Rutgers University Investing

Report #3

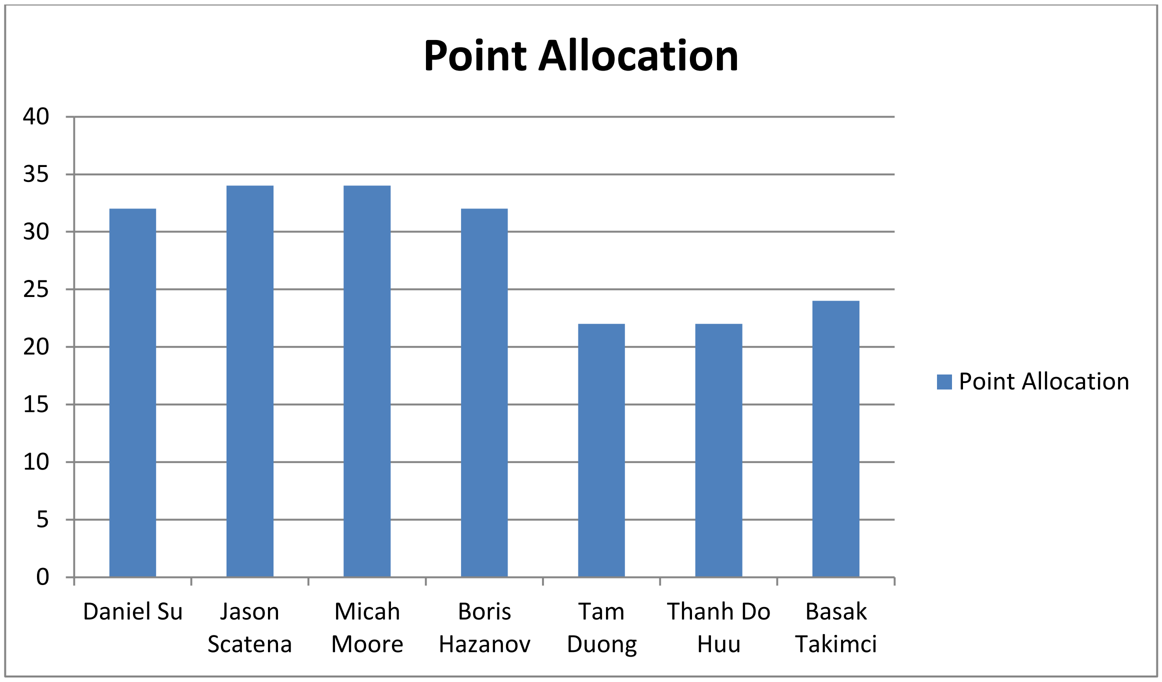
Group Members:

Daniel Su	Thanh Do Huu	Micah Moore
Jason Scatena	Boris Hazanov	Tam Duong
Basak Takimci		

Table of Contents

	Daniel Su	Jason Scatena	Micah Moore	Boris Hazanov	Tam Duong	Thanh Do Huu	Basak Takimci
Summary of Changes (5)	20%	20%					60%
CSR (6)					50%	50%	
Glossary of Terms (4)			100%				
System Req. (6)					100%		
Functional Req. (30)	50%	50%					
Effort Estimation (4)				50%	50%		
Domain Analysis (25)	50%	50%					
Interaction Diagrams (40)			50%	50%			
Class Diagrams (20)			50%	50%			
System Architecture (15)						100%	
Algorithms and Data Structures (4)						100%	
UI Design and Implementation (11)	31.8%	50%					18.2%
Design of Tests (12)					50%		50%
History of Work (5)					100%		
Project Management (13)							100%

Point allocation:
Daniel Su: 1+15+12.5+5.5 = 32
Jason Scatena: 1+15+12.5+5.5 = 34
Micah Moore: 4+20+10 = 34
Boris Hazanov: 2+20+10 = 32
Tam Duong: 3+6+2+6+5 = 22
Thanh Do Huu: 3+15+4 = 22
Bassak Takimici: 3+6+13+2 = 24


Point Allocation
40
35
30
25
20
15
10
5
0
Daniel Su
Jason Scatena
Micah Moore
Boris Hazanov
Tam Duong
Thanh Do Huu
Basak Takimci
Point Allocation

Summary of Changes

- Added effort estimation
- Modified the domain analysis to reflect changes in the design of our buying and selling system
- Modified the interaction diagrams to reflect our switch to the broker design pattern
- Created new classes in the class diagram to reflect the change to a broker design pattern
- Updated the user interface design to reflect changes made during development
- Changed Plan of Work to History of work
- Described the current status of the project and future work that we did not get the chance to implement.
- Added new references

Customer Statement of Requirements

Problem Statement

To Whom It May Concern,

As you no doubt know already, your team has been contracted to develop a stock market fantasy league software system for the Rutgers School of Business. Our department has decided to pursue this venture as a means of furthering our mission: “Rutgers Business School serves New Jersey and the world by cultivating business knowledge, ethical judgment, and global perspective in our students, scholarship, and society through innovative research and teaching and robust partnerships with businesses across our diverse communities.” We have decided that the best way to “cultivate business knowledge…and global perspective” in our students is to employ a stock market game.

There is great value in experience, especially in a topic such as investing in the stock market. The complexity of reading the market and making wise investments makes first-hand experience an extremely valuable learning tool. Students often times don’t have disposable income that they can afford to risk playing the market. In order to address this, the concept of a stock market game has come about, in which contestants are given a set amount of fictional money to “invest” in real stocks. At the end of the game, the participant with the most money wins. While disguised as a game, this activity provides students of finance an opportunity to make use of concepts learned in class and hone their abilities. The financial industry has changed considerably in recent times. While a trader a mere 20 years ago would rely on television, newspapers and telephones to gather market information traders today have the entire internet at their disposal. The birth of the modern web has allowed the populace to become generators of media in addition to consumers. It is now more possible than ever to gauge public opinion on companies and products. A myriad of online services exist that will allow a user to set up a stock market game at no cost. None of these other services, however, stress the importance of using the modern internet and social media as an information gathering tool. A stock market game that addresses this can better prepare students for real life trading. This can be done by building a stock market game service that integrates, at it’s very core, tools to make sense of the flood of information being posted to social media at all times. To not train students to use such a readily available source of pertinent information, would put our them at a major disadvantage.

In order for a stock market game to be usable, certain key functionalities will need to be included. There are three main roles users of our system will take: system administrator, league administrator, and player. The role of system administrator should be designed so that it can be undertaken by a professor or a member of the department who is assigned this extra responsibility. Whomever this role falls upon will, obviously, need a working understanding of the system at a fairly low-level since they will be the first resource in the event of a system malfunction. We understand that this role, in it's nature, requires a fairly large base of knowledge. We expect your team to either train personally, or provide documentation for the benefit of that individual. The system administrator will manage day to day activities relating to the hosting of the system, updates to the system, adding and removing league administrators, and monitoring system resource usage.

Each professor, that teaches one or more sections of a relevant course, will take on the role of league administrator. Extensive computer literacy cannot be assumed, and as such the user interface must put a heavy focus on being intuitive and easy to use. From this interface the professor must be able to control the entire game for his class sections. He or she will be able to manage player interaction, help players troubleshoot issues they may encounter, create a new league, and manually add or remove player accounts. The professor will have tools at his or her disposal that allow him or her to gain a big picture overview of the league. For example leaderboards and other league statistics. The league administrator will also be able to adjust rules and policies for his or her specific league so as to help it align more closely with the course syllabus.

Students will take on the role of system player. To avoid unnecessary work on the part of the league administrator, the system should allow players to register a new account and join the appropriate league with no administrator interaction. When a player accesses the web interface they will be presented with the functionality needed to research, buy, and sell stocks. The players will need to have options for data visualization and statistics tracking. A host of options for alerting the player of pertinent information using products they are already comfortable with should be explored, such as: twitter, SMS, and email.

As primarily an education tool, the game must attempt to train student players to best make use of the resources available to them. The market value of a company isn't exclusively based on how they are doing economically; public opinion greatly influences stock prices. As of last summer, twitter had over 210 million active users. The flood of tweets posted daily act as a free source of market research giving a survey of public opinion. Using this freely available market information, we want students, using our system, to be able to make use of functionality unavailable in previous stock market simulations and become more broadly informed investors.

We would like you to build a system that gives players a twitter-sourced sentiment score for all the stocks in their portfolio. In addition, players should be able to manually ask the system to watch the sentiment of prospective stocks. To further improve the player's experience, the game should also allow players to follow certain organizations, groups or fields of interest using public twitter accounts. The goal of this feature is to quickly and accurately inform players of recent changes in stocks that they own or are researching. This information will help players make informed decisions, using the public opinion of a company's success, as opposed to solely the news and financial data available through more traditional media.

In order to make the system easily accessible, we would like to make the game a web application, not requiring professors or students to download and install anything to their personal devices. Professors and students should interact with the game through a pragmatic and clean web interface that looks good and is easily usable across a host of devices. We would like this interface to be hosted on a server owned by the department and handled by the system administrator. We expect the interface to provide access to all the functionality of the system in a clear and intuitive way.

The functionality we expect from this system can be summarized as follows:

- For every user
    - Pre-login info page
    - Login
    - Registration
    - Account management
        - Change password
        - Change contact information
- For the system administrator
    - Add or remove professors as league administrators
    - View system usage statistics
    - Manually reset user passwords
- For the League administrators
    - Start, manage and end a league
    - Set league rules and policies
        - starting capital
        - start date
        - end date
        - league name
    - Manually add or remove players
    - View league statistics
- For the Player
    - View their portfolio
    - View competitor portfolios through a profile page
    - Initiate market orders
        - Short
        - Cover
    - Initiate conditional orders
        - Stop orders
        - Limit Orders
    - View twitter feeds of relevant organizations
    - View twitter sentiment scores of various organizations
    - View stock price history of any public company
    - view leaderboard and other league statistics
    - Communicate with league administrator
    - Access public league chat room

We are looking forward to seeing the system that your team develops and encourage you to contact us if you have any questions or updates.

Regards,
The Rutgers School of Business Development Board

Glossary of Terms

Stock Market League - A market simulation that allows users to practice trading and learn how the market works

Stock - A type of security that represents a claim on part of corporation’s assets and earnings

Ask Price - The minimum price that a seller or sellers are willing to receive for the security

Bid Price - The maximum price that a buyer or buyers are willing to pay for a security

Ticker Symbol - A stock symbol or ticker symbol is an abbreviation used to uniquely identify publicly traded shares of a particular stock on a particular stock market

User Groups:

Player - A standard user that participates in the leagues and has control over their personal profile and settings

League Administrator - The league administrator manages leagues that they have created and the players that participate in those leagues

System Administrator - The system administrator is the super user who has the highest privileges and can manage all other users as well as system settings

Orders:
Market Orders (Immediately executed):

Buy Order - An order to purchase a specific amount of stock

Sell Order - An order to sell a specific amount of stock

Short Order - An order where a sell is performed using borrowed stocks. The trader then expects the value to decrease and to profit by performing a cover order to return the loaned stocks at a lower price

Cover Order - An order where a buy is performed in order to cover / return stocks that were previously loaned to the trader

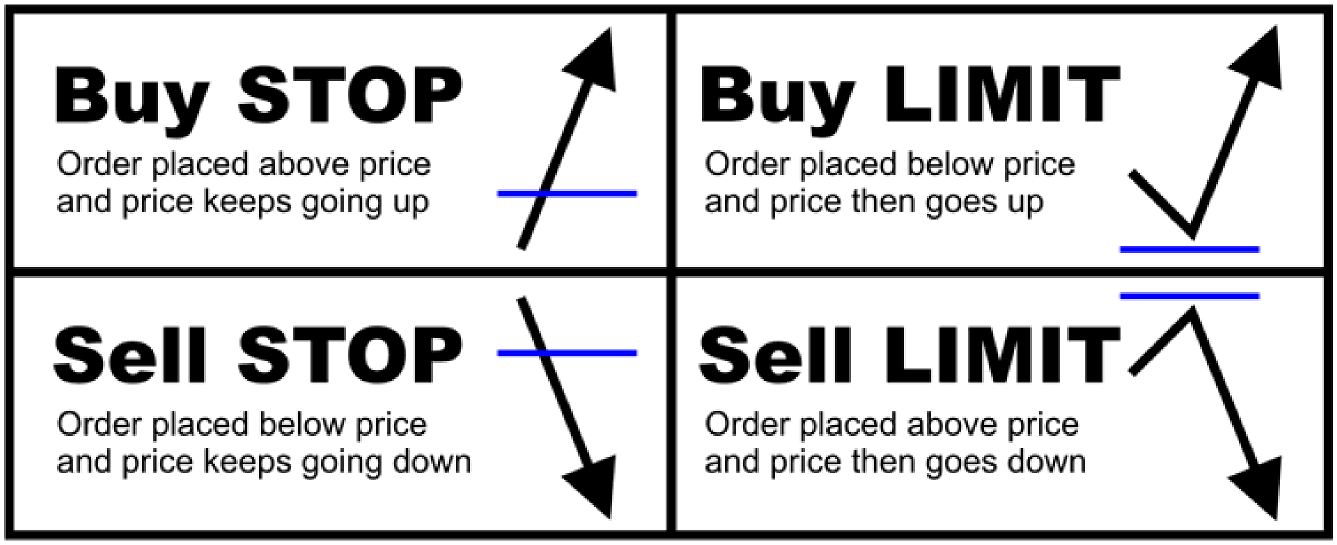
Conditional Orders (Executed on met condition):

Stop Order - An order that activates only when the security you want to buy or sell reaches the stop price

Limit Order - An order that sets the maximum or minimum at which to buy or sell a stock

*Limit orders guarantee the trade will be made at particular price while a stop order does not

Below is an image that allows better understanding of these terms


Buy STOP
Order placed above price and price keeps going up
Buy LIMIT
Order placed below price and price then goes up
Sell STOP
Order placed below price and price keeps going down
Sell LIMIT
Order placed above price and price then goes down

Additional information can be found at http://www.investopedia.com/

<u>System Requirements</u>

Functional Requirements

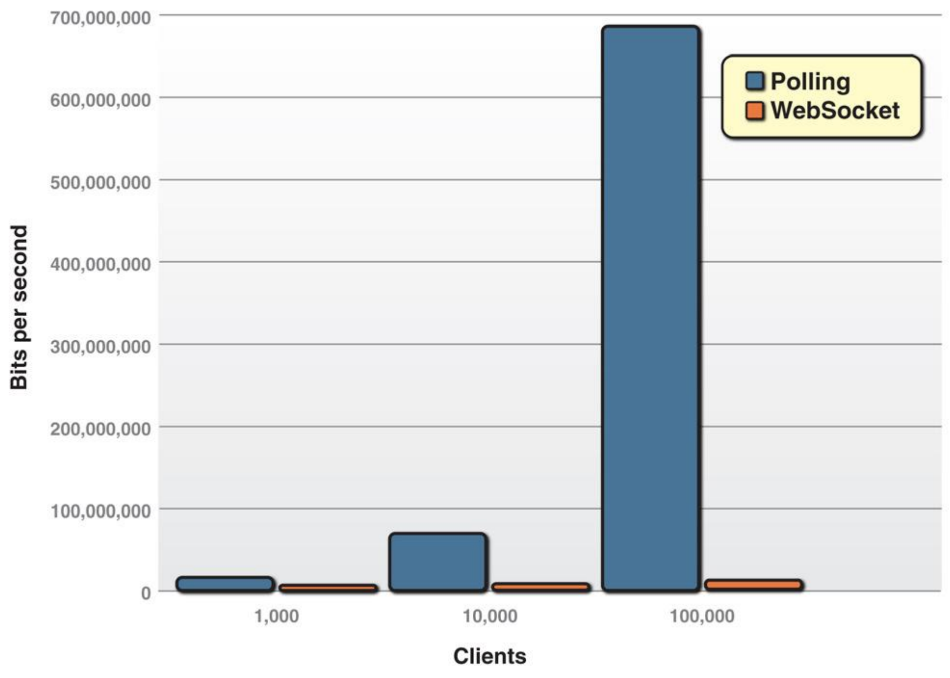
PW = Priority Weight (from 1 to 5)

ID	PW	Requirement
REQ-1	5	The system will provide an information page pre-login.
REQ-2	5	The system will allow users to log-in.
REQ-3	5	The system will provide a registration page for new users.
REQ-4	5	The system will allow users to change their password.
REQ-5	4	The system will allow users to change their contact information.
REQ-6	5	The system will allow the system administrator to add or remove professors as league administrators.
REQ-7	5	The system will allow the system administrator to view system usage statistics.
REQ-8	5	The system will allow systems administrator to manually change a user's password.
REQ-9	5	The system will allow to the league administrator to start, manage and end a league.
REQ-10	5	The system will allow to the league administrator to set league rules, including: • Starting capital. • Start date. • End date. • League name.
REQ-11	4	System will allow league administrator to add/remove players within their league manually.
REQ-12	3	System will allow league administrator to view leagues statistics.
REQ-13	3	System will allow to each user view his statistics.
REQ-14	3	System will allow to each user View competitor portfolios through a profile page

REQ-15	3	System will allow to each user Initiate market orders such as: • Short • Cover
REQ-16	2	System will allow to each user Initiate market orders such as: • Stop orders • Limit Orders
REQ-17	3	The system will allow users to View twitter feeds of relevant organizations
REQ-18	2	The system will allow users to View twitter sentiment scores of various organizations
REQ-19	3	The system will allow users to View stock price history of any public company
REQ-20	2	The system will allow users to View leaderboard and other league statistics
REQ-21	5	The system will allow for low latency real-time trades
REQ-22	5	The system will allow for scalability

Analysis on REQ-22

Stock market simulation systems done by previous groups all had one thing in common. They utilized the HTTP protocol to accomplish market orders. While this works fine in small scale deployments, when these systems are rolled into high volume environments, they will start to require massive costs to maintain. What we are proposing is to use a WebSocket implementation that reduces the overhead for each trade from up to kilobytes of data, down to just a few bytes. With a high enough volume of trading, this implementation can save a very significant amount of money.



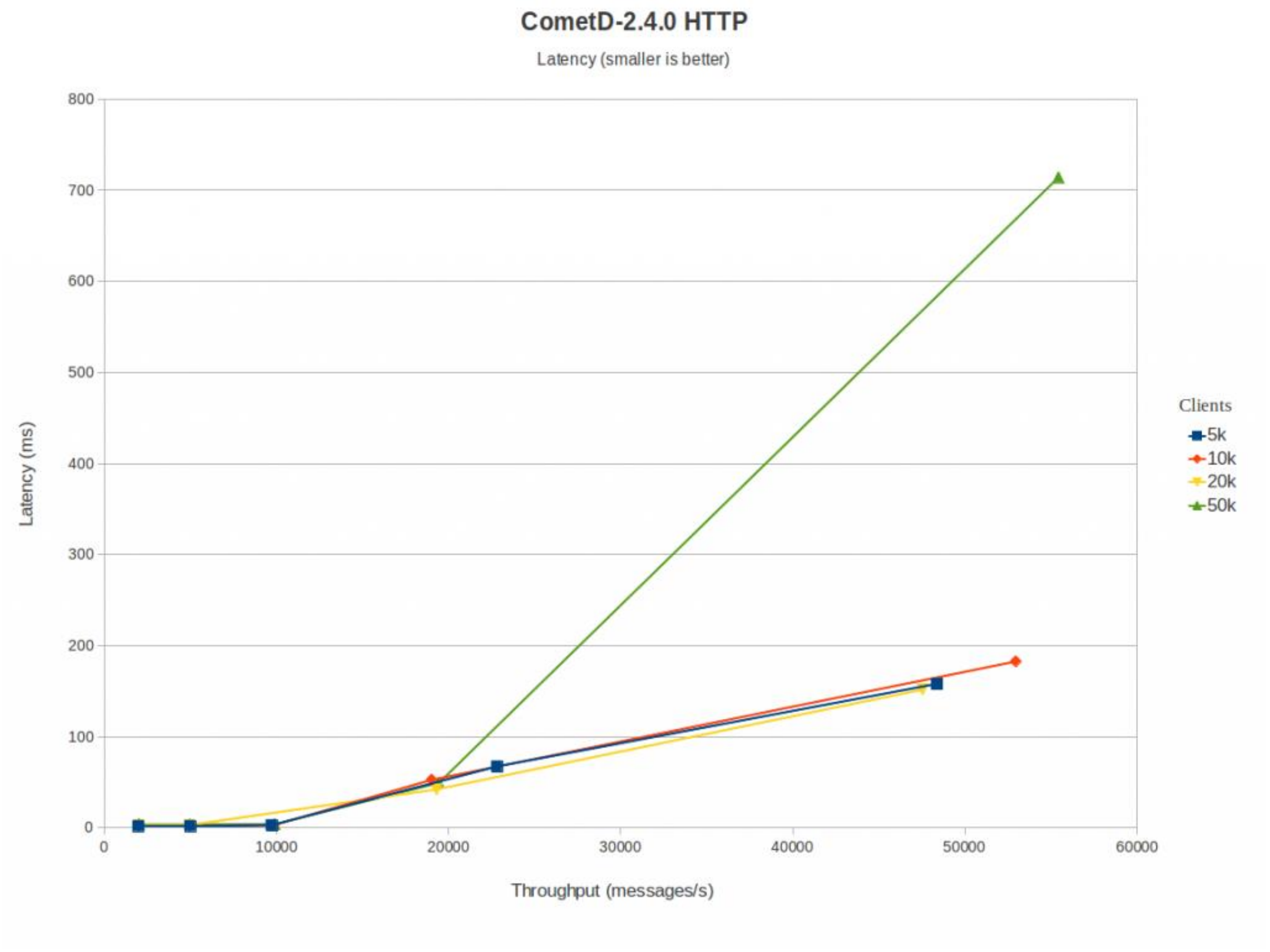
http://blog.kaazing.com/2010/02/24/5-signs-you-need-html5-web-sockets-part-2/

Analysis on REQ-21

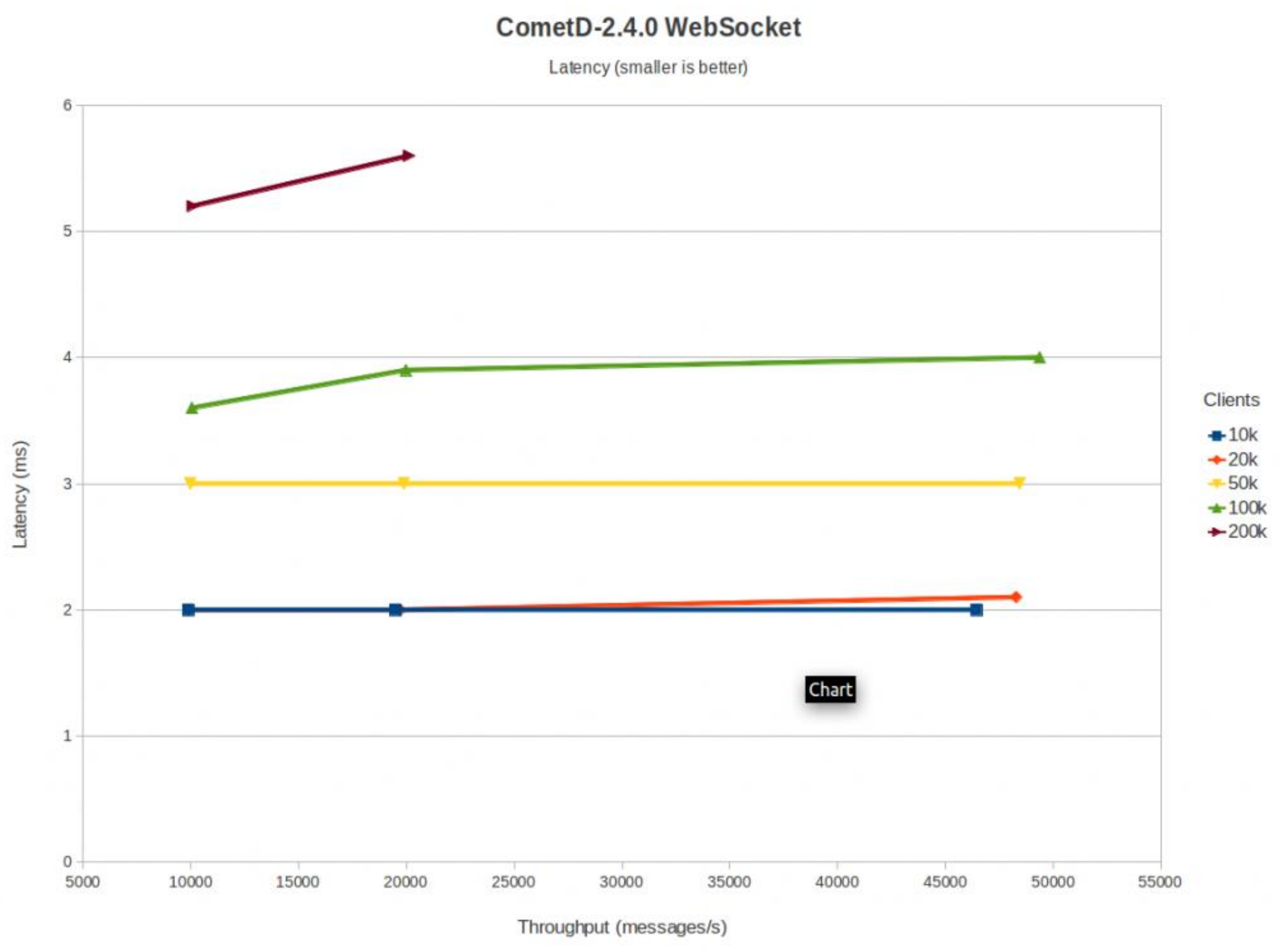
By using the WebSocket implementation, we also reduce the latency response down. This can have huge ramifications for a high frequency trading simulation. InformationWeek (http://www.informationweek.com/wall-streets-quest-to-process-data-at-the-speed-of-light/d/d-id/1054287) estimates that 1ms of latency can be worth up to $100 million per year to a major brokerage firm. The quest for low latency is so high, that firms are willing to lay down private fiber lines to improve latency response by just microseconds. (http://spreadnetworks.com/press-releases/10-04-2012-latency-improvements/)

While our implementation is not to conduct real world market trades, a low latency response does allow for more simulation options.

Below are simulations on the two different protocols that demonstrates the advantage of WebSocket



CometD-2.4.0 HTTP
Latency (smaller is better)
Latency (ms)
Throughput (messages/s)
0
100
200
300
400
500
600
700
800
0
10000
20000
30000
40000
50000
60000
Clients
5k
10k
20k
50k


CometD-2.4.0 WebSocket
Latency (smaller is better)
Latency (ms)
Throughput (messages/s)
Clients
10k
20k
50k
100k
200k
Chart

Non Functional Requirements

Functional
The big three web browsers will be supported so compatibility with a majority of users will be achieved. User security will be a major priority with features for authentication and encryption for sensitive data such as passkeys.

Usability
We will focus on providing a clean and consistent interface through CSS that will appeal to the user. AJAX will also be used to allow for immediate responses for user actions.

Reliability
Users will be given a confirmation message for sales transactions to allow for detection of user error. Error messages will be displayed to the user to notify failure of completing the proposed action.

Performance
There will be a focus on scalability and an efficient system for passing message between the client and the server (ie. for transactions).

Supportability
The front end will support access from mobile devices or devices with smaller resolutions. Maintenance support is also included through the admin control system.

PW = Priority Weight (from 1 to 5)

ID	PW	Requirement
REQ-23	3	Inter browser compatibility
REQ-24	5	User Security
REQ-25	2	Graphical Design
REQ-26	4	Responsiveness
REQ-27	3	Error Response
REQ-28	4	Scaling and Efficiency
REQ-29	3	Front End Interface
REQ-30	4	Maintenance Control

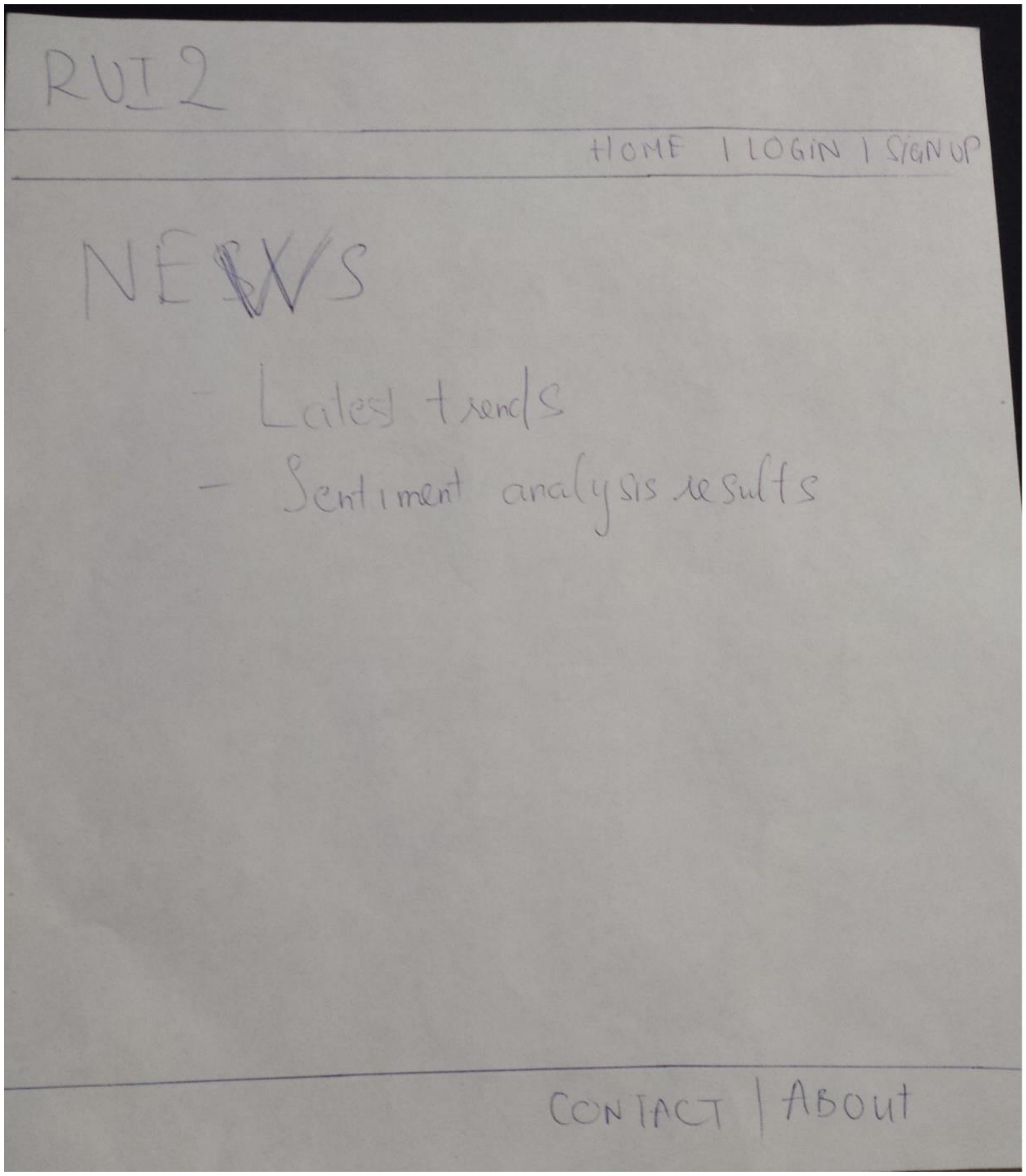
On-Screen Appearance Requirements

The on screen appearance design will primarily cater to those with a laptop and desktop system, for resolutions of 720p and greater. There will also be support for handheld and tablet devices through the use of responsive CSS in order provide these users with a functional interface. Dynamic data loading through JavaScript will also be used in order to minimize wait time for the user.

Technologies used will be restricted to those that are universally compatible. Flash and Java will not be utilized due to their limited compatibility and massive resource drain.

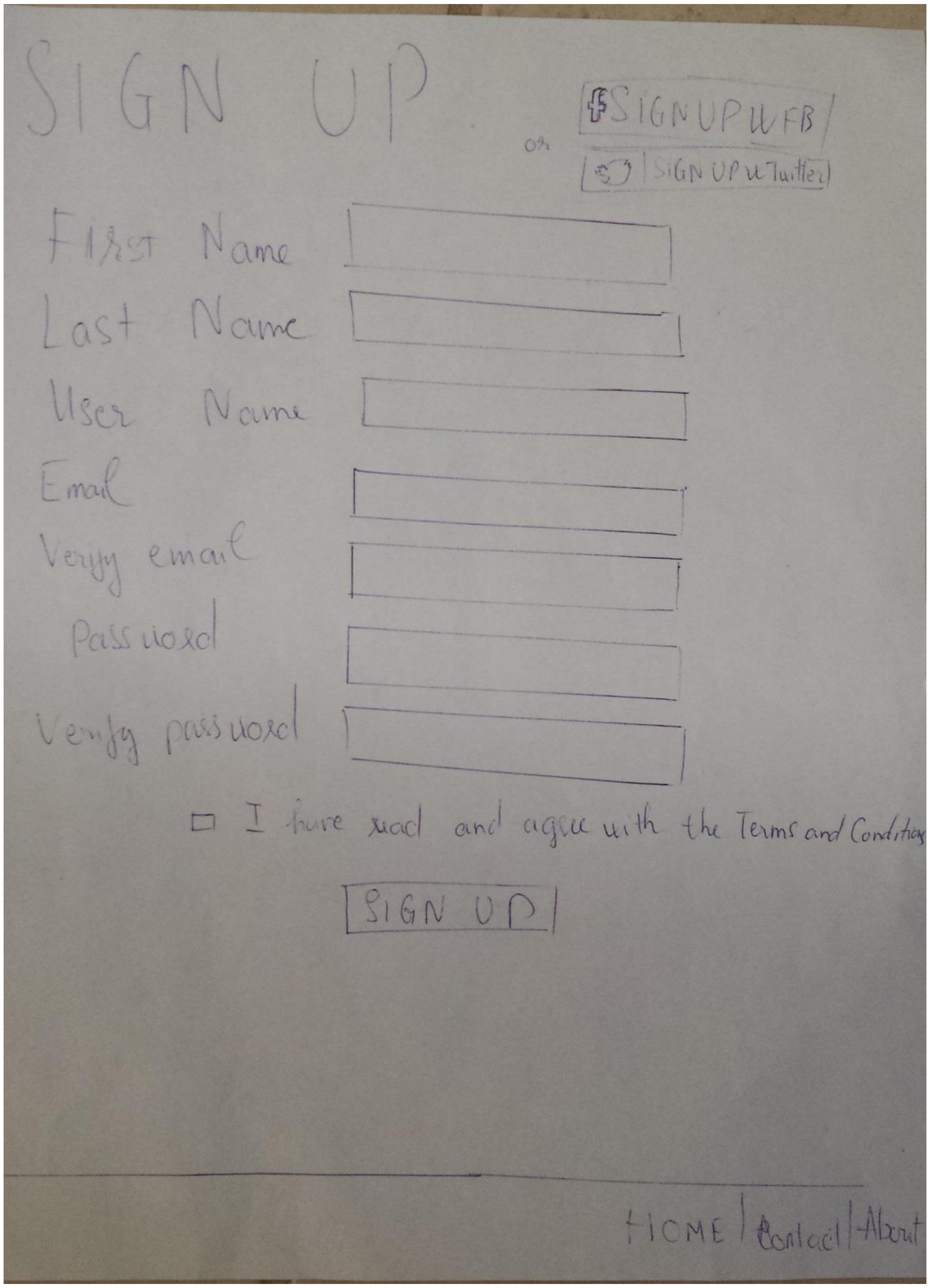
ID	PW	Requirement
REQ-31	3	Responsive CSS / Cross Device Compatibility
REQ-32	4	Rapid dynamic data updates

Home page

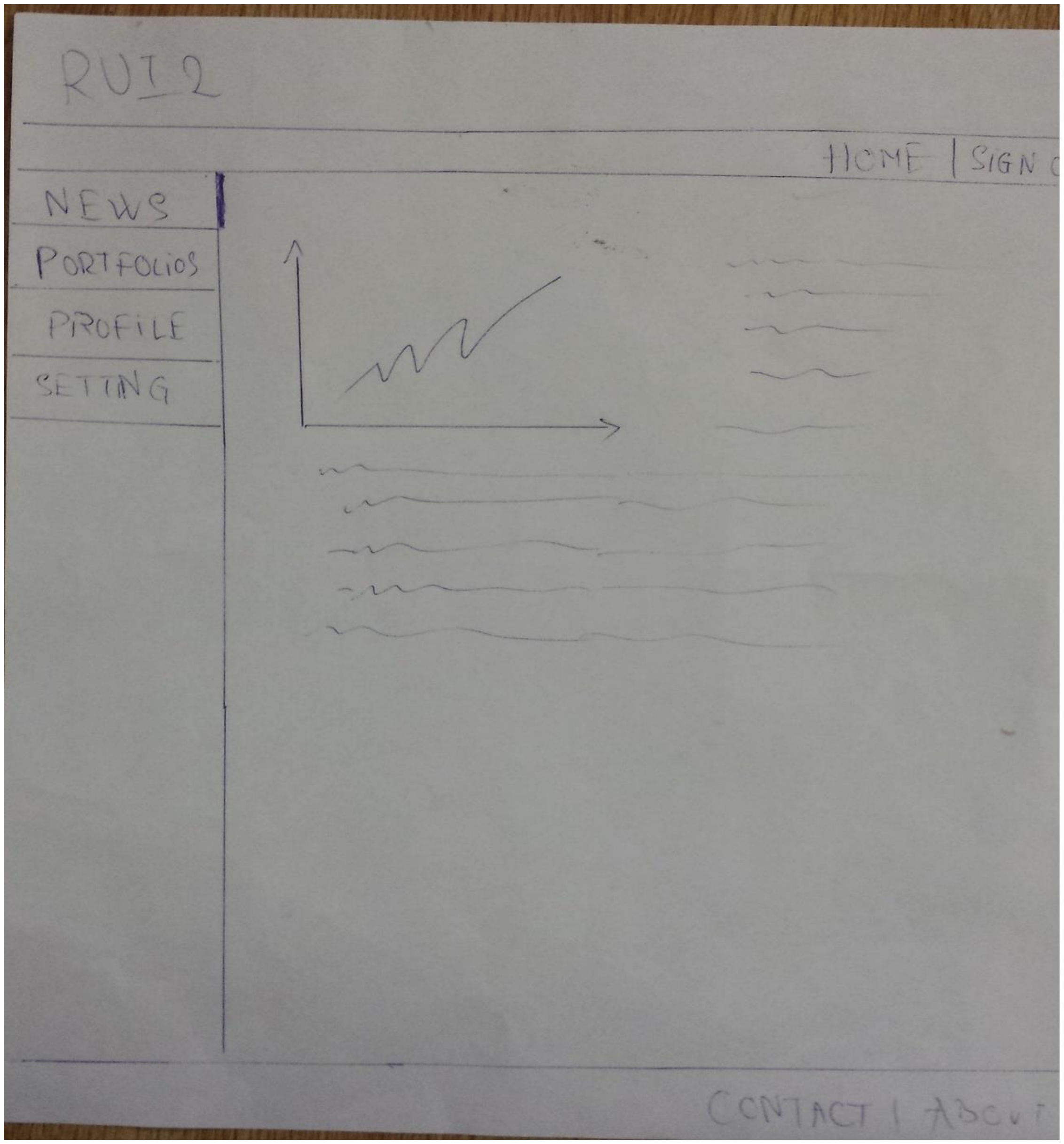


RUI 2
HOME | LOGIN | SIGNUP
NEWS
- Latest trends
- Sentiment analysis results
CONTACT | About

Signup Page


SIGN UP
or
SIGN UP w FB
SIGN UP w Twitter
First Name
Last Name
User Name
Email
Verify email
Password
Verify password
I have read and agree with the Terms and Conditions
SIGN UP
HOME | Contact | About

User’s Page


RUI2
HOME | SIGN
NEWS
PORTFOLIOS
PROFILE
SETTING
CONTACT | ABOUT

Functional Requirements Specification

Stakeholders:

Internal stakeholders:

- Owners: people who legally have the right to possess the web application. Owners' goals is contributing the software to multiple universities across the world, keep low cost maintenance and customer satisfaction.

- Managers: person who is in charge of affairs, , resources and expenditures of the web application. They interests include performance, growth, customer satisfaction, profit, cost, employees, and demand.

- Employees: person who is in charge of maintaining the web application, they will provide support to system administrators (university professors) , they will perform software updates to minimize troubleshooting. Their interests are reliability, working conditions, salary, working hours, job security, and benefits.

External stakeholders:

- Customers (Universities, students and professors): A person who is registered in the web application and creates his own stock portfolio, a system administrator (professor) can register without having stock portfolio and universities that will host the web application on their web page. . The customer's interests include software value, quality, reliability and service.

- Advertisers: An academic institute who calls the attention of the students to participate in daily activities. Finance companies who want to address students. Advertiser's interests are number of customers, detecting leading players.

- Ministry of Education- The executive policy making body in the United States. Ministry of education goal is to provide better finance studies platform in universities without putting in risk student money.
- Competitors: A company providing the identical products to universities. Their interests include profit, demand, customers and quality.
- Stock Researcher: An individual who researches the human behavior in the field of investment in stocks. His or her interests include investors, human behavior, and investing strategies.

- Actors and Goals:

- User: Any student which enrolled to Finance class.
  Type: Initiating
  Goal: Create an account

- Investor: A student who is authenticated using the login system and is interacting with the system, portfolio.
  Type: Initiating
  Goal: Login,join a league, monitor stocks and portfolio, buy and sell.

- League Administrator: Class professor who is authenticated using the login system and is interacting with the system.
  Type: Initiating.
  Goal: Login,create new league, monitor students portfolios, create and delete accounts.

- Yahoo Finance: The external source where real-world stock quotes are obtained at periodic time intervals.
  Type: Participating
  Goals: None.

- Database: A place where information about the various stocks such as price quotes,ticker symbols, and market name, are stored. Also, it stores a list of investors currently part of the system and their settings such as user id, passwords, email address, and other personal details.
  Type: Participating
  Goal: None.

- Email Server: A machine responsible for sending messages to investors via E-mail and SMS.
  Type: Participating
  Goal: None.

- Advertiser: An individual who interacts with the system through a user account and posts university related activities. Finance companies who interacts with the system through a user account and posts job openings.
  Type: Initiating

Goals: Login. Post Advertisements. Remove Advertisements.

Use Cases

UC-1: Register -- Allows a student to register an account and enter a game by filling out a form and entering a class code given out by a professor/league administrator
Derived from REQ-3

UC-2: Make Trades -- Allows a player to initiate trade orders, the system will then respond appropriately based on market conditions and status
Derived from REQ-15 & REQ-16

UC-3: Setup League -- Allow a league administrator to start a game and initialize settings
Derived from REQ-9 & REQ-10

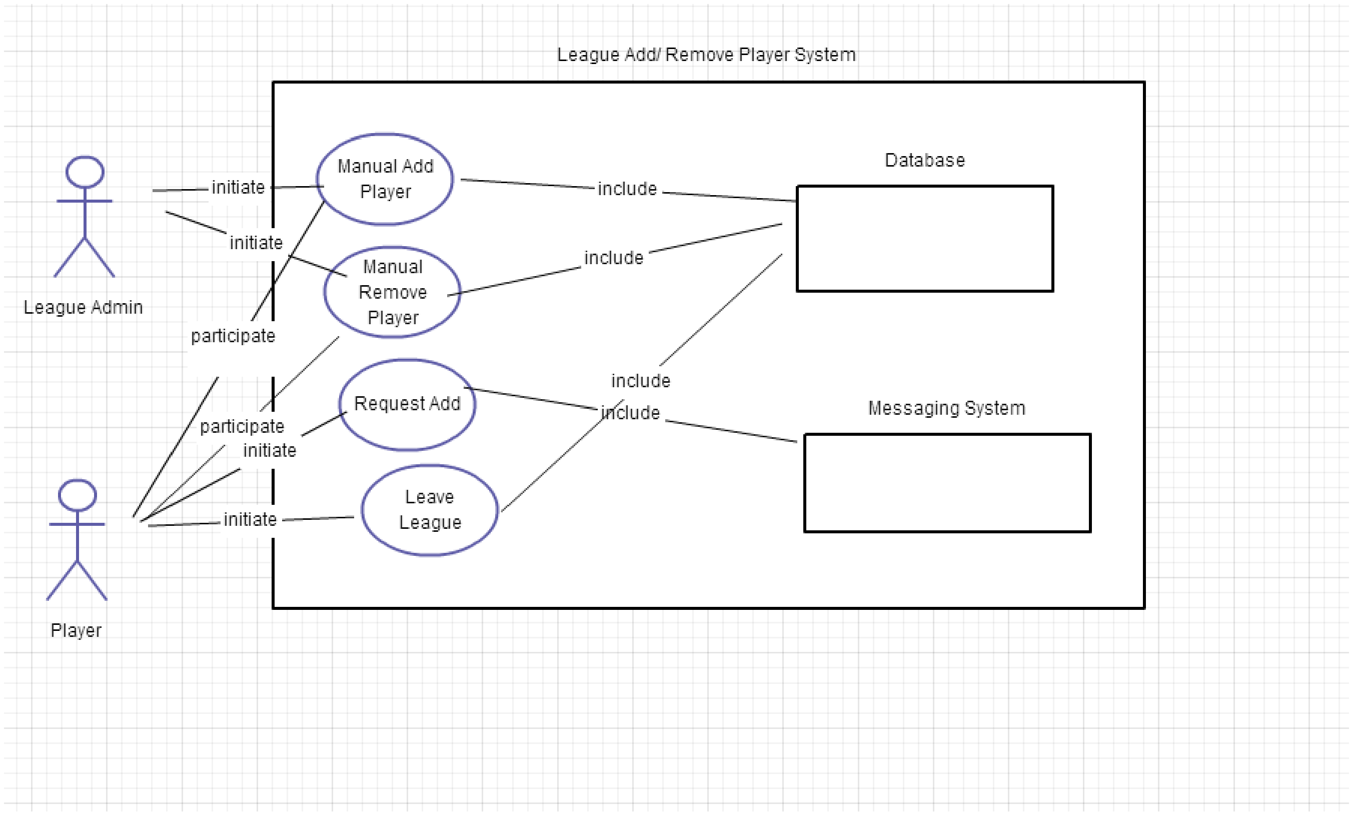
UC-4: View Profile -- Allows all users in a league to view the profiles of all other players in the league.
Derived from REQ-13 & REQ-14

UC-5: Manually add player to League -- Allow a league administrator to manually add a student’s account.
Derived from REQ-11

UC-6: Player Joining a League -- Allow a student account to join a league using a password supplied to them by their teacher, the league administrator.
Derived from REQ-11

UC-7: View League Stats -- Allow a player to exam a statistics and leaderboard page for the league they are in.
Derived from REQ-20

UC-8: Twitter Research -- Allows a user to make use of twitter streams for market research through a custom tool.
Derived from REQ-17 and REQ-18


League Add/ Remove Player System
League Admin
Player
Manual Add Player
Manual Remove Player
Request Add
Leave League
Database
Messaging System
initiate
initiate
participate
participate
initiate
initiate
include
include
include
include

Req't	PW	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8
REQ1	5	X							
REQ2	5	X							
REQ3	5	X	X						
REQ4	5	X					X		
REQ5	4	X							
REQ6	5								
REQ7	5								
REQ8	5								
REQ9	5								
REQ10	5			X					
REQ11	4		X	X					
REQ12	3		X			X			
REQ13	3						X	X	
REQ14	3				X			X	
REQ15	3				X				
REQ16	2								
REQ17	3								
REQ18	2		X						X
REQ19	3		X						X
REQ20	2								
REQ21	2								
REQ22	1							X	
MAX PW		5	5	5	3	4	5	3	3
Total PW		25	20	10	6	4	9	8	5

Use Case UC-2: Make Trades

Related Requirements: REQ-10, REQ-15, REQ-16
Initiating Actor: Player
Actor's Goal: Initiate a trade of some sort in a simulated stock market
Participating Actors: Database, Finance API
Preconditions: Player is a registered student
Successful End Condition: Database is updated to reflect portfolio and capital changes
Failed End Condition: Player is notified their trade could not be made and is not charged
Extension Points:

Flow of Events for Main Success Scenario
include::Login (UC-XX)

← 1. System displays an interface for the player to enter a stock ticker, share amount and select a market order type
→ 2. Player makes their selections on the market order screen
← 3. System polls the Finance API for market status
→ 4. Finance API indicates last known market price for ticker and market status
← 5. System writes trade to database, completing it if market is open and user has enough money (charging users account and adding to their portfolio). If market is closed it adds it to a pending trades list, to be completed when the market is open.
← 6. System outputs to display indicating to player the status of their trade.

Flow of Events for Extensions (Alternate Scenarios):

step 5a: User didn't have enough money
← 6. System indicates to user that they had insufficient funds through main

display and informs them the market order could not be made.

step 4a: Finance API couldn't be reached
← 6. System emails system administrator informing them of the error. Then indicates to user that stock prices are temporarily unavailable, and ask if they want to leave order pending or cancel it.
→ 7. User chooses one of two options.
← 8. System either cancels order or adds it to pending list on server.
← 9. System indicates to user that it has completed their choice successfully

Use Case UC-5: Administration- Manually Add Player to League

Related Requirements:	REQ-11
Initiating Actor:	League Administrator (LA)
Actor's Goal:	Add a player to the league
Participating Actors:	Database, Player
Preconditions:	Player exists Player is not in league database
Successful End Condition:	Player is added to league database

Flow of Events for Main Success Scenario:
← 1. System notifies league admin of add request and provides the player's profile and ID
→ 2. LA goes to the admin panel of the requested league, inputs player ID into "ADD PLAYER" functionality and submits
← 3. System verifies player ID is valid and sends player a join league request.
→ 4. Player verifies request
← 5. System enrolls player in the specified league, and notifies LA of confirmation

Flow of Events for Extensions (Alternate Scenarios):
2a. LA inputs invalid player ID
← 1. System detects invalid ID and displays an error

2b. LA inputs valid player ID that is already enrolled
← 1. System detects player is already enrolled and displays an error

4a. Player denies request
← 1. System notifies LA of rejected request

Use Case UC-6:	Player requests to join league
Related Requirements:	REQ-21
Initiating Actor:	Player
Actor's Goal:	Automatically enroll in specified league
Participating Actors:	Database, League Admin
Preconditions:	Player is logged in Failed password attempts is zero
Successful End Condition:	Player is added to league database

Flow of Events for Main Success Scenario:
← 1. System queries user for league ID
→ 2. Player inputs a league ID
← 3. System verifies league ID and queries user for league password
→ 4. Player inputs a league password and confirms action
← 5. System verifies password and enrolls player in league

Flow of Events for Extensions (Alternate Scenarios):

2a. Player enters invalid ID

← 1. System detects invalid ID, displays error, and prompts for re-entry of league ID

2b. Player enters valid ID, however is already enrolled in league

← 1. System detects valid ID and erroneous request. Error message is displayed

4a. Player enters invalid league password

← 1. System detects invalid password and gives the player the following options

- Re-enter league password
- Message League Admin for Administrative Manual Add

← 1a. System detects failed password attempts has reached limit.

- The event and offending player info is logged.

→ 1b. Player re-enters league password, return to main success state 3

→ 1c. Player request to contact league Admin

← a. System provides player with request form

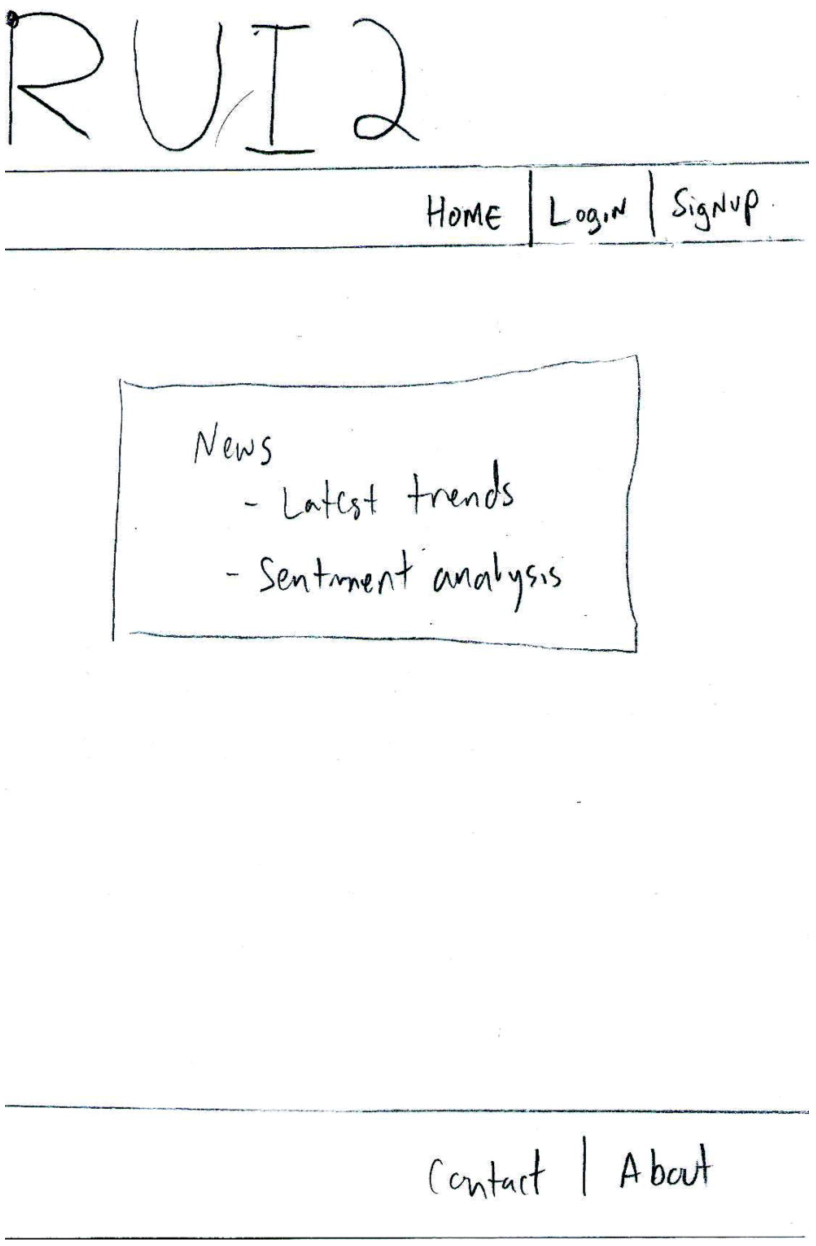
→ b. Player writes and submits message

c. Refer to Use Case 5

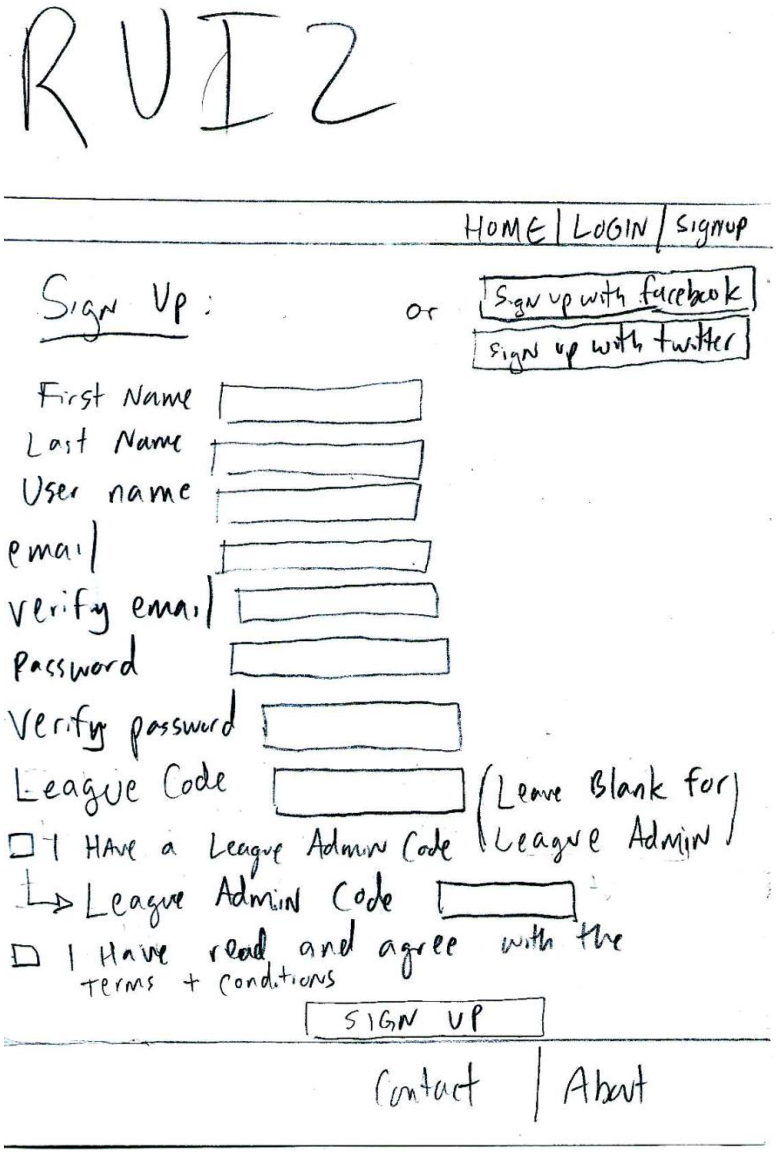
User Interface Specification

Preliminary Design

UC1 - User Registration:



For the first time user, they can click Signup on the welcome page which redirects him/her to the signup page.



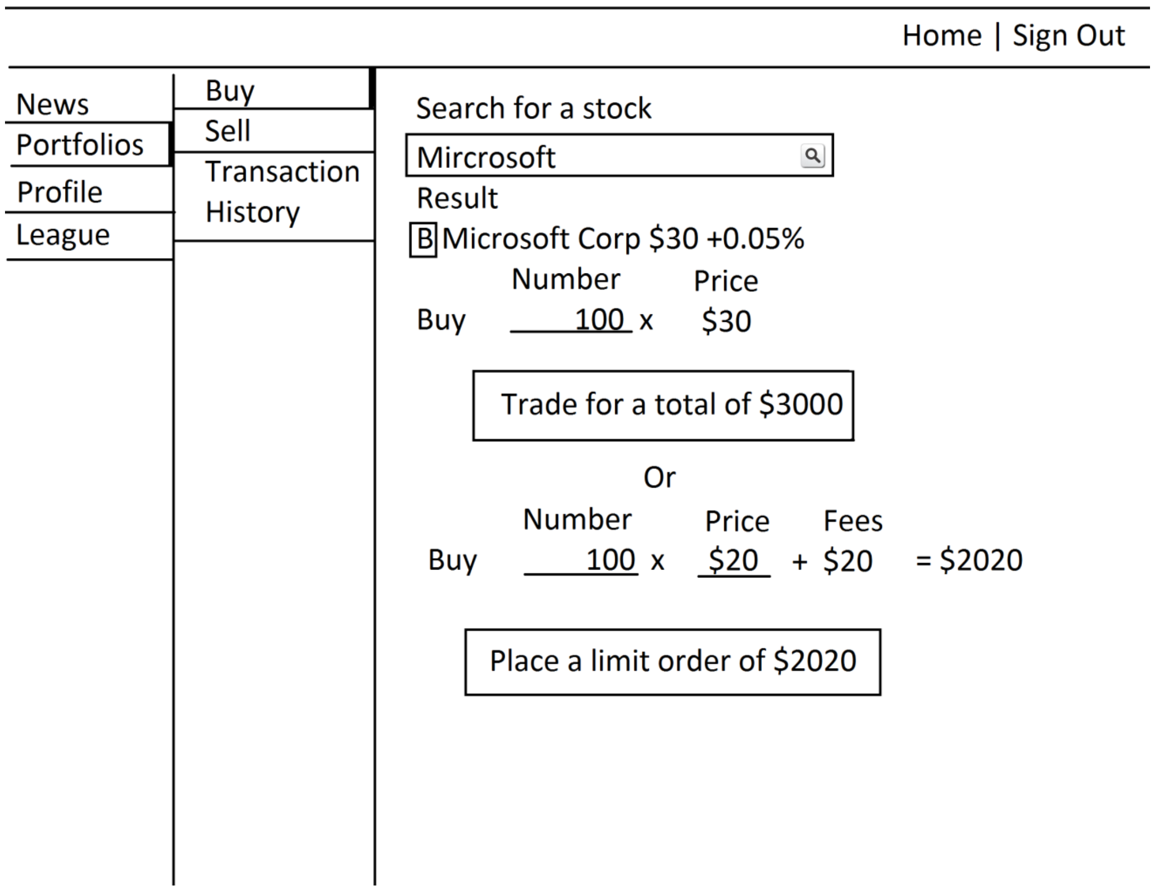
The player enters his/her first and last name, desired user name, email and password (and verify). The player will also be given a league code by the league administrator. The player must click the: I have read and agree with the terms and conditions before enabling the signup button. Alternatively the player can click signup with Facebook or signup with twitter, and the first 7 fields will be swapped with just their Facebook login and password.

When the league administrator signs up, he/she will enter the identical information to a player, but leave the league code blank, and click the box “I have a league admin code”, which will bring up another text box field ‘League Admin Code’, in which he/she will enter the league administrator code given to them by the system administrator. This will entitle the league administrator to be a super user and have access to the extra administrative options.

UC-2: Make Trades:

The player can make purchases by selecting PORTFOLIOS/ BUY tab. In the Buying page, they can search for a stock by typing its name into the search box. Clicking the B button on the left of the result will make the program receive the information of the stock. Player will then proceed with the transaction by typing in the amount they want to buy. Finally they can either make an immediate order by clicking “Trade for a total of __” button or select “Place this order as a Stop/Limit order” which they can later view in transaction history.

RUI2



Search for a stock

Mircrosoft

Result

B Microsoft Corp $30 +0.05%

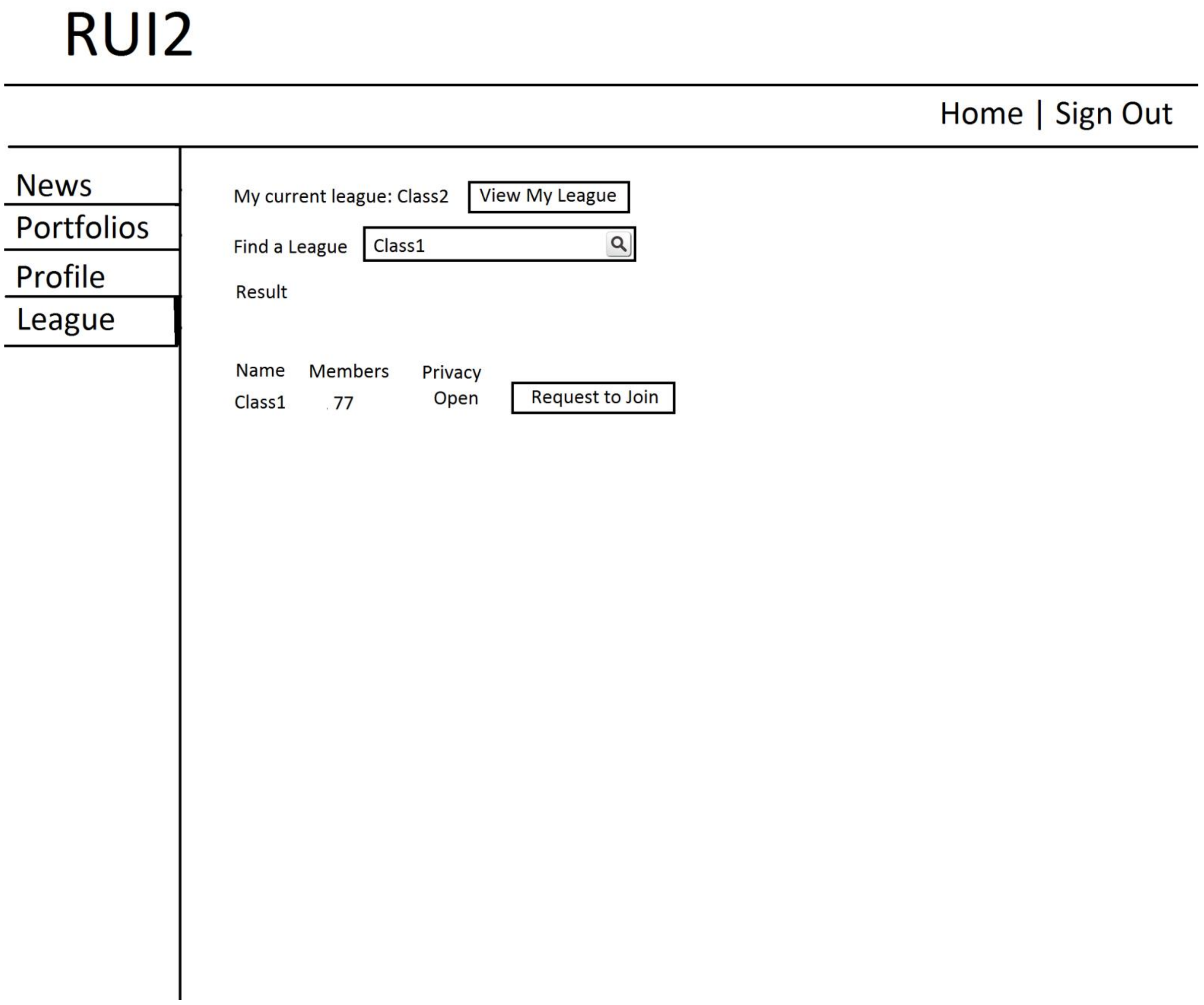
	Number		Price
Buy	100	x	$30

Or

	Number		Price		Fees	
Buy	100	x	$20	+	$20	= $2020

Similarly for Selling, players can click the “S” button left to an owned stock to sell them. They can trade it immediately with the market price or place a Stop/Limit order with an additional fee

UC-4: Allows all users in a league to view the profiles of all other players in the league.

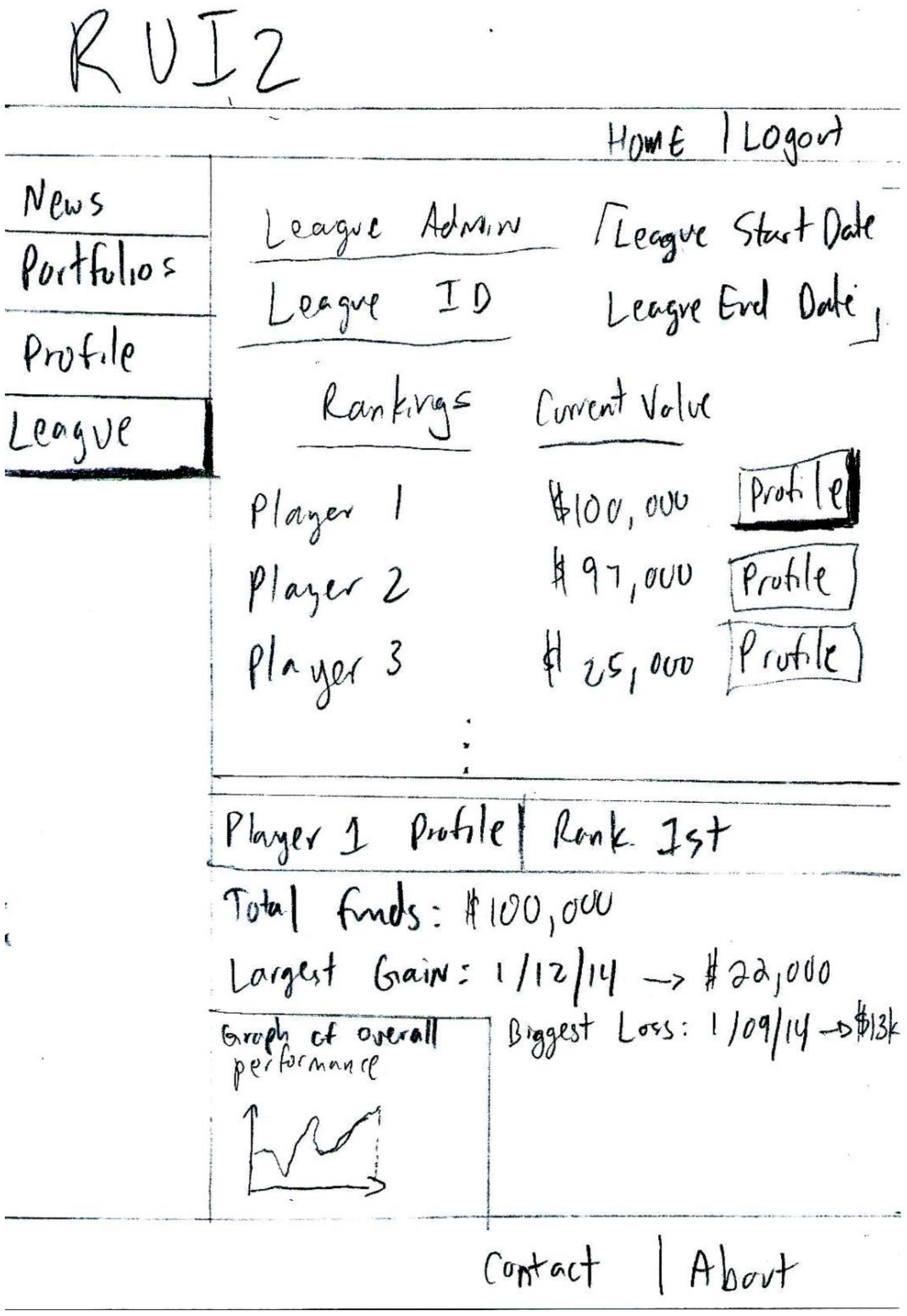


My current league: Class2

Find a League Class1

Result

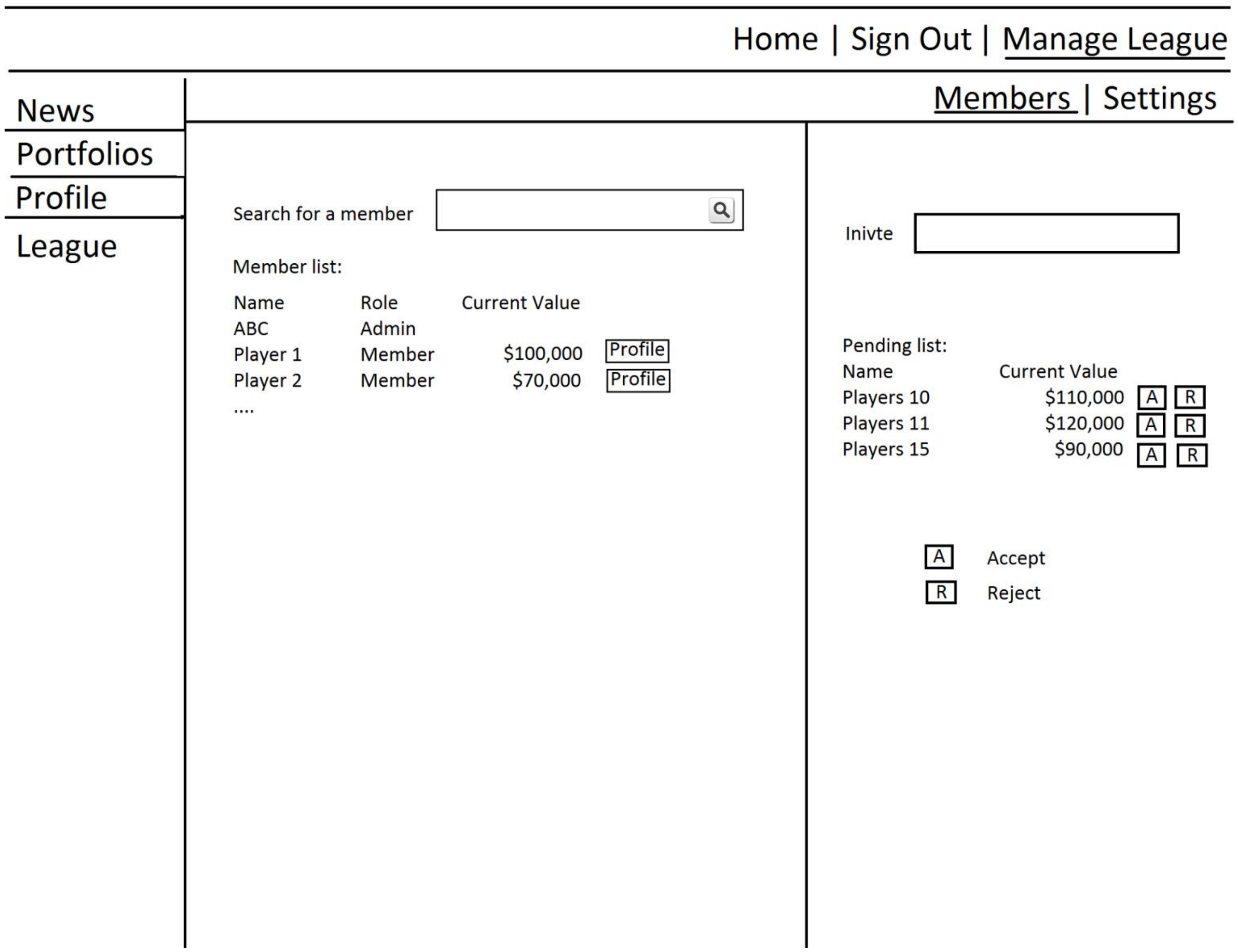
Name	Members	Privacy
Class1	77	Open



Once the player has logged in they can click the league tab on the left then select "View My League" button, which will bring up the current league admin's name, the league id, the start and end date of the league, and rankings of all the players in the league. Aligned with each of these players will be a button to view their individual profile. Whichever profile button is selected, that players profile will be displayed underneath the rankings. This profile will contain the players ranking, their total funds, their largest day for gain/loss and a graph of their overall performance for the whole league period. By default the league leader’s profile will be displayed when the league page is first loaded.

UC-3: Setup League -- Allow a league administrator to start a game and initialize settings
Since a league can have a large number of member, say 100 for a class, having a member’s profile displayed at the bottom of the page may give the admin a frustrating experience, especially moving back and forth viewing different users’ profile(player 30 and player 70 for example). Instead, i suggest that the profile button will lead the user to the player’s profile page in which they can see the mentioned profile in details

RUI2



Member list:

Name	Role	Current Value	
ABC	Admin		
Player 1	Member	$100,000	Profile
Player 2	Member	$70,000	Profile
....			

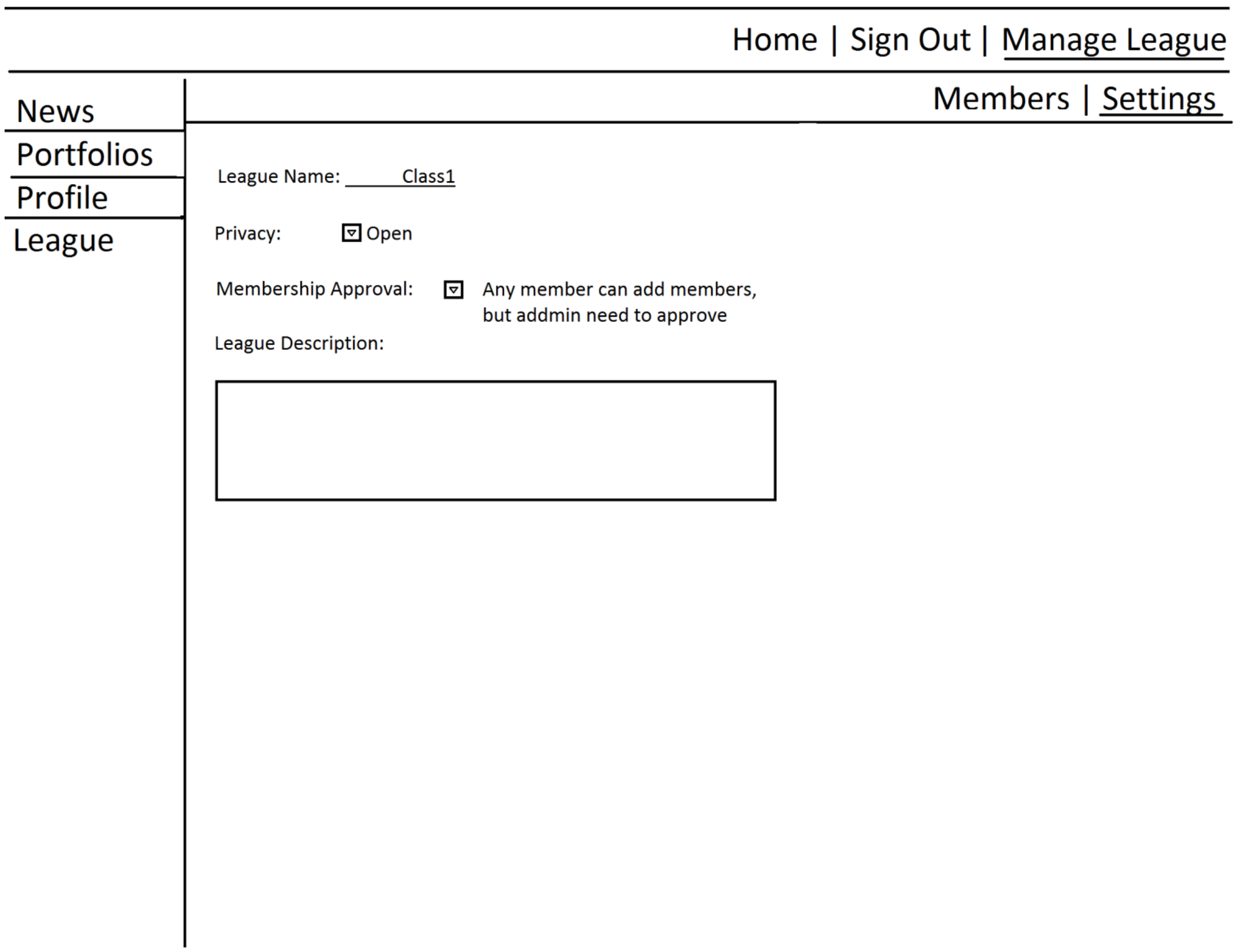
Pending list:

Name	Current Value		
Players 10	$110,000	A	R
Players 11	$120,000	A	R
Players 15	$90,000	A	R

Selecting Manage League/Members , the league admin can get an overview of the league’s members, he/she can view any member’s profile by click the profile button next

to the member on the member's list. The admin can also find a specific player quickly by typing the name into the search box.
The league's admin can also invite people to join the league by typing the player's username into the box next to invite. There is a pending list which list the players that want to join the league, the league admin can either accept or reject the request by click the corresponding button. (this seems to meet UC-5)

RUI2



League Name: Class1

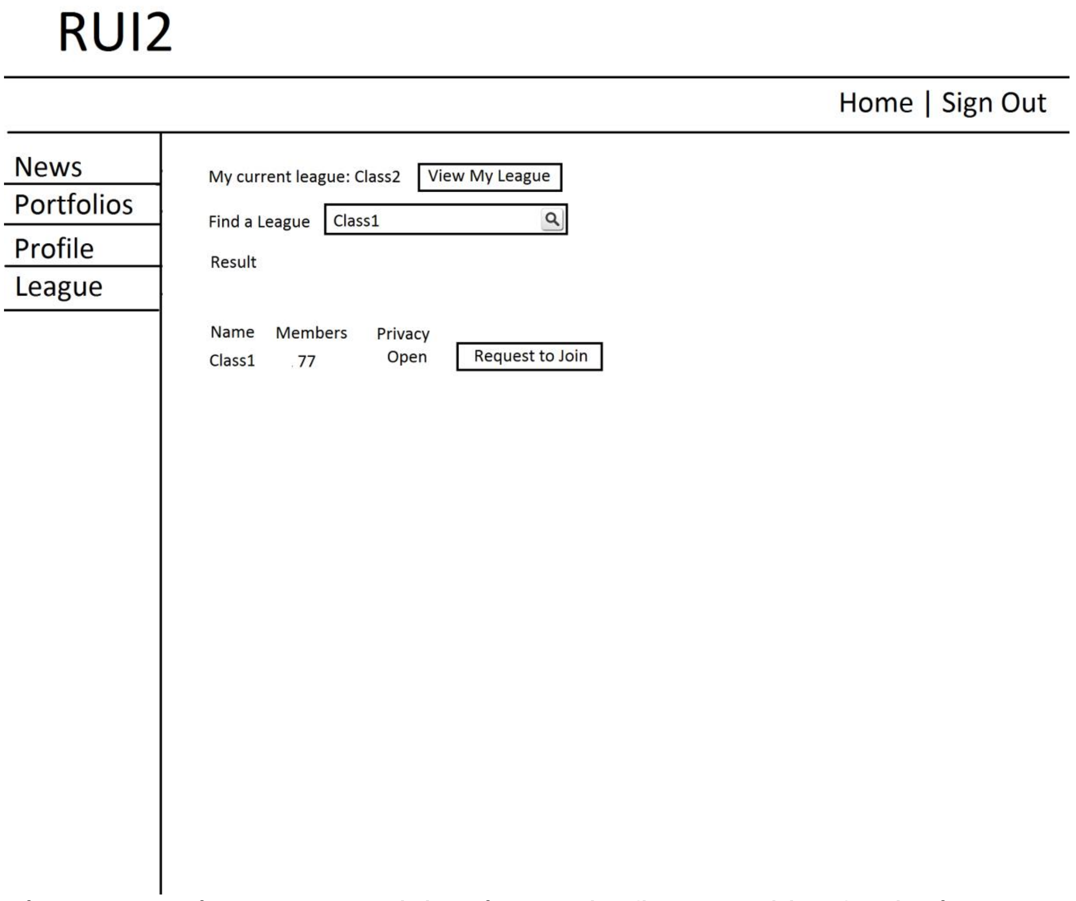
Privacy: Open

Membership Approval: Any member can add members, but addmin need to approve

League Description:

And if the league admin wants to change the league's name, privacy, membership approval setting and write a description for the league, he/she can choose the setting tab.

Use Case UC-6: Player requests to join league



My current league: Class2

Find a League: Class1

Result

Name	Members	Privacy
Class1	77	Open

Players can also request to join a league by first searching for the league using its name then click the "Request to join" button next to the result to send the request

User Effort Estimation

1. Sign Up: 4 mouse clicks, 86 keystrokes
a. Click Sign Up on the right corner of header
b. Data of Users : 20 keystrokes of first name and last name, 10 keystrokes of user name, 15 keystrokes of email,15 keystrokes of verify email, 11 keystrokes of password, 11 keystrokes of verify password.
c. Click League admin code
d. Input the league admin code, 4 keystrokes
e. Click to agree with terms and conditions
f. Click sign up to be done.

2. Login: 1 click, 21 keystrokes
a. Click Login on the right corner of header
b. 10 keystrokes of user name, 11 keystrokes of password.

3. League Portfolio: 2 clicks, 10 keystrokes
a. Click Portfolio at home page on left side.
b. put amount of shares want to sell/buy
c. click confirm what we did.

4. Setting: 2 clicks,
a. Click Setting at home page on left side
b. Click parts you want to setting.

5. Trade: 2 clicks, 10 keystrokes
a. click the company
b. choose buy/sell
c. put amount of shares you want to buy/sell, 10 keystrokes
d. click confirm

6. Profile: 1 click
a. click profile to see information
b. Edit information

7. Create new League: 2 clicks, 15 keystrokes
a. Click League tab
b. Enter league name
c. Click confirm

$$UCP = UUCP * TCF * ECF$$

Where UUCP = UAW + UUCW

Unadjusted Actor Weight:

Actor Name	Description of Relevant Characteristics	Complexity	Weight
Admin User	Admin user is using administrative web pages to interact with the system which is a form of GUI	Complex	3
Player User	The player is using the standard web pages of our application to make trades, a GUI	Complex	3
Yahoo! Finance	The service is accessed for market data using a provided API	Simple	1
MySQL Database	Database is interacted with through a database protocol	Average	2
Twitter	Twitter is accessed through a provided API	Simple	1

UAW = 2*complex+1*average+2*simple = 2*3+1*2+2*1 = 10

Unadjusted Use Case Weight:

Use Case	Description	Complexity	Weight
Register (UC-1)	Simple user interface. 5 steps to main success. 3 participating actors	Simple	5
Make Trades (UC-2)	Complex user interface. 15 steps to main success. 4 participating actors	Complex	15
Setup League (UC-3)	Complex user interface. 10 steps to main success. 3 participating actors	Complex	15
View Profile (UC-4)	Moderate user interface. 4 steps to main success. 2 participating actors	Average	10
Manually Add Player to League (UC-5)	Simple user interface. 3 steps to main success. 2 participating actors	Simple	5
Player Joining a League (UC-6)	Moderate user interface. 7 steps to main success. 3 participating actors	Average	10
View League Stats (UC-7)	Moderate user interface. 7 steps to main success. 2 participating actors	Average	10
Twitter Research (UC-8)	Complex user interface. 15 steps to main success. 3 participating actors	Complex	15

UUCW = 3*complex+3*average+2*simple = 3*15+3*10+2*5=85

Therefore:
UUCP = UAW+UUCW = 10+85=95

Technical Complexity Factor

Technical Factor	Description	Weight	Perceived Complexity	Calculated Factor
T1	Distributed due to web based nature	2	3	6
T2	Good performance is critical in order for trades to be accurate	1	4	4
T3	End-user expects efficiency but there are no exceptional demands	1	3	3
T4	Significant complexity in trade processing and twitter integration	1	4	4
T5	No reusability requirement	1	0	0
T6	Not very important only done once	0.5	2	1
T7	Very important since it is targeting non technical users	0.5	5	2.5
T8	Must work across a wide range of browsers and devices	2	4	8
T9	Must be easy to upgrade	1	3	3
T10	Concurrent use is required	1	5	5
T11	Security is of reasonable concern	1	3	3
T12	No third party usage	1	0	0
T13	No significant training necessary	1	2	2

TCF = 0.6+0.01*41.5=1.015

Environmental Complexity Factor

Environmental Factor	Description	Weight	Perceived Impact	Calculated Factor
E1	Unfamiliar with the development process; all first timers	1.5	0	0
E2	Very little familiarity with the stock market	0.5	1	0.5

E3	Moderate knowledge of Object oriented approach	1	2	2
E4	First time lead analyst	0.5	0	0
E5	Motivated within reason of other assignments	1	2	2
E6	Requirements evolving with time	2	3	6
E7	Team entirely consist of part time workers	-1	5	-5
E8	Programming languages used are complex and unfamiliar	-1	4	-4

ECF = 1.4+(-0.03*1.5) = 1.355

UCP = UUCP*TCF*ECF = 95*1.015*1.355 = 130.66

Project Duration = UCP*PF = 130.66 use case points *28 hours per point = 3,658.36

Domain Model

Domain Analysis

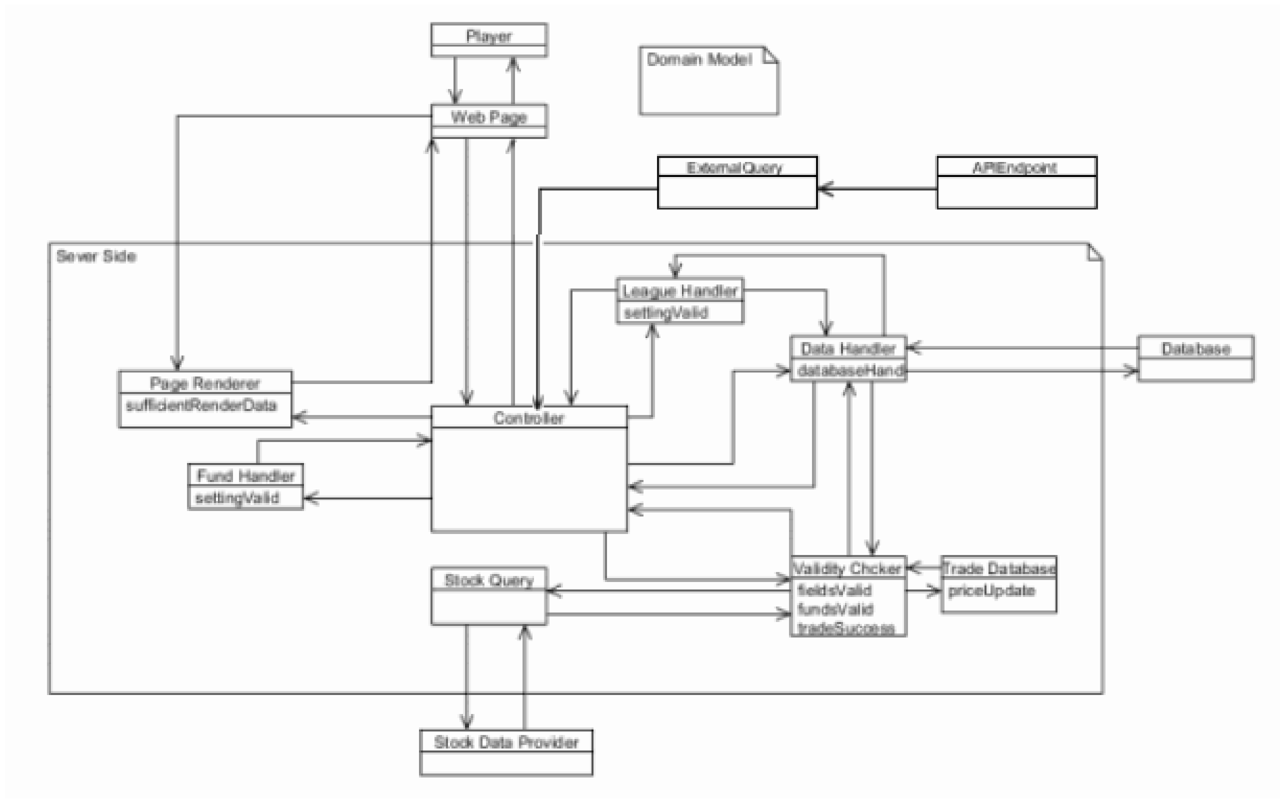


Figure 1: Domain Model

This is our general domain model which shows important objects and their interactions with others.

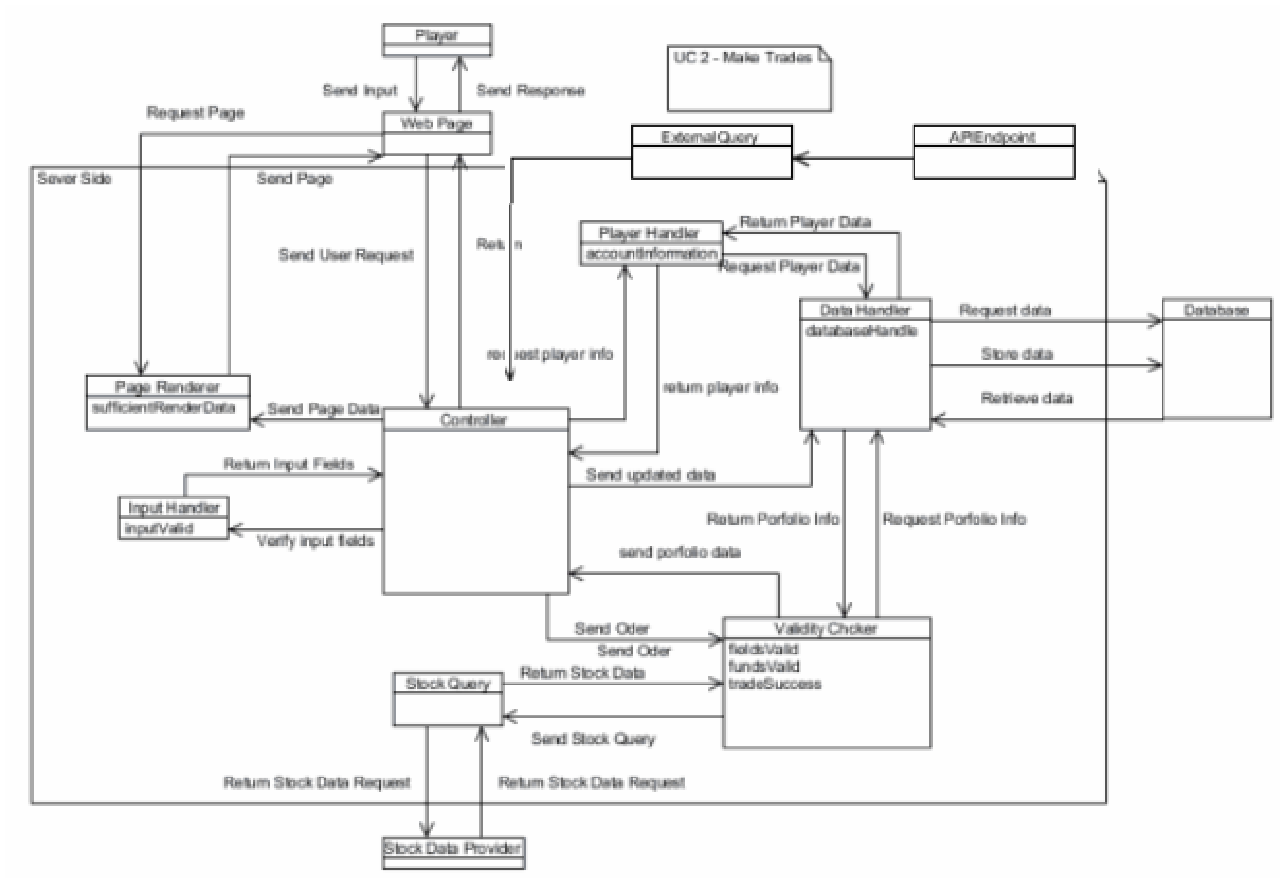


Figure 2 - Make Trade Model

Figure 2 represents our UC-2 Make Trades. When a player make a trade order on the web page, the order is sent to the controller, which directs the request to Validity Checker to check whether the request is valid and the player has enough funds or has enough stocks to sell. If all of above conditions are met, the validity sends a request to Data Handler to make the transaction by updating the player's data. The controller notified by Validity Checker that the transactions are made successfully, then send the updated data about player to Page Renderer to generate updated info displayed on user's web page.

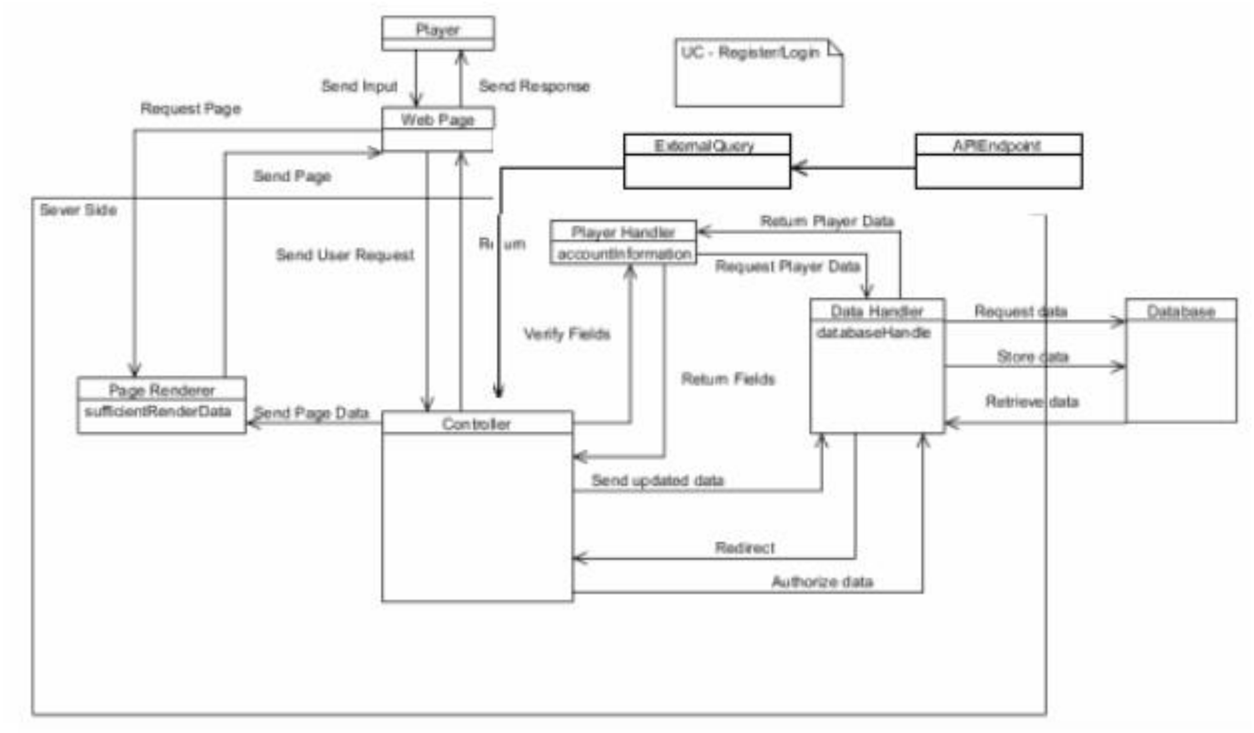


Figure 3 - Register/Login Model

Figure 3 represents UC-1 Register/Login
Register: Player fill in the form and send a register request through web page to the Controller. The Controller conveys the request to Player Handler, whom first verifies player's data validity and availability before sending creating new profile request to data handler. After the data processing processes are finished, the result is sent to the controllers whom relay the data to Page Renderer to generate a page displayed later on Player's web page.

Login: Similar to Register, player fills in form and sends a login request to controller. Controller relay the request to Player Handler to verify username and password. If the player typed in the
correct combination, the player is granted access to the system.

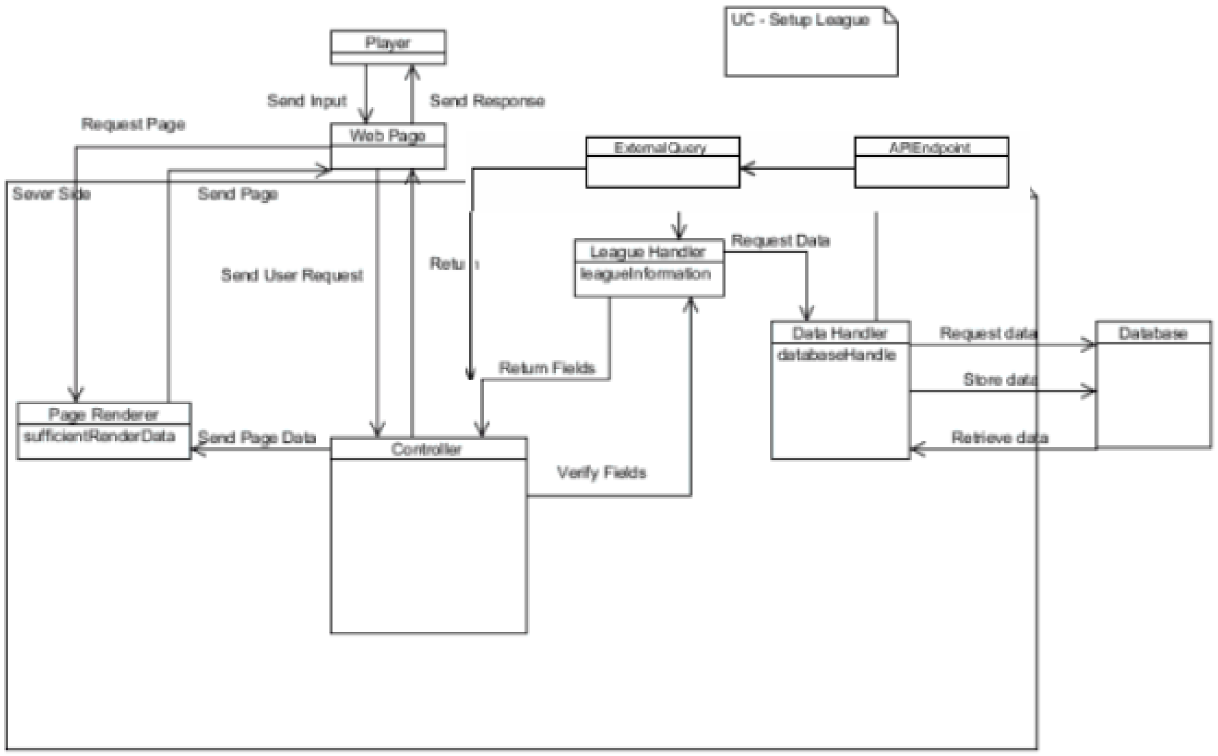


Figure 4 - League Model

Figure 4 represents UC-3 Setup League. To create a new league, the administrator (user) fill out a form then send the form via web page to controller. The controller replays the request to create a new league to League Handler, whom checks the validity) of the info (by comparing with data from database with help of Data Handler, then request an update in data. If the creating new league process is successful, the administrator will receive the updated info on web page with the data generated using page renderer

.

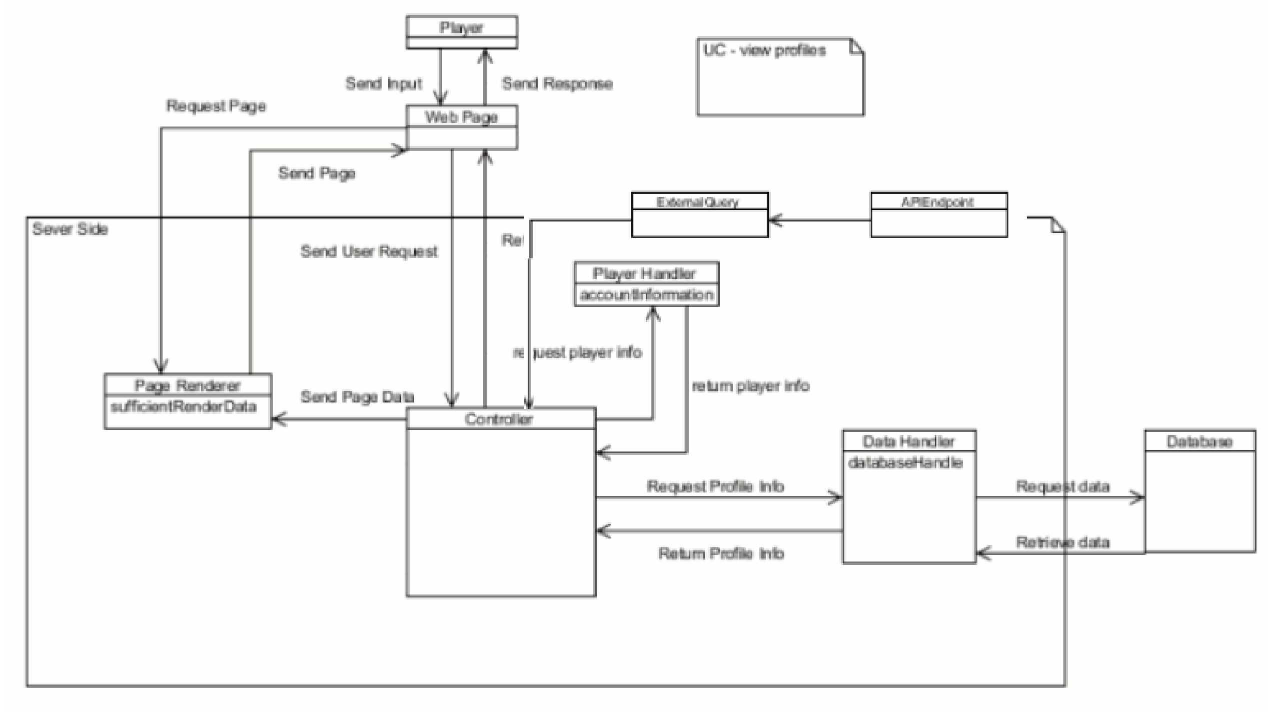


Figure 5 - View Profile Model

Figure 5 represents UC-4: Viewing player's profile. The player first sends the request to the system through Web Page. The request is directed to Controller. The Controller then request profile info from Data Handler. Data Handler conveys the request to Database and receive data back. The data is later sent to Page Renderer to Render the web pages contain for the player.

Concept Definitions

Player:
Definition: Somebody who wants to interact with the system
Responsibilities:

- Research stocks
- Request trades
- Manage Portfolios
- Manage League (League Administrator only)
- Navigate through website
- Request League info/ Player profile

Web Page:
Definition: a web document that is suitable for the World Wide Web and the web browsers.
Responsibilities:

- Receive player requests
- Send requests to Controller
- Send request for pages to PageRenderer
- Receive pages from the PageRenderer
- Pass data to the browsers to display to players

Page Renderer:
Definition: objects that process player's requests and render web pages from data accordingly
Responsibilities:

- Receive page data from the controller
- Process the data into easily viewable format
- Send the results to the Web Page

Controller
Definition: Objects that control the operations of the system base on the player's requests

Responsibilities:

- Receive player’s requests from the web page
- Send the page data to the Page Renderer
- Request/Receive Player Data from the Player Handler
- Send/Receive updated data to the Data Handler
- Send the player’s input to the Input Handler to verify

Validity Checker
Definition: Objects that test whether the inputs/ requests are valid before send the requests to other objects
Responsibilities:

- Receive requests/ orders from Controller and Verify them
- Send the requests/ orders to Stock query or Database handler depend on the requests/orders

Stock Query
Definition: Objects that fetch real-time stock info along with sentiment results
Responsibilities:

- Receive stock data request from Validity Checker
- Request/Receive stock data from stock data provider

Data Handler
Definition: Objects that deal with data related processes
Responsibilities:

- Receive requests and send data to controllers, League Handler and Player Handler
- Send request, retrieve and update data in database

League Handler
Definition: Objects that handle league related processes
Responsibilities:

- Receive league related requests and verify them
- Send requests to database handler
- Receive data from data handler

Stock Data Provider
Definition: Sources of real-time stock data that is accessible by the system
Responsibilities:
- Receive requests from Stock Query
- Send Data to Stock Query

Association Definitions

Concept Pair	**Association Description**	**Association Name**
Web Page ↔ Controller	Web page sends the user request to the controller to be processed and distributed. Controller sends return to web page to signify completion of process.	Send user request, return
Web Page ↔ Page Renderer	When the web page is signalled by the controller that the controller is finished processing the user request, the web page signals the page renderer to request the page. The page renderer sends the rendered page to the web page to be displayed.	Request page, send page
Controller→Page Renderer	Controller sends requests to Page Renderer	Convey requests
Controller ↔ League Handler	The controller sends the data inputted by the user (in this case a league admin) to the league handler to verify the validity (Correct data types,	Verify fields, Return fields

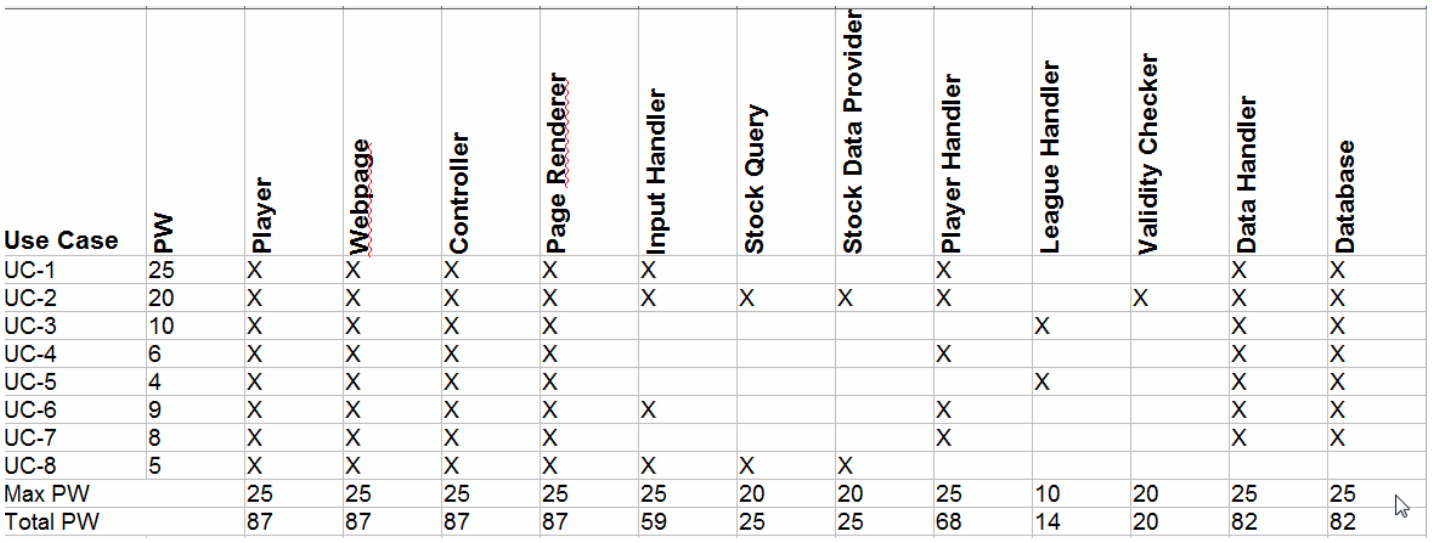
	valid dates etc.) and checks that it doesn’t conflict with other previous leagues in the database (conflicting league names for example).	
Controller ↔ Player Handler	controller sends requests for player data,	send request, receive data
Controller ↔ Data Handler	controller sends data requests from players to data handler and receive data back from data handler	send request, receive data
Data Handler ↔ Database	Request and retrieve data from database, store data	Convey requests
Validity Checker ↔ Stock Query	Validity Checker send validated requests to Stock Query and receive stock data back	Request data
Stock Query ↔ Stock Data Provider	Stock Query passess request to Stock Data Provider and receive back stocks data	Convey requests

Attribute Definitions

Concept	**Attribute**	**Meaning**
PageRenderer	sufficientRenderData	Used to determine whether the data retrieved is sufficient to generate web pages
InputHandler	inputValid	Used to determine whether the player’s input is valid(no special characters, appropriate length)
DataHandler	databaseHandle	Communicates back and forth between the database to

		handle requests.
Validity Checker	fieldsValid,fundsValid,tradeSuccess	Used to determine if the input fields are valid, if the player has enough funds to make the purchase and whether the trade is a success.
PlayerHandler	accountInformation	Player name, Role(whether the user is a league administrator), funds, transaction history, performance
LeagueHandler	leagueInformation	League’s name, members, status

Traceability Matrix



Use Case	PW	Player	Webpage	Controller	Page Renderer	Input Handler	Stock Query	Stock Data Provider	Player Handler	League Handler	Validity Checker	Data Handler	Database
UC-1	25	X	X	X	X	X			X			X	X
UC-2	20	X	X	X	X	X	X	X	X		X	X	X
UC-3	10	X	X	X	X					X		X	X
UC-4	6	X	X	X	X				X			X	X
UC-5	4	X	X	X	X					X		X	X
UC-6	9	X	X	X	X	X			X			X	X
UC-7	8	X	X	X	X				X			X	X
UC-8	5	X	X	X	X	X	X	X					
Max PW		25	25	25	25	25	20	20	25	10	20	25	25
Total PW		87	87	87	87	59	25	25	68	14	20	82	82

System Operation Contracts:

UC-1: Register:

- Preconditions:

-Players who have code given out by professor/league administration

- postconditions:

- Users have portfolio which contains users' information in the database

Operation: Joining a league

Precondition:

- The league exists
- The player has the league's password

Postcondition:

- Player successfully join in the league

UC-2: Make Trades:

- **Buy stocks**
- Preconditions:

- Users must have enough money to purchase stocks and bonds.
- There are Stocks that is available to purchase
- Transaction date is valid

- postconditions:

- Database has been updated with these changes
- Update Stocks inventory in database.

- **Sell Stocks**
- Preconditions:

- Users must have stocks that are available to sell
-

- postconditions:

- Database is updated with these changes
- Transaction date is valid

UC-3: Setup League :

- Preconditions:

- Input settings are valid

- postconditions:

- league informations are stored in database

Operation: manage league

- Preconditions:

- Users can control league privileges.

- Postconditions:

- League informations are valid in the database.

Operation: invite to League:

- Preconditions:

- Invitee must have account (users ID, league ID)
- invitee has valid portfolio

- Postconditions:

- None

Operation: View transaction:

- Preconditions:

- Initiating players is logged into the system

- Postconditions:

- Display information of players transaction

Operation: Administration and maintain website:

- preconditions:

- Player is league administration
- System administration is log in often

- -Postconditions:

- system administration control what website changing.

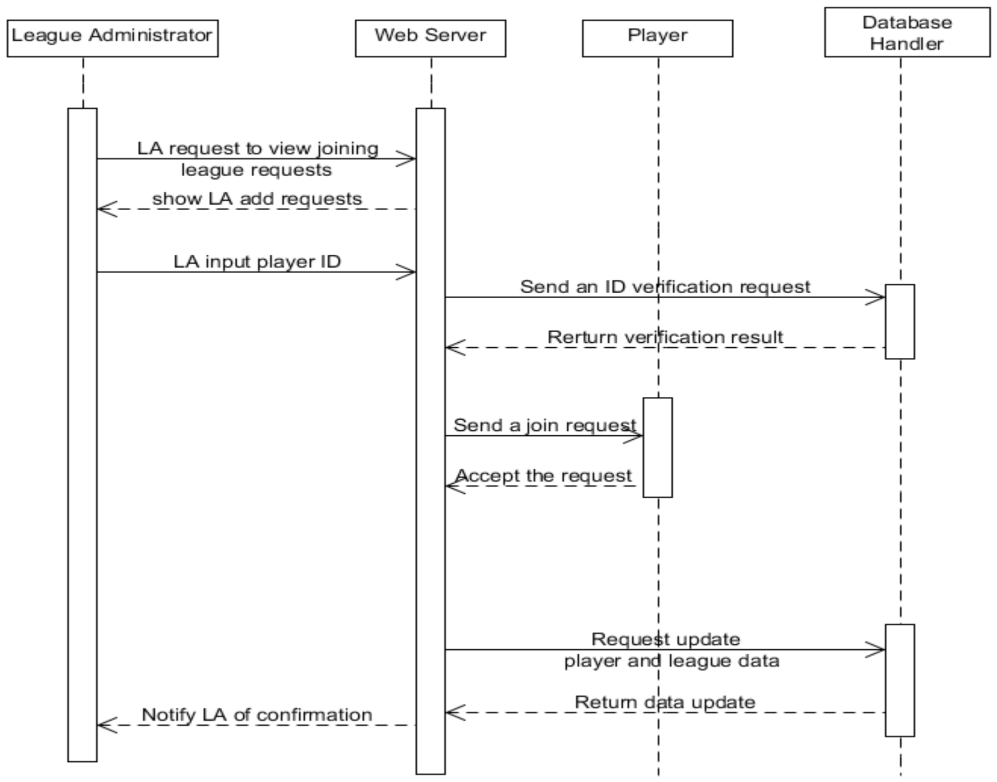
UC-4: View other player's profile

- preconditions:
  - Player is logged in
  - -The profile of the other player is available for viewing
- postconditions:
  - Display the profile of the other player

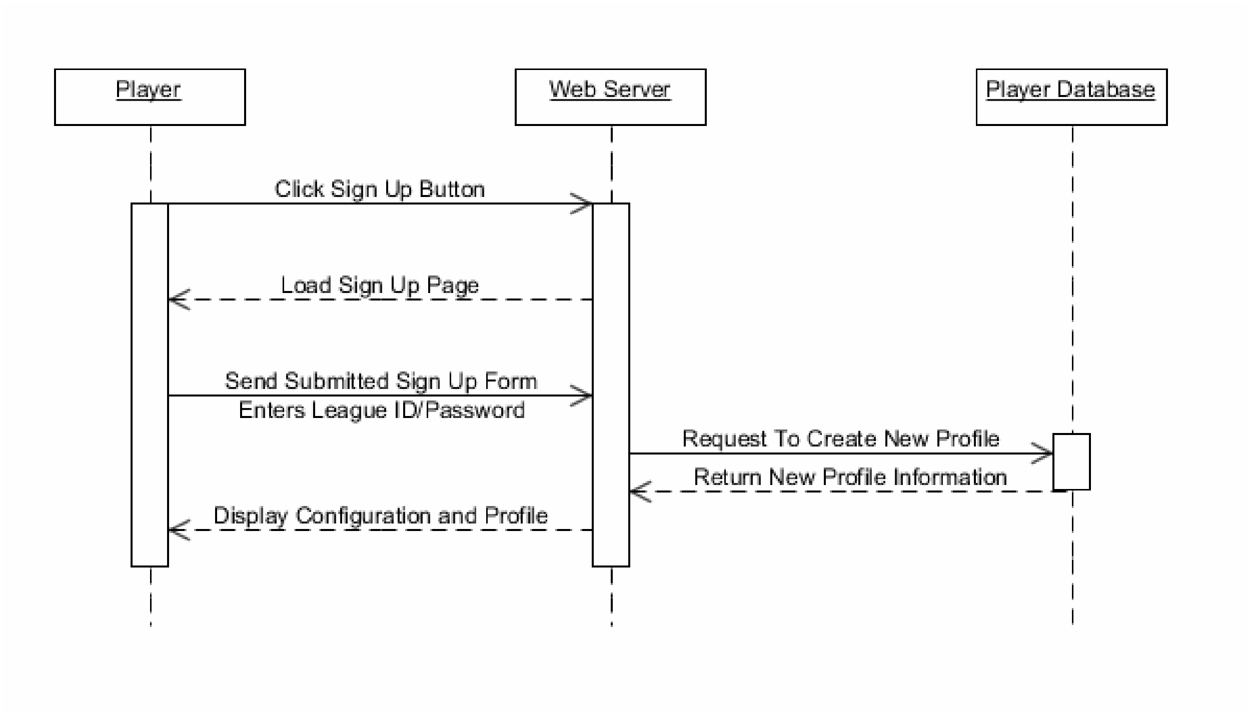
<u>Interaction Diagrams</u>

Administration- Manually Add Player to League UC-5:

To manually add a player to a league, the league administrator first requests the web server to view the player's league joining request. The Web server returns the request list. After viewing the list, the League Administrator inputs the player ID of the players he/she wants to add to the league. The Web Server sends verification requests to the database handler(who deals with the database). The database handler then sends the join requests to players. After each player has accepted a request, the Web Server sends a data update request to the database handler. The database handler then sends the updated info(that the player has been successfully been added to the league) back to the web server to be viewed by the league administrator(and player if they want to). If the league admin fails to input a valid player ID, the web server will notify him/her of the problem and request him/her to input a different player ID until he/she inputs a valid one. If the player has already joined the league, the web server will notify the league administrator of the error that the player is already in the league. If the player rejects the request, the Web Server will then send a notification to the league administrator that the player rejected.

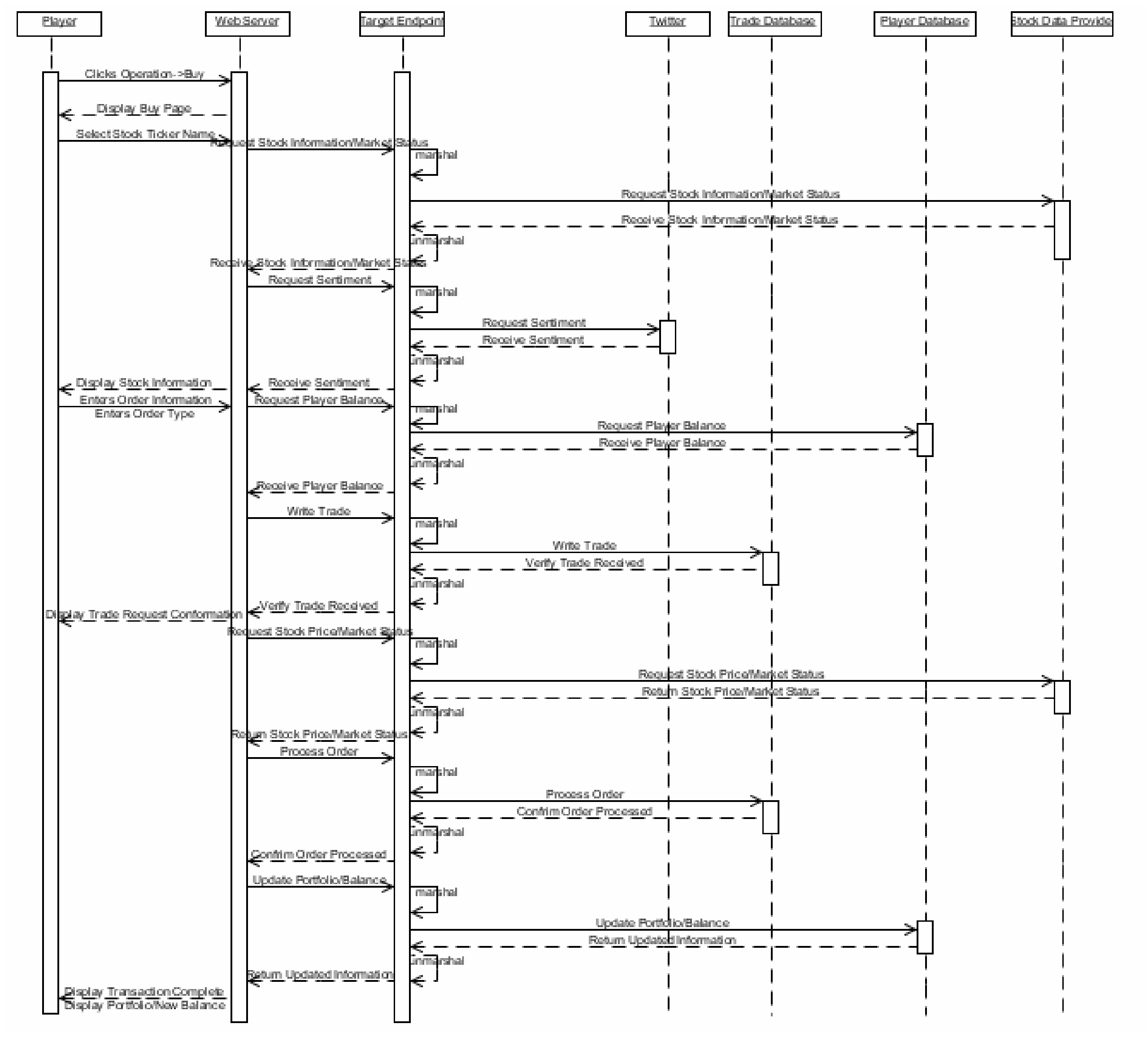

League Administrator
Web Server
Player
Database Handler
LA request to view joining league requests
show LA add requests
LA input player ID
Send an ID verification request
Rerturn verification result
Send a join request
Accept the request
Request update player and league data
Return data update
Notify LA of confirmation

UC-6 Player Requests to Join League:
The league admin will have a unique code/password when they have set the league up. It is assumed that the administrator will email each player in his league this ID/password. The player clicks the sign up button. The web server loads, and sends the sign up page back to the player to fill out. The player enters his information along with the league ID/password given by the administrator. If this league ID/password is entered incorrectly 3 times, a button will appear to email the league administrator to manually add you to the league (which corresponds to the 'Manually Add Player to League' sequence diagram). The web server verifies the player's data. Once verified, the server sends a request for a new player profile to
be created in the database. The database then returns this new profile information to the web server, which then in turn displays this information to the player, showing he's logged in.UC-6 System Sequence Diagram:


Player
Web Server
Player Database
Click Sign Up Button
Load Sign Up Page
Send Submitted Sign Up Form
Enters League ID/Password
Request To Create New Profile
Return New Profile Information
Display Configuration and Profile

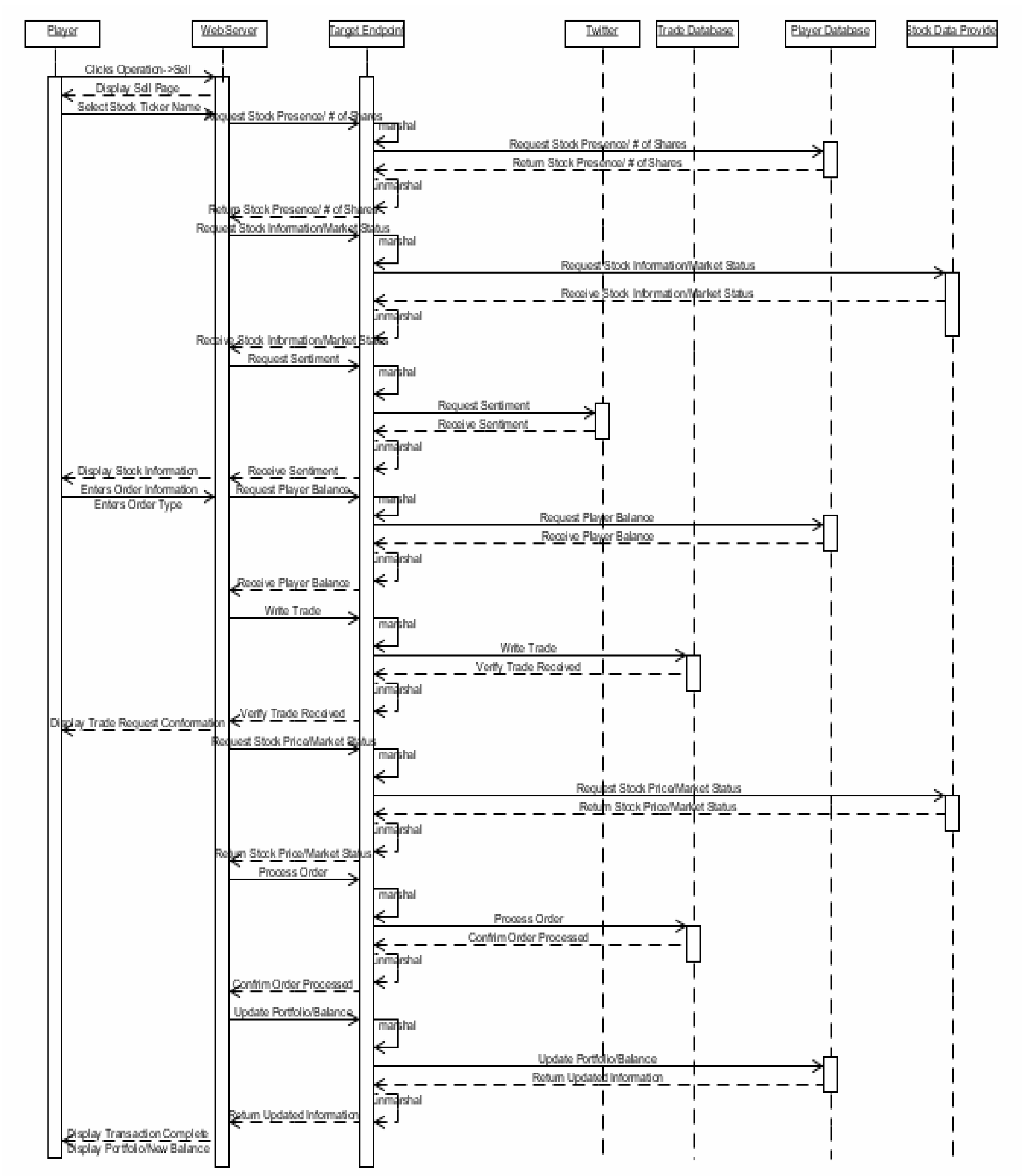
UC-2 Make Trades -- Buy (1 of 2):

The player clicks Portfolio->Buy Stocks. The server displays the buy page(just the stock ticker
name field at first). The player enters the stock ticker name. The server requests the stock information, and market status (whether the market is open or closed) from the a broker module TargetEndpoint, which marshals the request and forwards it to Yahoo! Finance. Upon receiving he data TargetEndpoint unmarshals it and sends the requested information to the server. The server requests sentiment data from twitter. Twitter sends the raw data to the server which analyzes it, using the NLTK library. The server then displays all the stock's information to the player. The player enters his desired buy share amount, boundaries, and order type. The server takes this information and first checks if the player has sufficient available funds by querying the player database. The player database then returns the player balance to the server. If the balance is enough, the server writes the trade request to the trade database.
The trade database verifies the buy request was received and sends the verification to the server, which relays it to display the confirmation of receipt to the player. The server then requests the stock price/market status again from the stock data provider through the TargetEnpoint broker module. This is to check whether the stock prices/market status has changed since the order had been submitted. The newly retrieved market status and stock price are returned to the server and as long as the market status hasn't changed and the stock price hasn't changed more than 0.5%(If the stock price has changed more than 0.5% it will send the user back to the Portfolio->Buy page to query for a stock ticker again) the server will send a order request to the trade database to execute the order. The trade database will send a confirmation of the order being processed to the server. The server will then update the player's portfolio/balance in the player database. The player database then sends the update portfolio/balance back to the server, which then, in turn, displays it to the player on his portfolio page.


Player
Web Server
Target Endpoint
Twitter
Trade Database
Player Database
Stock Data Provide
Clicks Operation->Buy
Display Buy Page
Select Stock Ticker Name
Request Stock Information/Market Status
marshal
Request Stock Information/Market Status
Receive Stock Information/Market Status
Unmarshal
Receive Stock Information/Market Status
Request Sentiment
marshal
Request Sentiment
Receive Sentiment
Unmarshal
Display Stock Information
Receive Sentiment
Enters Order Information
Enters Order Type
Request Player Balance
marshal
Request Player Balance
Receive Player Balance
Unmarshal
Receive Player Balance
Write Trade
marshal
Write Trade
Verify Trade Received
Unmarshal
Verify Trade Received
Display Trade Request Conformation
Request Stock Price/Market Status
marshal
Request Stock Price/Market Status
Return Stock Price/Market Status
Unmarshal
Return Stock Price/Market Status
Process Order
marshal
Process Order
Confrim Order Processed
Unmarshal
Confrim Order Processed
Update Portfolio/Balance
marshal
Update Portfolio/Balance
Return Updated Information
Unmarshal
Return Updated Information
Display Transaction Complete
Display Portfolio/New Balance

UC-2 Make Trades -- Sell (2 of 2):

The player clicks Portfolio->Sell Stocks. The server displays the Sell page(just the stock ticker name field at first). The player enters the stock ticker name. The server requests verification of the stock presence and number of shares from the player database(portfolio). This information is returned to the server. If the stock is present in the player portfolio the server requests the stock information and market status(whether the market is open or closed) from TargetEndpoint. The server then requests the sentiment from Twitter. Twitter sends the raw data to the server, which analyzes it, using NLTK library. The server displays all the stock's information to the player. The player enters his desired sell share amounts, boundaries, and order type. The server then writes the trade to the trade database. The trade database verifies the sell request was received and sends the verification to the server, which relays it to display the confirmation of receipt to the player. The server then requests the stock price/market status again from TargetEndpoint. This is to check whether the stock prices/market status has changed since the order had been submitted. The newly retrieved market status and stock price is returned to the server and as long as the market status hasn't changed and the stock price hasn't changed more than 0.5%(If the stock price has changed more than 0.5% it will send the user back to the Portfolio->Sell page to query for a stock ticker again) the server will send a sell request to the trade database to execute the order. The trade database will send a confirmation of the sale being processed to the server. The server will then update the player's portfolio/balance in the player database. The player database then sends the update portfolio/balance back to the server, which then, in turn, displays it to the player on his portfolio page.


Player
WebServer
Target Endpoint
Twitter
Trade Database
Player Database
Stock Data Provider
Clicks Operation->Sell
Display Sell Page
Select Stock Ticker Name
Request Stock Presence/ # of Shares
marshal
Request Stock Presence/ # of Shares
Return Stock Presence/ # of Shares
unmarshal
Return Stock Presence/ # of Shares
Request Stock Information/Market Status
marshal
Request Stock Information/Market Status
Receive Stock Information/Market Status
unmarshal
Receive Stock Information/Market Status
Request Sentiment
marshal
Request Sentiment
Receive Sentiment
unmarshal
Display Stock Information
Receive Sentiment
Enters Order Information
Enters Order Type
Request Player Balance
marshal
Request Player Balance
Receive Player Balance
unmarshal
Receive Player Balance
Write Trade
marshal
Write Trade
Verify Trade Received
unmarshal
Verify Trade Received
Display Trade Request Conformation
Request Stock Price/Market Status
marshal
Request Stock Price/Market Status
Return Stock Price/Market Status
unmarshal
Return Stock Price/Market Status
Process Order
marshal
Process Order
Confirm Order Processed
unmarshal
Confirm Order Processed
Update Portfolio/Balance
marshal
Update Portfolio/Balance
Return Updated Information
unmarshal
Return Updated Information
Display Transaction Complete
Display Portfolio/New Balance

Design Principles and Patterns:

In order best formulate our design we used a combination of deductive and inductive analyses. At the top down global level we wanted to ensure that the overarching architecture of our system was efficient. While our system isn't particularly large, merely using a bottom up approach would not lead to an optimal design. We let the abilities of our team and our time constraint guide our design. For instance we knew from the start that we would be employing the "communication through a common data element" communication pattern. This is because we could use a master database through which all our sub-systems can interact. We determined this to be more appropriate, at least partially, because we have multiple team members that have worked with database design in the past.

This top-down approach, in isolation, is not sufficient for a well designed system. This is why we employed inductive analysis for the design as well. This allowed us to analyze the problem, and come up with a solution that employees good design principles. Our design focuses primarily on incorporating high cohesion with an inclination towards low coupling. We incorporate various classes and objects to perform specialized roles and functions. Then these objects all communicate with the main controller in either returning values, requesting computations from other objects, etc.

For example, we plan to have an object responsible for data visualization. The main controller would receive the user data from the data request object, then this data would be fed to the data visualization object in order to generate user friendly data images / graphs. We will also have objects responsible for conducting trades and storing the trade results in the database. So in the case where a user has just conducted a trade and wishes to visualize his/her new portfolio data.
The flow would be like this:

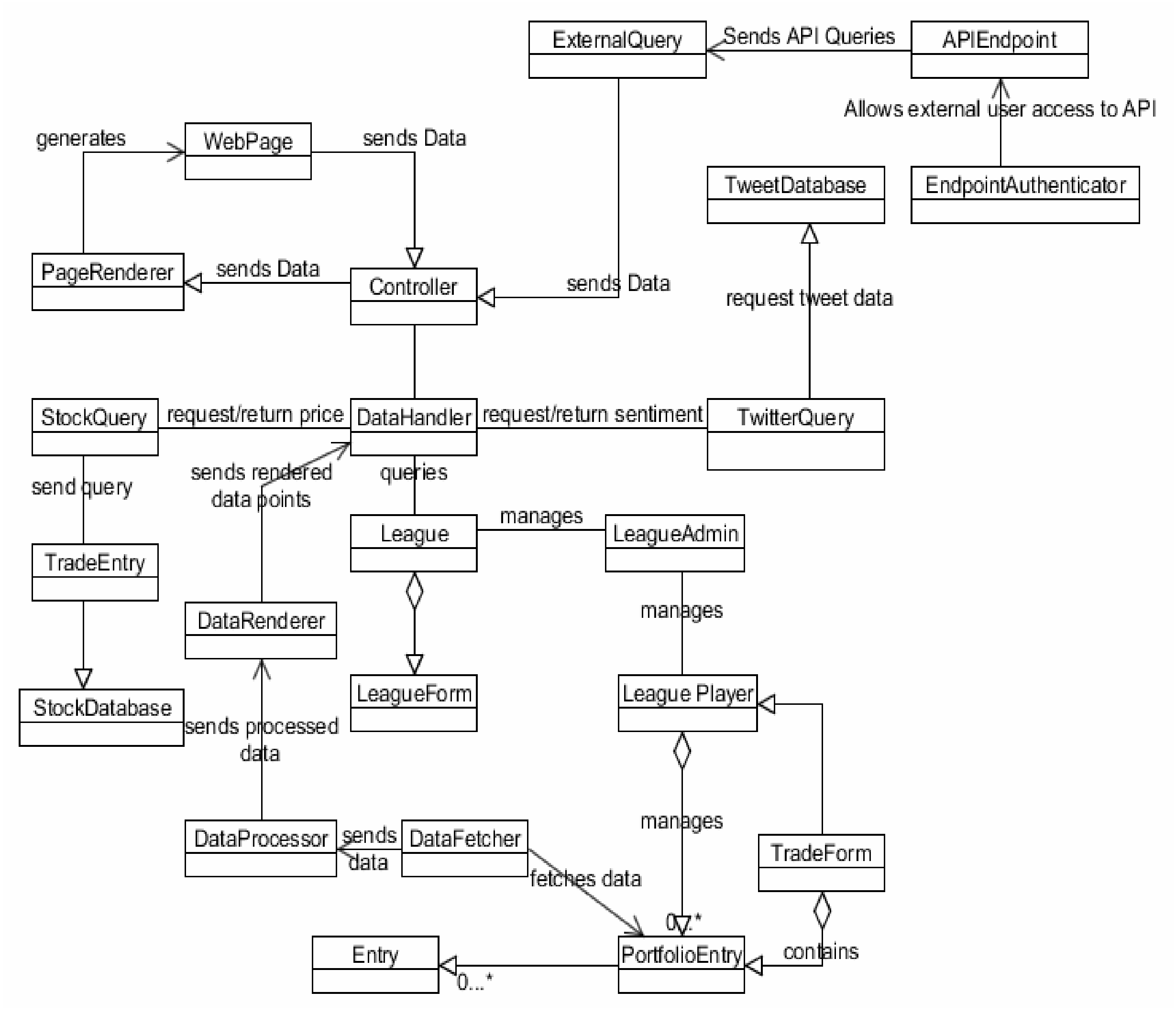
1. Object 1 performs trade and returns trade results to controller.
2. Controller calls object 2 to store trade results.
3. (User requests data visualization)
4. Object 3 fetches portfolio data, and feeds it to object
5. Object 4 then finishes the task and provides the user with the visualizations.

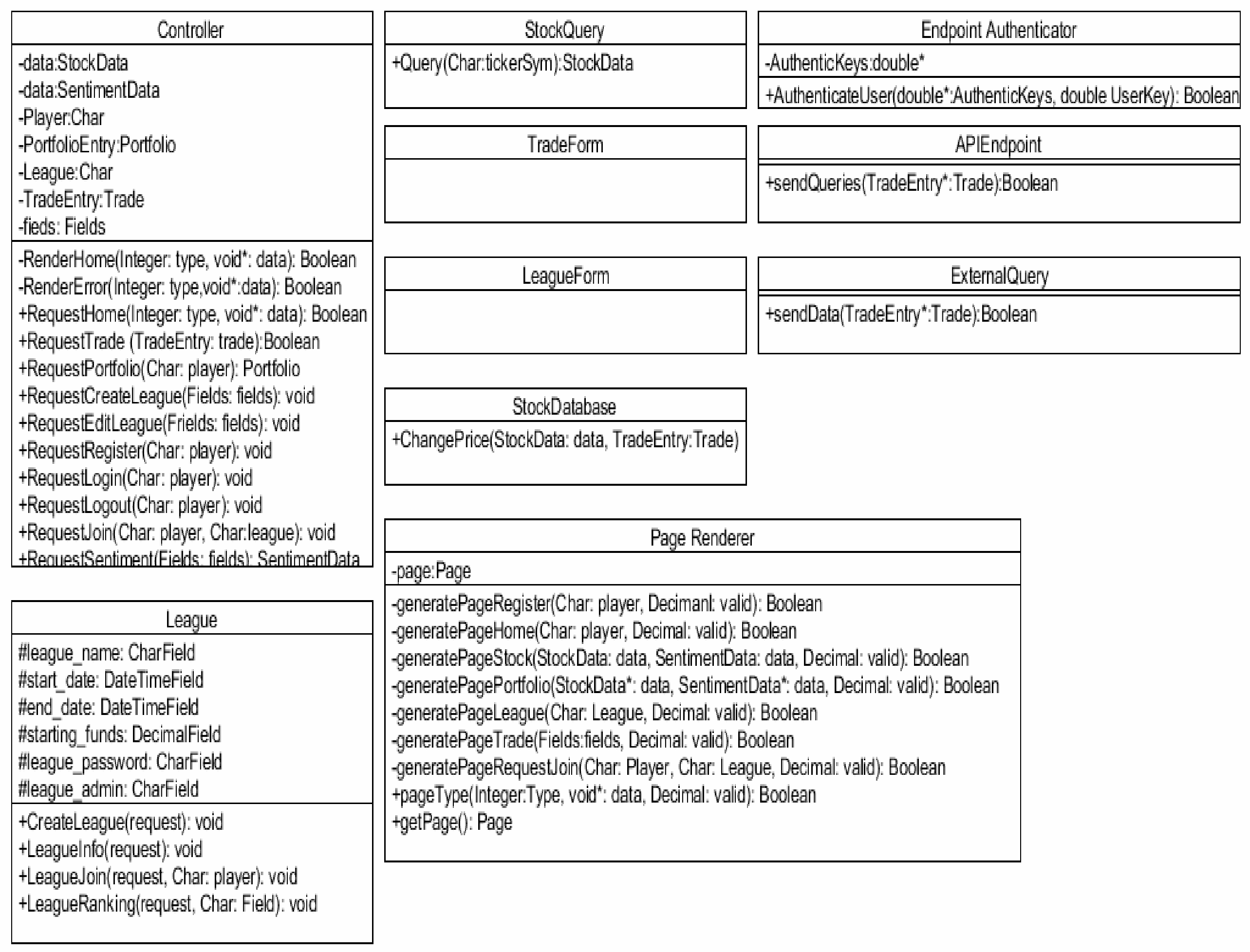
As for coupling, the goal is to have only the main controller communicating between all objects. The individual objects may need to communicate with each other, but it will be through the main controller. By adhering to these concepts, our product code will be easily read and maintained. Addition of new objects will also be streamlined due to the low coupling concept of having a main controller negotiate on the behalf of individual objects.

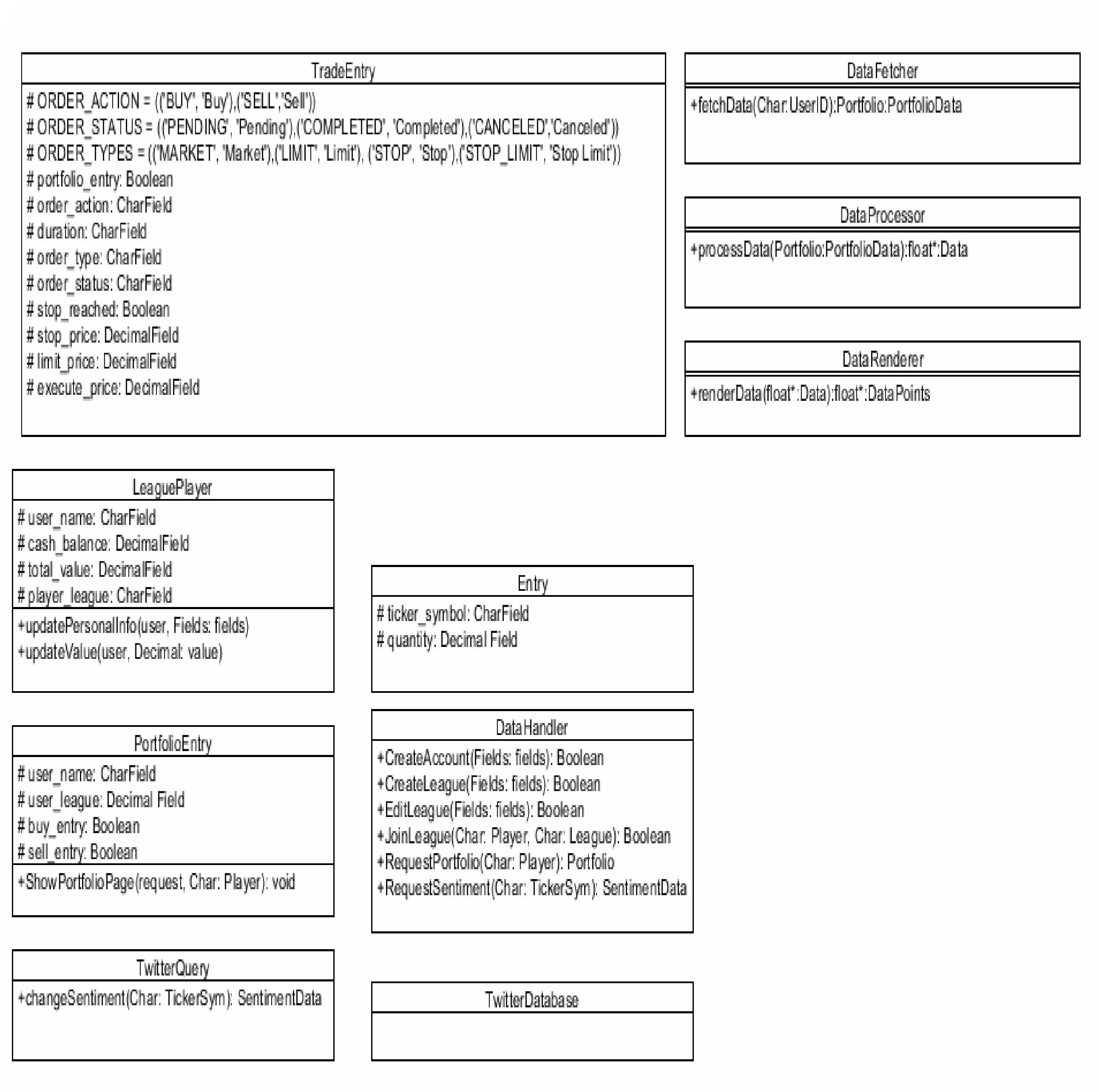
After learning about design patterns in lecture we began to look at how we could apply those to our project to improve it's over all design. The pattern that seemed most applicable to our software system was the broker pattern outline in section 5.4.1 of the text. In our new design rather than have the individual web pages that require stock data communicate directly with the source of this data the request is instead routed through a broker module. This module, which we know as TargetEndpoint provides multiple benefits to the system namely it provides greater flexibility, maintainability, and adaptability to the overall system. Additionally it affords us greater level of security. By moving the data gathering operations to the back end they become less accessible to a potential attacker. In the old design it would be significantly easier for someone to spoof a response from Yahoo! Fiance since all the buy and sell operations were taking place on the web page itself.

<u>Class Diagrams and Interface Specification</u>

Main Diagram




Controller
-data:StockData
-data:SentimentData
-Player:Char
-PortfolioEntry:Portfolio
-League:Char
-TradeEntry:Trade
-fieds: Fields
-RenderHome(Integer: type, void*: data): Boolean
-RenderError(Integer: type,void*:data): Boolean
+RequestHome(Integer: type, void*: data): Boolean
+RequestTrade (TradeEntry: trade):Boolean
+RequestPortfolio(Char: player): Portfolio
+RequestCreateLeague(Fields: fields): void
+RequestEditLeague(Frields: fields): void
+RequestRegister(Char: player): void
+RequestLogin(Char: player): void
+RequestLogout(Char: player): void
+RequestJoin(Char: player, Char:league): void
+RequestSentiment(Fields: fields): SentimentData
StockQuery
+Query(Char:tickerSym):StockData
Endpoint Authenticator
-AuthenticKeys:double*
+AuthenticateUser(double*:AuthenticKeys, double UserKey): Boolean
TradeForm
APIEndpoint
+sendQueries(TradeEntry*:Trade):Boolean
LeagueForm
ExternalQuery
+sendData(TradeEntry*:Trade):Boolean
StockDatabase
+ChangePrice(StockData: data, TradeEntry:Trade)
Page Renderer
-page:Page
-generatePageRegister(Char: player, Decimanl: valid): Boolean
-generatePageHome(Char: player, Decimal: valid): Boolean
-generatePageStock(StockData: data, SentimentData: data, Decimal: valid): Boolean
-generatePagePortfolio(StockData*: data, SentimentData*: data, Decimal: valid): Boolean
-generatePageLeague(Char: League, Decimal: valid): Boolean
-generatePageTrade(Fields:fields, Decimal: valid): Boolean
-generatePageRequestJoin(Char: Player, Char: League, Decimal: valid): Boolean
+pageType(Integer:Type, void*: data, Decimal: valid): Boolean
+getPage(): Page
League
#league_name: CharField
#start_date: DateTimeField
#end_date: DateTimeField
#starting_funds: DecimalField
#league_password: CharField
#league_admin: CharField
+CreateLeague(request): void
+LeagueInfo(request): void
+LeagueJoin(request, Char: player): void
+LeagueRanking(request, Char: Field): void


TradeEntry
# ORDER_ACTION = (('BUY', 'Buy'),('SELL','Sell'))
# ORDER_STATUS = (('PENDING', 'Pending'),('COMPLETED', 'Completed'),('CANCELED','Canceled'))
# ORDER_TYPES = (('MARKET', 'Market'),('LIMIT', 'Limit'), ('STOP', 'Stop'),('STOP_LIMIT', 'Stop Limit'))
# portfolio_entry: Boolean
# order_action: CharField
# duration: CharField
# order_type: CharField
# order_status: CharField
# stop_reached: Boolean
# stop_price: DecimalField
# limit_price: DecimalField
# execute_price: DecimalField
DataFetcher
+fetchData(Char:UserID):Portfolio:PortfolioData
DataProcessor
+processData(Portfolio:PortfolioData):float*:Data
DataRenderer
+renderData(float*:Data):float*:DataPoints
LeaguePlayer
# user_name: CharField
# cash_balance: DecimalField
# total_value: DecimalField
# player_league: CharField
+updatePersonalInfo(user, Fields: fields)
+updateValue(user, Decimal: value)
Entry
# ticker_symbol: CharField
# quantity: Decimal Field
PortfolioEntry
# user_name: CharField
# user_league: Decimal Field
# buy_entry: Boolean
# sell_entry: Boolean
+ShowPortfolioPage(request, Char: Player): void
DataHandler
+CreateAccount(Fields: fields): Boolean
+CreateLeague(Fields: fields): Boolean
+EditLeague(Fields: fields): Boolean
+JoinLeague(Char: Player, Char: League): Boolean
+RequestPortfolio(Char: Player): Portfolio
+RequestSentiment(Char: TickerSym): SentimentData
TwitterQuery
+changeSentiment(Char: TickerSym): SentimentData
TwitterDatabase

New domain model:

Add an external query object that interacts through an api endpoint which goes through the controller to perform queries / purchases.

Explanation:
We decided to improve our project by including an API which allows for extensibility and the ability for 3rd parties to use our functionality remotely. The way the API works is that queries are generated from the 3rd party and are collected at our target endpoint. The queries then are then validated and processed by the controller which then performs the relevant action. The benefit that this allows is users no longer are forced to go through our webpage to instruct the controller to perform operations on their account, instead they can now develop their own clients and their own user interface to connect with us. This is a much better design that is both robust and elegant.

class diagram:
add the external query + api endpoint.
Also add another object called endpoint authenticator

explanation:
The endpoint authenticator determines if the user has the credentials to perform the requested operation. This validates the user query to ensure that security is enforced and the user has permission to perform the operation.

also add:

data visualization:
objects:

data fetcher
data processor
data renderer
explanation:
Data from the user's portfolio is retrieved and processed in order to generate a graphical chart that displays the user's performance over time. Fetcher queries the database to obtain the relevant information using the user's id. The processor takes the database entries and formats them for the data renderer. Then the renderer processes the data and formulates data points which are then displayed graphically with a Javascript wrapper function.

OCL:

Data processor

Pre condition: data is fed in as an associative array where the corresponding column entry matches the column name

Post condition:
data is processed and converted into a direct format consisting of an associative array for the data point. The format is data value, time.

Endpoint authenticator

Pre condition:

Query is made at target endpoint and collected

Postcondition:
Query is authenticated and a return value of valid or invalid.

context Controller::RequestPortfolio(Char:player): Portfolio
pre: (player → Player.Account.Portfolio = pageRenderer)
- A player can see only his or her own portfolio

context Controller::RequestEdit(Fields: fields): void
pre: (LeaguePlayer → updatePersonalInfo = true)
- A player can update information

context DataHandler::CreateAccount(Fields: fields): Boolean
post: (hasPortfolio → true AND inLeague)
- The player will have portfolio for the league when register

context DataHandler::CreateLeague(Fields: fields): Boolean
pre: (league → name =field:_Name AND league→ the.player AND update())
- The league coordinator will have portfolio

context DataHandler::joinLeague(char; player, char:league): Portfolio
pre: (portfolioEntry→ username: char→ the.player AND update())
- The league coordinator will have portfolio

context TradeEntry inv:
if (playerBalance>= 0)
player Balance!= negative

```
          else
                    playerBalance = negative
```

- the player have either negative or not negative balance

context TradeEntry::orderAction((“BUY”, ‘Buy’),(“SELL”,’Sell’))
pre: (orderAction→buyOrder, AND orderAction→ sellOrder)
post: (tradeEntry::orderStatus → pending OR completed OR cancelled)

- the balance must be valid for the specific order
- when the request occurs (buy or sell) data must be called

context LeaguePlayer:: username (CharField)
player →updatePersonalInfo

- the player can update information

context LeaguePlayer:: updateValue(user, Fields: fields)
  player → cashBalance

- the player can update value

context LeaguePlayer inv:
  total value →decimalField

context LeaguePlayer inv:
  player league →charField

context PortfolioEntry inv:
  if (portfolioEntry→entry != stuck)
  stuck>=0
-a portfolio entry cannot have negative stuck

context portfolioEntry inv:
buyEntry→boolean()

context portfolioEntry inv:
sellEntry→boolean()

context entry inv:
tickerSymbol → charField

context entry inv:
quantity → decimalField

- entry uses for the number of stock the players buys

```
context TradeEntry inv:
   portfolioEntry →boolean

context TradeEntry inv:
   stopPrice → DecimalField

context TradeEntry inv:
stopPrice →decimalField


context TradeEntry inv:
limitPrice →decimalField

context TradeEntry inv:
executePrice →decimalField

context TradeEntry inv:
orderType →charField

context TradeEntry inv:
orderStatus→charField
```

Traceability Matrix

Class/Domain Concept	WebPage	PageRenderer	InputHandler	StockQuery	Stock DataProvidr	PlayerHandler	leagueHandler	ValidityChecker	DataHandler	DataBase
Webpage	X									
PageRender		X								
Controller	X	X								
DataHandler									X	
TwitterQuery									X	
TweetDataBase			X						X	
StockQuery									X	
Trade Entry				X					X	
StockDatabase									X	
League						X			X	
League Form									X	
League Admin						X			X	
League Player									X	
TradeForm									X	X
PortfolioEntry					X		X		X	
Entry									X	

Many of the classes are matching with DataHandler concept. It is because classes are interrogated by the DataHandler. In the domain model it is easy to see how each class is in the single entity. However, in the class diagram everything is more detailed because the class diagram gives more information. It is more understandable why classes are evolved from a single concept.

Architectural Styles

RU Investing uses architectural style with focus on Model/View/Controller approach.

3.1.1 Model/View/Controller

Model contains user information and communicates between several systems over a network. A client may initiate a communication session, while the server waits for requests from any client. The site database will be created using Django and the stock model will be made accessible by API calls to an external stock information provider.
View requests from the model the information it needs to generate an output. The view will be represented by HTML, CSS, and JavaScript. Controller is the controller logic will be implemented using Python.

3.1.2 Front and Back Ends:

Both the frontend and the backend are using the component-based architectural style by using design and development languages that allows them to be run independent of the platform they are on. And, The front-end component of our system is our Web UI. This is what the public will see. The back end consists of all the behind the scenes business logic for our app.

3.1.3 Event-driven Architecture

A software architecture pattern promoting the production, detection, consumption of, and reaction to events. Any change to the equilibrium of our system by the user is an event. In this way, the user acts as an event emitter (i.e. initiating buy, sells, creating leagues,etc.). The events are handled by the controller logic, which serves as the event consumer for these events. Another type of event that drives our application are changes in stock price. This is used to execute limit, stop, and stop limited orders.

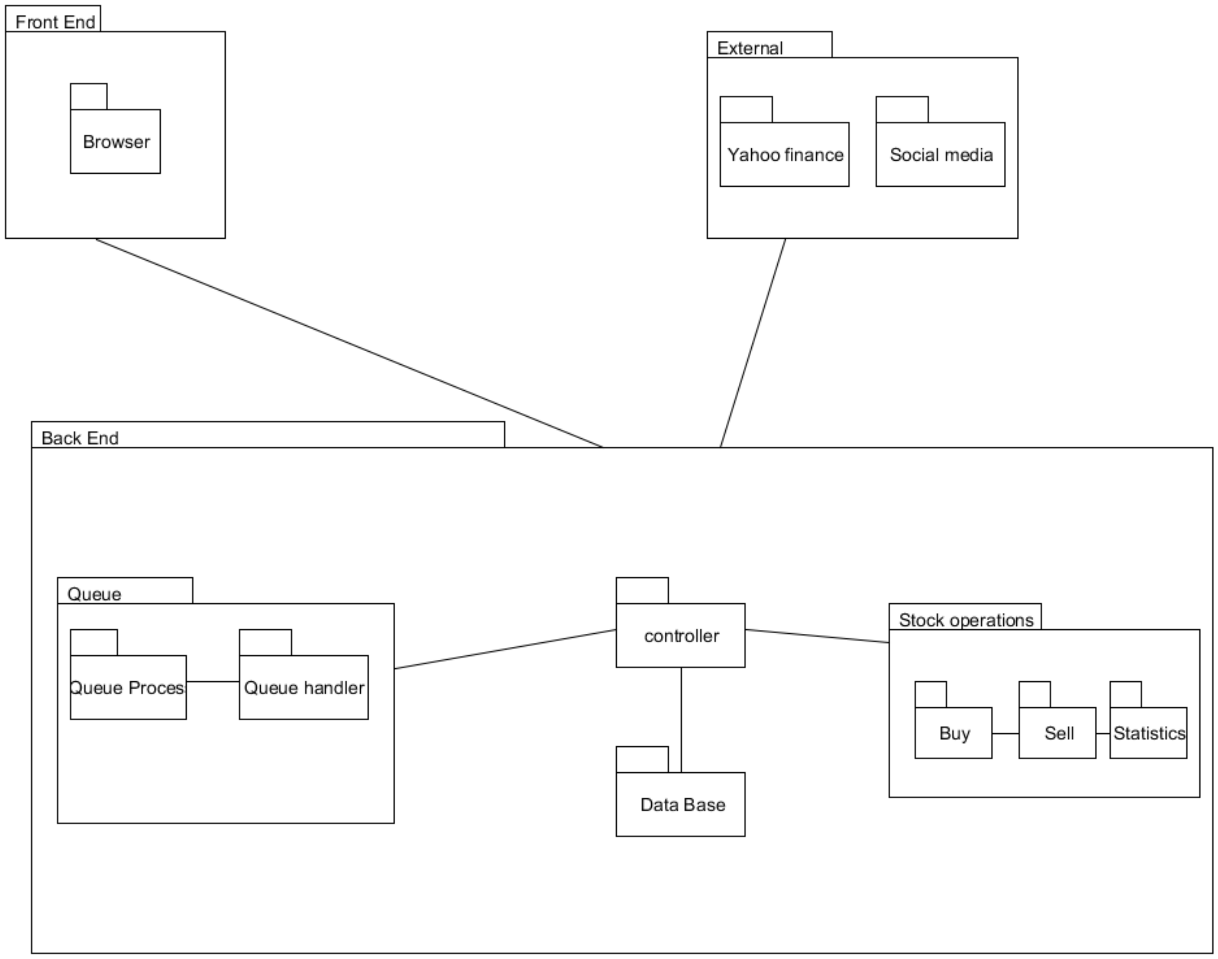
3.1.4 Object-oriented

The responsibilities are divided into different objects, which contain relevant information/data and behavior. In our application, we are planning to use object oriented approach, because it will make our work easier as well as efficient. We can represent Portfolio, Securities, League, and Orders as objects.

Identifying Subsystems

RU investing works as an educational game for finance classes at Rutgers University. The website contains three main subsystems: the visible side to the user which will be the front end of the website, the back end of the system where the technical operations take place and the external end which is used as data source.

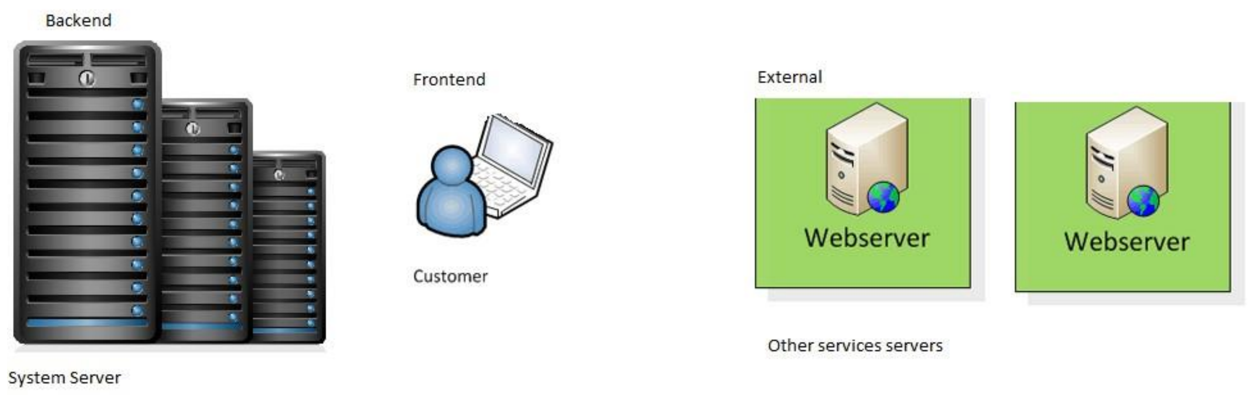
We can see that most of the essential operation occurs in the back end of the system. Essential part of the back end system is the controller which links requests from the queue as buying, selling and show statistics from the user and then assigning those requests to the request handler. The queue
will store all requests from the user then transfer them the controller which will assign those operations to the appropriate subsystems. Other
subsystems will have the ability to communicate with external databases like yahoo finance and social media networks.



The UML package diagram

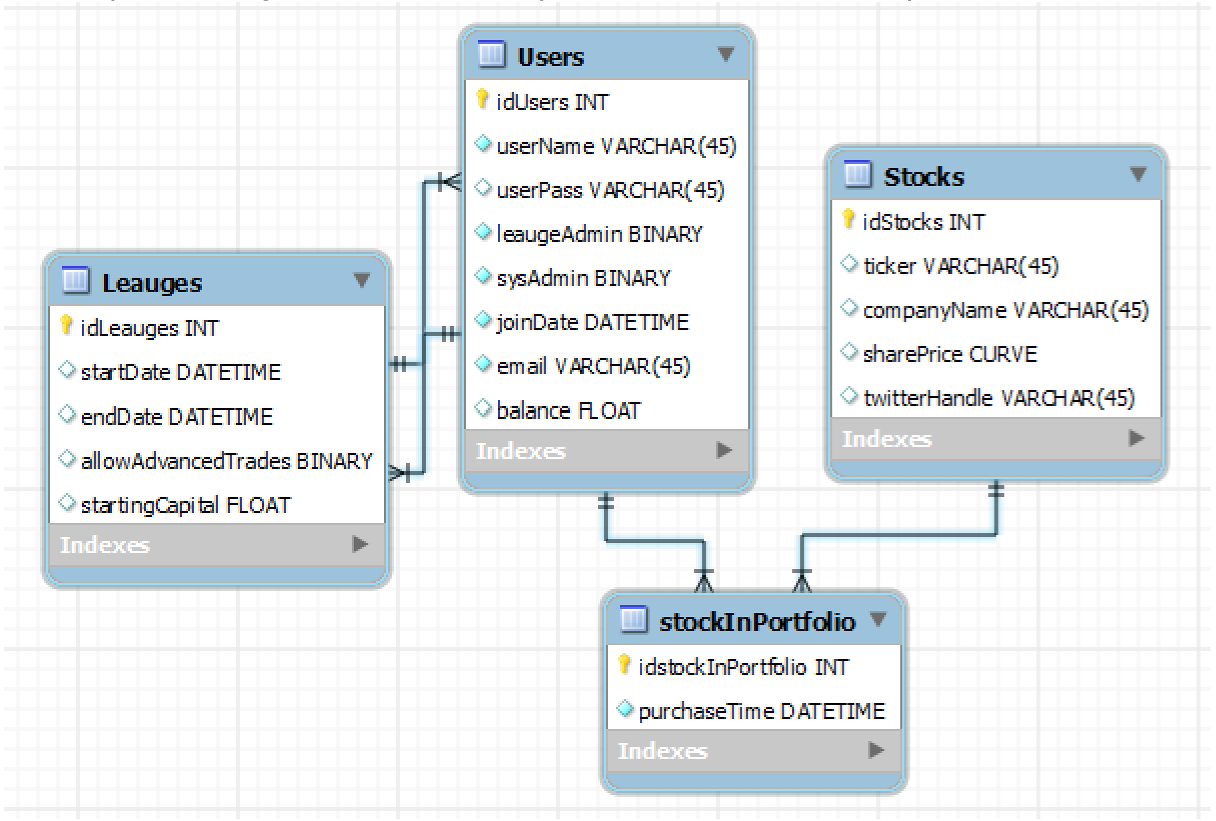
Mapping to Hardware

When running software that is based on a web page there is a standard used on how the hardware is mapped. The front end system will run on the user machine and the back end will run on the server.



Persistent Data Storage

Our project requires quite a bit of persistent storage. We will need to store records of every user and their portfolio, every valid stock, and every league. In order to make sure our system is robust, secure, and easily managed we will be using a time tested industry standard storage scheme. Specifically we will be using MySQL in order to implement a relational database. MySQL which is currently the most popular open-source database software in the world*. After careful analysis of our domain model, we have developed the following model for our planned database. Most of the model should be self-explanatory, to clarify one of the trickier aspects however, the stockInPortfolio table is used to implement a many to many function between stocks and users, while also allowing us to store the purchase time and date, to determine gains and losses over time. The reason we do not keep track of a current net worth, or rank is because all of this is highly variable and can
be computed using information already stored with minimal computation.



*source: http://www.mysql.com/

Network Protocols

We will be using two different network protocols to implement our market simulation. We will be using both HTTP and Websockets. We will have a web server that will serve HTML and PHP content to the user through the HTTP protocol. Most basic operations will be through HTTP such as displaying user information and profile data.

However we will implement our trading operations by using the Websocket Protocol. The Websocket Protocol allows for lower overhead as well as lower latency response. Both of which will make our system more efficient and robust in large scale operations.

Global Control Flow

Our system will be both procedure and event driven, depending on which modules are being in use. For example, registering an account will be a procedure driven process. On the other hand, performing real time transactions will be an event driven process where the communication channels are initiated upon the event generated by the user.

Our system will be a time dependent system. For one thing, the stock market opens and closes on a timer. Also a major part of our system will be the event driven real time transactions.

On the topic of threading, our system will be a multi-threaded system. We will need multi-threading to achieve optimal performance during times where there are multiple concurrent connections.

Hardware Requirements

We will be using an online hosting provider that will provide us with a PHP backend and a MYSQL database. The server must include the ZMQ library as we will be using it for passing messages between the client and the server.

The host server also must have a minimum of 1 TB storage space with options to expand upon new users. The network connection must also be extremely stable and at least 50Mbps in order to meet the demand for real time market transactions.

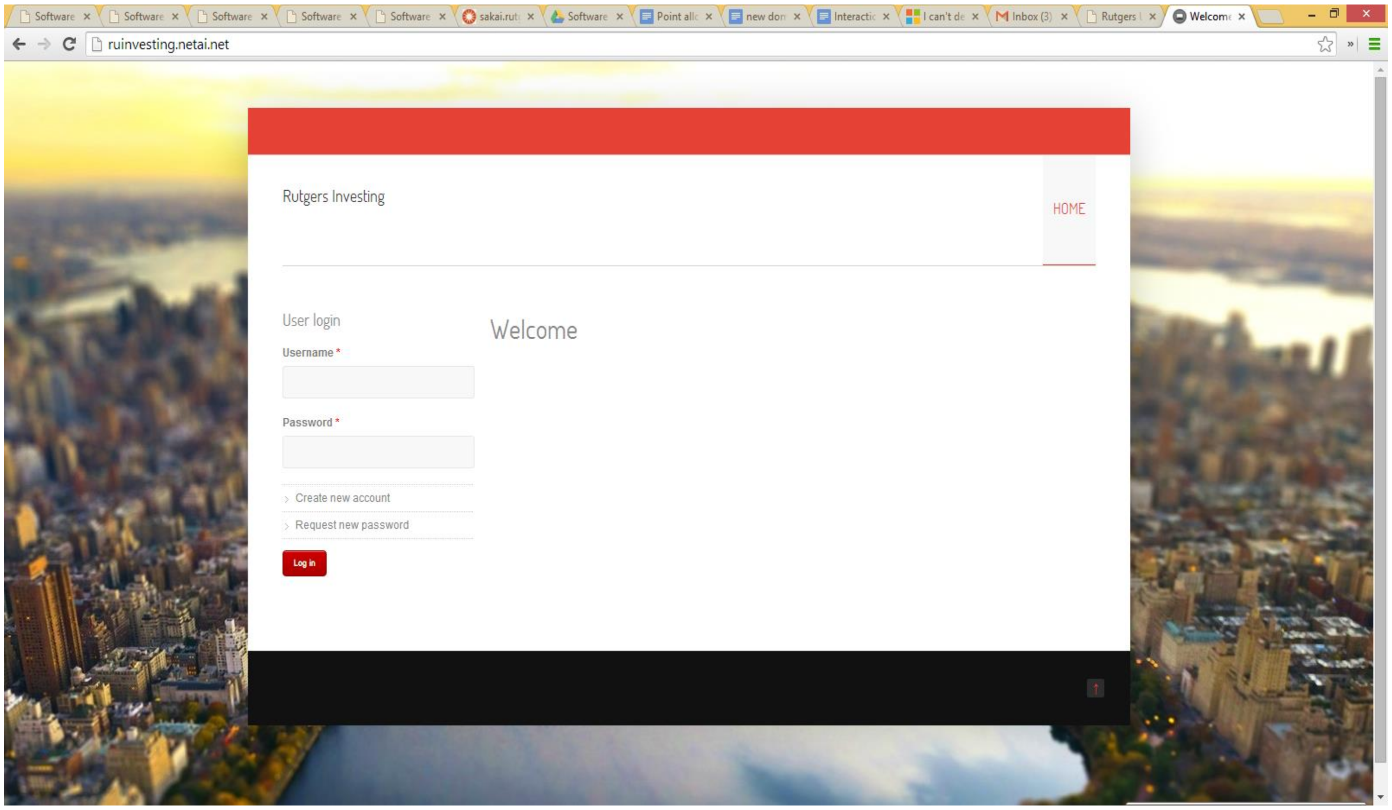
Algorithms and Data Structures

One of the key benefits of the architecture of our design is that it avoids using any sort of complex data structures. By using a database as a central communication hub we can simply use SQL request to pull arbitrary data, based on all sorts of variables. This avoids the need for trees and linked list and other complex data structures. Our php code will merely have to work in primitives, mainly the associative array. Like wise our algorithmic requirements are fairly simple. Most of our requirements such as keeping track of net worth and making trade involve simple addition and subtraction across entries returned from a SQL query.

User Interface Design

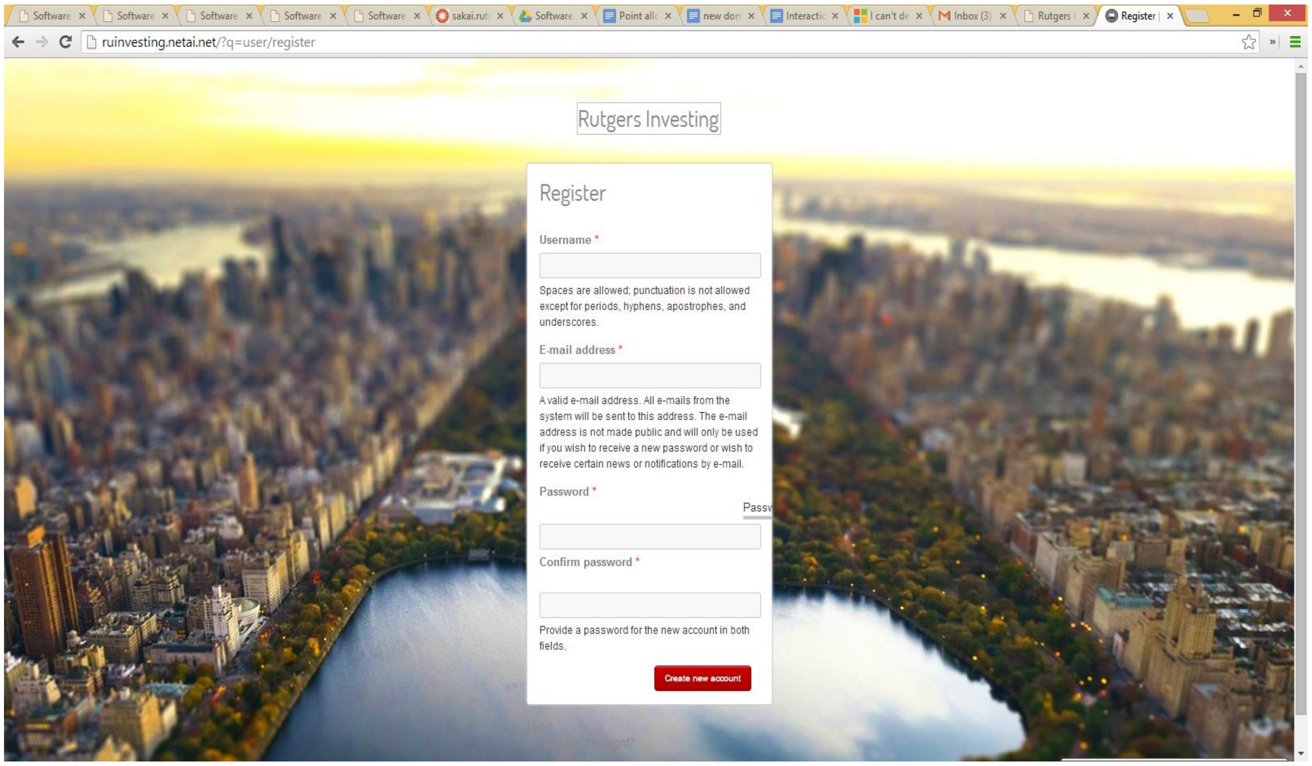
Home/Login Page

When the users first visits the website, this is the homepage that they will be greeted with. The home page allows people to log in and create a new account. The home page also will allow users to request their password if they forget it.



Create New Account:

The sign up page can be accessed from the home page by clicking “create a new account” link. The users will then be instructed to create a new account. The requirements for the creation are username, email address and password.



Register

Username *

Spaces are allowed; punctuation is not allowed except for periods, hyphens, apostrophes, and underscores.

E-mail address *

A valid e-mail address. All e-mails from the system will be sent to this address. The e-mail address is not made public and will only be used if you wish to receive a new password or wish to receive certain news or notifications by e-mail.

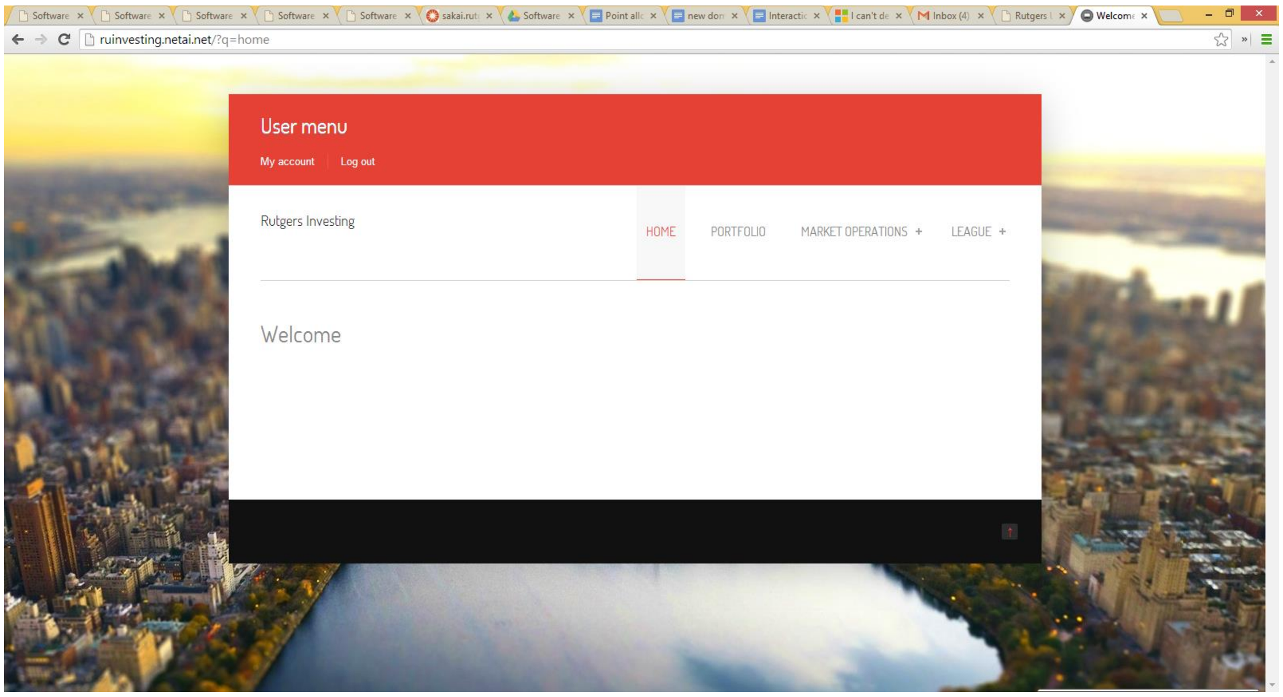
Password *

Confirm password *

Provide a password for the new account in both fields.

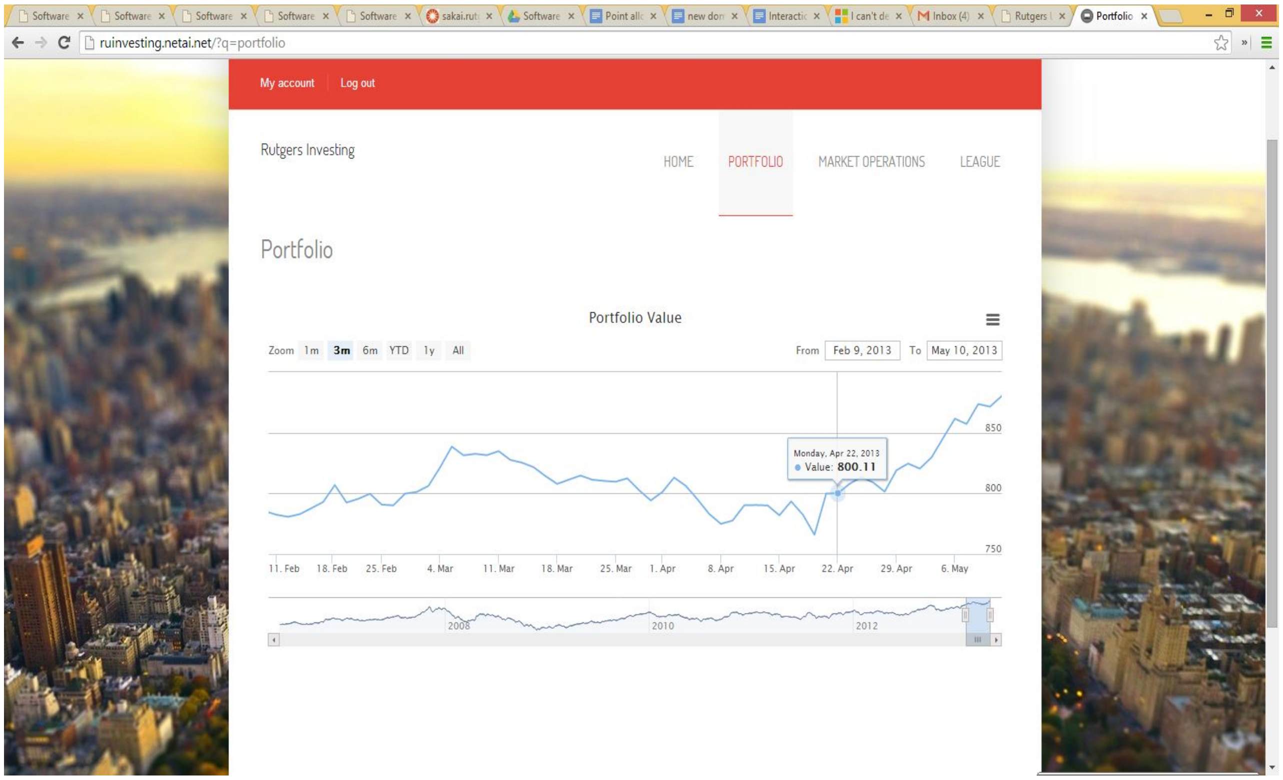
Home

When users click on the home page, he or she will get information about the home page..



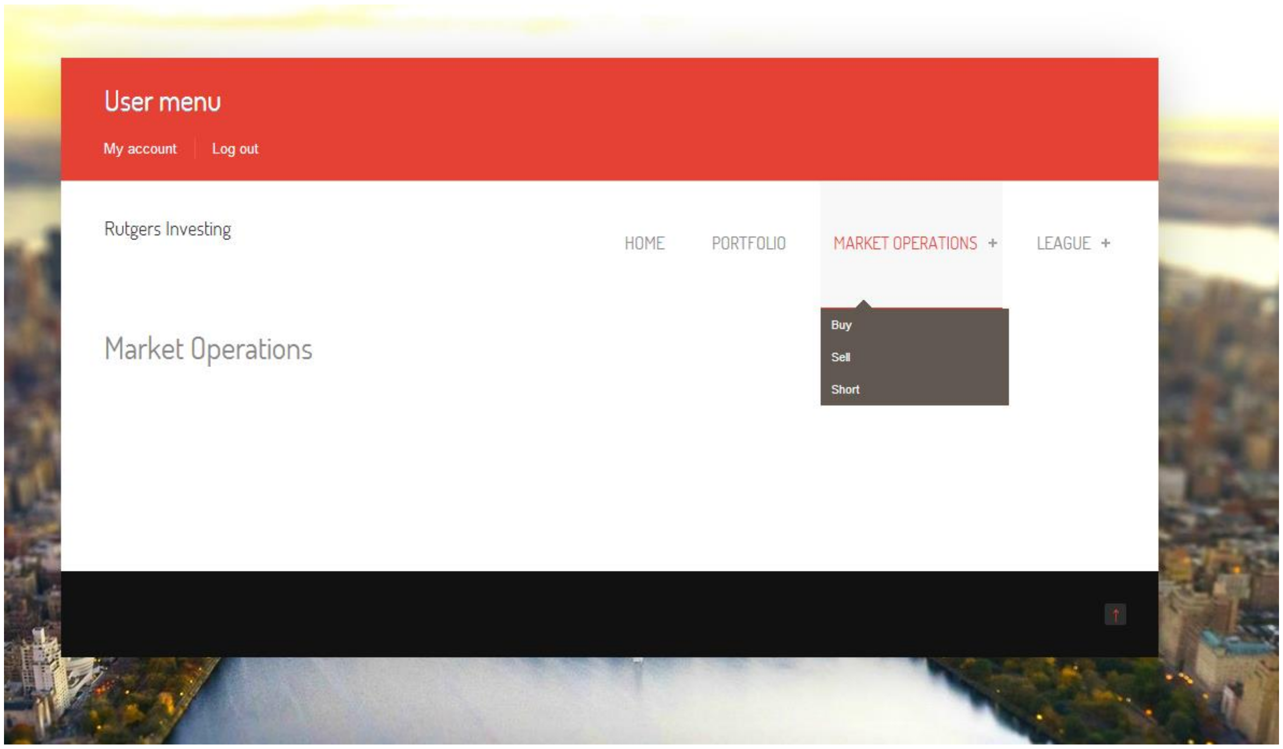
Portfolio Page

Portfolio page has general information about the user. User can see his or her balance, the day that the user joined the website, a performance graph that shows the user’s performance.



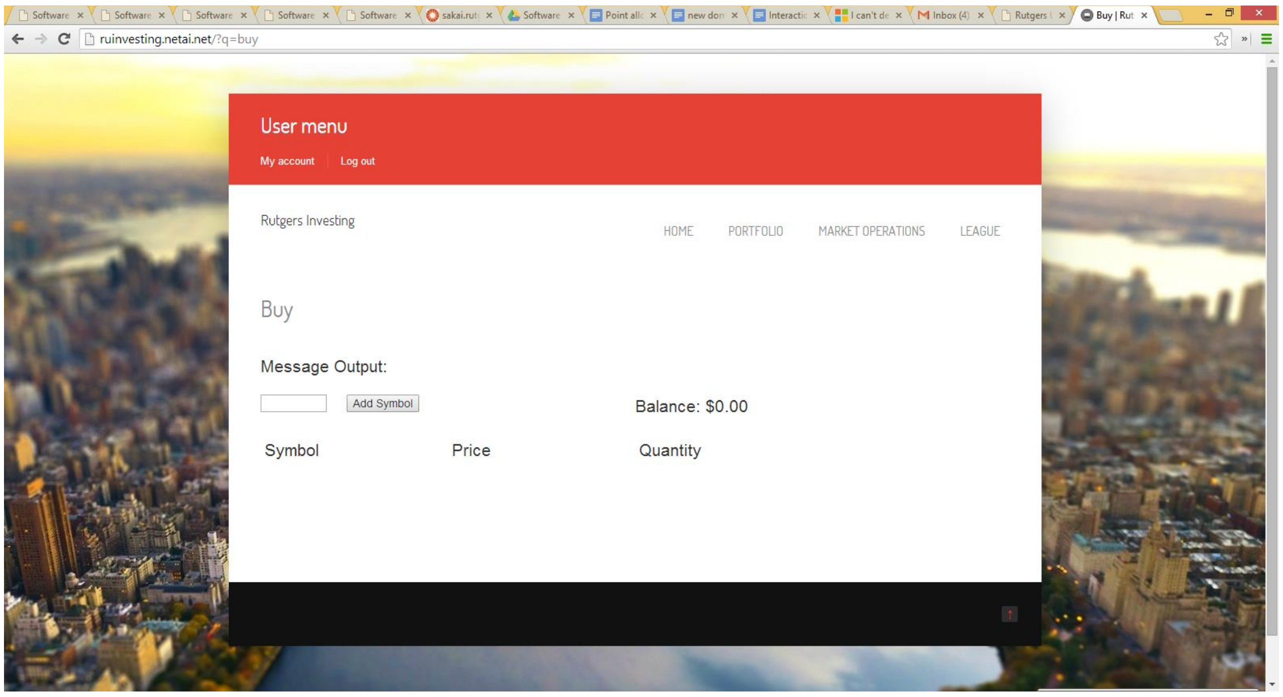
Market Operations:

When user clicking this button a sliding bar with useful trading operations appear.



Buy

The user can access this page by clicking on the buy button from the sliding bar. This page main purpose is for the user to search and buy stocks. After selecting the stocks and type in the trade amount, the users can either make an immediate transaction by clicking “Trade for a total of ….” or place a limit of order for a better profit by fill in the form and selecting the corresponding button.



Buy

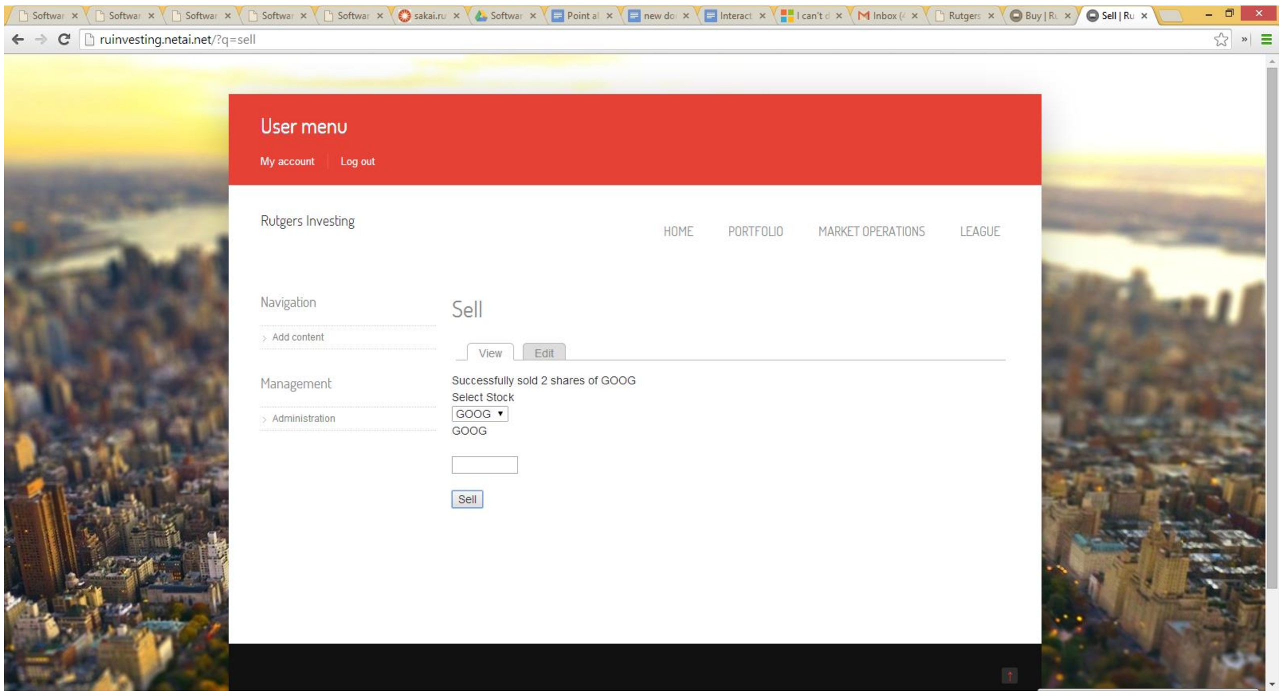
Message Output:

Balance: $0.00

Symbol Price Quantity

Sell

When the users select the sell button, they will be able to see their owned stocks. Similar to functions in “Buy”, you can make an immediate transaction by clicking “Sell for …” or place a limit/stop order buy fill in the form and selecting the corresponding button.



Successfully sold 2 shares of GOOG

Select Stock

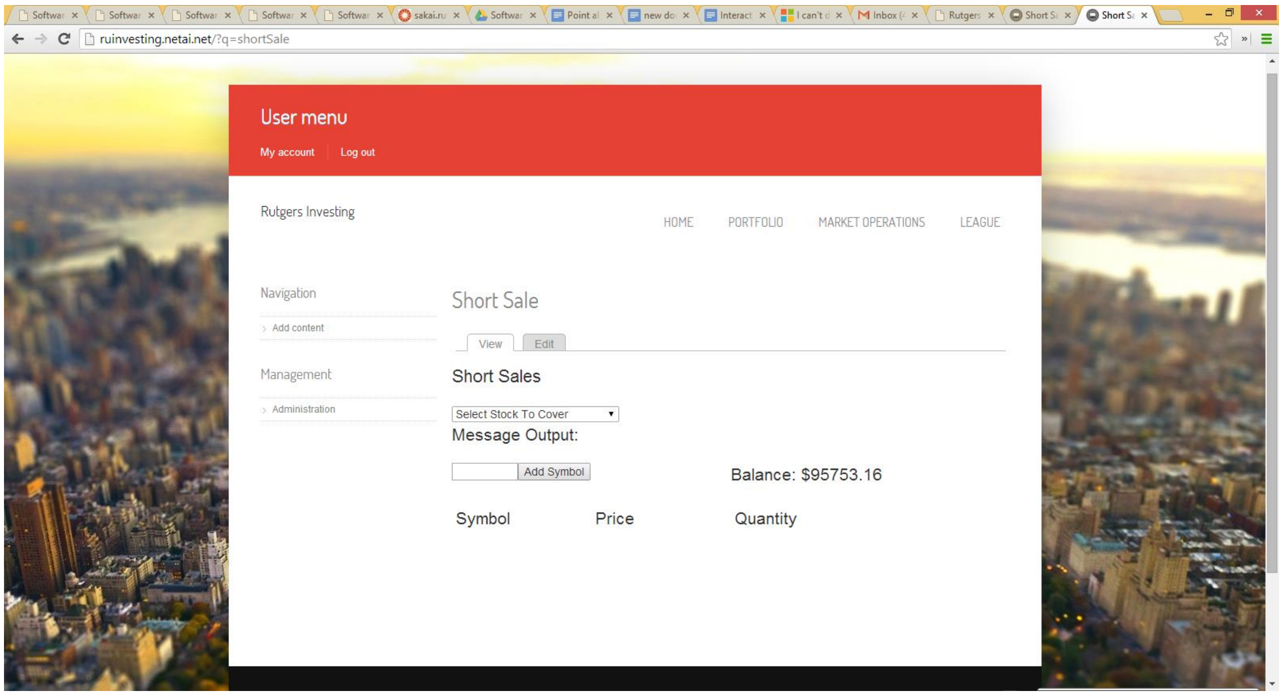
GOOG

GOOG

Sell

Short Sell:

By clicking this button the user can use short sale function.



Short Sale

Short Sales

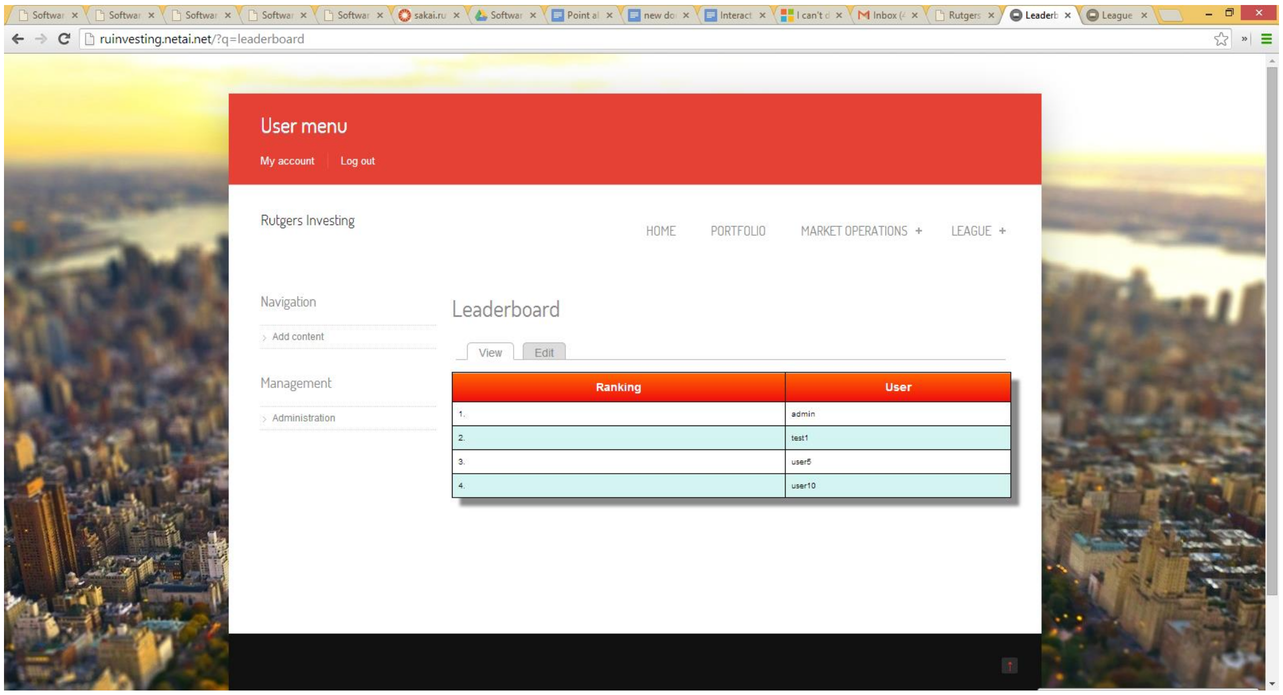
Message Output:

Balance: $95753.16

Symbol Price Quantity

League Leader Board

This page will display the ranking of the users.



ruinvesting.netai.net/?q=leaderboard

User menu

My account | Log out

Rutgers Investing

HOME PORTFOLIO MARKET OPERATIONS + LEAGUE +

Navigation

Add content

Management

Administration

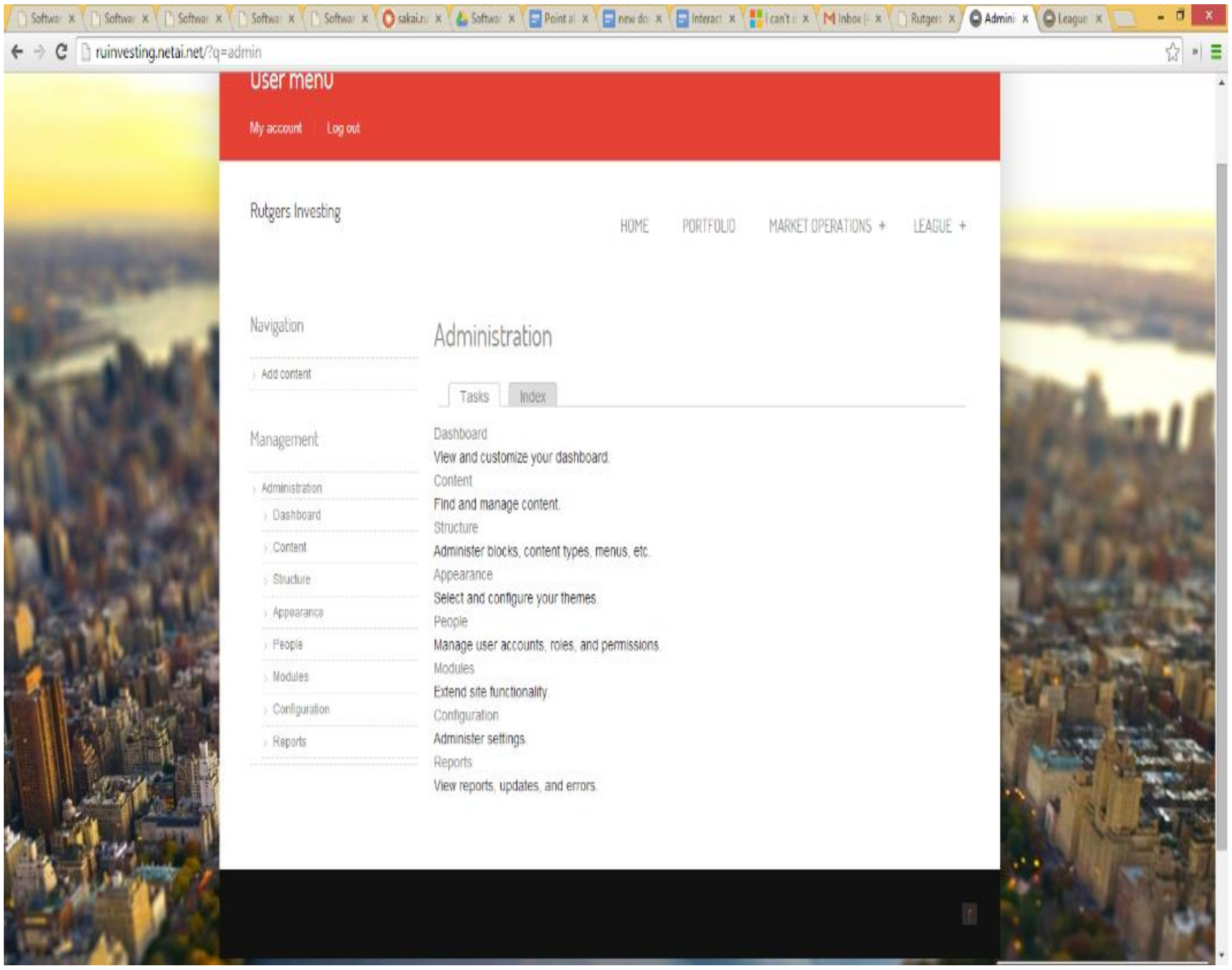
Leaderboard

View Edit

Ranking	User
1.	admin
2.	test1
3.	user5
4.	user10

Administrator Page:

Contain operations needed for running a league.



Administration

Dashboard
View and customize your dashboard.
Content
Find and manage content.
Structure
Administer blocks, content types, menus, etc.
Appearance
Select and configure your themes.
People
Manage user accounts, roles, and permissions.
Modules
Extend site functionality.
Configuration
Administer settings.
Reports
View reports, updates, and errors.

Test Cases Design

Test Cases

Test –case : TC-1 Function tested: Mouseover Pass/Fail criteria: The test passes when the user moves the cursor to a specified region and the region is highlighted and or dialog box pop up. The test fails if criteria above doesn't happen.	
Test Procedure	Expected Results
Test: • The cursor will be moved to a certain location. • Keep cursor on the same location for a certain period of time. Fail: • The cursor will be moved to a certain location.	Pass: • Word is highlighted. • Dialog box pops up and the relevant information displayed. Fail: • Word is not highlighted, script error.

Test Case -1

Test –case : TC-2
Function tested: Send request.
Pass/Fail criteria: The test passes when the user send http request and capture the response.

Test Procedure

Pass:

- The user clicks a hyper link button and waiting for response.
- When there is event caused by a user requesting an http page(refresh).

Fail:

- User hits the button or the hyper link again.

Test Case 2

Test –case : TC-3 Function tested: Check market time. Pass/Fail criteria: The server will be able to check the market time.
Test Procedure
Pass: • The system will ask from the function to deliver open/close market status. Fail: • The system will ask from the function to deliver open/close market status.

Test Case-3

Test –case : TC-4 Function tested: Form send. Pass/Fail criteria: The form will be sent and the received data will be checked for invalid entries.
Test Procedure
Pass: • The system send and receiving a form and accept it information Fail: • System cannot send form. • System cannot accept form.

Test Case-4

Test-case: TC-5 Function Tested: InvalidSymbol Pass/Fail criteria: The test passes if the user enters an invalid ticker symbol and the web page displays “Invalid Ticker Symbol”. The test fails if the user enters an invalid ticker symbol and the web page doesn’t display	
Test Procedure	Expected Results
Pass: • The user enters an invalid ticker symbol in the text box in the buy page of the website Fail: • The user enters an invalid ticker symbol in the text box in the buy page of the website	Pass: • The page displays the stock ticker symbol entered followed by “Invalid Ticker Symbol” message. Fail: • The page displays nothing, or it displays the stock ticker symbol entered and an invalid price

Test-case: TC-6 Function tested: FetchStockData Pass/Fail criteria: The test passes if the web page accurately displays the ticker symbol entered and the current stock price of that ticker next to it. The test fails if the web page displays the incorrect price or	
Test Procedure	Expected Results
Pass: • User enters a valid stock ticker symbol in the text box in the buy tab of the web page. Fail: • User enters a valid stock ticker symbol in the text box in the buy tab of the web page.	Pass: • Web page displays the stock ticker symbol followed by the current market price next to it. Fail: • Web page displays nothing, "Invalid Ticker Symbol", the incorrect ticker symbol, or the

Integration Testing

RUInvesting will be tested using the bottom up strategy. Each part will be tested individually first, then after integration(in case race conditions occur). Each test must also be run in each possible state and time(to make sure it behaves properly whether the market is open or closed). We implemented this strategy in hopes it will allow us to catch bugs at a lower level and pre-emptively fix the bugs at the top level.

mouseOver:
The mouseOver test will verify the mouseOver function is working properly (when the mouse is hovered over a certain field the field should become highlighted). This is important if there are many small fields close to each other, the user could have difficulty distinguishing between them, so the mouseOver functionality will help them differentiate.

sendRequest:
The sendRequest test will verify that our http request handler is working properly. If the user is unable to navigate throughout our website, they will not be able to access information, make trades, or even register for that matter.

checkMarketTime:
The checkMarketTime test will ensure that our system will be able to know when the market is open and when it is closed. This is critical because regular trades cannot occur when the market is closed.

formSend:
The formSend test will verify that the relevant form is sent to the correct address and verify that sent data is received.This test is critical because communication between user-system, system-server and system-system need to work accurately and fast.

InvalidSymbol:
The InvalidSymbol test will verify that if an invalid symbol is entered in the text field of the market transaction tab the web page will display an “Invalid Ticker Symbol” message. This test is important because we only want to display prices of valid stocks.

FetchStockData:
The FetchStockData test will confirm that when a valid stock ticker symbol is entered in

the text box of the market transaction tab of the web page, the current market price per share of that stock will be displayed. This is very important because it’s at the heart of our fantasy game. If the user is unable to view current stock prices, he/she will not be able to make informed buying decisions.

History of Work

Our group managed to accomplish our goals in a timely, organized, and consistent fashion.

The first goal of our group was to formulate a project idea and determine a system that would allow us to coordinate our efforts. We successfully accomplished this by creating the user interface and functional specifications that were included in report one. These specifications allowed us to have a clear idea as to how our project was to be approached on a high level. We also established a system where parts of the project would be broken up in to modules to which we assigned to pre-determined sub teams. These sub teams would work individually to complete their assigned module, and afterwards integration would be performed to combine modules and ensure compatibility. We also established a communication network through the use of Facebook, Email, and SMS. All of this was done from Feb 17 to Feb 23 .

As we finished the first report and established our vision for the project, we also started to deploy our web server. During the first week of March, we obtained hosting through the use of a free online hosting service and set up a PHP installation with MYSQL support. The next step was to determine version control and member permissions. We decided to opt for a modular local testing system where each team worked on their assigned module locally. Changes were only to be made on the remote server when functionality of the module was guaranteed. This allowed each individual sub team to have control as to how they wanted to approach the development of their module.

After preparing our remote server and determining access permissions, we then proceeded to work on the second report. This would still be during the first week of March, specifically the weekend of March 1st and 2nd. We accomplished many of our milestones during this time. We established a system for obtaining stock data from the Yahoo Finance API through the use of asynchronous XMLHttpRequests by using built in Javascript AJAX functions. Requests were made by using MYSQL queries to the Yahoo Finance API endpoint to retrieve relevant information. We also managed to implement all of the Market Operations as well as produce a working prototype of the leaderboard functionality. In the next week, we then collated all of our modules and integrated them together to produce the working application that was demonstrated during the first demo. We also had implemented our high quality user interface that gave our application a professional look.

In the month of April, we continued to progress in developing key functionality in our application such as the leaderboard and API endpoint. We also made progress toward Report 3 by reviewing our work and determining how our progress had evolved over the semester. Currently where we stand, our project is near the end of completion and there is only minor tweaks that need to be made in order to perfect it.

Current Status:

Our greatest strength was our ability to establish a clear communication network that allowed us to consistently meet deadlines. Our approach at partitioning the project into modules also proved to be a great idea as it allowed for project ownership as well as implementation of low coupling. We have met all of our goals and deadlines, and as a result we have produced a high quality product that we are proud of.

Key Accomplishments:

- Designed a user permission system for league users, league administrators, and system administrators
- Designed a functional and extendable API
- Designed a backend database system that stores entries for league functionality, user permissions, and user data
- Created working market operations that interact with the Yahoo Finance API to perform operations on our backend database structures
- Implemented a data visualization system that allows users to obtain a graphical representation of their performance
- Implemented Clean and Professional User Interface

Future Work:

We have a few ideas that we believe would be beneficial extensions to our core system. One of these ideas is to implement Twitter Sentiment analysis which would be a unique way to incorporate high level data interpretation algorithms into the application. Due to the fact that we believe this is a very promising idea, we have already established the implementation details and included Twitter sentiment in our interaction diagrams in the previous reports. However, because this is such an abstract and complex feature, we felt that we did not want to produce a low quality implementation. A great future goal would be to perform detailed analysis as to how we could use such a feature to improve our system and implement the feature in a well documented fashion.

We would also like to incorporate market analysis tools in the future. These tools are proven to be extremely useful in real world trading activities. Since the goal of our users are to obtain a simulation of real world trading, it would be a good idea to try to fully immerse them and provide them with all the resources that a real trader would have.

A very important feature of our system however was the API, we believe that it can prove to be immensely useful as future work will be crowd-sourced which allows anyone to improve our system and extend our work.

References

http://blog.kaazing.com/2010/02/24/5-signs-you-need-html5-web-sockets-part-2/

http://www.informationweek.com/wall-streets-quest-to-process-data-at-the-speed-of-light/d/d-id/1054287

https://webtide.intalio.com/2011/09/cometd-2-4-0-websocket-benchmarks/

http://spreadnetworks.com/press-releases/10-04-2012-latency-improvements/

http://www.investopedia.com/ - Used for financial term definition

http://www.umlet.com/