Diamond Hands Investment Fantasy League

Report #3

Software Engineering

14:332:452

Repository Link

Group 3:

Aarushi Pandey

Aarushi Satish

Apoorva Goel

Christine Mathews

David Lau

Jacques Scheire

Jawad Jamal

Krishna Prajapati

Riya Tayal

Sahitya Gande

Yati Patel

Yatri Patel

Table of Contents

1 Contribution Breakdown

All team members contributed equally to this project.

2 Summary of Changes

- Adjusted business goals to make our product more oriented around education
    - See UC-11 update
    - Tutorial slides for first-time users
    - AI algorithmic trading analysis, graphics, and explanations added as part of UC-4
- Truncated glossary of terms to only include terms specific to our system
- Further defined the technical requirements for users to access the application
- Edited UC-11 to further emphasize our educational business values
    - UC-11 now includes Summary page that allows users to see their progress in a more quantifiable manner, thus helping us align our application with our business goals and make it more educational

- Updated interaction diagrams to more accurately depict workflow
- Updated class diagram to include additional get/set methods for entities
- Removed feature to view past leagues, moved to future goals
    - This feature does not add as much to our application in terms of our business goals as other features, such as the recently modified UC-11
- Changed how private leagues are joined to make the User Interface more intuitive and meet REQ-27
    - Users now see all public and private leagues in a table and can join by simply clicking the button next to the league. They no longer need to copy-paste a league code to join a public/private league
- Replaced REQ-9 with a new requirement that allows user to sort and filter joinable leagues based on certain criteria, including league name, starting balance, start/end dates, and publicity
- Replaced REQ-16 with a new requirement that allows users to view how a bot will trade during the course of a stock's historical prices
- Removed dashboard term from glossary and feature from mock-ups
    - Dashboard feature has been replaced with Central League Pages

3 Customer Statement of Requirements

In the modern day, many students seem to have similar aspirations for their future. These young adolescents plan to go to college, get a degree, find a well-paying job, and simply live their lives working. Most of our time, as youths, is spent learning how to make money with hard work and dedication, but not nearly enough time learning how to make money with money. Investing is a key factor in achieving long-term financial security by creating multiple sources of income for yourself. For many, making sound investments is a critical step towards retirement and financial independence. The notion of investing in the stock market is actually a very well-known and well-tried passive form of income. However, even though millennials acknowledge the fact that investing in the stock market is an intelligent decision, many still have an abundance of valid concerns when it comes to investing, describing the market as complicated, risky, and overwhelming [1].

This fear similarly extends to older generations. Investing is a timeless decision and it is never too late to start. When one makes the conscious decision to set aside money to invest, they are developing discipline by displaying a clear concern for their future and financial wellbeing. Unfortunately, an April 2020 poll indicated that a mere 55% of Americans own stocks [2], which means that a significant portion of the population have absolutely no stake in the market. While investing is important for the young, who can tolerate more risk and have time on their hands to let their investments grow exponentially, it is equally as important for individuals that are closer

to retirement. They tend to have much more buying power than younger generations, which can contribute to larger returns. Even small percent gains for this demographic can result in a significant monetary gain. It may even provide income well after retirement.

This lack of financial literacy is one of the main reasons why many Americans face problems with saving and investing. Approximately half of the American population does not have enough money saved to retire comfortably. Financial literacy is crucial because it equips people with the knowledge and skills necessary to manage money and grow it effectively [3]. Additionally, it aids in making smart, well-calculated financial decisions that help them achieve financial freedom. To make matters worse, a large portion of Americans will not be able to rely on Social Security as a passive income in retirement, since the reserves are projected to be depleted by 2035 [4]. This makes it more imperative than ever for people of all age groups to have multiple sources of income, specifically passive incomes like investments.

While getting involved in the stock market does not require a vast knowledge of technology or years of study, getting involved without the proper background knowledge and experience can be a highly risky endeavour. Especially in light of recent news relating to GameStop and AMC, the market has proven itself to be quite volatile and unpredictable for rookie investors. With new, attractive and accessible online brokers like Robinhood, getting started with investing is the easiest it has ever been. Unfortunately this also means that there will be inexperienced investors that are tempted to throw their money at the market and make uninformed purchases. As we have seen in recent years, this can be a costly mistake and many young, eager investors have suffered as a result.

To combat this national problem, our goal is to offer a platform where both new and experienced investors can trade stocks with virtual currency to learn how to intelligently invest their capital.

New Investor Perspective:

> As an inexperienced investor, I have very little awareness of how stocks operate. I want to start investing in the stock market, but I'm terrified at the idea of losing money. I want to learn more about the market before I start investing, but I feel that the only effective way to learn would be through hands-on experience. An app like Diamond Hands Investment Fantasy League (DHIFL) would fix this problem for me entirely by letting me trade with virtual currency. I can watch my stocks rise and fall as if I had invested in the real market, yet still sleep soundly knowing that my money is safe.
>
> Additionally, I tend to find myself overwhelmed by the many resources that are at my disposal. There is so much material on the stock market and I have no understanding of its meaning or value at this stage in my stock market career. DHIFL would help me tremendously through not only their tooltips for important financial terms, but also through their specialized news feed that shows me exactly what information is relevant to my portfolio.

I also have friends who want to learn more about the market, so this would be a great opportunity to see how they would invest. We would be able to help each other learn more about the market by tracking each other's performance in comparison to ours. Through the app, I am also able to compete against my friends, which motivates me to learn more about the market in order to give myself an edge when climbing the league rankings. I also have the ability to see other users' portfolios, which I can use to size up other players and compare my performance to theirs.

Experienced Investor Perspective:

As an experienced investor, I have already bought and sold stocks through a real broker. I've learned enough to make a profit in the long run, but there are some risky strategies I would like to test out in order to refine my trading tactics. I want to experiment with these new ideas to find what generates the most profit, but I do not want to risk losing actual money in the process. DHIFL provides a stress-free environment where I can invest virtual money and see how my strategies might play out in the real stock market. The application has an extensive portfolio management system that allows me to place market and limit orders on equities. With a variety of options to play around with, there are an endless amount of scenarios that I could implement at no risk.

I would also appreciate having access to a community that will bring a fresh perspective and help me adapt to newer strategies. With the option to join/create leagues and engage in healthy competition, there are so many valuable insights that I can gather from my competitors. I can also compare my portfolio to others through the league rankings, and use other players as a metric to assess the effectiveness of my own investments.

I would want an application that resembles the online broker websites that I currently use to invest because I enjoy their design layout. I would also prefer to have access to metrics to assess my portfolio's growth. I feel that comparing my performance to index funds, such as the S&P 500, as well as assessing various graphs and historical data of all my equities and my portfolio gives me a more complete view of how much of my money's potential is being achieved. Additionally, the app offers an optional AI trading bot that can be added to my leagues. The bot's portfolio can be viewed as a real player that effectively follows a trading strategy, so it can be used as a baseline metric for my portfolio to be compared to.

While our application would help both new and experienced investors alike, the idea of a stock market fantasy league is not a novel idea. Trading stocks, competing with friends in leagues, and analyzing one's stock data are all features that have been implemented in many past projects, one such project being the Paramount Investments League from Spring 2014. However, our platform will improve upon previous implementations through a new feature and a more modern implementation. Notably, we offer users the option to include AI players in their leagues. This

gives new users the opportunity to experience the game alone, which would help those who may be apprehensive about joining leagues with real players. As for more experienced players, the AI component offers another benchmark to evaluate players' performances. The AI will exercise various algorithmic trading techniques, which will utilize data gathered from our stock price endpoint to determine future decisions. The feature that most resembles our AI system is in Group #2's Spring 2014 Project, where they analyzed Twitter posts to determine sentiment value of a stock. However, an AI bot is clearly a distinct approach to this project. It is also worth noting that the last published project to implement a stock market fantasy league was made seven years ago, so we also plan to use more current and cutting-edge technologies to implement our idea. We will also utilize the IEX Cloud API, a modern low-latency API, for gathering stock data and news. The combination of these new technologies will allow our application to have a modern feel with added user responsiveness.

Overall, investing in stocks may seem like a daunting and intimidating task. There are many financial terms, numbers, and graphs that may put off and even overwhelm new investors. For experienced users, potential losses may prevent them from trying new approaches. Our platform will strive to break these barriers by presenting all information in an intuitive, user-friendly, and visually appealing manner that will accelerate the learning process for users of all experience levels.

4 Glossary of Terms

4.1 League

A market simulation that has a set of rules decided by a league manager. It allows for several users to compete.

4.2 League Manager

League manager has control over the settings of the league. They may choose the player starting balance, fees for trading stocks, and limit the number of trades within a trading day. League managers also have control over privacy settings, such as determining who is able to join their league. League managers also have the ability to provide additional funds to players within the league as well as remove players within the league.

4.3 Portfolio

Detailed list of all equities owned by a player in a specific league. Will display profits, losses, total return as well as other key information associated with the equity

4.4 Market Order

An order placed for immediate execution in the market.

4.5 Buy Order

An order placed to purchase a specific number of shares of a stock.

4.6 Sell Order

An order placed to sell a specific number of shares of a stock.

4.7 Limit Order

An order to either buy/sell a stock at a user-defined price. Investors set a maximum amount they will pay for a stock when buying shares and a minimum amount when selling shares.

4.8 Transaction

An agreement between seller and buyer to exchange cash for an asset.

4.9 Ticker Symbol

A unique arrangement of letters used to represent a security for the purpose of trading.

4.10 Leaderboard

Ranking system within the league based on total portfolio value of each player.

4.11 Transaction History

Users can view transactions from all league members from current leagues.

4.12 Symbol Lookup

A search bar that allows users to access information about particular stocks.

4.13 Trading Bot

An artificial intelligence system that competes with other players in leagues to enhance their skill level.

5 System Requirements

5.1 Business Goals

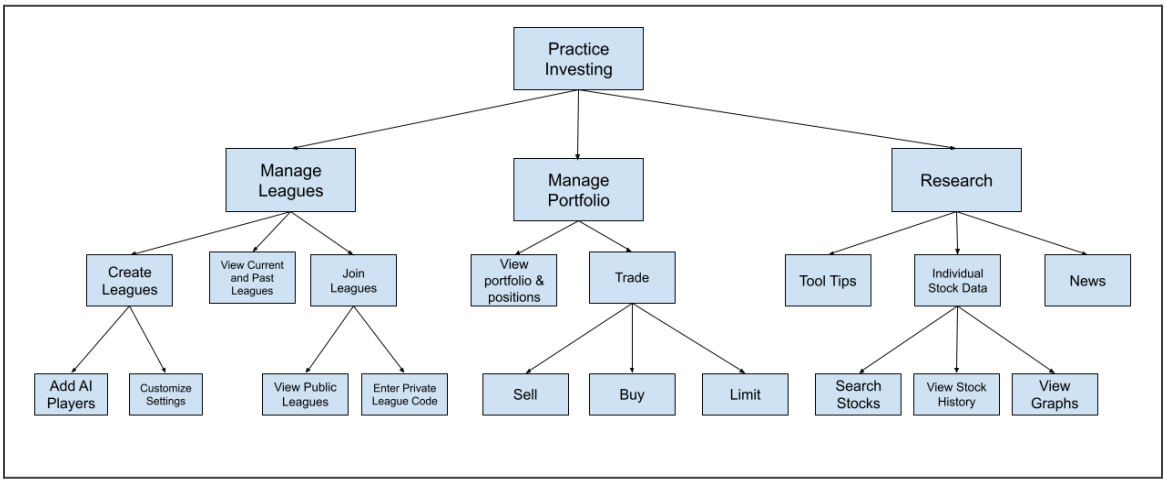

Figure 5.1. Business Goal Diagram

5.2 Enumerated Functional Requirements

PW = Priority Weight (from 1 to 5)

5.2.1 User Profiles

Table 5.1. User Profile Requirements

ID	PW	Requirement Description
REQ-1	2	The system will allow users to register an account and log in with their credentials.

5.2.2 Portfolio Management

Table 5.2. Portfolio Management Requirements

ID	PW	Requirement Description
REQ-2	5	The system will allow users to place market and limit orders on equities.
REQ-3	5	The system will allow users to view their positions, transactions, and portfolio value graph in each of their leagues.

5.2.3 Graphs & News

Table 5.3. Graphs and News Requirements		
ID	**PW**	**Requirement Description**
REQ-4	5	The system will allow users to view prices, graphs, and metrics for individual equities.
REQ-5	2	The system will provide users with a curated news feed for equities in their portfolio.
REQ-6	1	The system will provide users with definitions on financial terms and instruction on how to analyze the graphs.

5.2.4 League Creation w/ Customizable Settings

Table 5.4. League Creation and Customization Requirements		
ID	**PW**	**Requirement Description**
REQ-7	4	The system will allow users to create new leagues with settings to define limits on gameplay.
REQ-8	2	The system will allow users to join private leagues with a password from the league manager or public games with unknown users.
REQ-9	1	The system will allow users to filter and sort leagues by different settings, including league name, starting balance, start date, end date, and publicity.
REQ-10	3	The system will allow users to view the portfolio and positions of other users in their league.
REQ-11	2	The system will allow users to view league rankings through a league leaderboard.

5.2.5 Optional AI Players to Leagues

Table 5.5. AI Player Requirements		
ID	**PW**	**Requirement Description**
REQ-12	3	The system will allow league managers to add AI players to their league.

REQ-13	1	The system will allow the user to view how an AI player would trade a particular stock using the stock's historical data
REQ-14	1	The system will limit the AI player from exceeding a certain number of trades per day, determined by the league manager.
REQ-15	5	The AI Player should be able to decide what market trades to make throughout the day, regardless of user activity.
REQ-16	3	Multiple AI players in the same league should not have the exact same portfolio.

5.3 Enumerated Non-Functional Requirements

Table 5.6. Nonfunctional Requirements

ID	PW	Requirement Description
REQ-17	4	Users are able to access the application via any web browser compatible with ES6 Javascript, e.g., Chrome 51.
REQ-18	2	The application will be consistent across all web browsers.
REQ-19	2	User accounts will be secure and unable to be accessed by other users.
REQ-20	3	The user interface will be responsive with at least eighty percent of actions taking less than 1000 milliseconds to respond.
REQ-21	4	The system should be able to service at least 100 users simultaneously without performance degradation.
REQ-22	3	Equity price data from endpoints will be accessed in under 1000 milliseconds.
REQ-23	3	The system should be able to handle at least 500 transactions per user per day.

5.4 User Interface Requirements

PW = Priority Weight (from 1 to 5)

Table 5.7. User Interface Requirements

ID	PW	Requirements

REQ-24	1	Registration/Login Page: The user will be able to create an account to play the game.
REQ-25	3	Leagues: The user will be able to view active leagues.
REQ-26	2	League Creation + League Manager Settings: The user will be able to create leagues with settings of their choice.
REQ-27	2	Join League: The user will be able to join a specific league or a random league that is active.
REQ-28	5	Portfolio Overview: The user will be able to view their current holdings and net worth.
REQ-29	4	Order Placement: The user will be able to place orders.
REQ-30	4	Individual Stock: The user will be able to search up a specific stock and view data, graphs and history on it.
REQ-31	2	News: The user will have the option to view news articles based on their holdings.
REQ-32	3	Leaderboard: The user will be able to view their standing in the league.

5.4.1 Registration/Login Page

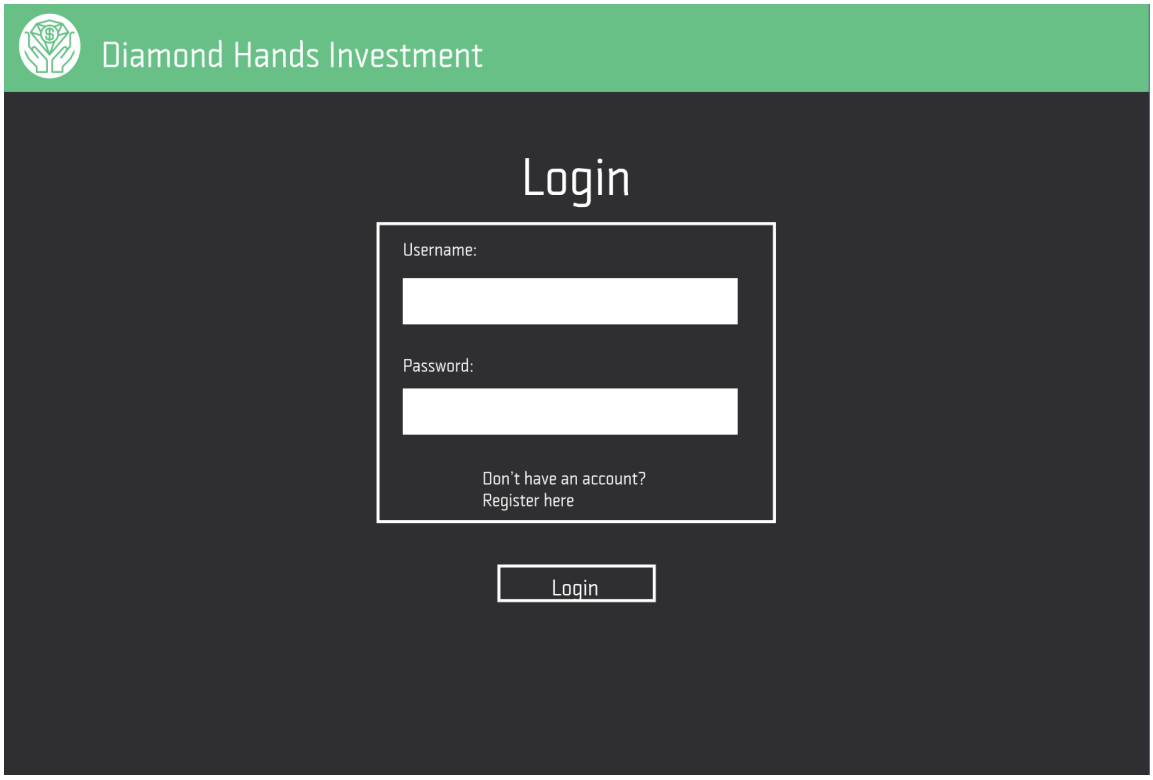

Figure 5.1.(a) Login Page

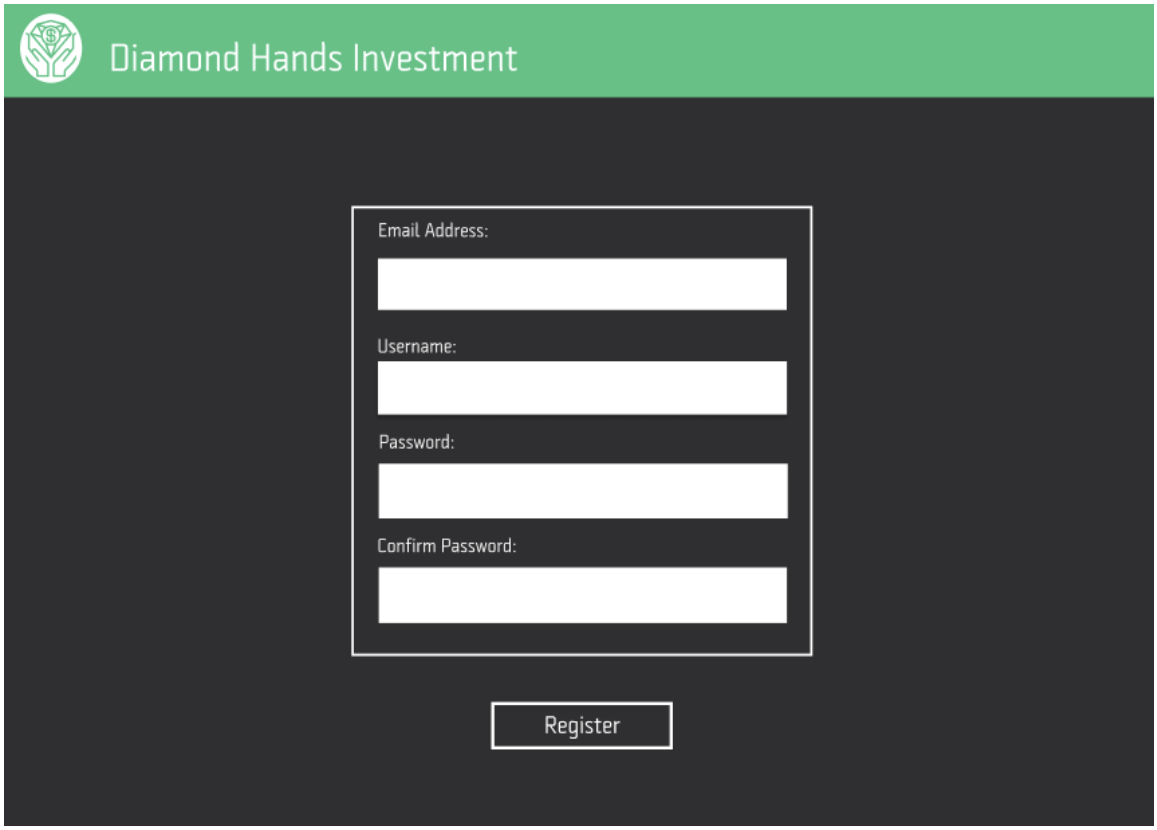

Figure 5.1.(b) Registration Page

The login page is shown in Figure 5.1(a). Here, users can put in their identification information to gain access to their fantasy league account. If they do not yet have an account, they can click “Don’t have an account? Register Here” to register. This will redirect the users to the page shown in Figure 5.1(b), where they can enter the information needed to create an account. Once they login, they will be directed to the leagues page shown below, in Figure 5.2.

5.4.2 Leagues

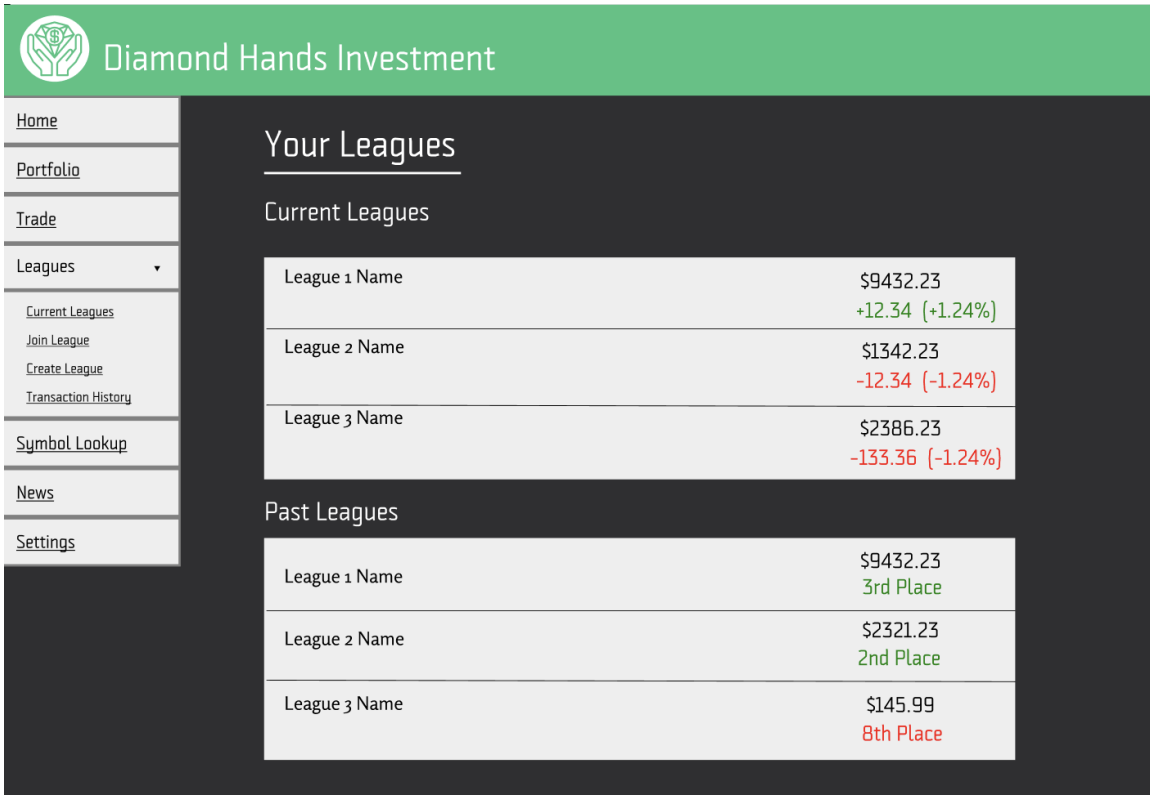

Figure 5.2. Leagues Page

The leagues page shows what leagues the user is participating in. There are options to create a league or join another league in the navigation bar on the left. If a league name is pressed, it will direct the user to the portfolio overview page for that specific league. If the create leagues button is pressed, the user will be prompted to the league creation and league manager settings slide below. If the join league button is pressed, the user is redirected to the join leagues page.

5.4.3 League Creation + League Manager Settings

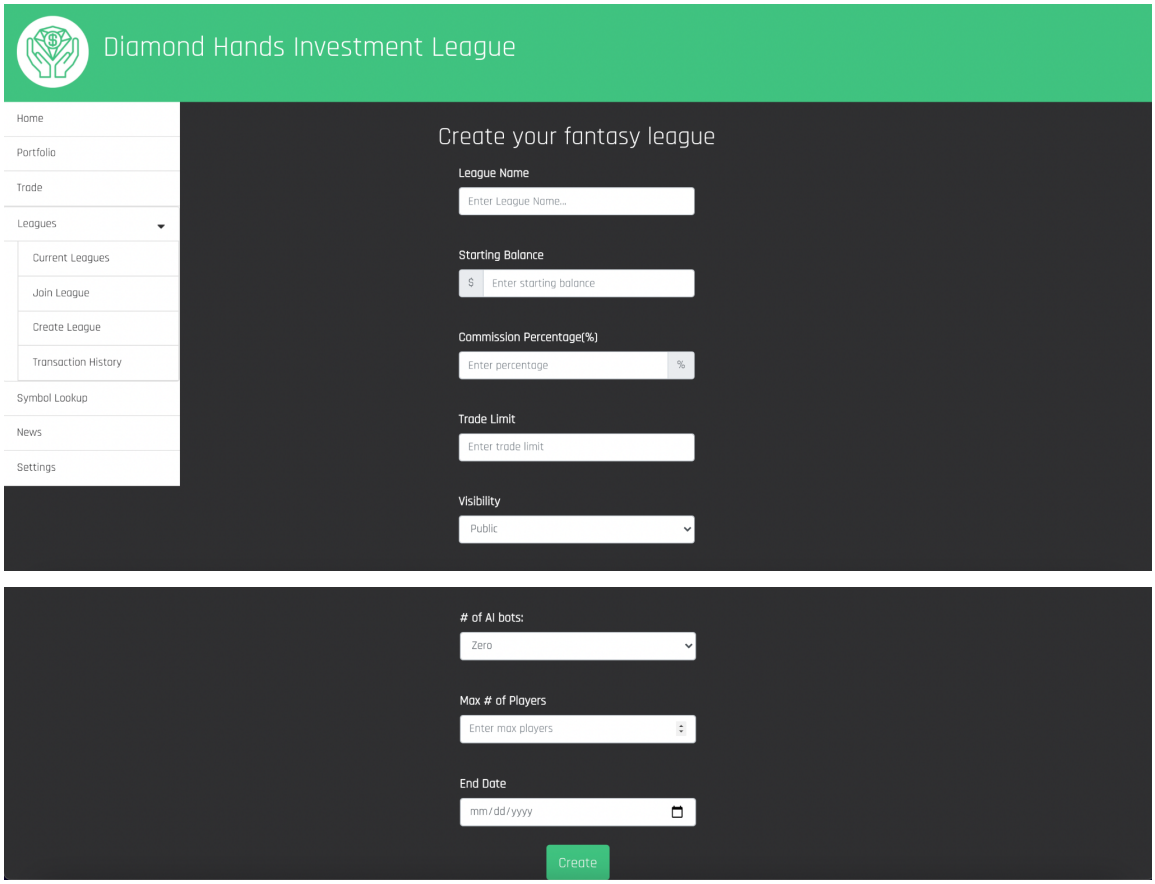

Figure 5.3. Leagues Page

The league creation page allows the league manager to create their league. The league manager has the ability to customize the league according to his/her preference and must fill out the fields above. League managers can set a custom starting balance for all the players in their league. The league manager will also have the ability to limit the amount of stocks each player can trade per day, the number of players that can participate in their league, as well as having the option to add AI players to their league.

5.4.4 Join Leagues

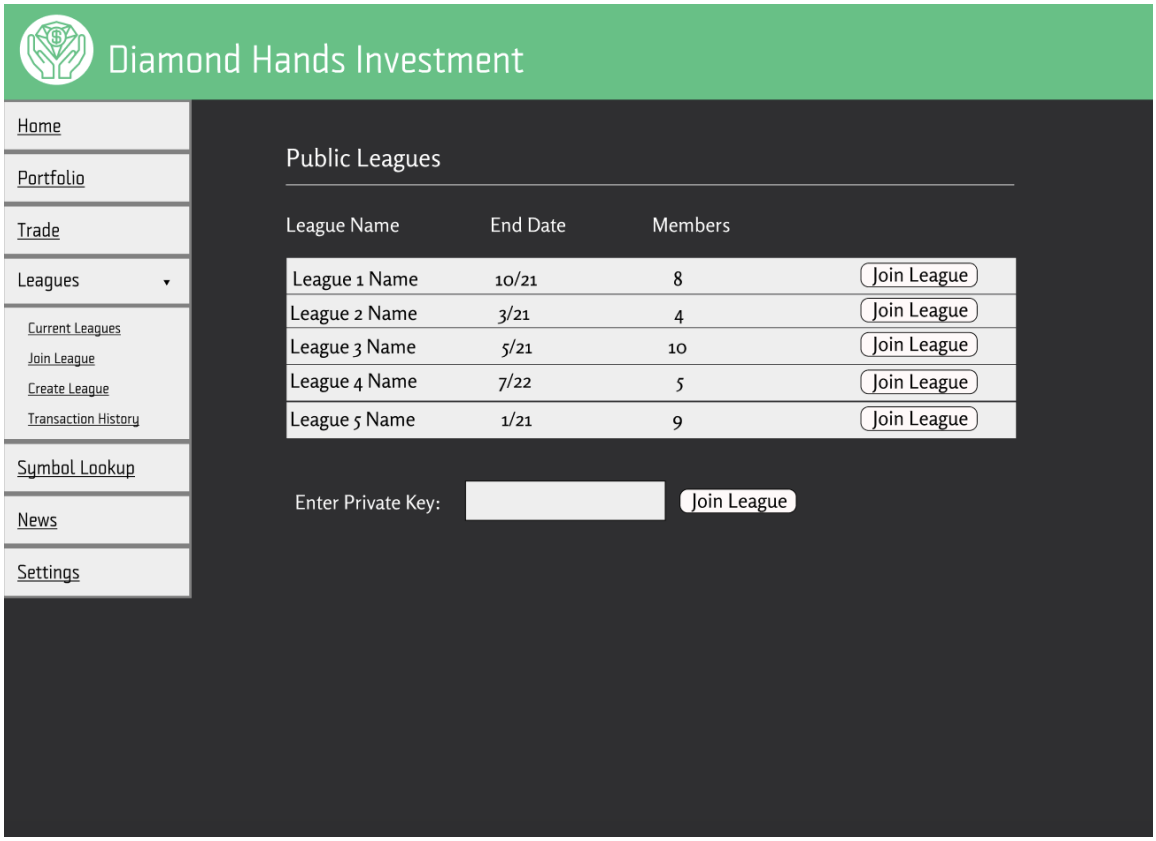

Figure 5.4. Leagues Page

The user can reach this page by clicking on "Join League" in the navigation bar on the left side of the page. On this page, the user will be able to join public or private leagues. To join a private league, the league manager will receive a code that will serve as a password for users attempting to join the league. When users enter that code, they will be allowed to join the private league.

5.4.5 Portfolio Overview

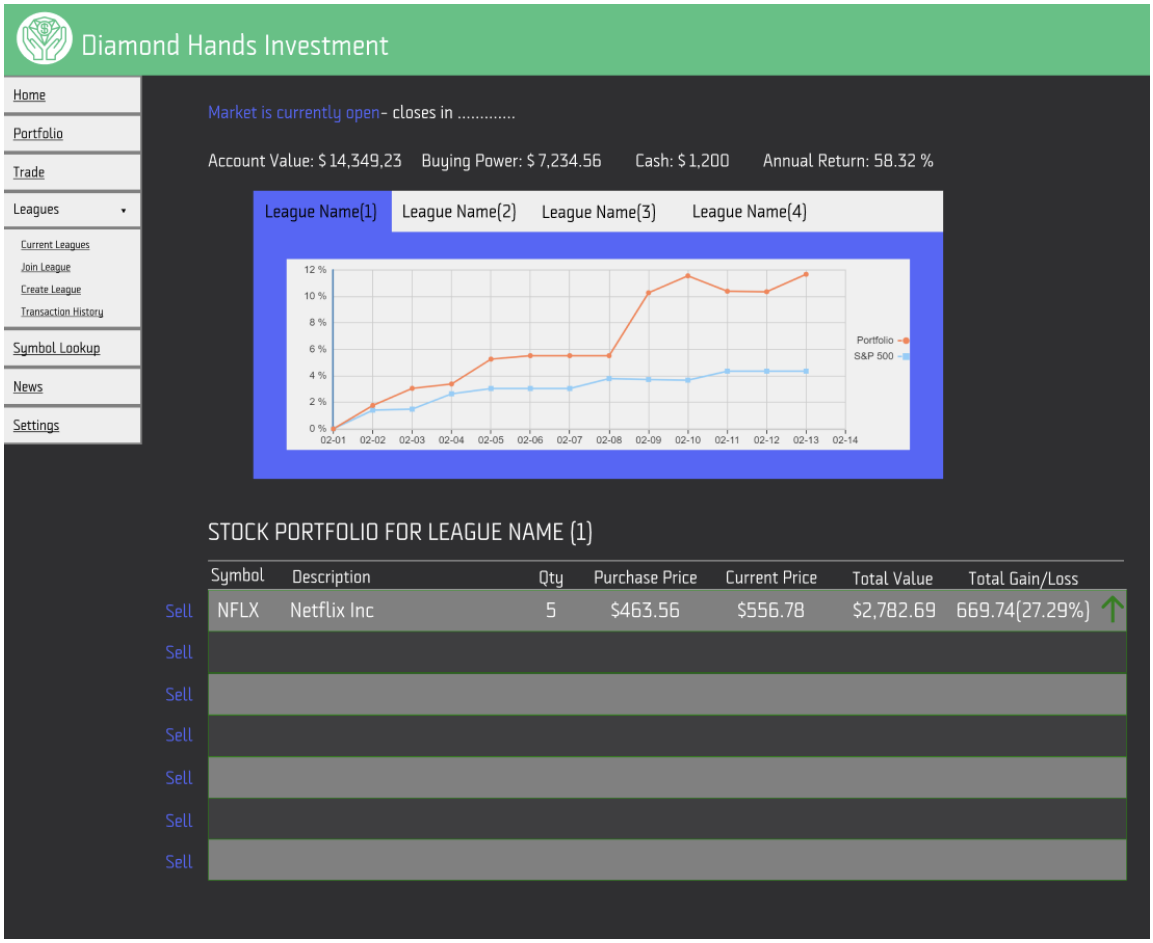

Figure 5.5. Portfolio Page

This page allows the user to view their progress. They will be able to see their portfolio value in that specific league, as well as their daily percent returns. Their portfolio value history will also be displayed in the form of a time series line graph. The user's current ranking will also be shown as well. If the badge is selected, the user will be directed to the leaderboard.

5.4.6 Order Placement

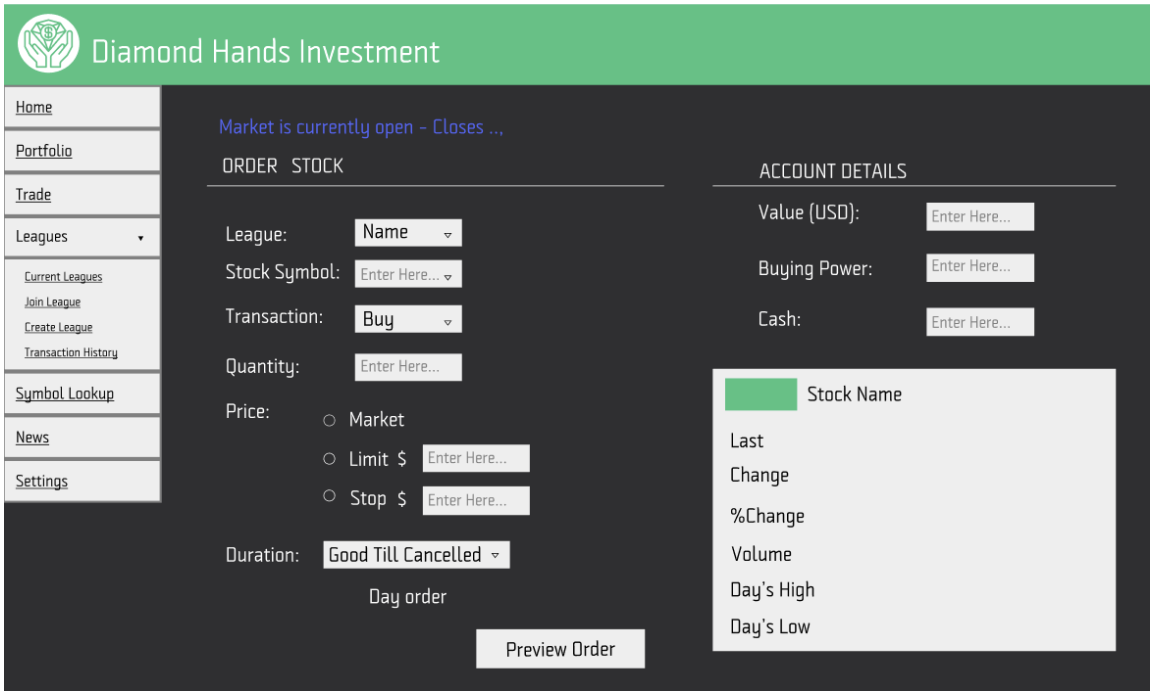

Figure 5.6. Trade Page

On this page, the user can place a trade. They can decide whether they want to buy or sell, the type of order and the number of shares they want to trade. Once the submit order button is clicked, the order is executed and a confirmation message will appear.

5.4.7 Individual Stock

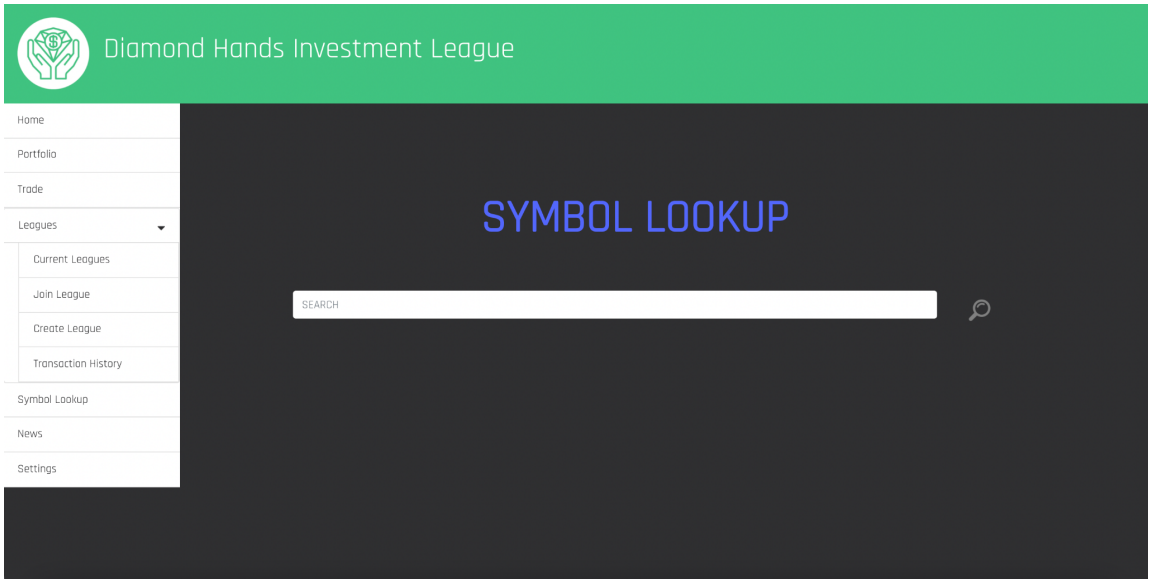

Figure 5.7(a) Symbol lookup page

On this page, the user will be able to search up information about different equities to help inform their decisions on what trades to make. The user will search up a name and a dropdown will appear with different equities that match the words that the user is typing in. After clicking the search button, the user will be taken to a new page with all the corresponding information about the equity.

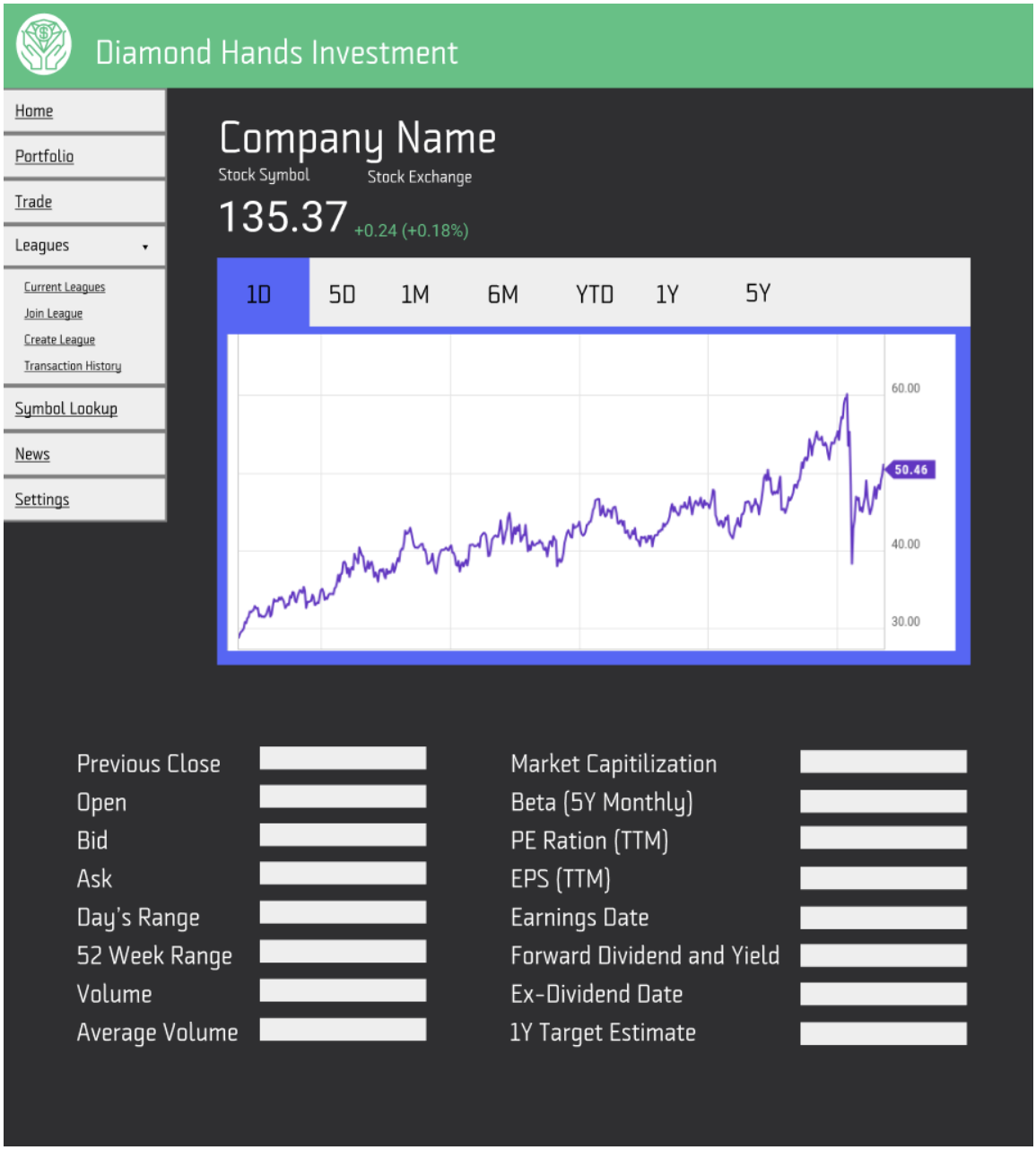

Figure 5.7.(b) Individual Stock Page

A user can look up symbols to find relevant stock information on the page shown in Figure 5.7(a). After searching a symbol, the user will be redirected to the individual stock page, shown in Figure 5.7(b). Here, the user will be able to see the progress of the stock. They can see how the stock is performing in the market. A graph will display the inclines and declines of all stocks, which are kept up to date with real world time. Users have the option to select the trade button to place an order on that stock, which will reroute them to the order placement page, or search up a different stock, which will reroute them to the

individual stock page. After viewing information about the stock, the user will be able to carefully make a decision and decide if they want to trade the stock or not.

5.4.8 Read News

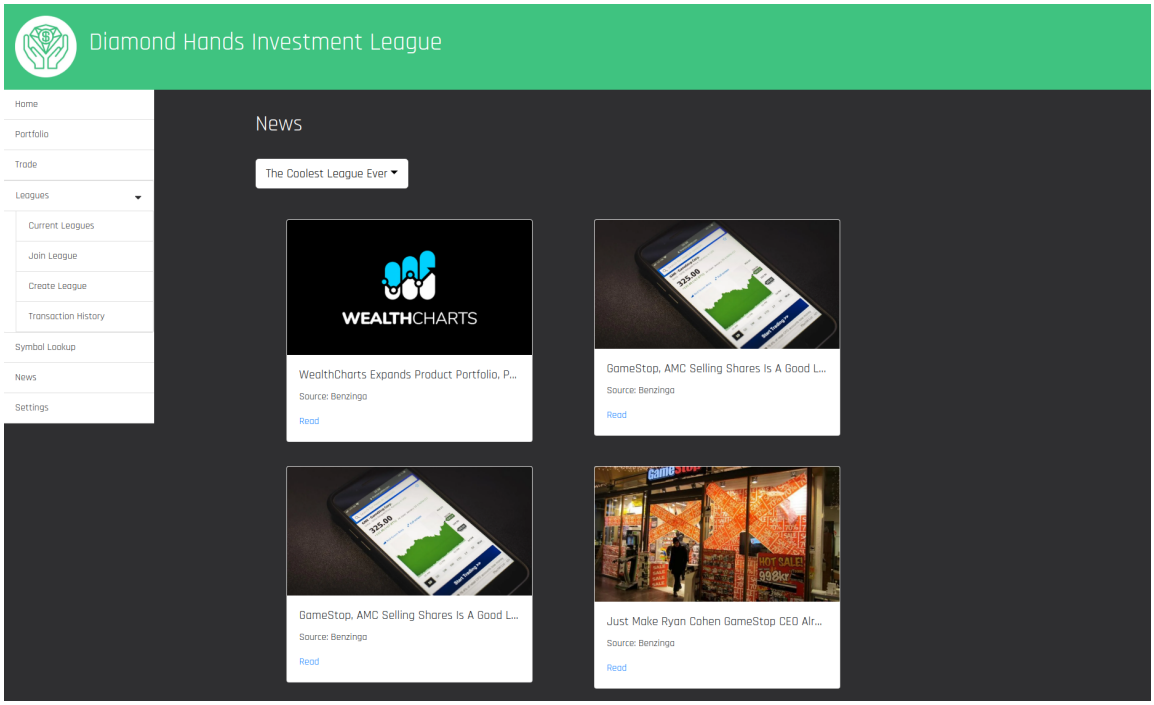

Figure 5.8. News Page

The general news page will allow users to keep up with the latest information related to stocks based on their account holdings. Once the article name is pressed, the article will be opened in another tab. This will allow the user to have a library of articles where they can obtain specific information about how certain stocks are performing. This will assist the user in making decisions concerning their stocks and give them information when is the best time to trade.

5.4.9 Leaderboard

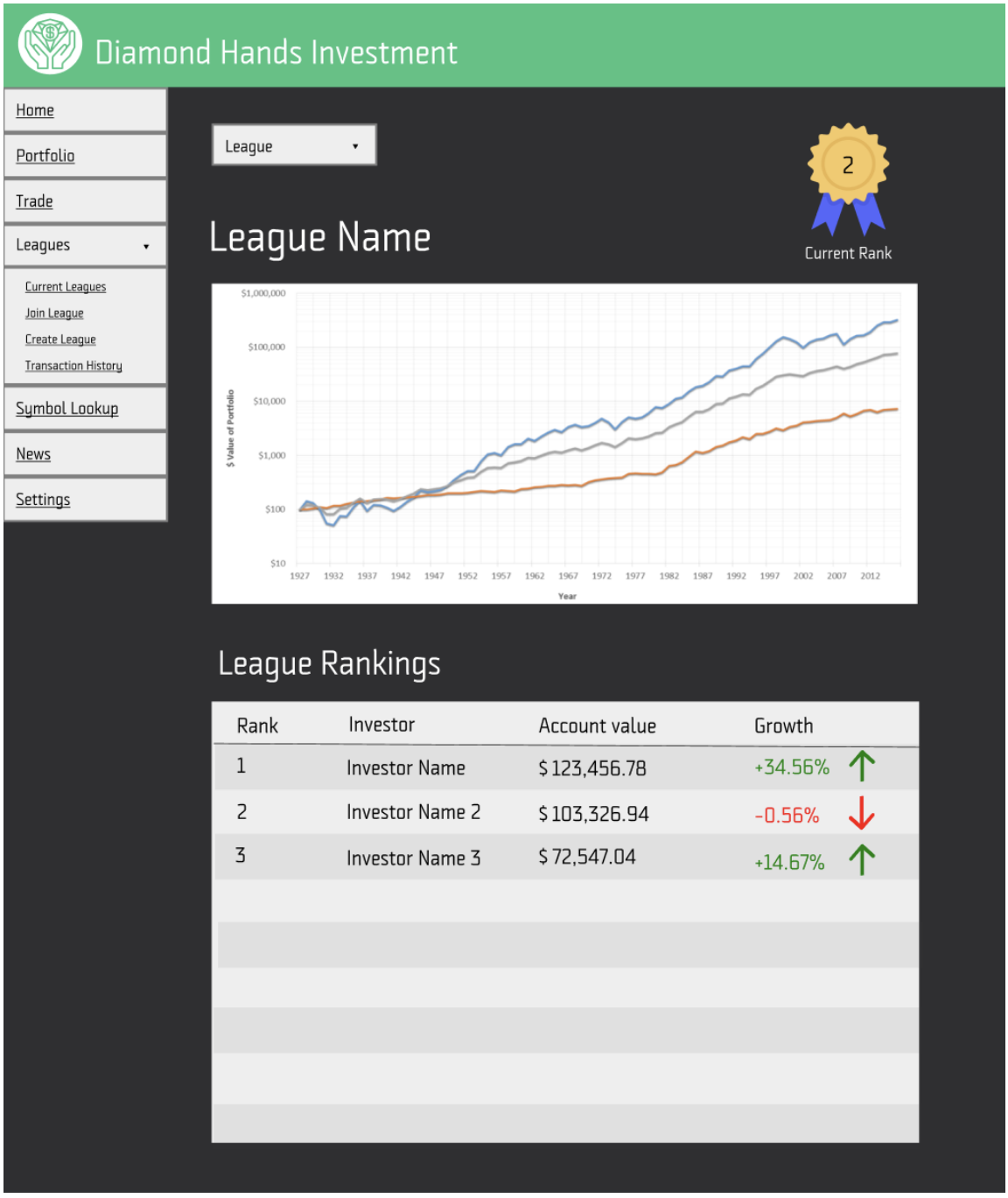

Figure 5.9. Leaderboard

The leaderboard will show the ranking of the user in their league with the most profitable holdings. This ranking is based on how they are performing in the stock market compared to the other members of the league. As users join multiple leagues at once, they will be able to see their ranking in each of the leagues. The leader board will include information

of the user's current rank, their account value, and the overall percentage, which shows whether the player lost money or gained money in that given day.

6 Functional Requirements Specification

6.1 Stakeholders

6.1.1 Inexperienced Investors

One of our primary target demographics are inexperienced investors. Inexperienced investors will be able to use our platform to learn about trading and get hands-on experience in a risk-free environment.

6.1.2 Experienced Investors

Experienced investors will also have a keen interest in the performance of our service. Diamond Hands Investment Fantasy League can serve as a way for experienced investors to experiment with new trading strategies. Additionally, the competitive social aspect of the fantasy league appeals to friend groups and coworkers who are interested in putting their investment skills to the test.

6.1.3 Educational Facilities

Educational facilities will want to utilize our platform as a tool to teach new students about investing. With more schools instituting financial literacy requirements, the demand for fun and educational investment tools is on the rise. Our fantasy league promotes engagement in the market, allowing students to actively exercise skills they have learned with no risk.

6.1.4 Sponsors

Our platform provides an excellent space for sponsors, such as popular online brokers like Robinhood or Fidelity, and advertisers to promote their products. With educational facilities and inexperienced traders in mind, our service allows sponsors to directly interact with a population of users who are just getting into the stock market and are interested in checking out new tools and products.

6.2 Actors & Goals

6.2.1 Initiating Actors:

Table 6.1. Initiating Actor Goals		
Actor	**Role**	**Goal**
Investor	The user is the one that is able to use the platform to trade equities, view their portfolio, and interact with other users in any league they create or join.	The goal of the user is to use the platform to improve their financial literacy and understanding of the market. There will also be the added bonus of healthy competition, which would entice any competitive spirits.
League Manager	The league manager is a user with special privileges, which allows them to dictate the rules and settings in a league that they create. A user can become a league manager by creating a league and inviting other players to it.	The goal of the league manager extends the goal of the user; the league manager is responsible for making sure that competition amongst users in their league stays fair throughout the active duration of the league.

6.2.2 Participating Actors:

Table 6.2. Participating actor goals	
Actor	**Role**
Stock Service	The stock service is a system that is responsible for providing stock data, including prices and news, to users.
Database	The database maintains information about user accounts, leagues, and stock data.
AI Service	The AI service is a system that is responsible for utilizing equity data and implementing algorithms in order to decide which equity to trade and when.

6.3 Use Cases

6.3.1 Use Case Diagram

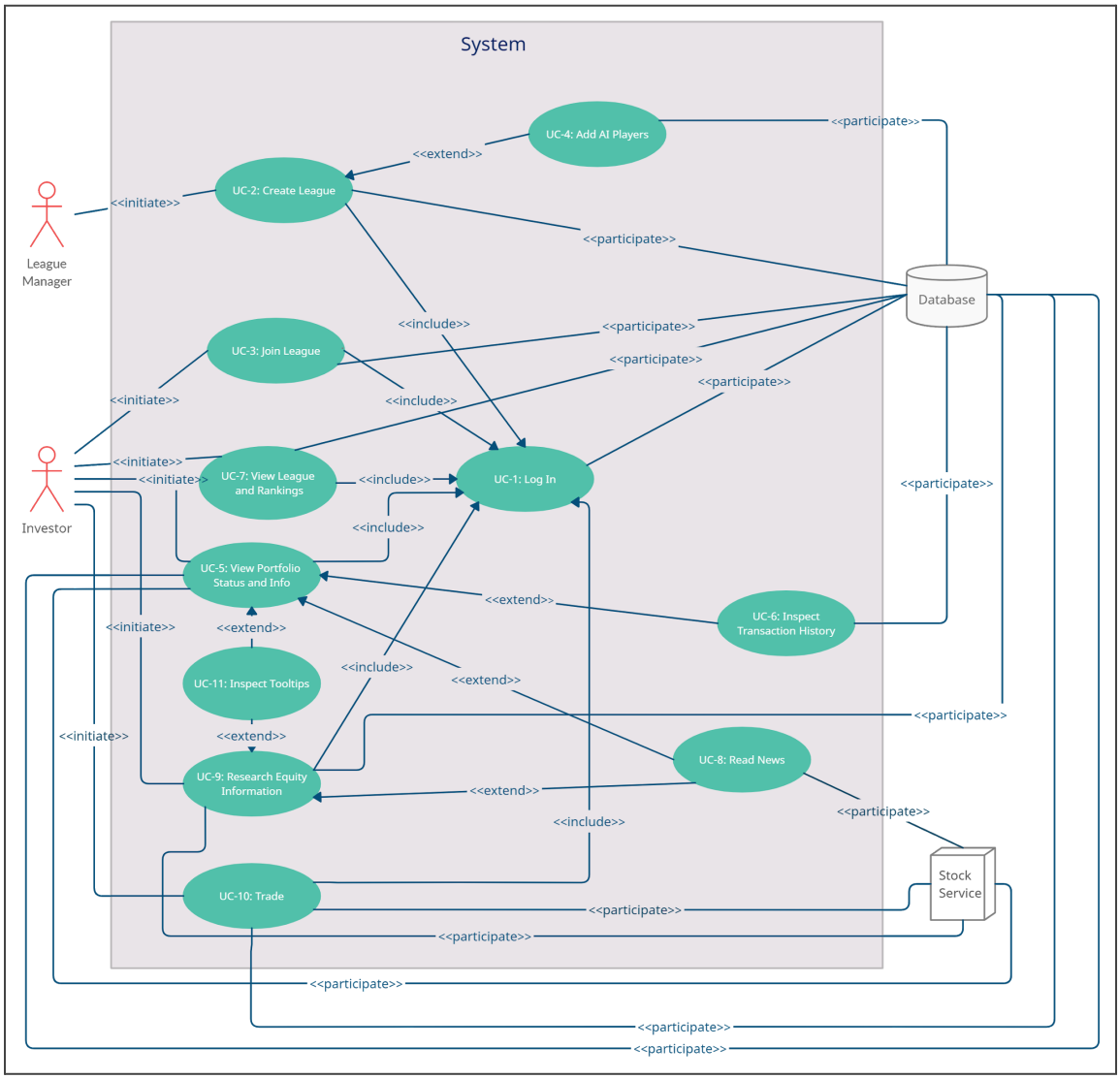

Figure 6.1. Use case diagram

6.3.2 Traceability Matrix

Table 6.3. Traceability matrix

Priority		UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11
REQ-1	2	X										

REQ-2	5					X					X	
REQ-3	5					X	X					
REQ-4	5								X	X		X
REQ-5	2					X			X			
REQ-6	1											X
REQ-7	4		X	X								
REQ-8	2		X	X								
REQ-9	1							X				
REQ-10	3					X		X				
REQ-11	2							X				
REQ-12	3		X		X							
REQ-13	1				X							
REQ-14	1				X							
REQ-15	5				X							
REQ-16	3				X							
REQ-17	4											
REQ-18	2											
REQ-19	2	X										
REQ-20	3											
REQ-21	4											
REQ-22	3					X		X		X	X	
REQ-23	3										X	
REQ-24	1	X										
REQ-25	3							X				
REQ-26	2		X		X							
REQ-27	2			X								

REQ-28	5					X						
REQ-29	4										X	
REQ-30	4									X		X
REQ-31	2					X			X			
REQ-32	3							X				
Sum:		5	11	8	15	25	5	15	9	12	15	10

6.3.3 Fully-Dressed Description

6.3.3.1 UC-1: Login

Related Requirements:	REQ-1, REQ-19, REQ-24
Initiating Actor:	Investor
Actor's Goal:	To log into account
Participating Actors:	Database
Preconditions:	The user has registered and created an account.
Postconditions:	The user is now viewing the home page and is able to access all functionality of the app such as looking up equity info, joining leagues, viewing financial news, etc.

Flow of Events for Main Success Scenario:

1) **User** enters credentials to access their account.
2) **System** verifies with the **database** by checking if entered credentials are correct.
3) **System** displays home page.

Flow of Events for Extensions (Alternate Scenarios or Contingencies):

1) **User** enters invalid credentials.
    a) **User** attempts to log in but enters incorrect username and/or password.
    b) **System** verifies user credentials with **Database.**
    c) **System** displays error message and prompts **User** to try again.

6.3.3.2 UC-2: Create League

Related Requirements:	REQ-7, REQ-8, REQ-12, REQ-26
Initiating Actor:	League Manager
Actor's Goal:	To create a league
Participating Actors:	Database
Preconditions:	The user is logged in and authenticated.
Postconditions:	The league has been created with desired settings and the league manager has been added to the league.

Flow of Events for Main Success Scenario:

1) **League Manager** enters desired league settings for the league they want to create.
2) **System** creates the league and saves it to the **database.**
3) **System** adds the **league manager** to the league and updates the **database.**
4) **System** displays a success message.

Flow of Events for Extensions (Alternate Scenarios or Contingencies):

1) **System** is unable to communicate with the **database.**
    a) **System** attempts to communicate with the **database.**
    b) **System** displays error message to **User.**

6.3.3.3 UC-3: Join League

Related Requirements:	REQ-7, REQ-8, REQ-27
Initiating Actor:	Investor
Actor's Goal:	To join a league and participate in the game
Participating Actors:	Database
Preconditions:	The user is logged in and authenticated.

Postconditions:	The user is now in the league and can compete with other league members and place trades.

Flow of Events for Main Success Scenario:

1) **User** selects the league that they want to join and enters the league key if the league is private.
2) **System** checks with the **database** to see if the user is already in that league.
3) **System** checks with the **database** to see if the league key is correct, if it is a private league.
4) **System** adds the user to the league and updates the league database.
5) **System** sends a success message.

Flow of Events for Extensions (Alternate Scenarios or Contingencies):

1) **System** is unable to communicate with the **database.**
    c) **System** attempts to communicate with the **database.**
    d) **System** displays error message to **User.**

6.3.3.4 UC-4: Add AI Players

Related Requirements:	REQ-12, REQ-13, REQ-14, REQ-15, REQ-16, REQ-26
Initiating Actor:	League Manager, AI Service
Actor's Goal:	To include AI Players in a league they create
Participating Actors:	Database
Preconditions:	The user is the league manager of the league in which they are adding AI players.
Postconditions:	One or more AI players are added to the league, each with restrictions on gameplay.

Flow of Events for Main Success Scenario:

1) **League Manager** chooses to add AI players to a league that they are managing.
2) **System** creates AI players and corresponding portfolios, and adds them to the **database** as users.
3) **System** updates league information in the **database**.
4) **System** communicates with the **AI service** to continuously monitor and update this AI player.
5) **System** sends a confirmation message to the **league manager**.

Flow of Events for Extensions (Alternate Scenarios or Contingencies):

1) **System** is unable to communicate with the **database.**
    a) **System** attempts to communicate with the **database.**
    b) **System** displays an error message to the **user.**

6.3.3.5 UC-5: View Portfolio Status & Information

Related Requirements:	REQ-2, REQ-3, REQ-5, REQ-10, REQ-28, REQ-31
Initiating Actor:	Investor
Actor's Goal:	To view one's current positions in an active league and keep track of their total return
Participating Actors:	Database, Stock Service
Preconditions:	The user is logged in, authenticated and a member of an active league.
Postconditions:	The user is now viewing a page that shows their total return on all their holdings in that particular league. A graph of portfolio history is also displayed as well as ranking and recent league activity.

Flow of Events for Main Success Scenario:

1) **User** selects a league from the page listing all active leagues.
2) **System** requests the user's current positions and league information from

the **Database.**

3) **System** requests all information relevant to the user's portfolio from the **Stock Service** e.g. historical price data, curated news, etc.
4) **System** displays **User's** portfolio and ranking in the league selected

Flow of Events for Extensions (Alternate Scenarios or Contingencies):

1. **System** is unable to communicate with the **Stock Service**.
    a. **System** connection attempt to **Stock Service** times out.
    b. **System** alerts **User** of failure to access resources.
2. **System** is unable to communicate with the **Database**
    a. **System** connection attempt to **Database** times out.
    b. **System** alerts **User** of failure to access resources.

6.3.3.6 UC-6: Inspect Transaction History

Related Requirements:	REQ-3
Initiating Actor:	Investor
Actor's Goal:	View their order history in their active leagues as well as recently executed orders of fellow league members. They can also see transactions made by all users in their past, inactive leagues.
Participating Actors:	Database
Preconditions:	The user must be signed into their account and must have been a part of a league.
Postconditions:	The investor is now viewing a page that shows all of their transaction history organized by individual leagues. This also allows them to see the transaction history of other players.

Flow of Events for Main Success Scenario:

1) **User** clicks on the league tab that they are interested in viewing the transaction history of
2) **System** requests all the portfolio's that match the league id from the **Database**
3) **Database** returns a list of order information sorted by the time that each

order was placed
4) **System** displays the order history list

Flow of Events for Extensions (Alternate Scenarios or Contingencies):

1) **System** is unable to communicate with the **Database**
    a) **System** connection attempt to **Database** times out.
    b) **System** alerts **User** of failure to access resources.

6.3.3.7 UC-7: View League & Rankings

Related Requirements:	REQ-9, REQ-10, REQ-11, REQ-25, REQ-32
Initiating Actor:	Investor
Actor's Goal:	To view league settings and standing in league
Participating Actors:	Stock Service, Database
Preconditions:	The user is logged in, authenticated and a member of an active league.
Postconditions:	The user is now viewing a page where the leaderboard is displayed and the current settings of the league.

Flow of Events for Main Success Scenario:

1) **User** selects a league from their current leagues list.
2) **System** requests league information and user portfolios from the **Database**, which returns the information.
3) **System** requests updated stock prices from the **Stock Service** to calculate player portfolio values for current leagues.
4) **System** displays league information and rankings.

Flow of Events for Extensions (Alternate Scenarios or Contingencies):

1. **System** is unable to communicate with the **Stock Service**.
    a. **System** connection attempt to **Stock Service** times out.
    b. **System** alerts **User** of failure to access resources.
2. **System** is unable to communicate with the **Database**
    a. **System** connection attempt to **Database** times out.

b. **System** alerts **User** of failure to access resources.

6.3.3.8 UC-8: News

Related Requirements:	REQ-4, REQ-5, REQ-31
Initiating Actor:	Investor
Actor's Goal:	To view recent news relevant to portfolio
Participating Actors:	Database, Stock service.
Preconditions:	The user is logged in and authenticated. For portfolio news, the user must have some equity holding in their portfolio.
Postconditions:	The user is viewing the news page with a curated list of articles specific to their portfolio.

Flow of Events for Main Success Scenario:

1) **User** clicks on the news tab to view their portfolio specific news.
2) **System** requests news data based on the **User** specific portfolio from the **Stock Service**.
3) **Database** stores news data obtained from the **Stock service.**
4) **System** displays the news articles.

Flow of Events for Extensions (Alternate Scenarios or Contingencies):

1) **System** is unable to communicate with the **Stock Service**.
    a) **System** connection attempt to **Stock Service** times out.
    b) **System** alerts **User** of failure to access resources.
2) **System** is unable to communicate with the **Database**
    a) **System** connection attempt to **Database** times out.
    b) **System** alerts **User** of failure to access resources.

6.3.3.9 UC-9: View Equity Information

Related Requirements:	REQ-4, REQ-22, REQ-30
Initiating Actor:	Investor

Actor's Goal:	To view stock quotes, history, and other financial data on a desired company
Participating Actors:	Stock Service, Database
Preconditions:	The user is logged in, authenticated and our stock service is functioning properly.
Postconditions:	The user is now viewing a page with financial data on an equity of their choice.

Flow of Events for Main Success Scenario:

1) **User** enters ticker symbol of their desired equity.
2) **System** requests **Database** to retrieve all cache data on the selected equity if available.
3) **System** validates cache data and requests current equity data from the **Stock service,** which returns updated information.
4) **System** transfers all updated information to cache in the **Database.**
5) **User** can view updated equity information.

Flow of Events for Extensions (Alternate Scenarios or Contingencies):

1. **User** enters invalid ticker symbol.
    a. **System** receives invalid notice from **Stock Service**
    b. **System** alerts **User** of the invalid ticker.

6.3.3.10 UC-10: Make Trades

Related Requirements:	REQ-2, REQ-22, REQ-23, REQ-29
Initiating Actor:	Investor
Actor's Goal:	To place a market or limit order for a specific league's portfolio
Participating Actors:	Stock Service, Database
Preconditions:	The user is logged in, authenticated, and a member of an active league.
Postconditions:	The user's portfolio is updated reflecting the

	changes in shares of the traded position.

Flow of Events for Main Success Scenario:

1) **User** enters the ticker symbol for their desired equity.
2) **System** requests **Stock Service** to validate the ticker symbol query, as well as display relevant information regarding the desired equity.
3) **User** selects whether they would like to buy or sell, their desired type of transaction, and the duration they would like the order to stay in effect.
4) **User** places their order.
5) **System** fulfills the order based on the most recent quote available from the stock service at any given time while the order is active.
6) **System** displays confirmation message to **User.**

Flow of Events for extensions (Alternate Scenarios or Contingencies):

1. **User** does not have enough capital to purchase shares of desired equity or sufficient shares to sell shares of desired equity.
    a. **System** validates request against **Database** data and determines the request is invalid.
    b. **System** alerts **User** of trade request rejection.
2. **User** enters an invalid ticker symbol.
    a. **System** receives invalid notice from **Stock Service**
    b. **System** alerts **User** of the invalid ticker.

6.3.3.11 UC-11: Tooltips and Summary Page

Related Requirements:	REQ-4, REQ-6, REQ-30
Initiating Actor:	Investor
Actor's Goal:	To learn about technical terms and review their recent performance results
Participating Actors:	Database, Stock Service, AI Service
Preconditions:	The user is logged in, authenticated, and a member of an active league.
Postconditions:	The user will be able to view the educational resource they requested (tooltip or summary)

Flow of Events for Main Success Scenario:

1) **User** clicks on tooltip indicator or summary page
2) **System** requests **Database** for tooltip or league information
    a) **System** calculates summary information using league information
3) **System** displays requested information to the **User**

Flow of Events for Extensions (Alternate Scenarios or Contingencies):

1. **System** is unable to connect to **Database**
    a. **System** receives connection error information
    b. **System** displays to **User** that a connection error occurred

6.4 System Sequence Diagrams

6.4.1 Use Case UC - 1: Login

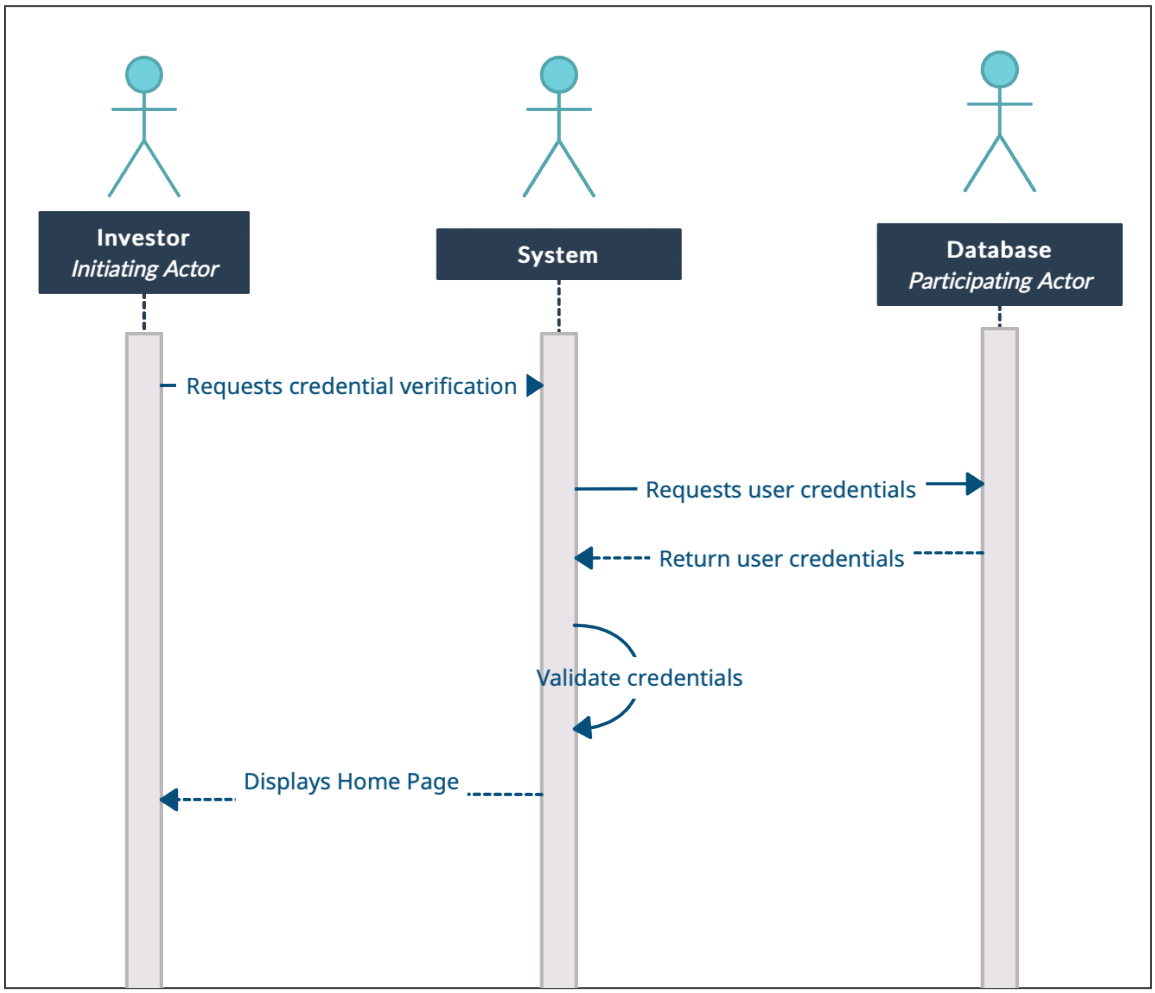

Figure 6.1.1. System Sequence Diagram: Login

6.4.2 Use Case UC - 2: Create League

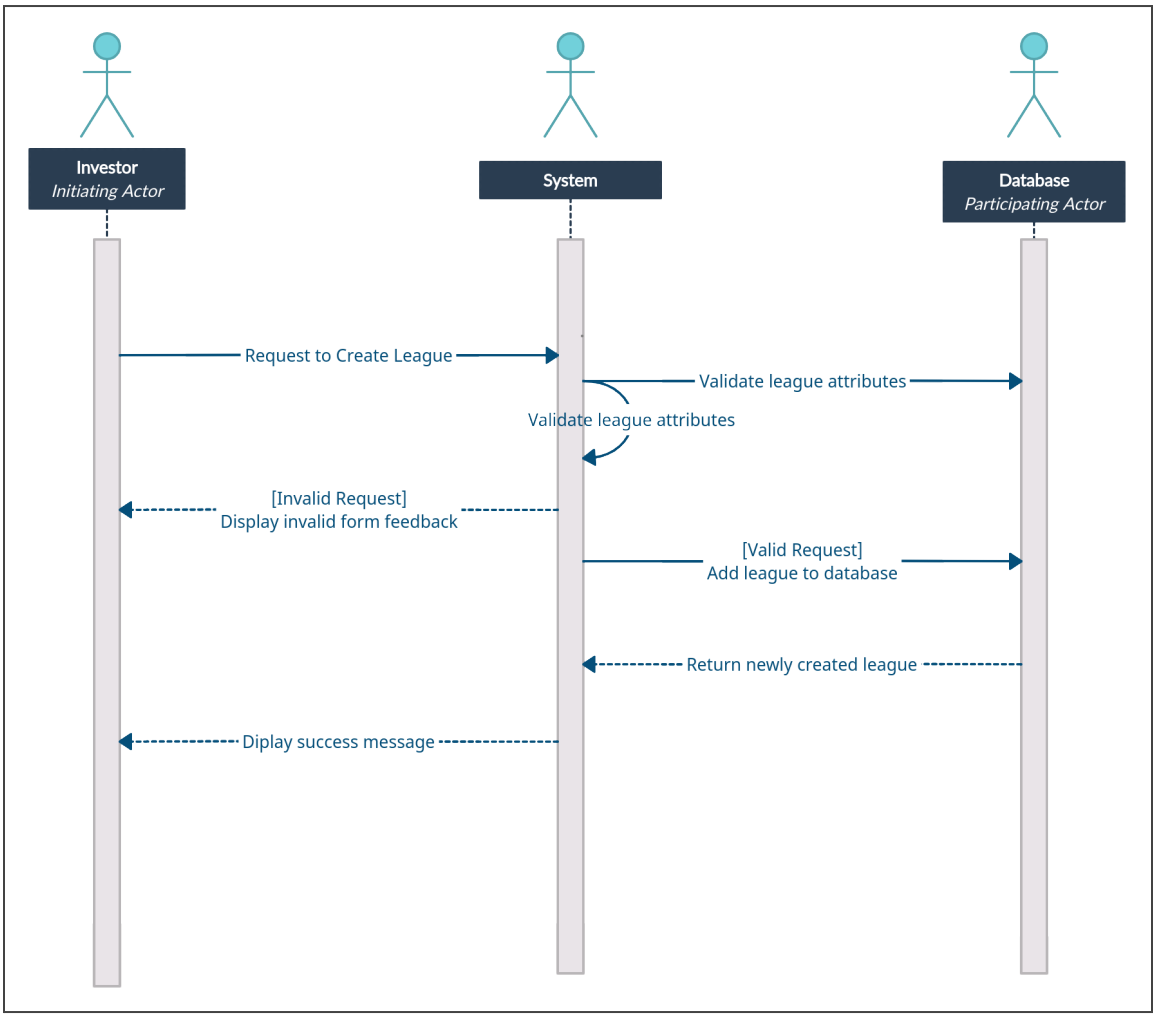

Figure 6.1.2. System Sequence Diagram: Create League

6.4.3 Use Case UC - 3: Join League

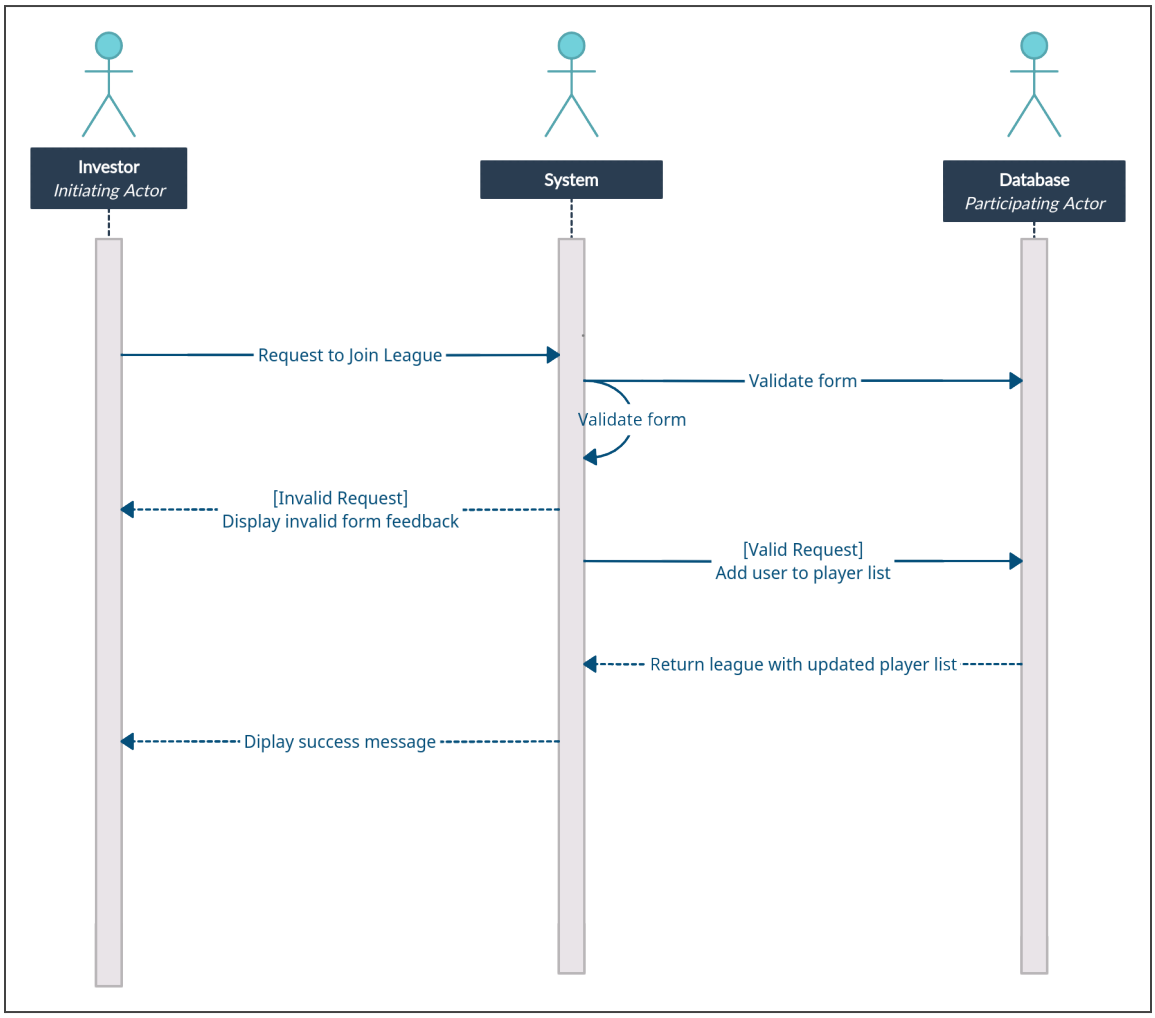

Figure 6.1.3. System Sequence Diagram: Join League

6.4.4 Use Case UC - 4: Add AI Players

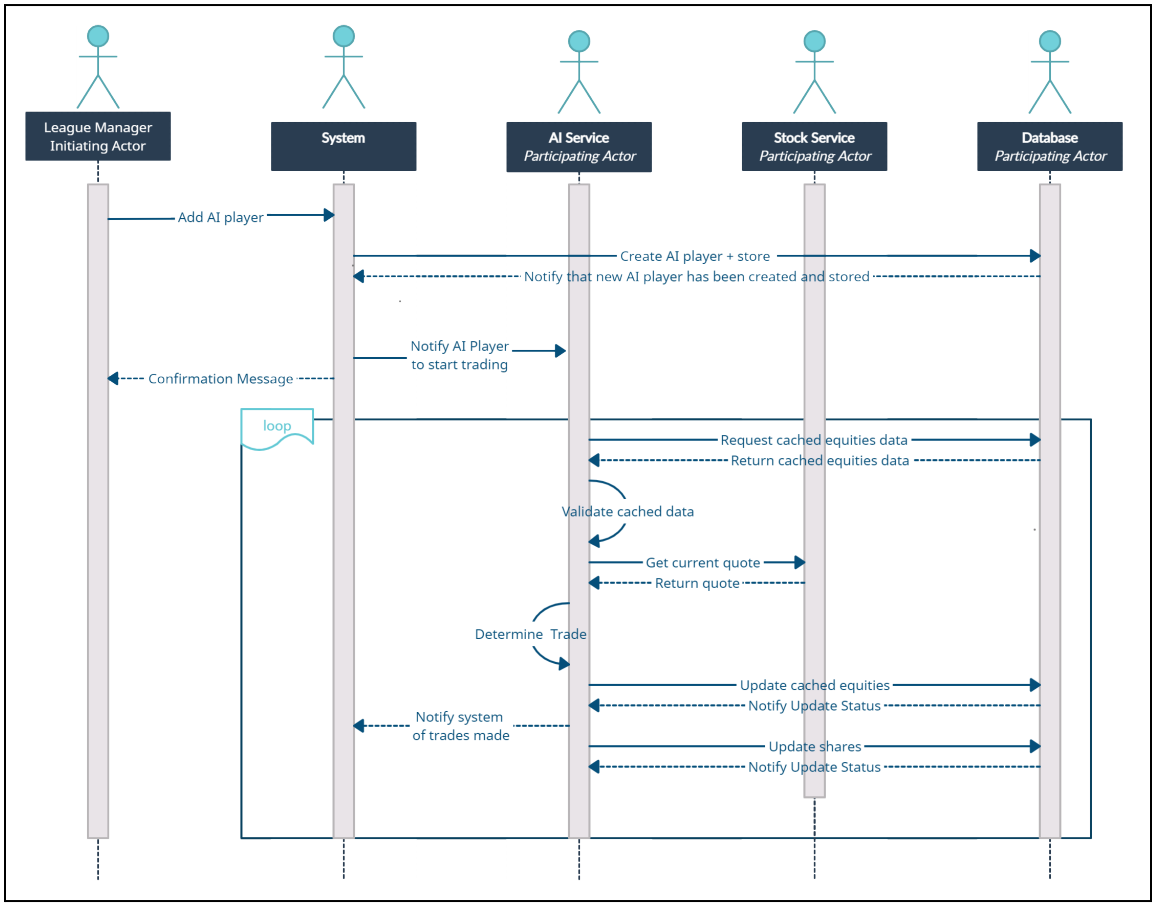

Figure 6.1.4. System Sequence Diagram: Add AI players

6.4.5 Use Case UC - 5: View Portfolio Status & Information

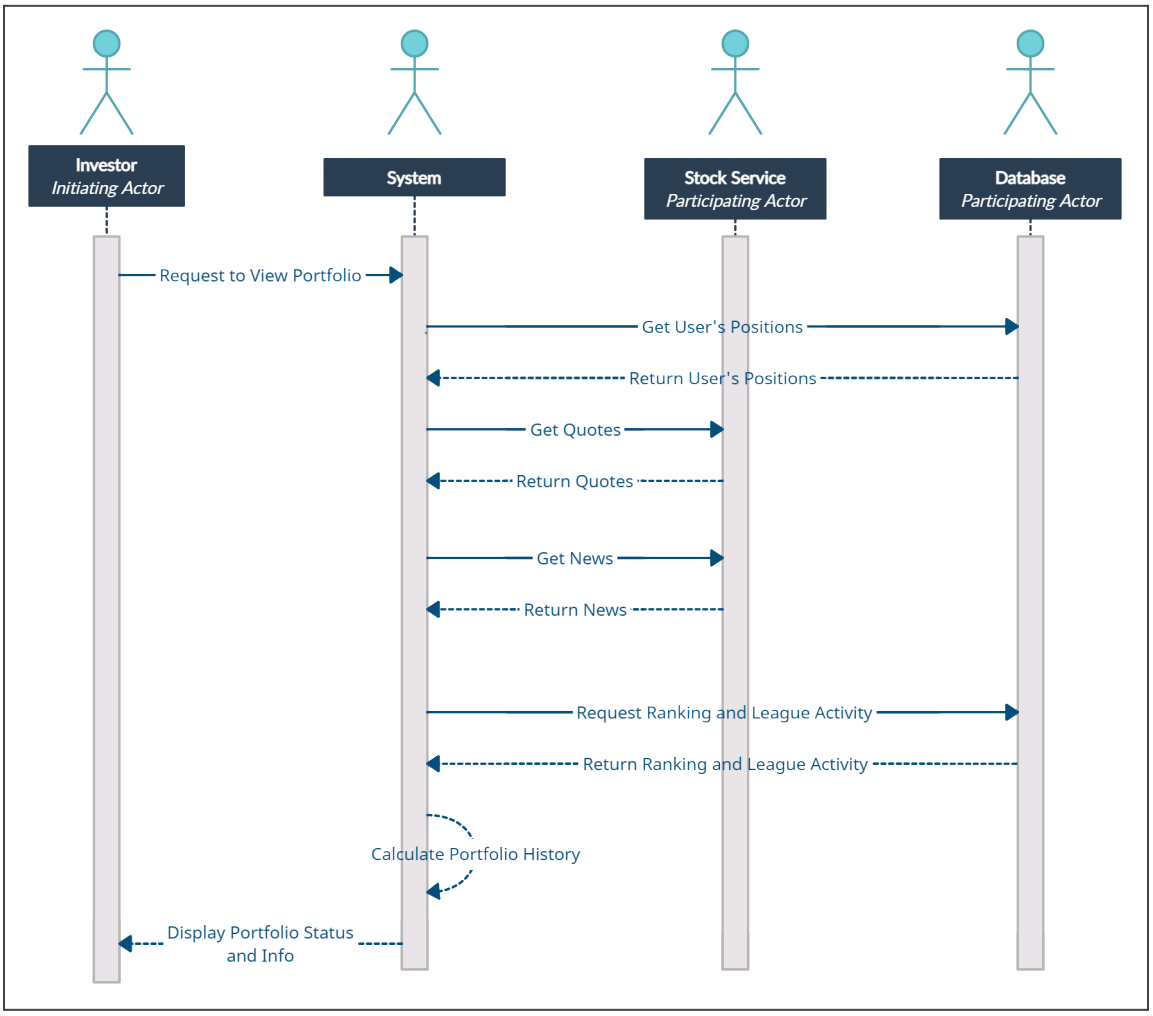

Figure 6.1.5. System sequence diagram: View Portfolio

6.4.6 Use Case UC - 6: Inspect Transaction History

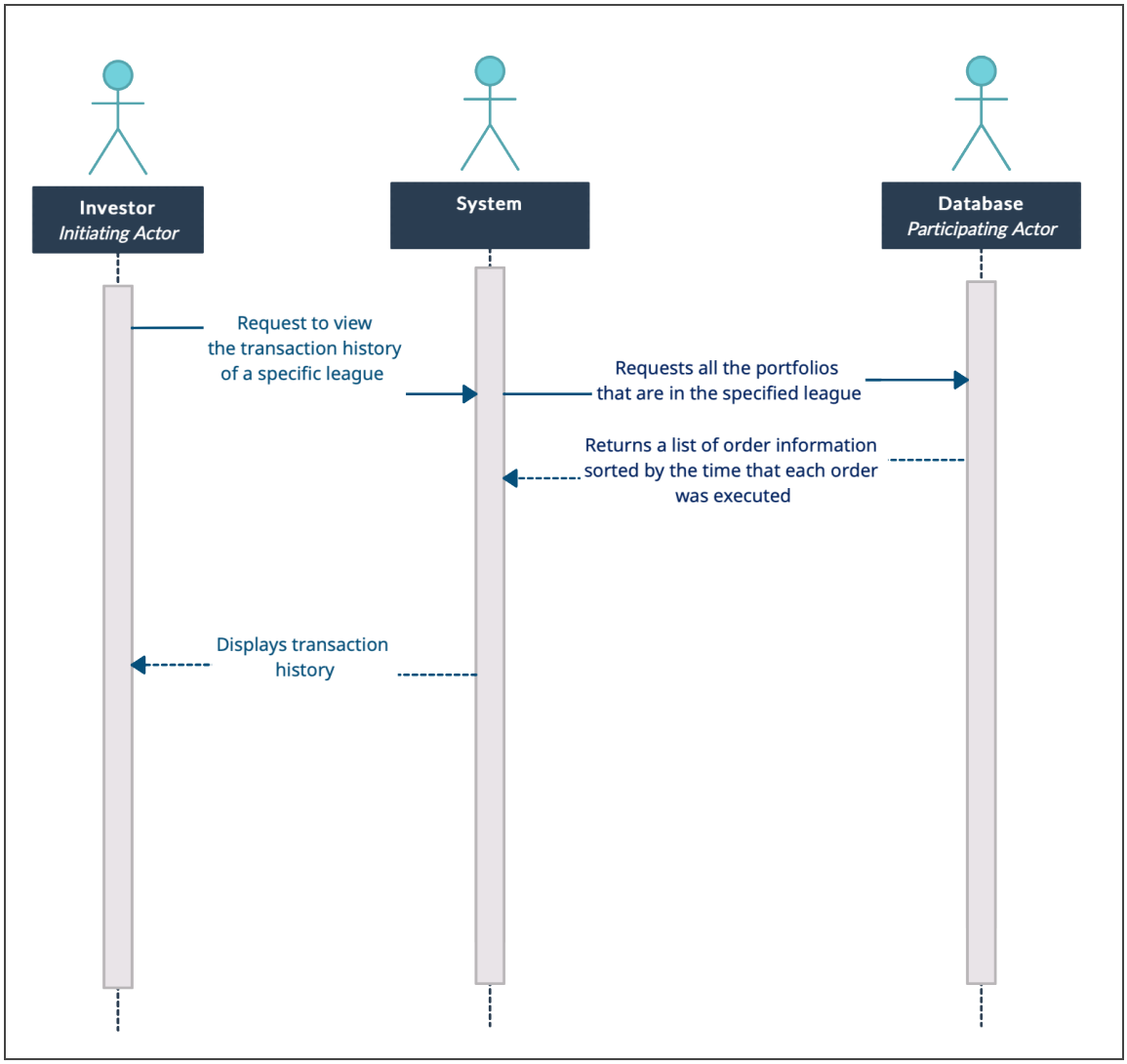

Figure 6.1.6. System sequence diagram: View League and Rankings

6.4.7 Use Case UC - 7: View League & Rankings

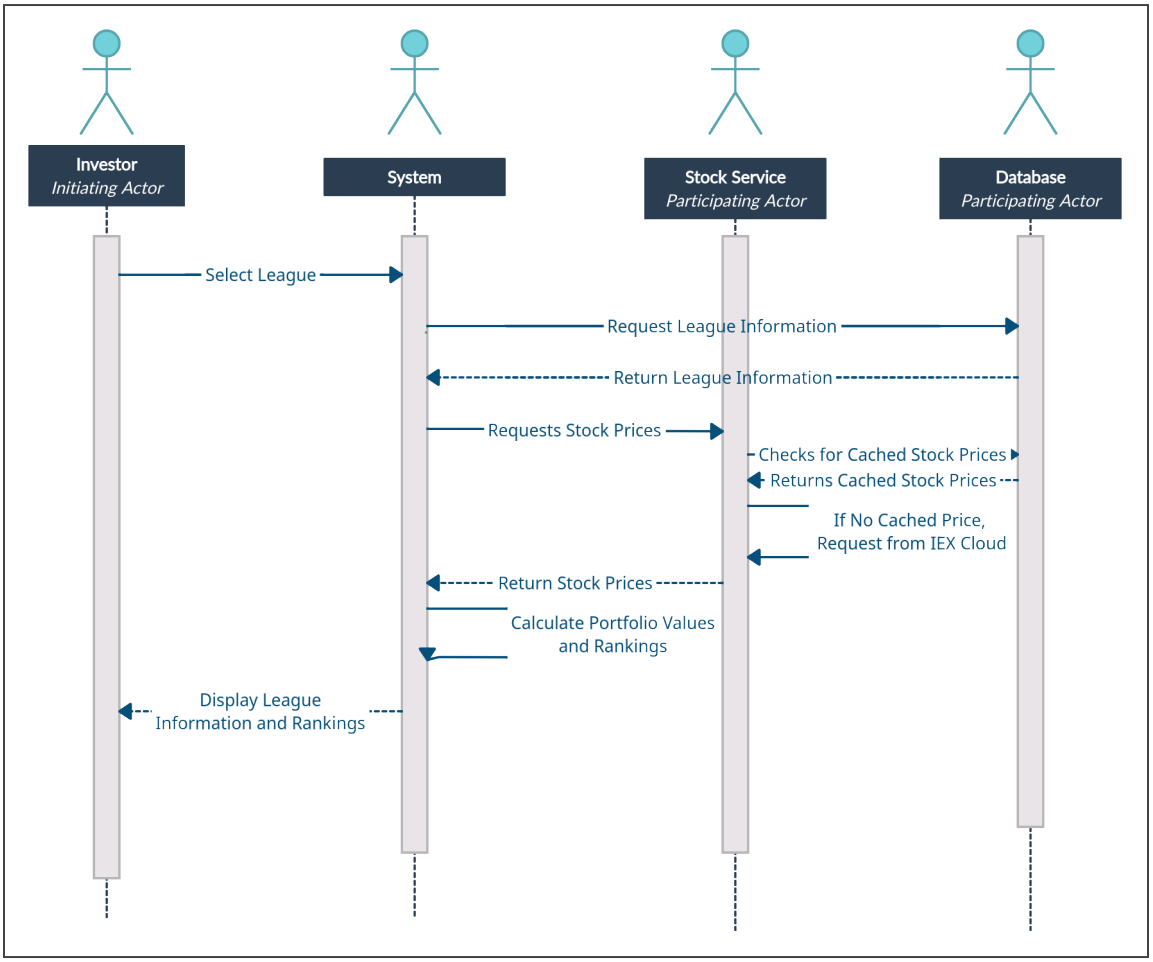

Figure 6.1.7. System sequence diagram: View League and Rankings

6.4.8 Use Case UC - 8: News

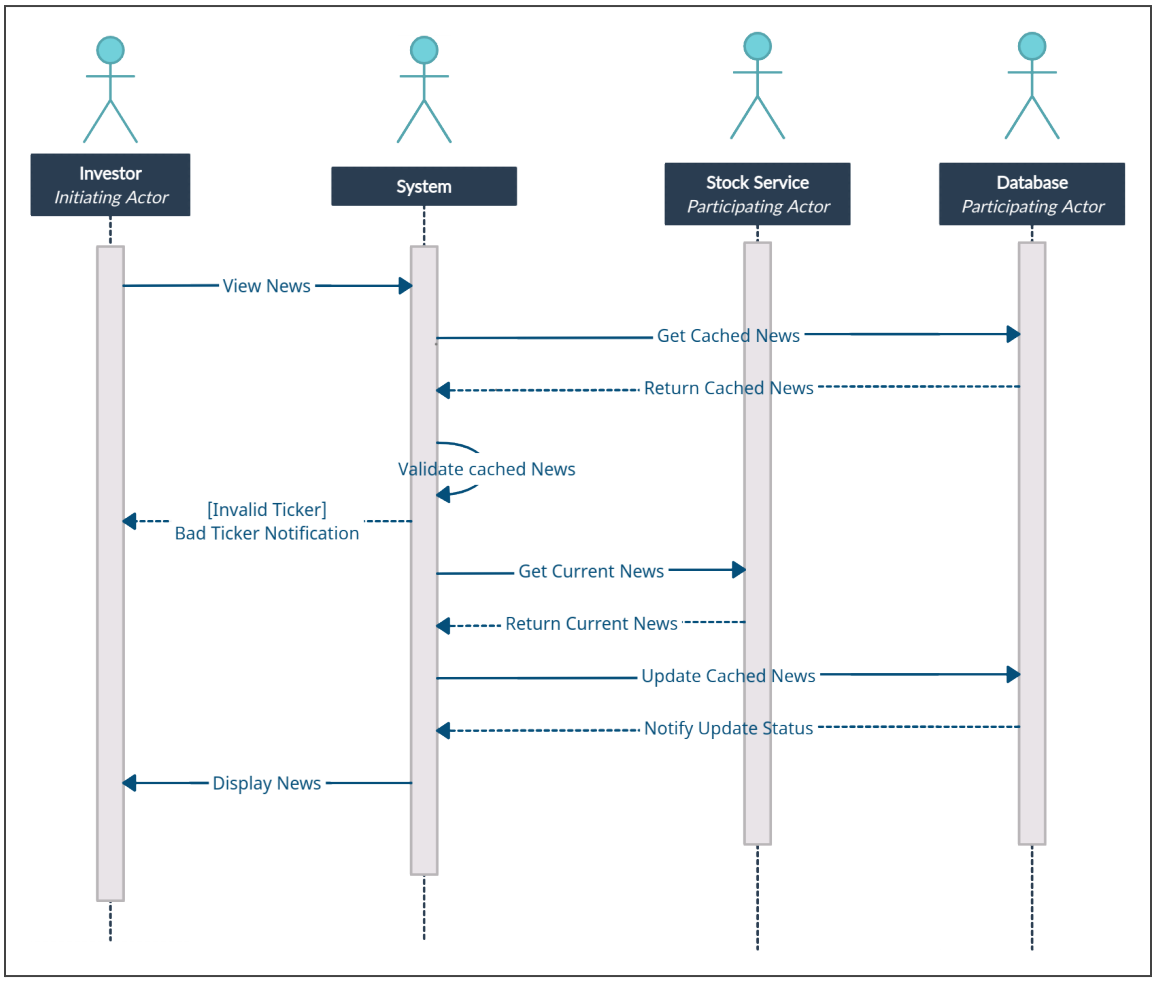

Figure 6.1.8. System Sequence Diagram: News

6.4.9 Use Case UC - 9: View Equity Information

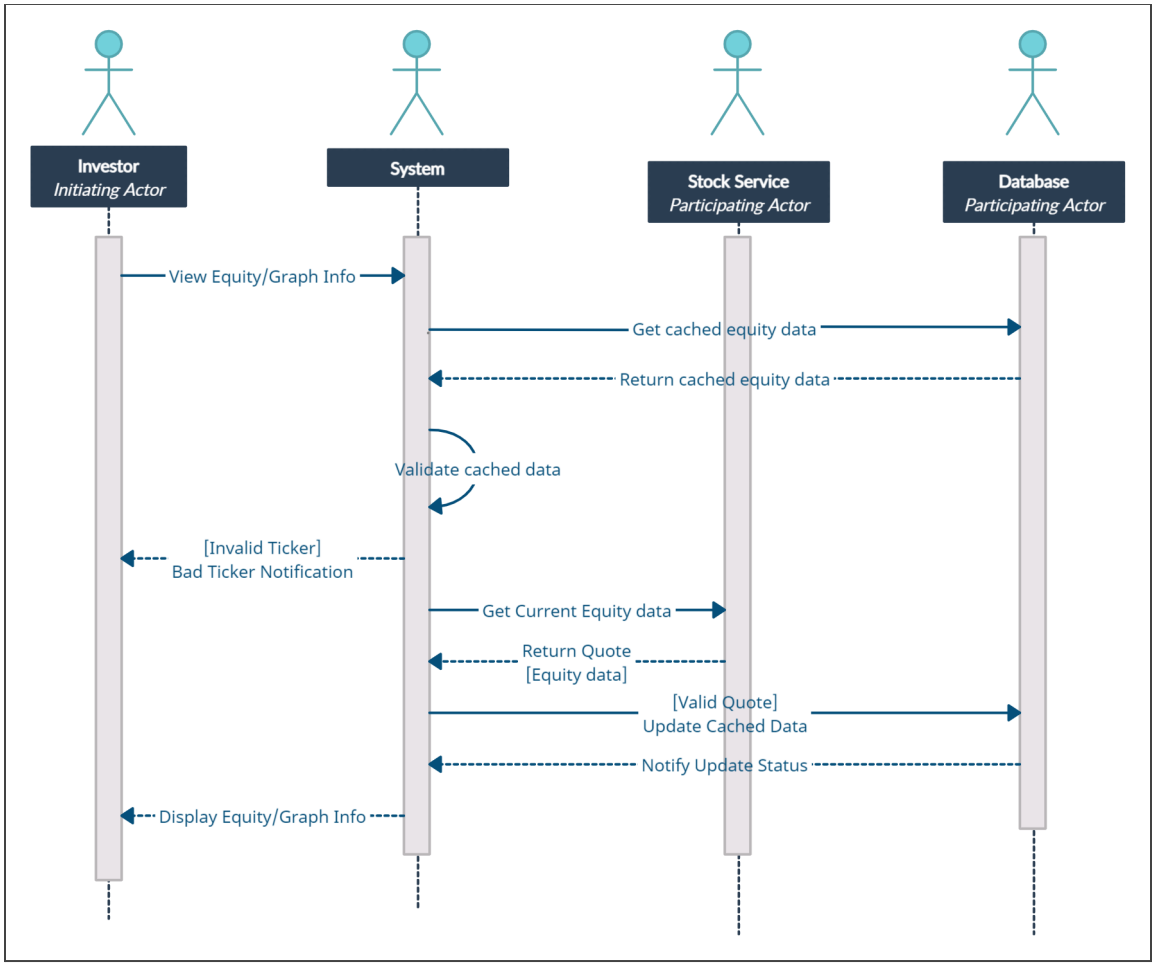

Figure 6.1.9. System sequence diagram: View Equity Information

6.4.10 Use Case UC - 10: Make Trades

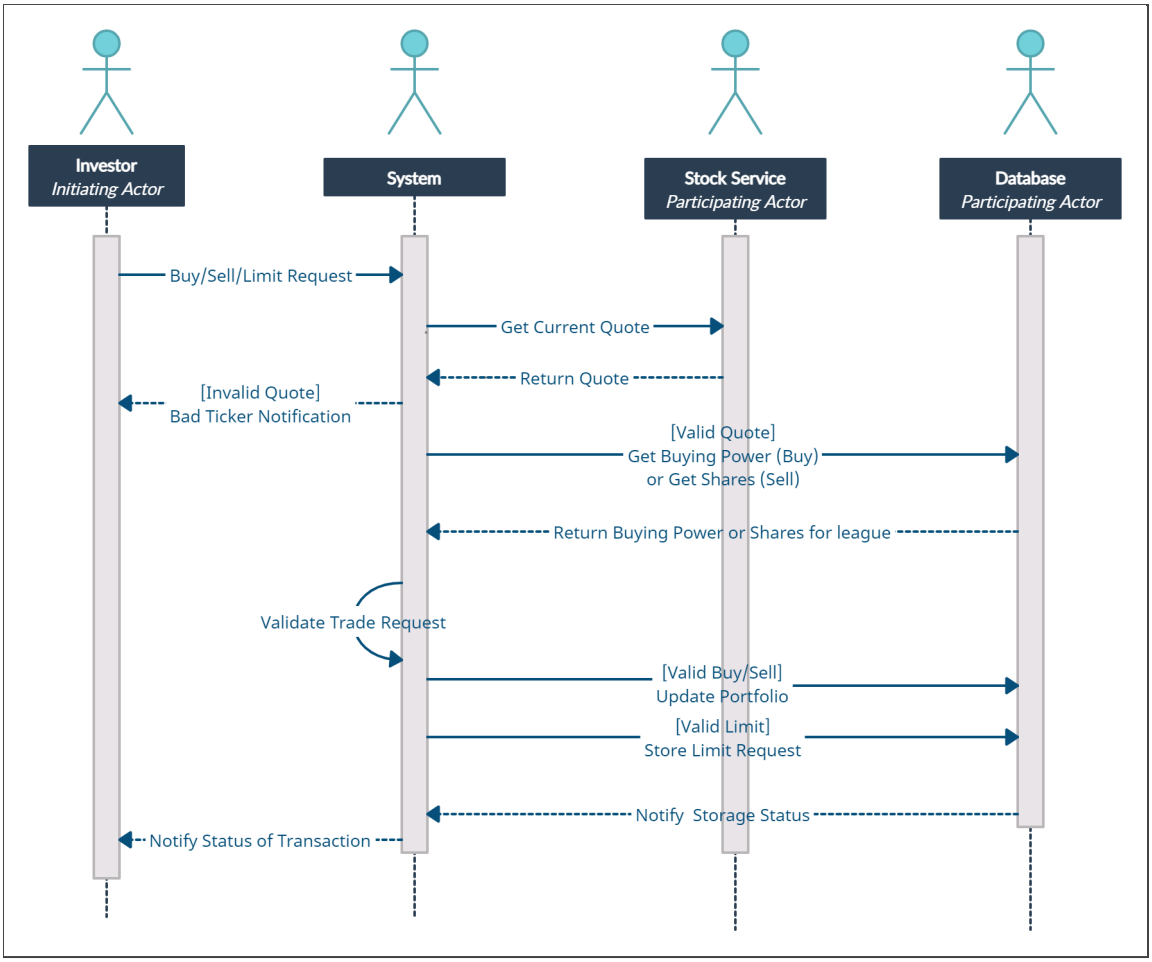

Figure 6.1.10. System sequence diagram: Trade

6.4.11 Use Case UC - 11: Tooltips and Summary Page

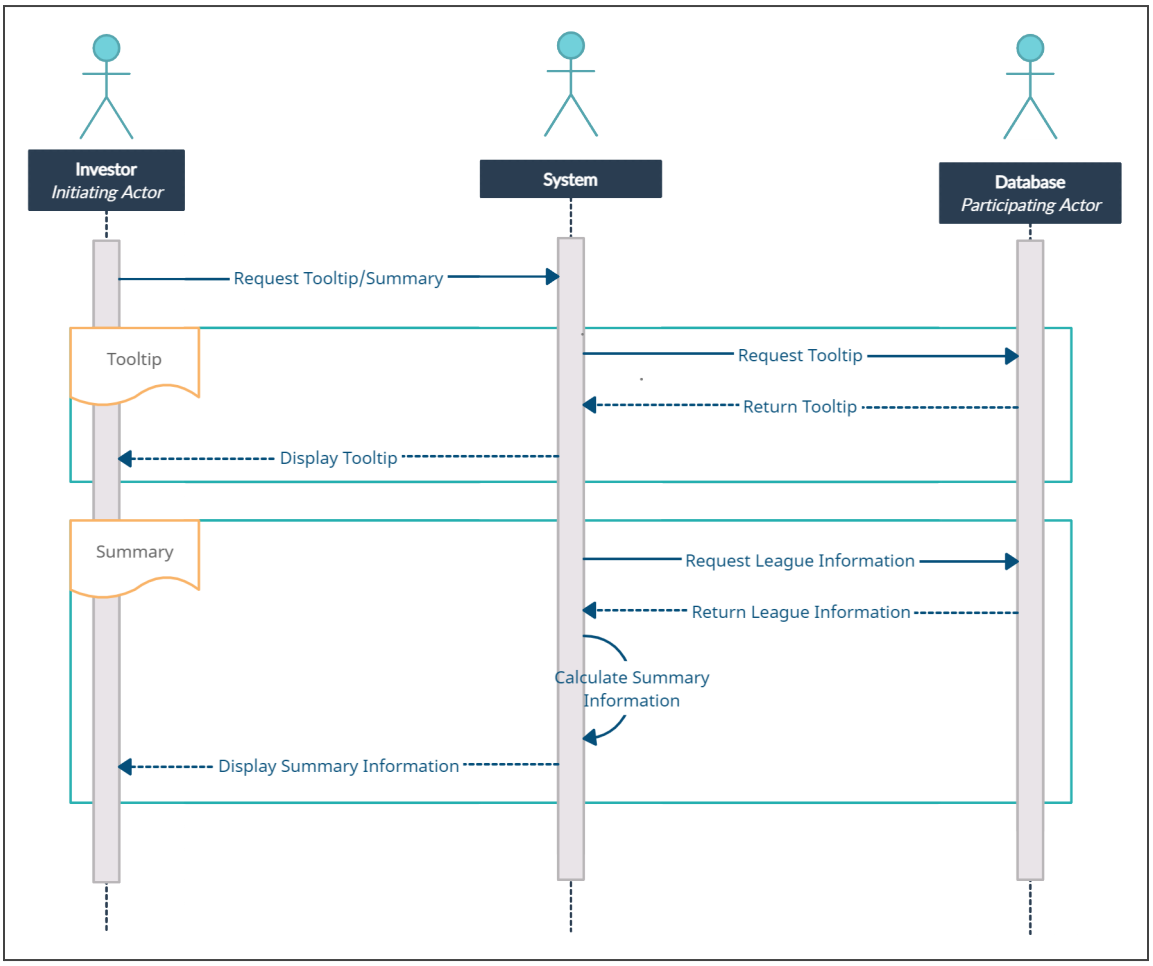

Figure 6.1.11. System Sequence Diagram: Tooltips and Summary

7 Effort Estimation Using Use Case Points

7.1 Unadjusted Use Case Points

Table 7.1. UUCP Values broken down by use cases

UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11
5	11	8	15	25	5	15	9	12	15	10

UUCP = 130

7.2 Technical Complexity Factor

Table 7.2. Technical complexity factor		
Technical Factor	**Description**	**Weight**
T1	Reusable code or design	1
T2	Easy to change	1
T3	Multiple services	2
T4	Concurrent use	1

TCF = C1 + C2 * Technical Factor Total → TCF = 0.55 * 0.25 * (1 + 1 + 2 + 1) = **0.8**

7.3 Environmental Complexity Factor

Table 7.3. Environmental complexity factor		
Environmental Factor	**Description**	**Weight**
E1	Familiar with development process	2
E2	Stable requirements	2
E3	Paradigm experience	1
E4	Unfamiliar programming language	-1

ECF = C1 + C2 * Environmental Factor Total → ECF = 1.25 + (-0.25) * (2 + 2 + 1 -1) = 0.25

7.4 Adjusted Use Case Points

UCP = round(TCF * ECF * UUCP)

Table 7.4. Adjusted use case points										
UC-1	**UC-2**	**UC-3**	**UC-4**	**UC-5**	**UC-6**	**UC-7**	**UC-8**	**UC-9**	**UC-10**	**UC-11**
1	2	2	3	5	1	3	2	2	3	2

7.4.1 Use Case 1

- Use Case Points: 1

- Duration = 1 * 28 = 28 hours

7.4.2 Use Case 2

- Use Case Points: 2
- Duration = 2 * 28 = 56 hours

7.4.3 Use Case 3

- Use Case Points: 2
- Duration = 2 * 28 = 56 hours

7.4.4 Use Case 4

- Use Case Points: 3
- Duration = 3 * 28 = 84 hours

7.4.5 Use Case 5

- Use Case Points: 5
- Duration = 5 * 28 = 140 hours

7.4.6 Use Case 6

- Use Case Points: 1
- Duration = 1 * 28 = 28 hours

7.4.7 Use Case 7

- Use Case Points: 3
- Duration = 3 * 28 = 84 hours

7.4.8 Use Case 8

- Use Case Points: 2
- Duration = 2 * 28 = 56 hours

7.4.9 Use Case 9

- Use Case Points: 2
- Duration = 2 * 28 = 56 hours

7.4.10 Use Case 10

- Use Case Points: 3
- Duration = 3 * 28 = 84 hours

7.4.11 Use Case 11

- Use Case Points: 2
- Duration = 2 * 28 = 56 hours

7.4.12 Total Duration

The total duration is calculated by adding all the duration calculations from sections 5.4.1 - 5.4.11. The calculation was made based on a 12 week development period with 12 members in the group.

28 + 56 + 56 + 84 + 140 + 28 + 84 + 56 + 56 + 84 + 56 = 728 hours

The above calculation clarifies that each team member worked approximately 5 hours/week.

8 Domain Analysis

8.1 Conceptual Model

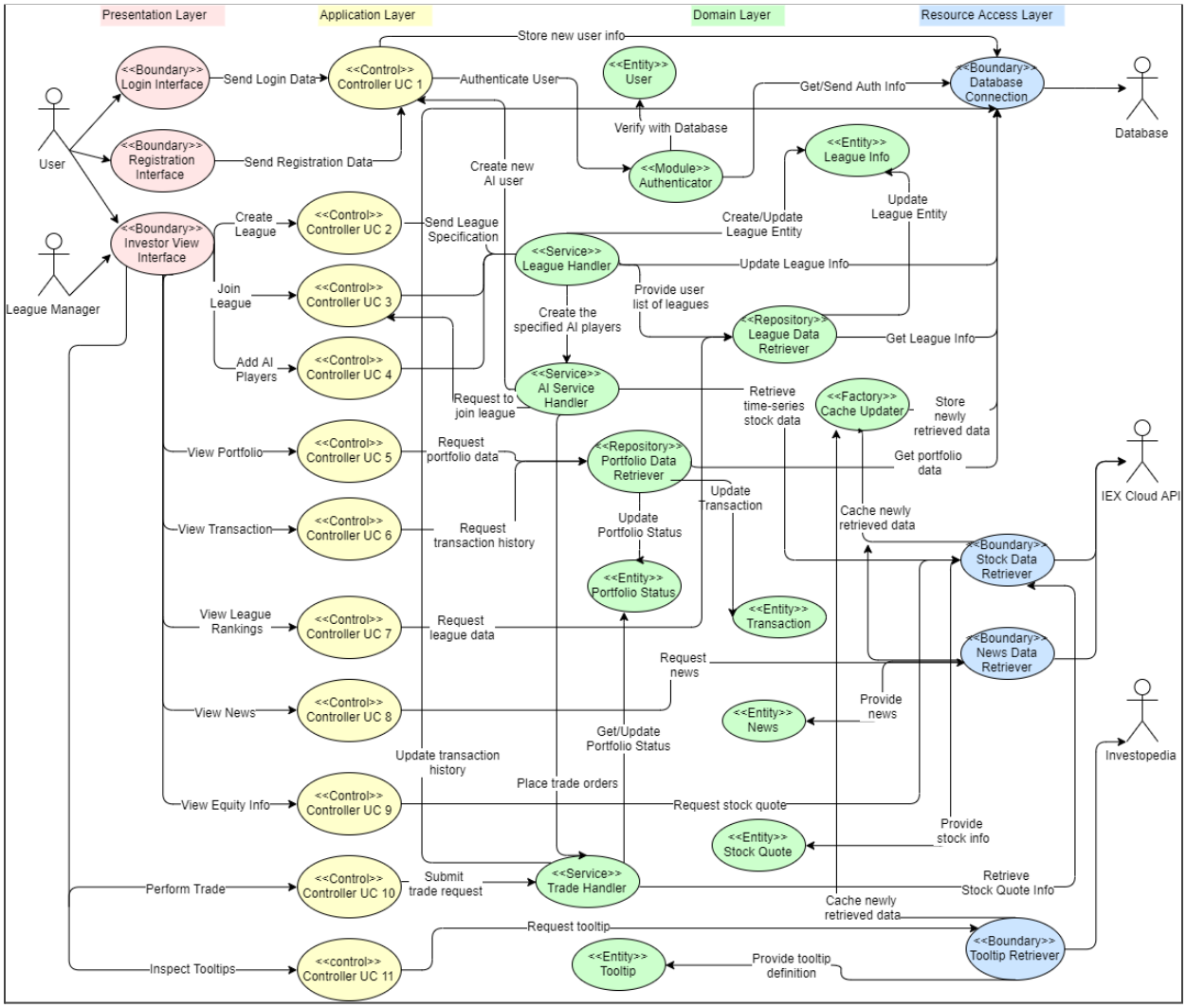

Figure 8.1. Conceptual model diagram

8.1.1 Concept Definitions

D = Doing, K = Knowing, N = Neither

Table 8.1. Concept Definitions		
Responsibility Description	**Type**	**Concept Name**

Determine proper handler for requested action and serve as a proxy for that concept to handle this data	D	Controller (UC 1 to 11)
Send and retrieve data requested from the database	D	Database Connection
Gather credentials to request authenticated access	K	Login Interface
Gather new user credentials to allow account creation	D	Registration Interface
Display investment tools and information related to a user's portfolio to the actor	N	Investor View Interface
Allows users to create and join leagues	D	League Handler
Present a form to allow the actor to place orders on equities	D	Trade Handler
Fetch real-time information pertaining to equities, such as prices and market volume	D	Stock Data Retriever
Gather news relevant to queried equities	D	News Data Retriever
Fetch information relevant to individual leagues	K	League Data Retriever
Fetch information relevant to individual portfolio in league	K	Portfolio Data Retriever
Fetch definitions from a financial website to be used as tooltips in the user interface	D	Tooltip Retriever
Container for information related to an order, such as equity, price, and volume	K	Transaction
Container for details about a particular league, such as the game settings and the user list	K	League Info
Verify whether or not valid credentials were given and allow access accordingly	D	Authenticator
Container for maintaining a user's holdings in a particular league	K	Portfolio Status
Container for information pertaining to equities, such as prices and market volume	K	Stock Quote
Container for news related to a given equity	K	News
Container for definitions of financial terms	K	Tooltip

Make trading decisions supported by mathematical models in all leagues containing AIs	D	AI Handler
Updating/Adding stock quote data and news data to database cache	D	Cache Updater

8.1.2 Association Definitions

Table 8.2. Association Definitions

Concept Pair	Association Description	Name Association
Controller ↔ Stock Data Retriever	Controller submits trade request and Stock Data Retriever retrieves stock quote info.	Request Stock Quote
Stock Quote ↔ Stock Data Receiver	Stock Data Receiver provides stock info.	Provide stock info
Controller ↔ News Data Retriever	Controller sends request to News Data Retriever to gather articles of news	Request news
News Data Retriever ↔ News	News Data Retriever provides news.	Provide news
News Data Retriever ↔ Cache Updater	News Data Retriever sends the updated news articles to the Cache Updater	Cache newly retrieved data
Controller ↔ Tooltip Retriever	Controller requests Tooltip Retriever to gather tooltip definitions	Request tooltip
Tooltip Retriever ↔ Tooltip	Tooltip retriever provides data for Tooltip	Provide tooltip definition
Tooltip Retriever ↔ Cache Updater	Tooltip retriever sends the gathered tooltips to the Cache Updater	Cache newly retrieved data
Login Interface ↔ Controller	Login Interface requests login info from the user and sends it to Controller.	Send Login Data
Controller ↔	Controller will convey the	Authenticate User

Authenticator	request to Authenticator to authenticate user	
Authenticator ↔ Database Connection	Database Connection stores or provides authorization information to the Authenticator	Get/Send Auth Info
Registration Interface ↔ Controller	Registration Interface will convey request to Controller to register the user and send registration data as well	Send Registration Data
Cache Updater ↔ Database Connection	The Cache Updater writes the updated information to the Database	Store newly retrieved data
Controller ↔ League Handler	Controller sends request to League Handler to allow users to join and create leagues	Send League Specification
League Handler ↔ Database Connection	League Handler sends the new league information to the Database	Update League Info
League Data Retriever ↔ League Info	League data retriever will update the league info	Update League Info
League Data Retriever ↔ Database Connection	The league Data Retriever requests data from the Database Connection	Get League Info
Controller ↔ Portfolio Data Retriever	Controller sends request to Portfolio Data Retriever to get portfolio information	Request Portfolio Data/Transaction History
Portfolio Data Retriever ↔ Database Connection	Portfolio Data Retriever requests data from Database Connection	Get Portfolio Data
Portfolio Data Retriever ↔ Portfolio Status	Portfolio data retriever grabs portfolio information to be displayed in portfolio status	Update Portfolio Status
Controller ↔ Trade Handler	Controller requests Trade Handler to execute the order	Submit Trade Request

Portfolio Data Retriever ↔ Transaction	Portfolio data retriever gets and updates transaction	Update Transaction
Trade Handler ↔ Database Connection	Trade Handler passes order data to be passed to the Database Connection for storage, updating transaction history and portfolio status	Update Transaction History
Trade Handler ↔ Stock Data Retriever	Trade Handler retrieves the most recent price of the current stock	Retrieve Stock Quote Info
Trade Handler ↔ Portfolio Status	Trade Handler gets and updates the current portfolio to reflect the desired user transaction	Get/Update Portfolio Status
AI Service Handler ↔ Stock Data Retriever	AI Service Handler needs recent time-series as well as historical time-series data in order to decide which trades to make	Retrieve time-series stock data
AI Service Handler ↔ Controller	Send info about new AI user so that it can create user and add it to league	Create new AI user and join league
Stock Data Retriever ↔ Cache Updater	Stock Data Retriever sends the updated stock data to the Database Connection	Cache newly retrieved data
AI Service Handler ↔ Trade Handler	AI Service Handler needs to place a trade order and must submit this request to trade handler	Place trade orders
League Handler ↔ AI Service Handler	League handler requests AI Service to create AI Players	Create Specified AI Players
League Handler ↔ League Data Retriever	League handler provides the user with a list of leagues	Provide User List of Leagues
Controller ↔ Database Connection	Controller sends new user info to Database Connection for storage (during the registration process)	Store new user info

6.1.3 Attribute Definitions

Table 8.3. Attribute Definitions

Concept	Attributes	Attribute Description
Login Interface	User's Identity	Credentials used for logging into a known user account
Registration Interface	New User Credentials	Email, username, and password for registering a new user account
League Handler	List of Joined Leagues	Enumerate the leagues that a user is currently participating in
	List of Available Leagues	Shows the leagues that are available
	League Creation Settings	List of specified settings for the requested league
Trade Handler	Transaction	Transaction entity, used to describe the parameters of an order
Stock Data Retriever	Ticker Symbol	Copied from Stock Quote entity, used to search what the current state of an equity is in order to fill in data
News Data Retriever	Ticker Symbol	Copied from News entity, used to retrieve news related to desired stock
League Data Retriever	Search Query	Used to search for a specific league or a list of leagues (i.e. all leagues with public visibility)
Portfolio Data Retriever	Portfolio ID	Used to identify which portfolio's data needs to be retrieved
Stock Quote	Ticker Symbol	Identifies what stock the data is related to

	Price Data	Contains price data from time range queried
News	Ticker Symbol	Used to identify which stock the data is related to
	List of Articles	Articles related to the ticker symbols queried
Transaction	Current Stock Data	Ticker Name, Trade Type, Trade Amount, Total Cost
	Date & Time	Used to store the current date and time of the transaction in the database, which can be viewed later
League Info	Settings	Game settings defined by the league manager
	Participating Users	List of users that are in the league
Portfolio Status	League ID	League that this portfolio is a part of
	Username	User that owns this portfolio
	Current Holdings	List of equities that a user owns, with their respective volume
Tooltip	Word	Word that is defined by the tooltip
	Definition	The body of the tooltip

8.1.3 Traceability Matrix

Table 8.3. Traceability Matrix

	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC- 9	UC-10	UC- 11
Controller	X	X	X	X	X	X	X	X	X	X	X
Database Connection	X	X	X	X	X	X	X	X	X	X	

Login Interface	X										
Registration Interface	X										
Investor View Interface								X	X	X	
League Handler		X	X	X							
Trade Handler										X	
League Data Retriever							X				
Portfolio Data Retriever					X	X				X	
Stock Data Retriever					X		X		X	X	
News Data Retriever								X	X		
Tooltip Retriever											X
Transaction						X				X	
League Info		X					X				
Authenticator	X										
Portfolio Status					X		X				
Stock Quote					X		X		X		
News								X	X		
Tooltip											X
AI Handler		X		X							
Cache Updater					X		X	X	X	X	X

With the exception of the Controller, all classes evolved directly from their domain concept. This is directly related to the fact that the domain concepts evolved from the sequence of actions required for each use case. Thus, it makes sense that each domain evolves into their own class.

The only exception to this was the Controller class, which resulted as a combination of domain concepts “Controllers UC 1 to 11”. There is no reason to separate all of our controllers in this manner aside from clarity in our diagrams and explanations. Thus, these domains get combined into one Controller class.

8.2 System Operation Contracts

Refer to the preconditions and postconditions of section 6.3.3 for the system contracts of each use case.

8.3 Data Model & Persistent Data Storage

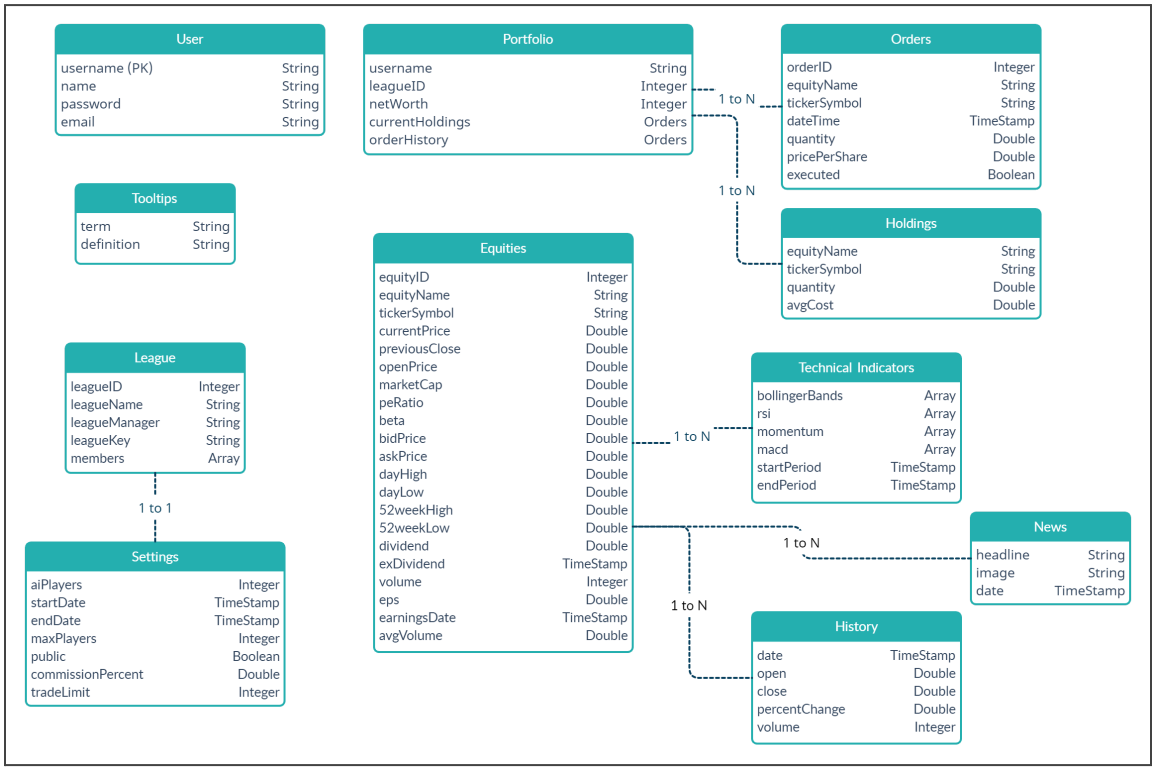

Figure 8.2. Data model diagram

Our system requires that data be saved to outlive a single execution of the system. This data includes user profile information, league information, portfolio holdings and equity data. We will be storing this information in a non-relational database provided by MongoDB Atlas, a cloud database service. Non-relational databases are ideal for storing a lot of data that will change frequently. In time it will improve operational efficiency and database performance over the relational model. The schema’s for these persistent objects are outlined below:

8.3.1 User Account Schema

```
User = {
        “username” : String,
        “email” : String,
        “password ” : String,
}
```

This schema defines the user profile. Users will be identified through unique usernames. AI trading bots will also be stored as unique, identifiable users as well. All users will have passwords so their accounts remain secure.

8.3.2 League Schema

```
League = {
        “leagueID”: Integer,
        “leagueName”: String,
        “leagueManager”: String,
        “leagueKey”: String,
        “members”: Array
        “settings”: [
        {
                “aiPlayer”: Integer,
                “startDate”: TimeStamp,
                “endDate”: TimeStamp,
                “maxPlayers”: Integer,
                “public”: Boolean
                “commissionPercent”: Double
                “tradeLimit”: Integer
        }
        ]
}
```

For every league, there will be a unique identifier which allows for multiple leagues to have the same name, if desired. All the usernames of members in the league will be stored. Each league will also have one league manager that controls the settings. Settings was included as an embedded document to describe the one-to-one relationship between a league and its settings. The league’s settings are viewed in context of the league and thus belongs to the league document.

8.3.3 Equity Schema

```
Equity = {
        “equityName”: Integer ,
        “tickerSymbol”: String,
        “currentPrice”: Double,
        “previousClose”: Double,
        “openPrice”: Double,
        “marketCap”: Double,
        “peRatio”: Double,
        “beta”: Double,
        “bidPrice”: Double,
        “askPrice”: Double,
        “dayHigh”: Double,
        “dayLow”: Double,
        “week52High”: Double,
        “week52Low”: Double,
        “dividend”: Double,
        “exDividend”: TimeStamp,
        “volume”: Integer,
        “eps”: Double,
        “earningsDate”: TimeStamp,
        “avgVolume”: Double ,
        “news”: [
        {
                “headline”: String,
                “image”: String,
                “date”: TimeStamp,
        }
        ]
        “historicalPrices”: [
                “fiveday”: [
                        {
                                “date”: TimeStamp,
                                “open”: Double,
                                “close”: Double,
                                “percentChange”: Double ,
                                “volume”: Integer,
                        }
                ],
                “onemonth”: [
```

```
        {
                "date": TimeStamp,
                "open": Double,
                "close": Double,
                "percentChange": Double ,
                "volume": Integer,
        }
],
"sixmonth": [
        {
                "date": TimeStamp,
                "open": Double,
                "close": Double,
                "percentChange": Double ,
                "volume": Integer,
        }
],
"yeartodate": [
        {
                "date": TimeStamp,
                "open": Double,
                "close": Double,
                "percentChange": Double ,
                "volume": Integer,
        }
],
"oneyear": [
        {
                "date": TimeStamp,
                "open": Double,
                "close": Double,
                "percentChange": Double ,
                "volume": Integer,
        }
],
"fiveyear": [
        {
                "date": TimeStamp,
                "open": Double,
                "close": Double,
```

```
                    “percentChange”: Double ,
                    “volume”: Integer,
                }
            ]
        ]
        “intradayPrices”: [
        {
            “time”: TimeStamp,
            “open”: Double,
            “close”: Double,
            “percentChange”: Double ,
            “volume”: Integer,

        }
        ]
}
```

Every equity can be identified by its unique ticker symbol. All the information about every equity will be stored in this document. News, historical prices and technical indicators are all embedded documents as this exemplifies the one-to-many relationship between the equity and each of these categories. Each equity will have multiple news articles stored and historical data that spans over a large time period.

8.3.4 Portfolio Schema

```
Portfolio = {
    “username”: String,
    “leagueID”: Integer,
    “cashAvailable” : Double,
    “netWorth”: Double,
    “currentHoldings”: [
    {
        “equityName”: String,
        “tickerSymbol”: String,
        “Quantity”: Double,
        “avgCost”: Double,
    }
    ]
    “orderHistory”: [
    {
```

```
            "orderID": Integer,
            "orderType": String,
            "tickerSymbol": String,
            "timePlaced": TimeStamp,
            "Quantity": Double,
            "pricePerShare": Double,
            "expiryDate": TimeStamp,
            "executed": Boolean,
            "activeLimitOrder": Boolean
        }
        ]
}
```

For every user participating in a league, there is one portfolio. We modeled the one-to-many relationship between a user's portfolio and their current holdings by embedding it as its own document. Every user in a league can purchase multiple different equities in that corresponding portfolio. Similarly, the one-to-many relationship between a user's portfolio and all their executed orders is also shown here by using embedding. A user in a league is allowed to place multiple orders corresponding to that portfolio. All order history will be saved in this document and can easily be retrieved per instance.

8.3.5 Tooltips Schema

```
Tooltips = {
        "term": String,
        "definition": String,
}
```

This schema defines our tooltips feature. Definitions for all the financial terms will be pulled from Investopedia and stored here. They will be identifiable by name.

8.4 Mathematical Model

Our AI bots place trades using 3 main algorithms: mean reversion, momentum trading and candlestick patterns (for more about the algorithms, see section 12.1). These algorithms make use of the following technical indicators which are calculated by using historical price data received from the yfinance library in Python. Our AI service has a Python backend where we retrieve the information through Python scripts and transfer it to the rest of the application. When this information is needed, it can easily be retrieved

and used to determine the best time to trade an equity. These indicators cannot be used alone to solely determine the price direction of an equity. However, when used together, they work to create a somewhat reliable basis for placing trades.

8.4.1 Bollinger Bands

The Bollinger Bands are a technical analysis tool that comprises of three trend lines: a lower band, a middle band and an upper band. The middle band represents the Simple Moving Average over a 20 day period, while the upper and lower bands are plotted two standard deviations away from the SMA, both positively and negatively [13]. These bands helped to identify oversold and overbought conditions. The upper and lower support lines help give traders confidence that the price is moving as expected. If the prices continually touch the upper band, it means the equity is being overbought, and if it continually touches the lower band, it means it is being oversold. The majority of the price action should occur between 2 of the bands. If prices breakout of this region, it is considered a major event. The bands expand if the price of the equity becomes volatile and will contract if the equity displays a tight trading pattern.

8.4.2 Simple Moving Average

The Simple Moving Average (SMA) is a technical indicator that shows the average of prices over a selected time period. It helps to determine if a stock trend will continue or if it is time for a reversal. A declining moving average predicts a downtrend, whereas a rising moving average would mean an uptrend. The SMA uses the arithmetic mean to smooth out price data over time. It is calculated by adding recent closing prices over a specified time period and then dividing by the number of time periods [14]. This time period is determined by the trader. The shorter the time periods, the more sensitive the average will be to price changes.

The *n-day simple moving average* for day d can be calculated from:

$$A_d = \frac{\sum_{i=1}^{n} M_{(d-i)+1}}{n}$$ where $n \leq d$ and M_i is the price on day i

8.4.3 Standard Deviation

The standard deviation is a statistic that helps measure the volatility of an equity. It measures the amount of variation relative to the average price [15]. If the prices are further away from the mean, this means that there is a larger spread of prices and thus a higher standard deviation. Higher standard deviations indicate

volatility in equity price action. Equities with low standard deviation are much more stable and generally have lower risk.

It can calculated as the square root of variance:

$$\text{SD} = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}}$$

where x_i is the price on a certain day in the time period, $\bar{x}$ is the mean price of the time period, and n is the number of days in the time period

8.4.4 Relative Strength Index

The relative strength index (RSI) is a technical indicator used to measure the magnitude of recent changes in price to determine if the equity is exhibiting overbought or oversold conditions. The RSI is typically measured on a scale of 0 to 100, where a value of 70 or above indicates an equity being overbought while a value of 30 or below indicates an oversold condition [16]. When an equity is overbought, this generally means that the equity is overvalued and is due for a reversal soon. When an equity is oversold, it means the equity is undervalued and has potential for a price bounce.

RSI can be calculated using the following formula:

$$RSI = 100 - \frac{100}{1+RS}$$

where RS = average gain / average loss

8.4.5 MACD

The moving average convergence divergence (MACD) is a momentum indicator that displays the relationship between two moving averages of an equity. It is represented in the form of a trend line. When prices cross above the line, it is time to buy and when prices fall below the line, it is time to sell. It ultimately helps traders determine whether the bullish or bearish movement is diminishing. The MACD is calculated by subtracting the 26 day period of the exponential moving average from the 12 day period of the exponential moving average [17].

9 Interaction Diagrams

9.1 Use Case UC - 1: Login

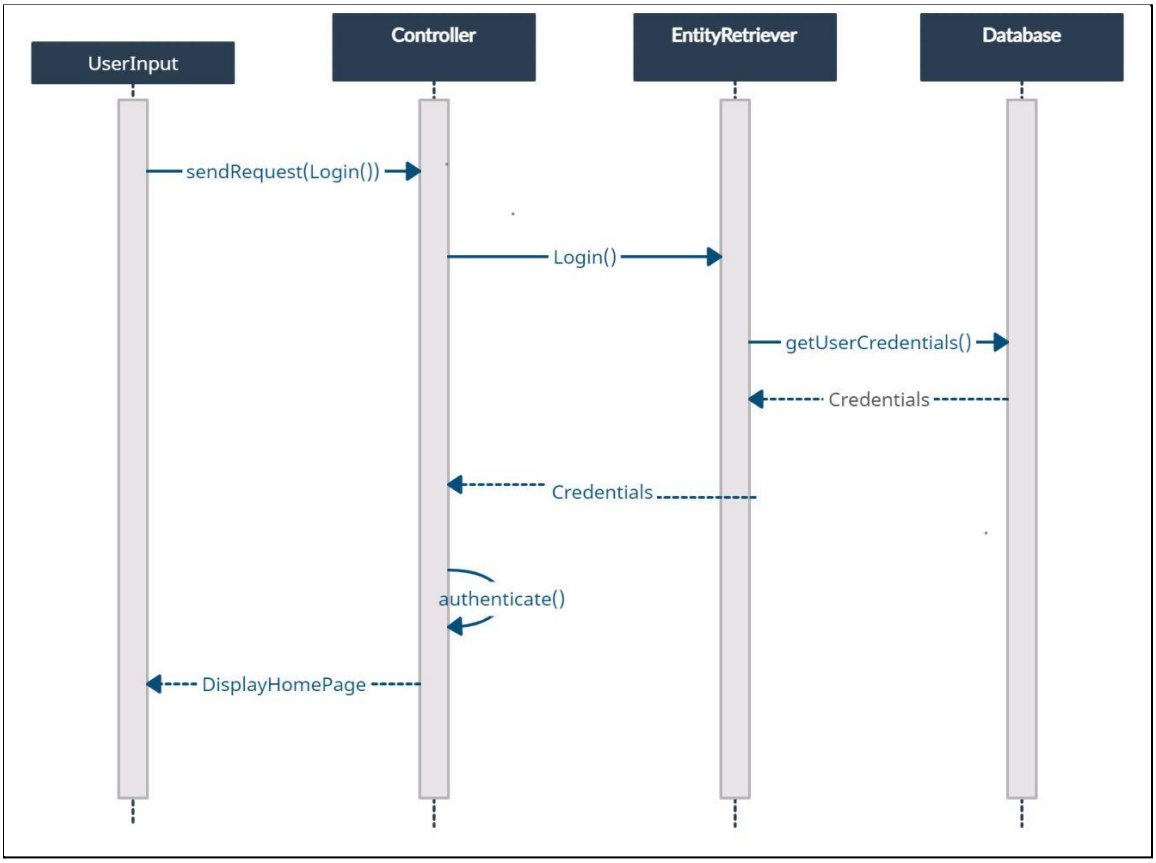

Figure 9.1. Interaction diagram: Login

The diagram above depicts the interaction between classes when the initiating actor comes across 'UC-1: Login'. The investor may choose to login after creating an account. They must enter their username and password which serves as the **UserInput.** This is handled by the **Controller** which then requests from the **Database** the password corresponding with the entered username. The request to the database is handled by the **EntityRetriever,** which retrieves the information from the database and then returns the credentials to the controller. The **Controller** verifies that the password the user entered is correct and then takes the user to the home page. The design principles employed in this use case are the **High Cohesion Principle** and the **Expert Doer Principle.** This use case follows the **Publisher-Subscriber Design Pattern**. The user/client is the subscriber or observer and waits for the response from the controller. This is shown through the indirect communication between them and the increased cohesion.

9.2 Use Case UC - 2: Create League

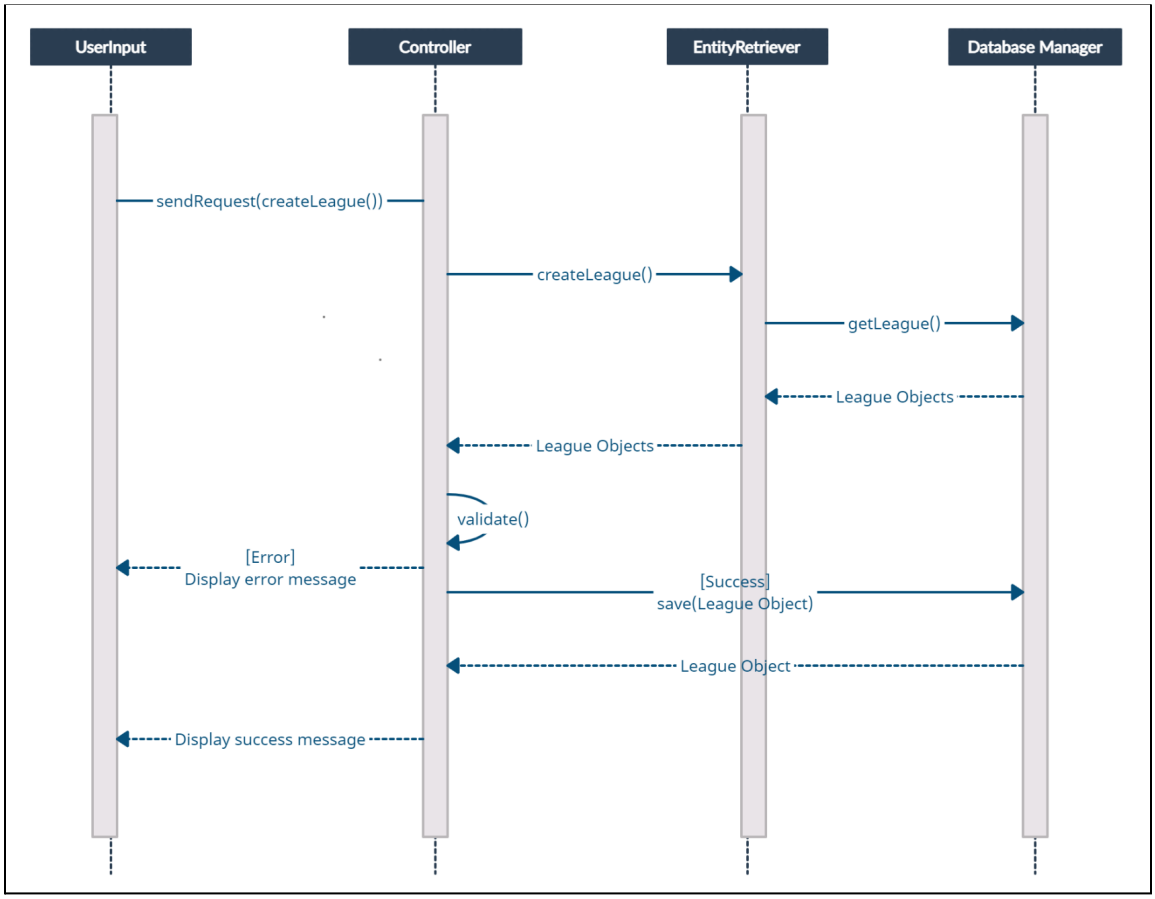

Figure 9.2. Interaction diagram: Create League

This diagram shows the interactions between classes in UC-2: Create League. A user can create a league on the Create League Page, which acts as **UserInput.** The **UserInput** sends a request to the controller, which handles the request and then contacts the **EntityRetriever** retrieves all leagues that exist in the database in order to validate against. These validation checks serve to make sure that there are no duplicate leagues and that all league attributes are valid. If this validation is successful, the user will receive feedback and the new league object will be saved to the **Database** and the user that created the league will be saved to the user list as a league manager. Otherwise, an error message is displayed. The **Expert Doer** and **High Cohesion Design Principles** are applied here. This use case also follows the **Publisher-Subscriber Design Pattern**.

9.3 Use Case UC - 3: Join League

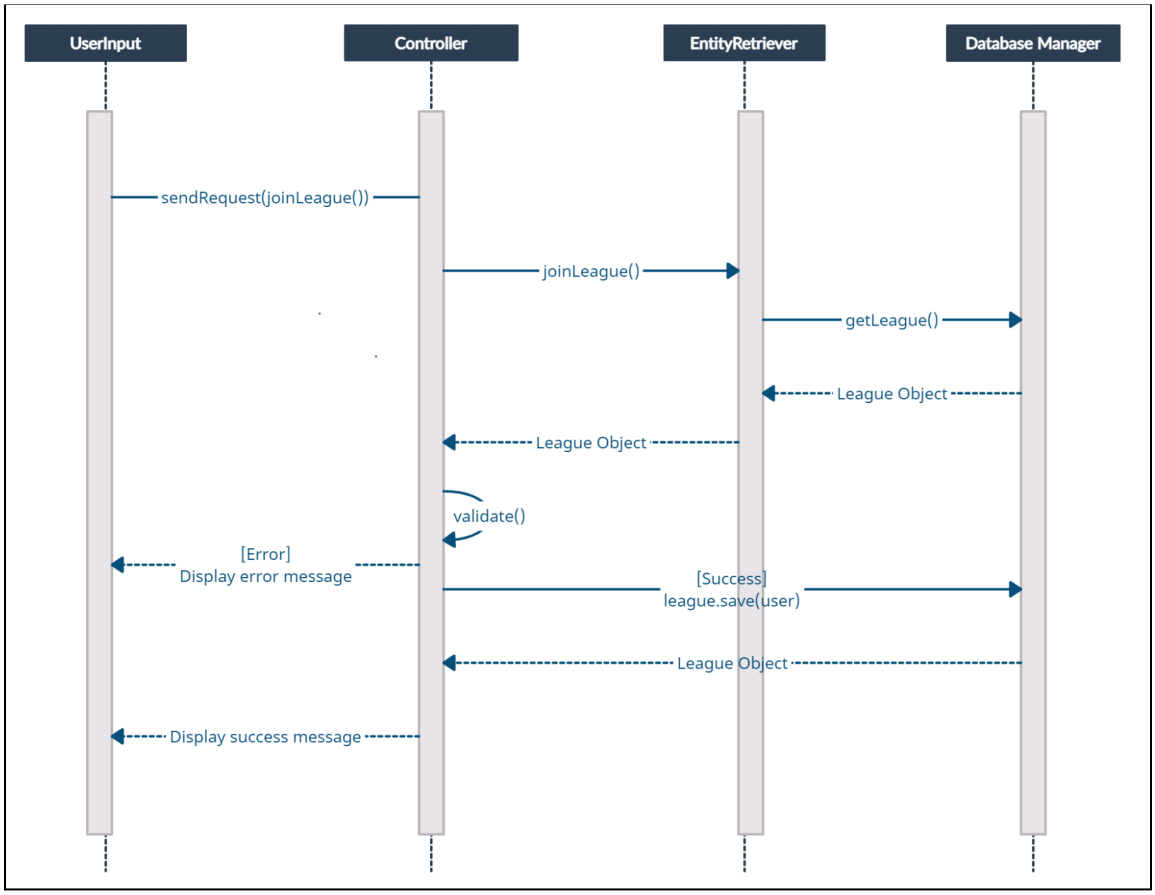

Figure 9.3. Interaction diagram: Join League

This diagram shows the interactions between classes in UC-3: Join League. A user can join a league on the Join League Page, which acts as **UserInput.** The **UserInput** sends a request to the **Controller**, which handles the request and then contacts the **Entity Retriever** to retrieve the league that the user would like to join. After the league is retrieved, validation is performed to check if the user is already part of the league and validate the league key if the league is private. If validation is successful, a success message is displayed and the user is saved to the user list in the respective league in the **Database**. Otherwise, display an error message. The **Expert Doer** and **High Cohesion Design Principles** are applied here. This use case also follows the **Publisher-Subscriber Design Pattern**.

9.4 Use Case UC - 4: Add AI Players

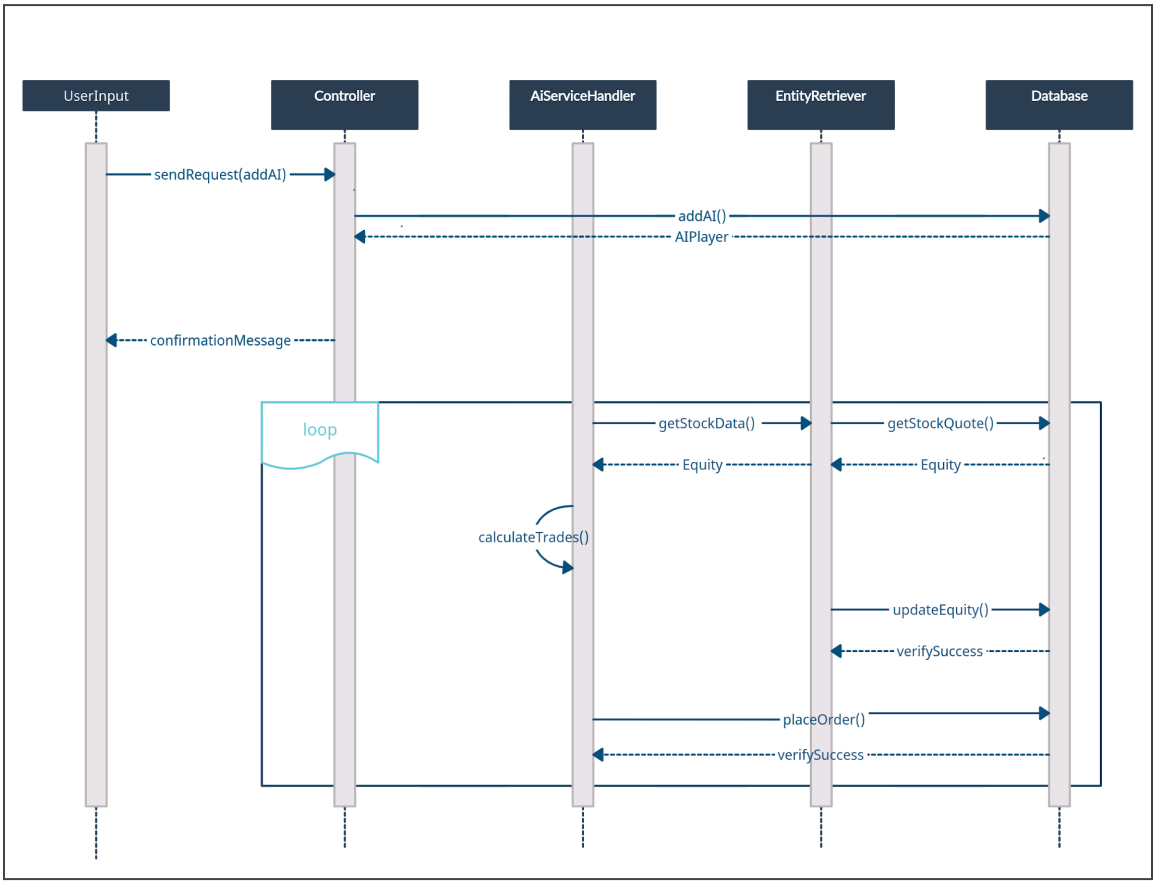

Figure 9.4. Interaction diagram: AI Players

The above diagram depicts the interactions between classes in UC-4: Add AI Players. A user that is a league manager may choose to add an AI player on the createLeague page of the **UserInput**. This is then handled by the **Controller** which creates and stores a new AI player in the **Database**. The user is notified of the creation through a confirmation message. The gameplay of the AI player is defined by a loop, in which the **AiServiceHandler** gets stock data and calculates what trades it should make. When it has determined the optimal trades, it will execute placeOrder() to place orders in the system. The **Expert Doer** and **High Cohesion Design Principles** are applied here. Each class has a focused and well-defined role that ensures that all information is routed properly and efficiently used. This use case also follows the **Publisher-Subscriber Design Pattern**.

9.5 Use Case UC - 5: View Portfolio Status & Information

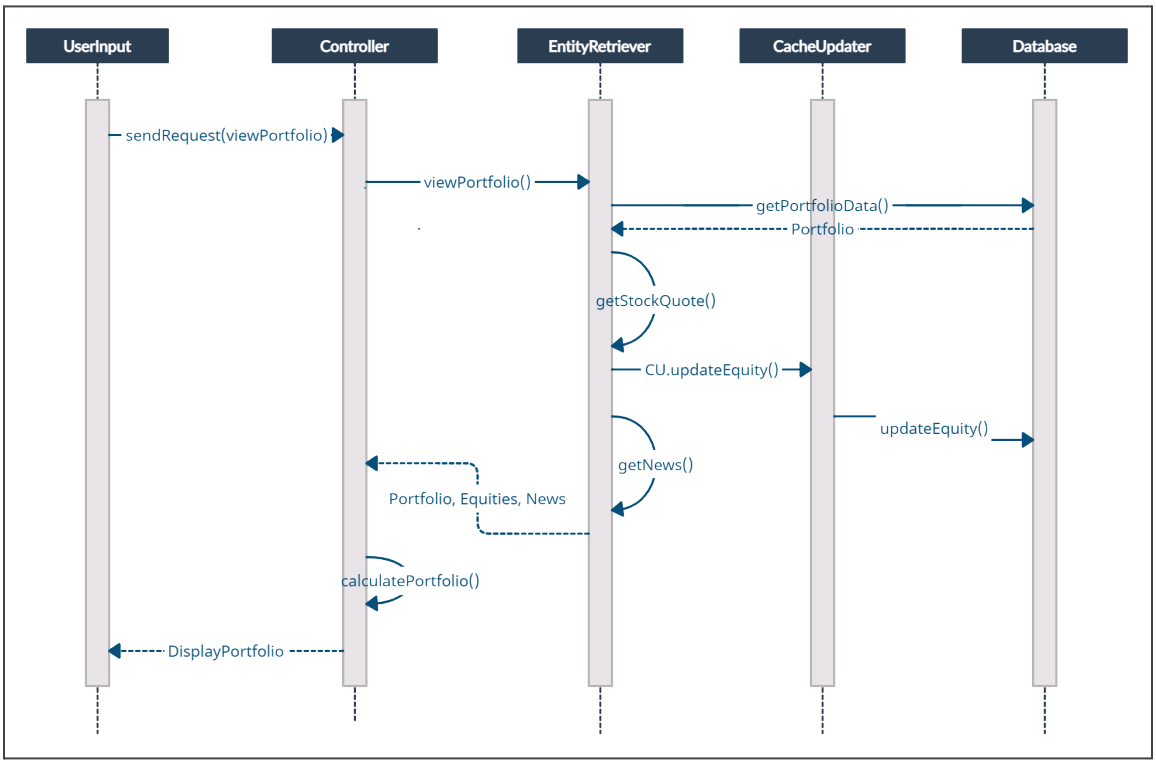

Figure 9.5. Interaction diagram: View Portfolio Information

This diagram shows the interaction between actors in 'UC-5: View Portfolio Status and Information' in order to view the Investor's current positions in an active league and keep track of their total return. Once the user is logged in and also a member of an active league, they will have access to a page that displays their portfolio. The **Controller** requests the user's current positions and league information from the **Database;** it also requests all relevant information from the **Entity Retriver** to the user's portfolio including curated news, historical price data, etc. The **Controller** finally displays the user's portfolio.

The design principles employed are the **High Cohesion Principle** and the **Expert Doer Principle**. The High Cohesion Principle is exhibited in all the actors which have specific tasks relevant to their functionalities, which limits the number of computational responsibilities. The Expert Doer principle is also exhibited by the **Entity Retriever** which is solely responsible for the specific function of providing updated news and equity information. This use case also follows the **Publisher-Subscriber Design Pattern**.

9.6 Use Case UC - 6: Inspect Transaction History

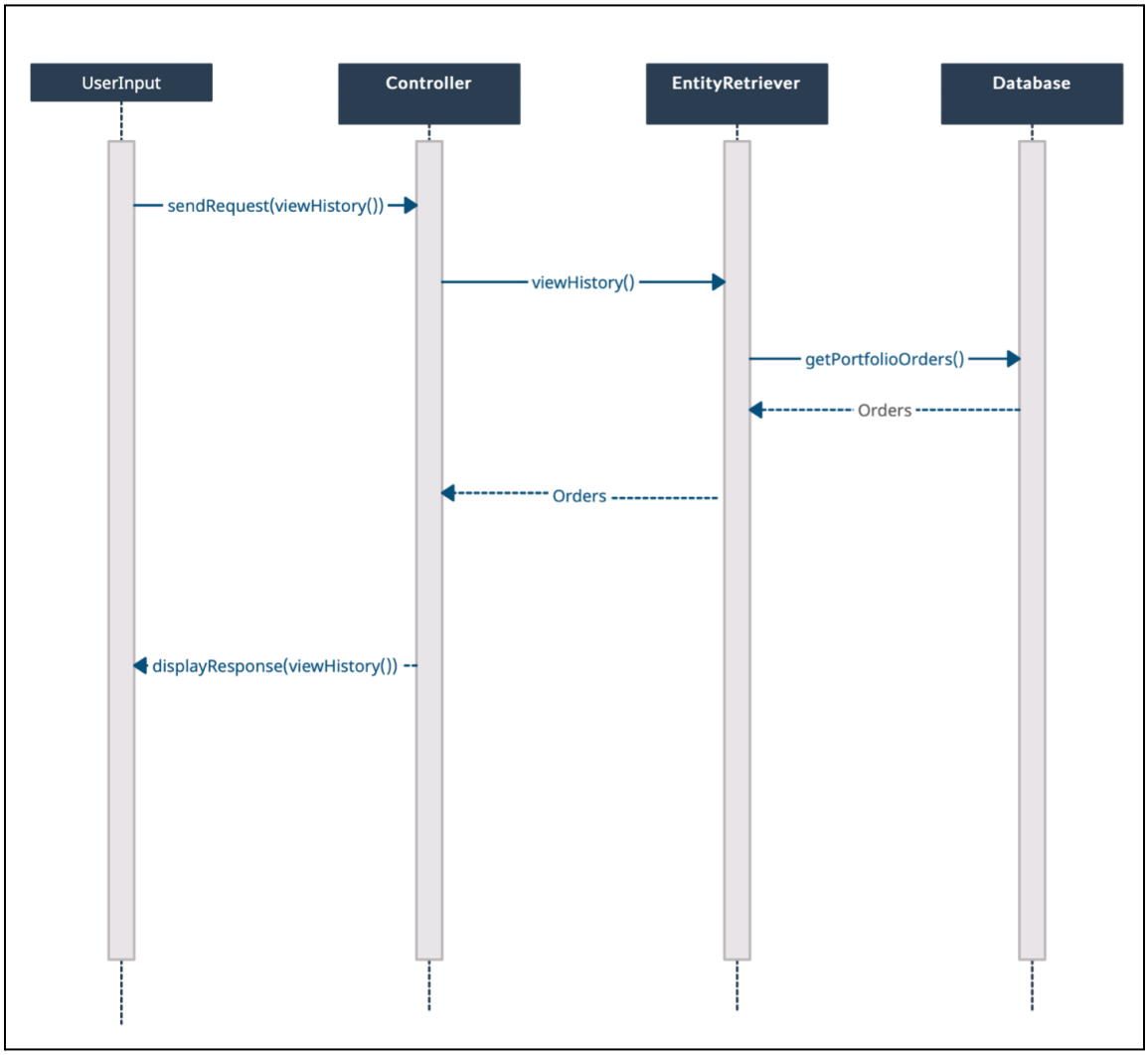

Figure 9.6. Interaction diagram: Inspect Transaction History

This diagram shows the interactions the actor will encounter in 'UC-6: Inspect Transaction History'. This will allow the actor to view the transaction histories of any of their past or current leagues. Once the user is logged in and selects a certain league to view the history of, the **Controller** requests the order history information. The **Controller** will request and pull information about the order history from the **Entity Controller**. The **Entity Controller** will retrieve this information from the **Database.** Then the **Entity Controller** will return the order history to the controller. Finally the **Controller** will display the order history. The design principles implemented here are the **Expert Doer Design** and **High Cohesion Design**. These designs focus on specified roles and that makes sure that the information is routed logically. This use case follows the **Publisher-Subscriber Design Pattern**.

9.7 Use Case UC - 7: View League & Rankings

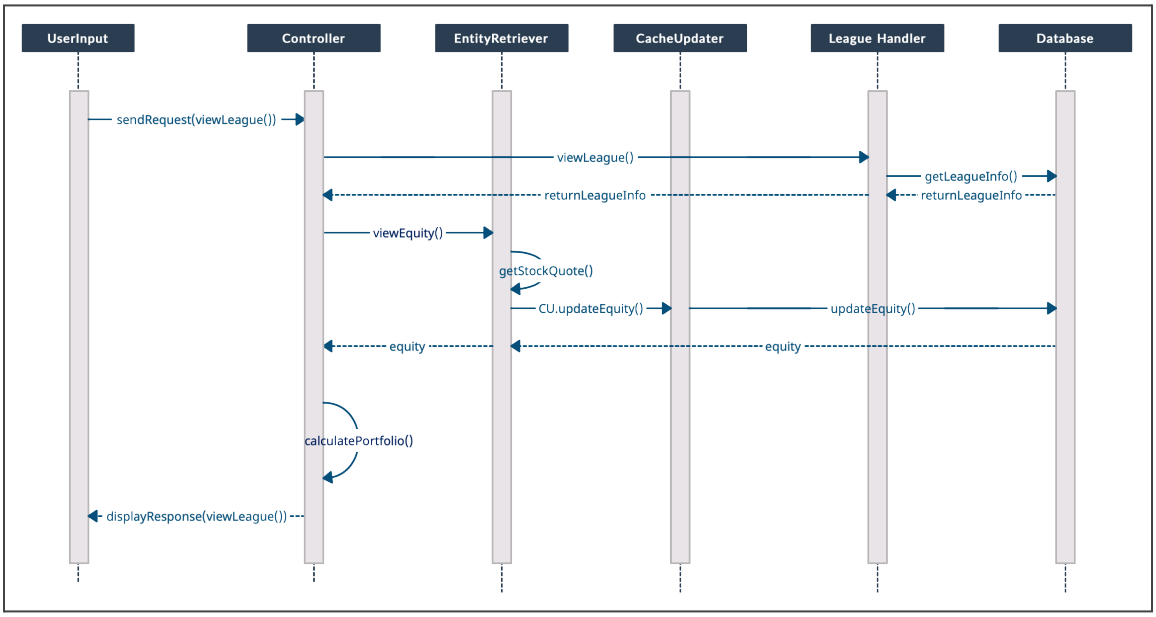

Figure 9.7. Interaction diagram: League and Rankings

This diagram shows the interactions the actor will encounter in 'UC-7:View League & Rankings'. This will allow the actor to view their rank in their current leagues as well as their league information. Once the user is logged in and selects a certain league the **Controller** requests the league information. The **Controller** will request and pull information about the stock prices from the **Entity Controller**. The **Entity Controller** will first check for cached stock prices from the database and then it will retrieve information from the database for cached prices. Then the **Entity Controller** will return the stock prices to the controller. Finally the **Controller** will display the user's leagues information and their ranking. The design principles implemented here are the **Expert Doer Design** and the **High Cohesion Design**. These designs focus on specified roles and that makes sure that the information is routed logically. This use case follows the **Publisher-Subscriber Design Pattern**.

9.8 Use Case UC - 8: News

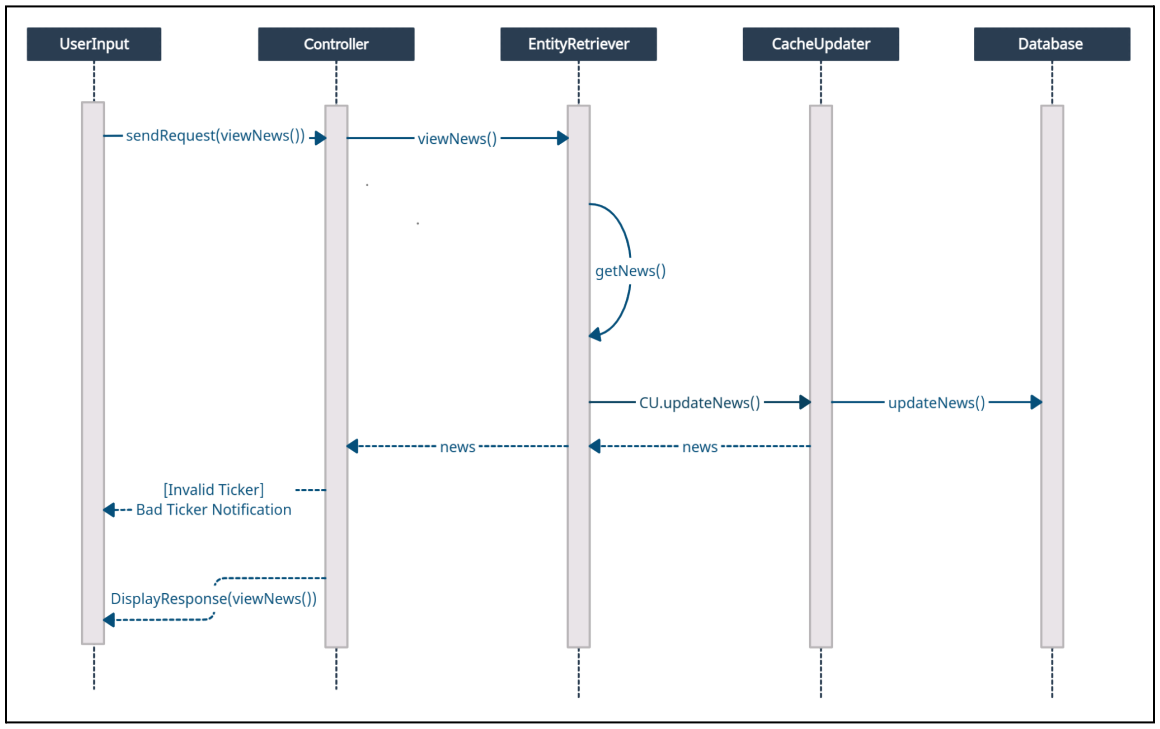

Figure 9.8. Interaction diagram: News

This diagram depicts the interaction between classes when the initiating actor encounters 'UC-8: News'. This provides functionality for users to view news that is relevant to an equity they are viewing or their entire portfolio. The **UserInput** sends a request to the **Controller** to view the news. The **EntityRetriever** then calls getNews() and calls upon the **CacheUpdater** to update the news. Then, it stores the updated news with updateNews() into the **Database** and returns it to the **Controller**. The **Controller** then checks if there is a valid ticker for the requested news and displays the news to the user. The design principles utilized in this use case are the **Expert Doer Principle**, the **High Cohesion Principle**. The use case follows the **Publisher-Subscriber Design Pattern**. The client is the subscriber and awaits any response sent by the controller. This is clearly exhibited through the sendRequest() function in the UserInput, which allows for indirect communication and helps to exemplify the high cohesion principle.

9.9 Use Case UC - 9: View Equity Information

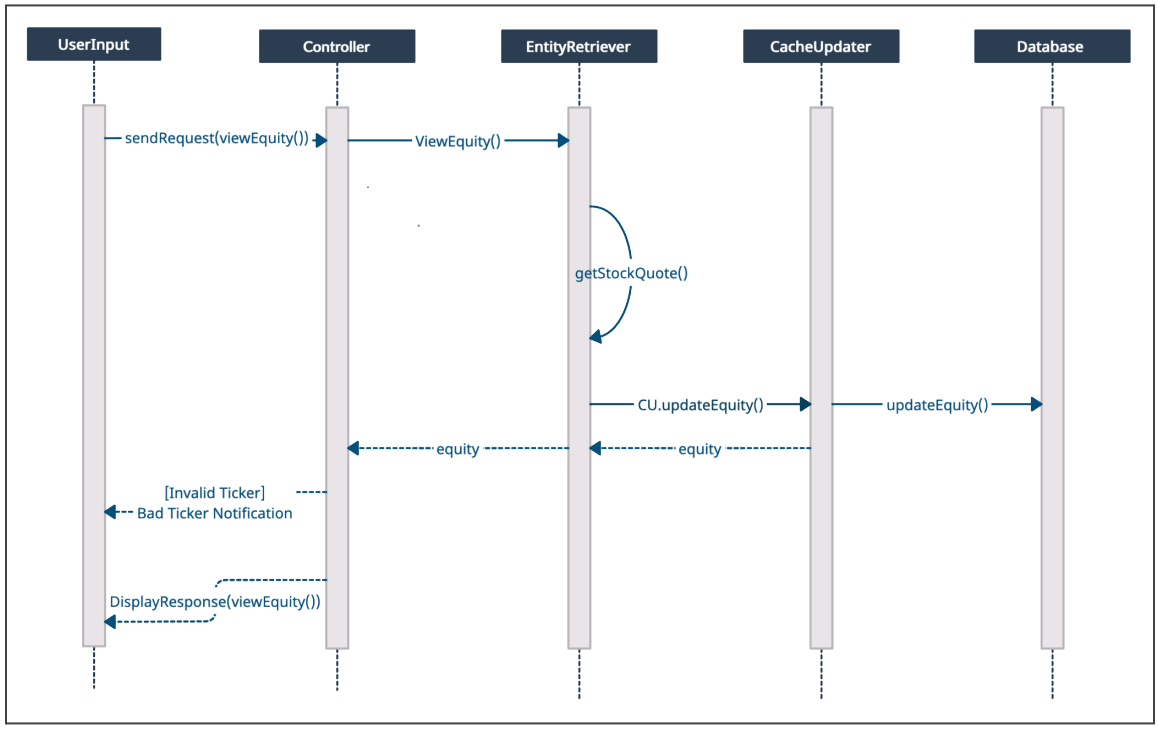

Figure 9.9. Interaction diagram: Equity Information

This diagram shows the interactions the actor will encounter in 'UC-9:View Equity Information.' The **userInput** sends a request to the **Controller** to view the stock information. The **EntityRetriever** calls getStockQuote() and then sends a request to **CacheUpdater** to update the equity. Then it gets the information for updateEquity() from the **Database**. The cache updater then returns the equity data to the controller. The controller checks if there is an invalid ticker and then it displays the equity information to the user. The design principles that are used are the **Expert Doer Principle** and the **High Cohesion Principle**. These principles are shown in the userInput and the API because both have specific tasks to complete. This use case also follows the **Publisher-Subscriber Design Pattern**.

9.10 Use Case UC - 10: Make Trades

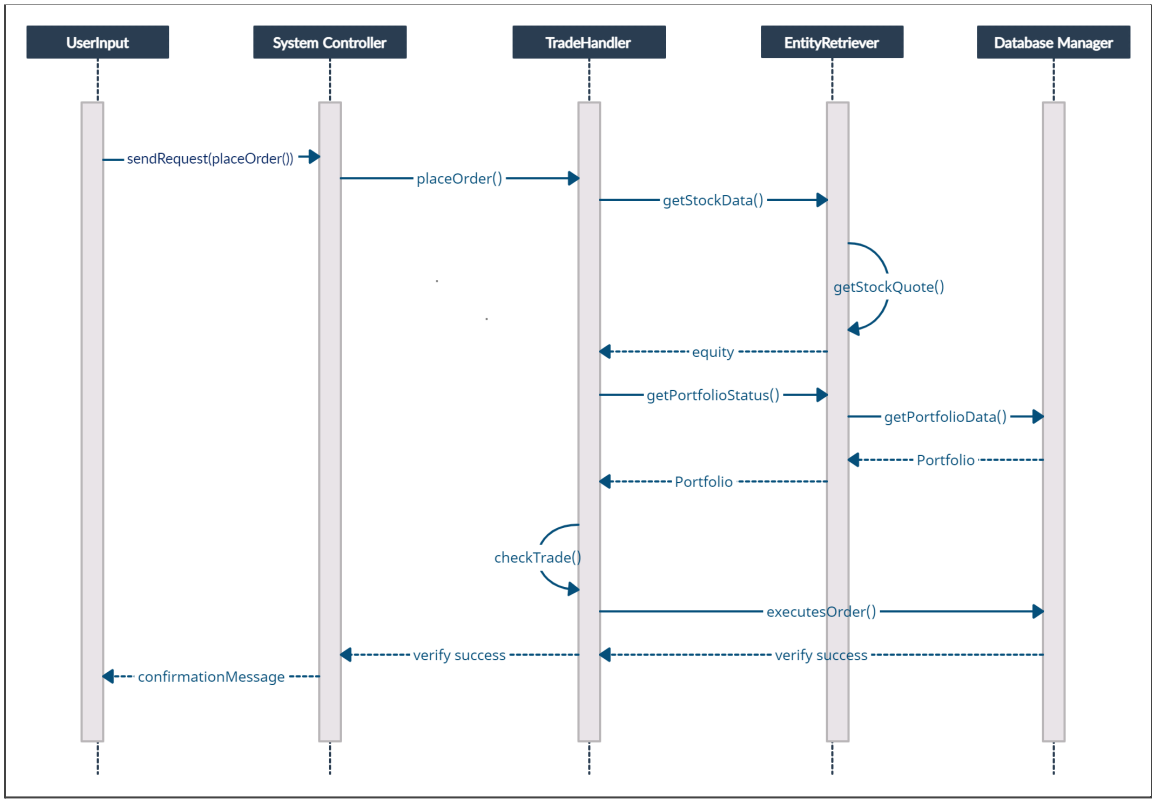

Figure 9.10. Interaction diagram: Trade

This diagram shows the interactions the actor will encounter in "UC-10: Make Trades". First, the **userInput** sends a place order request to the **System Controller**. The placeOrder function is then called from the controller to the trade handler. The **Trade Handler** then gets the stock quote from the **Entity Retriever** and returns it back to the Trade Handler. Before checkTrade occurs, the portfolio status must be updated from the **Database Manager**. Once checkTrade is completed, the system executes the order and the user is then sent a confirmation message. The design principles that are used are the **Expert Doer Principle** and the **High Cohesion Principle**. The principles are used because the diagram determines where to delegate responsibilities as it shows how each class receives specific tasks to be completed. This use case also follows the **Publisher-Subscriber Design Pattern**.

9.11 Use Case UC - 11: Tooltips and Summary

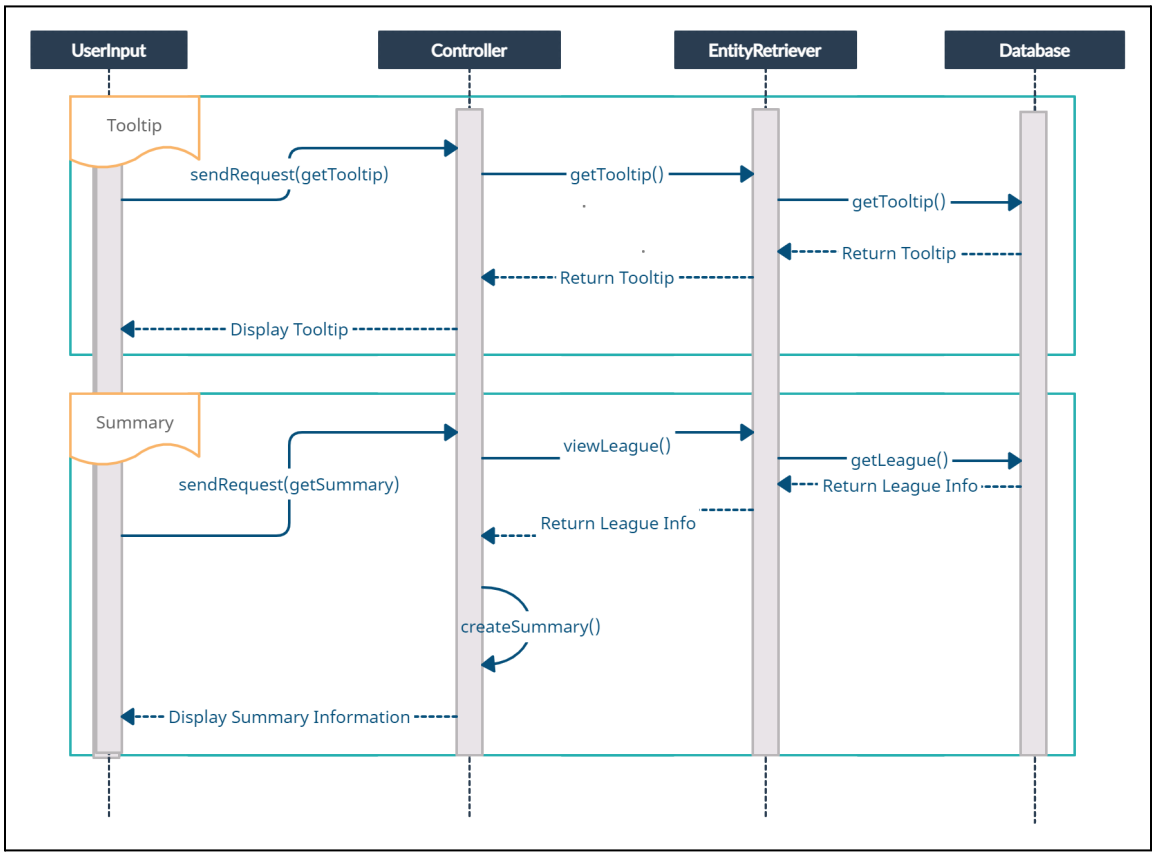

Figure 9.11. Interaction diagram: Tooltips & Summary

This diagram depicts the interaction between classes when the initiating actor encounters 'UC-11: Tooltips and Summary'. This provides functionality for users to view news that is relevant to an equity they are viewing or their entire portfolio. The **UserInput** sends a request to the **Controller** to view the tooltip or the summary. The **EntityRetriever** then calls getTooltip() if the request wants a tooltip, or it calls getLeague() for league information. After retrieving the information from the **Database**, the **EntityRetriever** returns the information back to the **Controller.** If a summary was requested, then the **Controller** must use the league information to generate the summary as the response. Otherwise, it will simply use the tooltip as the response. The **Controller** then sends back the response to **UserInput** for the initiating actor to see. The design principles utilized in this use case are the **Expert Doer Principle** and the **High Cohesion Principle**. The use case follows the **Publisher-Subscriber** design pattern. The client is the subscriber and awaits any response sent by the controller. This is clearly exhibited through the sendRequest() function in the UserInput, which allows for indirect communication and helps to exemplify the high cohesion principle.

10 Class Diagram & Interface Specification

10.1 Class Diagram

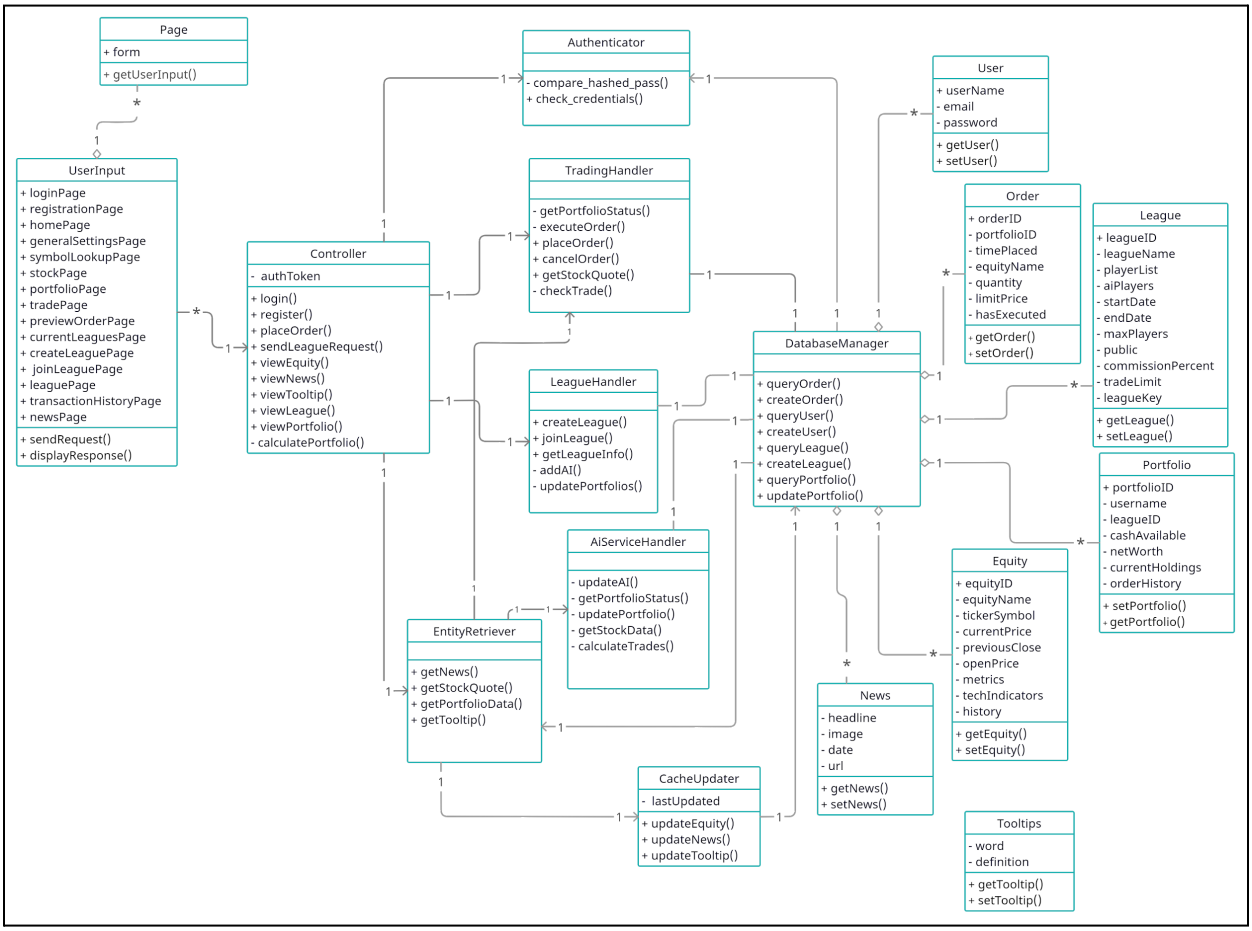

Figure 10.1 Class Diagram

10.2 Data Types and Operation Signatures

10.2.1 CD-1: AiServiceHandler

This class is responsible for operations regarding the AI players. The AI players will act as users who have their own portfolios. They will need to access historical data as well as technical indicators to make informed decisions on what equity to trade and when. This class will update the league when the AI player is created, keep track of all trades executed by the AI and update their corresponding portfolios accordingly.

Operations:

- updateAI(leagueName: String): Object
  Updates AI settings and information for the league.

- getPortfolioStatus(username: String, leagueName: String): Object
  Returns portfolio information such as cash available for trading, portfolio ID etc.

- updatePortfolio(portfolioID: Integer): Object
  Updates AI portfolio for every order executed: stores order history and updates current positions

- getStockData(equityID: Integer, ticker: String): Object
  Gets all relevant data on specific tickers.

- calculateTrades(portfolioID: Integer, metrics: Object, historicalPrices: Object, techIndictors: Object): Object
  Analyzes best time and price to execute a trade based on algorithms.

10.2.2 CD-2: Authenticator

This class is responsible for operations regarding user security. It checks all users credentials and compares the stored hashed passwords with the user inputted password to validate the user. An authentication key is returned which allows the user access to their account.

Operations:

- compare_hashed_pass(password1: String, password2: String): Boolean
  Determine if the inputted password is the same as the stored hashed password.

+ check_credentials(username: String, password: String): Object
  Check if the inputted credentials are a valid user, return an authentication key if valid.

10.2.3 CD-3: CacheUpdater

The CacheUpdater class is responsible for ensuring that the database is always up to date. If a request for data that has not been recently cached is made, the CacheUpdater retrieves updated data from a third-party data endpoint.

Attributes:

- lastUpdated: Date
  The date and time the cache last updated information stored in our database.

Operations:

+ updateEquity(equityID: Integer, ticker: String): Object
  Updates all available data on specific ticker

+ updateNews(equityID: Integer, ticker: String): Object
  Updates and replaces news articles on ticker

+ updateTooltip(tooltipID: Integer): Object
  Updates and replaces definition on tooltip

10.2.4 CD-4: Controller

This class is the heart of the back-end and serves as the connection between the front-end and the services. The controller's purpose is to get the user's request and send it to the appropriate handler. The controller will then send the handler's response back to the front-end.

Attributes:

- authToken: Object
  Token used by system to authenticate user identity.

Operations:

+ login(username: String, password: String): Boolean
  Allow users to login in using a valid username and password.

+ register(username: String, email: String, password: String): Boolean
  Allow new users to register a new account with a username, email, and password.

+ placeOrder(portfolioID: Integer, ticker: String, volume: Integer, orderOptions: Object): Boolean
  Create a trade order for a particular portfolio.

+ sendLeagueRequest(leagueID: Integer, requestBody: Object): Boolean
  Handle any given operation regarding leagues and league management.

+ viewEquity(equityID: Integer): Object
  Get equity information to display to the user.

+ viewNews(equityID: Integer): Object
  Get news to display to the user.

+ viewTooltip(tooltipID: Integer): Object
  Get the definition of a term to display on the tooltip.

+ viewPortfolio(portfolioID: Integer): Object
  Get portfolio information to display to user

- calculatePortfolio(portfolio ID: Integer, stockData: Equity): Object
  Calculates the current value of the portfolio to display to the use

10.2.5 CD-5: DatabaseManager

The DatabaseManager class serves as an interface between the classes that require data to operate and the database. This class is capable of querying and creating new entries in the database for any relevant information.

Operations:

+ queryOrder(orderID: Integer): Object
  Retrieve an order that was placed.

+ createOrder(portfolioID: Integer, orderOptions: Object): Boolean
  Create an order entry for a given portfolio with the given options.

+ queryUser(userID: Integer): Object
  Retrieve a user's information.

+ createUser(username: String, email: String, password: String): Boolean
  Create a user with the given credentials

+ queryLeague(leagueID: Integer): Object
  Retrieve information about a specified league.

+ createLeague(leagueName: String, leagueManager: User, leagueOptions: Object): Boolean
  Create a league with the name and options specified by the league manager.

+ queryPortfolio(portfolioID: Integer): Object
  Retrieve a portfolio, with its current holdings and value.

+ updatePortfolio(portfolioID: Integer, updatedValues: Object): Boolean
  Update a portfolio with the specified changes.

10.2.6 CD-6: EntityRetriever

This class is responsible for the retrieval of current data on equities from the API and ensures all out of date information is replaced and updated when the data is needed.

Operations:

+ getNews(equityID: Integer): Object
  Gets news articles on specific equities

+ getStockQuote(equityID: Integer, ticker: String) : Object
  Gets all relevant data on specific ticker

+ getPortfolioData(portfolioID: Integer) : Object
  Gets user portfolio data, including positions, portfolio value, etc.

+ getTooltip(tooltipID: Integer): Object
  Gets term definition to display on the tooltip

10.2.7 CD-7: Equity

This class is used as a container for all information regarding equity. The ticker symbol, name of the company, and price data is stored. Additionally, more specific research information such as market cap and P/E ratio is made available.

Attributes:

+ equityID: Integer
  The unique identifier for this equity.

- equityName: String
  The name of the company associated with this equity.

- tickerSymbol: String
  The ticker symbol of the equity.

- currentPrice: Double
  The current value of the equity.

- previousClose: Double
  The equity's final price on the preceding trading day.

- openPrice: Double
  The equity's price at market open.

- metrics: Object
  A list of financial statistics pertaining to equity such as P/E ratio, volume, market cap etc.

- techIndicators: Object
  A list of technical indicators and their values over different time periods.

- History: Object
  A list of all historical price data for different time periods.

Operations:

+ getEquity(): Object
  Get the equity

+ setEquity(Equity: Object): Boolean
  Set the equity

10.2.8 CD-8: League

This class is used as a container for all information regarding a league. All league settings specified by the league manager such as the time period of the league, the limit of trades per day, etc will be stored here. Also, you can find the list of members participating in the league and the password to join a league here.

Attributes:

+ leagueID: Integer
  The unique identifier for this league.

- leagueName: String
  The name of the league.

- playerList: Object
  A list that enumerates the players participating in this league.

- aiPlayers: Object
  The list of AI players that are participating in this league.

- startDate: Date
  The date when this league was created.

- endDate: Date
  The date when this league ends.

- maxPlayers: Integer

The maximum number of players allowed to participate in this league.

- public: Boolean
  True if any player can join the league, false otherwise.

- commissionPercent: Double
  The "brokerage fees" defined for this league.

- tradeLimit: Integer
  The maximum amount of money a user is allowed to use trading in a single day.

- leagueKey: String
  The password for the league, if provided.

Operations:

+ getLeague(): Object
  Get the league

+ setLeague(League: Object): Boolean
  Set the league

10.2.9 CD-9: LeagueHandler

This class is responsible for all operations regarding leagues, including the creation of leagues, joining leagues, and managing leagues. Portfolios are closely tied to the league they are a part of. Hence, this class also shares some responsibilities for ensuring that the portfolios are up to date.

Operations:

+ createLeague(leagueName: String, startingBalance: Double, commissionPercentage: Double, tradeLimit: Integer, startDate: Date, endDate: Date, username: String): Boolean
  Creates new league based on league manager's desired preferences

+ joinLeague(leagueName: String, leagueKey: String): Boolean
  Adds user to league if the passcode they entered was correct

+ getLeagueInfo(leagueID: Integer): Object
  Returns league information such as name, list of players, start & end data, trade limit etc.

+ addAI(leagueName: String): Boolean

Creates and adds AI players to league

- updatePortfolios(username: String, leagueName: String): Boolean
  Updates user portfolio for every order executed: stores order history and updates current positions

10.2.10 CD-10: News

This class is used to store news related to a company whose stock is traded. This news is presented to users in multiple locations in the application to help them make informed trading decisions.

Attributes:

- Headline: String
  Title of news article

- Image: String
  Link to image associated with news article

- Date: DateTime
  Date and time news article was published

- URL: String
  Link to news article webpage

Operations:

+ getNews(): Object
  Get the news

+ setNews(News: Object): Boolean
  Set the news

10.2.11 CD-11: Order

This class represents an order that was placed by a user. The portfolio of which this order was placed for is specified, along with the price and volume of the order.

Attributes:

+ orderID: Integer
  A unique identifier for this order.

- portfolioID: Integer

The portfolio of which this order is placed for.

- timePlace: Date
  The time that the order was placed.

- equityName: String
  The name of the equity that the order is for.

- quantity: Integer
  The number of shares for this order.

- limitPrice: Double
  The price at which to execute this order.

- hasExecuted: Boolean
  True when the order has been executed, false otherwise.

Operations:

+ getOrder(): Object
  Get the order

+ setOrder(Order: Object): Boolean
  Set the order

10.2.12 CD-12: Page

This class represents a page that is displayed to the user. Each page serves a specific role and allows the user to interface with all of the available operations.

Attributes:

+ Form: Object
  This is an HTML object that allows the user to enter information. Each form is linked to a specific API request.

Operations:

+ getUserInput(): JSON
  This function takes the form data and returns the data in a processable form.

10.2.13 CD-13: Portfolio

This class is used as a container for information regarding a user's portfolio for each league that they are participating in. It will keep track of their current worth, their current positions, as well as their order history.

Attributes:

+ portfolioID: Integer
  The unique identifier for this portfolio.

- Username: String
  The username of the user for which this portfolio belongs to.

- leagueID: Integer
  The league for which this portfolio is a part of.

- cashAvailable: Double
  The amount of cash available for trading.

- netWorth: Double
  The current total value of all cash and equities held in this portfolio.

- currentHoldings: Object
  A list of positions for this portfolio, with equity names and volume.

- orderHistory: Object
  A list of past transactions made in this portfolio.

Operations:

+ getPortfolio(): Object
  Get the portfolio

+ setPortfolio(Portfolio: Object): Boolean
  Set the portfolio

10.2.14 CD-14: TradingHandler

This class is responsible for all operations regarding trading in the game. All users, including AI players, will utilize the trading handler to place orders on equities. The trading handler will process these orders and execute them when the conditions are met.

Operations:

- getPortfolioStatus(username: String, leagueName: String): Object
  Returns portfolio information such as cash available for trading, portfolio ID etc.

- executeOrder(portfolioID: Integer, orderID: Integer, equityName: String, quantity: Integer, price: Double, timePlaced: DateTime): Boolean
  Executes order based on user's specifications

+ placeOrder(portfolioID: Integer, equityName: String, quantity: Integer, limitPrice: Double): Boolean
  Places order based on user's specifications

+ cancelOrder(orderID: Integer): Boolean
  Cancels order that user has placed

+ getStockQuote(tickerSymbol: String): Object
  Retrieves current stock price

- checkTrade(portfolioID: Integer, orderID: Integer, currentPrice: Double): Boolean
  Verifies if user has enough funds in their account to execute trade and makes sure current stock price matches limit price, if applicable

10.2.15 CD-15: User

This class is responsible for storing user information relevant for authentication and identification. All users access to their leagues and portfolio information is tied to data stored in this class.

Attributes:

+ userName: String
  The username of the user.

- email: String
  The email provided by the user upon registration.

- password: String
  An encrypted version of the user's password.

Operations:

+ getUser(): Object
  Get the user

+ setUser(User: Object): Boolean
  Set the user

10.2.16 CD-16: UserInput

This class is designed to take the user's input from the front-end client to create a request. This request is then sent to the proper controller in the backend to fulfill the request. This class then displays the server's response to the user.

Attributes:

+ loginPage: Page
  This is the page where the user can login to their account.

+ registrationPage: Page
  This is the page where a new player can register their account.

+ homePage: Page
  This is the page where a user can see their profile.

+ generalSettingsPage: Page
  This is the page where a user can edit the settings of the website.

+ symbolLookupPage: Page
  This is the page where a user can search for equities (leads to a stock page)

+ stockPage: Page
  This is the page where a user can view equity information.

+ portfolioPage: Page
  This is the page where a user can view their portfolio and holdings for a specific league

+ tradePage: Page
  This is the page where a user can make buy, sell, or limit requests to change their portfolio for a league.

+ previewOrderPage: Page
  This is the page where a user can review their order before the changes take place in their portfolio.

+ currentLeaguesPage: Page
  This is the page where a user can see all of the leagues that they are currently active in.

+ createLeaguePage: Page
  This is the page where a league manager can create a new league with the settings that they desire.

+ leaguePage: Page
  This is the page where a user can view the current status of a league, the rankings, and navigate to any of the portfolios within the league.

+ transactionHistoryPage: Page
  This is the page where a user can view all the transactions (buy, sell, limit) they have made for their portfolio while the league has been active.

+ newsPage: Page
  This is the page where a user can view all relevant news to their portfolio. The news is displayed based on their top holdings.

Operations:

+ sendRequest(JSON input): JSON
  This function grabs the user's request and turns it into an API call to the backend. After receiving the response, the function will display the updated information to the user.

+ displayResponse(JSON response): void
  This function directly edits the corresponding page to display the server's response to the user.

10.2.17 CD-17: Tooltip

This class is used as a container for information regarding terms that may be unfamiliar to inexperienced users. It keeps track of a word and its definition, and it is considered its own entity because the tooltips are not manually generated or placed on a page.

Attributes:

+ word: String
  The term that needs to be defined for an inexperienced investor

+ definition: String
  The API-requested meaning of the word that needs to be defined

Operations:

+ getTooltip(): Object

Get the tooltip entity from database

\+ setTooltip(Tooltip: Object): Boolean

Set the tooltip using a manually-made list of terms and API-requested definitions

10.3 Class Diagram Traceability Matrix

Putting the headers in this section would have made reading the matrix very difficult so instead we assigned each class name a number as referenced in section 4.2. Additionally, the only class name changes that occurred were for EntityRetriever and Page. The change for EntityRetriever is that we changed the name to be more general so that it could encompass all the different retrievers that are mentioned in the concept definition section in 2.1.1. The change for Page is that it encompasses all the different interfaces mentioned in the concept definition section. There were also some minor changes to names such as Stock Quote now falls under the class Equity instead of both having the same name. Similarly Database Connection falls under the class DatabaseManager. Portfolio Status falls under the class, Portfolio; Transaction falls under the class, Order; Trade Handler falls under the class, TradingHandler; League Info falls under the class, League.

Table 10.1. Traceability Matrix

	CD-1	CD-2	CD-3	CD-4	CD-5	CD-6	CD-7	CD-8	CD-9	CD-10	CD-11	CD-12	CD-13	CD-14	CD-15	CD-16	CD-17
Controller (UC 1-11)				X													
Database Connection					X												
Login Interface													X				
Registration Interface													X				
Investor View Interface													X				
League Handler									X								
Trade Handler														X			
Stock Data Retriever						X											
News Data Retriever						X											
League Data Retriever						X											

Portfolio Data Retriever						X											
Tooltip Retriever						X											
Transaction											X						
League Info								X									
Authenticator		X															
Portfolio Status												X					
Stock Quote							X										
News										X							
Tooltip																	X
AI Handler	X																
Cache Updater			X														

11 System Architecture & System Design

11.1 Identifying Subsystems

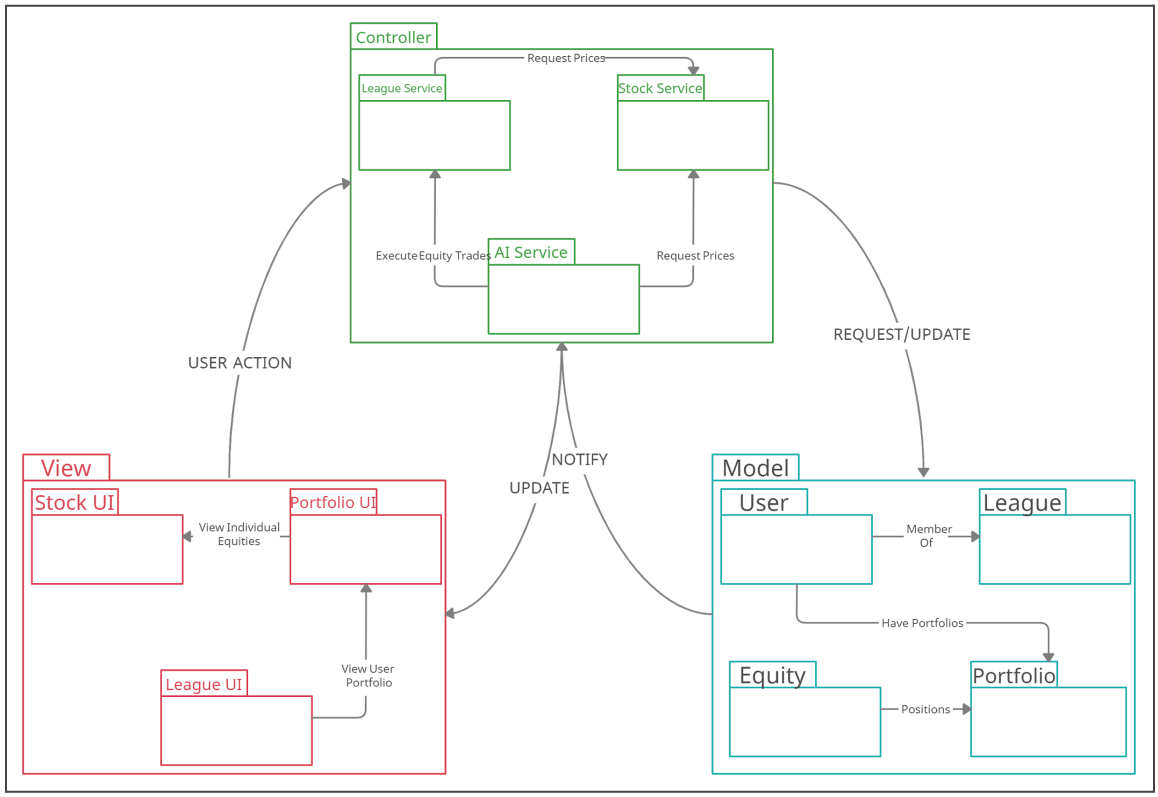

Figure 11.1. Diagram of subsystems

Our system is composed of three subsystems that follow the MVC model and is defined in greater detail in *9.2 Architecture Styles*:

The controller subsystem is comprised of the league service, the stock service, and the AI service. These three services handle all of the business logic within the application. The league service requests for prices and news from the stock service in order to handle trading. The AI service requests for prices and technical indicators from the stock service and utilizes the league service to execute orders.

The view subsystem consists of the stock UI, the portfolio UI, and the league UI. This subsystem provides users with methods that allow them to interact with the system. The league UI is the root, from where users can access their individual league portfolio through the portfolio UI. Users are able to view individual equities through the stock UI, which can be accessed from the portfolio UI.

The final subsystem is the model subsystem, which consists of users, leagues, and equities. This subsystem defines the types of data that we are storing. League models contain information on users participating in the league and their desired settings. Equity models contain historical price data and other relevant financial statistics such as technical indicators. User models keep track of account information and holdings in each league.

Utilizing this MVC architecture along with microservices results in a modular design that allows us to develop components independently of one another.

11.2 Architectural Styles

For our application, we are making use of a Model-View-Controller (MVC) architecture model as outlined above, which has become a popular design pattern for developing web applications. The view component represents the front end of the application and handles all of the application's UI logic. It presents to the user all the data received from the controller. The view component is responsible for displaying all the graphs, statistics, and metrics to the user. The model directly interacts with data from the database and handles all the business logic of the application and responds to requests from the controller. The controller component processes business logic and receives incoming requests from the user. The controller handles user interaction and input and acts as an interface between the model and the view.

Additionally, we will be incorporating a Client-Server model, since many users will want to access our application at once. The user/player will be the main client who interacts with our service, and the server will provide and store all the financial data that the users need upon request. This promotes seamless interaction between the client and infrastructure.

Lastly, we will be using a microservice architecture. Our plan is to create separate services for league logic and financial logic i.e. there will be a league service and a stock service. This allows us to develop these components of the application separately and independently of each other.

11.3 Mapping Subsystems to Hardware

The "View" subsystem gets mapped directly to the user's device. Our client will be intended for users on either smartphones or PC. Users will be able to use any device with a web browser to access our web application, but only resolutions that are commonly found on mobile or desktop computers will be supported, as mentioned later in Hardware Requirements (7.6). The user interface will be presented through web pages that are managed with React.

The “Controller” and “Model” subsystems are run by a server. This server will be run on a remote cloud-based machine and will consist of multiple microservices that will communicate with each other through REST APIs. Each microservice (Controllers) will be built in Node.js and will utilize Express for creating routes and building endpoints. The Controllers can also utilize said routes to communicate with MongoDB (Model) for persistent data storage.

11.4 Connectors and Network Protocols

Since the application we are building is a web application, we are using HTTP/HTTPS. This is the standard protocol for anything that is accessed through a web browser, which is why we chose to use it. Additionally, it allows us to use REST APIs to update and retrieve data. This is also another industry standard when it comes to web applications.

11.5 Global Control Flow

11.5.1 Order of Execution

Our system is almost entirely event-driven, as it mainly acts on requests made by the user to the server. Many of the features our system accommodates for are prompted by either the user or another part of the system such as the trading bot. When a user wants to place an order, view their portfolio, or search equity information, they must request the server which will take the necessary steps to facilitate the request. The only features that would be considered procedure-driven are registering and trading. Before any user can access other parts of our application, they must make an account and before they can make a trade, they must be part of a league. However, after making an account, they can trade any stock available to them at any time they want, which means that the system has to wait for the event to occur before it responds to it.

11.5.2 Time Dependency

Our system is also time dependent. The stock market itself only operates during a certain part of the day, so our system depends on this period (9:30AM – 4:00PM) to execute orders. Users can place orders before market open or even after market close, however, orders will only be fulfilled during regular trading hours. Additionally, our system supports real time transactions, the latest stock quotes and portfolio value, limit orders and our trading bot, all of which are heavily reliant on real-time timers. The limit orders and the AI trading bot will access the stock data at a period that will be specified later. Optimally, they will access the

stock data as frequently as possible. This period is constrained by the number of API calls that the stock service can support.

11.6 Hardware Requirements

Requirements for web application client:

- 5 Mbps network connection
- Minimum resolution: 1024 x 768
- Minimum requirements for a web browser compatible with ES6 Javascript

12 Algorithms and Data Structures

12.1 Algorithms

When a league is created, the AI bots are created and stored as users into the database. There is a boolean variable identifying whether the user is an AI bot. All the algorithms used in the application are outlined in sections 12.1.1 - 12.1.3. Each bot is also assigned an algorithm that it will use in the specified league. There will be no other bot in that league employing the same algorithm for making trades. Each AI player has its own portfolio and starts from scratch (like all other members in the league).

Every two minutes during market hours, the AI bots will check to make trades. There is a predetermined list of ticker symbols in which the AI bots will conduct their research. For every equity in the list, each of the three algorithms will determine a ‘Buy’, ‘Hold’ or ‘Sell’ rating for that specific equity. This is based on the technical indicators described in detail in section 8.4.

After the ratings have been calculated for each equity, each bot goes through its portfolio and places trades. For example, if there is a ‘Sell’ rating on a particular equity, the bot will check to see if it has a position in that stock and if so, it will sell its shares. If there is a ‘Buy’ rating, the bot will invest a certain fraction of their portfolio into the equity, and if there is a ‘Hold’ rating, it must not be a good time to place a trade so no action occurs.

12.1.1 Mean Reversion

Mean reversion is a mathematical method of trading implemented by many other successful quantitative hedge funds and value investors. It is rooted in the belief that the prices will eventually revert to the mean of the historic data and even out over time [7]. This can be applied to the strategy of ‘buying low and selling high’

[6]. When implementing this strategy, investors are attracted to prices that stray away from historical averages. For example, if the price of an equity dropped significantly due to the resignation of a CEO, investors would buy into the equity with the beliefs that the stock will return to its historical average in time.

There are many ways in which investors apply mean reversion. We plan to make use of technical indicators such as the Bollinger Bands, which encompass both the Simple Moving Average (SMA) and standard deviations. These indicators act as good levels to enter and exit trades.

12.1.2 Momentum Trading

Momentum trading is an algorithmic strategy utilized by investors to take advantage of an upwards or downwards trend in an equity's price. It is rooted in the belief that trends that head in one direction will continue to head in that one direction because of the momentum already behind them. When implementing this strategy, investors buy into equities that are rising and sell them when they believe they have peaked [8]. The main idea behind this is to find opportunities with short-term uptrends and sell out when the equity starts to lose momentum and volume, then repeat the process [9]. This strategy is heavily reliant on secure entry and exit points, as well as proper risk management.

There are many ways in which investors apply the momentum trading strategy. We plan to make use of technical indicators such as the Relative Strength Index (RSI) and the Moving Average Convergence Divergence (MACD) [11]. These indicators help us identify which of these spikes in prices are real and supported, while also helping us stay away from reversals.

12.1.3 Japanese Candlesticks

Japanese candlesticks are technical analysis charts that are used by investors to analyze the price movements of equities. They provide a graphical representation of the activity of the particular equity in terms of demand and supply over a set period of time. The candlesticks each have a body, which shows the distance between the open and close prices of the particular equity. They also keep track of the lows and highs of each time period. The patterns made by the candlesticks provide a visual perspective for predicting future market trends which help investors more accurately plan their entries and exits for profit maximization and loss reduction [12].

In our current implementation, we have decided to choose a set number of bullish and bearish candlestick patterns. Some of the patterns we identify include the engulfing, hammer & shooting star, and morning & evening star patterns. Each

pattern can give us more insight about the market and can help us realize market reversals.

We have a candle score in which we keep a cumulative sum of the signals we get whenever we recognize a bullish or bearish pattern based on the open, high, low, and close prices. If there is a bullish signal detected we add 1 to the sum, and if there is a bearish signal we subtract 1. If the sum of all signals is greater than 0 then it is a buy, if it is less than 0 then it is a sell, and if it is equal to 0 there is no action taken [10].

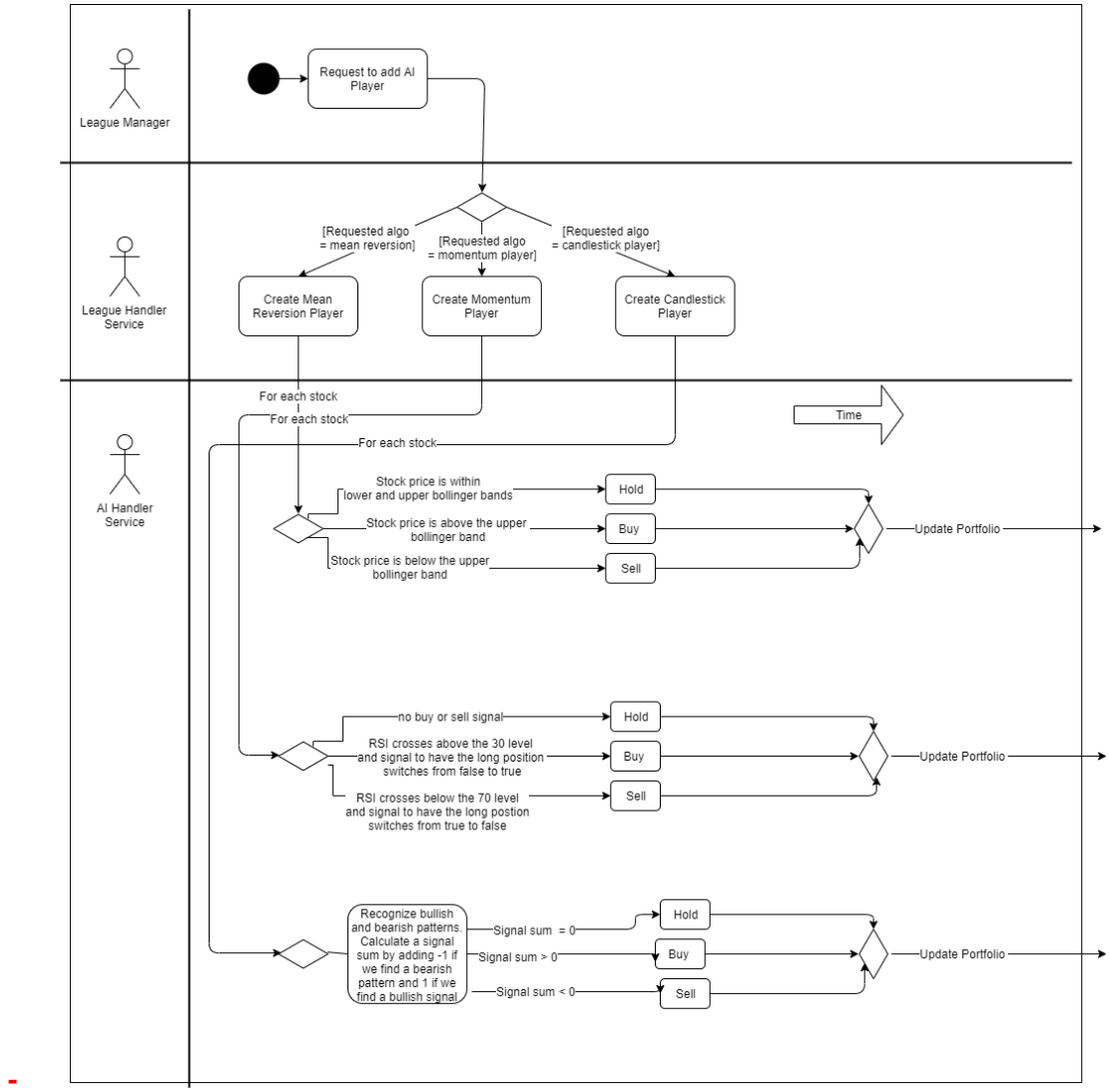

Figure 12.1. Diagram for the implementation of AI Algorithms

12.2 Data Structures

Our system is written almost entirely in Javascript and makes use of Javascript objects to handle most of our data structure requirements. Javascript objects behave similarly to

hash tables with key-value pairs but are implemented differently in each Javascript engine. This structure is both highly flexible and performant. Every entity in our system can be stored as an object with easily legible and accessible fields.

The only shortcoming of Javascript objects is that they are not directly iterable. To handle these cases, we used arrays. Like objects, arrays in Javascript are also very flexible, since they are similar to lists or arraylists in other languages and can dynamically grow. The combination of these two data structures encompassed all of our required use cases.

12.3 Concurrency

Since we are running a Node application, everything runs on a single thread. Despite this, multiple users are able to be served at the same time since the requests are non-blocking. The user will be served when in the next iteration of the event loop.

13 User Interface Design and Implementation

13.1 Page Design Updates

Below are many of the updates that we included in our final demo presentation. We added new pages to emphasize our business value and also updated past pages to make it more user friendly.

13.1.1 Centralized League Page

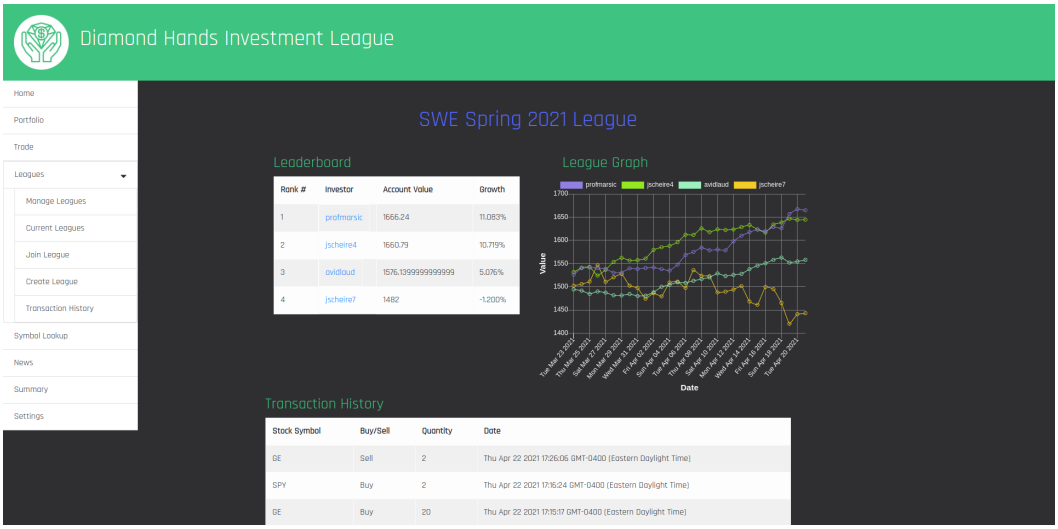

Figure 13.1. Centralized League Page

A new implementation made to the application is the Centralized League Page. To get to this page the user clicks on a specific league on the Current League Page and then the information about the pertaining league will be displayed. The information is displayed on a Leaderboard where the player can see what their rank is in the league. Players can also track their progress against their league mates as well as see their transaction history on the bottom of the page.

13.1.2 Summary Page

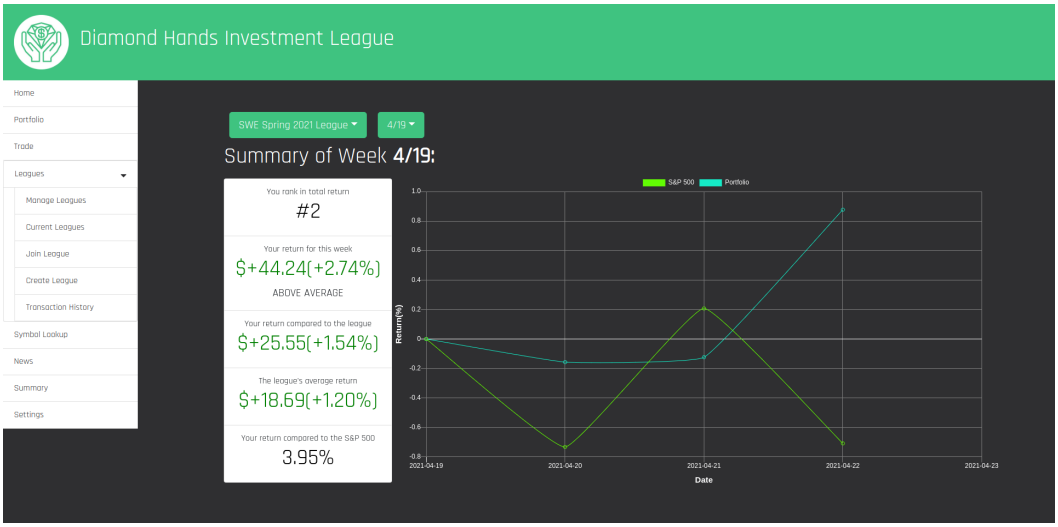

Figure 13.2. Summary Page

Based on feedback given to us during demo one, our group decided to implement a summary page. This page notifies users every week on their performance relative to other members in the league. Users can have the option of looking through previous weeks to see how they have improved. Additionally, users can easily see their performance ranked in comparison to S&P 500. The Summary page can be easily accessed by clicking on Summary in the navigation bar.

13.1.3 Tooltips

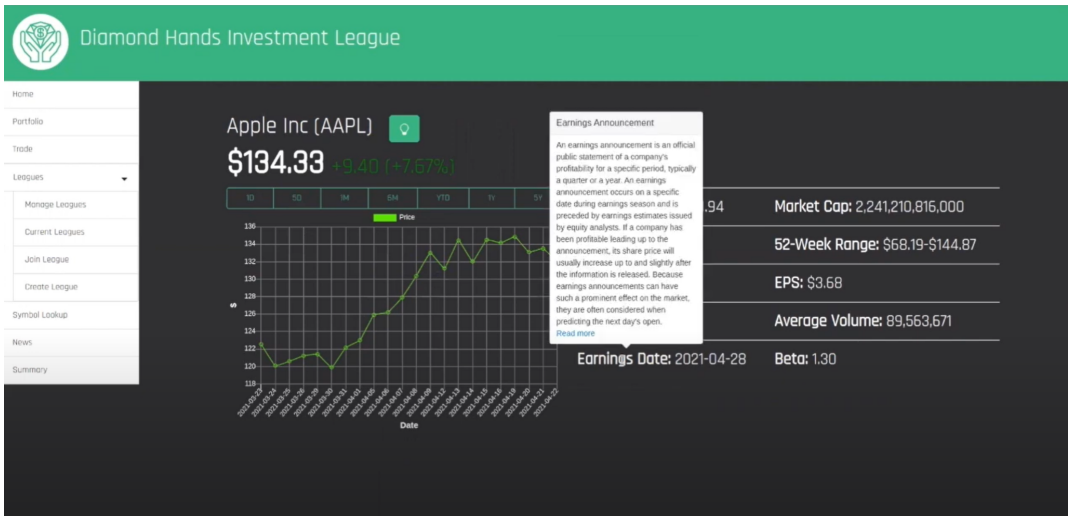

Figure 13.3. Tooltips

Tooltips was another feature that we were able to implement in our project. The tooltips can be seen by hovering over specific keywords. For example, when the user brings their cursor to the word Earnings Date, a small pop up will appear with a description and a link to more information regarding the hovered word. All the definitions are taken from Investopedia articles that correspond to that particular subject.

13.1.4 Tutorial

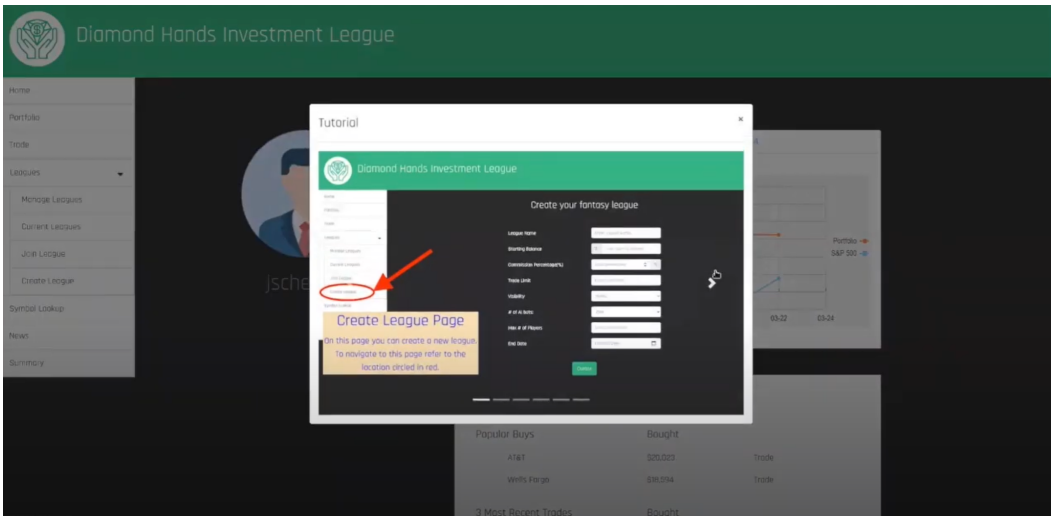

Figure 13.4. Tutorial

A new feature we added are the tutorial slides. These slides will show up in the beginning when the user first logs in to the application. The series of pictures in the slides show the user how to navigate through our application as well as provide disclaimers to the user.

13.1.5 News

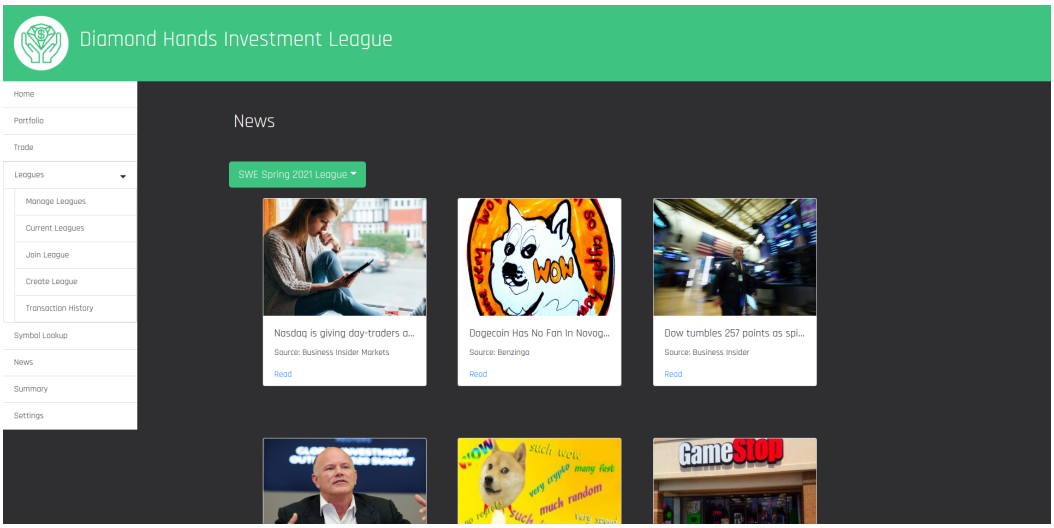

Figure 13.5. News Page

Another feature that we were able to add was the curated News Page. This page was created to give users the most updated news pertaining to the stocks the user purchased. Each one of the users league will produce different news articles relating to their transactions. In the above image, there are articles regarding GameStop because the user made some sort of transaction with game stop in the “SWE Spring 2021 League”. All of these articles are pulled from the IEX cloud API.

13.1.6 AI Bot

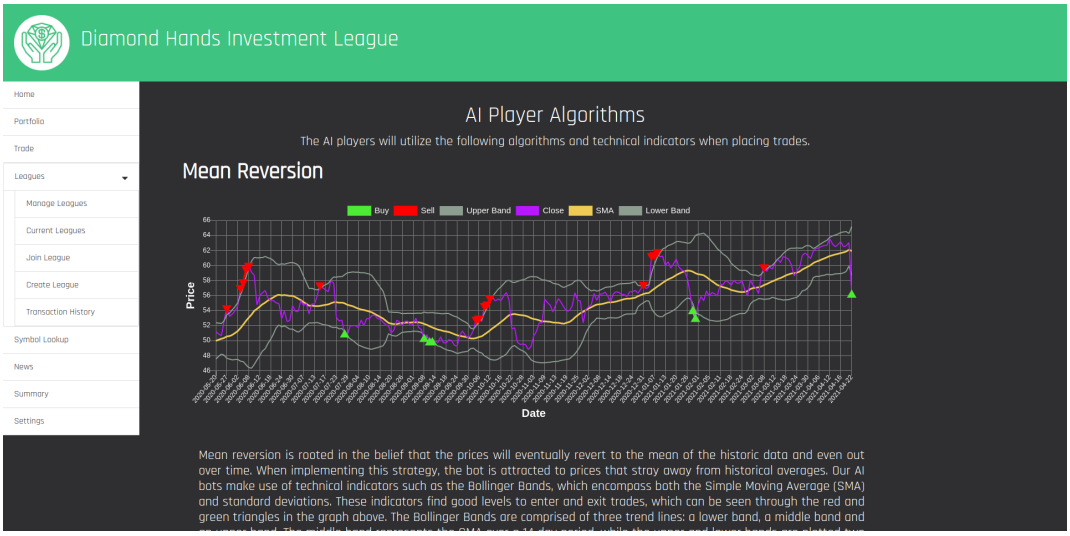

Figure 13.6. Algorithms Page for AI Bots

A newly added feature are the AI Bots. To see an explanation of how the bots work and the algorithms that are used we can look to the AI bot page. To navigate to this page the user can click on the light bulb on the symbol lookup page. On this page the user will be able to see an interactive graph. They can see where our bot would have traded the equity based on the current price and the buy & sell signals indicated in green and red, respectively. Under each graph, we have also included a concise description of the algorithm used and we also have a link to Investopedia where the user can obtain a stronger understanding of the algorithms.

13.1.7 Join League Page

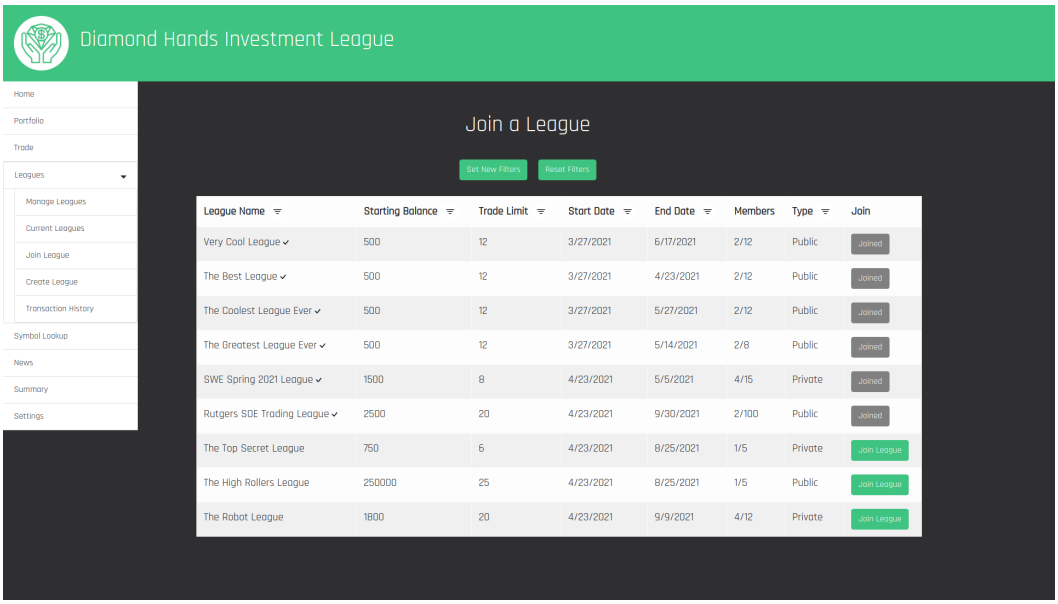

Figure 13.7. Join League Page

A new feature we added to the Join League page were filters. This is a way for users to find leagues easier and more efficiently. Some examples of the filters are league name, trade limit range, start and end dates of the league. We also show which leagues the user has already joined so users are able to clearly see what other leagues they can join. Additionally, we have also included the ability to sort columns in ascending or descending order. This is indicated by the filter sign that is in the following columns: “League Name”, “Starting Balance”, “Trade Limit”, “Start Date”, “End Date” & “Type”.

13.2 Page Efficiency

The original plan of the project was to use HTML and CSS to create the pages, as they were introductory languages that members of the team, unfamiliar with coding, could easily learn on their own time. We quickly realized that this would not be an efficient way to create the application and instead began focusing on familiarizing ourselves with React Bootstrap components. This would allow us to have a strong foundation for the user interface and would nicely package the frontend and backend components of our project.

In a similar fashion, we decided to use SCSS because this would reduce repetition in code and allow the team to import different design rules that we wanted to standardize for the application. One example of the design rules would be the overall theme of our application. We went forward with using a dark theme, so by setting up those rules in a

root SCSS file, we can ensure that every page has the same header, page color and display rules.

We also ensured that we can guarantee the user an enjoyable viewing experience, regardless of the device that they are using. Regardless, whether the user was using a laptop, desktop or phone the user would be able to view our application. We do recommend, however, that the user use either a laptop or desktop for a more pleasurable experience. We have taken great care in checking the look of each page on all the different devices that members in our group use. It has been difficult, but by trying to maintain this type of consistency early on, we hope to give ourselves time to focus on some of the more difficult implementation aspects of the application.

14 Design of Tests

14.1 Use Case Coverage

Table 14.1. Test Cases for each Use Case

Use Case Number	Test Case Number
UC-1	TC-2
UC-2	TC-5, TC-6, TC-7
UC-3	TC-7
UC-4	TC-5, TC-6
UC-5	TC-1, TC-7
UC-6	TC-7
UC-7	TC-1, TC-7
UC-8	TC-1, TC-7
UC-9	TC-1, TC-4, TC-7
UC-10	TC-1, TC-3, TC-7
UC-11	TC-8

14.2 Testing Use Cases

14.2.1 View Equity Information

Table 14.2.1 View Equity Information Test Cases		
Test Case Identifier:	TC-1	
Use Case Tested:	UC-5, UC-7, UC-8, UC-9, UC-10	
Pass/fail Criteria:	This integration test passes if the user is able to access all equity data, including prices, metrics, and news	
Input Data:	Equity ticker	
Test Procedure:		**Expected Result:**
1. Request equity prices from “FAKE” (an invalid ticker)		System returns error message of “invalid ticker”
2. Request equity prices from “TWTR” (a valid ticker)		System gathers recently cached prices or retrieves new prices and caches them; System displays prices to user
3. Request equity metrics from “FAKE” (an invalid ticker)		System returns error message of “invalid ticker”
4. Request equity prices from “TWTR” (a valid ticker)		System gathers recently cached metrics or retrieves new metrics and caches them; System displays metrics to user
5. Request equity news from “FAKE” (an invalid ticker)		System returns error message of “invalid ticker”
6. Request equity news from “TWTR” (a valid ticker)		System gathers recently cached news or retrieves new news and caches them; System displays news to user

14.2.2 Login and Registration

Table 14.2.2 Login/Registration Test Cases	
Test Case Identifier:	TC-2
Use Case Tested:	UC-1

Pass/fail Criteria:	The test passes if the user enters a valid username along with the correct associated password and/or creates an account with the appropriate credentials
Input Data:	Username, password

Test Procedure:	**Expected Result:**
1. Logging in with [jscheire3, monkey3] (valid username and incorrect password)	System returns error message of "invalid username or password" and is prompted to try again
2. Logging in with [marsic, password] (invalid username and some password)	System returns error message of "invalid username or password" and is prompted to try again
3. Registering with [ab, password123] (username length is less than 3)	System returns error message of "username must be minimum 3 characters" and is prompted to try again
4. Registering with [marsic, BinHu] (password length is less than 6)	System returns error message of "password must be minimum 6 characters" and is prompted to try again
5. Registering with [jscheire3, otherpassword] (a username that is already taken)	System returns error message "username already in use" and is prompted to try again
6. Logging in with [jscheire3, password123] (valid username with correct password)	System returns success message: {username: jschiere3, token: <JWT token>} The user is redirected to the home page

14.2.3 Trading

Table 14.2.3 Trading Page Test Cases

Test Case Identifier:	TC-3
Use Case Tested:	UC-10
Pass/fail Criteria:	This case tests whether the user is able to buy or sell a particular equity. The test passes if the transaction is executed and the user is able to view this purchase

	in their portfolio.
Input Data:	League Name, Equity Symbol, Transaction Type, Quantity and Price

Test Procedure:	**Expected Result:**
1. Do not select a league name from the dropdown menu	System returns an error message of “Cannot preview order without league name” and is prompted to try again
2. Do not enter the equity name.	System returns an error message of “Cannot preview order without existing stock symbol” and is prompted to try again
3. Do not select the transaction type from the dropdown menu	System returns an error message of “Cannot preview order without transaction type” and is prompted to try again
4. Do not type in the quantity.	System returns an error message of “Cannot preview order without a valid quantity” and is prompted to try again.
5. Do not select the price type	System returns an error message of “Cannot preview order without selecting a price type” and is prompted to try again.
6. Select “limit order” for the order type and do not specify a price.	System returns an error message of “Cannot preview order without a limit price or a stop stop price” and is prompted to try again.
7. “The Greatest League Ever” is selected, select “Buy” type, select quantity as 1, select symbol as “GME”, and select “GTC”.	System displays a success message “Order successfully placed!”

14.2.4 Stock Symbol Lookup

Table 14.2.4 Stock Symbol Lookup Test Cases	
Test Case Identifier:	TC-4
Use Case Tested:	UC- 9

Pass/fail Criteria:	This case tests whether the user is able to look-up information on a particular equity. The test passes if the information is displayed.
Input Data:	Equity Symbol

Test Procedure:	**Expected Result:**
1. Type in stock symbol “FAKE” (an invalid ticker)	System returns error message of “invalid stock symbol” and is prompted to try again
2. Type in stock symbol “TWTR” (a valid ticker)	System redirects user to “Stock Research” page of queried stock symbol

14.2.5 Create League

Table 14.2.5 League Creation Test Cases

Test Case Identifier:	TC-5
Use Case Tested:	UC-2, UC-4
Pass/fail Criteria:	This test passes if the user is able to successfully create a league with the desired parameters.
Input Data:	League Name, Starting Balance, Commission Percentage, Trade/Day Limit, Private/Public, # of AI bots, Start and End Date

Test Procedure:	**Expected Result:**
1. User enters “The Greatest League Ever” (a league name) that is already in use.	System returns error message of “already taken league name, please enter a new name”
2. User tries to put 1/1/2000 (an invalid start or end date) as a start or end date.	System returns error message of “Please provide a valid [start/end] date”
3. User tries to input a commission percentage of 200% (below 0% or above 100%)	System returns error message of “Please provide a valid comm percentage up to 2 decimal places”
4. User tries to leave “League Name” (a required input) blank.	System returns error message “Please provide a [appropriate value]”

5. User selects a league name of “My New League”, a starting balance of 20000, a start date of 5/3/2021, an end date of 12/15/2021, commission percentage of 5%, private league, and 2 AI bots (all valid inputs) and successfully hits the “Create League” button.	System stores new league information in the database, and the user becomes the league manager.
6. User tries to access create league page without proper authentication	System redirects user to login page and prompts them for authentication

14.2.6 Join League Page

Table 14.2.6 Join League Test Cases

Test Case Identifier:	TC-6
Use Case Tested:	UC-2, UC-4
Pass/fail Criteria:	This test passes if the user is able to successfully join a league with the desired parameters.
Input Data:	League Name, League Key(if applicable)

Test Procedure:	**Expected Result:**
1. User enters “CryptoLeague” which does not exist	System returns error message of “league not found”
2. User tries to join “The Private League” (a private league) without specifying a league key	System returns error message of “Invalid league key”
3. User tries to join “The Private League” (a private league) with “wrongpassword” (an incorrect league key)	System returns error message of “Invalid league key”
4. User enters “The Greatest League Ever” (a league name) that already exists	System stores user as member of the league and initializes a blank portfolio for the new member and returns a success message
5. User tries to join “The Private League”, an existing private league with a valid league key, 12345	System stores user as member of the league and initializes a blank portfolio for the new member and returns a success message

6. User tries to access join league page without proper authentication	System redirects user to login page and prompts them for authentication

14.2.7 Navigation Bar

Table 14.2.7 Navigation Bar Test Cases		
Test Case Identifier:	TC-7	
Use Case Tested:	UC-2, UC-3, UC-5, UC-6, UC-7, UC-8, UC-9, UC-10	
Pass/fail Criteria:	This case tests whether the user is able to navigate to each page within the application via the navigation bar.	
Input Data:	No input data	
Test Procedure:		**Expected Result:**
1. Logged in User clicks “Home”		This should route the user to the Home page. Their username should be displayed under their profile picture.
2. Logged in User clicks “Portfolio”		This should route the user to the Portfolio page. If the user clicks the dropdown, a list of their joined leagues should show.
3. Logged in User clicks “Trade”		This should route the user to the Trade page. If the user clicks the “League” dropdown, a list of their joined leagues should show.
4. A user clicks “Leagues”		The “League” components includes Manage Leagues, Current Leagues, Join League, and Create League. This should collapse the “League” components of the navigation bar if the components are already visible, or show these components if they are not visible.
5. Logged in User clicks “Manage League”		This should route the user to the Manage Leagues page. The user will be able to select the league to manage from a dropdown menu that includes all leagues

	they are the league manager of.
6. Logged in User clicks “Current Leagues”	This should route the user to the Current Leagues page. The user should see a table of all currently joined leagues, along with each league’s portfolio’s current value.
7. Logged in User clicks “Join League”	This should route the user to the Join Leagues page. The page should load a table of all leagues in the database
8. Logged in User clicks “Create League”	This should route the user to the Create Leagues page, where the user can see the form to create a league.
9. Logged in User clicks “Symbol Lookup”	This should route the user to the Symbol Lookup page, where the user can enter a ticker symbol to see equity data.
10. Logged in User clicks “News”	This should route the user to the News page, where they can select a league to view news relevant to their portfolio.
11. Logged in User clicks “Summary”	This should route the user to the Summary page, where the user can view their portfolio’s performance compared to a variety of metrics on a weekly basis. The user should be able to click the dropdown to select any league they have joined.
12. A user who is not logged in clicks on any of the buttons in the navigation bar (except “Leagues”)	This should route the user to the Login page, where they will see the option to log in or sign up.

14.2.8 AI Players

Table 14.2.8. AI Player Test Cases

Test Case Identifier:	TC-8
Use Case Tested:	UC-11
Pass/fail Criteria:	This case tests whether the AI players can be added to a league
Input Data:	No input data

Test Procedure:	Expected Result:
1. User navigates to create league page and selects “One”, “Two”, or “Three” for “# of AI bots field and clicks “Create”	The league will be successfully created. The appropriate number of AI players will be created by the AI service. User accounts will be created for these bots and their portfolios will be initialized in the league. The success message will be sent, which will be the league object itself (LO). LO.settings.aiPlayers will equal the number selected
2. User navigates to create league page and selects “Zero” for “# of AI bots field and clicks “Create”	No AI players will be created

15 History of Work, Current Status, and Future Work

15.1 History of Work

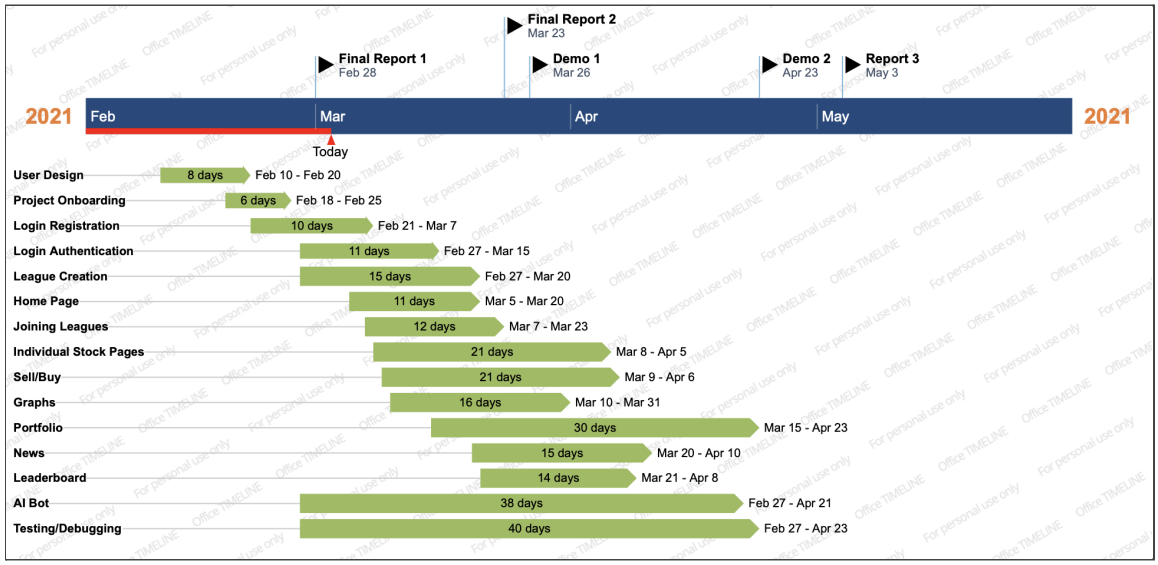

Figure 15.1. Gantt Chart for Past work

15.1.1 Use Case 1: Login

The login page has been implemented. The user can successfully login and retrieve their information. We have also incorporated authentication for the login page that makes use

of JSON Web Tokens, a popular token based authentication method. When a user logs in, a unique token will be stored in the browser and assigned to the user. When the user makes a request to the stock service or league service, they will pass said token in order to authorize these requests.

15.1.2 Use Case 2: Create League

The league creation page has been fully implemented. The user can successfully create a league to become a league manager, with the ability to set the league name, starting balance, commission percentage, trade limit, visibility of the league, maximum number of players, number of AI bots, and the ending date. Validation to prevent the user from inputting bad inputs has also been implemented for this page. There is also currently no limitation to the number of leagues that a user can create.

15.1.3 Use Case 3: Join League

The join league page has been successfully implemented. A player may join any existing league. For private leagues the player must have the league code that is provided via the player by the League Manager. For public leagues the player can choose the league and join. At this time there is no limit to the number of leagues that a player can participate in.

15.1.4 Use Case 4: Add AI Players

We have implemented the AI players into our leagues. A league manager can choose to add up to three AI players to their league. AI players will utilize selected algorithmic trading techniques, but will only be granted a limited number of actions per day.

15.1.5 Use Case 5: View Portfolio Status and Information

The view portfolio has been implemented completely. The user can choose the league for which they wish to see the stocks which have been bought. Once the user chooses their league, they can see a list of stocks with information relating to stocks quantity, current price, total value and total change.

15.1.6 Use Case 6: Inspect Transaction History

We have implemented the Transaction History on the Centralized League Page. The user can view their order history in their active leagues, as well as recently executed orders of fellow league members. They can also see transactions made by all users in their past, inactive leagues.

15.1.7 Use Case 7: View League & Rankings

We have implemented a leagues page that allows the user to be able to view all the active and past leagues that they have been a part of. When users click on a specific league they will navigate to, what we call, a Centralized League Page. This page provides all the relevant information for each participating league. It includes league rankings in the form of a table, a league graph that compares all players' portfolios and a user's transaction history for every trade made in that league.

15.1.8 Use Case 8: News

The news page has been completed. The user will be able to see curated news based on the different transactions they make. This news is presented to users to help them make informed trading decisions about the stocks they have already interacted with.

15.1.9 Use Case 9: View Equity Information

The research equity information has been completely implemented. The user can now view the equity information. This page shows the equities performance compared to S&P 500 as well as detailed information about the equity.

15.1.10 Use Case 10: Trades

We have finished working on the Trade page. The user can buy as many stocks at market price as well as use the additional features such as stop order, limit order, and day order.

15.1.11 Use Case 11: Tooltips & Summary Page

We have fully completed the development of the Summary Page. This page notifies users every week on their performance relative to other members in the league. The tooltips have been implemented as well. We hope that both of these features will provide users with helpful educational tips that will allow them to approve their performance.

15.2 Future Work

15.2.1 Machine Learning versus Algorithmic AI Players

Currently we have created algorithmic AI players for users to include and practice within leagues. However, in the future we would want to shift towards using machine learning AI players. We want users to get a well-rounded experience and although algorithmic AI players can teach users, it may not challenge them after they continually practice with these AI players. Machine learning AI Players will learn with every trade that they make and continue to challenge even our more experienced users.

15.2.2 Candlestick AI Players

In the future, we would like to supplement the candlestick algorithm with some other technical indicators. These reversal patterns should be used as an entry trigger to either buy or sell equities. We can make our judgements by following the TAE framework which stands for Trend, Area of Value, and Entry Trigger [13]. The trend criteria can be based off of the moving average and will let us know if we have a long or short bias. The area of value criteria takes into consideration the support/resistance of the market and the trendline [13]. The entry trigger will be any reversal patterns we can identify from the aforementioned candlestick patterns. Using all of the criteria together, we can make an informed decision when placing orders.

15.2.3 Achievements

Achievements was a feature that we wanted to include in our application, but after recognizing that our application lacked the necessary educational features we decided that this would be a goal we could work towards after giving our application a stronger foundation. This feature would provide users with league achievements and personal achievements. Some of the league achievements might include “best trade of the day” or “most money made this week”, while personal achievements may include “researched 50 different equities” or “logged onto the application every day this week”.

15.2.4 Derivatives Trading

Our system currently allows users to place market, limit, and stop orders on equities in the market. While this encompasses the majority of trading for most new users, we would like to also implement functionality for trading derivatives, such as futures and options. Due to the nature of derivatives trading, we would need a significant amount of educational resources to guide new traders. However, this would greatly increase our business value to users and allow us to target a market of users who are looking to try our derivatives trading.

15.2.4 Social Interaction

This was another feature that we would have liked to develop, but we did not have enough time to thoroughly focus on this feature. What we had envisioned was a chat feature that would allow players to interact with one another. There would be forums where players could ask questions and learn more from their peers. Players would be able to chat with one another in league chats and also in one-on-one chats. They would also be able to “friend” players to have easier access when attempting to add players to a league. This would be great for classroom settings because a teacher would be able to track the progress of students as a league manager and provide one-on-one help when required.

15.2.5 Interactive Tutorials Courses

To double-down on what our application offers in an educational setting we hoped to develop courses that users would have access to learn more about the stock market. They would also get the opportunity to watch videos to assess their performance using their progress history. It would be a great resource for students, inexperienced investors and even experienced investors.

15.2.6 Display Past League Information

Another feature we would have liked to develop given more time would be the ability for users to view past league information. This would allow them to view their past activities and make new decisions. The users would be able view their portfolio graph with respect to S&P 500 for the entire time they were in the league. They would also be able to see the transactions they made while they were part of the league. Users can also view the final leaderboard of that past league.

16 References

[1] J. Gobell, "The Investopedia Affluent Millennial Investing Survey," *Investopedia*, 20-Nov-2019. [Online]. Available: https://www.investopedia.com/the-investopedia-affluent-millennials-study-4769751. [Accessed: 09-Feb-2021]

[2] T. Ghilarducci, "Most Americans Don't Have A Real Stake In The Stock Market," *Forbes*, 16-Dec-2020. [Online]. Available: https://www.forbes.com/sites/teresaghilarducci/2020/08/31/most-americans-dont-have-a-real-stake-in-the-stock-market/?sh=68fc3bcd1154. [Accessed: 09-Feb-2021].

[3] S. Rose, "7 Expert Perspectives on Why Financial Literacy Is Important," *OppLoans*, 01-Sep-2020. [Online]. Available: https://www.opploans.com/oppu/articles/why-is-financial-literacy-important/#:~:text=Financial literacy is important because, a solid foundation for success.&text=Nearly half of Americans don't enough money to retire comfortably. [Accessed: 09-Feb-2021].

[4] "How Much Longer Will Social Security Last?," *AARP*, 22-Sep-2020. [Online]. Available: https://www.aarp.org/retirement/social-security/questions-answers/how-much-longer-will-social-security-be-around.html#:~:text=En español | According to the,will be depleted by 2035. [Accessed: 09-Feb-2021].

[5] *Investopedia*. [Online]. Available: https://www.investopedia.com/simulator. [Accessed: 09-Feb-2021].

[6] J. Chen, "Mean Reversion," *Investopedia,* 01-Feb-2021. [Online]. Available: https://www.investopedia.com/terms/m/meanreversion.asp#:~:text=Mean%20reversion%2C%20in%20finance%2C%20suggests,techniques%20to%20options%20pricing%20models. [Accessed: 04-Mar-2021]

[7] J. Marwood, "How To Build A Mean Reversion Trading Strategy", *Decoding Markets,* 04-April-2018. [Online]. Available: https://decodingmarkets.com/mean-reversion-trading-strategy. [Accessed: 04-Mar-2021]

[8] A. Barone, "Introduction To Momentum Trading", *Investopedia*, 15-May-2019. [Online]. Available: https://www.investopedia.com/trading/introduction-to-momentum-trading. [Accessed 04-Mar-2021]

[9] Rayner, "The Essential Guide to Momentum Trading", *TradingWithRayner*, 08-Oct-2020. [Online]. Available:

https://www.tradingwithrayner.com/the-essential-guide-to-momentum-trading. [Accessed: 04-Mar-2021]

[10] S. Guglani, "How to develop your first trading bot using python - by recognising candlestick patterns," *Medium*, 14-Aug-2019. [Online]. Available: https://sunil-guglani.medium.com/how-to-develop-your-first-trading-bot-using-python-by-recognising-candlestick-patterns-a755f7fa6674. [Accessed: 30-Apr-2021].

[11] S. Andersen, "Momentum trading with MACD and RSI - Yfinance & Python.," *Medium*, 29-Dec-2020. [Online]. Available: https://medium.com/analytics-vidhya/momentum-trading-with-macd-and-rsi-yfinance-python-e5203d2e1a8a. [Accessed: 30-Apr-2021].

[12] Rayner, "The Monster Guide to Candlestick Patterns," *TradingWithRayner*, 02-Oct-2020. [Online]. Available: https://www.tradingwithrayner.com/candlestick-patterns/. [Accessed: 04-Mar-2021].

[13] A. Hayes, "Bollinger Band®," *Investopedia*, 30-Apr-2021. [Online]. Available: https://www.investopedia.com/terms/b/bollingerbands.asp. [Accessed: 04-Mar-2021].

[14] J. Fernando, "Moving Average (MA) Definition," *Investopedia*, 29-Apr-2021. [Online]. Available: https://www.investopedia.com/terms/m/movingaverage.asp. [Accessed: 04-Mar-2021].

[15] M. Hargrave, "Standard Deviation," *Investopedia*, 20-Apr-2021. [Online]. Available: https://www.investopedia.com/terms/s/standarddeviation.asp. [Accessed: 04-Mar-2021].

[16] J. Fernando, "Relative Strength Index (RSI)," *Investopedia*, 29-Apr-2021. [Online]. Available: https://www.investopedia.com/terms/r/rsi.asp. [Accessed: 04-Mar-2021].

[17] J. Fernando, "Moving Average Convergence Divergence (MACD) Definition," *Investopedia*, 29-Apr-2021. [Online]. Available: https://www.investopedia.com/terms/m/macd.asp. [Accessed: 04-Mar-2021].