“Trade Kings”

Report 3

https://github.com/bsonani/TradeKings

Software Engineering (14:332:452)

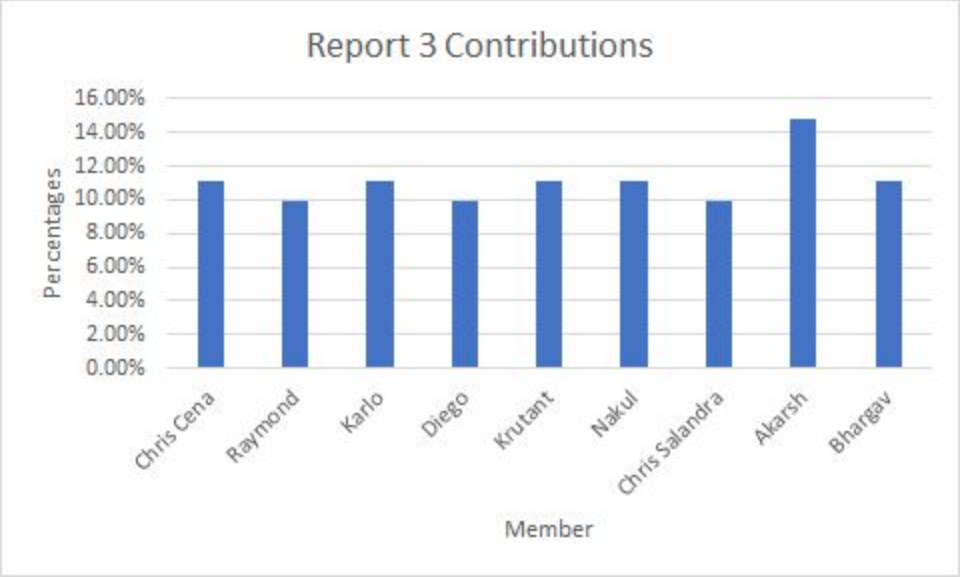
GROUP 13:
Christopher Cena
Raymond Del Rosario
Karlo Fernandez
Diego Ordonez
Krutant Patel
Nakul Patel
Chris Salandra
Akarsh Sardana
Bhargav Sonani

April 30, 2018

Table of Contents

Table of Contributions:

Members:	Chris Cena	Raymond	Karlo	Diego	Krutant	Nakul	Chris Salandra	Akarsh	Bhargav
Total Contribution (100%)	11.11%	9.90%	11.11%	9.90%	11.11%	11.11%	9.90%	14.75%	11.11%



Summary of Changes

1) Updated Problem Statement to reflect unique features that we have both implemented and brainstormed since Report 1. All of these features were presented and discussed during Demo 1.
2) Updated glossary of terms by not only adding new terms, but also redefining existing terms to add clarify and better reflect our product.
3) Updated System Requirements, by changing the descriptions of certain requirements to better reflect our actual implementation of them (and our updated goals) and by adding new system requirements as well.
4) Updated the Functional Requirements Specifications, by changing the descriptions of existing use cases to better reflect our implementation, as well as adding more use cases. Redid **all** of the system sequence diagrams to reflect the aforementioned updates and provide greater uniformity. Updated casual descriptions, fully dressed descriptions, and traceability matrix to match the aforementioned changes.
5) Added Effort Estimation using use case points which is comprised of the complexity of the use cases and actors on the system, technical complexity, and the environmental factor.
6) Updated the interaction diagrams to meet current standards including adding additional ones to better represent Trade King's functionalities as a whole.
7) Added both the implemented and planned design patterns to the interaction diagram section in order to accurately reflect which patterns were implemented in the report.
8) Updated the Domain Analysis to reflect new use cases and modified ones that more accurately represent the Trade Kings Website.
9) Algorithms and Data Structures were modified slightly to reflect new structures associated with Trade King's price alerts.
10) Many classes were added to Class Diagrams and Interface section in the format of the Object Constraint Language. Invariants, preconditions, and postconditions have been listed for the classes and functions.
11) User Interface and Implementation have been updated to accurately depict our second demo
12) Design Tests have been revised and updated accordingly to the Trade King's Design
13) Updated interaction diagram #3, (joining a league), to reflect the publisher-subscriber design pattern.
14) Update our History of Work as we explain in detail the amount of work done after the first demo.

15) We added Current Status and Future Work as what we have accomplished with this project and the class itself
16) Updated User Interface Design and Implementation, by adding two features, Analyze and forum, that was shown working in Demo 1.
17) Updated History of Work, Current Status, and Future Work, by adding current progress, key accomplishments and future implementations that will be added before Demo 2.
18) Added additional interaction diagrams that include implemented design patterns (5.1, 5.3, 5.6).
19) Added more references as we used additional sources since Demo 1

1 Customer Statement of Requirements

A. Problem Statement

At Trade Kings, our goal is not only to develop an engaging simulation where players gain exposure to the stock market, but also an education platform where users can take their investing know-how and finance knowledge to the next level. Customers will begin by first registering on our website and creating an account. The registration process should be simple, and only consists of the customer's email, age, username, and password. Once this information is put in, the user will automatically receive any email from Trade Kings confirming that his account has been registered.

Upon registering, users will be able to login from our site's homepage, using the username and password that they chose while registering. When a new user logs in to the Trade Kings website for the first time, he/she will automatically be in Tutorial Mode, which essentially shows new users how to navigate the site and use our tools. Specifically, Tutorial Mode will show new users how to create a league, join a league, research the stock/cryptocurrency market, track various securities by setting price alerts, execute trades (buying and selling securities), and taking part in the Investment Forum.

After going through Tutorial Mode, users are ready to get started and begin playing. To make the game more exciting, player will compete in leagues, which can be made up of either 8, 10, 12, or 16 players. Users can either join a public league with open spots or create their own, which can either be private (players must be invited) or public (anyone can join). If a user chooses to join a league, then that user will be taken to a screen that lists all the joinable leagues. The leagues will display how many users are in that league, when it will begin, how much capital each player will start out with, and what mode will users play in (Industry Mode, Traditional, Cryptocurrency Mode, or Equities Mode) and how the winner is decided. Therefore, users can make a decision on which league to join based off their preferences and what is available. If a league is private, then the user can message its League Manager and ask for permission to join the league. If a user chooses to create their own league, then that user is automatically the League Manager, meaning they choose the League name and decide the league settings (described further below). The game will simulate the basic functionality and provide many of the same features as the US stock and cryptocurrency markets, but instead of real money, users of the website will be able to invest with virtual money. This default amount of money that each player starts with is $10,000, however, this amount can be changed by the League Manager when creating a new league. Players in the same league will all start with the same amount of capital,

and will compete with each other to see who can make the smartest investments; their performance will be recorded via a league leaderboard, which is updated daily. Players can participate in multiple leagues, so that they can play with different friend groups or in leagues with different rules. For example, a particular player who is a student in college might be in one league with his friends, another league with his classmates, and a third league that is with random players where the rule is that users can only invest in cryptocurrencies ("Cryptocurrency Mode"). Likewise, leagues can be set up such that players are only able to invest in equities (common stock), which is called "Equities Mode". Another mode that we will also create is "Industry Mode". In this league setting, players are restricted to only investing a maximum of 30% of their initial capital in just one industry/sector. For instance, if a player starts with the default amount of money, $10,000, than they cannot invest more than $3000 in one industry(Consumer, TMT, Industrials, etc.) The final mode is called "Traditional", where users can invest in both Equities and Cryptocurrencies, and their only restriction is the amount of capital they have. There are two different ways that a league winner is decided (this is chosen at the beginning by the League Manager).

Users will have the opportunity to invest in the same stocks and cryptocurrencies that are featured on the major exchanges in the US. Users will be able to see recent prices on all securities and buy or sell accordingly. Players will also have multiple tools available to manage their portfolio and to perform research on securities before they make an investment. For example, users should be able to track the performance of individual stocks that they have taken a position in, as well as their overall portfolio. Moreover, when researching equities to buy, users should be able to clearly see the stock's metrics (such as P/E ratio, EPS, etc.) as well as how the stock's price has changed over time. Moreover, another feature that we will be introducing is price alerts. When a player is intrigued in taking a position in certain stock, but feels that the current share price is too high, then the player can still track that stock and be alerted via email when the stock's share price lowers into the player's target range. Similarly, the player can set up other alerts as well for the different metrics of the stock.

In addition to providing a simulation of real markets, what separates Trade Kings from other products and makes it unique is its commitment to providing a strong educational platform. The first way that we will do this is by providing daily market updates on our homepage. These updates will primarily consist of listing the current values of major indices, commodities, and major currencies (as well as important exchange rates). Furthermore, the market updates will also highlight any large movements in the price of a well known stock or cryptocurrency, and notable events in the US and world economies. Additionally, we will have an Investment Forum, which is a user-driven forum where members can make posts on their investment ideas with other users being able to comment below the posts. By allowing users to present and support investment ideas in a particular stock or cryptocurrency, we can generate insightful community discussions that encourage members to explore other valuation strategies that might be new to them. Finally, the site will also have a "Valuation Strategies" tab, which will present information

on how different valuation strategies are performed, such as SWOT analysis, Porter's 5 Forces, a Discounted Cash Flow Model, and more. This section of the website can only be updated by our Educational Content Contributor. The Educational Content Contributor, who will also serve as the moderator of the Investment Forum, will be chosen for their experience in investing and for their credentials in the field of finance.

Furthermore, unlike many other Trading Fantasy Leagues, Trade Kings allows users to execute more than just buy and sell trades. A new feature that we will be introducing is a "short sell". This strategy is an integral part of the investment world, so it is important that we provide this function in our product. A short sell is when an investor borrows shares from a broker and immediately sells them on the market. Then, when (or more importantly, "if") the share price of that stock decreases, the investor can purchase the same amount of shares that he/she originally borrowed and return them to the broker. The difference in share price at the time of borrowing and buying is the profit that the investor can potentially make. Furthermore, we also added the ability for users to invest in Treasury bonds. These fixed income investments are an extremely important part of our market, as it is the safest investment anyone can make.

We will also be introducing more unique features regarding the Social aspects of Trade Kings. In addition to having an investment forum, where users can engage in community discussions, we will also provide a social media integration, so that users can login from various social media websites, such as Facebook. Furthermore, we will allow the ability for users to connect their Trade Kings account to their Paypal account; this way, user can engage in leagues where the winner gets prize money (at the start of the league "season", each player will pay a certain amount, and the winner gets all the money).

Another unique feature that Trade Kings will be implementing is a basic forecaster for market trends for both stocks and cryptocurrencies. That is, we will be using Artificial Intelligence and using certain markets indicators to estimate future prices for both stock indices and cryptocurrencies.

B. Glossary of Terms

Ticker Symbol - A sequence of letters used to uniquely identify a stock or cryptocurrency.

Dividend - A payment paid regularly by a company to its shareholders using its profits. It is usually reported as a fixed percent of the share price.

Leaderboard - Ranking system to distinguish the most successful player from the rest.

Share - a unit or miniscule percentage of a company which can be used in transactions whether it be for selling or purchasing.

Share Price - the value equivalent in cryptocurrency that a particular share is worth at any given time.

Equity - a synonym for share price. A term referring to the value of an individual share of a stock.

Security - An asset maintained by a player which include debts, equities and derivatives which will each play different roles throughout this game.

Portfolio - Earnings, losses, averages, performance and other asset analysis are included in a player's portfolio. This will include a player's bio, homepage as well as other features associated to a particular player. The portfolio is basic summary of a player and what they have accomplished.

P/E Ratio - Ratio for the current share price relative to its per-share earnings. This ratio determines the value of a company.

EPS - Earnings per share or EPS is a percentage of stock's profit allocated to an individual who is in ownership of that stock.

Cryptocurrency - Digital asset designed to work as a medium of exchange in which encryption techniques are used to secure its transactions,and to verify the transfer of assets.

Buy - A transaction in which an individual user exchanges an amount of cryptocurrency for an individual or group of stocks. This is an essential component of this game.

Sell - The user no longer needs a stock and wishes to sell it at the desired price.

Limit - User makes an order to buy or even sell a stock at a determined pricer. A buy limit order can only be completed at the limit price or lower. Also a sell limit order can only be completed at the limit price or higher.

Stop - Operations that take affect if a particular stock rises above or falls below a particular target value. If triggered this operation will effectively purchase or sell the selected stock. This is mainly used to minimize losses for a player.

ROI - Return on Investment

League - A stock market simulation with fixed number of users in which all work to reach the same goal. The goal is to attain the most money at the end. All user, or investors, can choose to participate in a private league, or a public league in which users of 8 are made randomly.

League Manager (or League Moderator) - League Manager is one of the users in the league who is responsible for adding other users as well as selecting the settings for the particular league. The manager can only control the settings of his or her own league.

League Forum - A league chat where the the League Manager can notify all member of anything within the league. Normal league members mave have access to sending messages to the forum if the League Manager chooses to give the ability to do so. The League Manager will always have the right to delete anything from the forum for any given reasonable reason.

Player - A player is any user that has an account on the Trade Kings website.

Invite - Anyone who has access to do so, can invite other members to join the league. The amount of access given to all members to do so is determined though the rules set by the League Manager/ Moderator.

Price/Metric Alerts: Users can track the price and various metrics of stock/cryptocurrencies so that they are alerted (emailed), once the value of that price or metric meets the criteria chosen by the user (for instance, a user can choose to be alerted once the share price of Facebook, FB, rises above $180.00).

Short Sell: When an investor borrows shares from a broker and immediately sells them on the market. When the share price of that stock decreases, the investor can purchase the same amount of shares that he/she originally borrowed and return them to the broker. The difference in share price at the time of borrowing and buying is the profit that the investor can potentially make. This type of investment is much riskier, because theoretically, there is no limit to how high a share price can rise, so an investor risks to potentially lose a lot more money if the share price rises a significant amount instead of decreasing as the investor predicted.

2 System Requirements

A & B. Enumerated Functional & Non-functional Requirements (Combined)

Identifier	User Story	Priority
ST-1	As a new user (guest visitor), I can only participate in Trade Kings if I register for an account with my email.	5
ST-2	As a newly registered user, I will be given an introduction to the site and the different features offered by it (this is known as Tutorial Mode).	2
ST-3	As a customer, I can research and compare current stocks in order to make a decision on what investment I should make. This includes viewing not only numerical data on a particular stock/cryptocurrency, but also having access to related news articles and being provided with a list of similar stocks/cryptocurrency. For instance, if I am looking at FB (Facebook), then I will be given a list of similar technology stocks that I can compare Facebook with.	4
ST-4	As a competitor, I can view a current leaderboard in order to view the progress of my friends or foes.	2
ST-5	As a competitor, I can observe the profiles of my competition and view their portfolio and list of transactions	2
ST-6	As a league manager, I can create a league with a personalized name and settings.	2
ST-7	As a league manager, I can send invitations to other users via email.	4
ST-8	As a user, I can purchase and sell stocks based on current market prices. I can also execute a short sell for stocks.	5
ST-9	As a user, I can accept or decline an invitation to join a league.	3
ST-10	As a user, I should be able to clearly see the stock's metrics (such as P/E ratio, EPS, etc.) as well as how the stock's price has changed over time.	4

ST-11	As a user, I will be be alerted via email when the stock's share price, which I'm interested in, lowers into my user set target range	3
ST-12	As a user, I will be able to see recent prices on all securities.	3
ST-13	As a user, I can put stocks into watchlists and have the ability to track how well they do over time.	2
ST-14	As a user, I can customize my avatar in my profile to my liking.	1
ST-15	As a user, I can link any social media accounts or any relevant channels in my profile.	1
ST-16	As a user, I can add posts and comments to the Investment Forum.	3
ST-17	As an Educational Content Contributor & Forum Moderator, I can delete inappropriate Forum posts/comments and post educational material on investment strategies.	3
ST-18	As a user, I can link my Trade Kings account with my Paypal account, if I have one, and make league payments using that Paypal account.	1
ST-19	As a user, I can view predicted future prices for stock indices and cryptocurrencies, as provided by the Trade Kings Forecaster.	4

The images below (Figure 1&2) are temporary template of the on-screen requirements for the login page and user interface. Upon final release of this project the user will be able to see a background image with a login pattern that requires each player's username and password.This is to demonstrate what each user can expect upon entering Trade Kings website. Below also lists a forgot password for players who need an email to confirm their password and a signup option for new members. The requirements of this login page is specified in the table above.

Figure 1

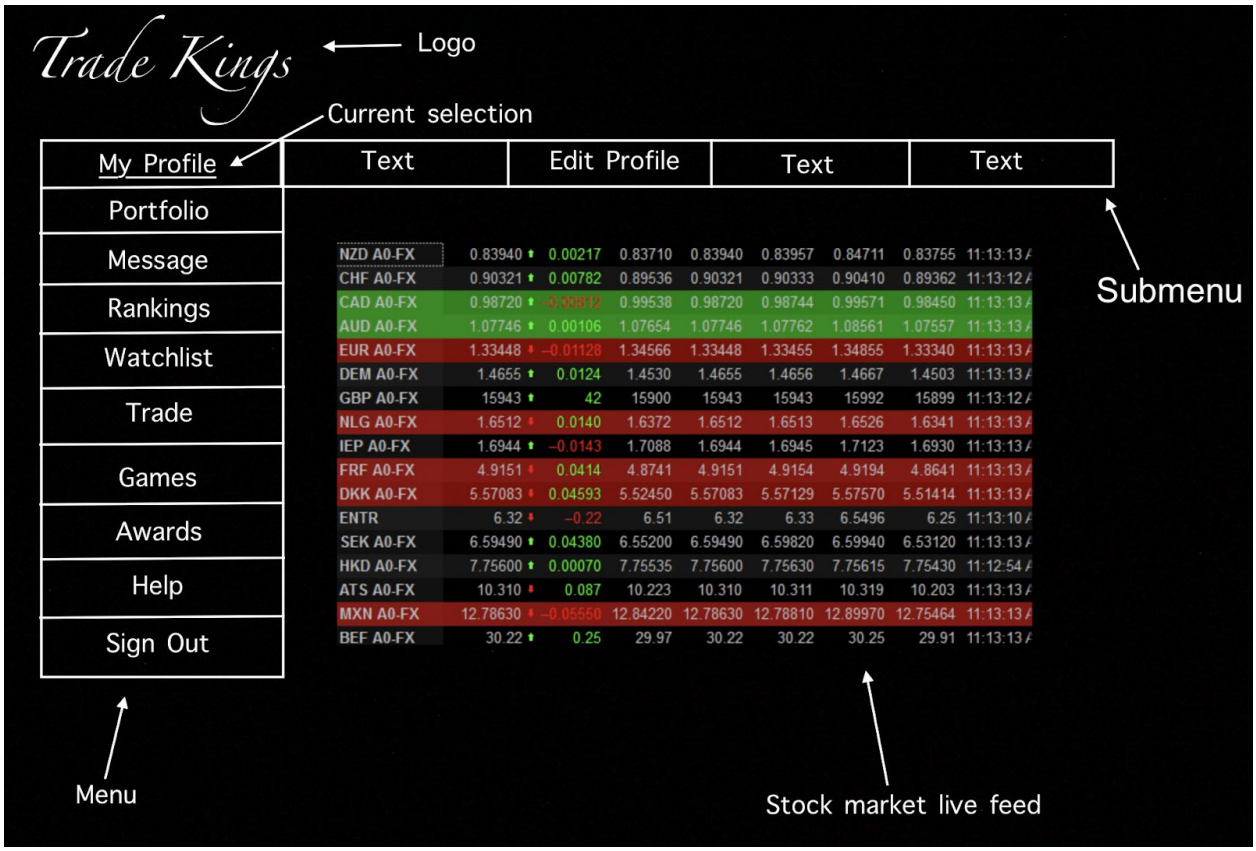


Figure 2

C. On-Screen Appearance Requirements

REQ-X	Description	Priority
OSR-1	The login screen will contain a portion for one to enter their username and password. Also, the option to sign up or recover a forgotten password or username will be present.	5
OSR-2	The in-game screen will contain tabs to show messages, games joined, rankings, portfolio, and stock analysis.	5
OSR-3	There will be a tutorial video for new users in order for them to gain a better understanding on how a stock market functions.	5
OSR-4	A web page will adjust to occupy any computer screen.	4
OSR-5	The main page will contain a stock market live feed.	4
OSR-6	There will be a leaderboard showing current ranking of the players in the particular league.	3

3 Functional Requirements Specification

A. Stakeholders:

A stakeholder is an individual, team, or organization with interests in or concerns related to the system-to-be of Trade Kings. The primary stakeholder and main end-user for our Trade Kings platform will of course be the customer, who can be anyone from a student who is a novice investor looking to gain initial exposure to the markets, or a more experienced investor wishing to expand their knowledge to new frontiers. The system architects is simply our nine person development team that will be building the system-to-be and subsequently testing its performance. We will also have a subset of our development team that will comprise the "future maintenance organization" of the system. The responsibilities of this group is mainly to update the different site appearances which will improve the clarity and viewability of the webpage. Furthermore, they will introduce new functionalities to the league, and update the finance API if the current one becomes outdated or defunct (such as the Yahoo Finance API).

The main external system that our Trade Kings system-to-be will interact with is the finance API, which will provide us with real-time stock and cryptocurrency data. Our site will display this data in a clean and organized manner through tables and various graphs which will illustrate not only the numbers but different patterns in order to make the users' analysis of the market and decision-making process easier. The finance API that we are planning on using is the Alpha Vantage API, which supports both current and past data for equities and digital currencies, as well as numerous technical indicators and metrics that will help users as they make investment decisions.

There are additional potential stakeholders that we hope to introduce in the future after our product hits the market and gains a certain amount of users. After reaching a certain level of success, we hope to add business schools as sponsors, starting with the Rutgers Business School (RBS). Sponsors will have a version of the site specific to their business school, where only students from that particular school can register with their student ID (for example RUID for RBS students) and participate in a league. As a result, these schools can use our site as another educational tool for their students because along with educational content, they will be exposed to a stock market simulation. Moreover, sponsors will have access to premium educational material that is not available to traditional users, as we continue to expand the educational content on our site. This will allow finance classes to incorporate investment fantasy leagues as well as investment forums into their curriculum with ease.

B. Actors and Goals

Player (Initiating and Participating): An end user who is an investor and is either currently in a league(s), was in a league(s), or is looking to join one.

> **Goals:** Joining a league, creating a league (becoming a League Manager), viewing portfolio, executing trades, creating a new discussion in the forum, contributing to an existing forum (becoming an Investment Forum Contributor).

League Manager (Initiating and Participating): Similar to a player, except this user decides to create his/her own league, instead of joining one.

> **Goals:** Deciding the league settings, such as number of players, starting amount of capital of every player, duration of the league, game-play mode, etc.

Guest Visitor (Initiating): An end user who either is not logged in or has not yet registered an account. User will be able to see the open leagues and the forum discussion topics, but cannot join any league or actually read the discussion.

> **Goals:** Registering an account, logging in to account, browsing the website to get some understanding and explore their interest in it

Site Administrator (Initiating): The site administrator will ensure that the website is operating smoothly which includes speed, quality of graphics, accuracy of data, etc.

> **Goals:** Managing and maintaining the performance of the website by monitoring the "error complaints" sent in by users (Trade Kings will have a special inbox dedicated to this) as well as researching, implementing, and testing new opportunities to improve the quality of league as a whole.

Finance API (Participating): The Alpha Vantage Finance API will provide the site with real-time stock and cryptocurrency data.

> **Goals:** Retrieving current market data and updating its database with the market data so that it can be easily transferred to users.

Investment Forum Contributor (Initiating and Participating): Players also have the ability of making posts in the investment forum, where they have community discussions on various discussion topics about the market in general or in relation to their game.

> **Goals:** Making posts on new investment ideas and market developments, commenting on existing forums with opinions, insight, and analytics

Educational Content Contributor & Forum Moderator (Initiating): Specially selected and qualified individuals (not a player) that will be in charge of providing and moderating much of the educational content on the Trade Kings website.

Goals: Moderating investment forum discussions, providing daily Market Updates, making periodic posts on the Investment Strategies page on various valuation methods and investing techniques

Database (Participating): Once a user registers, the user's registration information will be stored in a database. This information will be used to send emails (i.e. for confirming registration, sending price alerts, etc.)

Goals: Storing necessary player information

C. Use Cases

I. Casual Description of Use Cases:

UC-1: Registering: Allows a visitor to set up an account and become a member/user via a quick registration form. User needs to provide first name, last name, age, email address, password, and answers to two security questions. Upon successfully registration, new user will receive a confirmation email to the address that was provided. After registering, the new user will can now start using the site's services (joining leagues, market research, etc.) but the user is initially in Tutorial Mode.

Requirements: ST-1, ST-2

UC-2: Creating and Managing League: Allows a player to form a league with customizable settings that suit the users preferences including the league's name. Users can invite up to 16 friends to a game, in which they compete to buy and sell the latest stocks and cryptocurrencies in order to maximize profits. By creating the league, player becomes "League Manager". League Managers choose the total number of players in the league, each player's starting amount of capital, the league mode (industry mode, traditional, etc.), and how the winner is decided (first to reach certain amount of profit, top player on leaderboard after a certain period of time, etc.).

Requirements: ST-6, ST-7

UC-3: Joining a League: Allows a user to participate in various leagues including those set up by strangers and by friends. Users can only join public leagues, but can join a private league with an invitation. Also users are allowed to join multiple leagues at once as. User must join league before initial capital is distributed to every player (essentially before the "season" begins).

Requirements: ST-9

UC-4: Researching Security/Market Data: Allows users to retrieve live data on both stocks and cryptocurrencies, enabling them to make more informed investment decisions. Players can see the current and past price/value of the security, as well as other metrics such P/E ratio, EPS, volume traded, various graphics (tables and graphs), and more.

Requirements: ST-3, ST-10, ST-12

UC-5: Executing a Trade: Allows a user to perform a market trade such as buying or selling a particular stock/cryptocurrency of interest, as well as borrowing stock shares to execute a short sell. The system will automatically check that the user has enough capital to purchase the desired security or enough of the security that the user wants to sell. Trades will be executed at the real-time market prices.

Requirements: ST-8

UC-6: Starting/commenting on an Investment Discussion: Allows users to start a new discussion thread or comment on existing ones in the Investment Forum, regardless of if they are a current player (in an active league) or not. Users will be able to engage in discussions regarding various investment ideas and strategies.

Requirements: ST-16

UC-7: Posting Educational Content: Allows the "Education Content Contributor" and the "Forum Moderator" to post material on investment strategies as well as market trends and developments. Furthermore, the same person will serve as the moderator of the Investment Forum, removing any posts/comments that are deemed inappropriate.

Requirements: ST-17

UC-8: Setting Price/Metric Alerts: Allows a players to track certain securities of interest by setting alerts on them. Player can choose to be sent an email once the criteria of the tracking has been met. For instance, a user might be looking at buying a particular stock but believes that it is currently overvalued based on the current share price and P/E ratio. The player can then set an alert, so that if that stock's share price and/or P/E ratio decreases to a certain level, the player will be notified in case they want to execute a trade on that stock.

Requirements: ST-11, ST-13

UC-9: Competitor Performance: Allows a player to look at the portfolios of other players in the same league, as well as every competitor's ranking on the league leaderboard. When a player clicks on a league that he is a part of, the league leaderboard will be displayed. The user can then

click on any individual competitor's name on the Leaderboard to view that competitor's portfolio.

Requirements: ST-4, ST-5

UC-10: Account Customization: Allows players to customize their profiles and make them unique. Players can go to their "My Profile" tab and choose an avatar. Furthermore, users can connect their account to their social media profiles when registering their accounts. This would enable users to login to Trade Kings through their social media accounts. In addition, users can link their Trade Kings accounts with their Paypal accounts.

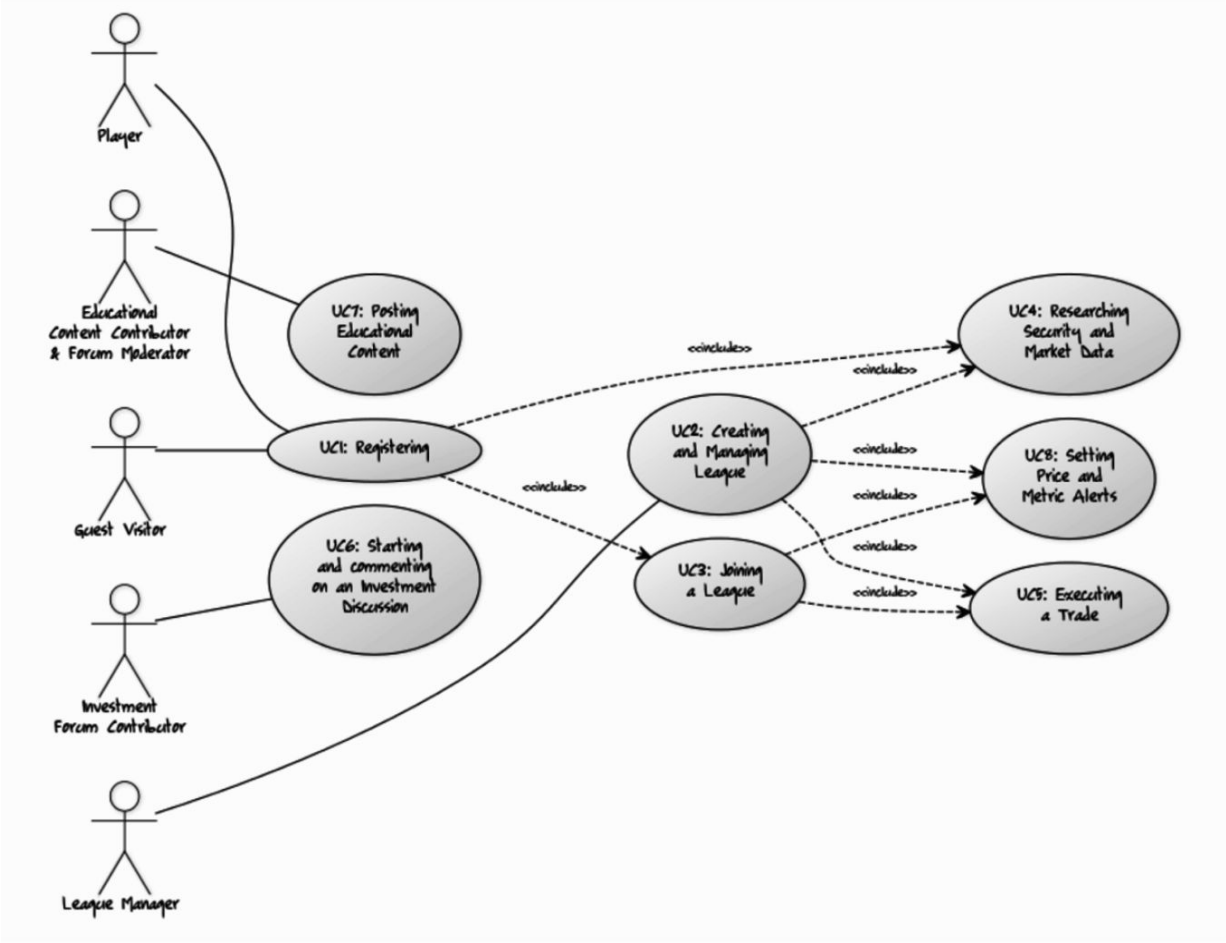
Requirements: ST-14, ST-15, ST-18

UC-11: Viewing Market Forecasts: Allows players to view future prices (up until a certain period in the future) for various stocks and cryptocurrencies. The Forecast page will display a chart with the security's price data, showing the historical values as well as the predicted future prices.

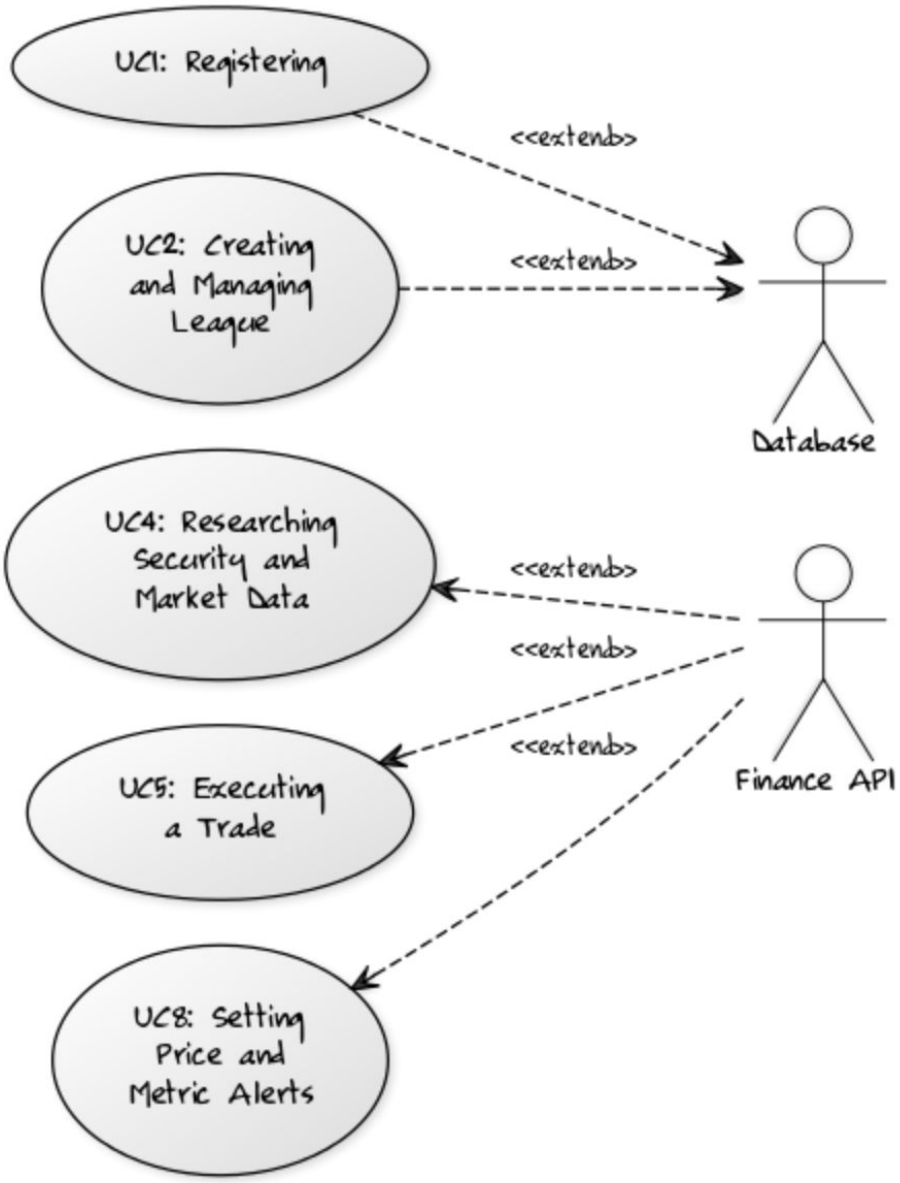
Requirements: ST-19

II. Use Case Diagrams:

Note: Use Case Diagrams 1 and 2 are technically one single diagram, but are simply shown here separately for clarity



Use Case Diagram 1



Use Case Diagram 2

III. Traceability Matrix:

Req't	PW	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11
ST-1	5	X										
ST-2	2	X										
ST-3	4				X							
ST-4	2									X		
ST-5	2									X		
ST-6	2		X									
ST-7	4		X	X								
ST-8	5					X						
ST-9	3			X								
ST-10	4				X							
ST-11	3								X			
ST-12	3				X							
ST-13	2								X			
ST-14	1										X	
ST-15	1										X	
ST-16	3						X	X				
ST-17	3							X				
ST-18	1										X	
ST-19	4											X
Max PW		5	4	4	4	5	3	3	3	2	1	4
Total PW		7	6	7	11	5	3	6	5	4	3	4

IV. Fully-Dressed Description of Use Cases:

Note: Only done for Use Cases 1, 2, 3, 4, 5, 7, and 8 as per instructions, because these use cases were deemed to be the most important and are on track to be finished by Demo 2.

Use Case UC-1:	**Registering**
Related Requirements	ST-1, ST-2
Initiating Actor	Guest Visitor, Player
Actor's Goals	To successfully input information into Trade Kings site and become a registered user.
Participating Actors	User Database, Email Server
Preconditions	Player's email is not already in the system's user database.
Postconditions	The user's information is loaded into the database. User receives confirmation email in inbox.

Flow of events for Main Success Scenario:

→ 1. **Guest Visitor** opens browser and visits the Trade Kings website.
→ 2. **Guest Visitor** enters email and password information to attempt login.
← 3. **System** searches user **Database** with email and password parameters and does not locate a **Player** with matching data.
← 4. **System** prompts **Guest Visitor** to register and loads registration page.
→ 5. **Guest Visitor** fills out registration fields: name, email, age, password, re-enter password, security question 1, and security question 2.
← 6. **System** creates new player entry and adds registration information to the **User Database.**
← 7. **System** tells **mail server** to send registration confirmation email to new **Player.**

Flow of events for Extensions (Alternate Scenarios):

→ 1. **Returning user** opens browser and visits the Trade Kings website.
→ 2. **Returning user** enters email and password information to attempt login.
← 3. **System** searches user **Database** with email and password parameters and is able to locate the **Player** with matching data.
← 4. System loads the **Player's** portfolio on the screen.

Use Case UC-2:	**Creating and Managing League**
Related Requirements	ST-6, ST-7
Initiating Actor	Player
Actor's Goals	Be able to join an existing league or create one's own.
Participating Actors	Players, League Database
Preconditions	Player exists in user database and is currently logged into the Trade Kings website.
Postconditions	League is created with corresponding title and league settings. League entry is added/updated to the League Database with the corresponding information. User Database is updated with league title added under player's League Manager role.

Flow of events for Main Success Scenario (Creating the League):

→ 1. **Guest Visitor** opens browser and visits the Trade Kings website and logs in.
→ 2. **Player** goes to "My Leagues" section of user profile and selects "Create League".
→ 3. **Player** chooses unique title for league and selects league settings.
← 4. **System** creates new league entry and adds league settings to the **League Database.**
← 5. **Systems** adds League Title under **League Manager** field for **Player** in the **User Database.**

Flow of events for Extensions (Alternate Scenarios - Managing the League):

→ 1. **Guest Visitor** opens browser and visits the Trade Kings website and logs in.
→ 2. **Player** goes to "My Leagues" section of user profile and selects "Managed Leagues."
→ 3. **Player** chooses specific league by clicking on the title of the league.
→ 4. **Player** edits the league settings and/or invites new players (if season has not started yet).
← 5. **System** updates data of corresponding league in the **League Database.**

Use Case UC-3:	**Joining a League**
Related Requirements	ST-9, ST-7
Initiating Actor	Player
Actor's Goals	To search and join a desired league
Participating Actors	Database, League Manager
Preconditions	Player is logged in.
Postconditions	One of the player's league slots is filled. Player now has capital in that particular league

Flow of Events for Main Success Scenario:

→ 1. **Player** searches for league via "Join League"
← 2. **System** searches **League Database** for with search parameters of public league and open spots
← 3. **System** outputs a page of joinable leagues
→ 4. **Player** clicks "Join" next to league of interest
→ 5. **Player** joins the league and gains initial amount of capital set by the **League Manager**
← 6. **League Database** is updated with new player's name and information

Flow of Events for Extensions (Alternate Scenario - User Invitation):

← 1. **League Manager** sends an invitation to **Player** to join the league
→ 2. **Player** clicks on "Accept Invitation"
→ 3. **Player** joins the league and gains initial amount of capital set by the **League Manager**
← 4. **League Database** is updated with new player's name and information

Use Case UC-4:	**Researching Security/Market Data**
Related Requirements	ST-3, ST-10, ST-12
Initiating Actor	Player
Actor's Goals	To view current and past s values as well as additional metrics
Participating Actors	Site Administrator, Finance API
Preconditions	Player is logged on to the site with his/her account Player has joined or created a league successfully
Postconditions	Player is more informed about their desired stock(s) and can make an educated decision on whether to buy or sell a stock

Flow of Events for Main Success Scenario (Search via Trade Market):

→ 1. **Player** searches for desired stock or cryptocurrency through "Stock/Cryptocurrency Market"
← 2. **System** outputs a page with a list of stocks and cryptocurrencies
→ 3. **Player** picks the desired stock or cryptocurrency to obtain information on
← 4. **System** displays information obtained from the **Finance API** on the stock or cryptocurrency
→ 5. **Player** analyzes information provided by the site which will include data and news on that stock or cryptocurrency

Flow of Events for Extensions (Alternate Scenarios - Investment Forums):

→ 1. **Player** searches for desired information on Investment Forum
← 2. **System** uses players search as keyword to search forum for related results
← 3. **System** displays acquired results
→ 4**.** **Player** selects forum thread to view and understand outside opinions and analysis on the specific subject desired

Use Case UC-5:	**Executing a Trade**
Related Requirements	ST-8
Initiating Actor	Player
Actor's Goals	To buy or sell a stock/cryptocurrency or borrow (short sell) a stock
Participating Actors	League Database, Finance API
Preconditions	Player must have enough capital to make trade
Postconditions	Trade is stored under Player in League Database and changes in value impact Player's standing

Flow of Events for Main Success Scenario:

→ 1. **Player** searches for a trade via "Stock/Cryptocurrency Market"
← 2. **System** outputs a page with a list of stocks and cryptocurrencies
→ 3. **Player** picks the desired stock to view
← 4. **System** displays information obtained from the **Finance API** on the stock or cryptocurrency
→ 5. **Player** starts buying or selling a desired stock or cryptocurrency if the player has the necessary capital to make the trade
← 6. **System** updates user trade history in the **League Database** with new trade
→ 7. **Player** gets an email (if setup) with trade details

Flow of Events for Extensions (Alternate Scenarios - Via Portfolio):

→ 1. **Player** opens his or her current stock/cryptocurrency portfolio
← 2. **System** searches **League Database** for player's current stocks and cryptocurrencies and displays them
→ 3. **Player** decides on a stock(s) or cryptocurrency he or she wants to buy or sell based on the **player's** available capital
← 4. **System** updates user trade history in the **League Database** with new trade
→ 5. **Player** gets an email (if setup) with trade details

Use Case UC-7:	**Posting Educational Content**
Related Requirements	ST-17, ST-16
Initiating Actor	Educational Content Contributor (ECC)
Actor's Goals	To provide content to site users in order to educate, provide advice, and expose them to investment strategies, understanding market trends, exploring new developments and frontiers, etc.
Participating Actors	Players
Preconditions	No new educational content has been posted on the site
Postconditions	New educational information can be found and analyzed by the users for the purposes of learning and/or making trading decisions Players can add opinions on forum threads

Flow of Events for Main Success Scenario:

→ 1. **ECC** enters site as Forum Moderator (Administrator)
← 2. **System** provides access to **ECC** if correct credentials provided
→ 3. **ECC** starts new forum thread with a post on a topic (strategies, trends, etc.) under the Investment Forum section of the site
← 4. **System** makes the new educational content available to the **Players** for viewing and contributing

Flow of Events for Extensions (Alternate Scenarios - Via News):

→ 1. **ECC** enters site as Forum Moderator (Administrator)
← 2. **System** provides access to **ECC** if correct credentials provided
→ 3. **ECC** takes valuable research (news, magazine articles, etc.) on the various topics and is able to post them on the homepage
← 4. **System** allows the **Players** to access the content that is posted

Use Case UC-11:	**Viewing Market Forecasts**
Related Requirements	ST-19
Initiating Actor	Player
Actor's Goals	To view future, predicted values of a requested (searched) stock or cryptocurrency, as well as the past/historical values of this particular security
Participating Actors	Finance API
Preconditions	Player is logged on to the site with his/her account
Postconditions	Future stock/cryptocurrencies values are displayed on user's screen

Flow of Events for Main Success Scenario:

→ 1. **Player** searches for desired stock or cryptocurrency in search bar on "Market Forecast Page".
← 2. **System** outputs a page with a list of stock and cryptocurrencies
→ 3. **Player** picks the desired stock or cryptocurrency to obtain information on
← 4. **System** displays information obtained from our forecaster on the stock or cryptocurrency

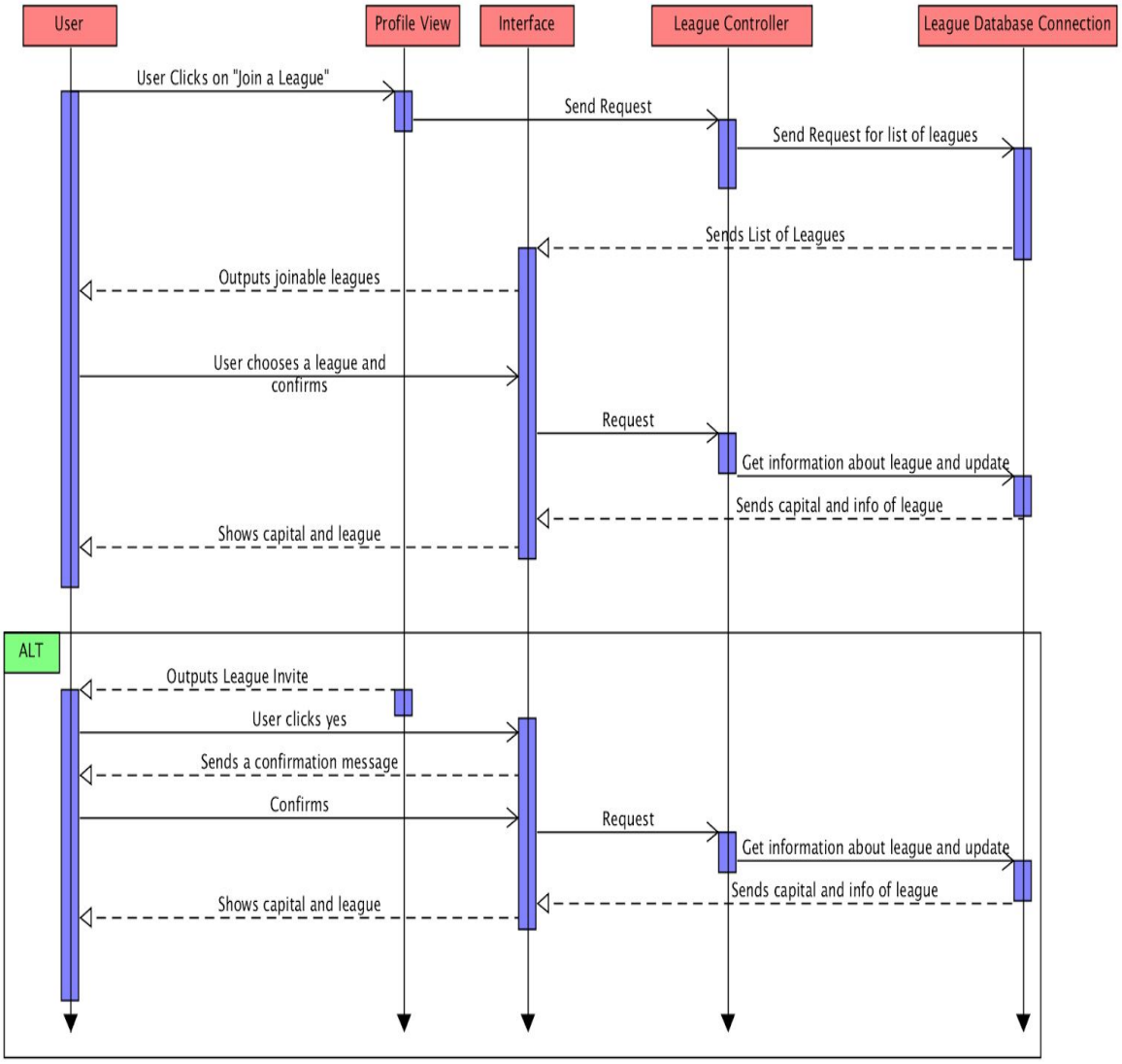
Flow of Events for Extensions (Alternate Scenarios - Investment Forums):

→ 1. **Player** searches for desired stock/cryptocurrency on Investment Forum
← 2. **System** uses players search as keyword to search forum for related results
← 3. **System** displays acquired results
→ 4. **Player** selects forum thread to view and understand other users' projections on the future value of the desired stock/cryptocurrency

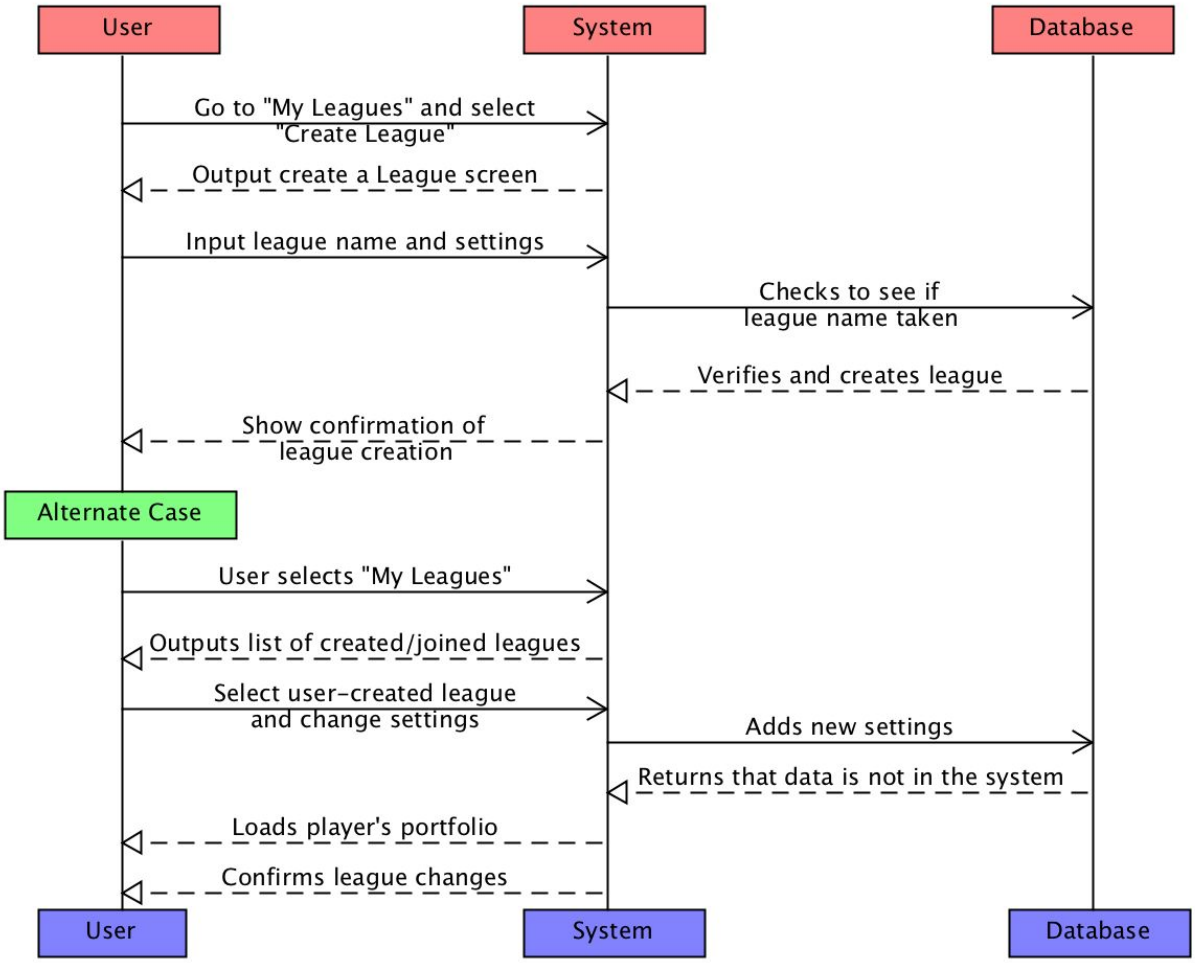
D. System Sequence Diagrams

Note: Only done for Use Cases 1, 2, 3, 4, 5, 7, and 8 as per instructions because these use cases were deemed to be the most important and are on track to be completed by Demo 2.

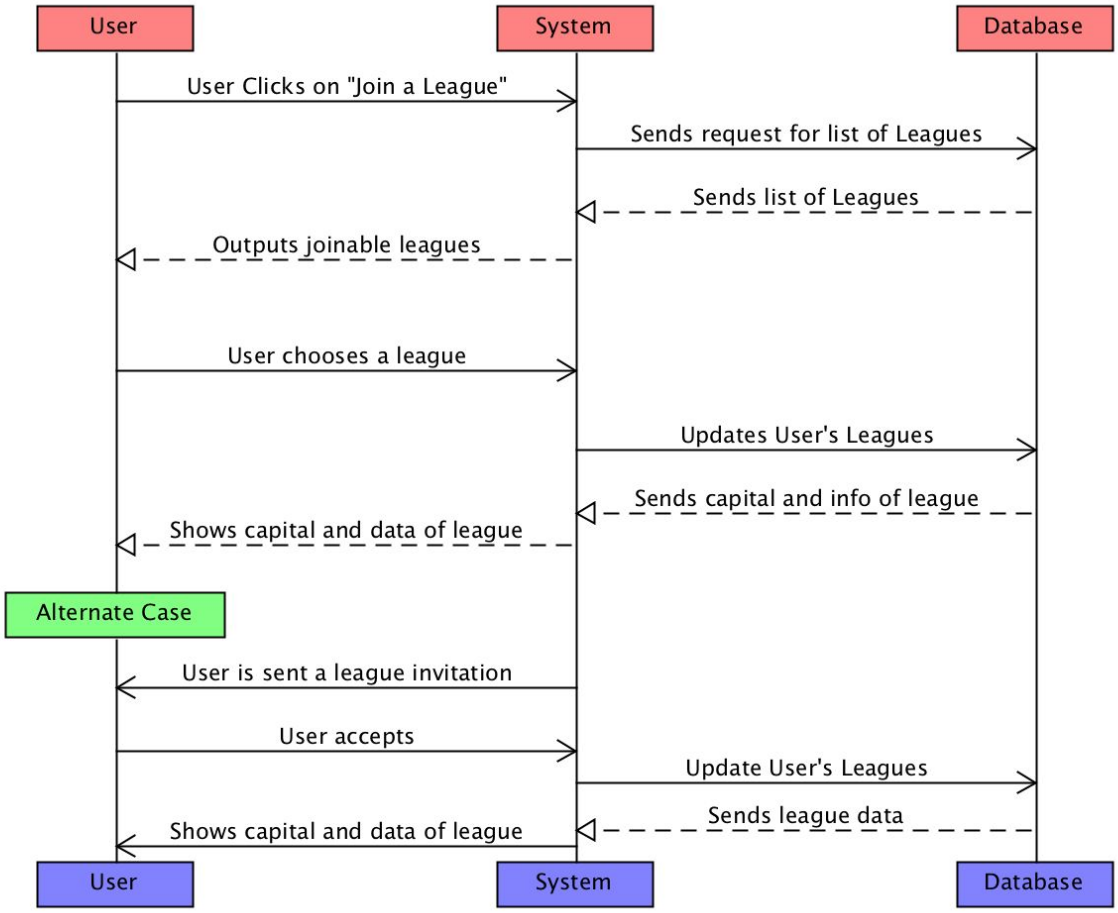
UC-1:



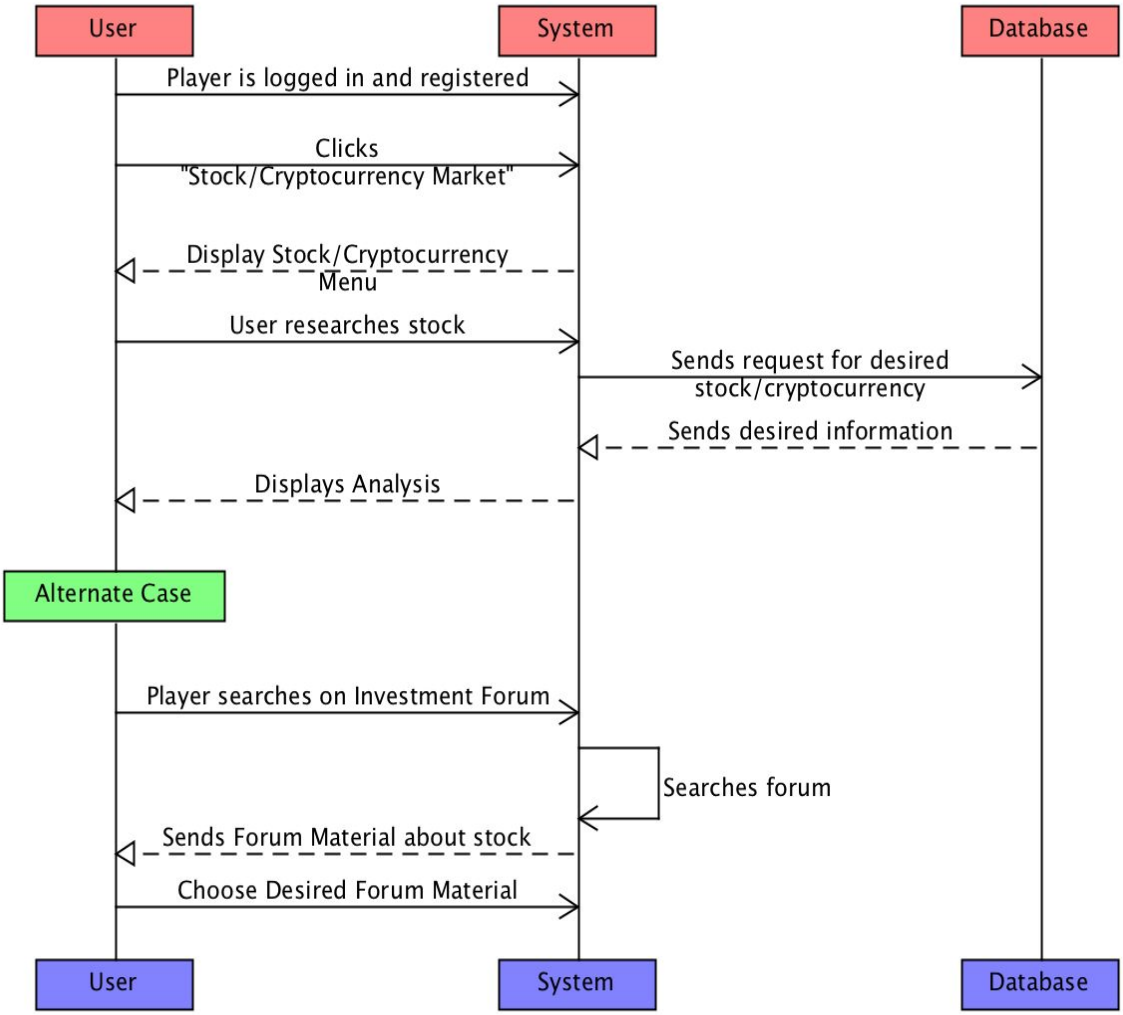
UC-2:


User
System
Database
Go to "My Leagues" and select "Create League"
Output create a League screen
Input league name and settings
Checks to see if league name taken
Verifies and creates league
Show confirmation of league creation
Alternate Case
User selects "My Leagues"
Outputs list of created/joined leagues
Select user-created league and change settings
Adds new settings
Returns that data is not in the system
Loads player's portfolio
Confirms league changes
User
System
Database

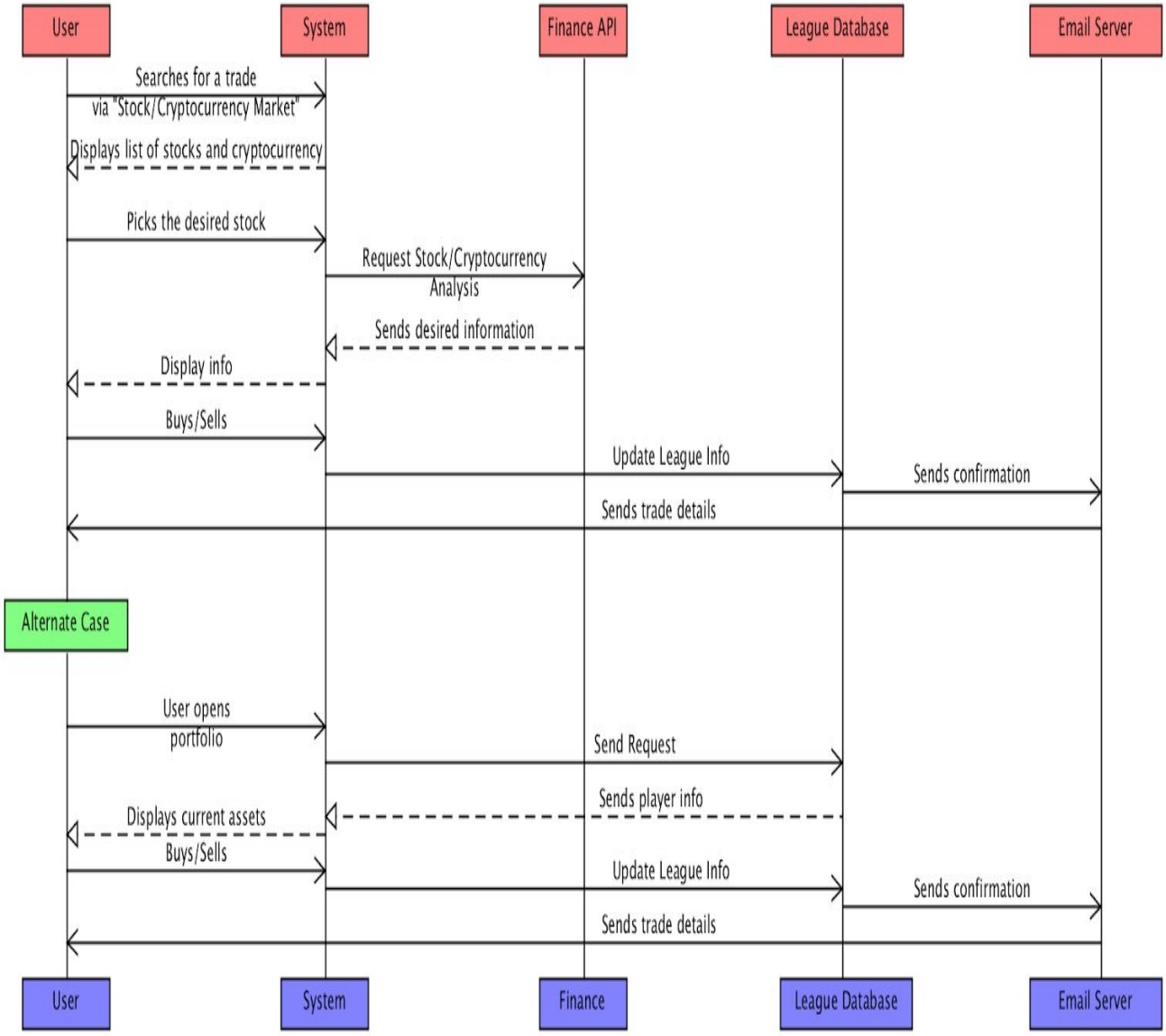
UC-3:


User
System
Database
User Clicks on "Join a League"
Sends request for list of Leagues
Sends list of Leagues
Outputs joinable leagues
User chooses a league
Updates User's Leagues
Sends capital and info of league
Shows capital and data of league
Alternate Case
User is sent a league invitation
User accepts
Update User's Leagues
Sends league data
Shows capital and data of league
User
System
Database

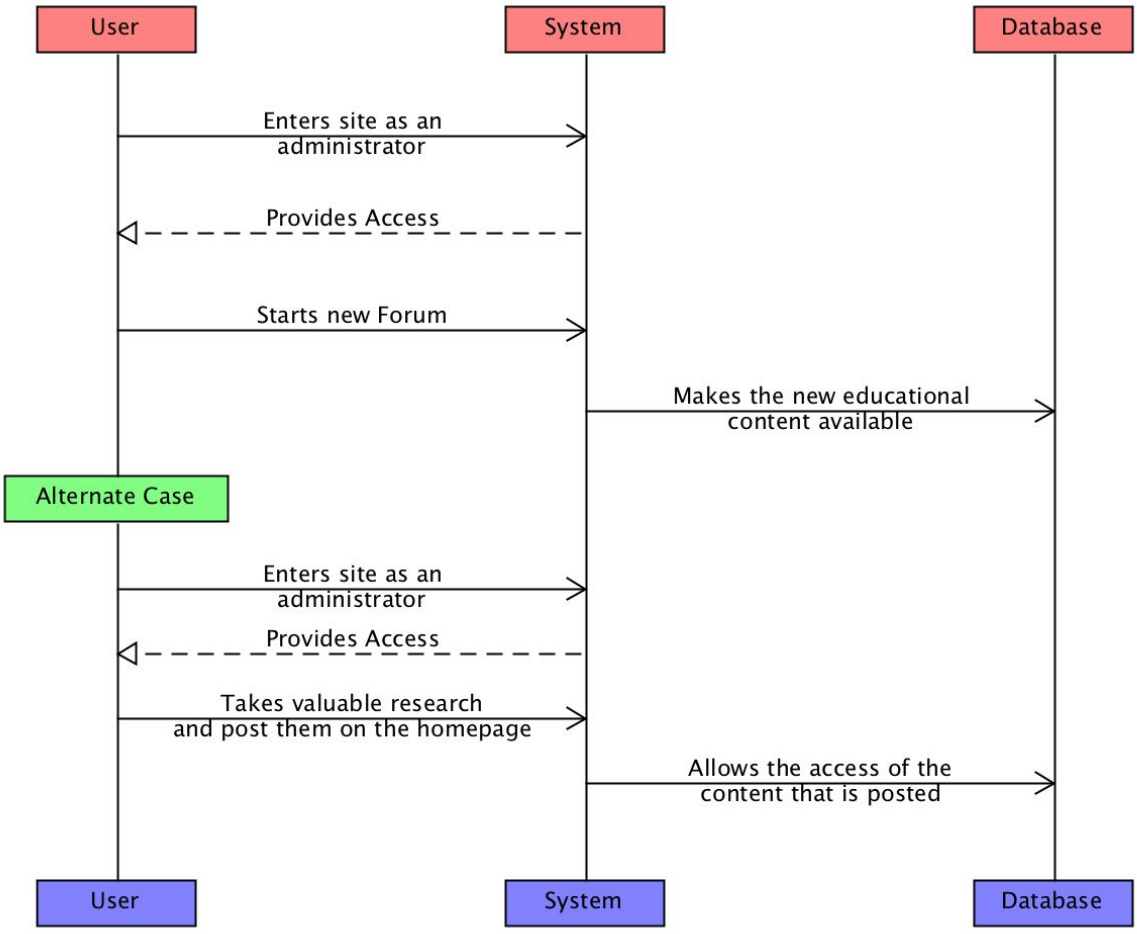
UC-4:


User
System
Database
Player is logged in and registered
Clicks
"Stock/Cryptocurrency Market"
Display Stock/Cryptocurrency
Menu
User researches stock
Sends request for desired
stock/cryptocurrency
Sends desired information
Displays Analysis
Alternate Case
Player searches on Investment Forum
Searches forum
Sends Forum Material about stock
Choose Desired Forum Material
User
System
Database

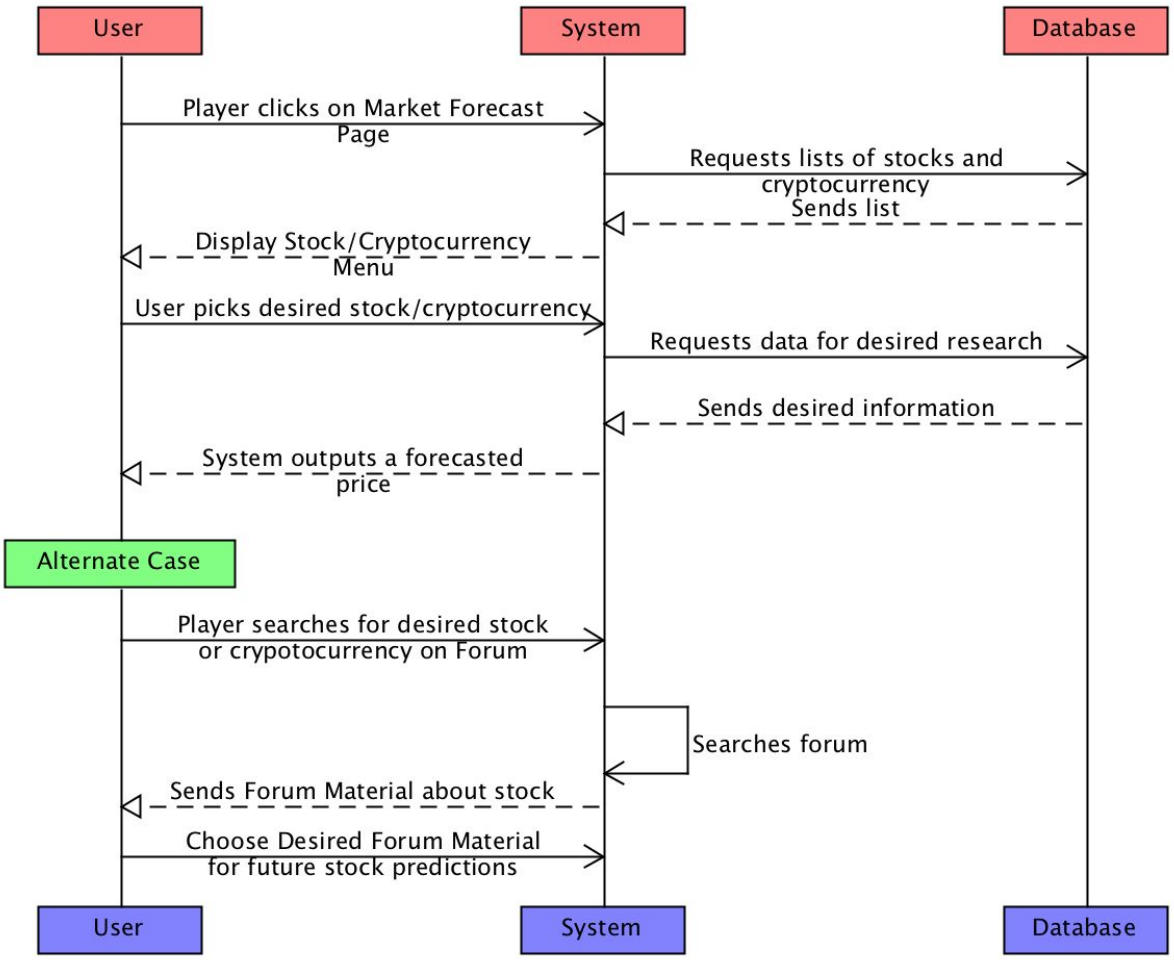
UC-5:


User
System
Finance API
League Database
Email Server
Searches for a trade via "Stock/Cryptocurrency Market"
Displays list of stocks and cryptocurrency
Picks the desired stock
Request Stock/Cryptocurrency Analysis
Sends desired information
Display info
Buys/Sells
Update League Info
Sends confirmation
Sends trade details
Alternate Case
User opens portfolio
Send Request
Sends player info
Displays current assets
Buys/Sells
Update League Info
Sends confirmation
Sends trade details
User
System
Finance
League Database
Email Server

UC-7:


User
System
Database
Enters site as an administrator
Provides Access
Starts new Forum
Makes the new educational content available
Alternate Case
Enters site as an administrator
Provides Access
Takes valuable research and post them on the homepage
Allows the access of the content that is posted
User
System
Database

UC-11:


User
System
Database
Player clicks on Market Forecast Page
Requests lists of stocks and cryptocurrency
Sends list
Display Stock/Cryptocurrency Menu
User picks desired stock/cryptocurrency
Requests data for desired research
Sends desired information
System outputs a forecasted price
Alternate Case
Player searches for desired stock or crypotocurrency on Forum
Searches forum
Sends Forum Material about stock
Choose Desired Forum Material for future stock predictions
User
System
Database

4 User Interface Specification

A. Preliminary Design

Trade Kings user interface will be the primary method of navigating through the website. This UI is specifically designed for simplicity and user convenience. The home page allows the user to execute almost all the actions available from one page including accessing other pages. Whether it is looking at the portfolio, managing the account, or even trading, this is all easily in the reach of the user. Along with this, as soon as the user logs in, the home page will display the information about the user's current stocks as well as the performance of the stocks. This serves as a reminder for every user of how many stocks they have and how well they are doing. At last, the color scheme is thoughtfully designed to make all the important information and options distinct and clearly visible. With black background, and white,green, or red text, the user will not have to look very hard to find something as it will be prominent.

Main Login or Register Page

The preliminary login page can be seen in Figure 4.1. On the bottom of the page, a digital stock ticker tape is displayed to quickly get a glimpse of how some stocks are performing. Users can then see a detailed report for each stock once they login. To login, returning users can simply insert their email address and password . Meanwhile, new users can easily create a new account with their name, email address, password, and two security questions. Another convenient method for logging in is with an existing social media account such as facebook, twitter, or google.

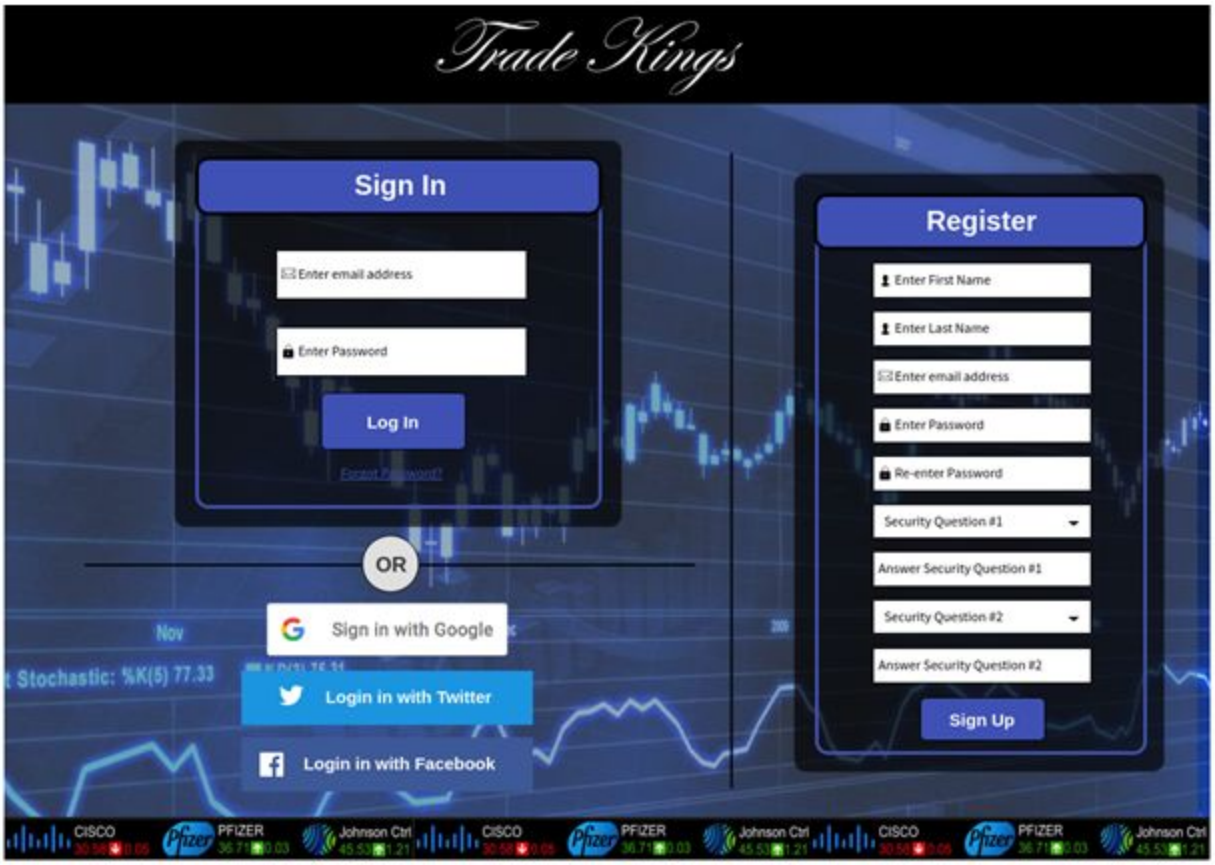


Figure 4.1 - Main Login or Register Page.3.

Home Page

The home page visible in Figure 4.2, displays a leaderboard, showing the users' overall rankings. Along with their rank, it shows each user's account value and how much money they have lost or gained as a percentage of the initial investment. Also on the home page is the header. The header, is in a horizontal container on top, right underneath the title. For consistency and convenience, it will remain the same throughout the website no matter what page the user is on. The header consists of all the options a user will want to use, including My Portfolio, Analyze, and Forum. After clicking an option besides "Home", a submenu will pop up below the menu, displaying more detailed options. The 'Trade Kings' logo will also be existent on all pages on the top of the website and clicking on it, will bring you back to the home page.

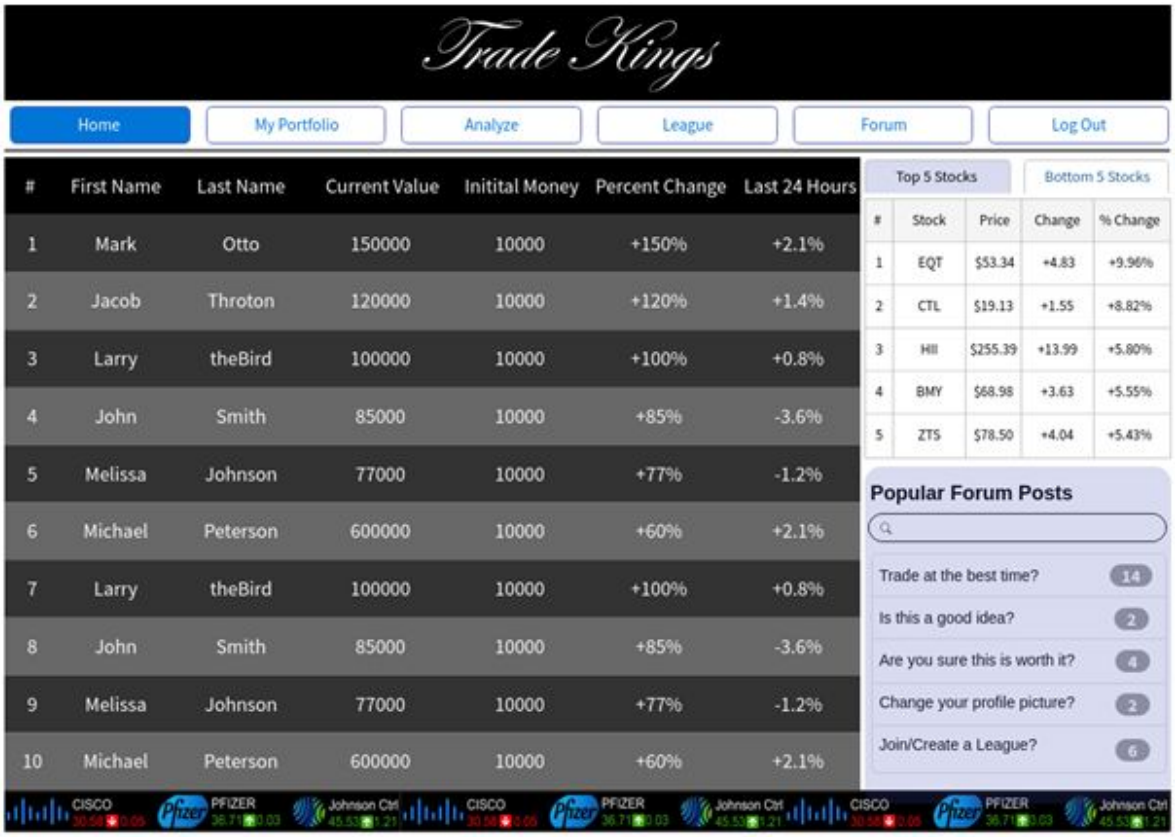


Figure 4.2 - Preliminary design for global header can be seen on top.

My Portfolio

The users portfolio view can be seen in Figure 4.3. A user's username is written on top of the page and his or her personal transactions and stock information is displayed on this page. As of right now, the portfolio will have sub-options of stocks, cryptocurrencies, and transactions history. Along with this, a graph is provided so the user can easily see his or her own performance. This graph will give them an idea of how to improve their rank in the specific league they are in.

Figure 4.3 - Preliminary image of Portfolio

Leagues

In Figure 4.4, users can select between the options of 'Create' or 'Join' in the League settings. If a user creates a league, then he or she will be able to send email invitations to other users and accept other users' requests. When creating a league, the user also has the option to keep the league public or private. Public means that all users can join, and private means users can only join by invitation. If a user selects 'Join', then the user can request to join an open public league which is managed by another user.

Figure 4.4 - Preliminary view of the options given to the user to get in a league.

Analyze

The “Analyze” tool is an option that players can use to research and learn more about their current assets/stocks and use it to decide whether they want to buy new assets. After clicking ‘Analyze’ a submenu will appear and user will have option whether to research on stocks, cryptocurrencies, or look at recent popular trades. While analyzing assets, users can for example lookup a specific stock by ticker. It will then display specific details about the stock such as current price, open and closing price, low and high, P/E ratio, a graph of price history, details of company’s quarterly financials, any related recent news to the company, and finally the status of any related markets. The purpose of this tool is to allow the user to do an in-depth analysis of the stock or cryptocurrency and make an informed trade. This is also facilitated by the “popular trades” option, as the users can get a sense of how the market is currently going. For convenience, users can see how many shares they currently own and can directly click to buy or sell from this page.

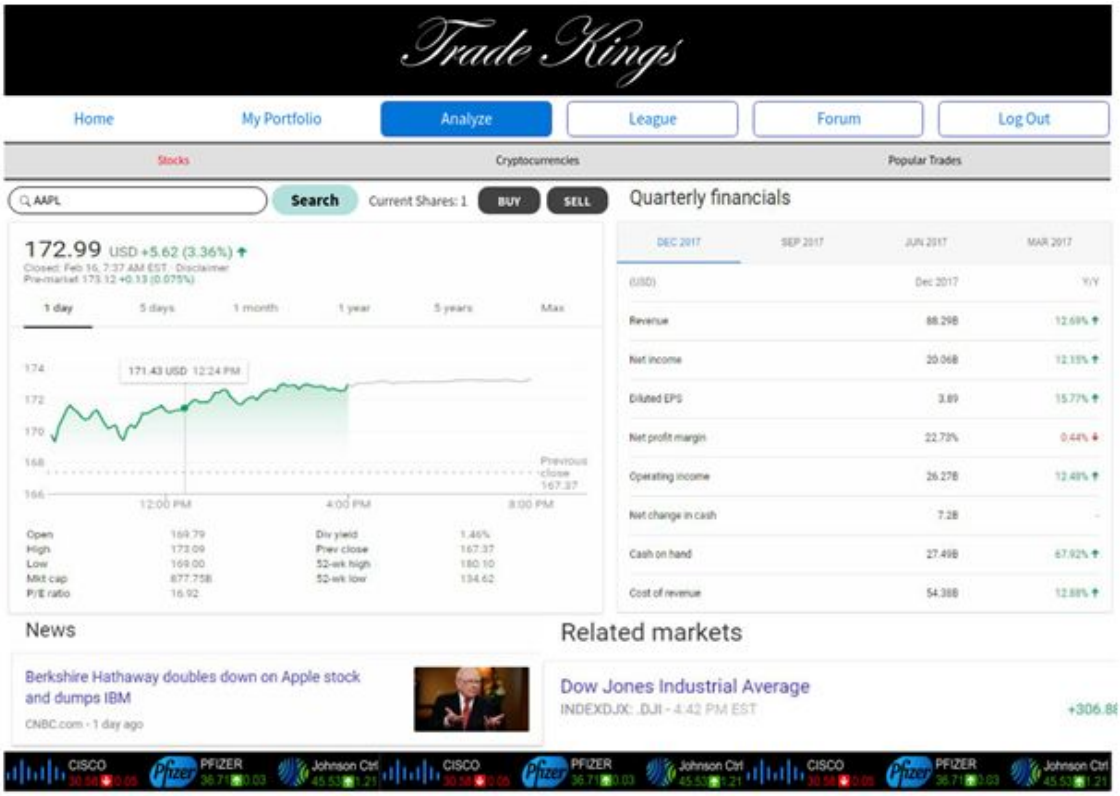


Figure 4.5 - Preliminary view of the Analyze tool

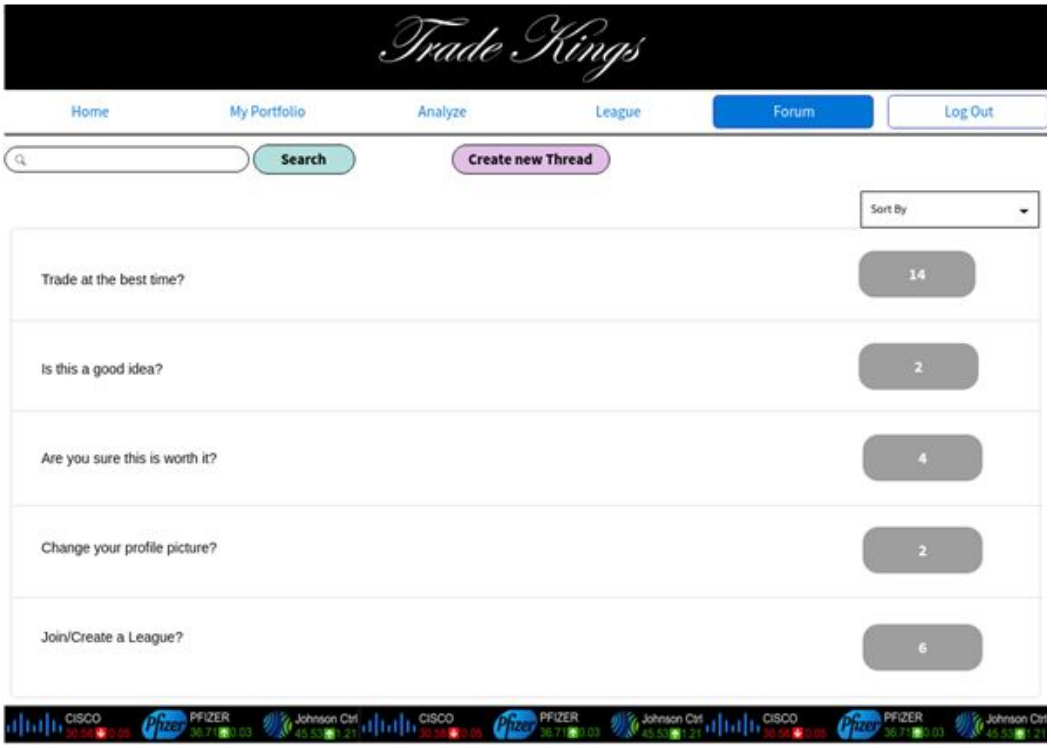


Figure 4.6 - Preliminary view of the Investment Forum

B. User Effort Estimation

User-Interface Navigation vs. Clerical Data Entry Comparison:

Scenario	Number of Clicks	Key input
Login	3	0-79
Register	10	50-80
Trade Stocks	3-4	4-15
Joining a League	2-3	0-20
Creating a League	5-10	20-40
Analyzing Assets	1-2	1-6
View Leaderboard	1	0

Register and Login

Assuming that user visited the website and wants to login if user had already signed up before.

- **Navigation:**
  1. Click on email address and type email id, 1 click and approximately 10 key input.
  2. Click on password and enter your password, 1 click and approximately 10 key input.
  3. Click on Login or hit enter on keyboard, 1 click or 1 key input

Assuming that user has no prior experience with the site and is the first time registering in the site:

- **Data Input:**
  1. Click on first name and type your first name, 1 click and up to 10 key inputs
  2. Click on last name and type your last name, 1 click and up to 10 key inputs
  3. Click on email address and type your email id, 1 click and up to 25 key inputs
  4. Click on password and enter your password, 1 click and up to 8 key inputs.
  5. Click on confirm password and enter your password again, 1 click and up to 8 key inputs.
  6. Click on security questions and select one and enter your answer, 1 click and up to 8 inputs. (Repeat another time for second security question)
  7. Click on Register, 1 click or 1 key input.

Trade Stocks

Assuming that user has logged in and wanted to purchase stock.

- **Navigation:**
    1. Go to menu and click on Analyze, 1 click
    2. Select buy or sell button, 1 click
    3. Click on input box to enter number of shares. 1 click
    4. Click submit to finalize trade, 1 click
- **Data Input:**
    1. Search for stock ticker name, 1-6 key input
    2. Enter Number of shares, 1-7 key input

Joining a League

Given that a user is logged in and wants to join a league:

- **Navigation:**
    1. Click League, 1 click
    2. Click Join, 1 click
    3. After finding league, click on it to choose it and finalize, 2 clicks
- **Data Input:**
    1. Click on league's name or enter its name on search bar, 1 click or up to 20 key input

Creating a League

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

- **Navigation:**
    1. Click League, 1 click
    2. Click 'Create a League', 1 click
    3. After choosing the settings, click submit, 1 click
- **Data Input:**
    1. Create league's name, approximately 20 key input
    2. Enter in settings for the league, 5-7 clicks

Analyzing Asset

Assuming that user is logged in and wants to analyze a particular asset

- **Navigation:**
    1. Click on Analyze, 1 click
- **Data Input:**
    1. Click on search bar and type or use filter settings, 1-3 clicks and/or up to 4 key inputs.
    2. Click on the stock to be researched, 1 click

Viewing Leaderboard

Assuming that user is logged in and is in the main page.

- **Navigation:**
    1. If not on the home page, click “home” 1 click
        a. Otherwise, no clicks required because after log-in, it defaults to home page with leaderboard

5 Effort Estimation using Use Case Points

Unadjusted Actor Weight (UAW)

Actor	**Complexity**	**Weight**
Player	Complex	3 pts
League Manager	Complex	3 pts
Visitor	Complex	3 pts
Site Administrator	Complex	3 pts
Investment Forum Contributor	Complex	3 pts
Educational Content Contributor	Complex	3 pts
Database	Average	2 pts
Finance API	Simple	1 pt
Total		21 pts

Unadjusted Use Case Weight (UUCW)

Use Case	**Complexity**	**Weight**
Registering	Average	10 pts
Creating & Managing League	Average	10 pts
Joining League	Average	10 pts
Researching Security/Market Data	Complex	15 pts
Executing Trade	Complex	15 pts
Posting in Investment Discussion	Simple	5 pts
Posting Educational Content	Simple	5 pts

Setting Price/Metric Alert	Average	10 pts
Competitor Performance	Simple	5 pts
Account Customization	Simple	5 pts
Viewing Market Forecasts	Complex	15 pts
Total		105 pts

Technical Complexity Factors

Factor	Description	Weight	Assessment	Impact
T1	Distributed System	2	3	6
T2	Performance Objectives	2	3	6
T3	End-user efficiency	1	4	4
T4	Complex Processing	1	2	2
T5	Reusable Code	1	1	2
T6	Easy to Install	0.5	0	0
T7	Easy to Use	0.5	4	2
T8	Portable	2	1	2
T9	Easy to Change	1	5	5
T10	Concurrent Use	1	5	5
T11	Security	1	4	4
T12	Access for Third Parties	1	0	0
T13	Training Needs	1	2	2
Total TFactor				40

Environmental Complexity Factors

Factor	Description	Weight	Assessment	Impact

E1	Familiar with development process	1.5	4	6
E2	Application experience	0.5	3	1.5
E3	Object-oriented experience	1	2	2
E4	Lead analyst capability	0.5	3	1.5
E5	Motivation	1	3	3
E6	Stable Requirements	2	2.5	5
E7	Part-time staff	-1	5	-5
E8	Difficult programming language	-1	2.5	-2.5
Total EFactor				11.5

Calculations

UCP = UUCP * TCF * EF

UUCP = UUCW + UAW

UUCP = 105 + 21 = 126

TCF = 0.6 + (0.01 * TFactor) = 0.6 + (0.01 * 40) = 1

EF = 1.4 + (-0.03 * EFactor) = 1.4 + (-0.03 * 11.5) = 1.055

UCP = 126 * 1 * 1.055 = 132.93

Duration = 132.93 * 28 = 3722.04

6 Domain Analysis

A. Domain Model

I. Concept Definitions:

Responsibility	Type	Concept Name
R1. Allow new users to register and existing users to login on homepage. Send confirmation email upon successful registration of new user via Mailing Queue.	D	User Controller
R2. Send data of newly registered users and updated data of existing players to Player Database Connection. Check status of player (whether or not they are a league manager or if they are a member in a league) when enabling player with certain controls of over a particular league.	K	User Controller
R3. Send data of leagues to League Database Connection regarding the settings that have been chosen and/or updated by League Manager.	K	League Controller
R4. Relay data between User Controller and Player Database. Send player data from User Controller to database. Respond to requests from User Controller to check if player is part of certain league and/or if they are the league manager.	D	Player Database Connection
R5. Send league data from League Controller to League Database.	D	League Database Connection
R6. Allows player actions based on league settings by checking league data via the League Database Connection (i.e. player cannot invest in equities if league settings is such that players can only trade cryptocurrencies).	D	League Controller
R7. A “scoreboard” with the rankings of all the players in a particular league.	K	Leaderboard
R8. Pulls real-time stock/cryptocurrency data from Finance API to allow users to research, trade, and track securities.	K	Finance API Adapter
R9. Sends new price/metric alerts (and corresponding data) to the Price Alert Database. System will frequently go through every “active” entry in the database to see if any alert has been triggered.	D	Price Alert Database Connection
R10. Queue requests for triggered price alert emails and registration confirmation emails.	K	Mailing Queue
R11. Record and execute various equity/cryptocurrency trade orders.	D	Trade System

R12. Displays the initial page of the website for users to login or create an account.	K	Home Page View
R13. Displays general information about the user's portfolio and profile.	K	Profile View
R14. Calculates future stock and cryptocurrency price predictions based on past and present data as well as market indicators which can then be used to make investing decisions.	K	Price Predictive Algorithm
R15. Displays future predicted prices based on stock and cryptocurrency forecasting algorithm.	K	Market Forecast View
R16. Allow users to connect social media and Paypal accounts to the website giving the user the ability to login and interact for the purposes of Trade Kings through social media platforms like Facebook and Twitter. Paypal allows users to make real money transactions based on competitive results, which is a big proponent in the fantasy sports universe.	D	Social Media Connection

II. Association Definitions:

Concept pair	**Association description**	**Association name**
Home Page View ↔ User Controller	Once the login/registration form is filled on the Home Page, the information is sent to the User Controller to verify the information and take the next steps accordingly.	conveys login data
User Controller ↔ Player Database Connection	User Controller verifies login data with Player Database Connection or sends data for registering user.	sends, stores
User Controller ↔ Mailing Queue	Upon successful registration of new user, User Controller enqueues a request for a confirmation email to be sent by the Mailing Queue	ends registration data
League Controller ↔ League Database Connection	League Controller add/updates league settings in the League Database via the League Database Connection. League Controller also checks settings stored in database when deciding the allowed actions a player has (i.e. if the league settings are such that players are only able to invest in cryptocurrencies, then the league controller will know this setting and implement this restriction be receiving the settings data from the database).	sends, stores
User Controller ↔ Profile View	Send information needed to display user's account and portfolio.	sends

League View ↔ League Controller	Join and create leagues from “League” page.	stores data
League Controller ↔ Leaderboard	Update leaderboard frequently to list the standings (rank) of every player in any particular league.	stores
Finance API Adapter ↔ Trade System	Updates interface with trades and data that are occuring in the user’s league	stores
Finance API Adapter ↔ Price Alert Database Connection	Update Prices with a Price Alert to the database that will be stored and used by the system for users	sends, stores
Price Alert Database Connection ↔ Mailing Queue	Send EPS, stock price or other stock related details to users via email assuming the price values have reached the target values initially set by the user.	sends
Finance API Adapter ↔ Price Predictive Algorithm	Send stock and cryptocurrency data to the predictive algorithm to provide it with the necessary information to analyze and make an educated prediction of future prices	sends
Price Predictive Algorithm ↔ Market Forecast View	Sends the stock or cryptocurrency prediction algorithm results to be displayed	sends
Social Media Connection ↔ Player Database Connection	Send the user’s social media and/or Paypal accounts credentials to the database to be stored and then requested when required for login or account usage permission or confirmation.	sends, stores

III. Attribute Definitions:

Responsibility	Attribute	Concept
R14. Used to determine the actor’s credentials involving how long they have been in the system. This may determine which data this actor will receive. If the user has registered (isRegisteredUser), but has never logged in (this will be recorded in the database), then when the user first logs in, the site will be in “Tutorial Mode”.	isNewUser	User Controller
R15. Used to determine if the actor has an account in	isRegisteredUser	User Controller

the site's database. If the user's username exists in the Player Database, then isRegistedUser is "true" and the system will check the associated password when allowing the user to log in.		
R16. Needed to determine if an actor is currently engaged in the system.	isLoggedIn	User Controller
R17. Needed to determine if an actor is not currently engaged in the system.	isLoggedOut	User Controller
R18. A method of determining whether an actor is currently in charge of hosting a stock fantasy league.	isLeagueManager	League Controller
R19. A method to determine whether an actor currently resides and is taking part in a stock league.	isInLeague	League Controller
R20. A method to check if user is the Educational Content Contributor. If "isECC" is true, then this user will be given additional privileges such as moderating the investment forum, and updating the "Market Updates" and "Valuation Strategies" tabs.	isECC	User Controller
R21. Allows an actor to check what stocks they currently have in ownership in their portfolio and it is viewable by other actors.	stocksOwned	Trade System
R22. List of current stocks being watched by a user including their EPS, share price and other financial statistics.	stocksTracked	Trade System
R23. Allows an actor to check what stocks they previously have sold in their portfolio and it is viewable by other actors.	stocksSold	Trade System
R24. Describes an actor's current performance relative to other actors participating in various stock fantasy leagues.	rank	Leaderboard
R25. Permission is given by the user to allow Trade Kings to access and verify the user through another account they own on a different social media application	isloggedinSC	User Controller
R26. The Mailing Queue will store the email address of the user that will be contacted when a price/metric	email_address	Mailing Queue

alert is triggered or for registration confirmation emails.		
R27. Displays all the information regarding, league, rankings, stocks, and any other actions that may have been taken within the account	display	Profile View
R28. Every league will have its own “leagueID”. Each league will be stored in the League Database by its leagueID. Whenever the system needs to pull league data from the database via the connection, it will refer to the specific league based on its leagueID.	leagueID	League Database Connection
R29. Every user will have its own “playerID”. Each player will be stored in the Player Database by its playerID. Whenever the system needs to pull league data from the database via the connection, it will refer to the specific league based on its playerID.	playerID	Player Database Connection
R30. Takes user given asset (either a stock or crypto currency) and uses the predictive algorithm which utilizes market data and indicators from the Finance APIs to determine a future price. The results are then displayed in the Market Forecast View.	predictAsset	Price Predictive Algorithm
R31. Lists all the social media accounts a player has linked to the Trade Kings site	linkedSMAccounts	Social Media Connection
R32. Determines whether the player currently has a linked social media account and is logged in. Allows the user to use the social media account through the website.	isLoggedInSM	Social Media Connection

The concept of 'user controller' has many attributes. Once the login or registration form is filled on the home Page, the information is sent to the user controller and the attribute "isNewUser" is set. Then, to check if the user's account is in the database, "isRegisterUser" attribute is set. Also, if the user successfully logs in or logs off, the attributes "isLoggedIn" and "isLoggedOff" are set, respectively. So to summarize, the user controller verifies login data with player database connection and sends data for registering user.

After a user joins and enters the *Trade Kings* league, he or she has the ability to become a manager of a specific league. So, it is very important to have 'league controller' attributes. League Controller adds league settings in the League Database via the League Database Connection and also checks settings stored in database when deciding the allowed actions a player has. So firstly, if an actor currently resides and is taking part in the stock league then

"isInLeague" attribute will be set. On top of that, if an actor is currently in charge of hosting a stock fantasy league, then the "isLeagueManager" attribute is set.

This fantasy league deals with trading and buying stocks so it makes sense to add a concept of 'trade system'. This concept consists of four attributes. First, the "stocksOwned" attribute allows the user to see all the stocks that he or she owns as well as other users in that league. Then if a user want to keep track of a specific stock, then the "stocksTracked" attribute will be turned on. Also, if a user ever sold a stock in the past, then that user along with others in the league are able to see it with the "stocksSold" attribute.

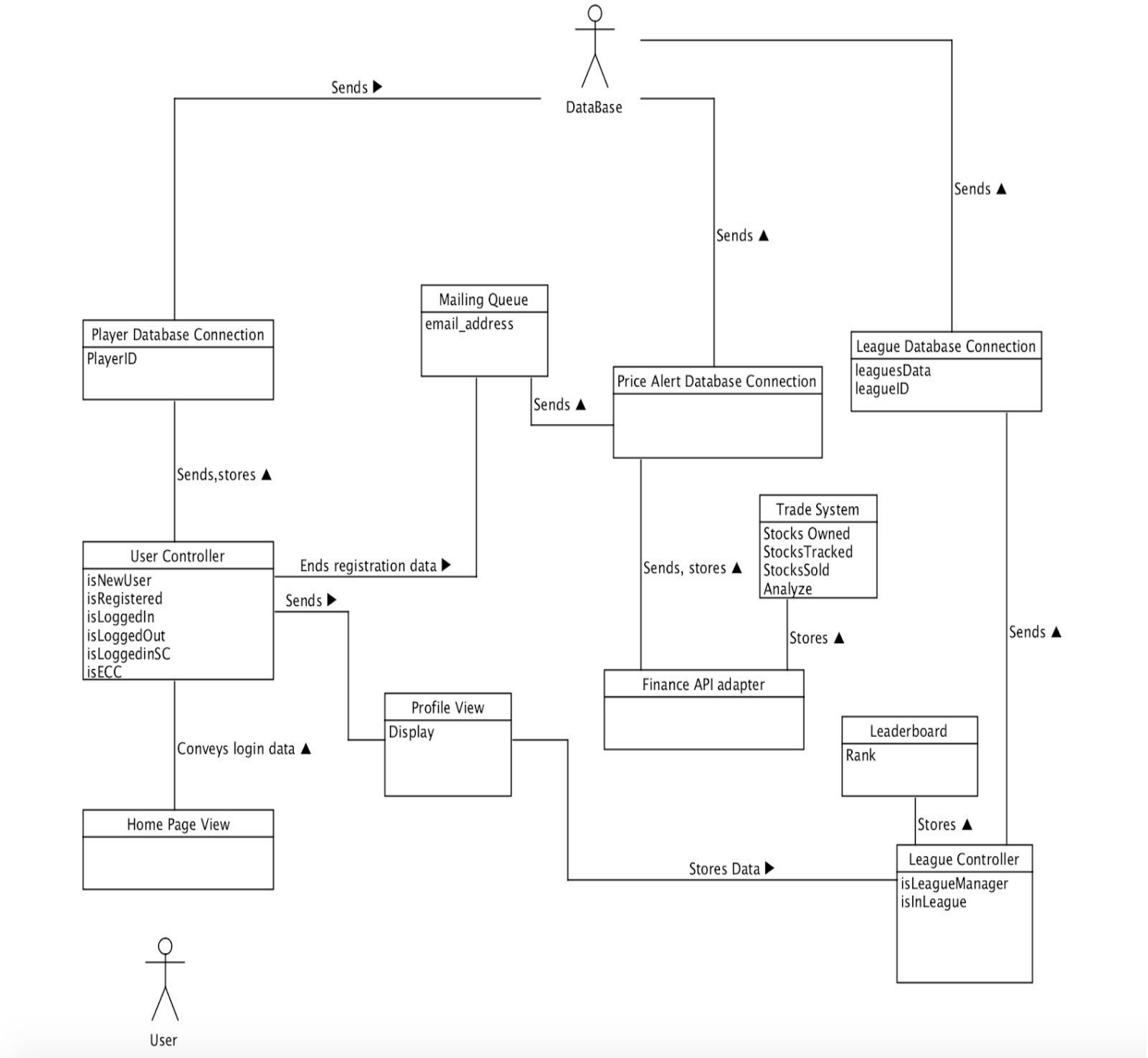
Within Trade Kings the ‘Leader Board’ concept enlists the “rank” attribute to assign individual users a numerical representation of their performance relative to other users. This attribute will allow the user to not only understand how they compare to their peers ,but also know which peers are performing the best. A user can then use this information to talk to their high performing peers in order to get tricks and tips in order to increase their own prominence. Another concept that will be used frequently by players is the 'Mailing Queue' which uses a “mail server” attribute in order to send notifications and alerts to users if for example a stock has reached a certain price. Another, example being a stocks EPS has reached a certain value.

The fantasy league also takes use of the 'Profile View' concept which uses the “display” attribute which displays all the information regarding, league, rankings, and stocks. Also, any other actions that may have been taken within the account are displayed in the Profile View. Essentially, a user can scroll through their profile using this attribute and view all relevant information regarding their stock fantasy leagues. One final concept in use would be the 'League Database Connection' concept that makes full use of the “leaguesData” attribute. The league’s data attributes stores all information that was given by either the manger or invested stocks that deal with either league options or trade investment regulations.

IV. Traceability Matrix:

Domain Concepts									
Use Cases	**PW**	**Account Controller**	**League Controller**	**Order System Controller**	**Login View**	**User Profile View**	**API Adaptor**	**Database Connection**	**Predictive Algorithm**
UC1	7	X			X			X	
UC2	6		X			X		X	
UC3	7		X			X			
UC4	11		X				X	X	X
UC5	5		X	X		X	X	X	
UC6	3					X		X	
UC7	6							X	
UC8	5			X		X	X	X	
UC9	4	X	X				X		
UC10	2	X			X	X		X	
UC11	4						X		X
Max PW		7	11	5	7	7	11	11	11
Total PW		13	28	10	9	28	29	45	15

V. Domain Diagram



B. System Operation Contracts

UC-1 Registering/Creating an Account

- *Preconditions*
    - Guest has no prior registrations to Trade Kings
- *Postconditions*
    - After registration, user data is inputted and stored in the player database
    - After clicking register, user is sent an account verification email
    - Subsequently, user is able to login via email and password

UC-2 Creating and Managing a League

- *Preconditions*
    - User must be registered on the Trade Kings website to continue further
    - Once the user has registered into the system, they will be presented with the option of creating and Managing the league.
    - User will then proceed to My League section and then Create League
    - After selecting the previous tabs, user then selects their Title and setting options for their newly created league
    - User will then proceed to My League section and then Manage Leagues
- *Postconditions*
    - Database is updated
    - My League section is updated

UC-3 Joining a League

- *Preconditions*
    - User has at least one league slot to join a league
    - User is registered with the website
- *Postconditions*
    - Once user joins a league, one of their league slots is taken up
    - User is sent a verification email
    - User's profile is updated with current leagues

UC-4 Researching Security/Market Data

- *Preconditions*
    - User is logged in to Trade Kings with his account
    - User was able to join/create a league
- *Postconditions*

- One of the players who joined the league is planning to buy stocks to make his initial investment

UC-5 Executing a Trade

- *Preconditions*
  - Player must have enough capital in cryptocurrency to make any trade
- *Postconditions*
  - After trading, user data is stored in the database
  - System updates user trade history
  - User receives an email with trade details

UC-7 Posting Educational Content

- *Preconditions*
  - Poster must be either an "Education Content Contributor" or a "Forum Moderator"
- *Postconditions*
  - Investment Forum and Strategies tab is updated with the newly posted content

UC-11 View Market Forecasts

- *Preconditions*
  - User is logged on to the site with his/her account
- *Postconditions*
  - Predicted future stock or cryptocurrencies price is displayed to user

C. Economic and Mathematical Models

Perfect Competition:

One of the most extensive concepts in Economics, the structures of perfect competition requires that no one player of our site will have an advantage over another which exists when has enough resources or power to control the market.

To fulfill the requirements of the Economic model of our project the following needs to be met:

- Players will have access to the same information regarding any stock via the Investment Forum and other educational content provided by the Educational Content Contributor(s).
- None of the players even the league creator can influence the market or the industries involved.
- No extra costs such as the commission of a broker or taxes will be charged when executing a trade.

- There will be an equal starting capital for every player in the league which is set by the League Manager. This ensures that every player starts at the same level and no one has an unfair advantage.

Understanding that these requirements can't be met in the real world is important because certain problems which are not in our control can cause the market to not be a perfect competition.

Stock/Cryptocurrency Pricing & Trading:

The method we use in determining the stock and cryptocurrency prices that the players will use is crucial because in reality the prices for buying the same stock (or cryptocurrency) can vary depending on various factors, but for our purposes every player should have to pay the same to preserve the principle of perfect competition.

- The stock and cryptocurrency prices used for trading will not require any modeling because the prices will be directly obtained from the Finance API (Alpha Vantage) as they change and be used as the trading price.
- When the player attempts to either buy or sell, the price taken from the Finance API will be multiplied by the number shares to determine the **cost** of buying or the **payout** for selling.
    - **cost** or **payout** = (price of stock/cryptocurrency) * (number of shares)
    - In the case of buying, the **cost** determined will be compared to the players capital, and only if the player has enough capital to carry out the transaction will the trade go through.
    - In the case of selling, the **payout** determined will be added to the players capital while the stock or cryptocurrency will be removed from the player's current portfolio.

Determining Rankings:

The rankings of players within a league or in general are determined by figuring out the value of the player's **total assets**. This value is determined by combining the player's currently available capital along with current value of all their holdings in the stock market and cryptocurrency exchange. The value of current holdings is determined by summing the product of each holding with its current price.

Total Assets = current capital + value of current holdings

Once the **total assets** for each player is determined, the rankings can be determined by simply ordering the players by the value of their **total assets** from greatest to least. Since the

market is constantly changing, we will be updating the calculations and these rankings very frequently to make sure the players can see the most accurate evaluation of their ranking.

7 Interaction Diagrams

Introduction:

Trade Kings has various actors in action to create a system which meets the desired functional requirements and customer problem statement. As described in further detail in the Functional Requirements Specification, the major actors are:

- Player
- League Manager
- Guest Visitor
- Site Administrator
- Finance API
- Investment Forum Contributor
- Educational Content Contributor/Forum Moderator
- Database

Together, they interact to ensure all the use cases are achieved by the site. In order to visualize and better understand the key use cases of Trade Kings, the following UML interaction diagrams are provided. First, there is the System Sequence Diagrams which are shown for the most crucial use cases. These sequence diagrams describe the timeline of interactions between each of the objects involved in order to carry out the functionality. The vertical aspect of the diagram is representative of time, while the horizontal aspect describes both which objects are communicating and what they are communicating.

For the majority of the use cases, the diagrams outline how different types of users interact with the User Interface, Finance API, and Database. The use cases described are:

1. Registering/Logging In
2. Creating and Managing League
3. Joining a League
4. Researching Security/Market Data
5. Executing a Trade
6. Posting Educational Content

These six use cases are required for the site to be functioning with the basic standards that have been described. The diagrams of each case show both the main and possible alternate success scenarios for the use cases. Each diagram is followed with a caption, further detailing the use case and its corresponding diagram.

UC1 - Registering

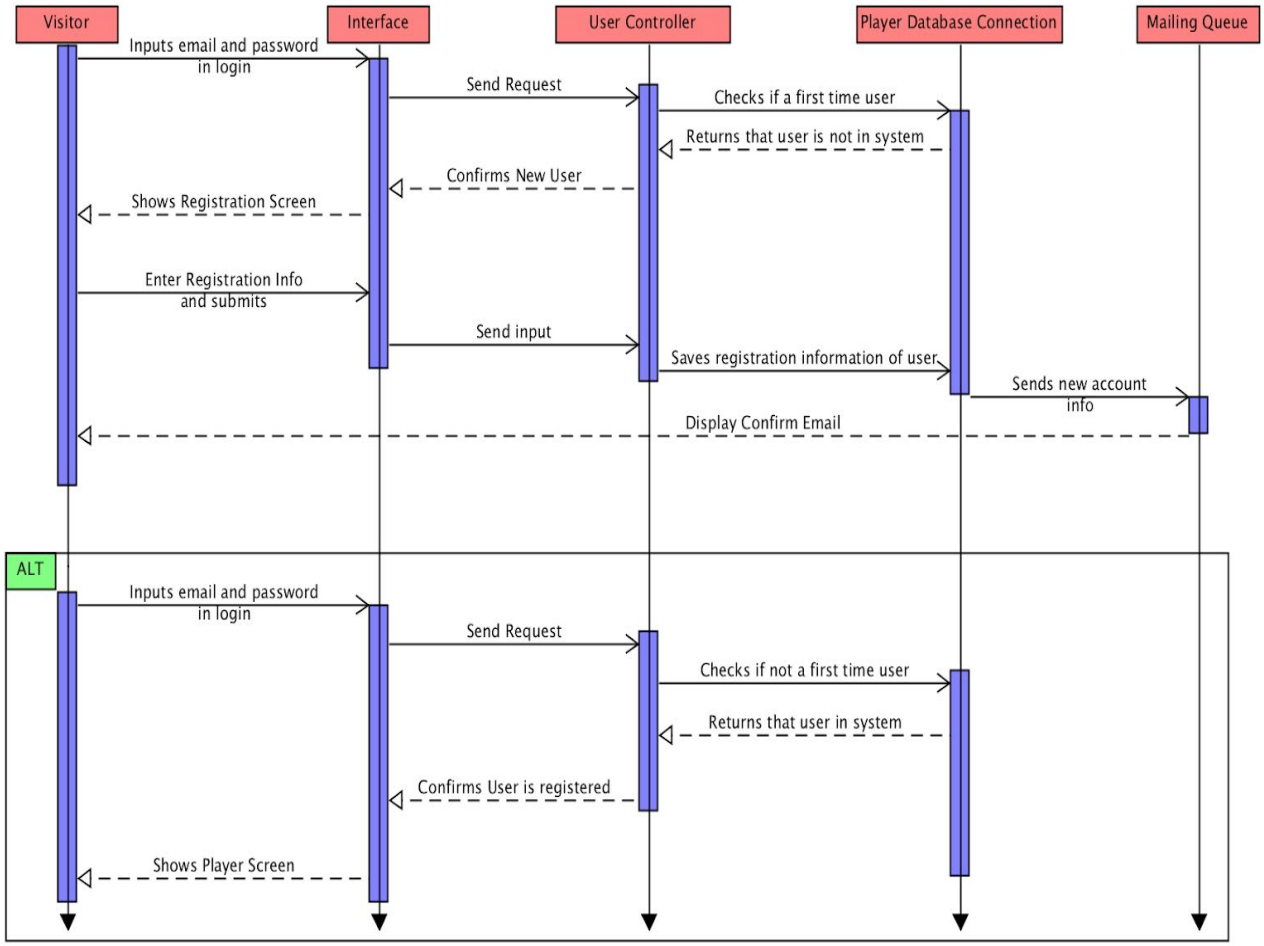


Figure 5.1A (Original)

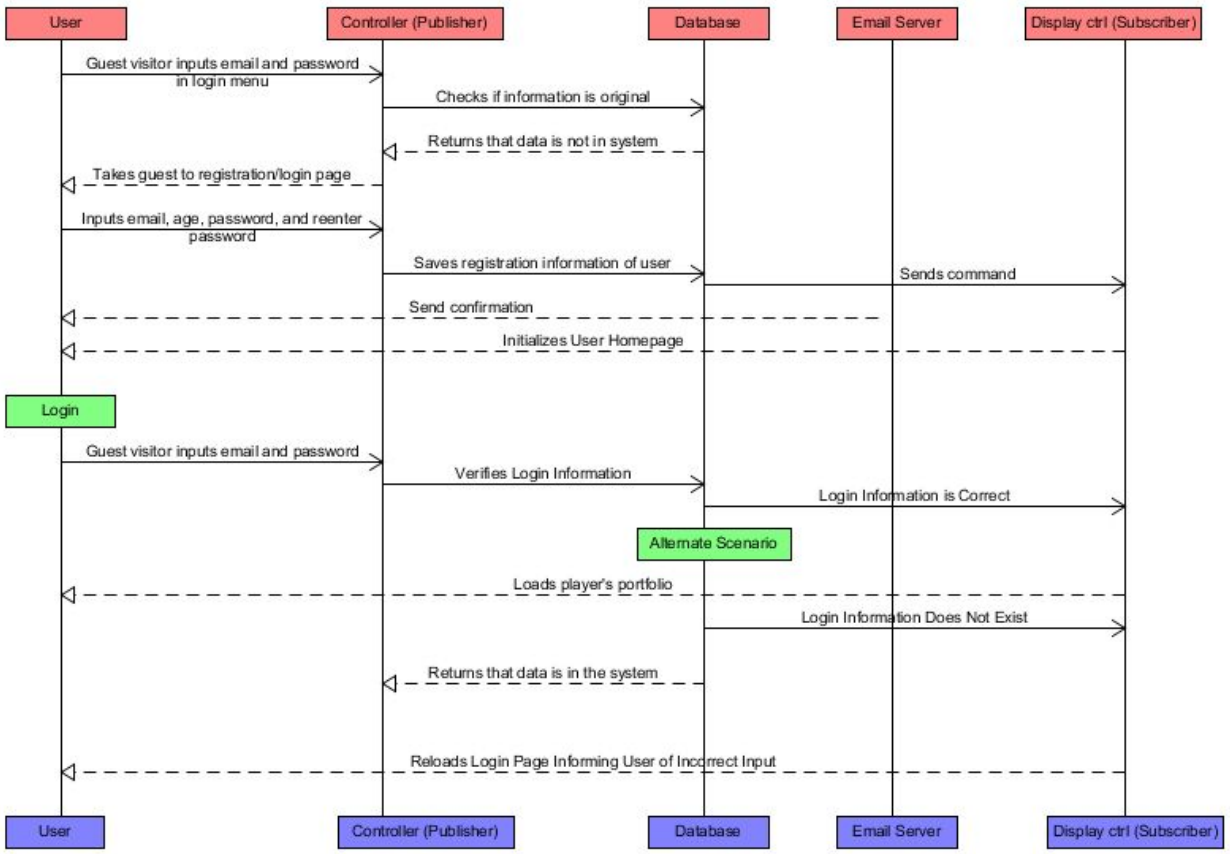


Figure 5.1B (Design Pattern Implemented)

The above diagrams is for Use Case 1- Registering. The visitor first gets on the home page and clicks “Create Account”. The user interface then sends a request to the user controller. Next, the user controller contacts the player database to see if the new user request is valid. The player database confirms that the user is indeed a new user. The user controller tells the interface to prompt the user to the registration page. The user inputs their email, password, email, security questions, and date of birth. The user is prompted to confirm the information. Once confirmed, the user controller and player database update and tell the mailing queue to send a confirmation email to the user that registration was successful.

In an alternate scenario, suppose a returning user tries to login. The user enters a password which is invalidated by the database. So the user controller asks user if they “Forgot Password?”. User continues process to recover account by entering their email and answering the original security questions. Once answers are verified with the database, user is prompted to set a new password. The database then gets updated with the user’s new password.

In this example, we see the expert doer principle being applied because the chain of communication is being shortened. For example, as the email system receives the new account information, it directly sends the confirmation email to the visitor to view which skips all the objects connected in between. The application of the expert doer principle also results in the the application of the lower coupling principle because the number of associations (connections) is decreased.

The second diagram above (Figure 5.1B) implements the Publisher-Subscriber design pattern in the diagram to show how the registration functionality works. The controller takes input from the user and publishes to the database and the display control which then communicate with the email server and back to the user respectively. The database is used to store and confirm credential while the display subscribes to the necessary information for the user to see.

UC2 - Creating and Managing League

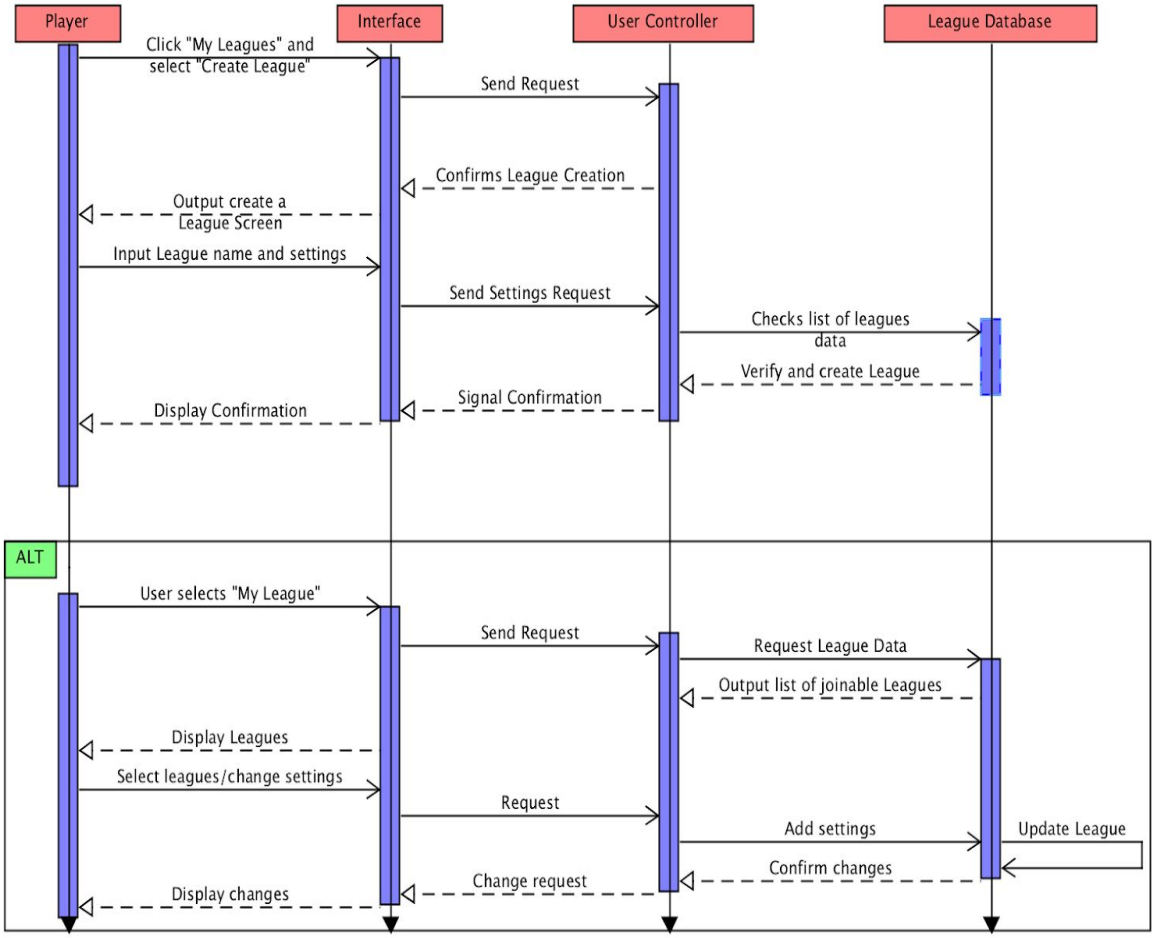


Figure 5.2

For this use case a player has two options: create a new league from scratch or modify the settings of an existing league. Based on these conditions, the user interacts with the user interface and indirectly with the user control and player database in order to store the league along with its settings in the database. Basically, this case deals with a player modifying their league to fit their current expectations. The database is the most significant domain in this case. Initially, the user selects to create league or

modify an existing one through the user interface. The requests are sent to the user control and upon a successful request the database updates the league information or creates new league entries. Eventually, the database sends a signal back to the user controller to the user interface showing a successful output message to user.

As depicted in Fig. 5.2, it is shown that high cohesion principle is implemented. Take for example the alternate scenario where the user attempts to modify the settings of an existing league. Then, the diagram shows how the work of changing the league settings is distributed across different objects. First, the interface receives input from the user, then it's passed to the user controller, which requests data from the database. Then the database sends information back which eventually gets back up to the user in the opposite order of how it was received. Thus, we see how each object communicates with other objects efficiently so that they all can work in cohesion.

UC3 - Joining a League

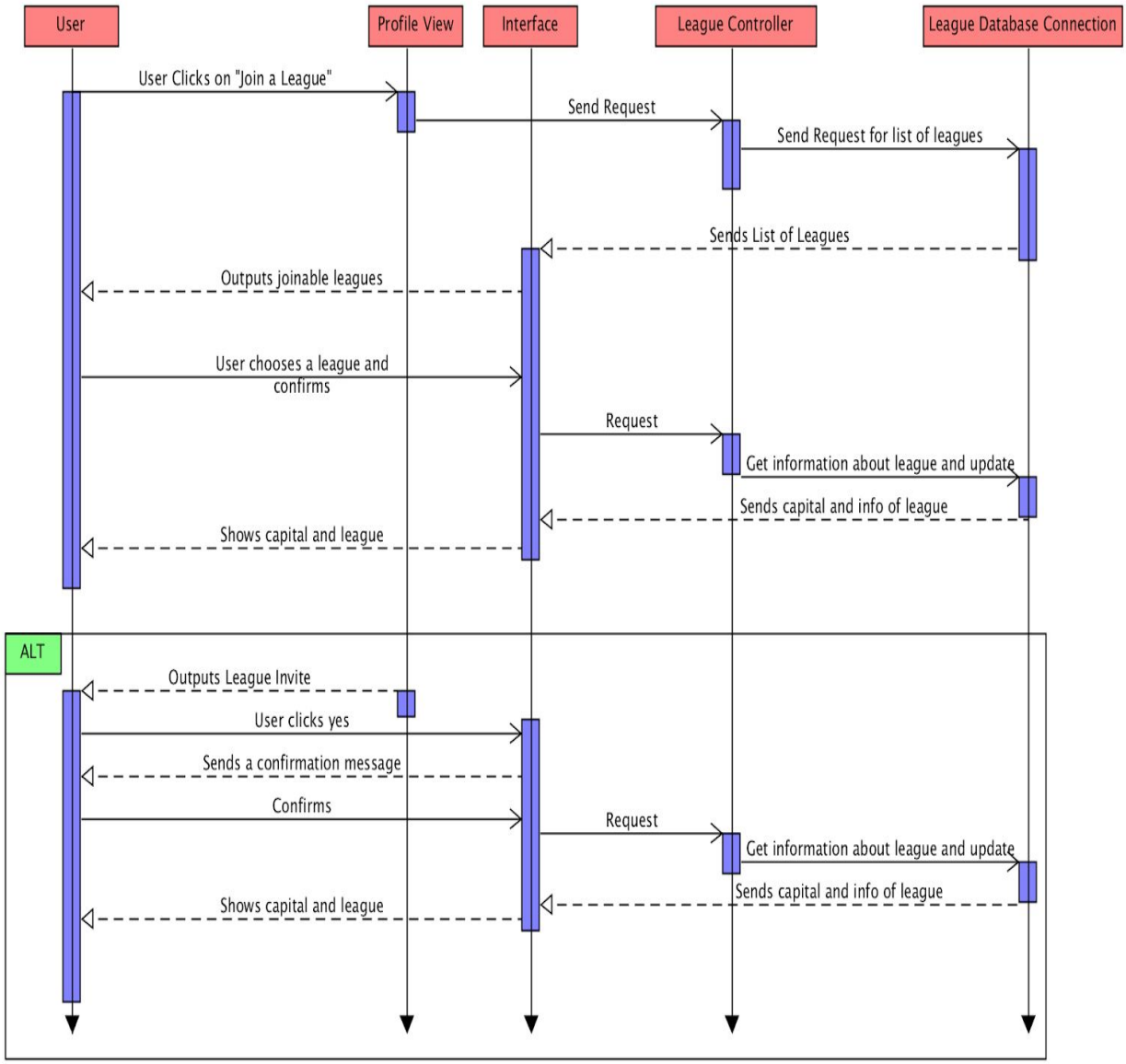


Figure 5.3A (ORIGINAL)

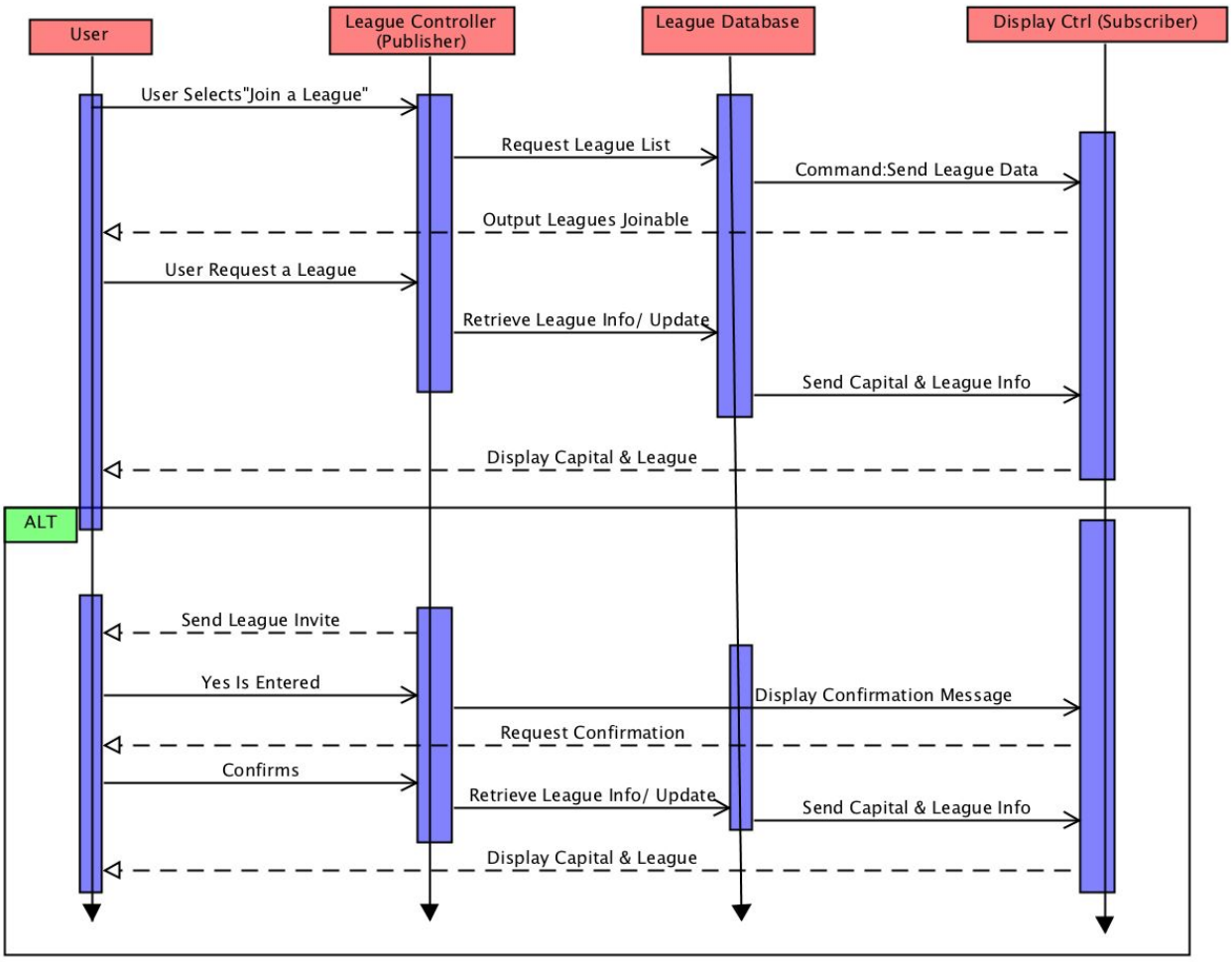


Figure 5.3B (Design Pattern Implemented)

The above diagram is for Use Case 3 (Joining a League). In the main scenario, the user's first action is to click "Join a League" in the profile view. The profile view then sends a request to the league controller. The league controller searches the league database via the league database connection and checks to find which leagues have empty slots for joining (based on the capacity value of each league). Once the joinable leagues are found, the league controller sends the league data to the user interface. The interface outputs a list of all of the joinable leagues. Once the user chooses a league, the interface asks for a confirmation. Once confirmed, the interface sends the request to the league controller. The league controller is updated and sends a request to the league database about the league's information. The league database is updated (new player's playerID is added to the proper leagueID) sends the information to the interface for the user to view.

In the alternate scenario, instead of the user clicking to join a particular league, the user can be sent a request by a League Manager and can join that league by accepting the request. First, the user

receives an invite from the League Manager to their interface page. When the user clicks “accept invitation”, the league controller updates the League Database by adding the player (playerID) to that league (leagueID). The League Database then sends back the settings of that league to the League Control, which displays the Join Confirmation and League Settings to the user’s interface page.

The design principles that we employed when assigning responsibilities to the objects are Expert Doer Principle and Low Coupling Principle. We used the Expert Doer Principle because here, the League Controller is central to the important actions. It handles the transfer of information from the user to the League Database, because it is most suited to act upon that information. Furthermore, Low Coupling Principle is employed because there is not any particular object in the system that is sending too many messages (taking on too many communication responsibilities).

The second diagram used (Figure 5.3B) implements the Publisher-Subscriber design pattern in the diagram to show how joining a league functionality is implemented. To start off the user initiates all commands. The league controller then waits for user input. Once input is retrieves such as a request to join a league the information is sent to the database which then outputs corresponding joinable leagues. This database works with the display controller ,the subscriber, in order to send the user the appropriate request of the list of leagues a user may join. If a user selects to join a league the corresponding database will be updated. In this way the user interacts directly with league controller that sends request back and forth from the league database. The database as well as the controller interact with to the display subscriber which retrieves information as commands. As a side note the profile view object was removed as its functionality seemed to be merged with the interface into the display subscriber in order to prevent redundancy.

UC4 - Researching Security/Market Data

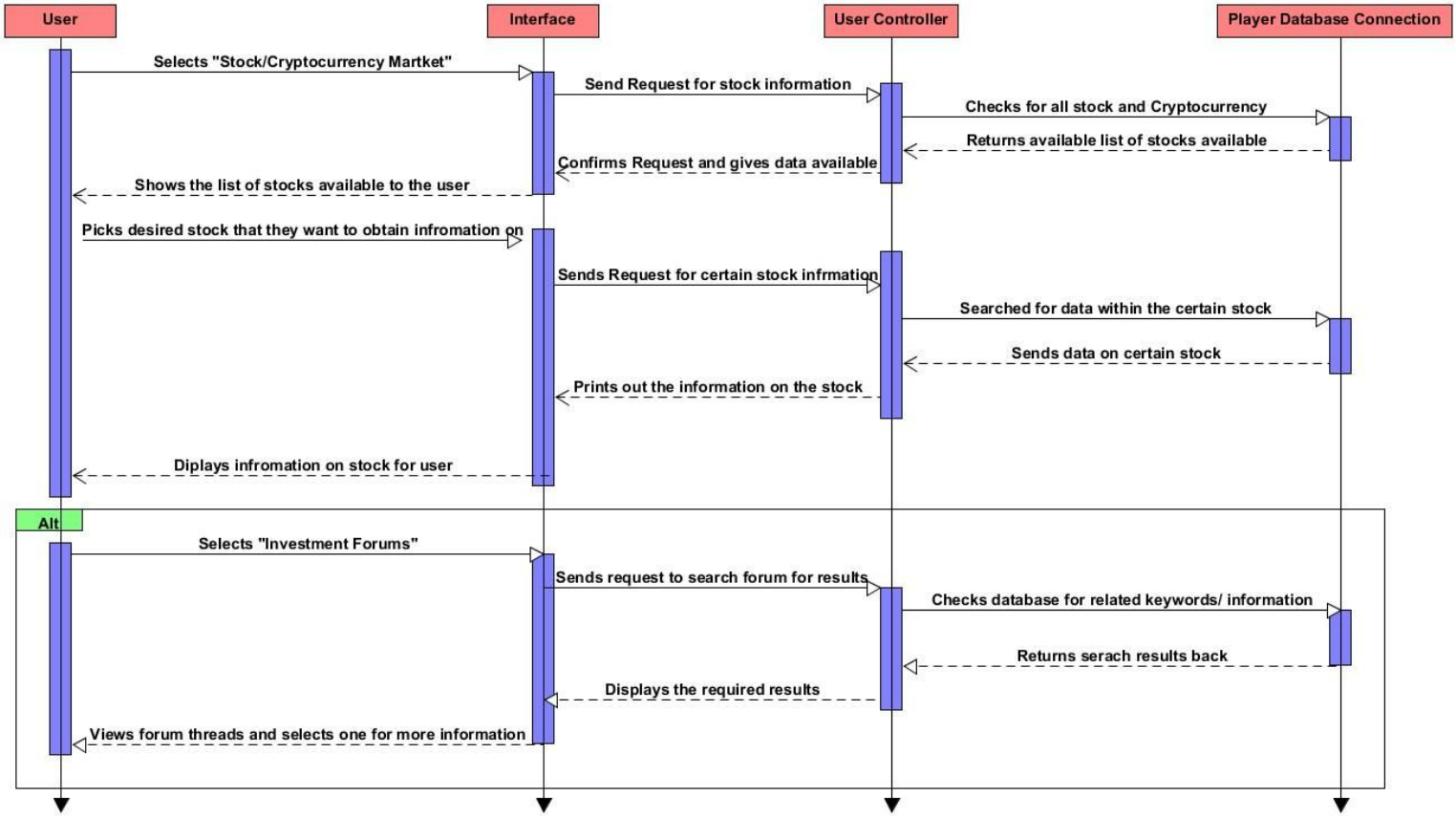


Figure 5.4

The diagram seen above is for Use Case 4. In this case, the user is trying to gain information on a certain stock that they want. For this, the user will select “Stock/Cryptocurrency Market”. From there, the user interface contacts the user controller. It informs that the user wants to be informed on available stocks. The User controller goes through the database for the website and selects all data given about stocks that can be viewed. The user controller shares the data with the interface which displays it to the user. The User then has the option to see further information on stocks they select. The interface once again informs the User controller that the user has a request to see certain information on a selected stock. The User controller goes into the database and obtains all the information stored on the selected stock. This data is handed off to the interface where it prints out the data in a appropriate way for the user. The user obtains the information and has the option to repeat the process for another stock. The user is also given a alternative option where they request for “Investment Forums”. From here the user can enter a keyword that will give them results of stocks, prices, and data that could be related to that keyword. This is just another way users can gain access to stock information through Trade Kings. This is another method that users can use to gain insight on investment strategies and market performance.

This use case implements high cohesion principle. When the user tries to research on a particular stock, each object communicates with each other until it receives the requested data from the database. Then the objects return the data back to the user. However, since each object simply communicates with another object to get the requested information, they are in high cohesion as they don’t take responsibility of computing anything. The only object that does computation is the database which requests a query, and returns the resulting information. All other objects simply communicate with each other to execute requests and return data.

UC5 - Executing a Trade

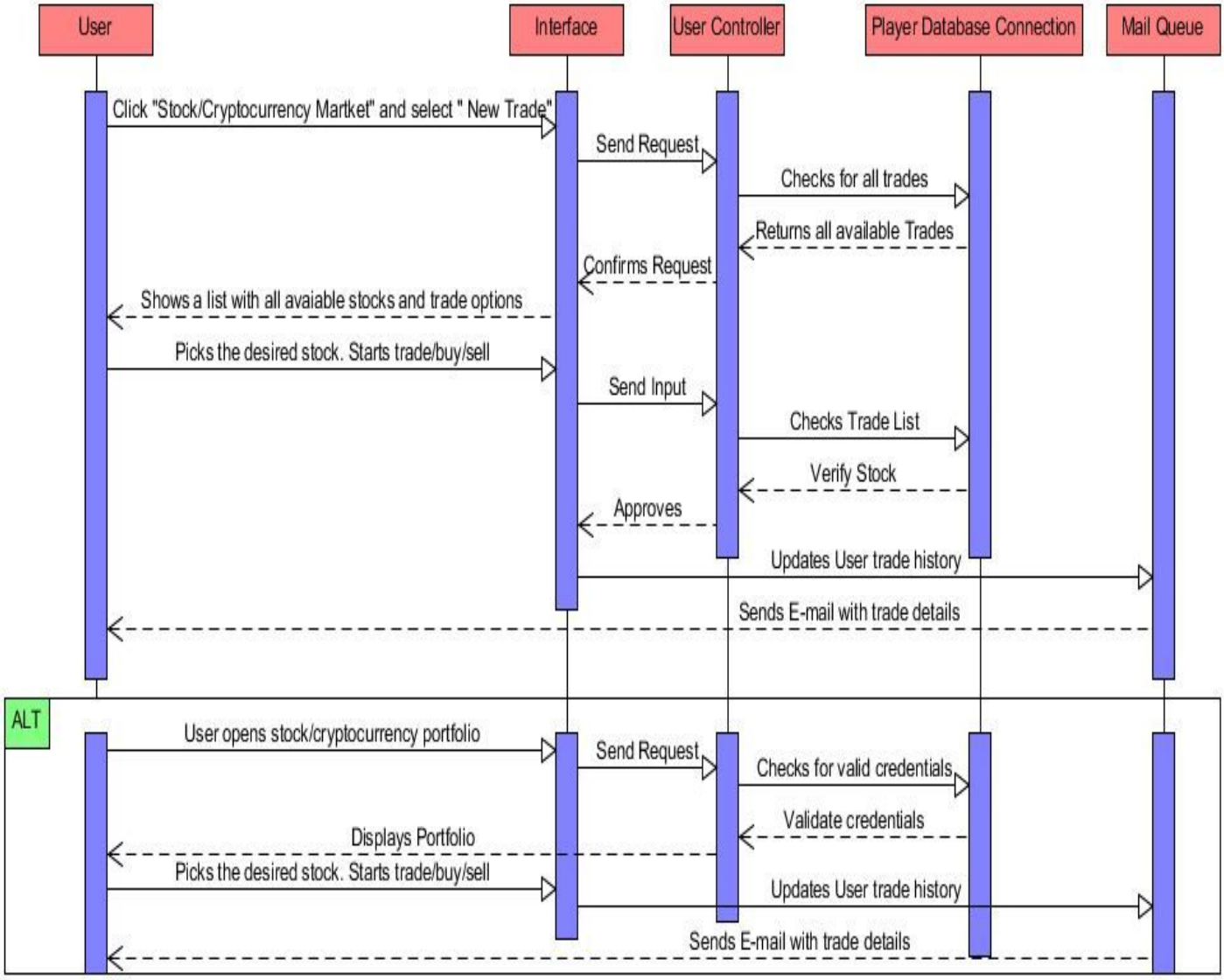


Figure 5.5

The above diagram is for Use Case 5 (Executing a Trade). The user's first action is to click on "Stock/Cryptocurrency Market" in order to search the market for a new trade. Then, the Interface outputs a page with a list of stocks and cryptocurrencies. While the Controller checks and verifies everything, the user picks the desired stock to view as the Interface displays information obtained from the Database on the stock or cryptocurrency. The user starts buying or selling a desired stock or cryptocurrency if the player has the necessary capital to make the trade. The league controller is updated and sends a request to the league database about the league's information. The league database sends the information to the interface for the user to view. Finally, User gets an email with trade details.

The alternate scenario begins with the user opening his/her portfolio. The database will obtain user portfolio information and output to the interface. The user can decide to trade directly from the

portfolio (select owned equity to buy or sell more of). If the user's capital meets the requirements, then the trade will go through, the database will be updated and the confirmation email is sent.

Here we see the high cohesion principle being used because the workloads is balanced across the objects as witnessed by the amount of actions each one takes upon itself.

UC7 - Posting Educational Content

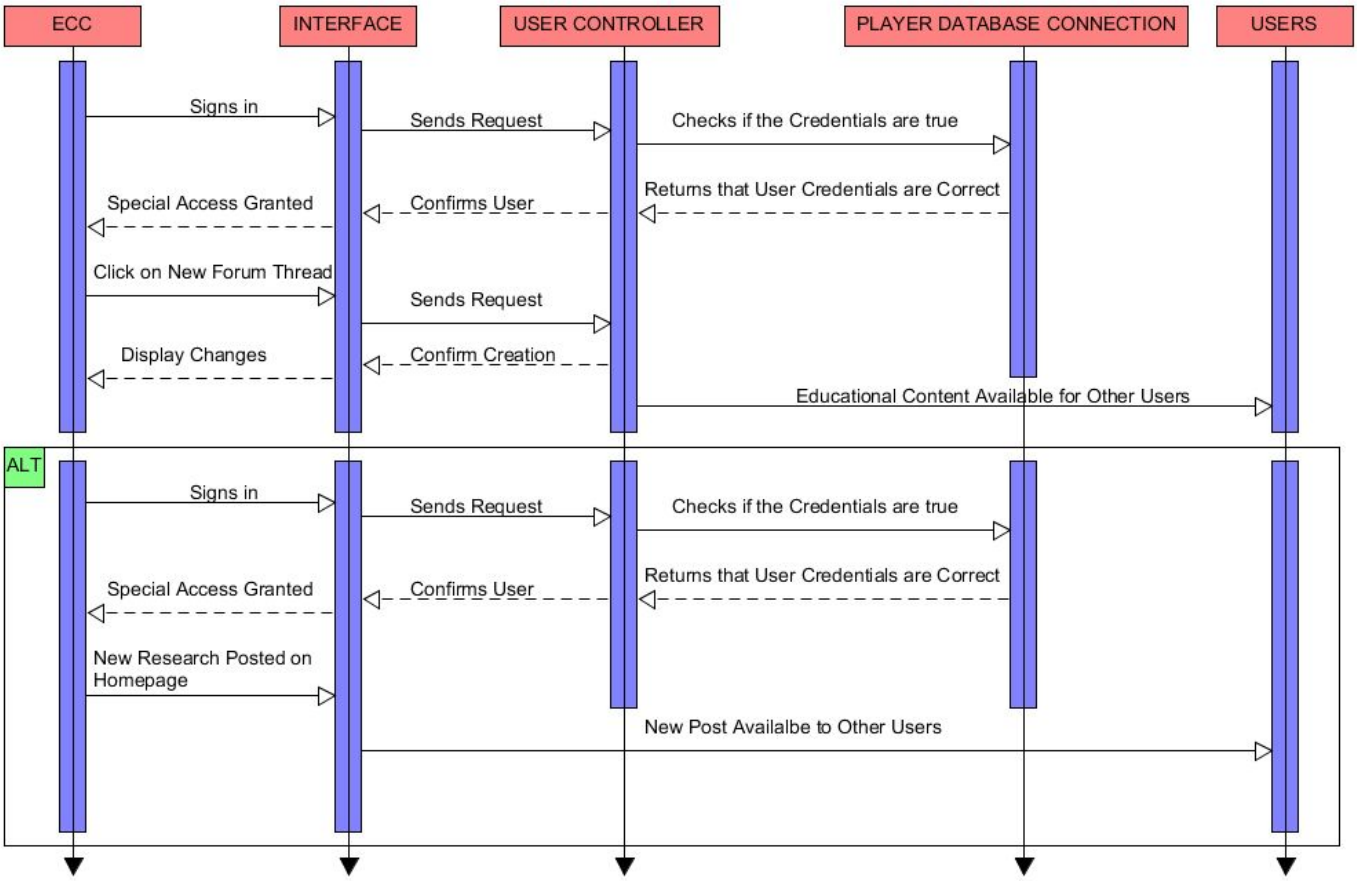


Figure 5.6

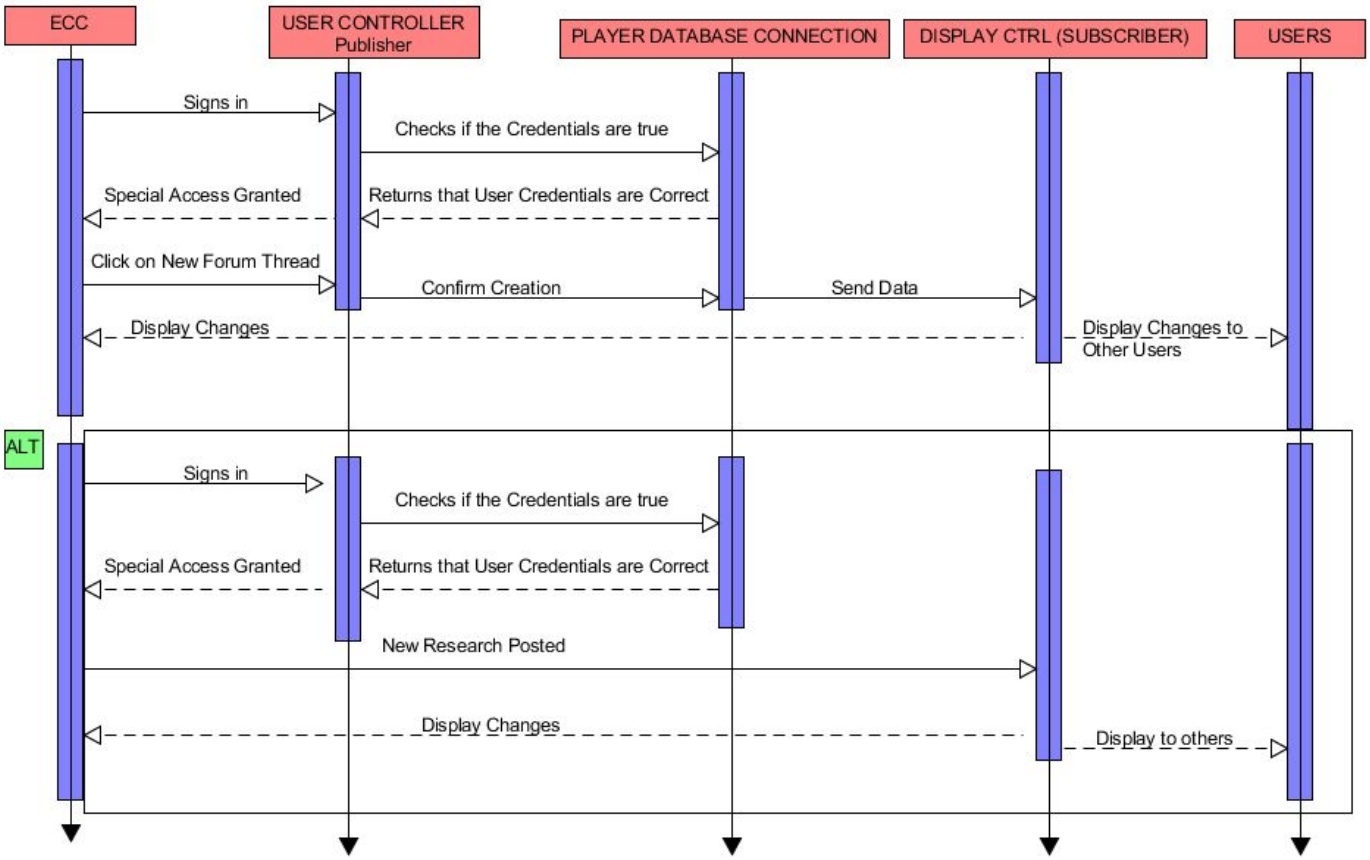


Figure 5.6B (Design Pattern Implemented)

For this use case, the Educational Content Contributor (ECC) has two ways in which he or she can provide news, tips, investment ideas, strategies etc. The primary way is to create a new post or thread in the Investment Forum section. In this main scenario, the Educational Content Contributor will first log in to the Trade Kings website. The User Controller will then send the login data to the Player Database and check if the corresponding playerID contains the special credentials. If so, then the User Controller will give access to the ECC to create a new post, detailing an Investment Idea. After creating the new discussion thread, the User Controller will display the content on the interface and other users will also be able to see it on their Discussion Forum Page.

The secondary way to provide educational content is through the homepage. In this alternate scenario, the User Controller will again verify that the user has the special credentials (via the League Database) when the user logs in. If the user does in fact have these credentials, the user will be given access to post Education Content. After this, the ECC will simply create a new post on the homepage and provide daily market updates and investment strategies. Finally, all the users will then see this on their homepage when they log in. Users will also be given access to the forum by being freeing to comment and receive comments from other users.

The design principles that we employed in assigning responsibilities to the objects are High Cohesion Principle and Low Coupling Principle. That is because, in the interaction diagram, there are 5 objects that are interacting with each other in the system, there is a relatively even amount of arrows between all of the objects (leaving the objects). This shows that there is not any particular object taking on too many responsibilities (both computation responsibilities and communication responsibilities).

The second diagram implements the Publisher-Subscriber design pattern to implement the posting of educational content. When a user signs in, the controller checks the credentials in the database and if correct, he or she is given the special access to create a thread. After creation, the publisher will store in the database which will then send to the subscriber to display the content back to the user and also to other users for their response. This is an efficient way of implementing this functionality.

UC11 - Viewing Market Forecasts

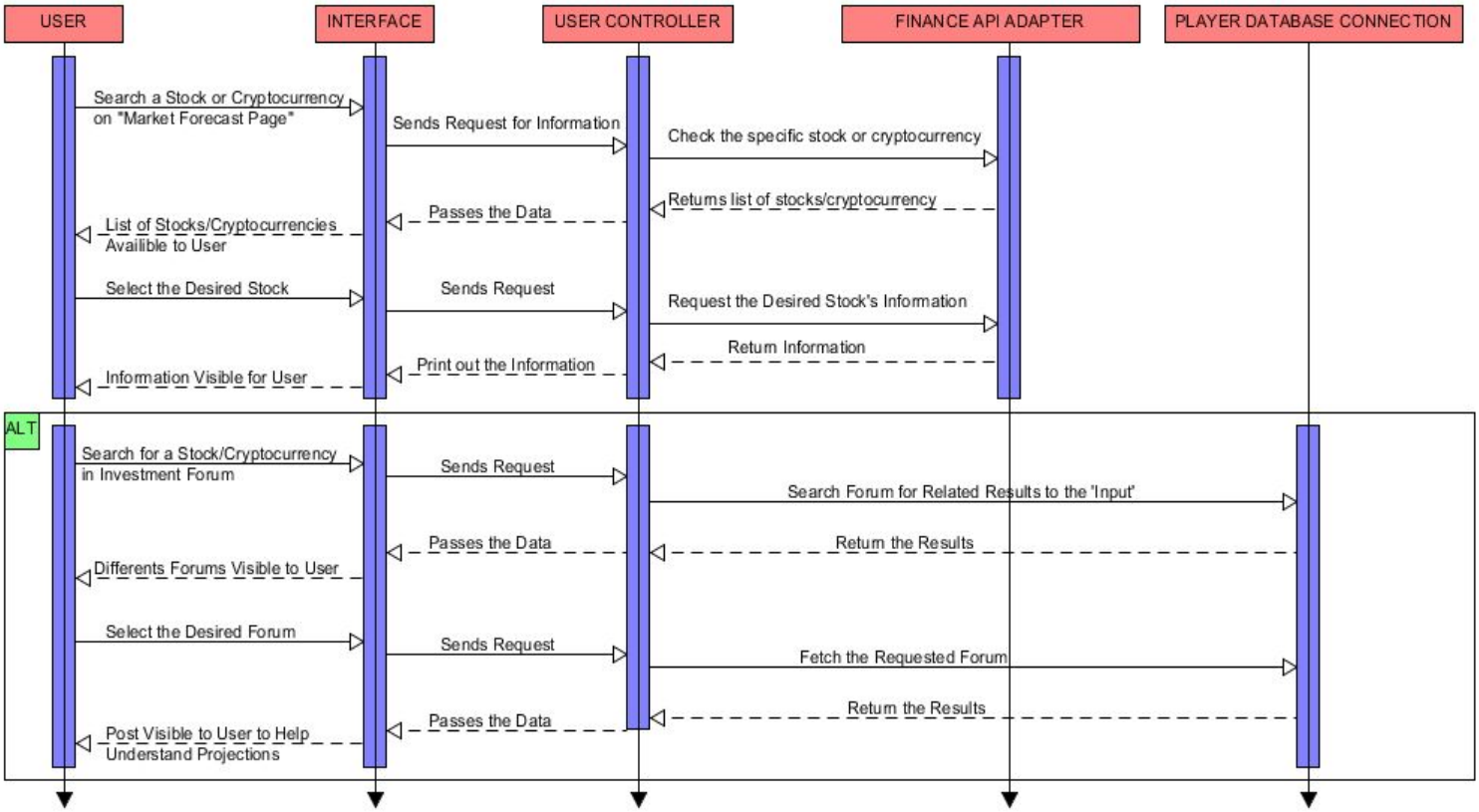


Figure 5.6

For this use case, the user has two ways in which he or she can learn more about future market forecasts. The first way a user can accomplish this is by searching for a stock or cryptocurrency on the "Market Forecast Page". After this, the user controller will check retrieve all the stocks or cryptocurrencies related to the user search from the finance API adapter. Then this list will be returned to the user controller which will print it on the screen for the user to see. Afterwards, the user will select a specific stock from that list and in return, the user will see information obtained from our forecaster on the stock or cryptocurrency. The secondary way of viewing market forecasts is through the Investment Forum. After searching for a stock in the search bar, the controller will look in the player database and gather all the posts that are related to the search of the user. The user then selects the specific post that he or she wishes to open and through this, he or she can view and understand other users' projections on the future value of the desired stock/cryptocurrency.

8 Class Diagram and Interface Specifications

A. Class Diagrams:

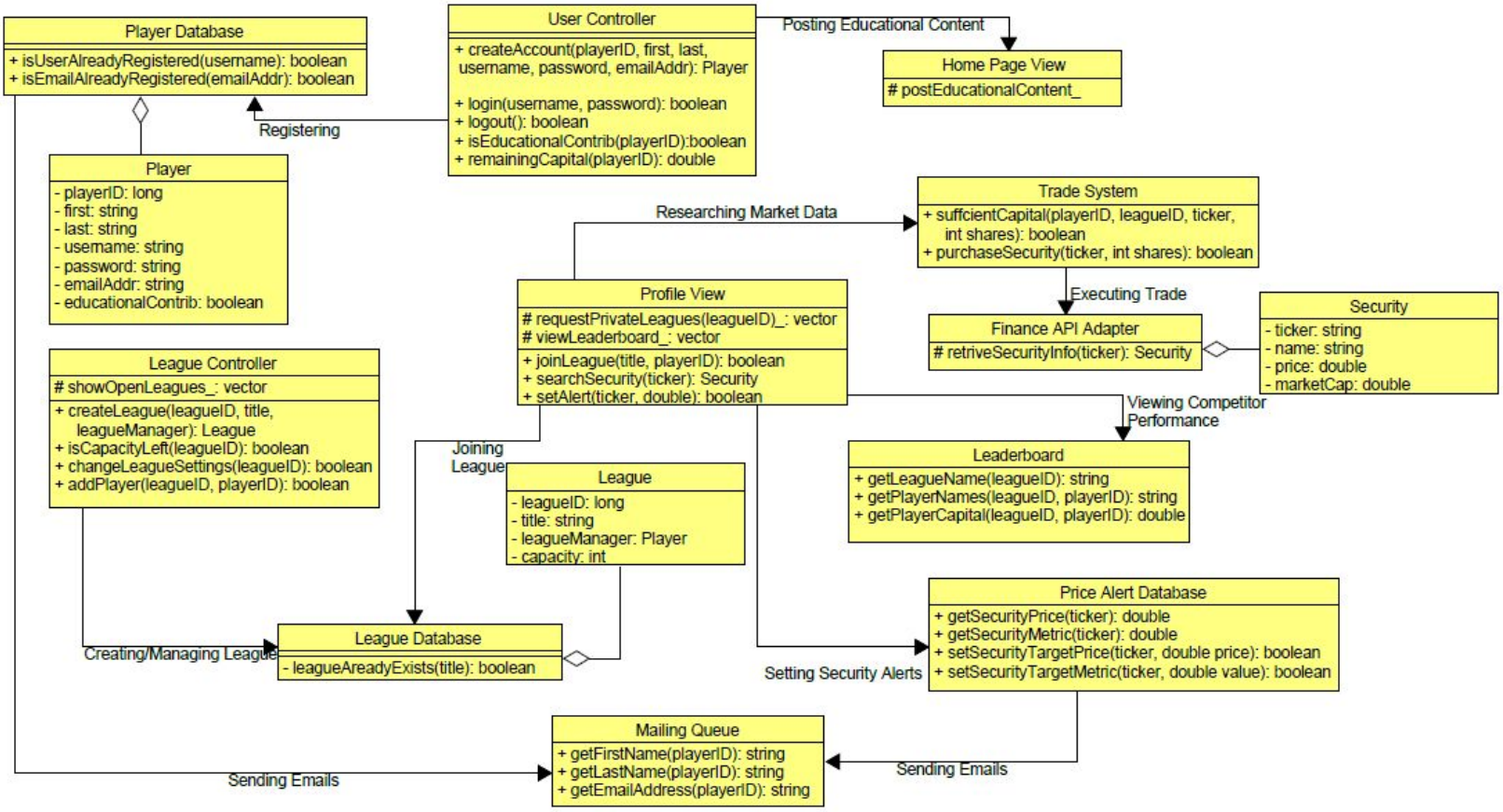


Figure 6

B. Data Types and Operation Signatures:

Class #1: Player/User:

This represents the Player/User object class (which is what will be stored in the Player Database, i.e. each tuple is a unique Player), for a user that has a registered account and can use the site's functionalities. The first attribute for the Player object is the playerID, a numerical identifier of data type long, which will serve as the primary key in the Player Database. The rest of the attributes are self-explanatory: first is the user's first name that they enter registering (data type string) and last is the user's last name that they enter registering (data type string). Username is the username that the user will write when registering (data type string); it must be

unique and this username is what will be displayed next to the player's avatar whenever he/she is logged in. Password is the password that the user will input and subsequently confirm when registering (data type string). Finally, emailAddr is the email address that the user has to input when registering, which will then be used to send a confirmation email to that user. The last attribute is educationalContrib: this attribute is too see if the user has the credential to post education content on the site's forum (daily market updates, investment strategies, etc.). The data type of this attribute is a boolean (true or false) and its default value will always be set to false (has to manually be set to true for a contributor).

Class#2: League:

This represents the League object class for every new league that has been created as well responsible for managing every single league in the system. This object is what will be stored in the League Database, i.e, each tuple is a unique league. The first attribute is the leagueID, a numerical identifier of data type long which will serve as the primary key in the League Database. The rest of the attributes are the title of the league (data type string), the leagueManager (data type Player), and finally capacity, which represents the total number of players allowed in the league, as set by the League Manager (data type int).

Class #3: Security:

This represents the Security object class for every different financial instrument that a player can invest in (purchase or sell or trade). Our platform/product will be offering two different securities: equities (stocks) and digital cryptocurrencies. The first attribute is the ticker of the security (data type string), which is how every security is listed on an exchange, and how the security is searched in the database/Finance API (such as VZ for Verizon). The second attribute is the actual name of the security (data type string). The final attribute is the marketCap, which is the market capitalization. The market capitalization is the total current value of the company (whose stock we're buying) or the total current value of the cryptocurrency (how much of that currency exists and has been mined). This value is derived by multiplying the current share price/currency value by the total number of shares/total amount of that currency.

Class #4: User Controller:

The User Controller class is what controls the ability of the user to perform the most basic functions of using the Trade Kings system. It also takes care of every single account created in the system. The first method/operation is createAccount, which is what allows a visitor to register. This method takes in the playerID, first, last, username, password, and emailAddr, which then creates a Player object that is stored in the Player Database. If the user is registered, then the User Controller can perform login/logout, which takes in the username and password as parameters and returns a boolean based on if the login is successful (the controller checks the parameters with the database to see if they are correct). Similarly, the method logout returns a

boolean as well but does not take in any parameters. The method isEducationalContrib(playerID) takes in a playerID and searched the Player Database using that ID, to see if the player has the proper credentials necessary to post education material; it returns a boolean value, based on whether or not the user has the credentials. Finally, remainingCapital takes in playerID as a parameter and checks if the player has enough capital left for a particular league. This method pulls information from the Player Database and is used to check if the user has enough cash to purchase a desired amount of a certain security.

Class #5: Player Database:

The Player Database class stores the Player objects that were defined earlier above and uses the playerID as the primary key. The first method is isUserAlreadyRegistered( ), which takes in the string username as a parameter. This method checks to see if a username is already taken when a new visitor is attempting to register, because every username needs to be unique; it returns a boolean based on whether the username already exists in the database. Similarly, the second method is isEmailAlreadyRegistered( ) which takes in the parameter emailAddr and checks if that email address is already associated with a registered account for the same reasons; again, it returns a boolean based on whether the email address already exists in the database.

Class #6: League Controller:

The League Controller controls the actions that can be performed surrounding leagues. The first method is createLeague, which takes in the parameters leagueID, title, and leagueManager. This method uses those parameters to create a League object which is then stored in the League Database. The second is method is isCapacityLeft, which takes in a leagueID; this method then uses that leagueID to check in the League Database, whether there are open spots in that particular league. This method then returns a boolean, indicating whether new users can join. The next method is changeLeagueSettings which takes in a leagueID; this method then uses that leagueID to change the data in the LeagueDatabase returns a boolean based on if it was successful. The last method is addPlayer which takes in a leagueID and playerID; this method adds the playerID Player to the league tuple associated with the leagueID, return a boolean based on whether it was successful.

Class #7: League Database:

The League Database class stores the League objects that were defined earlier above and uses the leagueID as the primary key. The first method is leagueAlreadyExists( ), which takes in the string title as a parameter. This method checks to see if title for a league already exists in the League Database, when a player attempts to create a new league, because every title has to be unique; this returns a boolean based on if the title already exists in the database.

Class #8: Profile View:

The Profile View class represents the page that a registered user will use once they have logged in and wish to view their account. From Profile View, a user can access league information and make moves regarding the stock market. To join a new league, two different methods can be accessed. The first method is joinLeague(title), where the user can be added to a league based on choosing from a list showing the various leagues (listed by league title). This method will return a data type boolean, true on success, and false depending if the user gets denied to join an specific league. If successful, the user's playerID will be added to the league (based on its corresponding leagueID) in the League Database. The above method is for joining public leagues, however, the user also has the option to request join a Private League. For this action, there is the method requestPrivateLeagues, which will return a list of the Private Leagues, from which the user can request permission to join. The third method is searchSecurity(ticker), which enables the user to research into the market data. The user can access information on various securities through this method, which takes in the security ticker as a parameter, and returns the security object, which contains all pertinent info regarding that security (this search is done through the Finance API). Finally, the last method is setAlert(ticker, double), which allows the user to set either a price alert or metric alert on a particular security. The first parameter is the ticker of the security, and the second parameter is a double representing the target value of the metric/price. This alert will return a boolean (true or false) based on if it is successful; if it is successful, then the alert will be stored in the Price Alert Database.

Class #9: Home Page View:

The Home Page View class displays the material that the Educational Content Contributor wishes to post. As a result, this class is very simple and it's only method is called postEducationalContent which simply outputs a formatted block of text that is inputted by the Educational Content Contributor.

Class #10: Leaderboard:

The Leaderboard class is another simple class, which displays the rankings of all the players within a particular league. The Leaderboard has three main methods that it calls so that it can retrieve the necessary information to display the rankings. The first method is getLeagueName(leagueID), which takes in leagueID as a parameter and uses this ID to find the corresponding league title from the League Database; the data type that the method returns is a string for the title. The next method is getPlayerNames(leagueID, playerID). Using these two IDs, the class is able to retrieve the actual names of all the players competing in the specified league; as expected, this method returns a string for the player names. Finally, the third method is getPlayerCapital(leagueID, playerID), which uses the two IDs to retrieve the current total amount of capital each particular player in the specified league has. The method returns a double (amount in USD), and it is this value that the competitors in a league will be ranked by.

Class #11: Trade System:

The Trade System class is what is used to actually execute a player's desired trades, and it interacts closely with the Finance API Adapter, which will be defined next. The Trade System class has two main methods. The first is sufficientCapital(playerID, ticker, int shares); this method is necessary because before executing a buy order for a player, the system first needs to check if the user has enough money to buy the desired number of shares. The method uses the playerID, the security ticker, and number of shares to retrieve this information and returns a boolean data type (true or false) based on whether the player has enough capital or not. The second method is purchaseSecurity(ticker, int shares) which actually executes the trade for the player. The method takes in the ticker of the security that the player is interested in trader and the number of shares that he or she wants to buy/shell. It then uses this information to execute the trade via real time data from the Finance API and returns a boolean based on if the action was successful or not.

Class #12: Finance API Adapter:

The Finance API Adapter Class is what allows our system to retrieve live, real-time market data, using the Alpha Vantage API. This class is extremely crucial to our system yet relatively simple in its implementation. The class' only method is retrieveSecurityInfo(ticker); this method works by taking in the security's ticker, and using that ticker to search and pull the corresponding security's data from Alpha Vantage. As expected, the method returns all of the relevant security information via the data type of a stock/security object.

Class #13: Price Alert Database:

The Price Alert Database class is what stores all of the different price and metric alert that a user might set up. The first method is getSecurityPrice(ticker) which retrievers the current price of a security based on the security's ticker; it therefore returns a double. Similarly, the second method is getSecurityMetric(ticker) which retrieves the current value for any metric (other than the price) of the security based on its ticker; this returns a double as well. Similarly, the system also needs to store the target value for the price or metric of the desired stock, which is does through two similar methods. These are setSecurityTargetPrice(ticker, double) and setSecurityTargetMetric(ticker, double). Both of these methods work the same way. They take in the ticker of the desired security and the value of the target price/metric, storing it the database. Both methods return a boolean as the result, based on whether or not the action was successful.

Class #14: Mailing Queue

The Mailing Queue class is what takes care of sending emails to the users. It sends emails to serve for two purposes: the first is to send a confirmation email upon a new player's successful registration, and the second is to send an email to a player when a price/metric alert

that they have set has been triggered (i.e., the conditions for the alert have been satisfied). The Mailing Queue class has three methods to retrieve the necessary information needed to send the email. All of this information will be retrieved from the Player Database, using the playerID (which is the primary key of the Player Database) as the input parameter. The first method is getFirstName(playerID), which uses the playerID to retrieve the player's first name (data type string). Similarly the second method is getLastName(playerID), which uses the playerID to retrieve the player's last name (data type string). Finally, the last method is getEmailAddress(playerID), which again uses the playerID, this time to retrieve the player's email address (data type string).

C. Design Patterns:

The following design patterns either were implemented, planned on being implemented or can be implemented theoretically given the group's use cases.

Publisher-Subscriber:

With the use of the publisher-subscriber design pattern one can easily isolate business logic from that of programing and algorithm logic. This can make it much easier for one to edit code more efficiently and make changes without having to search through an abundance of it. These edits include bug fixing, restructuring and deleting certain elements of different object classes. This can lead to users on Trade King's to interact with the system more efficiently meaning transactions such as a purchase of stock would be less time consuming. This is mainly because in the publisher-subscriber design pattern publish classes do not design functions specifically for one receiver. Instead, they design an output class that recieve can also access without knowing the exact nature of the publisher. This can be seen with executing a stock sequence diagram as the system is designed to act as a publisher involving the exchange of stocks and the subscribers can be seen as all the different stocks and cryptocurrency seen on the market.

Decorator Pattern:

For this pattern behavior can be added or removed to an individual object without affecting other objects from similar or even the same class. For this report separate functionalities were coded in both PHP and Javascript. This was made possible because the group made use of isolated objects such as Forum objects or API stock objects. This allows code to be edited more easily and that a flaw in changing one object does not branch over and affect other design objects. This can be seen in all the separate use cases and sequence diagram

considering most of them operate independently from one and other such as sequence diagram 7, educational content, and use case 4, research security and market data.

Object Relational Model Pattern:

This design pattern took use through phpmyadmin and mySQL which allowed us to simulate databases and queries without actually typing out the queries. Basically, mySQL allows one to create databases and tables through a simple U.I. This allowed the group to construct various tables for testing quickly and efficiently when building functionalities such as the Trade King's Forum or the League's Page. Later, in order to give the group more control over the database and to access the database from separate computers the group took use an Amazon based. This Amazon database can be accessed remotely and needs to be updated with specific queries. However, it allows for more accessible database access. The databases and forums taking use of this design pattern can be seen in the Posting Educational Content sequence diagram.

Model-View-Controller:

Using this architectural style is very crucial when developing or creating a website that has user interaction. This style divides an application into three categories;model, view, and controller. View is what the user gets to see and for this project, it is the UI of the website. Without a view, the website does not exist. Similarly,without a controller, the user can't use the website. In the Trade King's website, every input the user makes goes to the controller. Whether the user clicks on register ,use case 1, join a league, use case 2, buy a stock , or sell a stock, the input will be taken by the controller. It, then, will convert the input into specific commands for the model. Finally, the "model" is the core of this architecture as it manages the data and logic of the application. Without model, the users actions will not take place on the screen. Any action the the user makes goes through the model, where it is fully completed, and then it is shown on the view. This style is what makes a website useful.

Responsive UI Pattern:

With the help of bootstrap the group was able to efficiently load more advanced graphics for the Trade King's website. Bootstrap allows one to aid stunning designs to their project without sacrificing speed as the patterns are rendered almost instantaneously. Basically, this design allowed us to seperate user interactions and experiences with basic User Interface and other cosmetics. This is similar to how the publisher subscriber pattern is able to isolate different objects. Bootstrap was used for much of the U.I. especially in the central home page. This can be seen in sequence diagram case 1, registering, as it is utilized in the registration and home page of Trade Kings.

D. Traceability Matrix:

	User Controller	League Controller	Player Database	League Database	Leader Board	Finance API Adapter	Mailing Queue	Price Alert Database	Trade System	Home Page View	Profile View
User Controller	X										
League Controller		X									
Player Database Connection			X								
League Database Connection				X							
Leaderboard					X						
Finance API Adapter						X					
Mailing Queue							X				
Price Alert Database Connection								X			
Trade System									X		
Home Page View										X	
Profile View											X

Essentially, all of the domain concepts have been mapped onto the classes from the class diagrams. The only name changes are that we went from Player Database Connection to Player Database and League Database Connection to League Database; this is because we are more concerned with the databases themselves than the connection between them and the controllers. Furthermore, we can take a look at how all of the Domain Concepts have evolved into the Class

Diagram through the Use Cases we have defined earlier. We can trace this evolution using both the Traceability Matrix and Class Diagram. First we see that the “Registering” use case involves the User Controller, Player Database, and Player concepts/classes. The “Creating/Managing League” involves the League Controller, League Database, and League concepts/classes. The “Joining League” use case involves the Profile View, League Database, and League concepts/classes. The “Researching Market Data” use case involves the Profile View and Trade System concepts/classes. The “Executing Trade” use case involves the Trade System, Finance API Adapter, and Security concepts/classes. The “Posting Educational Content” use case involves the “User Controller” and “Home Page View” concepts/classes. The “Setting Security Alerts” use case involves the Profile View and Price Alert Database concepts/classes. Finally, the “Viewing Competitor Performance” use case involves the Profile View and Leaderboard concepts/classes.

E. Object Constraint Language (OCL)

Essentially, Trade Kings is composed by important constraint languages, and we will explain each class in detail:

Class #1: Player/User:

This class represents the Player/User. The User will be allow to register into the website by setting up an account according to its name, email and created password. Once the account is fully completed, the user will be allow to experience Trade Kings and its functionalities.

Invariants:
Context Player **inv;**
self.emailAddr.length() >= 10;
self.username.length() >= 5;
self.password.length() >= 8;
self.educationalContribub = FALSE; (default value)

Pre- and PostConditions:
None: Player is a simple object with no operations

Class#2: League:

This class represent the Leagues in general in Trade Kings. The League will have a set of created rules by the User or The league will contain weekly competitions for rewards at the end of each week. Each league will have its own capacity of players allowed in the league as set by the League Creator.

Invariants:
Context League **inv;**
self.emailAddr.length() >= 10;
self.capacity <= 5;

Pre- and PostConditions:
None: League is a simple object with no operations

Class #3: Security:

This class represent the security that is being used for different financial instrument that the user can invest in Trade Kings. As we said previously, we using two types of securities: Equities and digital cryptocurrency. We want to provide to the User new ways to invest on the Stock Market, as well as be able to provide educational content for the User's benefit.

Invariants:
Context Security **inv;**
self.price >= 0;
self.marketCap >= 0;

Pre- and PostConditions:
None: Security is a simple object with no operations

Class #4: User Controller:

This class is one of the most important in Trade Kings because it makes sure that each new or existed account is registered in the system. This controller has the ability to control the user's perform when she/he is using basic functions regarding with functionalities in Trade Kings. Furthermore, it confirms the User when they log in or log out, as well as it checks if the user has enough cash to perform purchases of stocks in Trade Kings.

Invariants:
None

Pre- and PostConditions:
context createAccount(playerID, first, last, username, password, emailAddr): Player
pre: NULL
post: initialized Player Object has been created

context login(username, password): boolean
pre: emailAddr, password initialized

post: boolean returned based on whether or not login was successful

Class #5: Player Database:

This class contains every single information of an existed or new user in Trade Kings. The Player Database will contain the history of the User's performance in a particular league, as well as purchases of stocks or personal information in the portfolio.

Invariants:

None

Pre- and PostConditions:

context isUserAlreadyRegistered(username): boolean

pre: username is an initialized string

post: boolean returned based on whether or not username is in Player Database

Class #6: League Controller:

This class is one of the most important in Trade Kings because it makes sure that each new or existed league is registered in the system. This controller has the ability to check for open spots in the League Database, as well as allowing the User to create new leagues with any preference. Furthermore, it confirms the User whenever another User decides to join his/her league or be able to block a User for misbehavior.

Invariants:

None

Pre- and PostConditions:

context createLeague(title, leagueManager): League

pre: title and leagueManager strings are initialized

post: initialized League Object has been created and stored in League Database

Class #7: League Database:

This class contains every single information of an existed or new league in the Trade Kings system.The League Database will save every action make by the User and other players in any particular league. Furthermore, it will also save the players status on the game and send it to the league controller to update the Leaderboard. Also this particular database will let know the User if the league has being created with the corrected parameters.

Invariants:

None

Pre- and PostConditions:
context leagueAlreadyExists(title): boolean
pre: title string has been initialized
post: initialized Player Object has been created

Class #8: Profile View:

For this class the user will be presented with a U.I. containing their personal information as well as information pertaining to their stocks and cryptocurrencies. This information includes current stocks, how much capital one has and other factors such as leaderboard rank.

Invariants:
None

Pre- and PostConditions:
context joinLeague(title, playerID): boolean
pre: title string has been initialized
post: player with playerID has been added to League Database

context setAlert(ticker, double): boolean
pre: ticker and double parameters defined
post: returns boolean value based on if setting alert was successful

Class #9: Home Page View:

In this class the user will be given the view of our homepage when a user signs in or when ever they command our website to go back to the home page. The home page will list all sections of the website along with a portal to enter those certain sections. The call is intended to make the transitions between portals in the website easier for the users/players.

Invariants:
None

Pre- and Post Conditions:
None

Class #10: Leaderboard:

This class is implemented by taking data from every Trade King's member. Then the data is compared and the player's with the greatest overall value are displayed on a board. This board consists of the player's email, avatar and their total score.

Invariants:

None

Pre- and PostConditions:

context getPlayerCapital(leagueID, playerID): double

pre: title string has been initialized

post: player with playerID has been added to League Database

Class #11: Trade System:

The trade system class will allow the user to make trades with the shares they either currently own in their player portfolio, or stocks/ cryptocurrency they are looking into purchasing. A player will only be allowed to make a trade if the action is approved by the system. Certain factors determining the approval are what the player owns within their portfolio and what information is located in the Trade Kings database.

Invariants:

sufficientCapital.shares > 0

Pre- and PostConditions:

context sufficientCapital(playerID, leagueID, ticker, int shares): boolean

pre: playerID, leagueID, ticker, and shares are all initialized/set to valid values

post: boolean returned based on whether user has enough capital to purchase desired number of shares

Class #12: Finance API Adapter:

The Finance API Adapter class, the website will take all current stock, cryptocurrency and price information from the API,"iex trading" for stocks and "alpha vantage" for crypto, to use for the its database. The website will capture the the information taken from the API the website is using, which will be later stored in the database for the rest of the system/ cases.

Invariants:

None

Pre- and PostConditions:
None

Class #13: Price Alert Database:

With this class users set target price values for stocks and cryptocurrencies. This is done on a form which uses SQL queries to store these values in the email database. Later on checks can be run to compare current stock values to the values in this database to see if they have reached the players target value.

Invariants:
None

Pre- and PostConditions:
context getSecurityPrice(ticker): double
pre: ticker set to valid string input
post: value of unit of security returned

context getSecurityMetric(ticker): double
pre: ticker set to valid string input
post: value of security's metric returned

context setSecurityTargetPrice(ticker, double price): boolean
pre: ticker set to valid string input
post: new alert added to Price Alert Database

context getSecurityTargetMatric(ticker, double value): boolean
pre: ticker set to valid string input
post: new alert added to Price Alert Database

Class #14: Mailing Queue

This class sends email to the user whenever the player /user takes part in a action that signals to the system to send that player an email. The functions that this class will be used is when a user needs confirm their account with a confirmation email and when the user selects to have stock updates on certain companies. This is when the users will receive an email containing information that would be relevant to a certain company they are interested in investing in.

Invariants:
None

Pre- and PostConditions:
context getFirstName(playerID): string
pre: playerID set to valid string input
post: first name string of corresponding player returned

context getLastName(playerID): string
pre: playerID set to valid string input
post: last name string of corresponding player returned

context getEmailAddress(playerID): string
pre: playerID set to valid string input
post: email address (emailAddr) string of corresponding player returned

9 System Architecture and System Design

A. Architectural Styles:

To allow this project to function at maximum capacity several Architectural styles will be used throughout this project. Implementing these styles plays a critical role in order to success of our project. The Architectural systems that will be used includes, but not limited to Data-Centric Design, Client Server Models, Model View Controller and Event Driven architecture. Each separate architecture plays a significant role in the overall project and they will be expanded upon in great detail in the upcoming parts of this report.

- **Data-Centric Design:**
  This is probably the most integral part of Trade Kings fantasy league because of how this site is centered around storing information in databases. Whether it be user login data, portfolio pictures, avatars, user achievements, leaderboard statistics, the databases are key. Also, along with user data such as trades, sells and purchases, the stock information will be stored in the databases using an API. As, it can be clearly seen without this databases storing an amalgamation of stock and user data Trade Kings fantasy league would not be possible. Overall, the databases will provide a personal host for individual users where they will be able to interact with the user interface and stocks using the Finance API.
- **Client-Server Model:**
  Within Trade Kings the various users, including Players and League Managers, will be interacting on a client-server model. Whether it be the user interface or various other stock exchange methods, the client-server relationship plays a significant role. This can be done via the Internet where the Trade Kings distributed application will allocate resources across a server to clients. This interaction between user, the client, and Trade Kings will use the Model-View-Controller in order to smoothly transition communication. Also, Trade Kings will have some systems not associated with clients on the side to ensure the site and its flow are functioning properly. Nevertheless, user will constantly be the primary client and will be the one to interact with other subsystems.
- **Model-View-Controller:**
  This is an architectural style that divides an application or a program into three different sections. The three parts include model, view, and controller. The user gets to see the "view" and use the "controller". The "view" is the UI and it is what the user sees on the website. So for this project, the "view" can be the homepage of the Trade Kings league, any informational diagrams or charts that are on the website. Then the "controller" accepts inputs from the user and converts that into specific commands for the model or

view. Finally, the "model" is the core of this architecture as it manages the data and logic of the application. Any action by the user goes through the model and then is shown on the view.

- **Event-Driven Architecture (EDA) :**
  One of the features of the Trade Kings League is that it will send out emails and notifications to all the users and for this, EDA will be used. This is an architecture style which promotes a reaction after a specific event occurs. An event could mean a big change in state. For example, whenever a stock price declines significantly, it will be considered as an event and then emails will be sent out to all the users who wish to take advantage of the situation. With this architecture, the application will become more responsive and interactive.
- **Object-Oriented:**
  Each feature of the project is divided up into different groups that relate to the feature. In our architecture, we decided to have feature-dedicated databases. For example, the trade system has its own database, the price-alert database, where all a player's stocks and stocks on the watchlist are stored. The league database holds a player's list of leagues joined as well as statistics they have performed in the league. Dividing these groups allows for an efficient division of labor.

B. Identifying Subsystems:

For the Trade Kings Project, we want to build a platform/system that will work on several interfaces. The subsystem identification is where we will be able to analyze each part of the software. Overall, there are two major parts to this system, the front-end and the back-end. We will have the user be able to see the Front End system of the project. This is where the user will be able to interact with the system whenever they choose to take any actions. The user interface should be the same for all devices no matter what operating system it is running on as they all access the system through the web. The website should be able to work on all devices, including smartphones and tablets. Keeping all this in mind, the website is primarily designed for a computer website browser style. When the user is presented with interface, the front end should include the login, registration, and the rest located in the diagram below. All users will equal opportunity to access each option in the front system. Through all this, we will be able to give the user a more pleasant experience when interacting with Trade Kings.
The Front End is reliant on the Back End system. The back end is where all the data will be stored and retrieved to the user. There are two major components to the back end. One is the database which stores all user related information and the second is accessing live financial data using the Alpha Vantage API. On a regular basis, the API must be used to update all users' portfolios, so they reflect accurate information. When a user researches on a specific stock, then it is the responsibility of the API to retrieve the correct information and send it back to the front

end to display it to the user. Similarly, on request to view one's portfolio, the database relays that information back to the UI. Hence, there is seamless integration between the back-end and front-end components as they rely on each other in order for the whole system to operate flawlessly. The other section is all the actions that user will be allowed to access when dealing with stocks with Trade Kings. They can either view a stock on it's previous data. They can also buy a certain stock for their portfolio. Lastly, they can sell a stock they once own that they no longer want possession of in their portfolio. All Actions are offered to users at all times, as long they past requirements. When an action is summed, the system controller will inform the rest of the Back End and Front end of what action the user is choosing to take.

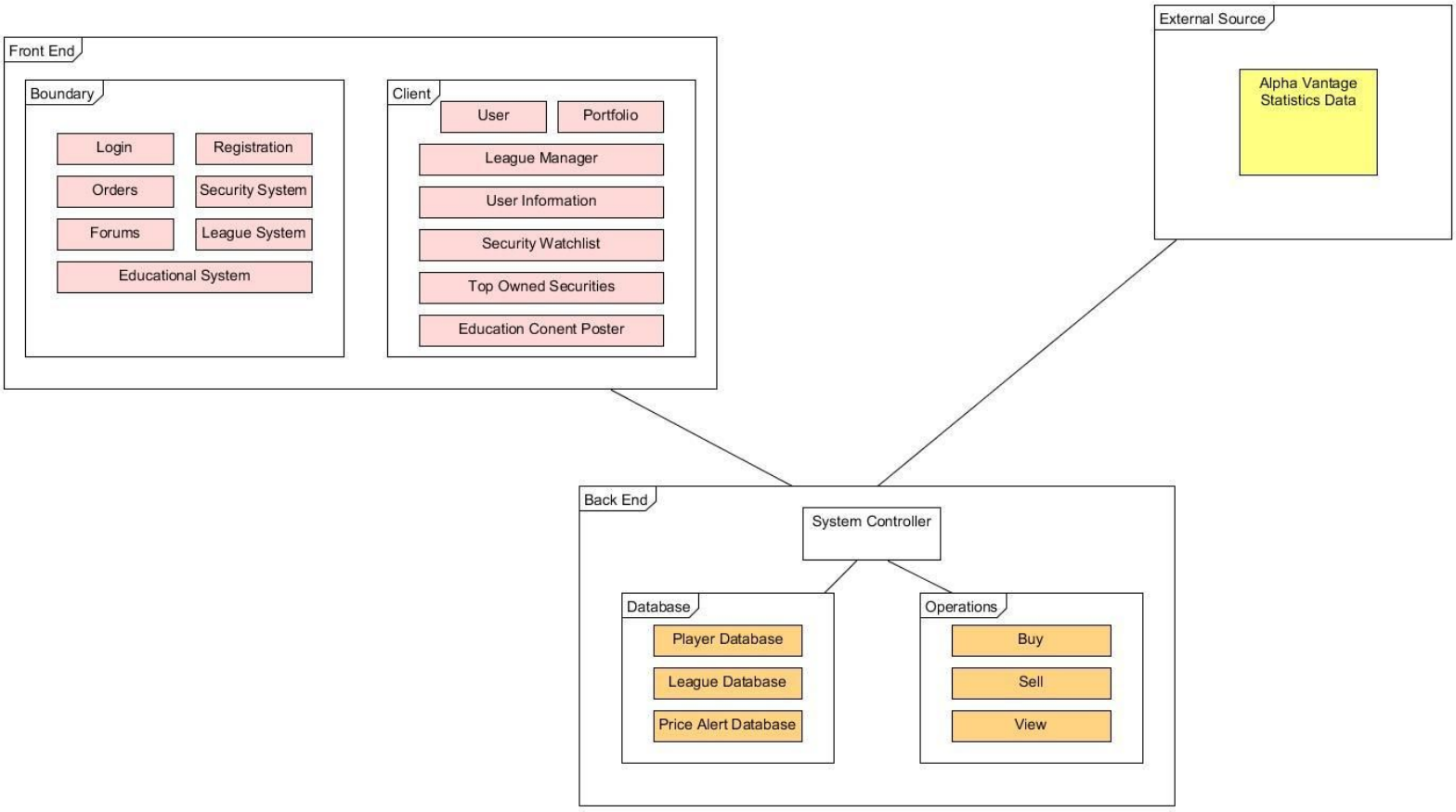


Figure 7

C. Mapping Subsystems to Hardware:

Trade Kings utilizes amazon's database service. The database is constantly running on a remote machine, however, the website can be opened from any machine. The system is divided into two separate parts, the front-end will handle the User Interface (UI) and the back-end will handle the database. When a user opens the website on their computer, they access the UI which is then connected to the database server located on a remote machine. The user directly interacts with the front-end which communicates with the controller and then relays information to the database server. This allows the user's actions on Graphical User Interface (GUI) to be reflected on the database. Therefore, a user will see the same information from anywhere they open the website as they all access the same database. The database is explained in further detail below.

D. Persistent Data Storage:

There will be a persistent data storage system which will need to outlive a single user's execution of the system. This will be very important as it will need to hold data such as users' login information, portfolio, trading history, league settings, and many other valuable pieces of information. Without this data being persistent, the website almost becomes completely non-functional as a user is no longer to sign-in or even execute a single trade. Trade Kings will consequently be using a relational database management system. In particular, the website will read and write from a MySQL database.

There are several operations that will require a query to the database such as viewing the leaderboard or user's portfolio. However, this query is sent in the back-end as user does not directly decide which query to write. So if the user were to decide to check their trade history, then there will be a query to retrieve the specific user's trade history and the results would be displayed in a table. Therefore, through UI interaction, the user indirectly decides what to query from the database. This is extremely valuable to this website's functionality because it sets a central location to keep data up to date. So regardless of which user or when the user chooses to interact with the website, they will all consistently receive the same real-time updated information.

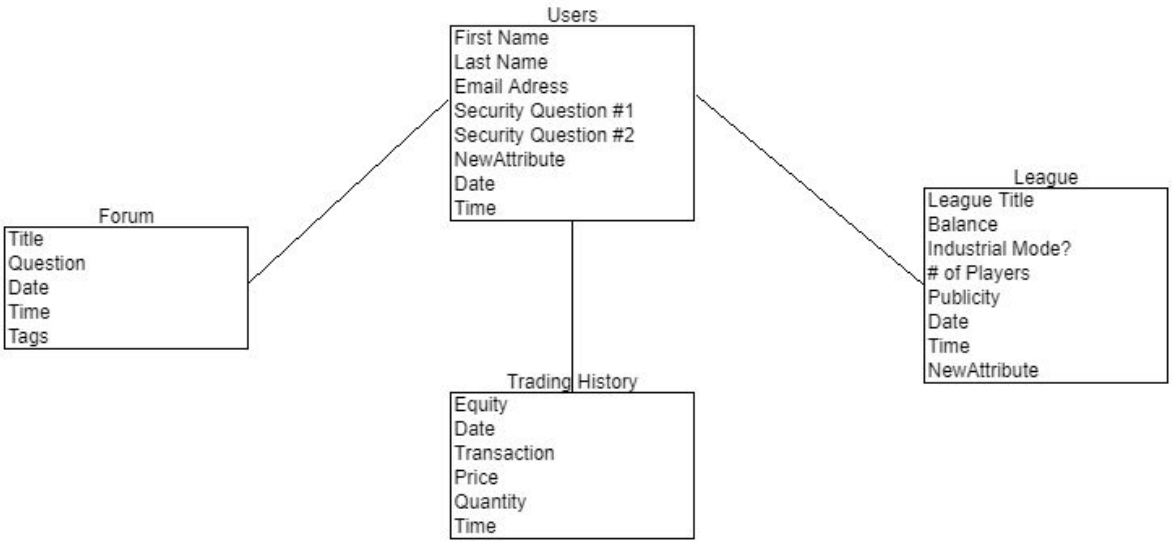


Figure 8

This diagram above is a database schema of how the relational database will store information

E. Network Protocol:

The primary protocol that Trade Kings will be using for our website is HyperText Transfer Protocol (HTTP). It is the basic protocol used by the World Wide Web (www) which defines the format and transmission of messages. When the Trade Kings Uniform Resource Locator (URL) is entered in a browser, using any kind of internet device, a command will be sent to HTTP. Everything on Trade Kings website can be accessed using any device through this protocol. Since Trade Kings uses PHP and MySQL, our system will be running on an Apache server. We decided to use PHP due to its strong connection with MySQL. Basically, as long as a user has an internet connection they can access this website whether it be a smartphone whether it is Android, IOS, Windows, Linux, or any other type of computing device running on a variety of different operating systems.

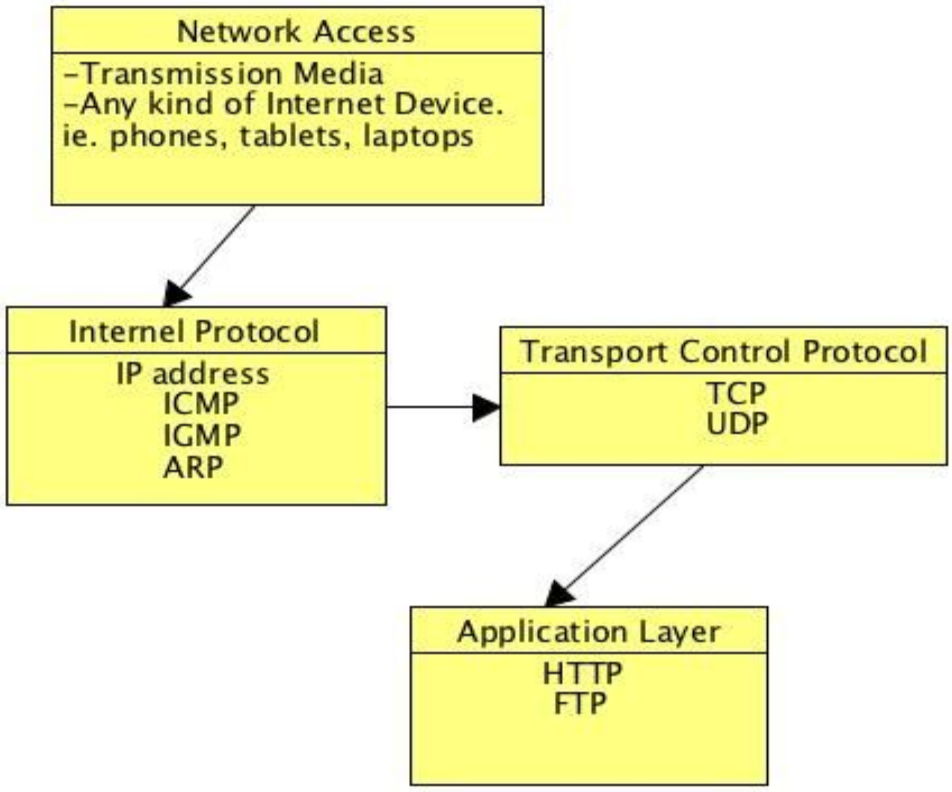


Figure 9

F. Global Control Flow:

Execution Order:

The majority of the Trade Kings' system will be implemented as event-driven. Most of the functionalities will be prompted by an event that results from an action by the user or the site. Also, these events do not necessarily need to occur in the same order to generate a desired action. For example, the execution of a trade is triggered by a Player when he or she views the stock or cryptocurrency page and initiates a trade (if capital amount permits). However, the events around the trade initiation can vary because a Player can access stock/cryptocurrency pages from various locations on the site such as from the "Stock/Cryptocurrency Market" tab or from one's own portfolio.

Certain functionalities are launched by the site itself such as the updating of the Leaderboard which is time dependent, or the removal of outdated content (news, forum posts, etc.). Another example is the email functionality, which can be triggered by confirmations or alerts that the system responds to when the user-set requirements are met.

Although the majority of the system is event-driven, some functionalities are procedure-driven or linear. These functionalities have a defined, step-by-step order.

- Registration: For a visitor to use the website and become a player, the necessary steps (providing information and confirmation) must be taken to register and become a part of the database
- Earning Achievement: The Player must meet a defined requirement list to qualify and obtain any achievement
- Posting on Investment Forum: The Player must go to the "Investment Forum" tab and initiate a reply to any specific forum

Time Dependency:

Trade Kings is heavily dependent on real-time systems considering it is a Stock Fantasy League that depends on current stock market prices. The site will have to adjust accordingly to steep rises or falls in stock prices throughout the date for the users convenience. So besides the timing system that comes with the Finance API, a few other features will depend on timers which are as follows:

- Leaderboard Timer: This timer will be set to update the Leaderboard periodically with the new rankings, adjusting to rank based on any updated financial standing for a player.
- Stock Market Operation Timer: This timer, which is based on the open and close times of the stock market, will dictate when users can execute trades, configuring the site accordingly.
- League timer: This timer will have several functions. First, it is used to mark the length of how long the league is available to be joined. Next, it will be used to mark length of a "season" which is used to eventually determine a winner (if the league has such a limit set).
- Market Update timer: This timer will be used to periodically access the Finance API to update the stock and cryptocurrency data displayed to the users.
- Achievement timer: This timer will be used to check periodically if a user has earned an achievement and it will display a corresponding appropriate message to the user who had reached the goal.
- Inactivity timer: This timer is started at select intervals of no input from the user. The player is logged out after a certain limit is reached.
- Old Forum Posts timer: This timer is activated as soon as a post on the forum is uploaded. After a certain time is reached, the forum post is placed at less priority in the page to allow for new forum posts to be commented.

G. Hardware Requirements:

The hardware requirements of the system are crucial to ensure high quality performance of the website. While the server-side requirements are more vital to the website's success, the client also has some basic requirements which will make the experience more friendly and enjoyable.

- **Client-Side Hardware Requirements:**
    - Internet Connection: The user will require connection to the Internet via Ethernet (LAN), WiFi (WAN), 4G (Cellular), etc. This connection will allow the user to access the website and enable the site to access the database and Finance API to update and display the proper site content. Furthermore, internet access enables the user to execute trades and do so with the correct market data. Also, it allows for the updating of the Investment Forum with information from the Educational Content Contributor and from Players. A higher frequency bandwidth and stronger Internet connection is necessary for ideal performance of the website regardless of web browser.
    - Devices: For a genuine user experience, simple devices such as an operating keyboard and mouse are required. These keyboard will be required if a user wants to search for a stock. While a mouse will be required if a user wishes to buy a stock by 'clicking' on the buy option. Also, a monitor or laptop screen that has a resolution better than 800x600 pixels is required for the website to be seen properly.
    - User's system should be able to run only javascript, as PHP is run on the server

- **Server-Side Hardware Requirements:**
    - Internet Connection: A connection is required for the Trade Kings website to function properly. Without a connection, simple tasks such as searching and trading stocks, updating the portfolio, or reading educational content is not possible. As of right now, since the project has not been completed, it can be predicted that a low band of frequency will be sufficient enough for the website to run properly.
    - Server Disk Space: For the website to run properly for a long time, then lots of storage will be required as many users will be registering and using the it. A predicted minimum storage of 100 GB will be required because as new users keep registering, the amount of information and data will keep increasing significantly. So to avoid any crashes to the server, an approximate minimum of 100 GB should be sufficient.

- System Memory: The system memory will contain the core memory components of the Trade Kings stock market fantasy league. Initially, the amount of data is limited, but as the site grows, a better form of storage will be necessary. Eventually, data will perhaps be allocated to cloud storage so that it can handle a larger amount of data efficiently.

10 Algorithms & Data Structures

A. Algorithms

There are various algorithms implemented in Trade Kings in order to achieve a finished product. It is important that these algorithms are accurate, realistic, and efficient to ensure the best performance of the system. The algorithms described are taken from the Mathematical Models section.

Stock & Cryptocurrency Trading:

During the execution of a trade, once the price of an equity or currency is obtained from the Finance API, the algorithm to determine the trade cost is used to determine to update the user portfolio.

1	Given: numberShares, stockPrice (or cryptoPrice), capital
2	If buying then
3	cost = (price of stock) * (number of shares)
4	If capital > cost then
5	capital = capital - cost
6	updatePortfolio()
7	return capital
8	If selling then
9	payout = (price of stock) * (number of shares)
10	capital = capital + payout
11	updatePortfolio()
12	return capital

Determining Rankings:

Periodically, the system uses the algorithm to determine rankings in order to update the league rankings.

1	Given: listOfStocks, stockPrice (or cryptoPrice), capital, numberOfShares
2	Loop through listOfStocks
3	assetValue = assetValue + (listOfStocks[i].stockPrice * numberOfShares)
4	totalValue = capital + assetValue
5	return totalValue

B. Data Structures

The various data structures used in implementing the Trade Kings system each play an integral role in its quality. Selecting the proper data structure to develop a functionality will directly impact the performance and, as a result, the user experience with the site.

Queues

- **Handling Trade Requests:**
  Throughout the stock trading window (time when the New York Stock Market is open), the Trade Kings system will be responsible for handling different types of trades from different users as they come in. Therefore, a queue data structure will need to be employed in order to store trade requests and handle them in an orderly fashion. Queues are ideal for this implementation because in the real world, the market is a first-come-first-serve system. In other words, whoever places their trade in first, and meets the requirements for the transactions is expected to have their trade go through first. Thus, because the queue is a FIFO (first-in-first-out) data structure, it fits the concept of handling trades in the order they are provided to the system.

- **Email Notification:**
  The Trade Kings system will have the capability to send email notification to the user, not only for confirmation purposes, but also for updates on tracked assets. Anytime a new user successfully joins the website (register) or a current user successfully completes an action (execute a trade), the system will send the confirmation email to the queue to be sent out. Similarly, users have the capability to track certain assets to via the statistics provided. Fore example, a user can track Tesla (TSLA) when the stock price hits a certain threshold, or a user can track Bitcoin until it falls to the user's specified price. In these situations, once the user condition is reached, emails are sent to the queue to inform the user of the condition being met. The existence of a queue will allow the sending of emails, especially when at high volume, in an organized manner. This is mainly done by

pulling user target inputs from a database and comparing this value to one pulled from an API. If the target value is reached an email notification will be sent.

Hash Table

- **User Asset Portfolio:**
  The user asset portfolio, containing the user's owned stock and cryptocurrencies, needs to be stored efficiently, as both the users and system will need to constantly access and retrieve information from the portfolio. Since the portfolio is constantly changing, the storage structure needs to have fast insertion, deletion, and search times. A Hash Table is the best fit for this implementation because it can store the the user's assets along with how much they have, its value, etc. easily. Furthermore, the hash table is searched via a key (which in this case can be the asset itself) and from there efficiently acquire the desired the data. As a result, the insertion and deletion times from the structure are fast. Since, the user portfolios will be constantly changing, this implementation will allow for smoother performance by the system will providing easy access.

- **User Profile:**
  Due the same reasons described above, the use of a Hash Table for the user profile, which contains the user's name, user ID, email address, password, etc. would be ideal. In this scenario, the key would best be defined as the user ID or email address which are required to be unique from all others in the database during the registration process.

Array

- **Finance API Data - Visuals:**
  Arrays will be used thoroughly in the implementation of Trade Kings because of all the numerical data being handled and manipulated. The Finance API provides various types of numerical stock and cryptocurrency data, but in order to make the user's ability to analyze and make decisions based on the data, visuals are crucial. Therefore, arrays can be used to store series of data and create visuals such as a line graph of a stock price over a three month period. Through these visuals, the user experience, can be more friendly, efficient, and educational.

- **Calculations:** Trade Kings will require a lot calculations to be run periodically in order to determine both stock and cryptocurrency trends as well to calculate player performance. For all the various calculation, arrays will play a significant role because many calculations will be iterative over a set of data and arrays are ideal for implementing these types of functions.

- **Email Comparison Data:**
  Target values that a user seeks to reach for certain stocks and cryptocurrencies will be stored in arrays associated with the users unique email. The array values are compared with values received from an API to see if their target price has been reached.

ArrayLists

- **Leaderboard/Achievements:**
  Trade Kings features a real-time leaderboard and achievements which are timely updated. Because the market as well as a user's decision-making can be volatile, uniform movements in the leaderboard are not expected. Instead, the leaderboard will change dynamically. Utilizing an ArrayList for implementing leaderboard is vital because similar to linked list, ArrayList adds the capability to change the size and add elements in between existing elements which regular Arrays do not have. With these abilities, an ArrayList make it easy to follow and apply the changes to leaderboard. Similarly, the amount of people who have reached an achievement will constantly change, and mostly grow as the site ages and the amount of users grows; thus, an ArrayList will allow the system to easily keep track of who has a specific achievement.

- **Real-time Data:**
  During the open hours of the market, the Trade Kings system will constantly be receiving data on a timer as it changes throughout the time period. Since this data is stored and manipulated to provide statistics and visuals, those stats and images will be changing as more data is acquired from the Finance API. ArrayLists are a great fit to implement this functionality because they can be dynamically modified.

11 User Interface Design and Implementation

Figure 11.1

Figure 11.2

In Figure 11.1 and Figure 11.2, both have similar UI. Figure 10.1 was the interface that was illustrated in report 1. However, one new feature that was implemented in Figure 11.2 to minimize user effort was the dropdown feature. Previously, it is noticeable that there is a sub-menu bar in Fig. 11.1, which provides the user different options to choose where to navigate. However, in order to save time and increase efficiency, there is new dropdown sub-menu in Fig. 11.2. This sub-menu is displayed while hovering over the menu, in this case, “My Portfolio”.

Suppose a scenario where user wishes to check their transaction history. In the previous interface, the user would have to navigate to the “My Portfolio”page and then click “Transactions History”in the sub-menu to reach there, requiring a total of 2 clicks. But, now with this new drop-down feature, user just has to hover over “My Portfolio”and then directly click on “Transactions History”, totaling to only 1 click.

Initially, this upgrade seems very minute. But when observing a user’s full experience on this website, it is expected that they will navigate to multiple pages. So for every page they navigate to, they save at the very least 1 click and suppose they navigated to 10 pages, then it minimizes navigation process to simply 10 clicks, which is half the original effort. Additionally, implementing this feature allows for even further detailed menus to be added, at no expense of user effort. This is exemplified in a scenario of navigating to the Buy/Sell described below.

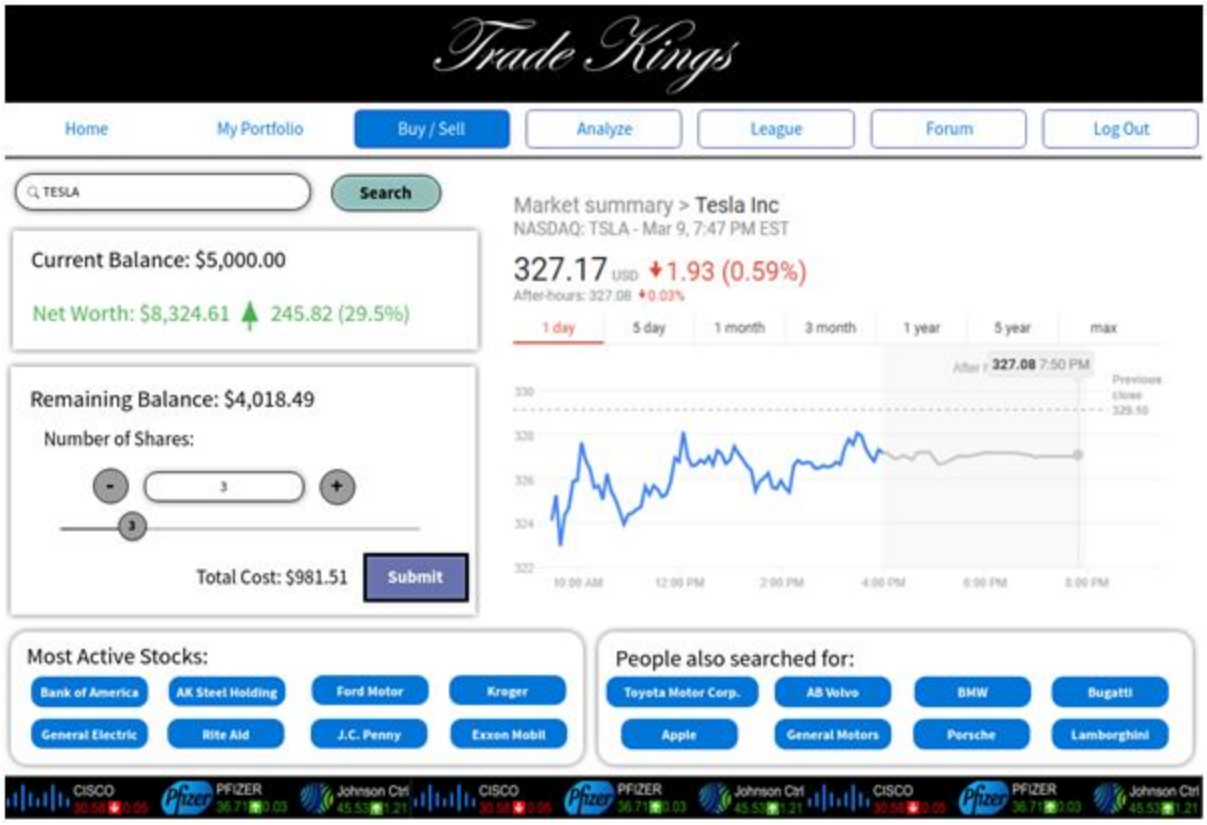


Figure 11.3

The image shown above in Figure 11.3 is a new page that was added. Previously, a user could only buy/sell stocks quickly from either their own portfolio or from the analyze page. However, when a user first opens the site, it was unclear on how to actually execute a trade and where to go. This is because the previous interface did not explicitly say where the user should go to buy or sell stocks. But since this is such a crucial function, the user should exactly know where to navigate to be able to execute a trade. Consequently, we implemented a separate page which allows user to directly buy and sell from. This page further allows user to get a glimpse of analysis on the stock as it displays a graph of the stock's price history and some related stocks. Another convenient feature that is a user can even choose to buy multiple different stocks at once. The UI shown in Fig. 11.3, displays an option for user to purchase a single stock at a time as they have to search for that company. In addition to this, a user can also choose to go to the "Buy Multiple Stocks"option, where they can simply enter in the ticker and the amount of shares for each stock they wish to purchase. In order to make this process faster, the ticker symbols will have auto-complete and the current stock price will be update live so it's easy for the user. This "Buy Multiple Stocks"option will be displayed in the drop-down menu of "Buy/Sell" and is shown when user hovers over it.

The impact of the drop-down feature discussed from Fig. 11.3 can also be seen here. Using the old interface style of the menu, if a user wanted to go to sell cryptocurrencies, they would have to click on "Buy/Sell"->"Sell"->"Cryptocurrencies", requiring a total of 3 clicks. However, with this hover-over feature, this navigation effort is reduced to just one click. This is because even a more detailed breakdown of a sub-menu can be shown on the dropdown. So this greatly increases ease of user access and reduces the amount of effort they need to take in order to navigate and interact with the Trade Kings website.

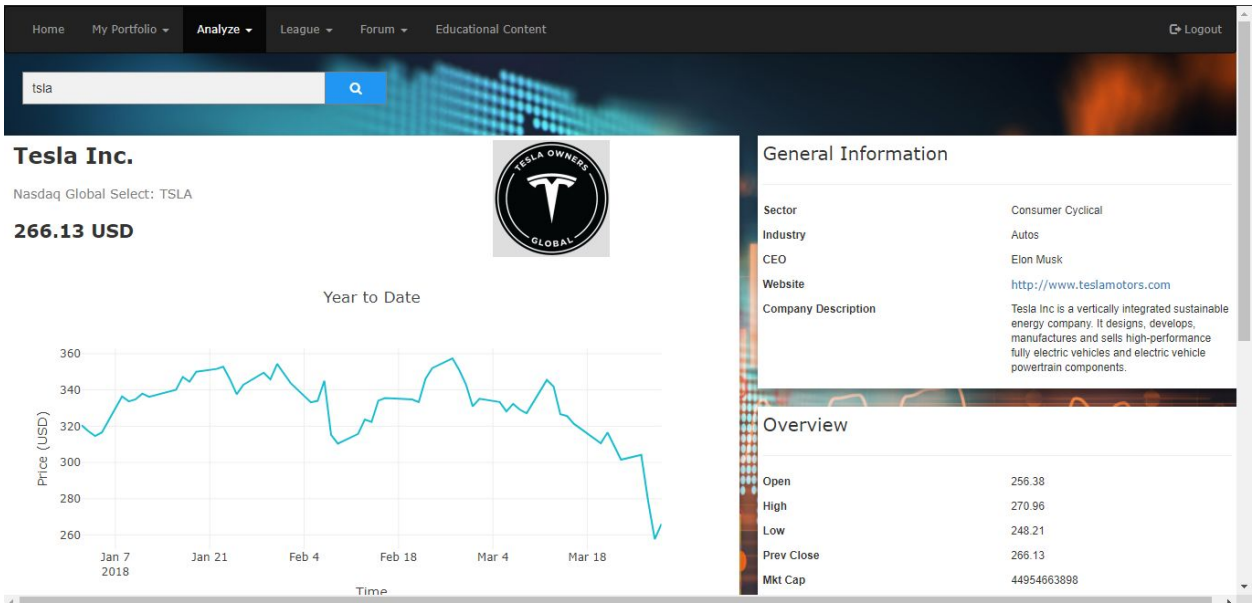


Figure 11.4

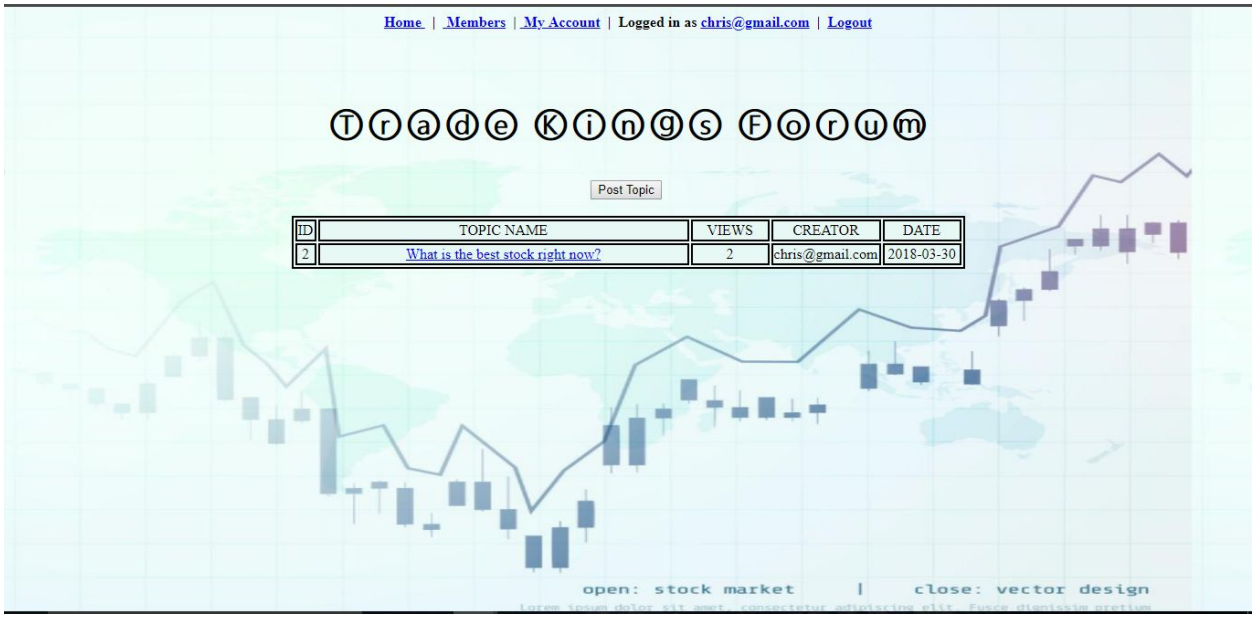


Figure 11.5

In Figures 11.4 and 11.5 are new features that were added before our first demo. The image shown in 11.4 allows the user to Analyze any stock by clicking Analyze in the menu and typing in a particular stock that user wants to examine. Another feature that was added is shown in Figure 11.5. Forum allows the user to post any topics like questions, update, and announcement on any league. This is accessible to any person who is registered in Trade Kings website.

There are numerous other methods to make the existing system more user-convenient and impeccable. The existing web technologies aids in implementation of this goal. One of such technology is JavaScript. JavaScript is a high-level programming language which is untyped and dynamic. Implementation of JavaScript along with CSS and foundation of HTML aids in making the system more robust. The advantage of JavaScript is that it is supported by all the browsers available today without attaching any plug-ins. JavaScript also supports object-oriented, imperative and functional programming styles. It includes the APIs for working with text, dates, arrays and regular expressions.

These vast functionalities of JavaScript enable us to make our system more efficient and user convenient. One of which include user-friendliness for filling the registration form, which can be transformed to improve its convenience. This includes setting up restrictions or constraints over different fields in the registration form, thus improving its accuracy. For example, the fields names cannot include special character types including @, $, %, etc. Also, they could have length restrictions in order to make it memory efficient. It also bears the potential to validate the email id of the user. Meanwhile, JavaScript can check if the password field and confirm password field matches. Thus, guiding the user for the wrong inputs (if any)

straight in the form itself, rather than submitting the form and validating on the server side, JavaScript validates the data on the client-side itself. These facilities aid in improving user-friendliness to a great extent.

The goal is to strive to implement more features and pages like shown above so that using the Trade Kings website becomes seamless. No user should be ever lost on how to navigate through the website to execute a particular task, as the website should be very self-explanatory and intuitive. With continuation of such improvements, the Trade Kings website will become significantly faster, efficient, and easy for the user to use.

12 Design of Tests:

One of the most important processes of our project is testing. It allows the originator to safeguard all the functionality, network security, and ensures the progression of the project. Our plan for testing was divided into five main parts. Testing this parts will be done by each subgroup as we merge all of these functionalities for our first demo.

- **Tour on the Website:** Site navigation should be possible, as well as user login/registration/logout. The plan of testing for this is to ensure the smooth navigation of Trade Kings UI.
- **Integration of Database and website:** Site should be able to recognize: registered users and new users, updated information of any current/new Leagues, user info and the user's portfolio.
- **Working Forum:** Site should have a Forum tab in which it will take users to a page full of interactive chat rooms. Also will give users an option to create their own forum or participate/comment on an existing one.
- **Integration of Finance API and website:** Site should display and interact with finance API (Alpha Vantage API) in which it will provide users with real-time stock and cryptocurrency data. This will help users to analyze, track, and invest in securities.
- **Email System:** Site should allow users to reset/change password, change email, and receive confirmation emails. Email can also be sent to users via alerts and notifications.

Consequently, all of the features and functionality of our project were defined, We then proceeded with analyzing and testing of each. The last thing we need for this testing is to know how we will integrate or merge every functionality that was made by different subgroups.

We have designed the Test Cases via the method of Unit Testing, where we will be testing whether each unit is able to correctly perform its desired actions. We start by first performing test cases on specific functions, and then develop more detailed test cases for larger units, which are comprised of these specific functions (called Integrated Test Cases). We have broken up the units by domain concepts/classes. Furthermore, within each of these Detailed Unit Test Cases, we have identified which use case they refer to. The reason we have decided to develop the test cases in this fashion, is so that we can first test small specific functions. Then, as we gradually build the system and merge various functionalities, the detailed test cases will be used to ensure that these larger, integrated unit are working correctly.

Function Test Cases:

TC-1: isUserAlreadyRegistered(username): boolean

Test Procedure	**Expected Results**
Call Function (Successful)	Searches Player Database for username and returns a boolean: true if username exists, false is username does not exist.
Call Function (Unsuccessful)	Does not correctly search Player Database: returns false when username exists and/or true when username does not exist.

TC-2: isEmailAlreadyRegistered(emailAddr): boolean

Test Procedure	**Expected Results**
Call Function (Successful)	Searches Player Database for email address and returns a boolean: true if email address exists, false is email address does not exist.
Call Function (Unsuccessful)	Does not correctly search Player Database: returns false when email address exists and/or true when email address does not exist.

TC-3: createLeague(leagueID, title, leagueManager): League

Test Procedure	**Expected Results**
Call Function (Successful)	Takes in parameters and returns League object with attributes equal to parameter values.
Call Function (Unsuccessful)	Does not return a properly defined League object.

TC-4: isCapacityLeft(leagueID): boolean

Test Procedure	**Expected Results**
Call Function (Successful)	Searches League Database based on leagueID. If corresponding capacity value > 0, return boolean true; if capacity value = 0, return boolean false.

Call Function (Unsuccessful)	Does not correctly search League Database: returns false when capacity value > 0 and/or returns true when capacity value = 0.

TC-5: changeLeagueSettings(leagueID): boolean

Test Procedure	**Expected Results**
Call Function (Successful)	Takes in a leagueID and changes the corresponding data associated with that leagueID in the League Database; returns boolean value of true.
Call Function (Unsuccessful)	Unable to change the corresponding data associated with the leagueID in the League Database; returns boolean value of false.

TC-6: addPlayer(leagueID, playerID): boolean

Test Procedure	**Expected Results**
Call Function (Successful)	Adds playerID data next to league corresponding to leagueID in the League Database; returns boolean value of true.
Call Function (Unsuccessful)	Unable to update League Database and add playerID value; returns boolean value of false.

TC-7: leagueAlreadyExists(title):boolean

Test Procedure	**Expected Results**
Call Function (Successful)	Searches League Database for title and returns a boolean: true if title exists, false is title does not exist.
Call Function (Unsuccessful)	Does not correctly search League Database: returns false when title exists and/or true when title does not exist.

TC-8: createAccount(playerID, first, last, username, password, emailAddr): Player

Test Procedure	**Expected Results**
Call Function (Successful)	Takes in parameters and returns Player object with attributes equal to parameter values.

Call Function (Unsuccessful)	Does not return a properly defined Player object.

TC-9: login(username, password): boolean

Test Procedure	**Expected Results**
Call Function (Successful)	Searches Player Database for inputted username and password. Returns true if username and password exist and correspond to same playerID; returns false otherwise.
Call Function (Unsuccessful)	Does not correctly search Player Database: returns false when username and password exist and correspond to same player ID and/or returns true otherwise.

TC-10: logout(): boolean

Test Procedure	**Expected Results**
Call Function (Successful)	Logs user out from account and returns boolean value of true.
Call Function (Unsuccessful)	Unable to logout user from account and returns boolean value of false.

TC-11: isEducationalContrib(playerID): boolean

Test Procedure	**Expected Results**
Call Function (Successful)	Searches Player Database based on playerID and checks if user has special credentials. Returns true if user has special credentials; returns false otherwise.
Call Function (Unsuccessful)	Does not correctly search Player Database; returns false when user has special credentials and/or true when the user does not have the special credentials.

TC-12: remainingCapital(playerID): double

Test Procedure	**Expected Results**

Call Function (Successful)	Searches Player Database based on playerID and returns corresponding capital value as a double.
Call Function (Unsuccessful)	Unable to find the player associated with the given playerID or cannot return the correct double walue.

TC-13: requestPrivateLeagues

Test Procedure	**Expected Results**
Call Function (Successful)	Queries League Database for private leagues and returns a display, listing the titles of all the private leagues.
Call Function (Unsuccessful)	Unable to properly query the League Database for private leagues and/or unable to display the list of private leagues.

TC-14: viewLeaderboard(leagueID)

Test Procedure	**Expected Results**
Call Function (Successful)	Searches League Database based on leagueID, and returns a display, listing all of the players in the league and their total capital, ranked from greatest amount of capital to least amount of capital.
Call Function (Unsuccessful)	Unable to find league in League Database and/or unable to display ranked list of all players in league.

TC-15: joinLeague(title, playerID): boolean

Test Procedure	**Expected Results**
Call Function (Successful)	Updates League Database by adding playerID, next to league corresponding to the inputted title.
Call Function (Unsuccessful)	Does not update League Database and add playerID to league corresponding to title, because capacity of league has already been reached (no open spots).

TC-16: searchSecurity(ticker): Security

Test Procedure	Expected Results
Call Function (Successful)	Searches Finance API using security's ticker as the search parameter. Returns security object containing all of the pertinent information about the search security.
Call Function (Unsuccessful)	Unable to retrieve a security object after searching for security using ticker; either ticker value is incorrect or Finance API does not contain this particular security.

TC-17: getFirstName(playerID): string

Test Procedure	Expected Results
Call Function (Successful)	Searches Player Database based on inputted playerID and returns the corresponding player's first name (attribute's name is "first").
Call Function (Unsuccessful)	Unable to find player in Player Database; playerID does not exist and is invalid.

TC-18: getLastName(playerID): string

Test Procedure	Expected Results
Call Function (Successful)	Searches Player Database based on inputted playerID and returns the corresponding player's last name (attribute's name is "last").
Call Function (Unsuccessful)	Unable to find player in Player Database; playerID does not exist and is invalid.

TC-19: getEmailAddress(playerID): string

Test Procedure	Expected Results
Call Function (Successful)	Searches Player Database based on inputted playerID and returns the corresponding player's email address (attribute's name is "emailAddr").

Call Function (Unsuccessful)	Unable to find player in Player Database; playerID does not exist and is invalid.

TC-20: postEducationalContent

Test Procedure	**Expected Results**
Call Function (Successful)	Displays text inputted/sent by Educational Content Contributor onto the Home Page of the Trade Kings website.
Call Function (Unsuccessful)	Unable (error) to display/out the content (text) inputted by the Educational Content Contributor.

TC-21: sufficientCapital(playerID, leagueID, ticker, int shares): boolean

Test Procedure	**Expected Results**
Call Function (Successful)	Searches League Database to find value of capital associated with playerID. If capital >= sharePriceOf(ticker)*shares, returns boolean value true. Otherwise, method returns boolean value false.
Call Function (Unsuccessful)	Does not correctly search League Database; returns false if capital >= sharePriceOf(ticker)*shares and/or true otherwise.

TC-22: purchaseSecurity(ticker, int shares): boolean

Test Procedure	**Expected Results**
Call Function (Successful)	Executes buy order for player based for desired security (given by ticker) and for the requested number of shares. Method adds order/position to the user's portfolio and returns true if successful.
Call Function (Unsuccessful)	Unable to execute buy order and returns false. Error occurs due to incorrect ticker and/or Finance API does not store this particular security.

TC-23: retrieveSecurityInfo(ticker): Security

Test Procedure	**Expected Results**

Call Function (Successful)	Searches Finance API using security's ticker as the search parameter. Returns security object containing all of the pertinent information about the search security.
Call Function (Unsuccessful)	Unable to retrieve a security object after searching for security using ticker; either ticker value is incorrect or Finance API does not contain this particular security.

TC-24: getLeagueName(leagueID): string

Test Procedure	**Expected Results**
Call Function (Successful)	Searches League Database based on inputted leagueID and returns the corresponding league's title.
Call Function (Unsuccessful)	Unable to find league in Player Database; leagueID does not exist and is invalid.

TC-25: getPlayerNames(leagueID, playerID)

Test Procedure	**Expected Results**
Call Function (Successful)	Searches League Database based on inputted leagueID and playerID and returns the corresponding player's name (first and last)
Call Function (Unsuccessful)	Unable to find player or league in League Database; leagueID and/or playerID does not exist and is invalid.

TC-26: getPlayerCapital(leagueID, playerID): double

Test Procedure	**Expected Results**
Call Function (Successful)	Searches League Database based on inputted leagueID and playerID and returns the corresponding player's amount of capital (data type double)
Call Function (Unsuccessful)	Unable to find player or league in League Database; leagueID and/or playerID does not exist and is invalid.

TC-27: getSecurityPrice(ticker): double

Test Procedure	Expected Results
Call Function (Successful)	Searches Finance API using security's ticker as the search parameter. Returns security's current price of data type double.
Call Function (Unsuccessful)	Unable to retrieve security's price value after searching for security using ticker; either ticker value is incorrect or Finance API does not contain this particular security.

TC-28: getSecurityMetric(ticker): double

Test Procedure	Expected Results
Call Function (Successful)	Searches Finance API using security's ticker as the search parameter. Returns security's metric value of data type double.
Call Function (Unsuccessful)	Unable to retrieve security's metric value after searching for security using ticker; either ticker value is incorrect or Finance API does not contain this particular security.

TC-29: setSecurityTargetPrice(ticker, double price): boolean

Test Procedure	Expected Results
Call Function (Successful)	Updates Price Alert Database to store new alert, with specified target price added to the corresponding ticker of the specified security. Returns boolean value true upon success.
Call Function (Unsuccessful)	Unable to updated Price Alert Database and store new alert. Returns boolean value false.

TC-30: setSecurityTargetMetric(ticker, double value): boolean

Test Procedure	Expected Results
Call Function (Successful)	Updates Price Alert Database to store new alert, with specified target metric value added to the corresponding ticker of the specified security. Returns boolean value true upon success.
Call Function (Unsuccessful)	Unable to updated Price Alert Database and store new alert. Returns boolean value false.

Integrated Test Cases:

Note: For the Integrated Test Cases, we have two scenarios: Success Scenario and Alternate Scenario. Success Scenario is when the action that the user wishes to perform is executed. However, Alternate Scenario does not mean that the test case has failed; rather, it defines what will happen when the conditions for a Success Scenario are not met. The test case fails when neither the Success nor the Alternate Scenario occurs, and instead, the system behaves unexpectedly. These failures are implicit and are not defined because of a result to limit redundancies.

- **User Controller**

Test Case Identifier: TC-31 **Use Cases Tested:** UC-1 **Pass / Fail Criteria:** The test passes if the user is able to successfully register an account **Input Data:** username, emailAddr	
Test Procedure:	**Expected Result:**
Success Scenario: User fills out all of the fields on the registration page and enters a unique username.	An account (player object) will be created for the user and it will be stored in the Player Database. An email request will be enqueued into the Mailing Queue so that a confirmation email can be sent to the user.
Alternate Scenario: User does not fill out all of the fields on the registration page and/or does not enter a unique username.	An account (player object) will not be created for the user. User will be prompted to either re-enter a username that is unique and/or make sure that all of the fields are filled out.

Test Case Identifier: TC-32 **Use Cases Tested:** UC-1 **Pass / Fail Criteria:** The test passes if the user is able to successfully register an account **Input Data:** password, emailAddr	
Test Procedure:	**Expected Result:**
Success Scenario:	An account (player object) will be created for

User fills out all of the fields on the registration page and enters a password that meets the criteria (minimum length, at least one lowercase letter, one uppercase letter, and a number or non-alphabet character).	the user and it will be stored in the Player Database. An email request will be enqueued into the Mailing Queue so that a confirmation email can be sent to the user.
Alternate Scenario: User does not fill out all of the fields on the registration page and/or does not enter a password that meets the criteria.	An account (player object) will not be created for the user. User will be prompted to either re-enter a username that is unique and/or make sure that all of the fields are filled out.

Test Case Identifier: TC-33 **Use Cases Tested:** UC-1 **Pass / Fail Criteria:** The test passes if the user is able to successfully register an account **Input Data:** emailAddr	
Test Procedure:	**Expected Result:**
Success Scenario: User fills out all of the fields on the registration page and enters a valid email address.	An account (player object) will be created for the user and it will be stored in the Player Database. A confirmation email will be sent to the user's email via the Mailing Queue.
Alternate Scenario: User does not fill out all of the fields on the registration page and/or does not enter a valid email address.	An account (player object) will not be created for the user, when the systems identifies that the email address does not exist. User will not receive a confirmation email and will have to attempt registration again until the provided email is valid.

Test Case Identifier: TC-34 **Use Cases Tested:** UC -1 **Pass / Fail Criteria:** The test passes if the user is able to successfully log into the "Trade Kings" website. **Input Data:** username, password	
Test Procedure:	**Expected Result:**

Success Scenario: User inputs the correct username and password (username and password both exist in the Player Database and are associated with the same playerID and presses "Log in".	The user controller will match the credentials in the database and the user will be logged in. The Homepage View will be displayed.
Alternate Scenario: User inputs the incorrect username and/or password and presses "Log in".	The user controller will not be able to match the credentials (via the Player Database) and will ask the user if he/she "Forgot Password" or would like to try again.

Test Case Identifier: TC-35 **Use Cases Tested:** UC-7 **Pass / Fail Criteria:** The test passes if the user is able to post educational content directly from the home page. **Input Data:** emailAddr, password, content (text)	
Test Procedure:	**Expected Result:**
Success Scenario: The user inputs the correct email address and password which is made for special access to the website. The user, or ECC, then writes a 'Title" and the content and presses post.	The User Controller will check the credentials in the Player Database. After it gives the special access, the new post will be created and displayed on the website via the Home Page View.
Alternate Scenario: The user inputs the incorrect email address and/or password. Also the user may or may not enter a 'Title' for the content and presses submit.	No new post or educational content will be published due to the incorrect credentials or due to not inputting a 'Title' for the content.

- **League Controller**

Test Case Identifier: TC-36 **Use Cases Tested:** UC -2 **Pass / Fail Criteria:** The test passes if the user is able to create a league using the proper input data **Input Data:** title, playerID, leagueManager, capacity	
Test Procedure:	**Expected Result:**

Success Scenario: Click on "Create League", enter a unique title and set the setting to user's preferences.	User now becomes the league manager and a new league object, with the defined title and capacity has been added in the League Database.
Alternate Scenario: Click on "Create League", and enter a non-unique title and/or ignore the Settings.	No new league objected will be created and stored in the League Database, and the user will be prompted fill out all the fields and/or input a unique title.

Test Case Identifier: TC-37 **Use Cases Tested:** UC -2 **Pass / Fail Criteria:** The test passes if the user is able to successfully change the existing settings. **Input Data:** leagueID, title, capacity	
Test Procedure:	**Expected Result:**
Success Scenario: User clicks on view leagues, selects a league, and makes couple or many significant changes to the settings and presses submit.	The League Controller will add the new settings/data to the League Database, which will then confirm the changes, and display the new settings back to the user.
Alternate Scenario: User clicks on view leagues, selects a league, and makes no changes to the settings and presses submit.	No changes will be made to the settings as there is no data to be altered/changed for the particular in the League Database. Hence, the user will be prompted to try again.

Test Case Identifier: TC-38 **Use Cases Tested:** UC -2 **Pass / Fail Criteria:** The test passes if a league manager is able to remove a disruptive player from a league. **Input Data:** leagueID, playerID	
Test Procedure:	**Expected Result:**
Success Scenario: User clicks on view leagues, selects a league, and scrolls down to the players. The user then removes a player from the league	The League Controller removes the banned player from the league by deleting the playerID for the specific league (leagueID).. The change

and clicks submit.	will eventually be confirmed and the current players will be displayed.
Alternate Scenario: User clicks on view leagues, selects a league, doesn't remove or add any players.	The current user count will remain the same and no changes will occur. The screen will prompt the user that no changes had occured.

Test Case Identifier: TC-39 **Use Cases Tested:** UC -2 **Pass / Fail Criteria:** The test passes if the League Manager is able to add/invite a player to an existing, created League **Input Data:** leagueID, title, playerID, username	
Test Procedure:	**Expected Result:**
Success Scenario: Click on "Add Player" and searches the market for a new trade	Player with corresponding username is added to the league in the League Database and the league's capacity value is updated.
Alternate Scenario: Click on "Add Player" and either enter a non existing usermax ame and/or league is at capacity.	Player will not be added to the league in the League Database. User will either be prompted to enter username again or given error message stated that "League is completely full".

- **Profile View**

Test Case Identifier: TC-40 **Use Cases Tested:** UC -3 **Pass / Fail Criteria:** The test passes if a user is successfully able to join their desired league. **Input Data:** playerID, leagueID, title, capacity	
Test Procedure:	**Expected Result:**
Success Scenario: User views the available public leagues and selects the one they want to enter by pressing join next to the title of the desired league.	The user is added to the league; the League Controller updates the League Database by adding the player's playerID next to the proper leagueID.

Alternate Scenario: User attempts to join the league, but the league is full with no slots available.	The user is not added to the league and the League Database is not changed/updated. System informs user that selected league is full and prompts user to select another one.

Test Case Identifier: TC-41 **Use Cases Tested:** UC -3 **Pass / Fail Criteria:** The tests passes if a user is successfully able to request permission to join a Private League **Input Data:** playerID, leagueID, title, emailAddr, leagueManager	
Test Procedure:	**Expected Result:**
Success Scenario: The User clicks “Display Private Leagues” and is shown a list of all the private leagues allowing requests. Player clicks “Request Permission to Join” next to desired league’s title	An email is sent to the League Manager of the requested league.
Alternate Scenario: The User clicks “Display Private Leagues” but there are currently no leagues that are allowing requests.	System informs user that no private league is current allowing requests is join and tells user to try again at another time.

Test Case Identifier: TC-42 **Use Cases Tested:** UC -4 **Pass / Fail Criteria:** The tests passes if a user is successfully able to search for a stock or cryptocurrency and receives current updated results on their prices and value of metrics. **Input Data:** playerID, ticker, name	
Test Procedure:	**Expected Result:**
Success Scenario: The player searches for desired stock or cryptocurrency through the Trade System using the ticker or name.	The system, using a finance API, outputs the corresponding Security and all the relevant data. The player is able to analyze the up-to-date data.

Alternate Scenario: User inputs an incorrect/non-existent ticker or security name, or the Finance API does not have track/store that particular security.	System will either prompt user to search again or inform the user that it does not have data on the particular security that had been requested.

Test Case Identifier: TC-43 **Use Cases Tested:** UC-6 **Pass / Fail Criteria:** This tests passes if a player is successfully able to access the investment forum and receive relevant information based on their search. **Input Data:** playerID, keywords/search parameters, discussion topics	
Test Procedure:	**Expected Result:**
Success Scenario: Player searches for desired information on Investment Forum using keywords and search parameters.	The system runs/searches the discussion threads against the search parameters based on the discussion titles and tags. System is able to find matches and displays them to user as a list of discussion topics.
Alternate Scenario: Player searches for desired information on Investment Forum, however, inputed keywords are not associated with any discussion threads.	Systems runs search but does not find any matching discussion threads. System either prompts user to refine search or informs user that desired discussion topic does not yet exist.

Test Case Identifier: TC-44 **Use Cases Tested:** UC-6 **Pass / Fail Criteria:** The test passes if the user is able to make a new post on the Investment Forum. **Input Data:** title (of post), playerID, content of post (text)	
Test Procedure:	**Expected Result:**
Success Scenario: User clicks on New Post. User is able to set “Title” of new post and add content.	The post made by the user is now on the forum posts. The system is updated to have the post in memory

Alternate Scenario: User clicks on New Post. There are no changes made when the user hits the button and the user is unable to add content.	No changes will be made to the forum as there are no new posts for the user controller to add. Hence, the user will be prompted to try again.

Test Case Identifier: TC-45 **Use Cases Tested:** UC-6 **Pass / Fail Criteria:** The test passes if the user is successfully post a comment on a discussion thread in the investment forum. **Input Data:** playerID, content of comment (text)	
Test Procedure:	**Expected Result:**
Success Scenario: User goes on a forum and clicks "New Comment". A box appears for the user to post a comment.	After the user clicks "New Comment", they are able to post desired comment. The forum is also updated with the comment.
Alternate Scenario: User goes on a forum and clicks "New Comment". Nothing happens when the submission is clicked.	No changes will be made to the forum as there are no new comments for the user controller to add. Hence, the user will be prompted to try again.

- **Trade System**

Test Case Identifier: TC-46 **Use Cases Tested:** UC -5 **Pass / Fail Criteria:** The test passes if the user is able to buy desired security at requested number of shares. **Input Data:** playerID, ticker, name, (number of) shares, capital	
Test Procedure:	**Expected Result:**
Success Scenario: Player searches for an existing security via the ticker or name and requests number of shares to purchase; player has sufficient capital to	System executes purchase using real-time market data and adds position to player's portfolio.

purchase desired amount of shares.	
Alternate Scenario: Player either searches for non existing security based on incorrect ticker or name, or the player does not have sufficient amount of capital to purchase desired amount of shares.	System is not able to execute purchase and add position to player's portfolio. System prompts user to search again or decrease amount of shares.

Test Case Identifier: TC-47 **Use Cases Tested:** UC-5 **Pass / Fail Criteria:** The test passes if the user is able to sell desired security at requested number of shares. **Input Data:** playerID, ticker, name, (number of) shares, capital	
Test Procedure:	**Expected Result:**
Success Scenario: Player searches for an existing security via the ticker or name and requests number of shares to sell; player has either equivalent or greater number of security's shares in portfolio.	System executes sale using real-time market data and liquidates/decreases position in player's portfolio.
Alternate Scenario: Player either searches for non existing security based on incorrect ticker or name, or the player does not have sufficient sufficient number of shares to sell desired amount of shares.	System is not able to execute sale and change position in player's portfolio. System prompts user to search again or decrease amount of shares.

Test Coverage:

The goal of our Test Coverage is to implement test cases that have tested every possible edge case of our method we will use. This is not realistic, however, so we try to cover all possible cases by implementing Equivalence Testing. In Equivalence Testing, we will divide all of the possible inputs for a particular method into equivalence groups, and confirm that the method behaves the same on each group. Such is the case with a majority of our methods and test cases, where each set of inputs can primarily be broken down into two equivalence classes: one valid equivalence class and one invalid equivalence class.

Integration Testing Strategy:

The integration testing strategy that we will implement is the Sandwich integration approach. The Sandwich approach is a combination of the top-down and bottom-up approaches. We do this by starting from both ends of the system and incrementally using components from the middle level (known as the target level) in both directions. We have chosen Sandwich integration because our goals for the first demo are to develop the basic User Interface of the system and the various databases that many intermediate functions will access and reference. Therefore, some of our objectives our on the top layer and some our on the bottom layer. As we progress with the project, we will build functions that use these database and smaller components. In addition, we will also build the functions that support various buttons and action on the user interfaces that users will interact with. As a result, as the project progresses, we will slowly make our way towards the target level. The way we will conduct this integration strategy is defined by our test cases above, where the function test cases are for testing the smaller components, and the more detailed Integrated Test Cases will be used as we merge these smaller functionalities into integrated units.

13 History of Work, Current Status, and Future Work

Throughout this spring semester the group completed most if not all of desired requirements and use cases. This was achieved by splitting the group of nine into 3 sub-groups of 3, where each sub-group took on their own functionalities. These functionalities ranged from email alerts to a forum for users to interact with each other.

Our first requirement was to complete Report 1 and Report 2 in which it was met on time. This will helped us out to come out with a successful demo that was held on March 29th that consisted on showing the following functionalities: Web Design, User Registration, Stock/Cryptocurrency Indices Information, Forum and Email Queue.

Also, while working on the separate reports the individual sub groups worked on their desired functionalities simultaneously. One group consisting of Diego, Chris C. and Chris S. worked on the email queue, forum, email alerts and user profile. The email queue used to verify a user was completed on March 12, 2018. On March 15th, 2018 the group was also able to complete an investment forum consisting of private and personal account pages for Trade Kings users to interact on. Prior to that 3 databases were constructed in order to store user data, user replies and topics posted by a user. Around the same time period another sub-group consisting of Bhargav, Ray and Karlo were able to implement a database that was accessible online through amazon as opposed to the original database that were tested using mySQL on a local host. The third group, consisting of Akarsh, Nakul, and Krutant, focused on the functionalities of Researching Stock/Cryptocurrency Markets, Executing Trades, and Forecasting market trends and price. The Research Page functionality was completed on March 30th, 2018, and the other two functionalities (Executing Trades and Forecasting) are currently on track to be completed by Demo 2.

The current status of our projects before demo 2 is still based on what we've have planned after demo 1. We, started of by re-grouping our team's subgroup depending on the functionality each member wants to work on. One subgroup is going to improve our User Integration and add some features on our website. Some of those features are registering through a social media account,and text message alerts Another subgroup will be working on upgrading our email alerts and portfolio. Email functionality will not only send notifications when a user registers, but also allows a user to set target prices for a certain stock and then receive notifications if a stock has reached the target value that was sent. The last subgroup will be working on trading stocks, it will allow user to buy and sell stock depending on users preferences the amount of stock a user can buy or sell can be adjusted.

Key Accomplishments:

- User Interface was created based on the designs from Reports 1 and 2.
- Login page is updated to be more user friendly and accessible to users.
- Implemented a database that was accessible online via Amazon as opposed to the original database that were tested using mySQL on a local host.
- The features Researching Stock/Cryptocurrency Markets, Executing Trades, and Forecasting market trends and price were added.
- Email alerts are able to send notifications when a new user registers.
- Analyzing stock were made available for users to use to examine a particular stock

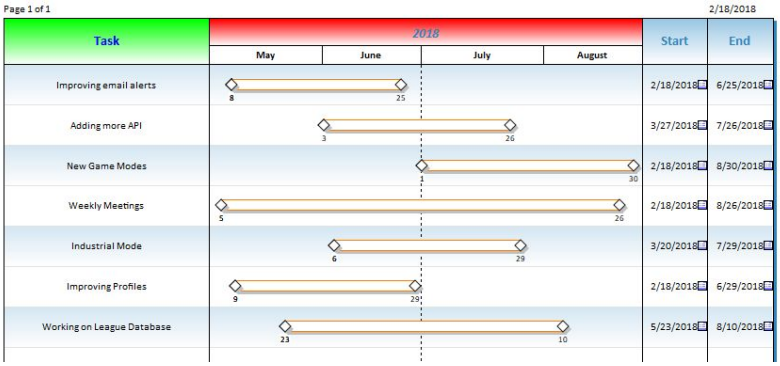


Figure 14.1

Though the development of Trade Kings will cease production in May, the team have planned realistic goals for future possibilities. Many of the key ideas and functions are only the start of what this team could accomplished if the project were to stay in production over the summer of 2018. More modes were wanted to be developed for the players. The team want the users to have a variety of options when using the Trade Kings website. The modes would have also allowed development as the key factors could play a role in the involvement and knowledge of our players. Along with the extra options, the team would plan on the user interface and clean up the site where it needs it the most. The website has come a long ways since the beginning of January, but there is always room for improvement. The site can be a little slow at times, and the team wanted to improve on performance as much as they could. The group thinks Trade Kings could be something that would be very easy to use no matter what age. But a team needs time in order to become successful at building a friendly user interface. The final improvement the team wanted to state is more work with APIs and using tools to the website's advantage. The project used many sources to gain valuable information. Whether this would deal around stock prices, or relevant incoming news about a player's investments. The group believed that the project grew strong on incoming courses from APIs. The team wished they could continue their great work with Trade Kings. But each group member knows that the site had to come to an end at some

point. If given the time and resources, the team hopes to revisit the projects and hopefully add more features that they promised.

14 References

[1] Stack Overflow, “Alternative to google finance api - Stack Overflow.” https://stackoverflow.com/questions/10040954/alternative-to-google-finance-api. [Online]
[2] Investopedia, “Bid-ask spread — Investopedia.” http://www.investopedia.com/terms/b/bid-askspread.asp. [Online].
[3] Investopedia, “Short (or Short Position) definition — Investopedia.” http://www.investopedia.com/terms/s/short.asp.[Online].
[4] Investopedia, “Limit order definition — Investopedia.” http://www.investopedia.com/terms/l/limitorder.asp. [Online].
[5] Investopedia, “Stop order definition — Investopedia.” http://www.investopedia.com/terms/s/stoporder.asp. [Online].
[6] Wikipedia, “Stakeholder.” http://en.wikipedia.org/wiki/Stakeholder_(corporate). [Online].
[7] Wikipedia, “Bootstrap (front-end framework).” http://en.wikipedia.org/wiki/Bootstrap_(front-end_framework). [Online].
[8] Wikipedia, “Openid.” http://en.wikipedia.org/wiki/Openid. [Online].
[9] Wikipedia, “Oauth.” http://en.wikipedia.org/wiki/OAuth. [Online].
[10] Wikipedia, “Application programming interface.” http://en.wikipedia.org/wiki/Application_programming_interface. [Online].
[11] Wikipedia, “Html.” http://en.wikipedia.org/wiki/Html. [Online].
[12] Wikipedia, “Css.” http://en.wikipedia.org/wiki/Css. [Online].
[13] Cohn, Mike. “Estimating With Use Case Points.” www.cs.cmu.edu/~jhm/DMS%202011/Presentations/Cohn%20-%20Estimating%20with%20Use%20Case%20Points_v2.pdf. [Online]