The Paramount Investments League

Final Report
Software Engineering
14:332:452

Team 1:

David Patrzeba
Eric Jacob
Evan Arbeitman
Christopher Mancuso
David Karivalis
Jesse Ziegler

May 1, 2014

Hyperlinks:

Webapp Link
Project Repository
Reports Repository

Revision History:

Version No.	Date of Revision
v.1.1	2/7/2014
v.1.2	2/16/2014
v.1.3	2/23/2014
v.2.1	3/2/2014
v.2.2	3/9/2014
v.2.3	3/16/2014
v.2.4	3/19/2014
v.3.1	4/19/2014

Contents

1 Customer Statement of Requirements

1.1 Problem Statement

The stock market, more specifically the New York Stock Exchange(NYSE) and the Nasdaq play a pivotal role in the American economy today. Both are signals of the strength of the private sector and consumer confidence. It is thus no surprise that more and more people want to be involved in these markets and attempt to increase their own wealth.

There is however a barrier to entry for many people, both young and old in participating. That is why with Paramount Investments League we are interested in a platform for interacting with these markets and providing educational interfaces for breaking down these barriers. Users should be able to easily register with the system and begin participating immediately. They should be given an imaginary cash portfolio where they can perform basic market orders such as buy and sell. These orders to should mimic real market orders as closely as possible and should include a brokers fee. More sophisticated market maneuvers should be unlocked as the user progresses through an achievements ladder.

Paramount Investments League is geared towards a wide array of audiences and expects a variety of users with varying knowledge levels to participate. In order to maintain appeal amongst these users the platform should provide rewards to users for acheiving particular goals. We would like to replicate the idea of achievements or trophies similar to the Microsoft xBox and Sony Playstation family of systems. These achievements can award users with new abilities or additional cash to their portfolio as they rise up the achievements ladder. Users should also be able to create leagues to help further enhance the competitiveness of the game.

Leagues exist to allow multiple users to compete against a subset of the global user base with individual league rules. This allows leagues to set particular goals in order to be declared the winner. Leagues will require a cash buy-in that will be pooled together and distributed to the winner(s) as seen fit by the league creator. To help facilitate these leagues, a leader board will be created for each individual league such that users can see their progress. In addition to league leader boards, mutliple global leaderboards will be available providing specific metrics of comparison.

To help facilitate a better understanding of markets, market metrics should be available to the user through news feeds of companies in their portfolio, interactive charts, and a live ticker of current trades happening on our platform. Users should be able to have granular control of email and social media updates.

The entire experience should be unified across mobile, tablet, and the desktop and combined with the above features provide an enthralling core experience for users to learn about the stock market.

1.2 Glossary of Terms

The following terms give a small overview of some of the items which will be necessary for fully understanding of the purpose of this software's design. The details provided should encapsulate the idealogy which is important to gaining full understanding of the goals and processes within the software designed. Further, the terms will also describe features and functionality as well as the important financial terms which are crucial to comprehension of the software and how it works.

Achievement – Any set goal reached by an investor. Achievement rewards can be managed by a league manager and may include badges, capital, equity, etc.

Transaction Ticker – Constantly updating scroll of most recent trades across the market. Users can observe market trends from global equities which may or may not already be in their portfolio.

Leaderboard – Global or league based ranking system determined by overall net worth of player.

Security – A tradable asset of any kind. Can include debts, equities, or derivatives. For the purpose of this game, we will be dealing primarily with equities.

Dividend – A payment made by a corporation to its shareholders, generally as a distribution of profit. It is usually distributed as a fixed percent of shareholder value.

Derivative – Any financial contract which derives its value from another asset or index.

- **Option** – Gives the user the option to buy or sell an asset at a specified price on or before a given date. The buyer and seller are both obligated to fulfill the transaction on the given date if the option is taken.

- **Future** - Allows the buyer to buy an asset at its current price and pay for it at that price in the future. A future is generally exchange traded. The buyer and seller are both obligated to fulfill the transaction on the given date if the future is taken.

- **Forward** – Allows the buyer to buy an asset at its current price and pay for it at that price in the future. A forward is a private agreement between buyer and seller not necessarily based around market equity. The buyer and seller are both obligated to fulfill the transaction on the given date if the future is taken.

League – A market simulation with a pre-determined rule set and several investors with a common goal to determine a winner. Goals can vary across leagues as determined by league managers. Investors can choose to opt into a private league, public league, or no league at all.

Portfolio – A detailed account of assets associated with a particular investor in a given league. Portfolios are unique to each user and will contain specific details such as earnings, losses, performance, averages, as well as detailed asset performances of equities within the given portfolio.

League manager – The league manager will have the responsibility of adding and/or removing investors from the league. League managers control settings, and victory conditions for a particular league. League managers maintain their manager status only for the league in which they have created.

Order – An investor must place an order for the purchase or sale of an asset.

Stock – A type of asset that represents equity in a company.

- **Ask Price** – The price at which a trader is willing to sell a stock.
- **Bid Price** – The price a trader is willing to pay for a stock.
- **Bid-Ask Spread** – The bid-ask spread describes the difference in price between the bid and the ask. These two prices are marginally different, but always with the ask being the more expensive of the two. It represents the friction inherent in trading a stock.[1]

Ticker Symbol – an abbreviation used to uniquely identify publicly traded shares of a particular stock on a particular stock market.

Symbol List – a list of a market/several market's ticker symbols.

Market Order – Any order placed for immediate market transaction.

- **Buy** – User has elected to purchase a particular stock and has placed a bid for that stock.
- **Sell** – User has elected to sell a particular stock and has posted an ask price for it.
- **Short** – Typically used by an investor who expects the value of a stock to decrease. The investors borrows shares of a particular stock and sells them at market price. The investor can then buy back the stock at a lower price profiting the difference. The investor is still responsible for the stock should its value increase.[2]

Limit - An investment which will only take place at a given price. An investor placing a buy limit will place a maximum amount the pay and an investor place a sell limit will place a minimum value for which the stock be sold. Limit orders are not guaranteed to ever process, and only do when the particular limit is reached.[3]

Stop – Orders which are activated if a particular stop falls below or rises above a particular price. It is used to minimize gains and losses for the investor.[4]

Share – A small percentage of a given company which can purchased or sold from other traders.

2 System Requirements

2.1 User Stories

The user stories written and elaborated below demonstrate several particular instances and requirements for program functionality, as well as a weight to measure relative importance of each requirement. In particular these functions are not necessarily written in order of particular weight or functional precedence but are simply a list of end user story requirements and relative weighted importance. It is important to observe that these cases will be elaborated on and referenced in further sections of this document. The following are told from the perspective of the user from his or her view with the intention of fully encapsulating what he or she should expect to be able to see or do upon entering and regularly using the referenced software.

Identifier	User Story	Weight
ST-1	As a user, I can create an account without registering with the website in order to participate in Paramount Investment League.	10 pts
ST-2	As a user, I can access the application across multiple platform paradigms so that I may continue to participate when I don't have access to a desktop computer.	10 pts
ST-3	As a user, I can join or create leagues with self-selected goals so that I may compete with others in a simulated stock market environment based on near real-time stock data.	10 pts
ST-4	As a user, I can search for companies by stock symbol and be presented with their current financial information so that I may research future investments.	6 pts
ST-5	As a user, I can browse a companies profile and view the performance data over a configurable span of time so that I may determine whether or not I want to invest in them.	6 pts
ST-6	As a user, I can buy or sell stocks so I may build my portfolio.	10 pts

ST-7	As a user, I can earn badges(achievements) that reward me with additional capital or new features for accomplishing predefined tasks.	10 pts
ST-8	As a user, I can manage my portfolio within a league to track my investments.	8 pts
ST-9	As a user, I can visually track my finances via graphs and charts so I may more easily manage my portfolio.	4 pts
ST-10	As a user new to the stock market, I will have access to an educational interface that teaches me about the stock market via pop-up dialogues.	6 pts
ST-11	As a user, I can see trades being made by all other users in real-time via a stock-ticker like marquee so I may have a quick overview of current trends.	3 pts
ST-12	As a user, I can see the performance of other users' portfolios so I may observe the investment habits of others.	2 pts
ST-13	As a user, I can view a portfolio leader board so I may have a summary of relative performance between users in my league.	1 pt
ST-14	As a user, I can opt to receive periodic e-mail notifications of my stock performance or trades so I may be kept up to date even when not actively viewing the site.	3 pts
ST-15	As a user, I can additionally link my account with social media sites so I may share my fantasy league experience with friends.	1 pt
ST-16	As a league manager, I can add league rules, a league name, and a league logo to personalize my league.	8 pts
ST-17	As a league manager, I may invit who I want to join.	8 pts
ST-18	As a league manager, I can create league announcements.	4 pts
ST-19	As a site administrator, I can view reports of and delete leagues that are inactive.	2 pts
ST-20	As a site administrator, I may post front page news or announcements.	3 pts
ST-21	As a site administrator, I may have access to a user count, number of active leagues, total leagues, quantity of daily transactions, the most/least popular stocks, and newly created so I may have reliable site statistics.	9 pts

ST-22	As a league manager, I can choose the specific victory conditions for a particular game (eg: first to a certain capital, net gain, or overall gain within a time). As a user I can view this condition and my progress toward victory.	5 pts

The above requirements outline a general list of requirements which we expect to reflect the core functionality of our software (with higher weighted items acting as higher priority and being implemented first). The ultimate goal of the software is to simulate that of a real world stock market with users having the options to perform and carry out the important and basic trading actions (see ST-6). We plan to add increased functionality when compared to years prior, however. With the addition of achievements, varied victory conditions, as well as increased leaderboard functionality Paramount Investments will appeal to a larger audience than that of years past (see ST-7, ST-13, ST-22). Notice that items such as administrative privileges as well as league creation and stock execution are prioritized with substantially higher priority with relation to our newly added functionality. This is because the core functionality of the software is absolutely crucial to it working. We will expand on the core as well Supplement requirements.

Core requirements

These requirements are absolutely crucial to the viability and progression of the software. That is the user can create and log into an account on a daily basis. We will use a basic authentication system to implement this. Importantly the user will be able to access this UI on multiple different platforms to ensure complete and smooth transitional access to the system with zero down time. (ST-1, ST-2)

The user will be able to access his or portfolio. (ST-6) From this portfolio, they can view their currently owned stocks as well as monitor the performance of their portfolio. They can view progress toward goal requirements and badges. (ST-8) From this location they can take action to buy and sell stock or perform short, stop, limits, etc.

League managers will have access to a specific configuration setup where they can choose victory conditions, league settings, and monitor progress of investors within the league. This functionality is core to the formation of leagues within the game. (ST-3, ST-16)

This project will NOT be its own market. In order to maintain the idea of perfect competition and unbiased market prices, all data will be taken from Yahoo! Finance to submit data and trades will be taken from here. This software is not intended to be a way for people to trade actual stock, rather just a resource for learning the market and tools of trade.

Supplemental Requirements

The user will be able to access social media integrated applications, and decide whether or not to keep their social media profile updated and informed with updates on progress from their fantasy league. (ST-15) They may also receive email updates with various progressions in the game (ST-14)

The user will be kept updated on the progress of other users to view their trades as well as recent market trades and trends. (ST-12, ST-11)

Users will also have access to on-site term explanation similar to that seen on Wikipedia. That is, they may scroll over an underlined term to find a brief definition and additional resources. (ST-10)

2.2 Nonfunctional Requirements

Functionality

Additional features for security will be enabled through the use of a OpenID and OAuth through a third-party library. There exists several packages for the purpose of authentication and authorization of users. Key authentication features are the ability to encrypt and store passwords, provide recovery options for users that have forgotten their password, and store a cookie to validate the session.

Usability

A key point in the design of this application is ease of use and appeal to the users. The application should be interactive, informative and consistent across multiple platform paradigms. Additionally the application will be used to provide the educational interfaces noted in ST-9 which should be able to be toggled on and off so that users can always view the information again.

Reliability

In order to ensure that there is no confusion to the user in the case of the internet or server failure, all transactions end with a final confirmation, and no changes to the account are made until after this confirmation. The user's portfolio will thus always be in a consistent state and will be restored when the user is able to log back in. A user that leaves the application and returns later will still be logged in. Server failure should also be dealt with by keeping backups of user data. Proper care should also be taken to handle a situation where a particular stock source is not available (i.e. Yahoo Finance).

Performance

In order to have a great performance, the website should be as lightweight as possible by keeping hardware demands to a minimum on both the client and server sides. For it to be efficient, any task initiated by the user should be completed in a timely manner. The web server should be able to serve concurrent requests especially when a large number of users are logged in. Any frameworks used should be lightweight but consideration should be taken not to prematurely optimize.

Supportability

It should be feasible to extend pr update any server components and include improved versions of modules which can be installed only by administrators. For scaling purposes, it should be made easy to include an additional number of servers to achieve load balancing. The system should be platform independent so that it is easy to move to newer technologies or the next versions of

web server. The system itself should also be backed up to a remote server for the sole purpose of extending functionality and testing new features in a controlled environment.

2.3 On-Screen Appearance Requirements

There are a few on screen requirements that will be universal to the entire site:

Identifier	Requirement
OSR-1	Every page has a scrolling ticker across the bottom of the page to update the user on stock movement.
OSR-2	Every page, with the exception of the login page, will have navigational links across the top, the user's username and their current position in the leaderboard.

There are also the following requirements for specific pages:

Identifier	Requirement
OSR-3	A custom 404 not found page will be displayed to a user when they try to access a URL/URI that doesn't exist or is not designed for them to be accessing.
OSR-4	On the portfolio page users will find currently owned stocks, charts and graphs, trade transcations, and a news feed.
OSR-5	The leaderboard view will contain users ranked by the top networth from their respective portfolios.
OSR-6	The login page will present the user with login icons representing the service they can use to log into our system, eg: google+, facebook, etc.

The below image gives a general representation of the on-screen requirements for the login page. You will note here that this is a general design strategy where a background image and login pattern will be debuted upon final release. The idea, however, is to demonstrate a framework for what the user can expect to see upon entering our webpage. More importantly, this framework will be the first impression the user has of our site and will be retouched and refined as necessary. The outline is shown above.

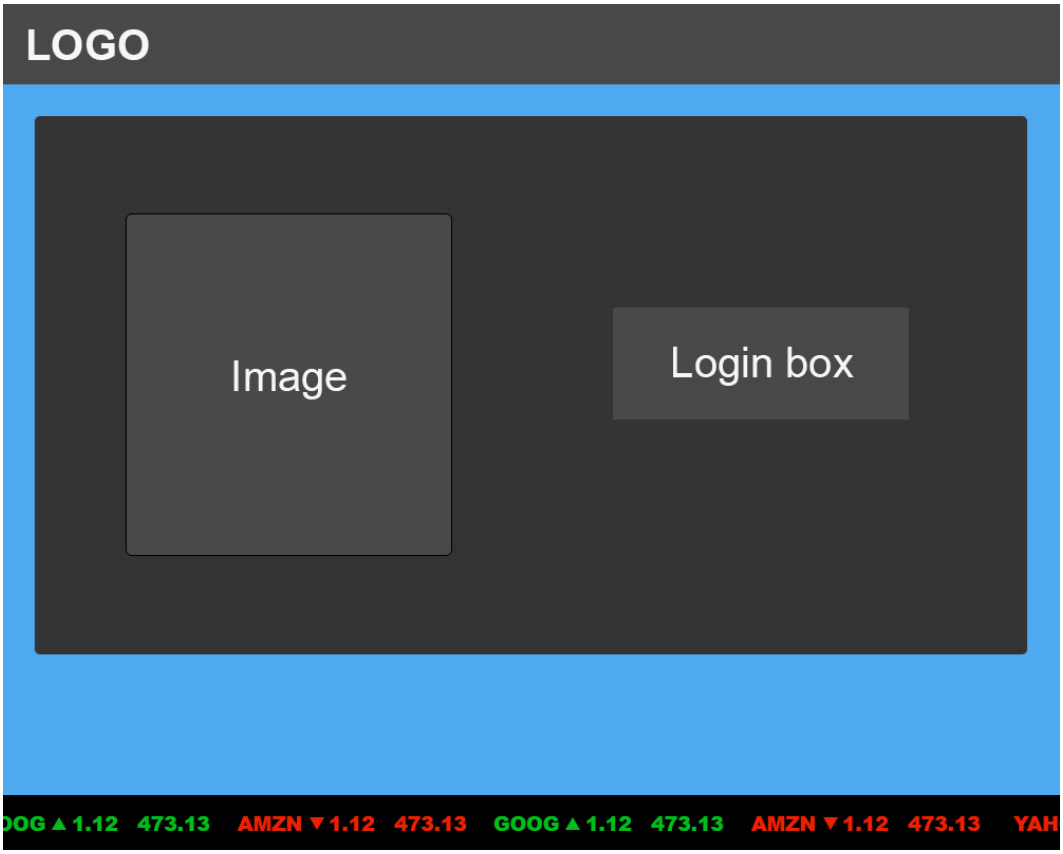


Figure 2.1: Basic on screen requirements of login page

3 Functional Requirements Specification

3.1 Stakeholders

In general, a stakeholder is someone in an organization or group that is affects or is affected by the decision made by that particular organization or group. For the Paramount Investments League, any user (investor) in a league will be considered a primary stakeholder because any transaction made within their league, will influence other users investors.

The target demographic for the software described tends to be centered on students and first time investors, as a result, it is unlikely that novice investors will participate in real trade without any experience. That being said, it is likely to see the software expand to take a large role in both university and pre-university classrooms, as a means of teaching general financial concepts. It would not be unlikely to see the game further expand to a larger range of users than other similar software due to increased functionality, addition of achievement and leaderboards, and ability to join with or without league functionality. Specifically, the addition of achievements leaves the user with the desire to return and spend additional time trading on the software.

The league will be a free service with the intention of eventually moving to a subtle-advertisement platform which will have no impact on the user. Once a substantial enough user-base is generated, it will not be unlikely to see advertisements begin to commence in order to bring revenue to the company. As a free service (with eventual advertisements) we expect the platform to attract the greatest number of users, and due to increased functionality, keep said users on the platform for the greatest amount of time. The software is targeted not only at students and potential investors, but at nearly everyone who desires to gain a greater understanding of the financial industry as well as those who would simply like to practice trading before executing in the real market.

3.2 Actors and Goals

Investor

A user who has an account in our servers and is logged in to their account. These will be regarded as our primary stakeholders.[5]

- Research the latest updates in the market
- View their portfolio

- Execute orders of any kind
- Join/create a league
- Take part in competitions

Guest

A visitor to the website who has either not logged in or just a simple visitor. Visitors will be regarded as secondary stakeholders.[5]

- Register and create an account using OpenID/OAuth2
- View the latest trades

Administrator

Also referred to as an Admin, is the person in charge of managing the operation of their respective field. For instance, a League admin is in charge of maintaining league settings.

League Administrator

Manages the leagues that they have created

- Can set league to be public/private
- Set the rules for the league

Site Administrator

Manages the overall website

- Ensure fair competition between leagues/players

Database System

Holds the information for the accounts of all users

- Insert information as accounts are created
- Push data back to views about users/events
- Store new data about about users/events

Financial API

Gives the stocks in our database up to date prices

- Fetch real world information and update our database accordingly

Browser

The middleman between user and system

- Present data to the user
- Retrieve data from the user

Yahoo! Finance

The unit that knows about current financial statistics

- Retrieve data about stocks

Queueing System

A subsystem for scheduling orders so as not to block user interactions.

- Place orders to be executed or canceled asynchronously
- Schedule events and mailings for system

3.3 Use Cases

Preface

Users will have instant access to the functionality of the site as soon as they have created an account and logged in. That is, they can perform all of the functions that an investor can perform. They will also have the ability to choose to create or join a league, though they are not required to in order to experience the full functionality of the software. This will give our software a broader demographic when compared to that of years prior. That being said, league creation and administrative user delegation will be an important part of the functionality of the program and will be described its own use case. Core functionalities of respect user types will be elaborated below.

Use Case Casual Descriptions

UC-1

Register/Create an account using OpenID/OAuth2 Allows a new user to register and/or create an account on the website server using OpenID/OAuth2. This should let players easily log in with their social media accounts (Facebook, Twitter, Instagram).

Derived from: ST-1, ST-2

Note: A player does not have to reauthorize if logging in from a computer they have previously logged in from, as long as the social media service theyre using is logged in to their account.

UC-2

Create/Join a League Allows a registered user to create a league according to their preferences. This use case should give the player creating the league freedom to chose many options that affect how the league functions, according to their liking. It makes the player creating the league the leagues administrator. If the user is not creating a league, this use case should allow them to join a league that already exists.

Derived from: ST-3

UC-3

View Market Data Allows a registered user to view market data for a given time period. The user should be allowed to view data on stocks, companies, and trades. If the user is viewing data after the markets have closed, the data returned is the last updated information. Otherwise, the data returned is live.

Derived From: ST-4, ST-5, ST-11

UC-4

Manage Portfolio Allows a registered user to manage their portfolio. This use case should let players view their position in the different leagues they are participating in. This can either be the global league or any player-created league they have joined.

Derived From: ST-8

UC-5

Place a Market Order Allows a registered user to place an order to buy/sell stocks, or any other advanced moves theyd like to make. The order must be placed within the operating hours of the market to obtain accurate quotes. The changes made to the portfolio after orders have been placed will be reflected immediately.

Derived From: ST-6

UC-6

Take Administrative Actions Allows the site administrator to take actions to promote the well being of the website. This use case could entail anything from kicking players out of leagues to deleting dormant leagues. There can only be one entity to take administrative actions on the website.

Derived From: ST-19, ST-20, ST-21

UC-7

Manage League Settings Allows the registered user who is also the manager of a league to take actions to maintain the well being of the league. This use case allows the league manager to change the league rules, kick players out of the league, or a multitude of other options given to league managers.

Derived From: ST-16, ST-17, ST-18

Fully-Dressed Use Cases

Having an account within our database is necessary for the user to experience functionality within the software. That being said, the user can create an account in different ways. They can choose to have the information imported by logging in through any OpenID/OAuth service service (eg: google, facebook, twitter, etc.) This will populate the database fields automatically with data driven from the external resources. Once the user has created an account, they can log in through OpenID/OAuth and will generally remain logged into the system as long as they remain logged into their service of their choice.

CG-BP01 A user can create an account such to access the full features of the game. They can return and log in with minimal burden and access all of the information previously stored.

Use Case UC-1	**Register/Create an account using OpenID/OAuth2**
Related Requirements:	ST-1, ST-2
Initiating Actor:	Guest
Actor's Goal:	Register with our servers
Participating Actors:	Guest, Database
Preconditions:	-Guest must not be a registered user
Postconditions:	-The **Database** is updated with guests information and logs the guess in as an **Investor**
	Flow of Events for Main Success Scenario:
→	1. **Guest** navigates to Paramount Investment League and logs in
←	2. **System** checks **database** for **investor** and isn't found
←	3. **System** retrieves OpenID/OAuth info and registers guest in **database** as an **investor**
→	4. **System** sends out confimation to user and displays starter portfolio
	Flow of Events for Alternate Scenarios:
→	1. **Investor** attempts to login
←	2. **System** checks **Database** and finds **investor**
←	3. **System** loads **investor** info from **database**

→	4. **System** displays users portfolio

Any user has the option at any time to create or join a league. The user who has requested to create a league will have elevated privileges versus a standard user. The league manager will be prompted to make a league with various setting options for victory condition, badges, achievements, etc. The league manager can also choose whether or not to make the league private. A private league will restrict users to those invited by the league manager, and will require a password to join. After initial setup, league managers will have minimal access to settings. That is, halfway through a league, the manager cannot decide to change the victory conditions. This prevents the league manager from abusing power to tip the scales in his or her favor.

CG-BP02 A user can create a league or join a league if he or she desires. Note that users may trade without a league, but has access to create or join a new league at any time. Leagues may be private or public.

Use Case UC-2	**Create/Join a League**
Related Requirements:	ST-3, ST-8, ST-16, ST-17, ST-18
Initiating Actor:	Investor
Actor's Goal:	Create or join a league to compete in
Participating Actors:	Database, other Investors
Preconditions:	-Investor is logged in
	-league is not created or user hasn't joined league
Postconditions:	-The league is created with the appropriate settings or
	-The **Investor** has joined the league
	-The **Database** has been updated
	Flow of Events for Main Success Scenario:
→	1. **Investor** navigates to and clicks on the create league dialogue.
→	2. **System** displays to the **Investor** the available options for creating a league.
→	3. **Investor** updates the settings, such as privacy, league name, number of spots, and managing users
←	4. **System** sends the updated settings to the **Database**
→	5. **System** sends confimation to the **Investor**
	Flow of Events for Alternate Scenarios:
3a. The **Investor** selects league settings that are disallowed, such as a league name that already exists.	
→	4. **System** informs user what settings are incorrect.
Investor wishes to join a league	

→	1. **Investor** navigates to league listing.
←	2. **System** updates **database** with **investors** info.
→	3. **System** confirms **investor** as part of league and displays league site.

Users can view raw market data which will be pulled from Yahoo Finance in near real time. Users can view company data either on their own portfolio page or through the companys specific info page. That is, the user can view detailed information of each stock or company before committing to a trade from a variety of sources. Users will be able to compare their portfolios performance to typical market trends from the Nasdaq, S&P 500, and DJIA. There will also be a stock ticker ribbon on the bottom of the screen for users to receive constant real time feeds of most recent trades happening within the market.

CG-BP03 A user can view market data in near-real time. They will have access to data taken from the Yahoo! Finance API and can view this data in their portfolio or by searching for stocks using tickers.

Use Case UC-3	**View Market Data**
Related Requirements:	ST-4, ST-5, ST-10, ST-11
Initiating Actor:	Investor
Actor's Goal:	View the latest information about stocks, companies, and trades
Participating Actors:	Database, Yahoo! Finance
Preconditions:	-Yahoo! Finance is accepting inquiries
	-Investor is logged in
Postconditions:	-None worth mentioning
	Flow of Events for Main Success Scenario:
→	1. **Investor** searches for a market term
←	2. **System** sends request to **database**
→	3. **System** returns suggested terms
→	4. **Investor** selects a term from suggested terms list and sends request
←	5. **System** sends request to **Yahoo! Finance**
←	6. **database** is updated.
	Flow of Events for Alternate Scenarios:
Search Fails	
←	6. **Yahoo! Finance** returns no results
→	7. **System** informs **investor** of search failure

User will be able to view all major items within their portfolio as well as place trades from

their portfolio page. From this page, a user can view detailed analysis and graphs of each of their respective stocks as well as their current rank within their league (if applicable) and globally. Users will also be able to place trades for respective companies through their portfolio page. Users can buy, sell, short, or carry out any additional action on any stock or security within the limits of their finances and league settings through this page (See UC-5). Users will also be able to customize and change views as well as add stock index comparisons to monitor their success vs market success.

CG-BP04 All users will have a portfolio which will house detailed information regarding all of their stocks and performance. They will be able to perform trades from this location and will also have access to the core functionality of the site through this page. It will essentially be the users home page.

Use Case UC-4	**Manage Portfolio**
Related Requirements:	ST-8, ST-9, ST-10, ST-12, ST-13, ST-14
Initiating Actor:	Investor
Actor's Goal:	Manage portfolio by viewing current standings/stocks/securities
Participating Actors:	Database, Yahoo! Finance
Preconditions:	-Yahoo! Finance is accepting inquiries
	-User is logged in
Postconditions:	-**Investor**'s portfolio is updated to reflect change in position
	Flow of Events for Main Success Scenario:
→	1. **Investor** navigates to their portfolio
←	2. **System** requests users portfolio from **database**
→	3. **System** displays portfolio to **investor**
→	4. **Investor** adjusts their portfolio
←	5. **System** updates the **database**
→	6. **System** displays confirmation of portfolio update

User should be able to place trades from various locations. That is, they may place it through their portfolio by typing in the ticker and quantity of shares. They may also navigate to a certain companys page and elect to purchase shares there. Selling shares should be done through the users portfolio where they may see the exact quantity of shares of each respective companies they own. Error messages will be thrown and orders not processed should a user request to buy more shares of a company the he or she can afford or the user attempts to sell more than he or she has. Main transactions will occur through the users portfolio.

CG-BP05 A user can place any market order he or she wishes to place (I.E. buy, sell, short, limit, stop, etc). These trades can be executed from multiple locations and at the prohibitions of the league rules.

Use Case UC-5	Place a Market Order
Related Requirements:	ST-6, ST-11
Initiating Actor:	Investor
Actor's Goal:	Place orders to buy/sell/short stocks, or place a stop/limit order
Participating Actors:	Database, Yahoo! Finance API
Preconditions:	-Investor is logged in
	-Yahoo! Finance is accepting inquiries
Postconditions:	-**Database** us updated with the users position
	Flow of Events for Main Success Scenario:
→	1. **Investor** enters a market order
←	2. **System** attempts to get update from **Yahoo! Finance**
←	3. **System** receives information back from **Yahoo! Finance**
←	4. **System** validates and records trade in **database**
→	5. **System** confirms trades and displays changes in **investor** portfolio
	Flow of Events for Alternate Scenarios:
3a. **System** doesn't recieve information back from **Yahoo! Finance**	
←	4. **System** notifies **investor** of failed request

Of the 3 user types, administrator is the highest and reserved only for developers and administrators of the software. Administrators have the ability to modify or delete leagues or specific users if the administrator feels that power is being abused. The administrator will also have elevated privileges to makes changes to the site. Their main purpose will be to suspend or ban users or leagues and ensure that the site is not being abused. This includes but is not limited to robot or AI users or user account spamming or advertising rather than trading properly.

CG-BP06 Administrators will have access to suspend, ban, or remove leagues or players completely. Administrators will have elevated privileges relative to other users and will use these privileges with the sole intention of maintaining the integrity of the software.

Use Case UC-6	Take Administrative Actions
Related Requirements:	ST-19, ST-20, ST-21
Initiating Actor:	Site Administrator
Actor's Goal:	Perform administrative work for the website, manage database
Participating Actors:	Database, Investors, League Manager
Preconditions:	-User is the site Administrator

	-Administrative actions need to be taken
Postconditions:	-Conflicts/Issues have been resolved
	Flow of Events for Main Success Scenario:
→	1. **Site Administrator** requests logs from **system**
←	2. **System** closes saves log file and returns log report

Depending on the specific role of the user. (I.e. investor, league manager), users should be able to customize several different items. That is, the league manager will have ultimate customization of the rules of his or her league, including but not limited to: victory conditions, achievements, and starting capital. These rules are to be established prior to the beginning of the league and not touched for the duration. League manager will also have a few other elevated privileges depending on whether he or she also falls into the investor role as well. Investors will be allowed to manage their own personal settings including but not limited to: email alerts, social media integration, and notifications.

CG-BP07 League managers will have access to a list of settings which they may select before the initiation of the league competition. They will have slightly elevated privileges from basic investors, but not enough to abuse power.

Use Case UC-7	**Manage League Settings**
Related Requirements:	ST-16, ST-17, ST-18
Initiating Actor:	League Manager
Actor's Goal:	Change league settings to the League Mangers preference
Participating Actors:	Database, other Investors
Preconditions:	-Initiating actor is the **League Manager**
	-There are outstanding abuse reports
Postconditions:	-The **Database** is updated to reflect the chagnes made.
	The abuse report shows that it has been resolved on the administration page
	Flow of Events for Main Success Scenario:
→	1. **League Manager** makes adjustment to league settings
←	2. **System** makes a update to the **Database**
→	3. **System** displays updated settings to the **league manager**

Traceability Matrix

The traceability matrix presented in *Figure 3.2* is based on only the full dressed use cases above and thus is only a partial representation of the complete project.

Requirements	Priority Weight	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7
ST-1	10	X						
ST-2	10	X						
ST-3	10		X					
ST-4	6			X				
ST-5	6			X				
ST-6	10					X		
ST-7	10							
ST-8	8		X		X			
ST-9	4				X			
ST-10	6			X	X			
ST-11	3			X		X		
ST-12	2				X			
ST-13	1				X			
ST-14	3				X			
ST-15	1							
ST-16	8		X					X
ST-17	8		X					X
ST-18	4		X					X
ST-19	2						X	
ST-20	3						X	
ST-21	9						X	
Total Priority		20	38	21	24	13	14	20

Figure 3.1: The traceability matrix for the given use cases.

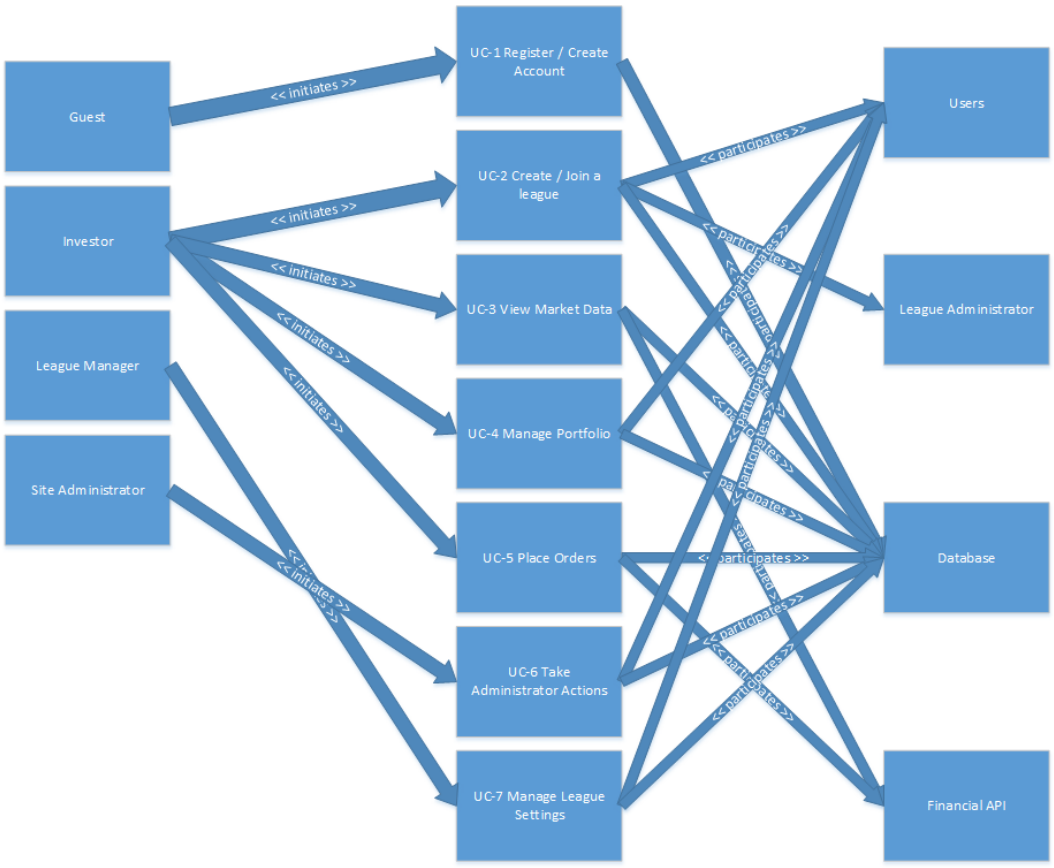


Figure 3.2: This graphic illustrates the relationships between the core actors of our platform.

3.4 System Sequence Diagrams

In the following sequence diagrams, we describe exactly the interactions between the key actors our system. It is important to note that most of the interaction between the user and system is facilitated by the browser. The user, through filling forms and button clicks, instructs the browser which requests to make to the system. In turn, the system communicates with the database to request the desired data, takes any required actions, and delivers the data to the browser for presentation to the user.

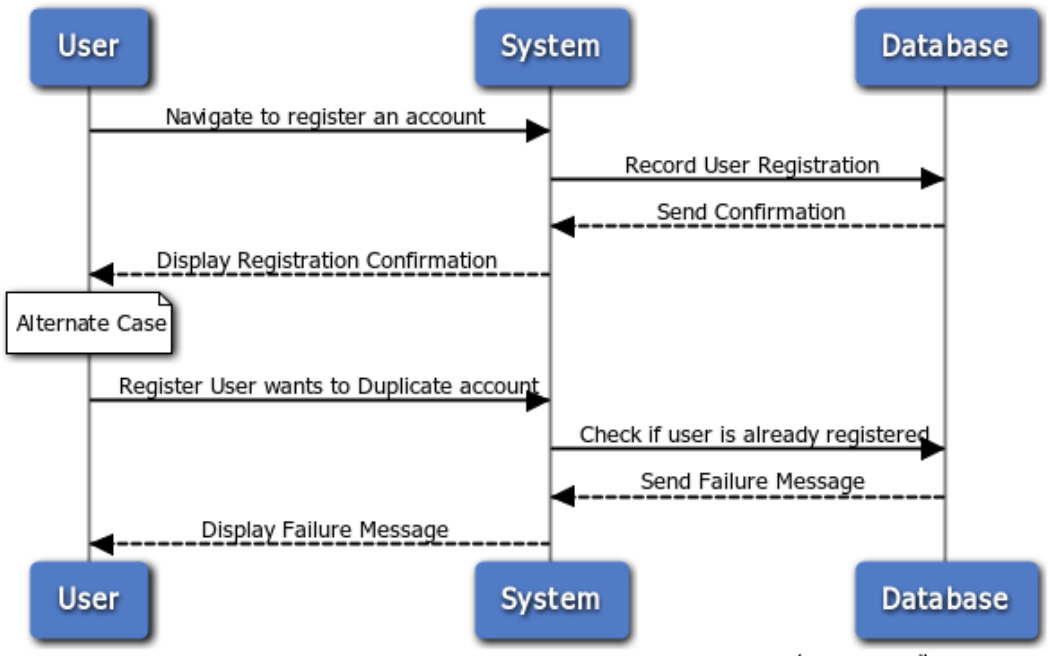


Figure 3.3: See UC-1 on page 20. When the user navigates to the login/register accounts page, this use case is triggered. The system makes it necessary to have an account before using Paramount Investments leagues. The System then takes the information that the user has input and sends them to the database, which then sends it back to the system to display on the screen to the user. If the system finds that the user that is registering with the same credentials as an existing account, the system throws up an error appropriately.

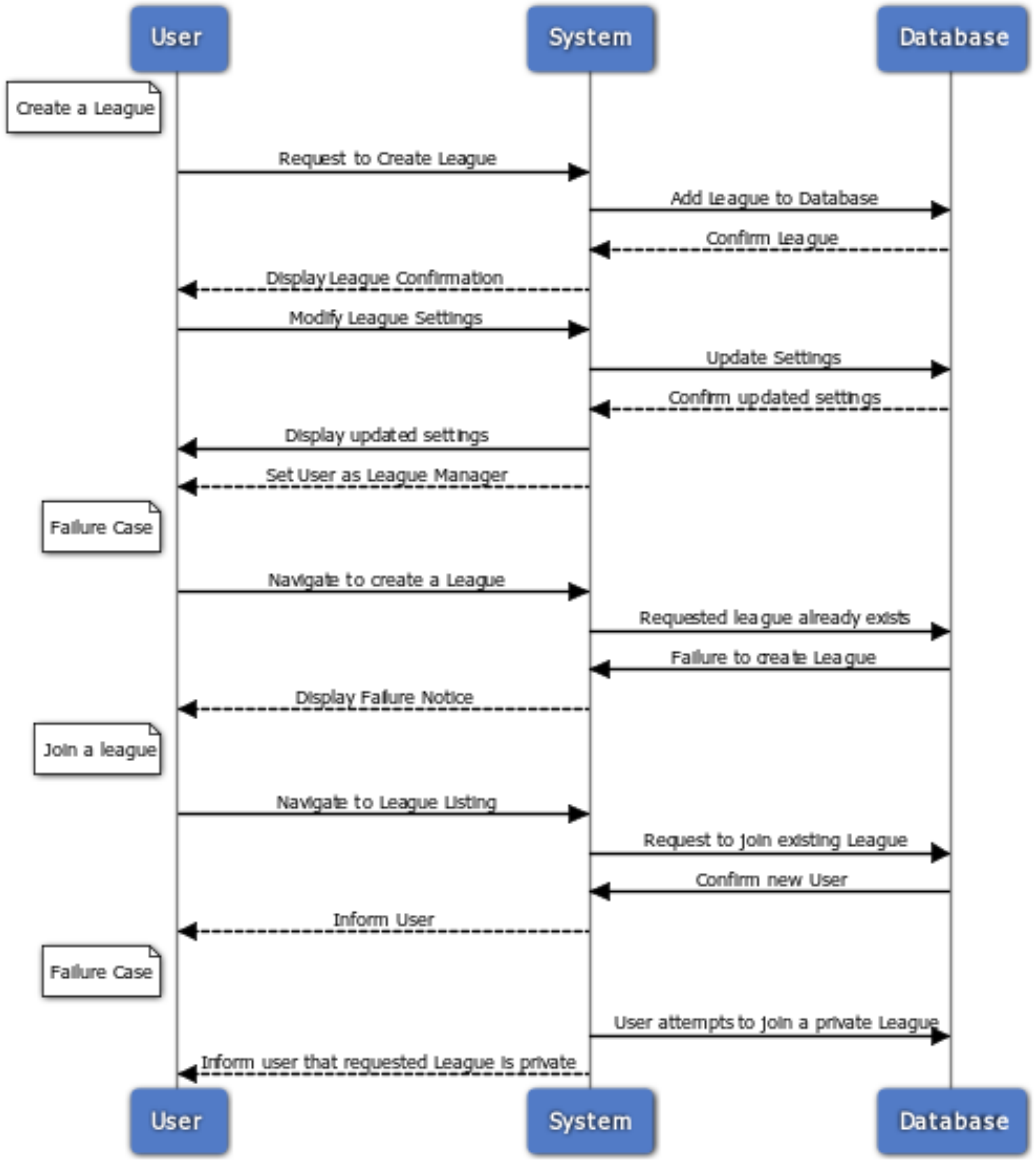


Figure 3.4: See UC-2 on page 21. This use case is triggered when the user navigates to the create league page. The user requests the system to create a league, which then sends appropriate data to the database. Once the data is stored successfully, the database sends a confirmation back to the system, which then displays an appropriate message to the user. If the user wants to join a league, the user requests the system appropriately, which then sends the data to the database regarding the right league and if successful sends the confirmation to the user.

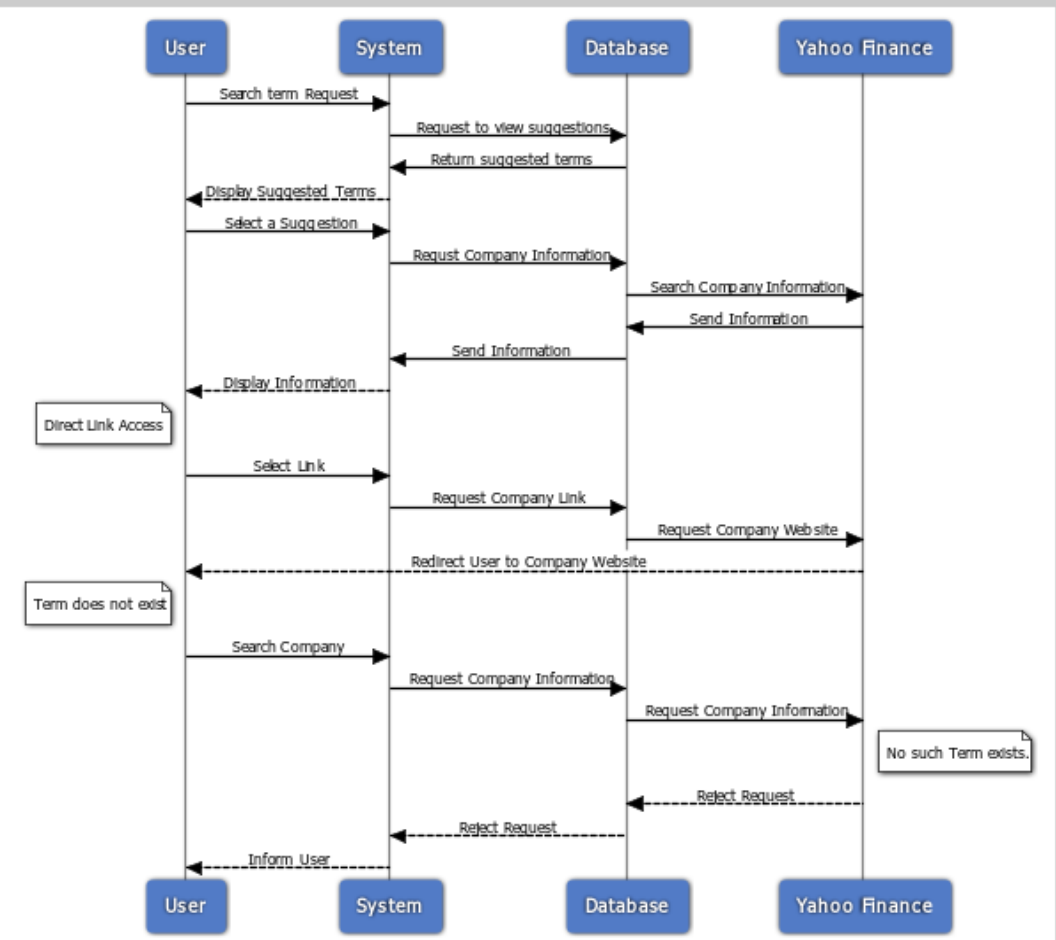


Figure 3.5: See UC-3 on page 46. When the user navigates to the research stock page, this use case is triggered. The user specifies to the system exactly what market data they would like to view. If the user wants to research off the companys website, then user will click on the hyperlink present in the system. If the user wants to view the data through the interface that Paramount Investment Provides, the system then pulls information from the Yahoo Finance API and then sends it back to the system to display to the user.

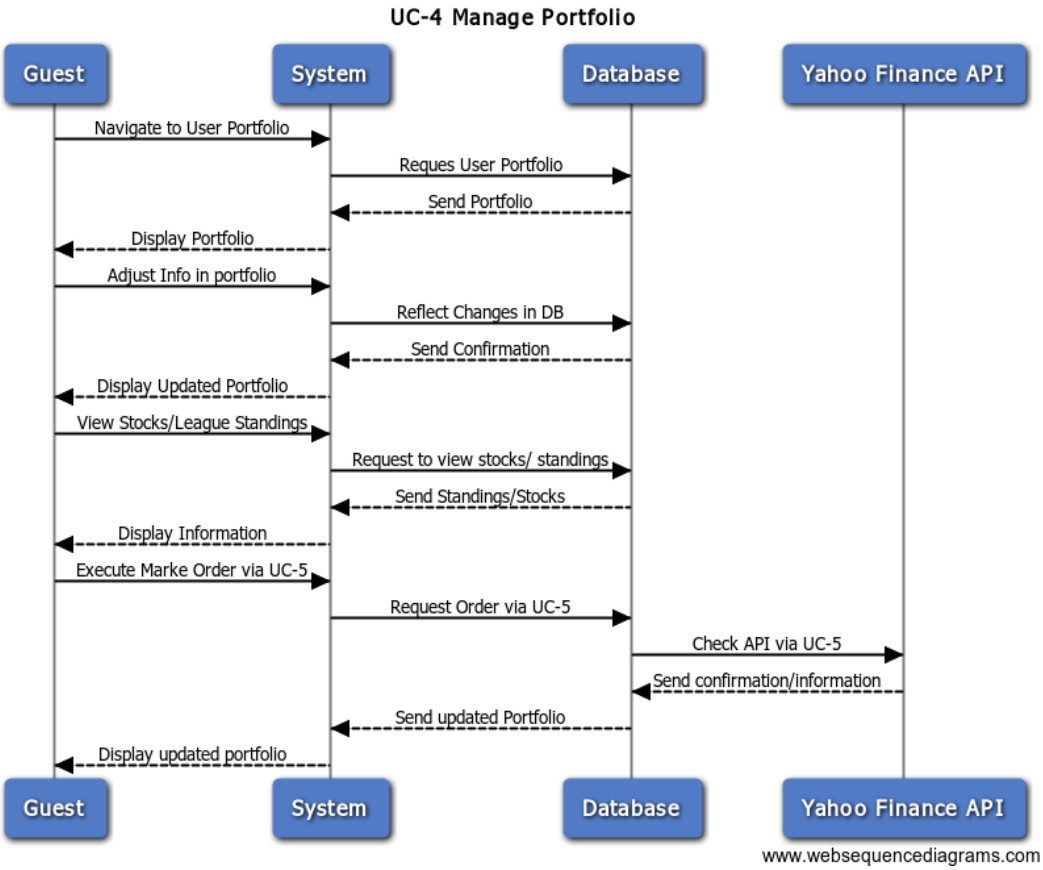


Figure 3.6: See UC-4 on page 23. This use case is triggered when the user goes to his/her own portfolio page. The user navigates to the view portfolio page. The system then requests the database to retrieve the users information. Once the database sends this data back to the system, the system displays it to the user. The user is now free to modify aspects of his/her portfolio. Once the user is finished modifying/updating their portfolio, the system will send the changes to the database, which will then store the information and save it.

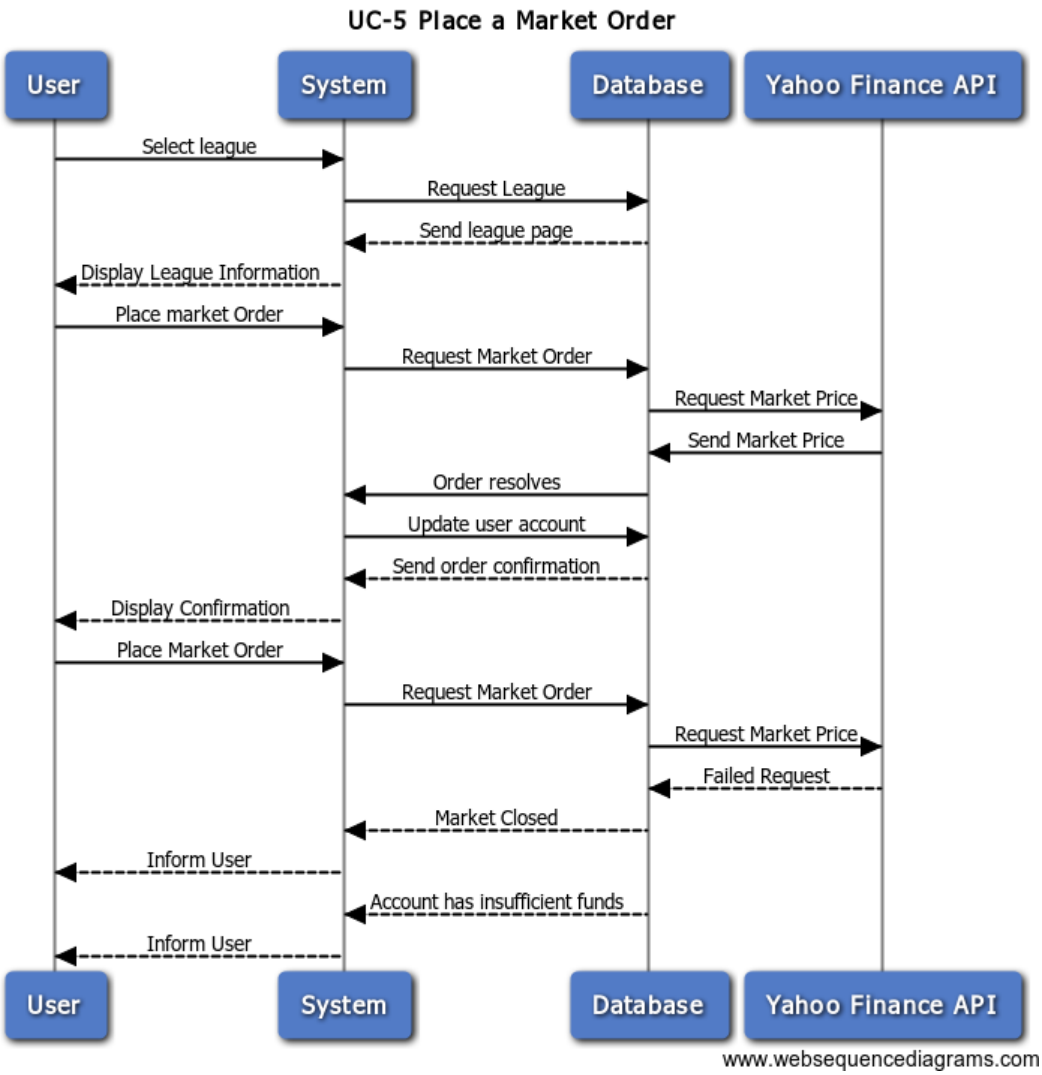


Figure 3.7: See UC-5 on page 23. This use case is triggered when the user goes to place a market order in the place order page. The user selects a league to place the order into. The system then displays the information to the user, who then requests to place a market order. The system then queries to Yahoo Finance API to retrieve information about the stock prices. The system then takes that information and processes it with what the user wants to do. If the order is completed successfully, it is appropriately displayed.

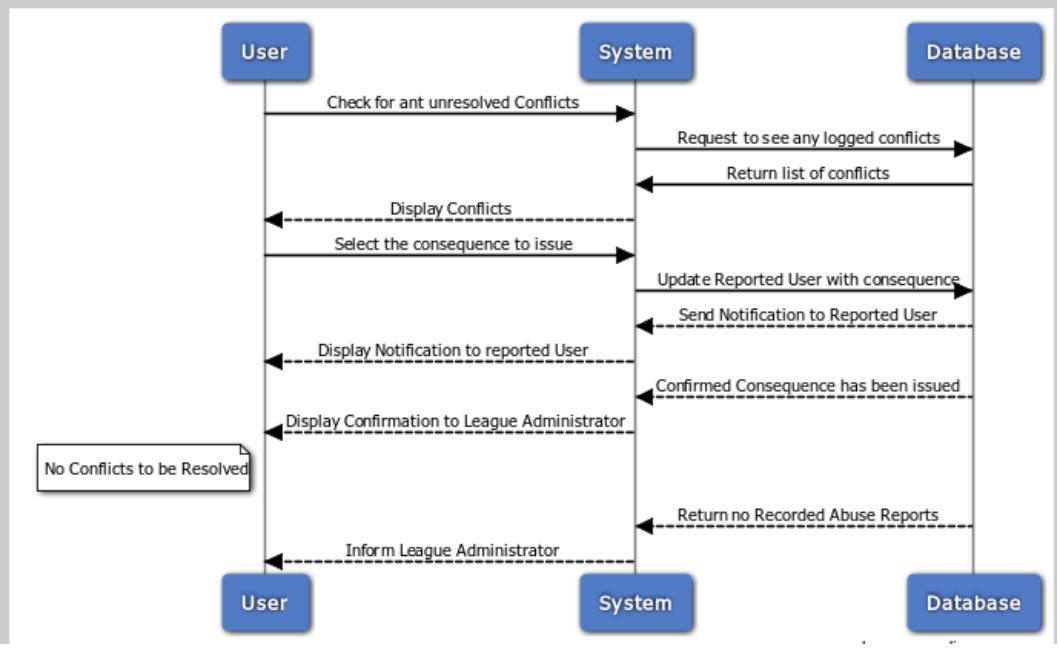


Figure 3.8: See UC-6 on page 24. This use case is triggered when the administrator of the website/ league wants to take action. The system first checks if the user logging in has administrative privileges in their respective group. If so, the system then looks in the database to check for any logged conflicts. If there are unresolved conflicts, the database returns them and then user can then view the conflicts. If there are no conflicts to be resolved, then display so appropriately to the administrator/logged in user.

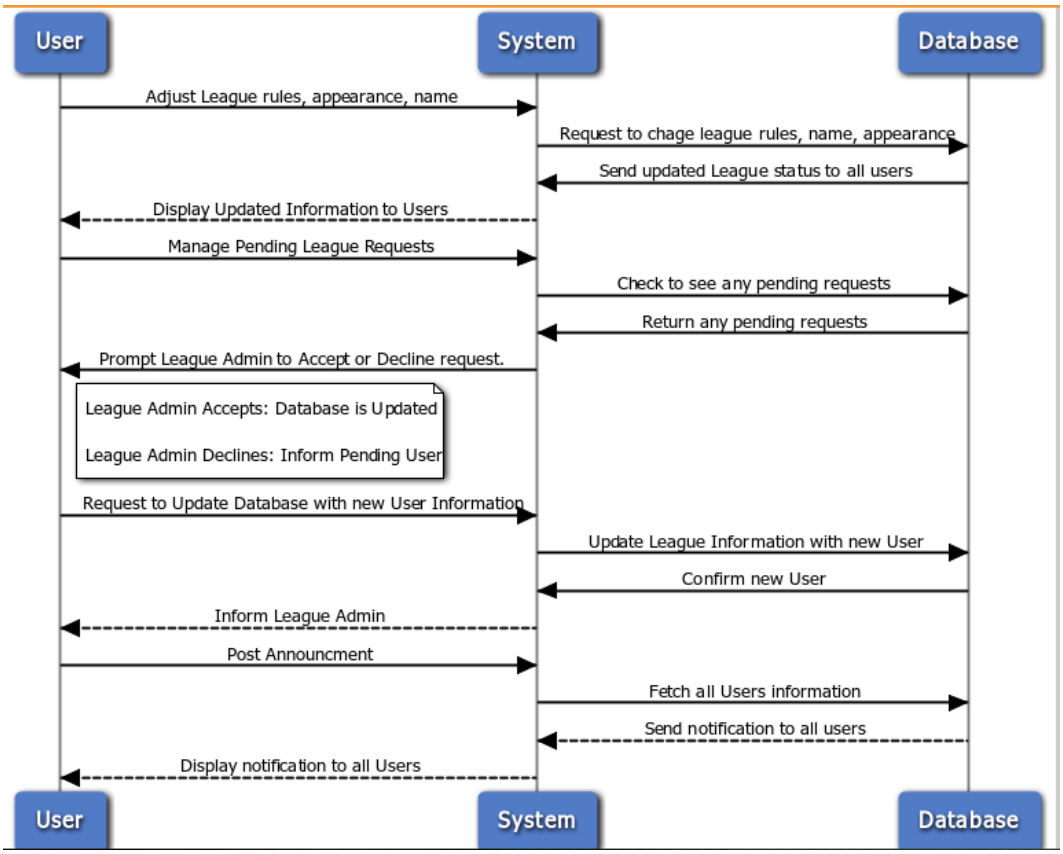


Figure 3.9: See UC-7 on page 25. This use case is triggered when the manager of a league wants to change the league settings. The system first checks to see if the user that is logged in, is the league manager and if so, the grants the user privilege to change league settings. The user then requests to change the league settings. The system retrieves the league data and modifies it as the user has requested.

4 User Interface Specification

4.1 Preliminary Design

The user interface (UI) for Paramount Invesments Leagues will act as a command center for users to interact with their portfolio, leagues they are a part of, and conduct research on potential orders. More specifically, the command center will act as the primary; but not the only; view for users to interact with the system. The command center will provide a snapshot of the users current portfolio and its value, their global rank, a dash to perform market orders, a news feed, and a graphing dash in order to quick analysis of stock performance. The UI will persist a users global rank across all views as well as a ticker of current trades being placed through the Paramount Investment League.

The UI should be lightweight so as not to burden our more restrictive target platforms of mobile and tablet. The colorscheme will be chosen to be easy on the viewer, though this is subjective, the colorscheme will be a basic pallet of grey/black/white/blue, tending toward pastel and web supported colors.

The UI will be built on top of Twitter's open source Bootstrap CSS[6] framework to help facilitate deleriving content to the three target platforms, desktop, mobile, and tablet. Bootstrap provides a mobile first design philosophy, but can be customized to target specific platforms.

Landing Page and Login

Paramount Invesment League is designed around allowing users to easily begin using the service, also know as "zero effort" resgistration. In order to accomplish this, the system does not require the user to register a new user name/account with our system, but instead piggybacks on OpenID[7] and OAuth[8] allowing users to use their Google, Facebook, Twitter, and other OpenID/OAuth accounts to login. You'll also notice that upon initial visit, the header is empty providing no navigation, this may be relaxed in the future to allow the user to explore some of the features of the website that don't require user authentication such as stock research. (*See figure 4.1*)

Global Header

The header (*see Figure 4.2*)across the website will remain persistant across the website once the user is logged into the system. Navigation is done between essentially 4 views in the following order, My Portfolio, Stock, League, Leaderboard,. These names are placeholders and will most likely be

Figure 4.1: First iteration of Landing/Login page.

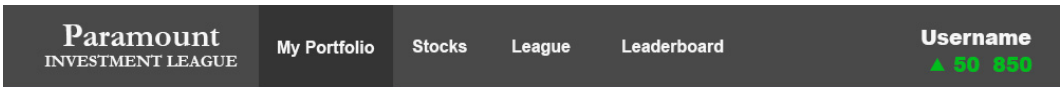


Figure 4.2: Preliminary design for a global header. This users is up 50 spots for the day.

My Portfolio, My Leauges, Leaderboards, Analyze Assets. The 'My Leagues' and 'Leaderboards' will be turned into drop downs as users expand into leagues to allow quick navigation.

The website name will also navigate to My Portfolio. The username will be replaced by the users actual username, and below it will be the users global rank. The rank will be highlighted in red or green depending on whether they have improved their position on the day, or it has declined. It will also indicate how many spots they have moved.

Global Ticker

One interesting feature of Paramount Investment Leagues will be its active ticker at the bottom of the website. This ticker will be seen in all views, including the Landing Page once there is enough volume to keep the ticker full. The ticker serves two goals, one for new users, and one for existing users. The first goal is to entice new users to participate by demonstrating that the app is being widely used. The second goal is to give a snapshot to existing users of assets that are "on

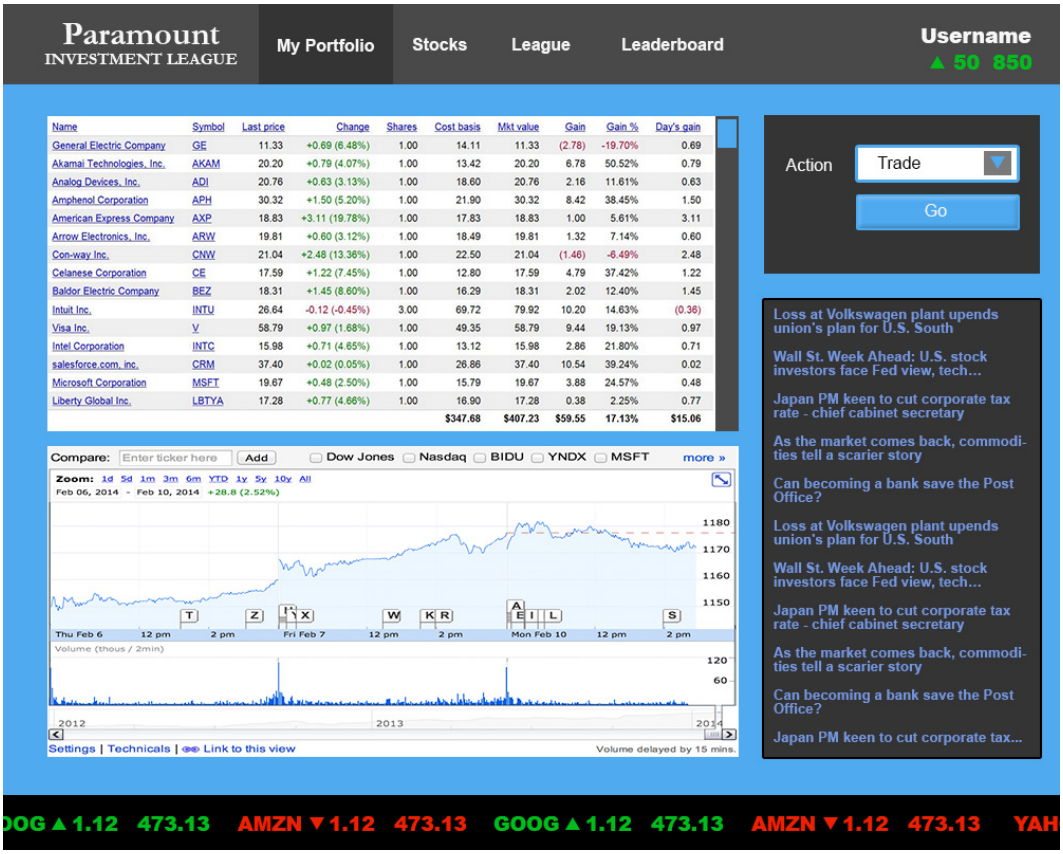


Figure 4.3: The preliminary design of the 'My Portfolio' view.

the move" so that they can attempt to remain competetive. The ticker can be seen at the bottom of all the figures.

My Portfolio

The 'My Portfolio' (*see Figure 4.3*) view of the website will act as the command center for a user wanting to get news about companies/assets in their portfolio, perform an order, or conduct quick graphical anaylsis of assets in their portfolio and compare them to any other asset available for trade through the platform.

More importantly, it provides a snapshot of the users portfolio including a scrollable list of all the assets inside the portfolio and a summary of said assets. In the future, assets will be 'clickable' and will take the user to a summary page of that asset, but that is not planned for the initial 2 iterations.

Leagues

The 'League' (*see Figure 4.4*) view will present a user that isn't a part of a league the ability to create a new league of join an existing league. Not shown in *Figure 4.4* is the view that a user

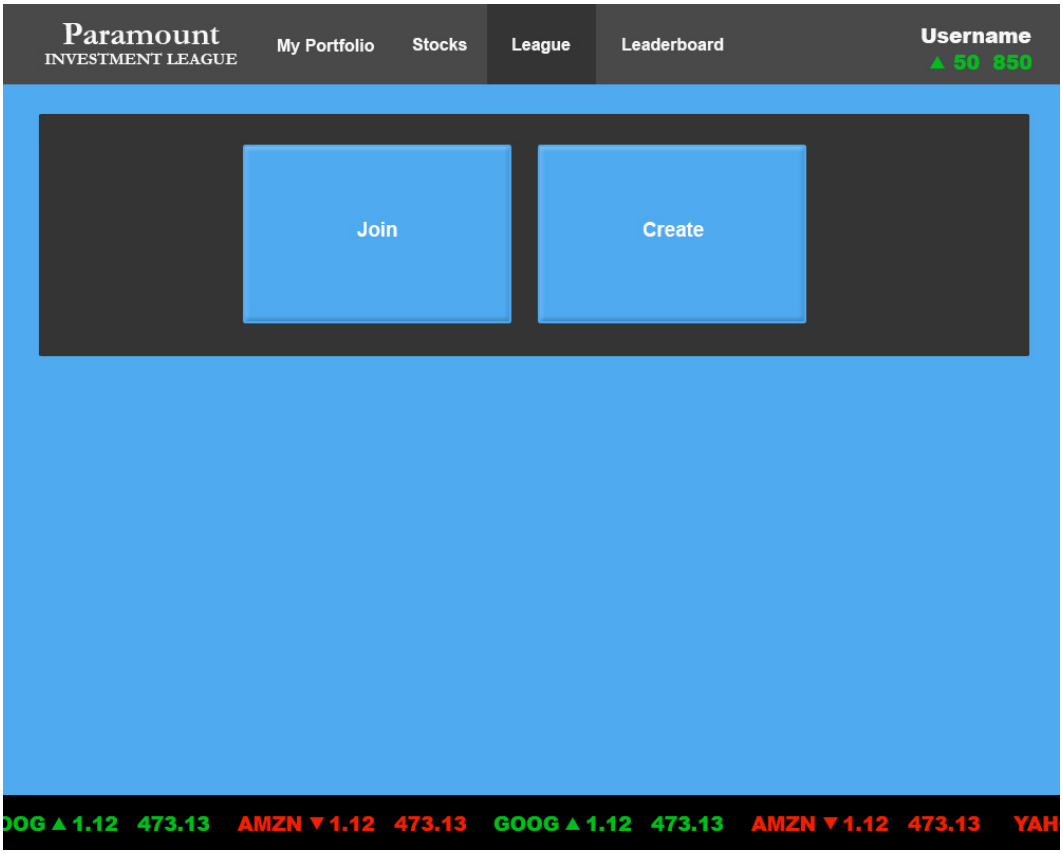


Figure 4.4: This is the league creation/join view. This would be the view presented to a user that is a part of no league yet.

who is a part of a league. This view will still persist the join/create dialogues, but will also present a list of all the leagues that user is a part of, their rank within said league, and their movement within said league.

Leaderboards

The 'Leaderboards' (*see Figure 4.5*) view will present the user with a partial view of the full leaderboard for a given league, or for every user. It will show their rank, their movement, the value of their portfolio as well as the same stats for all other users around them. The view will be scrollable if there are more records then can be displayed, and will center the user in the middle of the view unless they are at the top or bottom of the board.

Asset Analysis

The 'Stocks' view (*see Figure 4.5*) will be renamed to more align its function with its name, which is to analyze assets. It will a more in depth way of anaylzing an asset versus what is available in the 'My Portfolio' view. There will be a news feed at the bottom of assets that you are searching for. There will also be a more formal analysis of asset data presented including P/E ratio, 52 week

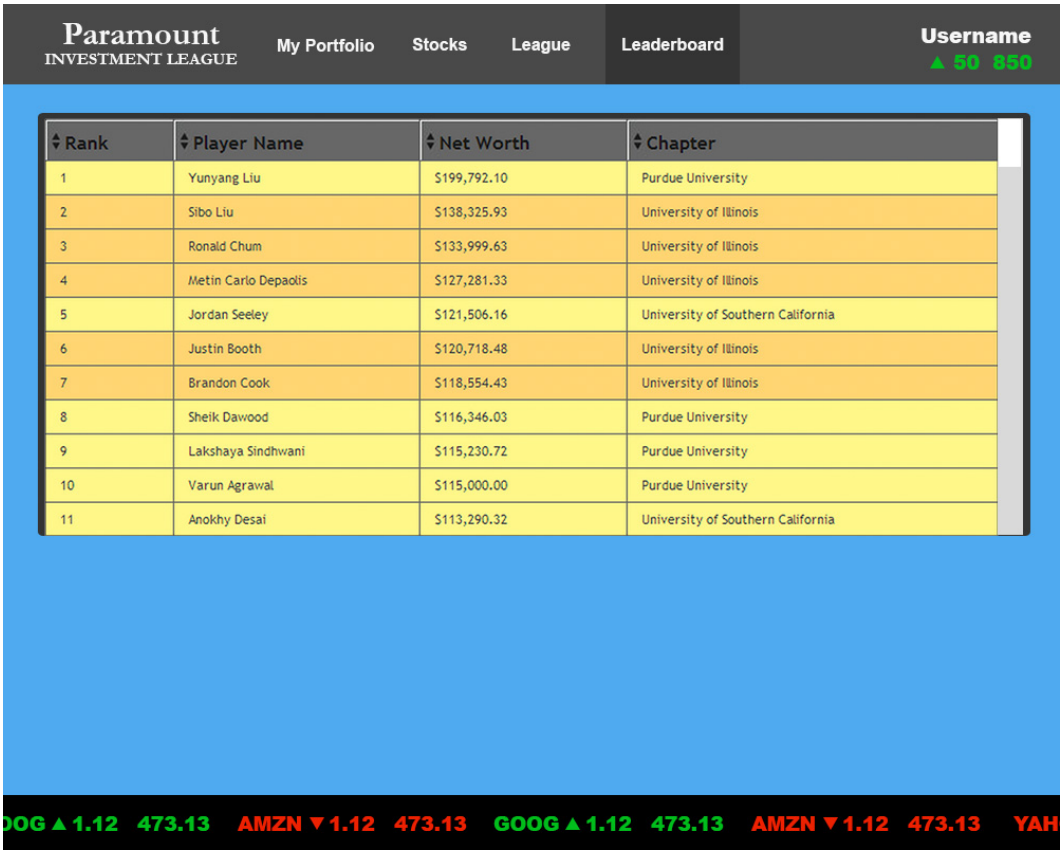


Figure 4.5: Here is the leaderboard view which will be the same for both leagues and global leaderboards. This view represents a global leader board. The colorscheme of this view here is incomplete and will fall inline with the remainder of the site.

range, Volume, EPS, etc. This isn't shown in the figure, but will one-half to two-thirds of the space that has been set aside for the news feed.

This is also one of the views and functionalities that has been identified to not require the user to be logged in. While it will not be availble to non-users in the intial product, it can be made available in future releases.

4.2 User Effort Estimation

Several of the most common usage scenarios for Paramount Investment Leagues:

Figure 4.6: The preliminary view for asset anaylsis.

Usage Scenario	Clicks	Keystrokes
Login & Register	2-3	0-1
Place an Order	4-6	2-12
Join a League	3-4	0-50
Create a new League	6-7	11-100
Analyze Asset	2	2-5
View Leaderboard	2	0

Login & Register

Assume the user has come to the domain and wishes to Login if already registered, or register if already a user:

- **Navigation:**
    1. Click on OAuth provider icon (Google, Facebook, Twitter, etc).
    2. Click on your account (optional for multiaccounts).
    3. Click on login, or hit enter.

Place an Order

Assume the user has already logged in and they wish to place an order:

- **Navigation:**
    1. Navigate to 'My Portfolio', 0-1 clicks.
- **Data Entry:**
    1. Select order type from drop down, 2 clicks
    2. Click textbox to enter asset name. 1 click
    3. Enter assets name eg: 'G', 'O', 'O', 'G', 1-4 keystrokes
    4. Press tab to specify number of shares, 1 keystroke (user could also execute 1 click)
    5. Enter the number of shares, 1-7 keystrokes
    6. Click execute, 1 click

Join a League

Assume that the user wishes to join a league and is logged in:

- **Navigation:**
    1. Click on League, 1 click
    2. Click on Join, 1 click
- **Data Entry:**
    1. Click on a League, or enter its name, 1 click or up to 50 keystrokes
    2. Click on confirmation dialogue, 1 click

Create a League

Assume that the user wishes to create a league and is logged in:

- **Navigation:**
    1. Click on League, 1 click
    2. Click on Create, 1 click

- **Data Entry:**
    1. Enter its name, 1-50 keystrokes
    2. Select ruleset from dropdown, 2 clicks
    3. Fill in parameters, 1-2 clicks and 10-50 keystrokes
    4. Click on confirmation dialogue, 1 click

Analyze an Asset

Assume that the user is logged in and they want to start an in depth analysis of an asset:

- **Navigation:**
    1. Click on Stock, 1 click
- **Data Entry:**
    1. Click on the textbox for entering an asset name, 1 click
    2. Enter an asset name, 1-4 keystrokes
    3. Hit enter, 1 keystroke

View Leaderboard

Assume that the user has logged in and wants to veiw a leaderboard:

- **Navigation:**
    1. Click on Leaderboard, 1 click
    2. Click on Select Legue/Global, 1 click

5 Domain Model

5.1 Concept Definitions

The Domain Model Concepts are derived from responsibilities contained in the Use Cases from Chapter 3.

Responsibility	Type	Concept
R1:Allow new user to create an account to partake in stock trading game as an Investor. And Login existing user.	D	Account Controller
R2: Initialize accounts with fixed amount of capital.	D	Account Controller
R3: Check if Investor is in a league or not.	K	League Controller
R4: Update new leagues at set interval and display to correct Investors.	K	League Controller
R5: Retrieve information about Stocks, Trades, and Companies.	K	Yahoo! Finanace Adapter
R6: Display information about users Stocks, Trades and Leagues	K	Player Profile View
R7: Record and Execute Buy/Sell/Short Stock trades	D	Order System Controller
R8: Display Welcome screen to create an account or log-in.	K	Login View

Domain Diagram

The shown figure (our domain model) illustrates how our system is broken down into different subsystems and how they interact with each other. Each box represents an entity in our system and each arrow represents a function that allows the entity to communicate with other entries in the system.

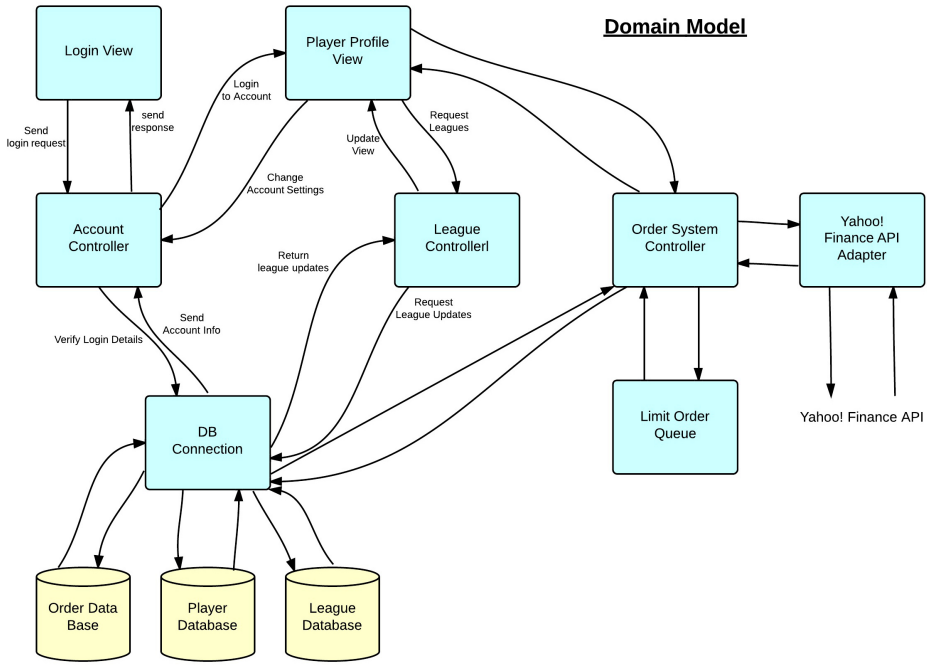


Figure 5.1: The domain model.

Account Controller

The first step for anyone using this system is to gain access by creating an Account. Account Creator is an interface that allows a player to create a new Investor account. It will check with the database for an account with the same details, and if not found it will proceed to create the account and store the details into the database.

Player Profile View

The Player Profile displays statistics and saved settings for the user that logs into The Paramount Investments League. Player Profile should query Yahoo! Finance Adaptor API to get data about Watchlist stocks, and any searched Target stock.

Login View

The Login View displays a UI to allow the user to login with OpenID. This will send a request to the account controller. If it fails the Login View will be updated to reflect this. If this succeeds

the user will now see the Player Profile View.

League Controller

League Manager will proceed to keep up to date display of rankings of every player in each league. These will be fetched from a database and displayed in the Player Profile. League Manager will allow Investors to join open leagues that they are not already in. This will require querying the database and updating if necessary. League manager may give our achievements based on certain accomplishments within the leagues.

Yahoo! Finance API Adapter

Yahoo! Finance API provides the almost real time stock data that our application is dependent on. Any downtime Yahoo! Finance experiences will affect our application. Yahoo! Finance API Adaptor serves as a translation between the CSV file that Yahoo! Finance produces through their API, and our application. Our Adaptor will take the spreadsheet Yahoo! Produces and convert the data into syntax that our application can understand. This adaptor is modular in order to allow multiple subsystems to make queries for live stock updates.

Order System Controller

Any order placed by an investor will go through the Order System which will sum the cost of the transaction and check that the account balance is satisfied. This will require communication with the Yahoo! Finance API Adaptor to get the current price of the target stock to be purchased/sold. If the order is a limit trade, we must define a way to check if the current price of the stock matches the limit price and execute the order.

Database Connection

We want to have a single subsystem maintain control of accessing the database to make retrieving information modular. This will allow expansion of the application to add additional functionality later, which may also need access to the database. This should provide a layer of security so each subsystem does not access the database directly. Additionally, the single subsystem will give us an interface for interacting with the database, without worrying about the underlying implementation of the database. In the long run, this will allow us to insert data to a different database (Ex. NoSQL) without spending time to refactor the code. Using a database connection will also help to increase security within our system by helping to prevent against attacks, such as SQL injections.

5.2 Association Definitions

Concept Pair	Association description	Association name
Login View <> Account Controller	Login View sends a login request. Account Controller can respond with Success or Failure.	sends
Account Controller <> DB Connection	Account Controller sends user login details. DB Connection sends account info or failure.	sends
Account Controller <> Player Profile View	Account Controller updates view to be the Player Profile View. (Profile View may also allow user to change some account settings).	updates
Player Profile View <> League Controller	User may send request to League Controller to update, join, or leave a league. League controller can update view with information.	sends
League Controller <> DB Connection	League controller request league statistics from DB Connection. DB Connection sends statistics	sends
Player Profile View <> Order System Controller	Player Profile View sends requests to the Order System Controller. Order System Controller updates the Profile View.	sends,
Order System Controller <> Yahoo! Finance API Adaptor	Order System Controller sends requests to API Adaptor about stock prices. API Adaptor returns information about the stock info.	sends
Order System Controller <> Limit Order Queue	Order System Controller creates the limit order queue when a limit order is placed but cannot be executed immediately.	creates

The associations of domain concepts are derived from the table above. The Account Controller takes information that a user enters into the Login View. This information is sent to check with the Database. The Account Controller can change the view to the Player Profile based on what information is stored in the database. From the Player Profile View a user can access account settings, leagues, and portfolio details. The league details are managed by the League Controller, which can allow requests to create/join a certain league. The league controller should also periodically update the Player Profile Views statistics of the leagues they are in. The Order System Controller allows the user to search market data, and attempt buy/sell/short trades. The Order System Controller must communicate to see if the user has sufficient funds. Limit trade may not be executed until the stock price reaches a certain value. If the limit trade cannot be executed immediately a Limit Order Queue is created and these orders will be placed in here. The Order System Controller communicates with the Yahoo! Finance API Adaptor to retrieve current quotes for stocks. Yahoo! Finance Servers must be online for this to work correctly.

5.3 Attributes Definitions

Responsibility	Attribute	Concept
R9: Know if user login failed	LoginFailed	Login View
R10: Player's name and OpenID	Name/OpenID	Player Profile View
R11: Players Account Balances (Cash Balance, Money Invested, Daily Change).	Account Summary	Player Profile View
R12: Stocks User added to WatchList.	WatchList	Player Profile View
R13: Stocks User owns.	Owned Stocks	Player Profile View
R14: Leagues and Rankings	leagueID/rank	Player Profile View
R15: Know if user is logged in	isLoggedIn	Account Controller
R16: Know which leagues user is in.	isInLeague	League Controller
R17: Know if user is creating a league	isCreatingLeague	League Controller

At the Login View it is good to know if a Login attempt failed so we can display the correct information to the user. When a Login is successful the Account Controller will set the isLoggedIn attribute, then interacts with the DB Connection to retrieve all information to populate the Player Profile Views attributes such as Name, OpenID, Account Summary, Owned Stocks, LeagueIDs, and rankings.

At the Player Profile View the user may attempt a variation of things such as changing account settings, joining/creating a league, making an order, or viewing market data. The League Controller has attributes isInLeague so it knows how to update data for the Profile view. It also has attribute isCreatingLeague, to submit this data through the Database Connection after a successful league creation.

The Player Profile View can also interact with the Order System Controller to view market data and place an order. The Order System Controller will use isValidValue to check if the account balance is sufficient to perform the operation. isValidStock checks to see if the symbol entered is a valid inquiry.

Yahoo! Finance API Adaptor communicates between the Order System Controller and the Yahoo Finance Server to deliver information about a particular stock. retrieveData is the attribute to search a stock value and then return it to the Order System Controller to perform a useful operation on it.

Database Connection has a retrieveData attribute to be able to read data from the database. There is also a writeData attribute to be able to store all information about the Players of the game.

5.4 Traceability Matrix

		DOMAIN CONCEPTS						
Use Case	PW	Account Controller	League Controller	Order System Controller	Login View	Player Profile View	Yahoo! API Adaptor	DB Connection
UC1	20	X			X			X
UC2	38	X	X			X		X
UC3	21			X		X	X	
UC4	24	X		X		X	X	X
UC5	13	X		X		X	X	X
UC6	14	X	X	X		X		X
UC7	20	X	X	X		X		X
Max PW		38	38	24	20	38	24	38
Total PW		129	72	92	20	130	58	129

Figure 5.2: The traceability matrix.

5.5 System Operation Contracts

UC-1 Register/Create an Account

- *Preconditions*
  - (join) If a new user is visiting the Paramount Investments League website (guest), they must first register with either OpenID/OAuth2 account before joining/creating a league with Paramounts Investments.
- *Postconditions*
  - After registration, the database is updated and logs the once previous guest, as an investor of Paramount Investments League.

UC-2 Create/Join League

- *Preconditions*
  - Investor must be logged into the Paramount Investments League website.
  - No more than one instance of the same League name can exist.
  - User hasnt joined a league yet.
- *Postconditions*
  - Investor has joined a league
  - Database has been updated
  - League has been set with selected settings.

UC-3 View Market Data

- *Preconditions*
  - Investor is logged in
  - Yahoo Finance is accepting inquiries.
- *Postconditions*
  - Query Stock Market Data

UC-4 Manage Portfolio

- *Preconditions*
  - User is logged into their Paramount Investments League account.
  - Yahoo Finance is accepting inquiries.
- *Postconditions*
  - Any adjustments made to the investors portfolio have been updated in the database.

UC-5 Place a Market Order

- *Preconditions*
  - User is logged into their Paramount Investments League account.
  - Investor has enough funds in their account to place a market order
  - Yahoo Finance is accepting inquiries.
- *Postconditions*
  - User profile is reflected with any change to funds or position.
  - Database has been updated with these changes.

UC-6 Take Administrative Actions

- *Preconditions*
  - User is the site administrator
  - An issue/conflict occurs and needs to be resolved.
  - There are outstanding abuse reports.
- *Postconditions*
  - Conflicts/Issues have been resolved
  - The reported user has been notified of any actions taken against them.

UC-7 Manage League Setting

- *Preconditions*
  - Initiating actor is the league manager.
  - League Manager is logged into their Paramount Investments League account.
- *Postconditions*
  - Database is updated to reflect any changes made to their account.
  - All users are notified of any changes made in their league.

5.6 Economic and Mathematical Models

Perfect Competition

One of the prevalent concepts in the stock market is the economic concept of perfect competition, which says that not any single participant has enough resources/power to control the market. To apply the concept of perfect competition to our project we will need the following requirements:

- Not one person can control the market or industries, segment, etc.
- Users can feel free to execute trades at their convenience without having to worry about extra costs
- Every individual has access to same stock information as other investors

- The selling price is the same as the buying price.

In the real world, none of these requirements can be met, as there is always some problem that prevents the market from being in perfect competition. The following are just some of the problems:

- There are high net worth individuals/companies who have enough capital to change the tide of a certain sector of the market. If one of these individuals suddenly decides to leave a particular market, the move may suddenly shift the market and effect other investors in that market.
- In the real world, users typically dont have direct access to stocks. They have a broker (electronic or human) who they interact with, who then have direct access to stocks. Users cant usually execute trades/buy stocks without worrying about extra costs because of the commissions charged by brokers when trading stocks.
- The world is not a fair place, and neither is the stock market. There are individuals who because of the field that they work in, have much more insight into a particular industry/stock. These individuals then sell this information to potential buyers in hopes that it gives them an edge in trading. This gives a huge disadvantage to those that dont have access to more information bout stocks.
- Lastly, in the real world, the selling price is never usually the same as the bid price. The Bid-Ask spread, the difference between the buying and selling price tends to be greater than 0.

All these factors lead the stock market away from perfect competition.

How do we plan to fix these issues to ensure a near-perfect competition?

- All investors start with the same amount of money, this way no one person by default has more power than anyone else
- No commission will be charged when the trades are executed for any investor
- Insider trading will be avoided by standardizing the stock information across the board
- The ask-bid spread will be 0, so the selling price is the same as the buying price

Mathematical Model:

- Stock Prices
  - There are no complicated mathematical models behind how the stock prices are determined in our platform. The market prices that are retrieved from Yahoo Finance are the prices that are available to users in Paramount Investments
- Achievements
  - Achievements in Paramount Investments each have their own mathematical model. There are no complicated algorithms behind how these achievements are attained. If the user has met the required conditions for a certain achievement, then they will be given that specific award.

- For example: Buy stocks whose P/E Ratio > 1

6 System Interaction Diagrams

6.1 Introduction

The interaction diagrams in the following section will outline the system interactions in the most important parts of our software. For each particular use case, we will outline the interactions among the systems and databases. Further, we will analyze multiple cases in which the systems will handle different scenarios. That is, it will show how the system handles both failure and success conditions. In the following scenarios, you will see the database, controller, and Yahoo! Finance API used in nearly every situation. Because this is a web-based and data based application, the database and controller become heavily prevalent. Users will need to log in and constantly access data pulled from the Yahoo! Finance API to have constantly refreshed and updated information. The diagrams below will accurately detail how this will be accomplished within the system.

6.2 Diagrams

Use Case 1

Shown in the sequence diagram for UC-1 begins with two options for the Guest. Either login or register an account. If a user attempts to register a new account the Account Controller is contacted with the users information. Then the Account Controller can attempt to check to make sure no duplicate login information exists in the database via the DB Connection module and if not it will store the new user information into the database. After this happens the user will be sent a confirmation email. Then the Account Controller will update the Login View.

If a user attempts to login, the Account Controller will attempt to authenticate the login details with details found in the database via the DB Connection module. If the details match correctly then the Account Controller will send the guest into investor mode and therefore displaying them the Player Profile View.

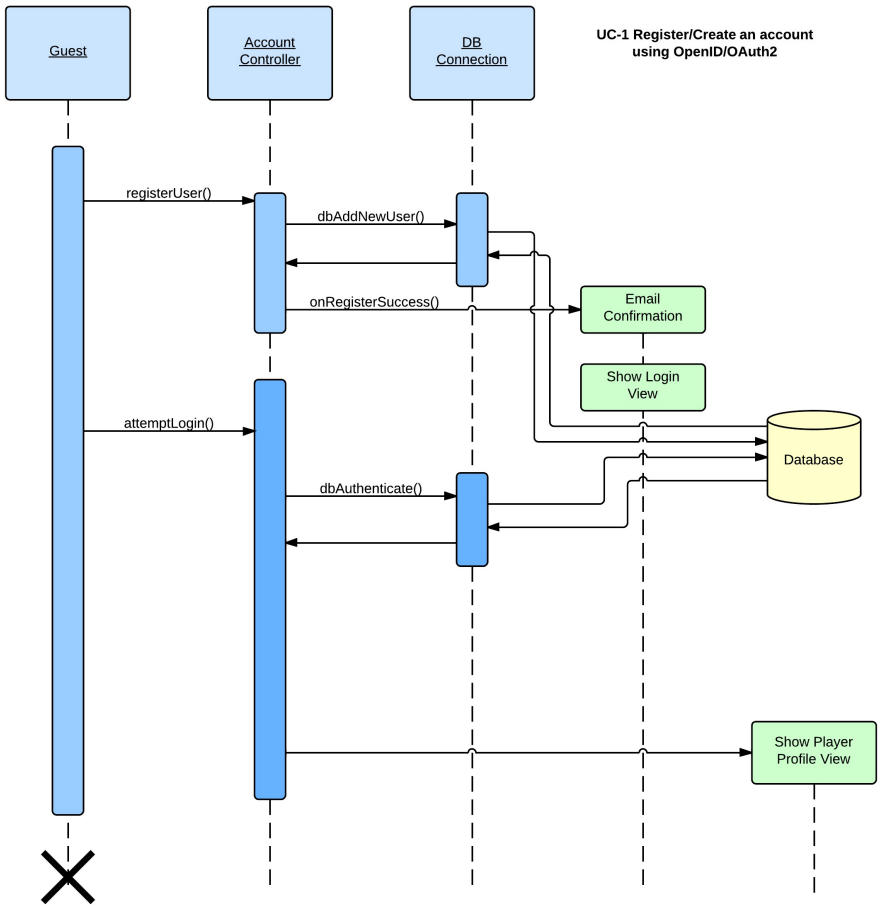


Figure 6.1: UC-1

Use Case 2

Shown in the sequence diagram for UC-2 is the flow of how to create an investment league. When an investor selects to create a league the League Controller will be contacted. This will update the Player Profile View display the available options for creating a league. After, there is a function updateSettings() which will create the league and process it in the database via the DB Connection and also allow settings to be updated for a league. Not shown in the diagram is

the alternative case of joining a league. The process to join a league is straightforward, where the league controller will show available leagues and then if an investor chooses to join they will be entered into the list in the database to associate with this league.

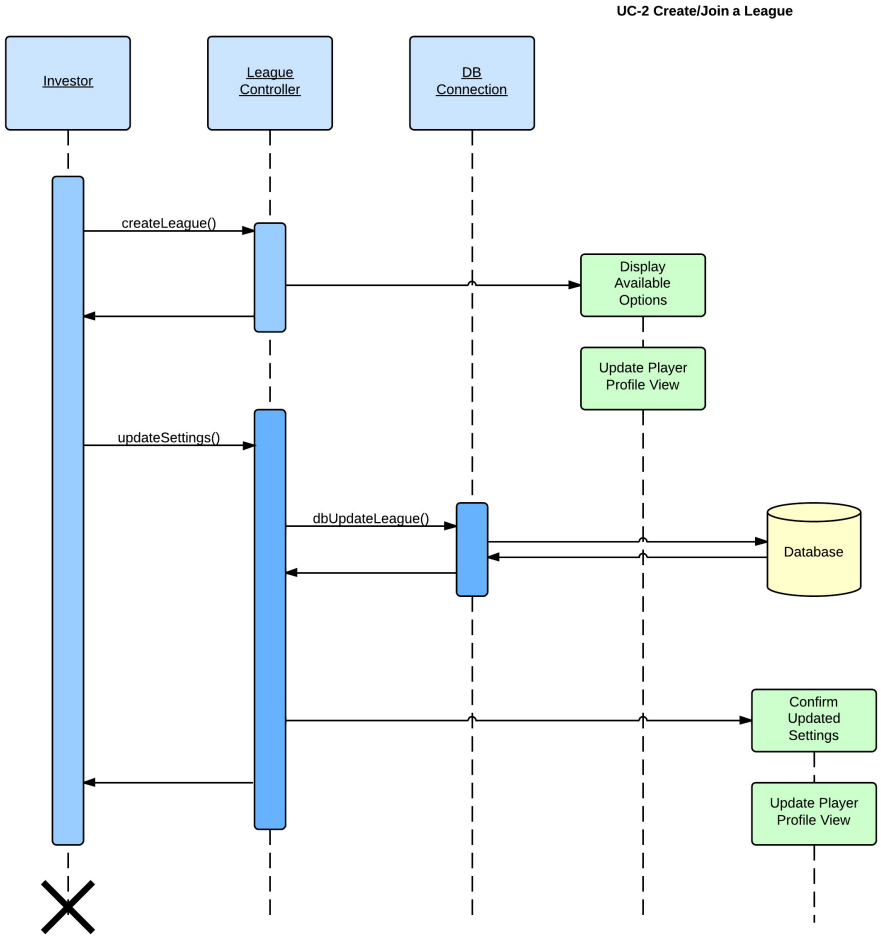


Figure 6.2: UC-2

Use Case 3

Viewing market data is accomplished by an investor searching a term. The Order System Controller then finds this term which is most likely a company name or stock symbol. The system will fetch matches from the database via the DB Connection module and display them from the user. The investor will choose a match. The Order System Controller takes the chosen term and requests its data from the Yahoo! Finance API via the Yahoo! Finance Adapter. The Order System Controller will update the database via the DB Connection module for this term, and then continue to show the Market Data View.

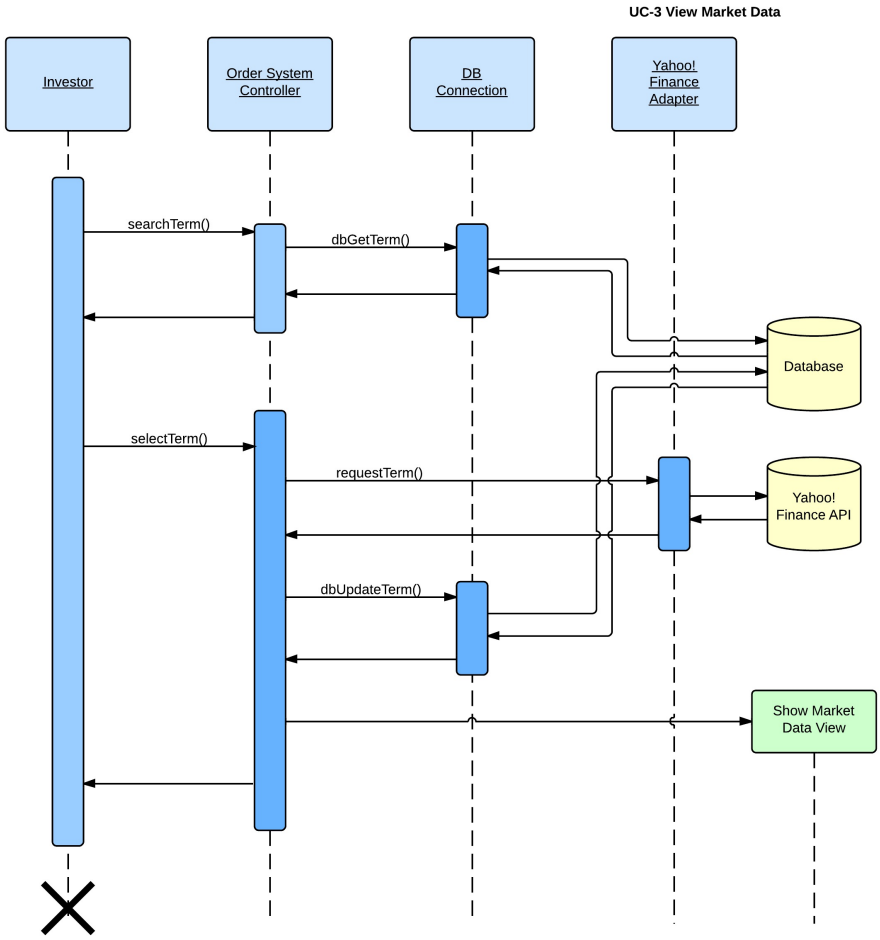


Figure 6.3: UC-3

Use Case 4

The investor should be able to view and make changes to their Portfolio View. When the user clicks to show portfolio, the Portfolio Controller will fetch the investors portfolio stocks from the database via the DB Connection module. The investor can also update their view of the portfolio and other settings.

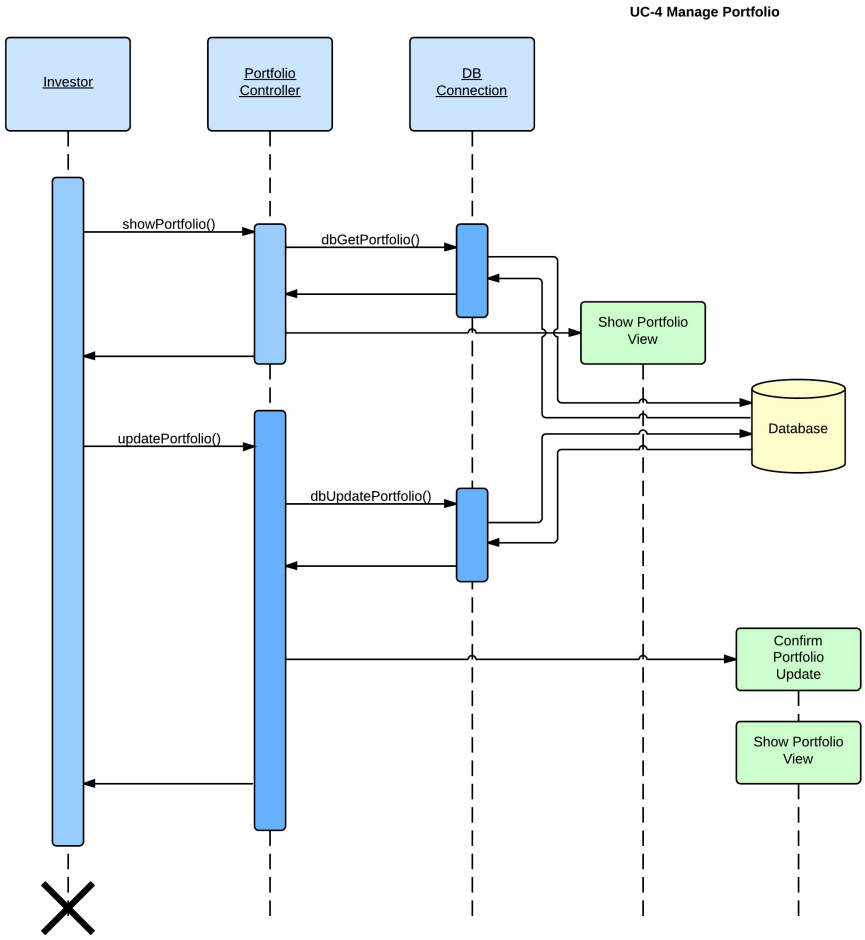


Figure 6.4: UC-4

Use Case 5

The investor needs to be able to place market orders. As soon as the investor places an order the Order System Controller contacts Yahoo! Finance API via the Yahoo! Finance Adapter to retrieve the current price of the stock. After the current price is found the Order System Controller must confirm with the database via the DB Connection module that the user has enough funds to make a buy order or enough stock to make the sell order. After the trade is confirmed information

will be stored about it in the database via the DB Connection module and the changes will be displayed in the investors portfolio.

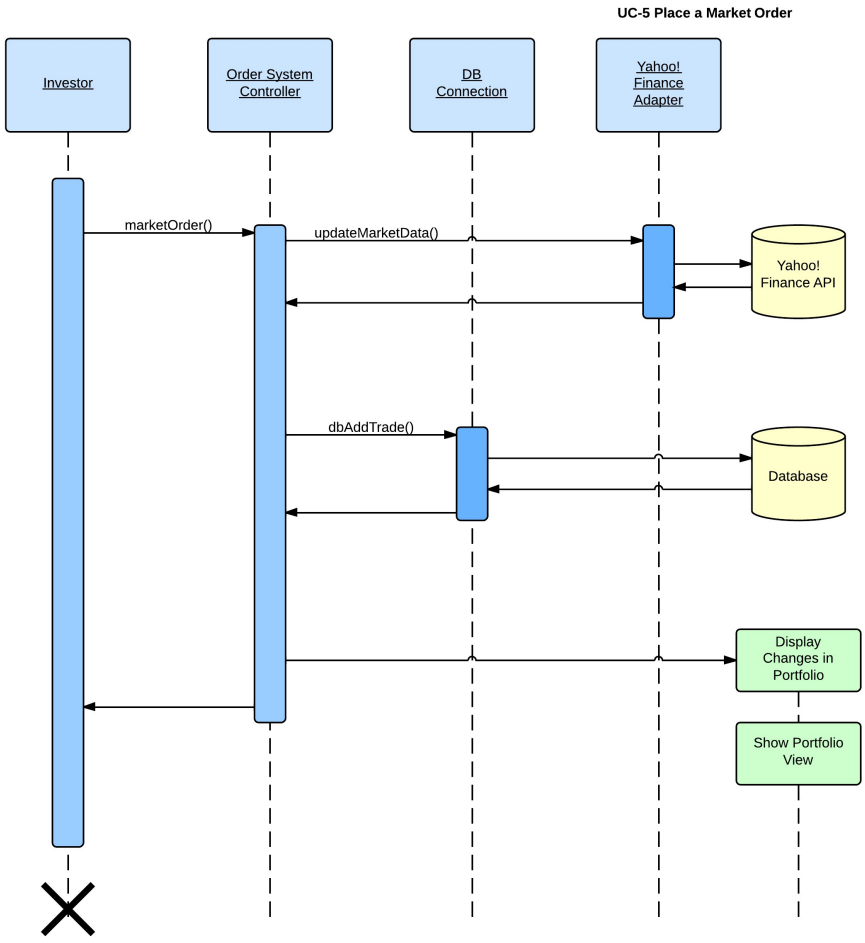


Figure 6.5: UC-5

6.3 Alternate Solution Diagramming

Software design shouldn't be about picking your first idea and going with it. You need to consider alternative solutions to the task at hand and pick the best one based on the known criteria. For this reason we are documenting some of our alternative solutions for historical reference.

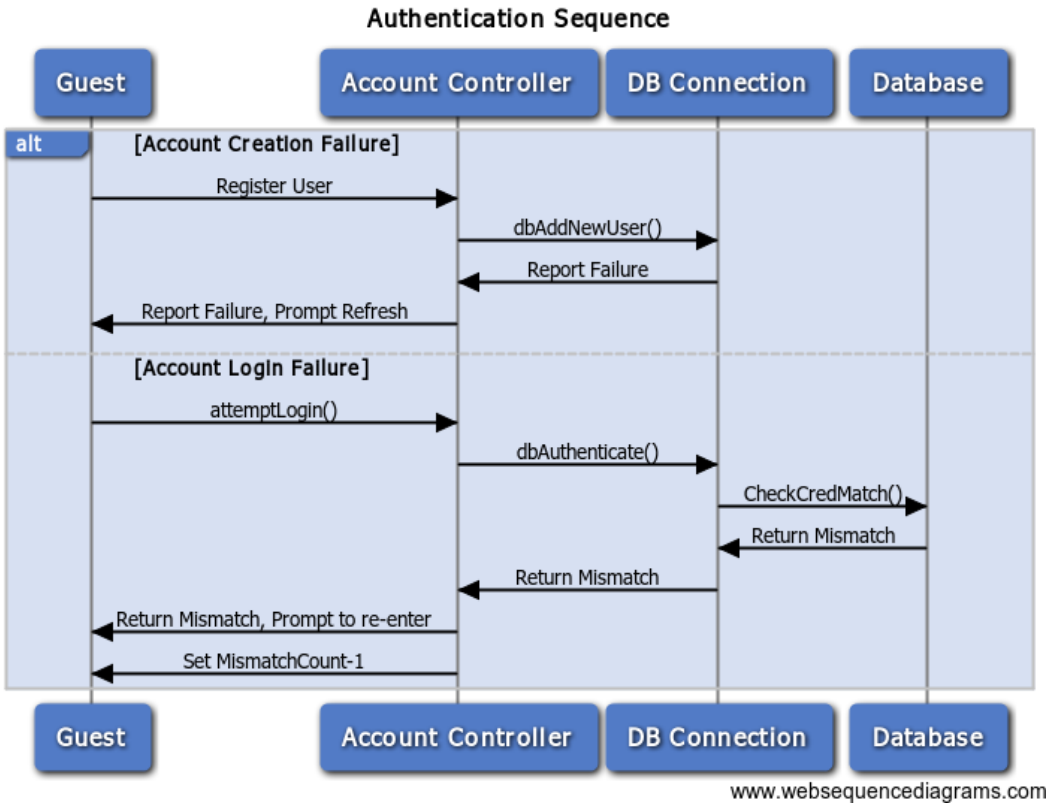


Figure 6.6: UC-1 alternate solution considered

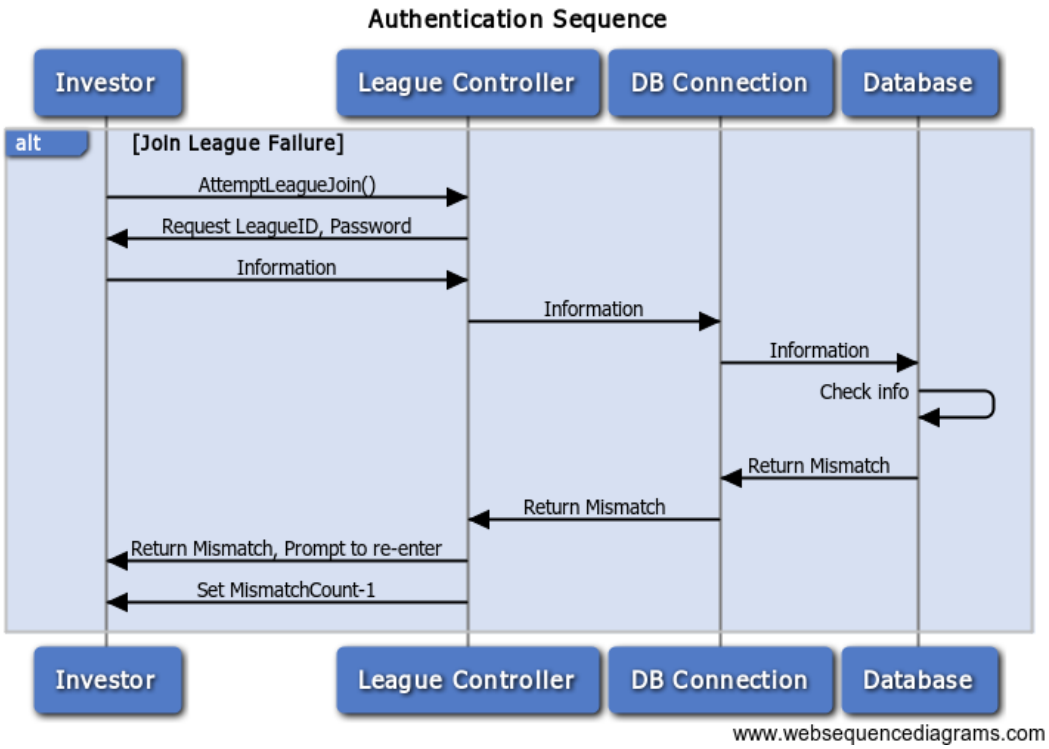


Figure 6.7: UC-2 alternate solution considered

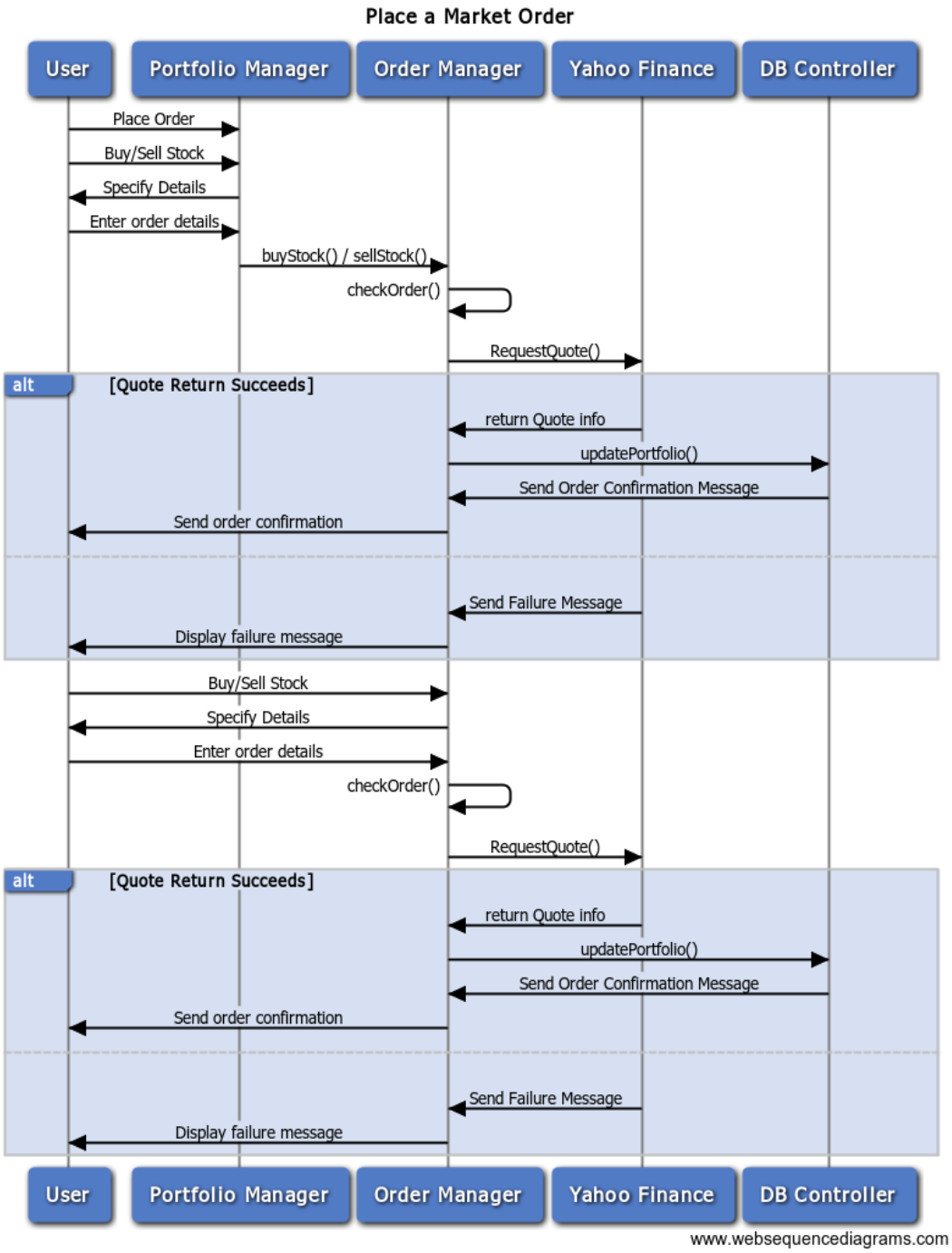


Figure 6.8: UC-3 alternate solution considered

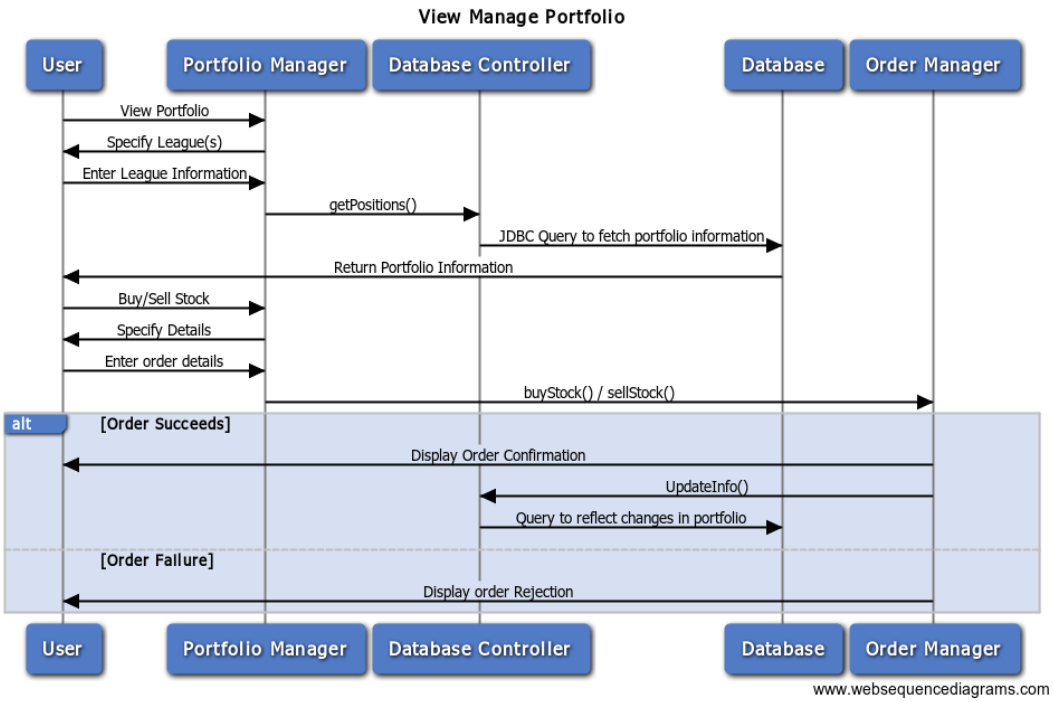


Figure 6.9: UC-4 alternate solution considered

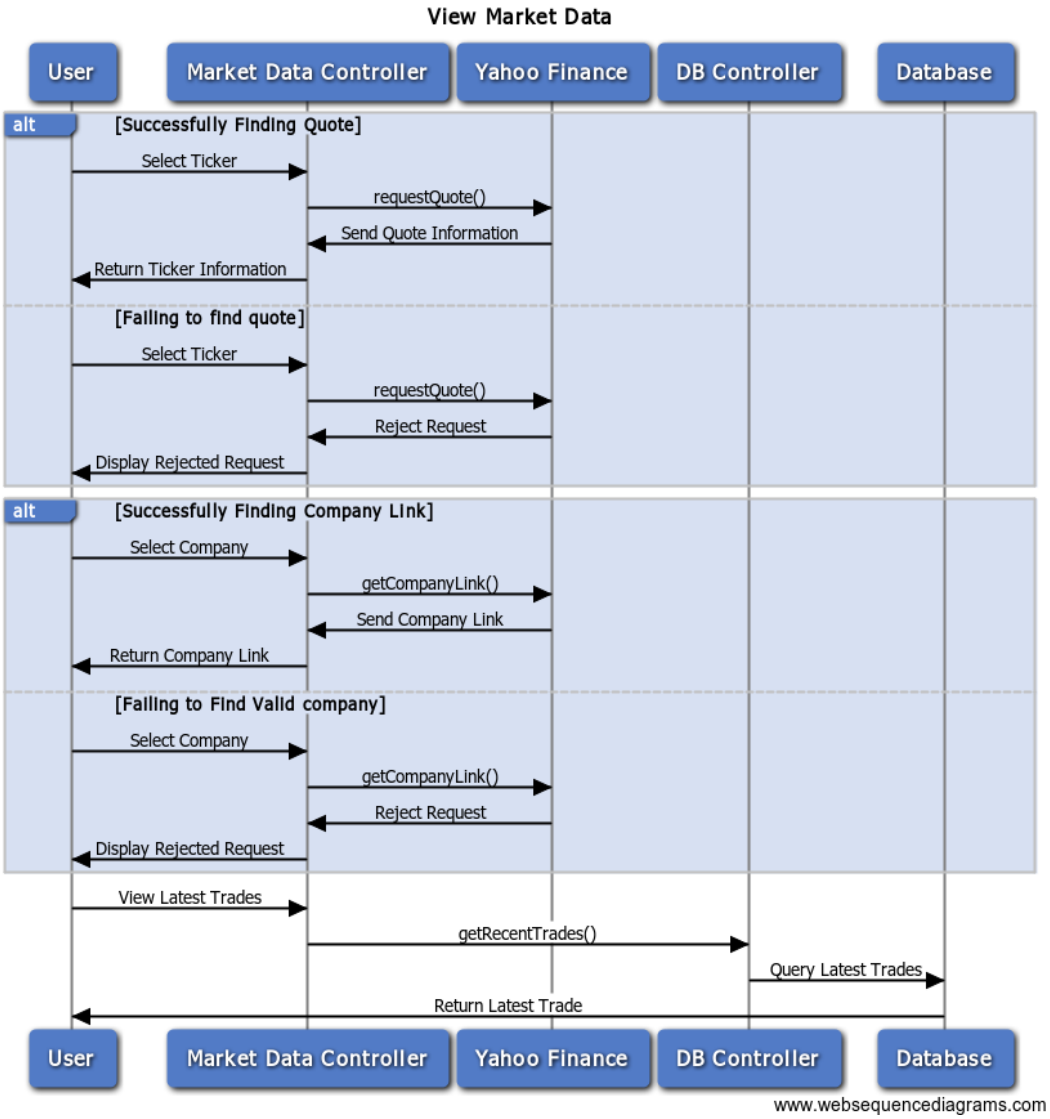
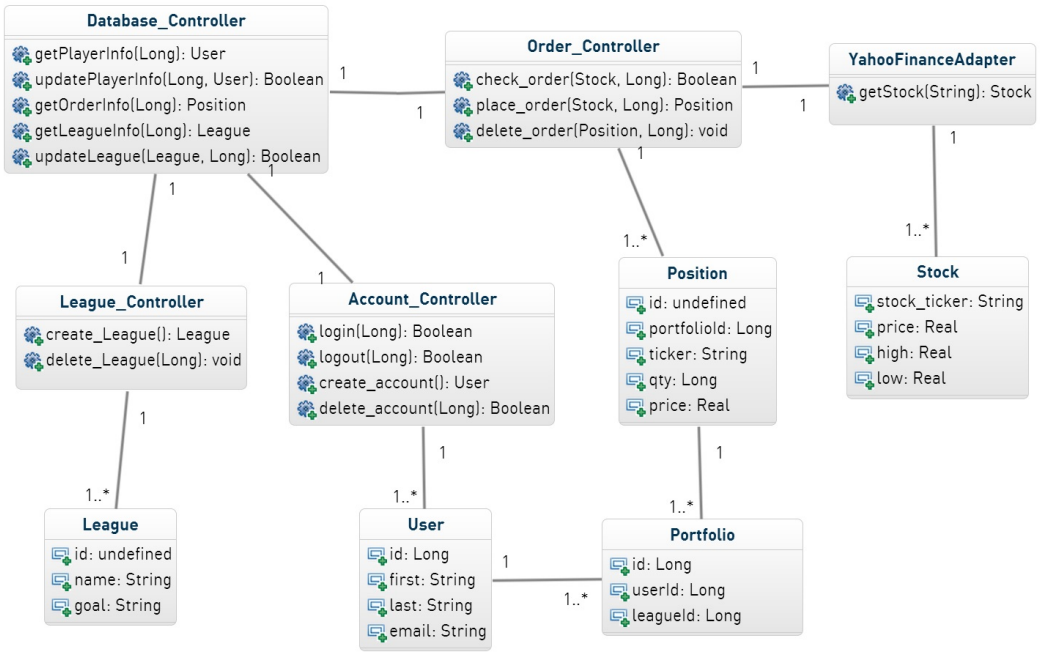


Figure 6.10: UC-5 alternate solution considered

7 Class Diagrams and Interface Specifications

7.1 Class Diagram



7.2 Class Data Types and Operation Signatures

Database Manager

Our database manager performs the function of managing the database. This can mean anything from adding user information into the database, retrieving information from the database and updating information in the database, regardless of whether the information deals with users/accounts, leagues, orders.

Methods

+ get_player_info(in user_id : long) : class User
This method is used when information needs to be retrieved for a specifc player

+ update_player_info(in user_id : int, in upd: class user) : bool
This method is used to update a users information, whether it be administrative or game related.

+ get_order_info(in transaction_id : int) : class transaction
This method takes in a transaction id and returns the information associated with that specific transaction.

+ update_league(in league_id : int, in leagueInfo : class league) : bool
This method returns the latest updates in the league

+ return_league_updates(in league_id : int) : class league
This method is used when a league needs to be updated with the newest information provided in the league in the input

Order Manager

Our order manager class is responsible for handling all the tasks related to orders/transactions. It is responsible for placing the order in the system and for moving old orders to the archive transactions table.

Methods

+ Check_order(in symbols: class Order) : bool
This method is simple used to check and make sure that the input order can be processed. It will check the users balance, etc.

+ place_order(in symbols: class Order) : bool
As the name suggests, this method is used to place an order in the system, with the information given by the input Stock class.

+ delete_order(in symbols: transaction_id): bool
As the name suggests, this method deletes an order from the system, assuming that it hasnt already

been processed. If it has then this function will return a false value.

+ Execute_order(in transaction_id : int) : bool
This method is responsible for actually getting the stock information from Yahoo Finance API and then changing the account/portfolios to reflect it accordingly.

League Manager

This class is responsible for managing all the leagues in the system. It has the authority to create leagues, delete leagues, and modify leagues as it is instruction to do so.

Methods

+ Create_league () : Class league
This function is used to create a league from scratch so that the user can create a league.

+ Delete_leagues(in league_id : int) : bool
As the name suggests, this method will delete the league matching the input

+ change_league_name(in league_id : int) : bool
This method is here solely for the purpose implied by its name. Its only function is to change the name of the league.

+ Change_league_manager (in league_id : int, in usr : class User) : bool
This function replaces the current league manager stored in the input league with the user specified in the input.

+ add_rules(in league_id : int) : bool
This function is here for the reason its name suggests. It is here just to add rules to a given league.

+ Delete_rules(in league_id : int) : bool
This method exists just to delete the rules in a league.

Account_Controller

This class exists to take care of any function that relates to accounts. This can mean creating an account, modifying an account, or even deleting

Methods

+ Login(in user_id : int) : bool
This function is used by the user to log into the system

+ logout(in user_id : int) : bool
This function is the opposite to the one above it, it is used by the User to log out of the system.

+ Verify_User(in User_id : int) : bool
Method to make sure that the person logging in or that the person who is logged in is not an imposter/fake.

+ Create_account() : class User
Creates an account with the current user

+ delete_account(in suser_id : int) : bool
Used to delete an account from the database system.

Yahoo Finanace Adapter

This class is responsible for obtaining market data from Yahoo Finance API. It consists of 3 functions to get quotes, get company information, and to get sector information.

Methods

+ Get_quote(in stock_ticker_id : string) : class quote
As the name suggests, this method is responsible for obtaining quote information about a given stock_ticker

+ get_company_info(in stock_ticker_id : string) : class Company
This method is responsible for getting market information about a specified company.

Stock

This class is responsible for representing a Stock. It has the authority to hold a ticker symbol, price, daily high price, and daily low price.

Attributes

+ String:stock_ticker
This attribute holds the ticker symbol as a character array.
+ double:price
This attribute holds the current value of stock corresponding to the ticker symbol.
+ double:high
This attribute holds the current High price of the stock on the market.
+ double:low
This attribute holds the current Low price of the stock on the market.

User

This class is responsible for representing a User. It has the authority to hold first name, last name, email address, and userId.

Attributes

+ long:id
A unique id to distinguish different users from one another.
+ String:first
This attribute holds the first name of the user.
+ String:last
This attribute holds the last name of the user.
+ double:low
This attribute holds the email address of the user.

Position

This class is responsible for representing a Position, which is a stock that a user owns. It has the authority to hold the user Id, portfolio Id, ticker symbol, quantity and price.

Attributes

+ long:id
A unique id to distinguish different users from one another.
+ long:portfolioId
A unique id to distinguish different users from one another.
+ String:ticker
This attribute holds the first name of the user.
+ long:qty
This attribute holds the quantity of the certain position.
+ double:price
This attribute holds the price that the position was purchased at.

Portfolio

This class is responsible for representing a Portfolio, which is the set of stocks that a user owns. It has the authority to hold the user Id, portfolio Id, ticker symbol, quantity and price.

Attributes

+ long:id
A unique ID to distinguish different portfolios from one another.
+ long:userId
A unique ID to distinguish who owns the portfolio.
+ String:leagueId
A unique ID to distinguish what league this portfolio is a part of.

League

This class is responsible for representing a League, which is a group that users can belong to, to compete with each other.

Attributes

+ long:id
A unique ID to distinguish different portfolios from one another.
+ String:name
This attribute holds the name of the league.
+ String:goal
This attribute holds the necessary requirement for a person to be declared the winner of a league.

8 System Architecture and System Design

8.1 Architectural Styles

In order to make the most efficient use of our software, we will couple several known software tools and principles into our design. The follow architecture types will be expanded in detail to not only reflect general functionality, but also to reflect functionality of the software as a whole. As explained, each will play a crucial role in the success of our software and will be largely derived from the necessities of the software. That being said, architectural systems will include (and may be expanded upon in the future) the Model View Controller, Data-Centric Design, Client-Server access, and RESTful design, with each architecture serving a small part of the whole result.

Model-View-Controller

The Model View Controller is a User Interface implementation method which will separate the software into 3 specific groups; that is: the model, view, and controller subsections. The view category is typically limited to UI specific output, i.e. a webpage with stock information. That being said, the model remains the core component of the MVC method which holds all of the data, functions, and tools. The controller simply takes the input and converts it into a command for either the model or the view.

The MVC method is ideal for this particular software because it allows the design to be broken down into smaller sub-problems. By splitting into 3 parts, we can separate UI functions, from database functions, and have all of them handled ultimately by the controller. Thus in terms of fluidity of the design, adding in the MVC allows each to be distinct and allows for the programming to be made far easier.

Data-Centric Design

Data is the fundamental backbone of Paramount investments. Stored within our database, will be numerous bouts of data, which will be necessary for all aspects of the software. The database needs to contain not only data pulled from the Yahoo! Finance API, but more importantly user specific data. Whenever the user logs in, they need to have access to a personal host of their own data. That includes but is not limited to complete portfolio, leagues, achievements, leaderboard, and settings. More importantly, the data needs to be stored in a way that it can be accessed by multiple subsystems whenever necessary. So in using this method, we can keep the data specific parts in the software abstract and easily accessible.

Client-Server Access

The user will be constantly interacting with the interface. All of the interactions are occurring, thus, on a client server basis. The user remains the primary client, and as such, constantly must interact with the other subsystems. All of the infrastructure provided by Paramount Investments will need to be accessed by the user. This ensures a smooth communication between each of the parts of the MVC and between client and infrastructure. Further, the infrastructure provided by Paramount investments will be able to access infrastructure of non-associative systems.

Representational State Transfer

As a software implementing a client server Access system, a REST system is also inherently implied. The RESTful design principles state that in addition to having a Client-Server Access system, the system has a scalability of components, that the interface is uniform, stateless, and cacheable. Using this method will employ a smooth, modular set of code. Using the interface specifications within the RESTful outline allows both the user and the designers to have streamline interactions with the interface. That is the user knows quite clearly what he or she is doing when say a link is clicked on a web page. The request is converted and sent out to the controller.

Importantly, the RESTful implementation can be implemented on multiple levels. And as is desired, this system will be able to work on Android and iOS as well as through standard web interfaces. Thus a smooth transition between these mediums is incredibly important. Thus whether a user places an order on his cell phone or online, he should be able to experience a uniform experience across all mediums. Using the RESTful system will help in this process.

8.2 Identifying Subsystems

Paramount investments aims to set its platform on multiple interfaces. As such, subsystem identification becomes an integral part of initial analysis of the software. On a thick layer, our platform exists with a front-end system and a back-end system. But on a much deeper level, we can see that, each of these subsystems can be broken down into still greater detail. Front end systems typically involve user interface, and object interactions with the user. Back-end will refer to all database schema, implementation and interactions with relevant hardware. Also included are non-associative items which are necessary to the success of our system.

Front-end systems are formally plain. The user interface which displays views and specific data to the user on multiple platform is included here. That is, it will contain different mappings and specific implementations for iOS and Android as well as natively for the Web. The front-end system will have to maintain constant communication with the back-end system to maintain consistency and retrieve data regularly. It needs to be able to successfully communicate information from commands given by the user and communicate them to the back end. The back end system will retrieve necessary data and information and return the data to the UI and user to project the page or information requested.

Our back end system will be broken down further and is easily considered the most important part of our infrastructure. Since we are using the MVC framework, the back end system is to be broken down into controller and database subsystems. Additionally, we will have the financial

retrieval system and queuing systems as previously outline. Thus, the bulk of the command processing is handled by our back-end subsystem. The back-end system must not only communicate among the subsystems within itself, but it must also communicate with the front-end UI system to respond to commands and also communicate with the non-associate systems as well.

Breaking down the subsystem further, we highlight the importance of the financial retrieval system, and the queue system. The financial retrieval system will communicate with Yahoo! Finance to retrieve relevant information as requested by the controller (whenever the controller receives an input from the front-end user). The queuing system will handle other processes and largely communication with non-associative systems. It will also be involved in queuing and handling all back-end processes and monitors to ensure that the correct commands are processed at the correct time. The success of these modules, the success of the entire back-end system, and the success of communication amongst the systems will be crucial for the overall success of the software.

8.3 Mapping Hardware to Subsystems

The Paramount Investments League is contained on a MySQL database server, which is stored on one machine. However, the system as a whole is spread across several machines. The system to be is divided into two separate sections: a front-end side that is run on the clients web browser of choice, and a back-end that runs on the server side of the database. The front-end is the main graphical user interface (GUI) between the system and the client. The front-end is responsible for communication between the GUI and the database for purposes such as confirming market orders and updating an investors portfolio. These changes in the front-end are reflected in the back-end side of the server. The back-end will handle proper execution of market orders and will updates users on each of their transactions.

8.4 Persistent Data Storage

The plan for data storage exists at the core of Paramount Investments. Since so much of our software depends on properly developed and updated data, it is of the utmost important that our database schema represent accurately all objects involved. That is, the data must accurately (at all times) reflect all relevant user data, stock information, ticker variables, league settings, achievements, leaderboards, and all other relevant objects.

Paramount Investments will make heavy use of the relational database MySQL. Relational databases are far more practical for the needs of this particular software. That is, relational databases consist of several indexed tables filled with various object attributes. As can be viewed in the class diagrams on the previous page, this is necessary for the large quantity of objects which will be present in the software. Tables will need to exist not only for user data and settings such as log in and league profiles, but also for stock and portfolio information. Further, these databases need to be constantly written and rewritten to ensure constantly updated and accurate information. Items such as leaderboards, and information which will be able to be viewed on each users portfolio need to constantly reflect accurate data.

The data will be retrieved from the respective database table in the form of a query. When a user inputs a command to retrieve data, a query must be placed, the table searched, and the eventual correct data value (or values) returned. For example if a user requests his or her settings,

it can query currently selected settings and return those values to the UI and to the user. If the user elects to make a change this will be sent back to the database, updated and saved for further access later. The same process can be mirrored and applied to all facets of the software. Several tables will be used for varying data as has been outlined in the diagrams above. The success of the software is dependent on the values being returned accurately and in the most updated form at all times. Because of that, the database must receive a regular feed from the Yahoo! Finance API in order to constantly update and reflect data when queries are placed. In doing so, users will have constantly accurate views of their portfolio performance, leaderboards, achievements, stock tickers, and recent trades going on throughout the league and entire user base. It is in this way that the Paramount Investment software will distinguish itself from others and retain functionality and efficient realization of its ultimate goals and requirements.

8.5 Network Protocol

As is standard for software of this type, Paramount Investments will uses the standard Hypertext Transfer Protocal (HTTP). HTTP acts by structuring text which is uses hyperlinks to communicate messages through text between nodes. While not necessarily unique or particular to our situation, it is still important to note that this will be the primary protocol between user and software interface. More importantly, the HTTP protocol will be used not only on web-based devices but also on Android and iOS devices as well. From any of these mediums, the users can access various webpages and links from the Paramount Investment website. They will be able to access, through this protocol, all relevant stock, portfolio, and relevant information through these pages and by using the HTTP protocol.

8.6 Global Control Flow

Execution Order

In general, the implementation of the system at Paramount Investments is for the most part, event-driven. All the features that the system has to offer must be triggered by some entity, whether it be the user themselves or some other part of the system. Overall, most of the event-driven characteristic comes from the user end of the system. Many of the functionalities (stock trading, portfolio viewing, league joining, etc..) can only be triggered by the user. There are however, some event-driven functionality that are initiated by the system. When the user places an order, it is processed and added into the database. From here, the system initiates the process of checking the order and then it uses Yahoo Finance API to retrieve market information about the stock, obtain a quote and then actually process the order.

There is some functionality that have to be executed in a defined order. Before the user starts investing, they must a few steps:

- Registration/creating an account: any user must register within our website before joining a league.
- Join a league: any user must first join a league before they can start investing.
- Achievements: any user must first complete the required criteria before they can be awarded with the achievement trophy.

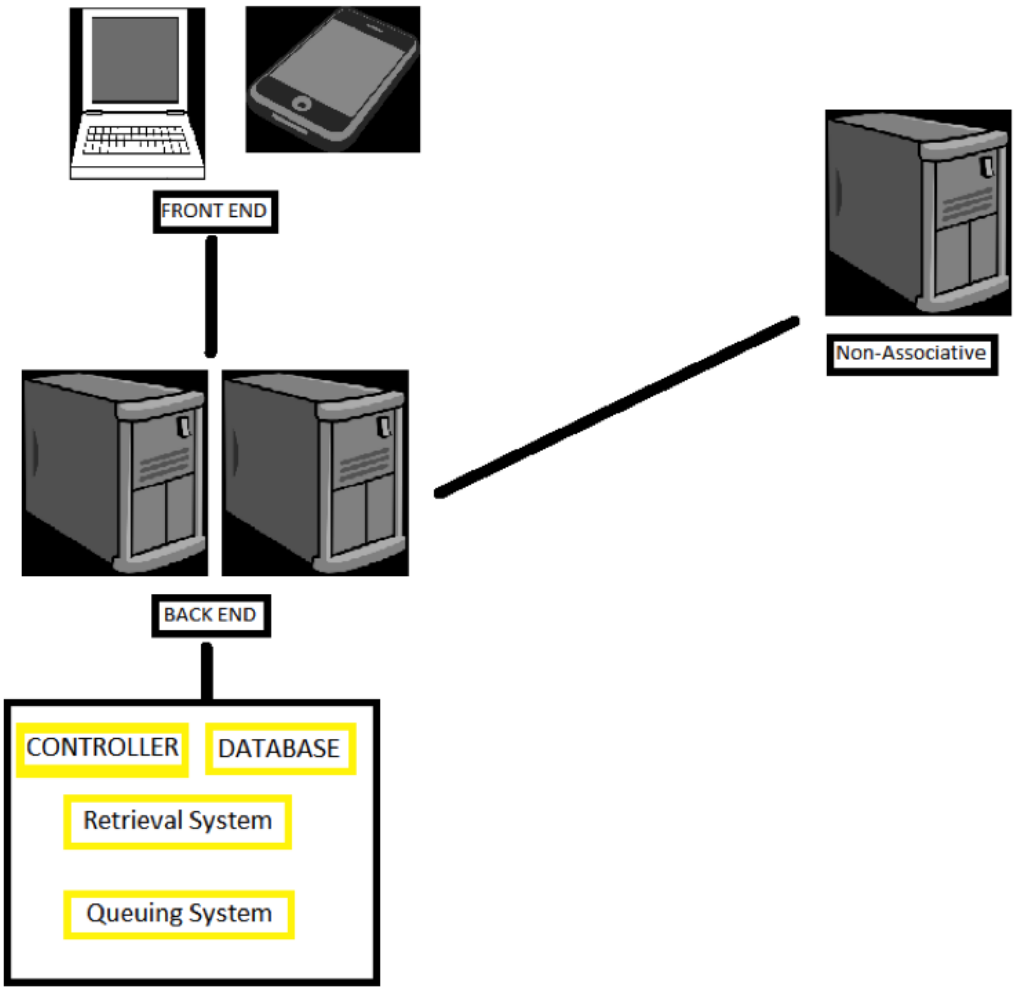


Figure 8.1: Diagram of the network protocol.

However, on the whole, our system is still definitively an event-driven one.

Time Dependency

In general, the system at Paramount Investments is very much a real-time system, but there are features that do not depend on time. The real-time system is very reliant on the stock market, which itself has certain times of operation. As the user is browsing the website, there are real-time timers that help the system process information that it is receiving.

- Achievement Timer: This timer is used at the end of the day to check for achievement specs for all the users. Achievements/ rewards will then be dished out accordingly.
- Stock Market open and close: The stock market has a time interval between when its open and when it is closed.
- Queuing system: the orders placed by users are placed into a queue. Depending on market conditions, this can place high loads on the server. To balance the server load, we must split the orders effectively. The timer in this system helps to check for unexecuted/ outstanding orders and then processes them.

Concurrency

There are sub-systems in our main system which have to be carefully thought of due to concurrency. The biggest of these is the queuing subsystem. This produces a concurrency issue because we have to make sure that no more than 1 order is being inserted into the queue at any given time. Likewise, we also have to make sure that no more than 1 order is being dequeued from a given queue. Other than this our system really does not need any synchronization. However, this may change as we are implementing our system.

8.7 Hardware Requirements

The hardware requirements on the server side are the main contribution to the operation of Paramount Investments League, leaving the client-side with minimal requirements. In fact, the only requirement of a client will that it runs a browser that is capable of running a modern web browswer.

Internet Connection

In order for Paramount Investments League to use any of its core functions (trading stocks, updating user portfolio, tracking administrative actions, etc.), an internet connection is required. Since most of the data being transferred is text (executable instructions), a low band of frequency is required. Note that a complete scalable analysis has not been performed on the system, so a low band of frequency is based off of the needs of the current website. For ideal performance, higher bandwidths of frequency should be used in order to reduce any overhead. A network connection between the server and the Yahoo Finance API is necessary during trade hours (9:30am - 4:00pm Monday through Friday), otherwise, no investors can perform a transaction.

Disk Space

The server must have adequate hard drive space to be able to store all of the database information. All data being stored is the sum of all program instructions for the system. 10 GB of storage space should be sufficient for the system.

System Memory

Since this system is in active development, there is limited concrete evidence that supports the overall performance of the system. The system will load copies of database stored information in order to operate over it. For better throughput, the memory should be managed using a Least

Recently Used scheme (LRU) in order to keep the system memory populated with useful information. A LRU scheme will release any bits of memory that havent been accessed in a long time, and it will replace it with information that is used more often. Also, any operations used on loaded information will also use up system memory. A minimum of 512 MB should be used for testing our system. In addition, as our user based expands, it is obvious that the system memory will also have to grow with it.

Client-side Hardware Requirements

The core hardware requirement on the client-side of the system will be an internet connection. This is essential for the client to be able to remotely connect to the server in order to access the database. Without an internet connection, no client will be able to use a web browser to visit the Paramount Investments League website. In addition to an internet connection, and for a friendly user experience, anyone on the client-side should have a functional mouse and keyboard, as well as a graphic display to see their portfolio. To display the Paramount Investments League website, a screen with a minimum resolution of 800x600 pixels is adequate.

9 Data Structures & Algorithms

9.1 Data Structures

In the implementation of the system at Paramount Investments League, there will be 3 main data structures in use. These 3 data structures are a Queue, ArrayList, and the HashMap.

Queue

The system at Paramount Investments league uses a queue data structure to hold all the orders/transactions that users may place during in the system during the day. Because of the FIFO (First in First out) property of the queue the orders that were placed first will the ones that are executed/processed first. This mimics the real world scenarios and will help capture part of the essence of stock market trading. At this stage in the implementation, we will be trying to accomplish this using the interface provided to use by the Queue Interface in Java. One of the requirements we require of this queue is that it be thread safe since there can be multiple users placing their orders at the same time into the same queue. If we find that there is a space limitation or lack of thread synchronization of this queue implementation, we will make an attempt to code the queue ourselves.

ArrayList

The system at Paramount Investments league will also be using an array data structure to keep track of the positions that an investor might have in a single stock. In most case, investors will have only one position per stock but there are scenarios where this is not true and the investor may have more than one position in a single stock. Because we are unsure of how many positions the investor might want, we need to be able to account for this using a data structure that has quick random reads but also has the capability to grow in size without restriction. This is achieved using the ArrayList class provided in Java. The ArrayList class has random read capability just like an array, but it also has the capability to grow indefinitely just like a linked list. Hence we will be using the ArrayList to keep track of the positions per stock.

HashMap

The system at Paramount Investments League will be using a HashMap to keep track of all the stocks that a user chooses to invest in for a given portfolio. This data structure needs to have quick insertion, delete, and read times. We chose the HashMap to accomplish this task because of the speed at which it is possible to insert, delete, and access information in a HashMap. Because there

are hundreds of different stocks, quick access to information is a necessity. We plan to accomplish this task using the HashMap class in Java.

9.2 Algorithms

At the time of this report there is only one interesting algorithm that has been designed and implemented. We expect as the project progress for this section to flesh out more and more, and include additional algorithms.

A Method For Reducing API Calls in a Highly Concurrent Environment

Our system relies on external API's[9] in order to accomplish the most central tasks, namely retrieving up-to-date stock data. Since we are using a free API, their are limits to the number of times that we can request information from the API without having our IP address[10] blocked. In order to limit the number of calls that are made, we need to cache the results on our servers.

In order to accomplish this we wrote a service that is concurrent and maintains a cache of stocks values on our server updating them periodically. Here is a brief overview of the algorithm:

Algorithm 1: Retrieve and Cache Stock Values

```
1  stockTicker ← End user does an operation that requires a stock value;
2  stock ← HashMap.get(stockTicker);
3  if stock exists then
4      return stock;
5  Synchronize;
6  if stock exists then
7      return stock;
8  stockValues ← YahooAPICall(stockTicker);
9  stock ← newStock(stockValues);
10 HashMap.put(stockTicker, stock);
11 return Stock;
```

In order to make the above algorithm work in a Concurrent environment, we synchronize[11] it in the critical section, that is, we only allow one user at a time to add something to the HashMap.

A Method For Invalidating and Updating an In Memory Data Structure in a Highly Concurrent Environment

In order to update the HashMap periodically, we run a background thread that sleeps for some defined amount of time, then runs. This background process, builds an entirely new HashMap, and once complete, replaces the out of date HashMap.

Algorithm 2: Update In Memory Data Structure

```
1 newStructure ← generate a new empty data structure;
2 oldStructure ← get a reference to the old structure;
3 for every object o : oldStructure do
4     newStructure.add(o.update());
5 oldStructure = newStructure;
```

10 User Interface Design & Implementation

10.1 Updated Pages

At this time, there has been no discernible progress from the UI mocks that were done in report 1. That said, we expect there to be many small, but substantial tweaks made to the final site once we begin doing user interaction studies. We also expect there to be minor changes made for the first demonstration, but this hasn't been implemented or finalized yet.

10.2 Efficiency of the Views

One thing that we need to concentrate on is ensuring that the website is fast for all users no matter what kind of device or connection the end user is using. For this reason, you will see a logical breakdown of the website which will allow us to cache elements of the site on the client side that generally won't change. We do this be separating the header, ticker, and the content of a given page. Since the header and ticker are the same across the entire site, they only need be loaded on the client a single time, and can be cached on the client side for the duration of the visit or longer.

The content of each individual page is dynamic, but by harnessing technologies like AJAX[12] and Comet[13], we are able to indicate to the user that the page is always reacting to their inputs without reloading the page. This again allows us to cache the resulting page on the client side, an perform updates as needed with minimal delay.

To further assist with reducing the load on clients, we will be using HTML[14] and CSS[15] to present our User Interface relying very minimally on pictures. Any picture that is displayed will be resized to the maximum allowed size and contained in an appropriate web format.

Finally, as discussed much throughout these reports, our goal is to be able to present our application across as many device as possible, including mobile, tablet, and desktop. We accomplish this by relying on the Twitter Bootstrap[6] CSS framework to help facilitate creating a responsive website.

Of course this all comes with a trade off, that is we won't support older browsers incapable of displaying and parsing HTML5/CSS3/JS or aren't web compliant with modern web standards. This should have minimal impact however, since most devices and users have a modern web browser, and those that don't generally don't fall into our target audience.

11 Design of Tests

No application is ever complete, but a big part of driving a project to a viable project is testing. Testing allows us to ensure expected functionality, check for possible security vulnerabilities, and prevent regression as the project moves forward. Attempting to launch a product without performing unit and integration testing, as well as "dog fooding"[16] an alpha version is a guarantee to have to putting out a buggy and sub par product. However, even with performing all the aforementioned, it is not possible to find and resolve every flaw before shipment. To this end, developers utilize *testing suites* in order to perform integration and unit testing in an efficient and effective manner.

A modern approach to this trade off is to build the feature set of an application around measurable, predefined tests. In this technique, known as Test-driven Development[17], developers iteratively define tests for intended future features, confirm that those features are not yet implemented (by running those tests), and then implementing the solutions. Though this approach does not test for all possible interplay between components, it is usually employed in high-paced development environments such as ours, where the coverage provided is usually respectable enough to prevent most problems.

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

11.1 Test Cases

The Paramount Investments League application is in active development, therefore, each test case specified is only applicable to existing functions during this stage of development. For the most thorough testing, we will perform unit tests on each component of the system currently in existence. The Paramount Investment League requires communication between Yahoo! Finance, our MySQL database, and our server, but unit testing these components in not efficient. Instead, we will perform integration tests on these units to see how they interact with each other.

Paramount Investments League will be using a Java/Scala Play Framework to develop our web application. The main reason for choosing Play Framework provides minimal resource consumption (CPU, memory, threads) and also supports big databases. Also, most of the team members are proficient in C++, so the transition to Java is doable.

11.2 Unit Tests

Database Manager

The tests listed below interact with our MySQL database, however they have no correlation to the implementation of the database.

1. **Test Case Identifier TC-1:**

   Function Tested: get player info(in user id : int) : class User

   Success/Fail Criteria A successful test is one that retrieves information about the requested player.

Test Procedure:	Expected Results
Call Function (Success)	Information requested matches the search criteria.
Call Function (Failure)	Information requested does not match the search criteria.

2. **Test Case Identifier TC-2:**

   Function Tested: update player info(in user id : int, in upd : class user):bool

   Success/Fail Criteria - A successful test is one that updates a player's information, whether it be an administrative action or game related.

Test Procedure:	Expected Results
Call Function (Success)	Player's profile is updated with new information. A value of true is returned after the function call.
Call Function (Failure)	Player's profile is not affected after attempted update. A value of false is returned after the function call.

3. **Test Case Identifier TC-3:**

   Function Tested: get order info(in transaction id : int):class transaction

Success/Fail Criteria - A successful test is one that returns the information associated with a specific transaction.

Test Procedure:	Expected Results
Call Function (Success)	Transaction information returned corresponds to transaction.id.
Call Function (Failure)	Transaction information isn't returned to the user.

4. **Test Case Identifier TC-4:**

   Function Tested: update league(in leagueInfo : class league):bool

   Success/Fail Criteria - This method is used when a league needs to be updated with the newest information provided.

Test Procedure:	Expected Results
Call Function (Success)	League information has been successfully updated. A value of true is returned after the function call.
Call Function (Failure)	League information has not changed from before. A value of false is returned after the function call.

5. **Test Case Identifier TC-5:**

   Function Tested: return league updates(in league id:int):class league

   Success/Fail Criteria - A successful test will return any league updates to the requested user.

Test Procedure:	Expected Results
Call Function (Success)	League updates are presented to the requesting user.
Call Function (Failure)	No data is presented to the user after function call.

Order Manager

The Order Manager is responsible for handling all tasks related to orders and transactions. The Order Manager is responsible for placing new orders in the system, as well as archiving old transactions in a table.

1. **Test Case Identifier TC-6:**

   Function Tested:Check order(in symbols: class Order) : bool

   Success/Fail Criteria A successful test will return a Boolean value of true corresponding to a valid user trade requests (buy, sell short, stop etc.).

Test Procedure:	Expected Results
Call Function (Success)	User is able to perform a valid transaction. A value of true is returned after the function call.
Call Function (Failure)	User will be notified that he/she will not be able to perform a valid transaction. (Ex. Not enough funds in their account). A value of false is returned after the function call.

2. **Test Case Identifier TC-7:**

   Function Tested: place order(in symbols: class Order) : bool

   Success/Fail Criteria - A successful test will allow the user to place a market order.

Test Procedure:	Expected Results
Call Function (Success)	Market order is placed, and a confirmation is sent to user. A value of true is returned after the function call.
Call Function (Failure)	Market order is not placed, and the user will be notified. A value of false is returned after the function call.

3. **Test Case Identifier TC-8:**

Function Tested: delete order(in symbols: transaction id): bool

Success/Fail Criteria - For a successful test, this method should delete an order from the system, assuming that it hasnt already been processed. If it has then this function will return a false value.

Test Procedure:	Expected Results
Call Function (Success)	Market order has been deleted from the queue. A value of true is returned after the function call.
Call Function (Failure)	Market order has already been recorded, user will be notified of the invalid transaction. A value of false is returned after the function call.

4. **Test Case Identifier TC-9:**

Function Tested: Execute order(in transaction id : int) : bool

Success/Fail Criteria - For a successful test, the system will obtain information from Yahoo! Finance and update a users portfolio accordingly.

Test Procedure:	Expected Results
Call Function (Success)	System retrieves data and updates the users portfolio. A value of true is returned after the function call.
Call Function (Failure)	System either does not retrieve information from database and or the users portfolio is not updated. A value of false is returned after the function call.

League Manager

This class is responsible for managing all the leagues in the system. It has the authority to create leagues, delete leagues, and modify leagues as it is instruction to do so.

1. **Test Case Identifier TC-10:**

Function Tested:Create league () : Class league

Success/Fail Criteria - A successful test is when the user can create a league from scratch.

Test Procedure:	Expected Results
Call Function (Success)	User is now the league manager, and their new league is added to their list of current leagues.
Call Function (Failure)	No new league is recorded in the system and the user will be notified that their attempt to create league has failed.

2. **Test Case Identifier TC-11:**

   Function Tested: reaturn league updates(in league id:int):class league

   Success/Fail Criteria - A successful test will delete the selected league.

Test Procedure:	Expected Results
Call Function (Success)	Selected league is deleted from the users list of league. A value of true is returned after the function call.
Call Function (Failure)	League will remain in the users list of league. A value of false is returned after the function call.

3. **Test Case Identifier TC-12:**

   Function Tested: change league name(in league id : int) : bool

   Success/Fail Criteria - A successful test will update the current league name with a modified one.

Test Procedure:	Expected Results
Call Function (Success)	League name has been changed and is reflected in the database. A value of true is returned after the function call.

Call Function (Failure)	League name has remained unchanged. A value of false is returned after the function call.

4. **Test Case Identifier TC-13:**

   Function Tested: Change league manager (in league id : int, in usr : class User) : bool

   Success/Fail Criteria - A successful test will change the current league manager with the new input league manager.

Test Procedure:	Expected Results
Call Function (Success)	League has a new manager, and all changes are reflected in database. A value of true is returned after the function call.
Call Function (Failure)	League manager remains unchanged. A value of false is returned after the function call.

5. **Test Case Identifier TC-14:**

   Function Tested:add rules(in league id : int) : bool

   Success/Fail Criteria -A successful test will add a new rule to the list of league rules already established.

Test Procedure:	Expected Results
Call Function (Success)	The newly added rule is reflected in the database. A value of true is returned after the function call.
Call Function (Failure)	The new rule to be added has not been added, and the database sees no changes in the list of rules. A value of false is returned after the function call.

6. **Test Case Identifier TC-15:**

Function Tested: Delete rules(in league id : int) : bool

Success/Fail Criteria - A successful test will delete a rule in the leagues list of rules.

Test Procedure:	Expected Results
Call Function (Success)	The selected rule is deleted, and the database is updated of the change. A value of true is returned after the function call.
Call Function (Failure)	The selected rule has not been removed and the database sees no changes. A value of false is returned after the function call.

Account Controller

This class exists to take care of any functions that involve any user accounts. Functions include, adding, modifying, or deleting an account.

1. **Test Case Identifier TC-16:**

   Function Tested: Login(in user id : int) : bool

   Success/Fail Criteria - A successful test will allow the user to visit their Paramount Investments League global portfolio.

Test Procedure:	Expected Results
Call Function (Success)	User is logged into the system and they can view their account. A value of true is returned after the function call.
Call Function (Failure)	User is not logged into the website. User may not have entered password correctly, or is not a registered user. A value of false is returned after the function call.

2. **Test Case Identifier TC-17:**

   Function Tested: logout(in user id : int) : bool

Success/Fail Criteria - A successful test will all the user to logout of their Paramount Investments League account.

Test Procedure:	Expected Results
Call Function (Success)	User is logged into the system and they can view their account. A value of true is returned after the function call.
Call Function (Failure)	User is not logged into the website. User may not have entered password correctly, or is not a registered user. A value of false is returned after the function call.

3. **Test Case Identifier TC-18:**

   Function Tested: Create account() : class User

   Success/Fail Criteria - A successful test will create a new user account.

Test Procedure:	Expected Results
Call Function (Success)	A former visitor to the Paramount Investments League website will now be a registered investor. A value of true is returned after the function call.
Call Function (Failure)	The request to make a new account has failed, and no new account will be reflected in the database. A value of false is returned after the function call.

4. **Test Case Identifier TC-19:**

   Function Tested: delete account(in user id : int) : bool

   Success/Fail Criteria - A successful test will delete the selected user account.

Test Procedure:	Expected Results

Call Function (Success)	An investor chooses to delete their account, and all portfolios will be deleted from the database. A value of true is returned after the function call.
Call Function (Failure)	The selected account remains in the system, the database doesnt lose the association with that user. A value of false is returned after the function call.

Yahoo Finance Adapter

This class is responsible for obtaining market data from Yahoo Finance API. It consists of three functions to get quotes, get company information, and to get sector information.

1. **Test Case Identifier TC-20:**

   Function Tested: Get quote(in stock ticker id : string) : class quote

   Success/Fail Criteria - A successful test will return the requested quote (stock) information to the user.

Test Procedure:	Expected Results
Call Function (Success)	Quote information is presented to the user. System requests to access information from Yahoo! Finance.
Call Function (Failure)	Quote information request does not go through and the user is notified of the error. System was not able to communicate with Yahoo! Finance.

2. **Test Case Identifier TC-21:**

   Function Tested: get company info(in stock ticker id : string) : class Company

   Success/Fail Criteria - A successful test will return the company information that the user requested.

Test Procedure:	Expected Results

Call Function (Success)	Company information is presented to the user. System can access company information from either database or link the user to the requested companys website.
Call Function (Failure)	Company information is not presented to the user. System failed to retrieve information from the database, or the company page link is invalid.

11.3 Test Coverage

The ideal test coverage would be to have a test that covers every edge case of every method. This is not only not feasible, it is impossible since it is not possible to actually know all the edge cases. Because of this we plan to test core functionality to provide a core amount of testing. Then through the use of alpha and beta build interactions with end users, we will be able to identify ways that user interact with the system that were not foreseen. We can then add additional testing to cover these new edge and use cases which will also help debug and prevent regression in the future.

11.4 Integration Testing

Integration testing will be done on a local developer machine by emulating the server environment. The system may not go live until the current system works in the integration environment. We accomplish this by having two branches of source code, master and dev. dev is the branch that all new work will be done on. From there, it will be pulled down into the local integration machine, tested and debugged. Once the system has been debugged, being sure to keep 1detailed logs of any system config changes needed, the source code will be pushed to master. Once pushed to master, any system config changes will be made on the production server in order to accomodate the new branch. Once those changes are made, master will be pulled into the production machine and a second round of integration testing will begin by launching the service on a developer port. If it passes all the tests, then the developer port will be shut down, and the system will relaunch the website on the normal http port.

12 History of Work, Current Status, & Future Work

12.1 History of Work

Throughout the semester we completed several of our planned milestones in a punctual, thorough, and consistent manner.

Our first planned milestone, completing the Report 1 Part 1 prior to 9 February 2014, was met on time. We continued to meet our report deadlines for Report 1 Part 2 and the full, compiled Report 1 by 16 February 2014 and 23 February 2013, respectively. For the second report, we successfully met our deadlines for Report 2 Part 1 (consisting of Sequence Diagrams, timing and communication diagrams) and Report 2 Part 2 (consisting of Class Diagrams, Interface, Architecture design, data structures, UI, tests and implementations) on 2 March 2014 and 9 March 2014, respectively.

As we met our initial Report deadlines, we also simultaneously began work on the initial build of *The Paramount Investment League*. We began with deploying our server environment, which took place between 22 February and 2 March 2014. While we were initially weighing our options between dedicated virtual private server (VPS) on Digital Ocean and Heroku, we determined that a dedicated virtual private server would better suit our needs and we successfully deployed it by 2 March 2013. In this time, we also finalized our plan for mockup-based views and the CSS/HTML plan.

Populating the server with Ubuntu 12.04.4, Play Framework, Twitter Bootstrap, and other supporting software, however, took a longer than anticipated. This is because we needed to not just deploy it to the live server, but also set up the enviornment on our local machines so that we could develop and test our code without running in production. We also documented all these details on our github wiki, which will allow future developers a walkthrough for getting setup with the project. This extended beyond our initial range of 1 March 2014 to 8 March 2013 and was completed by 13 March 2014.

We completed our full Report 2 deadline by 15 March 2014. After this point we shifted gears and committed to having *The Paramount Investmets League* Alpha build prepared for Demo 1. In the days between 13 March 2014 and 28 March 2014 we met several of our objectives. The Yahoo! Finance API was implemented, the MySQL database structure was deployed and populated. Our routing plan was completed, our views theme were implemented, and we successfully got users, and portfolios at a working state in which data could be utilized between them in our routing structure.

Our biggest strength was establishing manageable goals that still built on the strengths and failures of our predecessors. We chose a functional UI but were not overly-ambitious with functionality, otherwise; this enabled us to meet our goals in a punctual manner. We almost fully implemented our achievements system with unlockable functionality features is.

12.2 Current Status

Currently, *The Paramount Investments League* is a functioning web application. We have an online version of our latest builds on the domain http://192.241.248.22/ based on our Virtual Private server hosted on Digital Ocean. It features working orders, leagues, portfolios, and user systems with a fully-functional and responsive UI that can be used in tablets and phones in addition to personal computer to be user friendly for anyone on a smart device. Most core functionalities have been deployed and the current status is debugging and optimizing our website to address orders in an asynchronous fashion to maximize the efficiency of our application.

We have added graphs via Highcharts as well as leagues, improved pagination support, enhanced the UI based on user feedback, increased the amount of available achievements and will be able to demonstrate the ability to unlock functionality as one becomes an experienced investor.

12.3 Key Accomplishments

The Paramount Investment League unlocked the following achievements:

- Play Framework application for core Web Application functionality
- A barrier free site registration process through the use of OAuth
- A responsive UI usable on smart devices based on our custom flat theme
- A minimalistic and easy-to-use system for users, portfolios, and leagues
- An implementation of highcharts that presentably showed line-graph data

The Paramount Investment League also satisfied the following user stories:

- ST-1
- ST-2
- ST-3
- ST-4
- ST-5
- ST-6

- ST-7
- ST-8
- ST-10
- ST-12
- ST-13
- ST-17
- ST-22

The Paramount Investment League also implemented the following use cases:

- UC-1
- UC-2
- UC-3
- UC-4
- UC-5

12.4 Future Work

The final stretch for a release-build *The Paramount Investments League* would be an immersive tutorial system activating upon user registration. This would have extensively took advantage of AJAX and is the most complex type of functionality to deploy on a live version of *The Paramount Investment League*, hence it would have required careful, slow, and surgical-level development. This was one of our major post-demonstration goals, but the level of sophistication for a fully-functional live tutorial upon user registration was highly impractical for the remaining time we had.

We also would like to add an asynchronous processing system for orders so that we can support stop and limit options. This would be unlockable functionality but requires someone with the appropriate domain knowledge so that we can implement it as true-to-life as possible.

Finally, to maximize user retention, another future work goal would have been to implement social media and e-mail notification options so users can be reminded to check and update their portfolios and leagues on a regular basis. A mailer system was intended, however, development on that particular item fell in favor of achievemnets, which were implemented by the Alpha.

12.5 Project Management

As this project evolved, our contributions and interactions became more complex and intertwined. It would be impossible for us to break down the work and contributions that each individual did. It is for these reasons that we feel as a team it is safe to say that we equally contributed to the project and therefore request that equal contributions be applied for all members of this project.

References

[1] Investopedia, "Bid-ask spread — Investopedia." http://www.investopedia.com/terms/b/bid-askspread.asp. [Online; accessed 18 February 2013].

[2] Investopedia, "Short (or Short Position) definition — Investopedia." http://www.investopedia.com/terms/s/short.asp. [Online; accessed 22 February 2013].

[3] Investopedia, "Limit order definition — Investopedia." http://www.investopedia.com/terms/l/limitorder.asp. [Online; accessed 23 February 2013].

[4] Investopedia, "Stop order definition — Investopedia." http://www.investopedia.com/terms/s/stoporder.asp. [Online; accessed 22 Febrauary 2013].

[5] Wikipedia, "Stakeholder." http://en.wikipedia.org/wiki/Stakeholder_(corporate). [Online; accessed 19 March 2014].

[6] Wikipedia, "Bootstrap (front-end framework)." http://en.wikipedia.org/wiki/Bootstrap_(front-end_framework). [Online; accessed 23 February 2014].

[7] Wikipedia, "Openid." http://en.wikipedia.org/wiki/Openid. [Online; accessed 19 March 2014].

[8] Wikipedia, "Oauth." http://en.wikipedia.org/wiki/OAuth. [Online; accessed 23 February 2014].

[9] Wikipedia, "Application programming interface." http://en.wikipedia.org/wiki/Application_programming_interface. [Online; accessed 19 March 2014].

[10] Wikipedia, "Ip address." http://en.wikipedia.org/wiki/Ip_address. [Online; accessed 19 March 2014].

[11] Wikipedia, "Java concurrency - synchronization." http://en.wikipedia.org/wiki/Java_concurrency#Synchronization. [Online; accessed 19 March 2014].

[12] Wikipedia, "Ajax." http://en.wikipedia.org/wiki/Ajax_(programming). [Online; accessed 19 March 2014].

[13] Wikipedia, "Comet." http://en.wikipedia.org/wiki/Comet_(programming). [Online; accessed 19 March 2014].

[14] Wikipedia, "Html." http://en.wikipedia.org/wiki/Html. [Online; accessed 19 March 2014].

[15] Wikipedia, "Css." http://en.wikipedia.org/wiki/Css. [Online; accessed 19 March 2014].

[16] Wikipedia, "Eating your own dog food." http://en.wikipedia.org/wiki/Eating_your_own_dog_food. [Online; accessed 15 March 2014].

[17] Wikipedia, "Test-driven development." http://en.wikipedia.org/wiki/Test-driven_development. [Online; accessed 16 March 2014].