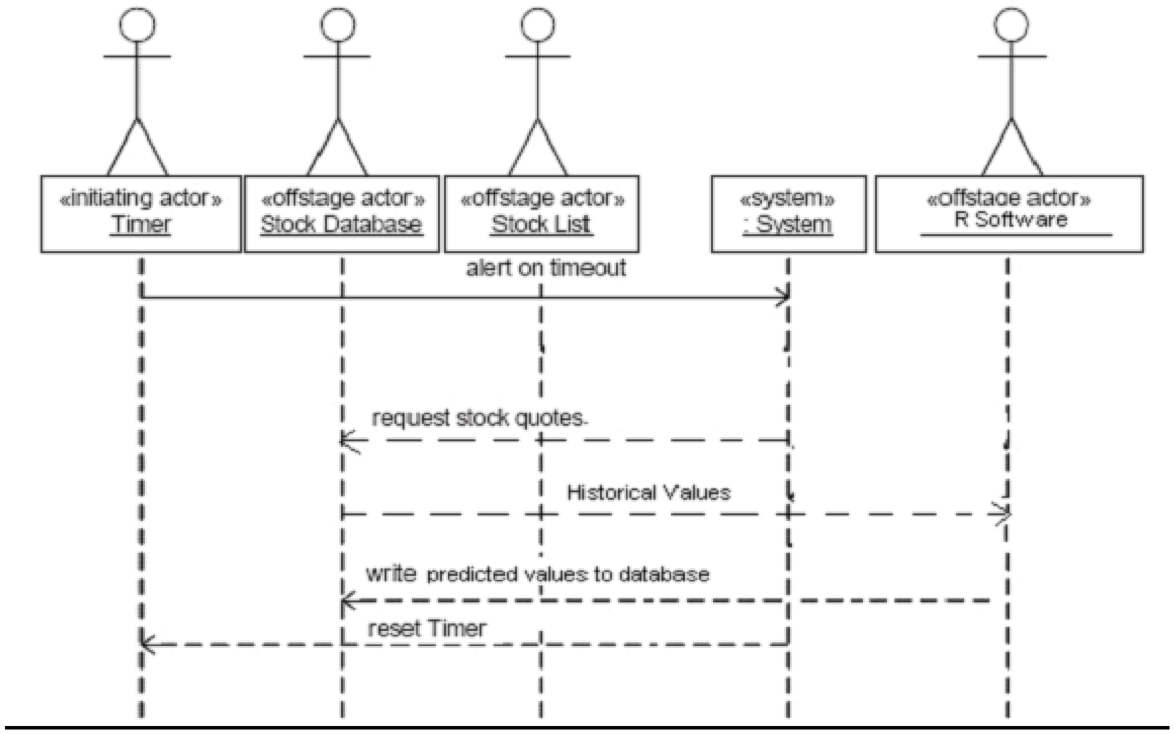
  ← 3. System calls Stock Database.

  ← 4. The System gets the Stock quotes.

  ← 5. The Stock quotes are displayed for the client.

  ← 6. System returns to System Menu.

Use Case 4: Generate predicted stock values and store it in Stock Database.


«initiating actor» Timer
«offstage actor» Stock Database
«offstage actor» Stock List
«system» : System
«offstage actor» R Software
alert on timeout
request stock quotes
Historical Values
write predicted values to database
reset Timer

- **Initiating Actor**: Timer, which is set for one day.
- **Actor's Goal**: To generate predicted stock quotes and store it in the Stock Database.
- **Participating actors**: Stock List, Stock Database, R.
- **Preconditions**: The Stock List has been defined. The Stock Database exists, implements the expected schema and R exists.
- **Trigger**: At the end of the day, the server administrator is started.
- **Post conditions**: The predicted values are stored in the Predicted values table.

  The Timer is reset.
- **Main Success Scenario:**

  → 1. The timer triggers the system administrator at the end of the day.

  ← 2. System calls Stock Database.

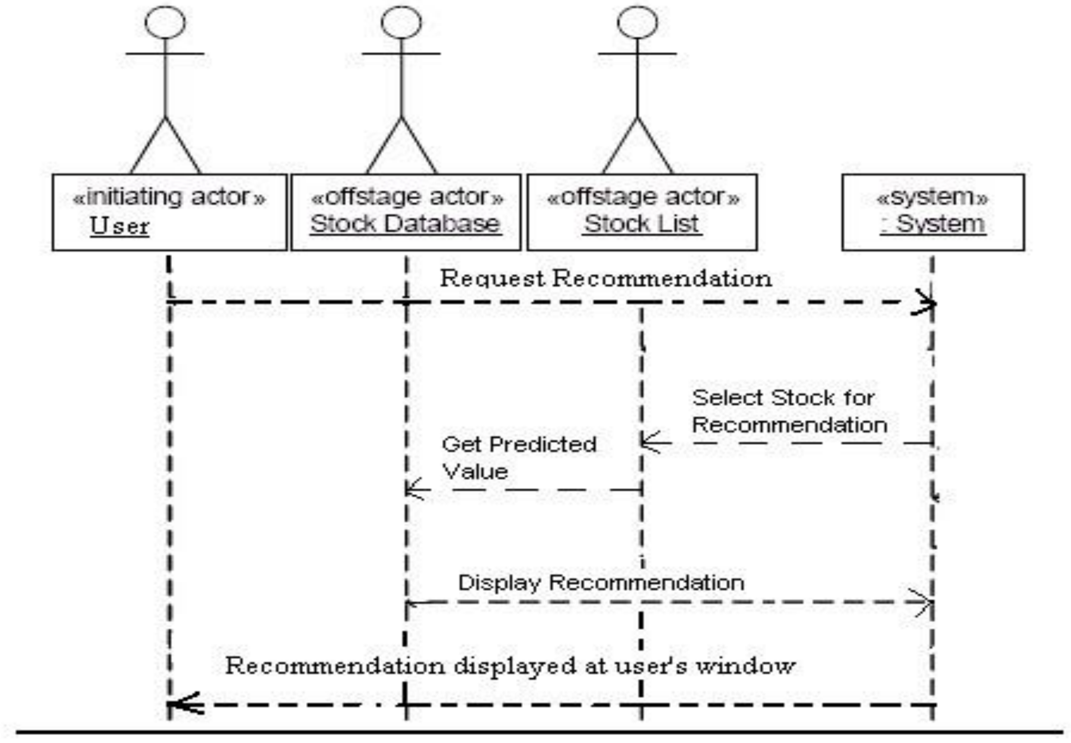
  ← 3. Stock quotes are given to R

  ← 4. The R generates the Predicted values

  ← 5. Predicted values are written to the Stock Database

  ← 6. System resets Timer.

Use Case 5: Suggestion



- **Initiating Actor**: The initiating actor is the Client, requesting for recommendation.
- **Actor's Goal**: To get recommendation for the selected stock and display on local machine.
- **Participating actors**: User, Stock List, Stock Database and Predicted Values table.
- **Preconditions**: Stock List has been defined. The Stock Database exists and implements the expected schema. Predicted values have been generated by R and stored in Predicted values table.
- **Post conditions**: Recommendations are generated for the requested stock. The recommendations are displayed at client's local machine. The System Menu is shown.
- **Main Success Scenario:**

→ 1. User initiates recommendation request by selecting the stock.

← 2. System reads Stock List.

← 3. System calls for Recommendation

← 4. System copies the information returned by R, stored in the Prediction values table.

← 5. Recommendation is displayed at the client's local machine.

← 6. System returns to System Menu.

Class Diagrams

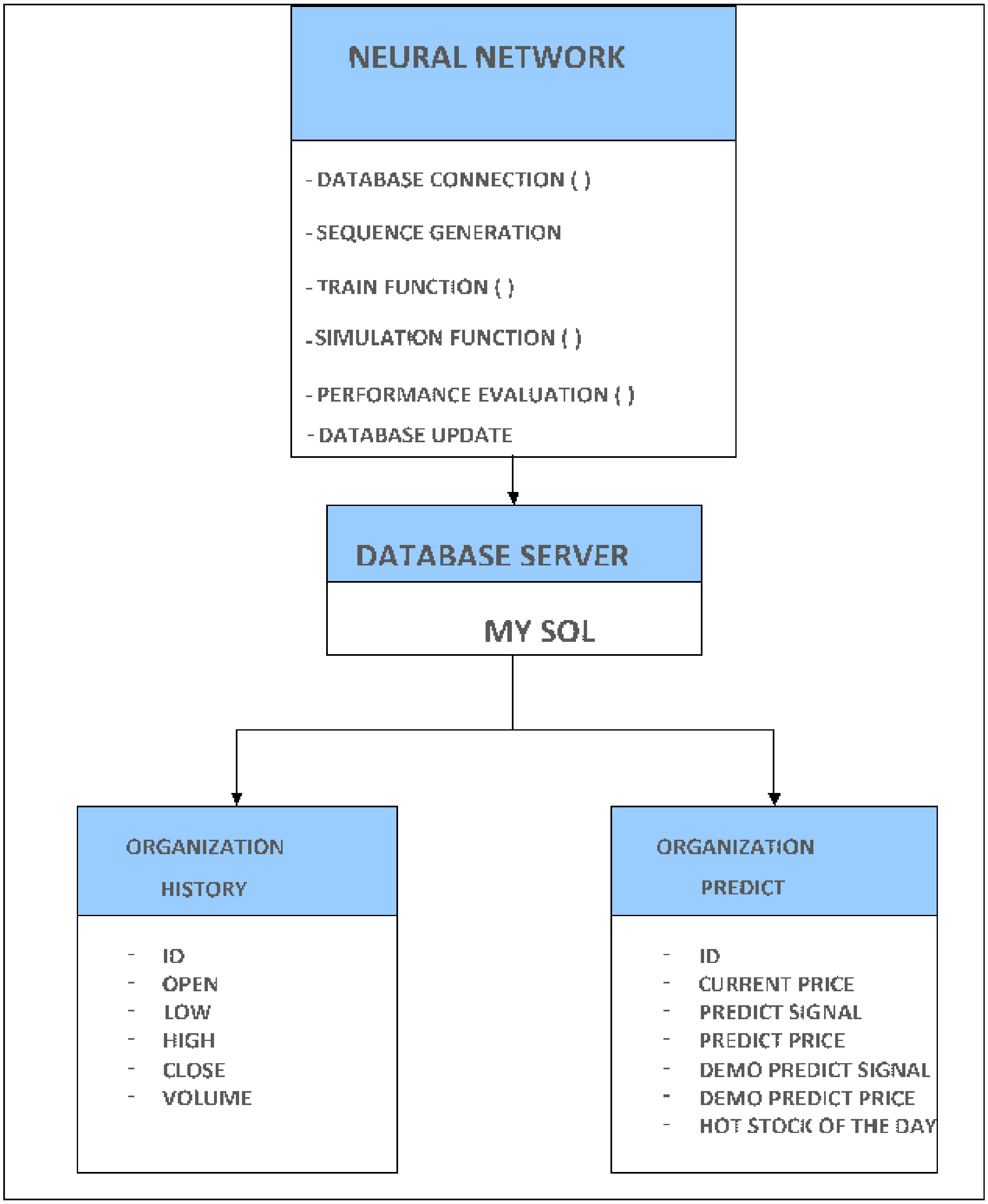
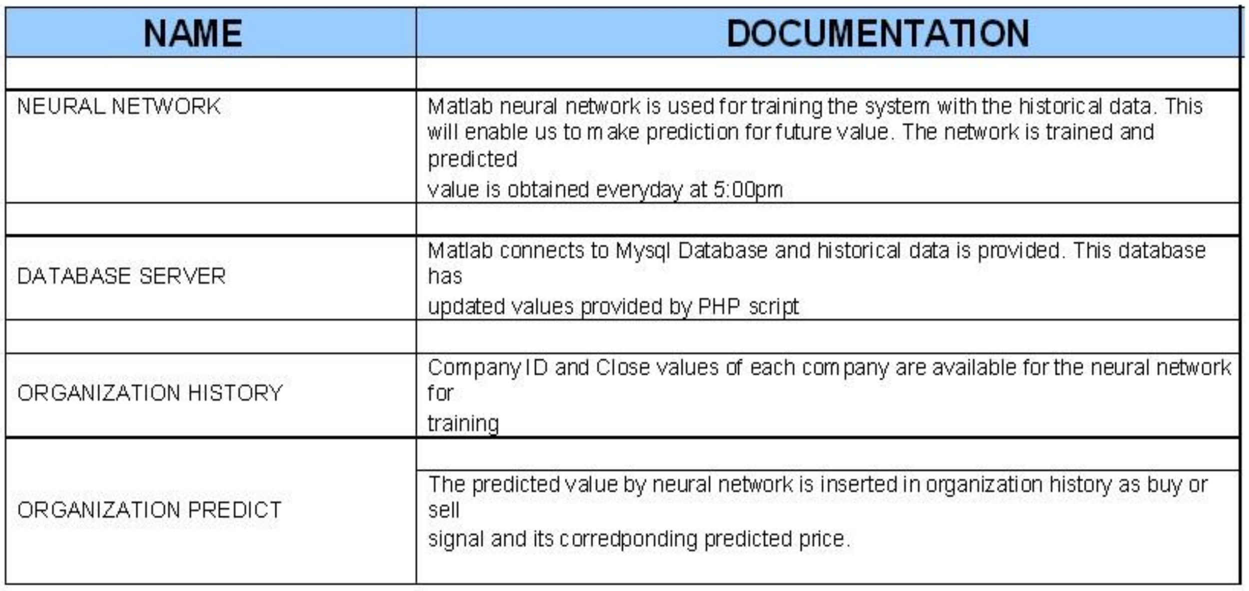


Fig 9 Class Diagram

Class Diagram Description:



NAME	DOCUMENTATION
NEURAL NETWORK	Matlab neural network is used for training the system with the historical data. This will enable us to m ake prediction for future value. The network is trained and predicted value is obtained everyday at 5:00pm
DATABASE SERVER	Matlab connects to Mysql Database and historical data is provided. This database has updated values provided by PHP script
ORGANIZATION HISTORY	Company ID and Close values of each com pany are available for the neural network for training
ORGANIZATION PREDICT	
	The predicted value by neural network is inserted in organization history as buy or sell signal and its corredponding predicted price.

System Architecture and System Design Architectural Styles

System Architecture and System Design Architectural Styles Structure:

Component-Based, Object-Oriented, Layered Architecture

The structure uses three architectural styles. The first, component-based, splits the application into reusable functional or logical components. The second, object oriented, is based on the division of the system into individual objects that contains data relevant only to the object, and is reusable. The third and final style, layered architecture, partitions the system into stacked groups or layers which allows for reusing of the layers elsewhere because there is no dependency between layers.

Domain: Domain Driven Design This is an object-oriented style that focuses on the business domain. This allows us define business aspects based on the business domain.

Deployment: Client/Server, N-Tier, 3-Tier When dealing with deployment there are three styles that are used. The first, client/server, separates the system into two applications in which the client can make requests to the server. The second and third, n-tier and 3-tier, are similar to layered style however each layer or tier in this case is located on a physically separate computer.

Communication: Service-Oriented Architecture (SOA), Message Bus The communication aspects of this application use the following architectural styles. The first, service-oriented architecture uses a system of contracts and messages to communicate between the client and the application. The second, message bus, allows the software system to send and receive message on multiple channels. This then allows the applications to interact without having to know any specific information about each other.

Identifying subsystems essentially the subsystems are as follows: Database, Prediction, User Database, Network Communication Server, Network Connection Client User Database, User Interface, and Social Network. Essentially what happens is that whenever a user wants to search a stock, the program hits the database. From here a prediction is made in regards to what the stock's price will be on the next day. The subsystems involved here would be the Prediction Calculator and the API Caller. The Network communication subsystems allow for connection to the server. The server is where the user database is, as well as the social network component.

Mapping Subsystems

Persistent Data Storage Data will need to be saved so that it can outlive a single execution of the system. Data this data includes a minimum of 6 months history for every stock, all data about all users, any post from any user ever posted, and all current predictions for stocks. If this data was lost the application would have to call to the API a few thousand times every time it starts up just to get data that it has already gotten before. This would be a huge waste of money and time. The social media of this application would cease to exist if all data was lost because there would never be any posts, nor would there even be users. A relational database will work well for what we need. We need to be able to store large amounts of data and be able to have relations between the data. For example, say a user has a favorite such as google. The will need to have a relation with all the data google has rather then re-saving the data for the user. For the projections it is extremely important that it has a relationship with all of the data from any stock as well as any data from the news feed. Allowing all of these relationships in our database will save time and storage space.

Network Protocol Initially we chose to use Keynote as our network protocol for this application. However, once we got our server set up and knew a little more about what we needed to do it was evident that there was no need to implement a full library. We decided it would be easier to just open TCP ports and have a direct connection.

Global Control Flow Our system is event-driven, and allows users to generate actions in different orders. Being an event-response type of system, the application doesn’t load any data until the user requests it. This allows the user to get right to what they wanted to do instead of having to go step-by-step through a process and waste time. Users can go into the application and can check out only the things they want to and exit without having to do the unnecessary steps.

Hardware Requirements the hardware requirements for the application are as follows: Screen size: This application will be fitted for the standard Android screen size; however will work with smaller or bigger screen sizes. Hard-drive space: The application will not require a lot of space on the mobile device; just about 10 Mb for the application to be installed and for certain data that needs to be stored on the device. Network connection: Minimum requirement is 3G connection which is about 0.4 Mbit/s. The hardware requirements for the server are as follows: Hard-drive space: The server will store all pertinent data in respect to the user and the Stocks, thus we will need a fair amount of room on the hard-drive about 500 Gb. RAM: 43 Minimum 4 Gb. Network connection: Minimum requirement is broadband connection which can range from 384 kbit/s to over 20 Mbit/s.

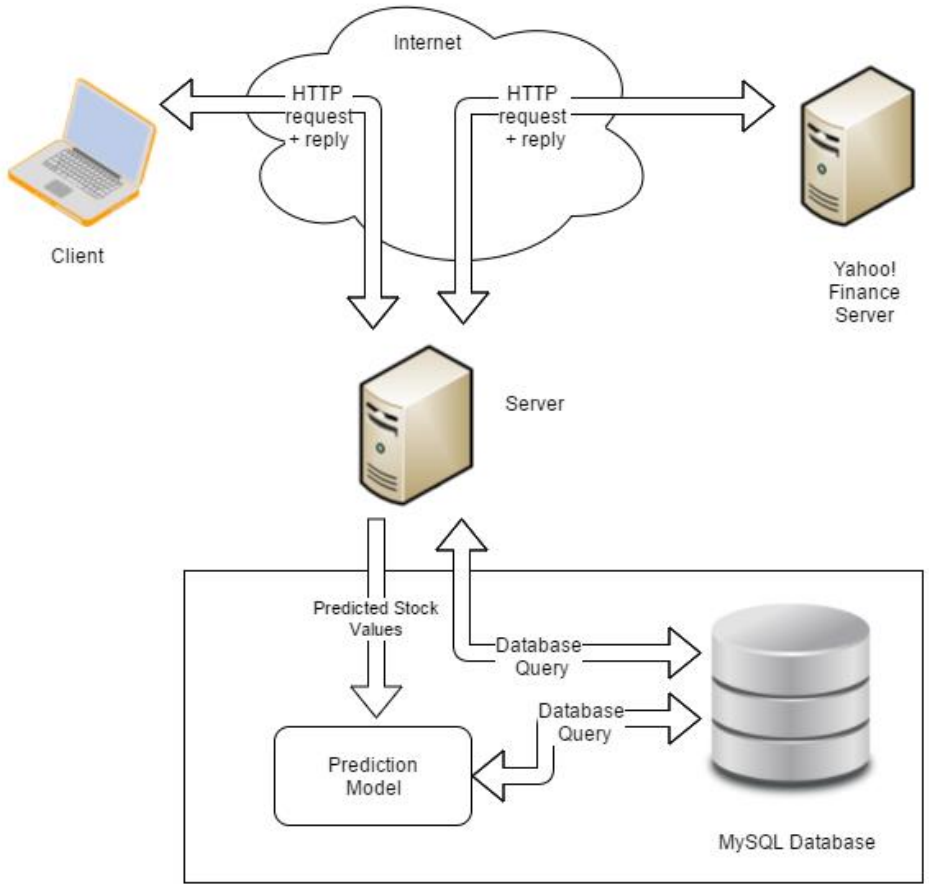


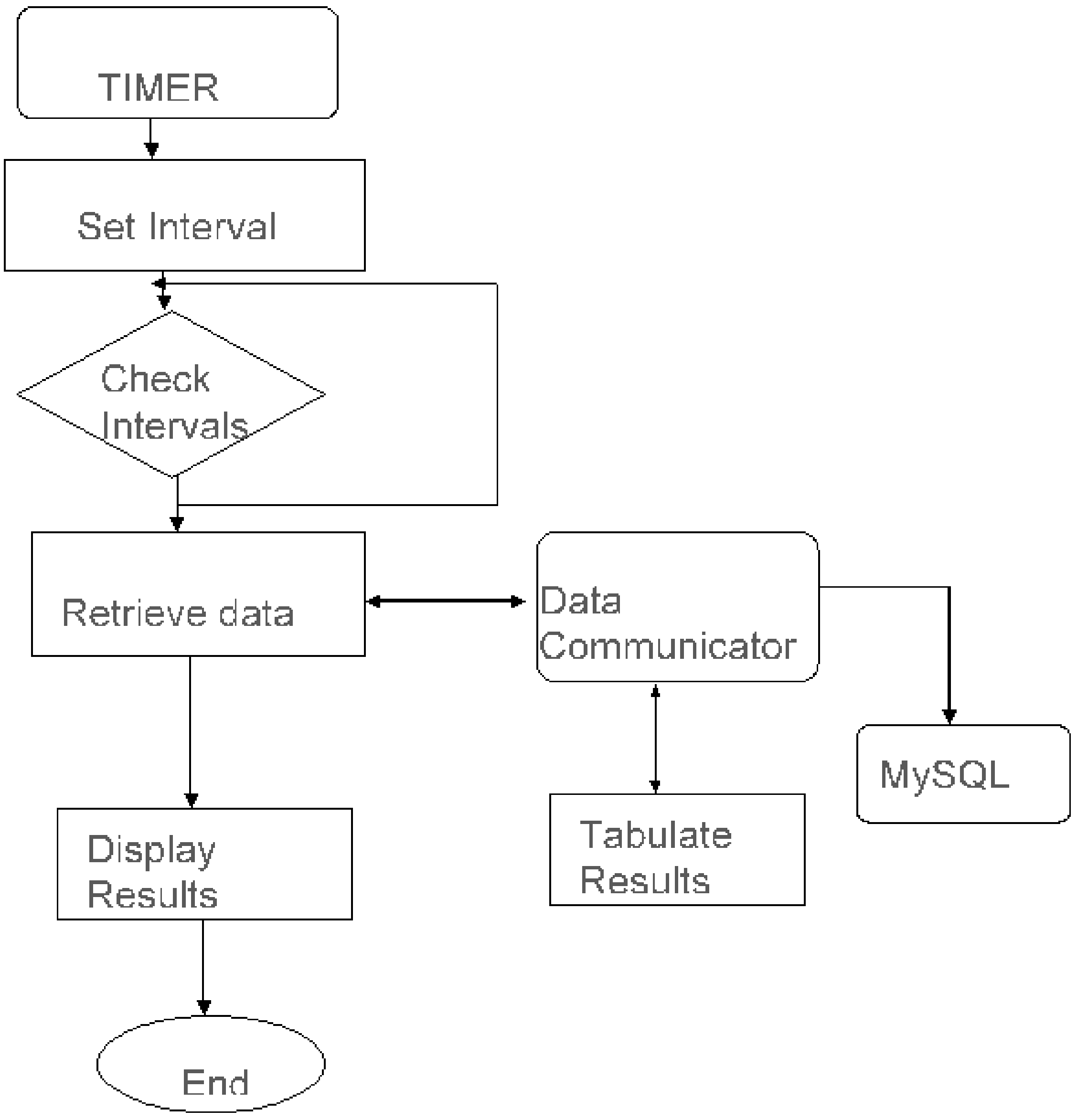
Figure 10 - Stock Prediction System.

The hardware and software components work hand in hand in achieving the deliverable. The client as well as the server houses an Apache Server. A PHP script is run on this apache server thereby establishing a connection between the server and the client. The user interface, a web page written as a HTML code in which the PHP script is embedded is where the user makes a query.

When a query is made the entry is passed as a variable in the PHP script. Once the message reaches the server, the server performs a real time invocation of Yahoo finance (http://finance.yahoo.com/). The historical data of the company is downloaded on to the MySQL database. The server houses the MySQL database and the R application. The R application makes use of these historical values and fits it into its prediction model thereby giving the required prediction. This predicted output is then stored in the database which in turn is retrieved by the client using another PHP script.

As an analogy, let us consider a restaurant. Here, the PHP can be compared to the steward, the Apache server being the master chef and the MySQL database being the stock room. When a guest (user) arrives, he places an order (makes a query) for a dish (company) he chooses. This order is carried to the chef (Apache server) by the steward (PHP). The guest places the order and waits for the dish to arrive. It is of no concern to him how the dish is prepared or what goes into it. The user in our case is completely oblivious to how the query is passed, or how the data is retrieved and processed. He only waits for the end results. The chef now has the details of the dish and he uses the ingredients (data) from his stock room (database) in order to prepare the dish. Once the dish is ready (predicted value), the steward (PHP) comes into picture again. He delivers the dish back to the guest. This highly interconnected system requires optimum synchronization to have the system up-and-running.

Algorithms and Data Structures



The main algorithm StockGuru uses is for the prediction of the how a certain stock will perform. This is carried out by a mathematical method known as neural networks. This method of using neural networks is also closely intertwined with another mathematical model called technical analysis. Before going any further one needs to understand how technical analysis works. Essentially for the system to understand how a particular stock is going to perform in the future, it first needs data points from the past (ideally at the bare minimum a month). Then using the data points of how the stock performed in the past a prediction of the future can be extrapolated based upon previous trends. Now for this algorithm the specific type of network we will be using is called multilayer perceptron. All that this means is that the input goes through many layers (i.e. operations or computations) until it reaches the output. Above is a pictorial representation. Now essentially what we do is we train our network the data about how stocks performed in the past, specifically on how stocks performed on specific dates. The more data that is inputted the greater the amount of nodes will be. This in turn would increase computation time but increase accuracy. Similarly, if less data about a stock's past performance is given 44 the faster it will compute but the less accuracy it will have. Like all systems it is necessary to find the balance between speed and performance when working with this algorithm. Data Structures Database The main database will hold all of the stock data implemented with a relational table. A relational table is the ideal structure to hold the stock data as we need to store many different numerical values all related to a single stock ticker stored as a string.

For example (CompanyName, TickerSymbol, OpenPrice, ClosePrice, High, Low, etc.) where all the values in a row correspond to a given stock. Ticker Symbols We will need a smaller database which contains all of the company names and their ticker symbols. This is best implemented with a Hashmap where each key of the map is either the company name or ticker symbol and the values are the corresponding ticker symbols. This will let a user search for a stock by either company name, like "Google," or by ticker symbol, like "GOOG." User Database The user database will store the usernames and passwords of the users on our application. It will also use a relational database to store a string email/username and its respective hashed password. A relational database is a good choice for this database as we will already be implementing the stock database with SQL, which will reduce the cost of setting it up, and the stored data is simple enough so it could be stored in any data structure. Word Frequency. To implement our neural network, we will likely use a Hashmap to store the frequency of found words. A hash map is a good implementation because it can be easily searched with efficient search methods like quicksort and its key-value setup is similar to the needed word-frequency setup for analysis with the neural network.

STOCK PREDICTION USING NEURAL NETWORKS

Predicting the stock market is not a simple task, mainly as a consequence of the close to random-walk behavior of a stock time series. Different techniques are being used in the trading community for prediction tasks. In recent years the concept of neural networks has emerged as one of them. We have developed an efficient tool for stock market forecasting based on Neural Networks.

Machine learning:

The machine learning module consists of two stages (repeated many times). First, we present the system with the input data and obtain the output. Second, we adjust the system, to make output closer to what it should be. The first stage is referred as feedforward, the second as back-propagation.

Neural Networks technology:

Prediction of stock market returns is an important issue in finance. Nowadays artificial neural networks (ANNs) have been popularly applied to finance problems such as stock exchange index prediction. An ANN model is a computer model whose architecture essentially mimics the learning capability of the human brain. The processing elements of an ANN resemble the biological structure of neurons and the internal operation of a human brain.

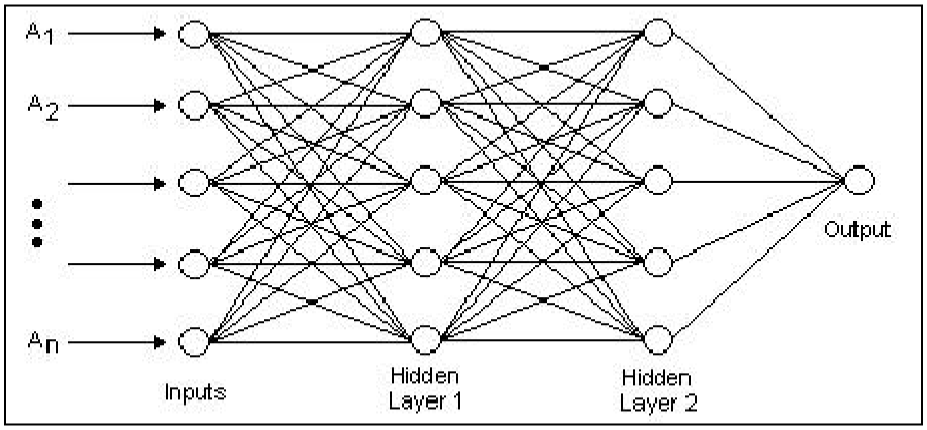
The use of ANN comes in because they can learn to detect complex patterns in data. It is able to work parallel with input variables and consequently handle large sets of data swiftly. The principal strength with the network is its ability to find patterns and irregularities as well as detecting multi-dimensional non-linear connections in data. In mathematical terms, they are universal non-linear function approximators meaning that given the right data and configured correctly; they can capture and model any input/output relationships.

This not only removes the need for human interpretation of charts or the series of rules for generating entry/exit signals but also provides a bridge to fundamental analysis as that type of data can be used as input. In addition, as ANNs are essentially non-linear statistical models, their accuracy and prediction capabilities can be both mathematically and empirically tested. This quality is extremely useful for modeling dynamical systems, e.g. the stock market.

Artificial Neural Network Approach:

One of the most important factors in constructing a neural network is deciding on what the network will learn. The goal of most of these networks is to decide when to buy or sell securities based on previous market indicators. The challenge is determining which indicators and input data will be used, and gathering enough training data to train the system appropriately.

We are using Multi-Layer Feed-Forward network topology for our project. Input layer is composed of a set of inputs that feed input patterns to the network. The inputs that we provide to our input layer is the sets of „closing prices‟ of a company. Following the input layer there will be at least one or more intermediate layers, often called hidden layers. Hidden layers will then be followed by an output layer, where the results can be achieved. In feed-forward networks all connections are unidirectional. These networks, also known as back-propagation networks, are mainly used for applications requiring static pattern classification. The main advantage of these networks is their ease of use and approximation of any input/output map. The main disadvantage is that they train slowly and require lots of training data.



Network Training and Testing:

Training a network involves presenting input patterns in a way so that the system minimizes its error and improves its performance. A multilayer feedforward network uses backpropagation training algorithm. Backpropagation is the process of back propagating errors through the system from the output layer towards the input layer during training. Backpropagation is necessary because hidden units have no training target value that can be used, so they must be trained based on errors from previous layers. The output layer is the only layer which has a target value for which to compare. As the errors are back propagated through the nodes, the connection weights are changed. Training occurs until the errors in the weights are sufficiently small to be accepted. It is interesting to note that the type of activation function used in the neural network nodes can be a factor on what data is being learned.

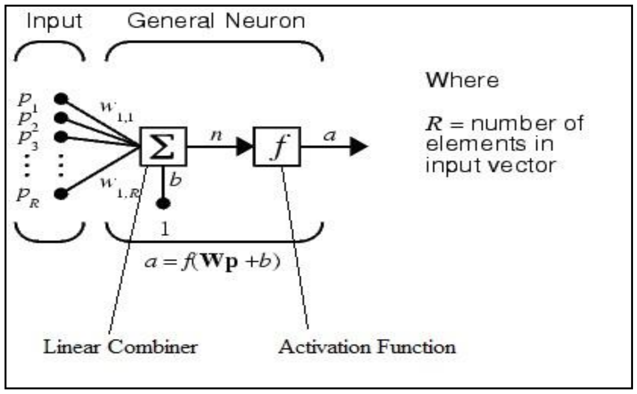
This process is performed internally by Encog library in java by using the train function. The function just accepts input parameters on which it trains network. As mentioned before we provide sequences of closing price of a particular company as an input parameter to the „train" function. We are providing 180 sequences, each sequence having 10 different closing prices which is randomly picked up. The train procedure involves training the network on most of the patterns (usually around 90%) and then testing the network on the remaining patterns. The network's performance on the test set is a good indication of its ability to generalize and handle data it has not been trained on. The system is trained on k-1 sets and tested on the remaining k set times, using a different test set each time.

Conditions for halt in training process:

- The maximum number of epochs (repetitions) is reached.
- The maximum amount of time is exceeded.
- Performance is minimized to the goal.
- The performance gradient falls below minimum performance gradient.
  - Validation performance has increased more than maximum validation failures since the last time it decreased (when using validation).

Neuron Models:

Below are the architectures of the network that are most commonly used with the backpropagation algorithm - the multilayer feedforward network.

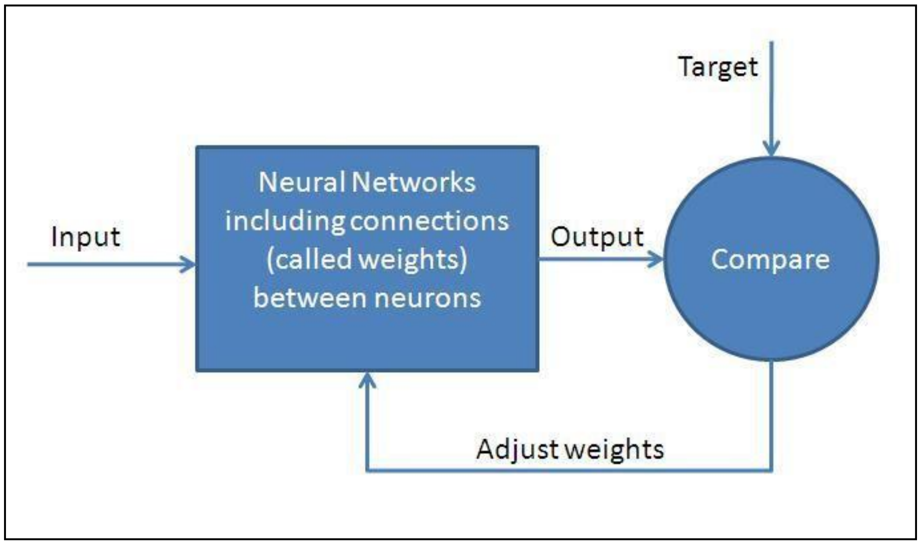


An elementary neuron with R inputs is shown alongside. Each input is weighted with an appropriate w. The sum of the weighted inputs and the bias forms the input to the transfer function f. Neurons may use any differentiable transfer function f to generate their output.

A **Linear Combiner**, which is a function that takes all inputs and produces a single value. A simple way of doing it is by adding together the Input (p1,p2…pR) multiplied by the Synaptic Weight (w1,w2….wR).

By applying the **Activation function**, it will take any input from minus infinity to plus infinity and squeeze it into the -1 to 1 or into 0 to 1 interval.

After the neuron in the first layer received its input, it applies the Linear Combiner and the Activation Function to the inputs and produces the Output. This output, as you can see from figure 19, will become the input for the neurons in the next layer. So the next layer will feed forward the data, to the next layer. And so on, until the last layer is reached.



Target
Input
Neural Networks including connections (called weights) between neurons
Output
Compare
Adjust weights

Fig : Basic Functionality of Neural Network

When we work with yesterday's price, we not only know the price for the "day - 1", but also the price we are trying to predict, called the „Desired output" of the Neural Net. When we compare the two values, we can compute the Error: dError = dDesiredOutput - dOutput;

Now we can adjust this particular neuron to work better with this particular input. For example, if the dError is 10% of the dOutput, we can we can increase all synaptic weights of the neuron by 10%.

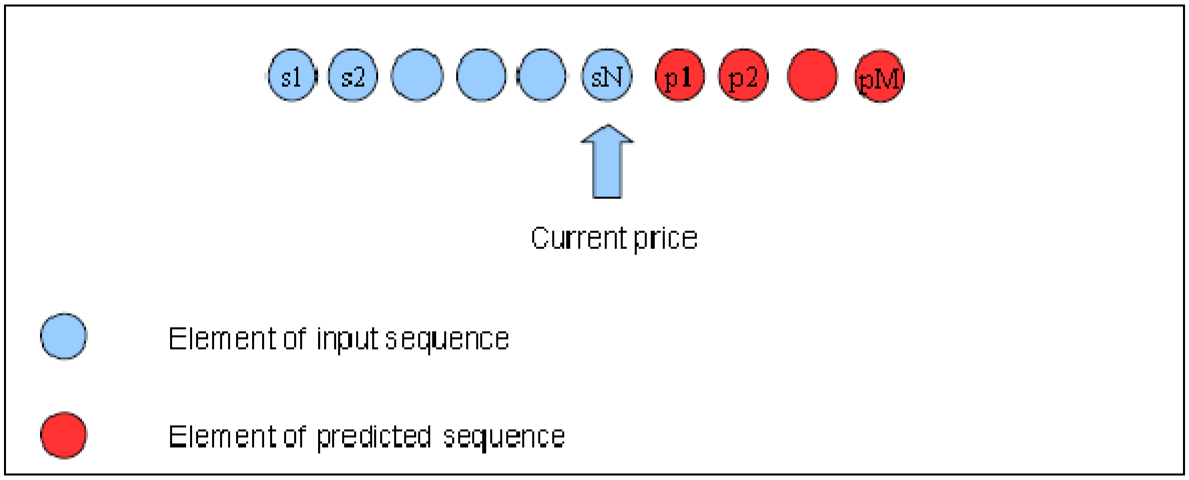
The problem with this approach is that the next input will require a different adjustment.

Hence we perform the following operation: dNewWeight = dOldWeight * dAdjustment * dLearningRate.

The learning rate (dLearningRate) is of importance of a single pattern. For example, we can set it to 0.01, then it will take 100 patterns to make a 10% adjustment.

Basic working of our model:

The model (neural network) accepts, as input, a sequence of given length N. The system can determine if at least one of future prices - within an observation window of fixed length M - will be higher or lower than current price.



Basic Working of our Model

A buy signal is given if at least one of red circles is greater than current price. A sell signal is given if at least one of red circles is lower than current price. Formally a buy signal is verified if (p1>sN) OR (p2>sN) OR ... OR (pM>sN), a sell signal is verified if (p1<sN) AND (p2<sN) AND.... AND (pM<sN).

The neural network memorizes or learns the sequences of the form s1, s2….sN and also memorizes the buy signal or sell signal. Now during the test phase the neural network takes sequences, predicts the output based on the training and then verifies if the predicted value matches with actual value. This where performance evaluation is done and if the neural network seems to be giving satisfactory performance then the final sequence is given to it and the network is asked to predict the future value.

Stock market prediction has been an important issue in the field of finance and mathematics due to its potential financial gain. As a vast amount of capital is traded through the stock market, the stock market is seen as a peak investment outlet. In addition, stock market prediction brings with it the challenge of proving whether the financial market is predictable or not. With the advent of faster computers and vast information over the Internet, stock markets have become more accessible to either strategic investors or the general public.

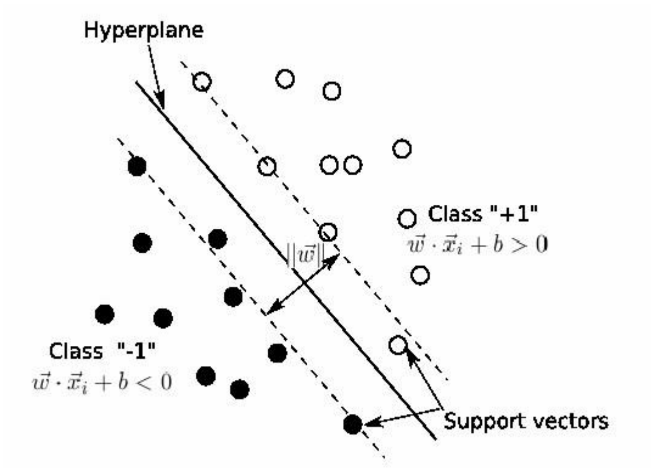
With advent of Machine Learning techniques, computers can "learn" various features and trends in data which can then be used to predict future values. The major focus of machine learning research is to extract information from data automatically, by computational and statistical methods. Various machine learning techniques like Support Vector Machines (SVMs) and Neural Networks (NNs) have been studied to analyze their capability to predict the stock markets.

An important requirement to be considered while designing the Stock predictor is the investment horizon of the user i.e. whether the user wishes for short term or long term investment. This issue is critical as one algorithm may suite short term investment strategy while it may fail miserably when applied for long term gains and vice versa. Keeping this in mind we have implemented two modules viz. SVM based long term predictor which helps the user for long term gains in a volatile market and Monte Carlo Predictor for short term gains in the market. For long term prediction a large amount of data needs to be analyzed. SVMs are more suited for long term prediction because they are easy to train, computationally less intensive and efficient as compared to other techniques like Neural Networks. The comparison of SVMs and Neural Network follows in next few sections. Also Monte Carlo method uses the correlation in stocks in a particular sector to predict the future values. As correlation in stock data holds good for short time and may or may not hold good for long periods, we use Monte Carlo method for short term prediction. Before doing so we present a sort of literature study carried out before implementing the algorithms.

Support Vector Machine based Stock Predictor:

There are various machine learning algorithms which are used for Stock prediction but amongst them the two most successful and commonly used are Support Vector Machines (SVMs) and Neural Networks (NNs). We carried out an initial literature survey of the two algorithms to figure out the best algorithm which suites our requirements.

Support Vector Machines:


Hyperplane
Class "+1"
$\vec{w} \cdot \vec{x}_i + b > 0$
$||\vec{w}||$
Class "-1"
$\vec{w} \cdot \vec{x}_i + b < 0$
Support vectors

Support Vector Machine classifier.

A Classifier is a function that maps input data samples to a defined set of object class. The task of the Classifier is to determine the value of the function for any valid input object after having seen a number of training examples (i.e., pairs of input and target output).

The classifier has to assign the input measurements to one of the classes learned in the training phase. In SVMs this classification of data is achieved by defining a plane known as a "hyperplane" that separates the data samples of different class. The closest data points to the hyperplane are called Support Vectors. The hyperplane is determined in a way to maximize the distance from the nearest data point to the plane. The maximized distance introduced, accommodates for any minor deviations in the data and also increases the ability of the classifier to learn. The hyperplane is of the form, w. x - b= 0, where w is the distance between the plane and the closest data point, x is the input data and b is the offset from the origin.

<u>Support Vector Machines Vs Neural Networks:</u>

Comparative studies to analyze the capabilities of the two classifiers are presented by researchers. NNs use feedback during training phase which may lead to errors due to iterative training while SVMs use mathematical kernels to separate the data, providing a global and unique solution. NN requires a skillful human to validate the results of classification during the training phase. NNs have the black box problem, which does not reveal the significance of each variable and the way they weigh independent variables. As the individual role of each variable cannot be determined, it is impossible to understand how the network produces future price of stock.

The main disadvantage of NN is that the weights determined to do classification tends to work for input data similar to training samples used. That is, NN tends to overfit the data set, reducing the learning and generalization capability of the network. In SVMs, the problem of overfitting doesn't occur because a mathematical kernel is used. Thus, SVMs are more suitable for Stock prediction than NNs.

As concluded from the above comparison, Support Vector Machines are more suited to the problem of Stock Prediction as compared to Neural Networks. We use Encog library functions in java to implement SVM.

SVM Encog package has 4 functions as follows,

<u>Stocktraindata</u>: This function provides sample data patterns to train our SVM. It has data resembling to Cup and Saucer, Ascending Triangle, Descending Triangle and Head and Shoulders. Extending the number of patterns to be predicted can very easily be achieved by including the data patterns in this function. Thus this function provides easy scaling and functionality of the application.

<u>train_stock_class</u>: This function provides the necessary parameters to train the SVM. We use one_vs_rest algorithm, Radial Basis Kernel and cross validation to make our SVM decision robust. The parameter details are very specific to Support Vector Machines and our out of scope of this document. Users are recommended to go through which gives a good insight into the parameters used. For our application we use optimal parameter which suite our goal of Pattern prediction.

<u>test_stock_class:</u> This function provides it with the trained SVM model and the test data i.e. the data whose pattern is to be predicted. In turn this function receives the pattern detected in the data.

<u>Main_script:</u> This is the main function which takes care of interaction with the database and the order of the execution of various functions.

Stock Prediction Using Technical Indicators

Definition: A technical indicator is any class of metrics whose value is derived from generic price activity in a stock or asset.

Technical indicators look to predict the future price levels, or simply the general price direction, of a security by looking at past patterns. Collectively called "technical", they are distinguished by the fact that they do not analyze any part of the fundamental business, like earnings, revenue and profit margins. Technical indicators are used most extensively by active traders in the market, as they are designed primarily for analyzing short-term price movements. To a long-term investor, most technical indicators are of little value, as they do nothing to shed light on the underlying business. The most effective uses of technical for a long-term investor are to help identify good entry and exit points for the stock by analyzing the long-term trend.

We select a distinguished choice of indicators which together provide us with recommendation\s for buying, selling or holding stocks. They are as follows:

Simple Moving averages: Moving averages are generally used to measure momentum and define areas of possible support and resistance. They are used to emphasize the direction of a trend and to smooth out price and volume fluctuations, or "noise", that can confuse interpretation. Typically, upward momentum is confirmed when a short-term average (e.g.15-day) crosses above a longer-term average (e.g. 50-day). Downward momentum is confirmed when a short-term average crosses below a long-term average. The most common way to interpreting the price moving average is to compare its dynamics to the price action. When the instrument price rises above its moving average, a buy signal appears, if the price falls below its moving average, what we have is a sell signal.

This trading system, which is based on the moving average, is not designed to provide entrance into the market right in its lowest point, and its exit right on the peak. It allows to act according to the following trend: to buy soon after the prices reach the bottom, and to sell soon after the prices have reached their peak.

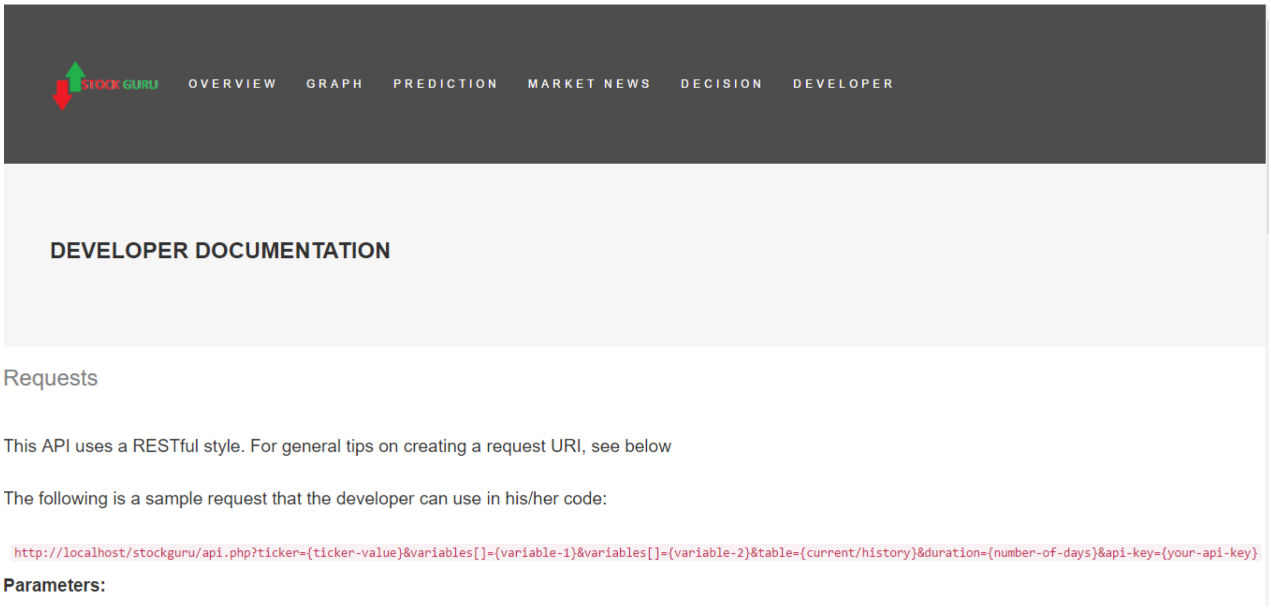
Bollinger Bands: Having evolved from the concept of trading bands, Bollinger Bands and the related indicators % band bandwidth can be used to measure the "highness" or "lowness" of the price relative to previous trades. Bollinger Bands are a volatility indicator similar to the Keltner channel.

Bollinger Bands consist of:

- an N-period moving average (MA)
- an upper band at K times an N-period standard deviation above the moving average ($MA + K\sigma$)
- a lower band at K times an N-period standard deviation below the moving average ($MA - K\sigma$)

Typical values for N and K are 20 and 2, respectively. The default choice for the average is a simple moving average, but other types of averages can be employed as needed. Exponential moving averages is a common second choice. Usually the same period is used for both the middle band and the calculation of standard deviation.

Web service Interface



DEVELOPER DOCUMENTATION

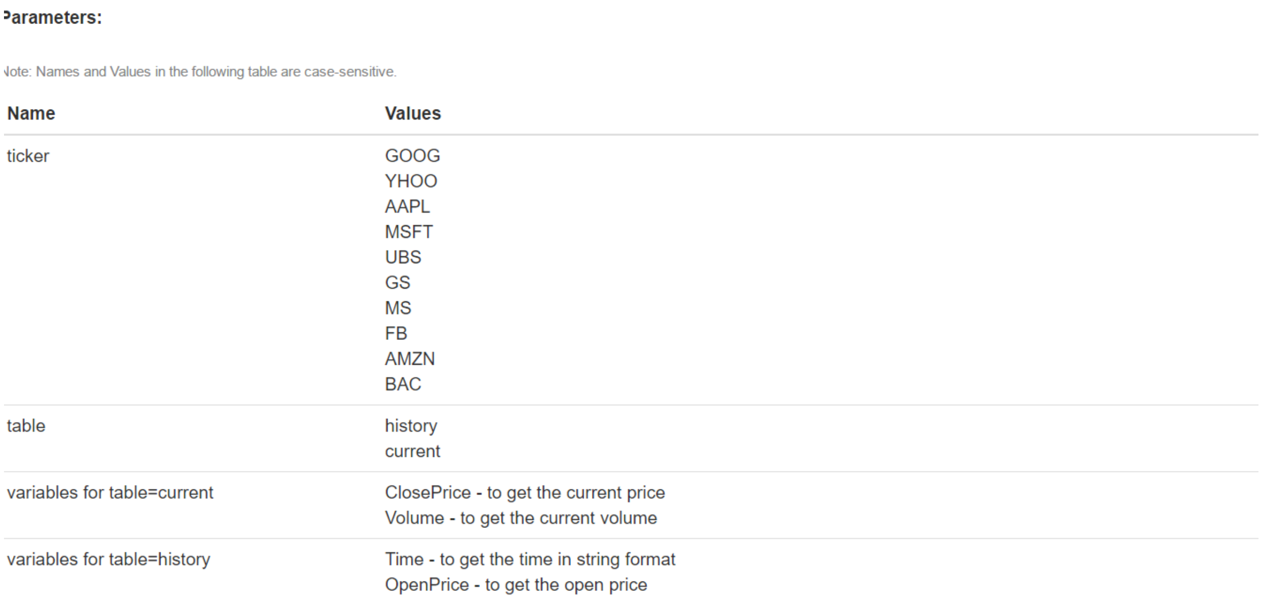
Requests

This API uses a RESTful style. For general tips on creating a request URI, see below

The following is a sample request that the developer can use in his/her code:

```
http://localhost/stockguru/api.php?ticker={ticker-value}&variables[]={variable-1}&variables[]={variable-2}&table={current/history}&duration={number-of-days}&api-key={your-api-key}
```

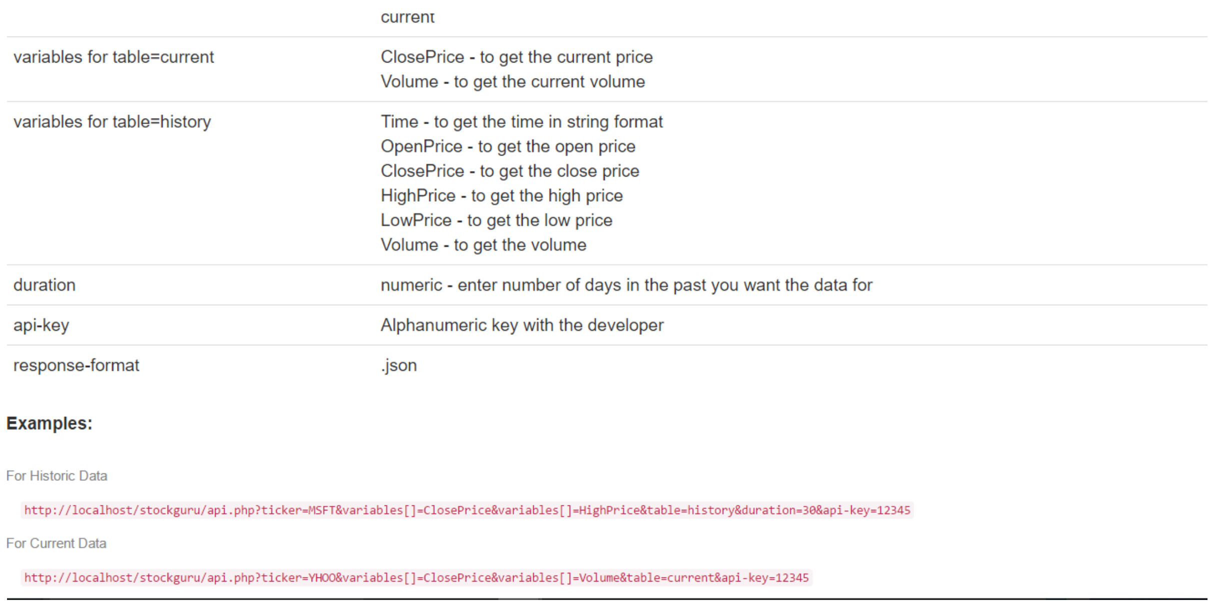
Parameters:



Parameters:

Note: Names and Values in the following table are case-sensitive.

Name	Values
ticker	GOOG YHOO AAPL MSFT UBS GS MS FB AMZN BAC
table	history current
variables for table=current	ClosePrice - to get the current price Volume - to get the current volume
variables for table=history	Time - to get the time in string format OpenPrice - to get the open price



	current
variables for table=current	ClosePrice - to get the current price Volume - to get the current volume
variables for table=history	Time - to get the time in string format OpenPrice - to get the open price ClosePrice - to get the close price HighPrice - to get the high price LowPrice - to get the low price Volume - to get the volume
duration	numeric - enter number of days in the past you want the data for
api-key	Alphanumeric key with the developer
response-format	.json

Examples:

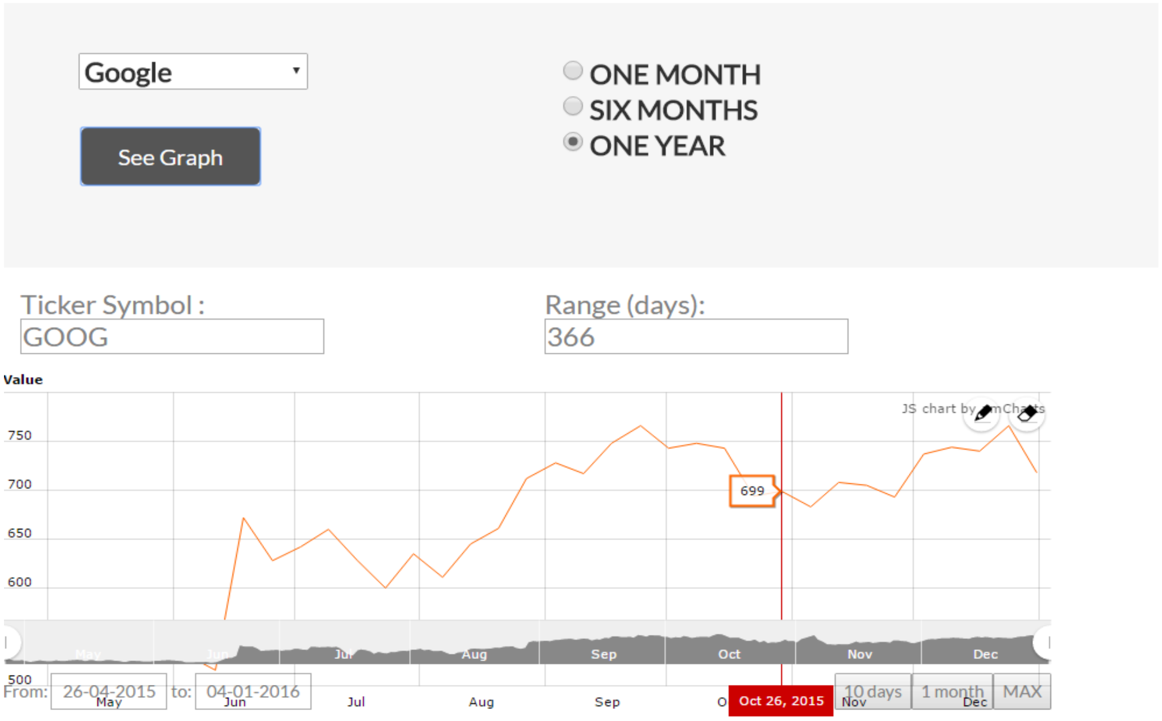
For Historic Data

```
http://localhost/stockguru/api.php?ticker=MSFT&variables[]=ClosePrice&variables[]=HighPrice&table=history&duration=30&api-key=12345
```

For Current Data

```
http://localhost/stockguru/api.php?ticker=YHOO&variables[]=ClosePrice&variables[]=Volume&table=current&api-key=12345
```

UI Design Implementation


Google

See Graph

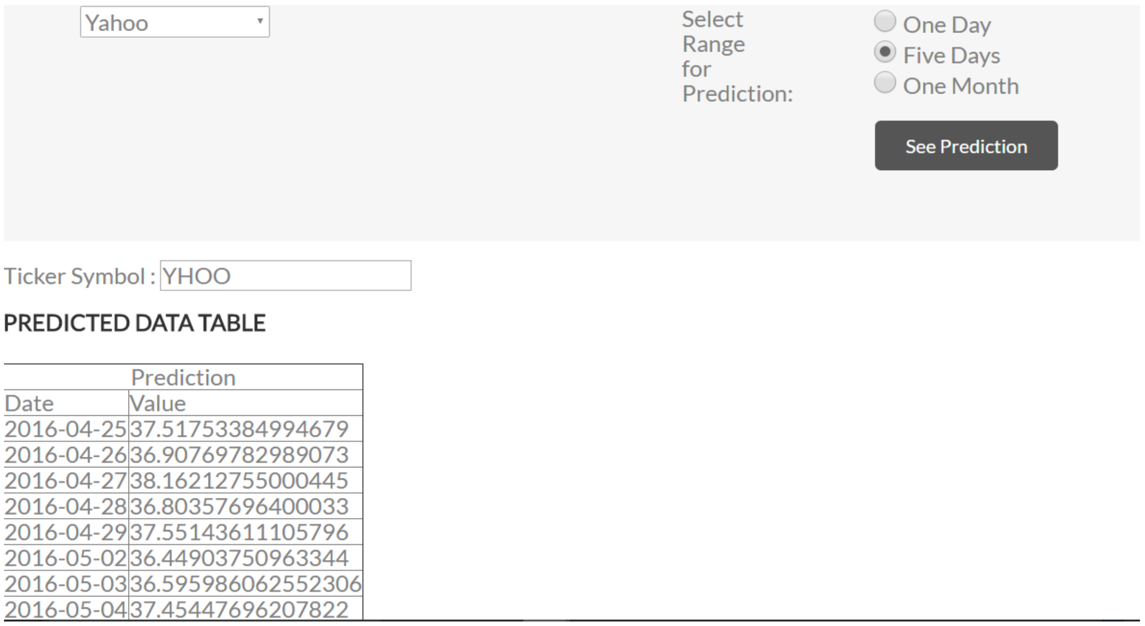
ONE MONTH
SIX MONTHS
[✓ ONE YEAR]

Ticker Symbol : GOOG

Range (days): 366

From: 26-04-2015 to: 04-01-2016

Figure – Displaying Trends


Yahoo

Select Range for Prediction:

One Day
[✓ Five Days]
One Month

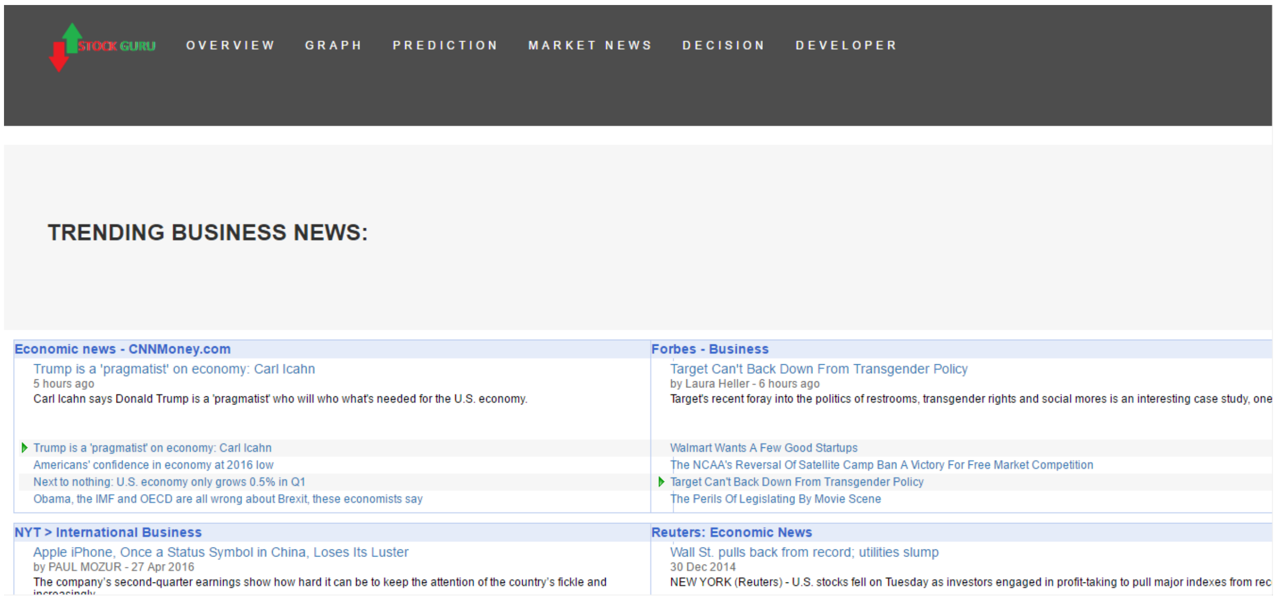
See Prediction

Ticker Symbol : YHOO

PREDICTED DATA TABLE

Date	Prediction Value
2016-04-25	37.51753384994679
2016-04-26	36.90769782989073
2016-04-27	38.16212755000445
2016-04-28	36.80357696400033
2016-04-29	37.55143611105796
2016-05-02	36.44903750963344
2016-05-03	36.595986062552306
2016-05-04	37.45447696207822

Figure – Predicted Data



TRENDING BUSINESS NEWS:

Economic news - CNNMoney.com

Trump is a 'pragmatist' on economy: Carl Icahn
5 hours ago
Carl Icahn says Donald Trump is a 'pragmatist' who will who what's needed for the U.S. economy.

- Trump is a 'pragmatist' on economy: Carl Icahn
- Americans' confidence in economy at 2016 low
- Next to nothing: U.S. economy only grows 0.5% in Q1
- Obama, the IMF and OECD are all wrong about Brexit, these economists say

Forbes - Business

Target Can't Back Down From Transgender Policy
by Laura Heller - 6 hours ago
Target's recent foray into the politics of restrooms, transgender rights and social mores is an interesting case study, one

- Walmart Wants A Few Good Startups
- The NCAA's Reversal Of Satellite Camp Ban A Victory For Free Market Competition
- Target Can't Back Down From Transgender Policy
- The Perils Of Legislating By Movie Scene

NYT > International Business

Apple iPhone, Once a Status Symbol in China, Loses Its Luster
by PAUL MOZUR - 27 Apr 2016
The company's second-quarter earnings show how hard it can be to keep the attention of the country's fickle and

Reuters: Economic News

Wall St. pulls back from record; utilities slump
30 Dec 2014
NEW YORK (Reuters) - U.S. stocks fell on Tuesday as investors engaged in profit-taking to pull major indexes from rec

Figure – Displaying Market News

Design of Tests

No application is complete until it has been tested as thoroughly as possible for security holes, broken functionality, and any other lacking features. Shipping without testing is a guarantee to have all manner of bugs and security holes. However, even with thorough testing, it is not usually possible to find and resolve every flaw before shipment. To this end, developers utilize testing suites to try and test programs efficiently and effectively. Tests can be designed for individual units and components as well as the broader system and the integration of the units. While not perfect for finding all flaws in a program (usually errors are discovered by looking for them, which generally requires either knowledge of an existing error or "luck" in an error making itself apparent during development), testing can serve to find almost all errors and flaws in an application.

However, developers face a dilemma. Developing an evolving application can cause existing tests to become outdated, while designing and running tests is time taken away from actually building the application.

A modern approach to this tradeoff is to build the feature set of an application around measurable, predefined tests. In this technique, known as Test-driven Development, developers iteratively define tests for intended future features, confirm that those features are not yet implemented (by running those tests), and then implementing the solutions. Though this approach does not (generally) test for all possible interplay between components, it is usually employed in high paced development environments such as ours, where the coverage provided is usually respectable enough to prevent most problems.

Accordingly, we first define the features and tests we plan on developing around, proceed to analyze the coverage offered by these tests, and then briefly discuss how we intend to test the integration of the components.

Test Cases

Due to project constraints, we cannot afford to thoroughly test existing packages for functionality we incorporated to streamline the design process. These packages include Ruby on Rails as well as Ruby gems (packages) for interfacing with Yahoo! Finance, various databases (ie MySQL, SQLite), the Resque queueing system, and other auxilliary package for Rails. Likewise, we cannot unit test the HTTP server we are using (Apache) and its Ruby extension (Phusion Passenger) or any of the databases. Rather we will focus on testing just the units of our application and their integration with each other.

Routing

As described earlier, Capital Games contains models for users, managers, administrators, trades, etc. Intrinsic to the Rails web framework we employ, most of these models are represented internally to the controller as "resources". At any point in time, any user (even a non-user!) could attempt to gain access to a resource to which they are not privileged, such as an administrator panel. Routing unit tests will confirm that only authorized users will be able to access restricted pages. As an extension of this premise, Routing unit tests will also confirm that pages with low privileges are accessible to all users and front-facing pages can be seen even without being logged in.

Database Models

Because of the data-centric design of Capital Games, protecting the integrity of the database entries is of the utmost importance. The Ruby on Rails framework has safeguards and validation for this purpose, but we still need to thoroughly unit test each of the models to ensure that only permissible combinations of attributes are able to be entered, and that proper error handling occurs to resolve attempts at improper attribute definition.

Queueing System

Capital Games heavily relies upon the queueing system to act as a computational highway for all asynchronous tasks. Due to the nature of this system we must prepare for race conditions; the different ways our data can be effected based upon the order of executing processes that are acting upon the queue. We will need to prepare a set of tests to express how the queue performs when open orders are altered by other processes during different phases of the queueing system. Based on our test results it might be necessary to implement data locking.

Finance Adapter

Whenever using external resources it is vital to understand the different ways in which they communicate not just when functioning as expected, but also when failing to perform properly. Since we do not have the ability to shut down the external Financial Adapter's Servers we can not run tests that give us feedback on what functionality to expect on failure. This leaves us without the ability to test the Financial Adapter and instead pro-actively safeguard against failure. Due to this we must build a wrapper that anticipates all perceivable failures coming from the Financial Adapter.

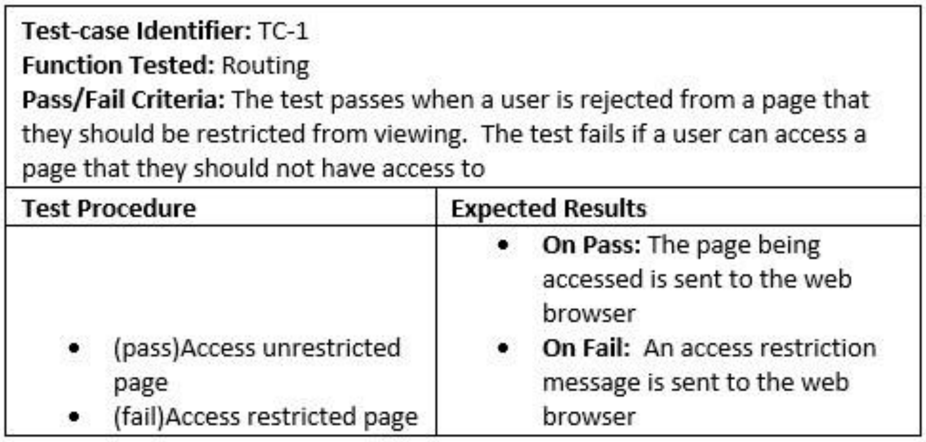
Test Coverage

In order to attain full functionality of Capital Games without bugs, we must be sure that none of its parts have errors themselves. Due to many dependencies such as other running processes, system states, and transitions, the same test will need to be preformed for each possible configuration to make sure that each part works in every possible scenario that it can be ran. This will require extending certain tests to run at the same time as background processes, and having parts called from all possible initiating parts. When working with integrated parts it is not simply enough to assume that parts will work once integrated just because they work independently. By extensively testing each possible run case we ensure that there are no points of failure once the system is launched.

Integration Testing

In order to achieve the most thorough testing, Capital Games will be tested using the bottom up strategy. Each part of Capital Games that we wrote will be extensively tested individually first. However simply testing each part individually is not enough due to race conditions and other integration issues that may exist in the systems described above. Because of this, parts must be tested after integration as well.

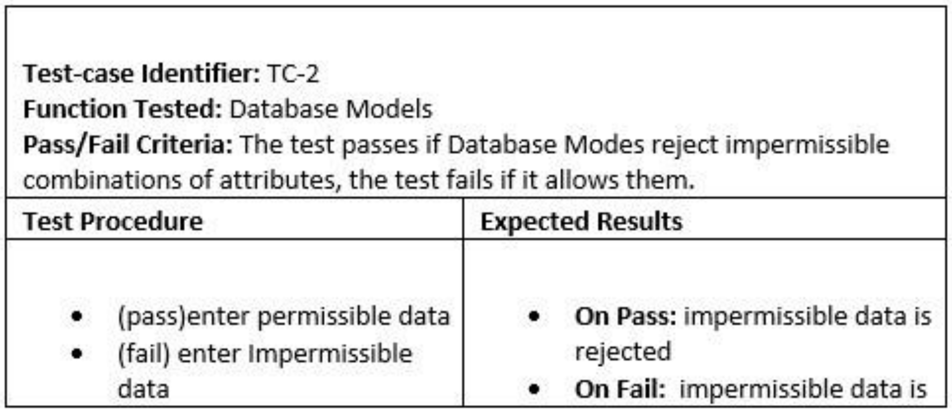
Knowing that functionality is state specific and transition specific for any state machine, each test must also be ran in all possible states. In addition to all previously listed conditions, tests need to be preformed at different times to make sure that functionality during backend asynchronous tasks do not have any bugs. We have chosen the bottom-up testing strategy based on the principle that bugs at the bottom level will dictate bugs at the top level, while bugs at the top level may very well be independent of bottom level performance. By carefully analyzing every part to part integration we can work our way up to a flawless design.



Test-case Identifier: TC-1
Function Tested: Routing
Pass/Fail Criteria: The test passes when a user is rejected from a page that they should be restricted from viewing. The test fails if a user can access a page that they should not have access to

Test Procedure	**Expected Results**
• (pass)Access unrestricted page • (fail)Access restricted page	• **On Pass:** The page being accessed is sent to the web browser • **On Fail:** An access restriction message is sent to the web browser

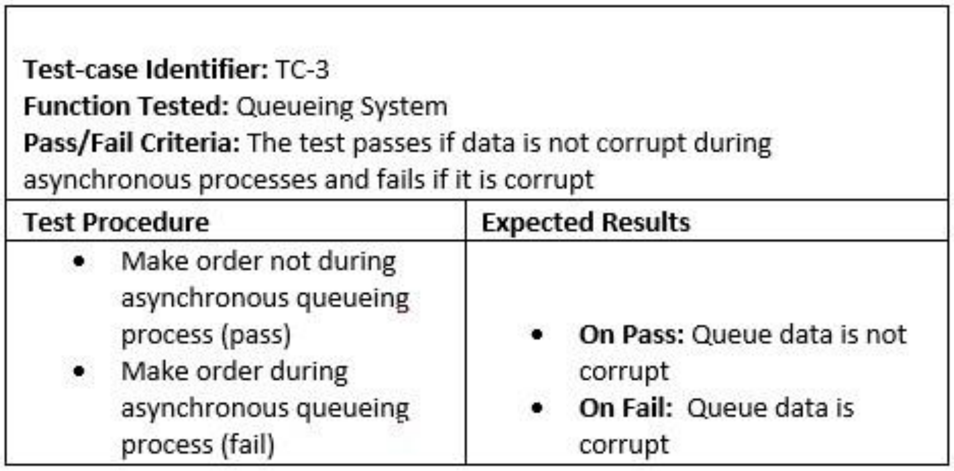
Test Case 1.



Test-case Identifier: TC-2
Function Tested: Database Models
Pass/Fail Criteria: The test passes if Database Modes reject impermissible combinations of attributes, the test fails if it allows them.

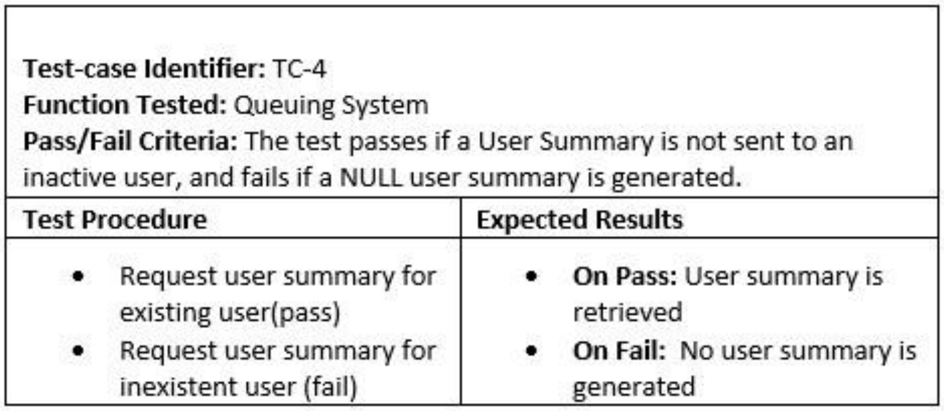
Test Procedure	**Expected Results**
• (pass)enter permissible data • (fail) enter Impermissible data	• **On Pass:** impermissible data is rejected • **On Fail:** impermissible data is

Test Case 2.


Test-case Identifier: TC-3
Function Tested: Queueing System
Pass/Fail Criteria: The test passes if data is not corrupt during asynchronous processes and fails if it is corrupt

Test Procedure	Expected Results
• Make order not during asynchronous queueing process (pass) • Make order during asynchronous queueing process (fail)	• **On Pass:** Queue data is not corrupt • **On Fail:** Queue data is corrupt

Test Case 3.


Test-case Identifier: TC-4
Function Tested: Queuing System
Pass/Fail Criteria: The test passes if a User Summary is not sent to an inactive user, and fails if a NULL user summary is generated.

Test Procedure	Expected Results
• Request user summary for existing user(pass) • Request user summary for inexistent user (fail)	• **On Pass:** User summary is retrieved • **On Fail:** No user summary is generated

Test Case 4.

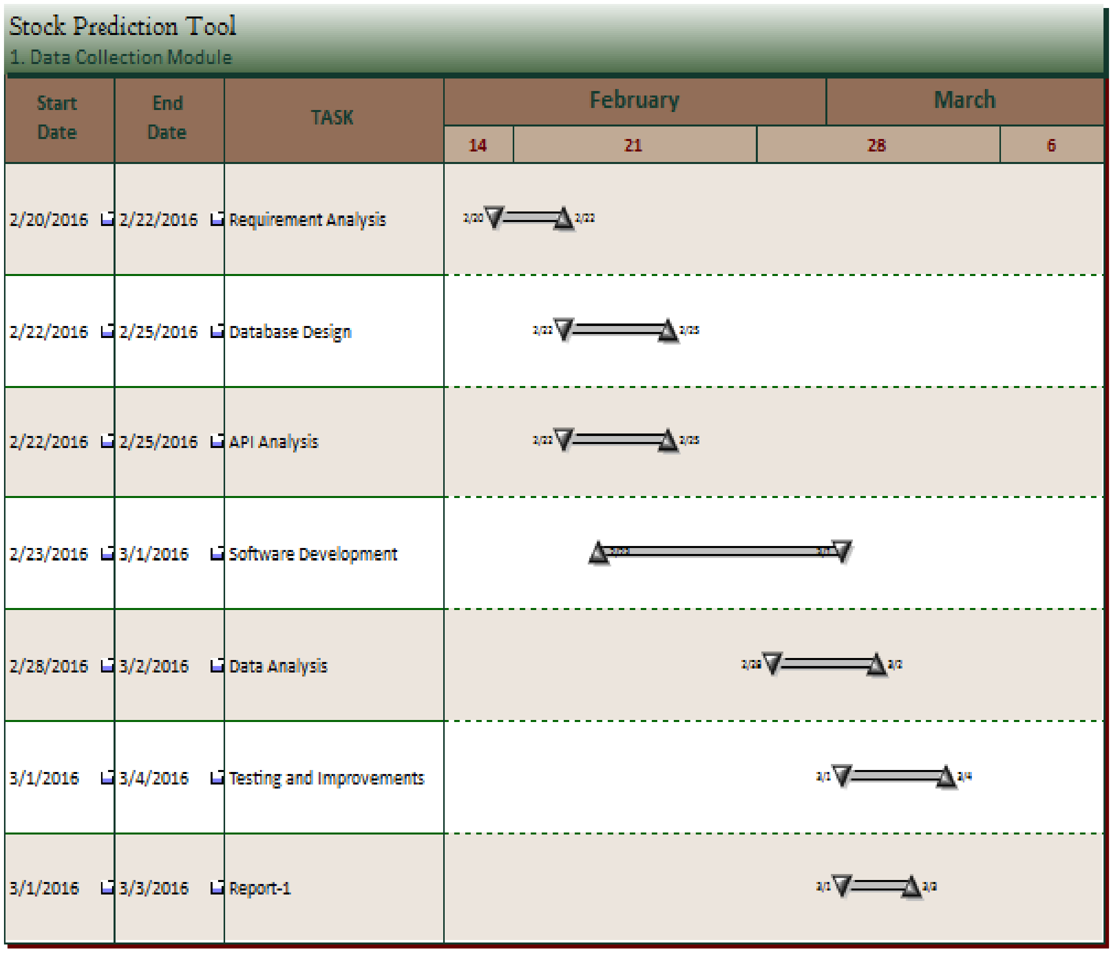
History of work, Current status, Future work

History of Work:

- **Dynamic data collection:** We have built a data feed algorithm which traverses financial web resources like Google finance and Yahoo finance; and collects data specific to the list of 10 companies for a time span of last one year, into the database.
- **Graphical support using Charts:** A customer can view the stock market history of any selected company in the form of a 2-D graph with the days of the previous year on the X-axis and corresponding closing price on the Y-axis. This visual display can help a customer to understand the dynamics of the stocks of an organization in a better way.
- **Friendly user interface:** We aim to educate the user on stock forecasting methodologies along with providing him with valuable prediction services. The designed website gathers a gamut of information like the latest news on stocks and companies, market indices, technology that went behind our prediction modules and the use of technical indicators. Navigating the website is kept easy and user can create his/her own account to access our services.
- **Web services:** Most of the websites today provide web services to the customers. Restful API services are being considered for deploying web service.
- **Stock forecasting by machine learning:** We have employed the latest and one of the most precise prediction technology with underlying 3-layered neural networks in our system to provide fair recommendations to end-user about buying or selling a particular stock. We successfully train the network using a wide range of real inputs from stock history and simulate the same for checking the reliability of our system. We also calculate the predicted price for any selected stock.
- **Secondary stock prediction modules:** We have implemented algorithms for 10 popular technical indicators in another module that works parallel to our central module. They are meant to provide more instantaneous prediction that might support the decision from neural network module.

Phase-I Plan:

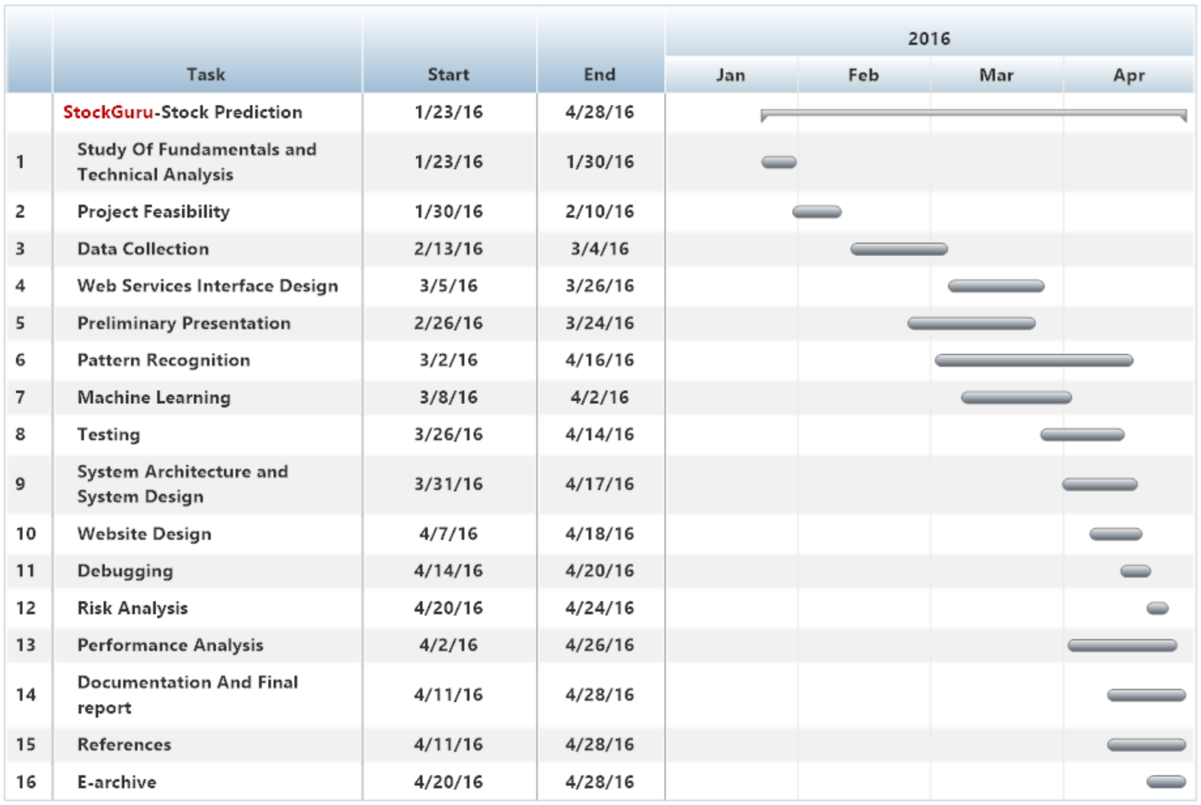
The following Gantt chart describes the plan of work from Project phase-1 open date until the submission date. The work was distributed equally among all the team members.


Stock Prediction Tool

1. Data Collection Module

Start Date	End Date	TASK
2/20/2016	2/22/2016	Requirement Analysis
2/22/2016	2/25/2016	Database Design
2/22/2016	2/25/2016	API Analysis
2/23/2016	3/1/2016	Software Development
2/28/2016	3/2/2016	Data Analysis
3/1/2016	3/4/2016	Testing and Improvements
3/1/2016	3/3/2016	Report-1

Phase-II Plan:



	Task	Start	End
	StockGuru-Stock Prediction	1/23/16	4/28/16
1	Study Of Fundamentals and Technical Analysis	1/23/16	1/30/16
2	Project Feasibility	1/30/16	2/10/16
3	Data Collection	2/13/16	3/4/16
4	Web Services Interface Design	3/5/16	3/26/16
5	Preliminary Presentation	2/26/16	3/24/16
6	Pattern Recognition	3/2/16	4/16/16
7	Machine Learning	3/8/16	4/2/16
8	Testing	3/26/16	4/14/16
9	System Architecture and System Design	3/31/16	4/17/16
10	Website Design	4/7/16	4/18/16
11	Debugging	4/14/16	4/20/16
12	Risk Analysis	4/20/16	4/24/16
13	Performance Analysis	4/2/16	4/26/16
14	Documentation And Final report	4/11/16	4/28/16
15	References	4/11/16	4/28/16
16	E-archive	4/20/16	4/28/16

Future Work:

- The system should track various stock movements simultaneously that might not be owned currently by customers and send alerts if it notices a favorable tilt in the graph that hints at immediate purchase.
- Developing customer profiles that should be maintained by the system and confidential customer data should be protected at all costs from prying software’s.
- The system should regularly send emails or short messages to the customer informing him about the current trends and future scenarios. It could feed the user current news and conduct surveys to judge its performance.
- If time permits, the system may provide in-detail risk analysis of user portfolios to generate Sharpe Ratio to evaluate their current and future risk standing in the market and take appropriate corrective measures.
- Improving the refreshing rate of the stock database.
- Trying to maintain the integrity of the database and other modules.
- Executing investors' requests as quickly as possible.

References

[1] http://www.taquote.com/public/download/techdefs.pdf

[2] Garth Garner, „Prediction of Closing Stock Prices‟, Portland State University department of Electrical and Computer Engineering

[3] http://www.traders.com/Documentation/RESource_docs/Glossary/glossary_TZ

[4] http://en.wikipedia.org/wiki/Technical_Analysis_Software_(Finance)

[5] Using Neural Networks to Forecast Stock Market Prices, Ramon Lawrence

[6] http://www.metaquotes.net/techanalysis/indicators/

[7] http://en.wikipedia.org/wiki/Technical_Analysis_Software_(Finance)

[8] http://www.bredemeyer.com

[9] http://www.agilemodeling.com/artifacts/useCaseDiagram.html

[10] http://aspnet.4guysfromrolla.com/articles/100803-1.aspx

[11] http://www.15seconds.com/issue/010430.htm

[12] http://msdn.microsoft.com/en-us/library/ms972326.aspx

[13] http://msdn.microsoft.com/en-us/webservices/default.aspx

[14] http://www.php.net/

[15] http://www.w3schools.com/PHP/php_get.asp

[16] http://devzone.zend.com/article/1081-Using-cURL-and-libcurl-with-PHP

[17] http://www.php-mysql-tutorial.com/connect-to-mysql-using-php.php

[18] http://phpwebservices.blogspot.com/

[19] http://www.w3schools.com/html/default.asp

[20] http://www.w3schools.com/xml/default.asp