StockProphit

Group 11

Name	**Email**
Asim Alvi	**asimalvi190@gmail.com**
Frank Bohn	**fbohn91@gmail.com**
Giovanni DeGrande	**degrangi@gmail.com**
Andrew Demoleas	**andrew.demoleas@gmail.com**
Eric Lee	**ericlee0492@gmail.com**
Sangit Patel	**ssgcrp@gmail.com**

Instructor: Prof. Ivan Marsic

Table of Contents

Individual Contributions Breakdown

All members contributed equally.

Summary of Changes

The following is a list of changes for this report:

- We removed the use case for top 5/bottom 5 because we got rid of that feature
- Also all of the other use cases were updated as well
- The interaction diagrams were updated with a description of the design pattern used
- Effort estimation using use case points was added
- Difficulties and Future section added to the report

Customer Statement of Requirements

Stocks are always changing and without proper research and persistence it is not possible to make smart investments. Time must be spent looking up the history of the stock to see how it performed in say the past week or the past month. Once you know how it has performed you can make guesses as to how it will perform in the future and decide whether or not you are willing to invest. In the busy world of today, people do not have the time to sit down and do all this research. They need a way to have a stock's history instantly assessed and processed into a prediction of how they will perform in the future.

The problem we are facing is that we have so many customers who are all looking for the next big thing to invest in and we just do not have the ability to accommodate every request. We need a way for the customers to easily get generic information from us as well as from other users, and do it in a way that is quick and efficient.

We are looking for a network that incorporates stock history and current events in order to accurately predict the price of a stock in the near future, which is all incorporated into a mobile device. We want a network that will allow users to post what they have found and also see what else is floating around the Internet that others have found. We also want this network to have the ability for notifications, favorites, to view projected stock rankings, and to search any stock.

We want this this network to have a Facebook / Twitter feel. We want users to be able to scroll through recent posts from other users that they are following. User should be able to post any articles related to a stock or a company, tag the company, and write a small synopsis of the article. These posts will be used to project the future stock prices, but will be most beneficial for the user. The user will be able to see what is going on in the world of stocks and will be able to make more educated investments based on what they have read.

It is becoming increasingly important to be able to view stock prices and perform research on new stocks while on the go. Customers are always calling while on the bus or the train and asking how their stocks are performing and asking for advice on what we think they should do. This is because people nowadays are constantly traveling from place to place and are not at home staring at a computer screen all day. They can check a few times a day when they are at their computers, but the problem with that is that the stock market is constantly changing. Something that we believe will help is making this network something that can be accessed by phones while our customers are on the go. A mobile network will allow our users to be constantly updated with the current trends and predictions of the stock market. No longer will it be a problem that our customers are not able to check on their stocks while on the go. They will be allowed to check on their stocks during their downtime, while standing in a line or sitting on a bus. This mobile network will be extremely beneficial to everyone involved, from the most diehard investors to even the simplest of investors. It will be especially beneficial to some of our customers who use the market to do day trading and penny trading. It is very important that they are able to buy and sell on the go because they may need to buy and sell at a seconds notice. This will allow them and as well as all users to constantly be able to check to see how their stocks. It will keep them informed and will eliminate the need for them to call our company to check in with how their stocks are doing.

The biggest part of this network is the stock projections. Most stock projection algorithms today only take into account price history when projecting how the application is going to perform. The problem with this is that the average stock does not change solely based on historical trends. There are a lot of other outside factors may drastically affect a stock's price. A company may be absorbed into another company; this may make investors believe that it is a safer investment because it is not incorporated into a larger more stable company. A company may be hit by a natural disaster such as an earthquake or a hurricane and may completely stop production. Sometimes things not even related to the specific company can affect their stock prices, such government policy. If the government changes something in its spending policy it can change investor's attitude towards how the economy itself is going to perform. This may make investors very confident in the economy and may make them invest more, or it may make investors very cautious and may make them pull their money out of the stock markets. We want our projection algorithm to take into account user input in order to make its projection. We want it to search through anything shared on the network such as new articles or even just posts. We want it to judge how positive or negative the article is or the post is and use it help with the projection.

Today's world has turned toward social media for almost everything involved in their live. It is a means of very efficiently spreading information and that is why it has been so effective. With the incorporation of social media into stock projections, there is nothing quite like this out there. This combined with top/bottom projections, stock searches, notifications, and favorites; all on a mobile device has the possibility to become something huge. We have the ability to revolutionize the way investors buy, sell, and research stock. We believe this application has the potential to be extremely successful.

Glossary of Terms

52-Week High/Low - The highest and lowest prices that a stock has traded at during the previous year.
http://www.investopedia.com/terms/1/52weekhighlow.asp

Algorithmic Trading - A trading system that utilizes very advanced mathematical models for making transaction decisions in the financial markets. The strict rules built into the model attempt to determine the optimal time for an order to be placed that will cause the least amount of impact on a stock's price. Large blocks of shares are usually purchased by dividing the large share block into smaller lots and allowing the complex algorithms to decide when the smaller blocks are to be purchased.
Read more:
http://www.investopedia.com/terms/a/algorithmictrading.asp#ixzz2Kcc9nHtS

Artificial Intelligence - An idea in the field of computer science that attempts to replicate the intricacies of the human mind through computer programming.

Artificial Neural Network - Neural networks is the developement of a computer system that models the human brain and its nervous system. also see (Artificial intelligence)

Autocorrelation - The correlation between the values of in a particular series and previous values across a certain interval.

Average volume- The amount of individual securities traded over a specified amount of time.

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

Bear Market - A securities market that perpetuates selling and declining prices based on the current fall of prices.

Beta - A measure of the volatility, or systematic risk, of a security or a portfolio in comparison to the market as a whole. A beta of 1 indicates that the security's price will move with the market. A beta of less than 1 means that the security will be less volatile than the market. A beta of greater than 1 indicates that the security's price will be more volatile than the market.
http://www.investopedia.com/terms/b/beta.asp

Bull Market - A securities market that perpetuates buying and rising prices based on the current increase in prices.

Buy and Hold - The acquisition of a tradable good for the long term rather than trying to profit over a quick turnover..

Charts - A image or display of a stock/security that plots price and/or volume (the number of shares sold).

Confidence Level - The probability that the value of a parameter falls within a specified range of values.

Data mining - Data processing using statistical algorithms to discover patterns and correlations in large, preexisting databases..

Dividend - A distribution of a portion of a company's earnings decided by the board of directors, to a class of its shareholders. The dividend is most often quoted in terms of the dollar amount each share receives (dividends per share). It can also be quoted in terms of a percent of the current market price, referred to as dividend yield.
http://www.investopedia.com/terms/d/dividend.asp

Earnings Per Share (EPS) - The portion of a company’s profit allocated to each outstanding share of a common stock. It is calculated as (Net Income - Dividends on Preferred Stock) / Average Outstanding Shares
http://www.investopedia.com/terms/e/eps.asp

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

Fundamental Analysis - The analytical method by which only the sales, earnings and the value of a given tradable's assets may be considered.

Equity - see (stock)
Read more: http://www.investopedia.com/terms/s/stock.asp#ixzz2KcQv75Fh

Institutional Ownership - The ownership of a company's stock by mutual funds, pension funds, and other institutional investors, generally expressed as percentage of outstanding shares.
http://financial-dictionary.thefreedictionary.com/Institutional+Ownership

Market Capitalization - The total dollar market value of all a company's outstanding shares It is calculated by multiplying a company's shares outstanding by the current market price of one share.
http://www.investopedia.com/terms/m/marketcapitalization.asp

Market Price - The current price at which an asset or service can be bought or sold. Economic theory contends that the market price converges at a point where the forces of supply and demand meet.
http://www.investopedia.com/terms/m/market-price.asp

Opening Price - The price at which a security first trades upon the opening of an exchange on a given trading day.
http://www.investopedia.com/terms/o/openingprice.asp

Outstanding Shares - Stock currently held by investors, including restricted shares owned by the company's officers and insiders, as well as those held by the public.
http://www.investopedia.com/terms/o/outstandingshares.asp

Pattern recognition-Task performed by a network trained to respond when an input vector close to a learned vector is presented. The network "recognizes" the input as one of the original target vectors.

Price-Earning Ratio (P/E Ratio) -A valuation ratio of a company's current share price compared to its per-share earnings. It is calculated as the Market Value divided by the Earnings Per Share (EPS)
http://www.investopedia.com/terms/p/price-earningsratio.asp

Profit - A financial benefit that is realized when the amount of revenue gained from a business activity exceeds the expenses, costs and taxes needed to sustain the activity. Any profit that is gained goes to the business's owners, who may or may not decide to spend it on the business.
Profit=Total Revenue-Total Expenses

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

Shares: see (stock)
Read more: http://www.investopedia.com/terms/s/stock.asp#ixzz2KcQv75Fh

Short Selling - The selling of a security that the seller does not own, or any sale that is completed by the delivery of a security borrowed by the seller. Short sellers assume that they will be able to buy the stock at a lower amount than the price at which they sold short.

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

Also known as "shares" or "equity."
Read more: http://www.investopedia.com/terms/s/stock.asp#ixzz2KcQv75Fh

Stock Symbols: see (Ticker symbols)
Read more: http://www.investopedia.com/terms/s/stocksymbol.asp#ixzz2KcTNCfge

Technical Analysis- A form of market analysis that studies demand and supply for securities and commodities based on trading volume and price studies using charts and modeling techniques.

Ticker symbols- A unique series of letters assigned to a security for trading purposes. NYSE and AMEX listed stocks have three characters or less. Nasdaq-listed securities have four or five characters. If a fifth letter appears, it identifies the security as other than a single issue of common stock. They are also known as "ticker symbols."

Ex: GOOG
Read more: http://www.investopedia.com/terms/s/stocksymbol.asp#ixzz2KcTNCfge

Total Expenses - 1. The economic costs that a business incurs through its operations to earn revenue. In order to maximize profits, businesses must attempt to reduce expenses without also cutting into revenues. Because expenses are such an important indicator of a business's operations, there are specific accounting rules on expense recognition.
2. Money spent or costs incurred that are tax-deductible and reduce taxable income.

Total Revenue- The amount of money that a company actually receives during a specific period, including discounts and deductions for returned merchandise. It is the "top line" or "gross income" figure from which costs are subtracted to determine net income. Revenue is calculated by multiplying the price at which goods or services are sold by the number of units or amount sold.

Volume - The amount of shares traded during a particular period of time. Simply put, the amount of shares traded between sellers and buyers as a measure of activity.
http://www.investopedia.com/terms/v/volume.asp

System Requirements

Functional Requirements

Label	Priority	Description
Algorithm	5	The system must take input data from the database and output a prediction.
Research	5	The system must search through articles and documents for keywords to help in making a prediction.
Updating	3	The system must pull data from the server when the user requests it.
Favorites	1	The system must store a list of stocks that the user chooses.
Search	3	The system must take the users input and search for the stock's data that the user requested.
Login	3	The system must take the users inputted login information and match it to the system, then either allow access or deny it.
Register	3	The system must allow new users to create an account.

Profile	3	The system must load the users profile once they have logged in.

	UC 1	UC 2	UC 3	UC 4	UC 5	UC 6	UC 7	UC 8	UC 9
Algorithm	X					X			
Research	X					X			
Updating	X	X	X	X	X	X		X	X
Favorites						X	X		
Search	X	X			X		X		
Login		X	X	X					
Register		X	X	X				X	X
Profile		X	X	X	X				X
Max Priority	5	5	3	3	3	5	3	3	3
Priority Weight	16	15	12	12	9	14	4	6	6

Nonfunctional Requirements

Label	Priority	Description
Functionality	5	The system must be an Android device with internet capabilities
Usability	5	The system shall use the standard Android user interface.
Reliability	3	The system will handle exceptions by prompting the user the option to either force close or wait for the process.
Performance	4	The highest acceptable latency time for the users will be 3 seconds for events not involving the internet and 30

		seconds for events that do involve the internet.
Supportability	1	The system will be maintained be the Development Team when dictated by major android development changes or stock exchange changes.
Implementation	5	The system will be implemented in Java, and will connect to a database that will also be implemented in Java.
Testability	4	The system will use the Android Emulator for testing.
Documentation	2	The system will be well documented via reports and possibly and API Documentation.

On-Screen Appearance Requirements

Label	Priority	Description
Home	5	This brings you back to the home screen of the application when it is clicked
Favs	3	This shows you the stocks that you have marked as favorites so that you can easily access and refer to them without searching for them again
Social	5	When you click social it takes you to a page to search for people, companies, and posts to find out if an outside factor will affect the price of the stock beyond our predictions
Post	5	This button allows you to create posts for the social network to inform others about stock information and news stories

	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9
Home	x								

Favs						x	x		
Social		x		x	x			x	x
Post			x						
Search	x	x				x	x		
Max Priority	5	5	5	5	5	5	5	5	5
Total Priority	10	10	5	5	5	8	8	5	5

Functional Requirements Specification

Stakeholders

There are many people that have an interest in this system. The following is a list of interested people or organizations:

1. Users/Investors
2. Administrators
3. Sponsors

Actors and Goals

The table below describes the actors in play and their goals.

Actor	Type	Goal
User/Investor	Initiating	To create or login into his or her account and be able to see and edit their information.
User/Investor	Initiating	To invest in stocks with as much data to help them make a decision of Sell, Buy, or Hold.
Administrator	Initiating	To login into the Admin account and view the application.
Administrator	Initiating	To manage and maintain the application and the server.

Profile	Participating	To provide relevant information about the user and his or her portfolio.
Predictor	Participating	To give the user information about the future of the stock to aid in their decision.
Database	Participating	To provide and store any relevant information about stocks and the users.

Use Case Casual Descriptions

User Side Use Cases:
Use Case 1: Search Stock
The user of the application wants to use the app to find the prediction of how a certain stock will do and see its history. Once the user is logged in they will be sent to the home page. The user will input the stock symbol or name of the stock in the search bar. Once the stock is input the user will hit search and the application will send a request to the database asking for all current information available for the stock. The database will then send the information to the application. The application will then display all current information associated with the searched stock, such as current price, current prediction, and a graph with the history of the stock for the past 6 months. Once the user is satisfied he can then exit the application or return to the home screen.
Responds to Requirements: Search, Updating

Use Case 2: Check News Feed
The user of the application wants to use the app to look through recent post. Once logged into the application the user will be brought to the home screen. The application will send a request to the database for the most recent posts by users in which our user is following. Once a list is compiled it is then sent to the application. The application then displays the posts for the user. On the home screen the user will see the a list of posts starting from the most recent. Once the user is satisfied he can then exit the application or return to the home screen.
Responds to Requirements: Social, Updating

Use Case 3: Post News Feed
The user of the application wants to use the app to post information. Once logged into the application the user will be brought to the home screen.The application will send a request to the database for the most recent posts by users in which our user is following. Once a list is compiled it is then sent to the application. The application then displays the posts for the

user. From the home screen the user will select Social. The user will then be brought to the Social Page. From this page the user can type in his post in the box. Once the post in completed, the user will then select Post to submit the post. Once submitted the application will send the information to the database. The database will store the information. Once the information is stored the database will send the application a notification of success. If the information is not stored correctly the database will then send the application a notification of failure. If it is a success the user will then be brought back to the Social Page. If it is a failure, the user will be brought back to the Social Page where he will be asked to input the information again. Once the user is satisfied he can then exit the application or return to the home screen.
Responds to Requirements: Social, Updating

Use Case 4: Visit Friends Profile
The user (a) of the application wants to visit another users(b) profile whom the user(a) is following. Once logged into the application the user will be brought to the home screen. The application will send a request to the database for the most recent posts by users in which our user is following. Once a list is compiled it is then sent to the application. The application then displays the posts for the user. From the home screen the user will select Social. The user will then select the "Following" button. The application will send a request to the database for the list of all users(b) our user(a) is currently following. Once the list is found it will be sent to the application. The application will then display this information on the Following Page. The user(a) can then search through the list for the user(b) they are looking for. The user(a) can then select that user(b). The application then sends a request to the database for the profile and most recent posts of the selected user(b). Once the information is compiled, the database will then send the information to the application. The application will then display the information to the user(a) on the Profile Page. Once the user is satisfied he can then exit the application or return to the list of who he is following.
Responds to Requirements: Social, Updating

Use Case 5: Search User / Follow User
The user (a) of the application wants to search for another user(b) and follow them. Once logged into the application the user will be brought to the home screen. The application will send a request to the database for the most recent posts by users in which our user is following. Once a list is compiled it is then sent to the application. The application then displays the posts for the user. From the home screen the user will select Social. The user will then select the "Search" button. The user will be brought to the Search Page. The user(a) can then use the search bar to search for any other user. Once the user submits the search, the application then sends a request to the database for any user by that name or user name. Once a list is compiled the database will send the list to the application. The

application will then display the results of the search. The user(a) can then select one of the results. Once a selection is made, the application will then send a request to the database for the profile and most recent posts of the selected user(b). Once the information is compiled the database will then send the information to the application. The application will then display the information to the user(a). If the user wants to follow the searched user(b) and have their posts show up on their news feed , then the user(a) may select the "Follow" button. Once the follow button is selected the application then sends this request to the database. The database will then add this user to list of users(b) they are currently following. Once the user is satisfied he can then exit the application or return to the search
Responds to Requirements: Social, Updating

Use Case 6: View Favorites
The user of the application wants to view the current price and projected price of all of his favorite stocks. Once logged in the user will be brought to the home screen. The application will send a request to the database for the most recent posts by users in which our user is following. Once a list is compiled it is then sent to the application. The application then displays the posts for the user. The user will then select the favorites tab. Once selected the application will send a request to the database for the current price and projected price of all stocks saved in favorites. Once the database compiles all the information, it will be sent to the application. The application will then display the information to the user. The user may then scroll though his favorites and view their current price and their projected price. The user may then select one of the favorites. The application will then send a request to the database for that stocks current price, current projection, price history, and other information. Once the database compiles this information, it will be sent to the application. The user will be brought to the Favorites Page and the application will then display the information. Once the user is satisfied he can then exit the application or return to the favorites screen.
Responds to Requirements: Favorites, Updating

Use Case 7: Add Favorites
The user of the application wants to add a stock to his favorites. Once logged in the user will be brought to the home screen. The application will send a request to the database for the most recent posts by users in which our user is following. Once a list is compiled it is then sent to the application. The application then displays the posts for the user. The user will input the stock symbol or name of the stock in the search bar. Once the stock is input the application will send a request to the database asking for all current information available for the stock. The database will then send the information to the application. The application will then display all current information associated with the searched stock, such as current price, current prediction, and a graph with the history of the stock for the

past 6 months. The user will then select the "Add to Favorites" check box. Once selected the stock name and symbol will be saved in favorites. Once the user is satisfied he can then exit the application or return to the home screen.
Responds to Requirements: Favorites, Updating

Use Case 8: Create Account
The user of the application wants to create a new account. The user will open the application and be brought to the login screen. Once at the login screen the user will then select "Create an Account." The user will then be brought to an account creation page. The user will fill out the required information, such as name, username, password, email address, job, qualifications, and others. Once completed and submitted, the information will be sent to the database. The database will then check the availability of the username. If the username is not available the database will send a notification to the application. The application will then ask the user to choose another username. If the username is available, then the database will set up the account, and log the user in. The user will then be brought to the home page.
Responds to Requirements: User, Updating

Use Case 9: Change User Information
The user wants to update the information on his profile. From the home screen the user will select "Settings". From the settings tab the user will select "Update Profile". The application will send a request to the database for information currently on the profile. The database will send the application this information. The application will then display the information and allow it to be edited. Once the user has completed his editing, he would submit the information. Once submitted the application will send the database the updated information. The database will save the information and send the application a notification of success of failure if it does not save properly. Once the application receives a failure notification, the user will be asked to re-submit the information. Once the application receives a success notification the user will be brought back to the home screen.
Responds to Requirements: User, Updating

Server Side Use Cases:

Use Case 1: Updating Stock History
Database needs most recent price for stock. The database decides that a current price for a specific stock is not current enough (done every 15 minutes). Database then request most current price from API. API sends back most current price. The database then saves that price in that stocks history.
Responds to Requirements: Updating

Use Case 2: Projection
Projections need to be made based on stock history and user input. Projection algorithm requests 6 month history from database. The database sends the history of the stock. The algorithm requests all recent posts for the database and information from other sources. The database sends all recent posts and information. The algorithm searches through any posts and information looking for information pertaining to the stock. Decision is made about how positive or negative the information found is. Once completed the algorithm calculates a projected price for a set amount of time. The projected price is sent to the database. The database saves the projected price in that stocks history.
Responds to Requirements: Updating, Algorithm, Research

Fully Dressed Descriptions

Use Case UC-1: Search Stock

Related Requirements: Updating, Search

Initiating Actor: User

Actor's Goal: To Search a Stock

Participating Actors: Predictor, Database,

Preconditions: User knows stock he wants to search

Success End Condition: Stock history and stock prediction is displayed

Failed End Condition: Stock symbol or name is invalid, error screen is displayed

Extension Points:

Flow of Events for Main Success Scenario:

include::Login()

← 1. **System** displays homescreen and allows user to choose any options on homescreen
or enter stock symbol or stock name into search bar

→ 2. **User** enters in a stock name or symbol in search bar

← 3. **System** finds stock history and **Prediction** in the **Database** and displays it for the **User**

Flow of Events for Extensions (Alternate Scenarios):

2a. User enters invalid stock or symbol

← 1. **System** signals error to **User** and keeps the **User** at the homescreen

Any Step: Network Connection Fails (System should continuously monitor the health of network connections)

← 1. **System** detects network failure and signals error to **User**

Use Case UC-2: Check News Feed

Related Requirements: Updating, Search

Initiating Actor: User

Actor's Goal: To view post and opinions on the current values of stock prices

Participating Actors: Database, User/Investor

Preconditions: User has an account on the social network and is following certain companies or people. The user has also logged in to their account.

Success End Condition: Post of other users or companies is revealed on the homepage

Failed End Condition: The user does not have an account or the user has not selected anyone or anything to follow

Extension Points:

Flow of Events for Main Success Scenario:

include::Login()

← 1. **System** displays home page and allows user to choose any options on homepage

→ 2. **User** scrolls up and down the home screen to read post

Flow of Events for Extensions (Alternate Scenarios):

2a. User does not login

← 1. **System** Homepage displays a blank news feed.

← 1. **System** prompts user to make an account or sign into view news feed

Use Case UC-4: Visit Friend's Profile

Related Requirements: Updating, Social

Initiating Actor: User

Actor's Goal: To view the profile of another user on the social netowork.

Participating Actors: Database, User/Investor

Preconditions: User has an account on the social network and the user has also logged in to their account.

Success End Condition: The profile of another user is displayed to the current user.

Failed End Condition: The user does not have an account and cannot access the social network

Extension Points: N/A

Flow of Events for Main Success Scenario:

include::Login()

← 1. **System** displays home page and allows user to choose any options on homepage

→ 2. **User** navigates to the social network page

→ 3. **User** searches for the user whose profile they wish to view.

← 4. **Database** searches for the inputted user and displays the profile of the user if a match is found.

Flow of Events for Extensions (Alternate Scenarios):

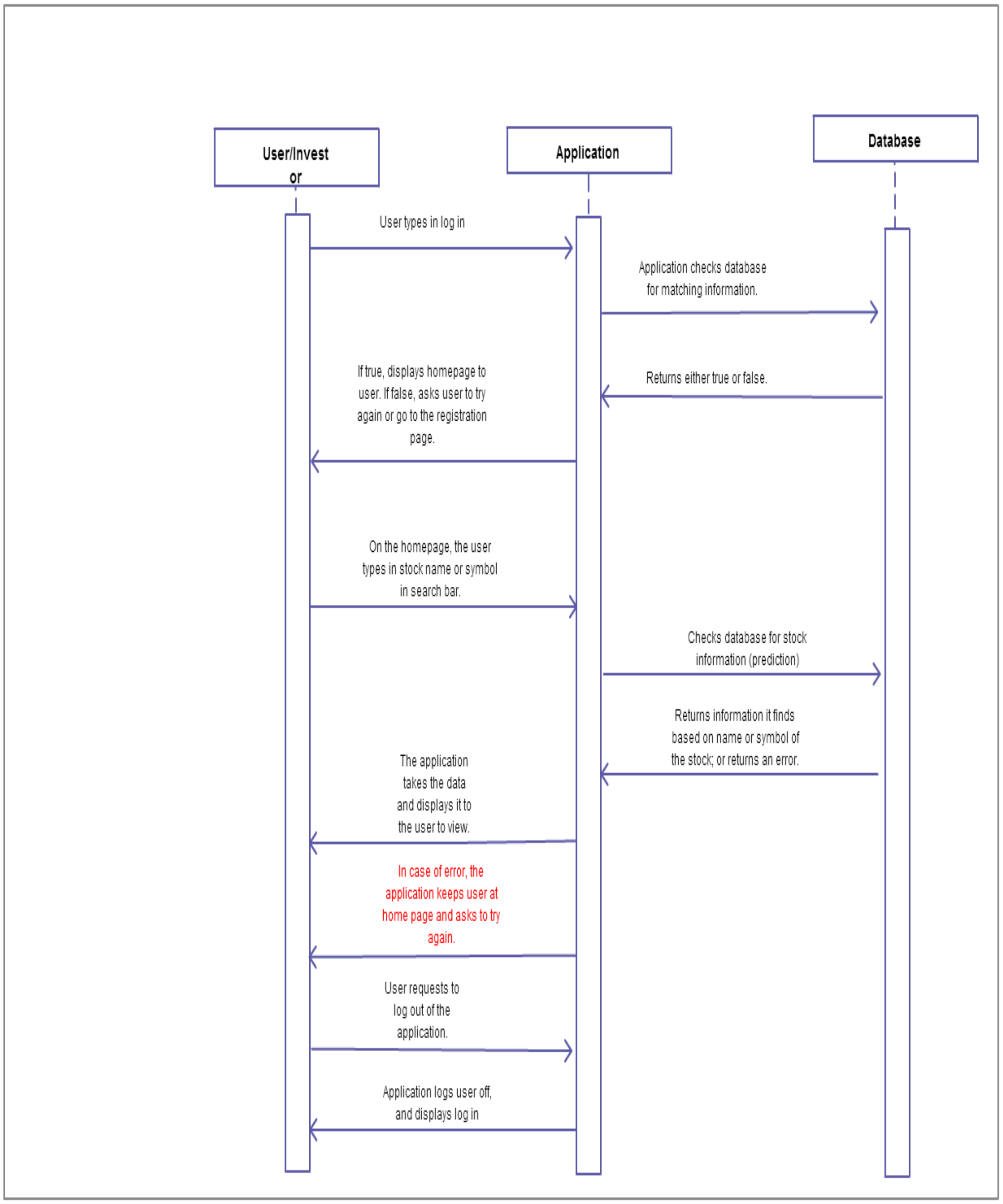
2a. User does not login

← 1. **System** redirects user to the login page when they attempt to access the social network page.

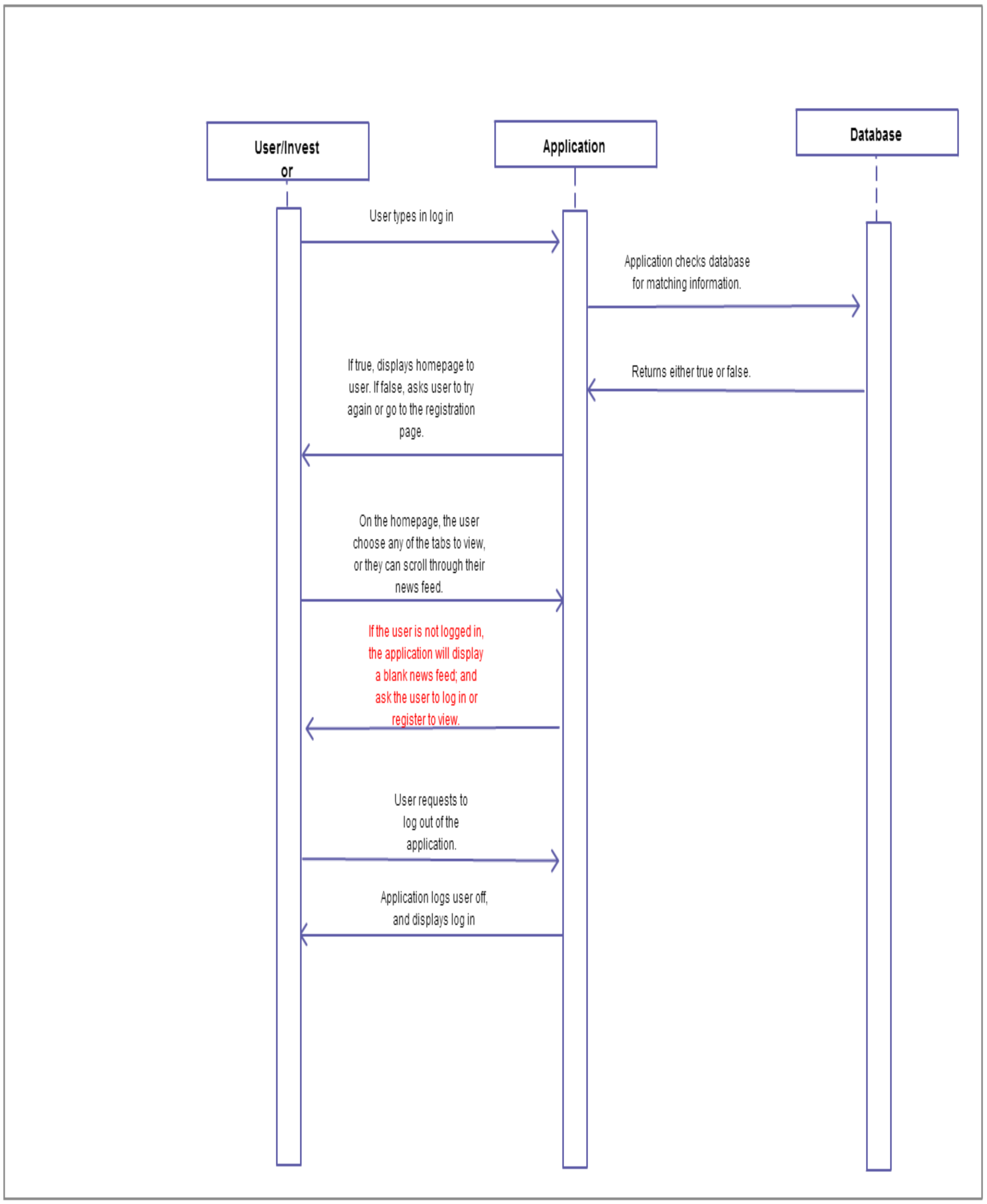
← 2. **System** prompts the user to login in order to access the social network.

System Sequence Diagrams:

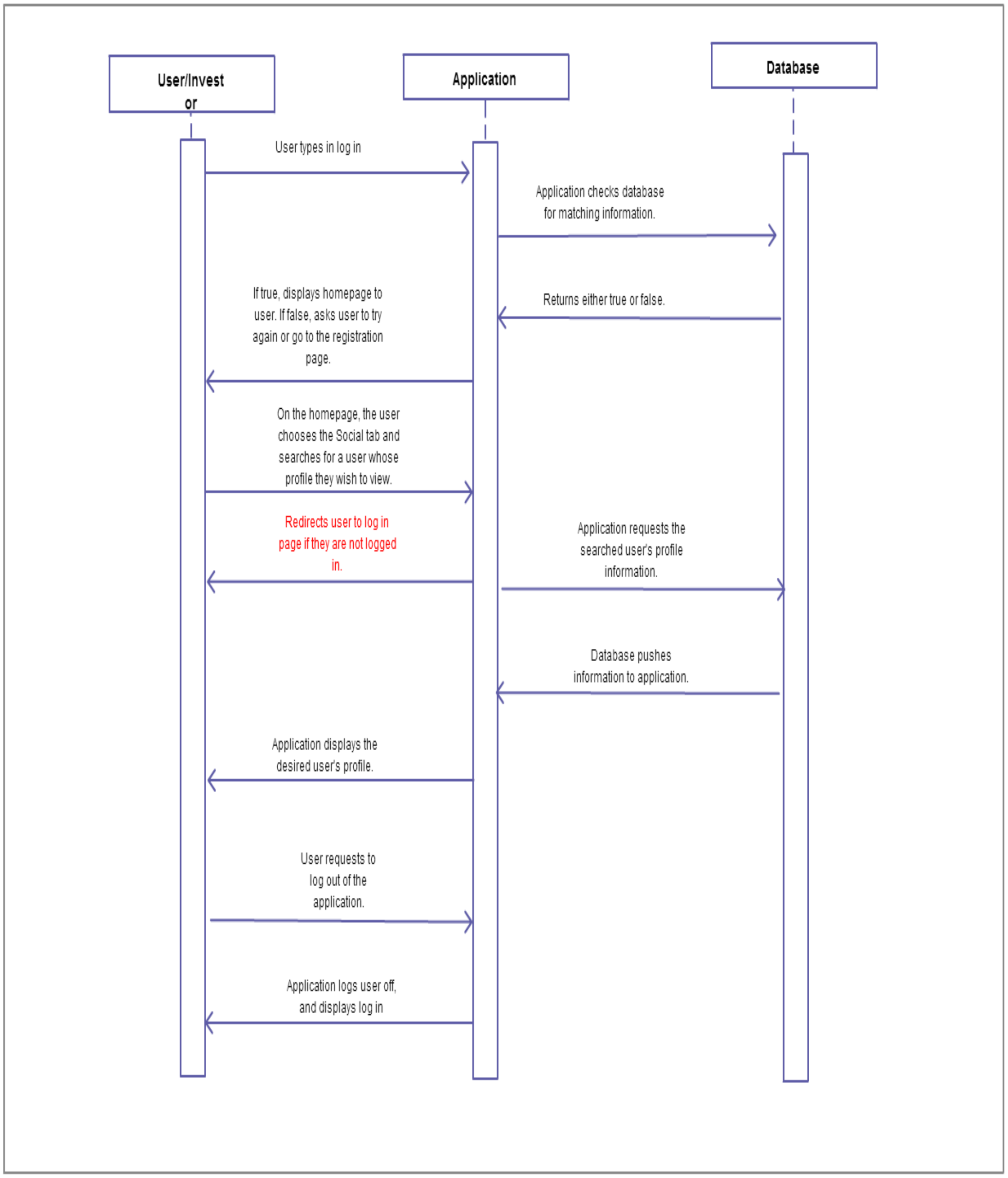
Use Case 1


User/Invest or
Application
Database
User types in log in
Application checks database for matching information.
Returns either true or false.
If true, displays homepage to user. If false, asks user to try again or go to the registration page.
On the homepage, the user types in stock name or symbol in search bar.
Checks database for stock information (prediction)
Returns information it finds based on name or symbol of the stock; or returns an error.
The application takes the data and displays it to the user to view.
In case of error, the application keeps user at home page and asks to try again.
User requests to log out of the application.
Application logs user off, and displays log in
[online diagramming & design] creately.com

Use Case 2:


User/Invest
or
Application
Database
User types in log in
Application checks database for matching information.
Returns either true or false.
If true, displays homepage to user. If false, asks user to try again or go to the registration page.
On the homepage, the user choose any of the tabs to view, or they can scroll through their news feed.
If the user is not logged in, the application will display a blank news feed; and ask the user to log in or register to view.
User requests to log out of the application.
Application logs user off, and displays log in

Use Case 4:


User/Investor
Application
Database
User types in log in
Application checks database for matching information.
Returns either true or false.
If true, displays homepage to user. If false, asks user to try again or go to the registration page.
On the homepage, the user chooses the Social tab and searches for a user whose profile they wish to view.
Redirects user to log in page if they are not logged in.
Application requests the searched user's profile information.
Database pushes information to application.
Application displays the desired user's profile.
User requests to log out of the application.
Application logs user off, and displays log in

Effort Estimation

Actor name	Description of relevant characteristics	Complexity	Weight
User/Investor	User/Investor is interacting with the system via a graphical user interface	Complex	3
Administrator	The actor is a person interacting via a graphical user interface.	Complex	3
Profile	Profile is another system interacting through a protocol.	Average	2
Predictor	Predictor is another system which interacts with our system through a defined application programming interface	Simple	1
Database	Database is another system interacting through a protocol.	Average	2
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 + 2 X 2 + 2 X 3 = 12

Use case	Description	Category	Weight
Search Stock (UC-1)	Simple user interface. 4 steps for the main success scenario. 3 participating actors (User, Predictor, Database)	Average	10
Check News Feed (UC-2)	Simple user interface. 3 steps for the all scenarios. 3 participating actors (User, Database)	Simple	5
Post News Feed (UC-3)	Moderate Interface Design. 3+2=5 steps for all success scenarios. Two participating actors (User, Database).	Average	10
Visit Friends	Moderate user interface. 4 steps for the main	Average	10

Profile (UC-4)	success scenario. 3 participating actors (User, Profile, Database).		
Search User/Follow User (UC-5)	Moderate user interface. 3+1=4 steps for all scenarios. 3 participating actors (User, Profile, Database).	Average	10
View Favorites (UC-6)	Moderate user interface. 4+3=7 steps for all success scenarios. 2 participating actor (User, Database).	Average	10
Add Favorites (UC-7)	Simple user interface. 2 steps for the main success scenario. 1 participating actor (User).	Simple	5
Create Account (UC-8)	Complex User Interface. 5+3=8 steps for all success scenarios. 2 Participating actors (User, Database)	Complex	15
Change User Information (UC-9)	Complex User Interface. 8+3=11 steps for all success scenarios. 3 Participating actors (User, Profile, Database)	Complex	15

UUCW(home access) = 2 X simple + 5 X average + 2 X complex = 2 X 5 + 5 X 10 + 2 X 15 = 90

UUCP = UAW + UUCW = 12 + 90 = 102

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 = 102 * 0.95 * 1.01 = 97.87 or 98 Use Case Points

Duration = UCP * PF = 98 * 28 = 2744 hours

Domain Analysis

Concept Definitions

Responsibility Description	**Type**	**Concept Name**
Searches the database for the requested stock the user inputed.	D	StockSearcher
Pulls relevant information for the stock after a search is conducted.	D	DataPuller
Uses the neural networking and data mining to determine an accurate prediction for a stock.	D	StockPredictor
Updates the user interface based upon user input or database updates.	D	InterfaceUpdater
Aggregates all the user posts relevant to a certain stock to both the proprietary StockProphit feed and external feeds like Twitter.	D	FeedAggregator
Allows user to post to the StockProphit feed	D	FeedPoster
Allows user to follow(constantly see on news feed) all of another user's posts.	D	FollowUser
Contains the user's profile information.	K	UserProfile
Allows user to search for another user using the username of the user they are looking for.	D	UserSearcher
Pulls the user's favorites list from the database.	D	FavoritesPuller

Adds a user’s stock selection to his/her favorites list, and updates the database accordingly.	D	FavoritesAdder
Goes through the registration process and allows user to create an account.	D	AccountCreator
Updates the users personal information.	D	InfoUpdater
Grabs current stock data from the API.	D	CurrentDataGrabber
Stores login information for user (cookie, timestamp, etc.)	K	LoginInfo

Association Desciptions

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
FeedPoster↔FeedAggregator	FeedPoster conveys status to FeedAggregator and FeedAggregator puts the data in a list to display in the news feed	stores
UserProfile↔FollowUser	User profile sends a request to FollowUser and FollowUser signifies that the selected user will be followed	updates, stores
FavoritesPuller↔DataPuller	FavoritesPuller conveys a list of the users favorites stocks and the DataPuller returns the values of the stocks that are flagged as favorites	retrieves, updates
FavoritesAdder↔FavoritesPuller	FavoritesAdder conveys a stock to be added as a user’s favorite to FavoritePuller and FavoritesPuller recognizes that stock as a favorite from that point on	stores, updates
AccountCreator↔UserProfile	AccountCreator conveys name, email, and	conveys-name,

	other information to UserProfile and UserProfile updates the displayed information on the user's profile page	conveys-email, updates, stores
CurrentDataGrabber↔DataPuller	CurrentDataGrabber updates DataPuller with the new stock data and DataPuller stores that information for access later	conveys-stockinfo, stores
LoginInfo↔InfoUpdater	LoginInfo conveys the current logged in user to InfoUpdater and InfoUpdater updates the application to display news relevant to the logged in user	conveys-logininfo, updates, stores

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 of data to post to the StockProphit feed Ticker symbol to search user posts to find relevant	newsFeed tickerSymbol	FeedAggregator
String to post to the feed	postString	FeedPoster
Integer ID of user to follow	userId	FollowUser
User's real name User email address User's followed list	userFullName userEmail stocksFollowed	UserProfile
String containing the username to search.	username	UserSearcher
Stock ticker symbol of stock to add	tickerSymbol	

linked list containing all favorited stocks	favoritesList	FavoritesAdder
Email address of the user (username is parsed from email address) User's password User's full name (optional)	userEmail userPassword userFullName	AccountCreator
Inherits data from AccountCreator	userEmail userPassword userFullName	InfoUpdater
List that contains all the stock tickers to query the API with to get current/historical stock data.	stockList	CurrentDataGrabber
User's login token User's ip address User's timestamp to determine timeout	loginToken ipAddress loginTime	LoginInfo

Traceability Matrix

	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9
StockSearcher	X								
DataPuller	X								
StockPredictor	X								
InterfaceUpdater	X	X	X	X	X	X	X	X	X

FeedAggregator		X							
FeedPoster			X						
FollowUser					X				
UserProfile				X	X			X	X
UserSearcher				X	X				
FavoritesPuller						X	X		
FavoritesAdder							X		
AccountCreator								X	
InfoUpdater				X	X			X	X
CurrentDataGrabber	X								
LoginInfo								X	X
Max Priority	5	5	3	3	3	5	3	3	3
Priority Weight	14	14	9	9	9	12	12	9	9

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.

Preconditions: Default homepage displays a blank news feed.

Post conditions:
- A news feed was created and displayed.

<u>Operation Contract – Search Function</u>

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

<u>Operation Contract – Changing Tabs</u>

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.

<u>Operation Contract – Log Off Function</u>

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)

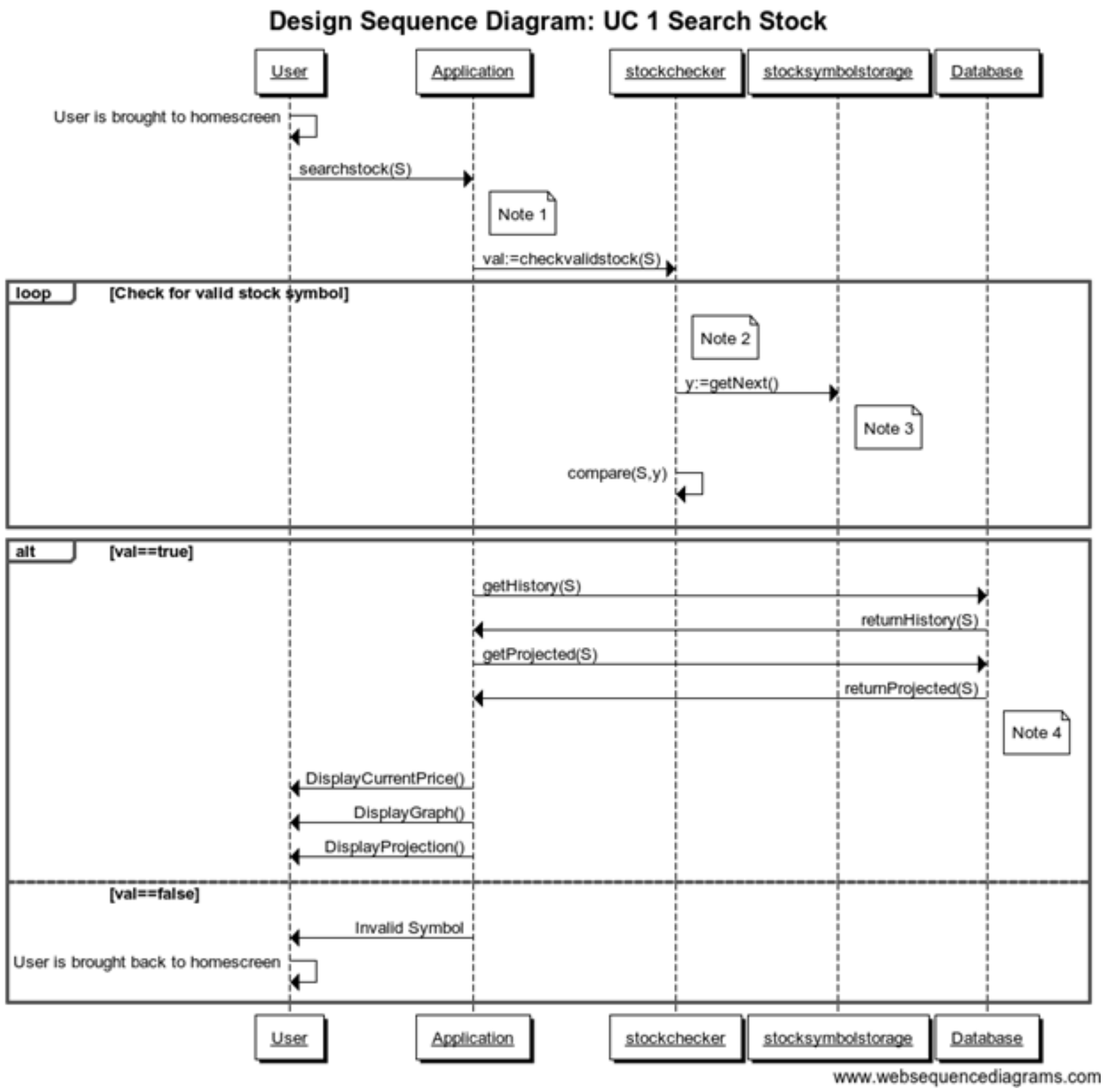
Exceptions: None.

Preconditions: Logging out will disconnect you from most of the features of the application.

Postconditions:
- The user's connection was ended.

Interaction Diagrams

Use Case 1

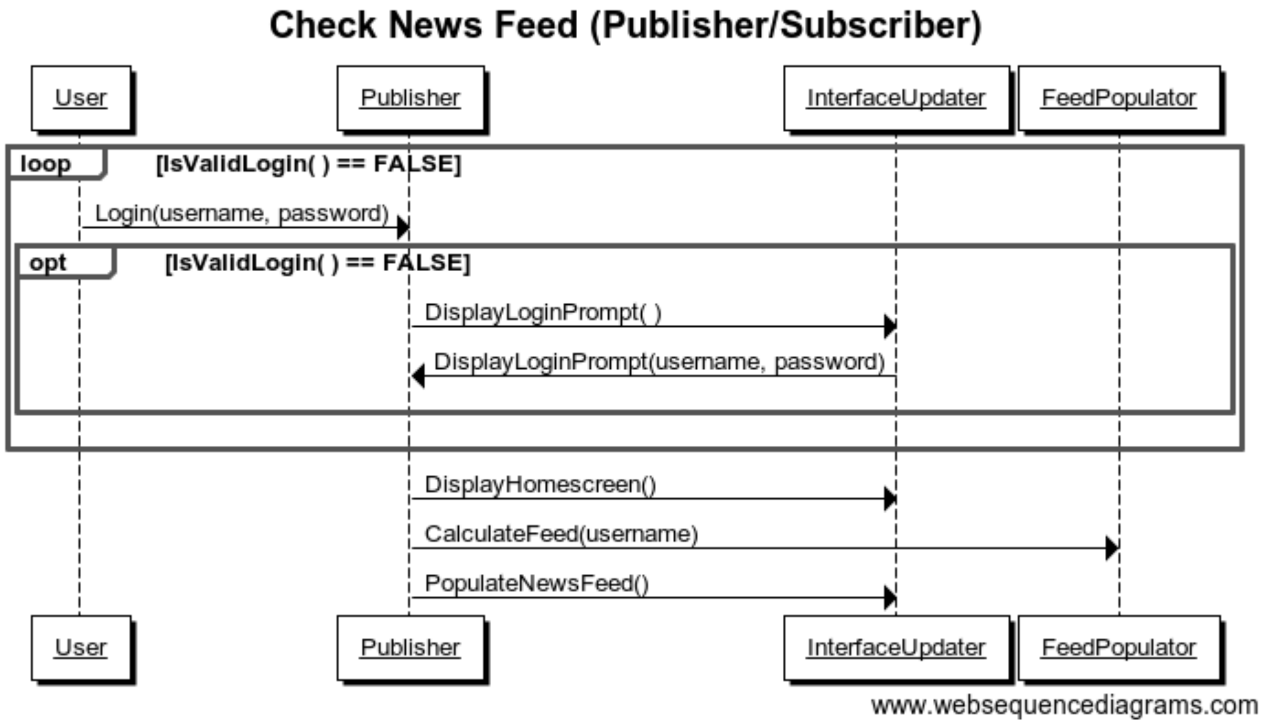


Note 1: The responsibilities of the application are to check if the stock entered by the user is valid, to request history and projections about the stock from the database, and to display the information to the user.
Note 2: The responsibility of stockchecker is to request a list of symbols from stocksymbolstorage and to compare with its inputted symbol to find if it is a valid symbol.
Note 3: The responsibility of stocksymbolstorage is to contain every symbol of every valid stock.

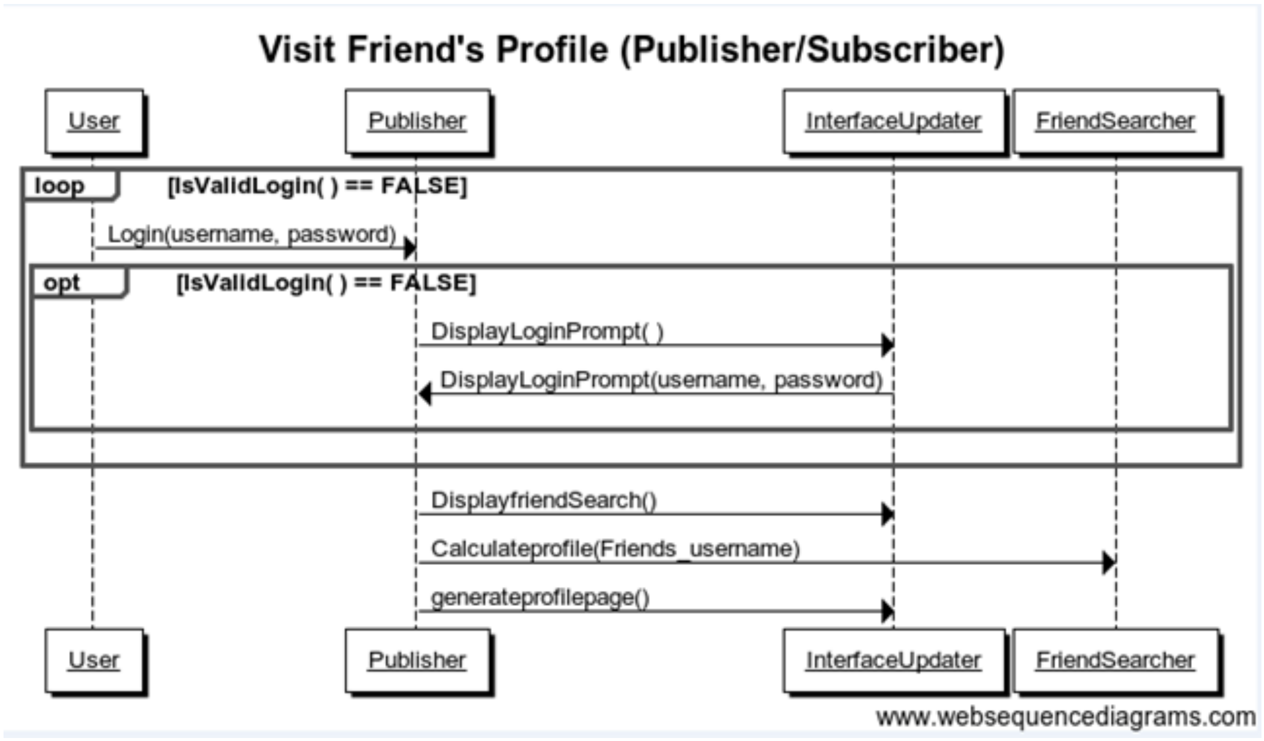
Note 4: The responsibility of the database is to contain all information about a valid stock, such as projected data and price history.

The design pattern used in the Use case is a State Pattern. It starts out in a state of requesting a stock. After the event of the stock being verified, it is put into a state of requesting the stock history. Once the history is found (event) then the data is sent to the application (state). Once the application has all history (event) the application requests stock prediction (state). Once the stock prediction in found (event) the data is sent to the application (state). Once that application has all the information (event) it is then displayed to the user.

This is a pretty basic form of the design. I used this design because most of the other designs would not have worked as well. Using a publisher subscriber pattern would have required the database to be a subscriber and had the database constantly checking for all the possibilities of requests that it could get. It is easier and faster just to send it a request. A decorator would have used additional processing of the results which was not needed, and would only slow it down. Command pattern may have also worked in this situation. It was a pretty basic use case and most designs probably would have worked just as well.


Check News Feed (Publisher/Subscriber)
User
Publisher
InterfaceUpdater
FeedPopulator
loop
[IsValidLogin( ) == FALSE]
Login(username, password)
opt
[IsValidLogin( ) == FALSE]
DisplayLoginPrompt( )
DisplayLoginPrompt(username, password)
DisplayHomescreen()
CalculateFeed(username)
PopulateNewsFeed()
www.websequencediagrams.com

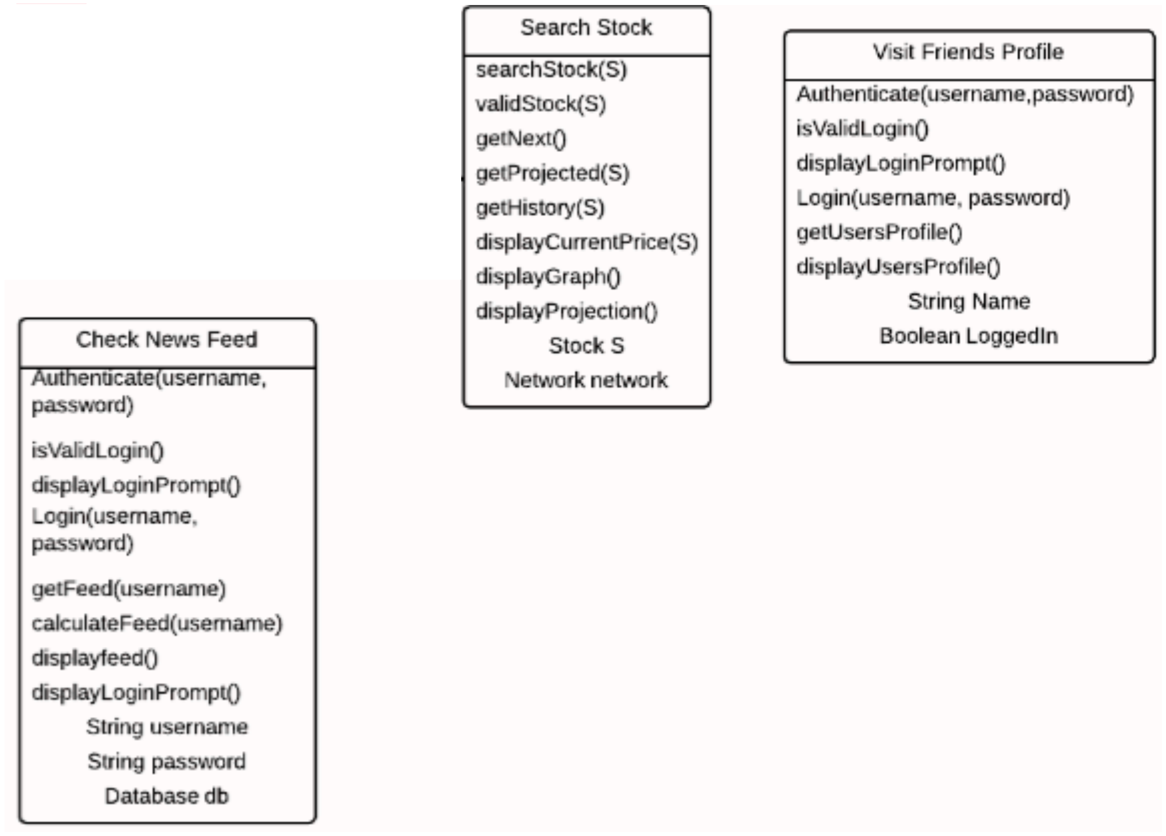
This use case is initiated when the user starts the application. A login screen is initially displayed and the user enters their login information. This information is passed to the publisher then the publisher and the initial conditions for the system are met. The publisher registers the InterfaceUpdater and FeedPopulator subscribers. The InterfaceUpdater displays the homescreen, which is the same for all users. The FeedPopulator gets the list of the people that a user is following and puts their most recent posts in an array to display to the logged in user. Then this information is used by the InterfaceUpdater to display the user's relevant news feed.


Visit Friend's Profile (Publisher/Subscriber)
User
Publisher
InterfaceUpdater
FriendSearcher
loop
[IsValidLogin( ) == FALSE]
Login(username, password)
opt
[IsValidLogin( ) == FALSE]
DisplayLoginPrompt( )
DisplayLoginPrompt(username, password)
DisplayfriendSearch()
Calculateprofile(Friends_username)
generateprofilepage()
User
Publisher
InterfaceUpdater
FriendSearcher
www.websequencediagrams.com

This use case is initiated when the user starts the application. A login screen is initially displayed and the user enters their login information. This information is passed to the publisher then the publisher and the initial conditions for the system are met. The publisher registers the InterfaceUpdater and FriendSearcher subscribers. The InterfaceUpdater displays the FriendSearch page, which is the same for all users. The generateprofile gets the person that a user is following and checks whether or not their following them based on the Calculateprofile function. Then this information is used by the InterfaceUpdater to display the user's profile. Publisher subscriber is a good pattern for this code because it utilizes loose coupling and other users friend information does not have to be accessed the user who is searching his friend.

Class Diagram and Interface Specification

Class Diagram



Data Type and Operation Signatures

Search Stock

-- Public searchStock(S)
The particular stock the user is searching for

--Public validStock(S)
Checks to see if the stock actually exists, by checking it against a list of valid stocks in a hash table

--Private getNext()
--Public getProjected(S)
Gets the predicted value of a stock

--Public getHistory(S)
Checks how stock performed in the past

--Public displayCurrentPrice(S)
--Public displayGraph()
--Public displayProjection()
Outputs the prediction to the UI

+ Public Stock S
+Private Network network
+Private Hashtable validStockList

Visit Friend's Profile

--Private Authenticate(username, password)
Does Authentication on username and password

--Private isValidLogin()
Checks if login is valid, this is where a token is checked to see if the user if login is valid

--Public displayLoginPrompt()
--Private Login(username, password)
Token is given to to user if info is valid

-- Public getUsersProfile()
--Public displayUsersProfile()
Displays another users profile on the UI

+Public String username
+Public String password
+Private Boolean LoggedIn
The local login token

Check News Feed

--Private Authenticate(username, password)
--Private isValidLogin()
--Public displayLoginPrompt()
--Private Login(username, password)
--Public getFeed(username)
Gets the news feed for the user whose name is passed in

--Private calculateFeed(username)
--Public displayFeed()
Displays feed to the UI

--Private displayLoginPrompt()
+Public String username
+Public String password
+Private Database db
**The Login database, containg private user info including password and email. (Passwords will be hashed for safety)*

OCL Contracts

context Search_Stock::SearchStock(ticker : String) : int[]
pre validStock(S) == true

context Visit_Friends_Profile::Authenticate(username : String, password : String) : boolean
pre displayLoginPrompt()
post LoggedIn== true

context Visit_Friends_Profile:getUsersProfile(user : String) : void
pre isValidLogin(user) == true
post displayUsersProfile()

context Check_News_feed::getFeed(username : String) : void
pre calculateFeed(username)
post displayFeed()

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.

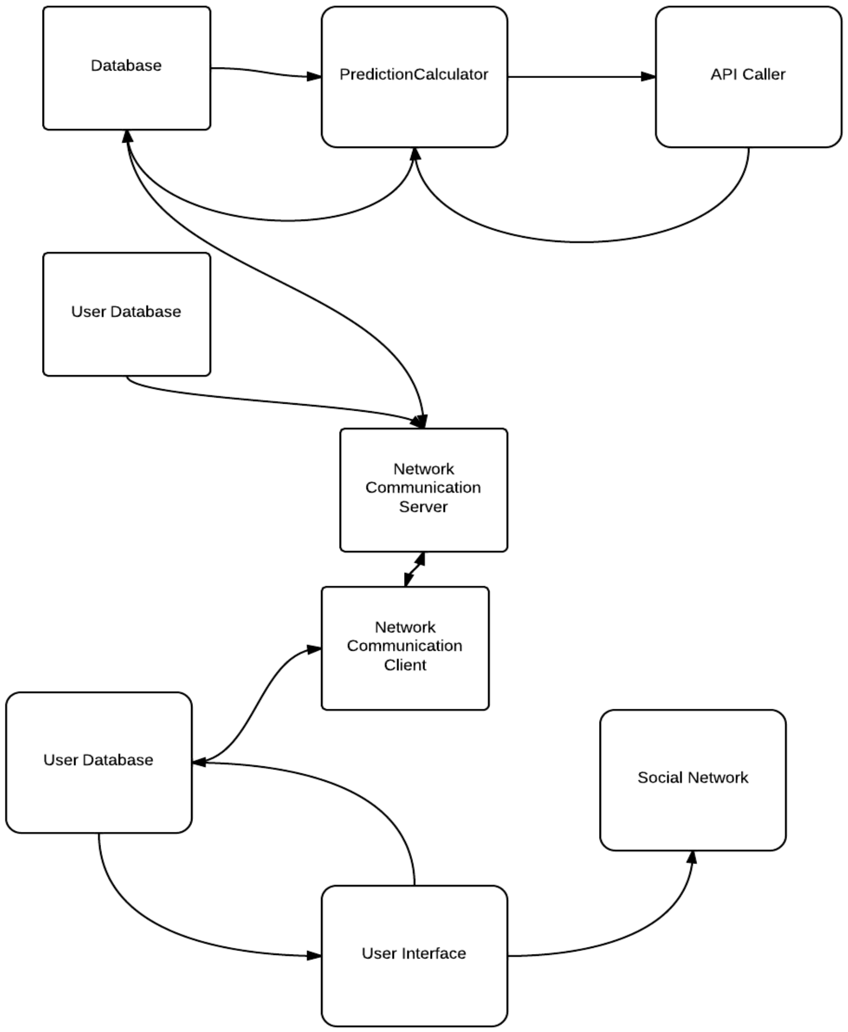
Communication:

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, PredictionCalculator, API Caller, 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 PredictionCalculator 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.



C. 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 Kryonet 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:**

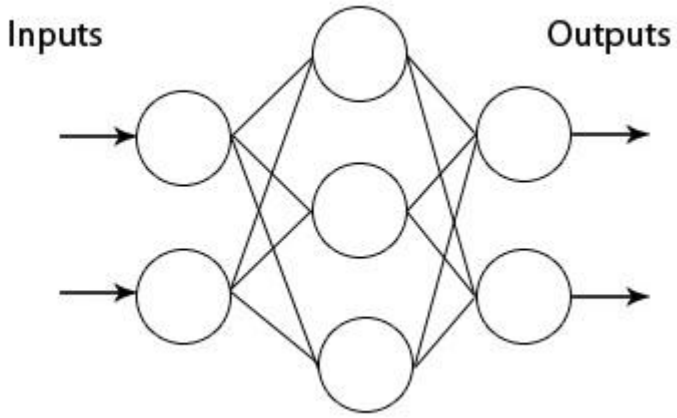
Minimum 4 Gb. **Network connection:** Minimum requirement is broadband connection which can range from 384 kbit/s to over 20 Mbit/s.

Algorithms and Data Structures

The main algorithm *Stock Prophit* 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 multi layer perceptron. All that this means is that the input goes through many layers (i.e. operations or computations) until it reaches the output.



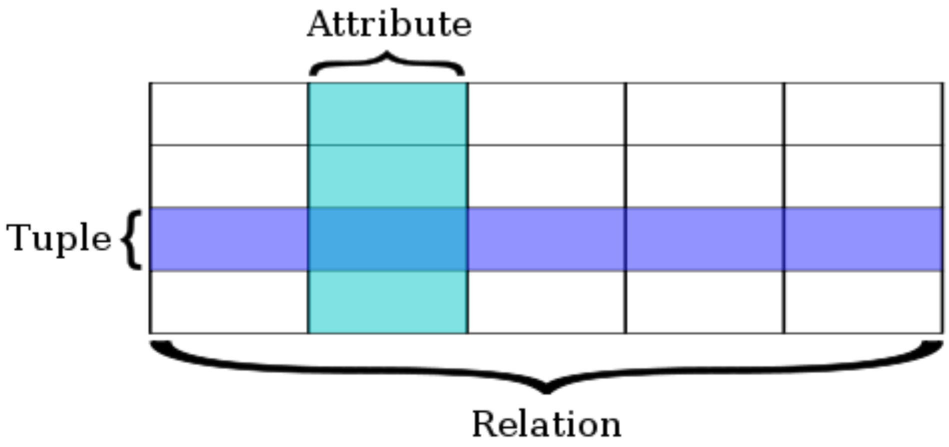
Above is a pictorial representation.

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 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

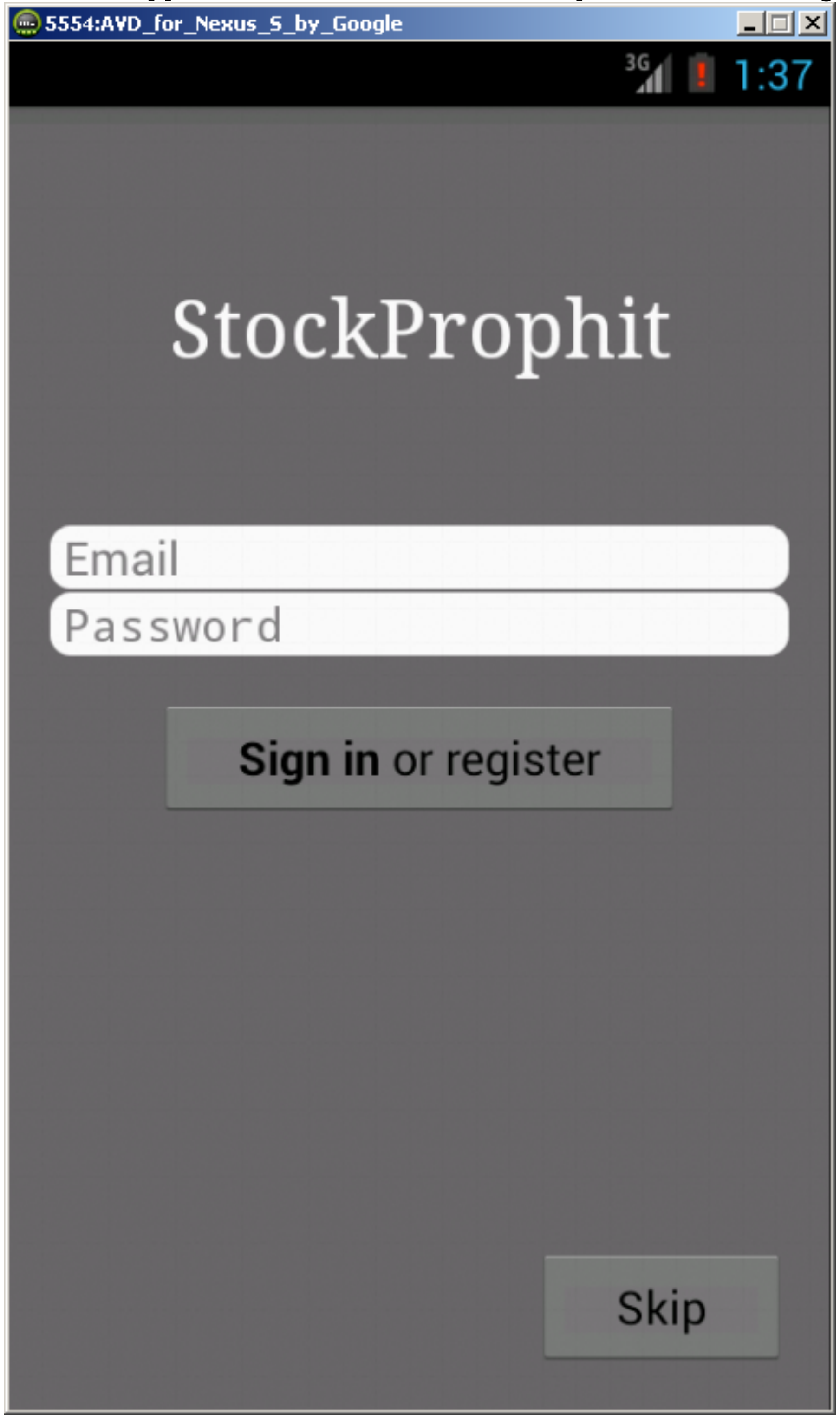
The user database will store the usernames and passwords of the users on our application. It will also use a relational database to store a string email/username and its respective hashed + salted password. A relational database is a good choice for this database as we will already be implementing the 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

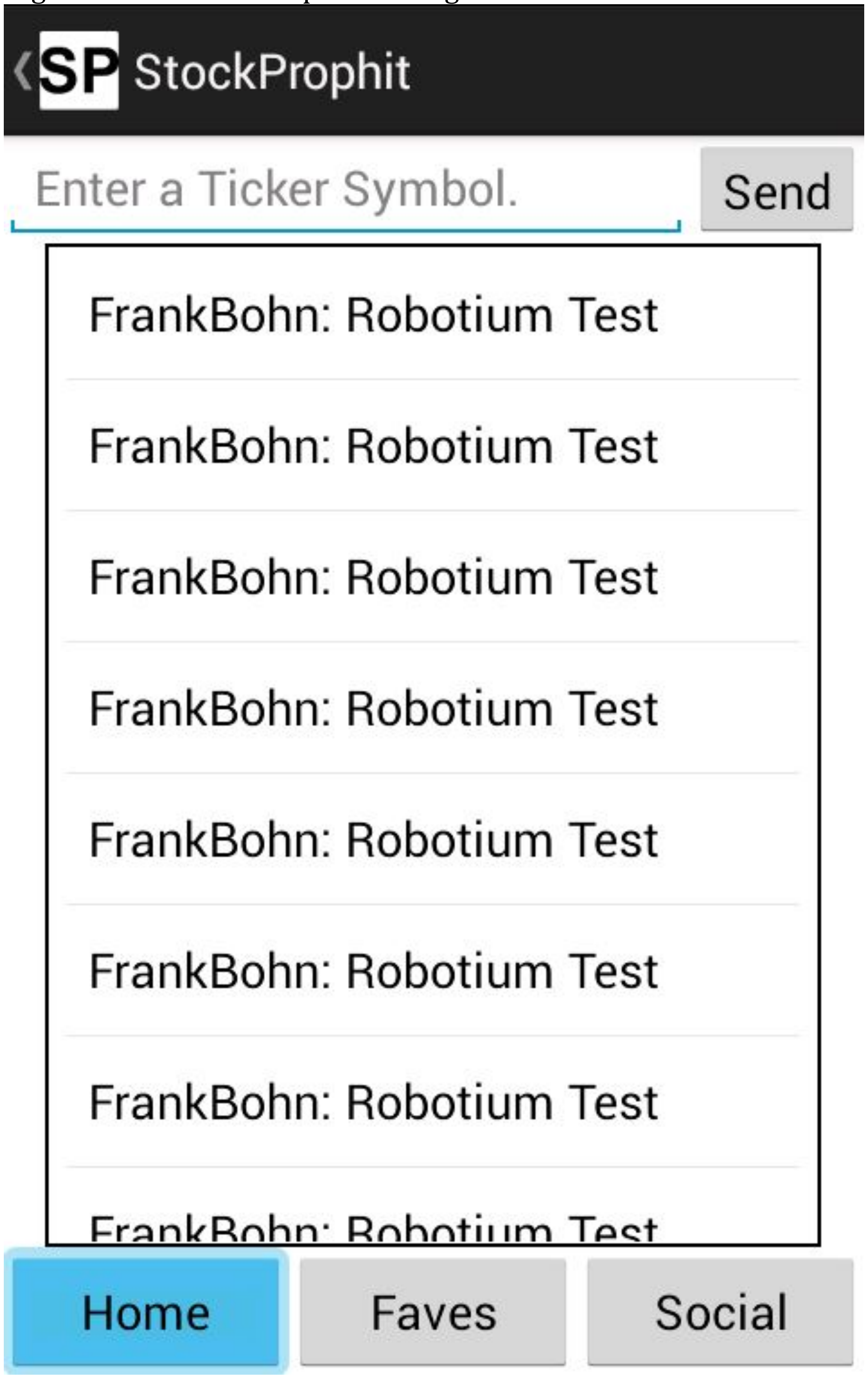
To implement our neural network, we will likely use a hashmap to store the frequency of found words. A hash map is a good implementation because it can be easily searched with efficient search methods like quicksort and its key-value setup is similar to the needed word-frequency setup for analysis with the neural network.

User Interface Design and implementation

When the application is launched, the user is presented with a login screen shown below:



The user has the option to create an account, login to an existing account, or skip registration. All three options bring the user to the home screen:



FrankBohn: Robotium Test

FrankBohn: Robotium Test

FrankBohn: Robotium Test

FrankBohn: Robotium Test

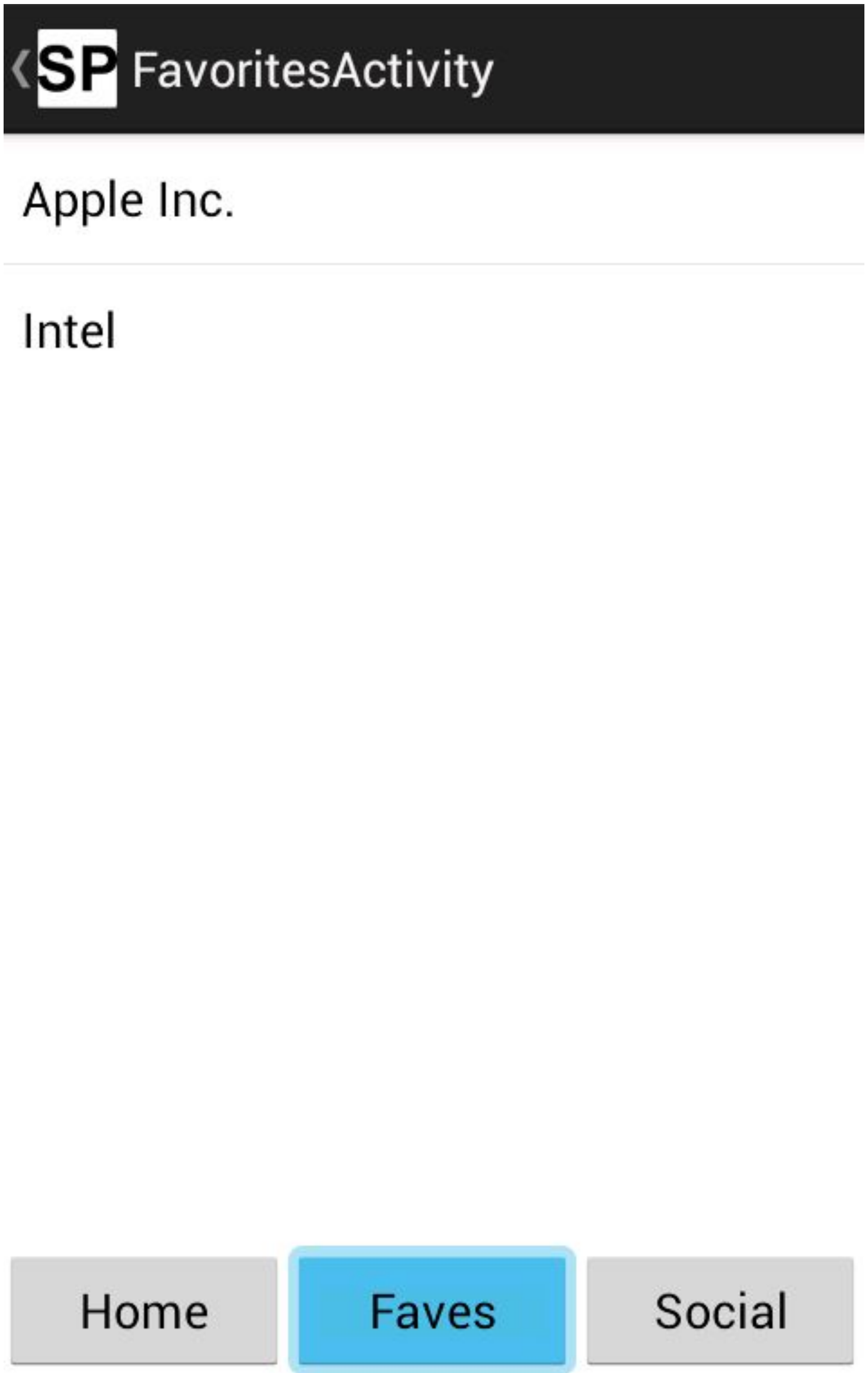
FrankBohn: Robotium Test

FrankBohn: Robotium Test

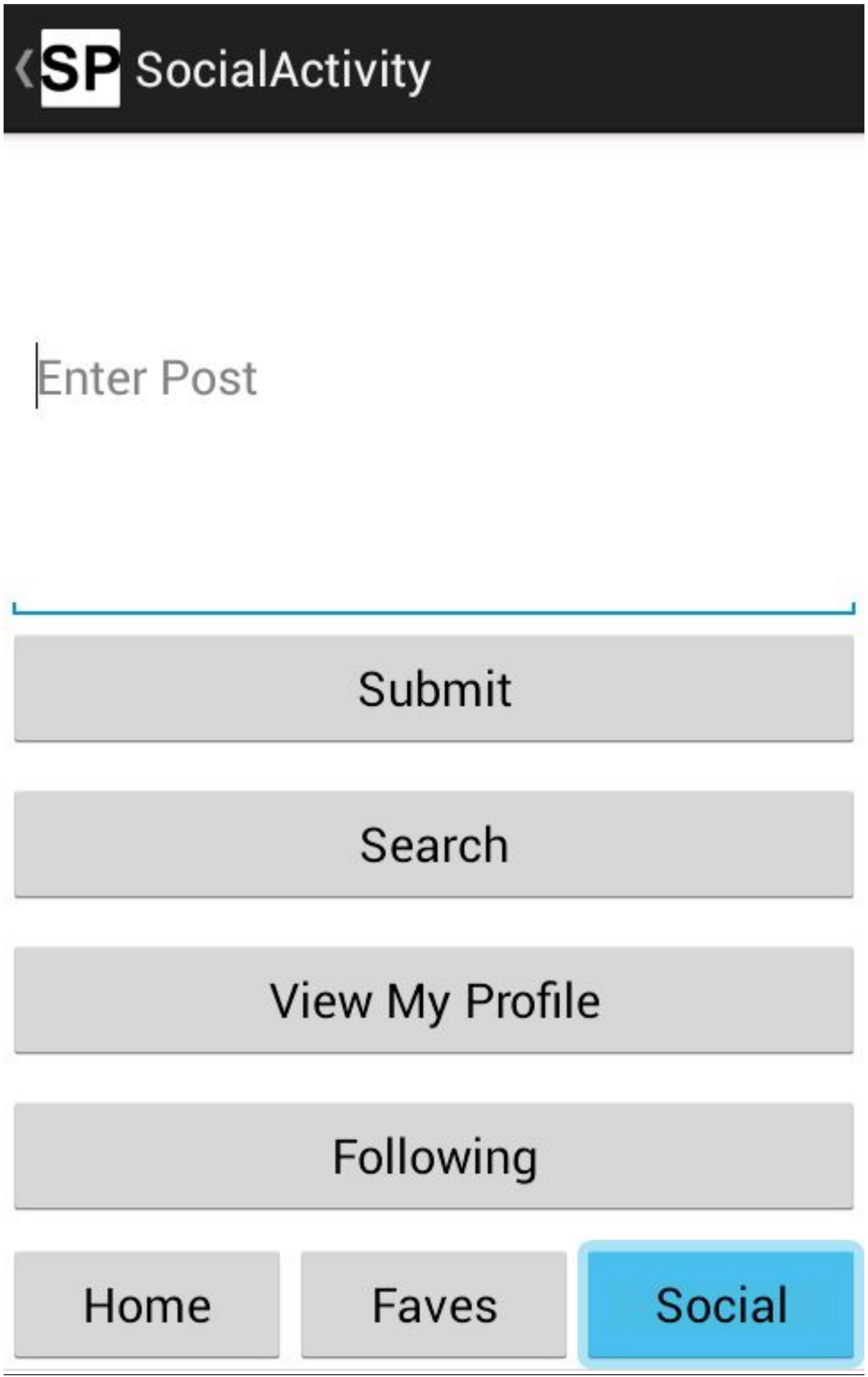
FrankBohn: Robotium Test

FrankBohn: Robotium Test

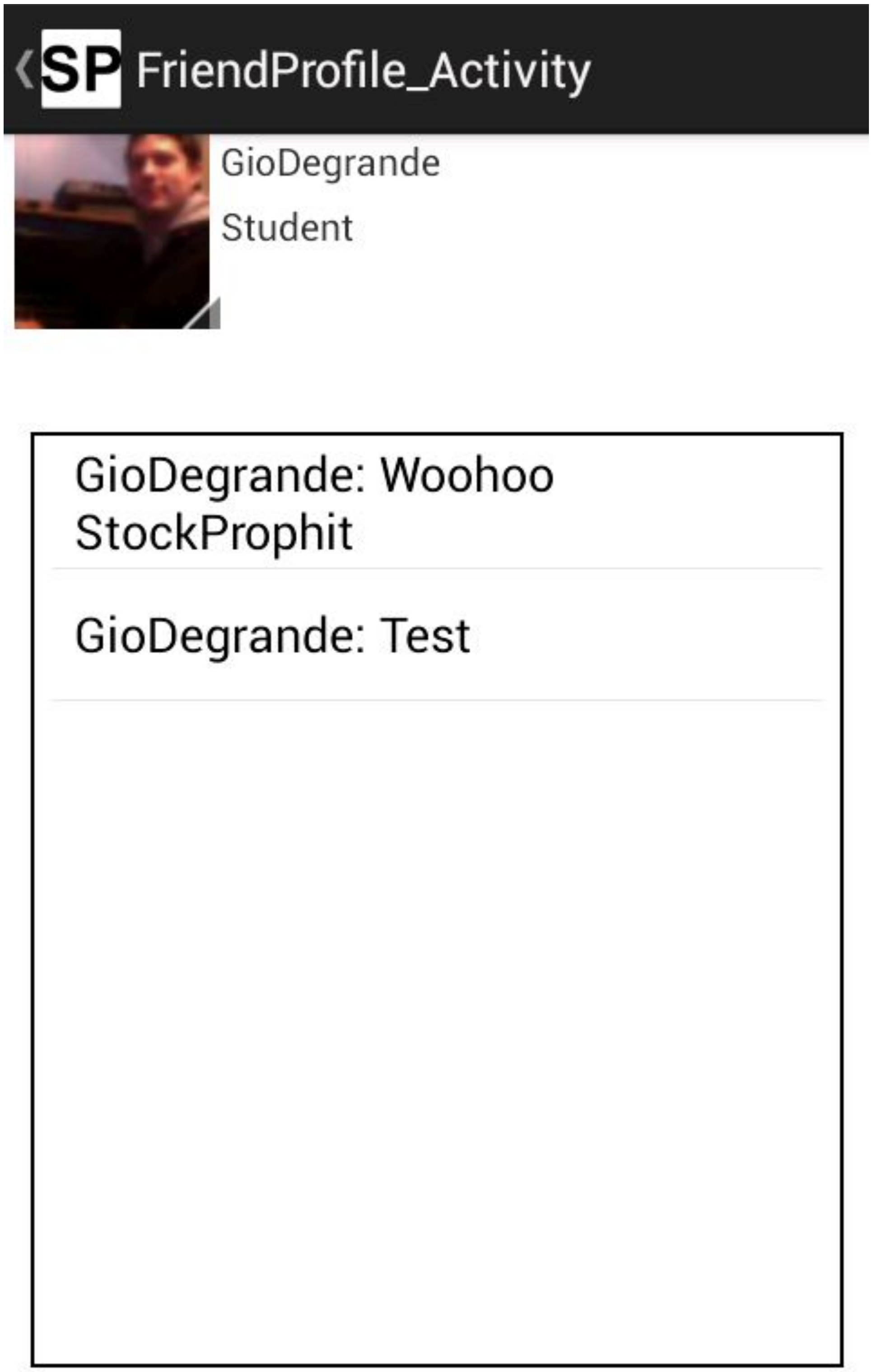
The home screen gives a display of the user's news feed (if logged in), options to navigate to the top/bottom 5, favorites, and social tab, as well as a search to check stock data for an entered stock.



The home screen gives a display of the user’s news feed (if logged in), options to navigate to the top/bottom 5, favorites, and social tab, as well as a search to check stock data for an entered stock.



The home screen gives a display of the user's news feed (if logged in), options to navigate to the favorites or social tab, as well as a search to check stock data for an entered stock.



GioDegrande

Student

GioDegrande: Woohoo StockProphit

GioDegrande: Test

The profile page allows a user to view his or her own profile or another user's profile that they have searched. From here they can see all of the specified user's posts and they have the option to follow that user, if it is not their own profile.

1. Login

After starting the app you will be brought to the Login Screen. Enter your Username and Password and click “Sign in”.

2. Navigation

The navigation bar consists of 3 buttons, “Home”, “Faves”, and “Social”. Click on any of the buttons and you will be navigated to that page. This bar is available only on these 3 main pages. To go back at any time click the “SP” button in the upper left hand corner.

3. Home

The home page consists of a stock search bar, a send button, and news feed. The search bar allows you to type in any stock you would like to search. Click on the auto complete text once you start typing to have the stock name entered for you. Once the stock is input hit the send button to submit the search, it will bring you to the stock page. The news feed displays the most recent posts by any users.

4. Favorites

The Faves page consist of a list of any stocks you have favorited. You can click on any of the stocks in the list and automatically be brought to the stock page.

5. Social (Post)

The social page consists of a “Enter Post” box, a “Submit” button, a “Search” button, a “View My Profile” button, and a “Following” button. The “Enter Post” box is where you would type in any message you would like to post. Once you have entered you post select “Submit” to submit the post. The “Search” button brings you to a page where you can search for other users. “View My Profile” brings you to your own profile. The “Following” button brings you to a page that displays any users you are following.

6. Stock Page

You can get to the stock page 2 ways, by searching a stock from “Home” or by selecting a stock from “Faves”. The stock page displays the Stock ticker, the most recent closing price, and the most recent predicted price. You can also choose to set the stock as a Favorite by selecting the check mark in the upper right hand corner labels “Favorite”. If the checkbox is checked then the stock will show up in your “Faves”. If it is unchecked then it will not be in your “Faves”.

7. Social Search

In order to reach the “Social Search” page you need to go to the “Social” tab and select “Search”. The “Social Search” page consists of a search bar and a “Search” button. To search a user enter in the Username into the search bar. Click on the auto complete text once you start typing to have the Username entered for you. Once entered click the “Search” button to submit the search and be brought to the profile page.

8. Following

In order to reach the “Following” page you need to go to the “Social” tab and select “Following”. The Following page consist of a list of any users you are following. You can click on any of the users in the list and automatically be brought to their Profile.

9. My Profile Page

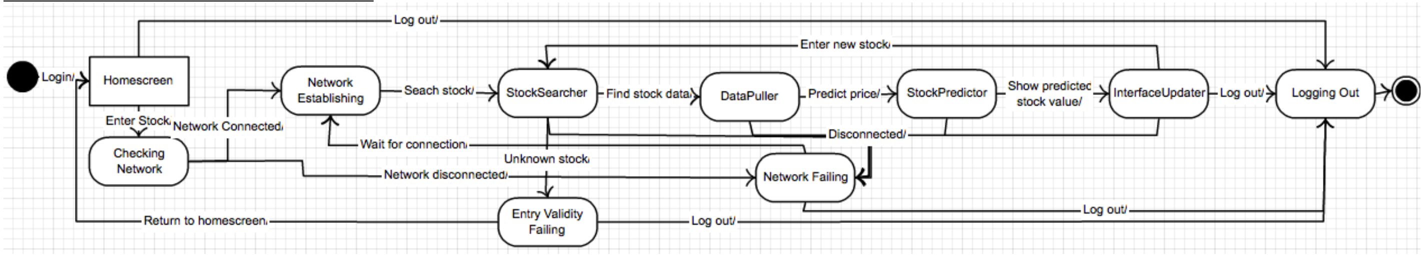
In order to reach the “My Profile” page you need to go to the “Social” tab and select “View My Profile”. You can also perform a search for yourself under the “Search” page. My Profile consist of a Profile Picture, a name, a short bio, and a news feed. The news feed displays all of the recent posts you have made.

10. Profile Page

There are two ways to get to a Users “Profile” page. You can select the user from your following or you can search for the user under the “Search” page. Profile consist of a Profile Picture, a name, a short bio, a news feed, and a following checkbox. The news feed displays all of the recent posts the user has made. You can also choose to follow or unfollow the uses by selecting the check mark in the upper right hand corner labels “Follow”. If the checkbox is checked then you are following the user and they will be displayed in your following tab. If it is unchecked you are not following the user and they will not show up in your favorites.

Design of Tests

Use Case 1 - Search Stock



In this state diagram the user first logs in and is directed to the home screen. After this the user must enter a stock that he or she wishes to search for. Immediately, the system checks the network to make sure there is a stable connection to fulfil the request. If the network is connected then the we move to the Network Establishing state and proceed with our stock search. Note that at every step in this diagram the network may fail and we must wait for a connection to re-establish in order to move forward from the StockSearcher state again. If this is not attainable then the user has the option to log off. After we have good connection we can move towards finding the stock that we requested. When we have found that stock, we pull the data and acquire that. Next we gather prediction from the

StockPredictor. Now the program updates the interface of the application and shows our search results. From here we have the option to search for another stock or decide to log off.

The first test case should describe going from the home screen to searching for a stock. When we enter a stock, after we have tested the connection initially, we must check to make sure the program correctly gathers the right information. We have to first compare the stock ticker symbol typed in to the actual ticker found by our search. We do this by gathering both symbols and comparing the strings to verify that they match. After this we pull the data from the stock and again compare it with the actual data from that specific stock and check to make sure that the information is the same. Next we compare the predictions made by our algorithm to predictions we have calculated out by ourselves to check the accuracy of the algorithm's predictions. Finally we check to make sure all of the predicted values show up on the screen properly by comparing what should appear to what actually does. If any of these processes fail, and error code will appear.

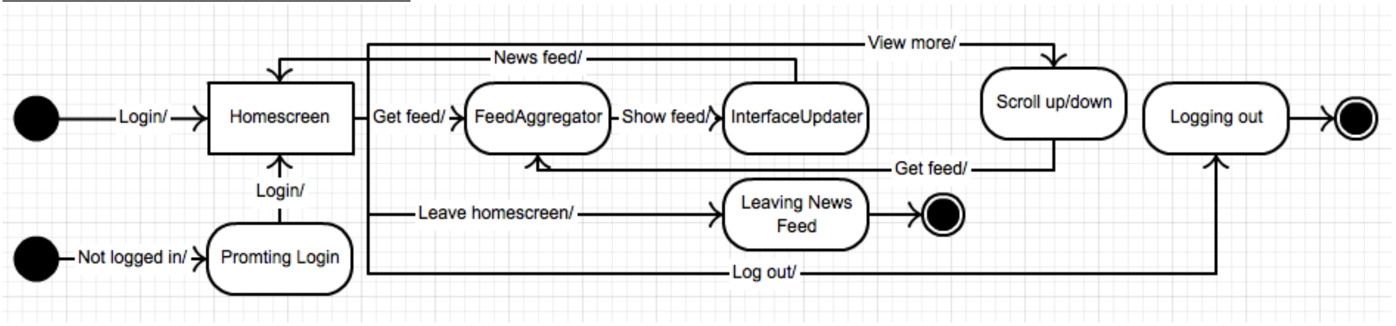
The next test case will be one checking if the stock entered is valid or not. If it is invalid it will be rejected and the user will be sent back to the home screen or have the option to log out. This pseudo code shows how we notice if the stock entered is a valid stock or not. We must compare the stock symbol typed in with a list of valid stock symbols. If the results come up false from our boolean check, then we reject the stock symbol.

```
public class StockSearcherTest {
        //test case to check if invalid stock will be rejected
        @Test public void
                checkStock_Homescreen_invalidStockRejected() {
                //1. set up
                Checker stock = new Checker(/*constructor params*/);
                //2. act
                Key invalidStock = new Key(/*constructor params*/);
                boolean result = stock.checkStock(invalidStock);
                //3. verify
                assertEqual(result,false);
        }
}
```

The next test case will be to check the network connectivity of the application at all states. If the network becomes disconnected then the user can wait until it re-establishes or log out. This pseudo code shows how we check to see if the network is connected properly. To find out if the network is connected or not we must run a loop through all of the states except the logging out state so that we can continuously check for network failure. While this is going on we must compare the network connection to a connected network connection and check to make sure it matches. If it does match then the network is indeed connected.

```
public class NetworkFailureTest{
        //test case to check if the network becomes disconnected
        @Test public void
                checkNetwork_anyState_networkConnected{
                //1. set up
                Network net = new Network(/*constructor params*/);
                //check connection at all states
                while(state != Logging Out){
                //2. act
                Key networkConnection = new Key(/*constructor params*/);
                boolean result = net.checkNetwork(networkConnection);
                //3. verify
                assertEqual(result,true);
                }
        }
}
```

Use Case 2 - Check News Feed



This state diagram shows the progression of checking the news feed in our application. The user starts off either logged in or not logged in. If the user is not logged in he or she will be prompted to log in before they have access to any information. If they are logged in they will open the application to a homescreen where the application will automatically start getting feed from the FeedAggregator. After it has the feed it will then show it in a proper manner on the Homescreen to the user. Once this has happened the user may decide to scroll up or down on the news feed which goes back to the FeedAggregator that loads more posts to be seen by the user. Finally, if the user would like to exit he or she may leave the home page or log out of the application.

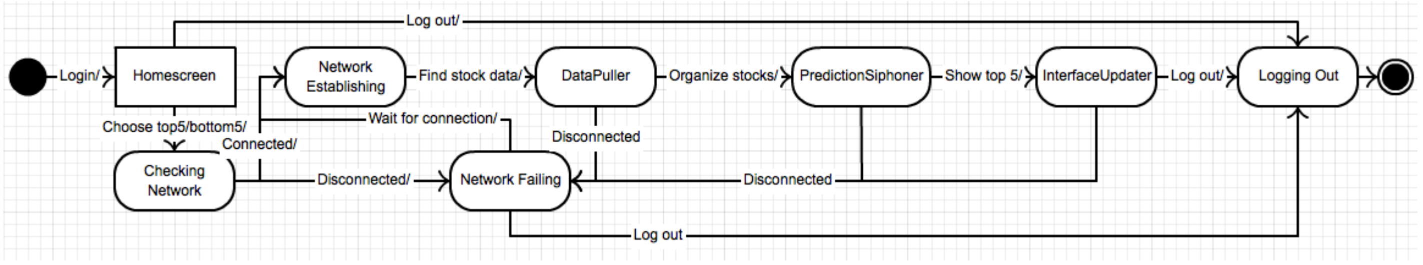
The first test case considers the proper checking of the news feed. This can only be completed after a few small checks are made. First we must make sure that the feed that is gathered corresponds to users' recent posts. To do this properly we would make posts from another account so that we know what would appear in the news feed of a user who is following the test user. We then would compare the gathered data to the data that we posted ourselves to make sure that it is the same. After this we would make sure that the

Homescreen interface is properly updated with the correct news feed as the program gathered. If it accurately compares, then it has succeeded. From here the user can either log off or continue to look at more posts by scrolling up or down. We would test these other posts the same way we would test the original ones by comparison. At any time the user may either log off or travel away from the homepage to end this cycle. If any of these checks fail, the user will receive an error.

The next test case for this use case will check if the user has logged in properly or not. If the user information is invalid it will be rejected and the user will be prompted to login again. This pseudo code shows how we notice if the login entered is a valid login or not. We must compare the login information typed in with a list of valid logins. If the results come up false from our boolean check, then we reject the login.

```
public class UserLoginTest {
        //test case to check if invalid login information will be rejected
        @Test public void
                checkLogin_LoginScreen_invalidLoginRejected() {
                //1. set up
                Checker login = new Checker(/*constructor params*/);
                //2. act
                Key invalidLogin = new Key(/*constructor params*/);
                boolean result = login.checkLogin(invalidLogin);
                //3. verify
                assertEqual(result,false);
        }
}
```

<u>Use Case 4 - Visit Friend's Profile</u>



In this state diagram we see the steps to visit a friend's profile page. The user starts off either logged in or not logged in. If the user is not logged in he or she will be prompted to log in before they have access to any information. If they are logged in they will open the application to a homescreen. From here they should click on the social button to be taken to the social page. From the social page they can search for another user to view their profile. Once they have entered a user to find, the application goes through UserSearcher to find the specific user that was requested. If the user does not exist or they have typed an invalid entry, the system will push them back to the social page where they can restart their search process or log off completely. If they do enter a valid name, however, the system loads the user's info using UserProfile and then shows the user's profile after it updates the interface. From here the user has the option to either return to the social screen again, decide to follow the user, or log off. Once they have followed the user they again can return to the social screen or log off of the program.

This next test case is to check to make sure we visit a friend's profile without problems. First we must load the social page and verify that the options that show up are the same options that we expect to by comparing the strings using boolean variables. After that we look for a user and we must compare the user found to the user searched for, if they match we move forward. Then we look at the user's profile that we found and compare the information with the actual information inputted into the system before hand. If all of the data and strings match then we know it is correct. This is where we must check that the information that we verified shows up on screen properly as it should. We compare the actual results with our expected ones. From here we have the option to follow the user if we are not already or just log out. If we decide to follow the user then we must check our following users afterwards to make sure the user we just followed has been added properly. If the user appears then the attempt was successful. After we follow the user we can either return to the social page or log out. If any of these conditions fail then the user is given an error message.

The test case for visiting a friend's profile will check if the user has logged in properly or not. If the user information is invalid it will be rejected and the user will be prompted to login again or have the option to log out. This pseudo code shows how we notice if the login entered is a valid login or not. We must compare the login information typed in with a list of valid logins. If the results come up false from our boolean check, then we reject the login.

```
public class UserLoginTest {
        //test case to check if invalid login information will be rejected
        @Test public void
                checkLogin_LoginScreen_invalidLoginRejected() {
                //1. set up
                Checker login = new Checker(/*constructor params*/);
                //2. act
                Key invalidLogin = new Key(/*constructor params*/);
                boolean result = login.checkLogin(invalidLogin);
                //3. verify
                assertEqual(result,false);
        }
}
```

This test case will be checking if the user/friend entered is valid or not. If it is invalid it will be rejected and the user will be sent back to the social screen. This pseudo code shows how we notice if the user is a valid user or not. We must compare the user typed in with a list of valid users. If the results come up false from our boolean check, then we reject the user entered.

```
public class UserSearcherTest {
        //test case to check if invalid user will be rejected
        @Test public void
                checkUser_LoadingSocial_invalidUserRejected() {
                //1. set up
                Checker user = new Checker(/*constructor params*/);
                //2. act
                Key invalidUser = new Key(/*constructor params*/);
                boolean result = user.checkUser(invalidUser);
                //3. verify
                assertEqual(result,false);
        }
}
```

After we have established how to test all of these circumstances, we must decide how to put them all together. Because of the complexity of our systems, we think it is best to use a bottom-up integration method within each state diagram and a vertical integration method to combine all of the test cases together to get the most efficient and simplest integration. Many of our tests depend on the test before them. The first tests we will run will be for test cases 1 and 6 we will first test for the network connection, while we are running the tests for logging in for cases 2 and 4 because the network and the login is necessary for all other parts of the state diagram to run. After that has been established then we will run the the stock searcher and user searcher tests from cases 1 and 4 respectively. Once these tests are all complete we can then run all of the first larger tests cases from each state diagram in the order described in each test case.

Test Number	Description	Class/Use Case
1.1	The user searches for a stock	Search Stock

1.1.1	The program checks if the data gathered is correct	DataPuller
1.1.2	The program checks to see if the predicted value is accurate	StockPredictor
1.1.3 2.1.2 4.1.3 6.1.3	The program makes sure the correct information is shown on screen	InterfaceUpdater
1.2	The program compares the entered stock ticker with the actual one	StockSearcher
1.3	The program checks if the network is connected	NetworkChecker
2.1	The program shows the news news feed	Check News Feed
2.1.1 2.1.3	The program compares gathered information to verified information	FeedAggregator Scroll up/down
2.2 4.2	The program check if the login information inputted is valid	LoginChecker
4.1	The user attempts to view another user's profile page	Visit Friend's Profile
4.1.1	The program verifies that the social page contains the same information that we would expect	Loading Social
4.1.2	The Program verifies the information about the user in question has placed on their profile	UserProfile
4.1.4	The program checks to verify that the user requested is indeed being followed	FollowUser
4.3	The program compares strings to verify the entered user is valid	UserSearcher

History of Work

Project Coordination and Progress Report:

Progress as of 3/3/2013

We have continued to work on the interface of our application. Most of the pages have been created. We are now beginning to code the application. (Eric and Gio)

We now have a rough model that is able to run on an actual android device. (Eric and Gio)

We have started to create the interface for all of the social aspects of the application and we expect these to be done within the next week or so, so that we can begin the actual coding of it.(Frank)

We are working on figuring out the database aspect of our project. We want to try to use the ECE server rather than the eden server. We are currently speaking with representatives from the ECE department to see if this is possible. (Gio and Asim)

We are currently figuring out how to properly use Neural Networks in order to get our algorithm functions and our social media search functions to work properly. (Asim and Sangit)

Progress as of 3/12/2013

We have continued work on the interface. We are expecting the interface to be finished by the end of the week so we can begin the coding during spring break. (Eric and Gio)

Our running model has had some updates to fix some bug we had during the switching of tabs. (Eric)

We are still waiting for responses about our database. We have found that use of the ECE server is fairly strict and getting access to is for students has been very difficult. We are leaning more towards the EDEN server. (Gio and Asim)

The algorithm for projection is being developed. Once developed it can be easily translated into code.(Asim and Sangit)

The social network interface is almost completed. It will be completed before spring break. Coding will begin over spring break. (Frank)

Progress as of 3/17/2013

The interface is mostly developed. Coding of the interface has begun. (Eric)

Database development has begun. We have decided to use a private server, we have not found any Rutgers servers to be very helpful. (Gio)

Code has been developed to take stock information and create a graphical representation of it. (Gio)

Algorithm coding has begun. We are working on using the KyroNet protocol. (Sangit)

Began working on the backend for prediction algorithm by implementing the neuroph framework. Currently using dummy data to work the prediction. Attempting to train the network with data from a stock's previous performance(Currently dummy data). Currently stock data is hardcoded, upon working the first step is to make it dynamic. (Asim)

Social Interface has been finished. Coding of the Social Network is going to begin in the next couple days. (Frank)

Current Status

Database is in the process of being set up on a private computer in the ece lab. We initially planned to have this part of the project done sooner however we had to change the server we were running on to complete the project. The initial server was insufficient for storing our database. The eden accounts limit what students can do on their servers so we had to obtain access to an ece server in the ee building. Our current database is being run from there.

Future Work

Our future plans for developing the stockprophit app are ambitious and possible. We plan to upgrade to a more powerful server to improve the runtime of the app and increase the amount of possible users running the app at a time. The current stockprophit app gets its stock values from the stocklytics api the eventual goal is to obtain the stock values directly to cut down the delay when trying to recieve up to date results. The stockprophits neural network will be improved further by increasing the number of location where articles are fetched and processed.

Project Coordination and Progress Report:

Progress Reports (All Reports)

Progress as of 3/3/2013

We have continued to work on the interface of our application. Most of the pages have been created. We are now beginning to code the application. (Eric and Gio)

We now have a rough model that is able to run on an actual android device. (Eric and Gio)

We have started to create the interface for all of the social aspects of the application and we expect these to be done within the next week or so, so that we can begin the actual coding of it.(Frank)

We are working on figuring out the database aspect of our project. We want to try to use the ECE server rather than the eden server. We are currently speaking with representatives from the ECE department to see if this is possible. (Gio and Asim)

We are currently figuring out how to properly use Neural Networks in order to get our algorithm functions and our social media search functions to work properly. (Asim and Sangit)

Progress as of 3/12/2013

We have continued work on the interface. We are expecting the interface to be finished by the end of the week so we can begin the coding during spring break. (Eric and Gio)

Our running model has had some updates to fix some bug we had during the switching of tabs. (Eric)

We are still waiting for responses about our database. We have found that use of the ECE server is fairly strict and getting access to is for students has been very difficult. We are leaning more towards the EDEN server. (Gio and Asim)

The algorithm for projection is being developed. Once developed it can be easily translated into code.(Asim and Sangit)

The social network interface is almost completed. It will be completed before spring break. Coding will begin over spring break. (Frank)

Progress as of 3/17/2013

The interface is mostly developed. Coding of the interface has begun. (Eric)

Database development has begun. We have decided to use a private server, we have not found any Rutgers servers to be very helpful. (Gio)

Code has been developed to take stock information and create a graphical representation of it. (Gio)

Algorithm coding has begun. We are working on using the KyroNet protocol. (Sangit)

Began working on the backend for prediction algorithm by implementing the neuroph framework. Currently using dummy data to work the prediction. Attempting to train the network with data from a stock's previous performance(Currently dummy data). Currently stock data is hardcoded, upon working the first step is to make it dynamic. (Asim)

Social Interface has been finished. Coding of the Social Network is going to begin in the next couple days. (Frank)

We have not finished too much of the application. We have most of the interface and have a rough version of our application running on an Android Device. We do not have any of our use cases implemented because we are still working on all backend things before we can worry about getting everything operational. We are planning on putting a good dent in the coding over spring break. We have begun our database programming. We have finished the interface for the social network. Coding for the algorithm has begun. We have figured out how we are going to implement our database and we are going to use a private server to do it.

Progress as of 4/2/2013

Prediction Algorithm - Our prediction algorithm is up and running. We tested it using a few days of data from a specific stock. The prediction was just a few dollars off (a percentage point or 2 because it was Google and they trade around $780).

Database - We have access to our database but we have not yet begun our coding for it.

Stock API - Since we do not have our database running we do not have anywhere to store any information we receive from the API.

Social Network - Most of the social network is up and running. We have the ability to post any comment that you want. The post will show up in the news feed of any user, eventually following and followers will be set up, so that only people who are following a user can see their posts. We also have the ability to view your own profile as well as others. On this profile you can see any posts the user has posted.

Main Interface - We have the ability to search stocks. Once a stock is searched and clicked on, the user is brought to a pay where it displays a price. As of right now that price is hardcoded in. Once we have the stock data in the database we will be able to pull from there in order to get a current price. We will also have the prediction of that stock and a graph of previous stock prices. We also have the ability to select favorites. Once you are on a stock page, you can select a button to add it to your favorites.

Top/Bottom 5 is still in progress, this was hard to do without the data but we were able to use some data hardcoded in to help us begin with that coding.

Progress as of 5/1/2013

Prediction Algorithm - Our prediction algorithm has been up and running since our before our demo. It was only using specific stock data because we did not have the ability to populate and use the database. Coding has begun for the addition of using social posts, news articles, and other social networks to influence the price of the predictions. We are anticipating having this done for Demo 2.

Database - We switched our database to the ECE server. We were having a problem accessing the one on the Engineering Server. Our database should be up and running by Friday. We were having some trouble coordinating all the work that needed to be done and it pushed our deadline for the database back about a week. This is one of the most important things to get done quickly because we need it for a lot of the other work we still have left.

Stock API - This is should be completed Friday along with the work on the Database.

Social Network - The Social Network has not been changed. We are waiting for the database to be completed so that we can put the usernames, profile information, as well as any posts on it. Once everything is on the database we will implement following and followers.

Main Interface - Not many updates have been done to the main interface. Again we are waiting for the database to be running so we can put information on there as well as get stock information from it. We have changed around some of the code to make it easier for the integration with the database.

By the time of our Demo we should have all of our use cases running. As of now we have most running, and a few others that still have some bugs that need to be worked out. However since the deadline is coming up so quickly we may need to cut something, starting with profile pictures and notifications. If everything goes smoothly, we shouldn't have a problem finishing everything, but nothing has gone smoothly for us yet.

Other Project Management Activities:

Coordinating Activities: Frank Bohn

We used a google spreadsheet in order to keep track of everything that needed to be done for the demo. Deadlines were set with respect to when they were due.

Organizing meetings: Frank Bohn

The team met every Wednesday. In our weekly meeting we would:

- Assign work for the week
- Discuss what has been worked on and what has been finished
- Set deadlines for important parts of the code
- Set deadlines for parts of the report
- Work on the Interface and coding
- Work on the Prediction Algorithm
- Assign work for the Demo

Plan of Work and Ghant Charts: Frank Bohn

Progress Reports: Frank Bohn

Combining all of the contributions into the correct files and formats: Gio Degrande

Plan of Work:

<u>Plan for Second Demo</u>

Get database up and running

Get API to populate database with stock history

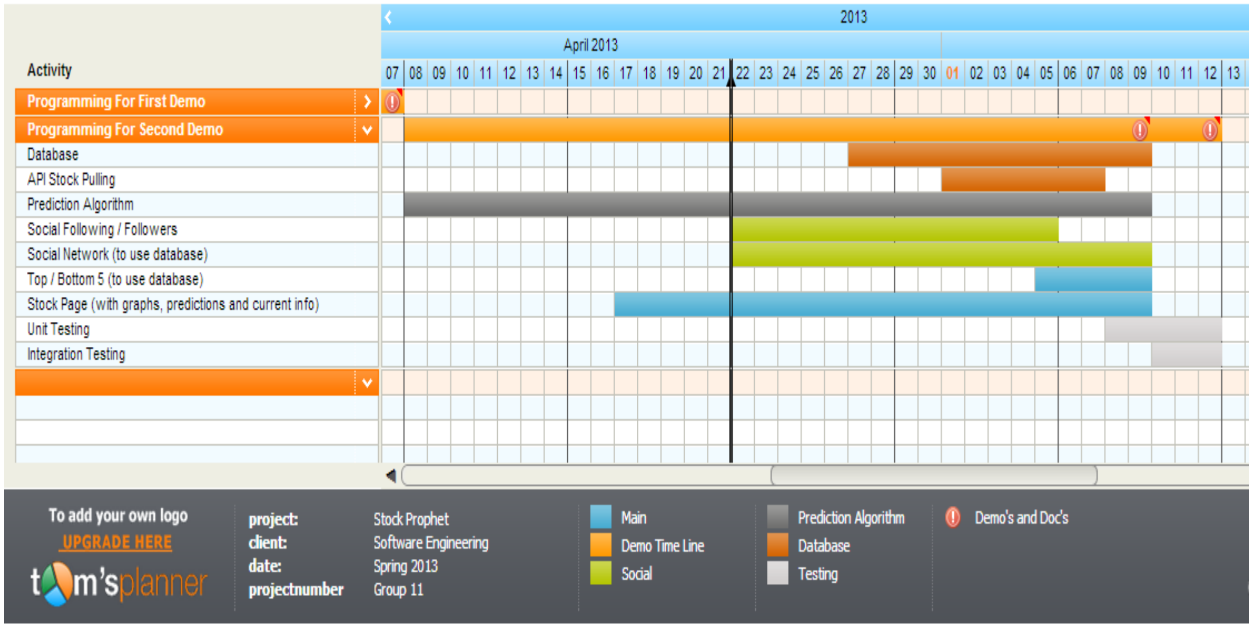
Get predictions to use news articles and posts in order to make the predictions

Move information currently stored on the device onto the database:

- Username and passwords
- All posts
- All User information

- Users Favorites

Get Following/Followers functional
Use stock history from the database to create a graph within the stock search function
Allow users to have profile pictures



project:	Stock Prophet
client:	Software Engineering
date:	Spring 2013
projectnumber	Group 11

Difficulties

Algorithm:

Neural Networks - To implement Neural Networks we found a library called Neuroph. Essentially Neuroph had a built-in API in which multiple functions existed that aided in the creation of a Neural Network. The algorithm for stock prediction was partially created by us, and partially found online from existing code. Online we found a piece of code for predicting the German Stock exchange. Our greatest difficulty was retooling this piece of code for the New York Stock Exchange. It took a couple of days for us to figure out what the code was precisely doing, and from there it took us another couple days to rework the code to suit our needs. We needed to both add and remove various aspects to and from the code so we could get where we wanted.

One of the main things that we had to do was make it so the user would be able to input a stock, and the data for that stock would be used in order to to get the predicted value of the stock. The code we found didn't allow for this, it only had dummy data for one stock. So we wrote a class which in turn did this, and through that we were able to make our algorithm quite dynamic. Furthermore of the challenges that we faced was figuring out how many past closing price data points we should use in our prediction. We realized that if we use too many then the time to compute the prediction would increase. Alternatively we also realized that if we didn't have enough data points then the prediction would not be accurate enough. So through trial and error we reached the conclusion that the ideal amount of data points would be 90 days worth of closing prices.

Neural Networks in it of themselves are a complicated topic, so prior to us even beginning to write code, we had to make sure we had a complete understanding of what neural networks were and how we could successfully implement them to our advantage. So we spent a couple of days researching Neural Networks, to gain a full grasp of their attributes. Only once this was done did we begin any coding. The biggest challenge was taking such a complicated concept and using it in conjunction with our algorithm.

Word Learning - To implement word learning into our algorithm we basically used an API for the New York Times. The way it works is once the user enters the stock name it gets inputted into the algorithm. The code now hits the API searching for any articles with that companies name in it, as well as a few key words that we had inputted that will serve as our way to tell if the companies stock is looking good or not. The API returns snippets of the articles as well as a word count for how many times the key words appeared in the articles. Taking the word count and applying the value these words have to it gives us the amount by which we increase or decrease the predicted price.

This was difficult because first we had to find an API that had the features we needed, the New York Times API was perfect because they have many articles on stocks and companies. Also the APIs had a built in feature that allowed us to search for a topic (stock name) and other words to filter through the article better (key words). Once we had the API, the issue became how to parse the JSON file that is outputted by the API and get the data we needed (word count). This was a bit difficult because we didn't want to use any external libraries to parse the JSON as that would make the file a little bulky. So after working on it for a couple of days, we finally figured out how to do it. We took the JSON file and parsed it into a string and applied a string split, telling it split the string every time it saw a colon. Once this was accomplished we were able to find get the string that contained the word count and now had to parse that into an integer so we could work with it in the algorithm. Now that we had the integer value we could apply its point value. Figuring out what point value to give this count took some time and testing to see what would get us a more accurate prediction. Once we had figured that out we could just add the new point value to the neural networks prediction and get a final more precise value for our prediction.

Stocks:

Favorites - The favorites page was more difficult to develop on an Android platform than it would have been as a simple Windows application. This was due to the fact that I needed to compile the list of favorites from an array of stock names that exists in an xml file. The array is first stored in a string array which is then converted to an arraylist. A plaintext file is read in which contains a string of 0s and 1s. The numbers are delimited by commas, spaces, and surrounded by square brackets which must be replaced before putting each digit into a new integer array. For every 0 in the integer array, the corresponding stock name in the same position in the arraylist is removed. The favorites list is then populated by the arraylist of stocks. One difficulty I faced while designing this method was that I didn't know how or if writing to text files was supported on Android, and the text files could only be viewed if you use the shell utility in the Android Debug Bridge. The largest difficulty was in learning a new platform and designing a user interface as well as doing the programming.

Data Collection:

Part 1 - The first part of data collection involves pulling the stock closing prices from an API called Stocklytics. Essentially how this works is that we created an account with http://stockalytics.com. From here we are given a RESTful URL, the parameters that we pass into the URL are the dates that we want quotes from, as well as the Stock Ticker for which we want the quotes. Once we invoke this URL we receive a JSON file with the Stock information. Once we have this we are now in possession of accurate and up to date stock opening prices, closing prices, and even volume. Even though all we need is the closing price, the other information can be useful in the future if we ever wish to implement any sort of graphs.

Part 2 - The second part of data collection requires us pulling information from news articles from the New York Times website. The New York Times offers a RESTful link with which developers can pass certain parameters and receive certain information from news articles. Collecting this data is key in regards to the Data Mining aspect of our algorithm. By passing the name of a stock ticker you can get articles in which that stock ticker appears. How the data collection algorithm here works is that it pulls at random any 500 articles from the New York Times database with the stock ticker name in order of newest first. It then searches these articles to see how many times the stock ticker name appears. Then once all of the instances of the Stock Ticker name are counted it is divided by 100. And this is the point value that is added onto the the predicted value that is obtained from the neural network.

Search - The stock search uses a hash map to create a relationship between a the passed string and the ticker symbol, the closing price, the position of the stock in the hashmap, and the predicted value. The string is passed from MainActivity.java to the DisplayMessageActivity.java. The hashmap is created and the passed string is used as the key in the map. The returned value, delimited by ':' is put into a string array and the ticker symbol is sent to the server to calculate the predicted value. A text file is read in to determine whether to set the state of the checkbox to checked or unchecked. This uses a similar method to the favorites in which it reads in a string of 0s and 1s. After the checkbox state is set, the page is displayed with all of the stock information. If the checkbox is toggled while on the page, the current state is found, and we write a new string of 0s and 1s to the faves.txt text file. Most of the problems that occurred while writing the code for stock search and favorites were due to learning how to use the Android operating system and develop android applications. These two features took a long time to

Database

Populating - Populating the database from the app was a little tricky but not as nearly as gathering data for the algorithm. We were able to save the data as an array and upload it to the database. Establishing the connection to upload the data was a little bit more difficult to do then creating the arrays. It was doable but soaked up a lot of time because they were a host of errors that we had to work through and fix.

Setting up- Setting Up the database was not too tricky, however it did require some foresight prior to doing any coding. The most difficult parts about setting up the database was trying to figure out, how many tables we would need, what would go in these tables, and what would be the optimal way to arrange the fields. In the end we settled on having separate tables for things such as user login data, social aspects, and small pieces of stock data. Pretty much we set up a fantastic framework for the database that is extremely extensible. Overall the coding here was not intensive at all it was just the pre-planning that took up the bulk of the time.

Client Server - This was one of the hardest things that we had to code in regards to client/server code. We reached the conclusion that storing all of our stock data (i.e. predictions, closing prices, etc.) on the database would be inefficient. However our prediction algorithm was located on the server because otherwise the number crunching would have to be done on the user's phone, which seemed far more inefficient than anything else. So we had to find a way to get the phone to tell the server which stock it wanted information on and then the server had to run the algorithm and return this information. We realized the most efficient way to do this would be to write a Client and Server. What these two programs did was that they would both open up a Socket on port 62000. Once this Socket was opened they would pass each their respective piece of data via a TCP stream. This was a bit tricky to do, mostly because it required knowledge of network protocol. It took a couple of days to finish, but once we got it to work it suited our needs very well.

Social:

Everything in Social was implemented and created by our group.

Posts -Social posts were originally supposed to be placed on they were originally supposed to be place onto the database and then called when a user logs in based on who the user was following. The reason this was not implemented was because we ran into some problems with the database, which did not leave enough time to implement it. Instead posts were implemented by using txt files saved onto the phone. Any post made would be saved to a main txt file which all of the posts by any user were saved to and also to a txt file for that specific user. This would allow any users logging into the same device to see what any other user has posted. When a user profile was visited, the specific users txt file would be displayed. The problems that came up with these txt files is being able to change the name of the txt file for the specific user this took a few hours to figure out.

Profiles - We came across a lot of problems with the profiles. We needed to be able to check to what user was logged in and pass this to the my profile page. We were able to use sharedpreferences to pass the username from login to the profile page. We also used sharedpreferences to pass any searched user to the profile.

The biggest problem we had was being able to follow users. Since each profile page was the same actual page but with different information on it, pressing the following button followed all users, fixing this problem took many hours to figure out. In order to fix this we created a hash map that gave each user an ID. When the checkbox was checked it took the username that was passed from sharedpreferences and checked the hashmap for the username. It then got the users ID. A text file was created with an array of ones and zeros. The ID was used as an index for that array and was set to a 1 for following and 0 for not following. This allowed a user to follow and not follow any and all users. We then ran across another problem where every user that would log into the device would have the same followers. We fixed this by creating a new txt file for every user.

We were not able to implement followers, which would have been a list of users following the logged in user. We tried to figure out how to check every txt file to see if each user was following a specific user. We were unsuccessful and we knew that it would have been easier to implement if it were on the database. Unfortunately we never were able to get it on the database, and for that reason we got rid of it for the second demo.

One of the last problems we had to overcome was to be able to go to the profile page from the list of followers. In order to do that we needed to pass a message from the following list to the profile. The profile was already being passed 2 other usernames, from search and from login. We needed to distinguish between each of these 3 usernames and figure out which one needed to be displayed. In order to do this we set up checks to see if the user was searching himself and that there was a message being sent from following. This took awhile to figure out since we had so many possibilities for the request.
`

Search - We came across a couple problems when we were programming the search function. The search function took whatever the user typed in to the search box and saved it in sharedpreferences. It was later received in the profile. The problem with this is that the user could search any other user even if they did not exist. A profile was still created for that non existant user. In order to fix that we added a check to make sure that the user being searched was in an array of the usernames. If it wasn't then the search was not performed. Fixing this problem did not take too long.

Future

Algorithm:

Word Learning - The word learning we have is already pretty good. If we were to continue to develop this application there are a few additions that could be made to this process to make the prediction more accurate and personal. Possibly allowing the user to input his/her own key words thus making it so that it's almost as if they were reading the New York Times and determining how they think the stock will go based on what they read. Thus giving it more of a real effect. Also allowing the users to share the key words, they feel mean most in predicting how the stock will change in price, with other users and say why they think these are the best words to use.

Stocks:

Favorites - The favorites could be updated to automatically import stocks from your brokerage account so you can easily view your portfolio. Also,

Search -

Database:

Populating - We faced a couple of different problems populating the database. The original API's we planned to use were not all applicable. The news source of CCN gave us trouble retrieving data because they limit the use of what independent programmers can do with their API. In place of CNN we chose Reddit as our news source. Reddit is a very good news source because it gathers articles from all over the internet. It has one of the largest databases of articles and information. We had more control over the API for the Reddit so it was a overall better choice. With the stocklytics API we had no problem getting the data the API was very easy to use and effective, however we had trouble gathering the stock prices automatically from the data because android does not have a built in JSON parser. To counteract that we had to install a package with the app that allows us to parse the data and retrieve the information of the stocks.

Setting up- One of the things that we could implement for the database in the future is using ORM (Object Relational Mapping). Basically what this means is that we would have objects in our database that each contain their own fields of data. An example would be that we could have a user object, the user object would have fields such as name, password, email, and favorites. Then favorites would also be an object, which contains a list of the user's favorite stocks. ORM is a very streamlined way of setting up the database.

Client Server - One of the things that we would like to do in the future is always have the Server.java file listening on the port in a continuous loop. Right now the way the server is set up is that it runs through once. Part of the reason we did this is because we don't waste the server resources by always running. However in a real world application it would be necessary for the server to be running at all times so that a user could get a prediction at any given moment. For that reason one of the goals would be to make it so that Server.java creates a new thread or instance of itself every time the user requests a prediction.

Social:

There is a lot of room for the expansion of the social aspect in the future. We had a lot of ideas to implement but not enough time. Below are some expansions we had in mind to make the social aspect of the app more integrated with the stock prediction.

Posts - One thing we could implement in the future is the ability to post predicted prices and links to specific stocks. This would allow users viewing the post to be able to quickly go to the stock page and see the information themselves. This would give the posts more meaning and give them more of a reason to be checked. In order to implement this we would just have to be able to link the stock symbol in the posts to the stock page. This would require a little bit of work but it is very possible to do using the framework that we have.

Another thing we could implement is that ability to post links to news articles about stocks and companies. This would be helpful because users will be exposed to a wider variety of stocks. This would easily implemented by allowing links in post that exit the app and bring you to the page on the internet with the article. We could also implement it by having the link send us to a new page in the application where the article would be loaded. With the framework we have already integrated this could easily be implemented.

Another thing that we would add in the future is the ability to comment on posts. This would allow for conversations to go on rather than back and forth posts. This would increase the amount of information sharing and would help keep all information accurate. This would be difficult to implement because we would have to figure out a way to save comment with the posts.

A last thing that we would implement would be the user of all the posts for further advance our prediction algorithm. Our algorithm already goes through and archive of articles to look for key words. It could also go through all of the posts looking for the key words. Using this social interface would be very helpful for this, unlike facebook or twitter. Facebook and twitter are not specific to stocks, so they may use the keywords in posts that are completely irrelevant. Our social aspect only deals with stocks. This would be easy to implement with what we already have in place. All we would need to do is put the posts online and have the algorithm look through them. This would help link the social aspect to the stock aspect of our program.

Profiles - One thing that could be implemented in the profiles would be displaying that users favorite stocks. This would allow other users to see what common stock interest they have and could help them decide whether they want to follow them or not. This could easily be implemented. All we would need to do is send the array of stock faves to the profile, which is easy with the way we have everything set up. We would need to do some rearranging of the profile page itself but other then that it should not present a problem.

On that note there is also the possibility of adding stock portfolios to the users profiles so that other users can see what stocks one person is investing in to decide if that user is one that you would have stocks in common with so that you can follow him or her if you so choose.

Another thing that we could implement in the profile would be the ability to change user information and bio. This would allow them to display what their interest in stocks are, what company they work for, and other other relevant information. This would also help for users decide whether they want to follow a user. This is easily implemented by making a edit profile page. The information could then be edited there and saved on the database.

Followers - One thing that we can add in the future is the ability to view who is following you. This could possibly allow users to find other users with similar interest. It would also allow users to see who their posts are going out to and could allow them to post more relevant information based on who is following them. This would also be easy to implement. Once all the following lists are on the database we could write a function to search for an index in each following list. Another way to implement this would be that every time a follow occurs and a following index is changed, it would also change a value in a followers array of the person being followed that would signify that this user is following them.

Database(Social) - The main thing that would need to be implemented in the future is the placement of all social information on the database. This would make the information constant from device to device. This is one of the main reasons that a lot of things were not implemented. Using the framework we now have in place, moving this information to the database would be very easy. Once on the database we would just need to change some of the code around to stop calling the text files and instead call the database.

References

Anchor:
http://www.anchor.com.au/hosting/support/CreatingAQuickMySQLRelationalDatabase

Architectural Styles:
http://en.wikipedia.org/wiki/Software_architecture#Examples_of_Architectural_Styles_.2F_Patterns

Arch Survey:
http://www.ics.uci.edu/~fielding/pubs/arch_survey_20.pdf
Designing Network:
http://stackoverflow.com/questions/2080421/designing-a-network-protocol-for-realtime-data-mobile-devices

Kryonet:
https://code.google.com/p/kryonet/

http://en.wikipedia.org/wiki/Internet_access#Technologies