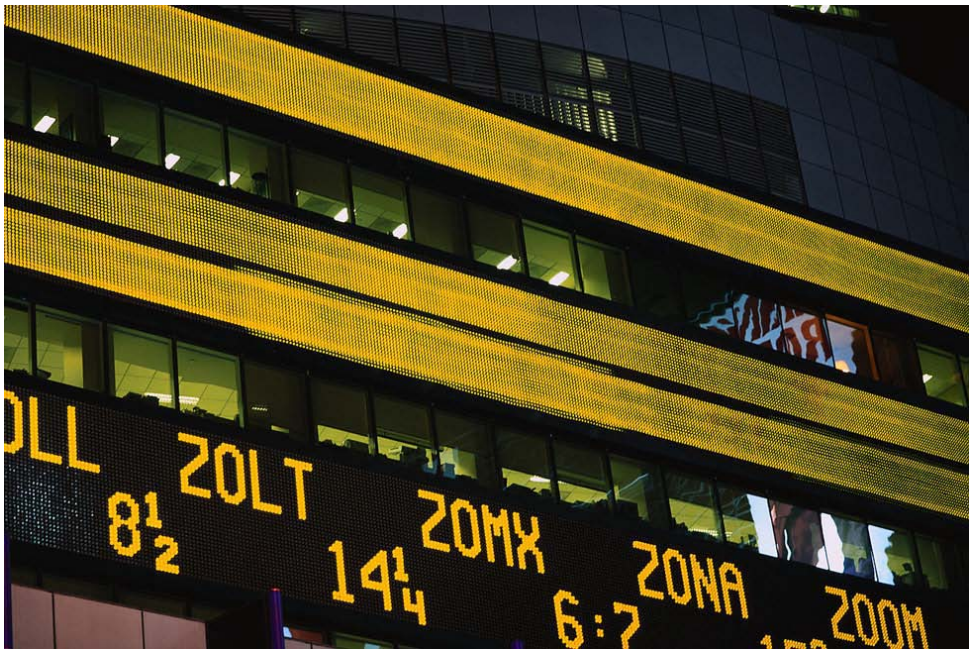
STOCK MARKET INVESTMENT FANTASY LEAGUE - REPORT 2



3/13/2009

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 2

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 2					
Application of Design Principles	34%		33%	33%	
UML notation	34%		33%	33%	
Prose Description	34%		33%	33%	
Class Diagrams	34%		33%	33%	
Data Types and Operation Signatures	34%		33%	33%	
Architectural Styles		100%			
UML Package Diagram		100%			
Mapping Hardware		100%			
Database Schema/ Persistent Storage		100%			
Other					100%
Algorithms and Data Structures					
User Interface design and Implementation					100%
Plan of Work			100%		
References		100%			
Project Management	20%	10%	20%		50%

FIGURE 1-1 - RESPONSIBILITY MATRIX

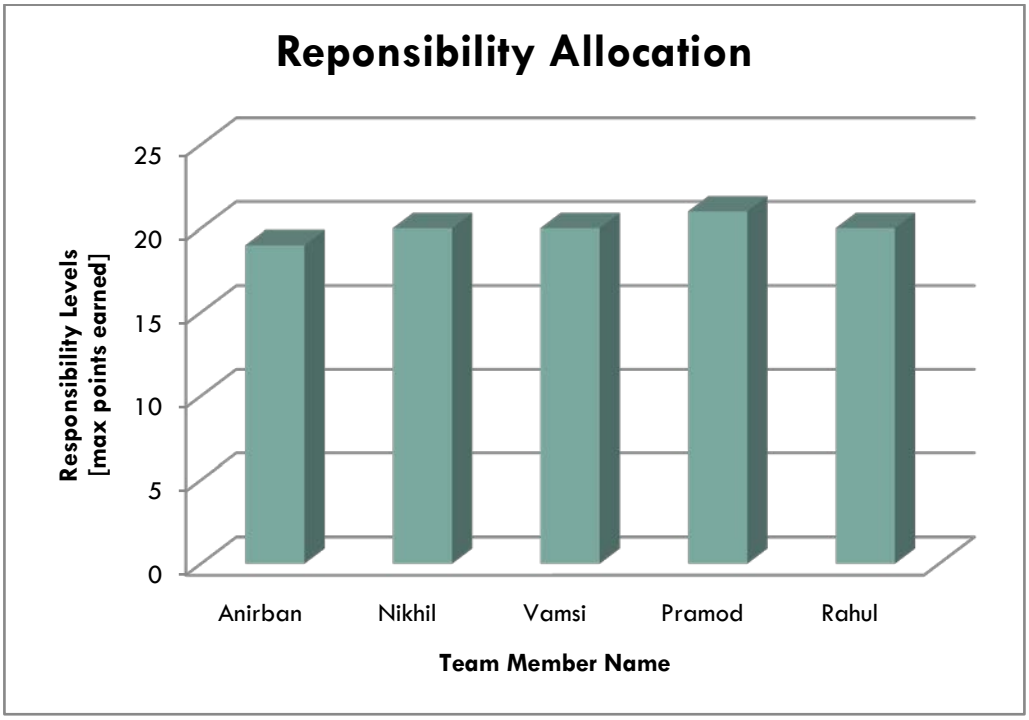


FIGURE 1-2 - RESPONSIBILITY ALLOCATION CHART

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) INTERACTION DIAGRAMS

a) List of Select Use Cases

UC 1 - Stock Database Update: The Stock Database will be updated on a timely basis. The data will be obtained using an external source. The update frequency will be determined according to the amount of resources available. (updated based on external timer actor)

UC 2 - Investor Database Update: The portfolio in the Investor Database will be updated on a timely basis. The data will be obtained using an external source. The update frequency will be determined according to the amount of resources available. (Is updated based on Stock Database and other internal considerations; inconsistencies with real world market must be addressed)

UC 3 - Open an Investor account, create profile: A user will go through the registration process to gain access to the system.

UC 4 – Setting preferences: Once logged in, the user will set account preferences and portfolio settings.

UC 5 - Disabling an existing account: A user will have the option to disable his or her account. The user's data will be discarded from the investor database.

UC 6 - Buy stocks (from market): User will be able to purchase stocks at market value. User has a choice of quantity.

UC 7 - Sell Stocks (to market): User will be able to sell stocks at market value. User has a choice of quantity.

UC 8 - Buy Stocks (from investors): User will be able to purchase at value set by other investors or make an offer.

UC 9 - Sell Stocks (to investors): User will be able to sell stocks to other investors. User has a choice of quantity and price.

UC 10 - Advertisements: Advertisements will be displayed to generate revenue using online services such as Google AdSense.

UC 11 - View Statistics:

A) View history: A list of previous transactions will be displayed.

B) Predictions: A forecast of an investor's net worth will be displayed according to financial models.

C) Recommendations: Suggestions such as what stocks to buy and what to sell will be provided.

D) Leaderboard: A sorted list of investors by his or her net worth, among other metrics.

b) Diagrams for UC-1, UC-2, UC-6, UC-7, and UC-11

There are many alternatives that were considered for the design of our system. We present here the major aspects of the Fully Dressed Use Cases.

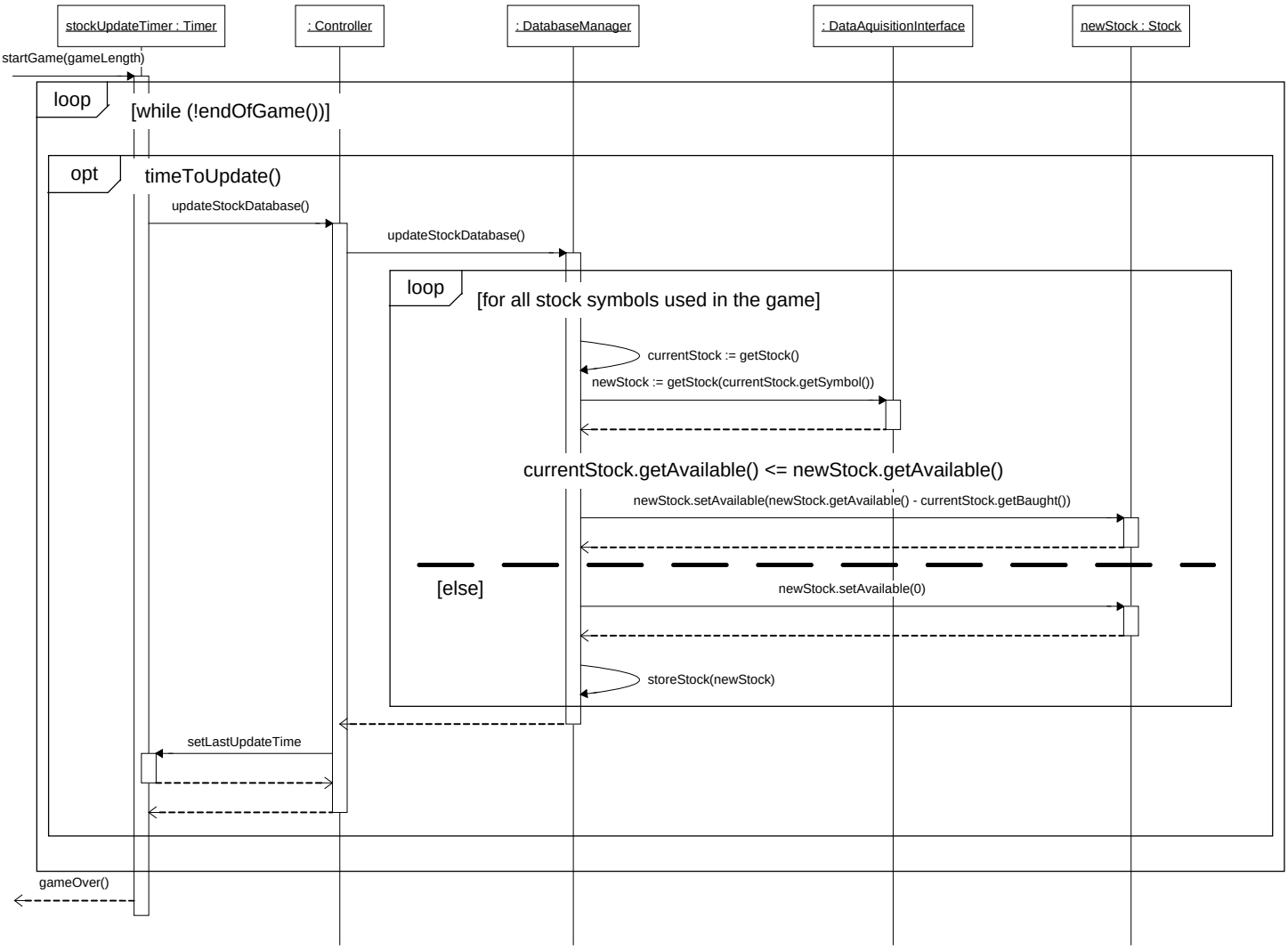


FIGURE 3-1 - UC-1 - STOCK DATABASE UPDATE

This interaction sequence diagram represents **UC 1 Stock Database Update.** An entire game period is depicted in this diagram. In order to maintain a real time environment in the fantasy game we use timers to initiate periodic updates the Stock Database. The Stock Database is an internal representation of the real world Stock Market; hence, this is a central aspect of our system. The controller in this diagram serves to forward command to the correct destination. Logically, the brunt of the work is handled by the DatabaseManager. The DatabaseManager must update the Stock objects stored in the Stock Database, for all the stocks that are being used for the game in session. During this update process there is an important condition that must be met. An updated "real world" stock must contain enough available shares. Inside our game, we subtract the internally purchased quantity from the "real world" available quantity to find the

"internal" available quantity of shares. If this "internal" available quantity is negative we know that our game is no longer consistent with the "real world" Stock Market. When this occurs we decide to update all information for the particular stock and set the "internal" available quantity to zero.

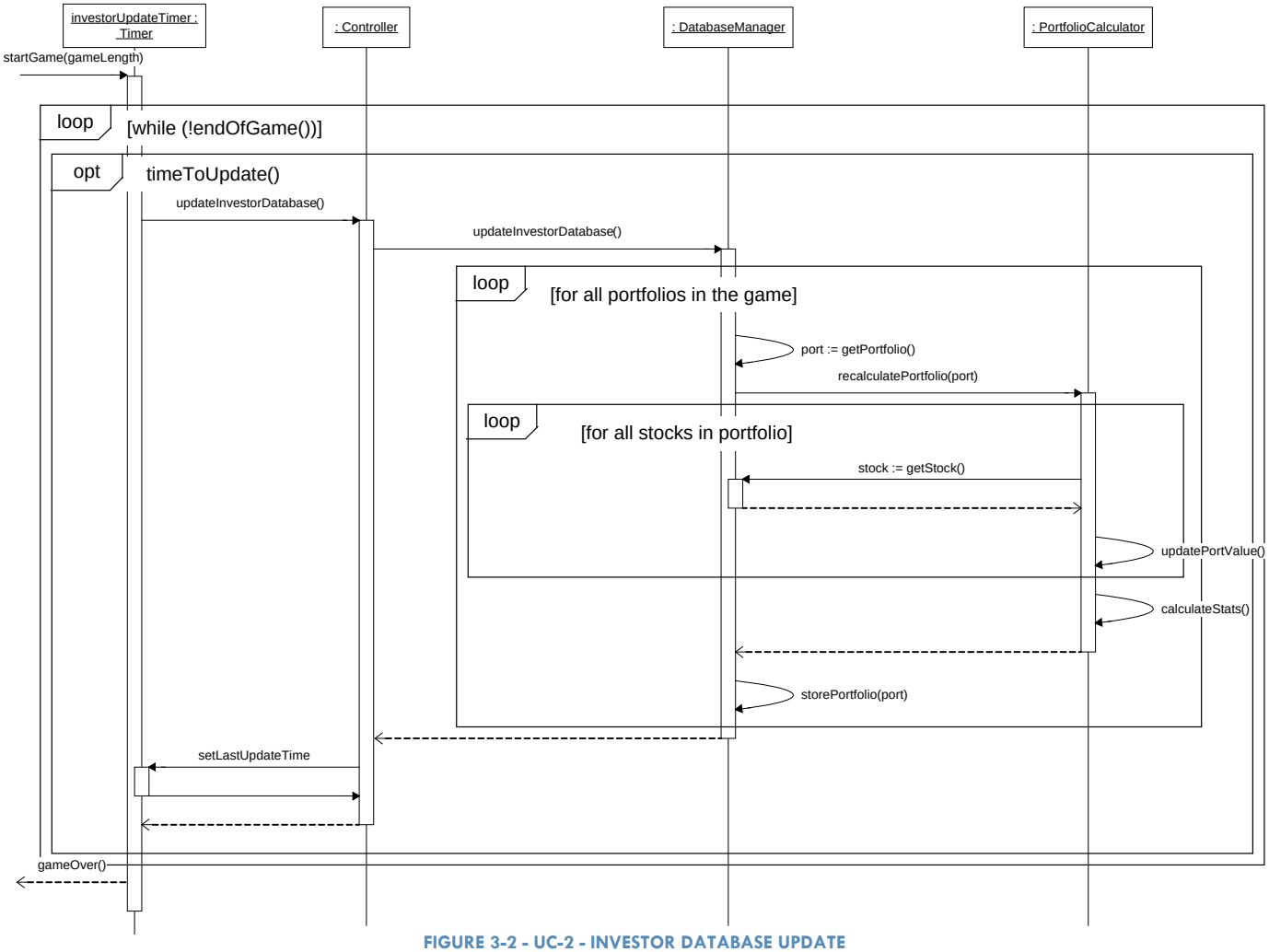


FIGURE 3-2 - UC-2 - INVESTOR DATABASE UPDATE

This interaction sequence diagram represents **UC 2 Investor Database Update**. An entire game period is depicted in this diagram. In order to maintain a real time environment in the fantasy game we use timers to initiate periodic updates the Investor Database. The Investor Database is where all investor information will be stored; hence, this is a central aspect of our system. The controller in this diagram serves to forward command to the correct destination. Logically, the brunt of the work is handled by the DatabaseManager. The DatabaseManager must update the Portfolio objects stored in the Stock Database, for all the portfolios that are part of the game in session. updatePortValue() which is called by the PortfolioCalculator performs the necessary adjustment, based on one stock, the value of the portfolio at hand. Once adjustments that pertain to individual stocks have been made, overall portfolio statistics are recalculated by the calculateStats() function call.

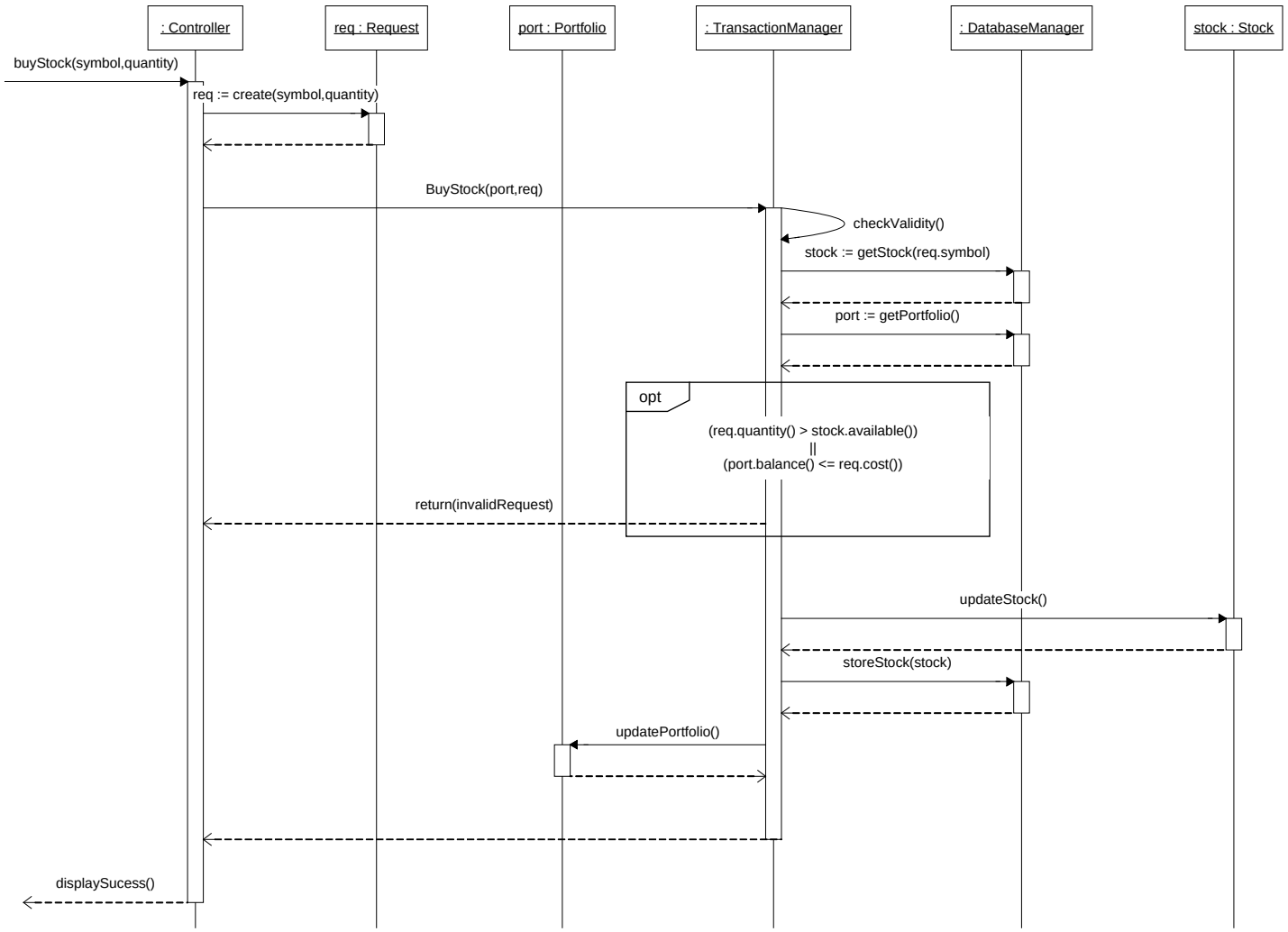


FIGURE 3-3 - UC-6 - BUY STOCK (MARKET)

This interaction sequence diagram represents **UC 6 Buy Stocks (from Market).** The timeline shown is that of when the user initiates the action to when he/she gets confirmation of it. The user first uses the interface to specify what stock to buy and how much (symbol and quantity). The controller passes this information along to create a request for the purchase. The transaction manager is notified of this request and checks its validity before proceeding to ask the DatabaseManager the current price. The transaction manager then accesses the portfolio through the DatabaseManager, and returns an invalid request to controller if the client does not have funds or the market doesn't have stock. If the manager thinks everything is fine, it updates the stock information by decrementing the quantity, updates the user's portfolio and the database, and returns to controller with success.

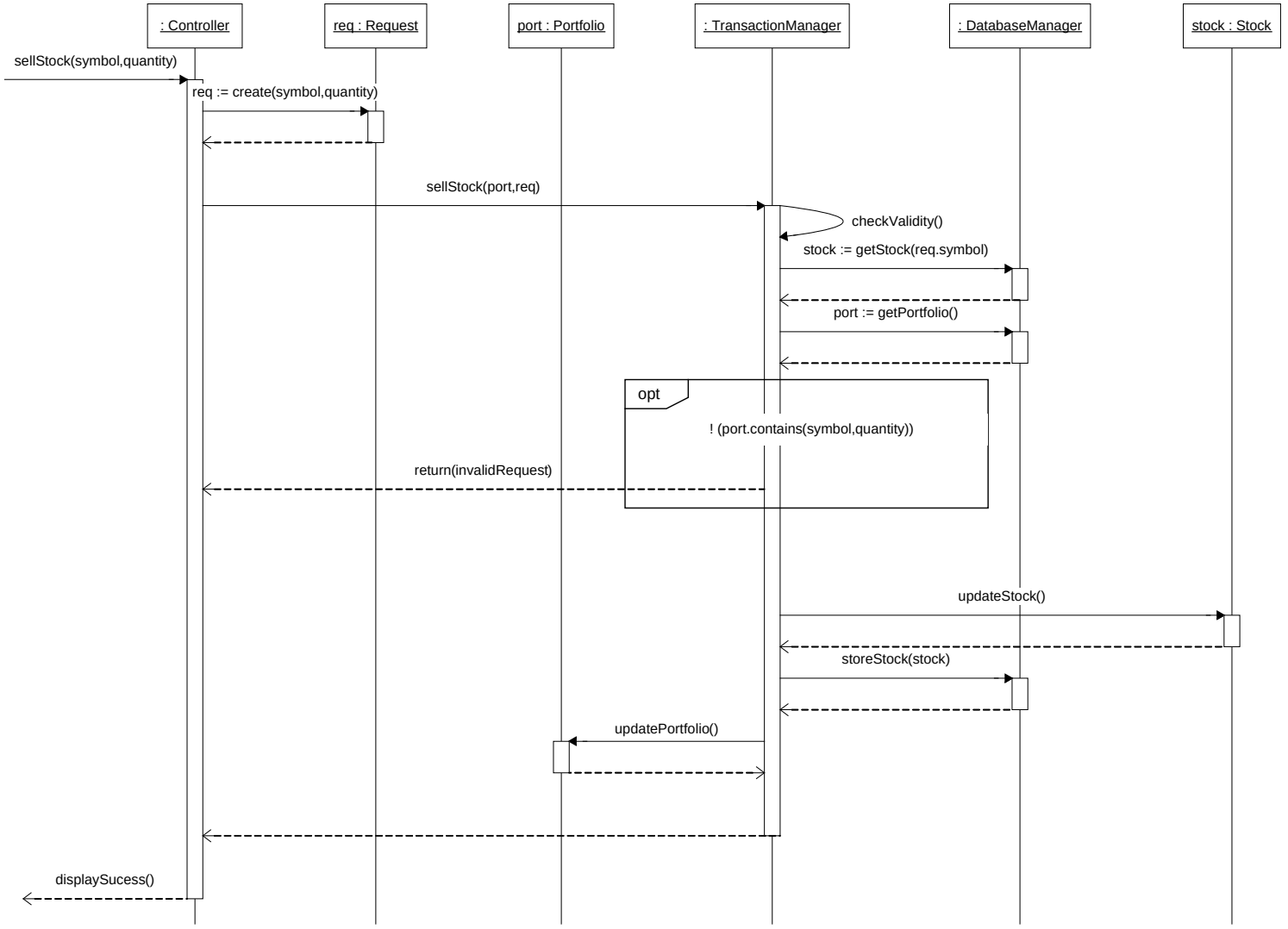


FIGURE 3-4 - UC-7 - SELL STOCK (MARKET)

This interaction sequence diagram represents **UC 7 Sell Stocks (to Market).** The timeline shown is that of when the user initiates the action to when he/she gets confirmation of it. The user first uses the interface to specify what stock to sell and how much (symbol and quantity). The controller passes this information along to create a request for the sale. The transaction manager is notified of this request and checks its validity before proceeding to ask the DatabaseManager the current price. The transaction manager then accesses the portfolio through the DatabaseManager, and returns an invalid request to controller if the client does not have the specified stock and quantity. If the manager thinks everything is fine, it updates the stock information by incrementing the quantity, updates the user's portfolio and the database, and returns to controller with success.

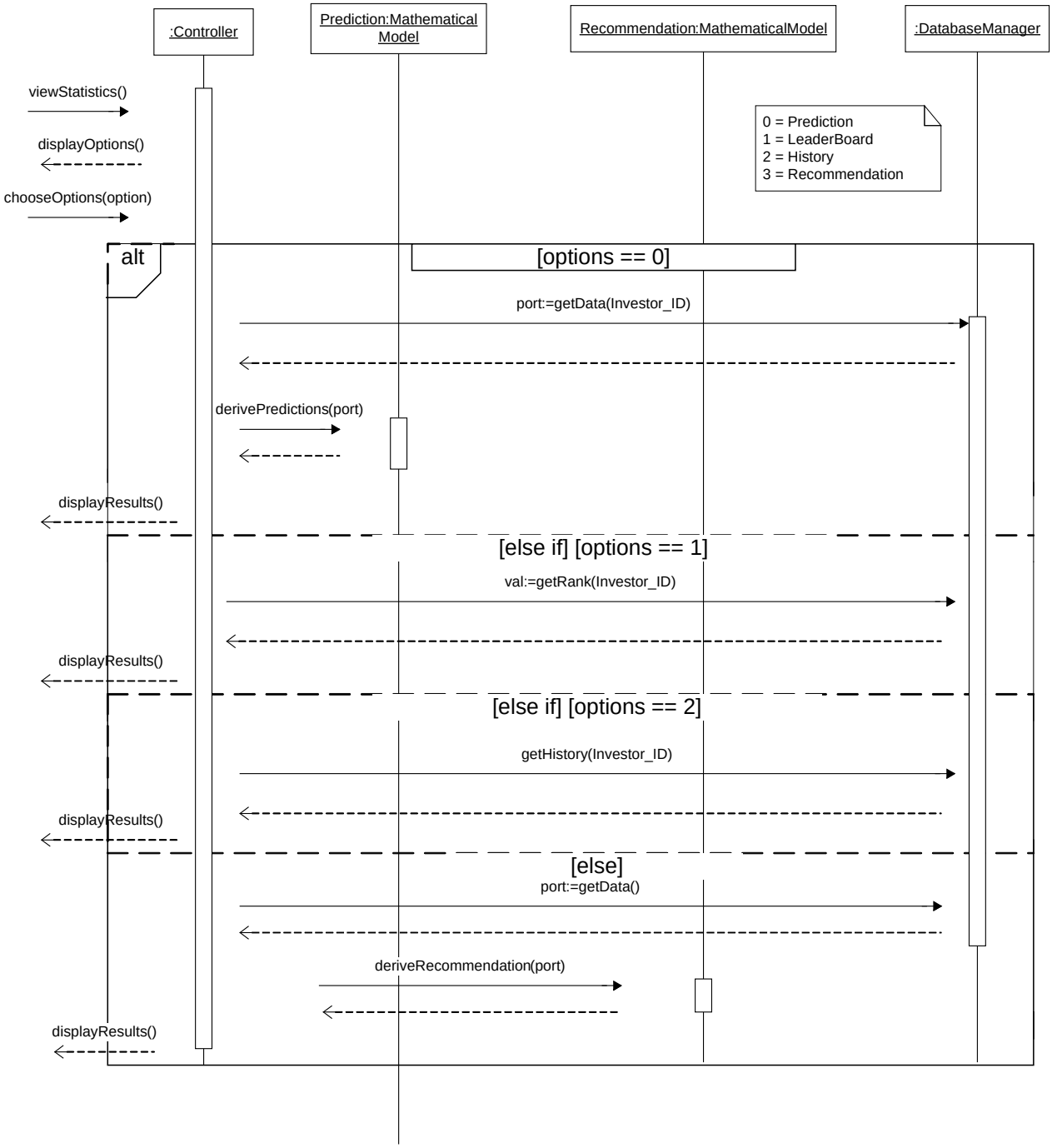


FIGURE 3-5 - UC-11 - VIEW STATISTICS

This interaction sequence diagram represents **UC 11 View Statistics.** A user wishes to see the statistics collected on his/her portfolio, and then the controller displays possible options for viewing the statistics. The user chooses various options and they are relayed back to the controller. Depending on the option chosen, the user can see history, predictions, recommendations, and a leaderboard, represented by the options value. Each selection has its own set of communication paths as seen above. Note that this is a preliminary diagram and may not have all requisite information.

4) CLASS DIAGRAM AND INTERFACE SPECIFICATION

a) Class Diagram

FIGURE 4-1 - CLASS DIAGRAM

b) Data Types and Operation Signatures (UML Format)

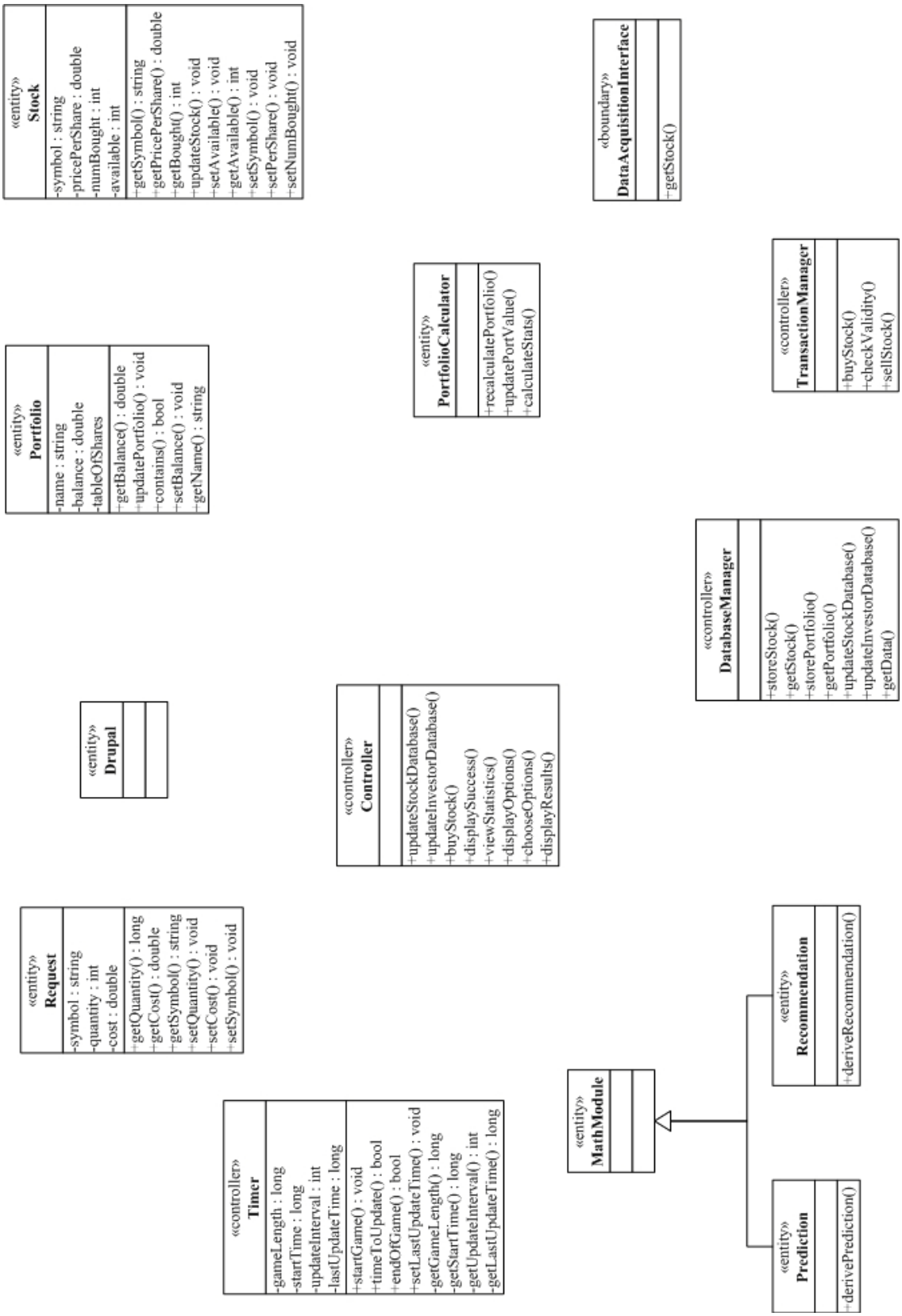


FIGURE 4-2 - DATA TYPES AND OPERATION SIGNATURES

The figure above (in a) Class Diagram) is the proposed the software class hierarchy. We note that the Math Module is a generalization of the Prediction and Recommendation classes, not an actual class.

5) 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

Although it was possible to identify the key subsystems through words, we chose to convey them using package diagrams as shown below because "A package diagram is worth a thousand words."

The package diagram shown just below is a pictorial conglomeration of all the concepts explained in the domain model and of all the classes shown in the class diagram. This diagram is drawn mainly to identify all the concepts that we analyzed during the first phase of the project. Moreover, the following diagram is divided into three packages: control, boundary and entity. The names of each package are taken directly from the three types of domain concepts: <<control>>, <<boundary>>, and <<entity>>. Needless to say, each domain concept is placed within its corresponding package name. For example, since controller is a domain concept of the type <<control>>, it will be placed under the control package.

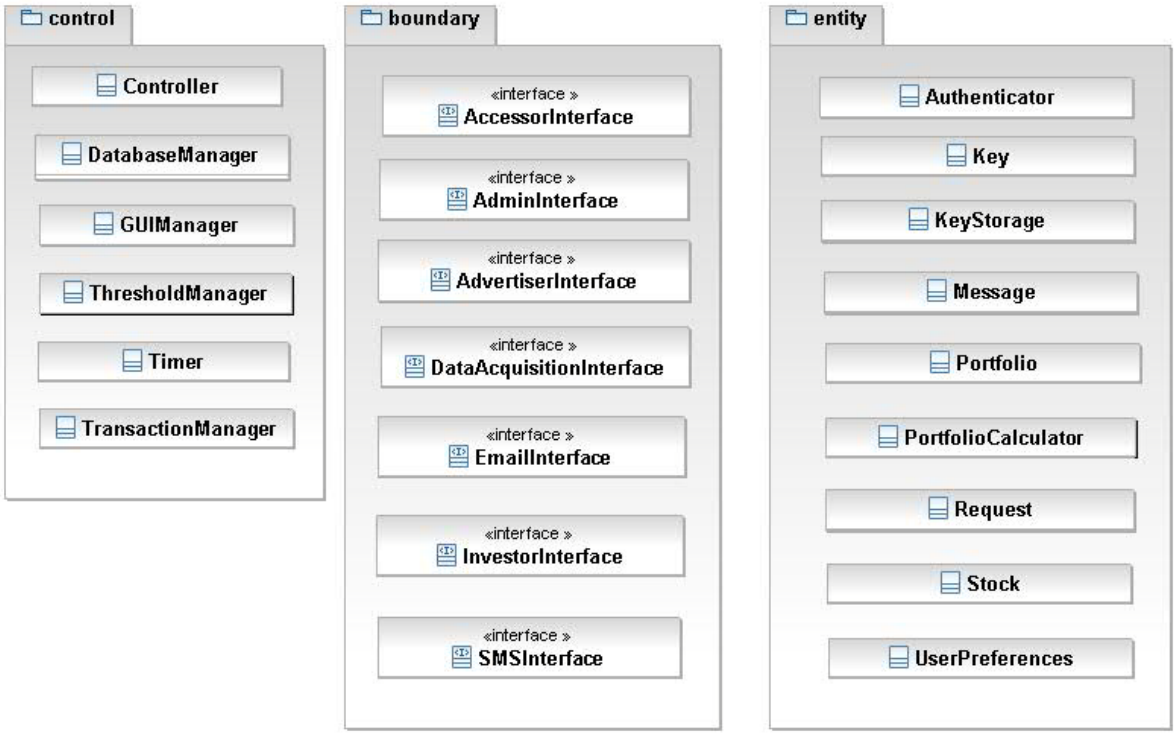


FIGURE 5-1 - INITIAL PACKAGE DIAGRAM

Although the package diagram shown above is a spectacular illustration of the entire system, it is not the correct representation of the type of implementation that we are planning to employ. We are using Drupal to create, maintain and run our game website. So, many of the key concepts shown above is being absorbed by the Drupal modules. These modules will automatically take care of the many of the boundary concepts and entities. Therefore, the following package diagram attempts to display the subsystems using Drupal - it is not known how much of the functionality Drupal is capable of taking over for the project, so please consider the diagram with this in mind.

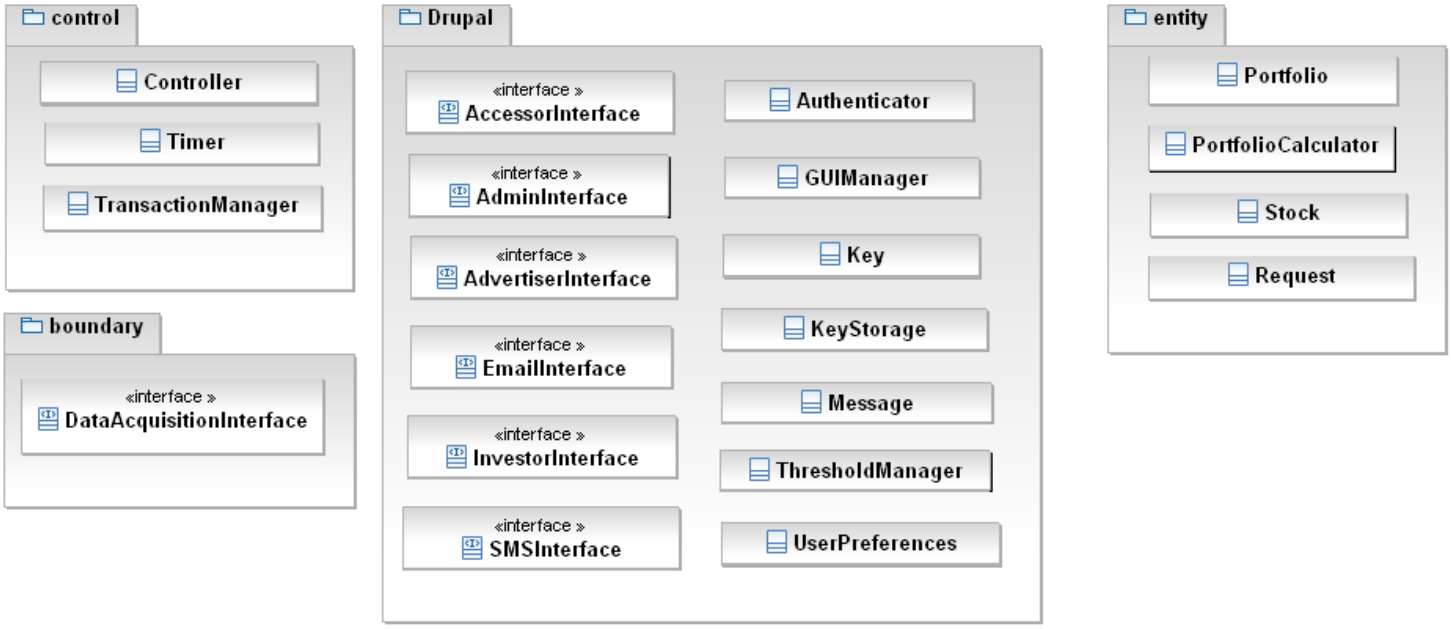


FIGURE 5-2 - UPDATED PACKAGE DIAGRAM

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 above. Although we may cook up more tables as this project progresses, the tables shown in the schema are finalized for now. Presently, we have two databases: investor database and stock database. The investor database holds personal information about each registered investor, each investor's portfolio and each investor's stock history. Similarly, the stock database contains information relating to all stocks available in our market. However, the investor database and stock database are divided into relational tables such as "INVESTOR" table, "PORTFOLIO" table and "INVESTOR HISTORY" table, "STOCK" table, and "STOCK HISTORY" table. Needless to say, each table is uniquely identified by a primary key and this is how each table will be accessed by the concepts within our system.

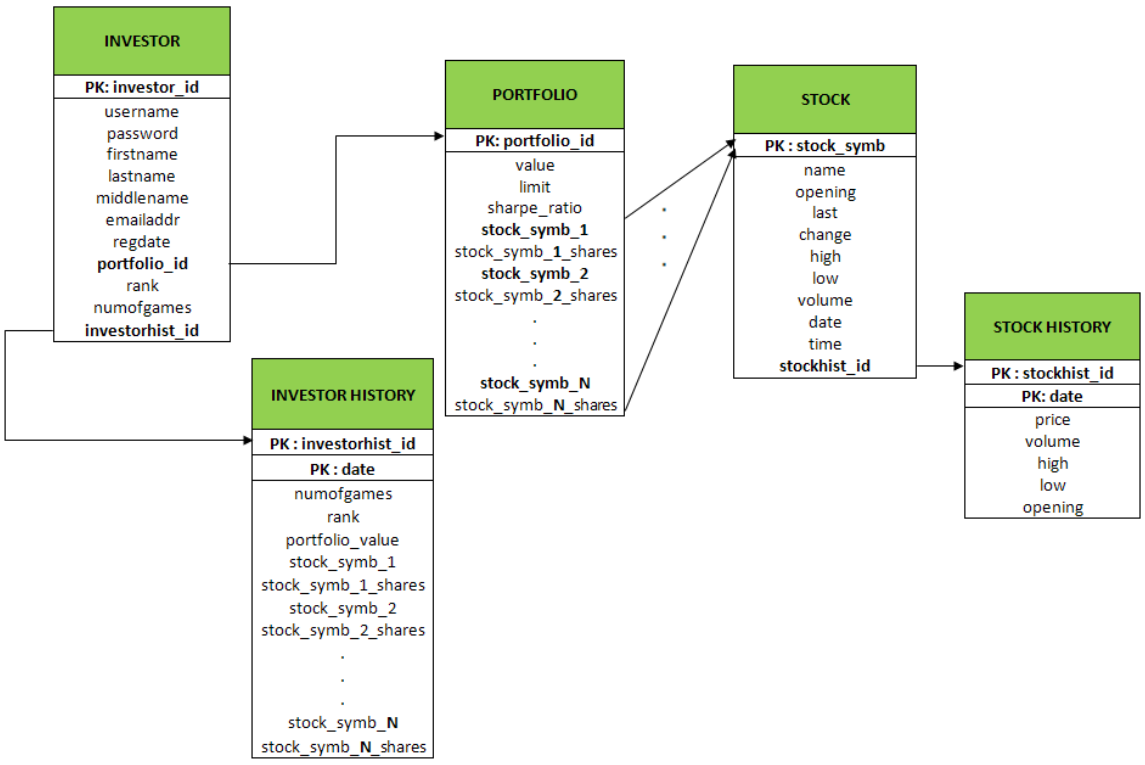


FIGURE 5-3 - SAMPLE DATABASE SCHEMA DIAGRAM

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.

6) 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: β_i = beta of stock i

β_p = beta of the portfolio

R_i = return on stock i

R_m = return on benchmark

σ_m = standard deviation of returns on benchmark

σ_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

σ_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.

7) USER INTERFACE DESIGN AND IMPLEMENTATION

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.

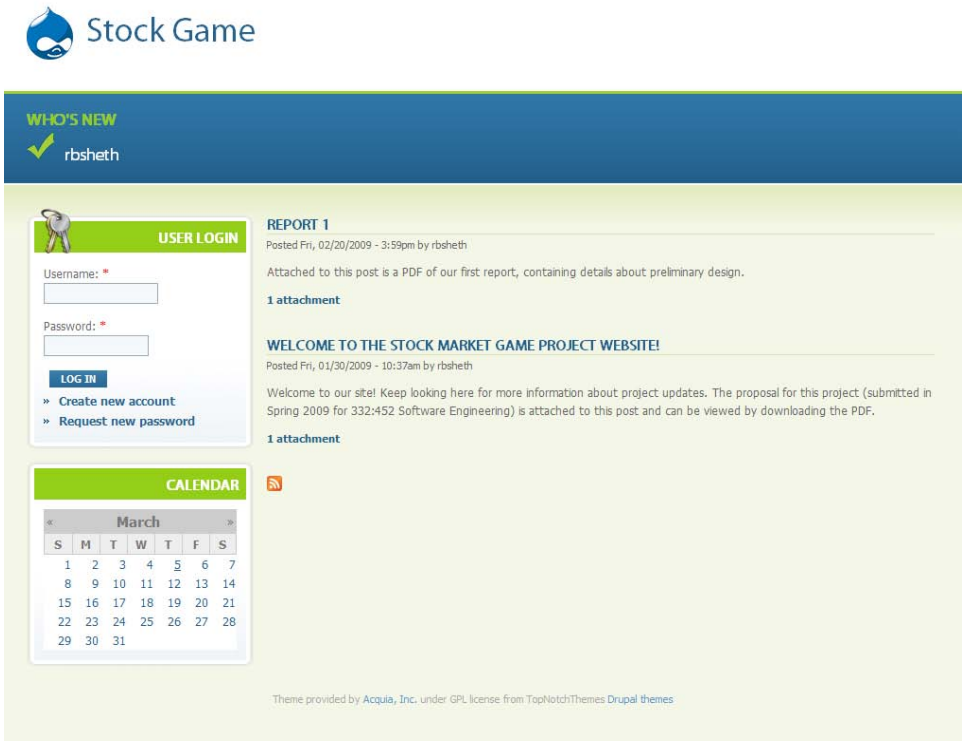


FIGURE 7-1 - FRONT PAGE OF THE WEBSITE

The figure above is the current front page of the website. The Drupal login block is shown on the left, and the calendar (tentative) 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. As is apparent, the interface is quite simplistic and most usage paths are hidden from an anonymous user. There are three ways to proceed 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.

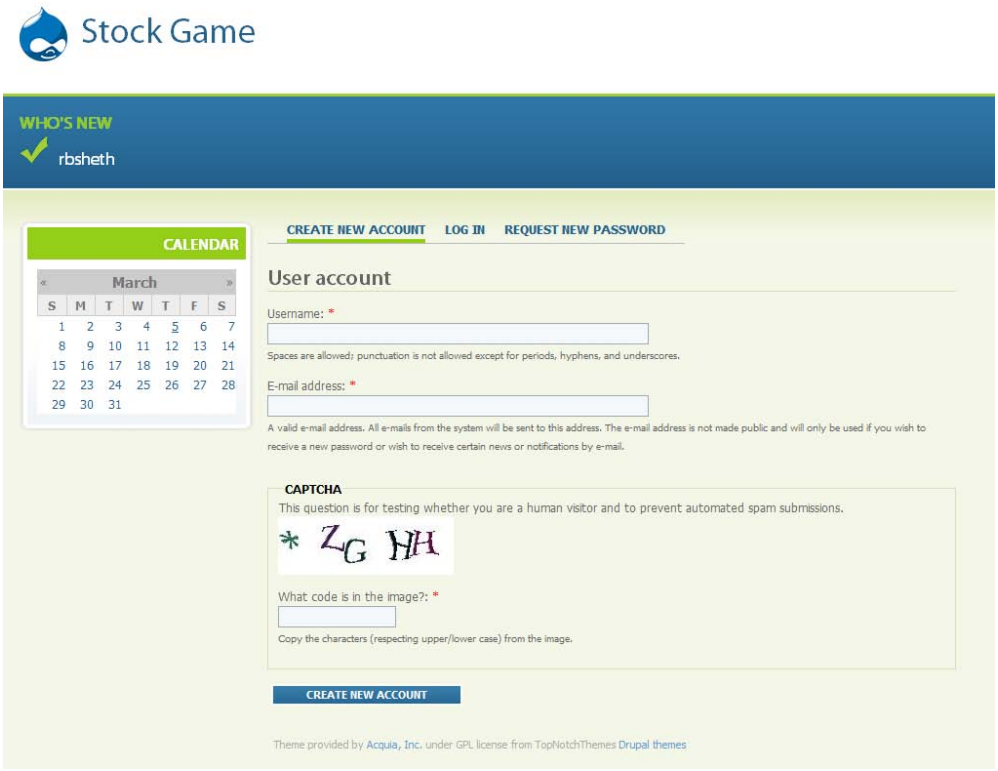


FIGURE 7-2 - REGISTER NEW ACCOUNT WEB PAGE

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. New users are featured on the front page of the site.

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.

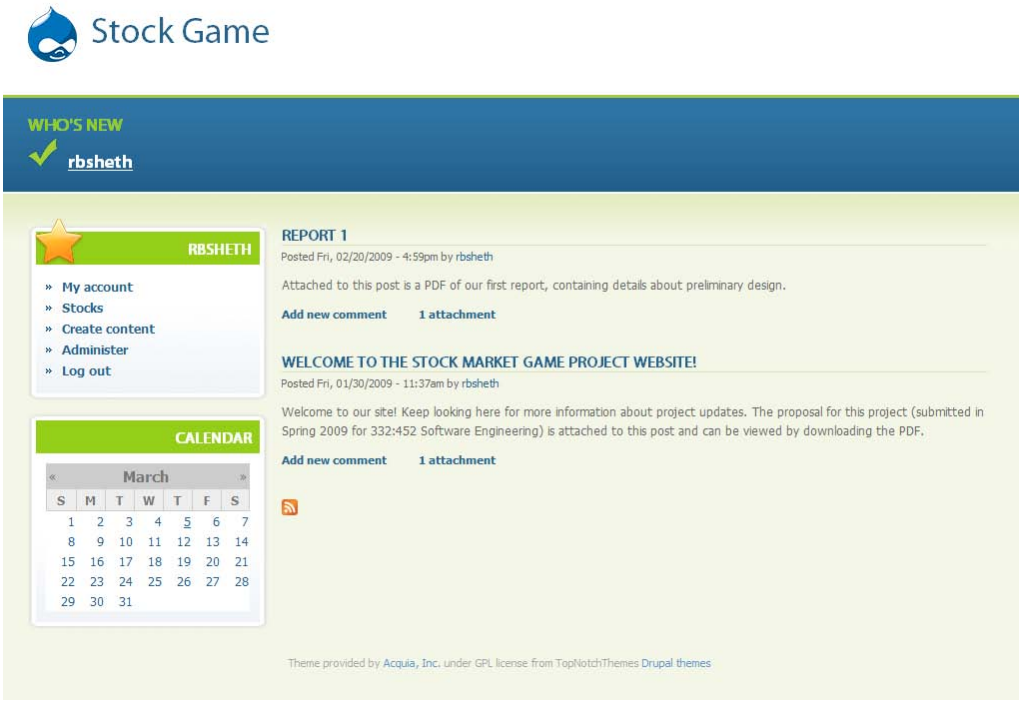


FIGURE 7-3 - POST-AUTHENTICATION WEB PAGE

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. For now, there is an option to look at the user's account, another to check stock prices, and some administrative features (for administrators only).

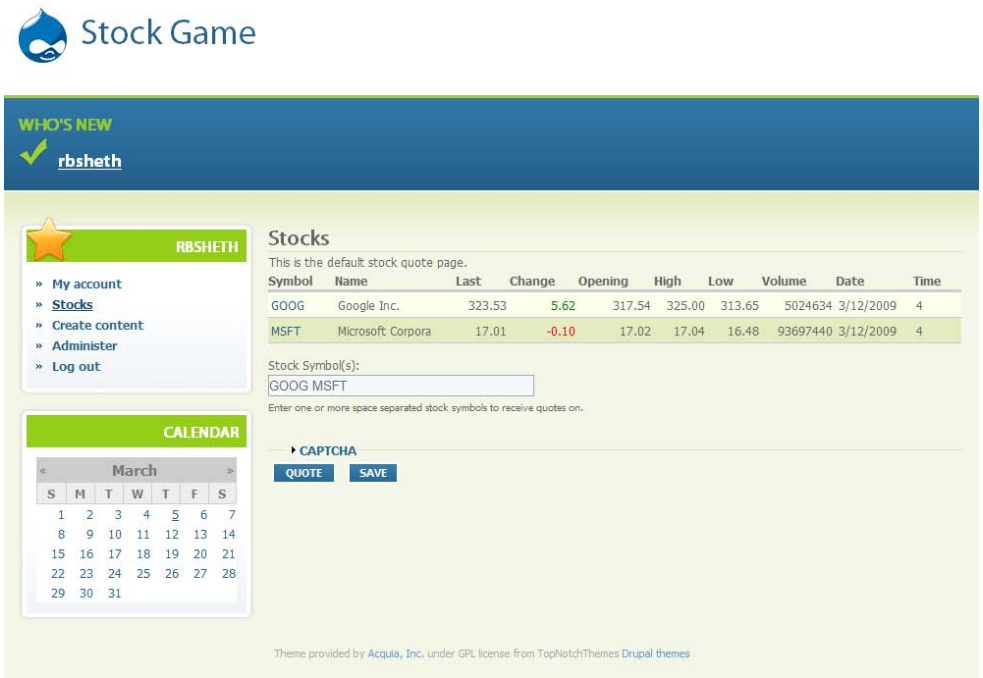


FIGURE 7-4 - STOCK QUOTE WEB PAGE

This is a temporary stock page to demonstrate some functionality of the stock quote retrieval system. It is undecided as to whether or not users will see this in the final product. There is a box to retrieve stock quotes by ticker, and the quotes are downloaded and shown on screen.

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 is very easy to do, requiring one click to perform the action. For game players, the interface is very intuitive and follows good design rules.

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.

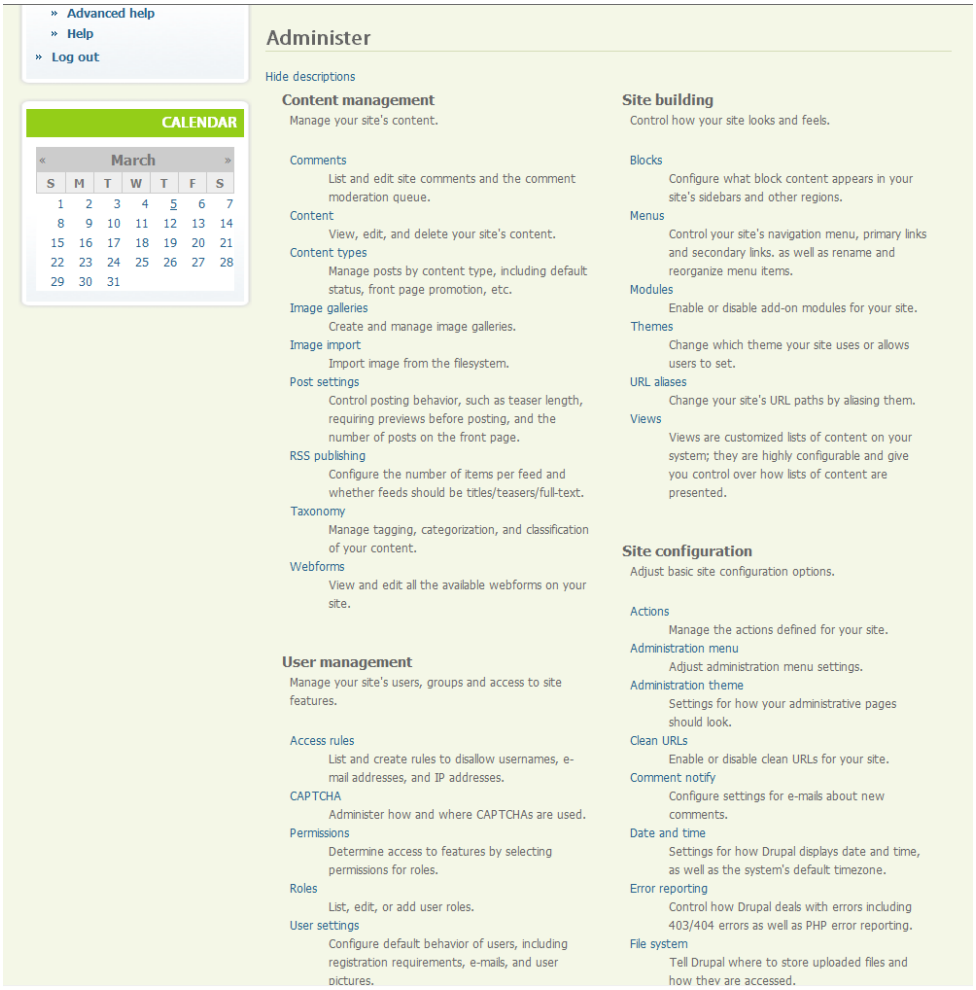


FIGURE 7-5 - A PORTION OF ONE ADMINISTRATIVE PAGE

8) PROGRESS REPORT AND PLAN OF WORK

a) Progress Report

Creating a Stock Market Investment Fantasy League consists of three parts as specified in the project description:

1. Part 1: Trader interaction (reviewing stock info and placing trading orders)
2. Part 2: Backend processing (tracking stock information, executing trading orders)
3. Part 3: Website operator and advertiser interaction (website management and advertisement placement)

Our group decided to use Drupal content managing system as a backend for our system. As Drupal is a robust framework, much of the administrative tasks can be completed easily.

List of Completed Use Cases:

Authentication System (UC 3, 4, 5):

- Drupal existing authentication system already includes individual user accounts, administration menus, flexible account privileges and easy administration.
- Our idea is to integrate this account system with associated Investor portfolio.

Stock Market Database Updates (UC 1):

- Stock market data is update using a Stock API which fetches data from Yahoo! Finance
- It needs to be configured to update automatically at set intervals and must be cached for increased performance

List of Uses Case In-Progress:

Investor Database Update (UC 2):

- Our idea is to design a "wrapper" around Drupal's existing account system.
- The Investor Database will be populated by an investors account data.

Buy/Sell Stocks (UC 6,7,8,9):

- Collection of classes for this use case will be implemented in PHP either as sub module or a module.
- This will then be integrated with a "game" module which will run on the website.

Statistics (UC 10):

- We will implement custom Drupal modules to handle statistics.
- Displays of graphs can be done using Charts API.

b) Plan of Work

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

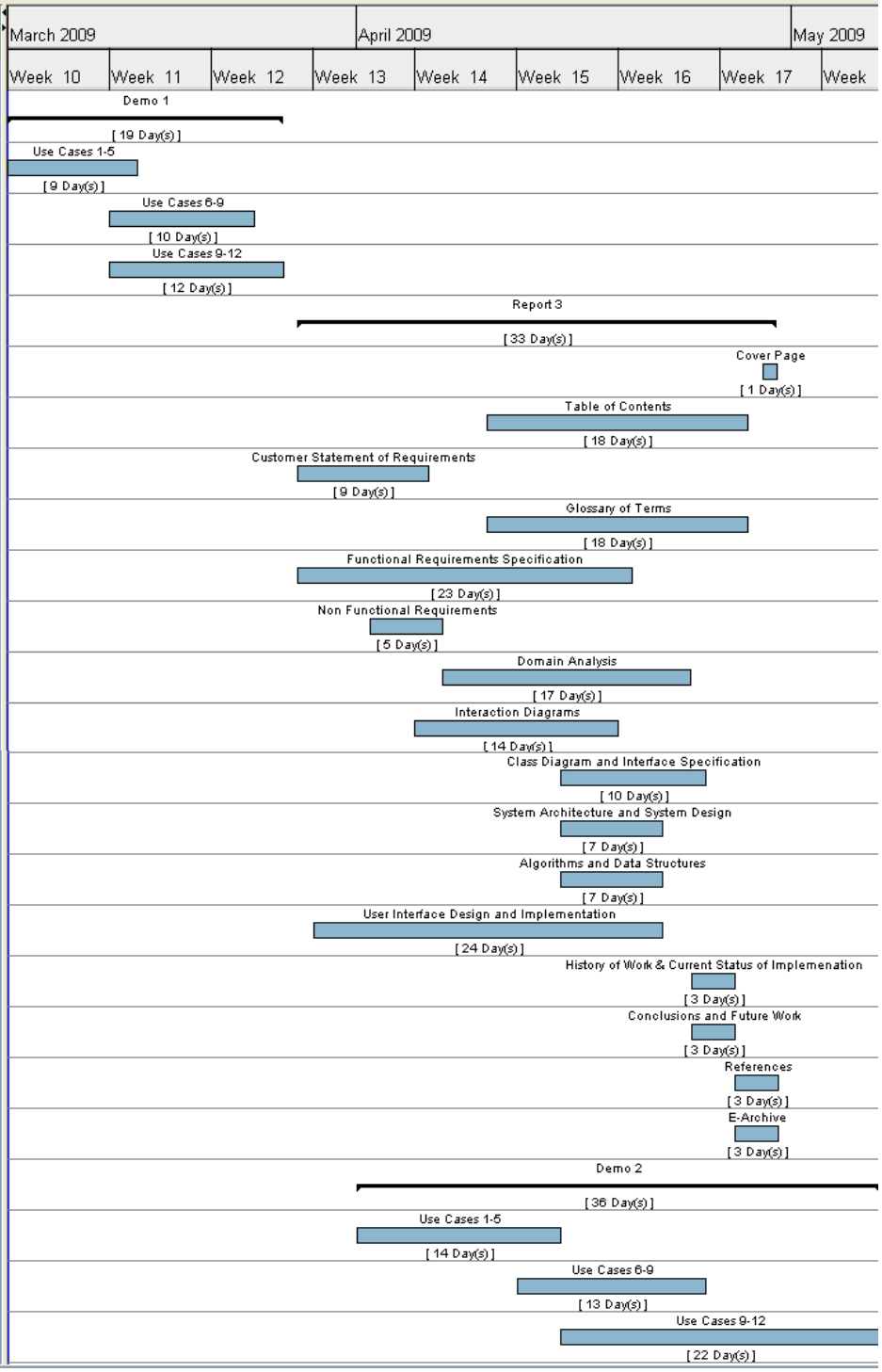


FIGURE 8-1 - GANTT CHART FOR PLAN OF WORK

c) Breakdown of Responsibilities

List the names of modules and classes that each team member is currently responsible for developing, coding, and testing (preliminary):

- Anirban Ghosh: Leaderboard block programming and design, buying/selling transactions
- Nikhil Kasthurirangan: Adding portfolios to Drupal users, buying/selling transactions, advertisement module
- Vamsi Kodamasimham: Backend processing module (getting data from financial source), E-Mail/SMS notification module
- Pramod Kulkarni: Value at risk calculation module, Graph API integration
- Rahul Sheth: Drupal subsystem configuration and administration, user interface layout and maintenance, advertisement module

Rahul Sheth will coordinate the integration, and is responsible for most of the documentation compilation and formatting.

Nikhil Kasthurirangan will perform the testing of the integrated system.

9) 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>.