Money Machine

Report #3

Group No. 6

Team Members

Name	**Email**
Rylan Uherek	rylan@scarletmail.rutgers.edu
Avinash Oza	avioza@scarletmail.rutgers.edu
Aakash Patel	Aak4shpatel@gmail.com
Mozam Todiwala	tmozam@scarletmail.rutgers.edu
Mandeep Desai	Mandeep.desai111@gmail.com
Pintu Patel	Php28@scarletmail.rutgers.edu

Instructor: Prof. Ivan Marsic

Project URL: https://sites.google.com/site/sespring13/

Revision History:

Version No.	**Date of Revision**
v.1 – Omitted Sections NOT Included	5/5/2013
v.2 – Full Report	5/12/2013

Individual Contributions Breakdown

Task/Group Member	Rylan	Avinash	Aakash	Mozam	Mandeep	Pintu
Summary of Changes (5 Points)	100%					
Sec 1: Customer Statement of Requirements (6 points)	100%					
Sec 2: Glossary of Terms (4 Points)	100%					
Sec 3: System Requirements (6 points)			60%		20%	20%
Sec 4: Functional Requirements Specification (30 Points)		50%		50%		
Sec 5: Effort Estimation using Use Case Points (4 points)					50%	50%
Sec 6: Domain Analysis (25 Points)					50%	50%
Sec 7: Interaction Diagrams (40 Points)		50%		50%		
Sec 8: Class Diagram & Interface Specification (20 Points)					50%	50%
Sec 9: System Architecture (15 Points)			60%		20%	20%
Sec 10: Algorithms & Data Structures (4 Points)[1]	---	---	---	---	---	---
Sec 11: User Interface Design & Implementation (11 Points)			100%			

[1] This section was not pertinent to our project.

Sec 12: Design of Tests (12 Points)			33%		33%	33%
Sec 13: History of Work, Current Status, & Future Work (5 Points)	100%					
Sec 14: References (5 Points)	100%					
Project Management (13 Points)	60%		40%			

Individual Point Allocation

Team Member	Points
Rylan	33
Avinash	33
Aakash	33
Mozam	33
Mandeep	33
Pintu	33

Individual Work Description, Project Management, & Notes

The following is a brief description of what each team member completed for Report #3:

Rylan:

- Compiled summary of changes to describe Reports #1-2 changes
- Updated CSR to reflect delivered system
- Updated Glossary to reflect updated CSR
- Compiled references as needed
- Created History of Work, Current Status, & Future Work sections as needed
- Project Management
    - Coordinated meetings / meeting times
    - Collated reports, documents, etc.
    - Dropbox / Document Control management
    - Represented group / contact point with TA & Dr. Marsic
    - Edited, modified styling, etc. on submitted documents

Avinash:

- Updated the functional requirements section as needed
- Updated the interactions diagram section as needed; included the use of design patterns

Aakash:

- Assisted updating System Requirements documentation
- Updated documentation on UI Design / Implementation
- Updated System Architecture & Design section
- Assisted in writing new test designs
- Project Management
    - Sent out meeting invites as needed, coordinated meetings & meeting times

Mozam:

- Updated the functional requirements section as needed
- Updated the interactions diagram section as needed; included the use of design patterns

Mandeep:

- Assisted updating System Requirements documentation
- Co-authored effort estimation section
- Updated Domain Analysis section as needed
- Updated Class Diagrams & Interface Spec. as needed
- Assisted in writing new test designs

Pintu:

- Assisted updating System Requirements documentation
- Co-authored effort estimation section
- Updated Domain Analysis section as needed
- Updated Class Diagrams & Interface Spec. as needed
- Assisted in writing new test designs

NOTES:

- The section “Algorithms and Data Structures” does not apply to our project. We have removed the section.

- The following sections have been omitted from this version of the report and will be available in the v.2 of this document:
    - Class Diagram & Interface Spec.
    - UI Design / Implementation
    - Design of Tests

- As much as possible, we have given sections of work to their original members to update. For example, if Rylan worked on the CSR during Report #1, he was given the CSR to update during Report #3. However, in the event a team member was given a section other than their own to work on for Report #3, the original member was partially credit for their section, as well as the updating team member.

- We removed a variety of proposed features. Features such as the ability to trade Options and Bonds were cut due to information availability; it costs money to buy information of such products (> $200 / month). Other features such as ‘Social Media Connectivity’ were cut due to simplicity. They involved placing a Facebook link on our product. We removed such features to focus on more robust features.

Table of Contents

Summary of Changes

- In CSR, removed any reference to a 'Tutorial System', 'Bonds', 'Options'. Removed the 'Advertiser Role'. Removed a reference to caching of stock prices. Removed mobile application development. Removed connection to social media. Removed references to 'chat' feature. Removed tool to 'suggest-a-stock'. Removed 'Morningstar' box. Updated the description of league management.

- Updated Glossary to remove terms not used in CSR. Updated definitions of terms to make them easily understandable.

- System architecture section edited to match MVC / minor editing corrections.

- Effort estimation section added.

- System requirements section edited to remove advertisements, tutorial system, challenge player. Updated to reflect system administrator.

- Domain Analysis section edited to reflect updated development. Removed references to advertisers, tutorial system, and challenge player features. Updated and redistributed responsibilities to reflect redeveloped objects.

- Functional Requirements Specification updated to remove unnecessary actors. Use cases updated to remove un-implemented UCs. Fully dressed UCs updated to remove non-implemented UCs. UCs were removed based on the updated System Requirements. UC Diagram updated.

- Interaction Diagrams updated to reflect implemented UCs. Detail added to each diagram description.

- History of Work, Current Status, & Future Work added.

1.0 Customer Statement of Requirements

Pig E-Bank

115 W 42nd Street
New York, NY 10036

Mr. Money Nickel
CEO, Pig E-Bank

The Virtual Stock Market Project, Group #6
Rutgers University

RE: Virtual Stock Market

Dear Project Group #6,

It is my pleasure to let you know that your team has been awarded a contract to develop a virtual stock market application for our bank. As your customer, we have a few requirements for the software which we would like to detail to better aid you in understanding our business requirements for the software.

As commonly known in our industry, there are a variety of virtual stock market applications available for use and purchase. However, each of these systems lack critical components which we, as a bank, need to use in order to better train our associates. Many associates who join our company out of college, understand basic market concepts, but lack the comfort of investing money, or providing investment advice. These associates tend to lack the knowledge of related stocks. We want this software to be able to teach them to feel comfortable investing. At a high level, the system should have these basic features:

- Allow the buying and selling of multiple market products (including derivatives, stocks, and bonds)
- The system should be 'easy-to-use', colorful, and fun

There are some general system requirements which we need:

- Ability to run on a web platform. We need to be able to access the game worldwide, without installing software.
- Host 'Investment Games', which are virtual games where users can play against each other by developing simple portfolios, and using buying and selling strategies to make money
- Provide 3 user roles (detailed later), 'Player', 'League Administrator', 'Administrator'
- A real-time trading system which gets market-prices within 5 seconds of actual accuracy
- Simple registration system to make an account, and start a league
- Other virtual investment platforms are complex, and require multiple clicks to find simple data (e.g. a player's portfolio). The platform should allow all users (regardless of role) to be able to access critical data with minimal clicks.
- Live ticket system to broker stocks, and perform complex orders such as limit orders

For each user, the system should provide the following functional requirements:

- **Player:**
  - Ability to join multiple leagues at once, and participate in each game individually
  - The player should be able to view the league standings, see how other users are performing, view league settings (the start / end dates, the amount of start money, the portfolios of other users)
  - Develop (view, and modify) multiple portfolios (for each league), via the buying and selling of stock (via normal, shorted orders), mutual funds, and ETFs
  - The portfolio should be clearly defined and have the player's holdings, ticker symbol, company name, shares / contracts owned, market value, and total portfolio value
  - Check the price of a market product
  - Access live finance news which may impact market prices
  - Ability to invite a friend to join their league
  - The player should be able to set a watch-list of stocks they may want to purchase.
- **League Administrator:**
  - Functionality to setup league settings (start and end date, initial funds), and league rules (which finance products can be bought – all, stocks only, etc. This should be accomplished by a check-box system or something similar).
  - A simple league management page where the users and settings can be managed.
  - Option to kick a user out of a game, in the event they are being unruly
  - The league administrator should have all the functionality of a player, and have all of the abilities of a player.
  - Leagues will all be set to public visibility (anyone, including non-league participants) should be able to see a league, members, and their portfolios
- **Administrator:**
  - The ability to delete a user or league
  - Functionality to disable / enable the platform for maintenance
  - A backup functionality to backup and restore the site
  - Functionality to disable the site as needed

We are looking forward to seeing your development of these features and functionalities. If you have any questions about our requirements, feel free to contact at us at my above address.

Regards,

Mr. Money Nickel
CEO, Pig E-Bank

2.0 Glossary of Terms

ETF – Exact duplicate of a mutual fund, and can be traded during investment hours.

Investment – The process of buying securities, in hopes of growing the invested money for the future.

League – A group of investors who play against each other. They are ranked by the growth of their individual portfolios.

Market – An interactive forum for buying and selling financial products.

Market Capitalization – How much the market values a company. The market cap is defined by the share price times the number of outstanding shares.

Mutual Fund – A fund which takes an investor's money, and invests it collectively, providing an equal return to each investor. A mutual fund cannot be traded during market hours.

Order / Limit Order – *See Trade*. A Limit Order is a trade set to execute when the market price of a security reaches a specified price.

Portfolio – A collection of stocks, bonds, derivatives, and mutual funds owned by a player. The value of the portfolio is the sum value of its contents.

Risk – The qualitative property of a security with respect to how probable it may or may not grow money over time. Typically, stocks are considered to have more risk than certain bonds. Whereas options are even more risky.

Security – A market product such as a stock, bond, ETF, Mutual Fund, Option, etc. which has some monetary value.

Share – A fraction of a publicly owned company which may be traded in a market.

Stocks – A share in a publicly owned company. The share can be bought by a player, and put into their portfolio.

Ticket / Trading System – A system which takes a user's trades and processes them. It exchanges the user's money in the portfolio for a security. The system is able to lookup the value of a security at a given time.

Trade – A transaction where a user exchanges funds (money) for a security.

3.0 System Requirements

3.1 Functional Requirements

ID	PW	Requirement
REQ-1	5	The system shall allow new Players to register an account with their email, which should be external to our website. Required information shall include a unique username, password that meets the guidelines, as well as Player's first and last name, birth date and gender. Upon completion of successful registration, the Player account balance shall be decided by Game Administrator.
REQ-2	5	The system shall support placement of order by filling out an order ticket. The order ticket should contain client's information, order type, quantity, price and additional instructions. The system shall periodically review the queued orders process them when conditions are met.
REQ-3	5	The system shall review the order queue periodically and: 1. If all the conditions are matched, convert order into a market order and execute. 2. Else if, the order is expired or cancelled, remove from the queue and mark it failed. 3. Else, none of above, leave untouched. If either 1 or 2 is executed, the system shall record the transaction and notify the Player by sending a confirmation message.
REQ-4	5	The system shall maintain a database of Player portfolios and transactions. The database will also include league status for each player.
REQ-5	4	The system shall support creation of new leagues or entry to existing leagues. Players shall be allowed to create leagues and specify duration, capital limits, allowed sectors and entrance fees. The system shall also keep track of leagues' status based on investment returns.
REQ-6	4	The system shall provide market data (price data, bid/ask sizes, volume and news feed of relevant articles) for set of companies.
REQ-7	3	The system will have FAQ page, where Players will learn how to use system.
REQ-8	1	The system shall allow Players to submit technical problems and comments to the system administrator.

3.2 Nonfunctional Requirements

ID	PW	Requirement
REQ-9	5	The system shall be simple to understand and use with minimal knowledge of a Player's learning curve. The layout of the page should be simple and easy to understand, and contain most of the contents on fewer pages.
REQ-10	5	The system shall maintain and store all the data and information on the system's database and not allow any data or information to be stored on Player's device. The system shall not allow Player to directly modify any data. Two copies of any record shall be kept in case of a failure.
REQ-11	3	The system shall be able to run on different platforms such as Windows, Unix, or Mac. It should the same theme and consistency between different browsers.
REQ-12	4	The system shall be efficient as possible, allowing Players to start a game within 5 clicks, buy a stock within 3 clicks, and view a portfolio in 2 clicks.

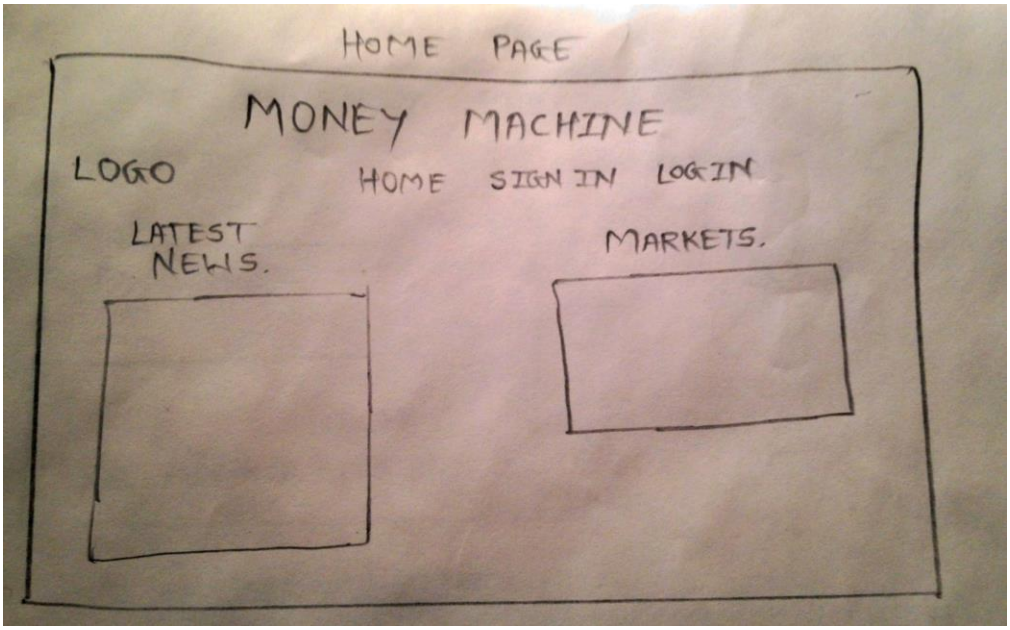
3.3 On-Screen Appearance Requirements

ID	PW	Requirement
REQ-13	5	The system must fit within a browser window of any browser.
REQ-14	3	The system must have a consistent look across different browsers and screen resolutions.

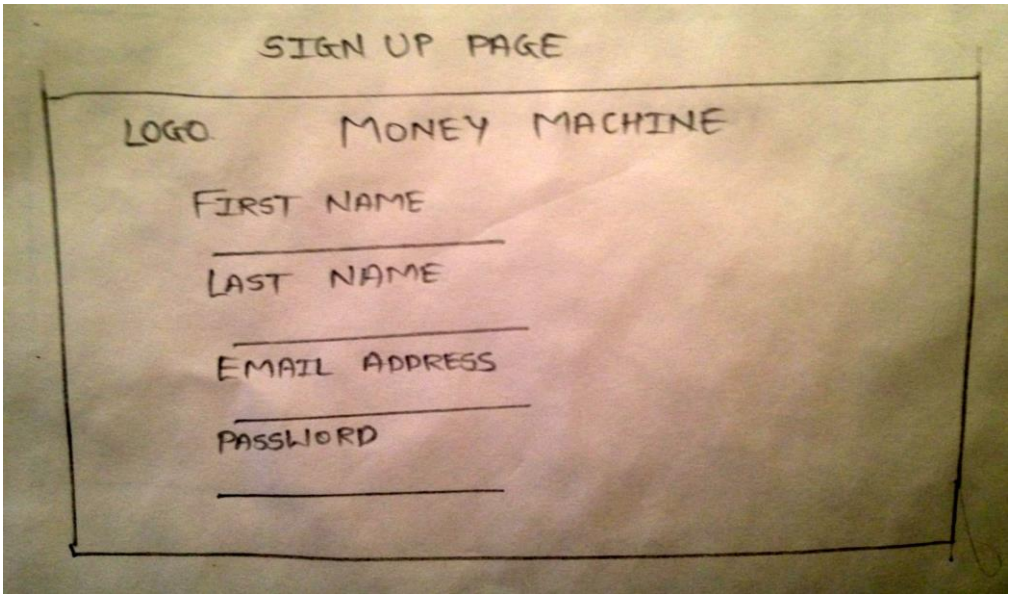
3.3.1 Screen Mockups

The following are mockups of specific pages from the project. They provide a rough idea of how specific, important pages of the project will look.

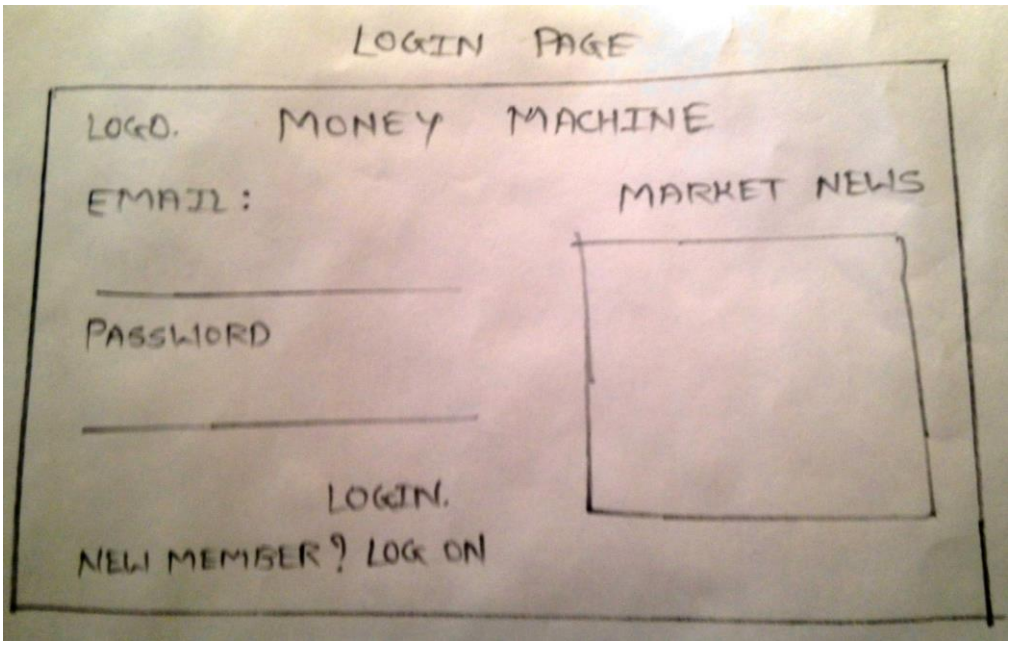
Home Page:



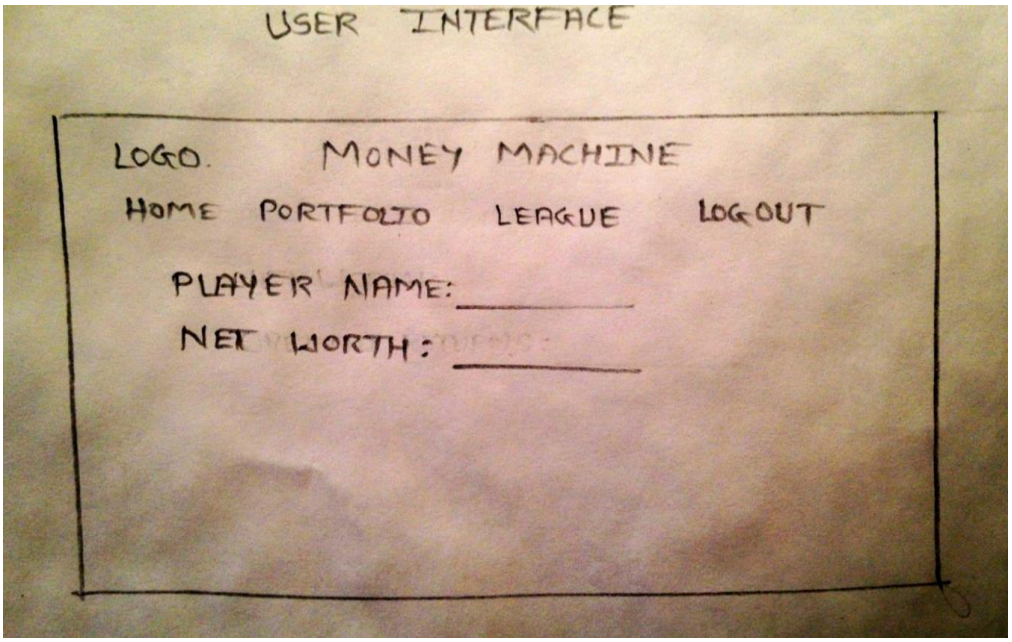
Sign Up Page:



Login Page:



League Interface Page (Dashboard):



4.0 Functional Requirements Specification

4.1 Stakeholders

- Potential Investors
- System Administrators

4.2 Actors & Goals

- **Player**:
  Type - Initiating Actor, Participating Actor
  Goals - Access security information, buy and sell securities, create investment games, and view watch lists

- **Visitor**:
  Type - Initiating Actor
  Goal - To register for full access to the system.

- **Game Administrator**:
  Type - Initiating Actor, Participating Actor
  Goals - Manage an investment game. Start or end an investment game.

- **System Administrator**:
  Type - Initiating Actor, Participating Actor
  Goals - Maintain web presence, view suggestions from players, and provide strategic enhancements to website operations.

- **Trade Database**:
  Type - Participating Actor

- **Player Database**:
  Type - Participating Actor

- **Security Data Provider**:
  Type - Participating Actor
  Goals - Provide information in relation to securities. Handle trade creation and modification.

- **Web Server**:
  Type - Participating Actor

4.3 Use Cases

4.3.1 Casual Description

Use Case UC-1: Register

Actor: Visitor (Initiating), Player Database (Participating), Web Server (Participating)
Goal: To register for a new account. A new player account will be created based on information provided from the visitor.

Use Case UC-2: Research Security

Actor: Player (Initiating), Security Data Provider (Participating), Web Server (Participating)
Goal: To provide information such as last price, bid/ask prices, fundamentals, charts, news, etc. Such information will be provided mainly from the Security Data Provider.

Use Case UC-3: Buy Security

Actor: Player (Initiating), Security Data Provider (Participating), Trade Database (Participating), Player Database (Participating), Web Server (Participating)
Goal: To purchase a security such as a bond, stock, option, etc. This will generate an order ticket which will contain order type (market, limit, buy to close, etc.) , security name/ ID, execution price, and time to expiry (Good Until Cancelled or Day Order). Prices will be provided from the Security Data Provider.

Use Case UC-4: Sell Security

Actor: Player (Initiating), Security Data Provider (Participating), Trade Database (Participating), Web Server (Participating)
Goal: To sell a security such as a bond, stock, option, etc. This will generate an order ticket which will contain order type (market, limit, sell to open, etc.), security name/ID, execution price, and time to expiry (Good Until Cancelled or Day Order). Prices will be provided from the Security Data Provider.

Use Case UC-5: View Portfolio

Actor: Player (Initiating), Security Data Provider (Participating), Trade Database (Participating), Player Database (Participating), Web Server (Participating)
Goal: To view current securities held, as well as available cash to withdraw/invest. This will be displayed for each league the player is a part of. Will also display current value of portfolios.

Use Case UC-6: View Transactions

Actor: Player (Initiating), Trade Database (Participating), Web Server (Participating)
Goal: To show pending, filled and cancelled transactions for the player.

Use Case UC-7: Create Investment Game

Actor: Player (Initiating), Player Database (Participating), Web Server (Participating)
Goal: To create games where an initiating player becomes the game administrator of the created game.

Use Case UC-8: Join Investment Game

Actor: Player (Initiating), Player Database (Participating), Web Server (Participating)
Goal: To join an investment game.

Use Case UC-9: Manage Investment Game

Actor: Game Administrator (Initiating), Player Database (Participating), Web Server (Participating)
Goal: To add/remove players from the game as well as accept/decline requests to join game.

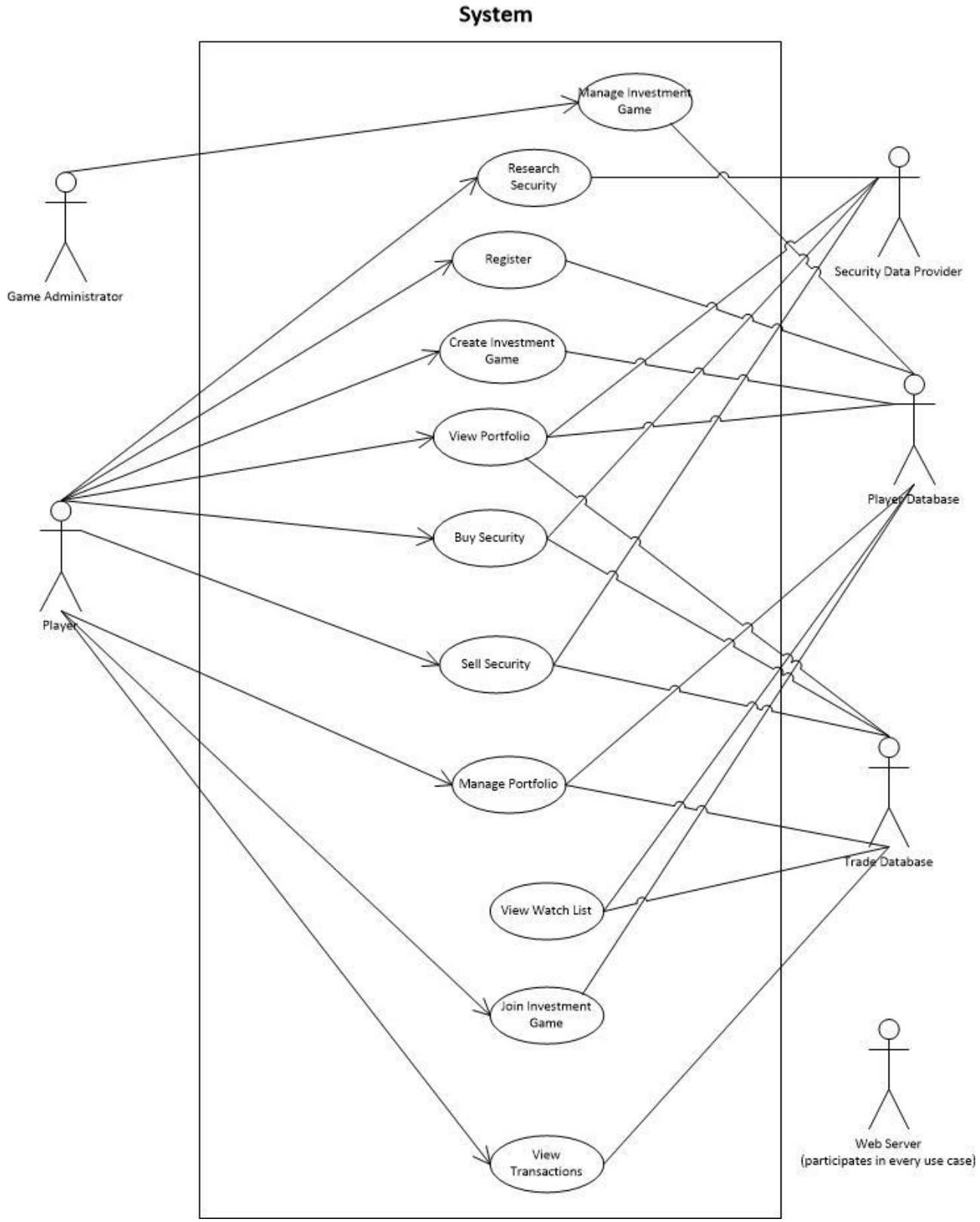
Use Case UC-10: Manage Portfolio

Actor: Player (Initiating), Player Database (Participating), Trade Database (Participating), Web Server (Participating)
Goal: To buy/sell & research securities.

Use Case UC-11: View Watch List

Actor: Player (Initiating), Trade Database (Participating), Player Database (Participating), Web Server (Participating)
Goal: To watch and track various security prices for securities which they may/may not have in their portfolio.

4.3.2 Use Case Diagram



4.3.3 Traceability Matrix

	R1	R2	R3	R4	R5	R6	R7		
PW	**5**	**5**	**5**	**5**	**4**	**4**	**4**	**Max**	**Total**
UC-01	x			x				5	10
UC-02						x		4	4
UC-03		x	x	x		x		5	19
UC-04		x	x	x		x		5	19
UC-05				x				5	5
UC-06			x	x				5	10
UC-07				x	x		x	5	13
UC-08				x	x			5	9
UC-09				x	x			5	9
UC-10				x				5	5
UC-11				x		x		5	9

4.3.4 Fully-Dressed Description

Use Case UC-1: Register

Related Requirements: REQ-1

Initiating Actor: Visitor

Initiating Actor's Goal: To register for a new account. A new player account will be created based on information provided from the visitor.

Participating Actors: Player Database, Web Server

Precondition: The visitor does not already have an account in the system.

Postcondition: The visitor successfully creates a new player profile and an appropriate entry is created in the Player database.

Flow of Events for Main Success Scenario:

1 → The visitor clicks the "Register" button or the visitor attempts to access a feature that is only for members.

2 ← The system provides the visitor with the registration page.

3 → The visitor submits the information to the system.

4 ← The system verifies the visitor's information and inserts this information into the Player Database.

5 ← The system provides confirmation to the visitor that their information was valid and a new profile was created successfully.

Flow of Events for Username/Email already in use:

1 → The visitor clicks the "Register" button or the visitor attempts to access a feature that is only for Players.

2 ← The system provides the visitor with the registration page.

3 → The visitor submits the information to the system.

4 ← The system attempts to verify the information. It finds that the username or email address is already in use.

5 ← The system generates an error and presents the registration page back to the user for editing.

Use Case UC-3: Buy Security

Related Requirements: REQ-2, REQ-3, REQ-4, REQ-6, REQ-7
Initiating Actor: Player
Initiating Actor's Goal: To buy to close or buy to open a position.
Participating Actors: Player Database, Trade Database, Security Data Provider, Web Server
Precondition: Player must have enough balance to purchase the security.
Postcondition: The cost of the order is debited from the player's total balance and an order ticket is generated.
Flow of Events for Success Scenario

1 → The player chooses a security from their portfolio or from one of the supported markets.
2 ← The Security Data Provider retrieves information for the chosen security such as bid/ask spread, last price, volume traded, and other metrics for the security.
3 → The player then fills out information such as order type, expiry date and number of securities to buy.
4 ← An order ticket is generated and forwarded to the trade database for processing and order confirmation is provided to the player as well as a temporary hold is placed on the player's account for the cost of the order.
5 ← The player is notified when their order is filled. The cost of the order is debited from the user's portfolio and corresponding security is added to the portfolio.

Flow of Events for Insufficient Funds

1 → The player chooses a security from their portfolio or from one of the supported markets.
2 ← The Security Data Provider retrieves information for the chosen security such as bid/ask spread, last price, volume traded, and other metrics for the security.
3 → The player then fills out information such as order type, expiry date and number of securities to buy.
4← The system determines that the player has insufficient funds to buy security. The order ticket is destroyed and order is re-forwarded to the player for editing.

Use Case UC-4: Sell Security

Related Requirements: REQ-2, REQ-3, REQ-4, REQ-6, REQ-7
Initiating Actor: Player
Initiating Actor's Goal: To sell a security such as a bond, stock, option, etc to close or open a position.
Participating Actors: Security Data Provider, Trade Database, Web Server
Precondition: The player must either own the security, or must have enough money to put into a margin account.
Postcondition: The cost of the order is credited to the player's total balance and an order ticket is generated.
Flow of Events for Main Success Scenario:

1 → The player chooses a security from their portfolio or from one of the supported markets.
2 ← The Security Data Provider retrieves information for the chosen security such as bid/ask spread, last price, volume traded, and other metrics from the security.
3 → The player then fills out information such as order type, expiry date and number of securities to sell.
4 ← An order ticket is generated and inserted into the trade database. The player is provided with an order confirmation that confirms their order has been placed.
5 ← The player is notified when their order is filled. The cost of the order is credited to the player's portfolio, and the corresponding security (if the player owned it originally) is removed.

Flow of Events For Not Enough Margin:

1 → The player chooses a security from their portfolio or from one of the supported markets.
2 ← The Security Data Provider retrieves information for the chosen security such as bid/ask

spread, last price, volume traded, and other metrics from the security.
3 → The player then fills out information such as order type, expiry date and number of securities to sell.
4 ← The system determines that the player does not have enough shares and does not have a high enough balance in their margin account. The order ticket is not placed for processing and is presented to the player for editing.

Flow of Events for Not Enough Stock (with no margin account):

1 → The player chooses a security from their portfolio or from one of the supported markets.
2 ← The Security Data Provider retrieves information for the chosen security such as bid/ask spread, last price, volume traded, and other metrics from the security.
3 → The player then fills out information such as order type, expiry date and number of securities to sell.
4 ← The system determines that the player does not have enough shares and does not have a margin account. The order ticket is not placed for processing and is presented to the player for editing.

Use Case UC-7: Create Investment Games

Related Requirements: REQ-5
Initiating Actor: Player
Initiating Actor's Goal: To initiate an investment game
Participating Actors: Player Database, Web Server
Precondition: Initiating player must be a registered user
Postcondition: The initiating player must become a game coordinator for the specific game and the game should be created.

Flow of Events for Main Success Scenario:

1 → The initiating player clicks on the tab "Create Game" and gets prompted to fill out a form which includes title of the game, a web url which directly links to the game, comment block (optional) and an expiry date .
2→ The game has a unique title and above field data is forwarded to the player database.
3 ← New data would be added to the player database and the player becomes the game administrator for the initiated game.
4 ← Player is notified that the game has been created.

Flow of Events for Duplicate Game Title

1 → The initiating player clicks on the tab "Create Game" and gets prompted to fill out a form which includes title of the game, a web url which directly links to the game, comment block (optional) and an expiry date .
2 ←The game title already exists in the player database and the initiating player is notified an "Invalid Name" error.
3 ← The Player is redirected to create game.

4.4 System Sequence Diagrams

UC-1: Register

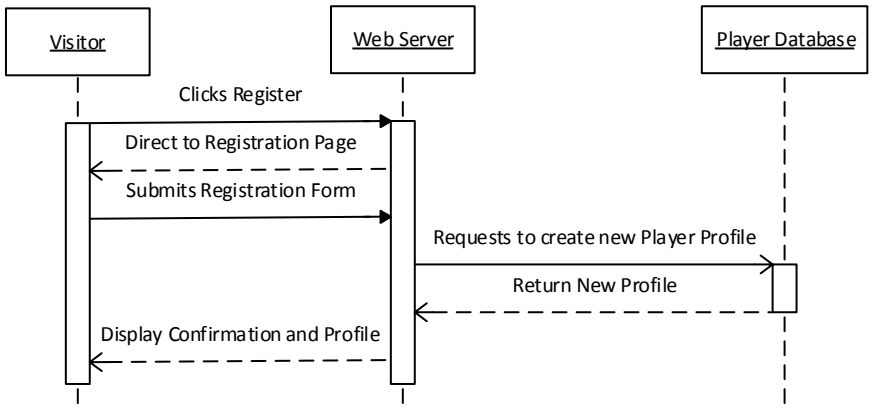
In this system sequence diagram, the visitor first navigates to the website. After reaching the website, the visitor clicks “Register”. After this, the visitor is presented with the registration page.

Once the user has submitted the registration page, the information provided is validated and is sent to the Player Database. The system then requests for a new player profile to be created for the visitor. The system then returns to the visitor that their profile creation was complete, and that they are now logged into the system.

The only alternate scenario to the main success scenario would be if any of the information entered by the user was invalid. In this situation, the system would return an error to the Visitor letting them know that there was an error in their submission. It would give the user another chance to submit the registration form.

As of now, no other implementations have been discussed, as the current one seems to be the most logical flow of events.

UC-1 System Sequence Diagram:

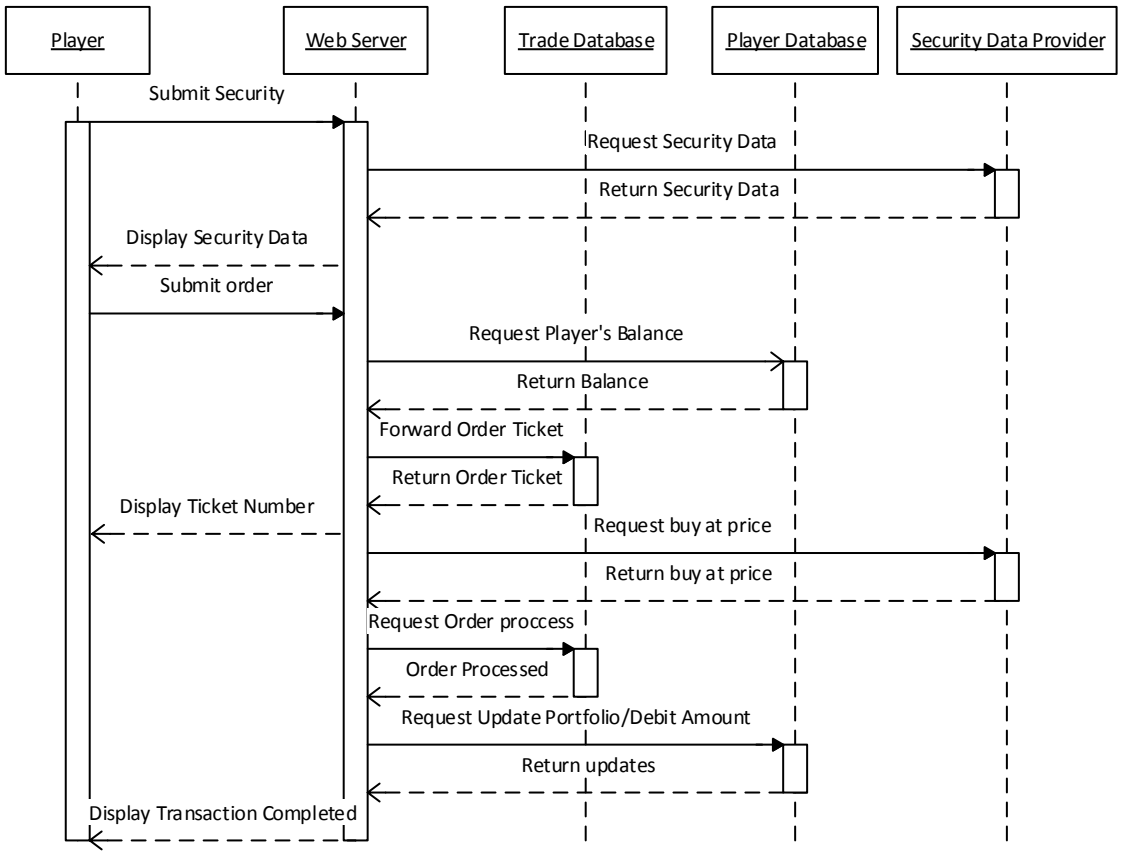


UC-3: Buy Security

In this system, the player selects a security of interest and in return the system should display the latest data of the security. This is done when the web server connects to the security data provider to read the data. The data is then displayed to the player. Then, the player must fill out order form which most importantly includes the buying price of the security. Upon clicking submit, the web server verifies if the player has enough balance to purchase. If the player has enough balance then the system requests an order ticket from the trade database for the particular security (securities). The player is displayed with a confirmation of order and order ticket number. The Web Server constantly reads the data of the particular security through the security data provider. Once, the parameters of the player match the current data, the system requests

the trade database to process the order. Order is then processed and player's portfolio is updated. A notification is also sent to the player informing that the transaction is completed.

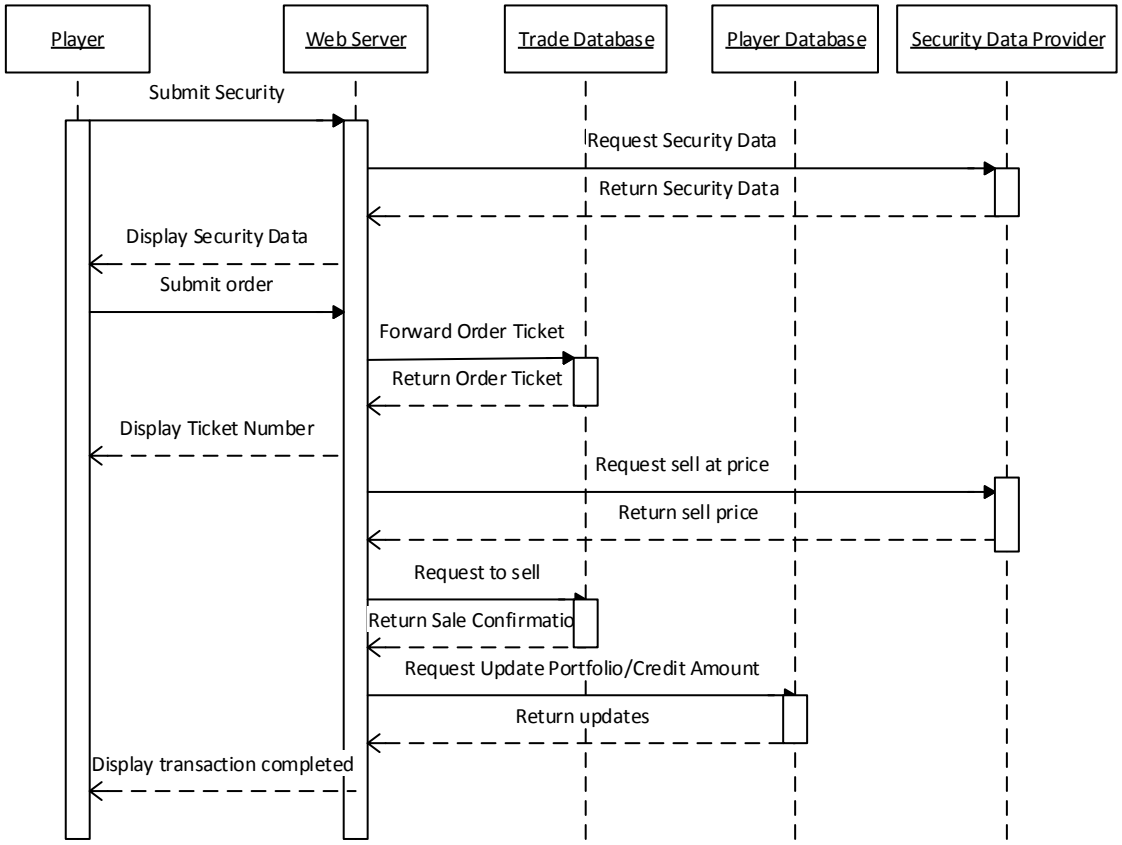
UC-3 System Sequence Diagram:



UC-4: Sell Security

In this system, the player selects a security of interest and in return the system should display the latest data of the security. This is done when the web server connects to the security data provider to read the data. The data is displayed to the player. Then, the player must fill out order form which most importantly includes the selling price of the security. A request for generating an order ticket is then sent out to the trade database. Once the order ticket has been generated the player is displayed confirmation of the order and the ticket number. The system constantly reads the data through the security data provider and once the price to sell his matched with the user's parameters the order is sent to be processed to the trade database.
Order is then processed and player's portfolio is updated. A notification is also sent to the player informing that the transaction is completed.

UC-4 System Sequence Diagram:

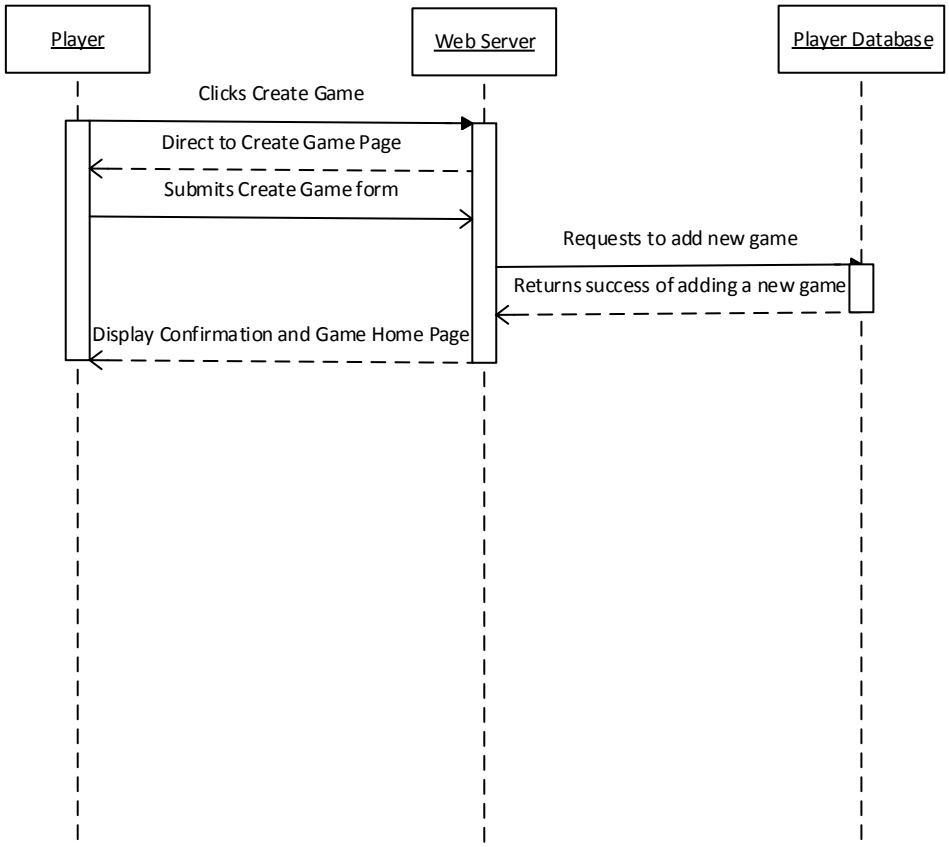


UC-7: Create Game

In this system sequence diagram, the Player requests for a new game to be created. The system then presents the user with the Create Game page. One the user has submitted the registration page, the information is validated and a request to create a new game is sent to the Player Database. The system then updated required fields in the Player Database and signals a success to the Web Server. The Web Server signals this back to the user with a confirmation that their game has successfully been create.

The only alternate scenario to the main success scenario would be if the game name the Player is trying to make is already taken. In this situation, the system would return an error after form submission letting the user know that their game name is already taken. It would ask the Player to choose another game name and go through the same process of revalidating.
As of now, no other implementations have been discussed, as the current one seems to be the most logical flow of events.

UC-7 System Sequence Diagram:



5.0 Effort Estimation

Use Case Points (UCP) method provides the ability to estimate the person-hours a software project requires based on its use cases. UCP method analyzes the use case actors, scenarios, nonfunctional requirements, and environmental factors and joins them in a simple equation:

$$UCP = UUCP \times TCF \times ECF$$

- Unadjusted Use Case Points (UUCP) – Measures complexity of functional requirements
- Technical Complexity Factor (TCF) – Measures complexity of nonfunctional requirements
- Environmental Complexity Factor (ECF) – Assesses development teams experience and their development environment

5.1 Unadjusted Use Case Points

UUCP are computed as a sum of the following two components:

- Unadjusted Actor Weight (UAW) – Combined complexity of all the actors in all the use cases
- Unadjusted Use Case Weight (UUCW) – Total number of activities contained in all the use case scenarios

$$UUCP = UAW + UUCW$$

5.1.1 Unadjusted Actor Weight

The weights for Actor classification are computed via the following table: Actor classification and associated weights

Actor	Description of Actor Type	Weight
Simple	The actor is another system which interacts with our system through a defined application programming interface (API)	1
Average	The actor is a person interacting through a text-based user interface, or another system interacting through a protocol, such as a network communication protocol	2
Complex	The actor is a person interacting via a graphical user interface	3

Actor Classification for Money Machine:

Actor	Description of Characteristics	Complexity	Weight
Player	Player is interacting with the system through a graphical user interface.	Complex	3
Game Administrator	League Admin is interacting with the system through a graphical user interface.	Complex	3
System Administrator	System Administrator is interacting with the system via a graphical user interface. (Creators of Money Machine)	Complex	3
Security Data Provider	Security Information Provider (Yahoo Finance) is interacting with the system through a network protocol.	Average	2
Trade Database	Database isanother system interacting through a protocol.	Average	2
Player Database	Database isanother system interacting through a protocol.	Average	2
Web Server	Web Serverisanother system interacting through HTTP	Average	2

UAW (Money Machine) = 4 x Average + 3 x Complex = 17

5.1.2 Unadjusted Use Case Weight

The weights for use cases are computed via the following table:

Use Case Category	Description of Category	Weight
Simple	Simple user interface. Up to one participating actor (plus initiating actor). Number of steps for the success scenario: no more than 3. If presently available, its domain model includes no more than 3 concepts.	5
Average	Moderate interface design. Two or more participating actors. Number of steps for the success scenario: 4 to 7. If presently available, its domain model includes between 5 and 10 concepts.	10
Complex	Complex user interface or processing. Three or more participating actors. Number of steps for the success scenario: at least 7. If available, its domain model includes at least 10 concepts.	15

Use case classification for Money Machine:

Use Case	Description	Complexity	Weight
Register (UC-1)	Simple user interface. 7 steps for main success scenario. Three participating actors (Player, Database).	Average	10
Research Security (UC-2)	Simple user interface. 4 steps for main success scenario. Three participating actors (Player, Database, Stock Info Provider).	Average	10
Buy Security (UC-3)	Simple user interface. 8 steps for main success scenario. Three participating actors (Player, Database, Stock Info Provider).	Complex	15
Sell Security (UC-4)	Simple user interface. 10 steps for main success scenario. Three participating actors (Player, Database, Stock Info Provider).	Complex	15
View Portfolio (UC-5)	Simple user interface. 6 steps for main success scenario. Three participating actors (Player, Database, Stock Info Provider).	Average	10
View Transactions (UC-6)	Simple user interface. 6 steps for main success scenario. Two participating actors (Player, Database).	Average	10
Create Investment Game (UC-7)	Simple user interface. 4 steps for main success scenario. Three participating actors (Investor, Database, Web Server).	Average	10
Join Investment League (UC-8)	Simple user interface. 5 steps for main success scenario. Three participating actors (Investor, Database, Web Server).	Average	10
Manage Investment Game (UC-9)	Moderate user interface. 4 steps for main success scenario. Two participating actors (League Coordinator, Database).	Average	10
Manage Portfolio (UC-10)	Moderate user interface. 4 steps for main success scenario. Two participating actors (FundManager, Database).	Average	10
View Watch-List (UC-11)	Simple user interface. 4 steps for main success scenario. Two participating actors (Investor, Database).	Simple	5

UUCW(Money Machine) = 1 x Simple + 8 x Average + 2 x Complex = 115

5.1.3 Computing Unadjusted Use Case Points

UAW (Money Machine) = 4 ∗ Average + 3 ∗ Complex = 17
UUCW(Money Machine) = 1 ∗ Simple + 8 * Average + 2 ∗ Complex = 115
UUCP(Money Machine) = UAW + UUCW = 17 + 115 = 132

5.2 Technical Complexity Factor

Technical Complexity Factor (TCF) is computed using thirteen standard technical factors to estimate the impact of productivity of the nonfunctional requirements for the application. We then need to assess the perceived complexity of each technical factor in the context of the project. A perceived complexity value is between 0 and 5, with 0 meaning trivial effort, 3 meaning average effort and 5 meaning major effort. Each factors weight is then multiplied by perceived complexity factor to produce calculated factor. Two constants

are used with TCF. The constants utilized are C1 = 0.6 and C2 = 0.01.

$$TCF = Constant1 + Constant2 \times TechnicalFactorTotal = C_1 + C_2 \cdot \sum_{i=1}^{13} W_i \cdot F_i$$

Technical complexity factors and their weights:

Technical Factor	Description	Weight
T1	Distributed system	2
T2	Performance objectives	1
T3	End-user efficiency	1
T4	Complex internal processing	1
T5	Reusable design or code	1
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	Portable	2
T9	Easy to change	1
T10	Concurrent use	1
T11	Special security features	1
T12	Provides direct access for third parties	1
T13	Special user training facilities are required	1

Technical complexity factors for Money Machine:
(PC = Perceived Complexity, CF = Calculated Factor)

Technical Factor	Description	Weight	PC	CF
T1	Distributed, web-based system	2	3	6
T2	User expects good performance, but will tolerate network latency	1	3	3
T3	End-user expects efficiency, which is achieved through caching.	1	4	4
T4	Internal processing required multiple interactions with other subsystems	1	4	4
T5	No requirement for reusability	1	0	0
T6	No user installation required	0.5	2	1
T7	Ease of use was very important	0.5	5	2.5
T8	Portable since it runs in a browser	2	2	4
T9	Relatively simple to add new features	1	2	2
T10	Concurrent use is required	1	4	4
T11	Security handled by Database	1	0	0
T12	No direct access for third parties	1	0	0
T13	No training required	1	0	0

TCF = 0.6 + (0.01 × 30.5) = 0.0905.

This results in a decrease of TCF by 9.50%.

5.3 Environment Complexity Factor

The Environment Complexity Factor (ECF) is computed utilizing eight standard environmental factors to measure the experience level of the people on the project and the stability of the project. We then need to assess the perceived impact based on perception that factor has on projects success. 1 means strong negative impact, 3 is average and 5 means strong positive impact.TCF is computed utilizing thirteen standard technical factors to estimate the impact of productivity of the nonfunctional requirements for the application. We then need to assess the perceived complexity of each technical factor in the context of the project. A perceived complexity value is between 0 and 5 with 0 meaning that it is irrelevant, 3 means average effort and 5 means major effort. Each factors weight is then multiplied by perceived complexity factor to produce calculated factor. Two constants are used with ECF. The constants utilized are C1 = 1.4 and C2 = -0.03.

$$ECF = Constant1 + Constant2 \times EnvironmentalFactorTotal = C_1 + C_2 \cdot \sum_{i=1}^{8} W_i \cdot F_i$$

Environmental complexity factors and their weights:

Environmental Factor	Description	Weight
E1	Familiar with the development process	1.5
E2	Application problem experience	0.5
E3	Paradigm experience	1
E4	Lead analyst capability	0.5
E5	Motivation	1
E6	Stable requirements	1
E7	Part-time staff	-1
E8	Difficult programming language	-1

Environmental Complexity Factors for Money Machine: PI = Perceived Impact, CF = Calculated Factor

Environment Factor	Description	Weight	PI	CF
E1	Beginner familiarity with UML-based development	1.5	1	1.5
E2	Half of team has familiarity	0.5	3	1.7
E3	Some knowledge of object-oriented approach	1	3	3
E4	Average lead analyst	0.5	2	1
E5	Highly motivated overall	1	4	4
E6	Requirements were stable	2	5	10
E7	Student staff (part-time)	-1	4	-4
E8	Used new programming languages but resources were readily available	-1	5	-5

ECF = 1.4 - (0.03 × 12.2) = 1.034
This results in an increase of UDP by 3.4%.

5.4 Calculating the Use Case Points

As mentioned earlier, UCP = UUCP × TCF × ECF.
From above calculations, UCP variables have the following values:

UUCP = 132
TCF = 0.905
ECF = 1.034
UCP = 132 × 0.905 × 1.034 = 123.52 or 124 use case points.

5.5 Deriving Project Duration from Use-Case Points

UCP is a measure of software size. We need to know the team's rate of progress through the use cases. We need to utilize the UCP and Productivity Factor (PF) to determine duration. The equation for computing Duration is:

Duration = UCP x PF

Productivity Factor is the ratio of development person-hours needed per use case point. Assuming a PF = 34, the duration of our system is computed as follows:

Duration = UCP x PF = 124 x 34 = 4216 person-hours for the development of the system.

This does not imply that the project will be completed in 4216/24 = 175 days 16 hours. A reasonable assumption is that each developer on average spent 18 hours per week on project tasks. With a team of six developers, this means the team makes 6 *18 = 108 hours per week. Dividing 4216 person-hours by 108 hours per week, we obtain the total of approximately 39 weeks to complete this project. We have spent 15 weeks approximately on the project so far which gives us 24 weeks left to complete this project according to our estimation. The reason for the large estimate is due to the highly over-estimated productivity factor.

6.0 Domain Analysis

6.1 Domain Model

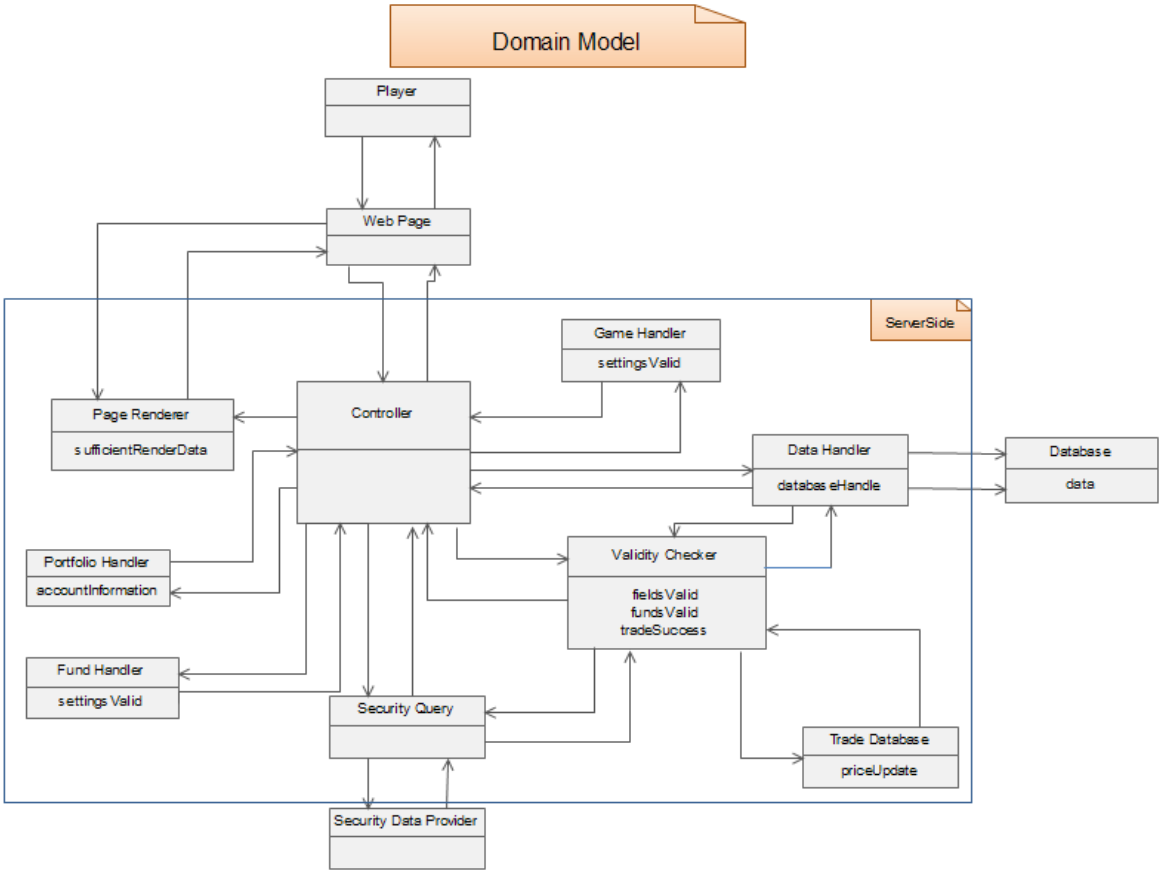


Figure 1: Domain Model

Figure 1, shows Money Machine's new, updated Domain Model. The subsequent diagrams give insight into how the concepts work to satisfy the key use cases of our updated website. The old domain model contained a Facebook concept, which is not in this Domain Model. The domain model is similar to the old domain model, but there are some changes to how the concepts will interact. In addition, the web server, web browser, and web framework is replaced with just web page. There are new concepts names added which are related to the new use case names.

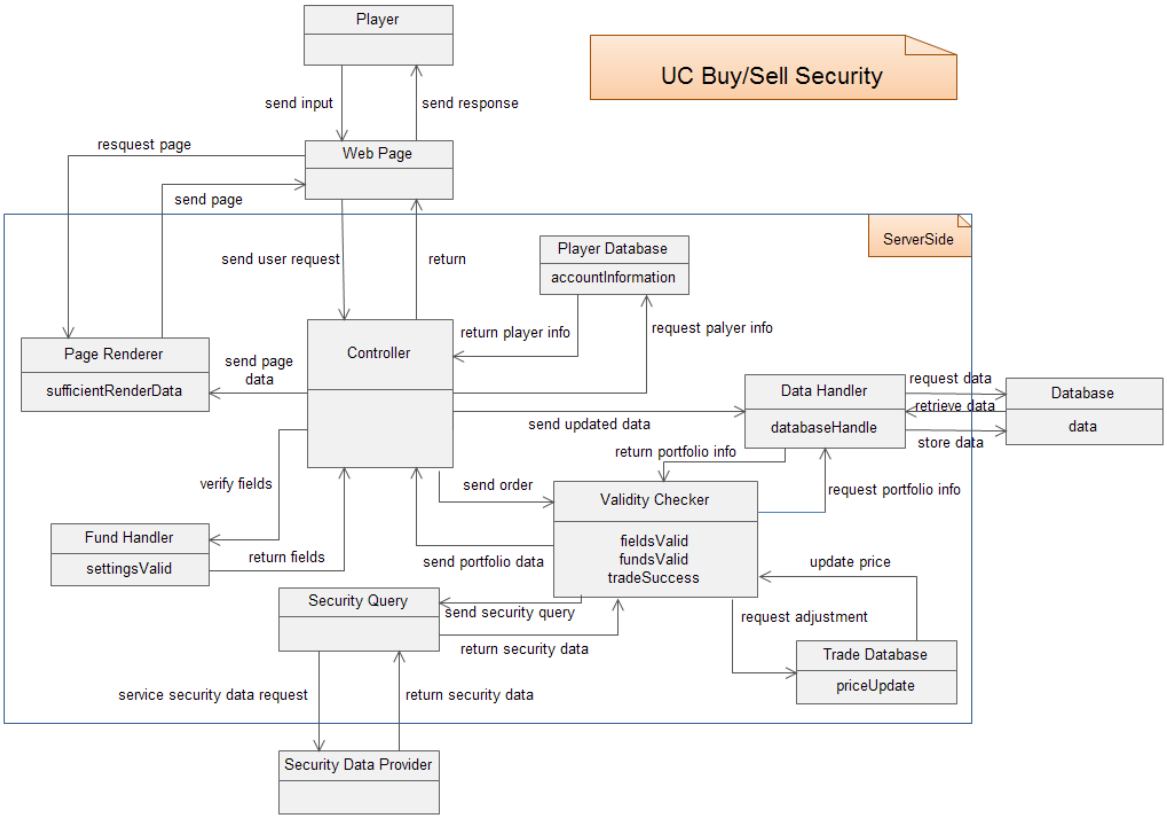


Figure 2: Buy/Sell Security

Figure 2, represent both our buy (UC-3) and sell (UC-4) use cases since they behave in the same way. The User fills out order information on the web page, and sends to request to order to the Controller. The controller relays the order to the Validity Checker so that it can send the corresponding security query to the Security Query concept, which fetches the necessary information from the remote Security Info Provider. The Validity Checker then sends a request to the Trade Database to adjust the stock price based on our current trade. Now the Validity Checker must retrieve the User's balance in order to verify the transaction is valid; it requests for the Data Handler to get this information from the Database. If the transaction is successful, the Controller tells the Data Handler to update the User's portfolio. Then the Controller will let the Page Renderer know what page to generate and pass necessary data. The Web Page is informed of the completion of the order and knows to request the page to be viewed from the Page Renderer.

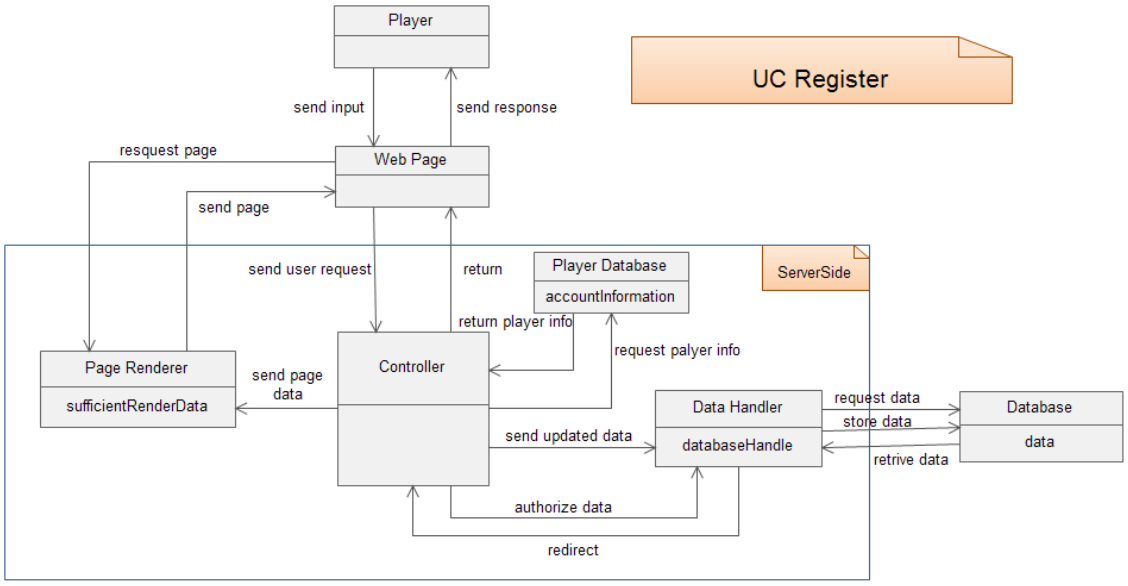


Figure 3: Register

The UC-6 Register is represented in Figure 3. First, the User tries to access the Player Portfolio, but he is not registered, so the web page tells the Controller to render the new registration page. The Controller will then send instructions to the Data Handler to a new account in our Database. The Controller also notifies to create a new Player Portfolio. The Controller then passes necessary data to the Page Renderer and informs Web Page that the process is complete. The Web Page will call for the Page Renderer to generate Player Portfolio page to be viewed.

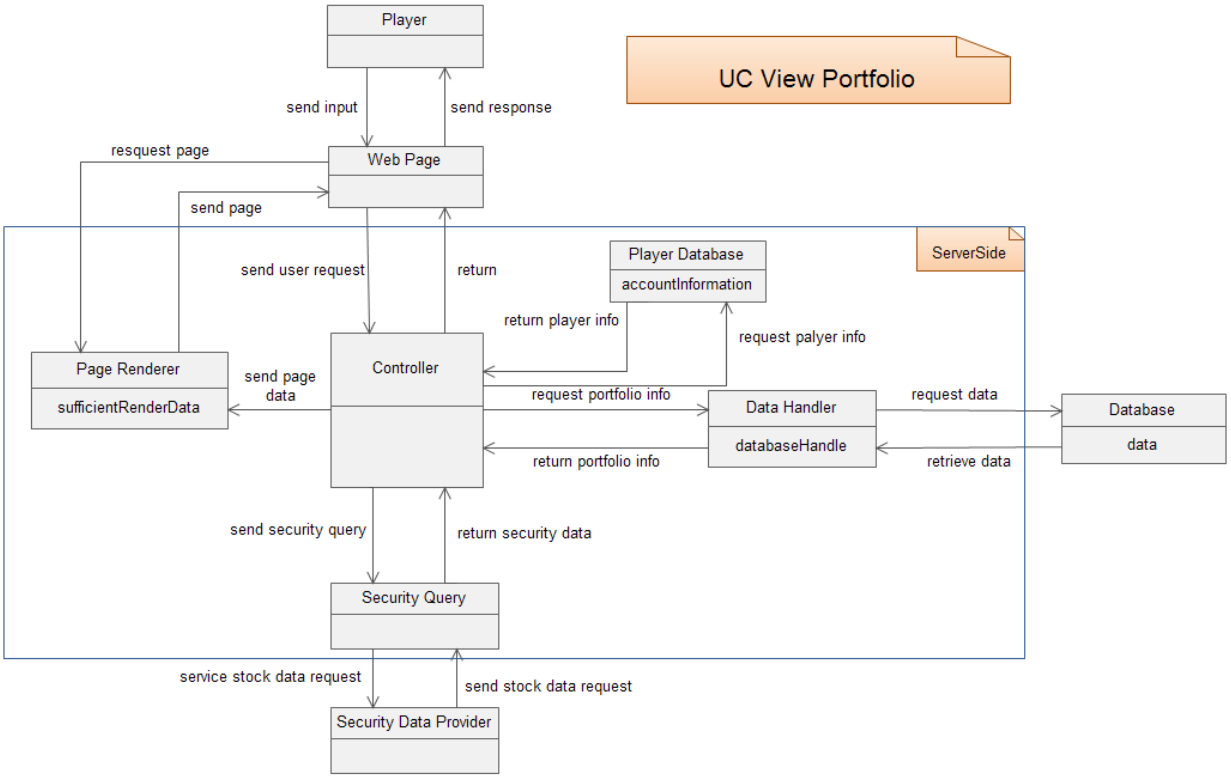


Figure 4: View Portfolio

Figure 4, shows the UC-5, View Portfolio. The Player queries about the portfolio from the Web Page, and this request gets sent to the Controller. To get the necessary data, the Controller will send a request for the portfolio info to the Data Handler, which obtains this data from the Database. The Controller will then query Security Query for each security held by the Player, which will obtain the necessary information from Security Info Provider. The portfolio is now ready to be viewed, so Controller gives the Page Renderer all necessary data and then lets the Web Page know the process has been finished. The Web Page requests the Page Renderer to create the required page to be viewed.

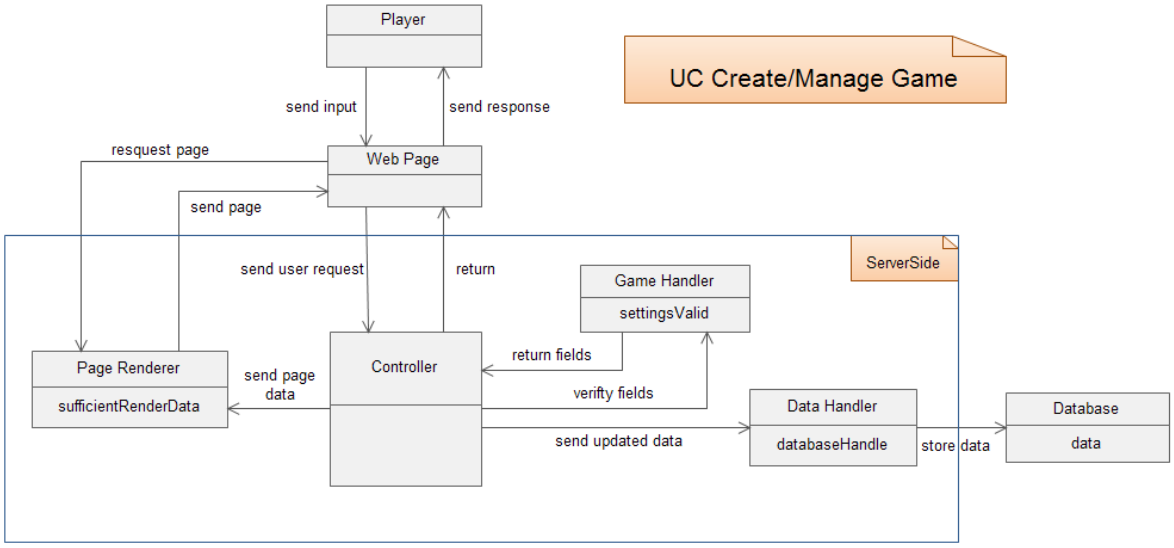


Figure 5: Create/Manage Game

Figure 5, represents the UC-7 and UC-10, the creation and management of Investment Games. The User fills in the necessary fields in order to create or change a Investment Game then the Web Page submits this info. The Controller will receive the request and call on the Game Handler to verify the validity of the fields. If there are no errors, the Controller will inform the Data Handler to store the new game or its new settings. Then (regardless of the validity of the fields), the Controller provides the necessary page data to the Page Renderer and informs the Web Page of the completion of the process. The Web Page calls for the Page Render to create the necessary page to be viewed.

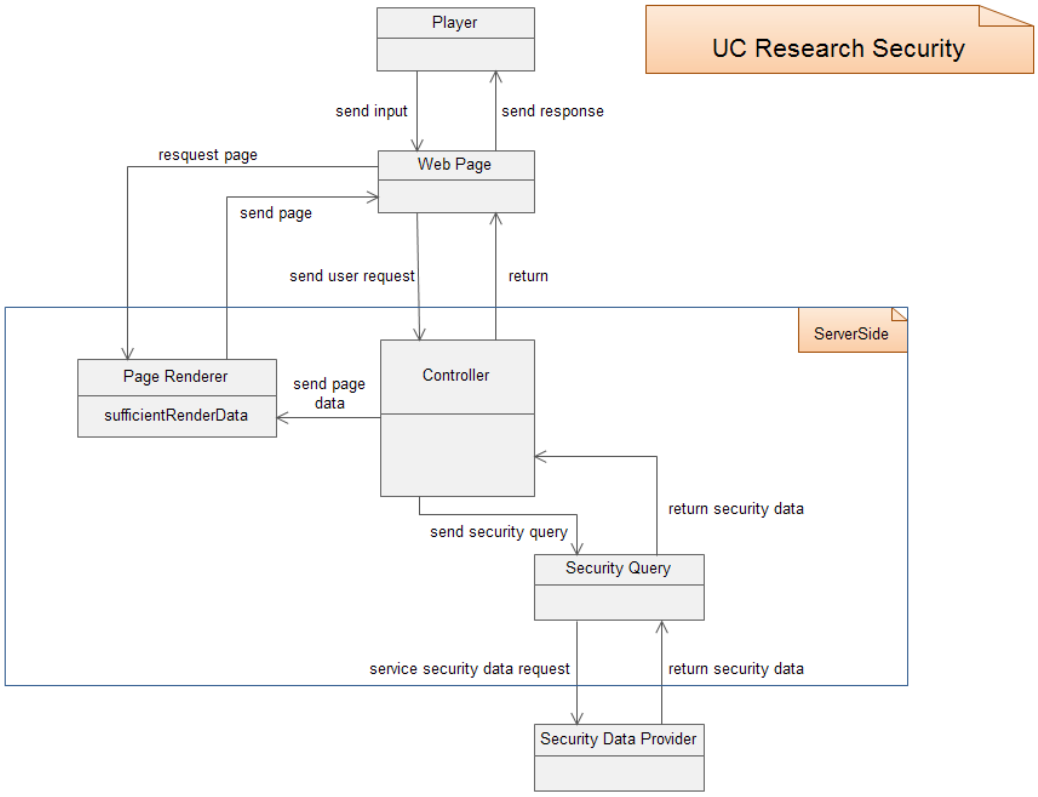


Figure 6: Research Security

Figure 7 shows the UC-3 Research Security. The security is requested through the Web Page by the User, which tells Controller to inform the Security Query to fetch the correct security data from Security Info Provider. Note that even an invalid ticker symbol will go through the same steps, the Security Query will just return N/A or 0 for all the fields. The Controller now sends the data to be rendered to the Page Renderer and then notifies the Web Page that the process is complete. The Web Page knows to request the page from the Page Renderer, which then services the request and generates the correct page to be viewed by the User.

6.1.1 Concept Definitions

"D" - *Doing* responsibilities.
"K" - *Knowing* responsibilities.

Player:
Definition: A person who uses or operates something
Responsibilities:

- Research stocks (D)
- Make requests for trades (D)
- Manage portfolio (K)
- Navigate through website (D)
- Manage Leagues (K)
- Go to FAQ page if needs help with system(D)

Web Page:
Definition: A document connected to the World Wide Web and viewable by anyone connected to the internet who has a web browser.
Responsibilities:

- Take requests from the Player (K)
- Send requests to the Controller (D)
- Send page requests to the Page Renderer (D)
- Update new webpage to be displayed when new page is rendered (K)

Page Renderer:
Definition: Page rendering is the process of generating a page from the database
Responsibilities:

- Receive the required data to generate new page (K)
- Convert the data into user-friendly format (D)
- Send rendered pages to the Web Page (D)

Controller:
Definition: Takes user request and creates a web page that is user-friendly.
Responsibilities:

- Request account creation (D)
- Receive Player requests from the Web Page (D)
- Request an order (K)
- Request stock queries (K)
- Send League requests to be validated (K)
- Inform Web Page when process is complete (D)
- Send page data to be rendered (D)

Stock Query:
Definition: Fetch Real time stock prices.
Responsibilities:

- Receive requests from the Player for stock data (K)
- Request information from Stock Info Provider (D)
- Send updated stock data to Player (D)

Validity Checker:
Definition: Routines in a data entry program that test the input is correct or not.
Responsibilities:

- Determine if sufficient amount of money is available for the transaction (K)
- Request updated stock price based on liquidity model (D)
- Request and receive portfolio data (K)
- Send queries for stock data (D)

Security Data Provider:
Definition: Manipulates the price to realistic real world price for slippage
Responsibilities:

- Determine new price (K)
- Send out updated stock information (D)
- Utilize algorithm to reflect realistic trades in the market (K)

Data Handler:
Definition: Communicates with database to service data requests
Responsibilities:

- Receive and send every kind of data used in system (D)
- Request data from Database (D)
- Send data to be stored in Database (K)

League Handler:
Definition: A Player who is allowed to create as well as participate in the Leagues.
Responsibilities:

- Receive initial or existing league requests (D)
- Determine if the requests are valid (K)
- Upon successful completion of Player's request, update database (K)
- Create a new league or let the Player participate in the other League (D)

Fund Handler:
Definition: A Player who handles his resources
Responsibilities:

- Receive requests for available money (D)
- Determine if requests are valid (K)
- Upon successful completion of Player's request, update the database (D)

6.1.2 Association Definitions

Concept Pair	Association Description	Association Name
Web Page ↔ Page Renderer	Request to visit page, sends rendered page	request page, send page
Web Page ↔ Controller	Passes the user's desired action, informs of process completion	send user request, return
Controller ↔ Page Renderer	Passes necessary data for page rendering	send page data
Controller ↔ Security Query	Asks for data on specific security, send data on specific security	send security query, return security data
Controller ↔ Validity Checker	Requests order to be carried out, passes new portfolio data	send order, send portfolio data
Controller ↔ Game Handler	Passes updated Game settings, validates updated settings	verify fields, return fields
Controller ↔ Player Database	Passes updated settings, validates updated settings	verify fields, return fields
Controller ↔ Data Handler	Passes updated data, ask for portfolio data to perform process, return altered portfolio data	send updated data, request portfolio info, return portfolio info
Security Query ↔ Security Info Provider	Asks for security data, return security data	send security data request, service security data request
Security Query ↔ Validity Checker	Asks for to query specific security, return security data	send security query, return security data
Validity Checker ↔ Data Handler	Asks for Player's portfolio information for validity purposes, passes user's portfolio information	request portfolio data, return portfolio data
Validity Checker ↔ Trade Database	Sends order information to determine adjusted price, return updated price	request adjustment, update price
Data Handler ↔ Database	Stores incoming data, request certain data, retrieve needed data	store data, request data, retrieve data

6.1.3 Attribute Definitions

Concept	**Attribute**	**Meaning**
Data Handler	databaseHandle	Interacts with the database to service data requests.
Database	Data	It includes all data used in the system, which includes League information, Player information, stock prices, fund settings, and transaction history etc.
Page Renderer	sufficientRenderData	Generates a page from database with updated information.
Trade Database	priceUpdate	Generates new price for the future orders.
Game Handler	settingsValid	Decides whether the Player's requests are valid for the given League.
Fund Handler	settingsValid	Decides whether the Player's requests are valid for the given amount of money.
Validity Checker	fieldsValid, fundsValid, tradeSuccess	Compares funds and prices to make sure a Player's request is valid.

6.1.4 Traceability Matrix

		Domain Concepts								
Use Case	**PW**	**Player**	**Webpage**	**Page Rendere**	**Controller**	**Stock Query**	**Validity Checker**	**Data Handler**	**League Handler**	**Fund Handler**
UC-01	10	X						X		
UC-02	4	X	X	X		X				
UC-03	19	X			X		X	X		X
UC-04	19	X			X		X	X		X
UC-05	5	X	X	X						
UC-06	10	X	X	X						
UC-07	13	X			X			X	X	
UC-8	9	X			X		X	X	X	
UC-9	9				X					
UC-10	5	X		X						
UC-11	9	X	X	X		X				
Max PW		19	10	10	19	9	19	19	13	19
Total PW		103	28	33	69	13	47	70	22	38

6.2 System Operation Contracts

Operation*: Register Player*

Preconditions:

- None

Postconditions:

- Player's background (name, age, etc.) information is stored in database

Operation*: Login*

Preconditions:

- Provide correct username and password

Postconditions:

- None

Operation: Access to FAQ page

Preconditions:

- Player has valid portfolio in database

Postconditions:

- None

Operation: Buying Securities

Preconditions:

- Player has enough cash available to purchase different stocks and bonds
- Enough Stocks available in the market to be purchased
- Transaction data is valid

Postconditions:

- Update database with Player's new stock holdings
- Update the Stock inventory in database

Operation: Selling Securities

Preconditions:

- Player has the enough stocks or bonds to sell
- Transaction data is valid

Postconditions:

- Update the database with Player's stock holdings
- Update the Stock inventory in database

Operation: Challenge Player

Preconditions:

- Player has valid portfolio in the database

Postconditions:

- Create a championship between Players

Operation: Query Stocks

Preconditions:

- Player has valid portfolio in the database

Postconditions:

- None

Operation: View Portfolio
Preconditions:
- Player has valid portfolio in the database

Postconditions:
- Let the Player change his portfolio and don't allow a player to make changes in another player's portfolio

Operation: Invite to League
Preconditions:
- Player has valid portfolio in database
- Player(Invitee) is already in the League

Postconditions:
- None

Operation: Create League
Preconditions:
- Player has valid portfolio in database

Postconditions:
- Database is updated with the new League information

Operation: Manage League
Preconditions:
- Player has access to the league privileges
- Player's request are valid

Postconditions:
- Upon successful completion of Player's request update database

Operation: View League Standings
Preconditions:
- There exists Leagues

Postconditions:
- None

Operation: Submit technical problems to Administrator
Preconditions:
- Player has valid portfolio in database

Postconditions:
- Send an Email to Administrator regarding the problem

7.0 Interaction Diagrams

UC-1: Register

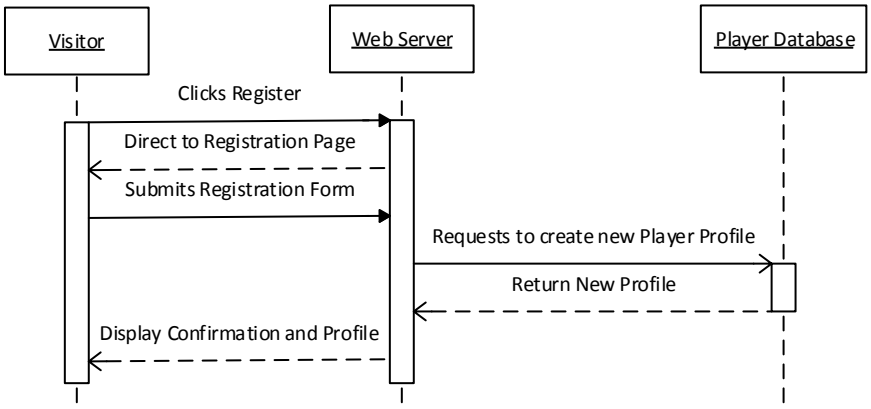
In this system sequence diagram, the visitor first navigates to the website. After reaching the website, the visitor clicks “Register”. After this, the visitor is presented with the registration page.

Once the user has submitted the registration page, the information provided is validated and is sent to the Player Database. The system then requests for a new player profile to be created for the visitor. The system then returns to the visitor that their profile creation was complete, and that they are now logged into the system.

The only alternate scenario to the main success scenario would be if any of the information entered by the user was invalid. In this situation, the system would return an error to the Visitor letting them know that there was an error in their submission. It would give the user another chance to submit the registration form.

As of now, no other implementations have been discussed, as the current one seems to be the most logical flow of events.

UC-1 System Sequence Diagram:

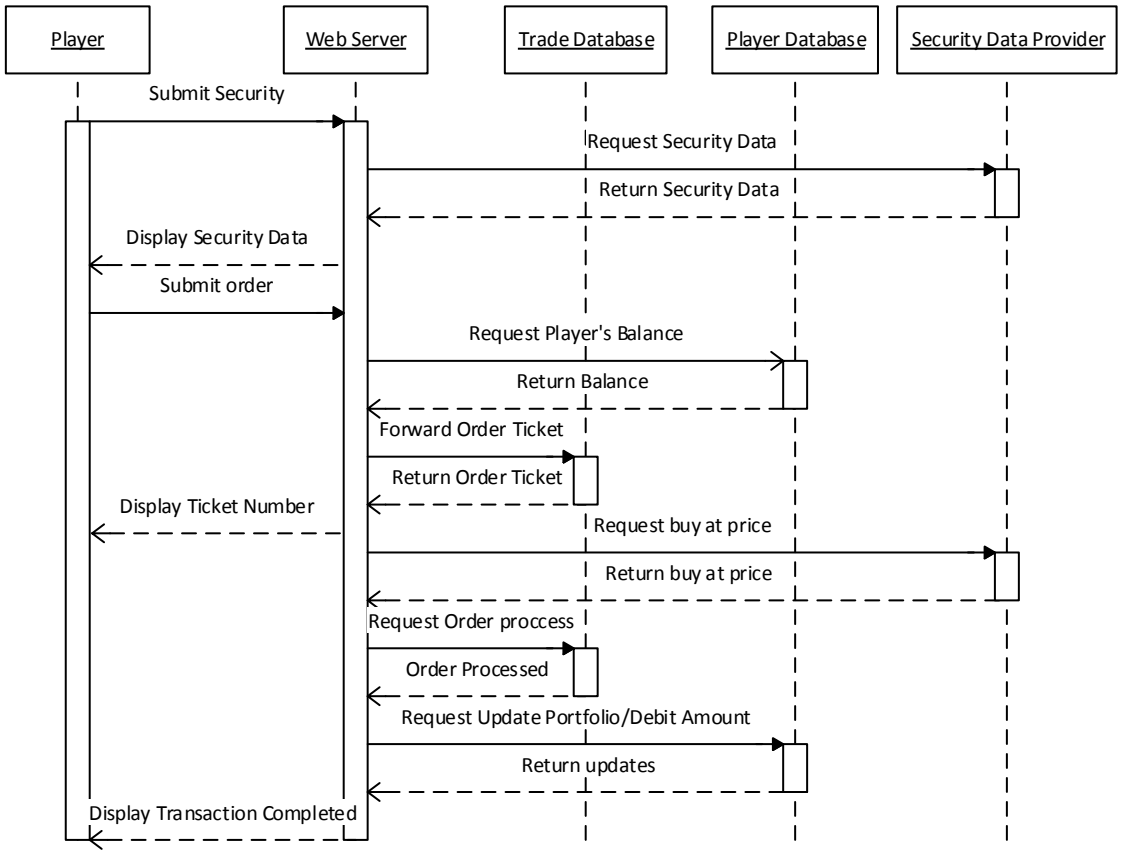


UC-3: Buy Security

In this system, the player selects a security of interest and in return the system should display the latest data of the security. This is done when the web server connects to the security data provider to read the data. The data is then displayed to the player. Then, the player must fill out order form which most importantly includes the buying price of the security. Upon clicking submit, the web server verifies if the player has enough balance to purchase. If the player has enough balance then the system requests an order ticket from the trade database for the particular security (securities). The player is displayed with a confirmation of order and order ticket number. The Web Server constantly reads the data of the particular security through the security data provider. Once, the parameters of the player match the current data, the system requests the trade database to process the order. Order is then processed and player's portfolio is updated. A notification is also sent to the player informing that the transaction is completed.

The above procedure was chosen for this process because it allows for loose coupling between the function, the ticket system, and the security data provider. Additionally, this method aligns with the Expert Doer principle. The principal allows there to be a specialized function that provides the ticket system and the security data. With this style the components work together very easily, yet work together just as well on their own.

UC-3 System Sequence Diagram:

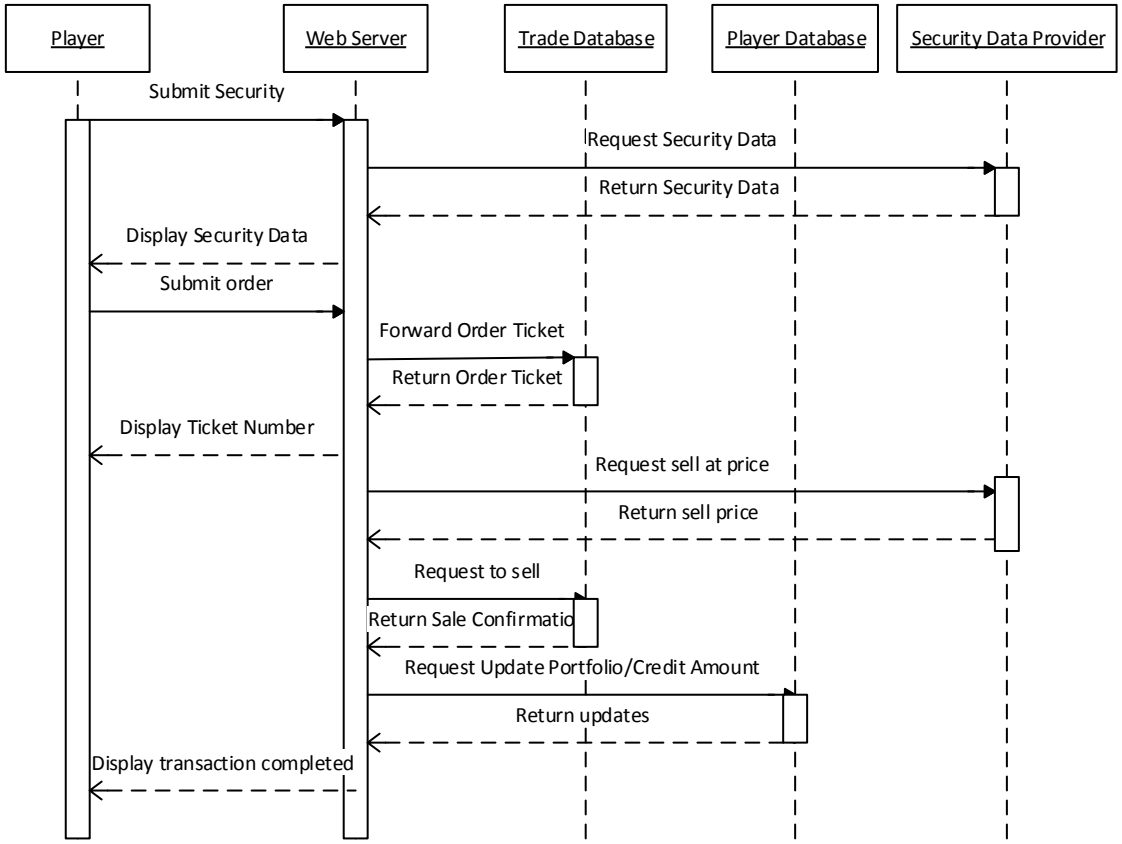


UC-4: Sell Security

In this system, the player selects a security of interest and in return the system should display the latest data of the security. This is done when the web server connects to the security data provider to read the data. The data is displayed to the player. Then, the player must fill out order form which most importantly includes the selling price of the security. A request for generating an order ticket is then sent out to the trade database. Once the order ticket has been generated the player is displayed confirmation of the order and the ticket number. The system constantly reads the data through the security data provider and once the price to sell his matched with the user's parameters the order is sent to be processed to the trade database.
Order is then processed and player's portfolio is updated. A notification is also sent to the player informing that the transaction is completed.

The above procedure was chosen for this process because it allows for loose coupling between the function, the ticket system, and the security data provider. Additionally, this method aligns with the Expert Doer principle. The principal allows there to be a specialized function that provides the ticket system and the security data. With this style the components work together very easily, yet work together just as well on their own.

UC-4 System Sequence Diagram:

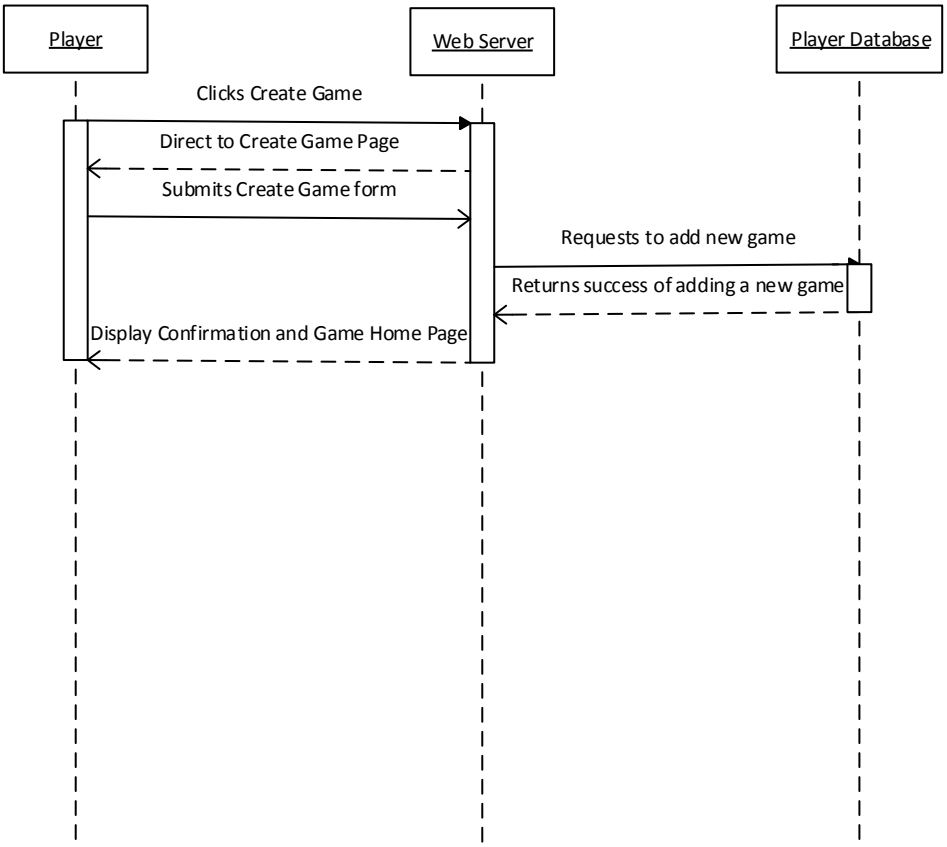


UC-7: Create Game

In this system sequence diagram, the Player requests for a new game to be created. The system then presents the user with the Create Game page. One the user has submitted the registration page, the information is validated and a request to create a new game is sent to the Player Database. The system then updated required fields in the Player Database and signals a success to the Web Server. The Web Server signals this back to the user with a confirmation that their game has successfully been create.

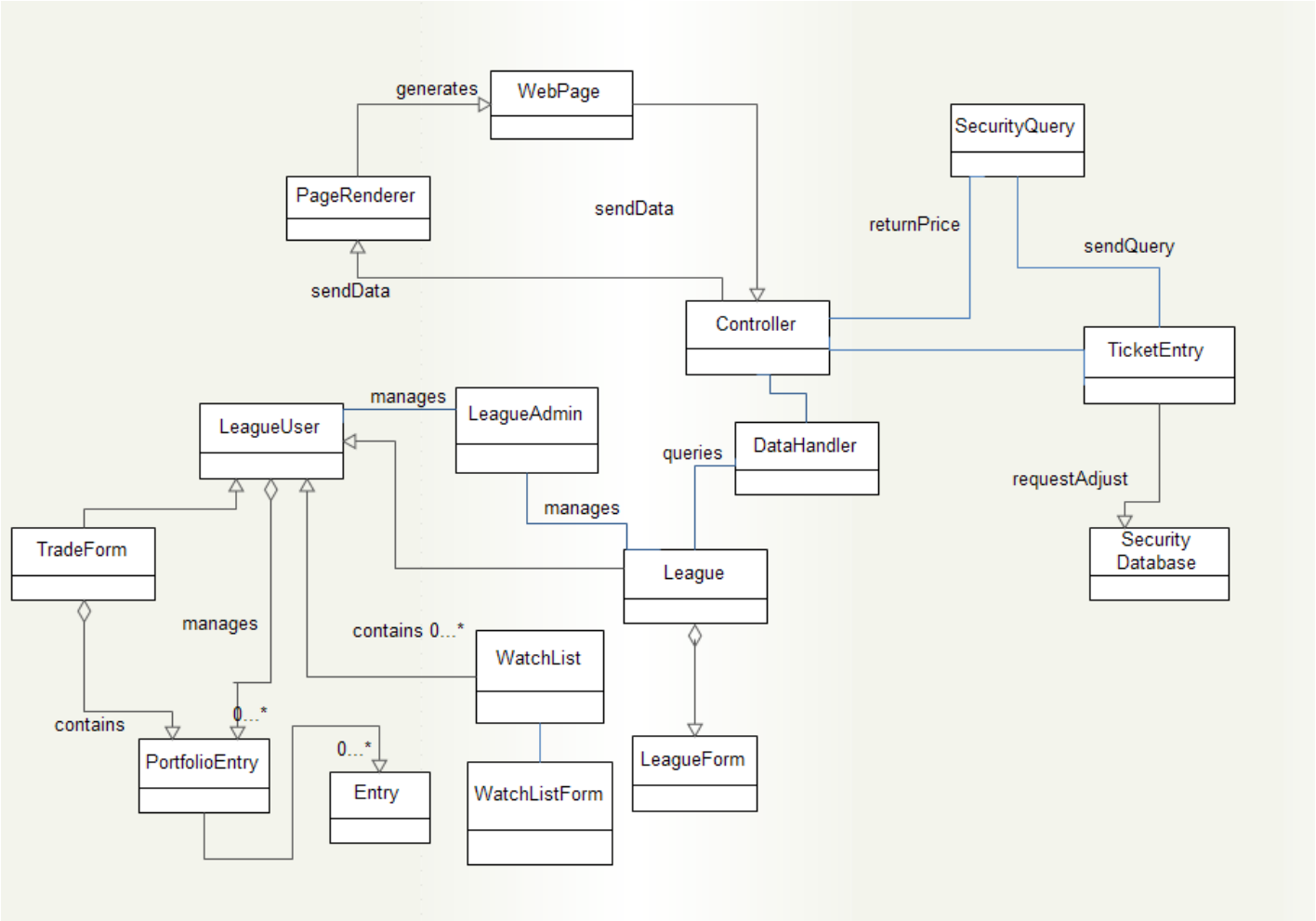
The only alternate scenario to the main success scenario would be if the game name the Player is trying to make is already taken. In this situation, the system would return an error after form submission letting the user know that their game name is already taken. It would ask the Player to choose another game name and go through the same process of revalidating.
As of now, no other implementations have been discussed, as the current one seems to be the most logical flow of events.

UC-7 System Sequence Diagram:



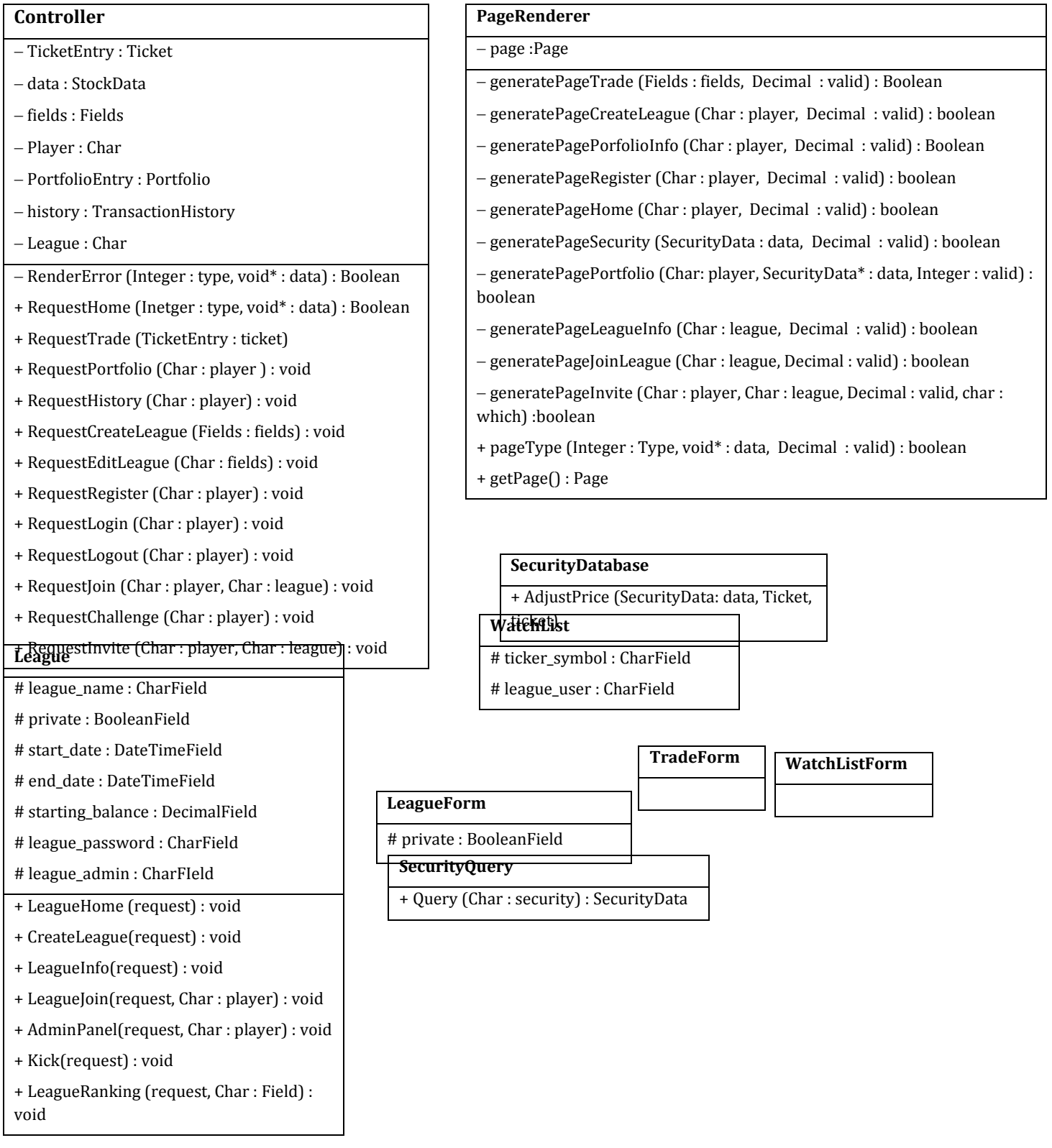
8.0 Class Diagram & Interface Specification

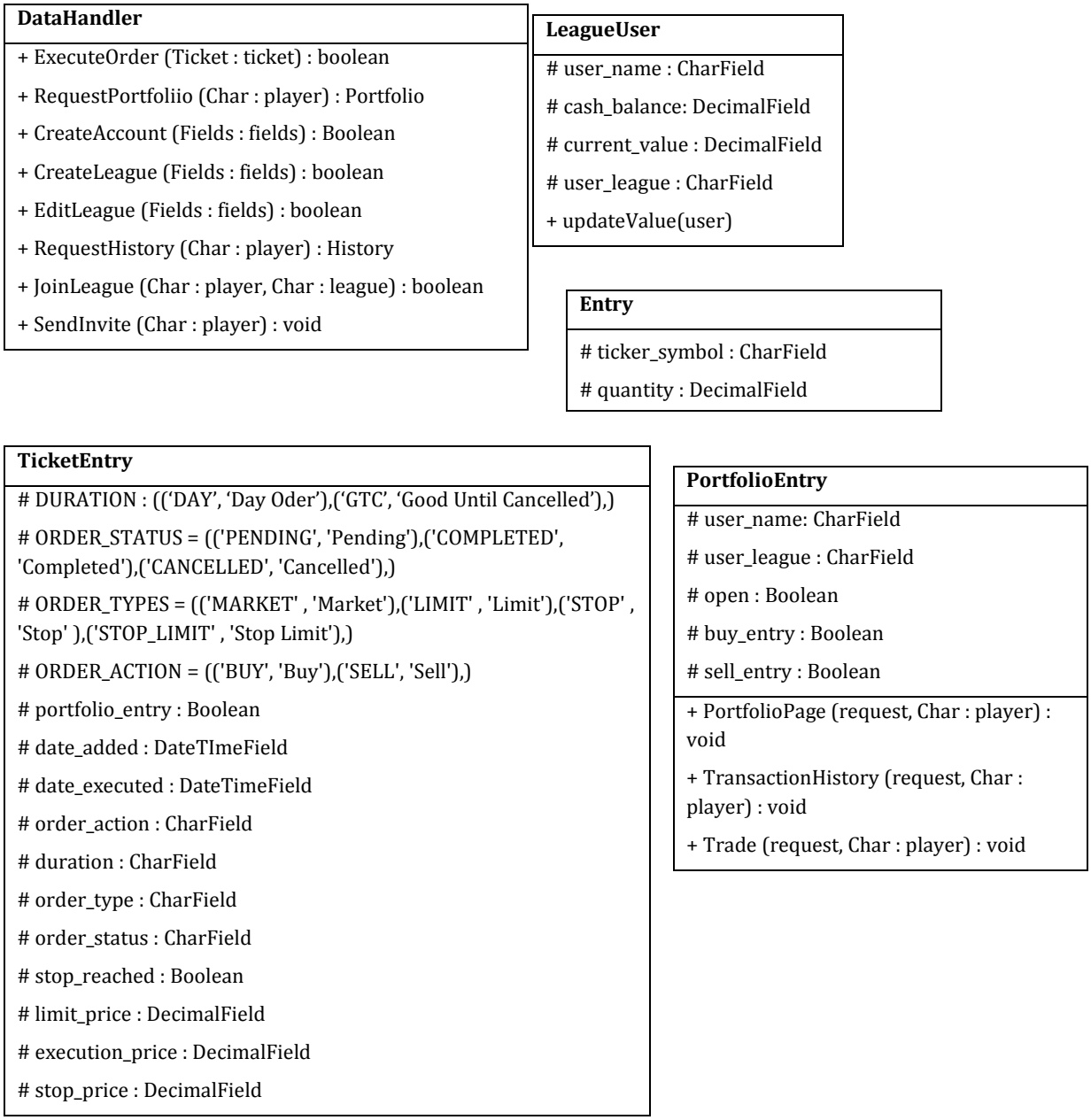
8.1 Class Diagram



A summary of important classes, their variables, and functions are shown in the *Data Types & Operation Signatures*.

8.2 Data Types & Operation Signatures




DataHandler
+ ExecuteOrder (Ticket : ticket) : boolean
+ RequestPortfoliio (Char : player) : Portfolio
+ CreateAccount (Fields : fields) : Boolean
+ CreateLeague (Fields : fields) : boolean
+ EditLeague (Fields : fields) : boolean
+ RequestHistory (Char : player) : History
+ JoinLeague (Char : player, Char : league) : boolean
+ SendInvite (Char : player) : void
LeagueUser
user_name : CharField
cash_balance: DecimalField
current_value : DecimalField
user_league : CharField
+ updateValue(user)
Entry
ticker_symbol : CharField
quantity : DecimalField
TicketEntry
DURATION : (('DAY', 'Day Oder'),('GTC', 'Good Until Cancelled'),)
ORDER_STATUS = (('PENDING', 'Pending'),('COMPLETED', 'Completed'),('CANCELLED', 'Cancelled'),)
ORDER_TYPES = (('MARKET' , 'Market'),('LIMIT' , 'Limit'),('STOP' , 'Stop'),('STOP_LIMIT' , 'Stop Limit'),)
ORDER_ACTION = (('BUY', 'Buy'),('SELL', 'Sell'),)
portfolio_entry : Boolean
date_added : DateTImeField
date_executed : DateTimeField
order_action : CharField
duration : CharField
order_type : CharField
order_status : CharField
stop_reached : Boolean
limit_price : DecimalField
execution_price : DecimalField
stop_price : DecimalField
PortfolioEntry
user_name: CharField
user_league : CharField
open : Boolean
buy_entry : Boolean
sell_entry : Boolean
+ PortfolioPage (request, Char : player) : void
+ TransactionHistory (request, Char : player) : void
+ Trade (request, Char : player) : void

1. Controller

Attributes

The controller plays the role of a town center, conveying messages back and forth between different domain concepts in the domain model. The role of controller is best accomplished, if the controller has a copy of all data that it handles as an attribute. Doing so will lower the chance of corrupting data.

– TicketEntry : Ticket

This is a copy of the order ticket that the player has just submitted.

– data : SecurityData

This is a copy of the data that the system queries from the Security Info Provider.

– fields : Fields

This is a copy of the fields that a Player or League Coordinator fills out during a creation / editing request.

– player : Char

This a copy of the player's username that the controller passes along to the data handler. It is used to find the Player object from inside the database.

– portfolio : Portfolio

This a copy of a Portfolio object that the controller passes along.

– history : TransactionHistory

This is a copy of a TransactionHistory object that the controller passes along.

– League : Char

This is a copy of the name of the league that the controller passes along

Methods

The controller has many methods which the web page calls in order to let the controller know that it has a request (all except for Render and RenderError). The controller will subsequently convey the message by calling another function.

– RenderHome (Integer : type, void* : data) : Boolean

The controller calls this method when it is ready to render a page which is by default home page. The arguments Integer represents the type of page that is displayed, and the pointer, points to a data structure containing the data necessary to construct the page.

– RenderError (Integer : type, void* : data) : Boolean

The purpose of this method is same as Render method, but instead of rendering the correct page it renders an error version of the page.

+ RequestTrade (TicketEntry : ticket)

Method used to request a buy/sell security.

+ RequestPortfolio (Char : player) : void

Method used to view a portfolio.

+ RequestCreateLeague (Char : fields) : void

Method used to create a league.

+ RequestEditLeague (Char : fields) : void

Method used to edit league settings.

+ RequestHistory (Char : player) : void

Method used to obtain/view transaction history.

+ RequestJoin (Char : player, Char : league) : void

Method used to request, join a league/game.

+ RequestChallenge (Char : player) : void

Method used to challenge another player.

+ RequestRegister (Char : player) : void

Method used to register a player.

+ RequestLogin (Char : player) : void

Method used to login a player.

+ RequestLogout (Char : player) : void

Method used to logout a player.

+ RequestFAQPage () : void

Method used to logout a player.

+ RequestAboutUs () : void

Method used to logout a player.

2. PageRenderer

Attributes

– page :Page

This is the current page that the web browser is displaying/will be displayed.

Methods

The valid parameter lets the page rendered know if the page that it should be generating is a success page or an error page.

– generatePageTrade (Fields : fields, Decimal : valid) : boolean

Method called to render a page displaying the results of an order.

– generateCreateLeague (Char : player, Decimal : valid) : boolean

Method called to render a page displaying the form to create a league.

– generatePagePortfolioInfo (Char : player, SecurityData* : data, Decimal : valid) : boolean

Method called to render a page displaying the results of a portfolio viewing.

– generatePageLeagueInfo (Char : league, Decimal : valid) : boolean

Method called to render a page displaying the results of a league information viewing.

– generatePageRegister (Char : player, Decimal : valid) : boolean

Method called to render a page displaying the form to create an account.

– generatePageHome (Char : player, Decimal : valid) : boolean

Method called to render a page displaying the results of a creation of a league or registration.

– generatePageJoinLeague (Char : league, Decimal : valid) : boolean

Method called to render a page displaying the results of joining of a league.

+ pageType (Integer : Type, void* : data, Integer : valid) : boolean

Method called by the controller in order to render a page with the given type, data, and whether it is an error or not.

+ getPage() : Page

Method called by the web page in order to retrieve the page it must display.

+ GenerateFAQPage () : void

Method called by the web page in order to retrieve the Frequently Asked Answers & Questions.

+ GenerateAboutUs () : void

Method called by the web page in order to retrieve the About Us page.

3. DataHandler

Methods

Methods called by the controller to access the information in the database.

+ ExecuteOrder (Ticket : ticket) : boolean

Method executes the ticket order by updating the player's portfolio accordingly.

+ RequestPortfoliio (Char : player) : Portfolio

Method called to request the portfolio data from the database.

+ CreateAccount (Fields : fields) : Boolean

Method called to request creation of new account and data to be stored in the database.

+ CreateLeague (Fields : fields) : boolean

Method called to request a new league creation and data to be stored in the database.

+ EditLeague (Fields : fields) : boolean

Method called to request the league settings be modified in the database.

+ EditFund (Fields : fields) : boolean

Method called to request the fund settings be modified in the database.

+ RequestHistory (Char : player) : History

Method called to request the transaction history from the database.

+ JoinLeague (Char : player, Char : league) : boolean

Method called to request that a player be added to a league in the database.

4. SecurityQuery

Methods

+ Query (Char : security) : SecurityData

Method called to request security data from the Security Info Provider. The data is forwarded straight to the class requesting it, and a copy is made within the Security Query.

5. SecurityDatabase

Methods

+ AdjustPrice (SecurityData : data, Ticket : ticket) : Decimal

Method called by the Validity Checker to modify the security price per share in accordance to how many the player plans to buy or sell.

6. WebPage

The web page contains a copy of various attributes that it receives from the player and forwards it on to the controller.

Attributes

– ticket : Ticket

This is a copy of an order ticket that the player fills out.

– fields : Fields

This is a copy of the league or fund settings the player fills out.

– playerinfo : PlayerInfo

This is a copy of the player info that the database provides.

– player : Char

This is a copy of the player's username.

– security : Char

This is a copy of the particular security that is requested by the player.

– league : Char

This is a copy of the league name that the player enters.

7. League

This class contains attributes and methods related to each league and used to generate the league pages.

Attributes

league_name : CharField

This is the league name.

private : BooleanField

This is the test is league is private.

start_date : DateTimeField

This is the league start date.

end_date : DateTimeField

This is the league end date.

starting_balance : DecimalField

This is the league starting balance for a particular league player.

league_password : CharField

This is the league password if league is private.

league_admin : CharFIeld

This is the league's admin who created the league.

Methods

+ LeagueHome (request) : void

This method lists all the leagues that are currently active.

+ CreateLeague(request) : void

This method is to create a league.

+ LeagueInfo(request) : void

This method is used to obtain the league info about a particular league including players in the league, start date, end date, etc.

+ LeagueJoin(request, Char : player) : void

This method is used to join a league.

+ AdminPanel(request, Char : player) : void

This method is used to access the league admin panel.

+ Kick(request) : void

This method is used to remove a player from a league by the league admin.

+ LeagueRanking (request, Char : Field) : void

This method is used to rank the player's in each league.

8. TicketEntry

This class contains attributes and methods related to completing a trade form and execute a trade. It also contains the different types of trades that could be handled.

Attributes

DURATION : ((‘DAY’, ‘Day Oder’),(‘GTC’, ‘Good Until Cancelled’),)

These are the durations of a ticket order.

ORDER_STATUS = (('PENDING', 'Pending'),('COMPLETED', 'Completed'),('CANCELLED', 'Cancelled'),)

These are the order status after a ticket order is submitted.

ORDER_TYPES = (('MARKET' , 'Market'),('LIMIT' , 'Limit'),('STOP' , 'Stop'),('STOP_LIMIT' , 'Stop Limit'),)

These are the different order types.

ORDER_ACTION = (('BUY', 'Buy'),('SELL', 'Sell'),)

These are the transaction of type that was placed buy/sell.

portfolio_entry : Boolean

This a valid bit: it lets the controller know if the ticket is a buy entry or sell entry.

date_executed : DateTimeField

This is the time and date of the ticket was successfully executed.

date_added : DateTimeField

This is the time and date of the ticket was submitted.

order_type : CharField

This is the type of transaction (example being stop order).

order_action : CharField

This is the type of order that was placed buy/sell.

order_type : CharField

This is for the type of order that is being placed. For example, Market Order, Limit Order, or Stop Order.

order_status : CharField

This is the to update the status, displays status of a transaction.

stop_reached : Boolean

This is the threshold change in price of the stock before the order is executed.

duration : CharField

This is the time; the trade took place, ex: day.

limit_price : DecimalField

This is the threshold price for a stock before the order is executed.

execution_price : DecimalField

The is the price the trade actually gets executed at.

stop_price : DecimalField

This is the threshold percent change of the stock before the order is executed.

9. PortfolioEntry

This class contains attributes and methods related to the creating a portfolio entry for each player and save the data related to each trade.

Attributes

user_name : CharField

This is the name of the player associated with a trade entry.

user_league : CharField

This is the name of the league the player associated with a trade entry.

open :Boolean

This is a Boolean field that checks if the trade is related to buy entry or sell entry.

buy_entry : Boolean

This is a Boolean field that checks if it is a buy entry.

sell_entry : Boolean

This is a Boolean filed that checks if it is a sell entry.

Methods

+ PortfolioPage (request, Char : player) : void

This displays the list of leagues the player is associated with.

+ TransactionHistory (request, Char : player) : void

This to the method used to get the transactions history and watch list of a player associated in a particular league.

+ Trade (request, Char : player) : void

This is method creates the portfolio entry for a particular trade executed by a player.

10. LeagueAdmin

The league coordinator does not have any special attributes or methods that make it not different from a player.

11. LeagueUser

This class contains attributes and methods related to the player in each league and used to generate the league pages.

Attributes

user_name : CharField

This is the username of the player.

cash_balance : DecimalField

This is the player's cash balance.

current_value : DecimalField

This is the player's current value in a portfolio.

user_league: CharField

This is the league the player is associated with.

Methods

+ updateValue()

This is a call to the API to update the current value of portfolio.

12. Entry

This is has the required attributes for a trade entry.
Attributes

ticker_symbol : CharField

This is the ticker symbol that is being matched in the API from the trade form.

quantity : DecimalField

This is the quantity that is requested to be traded in the trade form.

13. TradeForm

This class generates the trade form to be displayed when presented to the trade window.

14. WatchListForm

This class generates the watch list form to be displayed when presented to the portfolio detail page.

15. WatchList

This class generates the watch list to be displayed when presented to the portfolio detail page.
Attributes

ticker_symbol : CharField

This is the ticker symbol that is being matched in the API from the trade form.

league_user : CharField

This is the player that is associated with the watch list.

16. LeagueForm

This is to generate and create a create a league form.

Attributes

private : BooleanField

This is the checks if the league is private.

8.3 Traceability Matrix

Class / Domain Concept	WebPage	PageRender	ValidityChecker	StockQuery	DataHandler	GameHandler	TradeDatabase	FundtHanler
WebPage	X							
PageRender		X						
Controller	X	X						
TicketEntry			X					x
SecurityQuery				X				
SecurityDatabase							X	
DataHandler					X			
League					X	x		
LeagueUser					X			
PortfolioEntry					X			x
TradeForm					X			
WatchListForm					X			
WatchList					X			
LeagueForm					X			
LeagueAdmin						x		
Entry					X			

Many of the classes map back to the DataHandler concept they contain data that is queried by the DataHandler. The domain model represented the all the classes in single entity, but now they are show as separate entities in the class diagram. The class diagram gives more insight on the inner workings and details of our program.

8.3.1 Object Constrain Language (OCL)

context Controller::RequestPortfolio(string : investor) void

pre: (investor → InvestorAccount.portfolio = this.portfolio)

- You can only view your own portfolios

context Controller::RequestEditLeague(Fields : fields) void

pre: (InvestorAccount → LeagueCoordinator = true)

- You can only edit a league if you are the league coordinator

context DataHandler::ExecuteOrder(Ticket : ticket) : Boolean

pre: (ValidateSell())

post: (InvestorAccount.Update())

- The Investor's portfolio must be updated to accommodate bought/sold stocks

context DataHandler::CreateAccount(PlayerInfo : playerinfo) : Boolean

post: (hasPortfolio = true AND inGlobalLeague = 1)

- The investor will have a portfolio for the Global Public League upon registration

context DataHandler::CreateLeague(Fields : _elds) : Boolean

post: (league → name = field:League Name AND league ! this.member AND update())

- The league will be stored in our database (update), and the league coordinator will have a portfolio for that league.

context DataHandler::EditLeague(Fields : fields) : Boolean

post: (league → settings.update(fields))

- League settings will be updated in the database

context DataHandler::JoinLeague(String : investor, String : league) : Boolean

post: (league → this.member AND update())

- The User will now have a portfolio for the league

context ValidityChecker inv:

if(League)

self.balance ≥ 0

- The User will not have a negative balance

context Ticket inv:

self.numstock > 0

-A ticket can only exist for at least one share of a stock, as orders must include at least one share

context Ticket inv:

pricepershare > 0

- The price of a share is always greater than zero

context Shares inv:

pricepaid > 0

- The price of a share is always greater than zero

context Shares inv:

lasttrade!pricepaid > 0

- The price of a share is always greater than zero

context Shares inv:

changepercent ≥ -100

- Value of a stock can never go below zero, so the percent change will never be less than -100%

context Shares inv:

quantity > 0

- If there were no shares of the stock, it would not be kept track of context Shares inv:
totalgainpercent ≥ -100

- Value of a stock can never go below zero, so the percent change will never be less than -100%

9.0 System Architecture & System Design

9.1 Architectural Styles

Money machine utilizes architectural styles with a main focus on Model/View/Controller approach. In this part, we will take closer look into how Money machine incorporates these techniques in its implementation.

9.1.1 Model/View/Controller

Money Machine relies heavily on the Model/View/Controller architecture. The main view is the web interface that the user interacts with. Through this interface the user carries out various tasks as enumerated by our Use Cases. Various controllers will help the user interface with the main models of the software which are the league, the portfolio, and the ticketing system. The view will be represented by HTML, CSS, and Javascript. The controller logic will be implemented using Python. For the models, the site database will be created using Django (Sqlite abstraction) and the stock model will be made accessible by API calls to an external stock information provider. Most of our concepts fall into the controller category.

9.1.2 Front and Back Ends

The front end component mainly involves Web UI, which will be mainly seen by public. The back end consists of all the behind the scenes business logic for our application. For example, the controller to communicate with the database, controller must go through DataHandler. So, in that situation, we can conclude that DataHandler is working as front end of the database to controller.

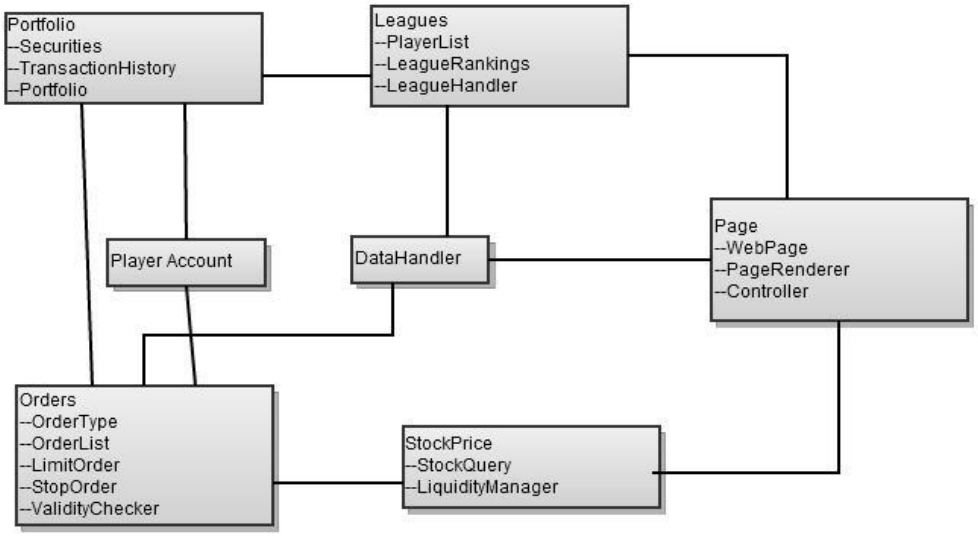
9.1.3 Event-driven Architecture

A Player acts as an event generator. A Player can buy securities, sell securities, create new leagues, participate in different leagues, and manage portfolios. Another subtle event-driven situation would be when the stock prices change throughout the day which changes the value of the Player's portfolio which also change their rank within the league. In addition, there is a FAQ page where Players can troubleshoot problems.

9.1.4 Object-oriented

In object-oriented design, 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. These objects are most important things in our project.

9.2 Identifying Subsystems



Page: (WebPage, PageRenderer, Controller)

Page is responsible for taking Player's requests and executing or transferring those requests to other subsystems. Page is subsystem which has broad relationship with the Player. Page will act as the middle-man to handle all queries and then pass them along to the appropriate subsystem which will render the requested page.

League: (PlayerList, LeagueHandler,LeagueRankings)

League is subsystem that is responsible for creating as well as updating all the objects that are associated with the League such as, League information, League Players, and Player rankings.

Portfolio: (Portfolio, TransactionHistory, Securities)

This subsystem is responsible for keeping track of Player's portfolio, securities, account balance and their past transactions.

Orders: (OrderType, OrderList, LimitOrder, StopOrder, ValidityChecker)

Orders keep track of the all the orders that have been placed in the past. This is also incorporated with the TransactionHistory of the Profolio subsystem. In addition, it lets the Player stop the order when the Player does not want to sell or buy Securities which he planned to buy for that price. In addition, this subsystem is responsible for validating the Player's buying securities request based upon available balance in Player's account.

StockPrice: (StockQuery, LiquidityManager)

This subsystem is mainly responsible for getting the updated stock prices and alter them based on liquidity. This system fetches information using the API for the stock information.

9.3 Mapping Subsystems to Hardware

The system is purely software based where the processes are first initiated by the Web Browser when the user requests an action to occur. The DataHandler, Controller, Stock Query, and Page Renderer will all be managed by the back-end server processes. The software is very flexible with very minimum hardware requirements. The more Players there the bigger the database needs to be which would mean that there would need to be a large amount of easily accessibly storage space. All the data inquires will be handled by the server.

9.4 Persistent Data Storage

Our database will store user's name, user's account balance, current stock prices and history of user's past transactions. For current stocks, the database will save current stock name, stock's ticker symbol, price and available quantity, price, date and time of the transaction. The system will be able to calculate Player's net worth, his stock holdings, his account balance, his standing in league, and his past transactions. In addition, the system will also suggest Player some securities based on his stock holdings. The database will store the information about the past transactions and different types of stocks sold as well as bought.

Name: Ivan Marsic

CashBalance: $26,615.00

Market Value: $73385.00

Stocks

Symbol	Qty	Price Paid	Date Bought	Current Price
GOOG	100	652.55	11/14/12	806.19
YHOO	100	19.35	12/20/12	21.94
XOM	70	88.50	2/27/13	89.23

Transaction History

Symbol	Transaction Type	Price	Quantity	Date
YHOO	Buy	19.35	100	12/20/12
XOM	Buy	88.50	70	2/27/13
F	Sell	34.83	200	2 /24/ 12
GOOG	Buy	652.55	100	11/14/12

9.5 Network Protocol

Money Machine will communicate with our application via HTTP. If the user is at our website, then they will be able to get the latest updates about the United States stock markets. In addition, if the user scrolls down on the home page, then they will be able to get the latest news related to stock markets. If the user are registered with our system then they can log in and then will be directed to their game, and if the user are not registered with our system then user will be asked to register with their background information. The system will validate the log in information and upon successful completion, login cookie stored on the user's browser which authenticates the user to experience game.

9.6 Global Control Flow

Our system is an event driven system in which it will wait for certain actions and then responds accordingly. A Player's portfolio will be updated every time webpage is updated. The database will be updated every time it will receive request for StockInfoProvider and then Player's portfolio will be updated. This process is similar for League updates. If a Player wants know about the securities then StockInfoProvider will be requested and then up to date information will be sent based on Player's request. Most of the events in our system are related to each other. Players' request are executed in the order they were received, which is like putting them in queue, and when the orders are executed then they will be removed from queue.

9.7 Hardware Requirements

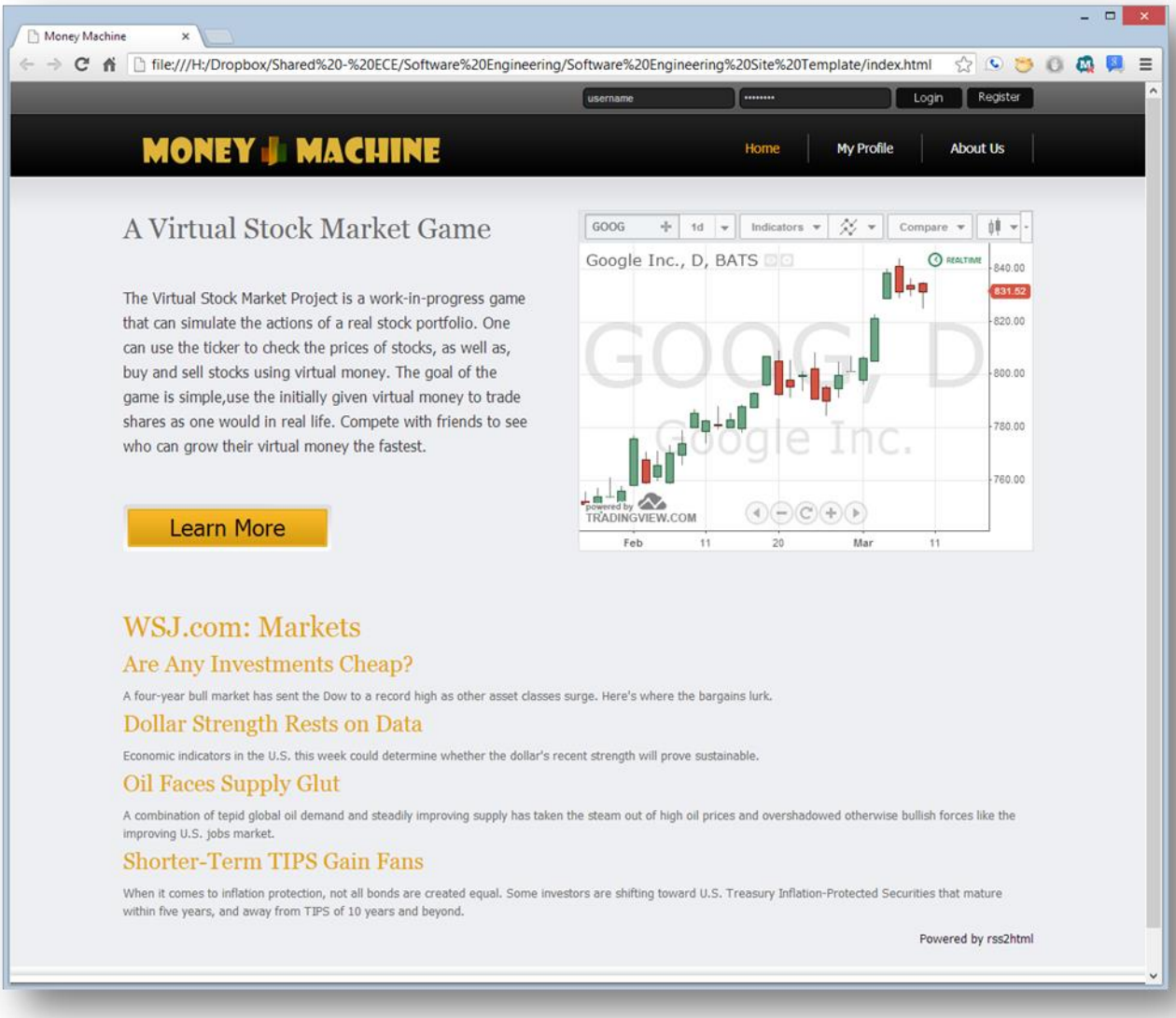
Our system will require only Internet connection and web browsers from our users. Our system will run on any web browser. Our system won't require any hardware space for this application. It will save all the information on our 'MoneyMachine' servers. Using their preferred browsers, without installing any software, Users easily connecting to their Internet, and enjoy and experience real life Stocks, and It will be an amazing experience for our users.

10.0 Algorithms & Data Structures

Our project does not have any true algorithms or data structures. We have chosen to forgo this section. Points for this section have been re-allocated into the User Interface Design & Implementation section as required. Please see the breakdown of responsibility to note the changes.

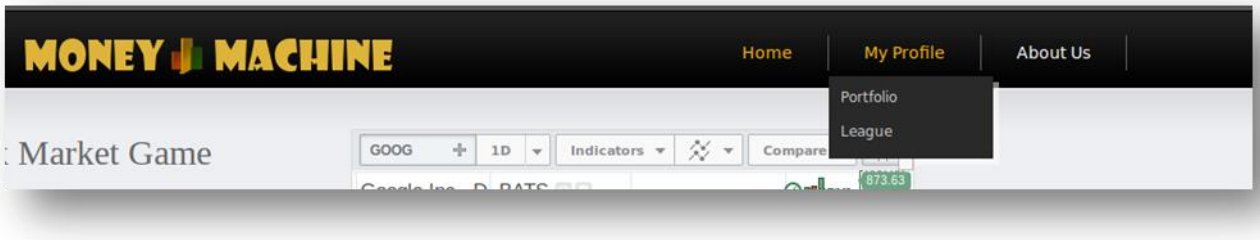
11.0 User Interface Design & Specification

11.1 Home Page



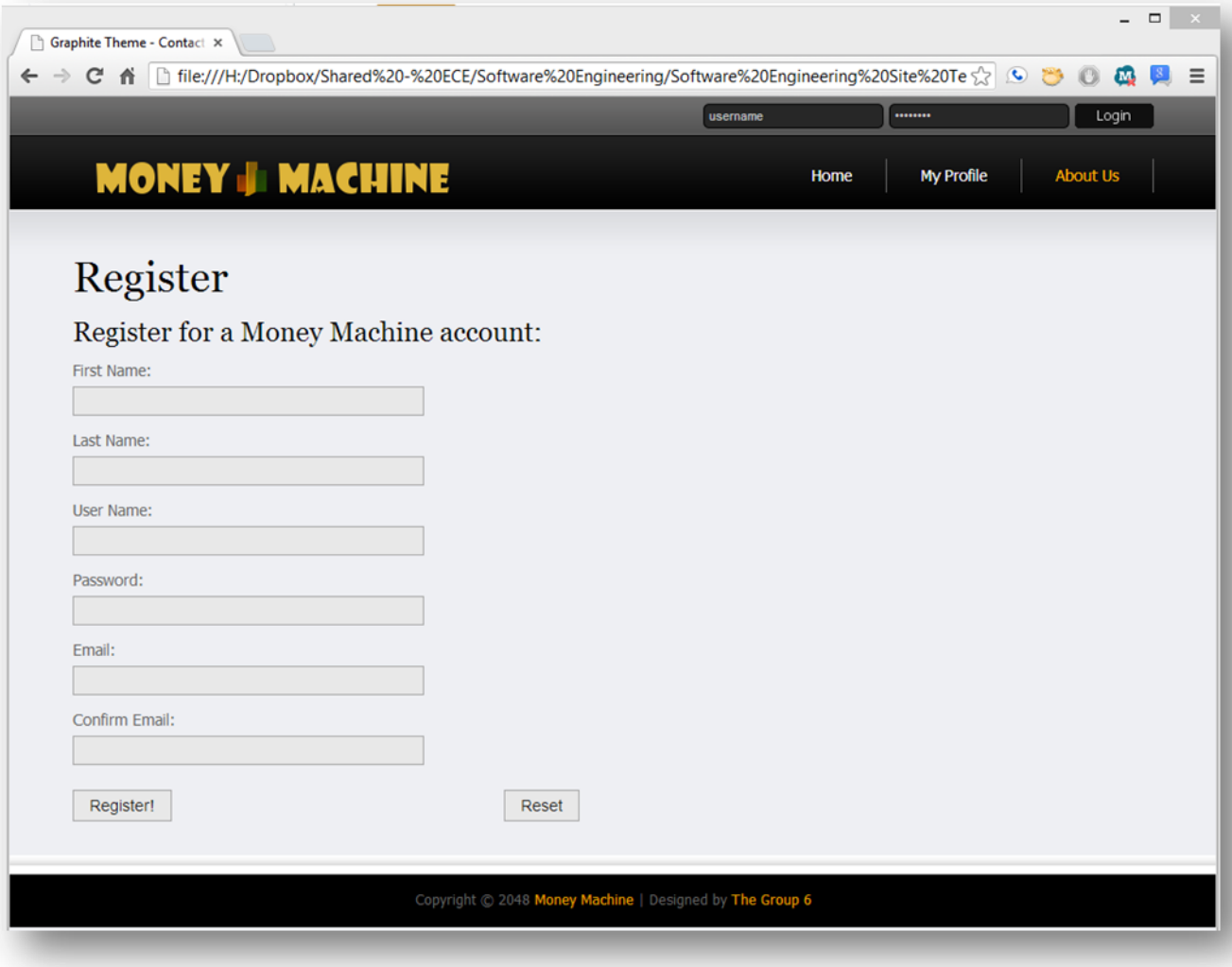
This is the home page of Money Machine, this is the first page that will be loaded when a user visits the site at first. The new UI offers a much cleaner look and provides quick access to stock ticker information directly from the home page which the previous UI design was not meant to do. The buttons have changed as well providing a more intuitive design. The original mockup of the home page was used as a skeleton for the content that was going to be displayed on the home page, major change in this UI is the layout of the content. Originally the News feeds were on the left half of the webpage while the stock prices and world markets information was on the right half of the page, but this has been changed with news feeds being on the bottom half of the page while displaying the welcome message and the stock information on the top half of the page.

11.2 Header Layout



The header has been changed which provides a log in system directly accessible within any page on the website given that the user is not currently logged in. Next change is the drop-down menu which has the options: Portfolio and League. This has been reduced from the previous design of including the *'Trade'* tab and *'Player Stats'* tab. This interface is much cleaner and simpler. The drop-down menu appears whenever the cursor is hovered over the "My Profile" button. This is different than the originally planned mockup of having just a single "My Profile" page which just had the tabbed panels that showed the 4 information tabs. The amount of clicks necessary with the new UI is the same as the previous UI; the user still has to click on 1 of the 4 options within the menu to perform the desired task. To access the subpages within the "My Profile" page it will only take 1 click from anywhere on the "Home" page or the "About Us" page. The "Register" button has also been added next to the "Login" button which has reduced the number of clicks required for the user to access the registration page from 2 clicks to 1 click.

11.3 Registration Page

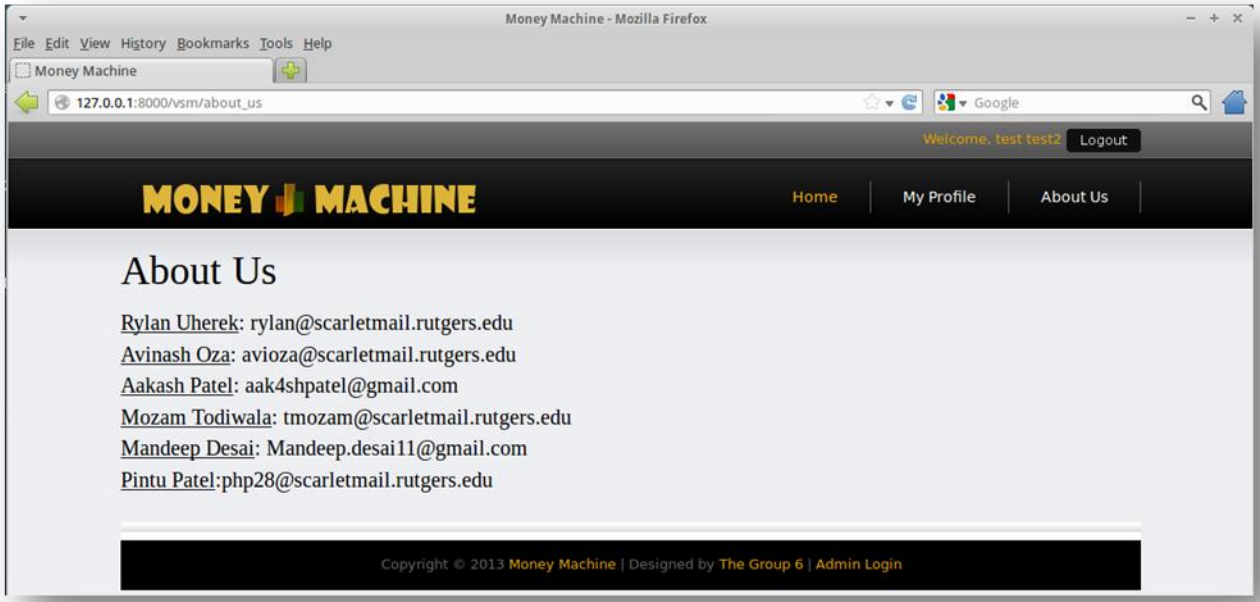


The registration page has been revised that provides more fields of input. The user has to enter an extra field, "User Name" and the "Confirm Password" from the previous UI Design has been changed to "Confirm Email" instead. This change has been made because the user can type the wrong password at first which can be recoverable via email, but if the user enters the wrong email address then that account can potentially belong to someone else or the user unable to access their account. So it is very important that the user enters the correct email address and have it be verified. A "Reset" button has been implemented should the user choose not to register for Money Machine. Once the user has registered he/she will be brought to the "My Profile" page which will implement Use Case 2, Use case 5 and Use Case 16.

HOW TO ACCESS THIS PAGE:

1) From the **Home** page, click on the **Register** button.

11.4 About Us Page



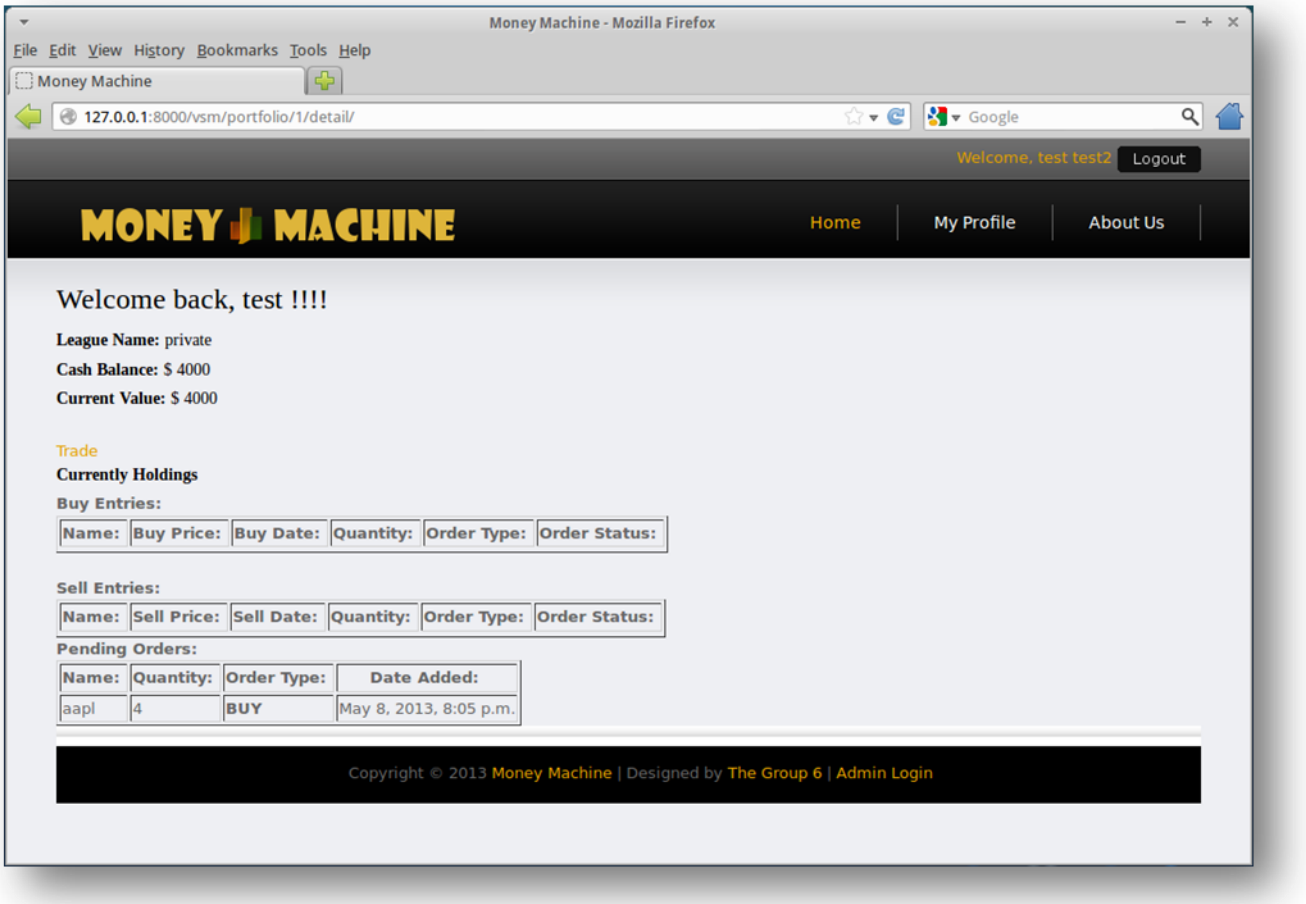
The “Help” page from the mockup has been changed to the “About Us” page instead which has the teams email addresses so it is easier for the player to contact one of the site administrators.

All the other changes that have been made are pure aesthetic changes which still provide the same number of clicks and menu traversals as before. UI minimizes the user effort in the sense that it is a simple interface while providing the tools necessary for the player to go through the registration process and enter a league in little to no time.

HOW TO ACCESS THIS PAGE:

1) From the **Home** page (or any page), click on the **About Us** button at the top.

11.5 Player Stats Page

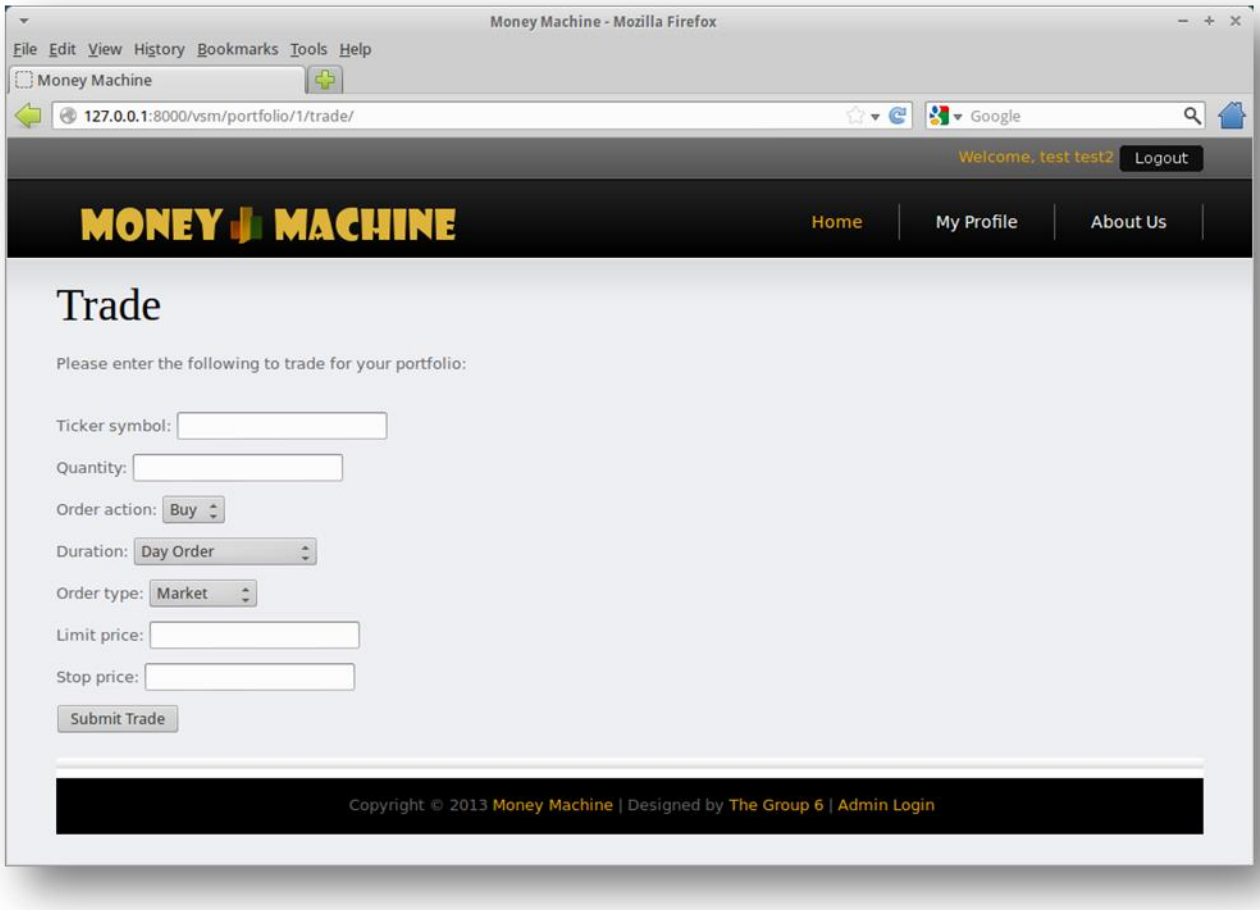


The "Player Stats" page can be viewed by hovering over the "My Profile" tab and clicking on the "Portfolio" page. This then displays all the leagues that the Player is part of. Then to view the stats of the Player for a particular league just click on the appropriate league. This page shows portfolio entries for a particular that the Player is a part of such as Buy Entries, Sell Entries, and the types of orders that the Player has placed. The page also shows the current value of the portfolio, as well as, the cash balance.

<u>HOW TO ACCESS THIS PAGE:</u>

1) From the **Home** page (or any page), hover over the **My Profile** tab.
2) Click on **Portfolio.**
3) Click on the **[League Name]** (in this case its '*private*')

11.6 Trade Page

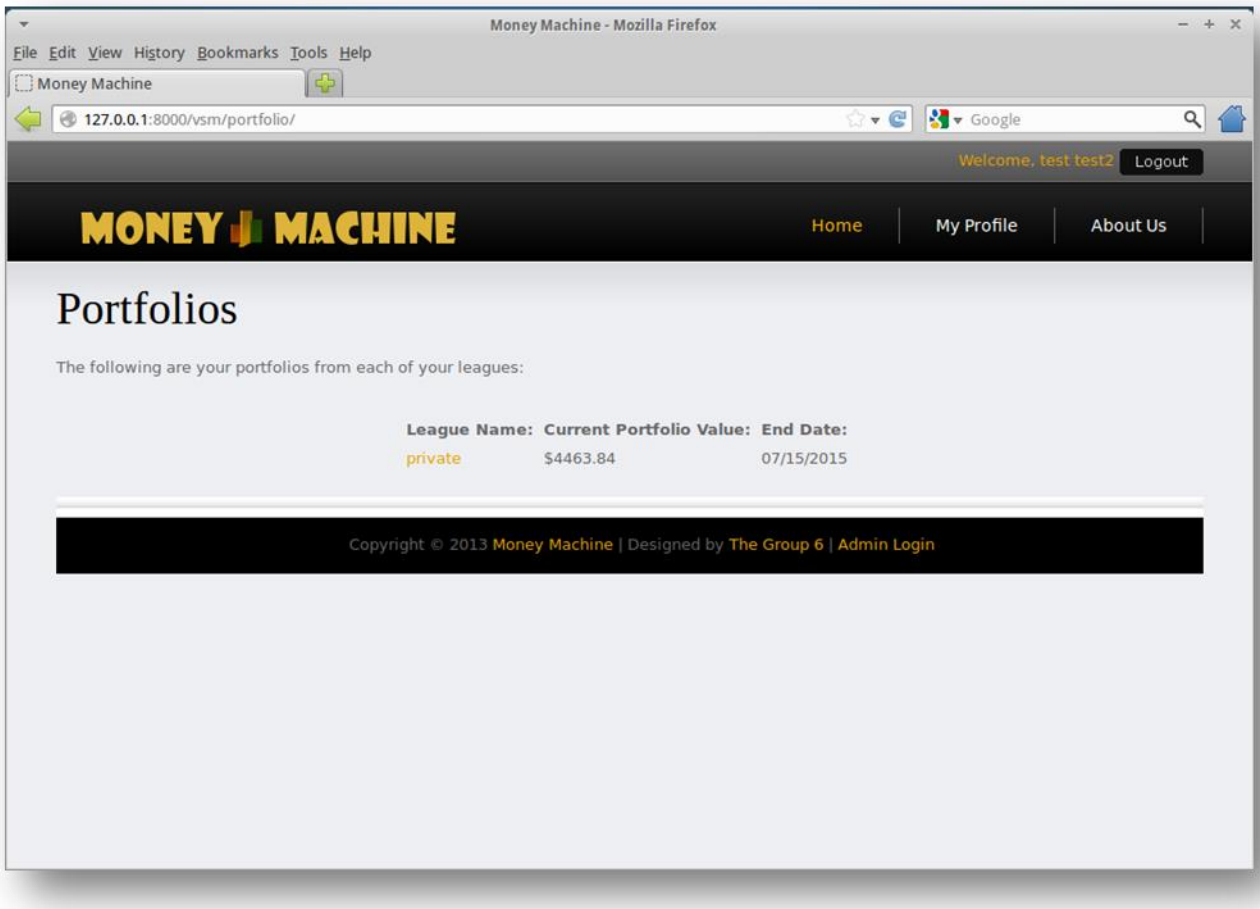


The "Trade" page can be accessed from the "My Profile" tab and selecting the Portfolio page and then selecting a league and then clicking on *'trade'*. This page has various different options that the Player can choose from. For '*Order action'* the Player can choose from Buy or Sell. For '*Duration'* the Player can choose from day order or good until canceled. For the '*Order type',* the Player can choose from Market, Limit, Stop, and Stop Limit. Various different types of orders can be placed. The limit price of the order can be placed as well. To access this page it requires 4 clicks, once all information is entered, the *'Submit Trade'* button should be clicked which will submit the order to the system to be processed.

<u>HOW TO ACCESS THIS PAGE:</u>

1) From the **Home** page (or any page), hover over the **My Profile** tab.
2) Click on **Portfolio.**
3) Choose a league from the **[League Name]** (in this case its '*private'*)
4) Click on **Trade**.

11.7 Portfolio Page

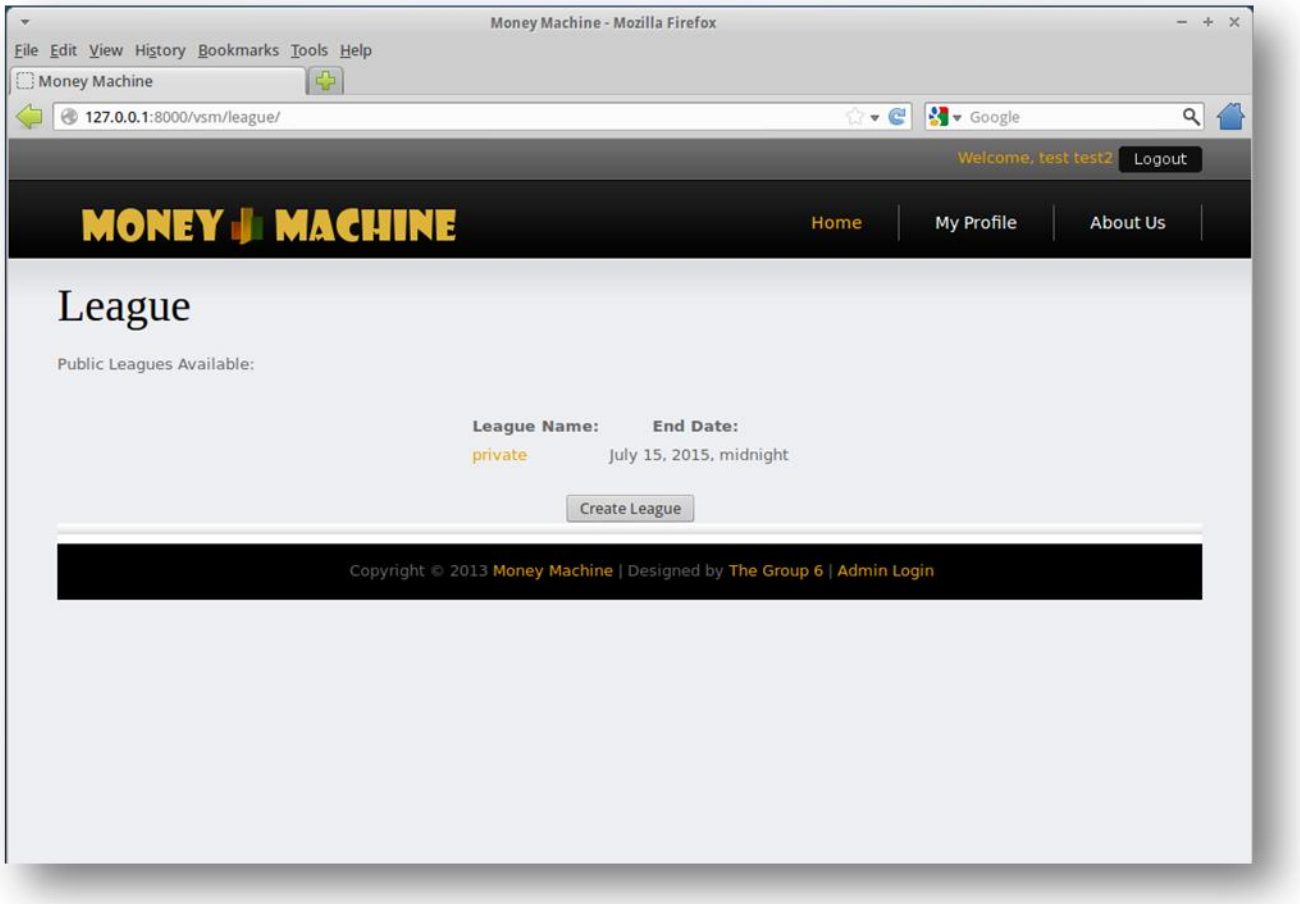


This page displays the general information of a Player's portfolio based on the leagues that the Player is currently a part of. This page shows the League Name, the Current Portfolio Value, and the End Date of the league. After the end date the portfolio will also expire since there will not be a league for there to be a portfolio. To access this page, simply hover over the *'My Profile'* tab and then click on *'Portfolio.'*

<u>HOW TO ACCESS THIS PAGE:</u>

1) From the **Home** page (or any page), hover over the **My Profile** button.
2) Click on **Portfolio.**

11.8 League Page

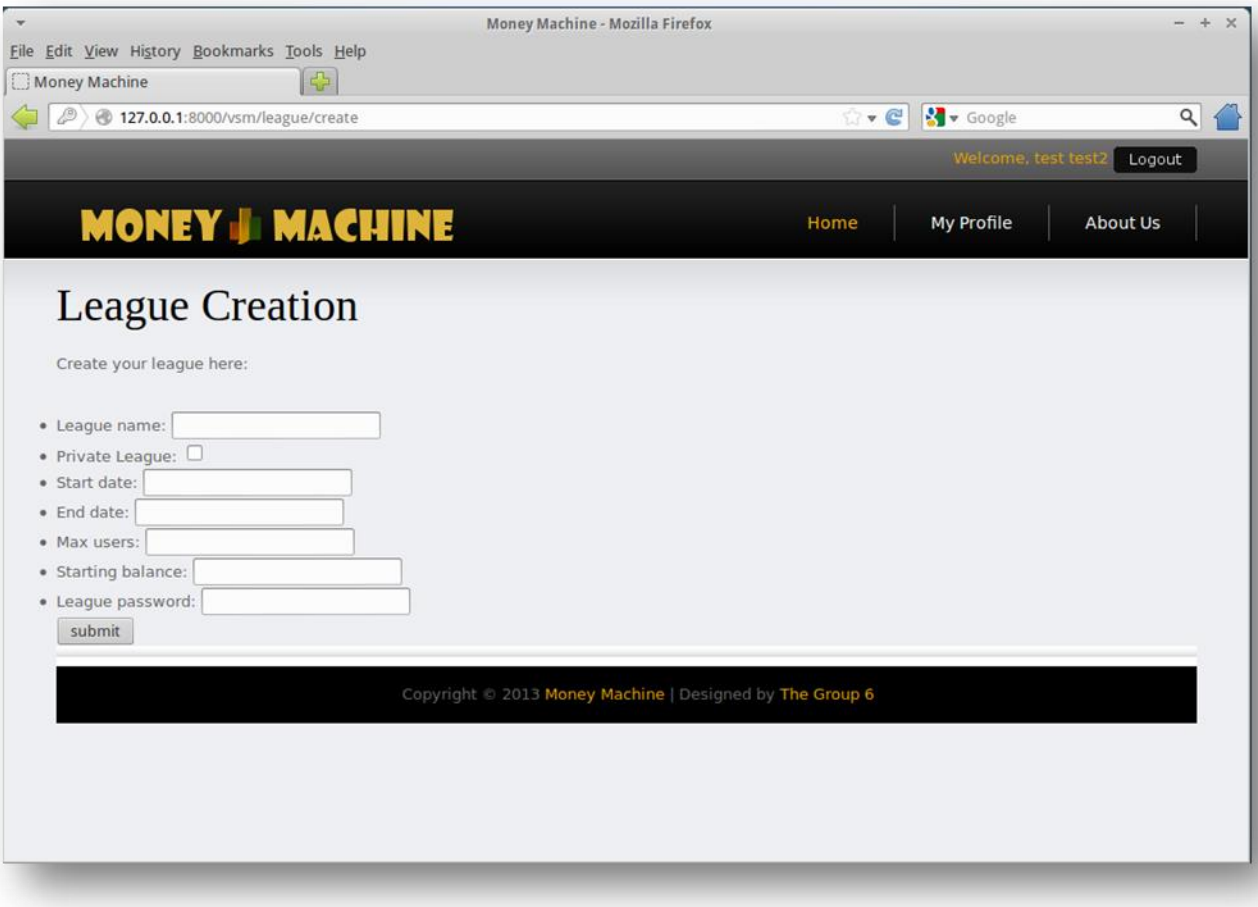


The league page offers players the ability to join different leagues and view the statistics of the currently joined league. To get to the league page the Player has to hover over the *My Profile* tab and selecting the *League* option. The Player is able to view the different types of leagues and also has the option to create a league.

HOW TO ACCESS THIS PAGE:

1) From the **Home** page (or any page), hover over the **My Profile** button.
2) Click on **League.**

11.9 League Creation Page

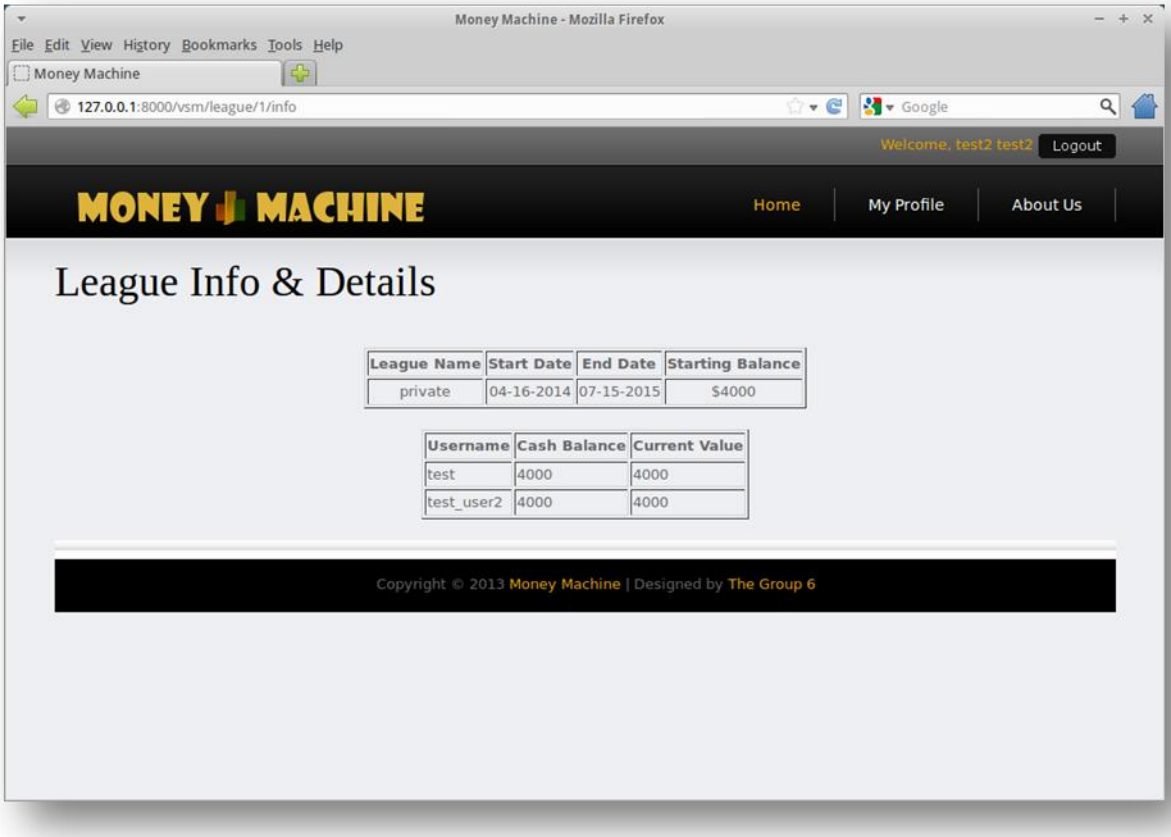


This page offers the Player the ability to create a league. The Player can specify if the league is to be private or a public league. If the league is set to private then a new Player who wants to join the league must enter the league password. As the sample image shows, there are various different fields that the Player must enter in order to create a league.

HOW TO ACCESS THIS PAGE:

1) From the **Home** page (or any page), hover over the **My Profile** button.
2) Click on **League.**
3) Then click on **Create League** button.

11.10 League Info Page



This is the page shows when a Player tries to join a private league (step 3 on How to Access this Page) the Player must first type in the league password that the league creator has specified. Only then the Player will be allowed to join a league and view its information.

12.0 Design of Tests

The following are the tests designed for our system. We plan on updating the tests as we continue to develop our software. These tests primarily encompass our unit testing scheme, however, there is a brief discussion of our integration testing technique.

12.1 Test Cases

Test Case: TC-01 [Log-In Page]

View Tested: virtualstockmark.views.login

Pass/Fail Criteria: This test case will check if the Player has provided correct user name and password successfully. However, the user must be registered with system.

Test Procedure	Results	Actions
Call Function	Pass	User should be able to log in to the system and able to see his portfolio.
Call Function	Fail	If user haven't provided correct user name and password that is registered with the system. This can also be a case when user click log-in button without providing any information. As results, it should notify user and request for the correct information showing (*) next to mandatory fields.

Test-case TC-02: [Validity Checker]

Function Tested: ticket_system.valid()

Pass/Fail Criteria: This test determines if Player has been able to successfully place an order in the market.

Test Procedure	Results	Actions
Call Function	Pass	Player has been successfully able to place an order in the market. The order will be placed only if Player has enough cash balance.
Call Function	Fail	If Player doesn't have enough cash balance, then it won't let Player to place an order.

Test Case: TC-03 [Create League]

View Tested: league.views.create_league

Pass/Fail Criteria: This test case will check to see if the inputs provided by the Player are valid or not.

Test Procedure	Results	Actions
Pass League Name Input	Pass	Player should be presented with the manage league page for the newly created league.
	Fail	The Player will be presented with an error page letting them know that the league name is already taken and will be given an opportunity to try again.

Test Case: TC-04 [Registration Page]

View Tested: virtualstockmark.views.register

Pass/Fail Criteria: This test case will check if the Player has provided correct values for the specified fields.

Test Procedure	Results	Actions
Pass Inputs for First Name	Pass	Player has entered correct values and can proceed to the next field.
	Fail	Display error message and player has to modify the field.
Pass Inputs for Last Name	Pass	Player has entered correct values and can proceed to the next field.
	Fail	Display error message and player has to modify the field.
Pass Inputs for User Name	Pass	Player has entered a unique username and can proceed to the next field.
	Fail	User name is taken and Player must choose a different username
Pass Inputs for Password	Pass	Player has entered a valid password and can proceed to the next field.
	Fail	Display error message and player has to modify the field.
Pass Inputs for Email	Pass	Player has entered correct email address and can proceed to the next field.
	Fail	Display error message and player has to modify the field.
Pass Inputs for Confirm Email	Pass	Player has re-entered the correct email address and can proceed to the next field.
	Fail	Display error message, player has not entered the same email address as in the previous field and player has to modify the field.
Click Register button	Pass	Player has entered all correct information and account is created and is forwarded to the Portfolio screen.

Test Case: TC-05 [Challenge Player]

View Tested: virtualstockmark.views.league

Pass/Fail Criteria: This test case will check if a challenge player request is successful or unsuccessful.

Test Procedure	Results	Actions
Pass Input Challenge	Pass	Player should be presented with a page with time until challenge begins and challenged player's name.
	Fail	The Player will be presented with an error page stating the player is already in a challenge with another player, or invalid challenge request.

Test Case: TC-06 [Data Handler]

View Tested: virtualstockmark.views.orderticket

Pass/Fail Criteria: The test passes if the test stub executes the ticket by updating the investor's portfolio accordingly

Test Procedure	Results	Actions
Execute Order	Pass	DataHandler executes order and updates investors portfolio and returns tree
	Fail	If unable to execute order, return false.

Test Case: TC-07 [Data Handler]

View Tested: virtualstockmark.views.portfolio

Pass/Fail Criteria: The test passes if the test stub request for portfolio data and it is retrieved from the database.

Test Procedure	Results	Actions
Request Portfolio Data	Pass	DataHandler request portfolio data and returns it from the database.
	Fail	If there is an error retrieving the data from the database, it should display an error that no data was returned.

Test Case: TC-08 [Data Handler]

View Tested: virtualstockmark.views.league

Pass/Fail Criteria: This test case will check if a challenge player request is successful or unsuccessful.

Test Procedure	Results	Actions
Request to Edit League Settings	Pass	DataHandler modifies league settings and returns true.
	Fail	DataHandler unable to modify league settings, returns false.

Test Case: TC-09 [DataHandler]

View Tested: virtualstockmark.views.portfolio

Pass/Fail Criteria: The test passes if the test stub request to view transaction history from the database is successful.

Test Procedure	Results	Actions
Request to View Transaction History	Pass	DataHandler returns the transaction history.
	Fail	DataHandler displays an error message, unable to retrieve transaction history.

Test Case: TC-10 [Data Handler]

View Tested: virtualstockmark.views.league

Pass/Fail Criteria: This test case will check if a challenge player request is successful or unsuccessful.

Test Procedure	Results	Actions
Request To Join League	Pass	DataHandler updates information in database about the league and returns true.
	Fail	DataHandler returns false if joining league encounters problem.

Test Case: TC-11 [Data Handler]

View Tested: virtualstockmark.views.league

Pass/Fail Criteria: This test case will check if a challenge player request is successful or unsuccessful.

Test Procedure	Results	Actions
Send invite Player to League	Pass	DataHandler adds the invite to the Players account in database.
	Fail	If fails, displays message unable to send or add invite the player.

Test Case: TC-12 [Validate Login]

View Tested: virtualstockmarket.views.login

Pass/Fail Criteria: To verify that the player has entered either or both username/password and either of these fields are not left blank.

Test Procedure	Results	Actions
Pass input Username/Password	Pass	Player has entered the username/password and is able to login.
	Fail	The Player will be presented with an error on the login page stating the player username and password fields are left blank.

Test Case: TC-13 [Validate Logout]

View Tested: virtualstockmarket.views.logout

Pass/Fail Criteria: To verify that the player is not able to click the back button after clicking the logout button.

Test Procedure	Results	Actions
Pass input Logout	Pass	Player has clicked on logout and is not able to click back after arriving at the logout page.
	Fail	The player is able to click on the back button in the browser even after successfully logging out.

12.2 Coverage of Tests

The test cases are planned to cover all of the possible models and views for every "app" in the Money Machine Project. However, due to the nature of the language being a MVC style language, it is very hard to test individual classes. Nonetheless, it is envisioned that all of the test cases will address all aspects of the application. It is planned that about 75% of the test cases will focus on transition states (ex- making sure that the application transitions from the league app to the portfolio app when the user wants to view a portfolio in a league.) The remaining 25% will be devoted to creating tests that test the UI specification.

12.3 Integration Testing Plan

The integration testing will be done with each component individually at first and then with other components for the project. The basic template of the website was written to make sure that each individual page can be accessed from another page. Once the templates for the sites are done, the team will begin to develop the actually methods and models that will be used for the project such as the Portfolio System, League System, Ticker System, Login System, and Registration System. The Registration system was first developed separately to test if the databases are working properly and then it was integrated into the website templates that were originally created so any visitor to the website can register for Money Machine. Once the Registration System is working it can be used in conjunction with the Login System for user authentication. After the Login System is created, testing is done on the system to see if a player is able to register properly and also able to login using the username that was created. Further testing of the authentication of the system has to be done to maintain a secure logout. For example, if an authenticated user clicks on the *Logout* button then the player must be brought back to the home page of the website and must be re-authenticated if the player decides to click on the *Back* button in the web browser.

Once the login system is working properly, the Portfolio System and the Ticker System are going to be developed separately, with a higher priority on the Ticker System. The Ticker System is one of the most important aspects of this project because it will handle all the queries for buying and selling stocks. The Ticker System has to be tested thoroughly to make sure that each buy and sell query is handled properly. Once this system is working properly it can be integrated with the Portfolio System, League System, and Registration System. At first, it will be tested with the Registration/Login System to make sure that each individual player is able to buy or sell stocks and if the Ticker System is able to reflect the transactions to the individual player. Once this test is done, it will be integrated with the Portfolio System, so the player can start

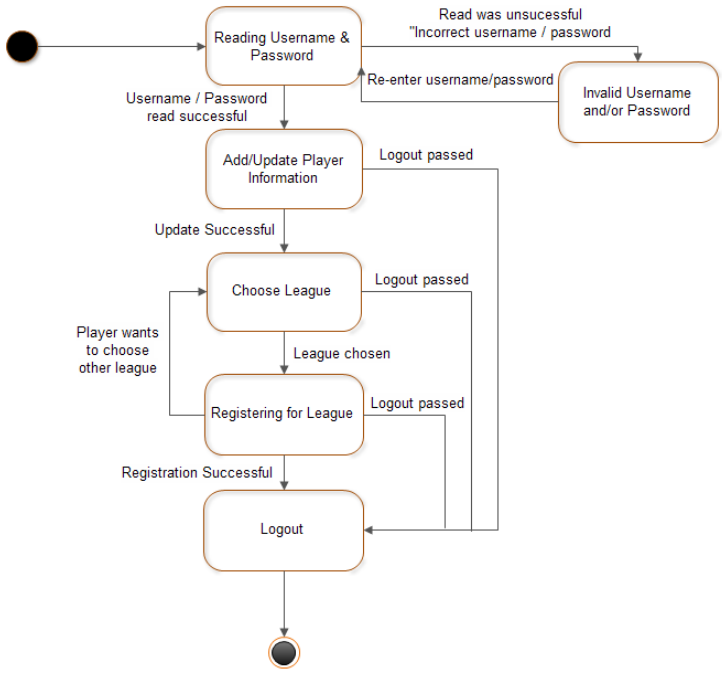
building his/her portfolio. While this testing and debugging of the integrated system is being done, the League System will also start to be developed by 2-3 group members so there is no delay in the software development. By the time the League System is done, it is expected that the integration between the Ticker System, Portfolio System, and the Login System are working in unison. The League System will be tested at first if it can handle faux data that the team will generate, such as, player names, net worth, rank, and daily loss or gain. If this test passes then it will be integrated with the all the previous systems to full complete the project. In the end, if all the systems are working, a player should be able to register for Money Machine account, create and maintain a portfolio, join a league, obtain statistics about currently joined leagues, and buy and sell shares and have the transactions reflect within a portfolio.

At first, each system will be tested individually with some sort of faux data that will be generated to make sure that individual system is working properly before being integrated with other systems. Testing and debugging is a major component of software development, however, if the debugging time is far too great than some aspects of the project might not get properly debugged due to project deadlines. Since each stage of the application development is being tested individually, hopefully this reduces the amount of errors when each system is integrated with one another.

12.4 Testing State Diagrams

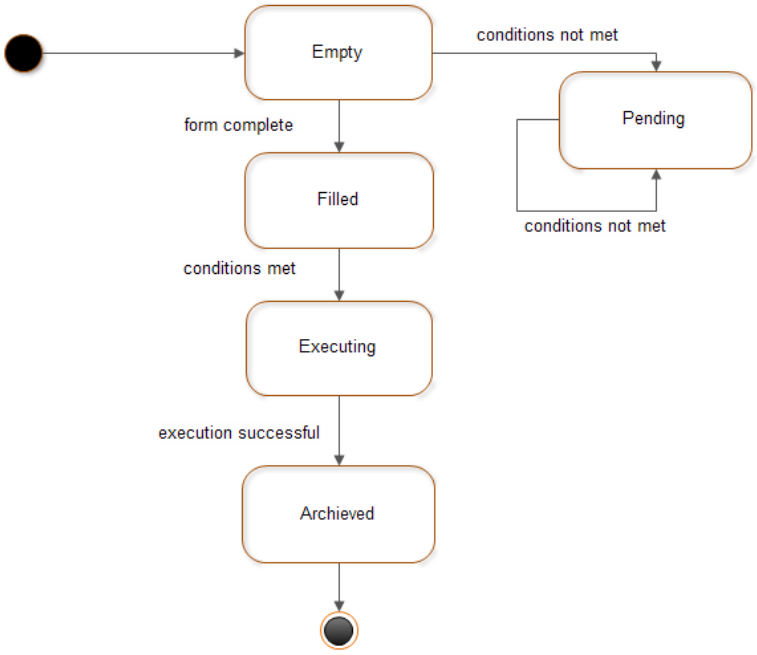
State Diagram: Registration / Login

The following state diagram below shows how registration and login by a player could be tested.



State Diagram: Order Ticket

The following state diagram below shows how a order ticket is tested.



13.0 History of Work, Current Status, & Future Work

Our project has had a very unique course of development. When we look back on it as a team, it was a smart idea to have a long list of features we planned on implementing. In the end, we ended up implementing less than half of the proposed features. The biggest issues involved the following:

1. Finding out how to implement features (we are all new to the programming language and framework we chose)
2. Finding free places to gather data from

Nevertheless, our team has met our goals and exceeded them in some cases. For our documentation timeline, we always hit the schedule. We were always ready before the due date.

Below is a copy of the coding sections of Reports #1 - #2. Attached is an additional column to tell whether a function was On-Time, Cut / Modified (Removed or Changed), or moved to another date (Final date given).

Task Name	Duration	Start	Finish	Actual Finish
Coding - Round #1				
Begin Coding (Everyone Ready w/ Django, Git, etc.)	0 days	Fri 3/1/13	Fri 3/1/13	On-Time
Setup simple Django Site	2 days	Fri 3/1/13	Sat 3/2/13	On-Time
Simple login system, user accounts, templates	5 days	Sun 3/3/13	Thu 3/7/13	On-Time
Create Portfolio Model, Buying / Selling Stocks	7 days	Thu 3/7/13	Fri 3/15/13	On-Time
Create Fetching of Stock & Options Prices	10 days	Mon 3/4/13	Fri 3/15/13	Cut / Modified
Create League System	6 days	Sat 3/16/13	Fri 3/22/13	On-Time
Develop Administrative Settings	7 days	Sat 3/23/13	Sat 3/30/13	5/1/2013
Tweaking / Adjustments	2 days	Sun 3/31/13	Mon 4/1/13	On-Time
Demo #1	0 days	Mon 4/1/13	Mon 4/1/13	
Coding - Round #2				
Suggest Security / Quiz Functionality	6 days	Tue 4/2/13	Tue 4/9/13	Cut / Modified
Tutorial System (1-2 Simple Tutorials)	6 days	Tue 4/2/13	Tue 4/9/13	Cut / Modified
Security Watch List	6 days	Tue 4/9/13	Tue 4/16/13	On-Time
Challenge Player	6 days	Tue 4/9/13	Tue 4/16/13	Cut / Modified
Administrative & Advertiser Functionality	11 days	Tue 4/16/13	Tue 4/30/13	Cut / Modified
Mobile Application	11 days	Tue 4/16/13	Tue 4/30/13	Cut / Modified
Tweaking / Adjustments	4 days	Thu 4/25/13	Tue 4/30/13	On-Time
Demo #2	0 days	Thu 5/2/13	Thu 5/2/13	

In coding round #1, we focused on getting a basic site setup. The changes were made for the following reasons:

- Options prices was removed. This also included removing bond prices. We could not find a free data source to provide us market information for either. All data sources charged $200+ for using their API. We looked into alternative methods, but it was unsuccessful. We decided to only implement the fetching of Stocks, ETFs, and Mutual Funds.

- The Administrative panel could not be implemented fast enough for the Demo #1. Instead, we decided to push it to Demo #2.

For round #2, we focused on enhancing features. The changes to the schedule were made for the following reasons:

- We determined that the 'Suggest a Security' functionality could not be completed using a simple algorithm.
- The 'Tutorial System' would take too much time to write flash / make videos. This is not true software engineering. We decided to replace the objective with a simple FAQ page.
- The 'Challenge Player' functionality would require an alert system / e-mail system to challenge another player. Our team could not determine a suitable method to implement an e-mail alert system using our programming language or framework.
- The Administrative functionality was kept in Demo #2. The Advertiser functionality was not. Page impressions / advertisements were not easy to implement in our setup. Likewise, creating a new user class for Advertisers would be very complex.
- Creating a mobile application would require remaking the site templates. This would be effectively doubling our project workload. It was cut.

To compensate for the features we cut, we decided to balance by implementing the following features (pulled from our original proposal!):

- Having a FAQ Page instead of tutorial system
- Having a robust administrative panel
- Creating a global ranking system
- Adding to our ticket system, which was not complete for Report #1 (only one type of trade had been implemented; more options were needed)
- Creating a private league implementation
- Creating a league management page to manage a league
- Portfolio watch lists for users to keep track of stocks they are interested in
- Minor bug fixes, and enhancements to the UI

These features were spaced out, and replaced the cut features. When contemplating whether to cut / modify a feature, we took into account the scope of the work required to implement a feature, the technical complexity of the feature (e.g. a tutorial system would be easy to implement, but take hours to create videos, flash games, etc.; not really software engineering), and the knowledge of our team. When possible, we substituted a feature with a different feature which was possible to implement.

Our project has met many of our goals. Some key accomplishments of our project:

- Developed a clean, easy to use UI for our project
- Instant market trading connectivity (ability to get live quotes from the market)
- A robust administration panel for the project website
- Ability to join multiple leagues, and fully trade common market products such as Stocks, ETFs, and Mutual Funds
- Global player ranking system to showcase best overall players and portfolios
- League privacy options
- Google stock research box
- Live finance news feed

Our project has a large amount of room for growth. The biggest areas for growth involve the features we did not have the time, or ability to implement. Our team spent many days looking for ways to get Bond and Options pricings. However, both are made of large amounts of frequently changing data. The data must be purchased at a nominal price (from a firm such as Bloomberg or Options House), and determine the data via an API. Fetching data from a page and parsing it is not an option; most places with Options data (e.g. Google Finance) obscure their source code to prevent sneaky developers from using such a tactic. Perhaps our team overlooked such a feature. It would be nice to include one of them.

Likewise, connectivity to a social media platform would be useful. Having a unified login using a Facebook API call would be great to bypass the registration system. Developing a message board for users to post questions, or having a chat box for the league would be useful.

Our project implemented a simple one-page FAQ. We planned on having a robust tutorial system, but we found that it would require too much time making videos / doing simple non-technical tasks to create. If this software were to be actually purchased by a company, a tutorial system would be in order (to use our software, and to teach end users how the stock market works).

To differentiate our product, a future revision could have 'challenge features'. Our team omitted these features due to time constraints.

Our team never had the opportunity to finish writing a complete trade system. Only 2 different trade types were implemented. There are multiple trade types, and a full trading system should be able to offer all of them.
Having an advertiser presence on the platform would be useful, as well as administrative settings to manage the associated advertiser user accounts.

Clearly, there are a variety of features to be implemented using our platform. However, because of our limited (3-month) time constraints, we barely had the opportunity to scratch the surface with such advanced ideas. Perhaps with more time, more experience, and more ingenuity, our team could deliver a more robust, and functional product.

14.0 References

1. Marsic, Ivan. Software Engineering. Rutgers University, unpublished. 2012. Web
2. "UML Class Diagram Help", Class Draw. Macrospark Solutions, n. d. Web. 03 Feb. 2013.
3. MarketWatch. MarketWatch, 18 Oct. 2011. Web. 29 Jan. 2013.
4. Group 6. Bears & Bulls. 03 May. 2012. PDF file
5. Group 2. Stockhop: The Stock Market fantasy League Game. n. d. PDF file