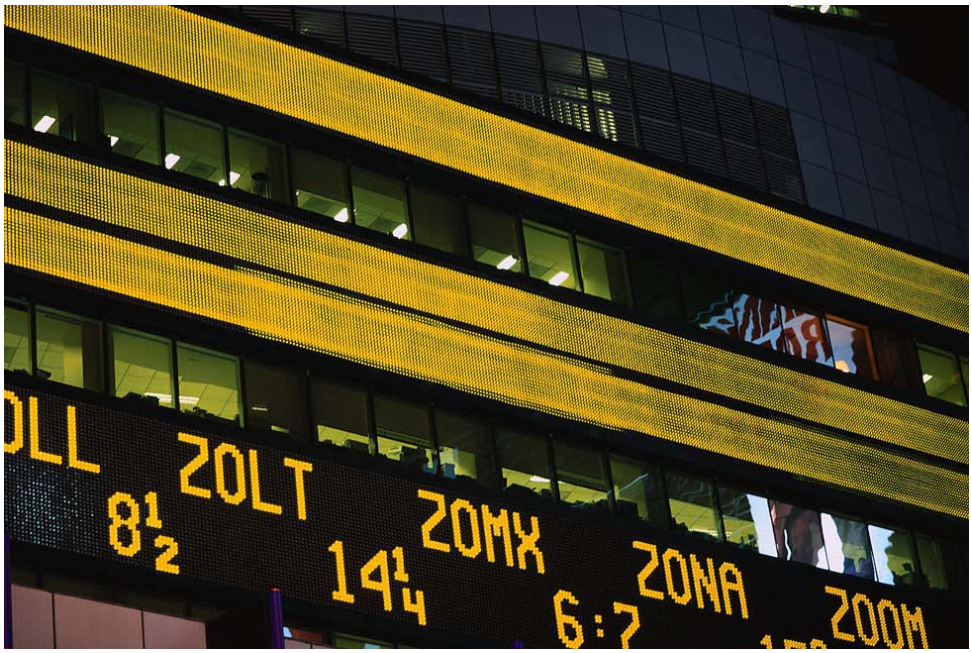
# STOCK MARKET INVESTMENT FANTASY LEAGUE - REPORT 3




5/2/2009

URL of the Project Website: http://rahul.rutgers.edu

Rutgers University - 332:452 - Software Engineering

Group #1

Anirban Ghosh, Nikhil Kasthurirangan, Vamsi Kodamasimham, Pramod Kulkarni, and Rahul Sheth

# 1) CONTRIBUTIONS BREAKDOWN FOR REPORT 3

All team members completed equally to this report, as is evident from the responsibility matrix and allocation chart below.

| Name | Anirban Ghosh | Nikhil Kasthurirangan | Vamsi Kodamasimham | Pramod Kulkarni | Rahul Sheth | - |
|---|---|---|---|---|---|---|
| Report 3 | | | | | | |
| Summary of Changes | 20% | 20% | 20% | 20% | 20% | |
| Customer State of Requirements | 50% | 50% | | | | |
| Glossary of Terms | 20% | 20% | 20% | 20% | 20% | |
| Functional Requirements | 50% | 50% | | | | |
| Non Functional Requirements | 50% | 50% | | | | |
| Domain Analysis | | | 50% | 50% | | |
| Interactions Diagrams | | | 50% | 50% | | |
| Class Diagram & Interface Specification | | | 50% | 50% | | |
| Data Types and Operation Signatures | | | 50% | 50% | | |
| Design Patterns | | | 50% | 50% | | |
| OCL Contracts | | | 100% | | | |
| Class Diagrams | | | | 100% | | |
| Architectural Styles | | 50% | | | 50% | |
| UML Package Diagram | | 100% | | | | |
| Mapping Hardware | | 50% | | | 50% | |
| Database Schema/ Persistent Storage | | 50% | | | 50% | |
| Other | | | | | 100% | |
| Algorithms and Data Structures | | | | 100% | | |
| User Interface design and Implementation | | | | | 100% | |
| History of Work | 100% | | | | | |
| Conclusion & Future Work | 70% | | | | 30% | |
| References | | 100% | | | | |
| Project Management | 20% | 10% | 20% | | 50% | |

FIGURE 1-1 - RESPONSIBILITY MATRIX

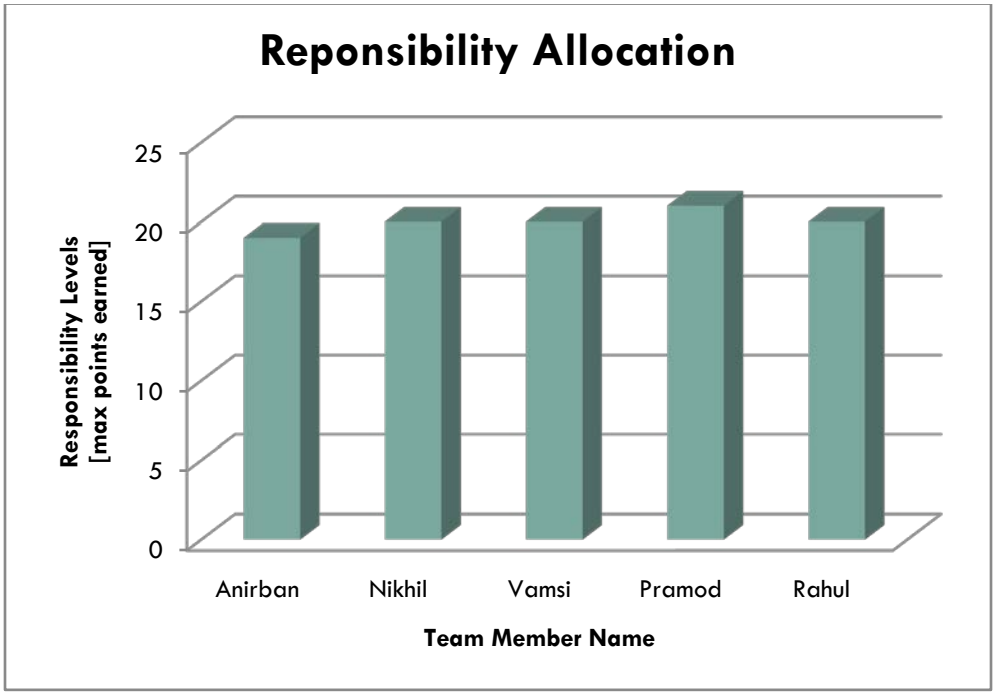



FIGURE 1-2 - RESPONSIBILITY ALLOCATION CHART

## Table of Contents

## 2) FOREWORD

This project will be implemented with the aid of Drupal (http://www.drupal.org), a content management system (CMS) designed for quick, standardized development of new websites. Drupal provides a framework for creation of websites through several modules, some of which are the "core" modules, and all Drupal code is written in PHP. Use cases regarding authentication and user interface design are finished in a correct fashion by the Drupal team, but the use of Drupal does not mean we will not be programming at all. We will be programming Drupal modules and making sure they work with existing modules properly to achieve our goals. Our project will be slightly different than most in terms of implementation techniques and responsibility assignment because each function will be assigned to a custom developed module. We chose to use Drupal instead of starting from scratch due to the fact that user interfaces and authentication, while important, are not as interesting or as challenging as programming the actual stock game itself.

## 3) SUMMARY OF CHANGES

- Revised Customer Statement of Requirements
- Added a another requirement [REQ 3]
- Edited Glossary
- Edited Actors and Goals
    1. Deleted Accessor and generalized User (now consistent with glossary)
    2. Combined Stock and Investor Database into Database
    3. Changed the name of External Database to Yahoo Finance
    4. Added an Advertiser Actor and described it.
- Edited Use-Cases (Casual Descriptions)
    1. Combined Stock and Investor Database Updates UCs into a Database UC. (now, UC -1)
    2. Combined Setting Pref. and Disabling Account UCs into Manage Account UC. (now, UC -3)
    3. Combined Buy Stocks (from market) and Buy Stocks (from Investors) UCs into Buy Stocks UC. (now, UC -4).
    4. Combined Sell Stocks (to market) and Sell Stocks (to Investors) UCs into Sell Stocks UC. (now, UC -5).
- Revised Use Case Diagram to reflect changes in Use Cases
- Completely redid Sequence Side Diagrams with new descriptions
- Revised Interaction Diagrams
    - Diagrams now reflect how Drupal works with the code
    - Added foreword to section
- Class Diagram was completely redone
- Added Use Case Points Section
- Added Traceability Matrix
- Added new breakdown of responsibilities, future work, history of work
- Revised Domain Analysis to reflect changes made in the implementation
- Persistent Data Storage section updated
- Updated User Interface Design section to describe new features and pages

# 4) CUSTOMER STATEMENT OF REQUIREMENTS

To Whom It May Concern,

Today, I am writing on behalf of demanding investors who are vigorously searching for ways to wisely endow in the stock market while working within thrifty portfolios. To meet this challenge, investors are seeking more effective financial tools that not only motivate them to invest in the stock market, but also provide intelligent strategies to overcome economic fiascos. Currently, one tool that meets these investor requirements is the stock market fantasy league game. This web-based game simulates a virtual stock world, creating an interwoven network of keen stock investors, through which they trade real-world stocks without the risk of losing real money.

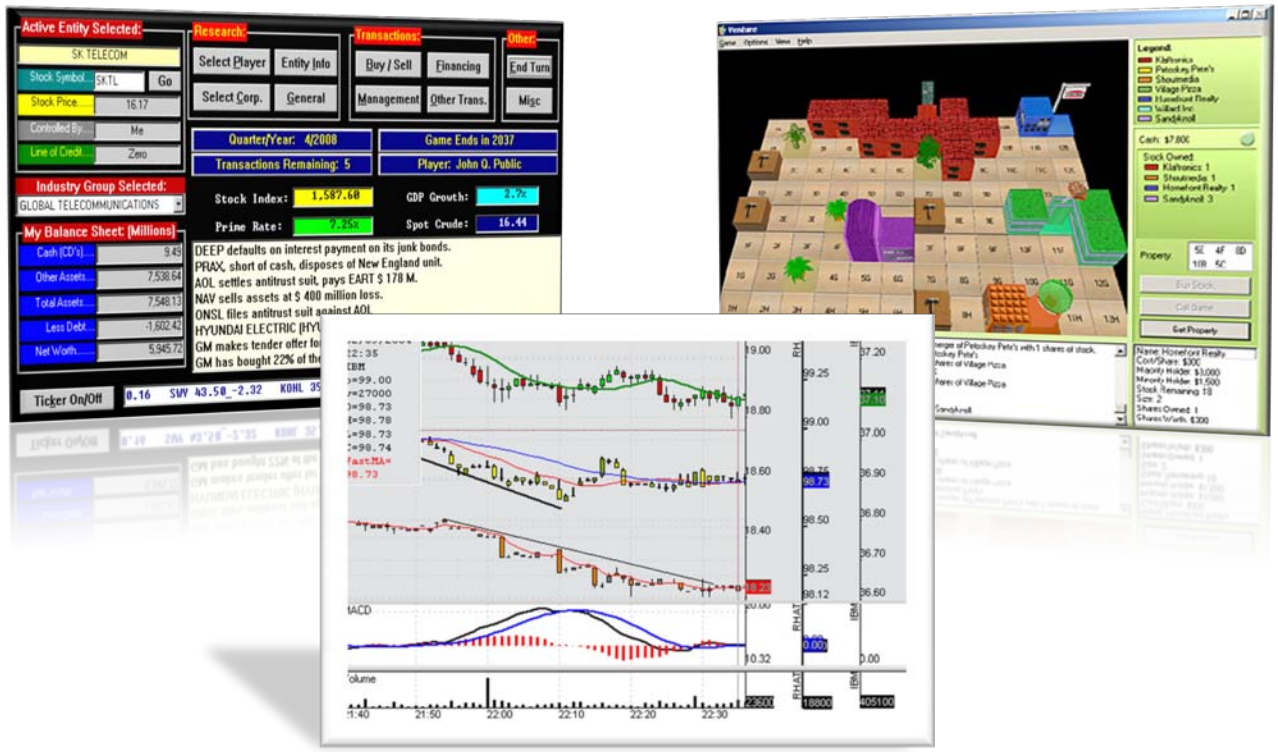



**FIGURE 4-1 - EXAMPLES OF CURRENT VIRTUAL STOCK MARKET GAMES** [1]

This fiscal tool has given stock investors a way to improve their investing strategies as well as their decision-making skills. Whether the investor is new to the financial markets or a stock investing guru, this is a powerful mechanism to build skills, to refine strategies, and to gain important investing experience. For

years, stock investors have successfully utilized this tool to enliven their saving and investing techniques. However, due to a recent surge of financial market debacles, the investors are in search of a more sophisticated and influential application that would keep them several steps ahead of the economic disasters. Our goal, here at Future Markets, Inc., is to develop such an application. I propose a web-based stock market fantasy league, where users and investors can trade fantasy stocks using virtual money in a similar way they would trade stocks in the real stock markets.

Our aim is for prospective investors to make the most out of our web application which mirrors the real stock marketplace. To utilize our application, a web user must complete a quick and easy registration process to create his or her private stock portfolio. Then, the registered user (referred as investor from here) is bestowed with $100,000 of virtual money. Now, the investor can begin playing by choosing among several options. He or she can just simply join a game that is already in progress (or he or she can create his or her own games or wait to be invited to a specific game)[1]. In any case, the investor is let loose in our virtual stock market; he or she can buy, sell, and modify his or her stock portfolio as he or she wishes. As the investor trades, our web application should keep track, update the portfolio and inform him or her about these updates. Moreover, our web application should provide real-time updates of investor's portfolio and as well as the virtual market through actual stock data from trusted and reliable external vendors. This can be performed by storing data in a database and by updating the database at regular intervals. These updates must be quick and reliable. One can maintain speed and consistency by keeping a simple mechanism for persistent data storage. For example, a single database that holds both the stocks in the market and the investor's portfolios can be updated with ease than updating several databases.[2]

[1] These extra features might be included in future versions. For now, a single game is played.

[2] Note: This is a suggestion not a requirement. Other forms of persistent storage are welcomed as long as they are reliable.

Needless to say, our application must provide every investor options to privately trace his or her stock history. A portfolio interface can be used to achieve this task. This interface must allow an investor to view all the stocks he or she owns, what his or her current portfolio value is, how much gain or loss he or she has made on each stocks and much more. Also, this interface should store past information and the success strategies that investors have employed. This is to keep investors informed of how they are currently performing, and perhaps remind them to make changes to their portfolios, if necessary. Using this historical data, our application should provide smart recommendations on future investment strategies through built-in mathematical models.[3]

Finally, through profits gained from stock advertisements, the application administrators should award the best investors on a regular basis. This provides an additional incentive to play the game and perform well. The intent here is for investors to be able to identify practices of the current tournament winner and possibly attempt to emulate his or her styles. On the other hand, the reward may be in the form of stock recommendations such as "players who bought this stock also bought these five others" or even more complex strategies.

Hopefully, the narrative above has given enough information about the demanding investor requirements. Optimistically, prospective investors who participate in this virtual stock market game will learn more than investing. As they progress, they will learn core concepts and acquire necessary skills relating to decision making, saving and investing that will help them succeed not only in the investment market, but also in life.

After conducting a survey about investor's individual requirements and about what they would like to see in a web-based stock market fantasy league application, a majority of investors advised eleven

[3] Models for Sharpe Ratio Analysis, Value at Risk Analysis must be present. Other models are always encouraged.

important requirements. I have attached these requirements below. I wish your team all the best in developing this web application.

Please keep me informed of your progress. Let me know if you have any questions.

Sincerely,

Benjamin Stock

Future Markets, Inc.

## a) List of Requirements:

**REQ 1: Want a stock market fantasy simulator website**

After using many stock predictors and stock market simulators, investors are looking for a much more effective and sophisticated financial tool that does it all: provide stock information, analyze, predict, recommend and finally reward.

**REQ 2: Ease of use (with tutorials and guides)**

The application must be user-friendly to a point where even a beginner will be able to perform his or her tasks without confusion. If not completely user-friendly, the application must provide a complete user tutorial.

**REQ 3: Stock Info**

The application must provide information on each stock available in the real stock market. It should present stock symbols, company names, stock history, stock values and prices etc…

**REQ 4: Reliability**

The web site must be reliable with frequent maintenance. More importantly, the application running on the website cannot frequently stall or crash.

**REQ 5: Graphical User Interface (GUI)**

A user interface with a picturesque layout must be available to perform the stock transactions, and to view stock options and graphs. Ideally, investors must be able to sell or buy stocks with a few mouse clicks.

**REQ 6: Separate investor portfolios with privacy**

Each investor must be able to register for the application. Once registered, a profile should be created. An investor portfolio must also be maintained, including previous and current stock transactions and amounts of money earned and lost. Also, privacy should be maintained for each investor profile and portfolio.

**REQ 7: Ability to automate the buying the selling through thresholds**

Users should have the option of setting threshold amounts. So, if the amount drops or rises beyond this threshold, then, the application should either buy or sell according to the set options of the user.

**REQ 8: Administrators to monitor activity**

A webmaster or an administrator must be available to answer questions and to supervise the game.

**REQ 9: Update and evaluate stocks**

A continuous update of stocks and portfolios must be provided. These updates must be accurate and reliable to the original stocks present in the stock market.

**REQ 10: Rewards**

The administrator or the web application itself should determine a winner of the month and award him or her through various ways.

**REQ 11: Advertisements**

The website must contain appropriate and interesting advertisements relating to finance and stock markets.

# 5) GLOSSARY OF TERMS

- **Actors** - External entities that the system can interact with.
- **Attributes** - properties of concepts, are usually for storage/accounting purposes or state information.
- **CAPTCHA –** a type of challenge-response test used to ensure that the response is not generated by a computer or robot. Usually, the user is asked to type a series of random characters as presented.
- **Class Diagram** – a type of static diagram, which describes the structure of a system. It illustrates system's classes, their attributes and the relationships among the classes.
- **Client-Server Architecture –** a type of system model that describes the relationship between two computers communicating over a network: two programs in which one program, client program, makes a request to another, the server program.
- **Component Based Software Engineering –** is a branch of the software engineering, dealing with decomposition of the systems into logical components with communication across them.
- **Concepts** - Entities that the system is made up of, for intercommunication and handling of events.
- **Controller** - Critical system component that manages communication of the system.
- **Database** – Stores information about the various stocks such as price quotes, ticker symbols, and market name. Also, stores a list of investors currently part of the system and their settings such as user id, passwords, email address, and other personal details.
- **Database Schema –** the structure of language supported by the database that defines the tables, the fields in each table, and the relationships between fields and tables.
- **DBMS** - Database Management System - name for a database package (like MySQL or Oracle)
- **Domain Model –** a conceptual model of a system, which describes the various entities involved within that system and their relationships.
- **Drupal –** Drupal is an open source content management system and modular application framework. Its modules are used to allow web designers to organize and display content.
- **Event-Driven Architecture –** a software engineering pattern that promotes the production and detection of events.
- **FURPS+** - stands for Functionality, Usability, Reliability, Performance, and Reliability (+ Others) - a standard method used to describe and classify non-functional requirements.
- **GOMS** - stands for Goals, Operators, Methods, and Selection Rules – a standard method used in creating user interfaces that should outline goals in an easy to read format for use in efficient interface design.
- **Google AdSense –** an external actor that enables advertisers to display relevant advertisements.
- **Graphical Analysis** - Using software to display graphs. This could be of a particular portfolio or a price of a stock over time. It is normally used for comparison as well as analyzing trends of a successful investor.
- **HTTP –** (Hypertext Transfer Protocol) an application level protocol for distributed and hypermedia information systems.

- **Interaction Diagram –** an overview diagram that illustrates the low-level sequence of events that occur within the system and its components.
- **Investor** - An individual who is authenticated using the login system and is interacting with the system with a portfolio.
- **MySQL –** an open-source relational database management system, running as a server to provide multiple accesses to numerous databases.
- **Net Worth** - The total value of an investor's portfolio. In this system, it is the sum of current market prices of an investor's stocks.
- **Package Diagram –** a static diagram that depicts the dependencies between packages that make up a model.
- **PHP –** (Hypertext Pre-Processor) a scripting language used to build dynamic web pages.
- **Portfolio** - A grouping of financial assets such as stocks, bonds, and cash equivalents, as well as their mutual, exchange-traded, and closed-fund counterparts.
- **Sequence Side Diagram –** a dynamic diagram that illustrates the sequence of events related to a single use case.
- **Share** - The percentage of an industry or market's total sales that is earned by a particular company over a specified time period.
- **Sharpe Ratio –** a measure of the excess return per unit of risk in an investment asset or a trading strategy. It is named after William Forsyth Sharpe.
- **Stock Market** - The market in which shares are issued and traded either through exchanges or over-the-counter transactions.
- **Stock Ticker –** is a mnemonic used to uniquely identify publicly-traded shares of a corporation.
- **System Administrator** - An individual who interacts with the system through an admin account and is responsible for maintaining and managing the system.
- **Timer** - Complex component that synchronizes activities based on system interval settings.
- **UML –** (Unified Modeling language) a standardized modeling language in Software Engineering.
- **Use Case** – a usage scenario of the system, showing an activity a user can perform and the required responses.
- **Use Case Diagram** – a behavioral diagram that summarizes and depicts the use case analysis of a system.
- **User -** Any individual using the internet for its various purposes or anyone who is not yet authenticated by our system.
- **Value at Risk –** is defined as a threshold value such that the probability that the *mark-to-market* loss on the portfolio over the given time horizon exceeds this value. *Mark-to-market* refers to accounting standards of assigning a value to a position held in a financial instrument based on the current fair market price for the instrument.

# 6) FUNCTIONAL REQUIREMENTS SPECIFICATION

## a) Stakeholders

### i) Internal Stakeholders

- Owners: A person or people who legally have the right to possess the web application. Owners' interests are profit, cost, performance, expandability, competitors and customer satisfaction.
- Managers: A person who is in charge of the affairs, resources and expenditures of the web application. They interests include performance, growth, customer satisfaction, profit, cost, employees, and demand.
- Employees: A person who maintains the web application by performing maintenance and support. Their interests are reliability, working conditions, salary, working hours, job security, and benefits.

### ii) External Stakeholders

- Customers (Users or Investors): A person who registers and creates his private stock portfolio using the web application. The customer's interests include value, quality, reliability and service.
- Advertisers: A company or person who calls the attention of the customer (who uses the web application) to a product or business relating to the stock market or financial industry. Advertiser's interests are number of customers, demand, cost, and competitors.
- Government: The executive policy-making body of the United States. Government's interests are taxation, legislation, unemployment, legality
- Competitors: A company providing the identical products (stock market financial tools) to the general public. Their interests include profit, demand, customers and quality.
- Stock Market: It is a public market where company's shares are traded at an agreed price. Its interests are investors, stock holders, buying and selling techniques, and the economy.
- Stock Researcher: An individual who researches the human behavior in the field of investment in stocks. His or her interests include investors, human behavior, and investing strategies.

## b) Actors and Goals

1. **User:** Any individual using the internet for its various purposes or anyone who is not yet authenticated by our system.

    - **Type:** Initiating
    - **Goal:** Create an account.

2. **Investor:** An individual who is authenticated using the login system and is interacting with the system with a portfolio.

    - **Type:** Initiating

- **Goals:** Login, delete an existing account, monitor stocks and portfolio, buy and sell.

3. **Yahoo Finance:** The external source where real-world stock quotes are obtained at periodic time intervals.

    - **Type:** Participating
    - **Goals:** None.

4. **Database:** A place where information about the various stocks such as price quotes, ticker symbols, and market name, are stored. Also, it stores a list of investors currently part of the system and their settings such as user id, passwords, email address, and other personal details.

    - **Type:** Participating
    - **Goal:** None.

5. **Email Server:** A machine responsible for sending messages to investors via E-mail and SMS.

    - **Type:** Participating
    - **Goal:** None.

6. **System Administrator:** An individual who interacts with the system through an admin account and is responsible for maintaining and managing the system.

    - **Type:** Initiating
    - **Goals:** Create a database. Create new games. Monitor games and usage. Perform regular maintenance.

7. **Advertiser:** An individual who interacts with the system through a user account and posts stock relating advertisements.

    - **Type:** Initiating
    - **Goals:** Login. Post Advertisements. Remove Advertisements.

8. **Google AdSense:** The external source by which advertisers can display pertaining advertisements on the webpage.

    - **Type:** Participating
    - **Goals:** Display advertisements.

## c) Use Cases

### i) Casual Descriptions

- **UC1 – Update Database:** The database will be updated on a timely basis. This process updates the stocks in the market as well as the investors' portfolios. The new data will be obtained from an external source, Yahoo Finance. The update frequency will be determined by the system according to the amount of resources available.

- **UC 2 – Create Account:** A user will go through a registration process to gain access to the system. Here, user refers to advertisers and as well as a regular users. Even though advertisers will be provided with an investor portfolio, it is useless to them as their goal is not playing the game.

- **UC 3 – Manage Portfolio:** Once logged in, the investor will be able to set or edit account preferences and portfolio settings. One of key settings is the option of threshold levels. When set, the system automatically buys or sells stocks when the threshold is reached. This is particularly useful for those investors who are on vacation or who are unable to immediately access the internet. The investor will also have the option of unsubscribing; he or she can delete his or her portfolio and account. If done, the investor's data, including the portfolio will be discarded from the database.

- **UC 4 - Buy Stocks:** Investors will be able to purchase stocks at market value from the market.

- **UC 5 - Sell Stocks:** Investors will be able to sell stocks at market value to the market.

- **UC 6 - View Statistics:** Investors will be able to view history and leader board. A list of previous transactions for a particular investor will be shown (to the particular investor) under his or her portfolio. A sorted list of investors, on a leader board, by his or her net worth for a particular game, among other metrics, will always be displayed on the webpage. In addition, a stock chart lookup with a myriad of options can be accessed at any time.

- **UC 7 – Manage Advertisements:** Advertisers will be able to display or remove advertisements from the webpage using online services such as Google Ad Sense.

- **UC 8 – Maintain webpage:** System Administrator will be able to supervise and edit settings of ongoing games. He or she will determine the suitable frequency of stock updates and manually override the system's default settings. He or she is also responsible for rewarding the best investors. On a daily basis, he or she will update the website.

### ii) Fully Dressed Descriptions

**UC 1 – Update Database:**

**Related Requirements:** REQ 9

**Initiating Actor:** System

**Actor's Goal:** To update stock information in the market and in the investors' portfolios.

**Participating Actors:** Yahoo Finance, Database.

**Pre-conditions:** (1) System timer is active.

(2) Database is non-empty.

**Post-conditions:** (1) Stock information is successfully updated.

(2) System timer is reset.

**Flow of Events for Main Success Scenario:**

1. System timer elapses.
2. ← System requests to connect with Yahoo Finance.
3. System establishes a successful connection with Yahoo Finance.
4. ← System requests stock data from Database.
5. → Database dispatches the requested data.
6. ← System prompts Yahoo Finance for updated stock information
7. →Yahoo Finance returns the updated information.
8. ← System stores the updated information into the Database.
9. ← System disconnects with Yahoo Finance.
10. System timer is reset.

**Flow of Events for Extensions (Alternate Scenario):**

3a. System cannot establish a connection with Yahoo Finance.

1. System cancels the update.
2. System timer is reset.

**UC 2 – Create Account:**

**Related Requirements:** REQ 6

**Initiating Actor:** User

**Actor's Goal:** To create an account.

**Participating Actors:** Database, Email Server.

**Pre-conditions:** (1) Database is non-empty.

(2) Email Server is active.

**Post-conditions:** (1) User has successfully created an account.

(2) He or she is given a portfolio.

**Flow of Events for Main Success Scenario:**

1. → User prompts the system to create an account.
2. ← System displays and initiates the registration process.
3. → User submits information.
4. ← System adds user info to Database.
5. ← System adds a new portfolio into the Database.
6. ← System updates portfolio to default settings.
7. ← System congratulates the user.

**Flow of Events for Extensions (Alternate Scenario 1):**

3a. User already has an account.

1. System cancels the registration process.
2. ← (a) System informs user of existing account. (b) System displays options for password retrieval.
3. → User requests password retrieval.
4. ← System signals Email Server to dispatch a custom e-mail.

**Flow of Events for Extensions (Alternate Scenario 2):**

3a. User provides invalid information (including CAPTCHA).

1. ← System informs user of invalid entries.
2. ← System again prompts user for valid information.

**UC 3 – Manage Portfolio:**

**Related Requirements:** REQ 6 and REQ 7

**Initiating Actor:** Investor

**Actor's Goal:** To edit portfolio settings. To unsubscribe.

**Participating Actors:** Database and Email Server.

**Pre-conditions:** (1) Database is non-empty.

(2) Investor account is active.

**Post-conditions:** (1) User has successfully edited his or her portfolio settings.

**Flow of Events for Main Success Scenario:**

1. → Investor prompts the system to edit portfolio settings.
2. ← System displays Investor's current portfolio settings.
3. → User edits settings.
4. ← System updates Database with new information.
5. ← System displays the updated portfolio.

**Flow of Events for Extensions (Alternate Scenario):**

1a. Investor requests to unsubscribe.

1. ← System requests a confirmation.
2. → Investor confirms.
3. ← (a) System deletes investor from the database. (b) System discards all other data and the portfolio.
4. ← System informs the investor.

**UC 4- Buy Stocks:**

**Related Requirements:** REQ 3 and REQ 5

**Initiating Actor:** Investor.

**Actor's Goal:** To buy shares of a company's stocks from the market.

**Participating Actors:** Database.

**Pre-conditions:** (1) Desired company stock available in the market.

(2) Desired number of shares available in the market.

**Post-conditions:** (1) Transaction reflected in investor's portfolio.

(2) Transaction reflected in the market; number of shares available is deducted.

**Flow of Events for Main Success Scenario:**

1. → Investor decides to purchase some shares of a company's stock.
2. ← System requests investor information from database.
3. → Database retrieves the requested portfolio.
4. ← System requests stock information from database.
5. → Database retrieves number of available shares.
6. System checks investor's portfolio for sufficient funds.
7. System queues the transaction in a transaction cart.
8. System processes the transactions in the cart (at regular intervals).
9. ← System modifies portfolio and updates database.
10. ← System modifies number of available shares and updates database.
11. ← System displays the purchases.

**Flow of Events for Extensions (Alternate Scenario):**

6a. Investor's portfolio has insufficient funds.

1. System cancels the entire transaction.
2. ← System notifies investor of insufficient funds.

**UC 5- Sell stocks:**

**Related Requirements:** REQ 3 and REQ 5

**Initiating Actor:** Investor.

**Actor's Goal:** To sell shares of a company's stocks to the market.

**Participating Actors:** Database.

**Pre-conditions:** (1) Portfolio is non-empty.

**Post-conditions:** (1) Transaction reflected in investor's portfolio.

(2) Transaction reflected in the market; number of shares is incremented.

**Flow of Events for Main Success Scenario:**

1. → Investor decides to sell some number of shares of a company's stock.
2. ← System requests investor information from database.
3. → Database retrieves the requested portfolio.
4. ← System requests stock information from database.
5. → Database retrieves number of available shares.
6. System queues the transaction in a transaction cart.
7. System processes the transactions in the cart (at regular intervals).
8. ← System modifies portfolio and updates database.
9. ← System modifies number of available shares and updates database.
10. ← System displays the sales.

**Flow of Events for Extensions (Alternate Scenario 1):**

3a. Investor's portfolio has insufficient shares.

1. System cancels entire transaction.
2. ← System notifies investor of an invalid sale.

**Flow of Events for Extensions (Alternate Scenario 2):**

3a. Investor's portfolio does not contain the stock to sell.

1. System cancels entire transaction.
2. ← System notifies investor of an invalid sale.

**UC 6- View Statistics:**

**Related Requirements:** REQ 5 and REQ 6

**Initiating Actor:** Investor

**Actor's Goal:** To view miscellaneous information about portfolio.

**Participating Actors:** Yahoo Finance.

**Pre-conditions:** (1) Investor has a non-empty portfolio.

(2) Successful connections to Yahoo Finance.

**Post-conditions:** (1) Desired data will be displayed immediately.

**Flow of Events for Main Success Scenario:**

1. → Investor decides to view trading statistics.
2. Investor maneuvers to portfolio webpage.
3. ← System displays transaction history to the investor.

**Flow of Events for Extensions (Alternate Scenario):**

1a. Investor decides to view stock statistics.

1. ← System requests investor to enter a stock symbol.
2. ← System requests investor to choose options.
3. → Investor chooses the necessary options.
4. System connects to Yahoo Finance.
5. → System requests graphical data.
6. ← Yahoo Finance dispatches charts and statistics.
7. ← System displays stock charts and statistics.

**UC 7- Manage Advertisements:**

**Related Requirements:** REQ 11

**Initiating Actor:** Advertiser

**Actor's Goal:** To post, remove or modify advertisements on the webpage.

**Participating Actors:** Google AdSense.

**Pre-conditions:** (1) Advertisements do not violate any policies.

(2) Posting and removal of advertisements is pre-approved.

**Post-conditions:** (1) Advertisement changes will be immediately displayed.

**Flow of Events for Main Success Scenario:**

1. → Advertiser decides to add or remove advertisements.
2. System waits for responses from Google AdSense.
3. Advertiser makes the necessary changes on Google AdSense.
4. → Google AdSense dispatches the amendments.
5. ← System displays the updated advertisements.

**Flow of Events for Extensions (Alternate Scenario):**

1a. Advertiser decides to modify advertisements on the system.

1. System fails to provide appropriate interface.
2. System cancels task.
3. ← System notifies advertiser of technical error.

**UC 8- Maintain webpage:**

**Related Requirements:** REQ 8 and REQ 10

**Initiating Actor:** System Administrator

**Actor's Goals:** (1) To maintain and supervise the ongoing games.

(2) Edit settings.

(3) Reward investors.

**Participating Actors:** System, Email Server

**Pre-conditions:** (1) Email Server is active

(2) Number of ongoing games is non-empty.

**Post-conditions:** (1) Amendments made will be immediately reflected.

**Flow of Events for Main Success Scenario:**

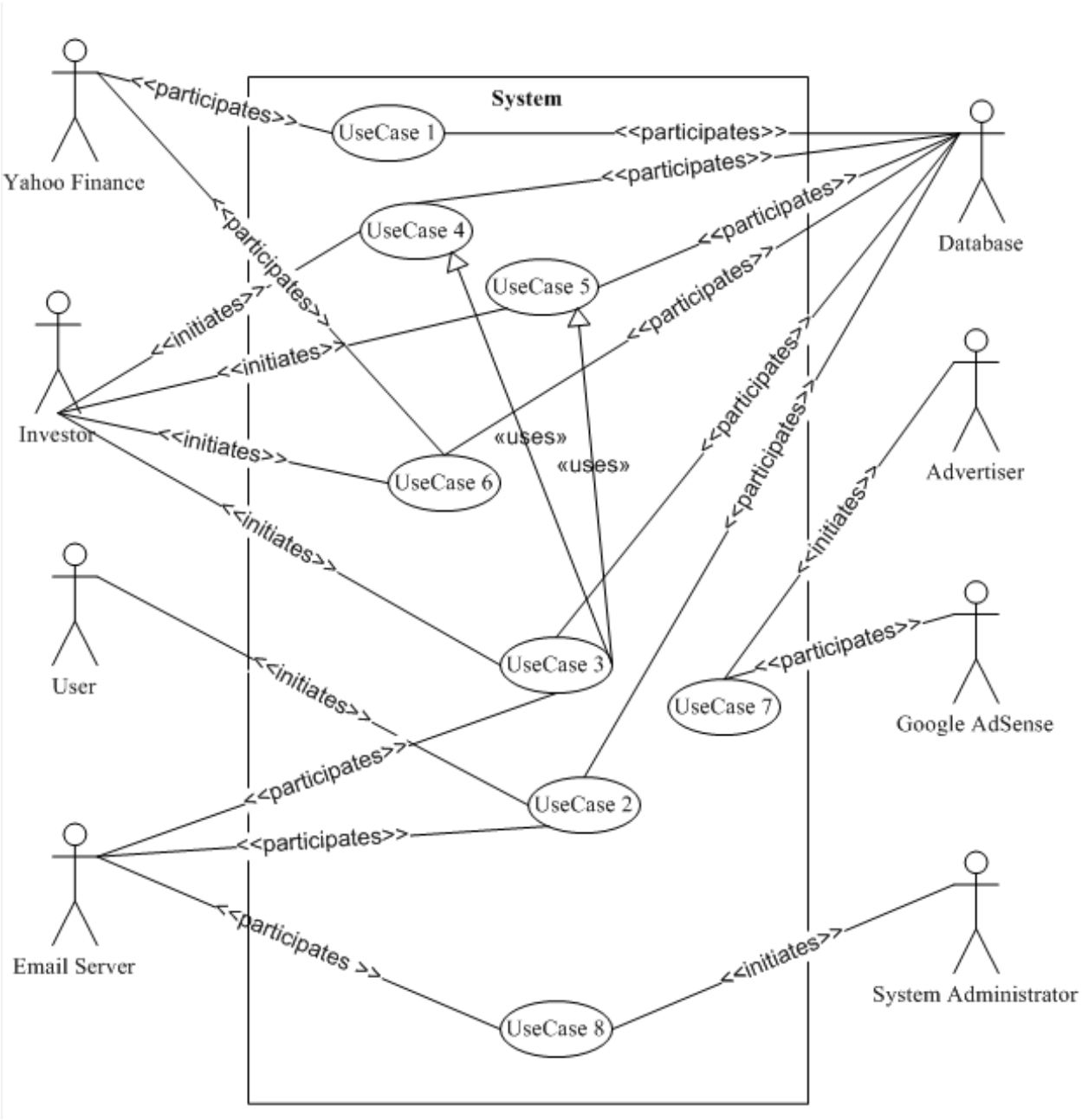
1. → Administrator decides edit system settings.
2. System waits for amendments.
3. → Administrator submits the necessary changes.
4. System executes changes and updates the webpage.
5. System displays the updated webpage.

**Flow of Events for Extensions (Alternate Scenario):**

1a. Administrator decides to reward investors.

1. → Administrator picks investors to reward.
2. ← System prompts for a reward.
3. → Administrator picks and submits a reward.
4. ← System signals the Email Server to notify investors.
5. → Email Server dispatches the congratulatory emails

### iii) Use Case Diagram



### iv) System Requirements - Use Case Traceability Matrix

Below is a traceability matrix for the requirements listed above.

| Req. | Requirement Description | Related Use Cases | Implementation Status |
|---|---|---|---|
| 1 | Want a stock market fantasy simulator website | ALL | Partial |
| 2 | Ease of use (with tutorials and guides) | UC-2, 3, 4, 5, 6, 7 | Partial |
| 3 | Stock Info | UC-1, 3 | Full |
| 4 | Reliability | UC-1, 8 | Full |
| 5 | Graphical User Interface (GUI) | N/A | Full |
| 6 | Separate investor portfolios with privacy | UC-2, 3, 6 | Full |
| 7 | Ability to automate the buying the selling through thresholds | UC-3 | Full |
| 8 | Administrators to monitor activity | UC-8 | Full |
| 9 | Update and evaluate stocks | UC-1, 3, 4, 5, 6 | Full |
| 10 | Rewards | N/A | Partial |
| 11 | Advertisements | UC-7 | Full |

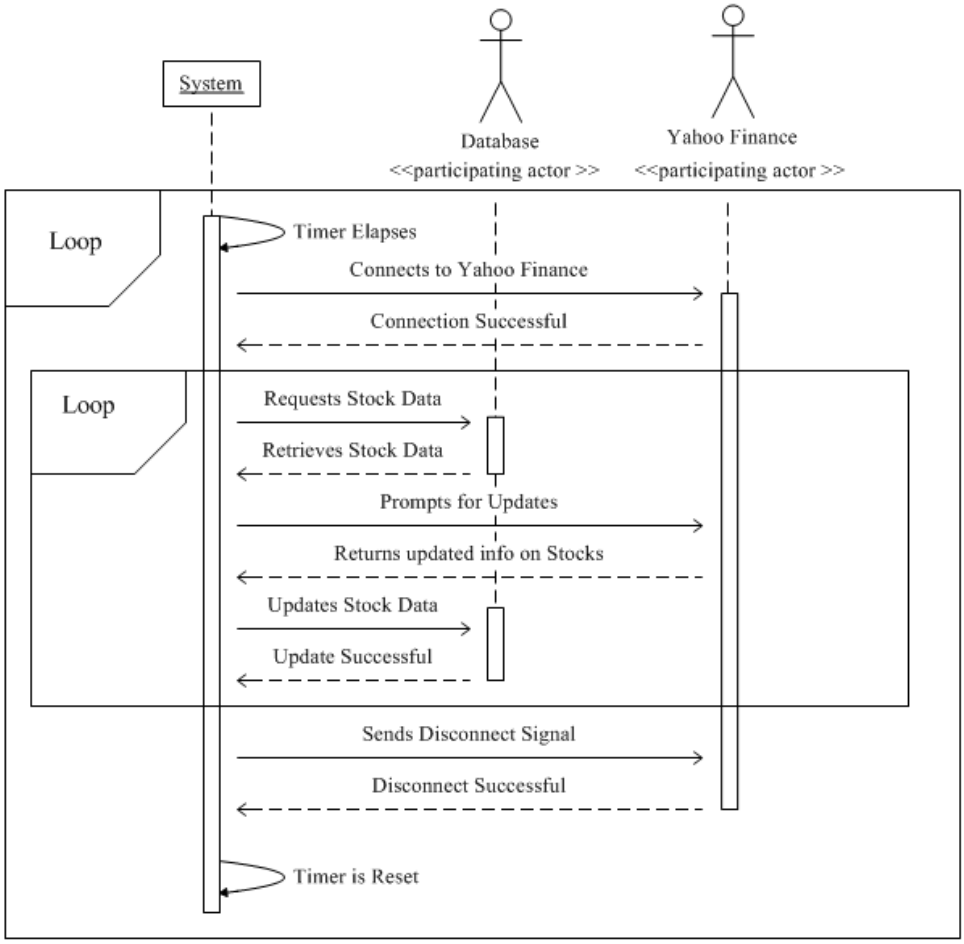
## d) System Sequence Diagrams

**UC – 1: Update Database**

**Main Success Scenario:**

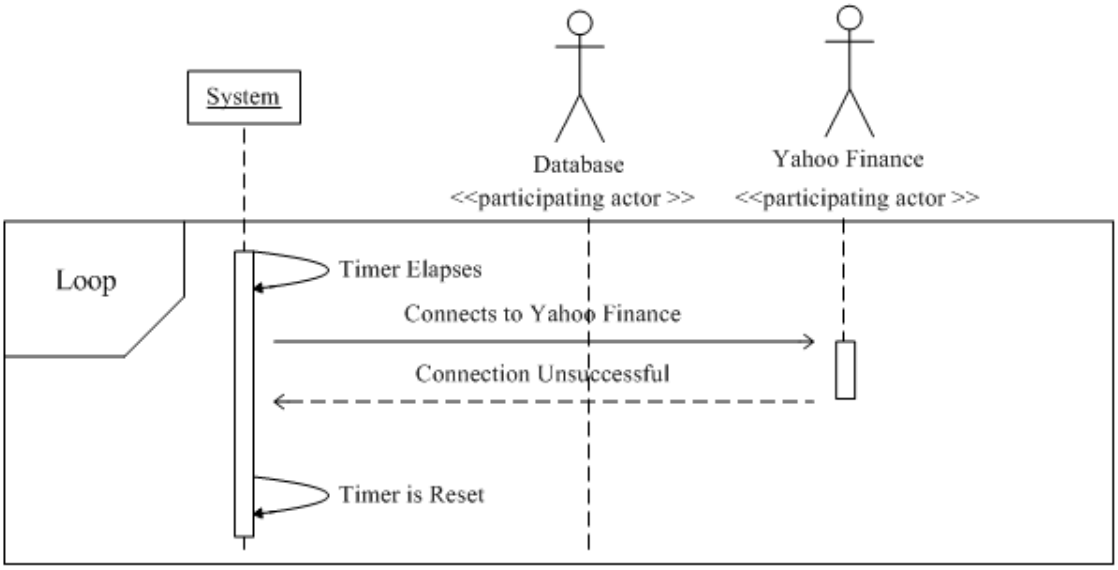
The figure below describes the update of the database at regular time intervals (the default interval settings are 15 minutes). Accurate update of the database is crucial to maintain the integrity of the application. So, a trusted external source, Yahoo Finance, is utilized to obtain the information of the real world stocks. At specific time intervals, the system initiates the update process. The system attempts a connection to Yahoo Finance. Upon success, it retrieves stock information from Yahoo Finance. The information is passed to the database, where it is stored. Once the update process completes, the system resets the update interval.

**Note:** *Only the stocks used in the Fantasy League are updated.*



**Alternate Scenario (Unsuccessful Connection to Yahoo Finance):**

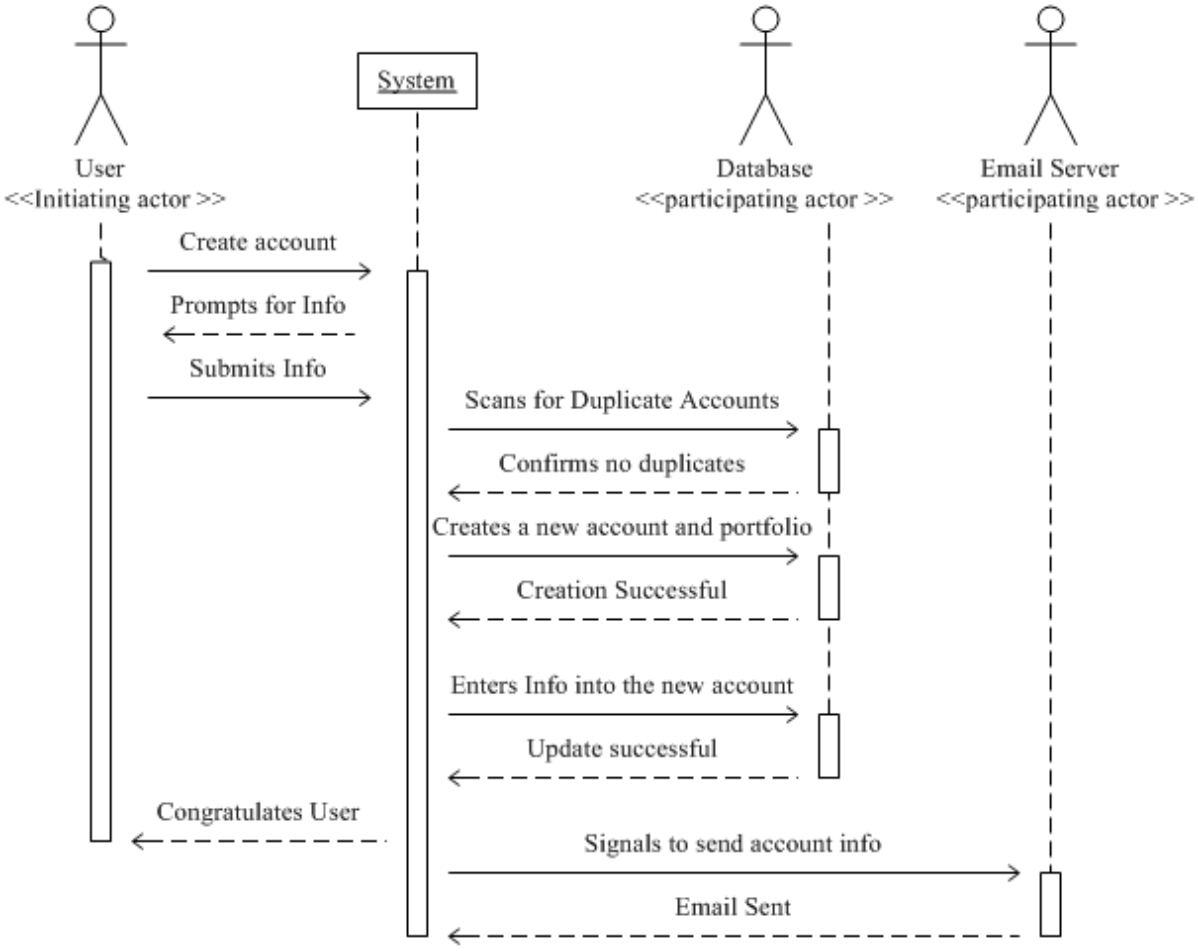
The figure below describes the alternate scenario of the database updates where connection to Yahoo Finance could not be established. In this situation, the system reacts by terminating the entire update process. Then, the system resets the update timer so that the update can be restarted at a later time.



**UC – 2: Create Account**

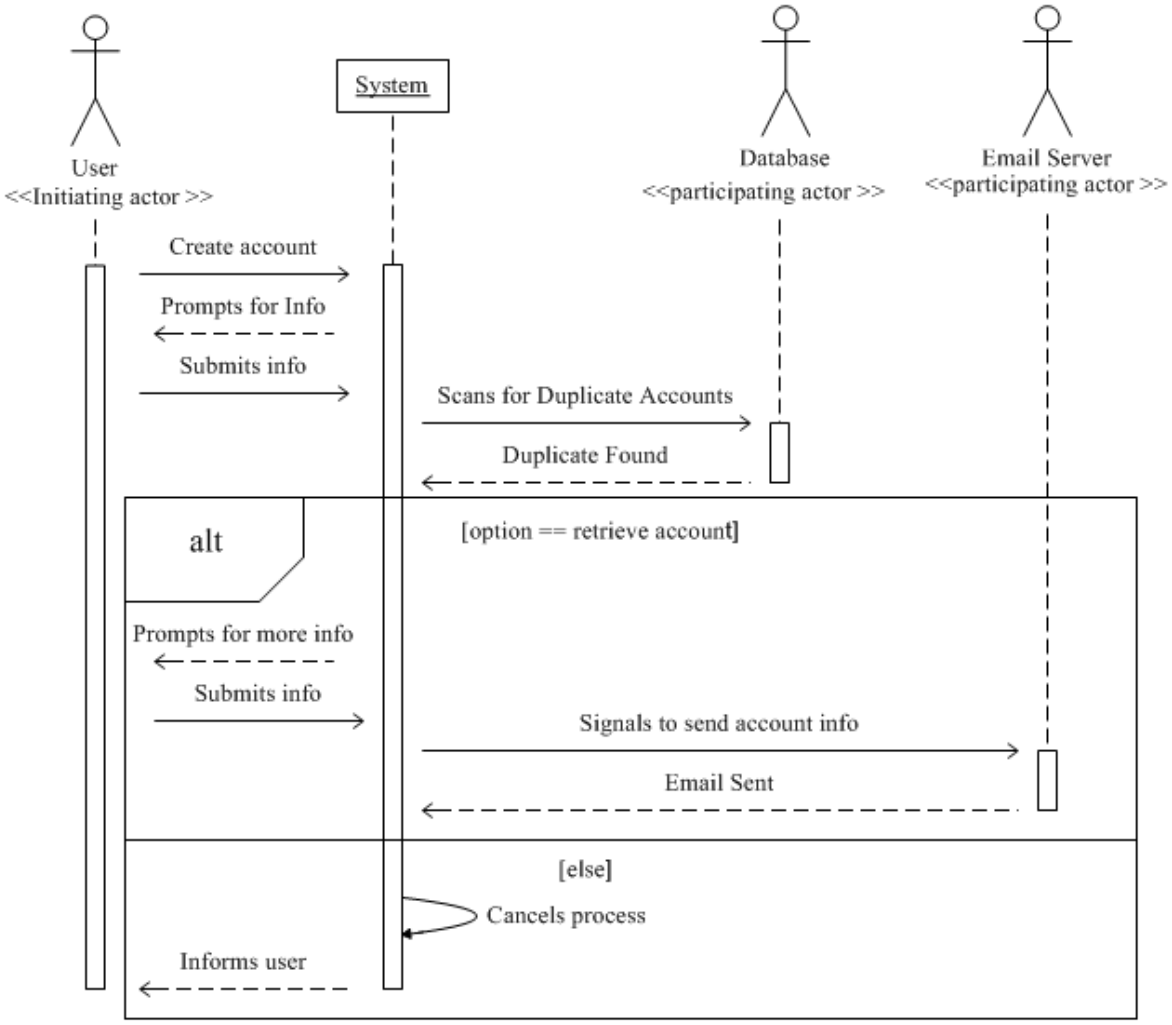
**Main Success Scenario:**

The following diagram illustrates a creation of an investor account and portfolio. A user starts the registration process by clicking on create account button. The system prompts user for personal information. Once the user submits the information (assumed to be valid here), the system creates a new entry in the database and updates it with the provided information, when no duplicates are found. The user is also provided with a portfolio, valuing at $ 100,000. Finally, an email, containing account info and logic instructions, is sent to the user's email address.



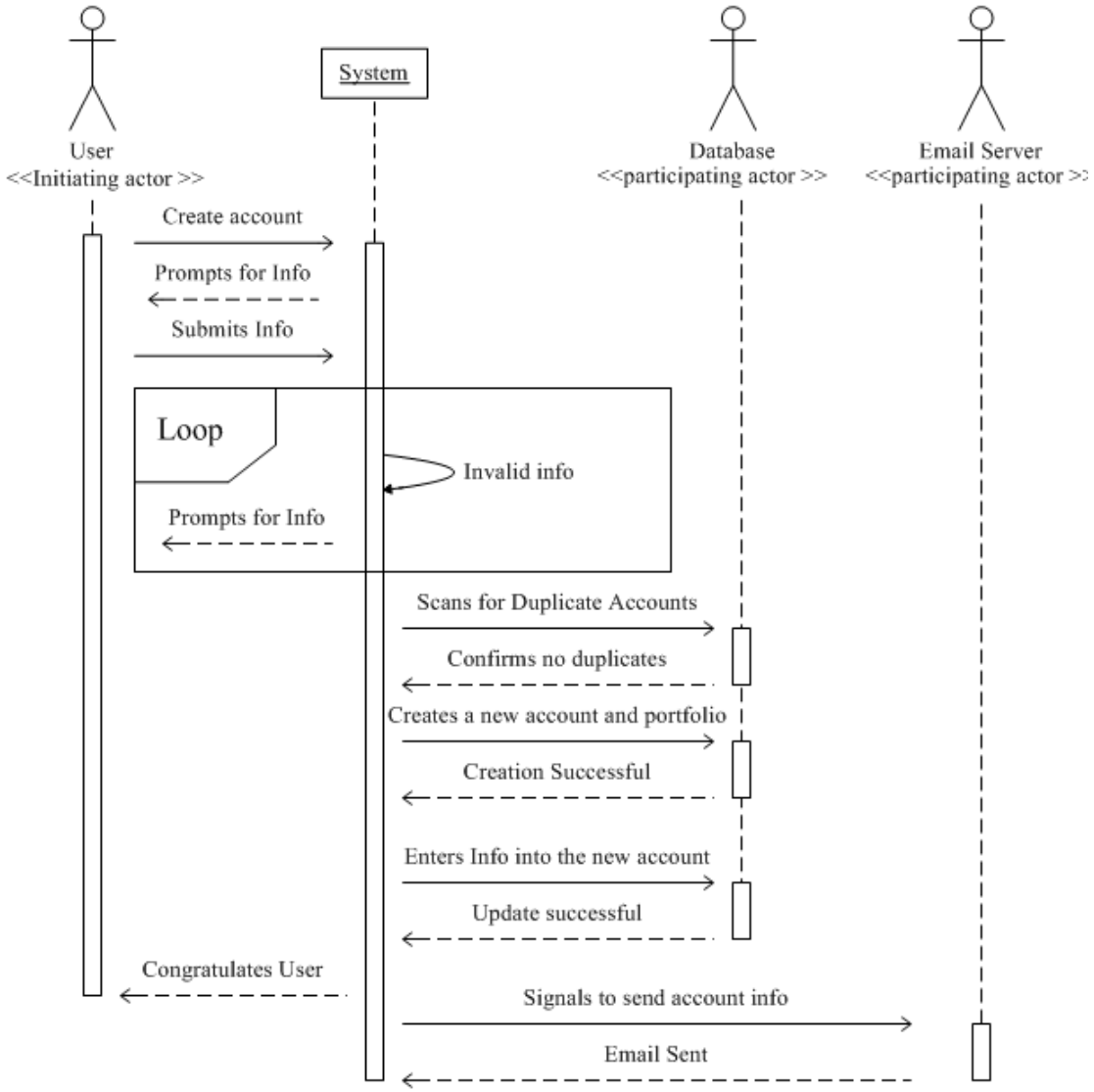
**Alternate Scenario I (User already has an account):**

As the user information is submitted to the system, the system checks for duplicate accounts. In this case, the system finds that the user already holds an account. Then, the system provides the option of retrieving account information. An email, containing the account information, is sent to the user's email address.



**Alternate Scenario II (User provides invalid information):**

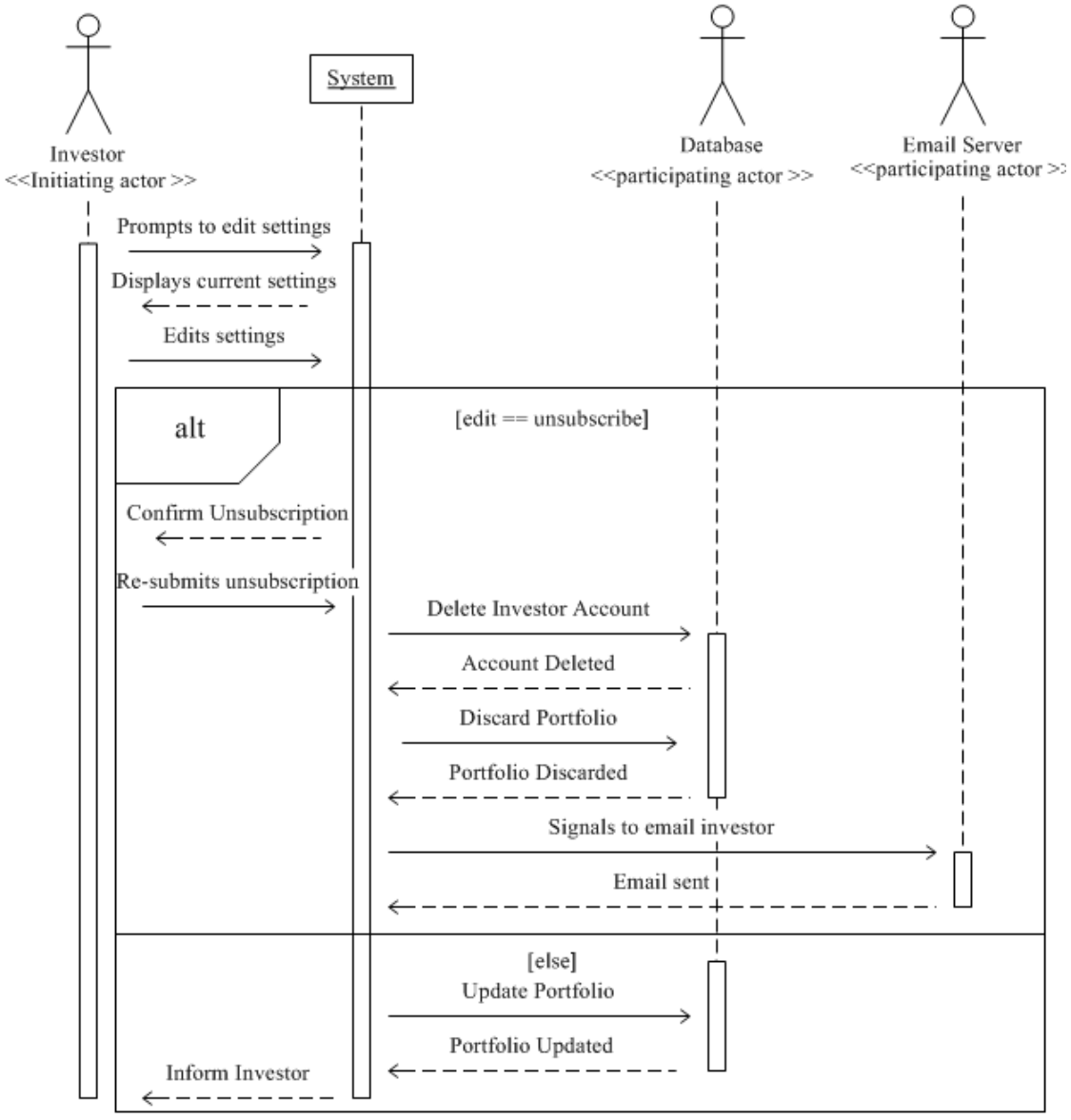
As the user information is submitted to the system, the system checks for validity. In this process, the system finds that the user entered incorrect information. Then, the system provides the option of resubmission after correction.



## UC – 3: Manage Portfolio

### Main Success Scenario and Alternate Scenario (Investor selects to unsubscribe):

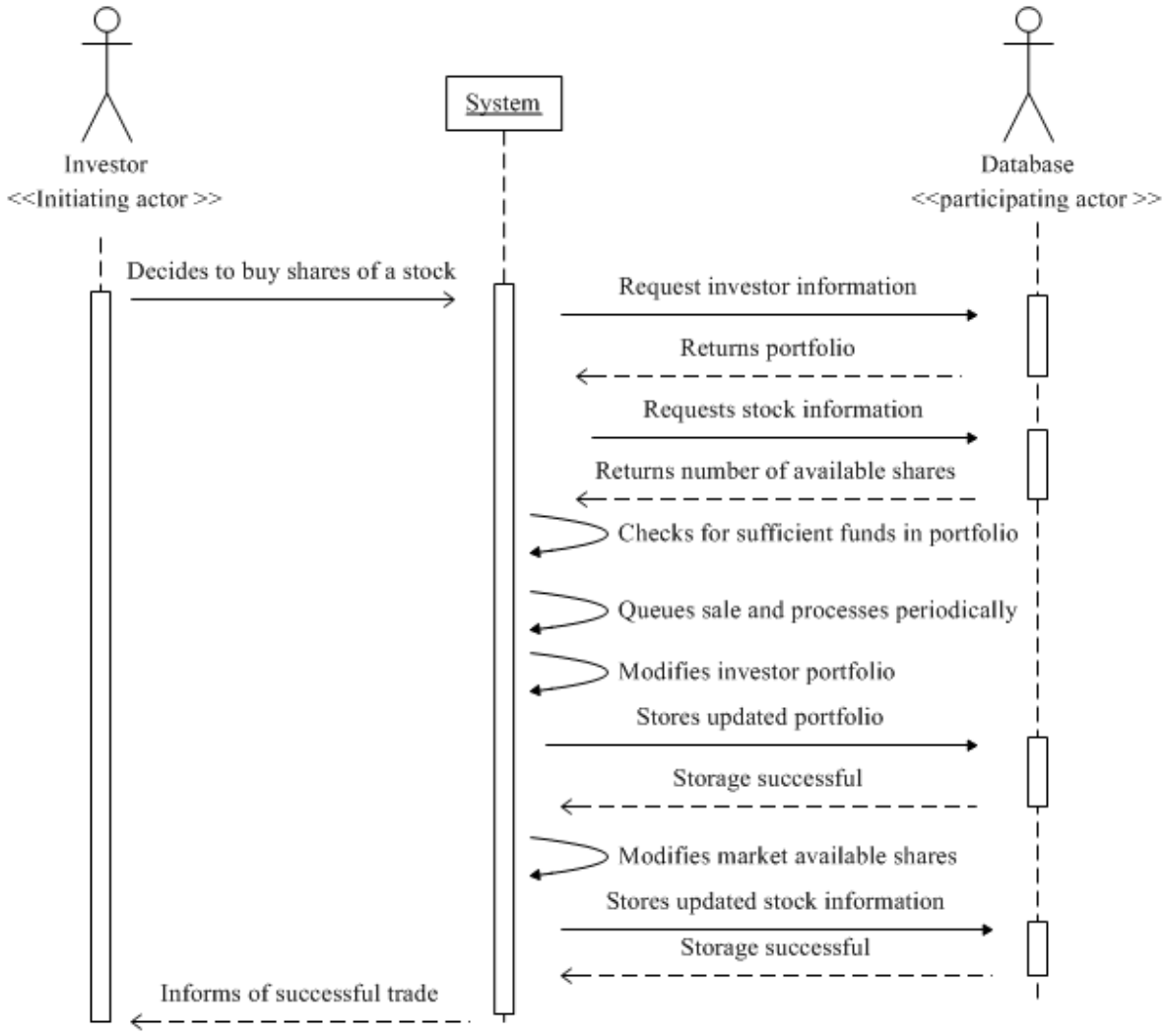
The following diagram illustrates the portfolio management by an investor. He or she edits settings and customizes his or her portfolio. Once the investor submits the changes, it is immediately reflected in the database. One of the key settings in the portfolio is the threshold levels. When set, this automates the buying and selling transactions. If the user selects to unsubscribe, then his or her account is deleted from the database. Moreover, his or her portfolio is discarded.



**UC – 4: Buy Stocks**

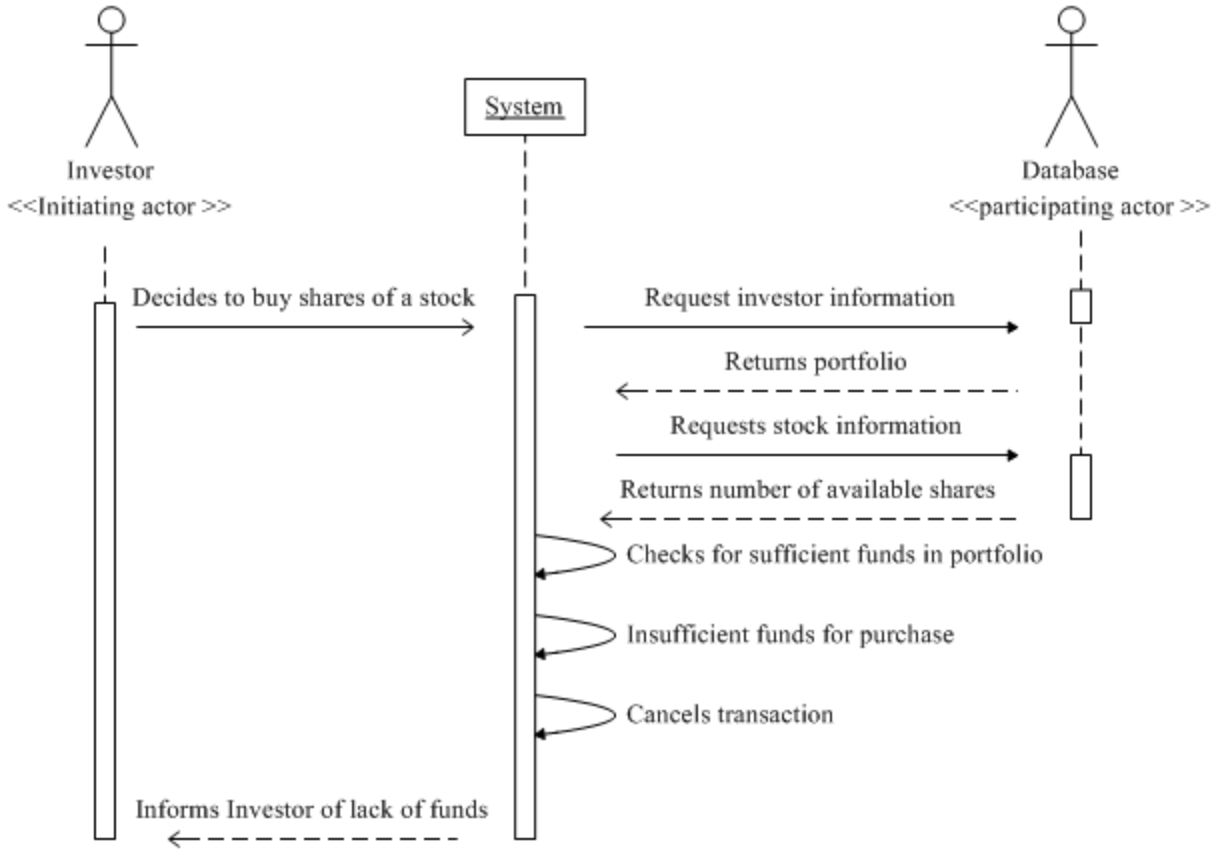
**Main Success Scenario:**

This diagram shows the purchase of a stock from the market by an Investor. The individual requests the particular stock data. The system retrieves the corresponding information from the database. When the individual decides to make a purchase, the system checks his or her portfolio treasury for adequate funds. Adequate funds are present (main success scenario). The system makes relevant modifications to the investor's portfolio as well as to the virtual market.



**Alternate Scenario I (Insufficient Funds):**

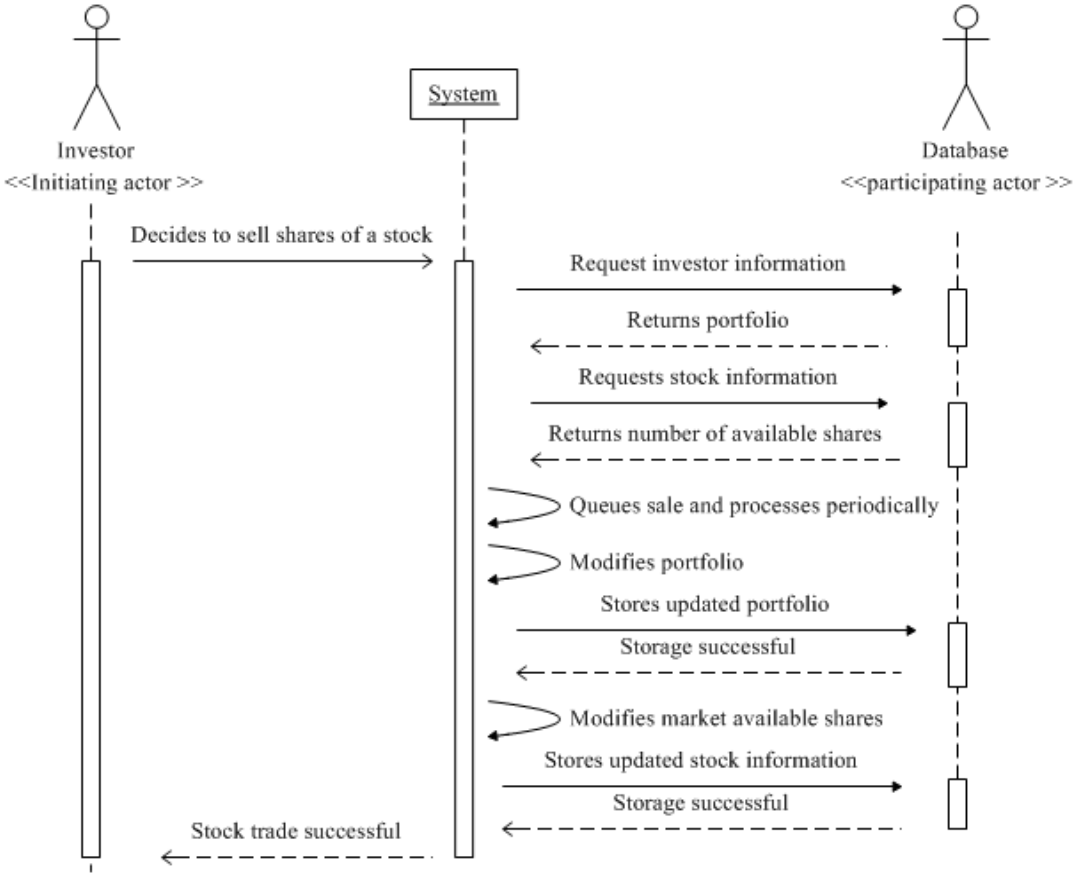
This diagram portrays the situation when events deviate from their prescribed path show in *Buy Stocks* diagram above. The investor requests stock data and the system retrieves the corresponding information from the stock database. When the individual decides to make a purchase, the system checks his or her portfolio for adequate funds. There is a lack of funds. The system cancels the purchase and notifies the investor.



**UC – 5: Sell Stocks**

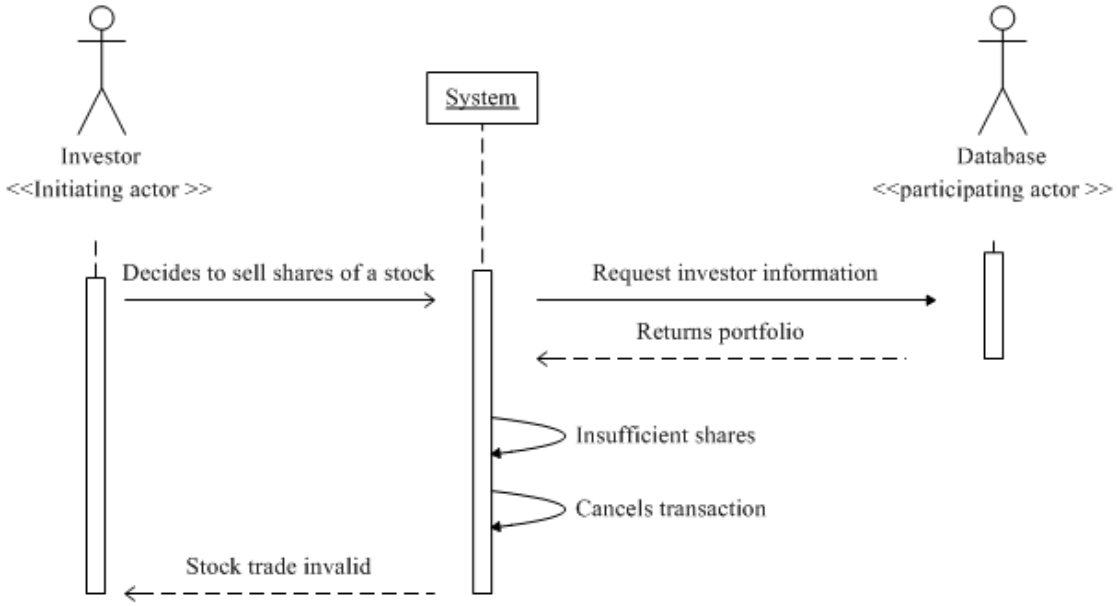
**Main Success Scenario:**

This diagram shows the sale of a stock to the market by an Investor. The individual requests the particular stock data. The system retrieves the corresponding information from the database. When the individual decides to sell, the system checks his or her portfolio for the amount of shares. Adequate shares are present (main success scenario). The system makes relevant modifications to the investor's portfolio as well as to the virtual market.



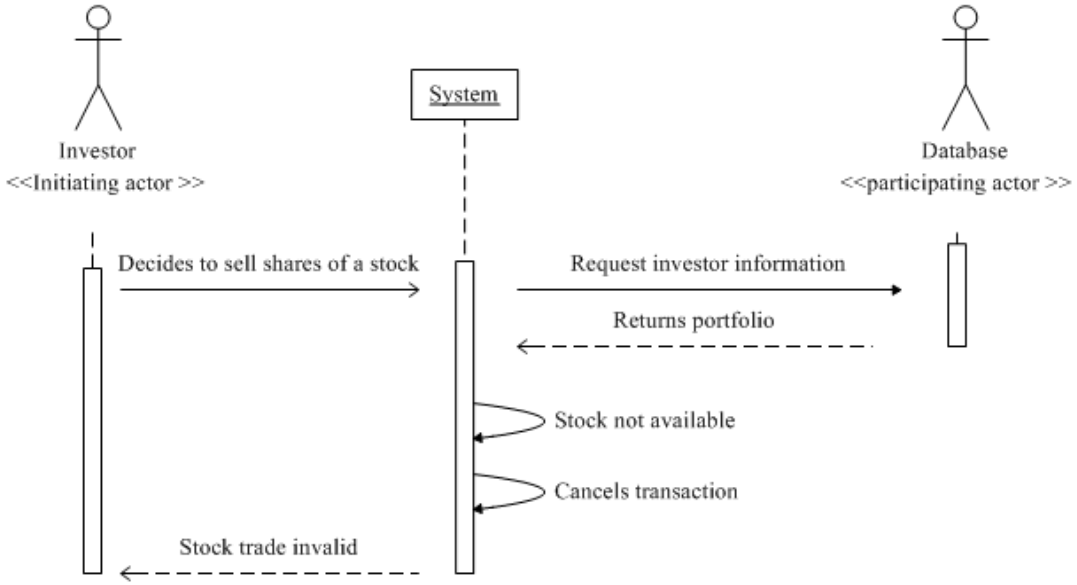
**Alternate Scenario I (Insufficient Funds):**

This diagram shows the sale of a stock to the market by an Investor when the investor's portfolio lacks shares to sell. The system notices this discrepancy and cancels the entire transaction. Finally, the investor is notified of the invalid transaction.




**Alternate Scenario II (Lack of Stock to sell):**

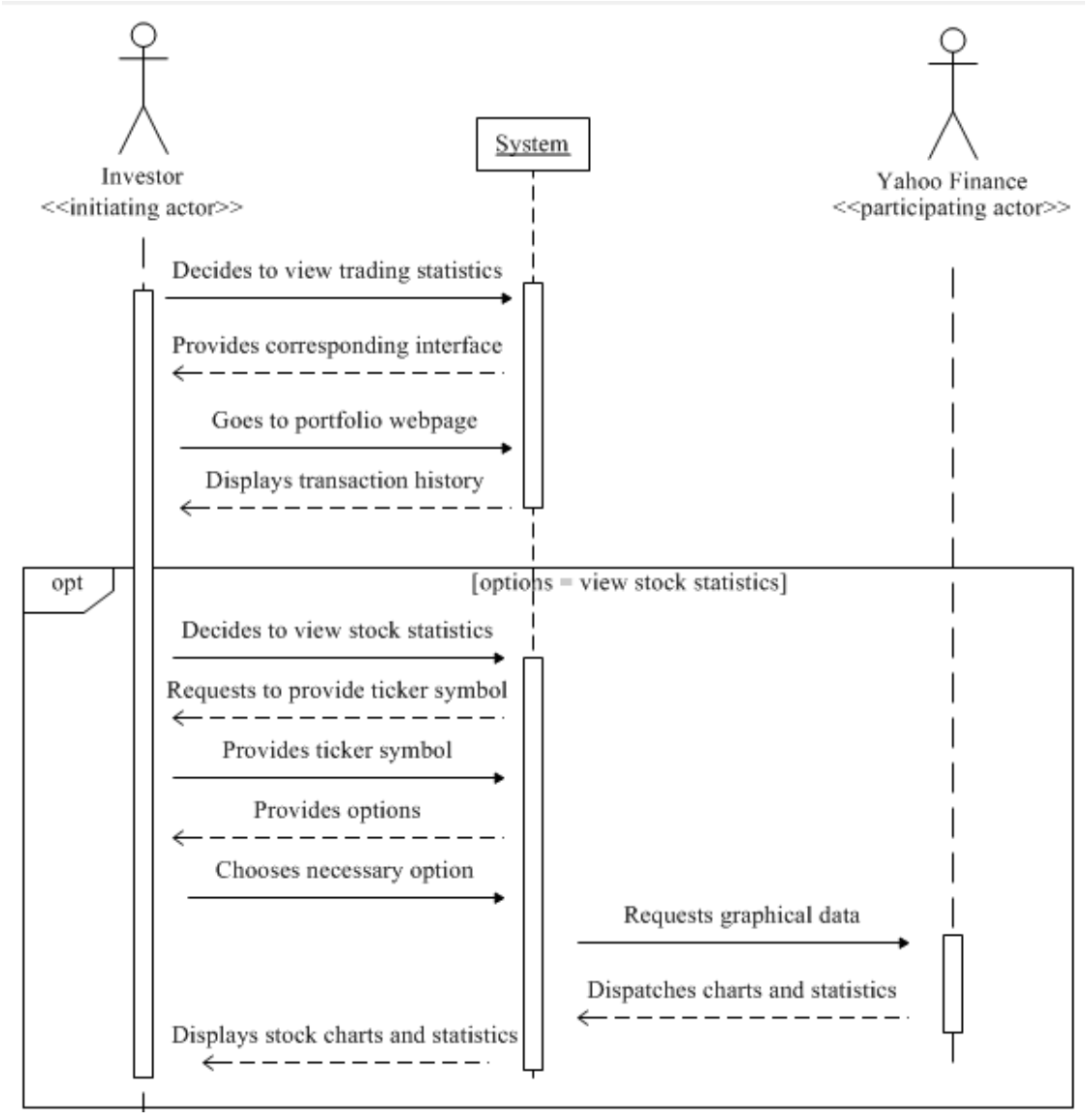
This diagram shows the sale of a stock to the market by an Investor when the investor's portfolio lacks the corresponding stock. The system notices this discrepancy and cancels the entire transaction. Finally, the investor is notified of this invalid transaction.



## UC – 6: View Statistics

### Main Success Scenario and Alternate Scenario (View Stock Statistics):

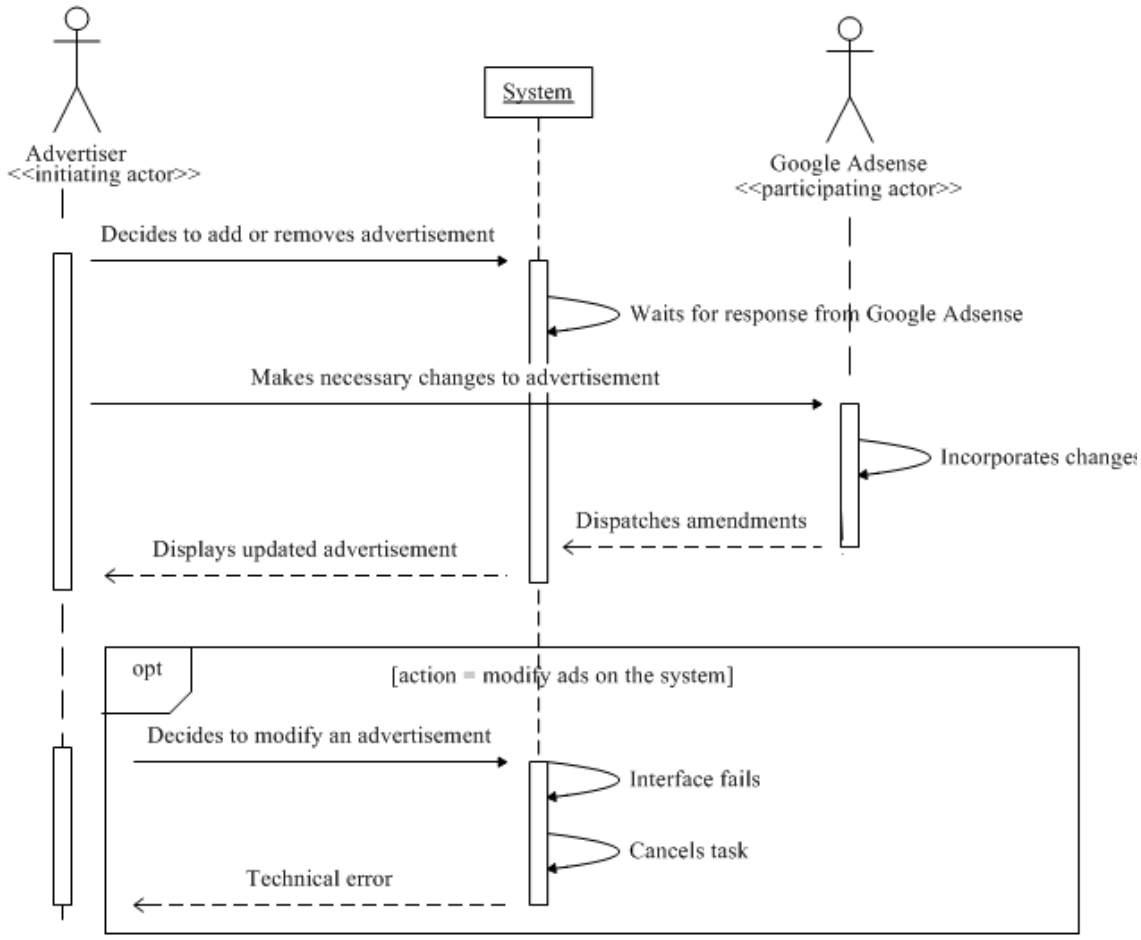
This use case allows an investor to observe certain miscellaneous information about stocks and his portfolio. An investor may choose to view his trading statistics or his stock statistics (alternate scenario). He or she navigates to his or her portfolio where the system displays the transaction history. Also, under stock chart, the system asks for a stock symbol and other options. Upon receiving all requests from the investor, the system contacts Yahoo Finance, requesting graphical data. System would then display these results to the investor.



**UC – 7: Manage Advertisements**

**Main Success Scenario and Alternate Scenario (Modify ads on system):**

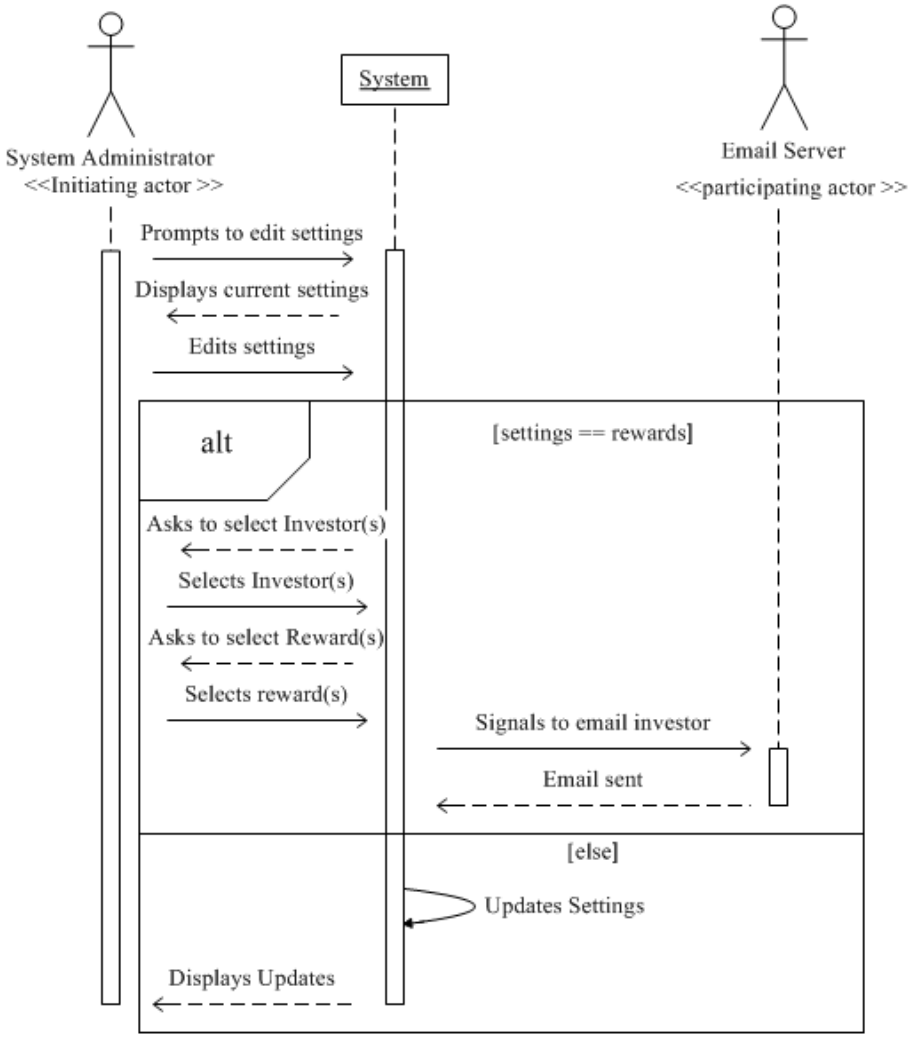
This diagram shows the steps involved in managing advertisements. An advertiser decides to add or remove advertisements. The system processes the request and prompts advertiser to make necessary changes at the external source, Google AdSense. Advertiser makes the corresponding changes on Google AdSense. Upon incorporating the changes, Google AdSense dispatches the updated advertisement to the system. The system consequently displays the updated advertisement. A technical error is thrown when the advertiser tries to modify the advertisement on the system itself as it fails to provide the appropriate interface.



**UC – 8: Maintain webpage**

**Main Success Scenario and Alternate Scenario (Modify ads on system):**

The following diagram illustrates some degree of system administrator work. The administrator may choose to modify system settings. The system would process the request and prompt the administrator to execute the changes. Administrator then would make necessary modifications. System would subsequently incorporate the changes and display an updated webpage. An alternate scenario could be when the administrator decides to reward an investor based on performance. The system would prompt for a reward. The administrator then picks and submits a reward. System processes the request and signals the Email server, which subsequently sends a congratulatory email to the investor.



# 7) NON-FUNCTIONAL REQUIREMENTS - ***FURPS+***

***Functionality*** - See Functional Requirements above.

***Usability***

The website must be easy to navigate, a delight to use, and pleasing to the eye. This requires a small amount of aesthetic design, as well as forethought into navigation, readability, etc. The user interface should be designed intelligently to meet these criteria.

***Reliability***

In case of server failure or intrusion, the site should keep a backup of users' data as well as a site-wide snapshot for recovery purposes. These backups should be created routinely and automatically for quick restorations of service. In addition, the data sources for the site should not be limited to one option, in case that source is no longer available (i.e. Yahoo).

***Performance***

The website should run on a web server without too many hardware demands, as the site itself should be as lightweight and efficient as possible. Users should be able to complete their tasks within a reasonable amount of time, even when many others are logged in/the server is updating quote and prediction information.

***Supportability***

The site and its server components should have the ability to be extended and improved using modules that either users or administrators can write, but only administrators may install them. This would give the system more flexibility for making future additions and updates. Having a system that can work on many servers to handle a large load may also be a good feature. If this is not possible, perhaps dedicating each of many servers to a specific task (like database management, stock updates, front-end, etc.)

The site should not be coded in a way so that future web servers cannot run it. Thus the site should use modern, standardized languages and protocols to accomplish the tasks defined.

***+ (Other)***

Each user's portfolio and history should be private. Their login credentials should be stored in a secure manner so that malicious users cannot break into the system and change things as they wish. Login attempts should be monitored and restricted to prevent dictionary/brute-force attacks on the system. IPs should be logged to prevent abuse of account creation tools and multiple accounts. Technologies like "CAPTCHA" should be used to prevent spam and unwanted posts/comments on the site.

# 8) USE CASE POINTS

*UCP = UUCP × TCF × ECF*

**UUCP: Unadjusted Use Case Points**

$$UUCP = UAW + UUCW$$

**Unadjusted Actor Weight (UAW):** An actor in a use case can be a person, a software program or a hardware device. The weight for an actor depends on the sophistication of the interface between the actor and the system. We present the UAW for the *Stock Market Fantasy League* project:

| Actors | Description | Complexity | Weight |
|---|---|---|---|
| **User** | Any individual who does not have an account yet. So, he plays very minimal role. | Simple | 1 |
| **Investor** | An individual who is authenticated using the login system and is interacting with the system with a portfolio using GUI. | Complex | 3 |
| **Yahoo Finance** | The external source where stock information is obtained. This interacts with our system using a Stock API. | Average | 2 |
| **Database** | A place where information about the various stocks and investors are stored. This interacts with a protocol. | Average | 2 |
| **Email Server** | A machine responsible for sending messages to investors via E-mail and SMS. This interacts through an API. | Average | 2 |
| **System Administrator** | An individual who interacts with the system through an admin account and is responsible for maintaining and managing the system. | Complex | 3 |
| **Advertiser** | An individual who interacts with the system through a user account and posts stock relating advertisements. He or she will perform this at Google. | Simple | 1 |
| **Google AdSense** | The external source by which advertisers can display pertaining advertisements on the system. Communication through Protocol. | Average | 2 |

In accordance with the data above, the UAW is calculated as follows:

UAW(*Fantasy league*) = ∑ [(Complexity Weight) x (# of Actors associated with Complexity)]

UAW(*Fantasy league*) = [(1) x (2)] + [(2) x (4)] + [(3) x (2)] = $\boxed{16}$

**Unadjusted Use Case Weight (UUCW):** The complexity level of the use cases is primarily derived from the number of steps in the main success scenario. Nonetheless, the number of participating actors, and the number of steps in the alternate scenario play a considerable role as well. Below we present the UUCW for our project:

| Use Case | Description | Category | Weight |
|---|---|---|---|
| **Update Database (UC-1)** | No user interface. Two participating actors. Ten steps for the main success scenario. | Simple | *5* |
| **Create Account (UC-2)** | Simple user interface. Two participating actors. Seven steps for the main success scenario. | Simple | *5* |
| **Manage Portfolio (UC-3)** | Simple user interface. Two participating actors. Five steps for the main success scenario. | Average | 10 |
| **Buy Stocks (UC-4)** | Moderate user interface. One participating actor. Eleven steps for the main success scenario. | Complex | 15 |
| **Sell Stocks (UC-5)** | Moderate user interface. One participating actor. Ten steps for the main success scenario. | Complex | 15 |
| **View Statistics (UC-6)** | Complex user interface. One participating actor. Three steps for the main success scenario. | Average | 10 |
| **Manage Advertisements (UC-7)** | No user interface. One participating actor. Five steps for the main success scenario. | Simple | *5* |
| **Maintain Webpage (UC-8)** | Complex user interface. Two participating actors. Five steps for the main success scenario. | Complex | 15 |

Based on the data above, the UUCW is as follows:
UUCW(*Fantasy league*) = ∑ (Complexity Weight) x (# of Use Cases in Complexity category)
UUCW(*Fantasy league*) = [(5) x (2)] + [(10) x (2)] + [(15) x (3)] = 75

$$UUCP = UUCW + UAW = 75 + 16 = 91$$

**TCF: Technical Complexity Factor**

$$TCF = \text{Constant-1} + \text{Constant-2} \times \text{Technical Factor Total} = C_1 + C_2 \cdot \sum_{i=1}^{13} W_i \cdot F_i$$

where,

Constant-1 ($C_1$) = 0.6

Constant-2 ($C_2$) = 0.01

Technical factors identify and estimate the impact on productivity of the overall project due to the involvement of non-functional requirements (FURPS+). There are thirteen standard technical factors. We present the TCF of our project below:

| Technical Factor | Description | Weight | Perceived Complexity | Calculated Factor |
|---|---|---|---|---|
| **T1** | Distributed system | 2 | 2 | 4 |
| **T2** | Performance objectives (Users expect exceptional Performance) | 1 | 4 | 4 |
| **T3** | End-user efficiency (End-user expects demanding efficiency) | 1 | 4 | 4 |
| **T4** | Complex internal processing | 1 | 5 | 5 |
| **T5** | Reusable design or code | 1 | 2 | 2 |
| **T6** | Easy to install | 0.5 | 0 | 0 |
| **T7** | Easy to use (Ease of use is very important) | 0.5 | 3 | 1.5 |
| **T8** | Portable | 2 | 0 | 0 |
| **T9** | Easy to change | 1 | 1 | 1 |
| **T10** | Concurrent use | 1 | 2 | 2 |
| **T11** | Special security features | 1 | 2 | 2 |
| **T12** | Provides direct access for third parties | 1 | 0 | 0 |
| **T13** | Special user training facilities are required | 1 | 0 | 0 |
| | | | **TOTAL** | 25.5 |

Based on the data above, the TCF is as follows:

TCF(*Fantasy League*) = 0.6 + [(0.01) x 25.5] = 0.855

**ECF: Environmental Complexity Factor**

$$ECF = \text{Constant-1} + \text{Constant-2} \times \text{Environmental Factor Total} = C_1 + C_2 \cdot \sum_{i=1}^{8} W_i \cdot F_i$$

where,

Constant-1 ($C_1$) = 1.4

Constant-2 ($C_2$) = –0.03

Environmental factors identify and estimate the impact on productivity of the overall project due to the experience of the development team. There are eight standard technical factors. We present the ECF of our project below:

| Environmental Factor | Description | Weight | Perceived Impact | Calculated Factor |
|---|---|---|---|---|
| **E1** | Beginner familiarity with the UML based development | 1.5 | 1 | 1.5 |
| **E2** | Some familiarity with application problem | 0.5 | 1 | 0.5 |
| **E3** | High knowledge of object-oriented approach | 1 | 5 | 5 |
| **E4** | Beginner lead analyst | 0.5 | 1 | 0.5 |
| **E5** | High Motivation | 1 | 5 | 5 |
| **E6** | Stable requirements expected | 2 | 2 | 4 |
| **E7** | No part-time staff will be involved | -1 | 0 | 0 |
| **E8** | Programming language of high difficulty will be used | -1 | 5 | -5 |
| | | | **TOTAL** | 11.5 |

Based on the data above, the TCF is as follows:

ECF(*Fantasy League*) = 1.4 + [(-0.03) x 11.5] = 1.055

**Calculating the Use Case Points (UCP)**

*Total UCP = UUCP × TCF × ECF*
*= 91* × 0.855 × 1.055 = 82.08 = 82 UCPs

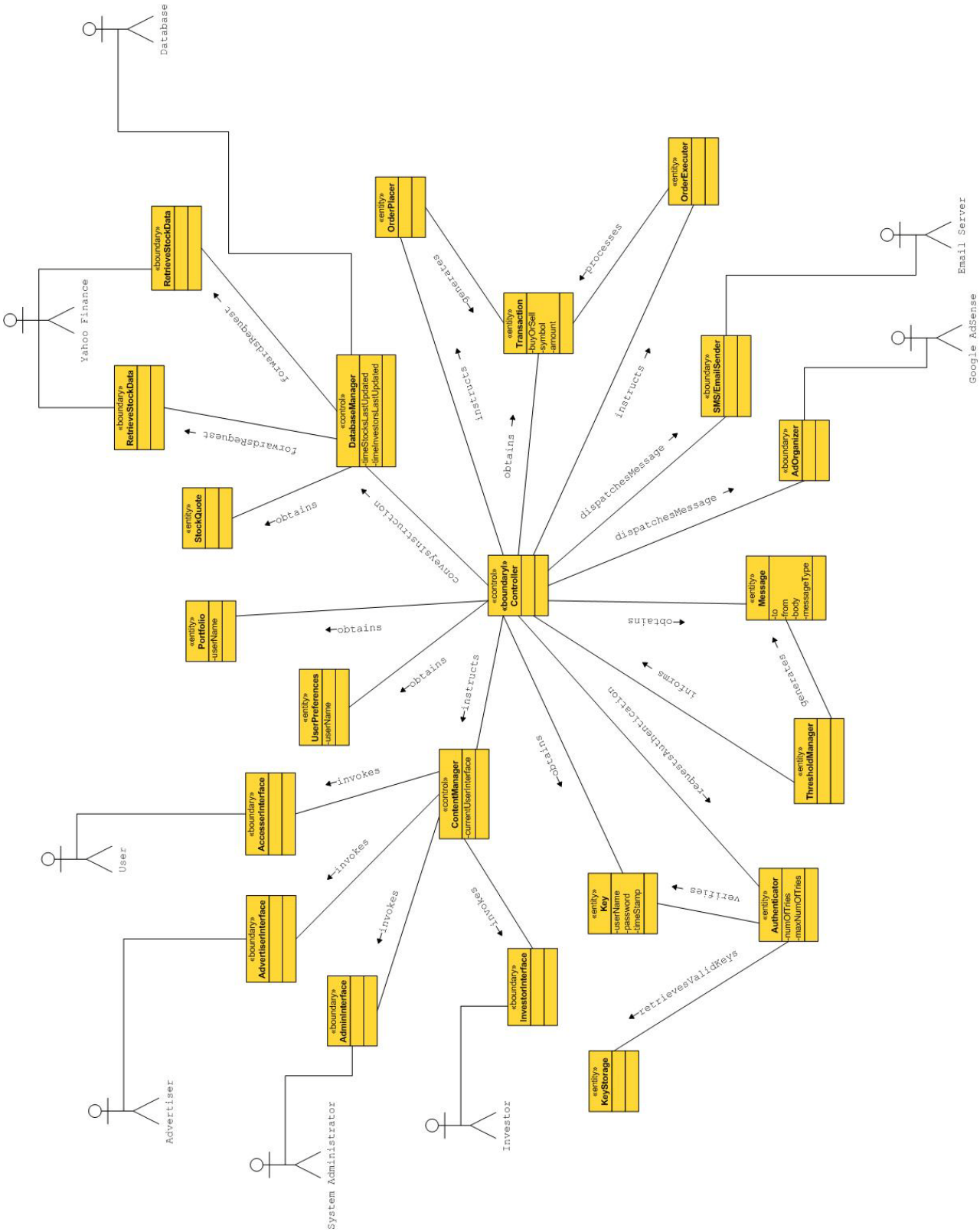
**Deriving Project Duration from Use Case Points**

*Duration = UCP × PF*

PF is the productivity factor. Although this is another factor that needs to be estimated, we were told to assume the Productivity Factor as 28 hours per point.
Therefore,
**Total Duration of *Fantasy League*** = 82 × 28 = 2296 hours
= 96 days
= 3.08 months = **3 Months** *(31 day months)*

# 9) DOMAIN ANALYSIS

## a) Domain Model



**NOTE:** *In our presentation of Domain Analysis here, we refer to the "User" Actor as an "Accessor".*

### i) Concepts Definitions

(One Responsibility is mapped to one Concept)

Type = Responsibility Type → "D" = Doing, "K" = Knowing

| | Concept Name | Type | Boundary? | Responsibility Description |
|---|---|---|---|---|
| 1 | Controller | D | No | Coordinate actions and information related to a logical subset of the Use Cases |
| 2 | ContentManager | D | No | Coordinates with the controller to present the correct user interface |
| 3 | AccessorInterface | D | Yes | System interface for the Accessor actor; the interface with which all actors interact before identifying themselves to the system |
| 4 | AdminInterface | D | Yes | System interface for the System Administrator actor |
| 5 | InvestorInterface | D | Yes | System interface for the Investor actor |
| 6 | AdvertiserInterface | D | Yes | System interface for the Advertiser actor |
| 7 | RetrieveStockData | D | Yes | System interface to the source for stock information; this is the mechanism used for retrieving one stock quote from Yahoo Finance |
| 8 | RetrieveStockChart | D | Yes | System interface to the source for stock information; this is the mechanism used for retrieving a stock data from Yahoo Finance in the form of charts |
| 9 | DatabaseManager | D | Yes | System interface to the database; used for retrieving data from, and storing data in, database |
| 10 | Authenticator | D | No | Uses the User Actor's input to authenticate the User Actor as an Investor, Administrator, etc. |
| 11 | AdOrganizer | D | Yes | Organizes Advertisements (which are retrieved based on Google Adsense) on user interfaces |
| 13 | Key | K | No | Contain the authentication information inputted by an Accessor |
| 14 | KeyStorage | K | No | Internal database for securely storing valid authentication keys for Investors, System Administrators, and Advertisers |
| 15 | SMS/EmailSender | D | Yes | System interface to the Email and SMS systems |

| | | | | |
|---|---|---|---|---|
| 16 | UserPreferences | K | No | Holds the preferences of an investor |
| 17 | ThresholdManager | D | No | Updates users' profiles based on their preferences, and current stock data; notifies email & sms systems under appropriate conditions |
| 18 | Portfolio | K | No | Contains the portfolio of a user; this is information which will be stored in the database |
| 19 | Message | K | No | Contains message to be dispatched by the controller to the SMS/EmailSender |
| 20 | StockQuote | K | No | Contains information pertaining to one stock quote; this is data which will be stored in the database |
| 21 | Transaction | K | No | Contains information pertaining to a particular transaction initiated by an Investor |
| 22 | OrderPlacer | D | No | Uses transaction request information and submits the order to OrderExecuter |
| 23 | OrderExecuter | D | No | Performs transactions, which are submitted by OrderPlacer, at the appropriate time |

### ii) Association Definitions

| Concept | Relation [direction: →] | Concept |
|---|---|---|
| Controller | instructs | ContentManager |
| Controller | obtains | UserPreferences |
| Controller | obtains | Portfolio |
| Controller | conveysInstruction | DatabaseManager |
| ThresholdManager | informs | Controller |
| Controller | dispatchesMessage | SMS/EmailSender |
| Controller | requestsAuthentication | Authenticator |
| Controller | obtains | Key |
| Controller | obtains | Message |
| Controller | obtains | Transaction |
| Authenticator | verifies | Key |
| Authenticator | retrievesValidKeys | KeyStogage |
| ContentManager | invokes | AccessorInterface |
| ContentManager | invokes | InvestorInterface |
| ContentManager | invokes | AdvertiserInterface |
| ContentManager | invokes | AdminInterface |
| DatabaseManager | forwardsRequest | RetrieveStockData |
| DatabaseManager | forwardsRequest | RetrieveStockChart |
| DatabaseManager | generates | StockQuote |

| ThresholdManager | generates | Message |
|---|---|---|
| AdOrganizer | instructs | ContentManager |
| Controller | instructs | OrderPlacer |
| Controller | instructs | OrderExecuter |
| OrderPlacer | generates | Transaction |
| OrderExecuter | processes | Transaction |

### iii) Attribute Definitions

| Concept | Attributes |
|---|---|
| Controller | |
| ContentManager | (1) `currentUserInterface`: indicates which user interface is being presented to the user of the system |
| AccessorInterface | |
| AdminInterface | |
| InvestorInterface | |
| AdvertiserInterface | |
| RetrieveStockData | |
| RetrieveStockChart | |
| DatabaseManager | (1) `timeStocksLastUpdated`: time stamp representing when the Stock Database was last updated<br>(2) `timeInvestorsLastUpdated`: time stamp representing when the Investor Database was last updated |
| Authenticator | (1) `numOfTries`: number of failed login attempts during current session<br>(2) `maxNumTries`: number of failed login attempts before user account is temporarily locked |
| AdOrganizer | (1) `publisherId`: administrators publisher id given by AdSense |
| Key | (1) `userName`: a user's login id<br>(2) `password`: a user's authentication key<br>(3) `timeStamp`: time stamp of current login attempt |

| | |
|---|---|
| KeyStorage | |
| SMS/EmailSender | |
| UserPreferences | (1) `userName`: a user's login id |
| ThresholdManager | |
| Portfolio | (1) `userName`: a user's login id |
| Message | (1) `to`: target recipient identifier<br>(2) `from`: identifiable system name<br>(3) `body`: contents of the message<br>(4) `messageType`: indicates whether message is for email, or sms transmissions |
| | (1) `buyOrSell`: indicates which type of transaction<br>(2) `symbol`: indicates which stock symbol<br>(3) `amount`: indicates amount |
| StockQuote | |
| OrderPlacer | |
| OrderExecuter | |

## b) System Operation Contracts

| Operation: UC1 – Update Database |
|---|
| Preconditions:<br>• Internal system timer elapses<br>• Database contains table of stock symbols |
| Postconditions:<br>• Internal system timer is reset<br>• Database contains the necessary up-to-date information associated with relevant stock symbols |

| Operation: UC2 – Create Account |
|---|
| Preconditions:<br>• Email server is in proper functioning order |
| Postconditions:<br>• If outcome is successful<br>- The system has records of a new account and its associated information<br>- The spending power for the portfolio of the account is set to $100,000<br>• If outcome is not successful<br>- No changes have been made to the system |

| Operation: | UC3 – Manage Portfolio |
|---|---|
| Preconditions: | • The initiating Investor has an account in the system and is logged into the system |
| Postconditions: | • The Investor account, stored in the database, reflects the requested changes |

| Operation: | UC4 – Buy Stocks |
|---|---|
| Preconditions: | • Initiating Investor is logged into the system<br>• Shares of desired stock are available in sufficient quantity<br>• The Investor portfolio contains sufficient funds |
| Postconditions: | • Appropriate changes are made to the relevant Investor portfolio<br>- spending power of the Investor is decreased by the transaction price<br>- a commission (system fee) is applied<br>- the number of available shares (in the database) of the corresponding stock is reduced by the correct amount |

| Operation: | UC5 – Sell Stocks |
|---|---|
| Preconditions: | • Initiating Investor is logged into the system<br>• The Investor portfolio contains a sufficient number shares of the stock to be sold |
| Postconditions: | • Appropriate changes are made to the relevant Investor portfolio<br>- spending power of the Investor is increased by the transaction price<br>- a commission (system fee) is applied<br>- the number of available shares (in the database) of the corresponding stock is increased by the correct amount |

| Operation: | UC6 – View Statistics |
|---|---|
| Preconditions: | • Initiating Investor is logged into the system<br>• Investor portfolio has displayable information |
| Postconditions: | • Investor Actor views information which is displayed |

| Operation: | UC7 – Manage Advertisements |
|---|---|
| Preconditions: | • Initiating Advertiser is logged into the system<br>• Advertiser has pre-approved advertisements |
| Postconditions: | • Changes in advertisement options are reflected in the user interfaces for all users accessing the system |

| Operation: | UC8 – Maintain Webpage |
|---|---|
| Preconditions: | • System Administrator is logged into the system<br>• Email server is online and operational |
| Postconditions: | • System reflects all changes made by the administrator<br>• If applicable, all intended emails are sent via the Email Server |

# 10) INTERACTION DIAGRAMS

## a) List of Select Use Cases

See Section 6c for descriptions of these use cases.

- **UC 1 – Update Database**
- **UC 2 – Create Account**
- **UC 3 – Manage Portfolio**
- **UC 4 - Buy Stocks**
- **UC 5 - Sell Stocks**
- **UC 6 - View Statistics**
- **UC 7 – Manage Advertisements**
- **UC 8 – Maintain webpage**

## b) Diagrams for Selected Use Cases

Each diagram assumes that Drupal is an "object" that communicates with our modules through its function calls. For all intents and purposes, other modules loaded into Drupal, including but not limited to Drupal core, will be assumed to be contained in the Drupal object, unless explicitly shown.

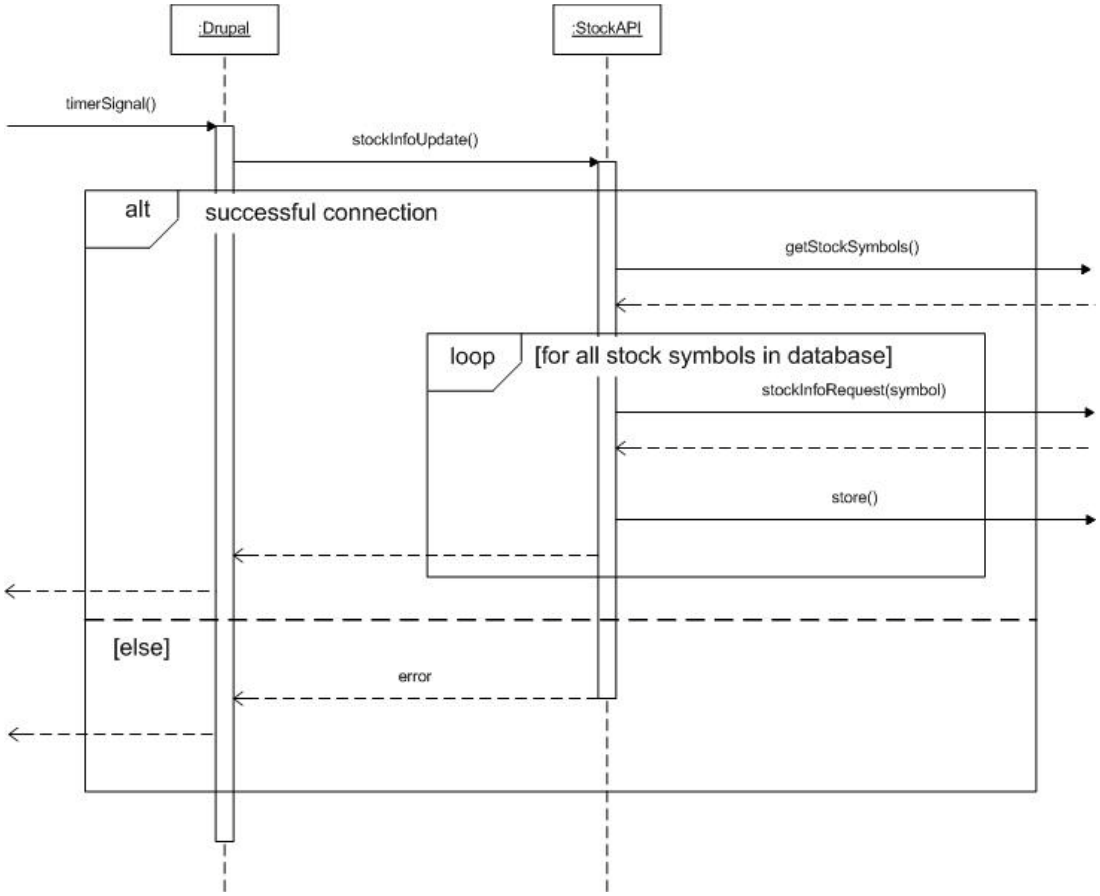



FIGURE 10-1 - INTERACTION SEQUENCE DIAGRAM FOR USE CASE 1

This interaction sequence diagram represents **UC 1 Database Update**. One iteration of the update process is depicted in this diagram. In order to maintain a real time environment in the fantasy game we use a timer to initiate periodic updates the Database. The Database holds is an internal representation of the real world Stock Market; hence, this is a central aspect of our system. Drupal, a central component of the system, in this diagram, serves to forward the timer's signal to the StockAPI module which in turn can the necessary action. Logically, the brunt of the work is handled by the StockAPI module. The StockAPI module must retrieve the table of stock symbols currently stored in the Database, query Yahoo! Finance for updated information for each symbol, and store the new data back into the table.

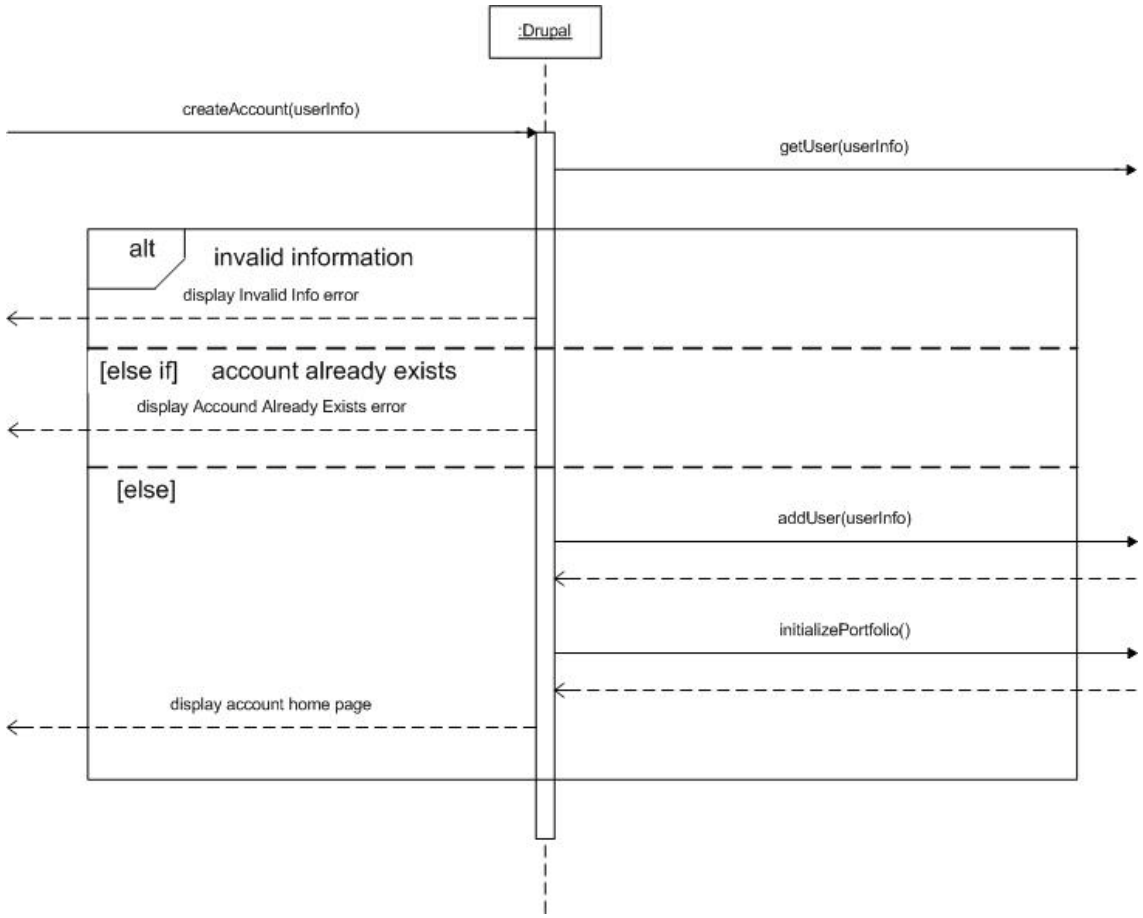



FIGURE 10-2 - INTERACTION SEQUENCE DIAGRAM FOR USE CASE 2

This interaction sequence diagram represents **UC 2 Create Account.** The timeline shown is that of when the user accessing our website initiates the action of creating an account to when he/she gets confirmation of successfully creating a new account. This diagram involves only the Drupal system which accepts the information given by an Accessor, and attempts to find the user in the system database. If invalid information was provided, an error is displayed indicating the same. If the creation of a new account cannot be complete because of an already existing account, the appropriate error message is shown. If there are no problems during the process, a new account is set up, and the associated Investor Profile is initialized with no stocks and $100,000 of spending power.

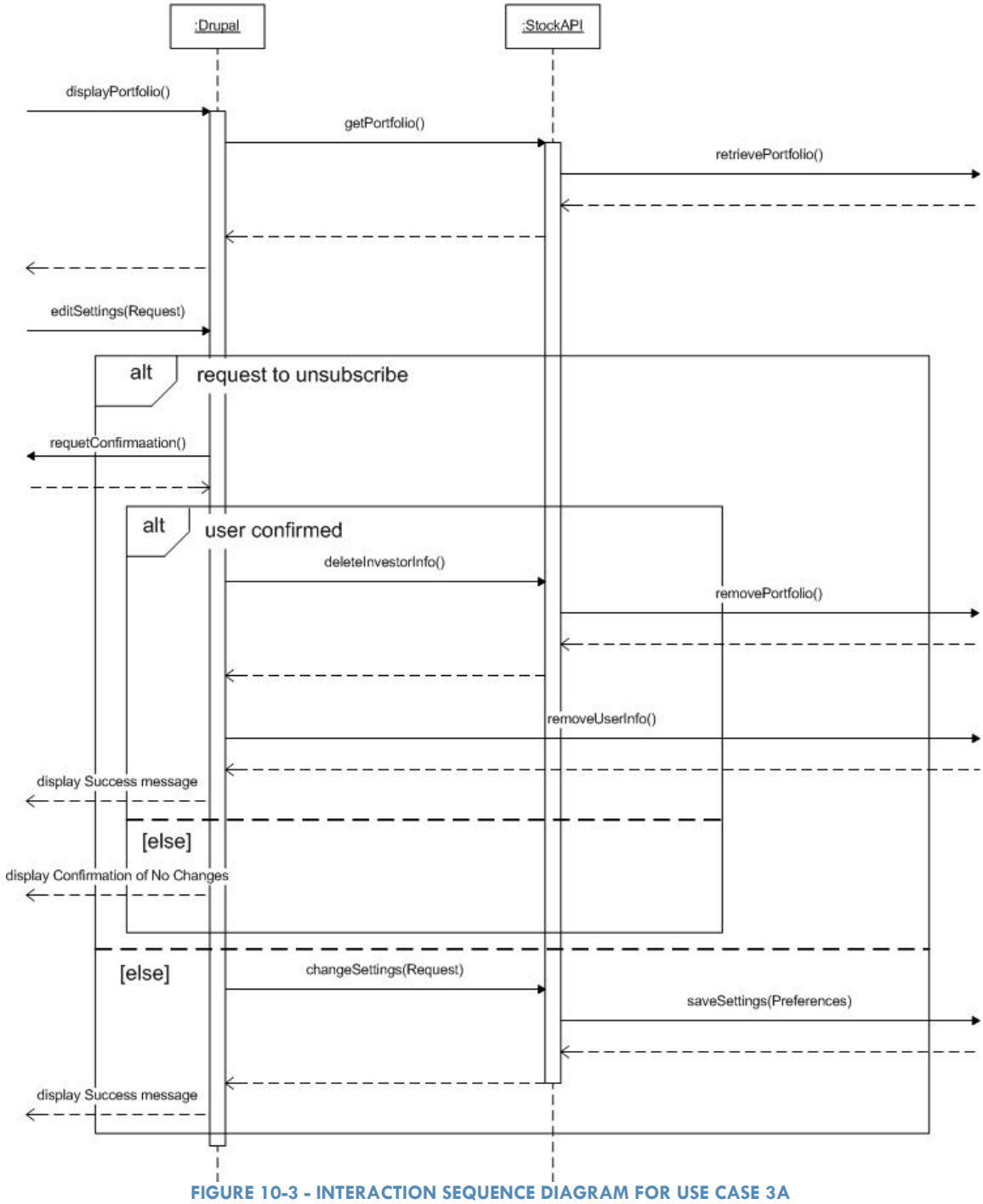



FIGURE 10-3 - INTERACTION SEQUENCE DIAGRAM FOR USE CASE 3A

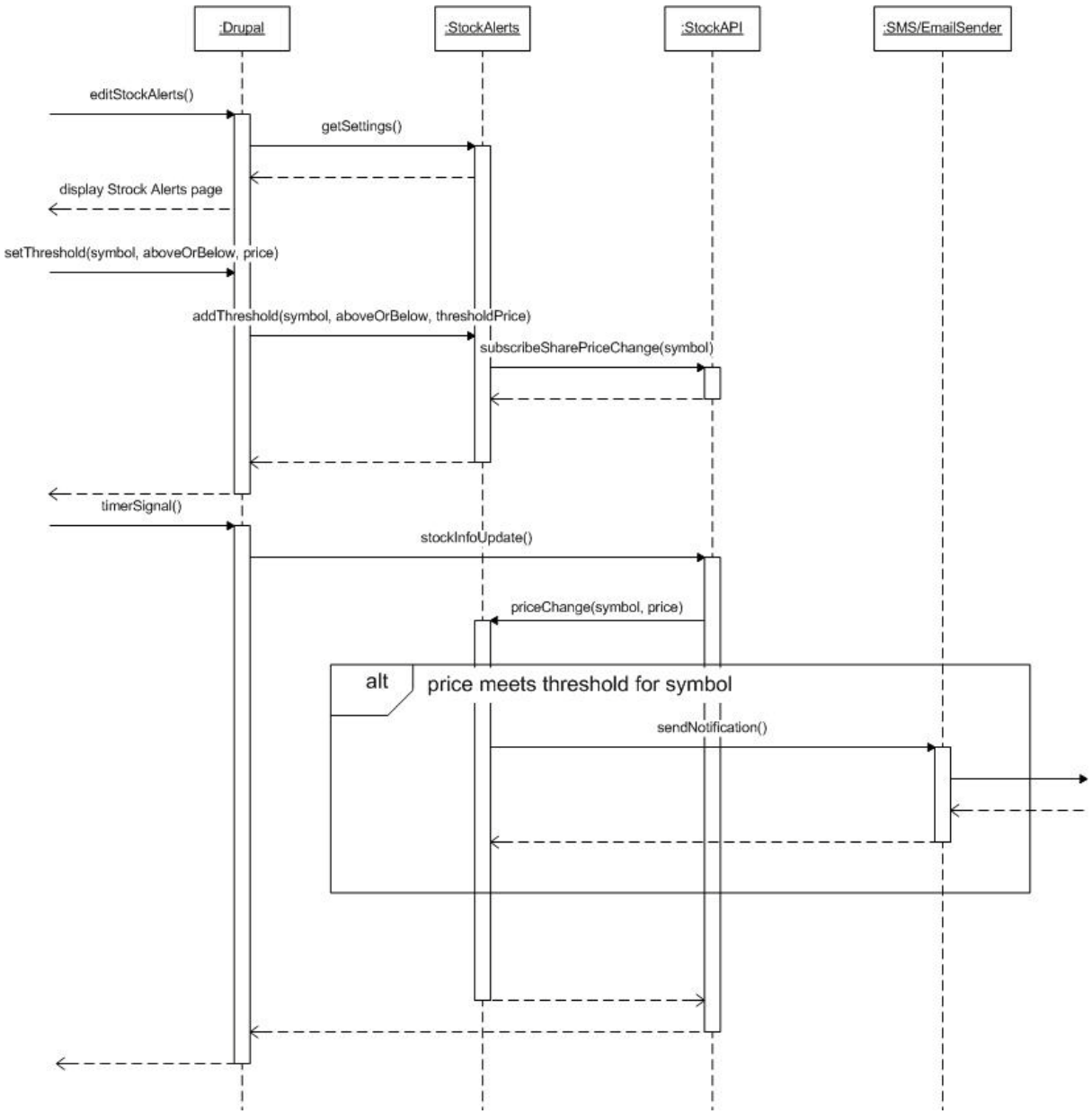



FIGURE 10-4 - INTERACTION SEQUENCE DIAGRAM FOR USE CASE 3B

This two interaction sequence diagrams represent **UC 3 Manage Portfolio.**
First Figure - The timeline shown is that of when the user submits a request to change some settings to when he/she gets confirmation of successfully applying the changes to their account/portfolio. The user first uses the interface to specify what changes are to be made. Drupal processes most of the changes requested, however requesting to unsubscribe from the system requires the StockAPI. In this case Drupal confirms the action and passes the request on to the StockAPI. The StockAPI then removes the Investor information from the database. Once everything is completed, a confirmation of the actions taken is printed.
Second Figure - The timeline shown is that of when the user submits a request to change some threshold settings to when he/she gets confirmation of successfully applying the changes to their

account/portfolio. The user first uses the interface to specify which stock symbol, what price, and whether the threshold is a high or low value. To accomplish the threshold management, the StockAlerts module subscribes to the StockAPI for receiving information about when the price for the specified stock symbol changes. When the timer signal comes for evaluating threshold values, StockAlrets check if thresholds are met, and if so, it uses the SMS/EmailSender module to send the appropriate notification to the Investor.

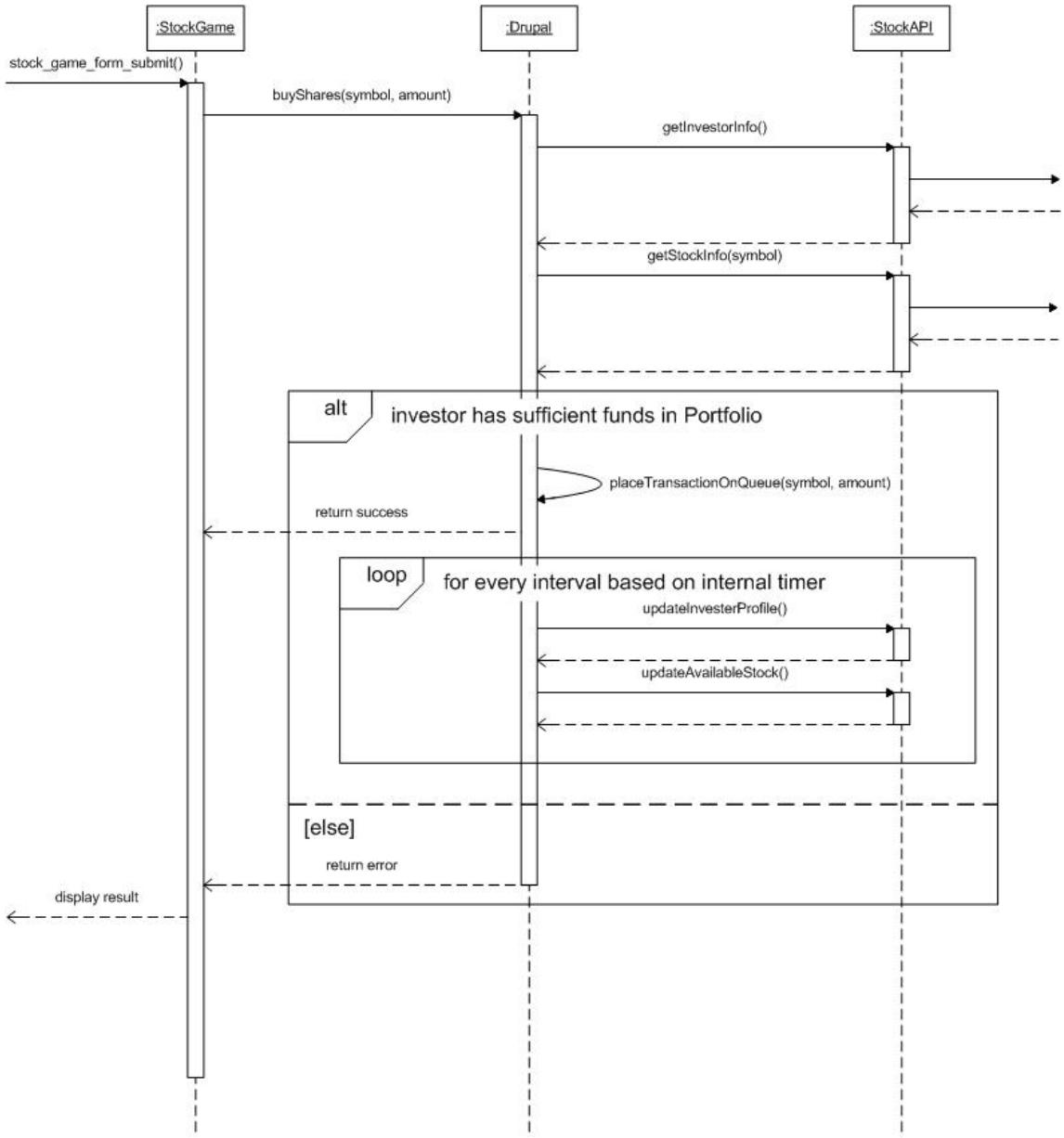



FIGURE 10-5 - INTERACTION SEQUENCE DIAGRAM FOR USE CASE 4

This interaction sequence diagram represents **UC 4 Buy Stocks.** The timeline shown is that of when the user initiates the action to when he/she gets confirmation of a submitted order. The user first uses the interface to specify what stock to buy and how much (symbol and quantity). Drupal passes this information along to the StockAPI. The current status of the Investor making placing the order is retrieved. The validity of the request is checked. And finally, the order is placed in a

queue for processing. The transaction is executed at an appropriate time decided on by the system.

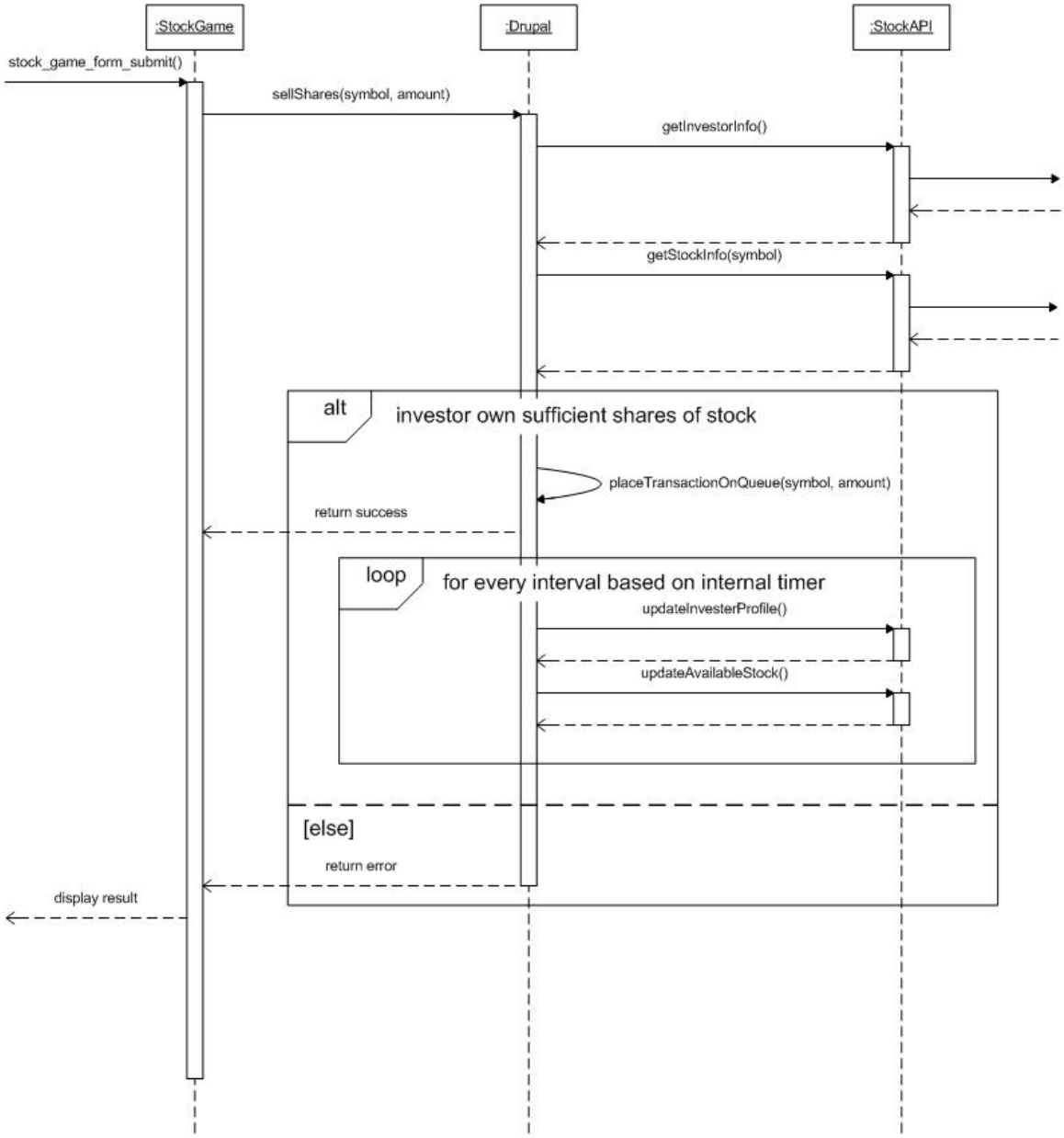



FIGURE 10-6 - INTERACTION SEQUENCE DIAGRAM FOR USE CASE 5

This interaction sequence diagram represents **UC 5 Sell Stocks.** The timeline shown is that of when the user initiates the action to when he/she gets confirmation of a submitted order. The user first uses the interface to specify what stock to sell and how much (symbol and quantity). Drupal passes this information along to the StockAPI. The current status of the Investor making placing the order is retrieved. The validity of the request is checked. And finally, the order is placed in a queue for processing. The transaction is executed at an appropriate time decided on by the system.

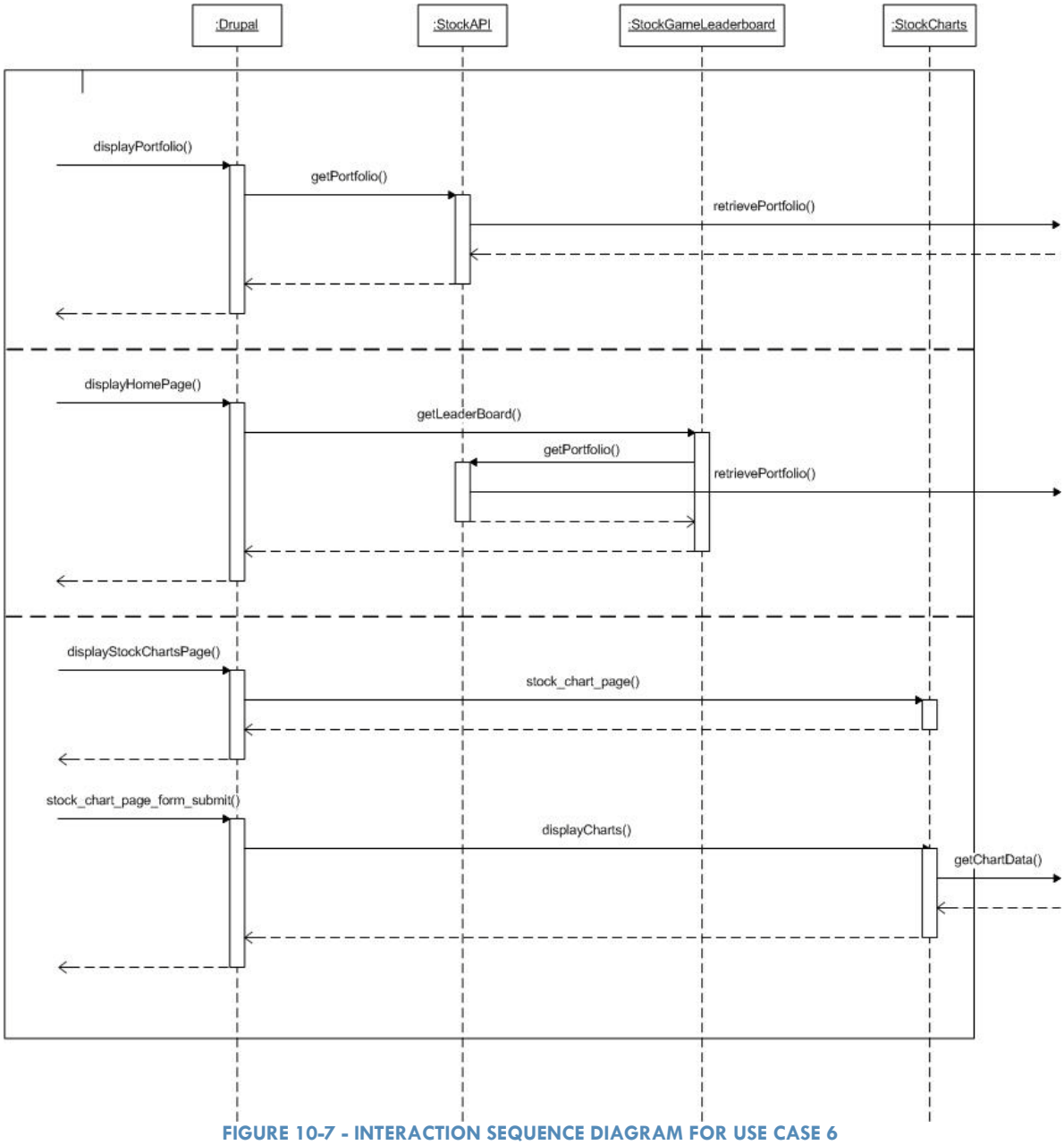



FIGURE 10-7 - INTERACTION SEQUENCE DIAGRAM FOR USE CASE 6

This interaction sequence diagram represents **UC 6 View Statistics.** A user wishes to see the statistics collected on his/her portfolio, and Drupal displays possible options for viewing the statistics. The user chooses various options and those selections are forwarded to the appropriate entities. Depending on the option chosen, the user can see their transaction history, leaderboard, and stock history. Each selection has its own set of communication paths as seen above.

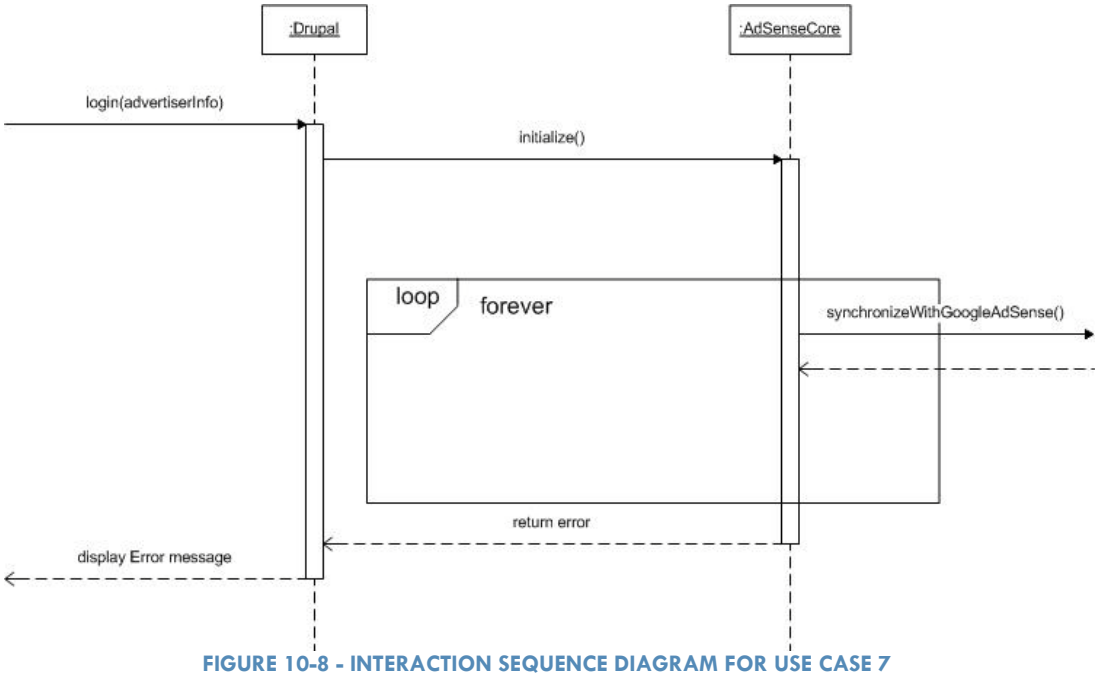



FIGURE 10-8 - INTERACTION SEQUENCE DIAGRAM FOR USE CASE 7

This interaction sequence diagram represents **UC 7 Manage Advertisements.** An advertiser logs in to the system with a special privileged account. The advertiser clicks on Google AdSense Manager, which displays the configuration of where the advertisements are placed on the website. Then, advertisements are retrieved using dynamic embedded html code.

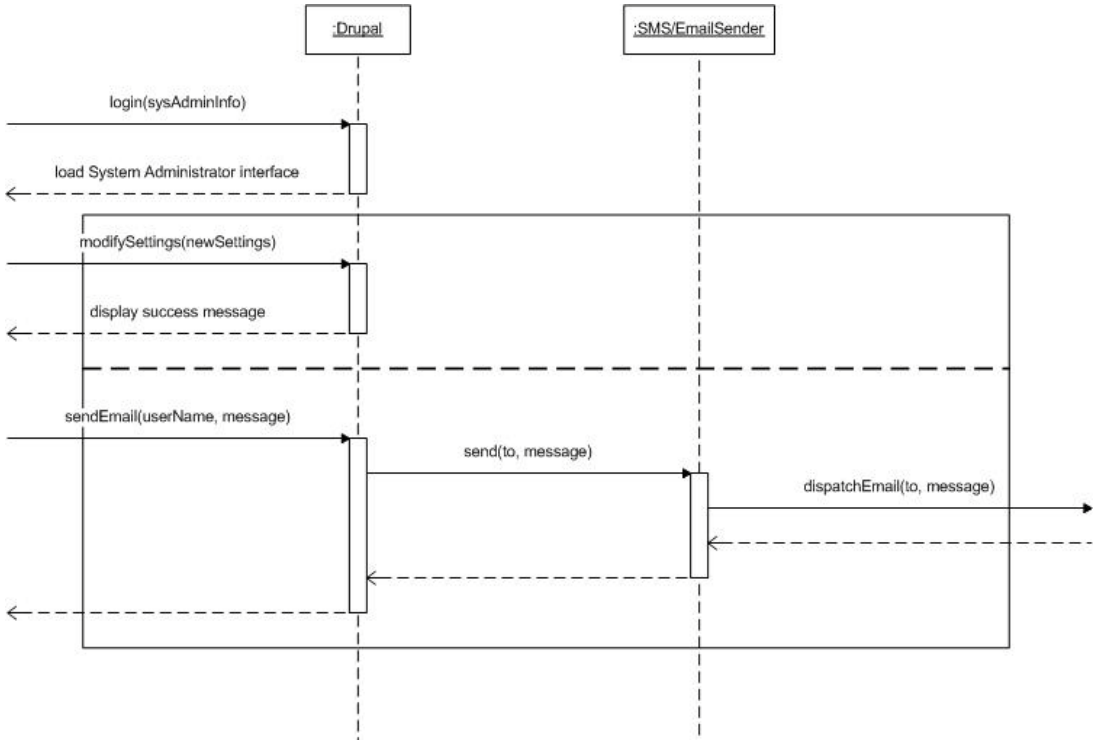
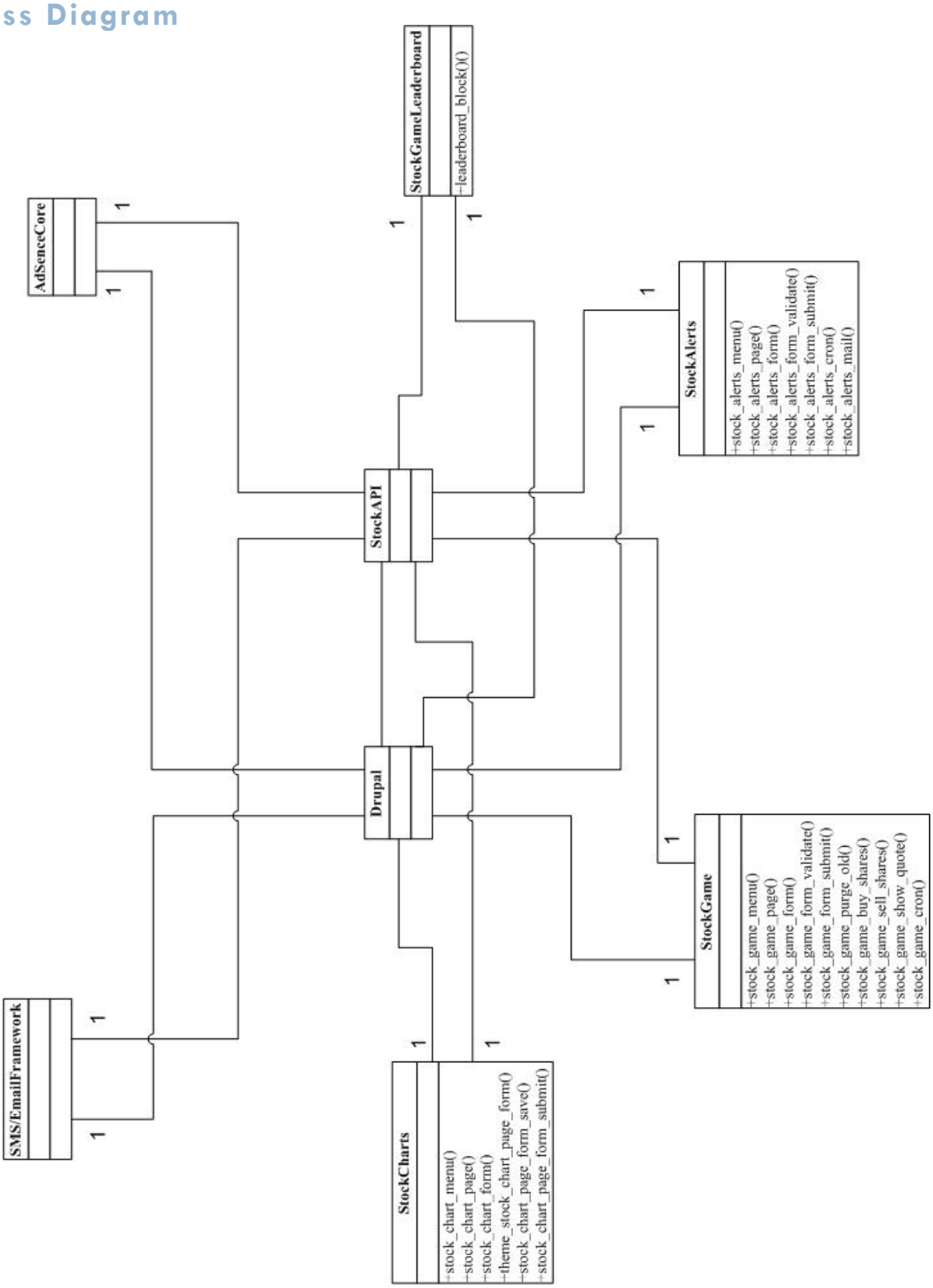



FIGURE 10-9 - INTERACTION SEQUENCE DIAGRAM FOR USE CASE 8

This interaction sequence diagram represents **UC 8 Maintain Webpage.** An administrator logs in to the system with a special privileged account. The administrator can perform various maintenance tasks, scheduling tasks, and content management. Possible task include sending site update emails to all users, selecting Investor of the month, and cleaning outdated pages. The above diagram illustrates the functionality of emailing all users in a subscribed to group list.

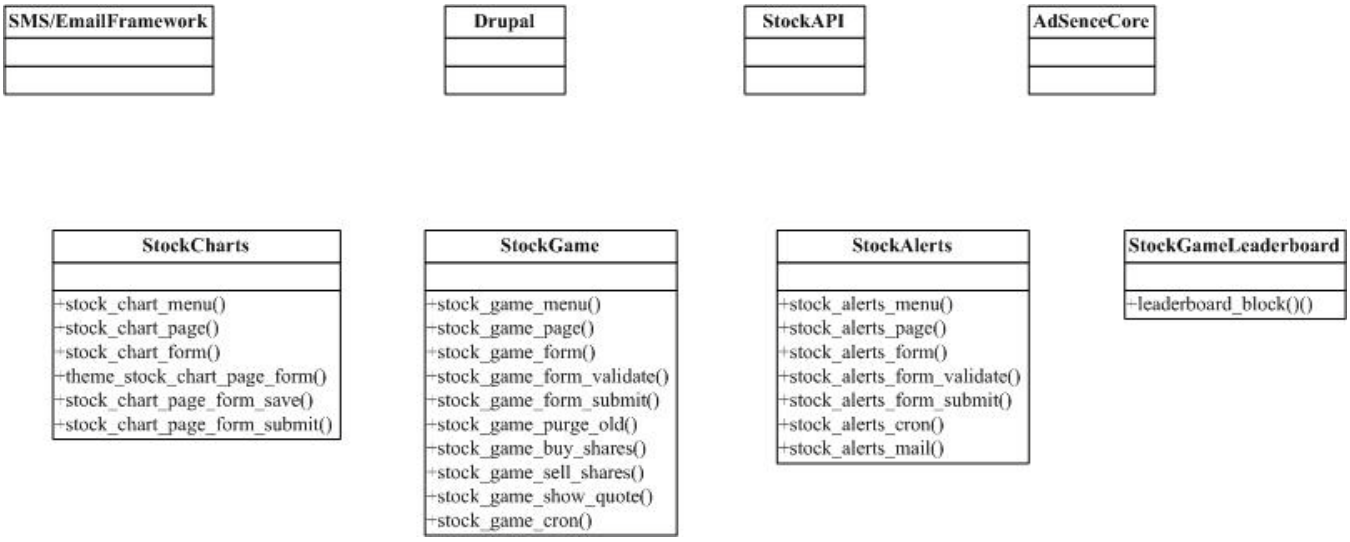
# 11) CLASS DIAGRAM AND INTERFACE SPECIFICATIONS

## a) Class Diagram




Note that the "classes" in this diagram are actually the programmed Drupal modules. The modules do not need to store private data as everything is inserted into the MySQL database, and all methods are public because there is no reason to hide any of the module functions from the Drupal core.

## b) Data Types and Operation Signatures




## c) Design Patterns

A design pattern is a common software engineering practice for reusing solution to problems in software design. Often, it can be a generic template that can be modified to solve a subset of problems. In this project, we apply a popular design pattern: Indirect Communication: Publisher-Subscriber

**Indirect Communication: Publisher-Subscriber**

Publisher-Subscriber pattern is a technique used to implement indirect communication where objects are not programmed to send messages to specific receivers, rather send messages to any subscribers of a certain interest or topic. There are three main components: publishers, subscribers, and events.

Publisher – Knows events sources, knows subscribers, registers/unregisters subscribers, notifies subscribers of events.

Subscribers – Knows events of interest, knows publisher, resisters/unregisters with publisher, processes received event notifications.

Events – The types of information that a subscriber want to be informed whenever it occurs.

**Application of Publisher-Subscriber**

One of the functionalities implemented in this project is automatic email and SMS notifications based on user threshold stock values. The software solution to this problem can be realized using the publisher-subscriber model. Following is the communication infrastructure highlighting the publishers and subscribers.

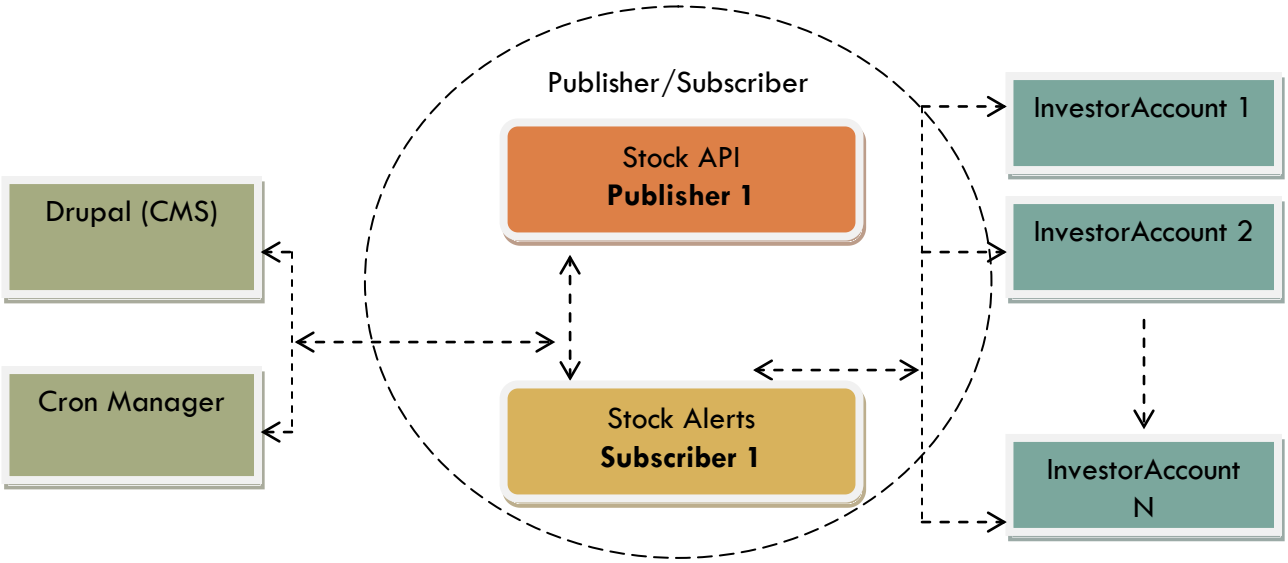



FIGURE 11-1 - COMMUNICATION INFRASTRUCTURE

The diagram above shows the communication implemented as a publish/subscribe model. The Stock Alerts module is associated with every investor account. Each investor once authenticated, can select his/her user preferences. Here, automatic notifications of change in stock prices can be selected by specifying upper/lower thresholds for a given stock. The stock alerts module can avoid polling the database for changes in stock values by subscribing the Stock API module for given thresholds. The Stock API module implements the stock prices updates retrieved from Yahoo Finance. Hence, the Stock Alerts is a subscriber to Stock API. When an event occurs (user threshold is met), a message is generated by Stock API and sent to the Stock Alerts. Hence, the Stock API is a publisher to the Stock Alerts. Stock API knows the events of interest (thresholds), and it subscribers (Stock Alerts). Finally, the Cron Manager is a crucial component responsible for scheduling the Stock API module to retrieve stock information. For example, every five minutes, Cron Manager runs the Stock API module to update out database.

### d) Object Constraint Language Contracts

Some of the contracts below describe Drupal's behavior in giving a PHP variable to the function and expecting it to be filled with data, be it a HTML table, page, form, or something else. The exception to this is the cron function, which is run by a timer and does not need to return anything.

**Leaderboard**

```
context Leaderboard inv:
     self.getNumOfTopUsers() >= 0

context Leaderboard:: leaderboard_block () : boolean pre:
     self.getLastUpdated() == timestamp_old

context Leaderboard:: leaderboard_block () : boolean post:
```

```
    self.getLastUpdated() = getLastUpdated() @pre + time_elapsed
```

### StockAlerts

```
context StockAlerts inv:
    drupal.menuVisible() == true

context StockAlerts::stock_alerts_menu() : boolean pre:
    drupal.containsMenu(stock_alerts) == false

context StockAlerts::stock_alerts_menu() : boolean post:
    drupal.containsMenu(stock_alerts) == true

context StockAlerts::stock_alerts_page() : boolean pre:
    drupal.containsPageList(stock_alerts) == false

context StockAlerts::stock_alerts_page() : boolean post:
    drupal.containsPageList(stock_alerts) == true

context StockAlerts::stock_alerts_form() : boolean pre:
    drupal.containsForm(stock_alerts) == false

context StockAlerts::stock_alerts_form() : boolean post:
    drupal.containsForm(stock_alerts) == true

context StockAlerts::stock_alerts_form_validate(id: FormId, state:
FormState) : boolean pre:
    form.select(id) > 0

context StockAlerts::stock_alerts_form_validate(id: FormId, state:
FormState) : boolean post:
    form.select(id).state.equals(“ready”) == 0

context StockAlerts::stock_alerts_form_submit(id: FormId, state:
FormState) : boolean pre:
    form.select(id).state.equals(“submitted”) != 0

context StockAlerts::stock_alerts_ form_submit (id: FormId, state:
FormState) : boolean post:
    form.select(id).state.equals(“submitted”) == 0


context StockAlerts::stock_alerts_form_cron() : boolean pre:
    drupal.cron.getCronTimeRemaining() > 0

context StockAlerts::stock_alerts_ form_cron () : boolean post:
    drupal.cron.resetCronTimer() > 0
```

```
context StockAlerts::stock_alerts_form_mail(k: Key, msg: Message, p:
Params) : boolean pre:
     drupal.mail.isReady() > 0

context StockAlerts::stock_alerts_ form_mail (k: Key, msg: Message,
p: Params) : boolean post:
     drupal.mail.isSent(k) > 0
```

### StockChart

```
context StockChart inv:
     self.isVisible() == true

context StockChart:: stock_chart_block () : boolean pre:
     self.getLastUpdated() == timestamp_old

context StockChart:: stock_chart_block () : boolean post:
     self.getLastUpdated() = getLastUpdated() @pre + time_elapsed


context StockChart:: stock_chart_page () : boolean pre:
     drupal.containsPageList(stock_chart) == false

context StockChart:: stock_chart_page () : boolean post:
     drupal.containsPageList(stock_chart) == true

context StockChart:: stock_chart_form () : boolean pre:
     drupal.containsForm(stock_alerts) == false

context StockChart:: stock_chart_form () : boolean post:
     drupal.containsForm(stock_chart) == true

context StockChart:: stock_chart_page () : boolean pre:
     drupal.containsPageList(stock_chart) == false

context StockChart:: stock_chart_page () : boolean post:
     drupal.containsPageList(stock_chart) == true

context StockChart:: stock_chart_page_form_save (id: FormId, state:
FormState) : boolean pre:
     drupal.form.select(id).state.equals(“saved”) !=0

context StockChart:: stock_chart_page_form_save (id: FormId, state:
FormState) : boolean post:
     drupal.form.select(id).state.equals(“saved”) ==0
```

```
context StockChart:: stock_chart_page_form_submit (id: FormId, state:
FormState) : boolean pre:
     drupal.form.select(id).state.equals(“ready”) !=0

context StockChart:: stock_chart_page_form_submit (id: FormId, state:
FormState) : boolean post:
     drupal.form.select(id).state.equals(“ready”) ==0
```

### StockGame

```
context StockGame inv:
     self.state.equals(“running”) == 0

context StockGame::stock_game_menu() : boolean pre:
     drupal.containsMenu(stock_game) == false

context StockGame::stock_game_menu() : boolean post:
     drupal.containsMenu(stock_game) == true

context StockGame::stock_game_page() : boolean pre:
     drupal.containsPageList(stock_game) == false

context StockGame::stock_game_page() : boolean post:
     drupal.containsPageList(stock_alerts) == true

context StockGame::stock_game_form() : boolean pre:
     drupal.containsForm(stock_alerts) == false

context StockGame::stock_game_form() : boolean post:
     drupal.containsForm(stock_alerts) == true

context StockGame::stock_game_form_validate(id: FormId, state:
FormState) : boolean pre:
          form.select(id) > 0

context StockGame::stock_game_form_validate(id: FormId, state:
FormState) : boolean post:
     form.select(id).state.equals(“ready”) == 0

context StockGame::stock_game_form_submit(id: FormId, state:
FormState) : boolean pre:
     form.select(id).state.equals(“submitted”) != 0

context StockGame::stock_game_ form_submit (id: FormId, state:
FormState) : boolean post:
     form.select(id).state.equals(“submitted”) == 0
```

```
context StockGame::stock_game_purge_old() : boolean pre:
     self.needsPurge() == true

context StockGame::stock_game_purge_old() : boolean post:
     self.needsPurge() == false

context StockGame::stock_game_buy_shares() : boolean pre:
     self.needsUpdateAccount() == true

context StockGame::stock_game_buy_shares() : boolean post:
     self.needsUpdateAccount() == false

context StockGame::stock_game_sell_shares() : boolean pre:
          self.needsUpdateAccount() == false

context StockGame::stock_game_sell_shares() : boolean post:
     self.needsUpdateAccount() == true

context StockGame::stock_game_show_quote() : boolean pre:
     self.needsUpdateDisplay() == true

context StockGame::stock_game_show_quote() : boolean post:
     self.needsUpdateDisplay() == false

context StockGame::stock_game_form_cron() : boolean pre:
     drupal.cron.getCronTimeRemaining() > 0

context StockGame::stock_game_ form_cron () : boolean post:
     drupal.cron.resetCronTimer() > 0
```

# 12) SYSTEM ARCHITECTURE AND DESIGN

## a) Architectural Styles

Software architecture refers to the composition of the system under discussion. Every system's configuration is unique due to the stakeholders' desires it supports. Each implementation varies the system's attributes, resulting in a (possible) different architecture. Some common software architectures are *Client-server*, *Event-Driven, Database Centric*, *Component Based*, and *Distributed Computing*.

The definitions of some of these common software styles are described here.

- The client-server software architecture model communicates between several systems over a network. A client may initiate a communication session, while the server waits for requests from any client (Wikipedia).
- Component based software engineering (CBSE) is defined as a type of software architecture style with emphasis on decomposition of the engineered systems into functional or logical components with well-defined interfaces used for communication across the components (Wikipedia).
- Event-driven architecture (EDA) is a software architecture pattern promoting the production, detection, consumption of, and reaction to events (Wikipedia).

Our *Stock Market Fantasy League* system is a cross breed of several software architectural styles. One may right away question this type of classification: Why several groupings? There are several reasons why we cannot categorize our system into a single software architectural style. Firstly, our system is a webpage. This representation places our system under the client-server representation. Secondly, our webpage comprises of several interconnected modules that communicate with each other, putting our system also in component based software architecture. Finally, our stock fantasy game is driven by events such as buying, trading, viewing, updating and possibly predicting. These events need immediate response from our system, thus, placing, yet again, our system under the event driven architecture.

In essence, the combination of all three architectures fits the style of programming and the tools that we are planning to employ. This type of solution is called the top down approach where the system is analyzed globally first and then broken down into several smaller components to facilitate testability, extensibility, and maintainability.

## b) Identifying Subsystems

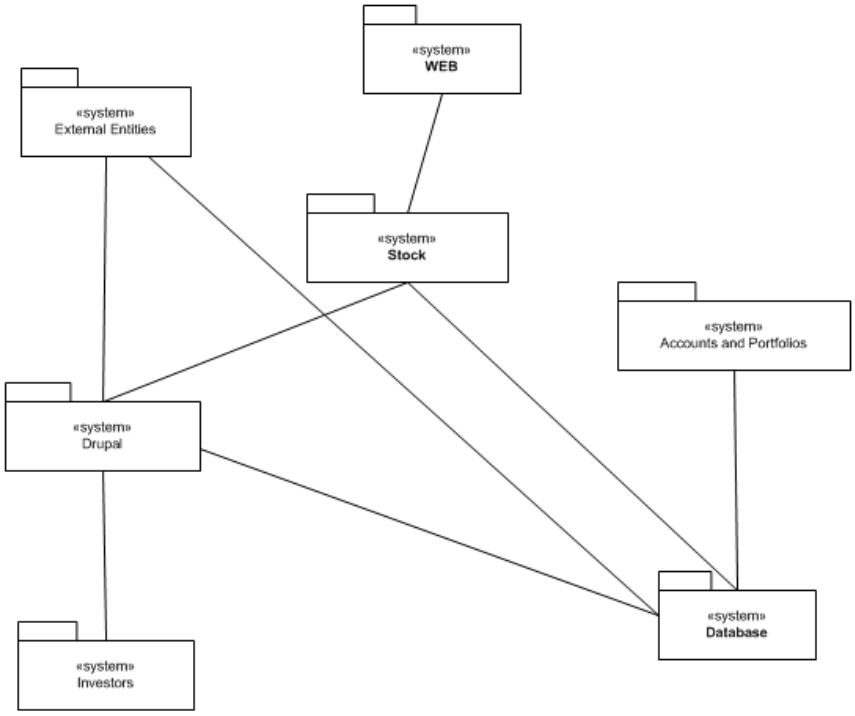



FIGURE 12-1 - OVERALL PACKAGE DIAGRAM FOR DRUPAL IMPLEMENTATION

We are using Drupal to create, maintain and run our game website. So, many of the key concepts are being absorbed by the Drupal modules (shown above in "Drupal"). These modules will automatically take care of the many of the boundary concepts and entities. Some of the external entities in our system are also shown, separating concepts inside the system.

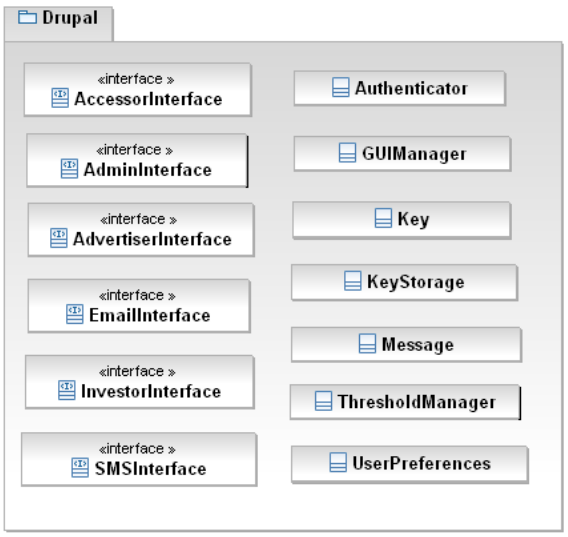



FIGURE 12-2 - INSIDE THE DRUPAL PACKAGE

## c) Mapping Subsystems to Hardware

Client/server model describes the relationship between two computer programs in which one program (the client), makes a service request from another program (the server), which fulfills the request. Although the client/server idea can be used by programs within a single computer, it is a more important idea in a network.

In a network, the client/server model provides a convenient way to interconnect programs that are distributed efficiently across different locations. Computer transactions using the client/server model are very common. Since our Stock Market Fantasy League is a web-based game, it definitely follows the web client/server model and needs to run on multiple computers (or subsystems).

The web browser (currently on our stock market website) is the client that requests services from our web server. However, this web server is a general name for a subsystem of its own. The web server needs to not only retrieve information from the stock market and investor databases, but also needs to communicate with the external web server to get frequent stock updates.

We are using Drupal to put into practice the client/server model and to maintain the stock market website. Drupal's internal modules will access the MySQL database and satisfy the client requests accordingly. In other words, our system can be invoked from multiple computers and the Drupal implementation will handle all of them accordingly.

## d) Persistent Data Storage

Our system needs to preserve important data beyond a single execution. So, we require both persistent data storage mechanisms as well as databases to hold critical information such as investor personal information, investor portfolios and stock information and history.

To incorporate such a facility, we will be storing all the data under a Relational Database Management System (RDBMS) called MySQL. A brief definition of an RDBMS is such that it is a database in which all the data and relations among them is stored in the form of tables. To find the necessary data, we provide a single or multiple Primary Key(s) that uniquely identifies the table the data is stored within. This simple mechanism allows us to find necessary data in a breeze, rather than searching billions of files sequentially. Therefore, we will be using relational tables as our persistent data storage mechanism.

The relational database schema that we are employing is shown here. The tables shown in the schema are only parts of the entire database that drupal holds. Although one cannot witness the entire database, one can guess the structure of the rest of the database using the schema shown. We have six important tables, namely "USERS," "STOCKAPI," "STOCK_GAME," "STOCK_GAME_HOLDINGS," "STOCK_ALERTS," and "STOCK_GAME_ORDERS".

The USERS table contains account information of all investors and advertisers. The STOCKAPI table comprises of all the stocks available in the game. STOCK_GAME table and

STOCK_GAME_HOLDINGS table hold data within an investor's portfolio. STOCK_ALERTS table deals with the maintenance of threshold levels that automate buying and selling. Finally, STOCK_GAME_ORDERS table maintains the transaction queue, which both serves as a shopping cart and as a transaction history lookup. Needless to say, some tables are uniquely identified by a primary key and this is how these tables will be accessed by the system.

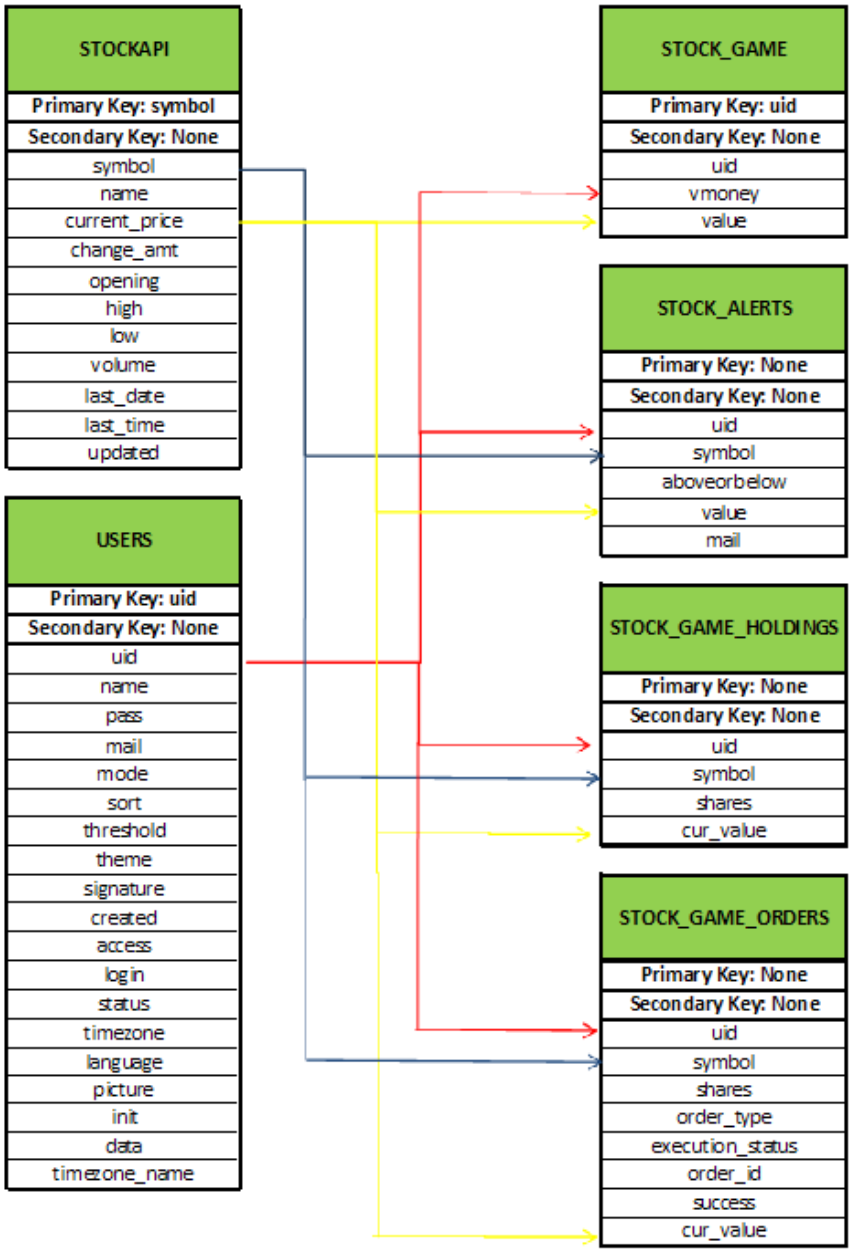



FIGURE 12-3 - DATABASE SCHEMA DIAGRAM

## e) Network Protocol

The system that is being developed will use standard TCP/IP sockets to communicate data over Hyper Text Transfer Protocol (HTTP) from a server to multiple clients, as well as from an external server to this server. The File Transfer Protocol (FTP), in conjunction with SSH (Secure Shell), may be used for administrative purposes.

The system's use of HTTP for communication is well justified as the standard for Internet and web-based applications is HTTP. The protocol provides a simple method to request and send data, which is exactly what is needed.

A brief description of HTTP:

Request Message Format:

A HTTP request is sent from the client to the server, usually requesting a webpage to be delivered. The format is COMMAND PATHNAME HTTP/VERSION.

Sample Request:

```
GET /index.html HTTP/1.1
Host: www.example.com
...
```

This sample request shows a client that wants to download a file index.html from the server. There is a lot of other information given, but the most important line is the first one above. The most used commands in HTTP are GET and POST, which request a file from the server and send information to the server (ex. webform) respectively.

Sample Server Response Header:

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix)  (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

The server response to the client's request is also quite simple. The required header line is HTTP/VERSION 200 OK, which signals to the client that the request was successful and a file is being returned. The rest of the response will be the actual file. If it is a file type that a browser can recognize (like HTML), the browser will not prompt the user to download the file and will

instead render the contents appropriately on-screen. The remainder of the server response is extra information that the server can tell the client, but is not required.

## f) Global Control Flow

The system is an event-driven one, waiting for any user input and acting accordingly. Multiple users can login and act on their portfolios at will, all at the same time. There are timers in the system, including those internal to Drupal. Drupal has the Linux command "cron" perform most of its scheduled activities, initiating module updates and various other features. The timer for the stock information and updates can be integrated into Drupal's list, or done on its own.

The system aims to be real-time, but due to hardware and time constraints, it is not actually possible. What will be done is a periodic update of information, with a period as short as possible. The stock information database will retrieve updates from the financial data source every 20 minutes or less. The large period is due to the large amount of information to be transferred and the limited resources the server will be running on. The leader board can be updated every 2-3 minutes, and other statistics can be updated with a similar period.

The Apache web server does use multiple threads to handle concurrent user requests. Exactly how it works and how the threads are synchronized are out of the scope of this report. However, this functionality extends to the applications served, including PHP and thus Drupal and the developed modules.

## g) Hardware Requirements

The server is required to run Apache with PHP, which by itself can run on a 486 with 8MB of RAM or so. Assuming that the system will be run on a machine with fairly decent specifications, the website's experience should be well presented.

Recommended Hardware Specifications for the Server:

Processor: Intel Pentium III 1GHz or equivalent

Memory: 512MB of memory or greater

100MB Available Hard Drive Space

Internet Connection/Networking Capability (>10KBps Upload, >100KBps Download)

These requirements are not minimal, but are recommended for the best server experience.

Clients may access the site through any web-enabled device, though the best user experience can be achieved using a computer with a display resolution of 1024x768 or greater.

# 13) ALGORITHMS AND DATA STRUCTURES

## a) Algorithms

Value-at-Risk

Value-At-Risk (VaR, or VAR) is a technique used to estimate the probability of portfolio losses based on the statistical analysis of historical price trends and volatilities. VaR is commonly used by banks, security firms, and companies that trade commodities. VaR is used to measure risk while it happens and is an important consideration when firms make trading decisions. Most often VaR is an indicator of the "risk of loss" on a specific portfolio of financial assets.

Calculation of VaR: using the variance covariance method

1.) Assume normal distribution for stock returns

2.) Obtain estimate of Expected Return (average return) and Standard Deviation

3.) Find worst 5% on the normal curve (for 95% confidence, number of standard deviations is -1.65σ)

Sharpe-Ratio

The Sharpe ratio tells us whether a portfolio's returns are due to smart investment decisions or a result of excess risk. This measurement is very useful because although one portfolio or fund can reap higher returns than its peers, it is only a good investment if those higher returns do not come with too much additional risk. The Sharpe ratio has as its principle advantage that it is directly computable from any observed series of returns without need for additional information surrounding the source of profitability.

Calculation of Sharpe-Ratio for a give portfolio using the Single Index Model (SIM)

1.) Find portfolio variance

$$\beta_i = \frac{COV(R_i, R_m)}{\sigma_m^2}$$

$$\beta_p = \sum_{i=1}^{n} x_i \beta_i$$

$$\sigma_p^2 \approx \beta_p^2 \sigma_m^2$$

Where: $\beta_i$ = beta of stock i

$\beta_p$ = beta of the portfolio

$R_i$ = return on stock i

$R_m$ = return on benchmark

$\sigma_m$ = standard deviation of returns on benchmark

$\sigma_p$ = portfolio standard deviation

$x_i$ = portion of the portfolio invested in stock i

2.) Find the expected (or average) return of the given portfolio

$$r_p = E(R_p) = \sum_{i=1}^{n} [x_i E(R_i)]$$

$$\sum_{i=1}^{n} x_i = 1$$

Where: $R_p$ = return on the portfolio

$R_i$ = return on investment i

$x_i$ = portion of the portfolio invested in stock i

3.) Find the Sharpe-Ratio as follows

$$S = \frac{r_p - r_f}{\sigma_p}$$

Where: $S$ = the Sharpe-Ratio

$r_p = R_m$ = expected portfolio return

$r_f = R_m$ = risk free rate

$\sigma_p$ = portfolio standard deviation

Searching Database

We will be utilizing the binary search algorithm when storing to and retrieving from the database. Because we will be storing multiple tables in the database system to emulate multiple databases, we can guarantee the proper sorting of the tables. We do not enumerate the specifics of the binary search algorithm as this searching technique is well known. For details on the binary search algorithm refer to [1].

## b) Data Structures

The system being developed will use a database to store the bulk of its data. The MySQL database will contain tables holding the information being stored/retrieved. There are no plans to use other structures at this time.

# 14) USER INTERFACE DESIGN AND IMPLEMENTATION

The goal of the user interface is to make the site look good and easy to use. This can be achieved using relatively simple web pages and forms to display and capture data, with a minimal amount of clicks/keystrokes. The typical user would point his/her browser to the game's website, log in, and start playing. No special equipment is needed, just a computer with an internet connection.

## a) Preliminary Design

The terms used below are not exact matches for those found on the site, as this is the preliminary design phase.

Sample Navigational Paths (all cases assume user is already on website's front page):

Start -> Log In -> View Portfolio -> Logout

Start -> Log In -> Manage Portfolio -> Buy Stock -> Logout

Start -> Log In -> Manage Portfolio -> Sell Stock -> Logout

Start -> Log In -> My Account -> Edit Notifications -> Configure -> Logout

Start -> Log In -> My Account -> Disable -> Logout

Start -> Log In -> Edit Advertisements -> Logout (Advertisers Only)

Start -> Log In -> My Account -> View history -> Logout

Start -> Log In -> View Recommendations -> Logout

Start -> Log In -> View Recommendations -> Manage Portfolio -> Buy/Sell Stock -> Logout

Start -> View Leaderboard

Start -> View New Users

...etc.

## b) User Effort Estimation

All of the following GOMS (Goals, Operators, Methods, and Selection Rules) Methods are done with older knowledge of the site - it has changed since implementation, but not by much. These methods are modeled loosely after the ones found on the page about GOMS [16].

NOTE: Keystrokes for data entry are omitted because they are variable. The minimal keystrokes/clicks for a given method ignoring data entry are described below. It is evident that most of the work the user has to do is data entry related, with minimal UI overhead.

Method for Goal: Log In/Authenticate - requires at least 2 clicks and 1 keystroke

1. Move cursor to Username text field and click mouse (context)
2. Enter username into text field using keyboard
3. Press "TAB" to move to password field
4. Enter password into text field using keyboard
5. Click "Log In" button
6. Return with goal accomplished

Method for Goal: Log Out of Account - 1 mouse click

1. Move cursor to "Log Out" link on page (context)
2. Click mouse button
3. Return with goal accomplished

Method for Goal: Create New Account - requires at least 3 clicks and 1 keystroke

1. Move cursor to "Create New Account" link on page and click mouse (context)
2. Click on the first input field
3. Enter desired information
4. Press "TAB" to go to next field
5. Repeat steps 4 and 5 until form is complete
6. Submit form by clicking "Submit" button
7. Return with goal accomplished

Method for Goal: Add to/Purchase stocks from market for a portfolio - requires at least 4 clicks and 1 keystroke

1. Accomplish goal: Log In/Authenticate
2. Move to and click on "Manage Portfolio"
3. Move to and click on "Buy Stocks"
4. Move to and click on stock ticker text field
5. Enter stock name/ticker symbol
6. Press "TAB" to move to next field
7. Repeat 5 and 6 for all required fields (likely ticker symbol, number of shares/$ amount, when to purchase)

8. Click "Submit"
9. Return with goal accomplished

Method for Goal: Remove/Sell stock to market from a portfolio - requires at least 5 mouse clicks

1. Accomplish goal: Log In/Authenticate
2. Move to and click on "Manage Portfolio"
3. Move to and click on "Sell Stocks"
4. Select desired stocks using checkboxes and mouse
5. Move to and click on "Confirm Sale"
6. Read page information, confirm as correct, then move and click "Submit" button
7. Return with goal accomplished

Method for Goal: Setup a New E-Mail/SMS Notification - requires at least 5 mouse clicks

1. Accomplish goal: Log In/Authenticate
2. Move to and click on "My Account"
3. Move to and click on "Edit"/"Manage Profile"
4. Move to and click on "Notifications"
5. Recall desired stock ticker symbol
6. Enter ticker symbol into text field
7. Move to and click on "E-Mail" and/or "SMS"
8. Move to and click on "Submit"
9. Return with goal accomplished

As stated previously, the goal of a user interface is to make the site look good and easy to use. The screen mock-ups from report one are indeed followed here, for the most part. As the reader should know, the system will be implemented using a prepackaged system called Drupal, enabling the use of tested and standardized layouts and themes. Doing so also allows more of the programming focus to be on the stock market algorithms, graph generation, market updates, etc. instead of the standard user interface and authentication code. Careful consideration still needs to be given to placement of objects, so the job is made slightly easier but is not automatically finished; Drupal does NOT allow for ignoring user interface design.

Drupal combines CSS (Cascading Style Sheets), PHP, HTML, and many other technologies to present the user with a clean, modular, and extensible interface that is way beyond what can be written by hand in one semester. As such, the interface components have been implemented and is already complete. The changes to the interface that will be made in the future relates to the layout of blocks (login, calendar, etc.) and theme (colors, font styles) used. One major feature to be added later is the advertisements on each page.

One subtle feature than can potentially reduce user effort is the ability to use "clean URLs" with Drupal. This means a standard URL - http://www.example.com/home/index.php?x=5&y=5&dest=user can be compacted into http://www.example.com/home. If the user wants to jump to a page quickly, he/she can just type the name of it in without using a long and convoluted URL.

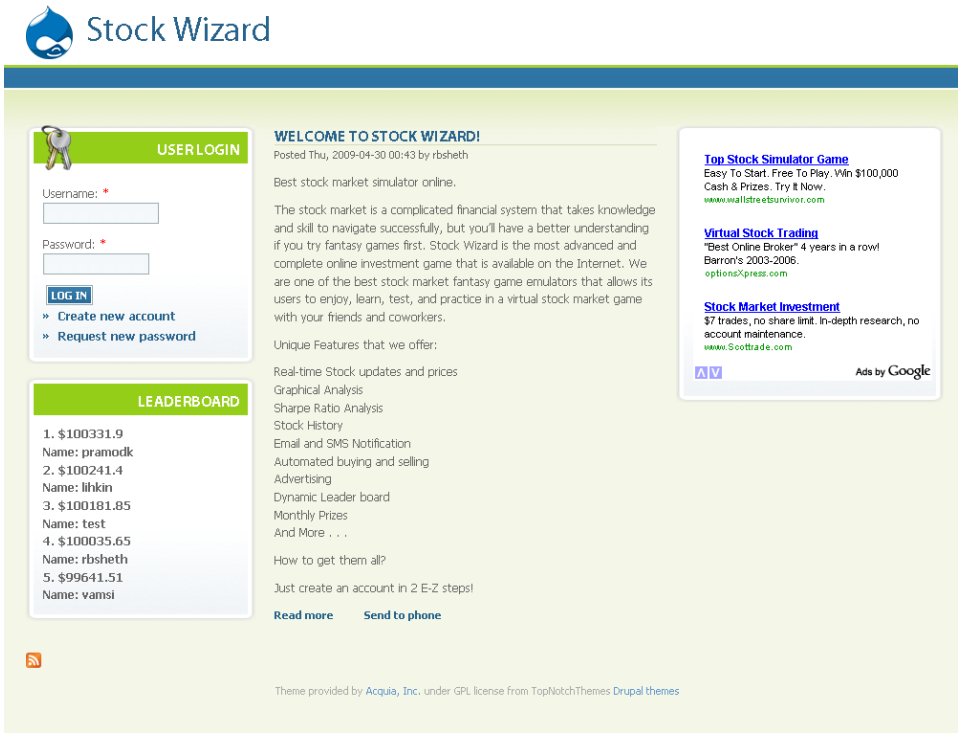



FIGURE 14-1 - FRONT PAGE OF THE WEBSITE

The figure above is the current front page of the website. The Drupal login block is shown on the left, and the leader board block is shown under it. The site announcements are shown in the

middle, but these may be moved to a "developer's blog" area in the future. A Google AdSense block is shown on the right. As is apparent, the interface is quite simplistic and most usage paths are hidden from an anonymous user. There are three ways to proceed on our site from this page - login using the username and password boxes on the left hand side, click on create a new account, and click on request a new password. The simple set of options is sufficient for the average user who must login to access their portfolio, any new user who wishes to register, and the occasional user who forgot his/her password.

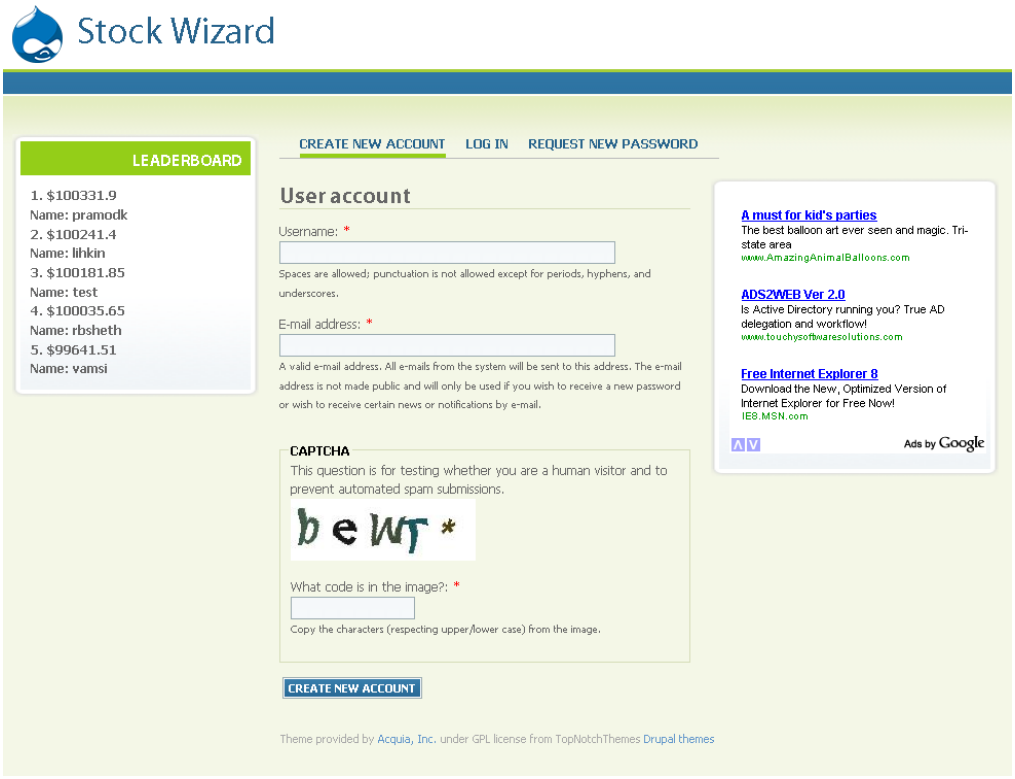



FIGURE 14-2 - REGISTER NEW ACCOUNT WEB PAGE

This figure shows the page displayed to a new user wishing to log in. There is a desired username field, e-mail address, and a CAPTCHA for stopping automated account creation. A password is then shown to the new user, who can then login.

The request new password option allows users to enter their e-mail address and receive a newly reset password for their account through their e-mail.

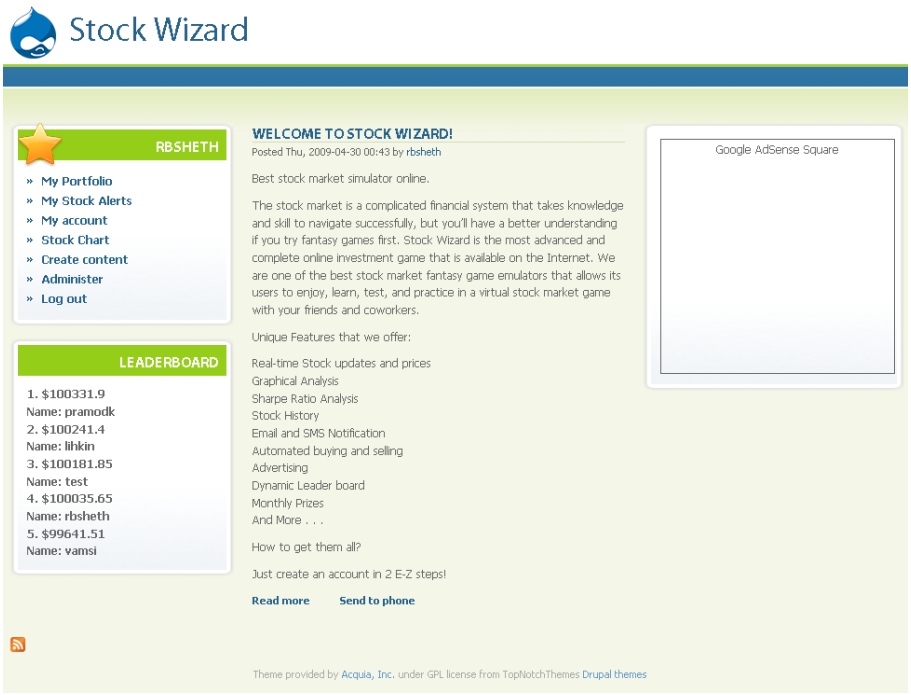



FIGURE 14-3 - POST-AUTHENTICATION WEB PAGE

After authentication, the user is presented with the page he/she was on before logging in - in this case, the front page. The account options are shown on the left. There are options to look at the user's account, check thresholds, see stock charts, and some administrative features (for administrators only).

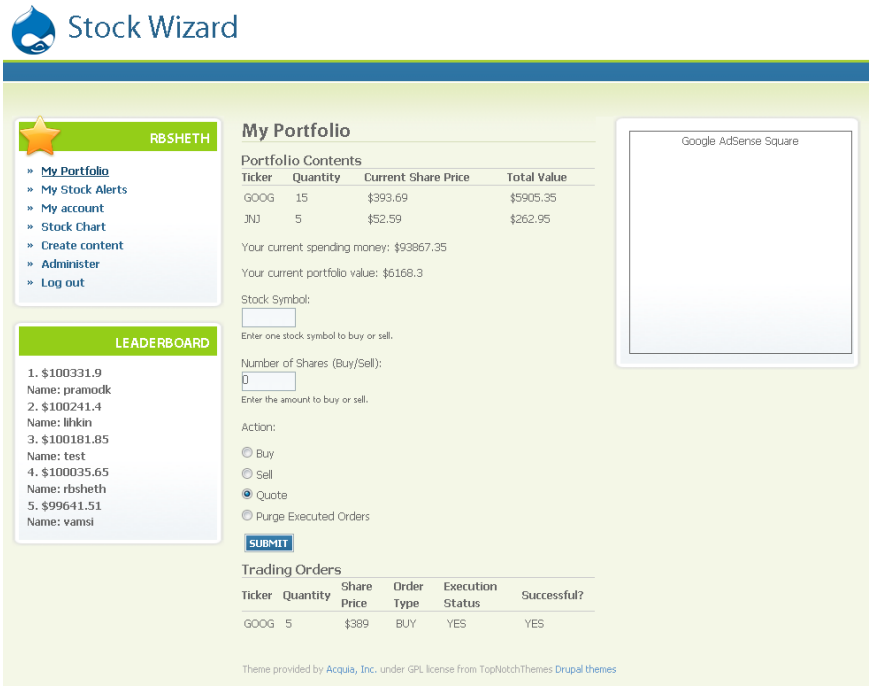



FIGURE 14-4 - MY PORTFOLIO WEB PAGE

This is a portfolio page that the user can buy and sell stocks from. One can also get stock quotes based on ticker symbol from here. An order that is placed is shown in the Trading Orders table and updated upon execution. Stocks in the user's portfolio are shown on the top table.

As can be seen in each figure, user effort has not been increased - the user interface follows the guidelines stated previously regarding user effort. Logging out, buying, and selling are very easy to do, requiring one click to perform the actions. For game players, the interface is very intuitive and follows good design rules.

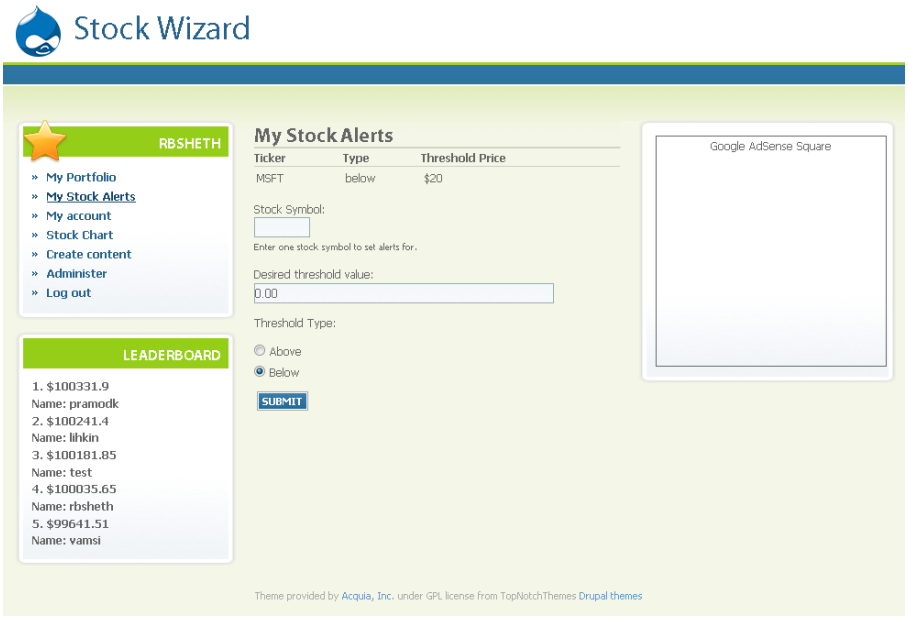



FIGURE 7-5 - STOCK ALERTS WEB PAGE

This is the stock alerts page as displayed to a user. It is very simple and easy to use - enter a ticker symbol, a price, and select if a notification is to be sent out when the stock is above or below the threshold.

As shown below, the stock charts page has many options in a cluttered interface. This is due to our basing of the code on existing code, having many options. It can be improved by reducing the number of options and cleaning up the chart display area.

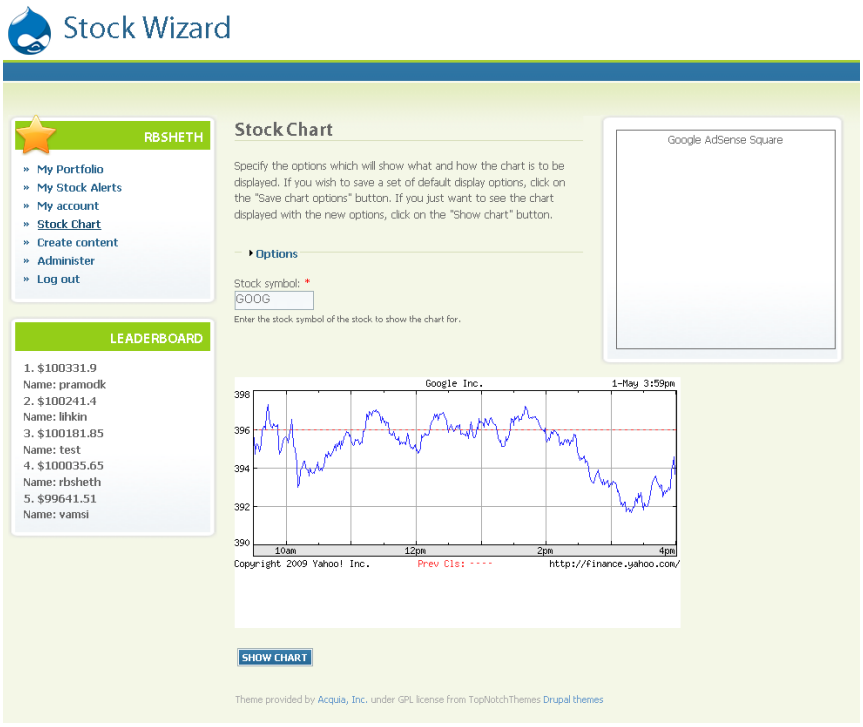



FIGURE 7-6 - STOCK CHART PAGE

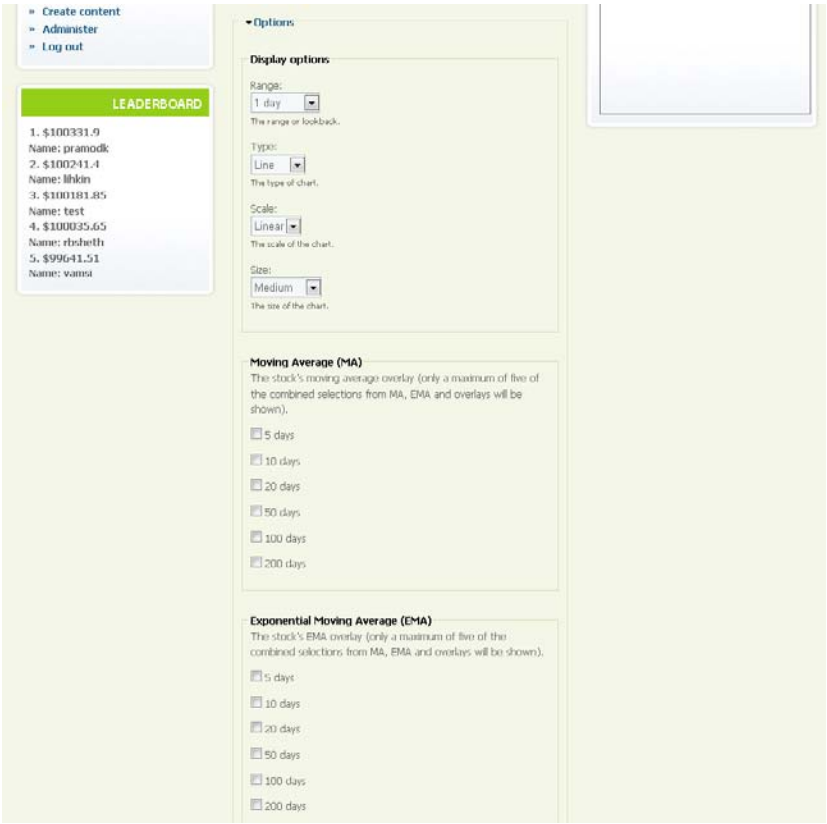



FIGURE 7-7 - SOME OF THE STOCK CHART OPTIONS

The only area where user effort is greatly increased is the administrative area. Drupal has a myriad of configuration options and entry points (e.g. text setting files on disk, interface based check boxes and radio buttons, custom code). Any administrator of the site must become familiar with the number of options Drupal provides for administering the site. This is not too much of a problem because it deals with a very select amount of "users" - most people using the site do not need to read much to start playing.

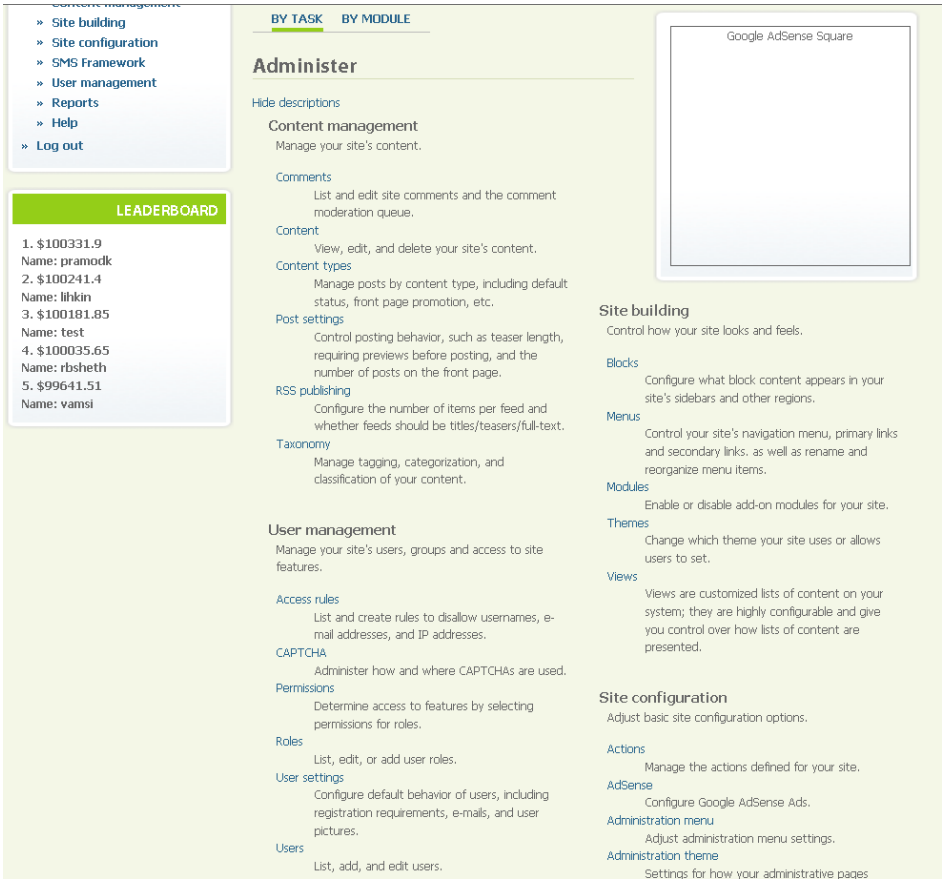
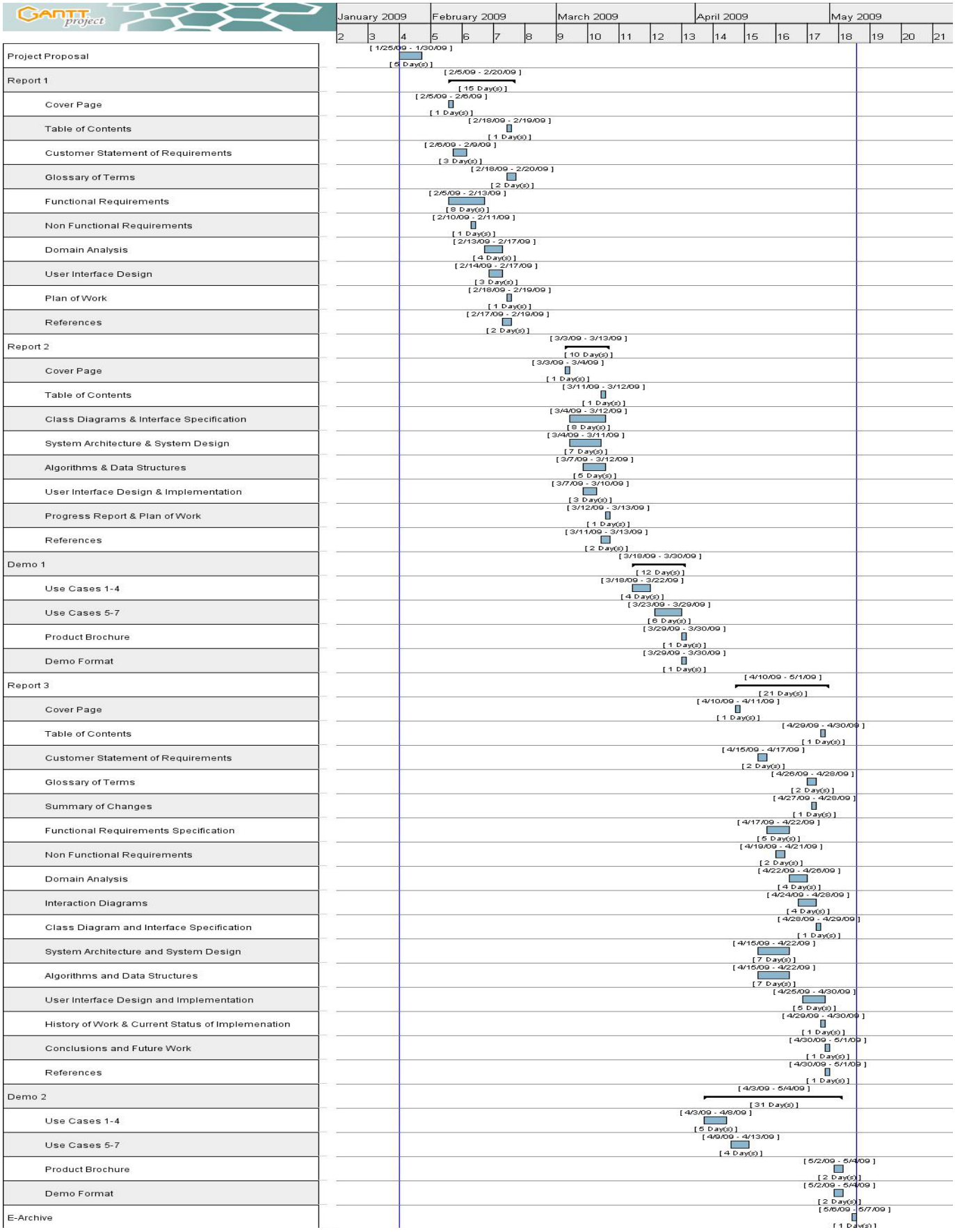
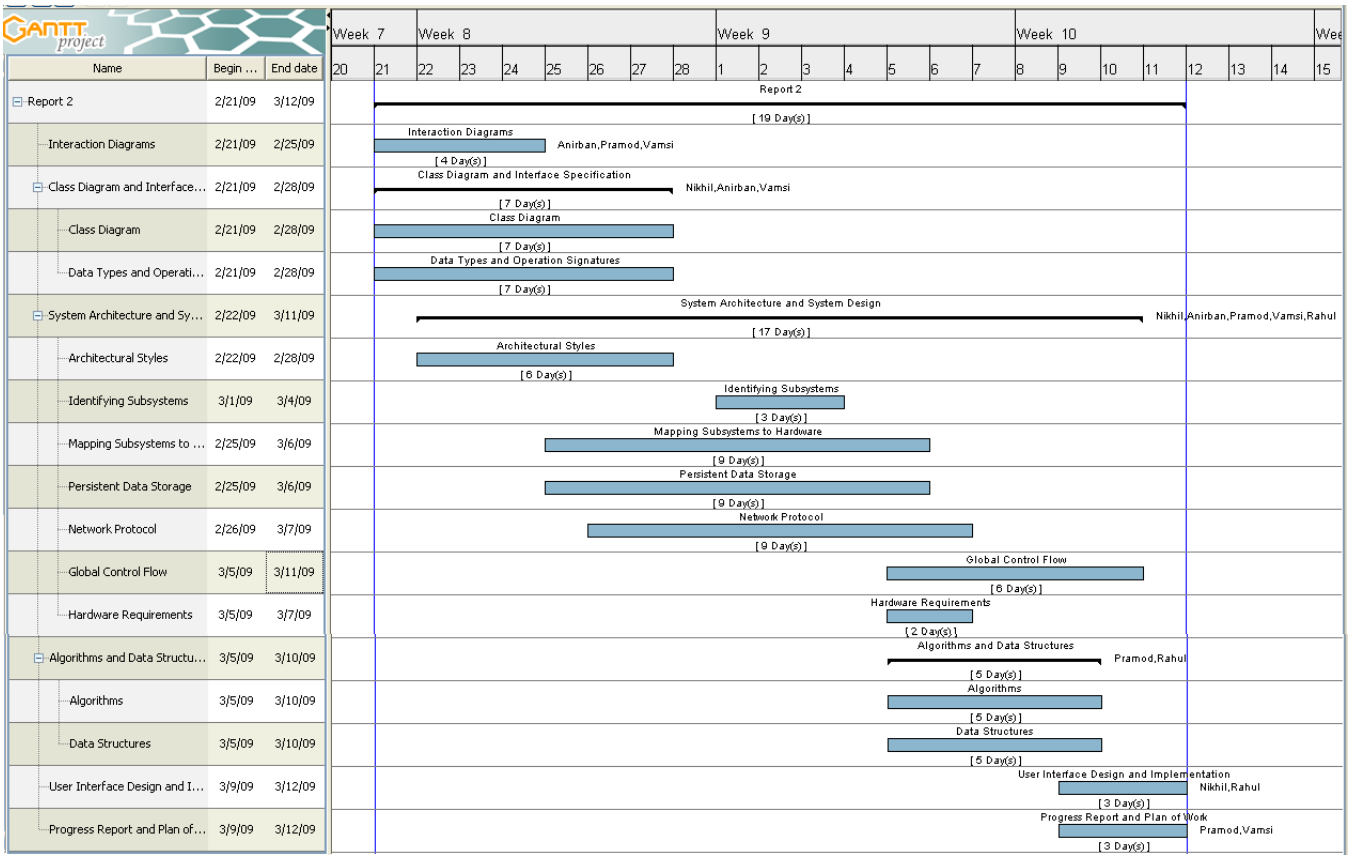


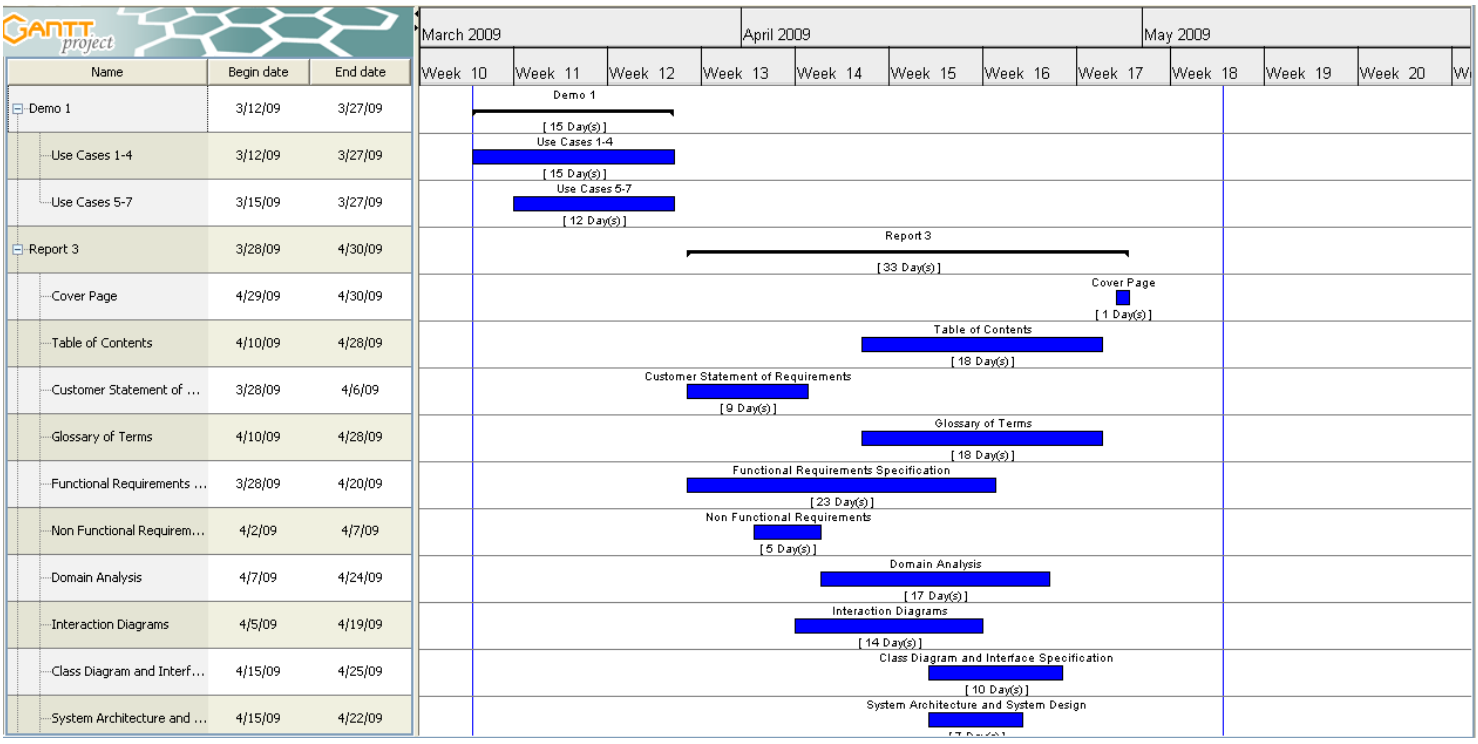

FIGURE 7-8 - A PORTION OF ONE ADMINISTRATIVE PAGE

# 15) HISTORY OF WORK AND CURRENT STATUS OF IMPLEMENTATION

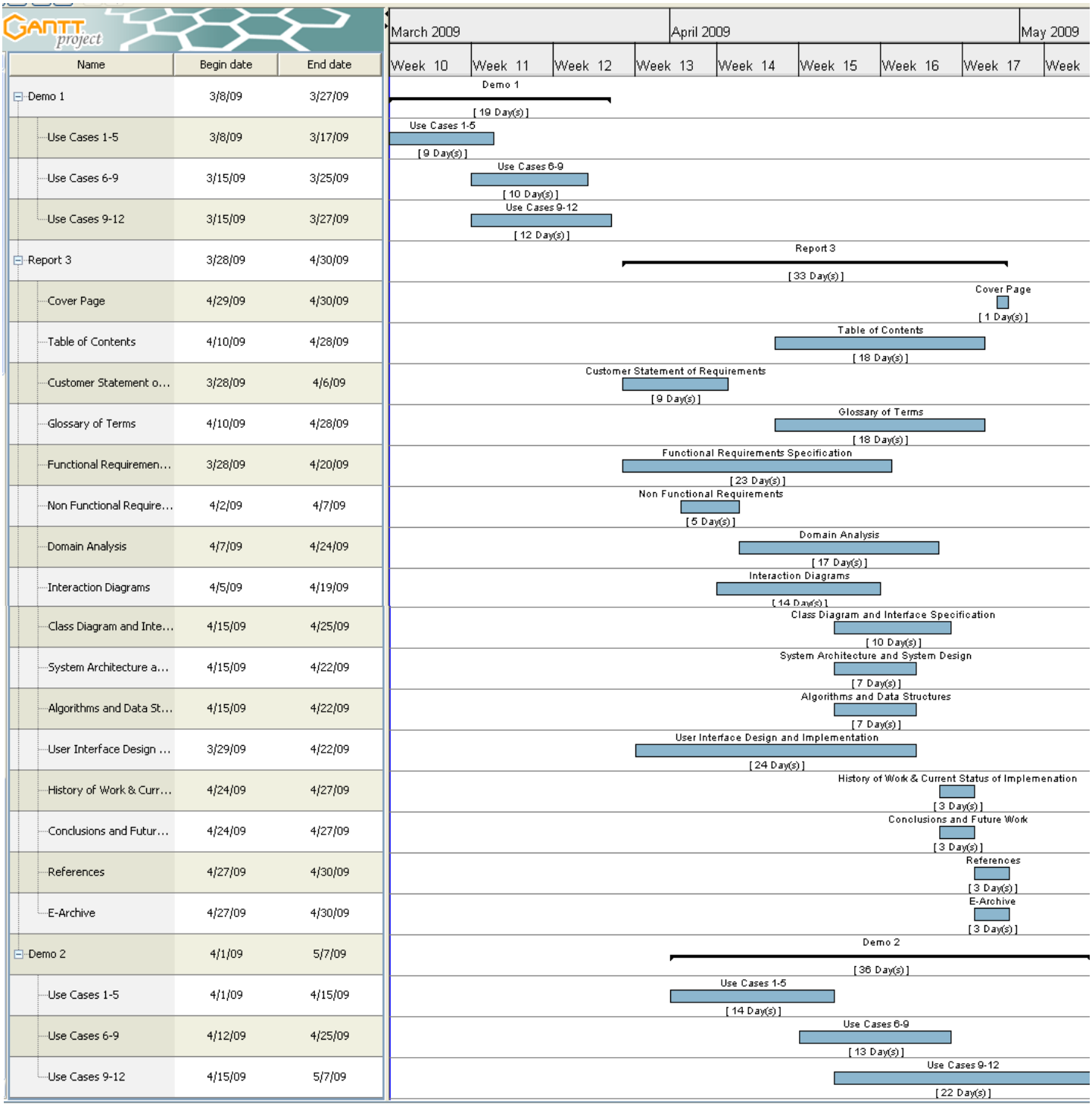


## From Report 1



From Report 2




| Name | Begin date | End date |
|---|---|---|
| Demo 1 | 3/8/09 | 3/27/09 |
| Use Cases 1-5 | 3/8/09 | 3/17/09 |
| Use Cases 6-9 | 3/15/09 | 3/25/09 |
| Use Cases 9-12 | 3/15/09 | 3/27/09 |
| Report 3 | 3/28/09 | 4/30/09 |
| Cover Page | 4/29/09 | 4/30/09 |
| Table of Contents | 4/10/09 | 4/28/09 |
| Customer Statement o... | 3/28/09 | 4/6/09 |
| Glossary of Terms | 4/10/09 | 4/28/09 |
| Functional Requiremen... | 3/28/09 | 4/20/09 |
| Non Functional Require... | 4/2/09 | 4/7/09 |
| Domain Analysis | 4/7/09 | 4/24/09 |
| Interaction Diagrams | 4/5/09 | 4/19/09 |
| Class Diagram and Inte... | 4/15/09 | 4/25/09 |
| System Architecture a... | 4/15/09 | 4/22/09 |
| Algorithms and Data St... | 4/15/09 | 4/22/09 |
| User Interface Design ... | 3/29/09 | 4/22/09 |
| History of Work & Curr... | 4/24/09 | 4/27/09 |
| Conclusions and Futur... | 4/24/09 | 4/27/09 |
| References | 4/27/09 | 4/30/09 |
| E-Archive | 4/27/09 | 4/30/09 |
| Demo 2 | 4/1/09 | 5/7/09 |
| Use Cases 1-5 | 4/1/09 | 4/15/09 |
| Use Cases 6-9 | 4/12/09 | 4/25/09 |
| Use Cases 9-12 | 4/15/09 | 5/7/09 |

As one can clearly see from the figures above, the plan of work from the previous two reports and the actual work conducted are not similar. The discrepancy, which is not unusual, is because of several factors such as time constraints, technical challenges, non-uniform experience level among the team members, and non-mutual contribution time among the team members. Nonetheless, like a professional corporate team, all project deliverables were completed within the prescribed deadlines and the quality of work was widely appreciated, with some degree of constructive criticism.

Below we present a list of the key accomplishments of this entire project:

- **Account Creation:** The account creation process involves allocating space for the new investor in the MySQL database with certain pre-defined conditions.
- **Obtaining Stock Information and Charts:** The web-based system accepts a valid company ticker symbol and retrieves the corresponding stock price from Yahoo Finance. Other information regarding the stock (charts) can also be obtained from Yahoo Finance if desired.
- **Stock Trade:** An investor can easily buy and sell shares of compnay stocks, and his or her portfolio gets modified accordingly. All information about an investor is stored in the MySQL database.
- **Updates:** All stock data in investor portfolios get updated through Yahoo Finance at a periodic time interval of fifteen minutes.
- **SMS/Email Notification:** An investor can send emails and SMSes through the web system. This feature subsequently allows the idea of automatic email/SMS notifications to investors at desired circumstances.
- **Activating Threshold:** An investor can set up threshold values on a certain stock, which, once crossed, prompts the system to send an email or SMS notification to the investor. This features allows an investor to be aware of the ever-changing environment of the game even without a computer.

# 16) CONCLUSIONS AND FUTURE WORK

Learning Drupal, PHP, and MySQL for this project was quite demanding and very interesting. The main challenges arose in the implementation of modules for Drupal, as it has a different type of system and ideology (write hooks in a module instead of functions in a class) for development. Working with Linux, the cron function, and the Apache web server were also good learning experiences. One notable challenge we had to face for the second iteration was to get the stock data constantly updated, as well as our portfolios, the leaderboard, and the stock alerts page.

Using cron to "activate" Drupal's timer system and our modules was hard to learn about and implement. Another challenge was the "white screen of death" (something Drupal is notorious for), which meant Drupal crashed and couldn't even display an interface to help fix the problem. This resulted in several reinstalls of the website, which is typical for a Drupal development site but very inconvenient.

The techniques we learned in this class helped us to split requirements into appropriate classes/modules, easily divide the work, and then use many individual building blocks to create an impressive final product. We were also allowed to use some preexisting code to provide a nice interface and authentication while meeting the stated requirements, which displays the flexibility of software design and the importance of code quality for reuse. Learning UML from books and websites is clearly not enough - it must be applied to a software project to even begin to see what it really is, and that was one of the purposes of doing this project. It was also clear that the standards for software design in place now are always changing and being improved upon by many people.

It would have helped to know a little more beforehand about the trading environment we were modeling, as well as having a class that teaches the basics of PHP, MySQL, Apache, and maybe even Drupal. Having a smaller scale project may also make concepts clearer, instead of trying to do everything and possibly being too general. More time would also have helped.

Due to time constraints, we were unable to implement several other interesting features for our application. The future work on this project would, henceforth, revolve around the implementation of the missing features. A brief description of each of these features is present below:

**Recommendation:** We plan on developing a feature through which our web application will examine current stock holdings of an individual investor, and provide recommendations to him or her regarding future stock-trade decisions. Based on the past behavior of the stock in question, our system would provide the investor with two options: Either more investment into the company, or relinquishment of the current shares. The degree of relinquishment would depend on the past behavior of the stock and the mathematical model used for the feature. The model would be based on one of the various forms of mathematical fuzzy logic.

**Tutorials:** Keeping in mind that majority of our customers would be amateur or potential stock brokers, we intend to provide them with on-line tutorials to better understand the game and the

general stock market. The tutorial would contain an in-depth description of our game, and concise information about the U.S. Stock Market and its functionalities. We believe the tutorial would play a significant role in alleviating customers' trouble in understanding the intricacies of the game and the general stock market.

**Predictions:** Similar to recommendation, we plan on developing a tool through which our customers can assess the risks and benefits of investing in a particular company. Based on past behavior and mathematical model, this feature will provide predictions about the stock in question. The mathematical model would be based on one of the various forms of mathematical fuzzy logic.

**Multiple game environments:** The current implementation of the game only allows one game session to run at a time. Customers joining our web system directly become part of the on-going game. We plan on implementing the idea where several game sessions can run simultaneously. Customers will then have the options of joining an on-going game or creating a new game. This feature will put our system at par with all other fantasy game systems available.

**International Markets:** In this age of rapid globalization, it would be of utmost importance to look beyond the borders of the country. We plan on incorporating several key international stock markets into our gaming system. Our customers would, therefore, get exposed to the stock-trading conducted outside of the United States. The experience and knowledge gained can consequently put our customers at an advantage in the real-world stock business. Apart from time zone differences and currency conversions, all other functionalities of the game will remain intact.

**Stock Trade among investors:** The current implementation of the game only allows our customers to trade with the market. We plan of adding the feature where customers can trade stocks among themselves, rather than just the market. Such a feature would enhance the dynamics of the game, and further consolidate our purpose for the entire project.

# 17) REFERENCES

Following is a list of references that have been used for any one of the parts of this report.


1. Binary Search
   "Binary search algorithm." Wikipedia, the free encyclopedia. 13 Mar. 2009 <http://en.wikipedia.org/wiki/Binary_search>.

2. Component-based Software Engineering
   "Component-based software engineering." Wikipedia, the free encyclopedia. 07 Mar. 2009 <http://en.wikipedia.org/wiki/Software_componentry>.

3. Client-Server
   "Client-server." Wikipedia, the free encyclopedia. 07 Mar. 2009 <http://en.wikipedia.org/wiki/Client-server>.

4. Event-driven Architecture
   "Event-driven architecture." Wikipedia, the free encyclopedia. 07 Mar. 2009 <http://en.wikipedia.org/wiki/Event_Driven_Architecture>.

5. Hypertext Transfer Protocol
   "Hypertext Transfer Protocol." Wikipedia, the free encyclopedia. 12 Mar. 2009 <http://en.wikipedia.org/wiki/HTTP>.

6. Introduction to Value at Risk (VAR)
   "Introduction to Value at Risk (VAR) - Part 1." Welcome to Investopedia.com - Your Source for Investing Education. 13 Mar. 2009 <http://www.investopedia.com/articles/04/092904.asp>.

7. Mathematics of Diversification
   "The Mathematics of Diversification." Accounting - 21e. 13 Mar. 2009 <http://www.swlearning.com/finance/strong/portfolio3e/powerpoint/ch06.ppt>.

8. MySQL
   "MySQL." Wikipedia, the free encyclopedia. 13 Mar. 2009 <http://en.wikipedia.org/wiki/MySQL>.

9. Pilone, Dan, and Neil Pitman. UML 2.0 in a Nutshell (In a Nutshell (O'Reilly)). North Mankato: O'Reilly Media, Inc., 2005.

10. Relational Database Management System
    "Relational database management system." Wikipedia, the free encyclopedia. 13 Mar. 2009 <http://en.wikipedia.org/wiki/RDBMS>.

11. Top Reasons to Use MySQL
"MySQL :: Top Reasons to Use MySQL." MySQL :: The world's most popular open source database. 13 Mar. 2009 <http://www.mysql.com/why-mysql/topreasons.html>.

12. Value at Risk (VaR)
"Value at Risk (VaR)." Welcome to Investopedia.com - Your Source for Investing Education. 13 Mar. 2009 <http://www.investopedia.com/terms/v/var.asp>.

13. Value at Risk
"Value at risk." Wikipedia, the free encyclopedia. 13 Mar. 2009 <http://en.wikipedia.org/wiki/Value_at_Risk>.

14. What is client/server?
"What is client/server? - a definition from Whatis.com." Network Management: Covering today's Network topics. 13 Mar. 2009 <http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci211796,00.html>.

15. FURPS

"FURPS." Wikipedia, the free encyclopedia. 10 Feb. 2009 <http://en.wikipedia.org/wiki/Furps>.

16. GOMS

"Tutorial GOMS (Kieras)." Home — College of Computing. 15 Feb. 2009 <http://www.cc.gatech.edu/computing/classes/cs6751_98_fall/handouts/GOMS-Kieras.html>.

17. UML 2.0 in a Nutshell

Pilone, Dan, and Neil Pitman. UML 2.0 in a Nutshell. Danbury: O'Reilly Media, Incorporated, 2005.

18. Sharpe Ratio

"Sharpe Ratio." Investopedia. 19 Feb. 2009 <http://www.investopedia.com/terms/s/sharperatio.asp >.

19. Stock Market Prediction

"Stock Market Prediction." Wikipedia, the free encyclopedia. 19 Feb. 2009 <http://en.wikipedia.org/wiki/Stock_market_prediction>.