“Titan Trading: A Simulated Trading Experience Enhanced by Artificial Intelligence”

Report 3: Part 2

By

Group #7

Steven Adler

Kristene Aguinaldo

Timothy Liu

Nicholas Lurski

Avanish Mishra

Safa Shaikh

Brooks Tawil

Kristian Wu

David Zhang

Table of Contents

Project Management

All team members contributed equally to this report.

Links to Project Files

[User Documentation]
[Reports]
[Integration Tests]
[Main Source Code Repository]

Revision History

Version	Date of Revison
v.1.1	02/04/18
v.1.2	02/11/18
v.1.3	02/18/18
v.2.1	02/25/18
v.2.2	03/04/18
v.2.3	03/11/18
v.3.1	04/22/18
v.3.2	04/30/18

Summary of Changes

- Edited Customer Statement of Requirements with additional requested features by the user, including artificial intelligence and machine learning.
- Updated System Requirements and User Stories - ST (32,33,34) - to reflect the addition of artificial intelligence, AI shop, and text capability to our application.
- Removed On Screen Requirements that were irrelevant to the Titan Trading Product (OSR-3, 9) and updated requirements to reflect the addition of artificial intelligence and shop
- Updated Glossary and Actors with addition of new terms including TitanCoin, AI, Twilio text API, etc.
- Updated Use Cases(14,15,16,17)
    - Use Case Diagram
    - Fully dressed UC (14,15,16)
    - System Seq Diagram (UC14)
- Updated all of the database tables in system architecture section. Added profile table and Entity-Relationship diagram along with descriptions of tables
- Updated Domain Model diagram to include AI purchasing and competing capability
- Updated Concept Definitions and Responsibilities to reflect addition of AI and trophies: (TCoinCalculator, ShopControl, AIControl, ShopTable, AITable, InventoryTable)
    - Added Domain Concepts and responsibilities
    - Adjusted concept associations, attribute definitions, traceability matrix, and domain model accordingly
    - Updated system operation contracts from new use cases
    - Updated system sequence diagrams depending on implementation changes and to feature various design patterns
- Updated Effort Estimation Point Calculations
- Updated (user case x requirements) and (domain concept x use case) traceability matrices
- Updated Algorithms and Data Structures to include the algorithms used for medium and challenging-level artificial intelligence, which include descriptions of linear regression and convolutional neural networks
- Revised class diagram and relevant data types and operation signatures according to new extensions and domain model
- Added 9.6 Object Constraint Language Contracts with preconditions and postconditions

- Updated User Interface Implementation by adding actual website implementation and adding UC-8, UC-14, UC-15 to screen mockups.
- Updated Design of Tests section to include testing for all new features in the application. These new tests include verification of shop upgrade purchases going through and Artificial Intelligence upgrade transactions.

1 CUSTOMER STATEMENT OF REQUIREMENTS

Dear Group #7,

Here at the Rutgers Investments Co., our employees trade on the stock market everyday and have allowed our company to survive in the competitive field. With our current success, we are looking to train the next generation of traders and finance experts. We believe that the best way to learn about trading is by participating yourself. With experience, traders will know the right stocks to choose that can yield an average of above 10% interest each year. They will also understand the risks involved in trading, and how investing in the stock market during a market boom can exponentially increase your net worth while a market crash can cripple your investments. However, learning by doing is not the best way to approach this problem. Through this process, people can lose a lot of money with failed investments, and there are even those who do not have the financial capability to begin trading. In order to allow everyone an opportunity to experience stock market trading and learn more about the financial industry, we want to create a stock market fantasy game, and we would like to commission you to create it. Our vision is to provide a stock market fantasy game with a twist: incorporating artificial intelligence and cryptocurrency trading to aspiring investors of all experience levels.

With *Titan Trading*, investors, traders, and students of all experience levels will have a chance to interact with the stock market in a fun, competitive manner without using their actual finances. We want our game to teach the novice trader the basics of investments and give them a platform to try things out before committing to investing in real life. For the more experienced day-trader, our game will give them a chance to try out some new investment strategies, or compete against their colleagues without risking their real assets. We can bring together traders of all levels in a no risk environment where they can really learn about investment trading. The market includes a wide range of assets from commodities, bonds, currency, etc. each of which follow different trends and trading practices. On top of the diverse investment options, there are also the tools, which include market limits, buy/sell orders, dividend pay comparisons, and graphs and data visualization tools, that investors use in their work on the market floor. Even for an experienced trader, the stock market is a complex and ever growing challenge, so being able to practice and learn using real data and tools, without risking personal loss, is what we want our game to provide.

So far, what we're asking you to do has already been done by a lot of different companies, and we want to take our product to the next level in order to give our users a more engaging way to practice trading. First, we want this program to be like a game where you compete against other people, and depending on your standings, you can earn rewards. By including the competition aspect, we hope to incentivize players to make strategic and profitable investments, instead of making risky and unrealistic trades. Furthermore, we want to have a universal league of traders that includes every user of our product. This, along with the competitive nature of the game, will provide for a more realistic trading environment for the players participating in our game. We also want to allow users to create private leagues so they can compete with people their friends or colleagues. This will allow people to participate in competitions with people of their own skill level rather than all the other people in the world. The big thing we hope to incorporate into our game is the use of AI. Players can try their hand at competing against an AI

to see who is the better trader. A default AI will be included in games should the players select it as an option in their league. In addition to the default AI, players can purchase other AIs. From our digital store, players can purchase "TitanCoins." TitanCoins are used to buy in game items, such as buying text integration to perform transactions. With the different features our game provides, we hope that this encourages beginners to begin learning about trading with their friends, and eventually participate in real trading.

Currently, artificial intelligence and machine learning have given large companies an advantage when deciding which trades to execute. We want to bring this technology directly to the users, so that they may learn from the AI's computing powers. We would like the user to be able to compete against an AI and view the data in which it uses to make decisions. By displaying the data in which the AI uses to make predictions, any novice or expert trader will get new insight to this upcoming technology and will further understand its impacts on today's finance society.

In addition to the AI feature, the Titan Shop will bring new revolutions to the stock simulator industry. Not only will the Titan Shop provide incentives for these traders to become more invested in the game, but it would also allow the game developers and company to monetize the game without using advertisements. By using the Freemium business model, this company can sell new and fresh features without players feeling contracted to a monthly subscription fee. These purchases will be easily accessible by connecting to one's PayPal account.

The second thing we want our product to include is to incorporate cryptocurrency trading. Today, cryptocurrencies have taken the spotlight with their exponential growth rates and the news cycle dominated by cryptocurrency analysis once Bitcoin peaked at 19,500 dollars per coin. Bitcoin is just one of the many cryptocurrencies available to the public, albeit one of the most well known ones. Cryptocurrencies still rather new to our society, but anyone can invest in them and certain coins have shown 20,000 times return for the investor. Now that the cost for one coin is so high, finding a good entry point to investing is difficult so we want to give our users the ability to practice trading cryptocurrency before deciding whether or not they want to make the investment in real life.

Although the whole premise of this project is just a game, there must be definite rules to guarantee fair play. In terms of leagues, every user registered on the website will enter a *universal league*. In this league, each user enters the stock market (without cryptocurrency) with the same buying power. To ensure that other users are not unjustly punished for playing in market slumps, this universal league will operate in seasons. At a season start, any currently registered user on the site will automatically be registered into the universal league for that season, and anyone that registers during a current season of the universal league will not be able to participate until the next season starts. The rules for the universal league should also be made to mirror the real-world as closely as possible. Other private leagues can be made by users, and these leagues will have preferences set by the league creator to dictate the rules for that game. Rules for one league may seem cheap or unfair to other players, so allowing private, customizable leagues is important to ensure that everyone plays their own fair game. These rules can include: the exclusion of certain stocks, the time needed to conduct transactions, interest, dividends, loans, and more.

This product needs to solidify competitiveness to ensure game longevity. For all leagues, leaderboards will be included to rank players. Each player's profile will record their league

rankings as a way to show off his or her hard work. A universal leaderboard will also be included that factors in all seasons of the universal league as a way for users to compare each other across the website. Rewards for placing well within notable leagues will also be included, such as site announcements and extra features. Special leagues will also be provided, which only invite users that have placed extraordinarily well in the universal league. This will give a desire for users to differentiate themselves from the competition by providing a more intense league.

Although the primary motive to play the game is for the competition between users, the product should also cater to beginners to ease the learning curve. After a user has registered on the site and is currently playing in a league, the website will have several pages that serve as the game manual and help guide. These pages will display videos that explain the stock market, how to buy and sell stocks, and other general stock market essentials. Leagues other than the universal league will also be provided specifically to beginners to meet and play with other eager, newer users.

The overall aim of this game is to both educate users about the stock market and to provide a competitive environment for enthusiasts to demonstrate their market skills, while not being as cutthroat as the real world market. The advantage over current stock market games will be the ability to trade cryptocurrencies, artificial intelligence, a monetized game, a more refined competitive aspect, and a beginner-friendly environment.

Sincerely,

The Rutgers Investments Co.

2 GLOSSARY OF TERMS

League - A collection of aspiring investors participating in an instance of a stock market simulation. Each league will come with its own pre-determined rule set and goals. A winner will be determined at the end of a season.

Universal - Every investor will be automatically slated into this league. It will involve all active users on the program.

Private - League will be created and maintained by a League Administrator.

League Manager - The creator of a private league. He/she will be responsible for inviting participants and designating the ruleset including victory conditions and timespan. Users are only admins for the league he/she has created.

Cryptocurrency - A form of virtual currency that uses cryptography to secure transactions, regulate the generation of units of currency, and verify transfer of assets. It is susceptible to volatile prices.

Microtransactions - The profit model of offering small features of a game for sale for increased profits. The player will be able to purchase additions to our game such as artificial intelligence difficulties through microtransactions.

Machine Learning - A method of generating an algorithm or rules that a computer opponent will follow when making trades to compete against human players. Different difficulties will be created for the computers by using different Machine Learning techniques.

Email and Text Notifications - The player will be allowed to sign up for email or text communication in where they will be able to interact with their portfolio through text or email messages.

Receipt - PDF or email record of a transaction that a player made that they can save for record keeping.

TitanCoin - currency the player can purchase to exchange for various microtransaction upgrades to the game.

Investor - One who puts money to use through purchases with the expectation of profits

Assets - Property owned by company or person, in our case, the user. It is regarded as having value.

Commodity - A basic good such as a food, metal, or agricultural product that investors can buy and sell.

Bonds - A fixed monetary investment that an investor loans for a defined period of time at a specific interest rate

Buy Order - Tool in the game used to purchase stock or cryptocurrency. It will automatically handle acquisition of assets. Buying power will update to reflect this transaction.

Sell Orders - Tool in the game used to sell stock or cryptocurrency. It will automatically handle the sales of assets. Buying power will update to reflect transaction.

Buying Power - Amount of free spending money available for use for an investor.

Season - A division of the year marked by changes in weather and daylight hours. This will be used as the length of time for the Universal League. Approximately three months.

Machine Learning - The use of statistical techniques to give computer systems the ability to "learn" from their actions and input data

Artificial Intelligence - A computer system that is able to perform a task that normally require human intelligence such as decision-making.

3 WHAT SETS US APART

Titan Trading is composed of a group of passionate engineers looking to develop a competent and working product for the enjoyment of users. Our product is above all else, a fun, educational, and rewarding gaming experience that does not simply aim to provide a realistic trading experience. Whereas other projects focus on realism and modeling real-life stock trading, we at Titan Trading aim to gamify the system and introducing popular aspects from modern video games to give user, above all, a fun experience. To do this we implement some important gaming concepts in our trading app that typically would not be seen in such an environment, summarized below:

1. **Artificial Intelligence** - Good games merely display goals to the player, great games challenge the user! Our product makes use of Tensorflow, an open source machine learning framework designed and developed by the Google Brain Team. This framework allows us to use machine learning to create realistic and challenging AI opponents of different difficulty scales. Our hardest AI opponent can even make use of live Twitter and news resources to make educated guesses about a stock before buying or selling. These AI opponents add a new dimension to the typically drab and singular gaming experience that are given by other stock trading apps. As of writing of this document, no other stock trading app on the market, or created for the purposes of this class, makes use of Tensorflow to create a realistic AI opponent using machine learning.
2. **Free-to-play shop interface** - The ability for players to spend real money on in-game items has become commonplace in modern games media. However the implementation of this in a stock trading game has yet to be seen. As such we found it appropriate to add this as a key component of the product. To do this, we created a virtual currency known as TitanCoins, which can be purchased with real funds in a Paypal account. The inclusion of an in-game store gives users a chance to get ahead and enjoy the experience for a nominal fee. TitanCoins can also be earned by being a top trader in a league.
3. **SMS integration** - Most applications restrict the user to only using a single tied down interface to make trades. However, Titan Trading aims to differentiate itself by opening up the interface to taking input from various input methods, including vectors. Should any user feel the sudden urge to act on an instinct, or if news breaks and they aren't near their computer, they can use our integration with Twilio to make trades over SMS. A simple text and your trades can be specified and made.
4. **Social media and email integration** - Through the use of unique identifiers for private leagues, we have created a working system that allows users to share the experience with their friends. Facebook, Twitter and LinkedIn all present diffrent avenues for inviting users and getting more people involved in the fun. We also make use of email as a way of inviting friends. All that you need is the name of your league and the password, allowing users to share with their friends in a simple and easy way.
5. **Paypal integration** - The use of real real money to fulfill shop requests needs a way to process these transactions in a secure and safe matter. Our product is able to integrate seamlessly with Paypal's Merchant interface to generate real charges to either a Paypal account or, using the Paypal platform, a real credit or debit card.

4 SYSTEM REQUIREMENTS

4.1 User Stories

Identifier	User Story	Point Value
ST-1	As a user, I can create an account by registering my email address with the website in order to save my information and play history.	2
ST-2	As a user, I can login to my account through a login page so I can verify my credentials via my username and password.	1
ST-3	As a user, I can view an About page that details information about the team who developed *Titan Trading* to learn more about the motivation behind this product.	2
ST-4	As a user, I can view an Instruction/FAQ page that details different features of the application in order to receive guidance on how to play the game.	2
ST-5	As a user, I will automatically be entered into a universal league - league in which all users compete - with a fixed starting balance so I can immediately start gameplay and make transactions.	4
ST-6	As a user, I can join a league either by creating a new league and becoming a league manager or by being invited to a league by another league manager, with the exception of the universal league.	4
ST-7	As a user, I can create a private league with an unlimited amount of members and automatically be designated as a league manager upon league creation in order to manage league details and rules.	3
ST-8	As a league manager, I should be able to set rules for my league, including a starting balance, and league name, in order to manage gameplay among a smaller group.	3
ST-9	As a league manager, I should be able to invite other users by email address to start a competition with additional players in my league.	2
ST-10	As a league manager, I would like to create private cryptocurrency leagues that only allow trading of cryptocurrency in order to isolate the volatile market of cryptocurrency from the regular market. This can be designated as a setting during league creation.	4
ST-11	As a league manager, I can set an end date for each of my privately	1

	owned leagues in order to define the end of the game. The player with the most money wins.	
ST-12	As a league manager, I can post or send announcements to the members of my league.	3
ST-13	As a user, I can access a Dashboard page that allows me to see the leagues I am competing in, my rank and balance in each league, and my statistics so I can keep track of my progress.	3
ST-14	As a user, I can view a page with current stock market trends so I can make an educated decision about which stock to buy.	3
ST-15	As a user, I can change my personal information and avatar to personalize what my profile looks like to other users and how it is displayed on leaderboards.	3
ST-16	As a user, I can view a leaderboard for each of my leagues that ranks the top 10 people with the most money in the league, and it must update all the time.	4
ST-17	As a user, I should receive notifications for selected stocks to know when they have exceeded or dropped past a threshold value set by me.	5
ST-18	As a user, I can access a transaction page in order to buy new stocks and sell stocks that I own.	9
ST-19	As a user, I can search companies either by their full name or acronym so I do not have to spend extra time searching for a company.	2
ST-20	As a user, I can receive accomplishment trophies once I reach specific milestones and achievements within the universal league so I am motivated to continue playing.	4
ST-21	As a visiting user, I should be able to view the home page, as well as view the statistics for the universal league (even if I do not have an account) in order to view the competition the game provides.	2
ST-22	As a site administrator, I can change universal league settings and view and delete inactive leagues to remove unwanted data.	3
ST-23	As a site administrator, I can access data about site statistics, including the number of users, active and non-active leagues, and transaction details in order to display accurate statistics on my website.	3
ST-24	As a user, I can view a glossary page that will tell me the key terms	1

	needed to start investing.	
ST-25	As a user, I can view an investing guide page that will tell me how the stock market and cryptocurrency work and how to invest properly.	1
ST-26	As a user, I can compete against an easy, medium, or difficult artificial intelligence in my private league.	2
ST-27	As an artificial intelligence, I can predict the best stocks or cryptocurrencies by analyzing monthly stock data, as well as human sentiment in recent tweets and news articles.	4
ST-28	As a user, I can buy new features from a Titan Trading Shop, such as text integration to submit transactions or to buy AIs to trade for me.	4
ST-29	As a user, I can connect my PayPal account so that I can purchase virtual Titan Bucks with real money.	2
ST-30	As a user, I can print out transaction receipts so I can keep track of muy buy and sell orders.	1
ST-31	As a user, I can access the website on any computer, as long as there is Internet.	1
ST-32	As a user, I would like to purchase and compete against an AI so I can gage my level of stock trading experience.	9
ST-33	As a user, I would like to be able to purchase text compatibility and link my phone number to my account so I can trade through my phone.	6
ST-34	As a player, I would like to be able to get receipts of all my transactions so I can track of the sales and purchases I have made.	4

4.2 Non-Functional Requirements

Functionality

In order to be able to accommodate for a growing user base we will be creating our web application around a Postgresql database, which is known in the software industry for its ease of scale. Additionally we will be creating private leagues that up to 20 people can join with their friends in order to make the game more personal for users. We will also be using password hashing and salting in order to protect user credentials.

Usability

We will be using Django, which is a Python web-application platform in order to create our website's Model View Controller. Along with this we will be using Bootstrap to create a good looking website that will be smooth and easy to navigate and use.

Reliability

We will be prototyping our website on a heroku instance, and deploy it to Amazon Web Services (AWS) for maximum uptime. All transactions will be performed and verified server side and not client side so that someone could not cheat and manipulate the trades going on. We will also keep backups of user data in case of a server failure.

Performance

Since we are using Postgresql, we have the ability to scale up and provide for a large number of users on the fly with great ease. Additionally, python makes efficient web-server applications so we will not have to worry all that much about demands on the server.

Supportability

Python provides a lot of debug tools, and provides a lot of information when things go wrong which will benefit us greatly when debugging our program. We will also be able to provide users with error messages when things do go wrong so that they know that their trade has not gone thru or various other errors that could happen.

4.3 On-Screen Appearance Requirements

On-screen appearance requirements detail requirements related to the user interface experience and are written from an objective perspective. These requirements have been separated into two categories: all site requirements and page-to-page requirements. All site requirements are those that apply to every single page across the website domain, while page-to-page requirements are those specific to different web pages.

We have brainstormed the formats for 5 different types of web pages: the homepage, dashboard page, individual league page, buy page, and sell page. We also intend to include a custom 404 not-found page, FAQ page, glossary page, and instruction page, whose requirements are also listed below.

4.3.1 All Site Requirements

Identifier	Requirement	Points
OSR-1	Every page will have navigational tabs at the top of the screen to access the home, about (dropdown with about us, FAQ page, instruction page, glossary), and dashboard, as well as a drop down menu to access their profile settings or log out.	1
OSR-2	Every page will have the logo of *Titan Trading* in the top left corner.	1

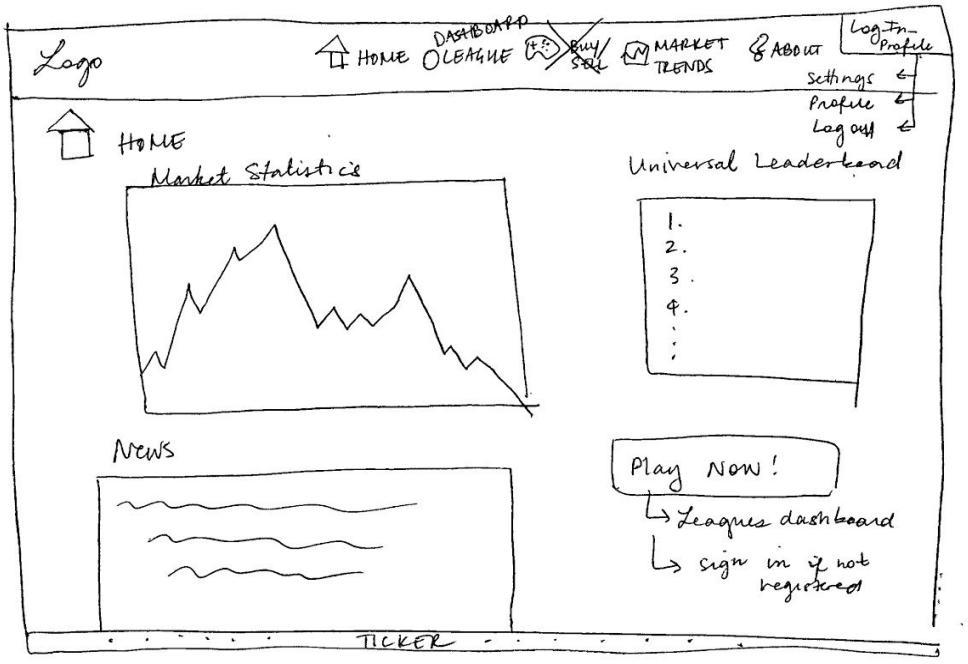
4.3.2 Page to Page Requirements

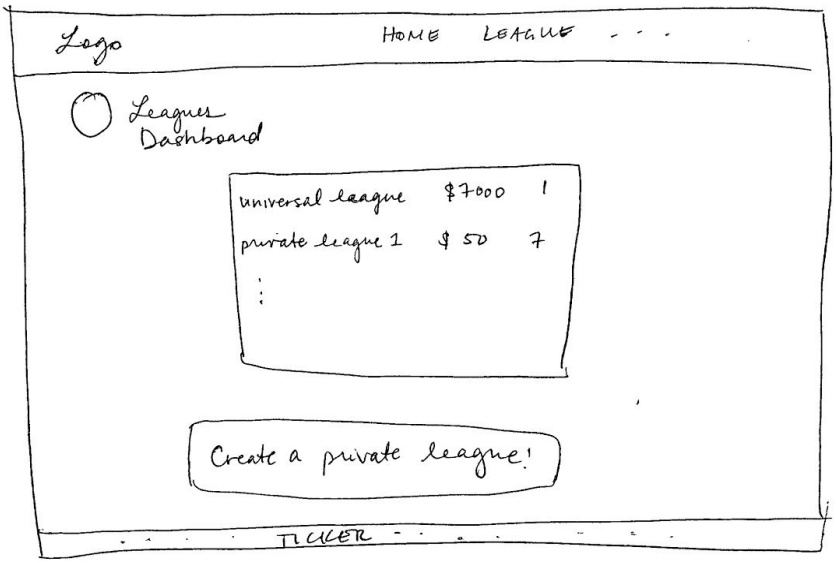
Identifier	Requirement	Points
OSR-3	A custom 404 not found page will be displayed to a user when they try to access a URL/URI that does not exist or is not designed for them to be accessing.	1
OSR-4	The homepage should include current market trends and articles that are tailored to my interests, as well as the leaderboard for the universal league.	3
OSR-5	The dashboard page should contain a list of leagues the user is competing in and a button to create a new private league.	2
OSR-6	Every individual league page should present information about the user's current buying power, total worth, and rank within the league.	2
OSR-7	Every individual league page should present buttons to buy or sell stocks.	1
OSR-8	Every individual league page should present a leaderboard of the top ranking players within the league and a list of league settings, eg: league end date, starting balance, league administrator, etc.	3
OSR-9	The buy page should present information about the user's current buying power, total worth, and rank within the league.	2

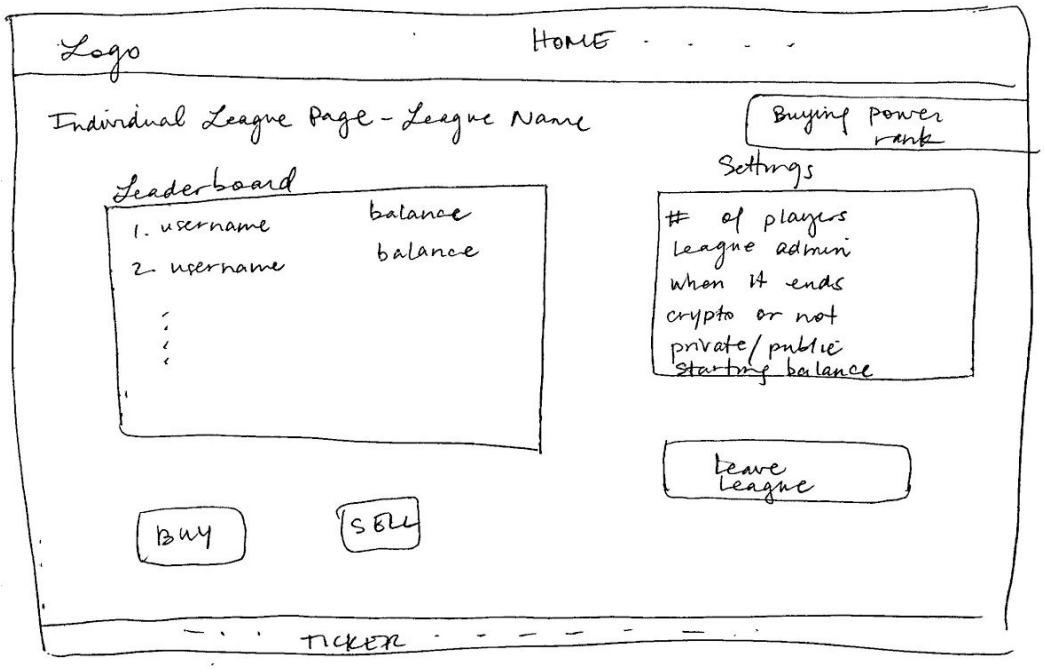
OSR-10	The buy page should contain a widget that will return market statistics about the stock the user searched for.	2
OSR-11	The buy page should present a form that allows the user to submit buy orders.	2
OSR-12	The sell page should present information about the user's current buying power, total worth, and rank within the league.	2
OSR-13	The individual league page should have a portfolio of all the stocks the players owns within a specific league and should have a button to submit sell orders.	4
OSR-14	The About page should include the mission statement behind Titan Trading.	4
OSR-15	The FAQ page should present answers to common questions such as, "What is the Universal League? What is a private League? I'm new to investing, is there a place I can go to learn more about stocks? What is cryptocurrency and how is it involved in our game?"	3
OSR-16	The glossary page will have a list of trading terms and definitions that will be sorted alphabetically.	2
OSR-17	Each user should have a profile page to display my trophies, username, and a description about me.	2
OSR-18	The shop page will display different features you can purchase with Titan Bucks.	4
OSR-19	The artificial intelligence page will be accessible from each private league page, and it will showcase the artificial intelligence's transactions and the data used to make its decision.	3

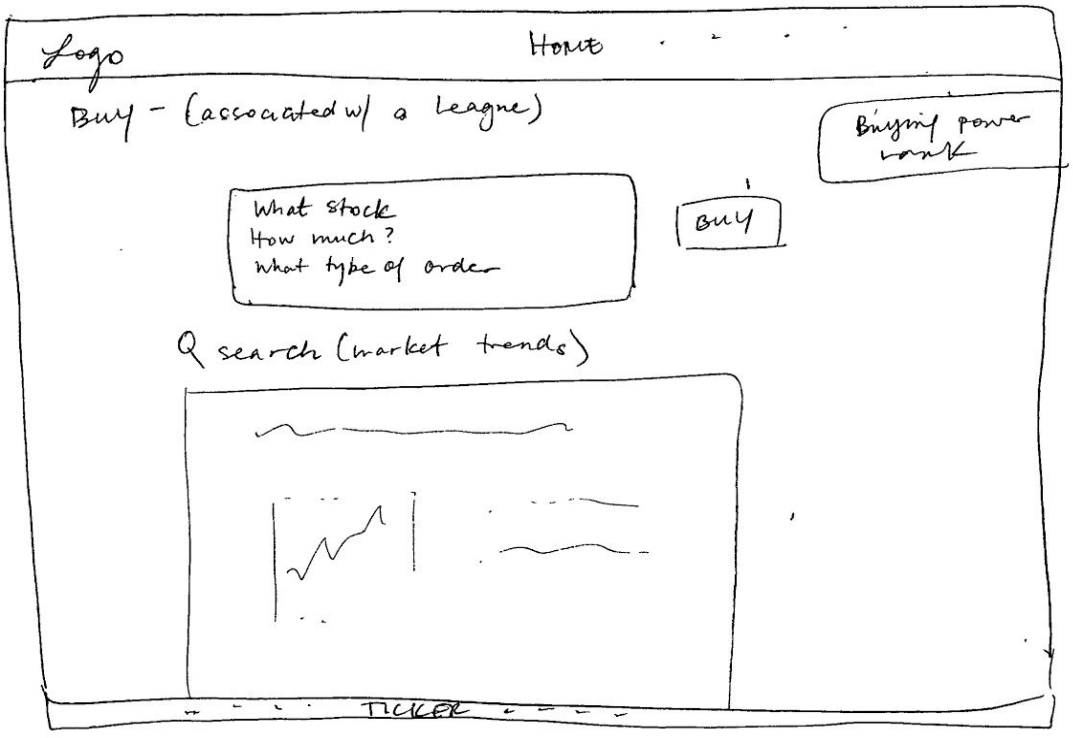
4.3.3 Sketches of User Webpages

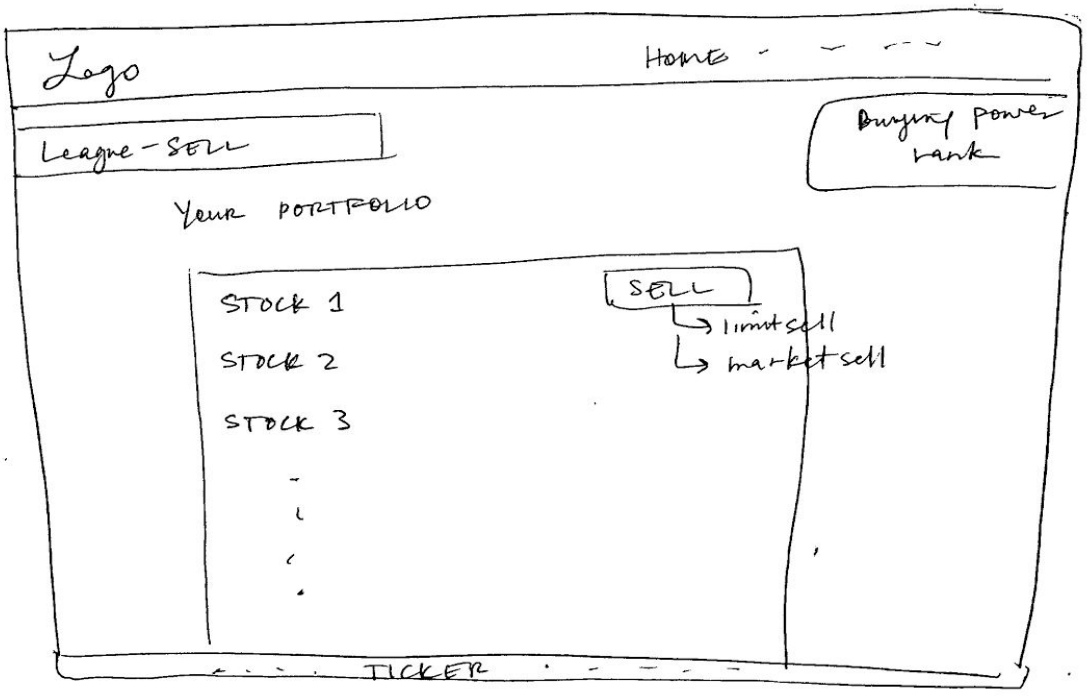
Brainstormed sketches of Titan Trading webpages are shown below. They include: the homepage, the leagues dashboard page, an individual league page, buy page, and sell page.


Logo
HOME
DASHBOARD
LEAGUE
BUY/SELL
MARKET TRENDS
ABOUT
Log In-Profile
Settings
Profile
Log out
HOME
Market Statistics
Universal Leaderboard
1.
2.
3.
4.
News
Play Now!
Leagues dashboard
sign in if not registered
TICKER


Logo
HOME
LEAGUE
Leagues Dashboard
universal league $7000 1
private league 1 $ 50 7
Create a private league!
TICKER

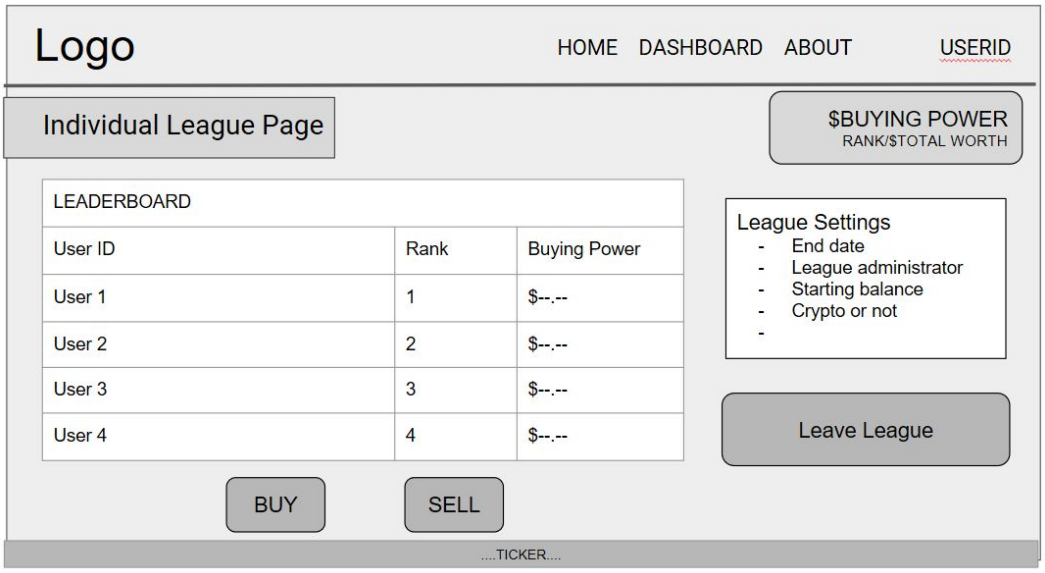

Logo
HOME
Individual League Page - League Name
Buying power
rank
Leaderboard
1. username
balance
2. username
balance
Settings
# of players
League admin
When it ends
crypto or not
private/public
starting balance
Leave League
BUY
SELL
TICKER

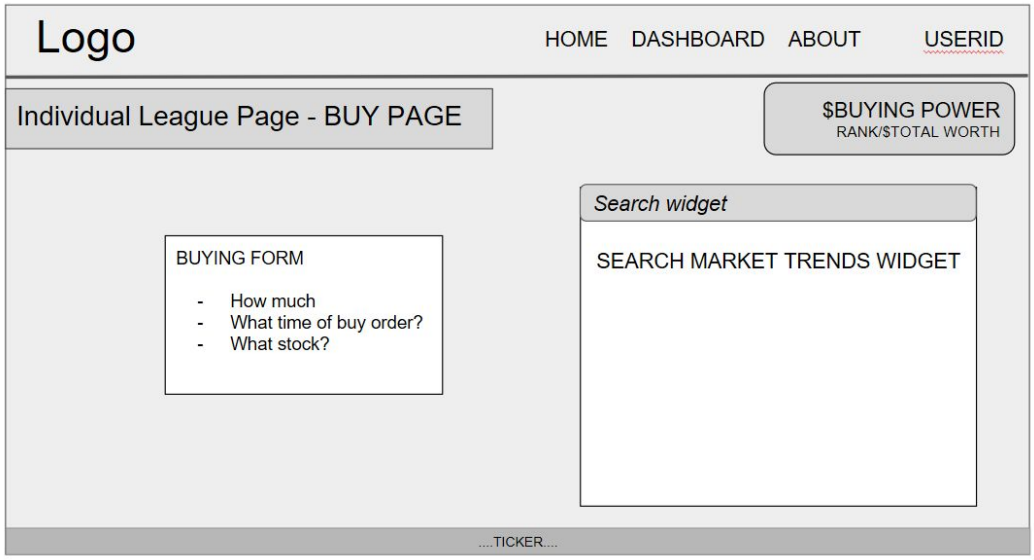

Logo
HOME
Buy - (associated w/ a League)
Buying power
rank
What stock
How much?
What type of order
BUY
Q search (market trends)
TICKER

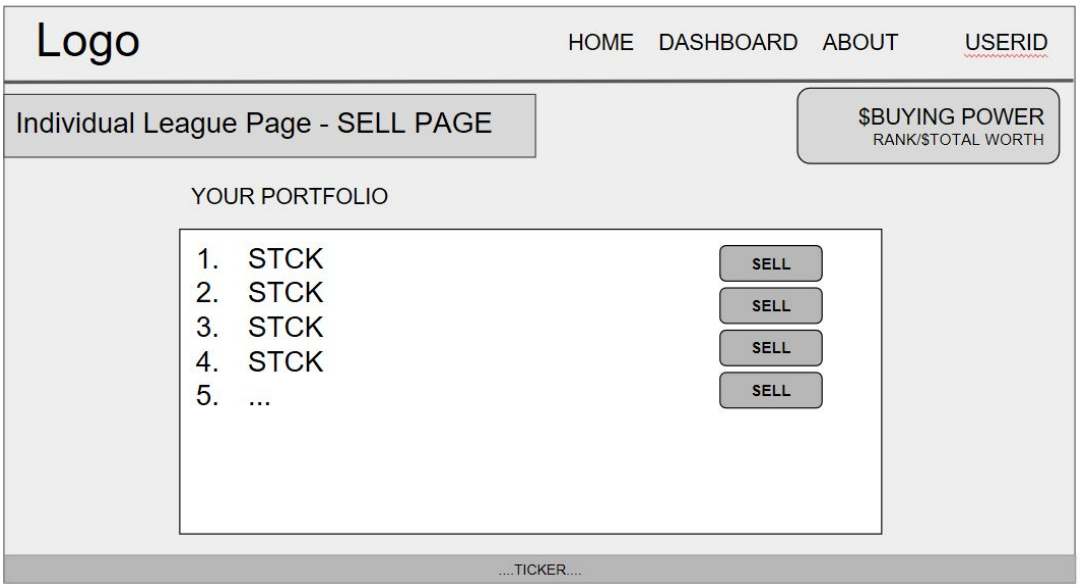

Logo
Home
League - SELL
Buying power
rank
Your portfolio
STOCK 1
STOCK 2
STOCK 3
SELL
limit sell
market sell
TICKER

4.3.4 First Draft UI Electronic Representation

After finalizing a first draft of the UI layout, we made an electronic representation







4.4 Acceptance Tests

Acceptance tests that the customer will run to check that the system meets the requirements are as follows. Note, however, that these test cases provide only a coarse description of how a requirement will be tested. Acceptance tests will follow a naming convention of AT-X.Y where 'X' is the number of the requirement that is being tested, and 'Y' is the number for the given test. Following the requirements table the following acceptance tests have been formulated:

Acceptance Tests for ST-1

AT-1.1 - Register into the system using a valid email address and fill out required information. (Pass: New account information is registered in database)

Acceptance Tests for ST-2

AT-2.1 - Using a pre-existing account, log in using a correct email and password.(Pass: User information is able to be pulled based on the logged in user)

AT-2.2 - Attempt to log in with an incorrect email and password. (Pass: The system rejects an incorrect combination of email and/or password)

Acceptance Tests for ST-3

AT-3.1 - Have a test user navigate to the about page and attempt to give a summary of the product based on information on the about page. Match this user description to some standard

description given by the development team. (Pass: The test user description matches closely to the desired developer description)

Acceptance Tests for ST-4

AT-4.1 - Have a test user, unfamiliar with the product, navigate to the FAQ and attempt to understand and answer their questions using the given FAQ. (Pass: No questions that are obvious are left unanswered and not on the FAQ)

Acceptance Tests for ST-5

AT-5.1 - A new user is automatically entered into the universal league, and their statistics are viewable on a global leaderboard. (Pass: User A is able to see the standing of User B on the global leaderboard)

AT-5.2 - A given logged in user is able to check their own standing in the universal league. (Pass: User A can navigate to the global leaderboards and see their standing)

Acceptance Tests for ST-6

AT-6.1 - A league administrator can invite others to join the league over email.

AT-6.2 - A league administrator can invite others to join the league using an existing username.

AT-6.3 - Non pre-registered users can sign up for the website and join a league they were invited to over email.

AT-6.4 - Pre-registered users can sign up for the website and join a league they were invited to over email or through their existing username.

(Pass cases all involve seeing a user either register and be shown asa player in the league, or an already registered user is put into the league.)

Acceptance Tests for ST-7

AT-7.1 - Ensure that any user can create their own league, becoming the league administrator. (Pass: the user creating the league is placed asd a player and is given league administrative permissions.)

AT-7.2 - League should be limited to the 20 player limit.

Acceptance Tests for ST-8

AT-8.1 - Preferences and settings for a given league are set upon creation. (Pass: The league admin can set the settings of the league upon its creation.)

AT-8.2 - Certain preferences for a given league are changed after the creation of the league. (Pass: Settings not critical to be set at the creation can be adjusted.)

Acceptance Tests for ST-9

See **AT-6.3** and **AT-6.5**

Acceptance Tests for ST-10

AT-10.1 - Ensure that when creating a league, the administrator is able to specify the use of cryptocurrencies in the league. (Pass: The setting is marked and the league will use cryptocurrencies instead of stock information)

Acceptance Tests for ST-11

See **AT-8.1** and **AT-8.2** fro preference setting. This is simply another preference to set and ensure that it is active.

Acceptance Tests for ST-12

AT-12.1 - League administrator can write up their own message or use a predefined message for certain actions.

AT-12.2 - Announcements can be sent by the league administrator to the players in the league (Pass: Users receive notification and message from the admin)

Acceptance Tests for ST-13

AT-13.1 - A user can navigate from any point on the website to their leagues page.

AT-13.2 - Ensure that a user can clearly see the leagues that they are a part of.

AT-13.3 - Ensure that a user get statistics and information regarding each league.

(Pass: For each of these the pass is straightforward and involves proper displaying of the information for a given user by the system and the ability for the user to navigate the UI.)

Acceptance Tests for ST-14

AT-14.1 - Statistics about individual stocks can be navigated to.

AT-14.2 - Market trends can be navigated to and viewed using graphs and data visualization.

AT-14.3 - A user can navigate the UI and easily reach the market trends page

Acceptance Tests for ST-15

AT-15.1 - From the user account page, view the current profile picture and allow for selection of a new profile picture.

AT-15.2 - User should also be allowed to change their display name.

AT-15.3 - Other personal information on the account page should be changeable.

(Pass: Successful change of any of the information on the account)

Acceptance Tests for ST-16

AT-16.1 - A specific league's leaderboard can be navigated to and is displayed to the user.

AT-16.2 - The timing system updates the current standings at midnight of each trading day, or at the end of the trading day. (Pass: League 'A' is given new standings and updates automatically based on the time of day)

Acceptance Tests for ST-17

AT-17.1 - Notification of stock price drops below/above a certain level, automatically set. (Pass: Uses holding the stock or interested in a stock are notified when an pre-set threshold is met.)

Acceptance Tests for ST-18

AT-18.1 - Users can navigate to the transactions page.

AT-18.2 - Users can buy stock from the transactions page.

AT-18.3 - Users can sell stock from the transactions page.

Acceptance Tests for ST-19

AT-19.1 - Search functionality supports stock tickers and full company names. (Pass: The system can successfully return the company page based on a predetermined list of companies of varying popularity)

Acceptance Tests for ST-20

AT-20.1 - A user's achievements can be navigated to and viewed.

AT-20.2 - Notifications for getting an achievement are properly distributed and displayed.

AT-20.3 - Achievement progress is properly tracked and displayed.

Acceptance Tests for ST-21

AT-21.1 - A user that is not logged in is served the home page.

AT-21.2 - The home page displays the global leaderboard, which a non-logged user can still view. (Pass: A non-logged in user is served no user specific data and is instead given the standard homepage.)

Acceptance Tests for ST-22

AT-22.1 - Site administrators have ultimate ability to view and delete leagues (Pass: A site administrator account deletes a league that they are not a part of)

Acceptance Tests for ST-23

AT-23.1 - Visualization about the backend active users, number of leagues, stock trades and transaction details can be accessed by any site admin. (Pass: An admin is able to view a transaction in any given league on multiple occasions. A site admin can view the data for any randomly given league.)

Acceptance Tests for ST-24

AT-24.1 - Glossary page can be navigated to from anywhere on the site.

Acceptance Tests for ST-25

AT-25.1 - Investing guide page can be navigated to from anywhere on the site.

AT-25.2 - The investing guide provides a good getting started point and offers plenty of resources for a novice trader to learn and grow. (Pass: Give a novice test user access to the guide

and perform a user test. The guide should provide the novice a starting point where they would feel comfortable jumping into using the product.)

5 FUNCTIONAL REQUIREMENTS SPECIFICATION

5.1 Stakeholders

The main stakeholders of Titan Trading are novice traders who want to learn more about the stock market and gain trading experience. Through this application, they will be able to participate in the stock exchange without using real currency. Through their support and use of our site, we will be able to better simulate the stock market and create a more genuine experience. In addition to those using Titan Trading for learning, there are those who can use this application as a teaching tool. Teachers or people teaching others about the stock market can use Titan Trading to expose others to the stock market and how trading works. Furthermore, Rutgers Investments Co. is a stakeholder of Titan Trading. Since they are the ones commissioning the creation of this application, the success of Titan Trading directly affects the investment they made in the program. On a similar note, we, the developers of this application, are stakeholders of Titan Trading since we are invested in the creation of the program. The success or failure of Titan Trading affects our reputation as programmers, and whether or not the application is popular is a reflection to our abilities to create a good UI and program.

5.2 Actors

Primary Actors:

For every use case there are direct actors who are involved in the use case/user story

Site Administrator - Owns the website and can edit the user interface and universal league settings. Is a league manager for the universal league.

User - Someone who has an account and can become a player by joining leagues.

League Manager - A user who owns a private league and can adjust any settings for the league.

Player - A user who is a member of a league.

Universal Player - A user is by default a player in the universal league.

Private League - A player profile created for a private league and has a player ID unique to this league.

Artificial Intelligence Opponent - A bot created to play against the player based on rules generated through machine learning. Different difficulty levels will be available for different prices.

Guest - Someone who visits the website without creating an account. Can view our homepage containing market statistics.

Secondary Actors

The secondary actors participates in the use case but does not initiate it. Subtypes include supporting actors and offstage actors. They help achieve the goal or must know about the outcome.

Finance API - Service to request current market data and stock prices. Includes information about stock time series data, cryptocurrencies, stock technical indicators, and sector performances. We plan to use the Alpha Vantage API.

Database - Stores all information related to app usage, including user data, leagues, league statistics (leaderboard, player rank), and user-league associations.

Microtransaction Market - allows users to purchase AI opponents and other add-ons for the game through Titan Bucks.

Email API - Service to allow allow sending players email. Email is used to send transaction records to players, and to invite other players to private leagues.

Twilio Text API - Service to allow direct messaging to players through text. Players can use this API to play the game, through means such as buying or selling stocks.

TitanCoin - currency user uses to purchase various microtransactions. Used as rewards when players win in a league.

Transaction Receipts - PDF or email record that provides a player with confirmation that they have made a transaction and provides details for that transaction.

Browser - Provides a user interface for users to interact with app HTML pages.

Widgets - Website modules that display relevant data, including news, stock information, and market trends.

5.3 Use Cases

Our application has two primary modes of access - guest and user. Guests can view the homepage, About page, and FAQ page. To use the full functionality of our application, guests sign up through a login page. Once signed up, users are automatically entered into a universal league, managed by the site administrator. Users can also create their own private leagues and invite other users to private leagues.

Potential use cases:

* indicates that the use case is important and should be described in a "Fully-Dressed" manner.

1. *Create a league (User)
2. Sign up (Guest)
3. Login (User)
4. Logout (User)
5. *Invite user to League (Manager)
6. *View statistics (Guest/User)
7. *Buy stock (Player)
8. *Sell stock (Player)
9. *View Dashboard and Leaderboards (User)
10. Accept/decline league invitation (User)
11. Search stock (Player)
12. *View FAQ/Instructions (User)
13. Unlock Trophy (User)
14. Buying and Competing Against an AI (User/Player)
15. Purchasing and Linking Text Compatibility From the TitanCoins Store (User)
16. Buying/Selling a stock using Text Compatibility (User)
17. Printing a Transaction Receipt (Player)

Casual Descriptions

***Note: ST-23 is related to all use cases because it is a general requirement for the system administrator.

UC1 - Create a league (ST-6,7,8,10,11,22, 31)

A user can create a new private league, and select it as a cryptocurrency league or regular stock league, set the starting balance, and set an end date for the league. The end date shall dictate who has the most money from all players in the league, declaring the winner.

UC2 - Sign Up (ST-1,5, 31)

A visitor can create a new account by clicking the “Sign Up” button on the homepage. To create an account, the visitor registers with their email address and chooses a username and password. After account creation, the now user will automatically be entered in a universal league.

UC3 - Login (ST-1, 2, 31)
After account creation, a visitor can log into an account through a login page by entering in a valid username and respective password. The system will update the website to reflect the visitor is now a user.

UC4 - Logout (ST-31)
The user can log out of an account by clicking the “Logout” button on the website. The system will redirect the user to the homepage and reflect that the user is now a visitor.

UC5 - Invite User to League (ST-6,7,9,31)
A league manager can invite users to a league by clicking “Add Users” on their league page. Then, the league manager will enter in the desired users’ emails. The system will email invites to these users, which the users can accept via hyperlink. If the email is not registered, the user can create an account directly from the invite.
UC6 - View Statistics (ST-14,21)
A user or guest can view market statistics by clicking the “Home” button. The system will pull the current market trends from the Finance API and redirect the user to the homepage which will display these trends.

UC7 - Buy Stock (ST-5,18,27,31)
A user can buy stocks on any league by clicking the buy button for the league. The system will redirect the user to a buy form, which the user will fill out with their desired stock name and quantity to purchase. The system will return the order total from the Finance API, and the user can submit the purchase given they have enough capital. The system will reflect the transaction in the database.

UC8 - Sell Stock (ST-5,18,27,31)
A user can sell stocks on any league by clicking the sell button for the league. The system will load a page displaying the user’s league portfolio. The user can then click the sell button from any stock in the portfolio and fill out the form to complete the selling process. The system will reflect the transaction in the database.

UC9 - View Dashboard and Leaderboards (ST-5,12,13,15,16,17,20,21,31)
A user can view the dashboard by clicking the “Dashboard” button, which lets the system redirect them to the dashboard page. From there, the user can select a league to view the league page, which includes the leaderboard.

UC10 - Accept/Decline (ST-6,9,31, OSR-19)
A user can accept or decline any pending league invites from league managers on the user’s respective profile page. To accept or decline, click the respective button and the system will reflect the decision.

UC11 - Search Stock (ST-18,19,31, OSR-11)
On the buy page, a user can search for a stock through the search widget. The system will match the user’s search against the names and symbols of stocks from the Finance API, and it will either display matching stocks or a failed search error message.

UC12 - View Instructions/FAQ (ST-3,4,24,25,31, OSR-15,16,18)
A user can hover over the “About” button, which will show buttons for the instruction and FAQ pages. The user can then click the chosen page to be redirected to that page.

UC13 - Unlock Trophy (ST-20,31)
A user will receive trophies after reaching milestones in the universal league or overall performance (e.g. doubling money in a league.).

UC14 - Buying and Competing Against an AI (ST-31,32)
A user can buy an AI of varying difficulties (easy, medium, or hard) and compete against it in a private league of choice.

UC15 - Purchasing and Linking Text Compatibility From the TitanCoins Store (ST-28,31,33)
A user can buy text compatibility from the TitanCoins store and link their phone number to their account. The Text Compatibility program will send a text to the number notifying them that the link has been completed.

UC16 - Buying/Selling a stock using Text Compatibility (ST-31,33)
A user can buy or sell a stock of their choice by texting the TitanTrading text compatibility number. The number will text them back with information on the transaction.

UC17 - Printing a Transaction Receipt (ST-30,31,34)

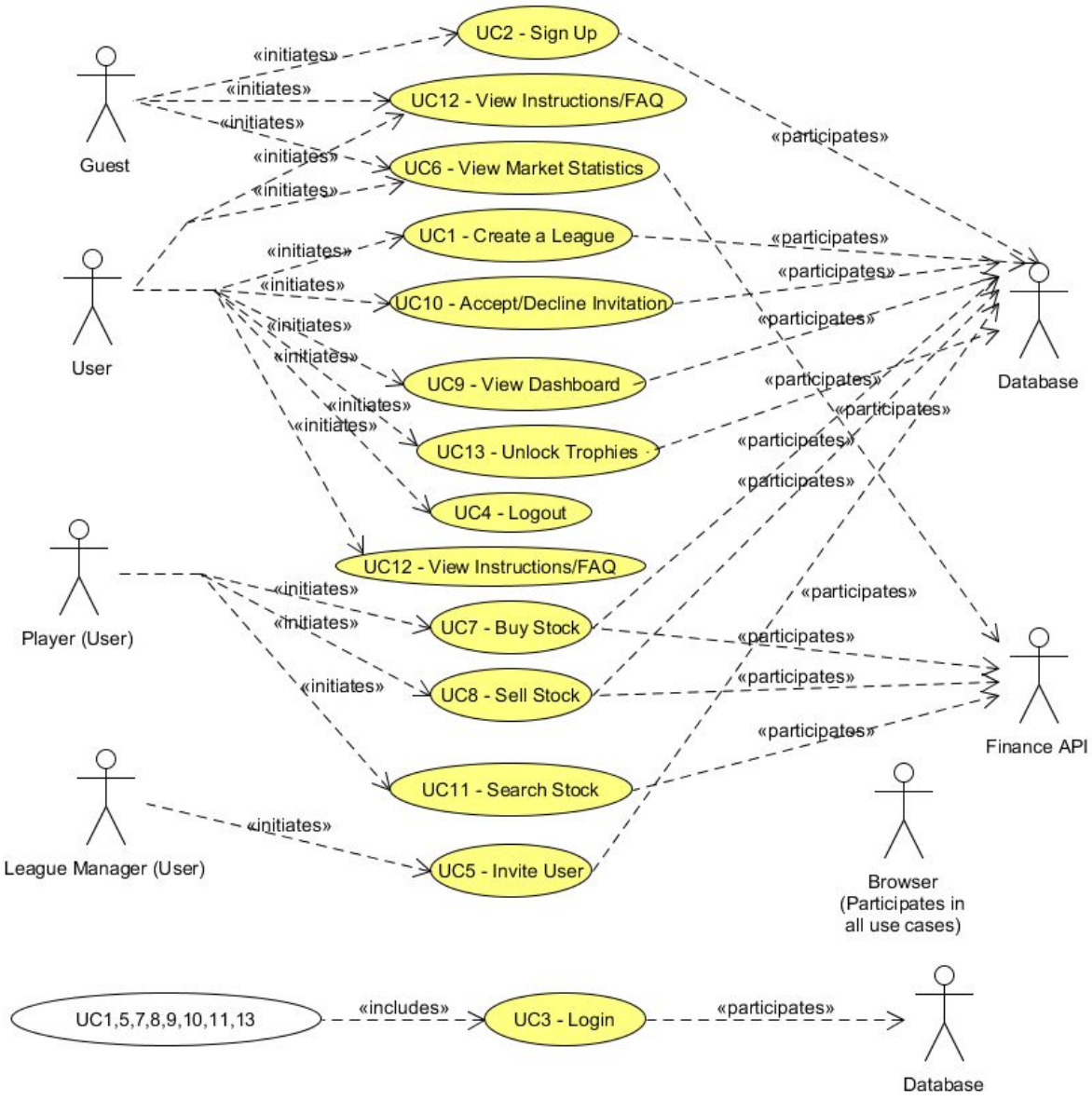
After a user completes a transaction, they will be redirected to a transaction receipt page where they will be able to print out the receipt.

UC18 - Connecting to PayPal (ST-31,29)

A user can connect their PayPal account to Titan Trading to purchase features from the Titan Trading Shop.

5.4 Use Case Diagram

5.4.1 Primary Use Cases



5.4.2 Secondary Use Cases

5.5 Traceability Matrix

User Story	Point Value	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC13	UC14	UC15	UC16	UC17	UC18
ST-1	2		X	X															
ST-2	1			X															
ST-3	2												X						
ST-4	2												X						
ST-5	4		X					X	X	X									
ST-6	4	X				X					X								
ST-7	3	X				X													
ST-8	3	X																	
ST-9	2					X					X								
ST-10	4	X																	
ST-11	1	X																	
ST-12	3									X									
ST-13	3									X									
ST-14	3						X												
ST-15	3									X									
ST-16	4									X									
ST-17	5									X									
ST-18	9							X	X			X							
ST-19	2											X							
ST-20	4									X				X					
ST-21	2						X			X									
ST-22	3	X																	
ST-23	3	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
ST-24	1												X						
ST-25	1												X						
ST-26	2																		
ST-27	4							X	X										
ST-28	4															X			
ST-29	2																		X
ST-30	1																	X	
ST-31	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
ST-32	9														X				
ST-33	6															X	X		
ST-34	4																	X	
Max PW		4	4	3	3	4	3	9	9	5	4	9	3	4	9	6	6	6	3
TOTAL PW		22	10	7	4	14	9	21	21	32	10	15	10	8	13	14	10	11	6

UC#1

Use Case UC#1 - Create a League
Related Requirements: ST-6 (User creates league), ST-8 (Set rules), ST-10 (Set as Crypto), ST-11 (set end date) **Initiating/Primary Actors:** User **Goal:** To create a new private league complete with settings **Participating Actors:** Database, Browser **Preconditions:** User is involved in fewer leagues than the league limit **Postconditions:** User is a league manager for the new league. User does not exceed the league limit.
Flow of Events For Main Success Scenario
→ User clicks the “Create a League” button ← System calls up the form for creating a new league. This includes the initial settings that are required to be set. → User completes the form by putting in their prefered settings. → User confirms the form is complete and that all settings are correct. ← System takes form data and updates the database with the new league ← System adds the newly created league to the user’s dashboard.

UC#4

Use Case UC#4 - Logout
Related Requirements: ST-2(User login), ST-21(Viewing the homepage as a visitor) **Initiating/Primary Actors:** User **Goal:** To logout of the user's account and allow access for another account to login **Participating Actors:** Browser **Preconditions:** User is already logged on to their account on the website **Postconditions:** User successfully logged out and returned to the visitor homepage
Flow of Events For Main Success Scenario
→ 1.) User clicks the "Logout" button ← 2.) System updates to reflect a visitor instead of a user → 3.)User is redirected to the visitor homepage

UC#5

Use Case UC#5 - Invite User to League
Related Requirements: ST-1(User registration), ST-6(User creates League), ST-7(League Manager designation), ST-8(League requirements), ST-9(League Invites) **Initiating/Primary Actors:** League Manager, User **Goal:** To invite a user to participate in the league that the League Manager created **Participating Actors:** Database, Browser **Preconditions:** League Manager invites a user that is not already in their private league **Postconditions:** User has been successfully added to the private league. The dashboard updates to reflect the league addition.
Flow of Events For Main Success Scenario
→ 1.) League Manager clicks the “Add Users” tab → 2.) League Manager adds valid email addresses for automated invites to be sent out → 3.) Users respond to the invite by clicking the hyperlink in the email ← 4.) System will update their addition to a new private league
Flow of Events For Extensions (Alternative Scenarios)
→ 4a) The user is new and will register his/her information to create a new account ← 5a.) System will update their addition to a new private league

UC#6

Use Case UC#6 - View Market Statistics
Related Requirements: ST-14 (View market statistics page), OSR-5 (Homepage) **Initiating/Primary Actors:** User, Guest **Goal:** To view the market statistics page **Participating Actors:** Finance API, Browser **Preconditions:** User/Guest is on any page of the website. **Postconditions:** User/Guest is on the market statistics page (homepage).
Flow of Events For Main Success Scenario
→ User clicks the “Home” button ← System pulls the current stock trends from the Finance API. ← System redirects user to the visitor homepage.

UC#7

Use Case UC#7 - Buying a Stock
Related Requirements: ST-18(Accessing a Transaction Page and performing a transaction) **Initiating/Primary Actors:** User **Goal:** To buy a stock for a league. **Participating Actors:** Database, Browser, Finance API **Preconditions:** User is on the League Dashboard page. **Postconditions:** User has bought the desired stock and the Database reflects the purchase.
Flow of Events For Main Success Scenario
→ 1.)User clicks the specific league to buy a stock for. ← 2.)System redirects the user to the individual league page of the league that was clicked. → 3.)User clicks the buy button. ← 4.)System redirects the user to individual buy page for that league and renders a buy form.. → 5.)User enters the stock and amount of shares desired for purchase. ← 6.) System pulls from the Finance API to display the price of each share and total price of the amount of desired shares. → 7.)User submits the information needed to buy a stock and can afford the stock. ← 8.)System inputs the purchase into the Database,subtracts from the user's current balance. ← 9.)System sends the user a transaction complete message.
Flow of Events For Extensions (Alternative Scenarios)
→ 7a.)User submits the information needed to buy a stock, but does not have enough money to afford the stock. ← 8a.)System sends the user a transaction failed message.

UC#8

Use Case UC#8 - Selling a Stock
Related Requirements: ST-18(Accessing a Transaction Page and performing a transaction), OSR-14 **Initiating/Primary Actors:** User **Goal:** To sell a stock from a league. **Participating Actors:** Database, Browser, Finance API **Preconditions:** User is on the League Dashboard page. **Postconditions:** User has sold the desired stock and the Database reflects the sale.
Flow of Events For Main Success Scenario
→ 1.)User clicks a specific league to sell a stock for. ← 2.)System redirects the user to the individual league page of the league that was clicked. → 3.)User clicks the sell button. ← 4.)System retrieves user's owned stock information from Database. ← 5.)System redirects the user to individual sell page for that league. → 6.)User clicks the sell button that appears next to each of the user's owned stocks for the desired stock. ← 7.) System retrieves the stock information that the user is trying to sell from the Finance API. ← 8.) System prompts the user with a sell form that includes the individual share value of the stock. → 9.) User fills out the form and submits. ← 10.)System inputs the sale into the Database, adds to the user's current balance, deletes the stock from the individual sell page. ← 11.)System sends the user a transaction complete message.

UC#9

Use Case UC#9 - Viewing Dashboard and League Leaderboards
Related Requirements: ST-13 (accessing league page and statistics), ST-16 (access specific private league leaderboards), OSR-7, OSR-9 **Initiating/Primary Actors:** User **Goal:** To find league ranking information **Participating Actors:** Database, Browser **Preconditions:** User is on any page of the website. **Postconditions:** User is successfully presented information on current league rankings.
Flow of Events For Main Success Scenario
→ 1.) User clicks the “Dashboard” button. ← 2.) System redirects the user to the Dashboard page. ← 3.) System pulls information from the Database about the user’s rank and buying power in each league. → 4.) User clicks the universal league. ← 5.) System redirects the user to the universal league individual page. ← 6.) System pulls leadership information from the Database about the universal league.
Flow of Events For Extensions (Alternative Scenarios)
→ 4a.) User clicks a private league. ← 5a.) System redirects the user to an individual private league page. ← 6a.) System pulls leadership information from the Database about the private league.

UC#10

Use Case UC#10 - Accept/Decline Invitation
Related Requirements: ST-6 (can receives invites by a league manager), ST-9 (manager can send invites to users), OSR-19 (profile page) **Initiating/Primary Actors:** User **Goal:** To accept or decline an invitation from a league manager **Participating Actors:** Database, Browser **Preconditions:** User is viewing his profile page. **Postconditions:** User may be entered into the given league depending on the choice made, and the database reflects the league changes made.
Flow of Events For Main Success Scenario
→ 1.) User clicks the "Accept" button corresponding to the league invite that he wants to accept. ← 2.) System reflects the addition of the user to the specified league in the database. → 3.) System removes the invite from the user's profile page and notifies the user that their addition to the league was successful.
Flow of Events For Extensions (Alternative Scenarios)
← 1a.) User clicks the "Decline" button corresponding to the league invite that he wants to reject. → 2a..) System removes the invite from the user's profile page and notifies the user that the league invite was successfully declined.

UC#11

Use Case UC#11 - Search Stock
Related Requirements: ST-18 (Using pages that conduct transactions), ST-19 (buy stock), OSR-11 (Search widget) **Initiating/Primary Actors:** Player **Goal:** To search for a stock by name or symbol to purchase **Participating Actors:** Finance API, Browser **Preconditions:** Player is viewing the buy page for some league. **Postconditions:** None
Flow of Events For Main Success Scenario
→ 1.) Player enters in their desired search terms into the search widget text box and presses search. ← 2.) System takes the search terms and pulls any stocks that match by name or symbol from the Finance API. ← 3.) System updates the player's search widget in the buy page with the matching stocks.
Flow of Events For Extensions (Alternative Scenarios)
← 2a.) System does not find any stocks that match by name or symbol from the Finance API. ← 3a.) System updates the player's search widget in the buy page with an error specifying the issue.

UC#12

Use Case UC#12 - View Instructions/FAQ
Related Requirements: ST-3, ST-4 (Can view an Instructions/FAQ page), ST-24(glossary), ST-25 (investing guide), OSR-15 (Titan Trading Lore), OSR-16 (FAQ answers), OSR-18 (Instruction/FAQ Page) **Initiating/Primary Actors:** Guest, User **Goal:** To view the Instructions/FAQ page **Participating Actors:** Browser **Preconditions:** Player is on any page of the website. **Postconditions:** Player is viewing the Instructions/FAQ page.
Flow of Events For Main Success Scenario
→ 1.) Guest/User hovers over the “About” button. ← 2.) System displays a dropdown menu with several pages, including the instruction and FAQ pages. → 3.) Guest/User clicks the chosen page. ← 4.) System redirects the user to the specified page.

UC#14

Use Case UC#14 - Buying and Competing Against an AI
Related Requirements: ST-32 (interacting with an AI) **Initiating/Primary Actors:** User **Goal:** To buy and link an AI to a private league. **Participating Actors:** AI, Browser, Database **Preconditions:** Player is on any page of the website. **Postconditions:** Player has bought an AI and linked it with a private league.
Flow of Events For Main Success Scenario
→ 1.) Player clicks on the TitanCoins Store tab in the navbar. ← 2.) System redirects the player to the store. → 3.) Player clicks on an AI (easy, medium, or hard) to buy from the store. ← 4.) System checks if the user has enough TitanCoins and if so prompts the user for the league name the AI should compete in. → 5.) Player enters the league name and confirms the purchase ← 6.) System checks if the buyer is a member of the league name given and if so creates an AI of selected the difficulty and inserts it into the league on the database.
Flow of Events For Extensions (Alternative Scenarios)
← 4a.) System checks if the player has enough TitanCoins and if not prompts the Player to buy more TitanCoins ← 6b.) System checks if the buyer is a member of the league name given and if not displays an error message to the Player.

UC#15

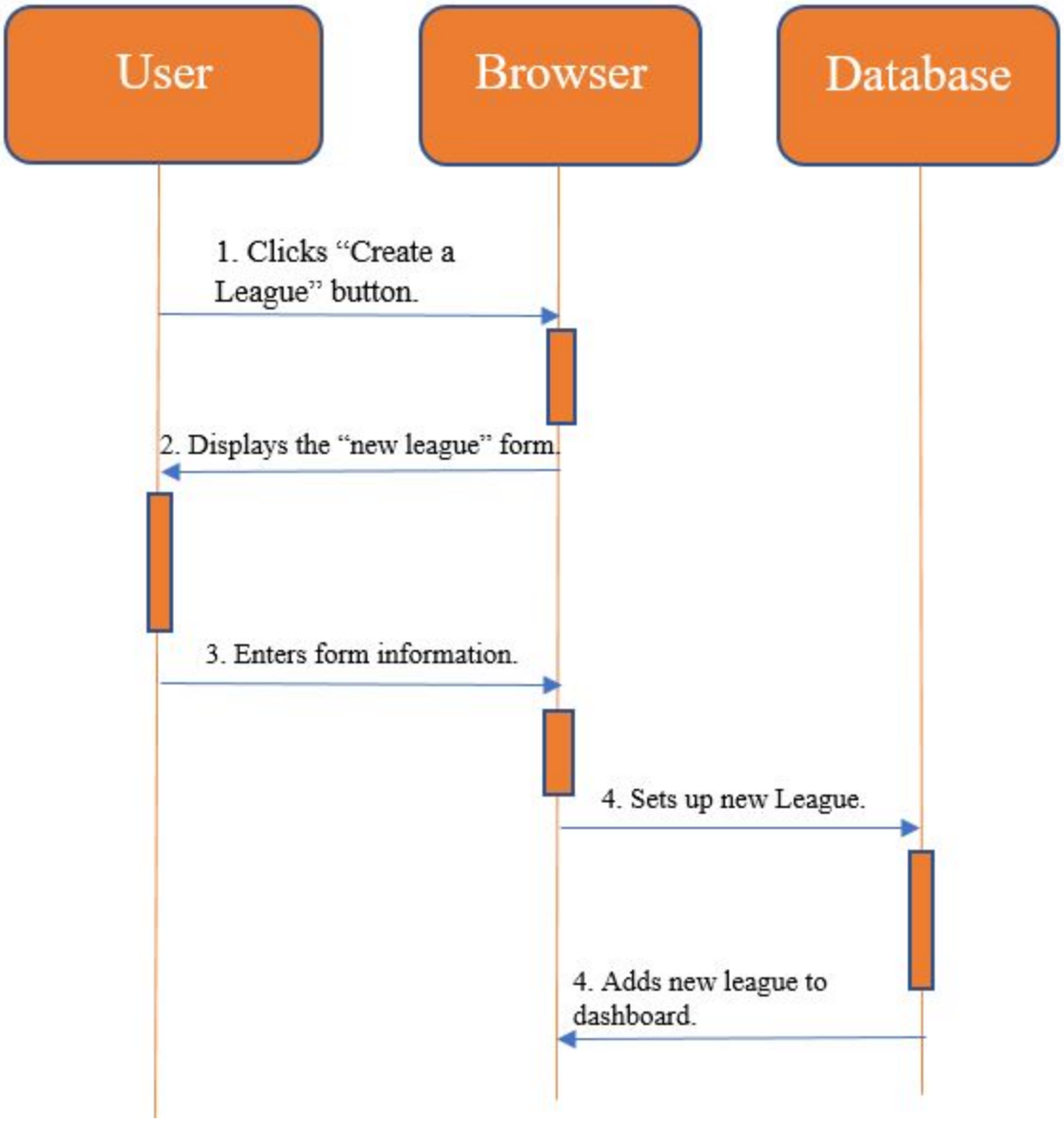
Use Case UC#15 - Purchasing and Link Text Compatibility
Related Requirements: ST-33 (purchasing and linking text compatibility) **Initiating/Primary Actors:** User **Goal:** To buy and link an AI to a private league. **Participating Actors:** Browser, Database, Twilio Text API **Preconditions:** Player is on any page of the website. **Postconditions:** Player has bought an Text compatibility and linked it with a phone number.
Flow of Events For Main Success Scenario
→ 1.) User clicks on the TitanCoins Store tab in the navbar. ← 2.) System redirects the player to the store. → 3.) User clicks on Text Compatibility to buy from the store. ← 4.) System checks if the player has enough TitanCoins and if so prompts the User for a phone number to link the account with. → 5.) Player enters a phone number and confirms purchase. ← 6.) System uses Twilio API to send a greeting message along with instructions on how to use text compatibility to the given phone number and updates the the phone number User attribute in the database.
Flow of Events For Extensions (Alternative Scenarios)
← 4a.) System checks if the player has enough TitanCoins and if not prompts the Player to buy more TitanCoins.

UC#16

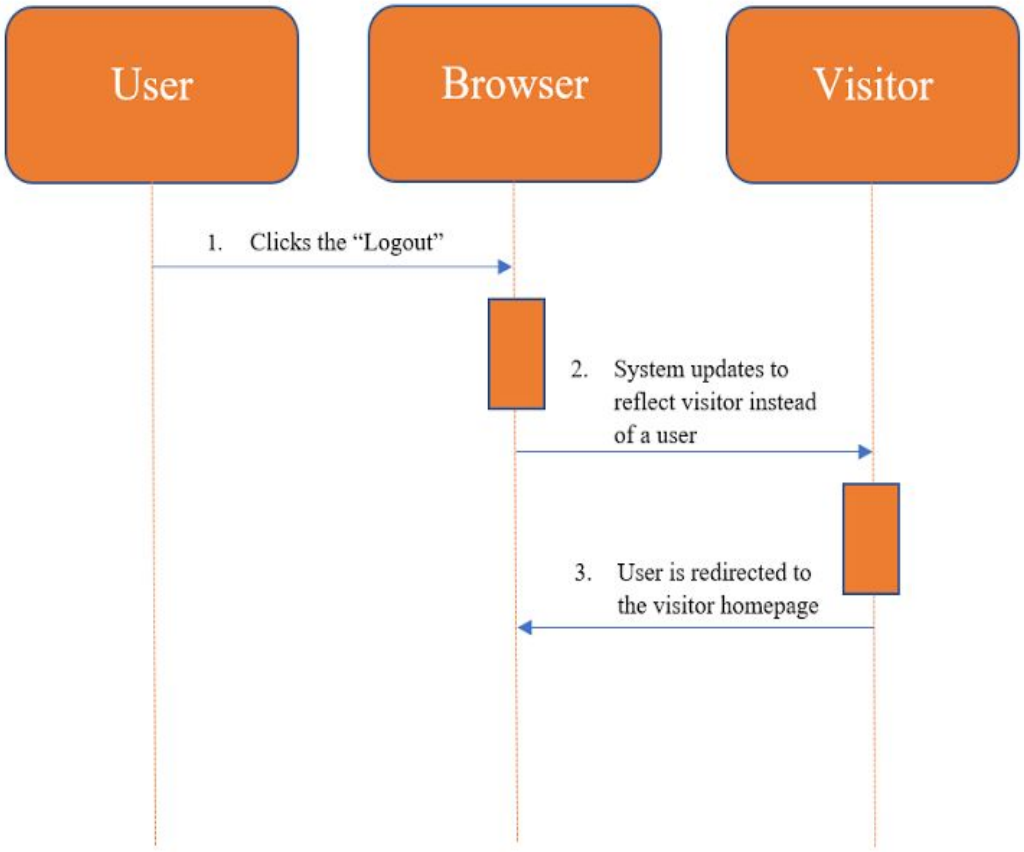
Use Case UC#16 - Buying a stock using Text Compatibility
Related Requirements: ST-33 (transactions using text compatibility) **Initiating/Primary Actors:** User **Goal:** To buy a stock using Text Compatibility **Participating Actors:** Browser, Database, Player, Twilio API **Preconditions:** User has a phone that is linked with Text Compatibility **Postconditions:** Player has bought a stock using Text Compatibility.
Flow of Events For Main Success Scenario
→ 1.) Player texts the Titan Trading Text Compatibility number with a stock order. ← 2.) System uses Twilio API to obtain the text message and send back a order is pending message. ← 3.) System processes the message and checks for the correct syntax. ← 4.) System checks if the given buy information is valid for the user and if so performs the buy operation (UC7 - Steps:5,6,8). ← 5.) System uses Twilio API to send back a transaction complete message with transaction details.
Flow of Events For Extensions (Alternative Scenarios)
← 3a.) System processes the message and uses Twilio API to send an error message back to the User for incorrect syntax. ← 4b.) System checks if the given buy information is valid for the user and if not, uses Twilio API to send back an invalid buy information message.

5.6 System Sequence Diagrams

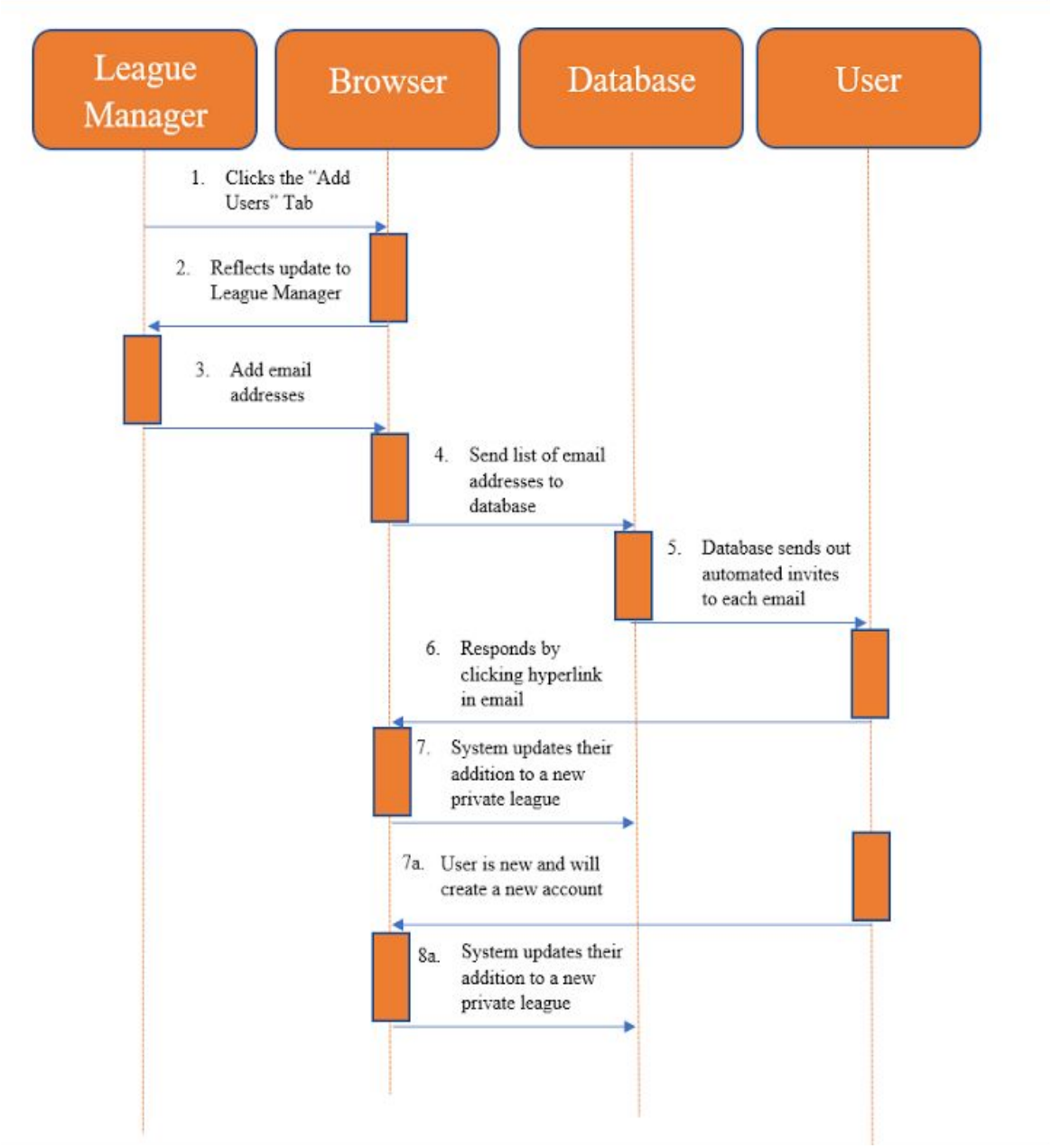
UC1 - Create a League



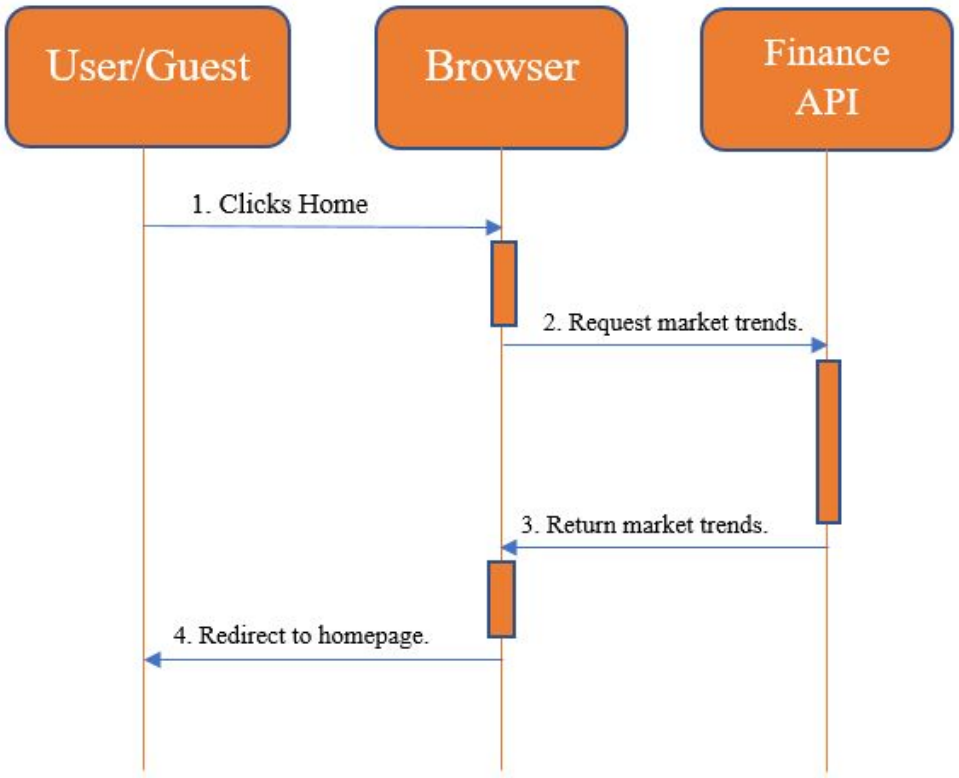
UC4 - Logout


User
Browser
Visitor
1. Clicks the "Logout"
2. System updates to reflect visitor instead of a user
3. User is redirected to the visitor homepage

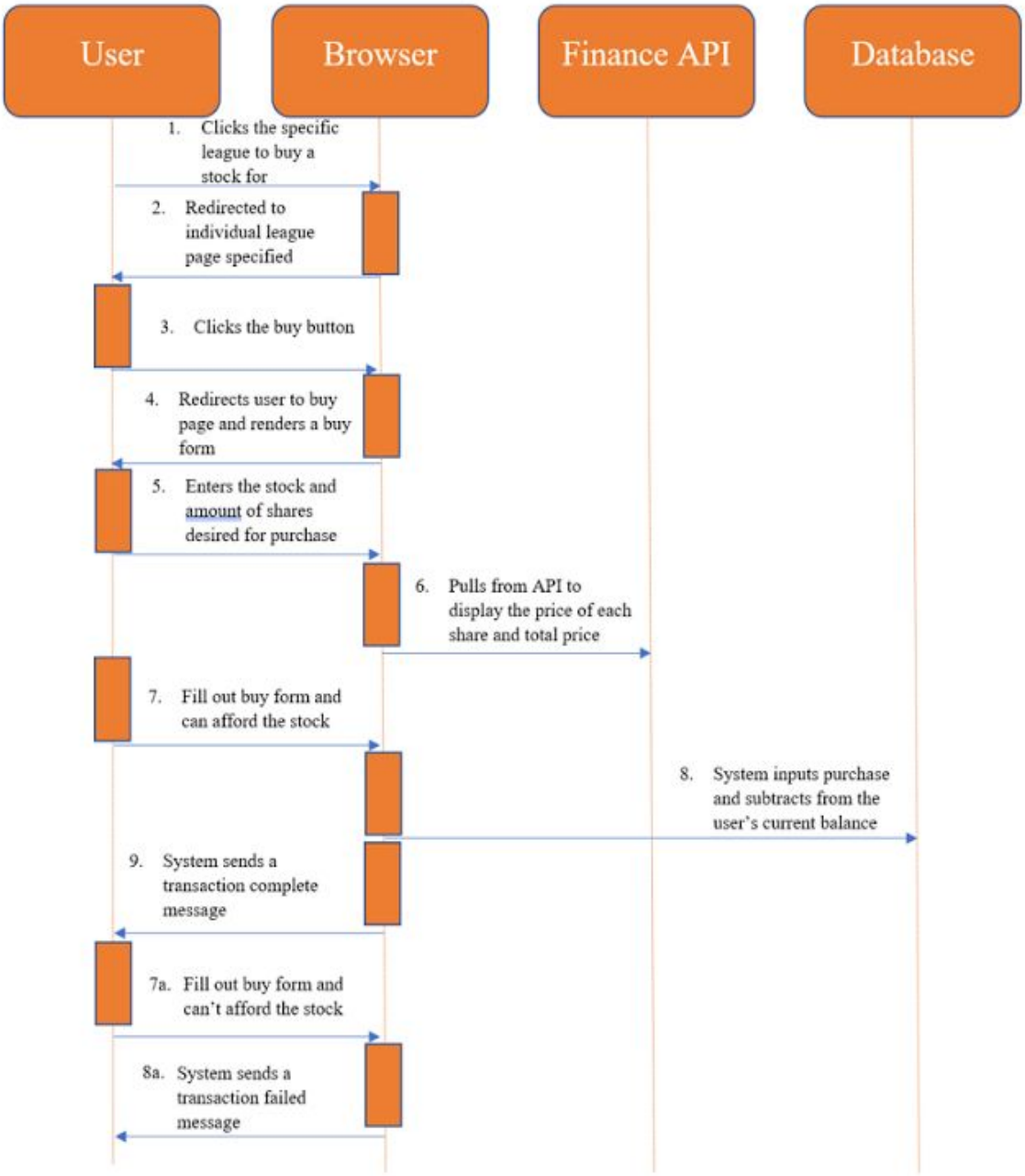
UC5 - Invite User to League



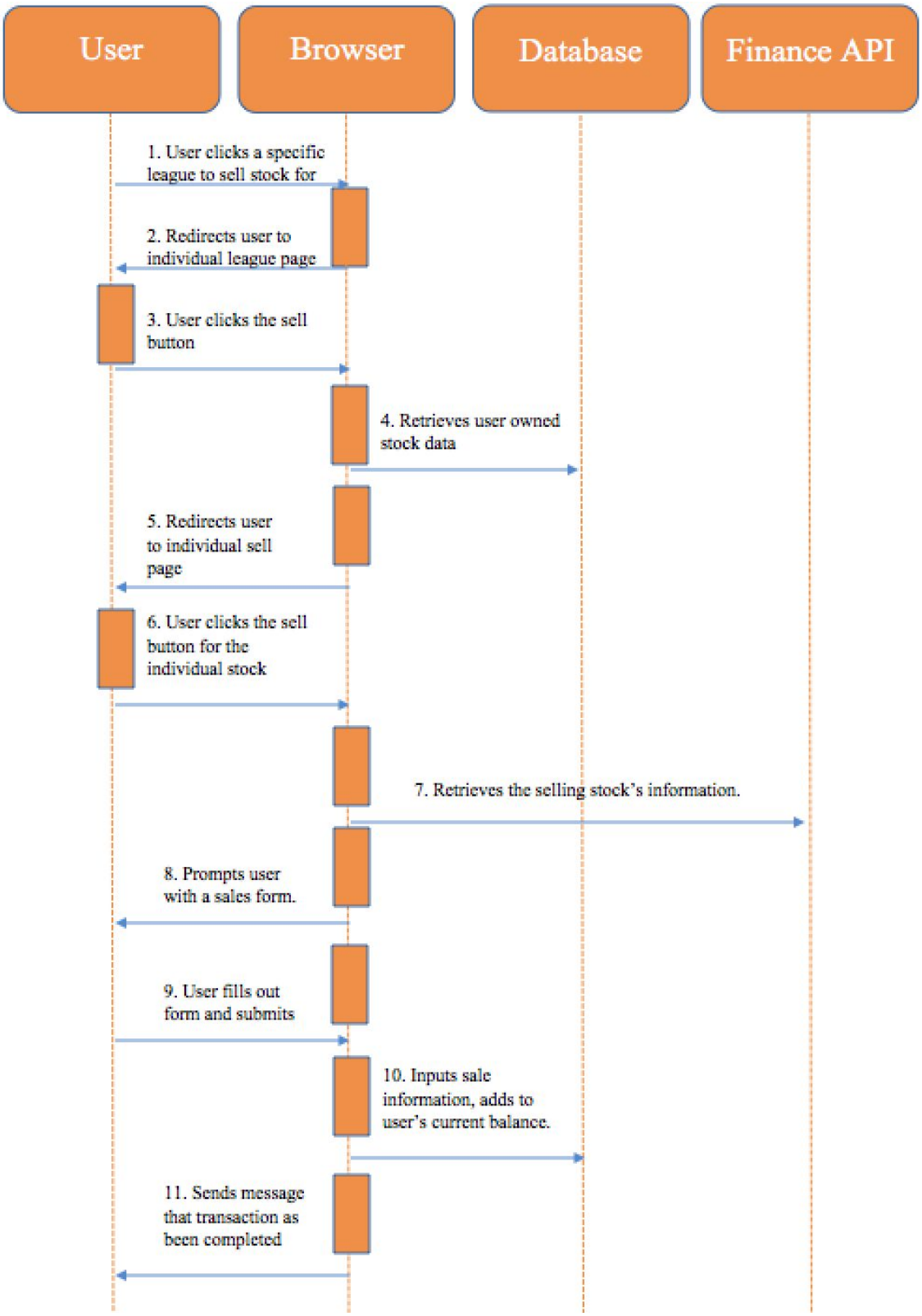
UC6 - View Market Statistics



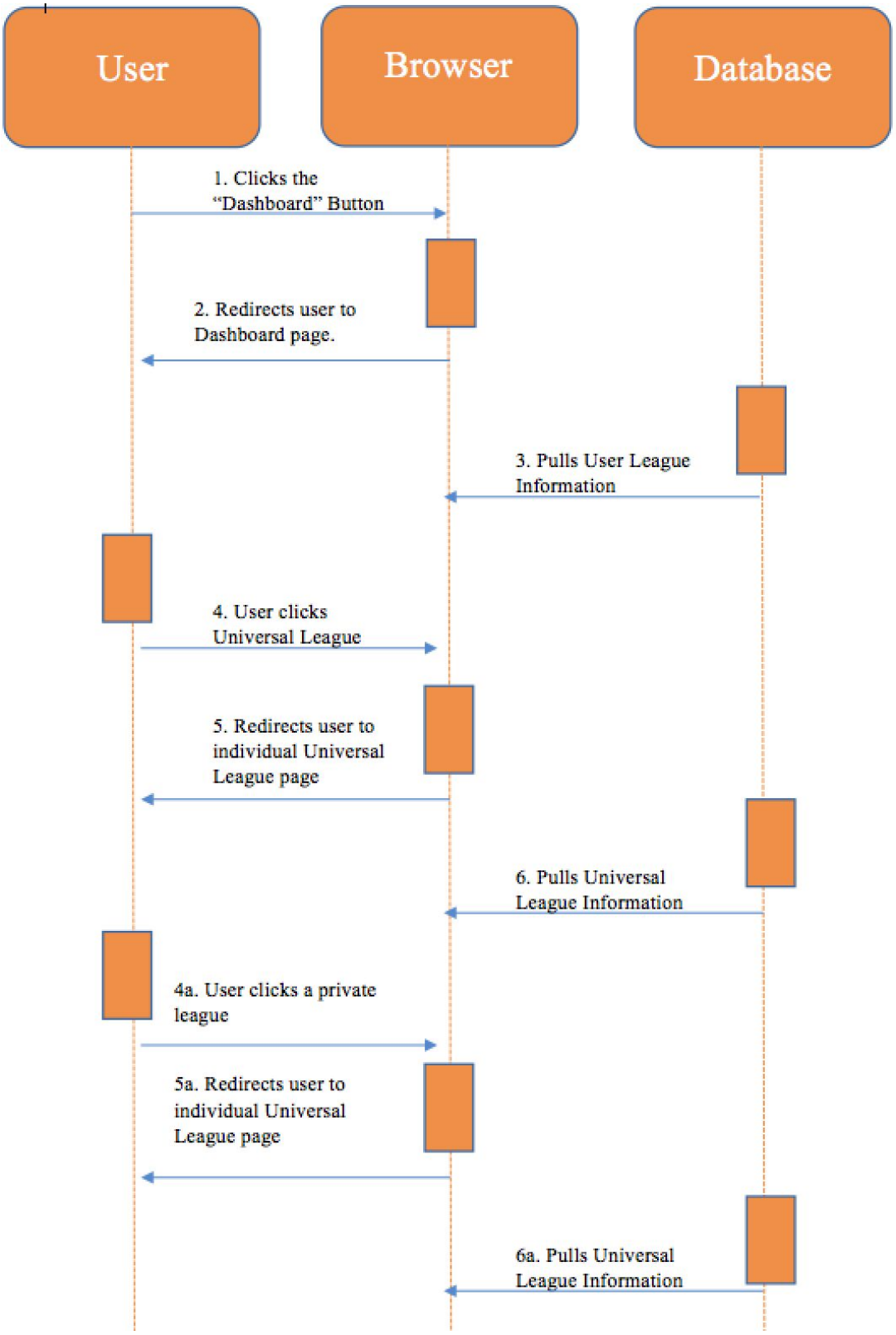
UC7 - Buying a Stock



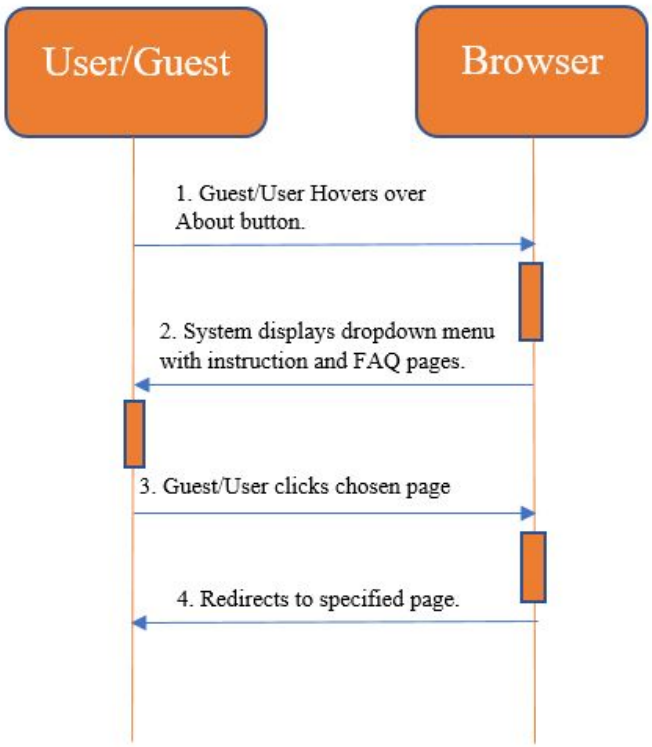
UC8 - Selling a Stock



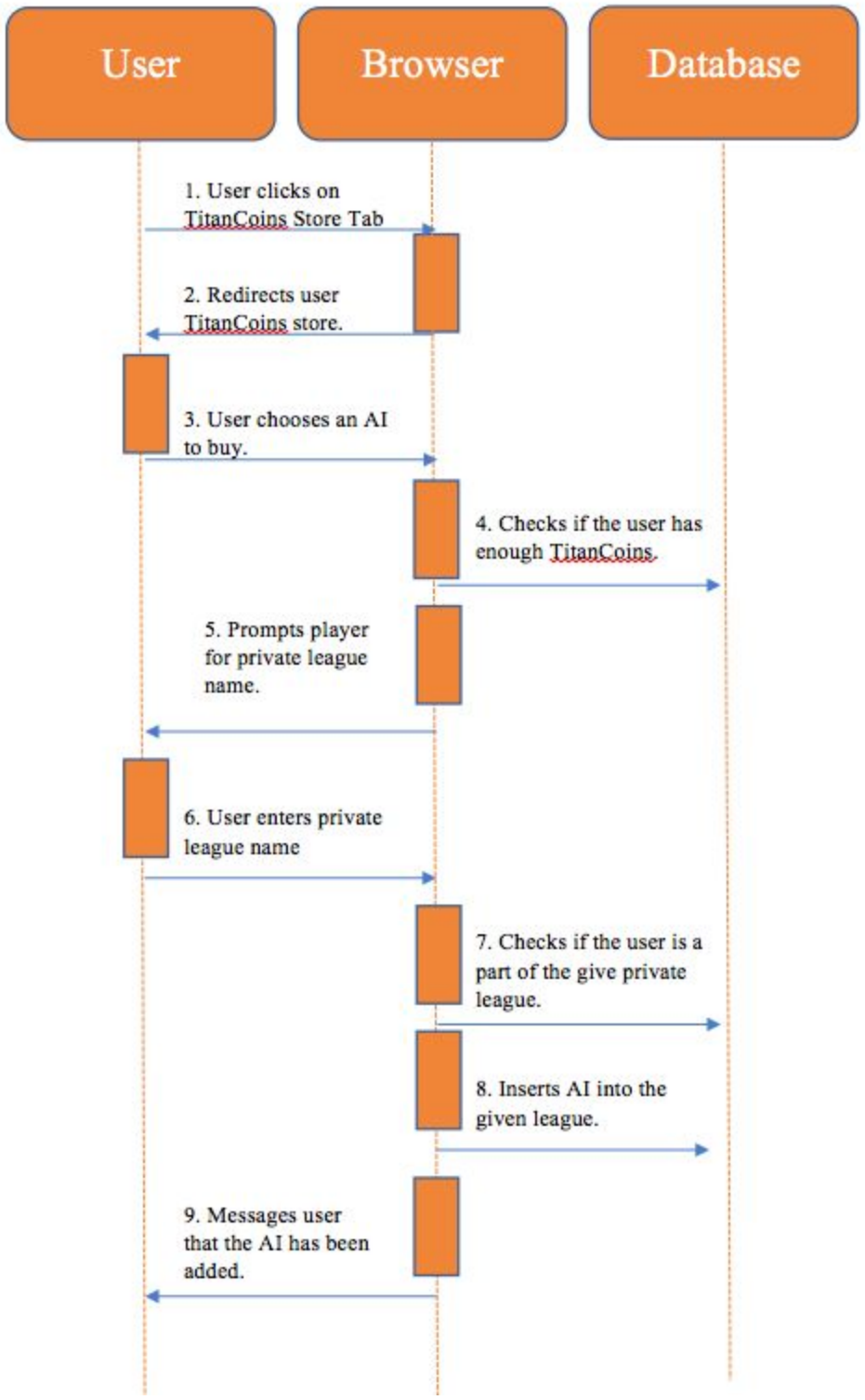
UC9 - View Dashboard and Leaderboards



UC12 - View Instructions/FAQ



UC14 - Buying and Competing Against an AI



6 USER EFFORT ESTIMATION AND USER INTERFACE

6.1 User Interface Design

Titan Trading is an online stock trading fantasy game designed to give users a look into the world of trading on the stock market. Our product is similar to what one would find in a normal fantasy sports application, where users are competing in fantasy leagues alongside their friends in an attempt to come out on top. The trades and portfolios are all fantasy, but the data and numbers are pulled in real time. Each league has its own dashboard where players can view their portfolio, buy/sell stocks, and see how their assets stack up to the rest of the league, and even the world. There are also plenty of resources to get started so that even a novice can jump right in.

At Titan Trading, we consider user effort in determining how intuitive it is to scroll from page to page. A primary concept used in our design is league compartmentalization. As a user, you can have multiple leagues; however, you must "enter" a league in order to access league-specific pages or capabilities, such as buying, selling, or viewing your artificial intelligence.

Our user interface is designed using html, css, and javascript. We use Bootstrap to its styling to our webpages. These style elements help bring the page to life and encourage a dynamic web design.

Pages for the User

Home Page

The home page is the landing page for the Titan Trading website, and it is accessible both to visitors and registered users. It contains the most up-to-date market trends and daily stock market news, and displays the top 10 players within the Universal League. From this page, users can choose to navigate to their league dashboard, the about page, or their profile page.

Global Navigation Pane

The global navigation pane is displayed on all webpages within the Titan Trading domain. It allows for easy navigation across important pages, such as the homepage, the league dashboard, the about page, or the user's profile page. If visitors click on the

“profile page” tab, this will redirect to a page prompting the guest to register for an account.

League Dashboard

The league dashboard’s primary function is to navigate players to a private or Universal league page. The dashboard lists all private leagues that the user is competing in along with the Universal league, which the user can click on to be directed to that individual league page. The dashboard will also display the user’s total worth and rank within each league. In addition, the dashboard also has a “Create New League” button, so users can create a new private league and become a league manager.

If a guest visits the league dashboard, the Universal league will be listed because visitors can be spectators within the Universal league.

Shop

The shop is the page where a user can buy add-ons for their gaming experience. This includes new AIs to compete against and TitanCoins, a virtual currency that can be purchased using real USD. This shop is a simple table setup that displays the products and gives small descriptions to each of them.

About (Dropdown to About Us, FAQ, Instruction, Glossary)

This dropdown menu can be found on the menu bar and gives the user access to the FQ, Instruction guide, and Glossary. All of these pages are there to serve an educational role whether its about Titan Trading, the stock market, or even the words commonly found in the finance world.

About Us

The “About Us” page is meant to serve as more of a static advertisement of Titan Trading. In this page we lay out our Mission Statement, as well as some history of the company. There will also be a short description of all of the hardworking developers who put their time into making Titan Trading what it is today.

FAQ

A FAQ page is a commonly employed tactic in modern websites. Our FAQ page will encompass questions regarding the website itself, trading in general, and how to setup and design your own league.

Glossary
In the financial world there are many terms that are ambiguous and foreign to a novice. Yet these terms are important for anyone who wants to get involved with the market. With this in mind, the Glossary page will serve as the layman's dictionary. Terms and ideas that are common in the finance world will be collected together and given a simple definition, so that even a novice can navigate the world of finance.

Profile Page
Any user on the site will want to keep track of their stats, achievements and settings. The profile page will serve as the central point that any user can change their personal preferences, view their achievement progress, and see their lifetime stats. It is essentially a running count of an individual user and includes other features such as a profile picture and username that other users will be able to see.

Pages for the Player

Individual League Pages
The individual league page is the home base for each privately owned league. Here, a player can view their competitors' current buying power, and decide to either buy or sell stocks. It shows the progress and rank of all players within the league, has options to navigate to a buy or sell page, and displays important league settings. A league manager can choose the day when the competition will end, the starting balance that each player begins with, and if the players will be trading cryptocurrency or normal stocks.

Artificial Intelligence Page
If you have purchased an artificial intelligence, whether it be easy, medium, or hard, you can view this artificial intelligence's transactions, assets, and the data analysis it underwent to make its decision. This page is accessible only from the individual league page, as the artificial intelligence is specific to the league. Although all other player data within your league is kept private, the artificial intelligence's data is public. This enforces our vision of having an education experience, so other players may can learn from the AI's habits.

Buy Page
When a user wishes to buy a stock, they have to be shown statistics and information regarding the company as well as direct information about important numbers such as the volume and price of a given stock. The Buy Page is where a user can not only see all

of this information, but also execute a buy order on a stock. This page is reached through a given league's dashboard and will be separated from the dashboards of other leagues. The interface will indicate to the user which league the transaction is occurring in. This prevents the user from mistakenly transaction in one league when it should have occured in another.

Sell Page

To stay competitive in the market, some players may decide to sell some of their assets, and this function will be available on the Sell Page. This page displays a portfolio of the player's current assets and has an option to submit limit or market sell orders. This page can only be accessed from an individual league page, as each sell order is specific to one league at a time.

Step By Step Walkthrough with Screen Mockups

Use Case #1

Use Case #1: Creating a new league

Flow of events:

Navigation

→ User clicks the "Create a League" button

← System calls up the form for creating a new league. This includes the initial settings that are required to be set.

Data Entry

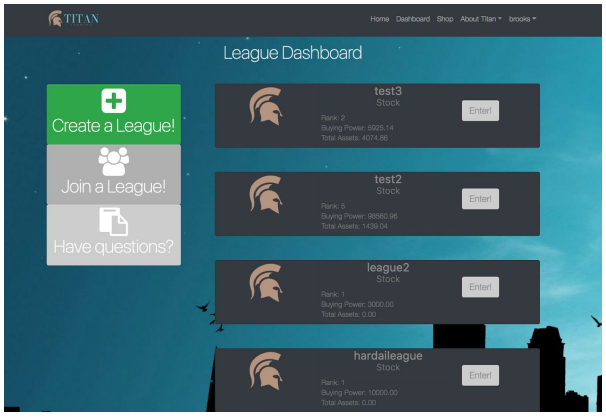
→ User completes the form by putting in their prefered settings.

→ User confirms the form is complete and that all settings are correct.

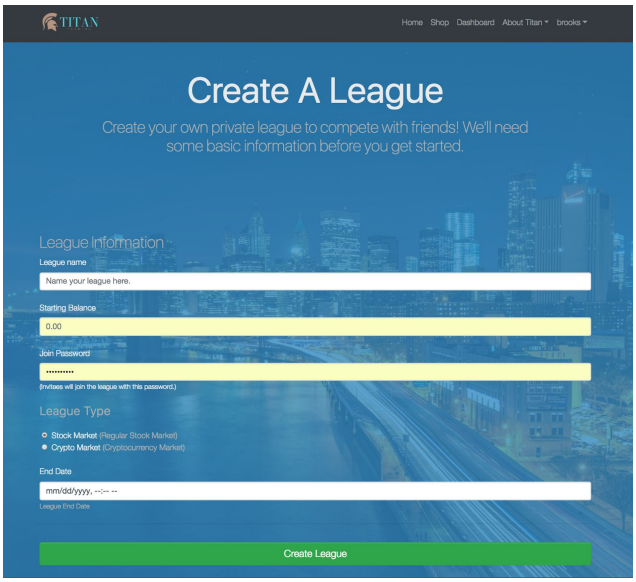
← System takes form data and updates the database with the new league

← System adds the newly created league to the user's dashboard.

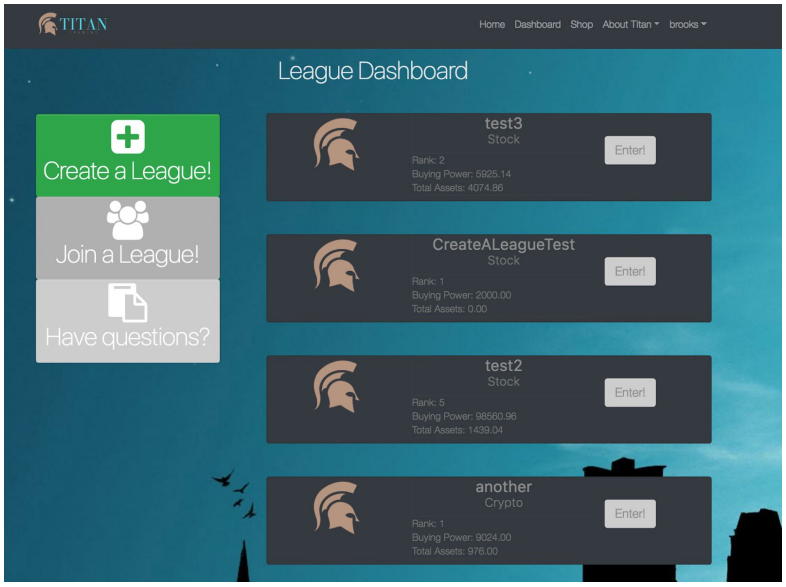
→ User clicks the "Create a League" button



← System calls up the form for creating a new league.
This includes the initial settings that are required to be set.
→ User completes the form by putting in their prefered settings.
→ User confirms the form is complete and that all settings are correct.



← System takes form data and updates the database with the new league
← System adds the newly created league to the user's dashboard.



Use Case #8

Use Case #8: Selling a Stock

Flow of events:

Navigation

→ User clicks on a specific league to sell a stock for.

← Systems redirects the user to the individual league page.

→ User looks through current assets and decides what to sell. User clicks on sell button.

← System redirects the user to individual sell page for that league.

Data Entry

→ User chooses how many assets to sell and fills out form.

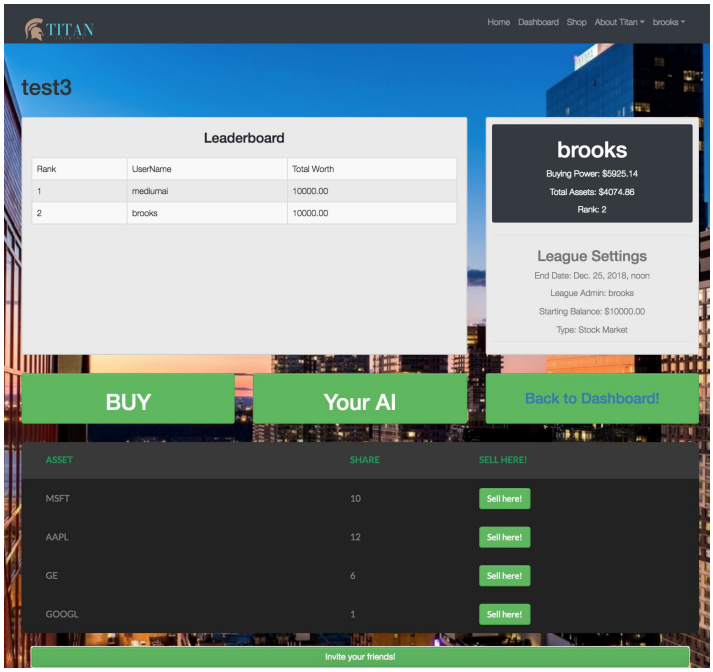
← Systems inputs sale into the Database.

→ User clicks on a specific league to sell a stock for.

← Systems redirects the user to the individual league page.

→ User looks through current assets and decides what to sell.

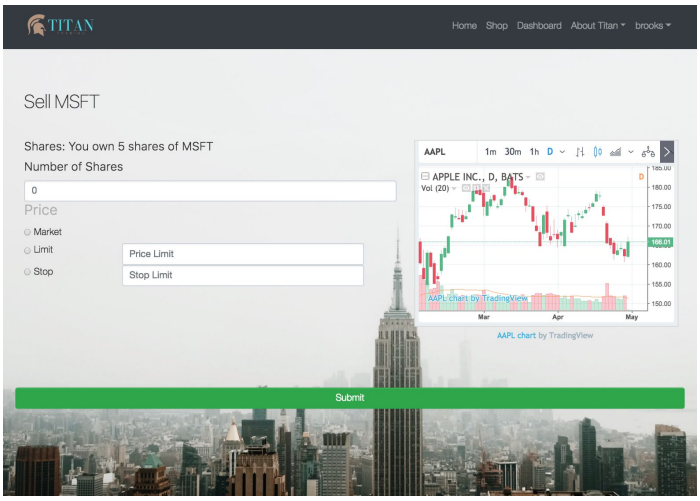
User clicks on sell button.



← System redirects the user to individual sell page for that league.

→ User chooses how many assets to sell and fills out form.

← Systems inputs sale into the Database.



Use Case #14

Use Case #14: Buying and Competing Against an AI

Flow of events:

Navigation

→ User clicks on the TitanShop tab in the navbar.

← System renders the shop page from this request and displays the store.

→ User looks at the shop packages and clicks on the package the user wants to purchase.

Data Entry

← System displays purchase form.

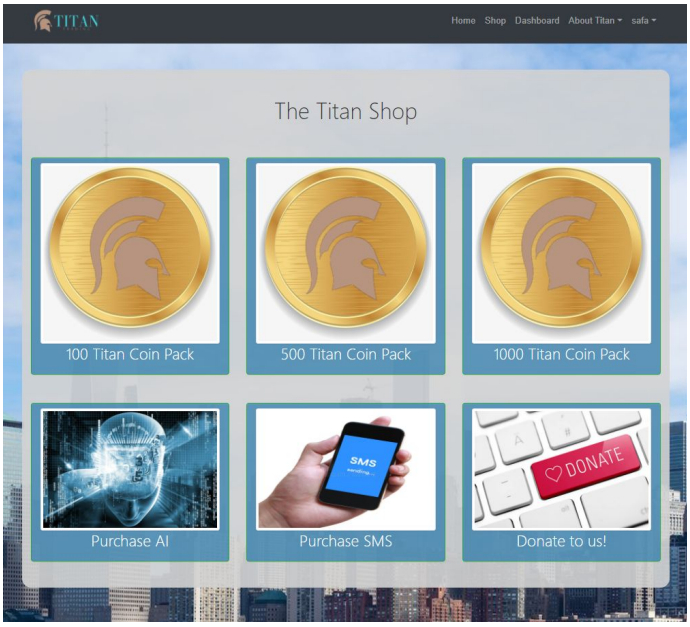
→ User completes form.

← System takes form data and updates the database with the new league

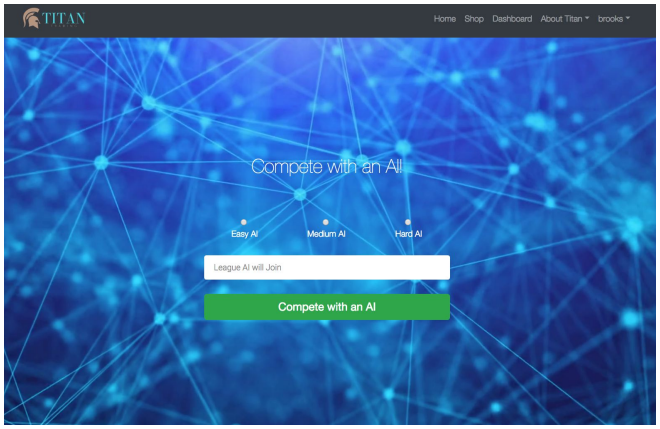
← User views the AI on the individual league page.

→ User clicks on the TitanShop tab in the navbar.

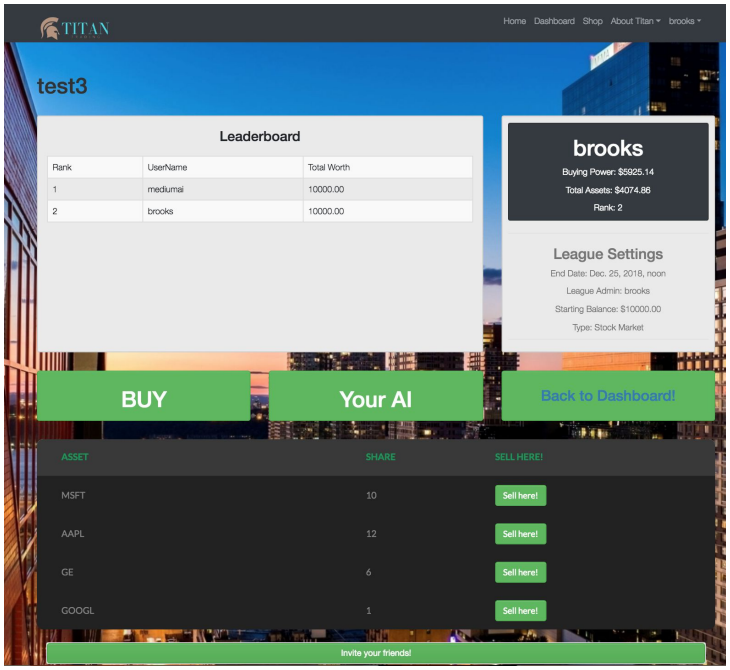
← System renders the shop page from this request and displays the store.



← System displays purchase form.
→ User completes form, choosing whether it wants to purchase an easy, medium, or hard AI.



← System takes form data and updates the database with the new league
→ User views the AI on the individual league page.



Use Case #15

Use Case #15: Purchasing and Link Text Compatibility

Flow of events:

Navigation

→ User clicks on the TitanShop tab in the navbar.

← System renders the shop page from this request and displays the store.

→ User looks at the shop packages and clicks on the package the user wants to purchase.

← System displays purchase form.

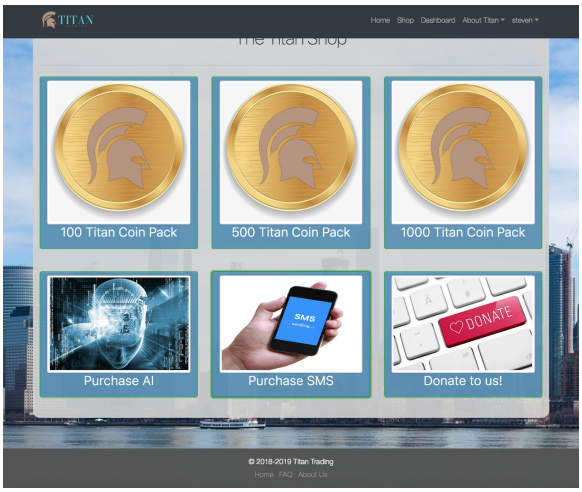
Data Entry

→ User completes form, indicating the amount of money used to purchase text compatibility.

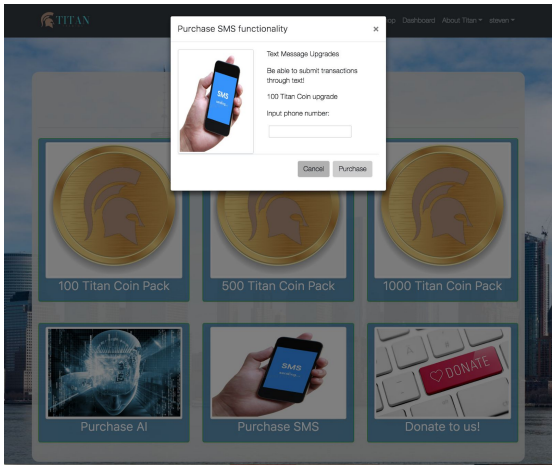
← System checks if the player has enough TitanCoins to purchase.

→ User enters phone number and confirms purchase.

After the user clicks on the shop and text compatibility, the following form will display.



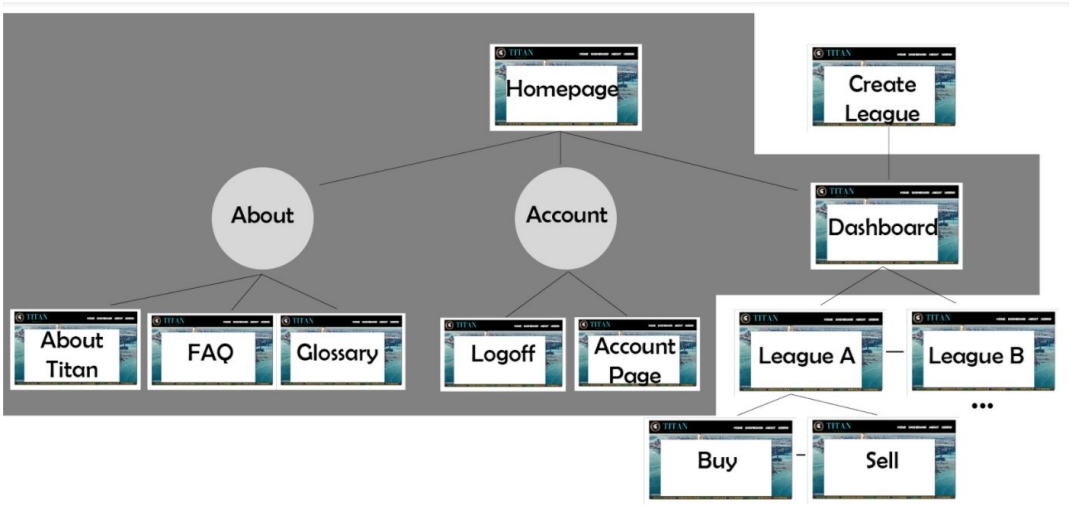
← System displays purchase form.
→ User completes form, indicating the amount of money used to purchase text compatibility.
← System checks if the player has enough TitanCoins to purchase.
→ User enters phone number and confirms purchase.



6.2 User Effort Estimation

Navigation Tree

Below is a navigation tree, showing how each webpage is connected to the others. Lines between web pages indicate it is accessible from page to page by one click. Web pages within the shaded box are also accessible to each other by one click, due to the global navigation pane.



* Shaded region is one click accessible from the menu bar

Effort Estimation for Use Case #1: Creating a new league:
This will take 3 clicks along with filling out the form for their preferred settings.
2 of the clicks are for user interface navigation, while 1 click would be involved with clerical data entry.

Effort Estimation for Creating a new account:
It will take 1 click to get to the account page, another to select create an account. Then the user must enter clerical information for the account details. This is 3 clicks total.

6.3 Calculations

In order to properly assess the effort that is necessary for the user to assert in order to use our product, we will use the points system set out in the Use Case section. This gives us a good way of quantitatively estimating the amount of effort used to do a certain task, and see where we can improve our product to make it easier for the user to use. We will use a multiplicative relationship in order to collapse all the different weighted points and ratings into one final factor representing the total amount of effort a user must put in to use the product.

$$TotalEffortPoints = Unweighted\ Use\ Case\ Points \cdot Technical\ Complexity\ Factor \cdot Environmental\ Complexity\ Factor$$

Unweighted Use Case Points

Actor	Description	Complexity	Weight

League Manager	To manage a league a user needs to be able to access and edit settings along with inviting other users through a Graphical User Interface	Complex	4pts
Private League	For a player to join a private league, they must have a portal to join consisting of a league code and password	Simple	1 pt
Artifical Intelligence Opponent	Artifical Intelligence Opponent interfaces and interacts through the user by being an opponent. User must purchase opponent.	Average	2pts
Guest	A user visiting the website must be able to view certain elements without logging in.	Simple	1pt
Email and Text Message Use	User must be able to receive receipts by email and trade stocks through text message API provided by Twilio.	Complex	3pt
Finance API	Application fetches stock and finance data from Alphavantage through their provided API	Simple	1pt

Technical Complexity Factors

Technical Factor	**Description**	**Complexity**	**Weight**
Database Management	System stores data in a database system and serves content to the end user	3	3
Application Performance	Users expect the application to be responsive and operate fast.	4	4
Ease of use	Users must be able to use the application without much tutorial	2	4
Ease of setup	Since the end user must not have to set up anything, this is not that important.	2	1
User data security	All passwords are hashed and salted when stored	1	3
Multiple end user use	Since this is a web application many users must be able to access at once	2	4
Microtransaction	User must be able to purchase Artificial Intelligence upgrades through a store	2	2

Environmental Complexity Factors

Technical Factor	Description	Complexity	Weight
Web framework experience	Developers must be familiar with web frameworks as that is a lot of the project.	2	2
Motivation	Motivation is important to be able to finish a project.	1	2
Availability	Developers must be able to attend meetings.	1	3
Machine Learning experience	Since machine learning is a big part of the user experience, developers must be able to know how to utilize it.	3	4
Modern Languages	Developers must know development practices in modern languages.	1	1
Mobile user experience	Since user must be able to use text or email messages to interact with the program, developers must be able to properly design a mobile experience.	2	4

Total Effort Weight:

$$UUCP = 4 \cdot 2 + 1 + 2 \cdot 2 + 1 + 3 \cdot 3 + 1 = 24$$

$$TCF = 3 \cdot 3 + 4 \cdot 4 + 2 \cdot 4 + 2 + 3 + 2 \cdot 4 + 2 \cdot 2 = 50$$

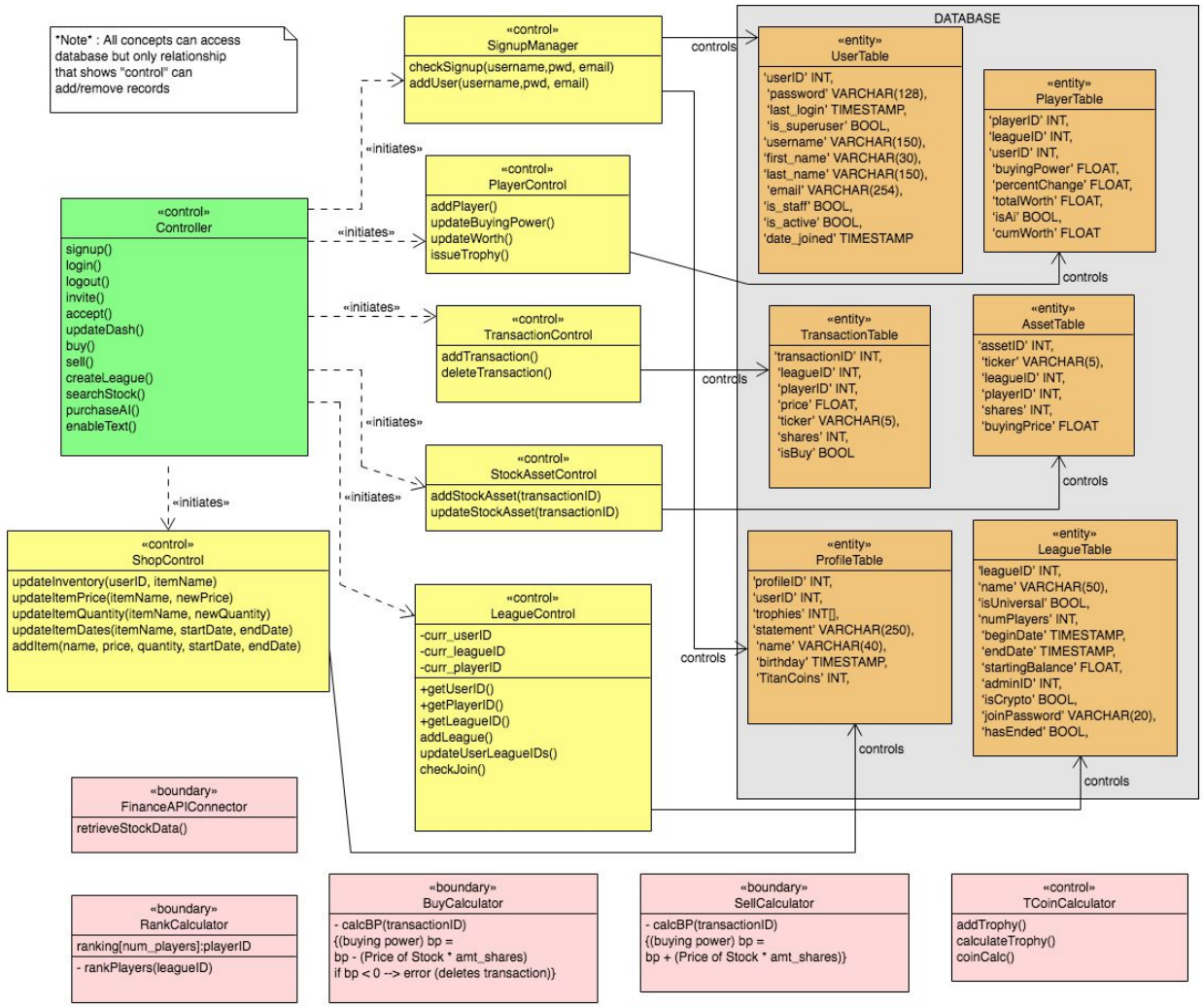
$$ECF = 2 \cdot 2 + 1 \cdot 2 + 1 \cdot 3 + 3 \cdot 4 + 1 \cdot 1 + 2 \cdot 4 = 30$$

$$Total = 24 + 50 + 30 = 104$$

7 DOMAIN ANALYSIS

7.1 Domain Model

7.1.1 Domain Model - Diagram



7.1.2 Concept Definitions and Responsibilities

Responsibility	Type	Concept Name
Delegates work to other concepts, logically groups use cases	D	Controller
Stores user account information:{ userID,username, password, email, leagueID1, leagueID2, leagueID3, leagueID4}	K	User Table
Verify whether or not the username and password are valid by the user.	D	LoginManager
Verifies whether or not the username or email is taken when attempting to	D	SignupManager

create an account.		
Adds a new user record with the given username, password, and email.	D	SignupManager
Orders the totalworth of each player in a given league and lists them in order(stock assets + buying power)	D	RankingCalculator
Stores league setting information:{settingID, leagueID, begin date, starting balance, end date, admin,crypto or not, joinPassword}	K	Setting Table
Stores player information:{playerID, userID, buying power, percent change, total worth}	K	Player Table
Adds a new player record with the given userID, buying power, percent change, total worth.	D	PlayerControl
Updates player record buying power	D	PlayerControl
Multiplies player's stocks by the amount of shares for each and sums the total.	D	PlayerControl
Stores league information:{leagueID, playerIDs, settingID, joinPassword}	K	League Table
Adds a new setting record with the given league name, begin date, ending date, starting balance, admin, crypto or not.	D	LeagueControl
Adds a league record with the given playerID and settingID	D	LeagueControl
Updates NULL field of leagueid <#>(first thats NULL in user record) with leagueid.	D	LeagueControl
Verifies whether or not the league name and joinpassword are valid by the user.	D	LeagueControl
Keeps track of the current leagueid and playerID.	D	LeagueControl
Retrieves stock information on a given stock name.	D	FinAPIConnector
Stores transaction record information:{transactionID, leagueID, amount of transaction, stock, shares, buy/sell}	K	Transaction Table
Adds a transaction record with given amount of transaction,stock, shares.	D	TransactionControl
Stores stock asset information:{stockassetID, playerID, stock, amount of shares}	K	Asset Table
Adds a new stock asset record with given information stock, amount of shares	D	StockAssetControl
Updates stock asset record or deletes the record if amount of shares = 0	D	StockAssetControl
Multiplies price of stock by amount of shares and subtracts from buying power. If buying power is a negative number, will throw error.	D	BuyCalculator
Multiplies price of stock by amount of shares and adds to buying power.	D	SellCalculator
Stores transaction record information:{profileID, userID, trophies[], statement, name, birthday, listAI[]}	K	ProfileTable
Updates number of TitanCoins and adds trophy to user profile if user reaches winning conditions	D	TCoinCalculator

Updates and adds an entry to the AI table	D	AIControl
Deletes an entry in AI table when AI expires; checks periodically	D	AIControl
Stores AI information: { playerID, AIdifficulty, startdate, expirationdate}	K	AI Table
Updates user's inventory depending on user purchases.	D	ShopControl
Updates store periodically to inform users of daily deals on TitanCoins.	D	ShopControl
Keeps track of shop items and their respective prices {name, price, quantity, start date, end date}	K	ShopTable
Stores information concerning user and player purchases {userID, numTitanCoins, item1, item2, ...}	K	InventoryTable

Concept Name	**Pages Used On**
Controller	N/A
User Table	Login/SignUp
LoginManager	Login Page
SignupManager	Sign Up Page
SignupManager	Sign Up Page
RankingCalculator	Home/Dashboard/Individual League Page
Setting Table	CreateLeague/
Player Table	Create League/Join League
PlayerControl	Create League/Join League
PlayerControl	Dashboard/Individual League/Buy/Sell
PlayerControl	Dashboard/Individual League
League Table	Create League/Join League/Buy/Sell
LeagueControl	Create League
LeagueControl	Create League/Join League
LeagueControl	Create League/Join League
LeagueControl	Join League
LeagueControl	Dashboard/Individual League
FinAPIConnector	Buy/Sell
Transaction Table	Buy/Sell
TransactionControl	Buy/Sell
Asset Table	Buy/Sell
StockAssetControl	Buy/Sell

StockAssetControl	Buy/Sell
BuyCalculator	Buy
SellCalculator	Sell
ProfileTable	Profile/Store
TCoinCalculator	Shop/Profile
AIControl	Shop/Buy/Sell
AI Table	Shop
ShopControl	Shop
ShopTable	Shop
InventoryTable	Shop

7.1.3 Concept Associations

Concept 1	**Association Description**	**Association Name**
Login Manager <-> User Table	Login Manager looks up user in User Table and verifies whether the username and password are valid.	Verification
Signup Manager <-> User Table	Signup Manager looks up whether a username or email has been taken when creating an account. If signup is successful, a user is added to the user table. If not, an unavailable message is returned.	Confirmation
RankingCalculator <-> Player Table	RankingsCalculator takes each player's total worth from Player Table to calculate rankings.	Calculation
Player Control <-> Player Table	Player Control adds a new player record to the Player Table	Creation
Player Control <-> Player Table	Player Control updates the buying power of a player record after a transaction.	Updates
Player Control <-> Asset Table	Player Control multiplies a player's number of stocks by the stock price for all stocks from the Asset Table to calculate the total worth of a player.	Calculation
League Control <-> Setting Table	League Control adds a new setting record to the setting table when a league is created.	Creation
League Control <-> League Table	League Control updates the next NULL league entry in the League Table with a new leagueid associated with the creation of a new league.	Updates
League Control <-> League Table	League Control verifies whether the leagueid and and password inputted by a user is correct by checking the League Table.	Verification
League Control <-> Player Control	League Control tells Player Control to add a player entry when a new user is verified to join the league.	Creation

League Control <-> League Table	League Control updates a League Table entry by adding the playerID returned by Player Control.	Updates
Transaction Control <-> Transaction Table	Transaction Control adds a transaction record to the Transaction Table whenever transactions occur in a league.	Creation
StockAsset Control <-> Asset Table	StockAsset Control adds a new stock asset record to the Asset Table	Creation
StockAsset Control <-> Asset Table	StockAsset Control updates a stock asset record in the Asset Table with a new number of shares or the updates price of a stock. If the number of shares equals 0, then it will delete the entry	Updates
BuyCalculator <-> FinAPIConnector	BuyCalculator multiplies the number of shares by the stock price taken from the FinAPIConnector to to find the total cost of the stock bought.	Calculation
BuyCalculator <-> PlayerTable	BuyCalculator finds the player in the PlayerTable making the buy transaction and subtracts the total cost of the stock bought from the buying power. If the buying power is less than the total cost of the stock bought, it will throw an error.	Updates
SellCalculator <-> Asset Table	SellCalculator verifies that the player has the assets specified in the Asset Table	Verification
SellCalculator <-> FinAPIConnector	BuyCalculator multiplies the number of shares by the stock price taken from the FinAPIConnector to to find the total cost of the stock sold.	Calculation
SellCalculator <-> PlayerTable	SellCalculator finds the player in the PlayerTable making the sell transaction and adds the total cost of the stock sold to the buying power.	Updates
TrophyCalculator <-> ProfileTable	TrophyCalculator adds a trophy to a user's list of trophies when a user wins in a particular league.	Updates
AIControl <-> AITable	AIControl updates an AI record in the AI Table with the playerID, AIdifficulty, star tdate, and expiration date whenever a player buys an AI from the shop.	Updates
ShopControl <-> ShopTable	ShopControl updates ShopTable for item additions, sales, and deals.	Updates
ShopControl <-> InventoryTable	ShopControl adds a “1” to the respective item attribute in the InventoryTable for a user’s tuple (denoted by UserID) whenever the user purchases the item.	Updates

7.1.4 Attribute Definitions

Concept	Attribute	Attribute Description
User Table	UserInformation	Contains userID, username, password, email, league

		IDs
LoginManager	VerificationParameters	Username, Password
SignupManager	VerificationParameters	Username, Email
RankingCalculator	ComparisonParameters	Stock assets, Buying power
Setting Table	SettingInformation	Contains settingID, leagueID, begin date, end date, starting balance, admin, crypto or not, joinpassword
Player Table	PlayerInformation	Contains playerID, userID, buying power, percent change, total worth
PlayerControl	AddPlayerParameters	PlayerID, userID, buying power, percent change, total worth, isPlayer
PlayerControl	UpdatePlayerParameters	PlayerID, userID, buying power, percent change, total worth, isPlayer
PlayerControl	CalculatorParameters	Percent change, total worth
League Table	LeagueInformation	Contains leagueID, playerIDs, settingID, joinPassword
LeagueControl	SettingAdderParameters	League name, begin date, end date, starting balance, admin, crypto or not
League Control	UpdateLeagueParameters	PlayerID, settingID
League Control	JoinChecker Parameters	League name, password
FinAPIConnector	StockName	StockName used to retrieve information from the API
Transaction Table	TransactionInformation	Contains transactionID, leagueID, # of transactions, stock, shares buy/sell
TransactionControl	historyHandle	Maintains and updates the transaction history of players across every league
Asset Table	AssetInformation	Contains stock AssetID, playerID, stock, # of shares
StockAssetControl	AssetAdder Parameters	PlayerID, stock, amount of shares
StockAssetControl	AssetUpdater Parameters	PlayerID, stock, amount of shares
BuyCalculator	BuyCalculator Parameters	PlayerID, buyingpower, stock, amount of shares
SellCalculator	SellCalculator Parameters	PlayerID, buyingpower, stock, amount of shares owned
AIControl	AddAIParameters	UserID, AIdifficulty, startdate, expirationdate

AI Table	AIInformation	Contains UserID, AIdifficulty, startdate, expirationdate
ShopControl	AddInventoryParameters	UserID, numTitanCoins, item1, item2, …
ShopControl	AddShopParameters	Name, price, quantity, start date, end date
Shop Table	ShopInformation	Contains name, price, quantity, start date, end date
Inventory Table	InventoryInformation	Contains userID, numTitanCoins, item1, item2, …

7.1.5 Traceability Matrix

	Domain Concepts																					
Use Case	**PW**	Controller	User Table	Login Manager	Signup Manager	Ranking Calculator	Setting Table	Player Table	Player Control	League Table	League Control	Financial API Connector	Transaction Table	Transaction Control	Asset Table	Stock Asset Control	Buy Calculator	Sell Calculator	AI Control	AI Table	Shop Control	Shop Table	Inventory Table
UC-1	22	x					x		x	x	x												
UC-2	10	x	x		x																		
UC-3	7	x	x	x																			
UC-4	4	x	x	x																			
UC-5	14	x									x												
UC-6	9	x										x											
UC-7	21	x							x			x	x	x	x	x	x		x	x			
UC-8	21	x							x			x	x	x	x	x		x	x	x			
UC-9	32	x						x		x													
UC-10	10	x									x												
UC-11	15	x							x			x											
UC-12	10	x																					
UC-13	8	x				x					x												
UC-14	13	x																	x	x	x	x	x
UC-15	14	x																			x	x	x
UC-16	10	x											x	x	x	x					x	x	x
UC-17	11	x											x	x									
UC-18	6	x																			x	x	x

- In this traceability matrix, we show how our use cases map to our domain concepts. Our first domain concept is a Controller. The controller is the central connection between all activities of our program, thus it is mapped to each use case.

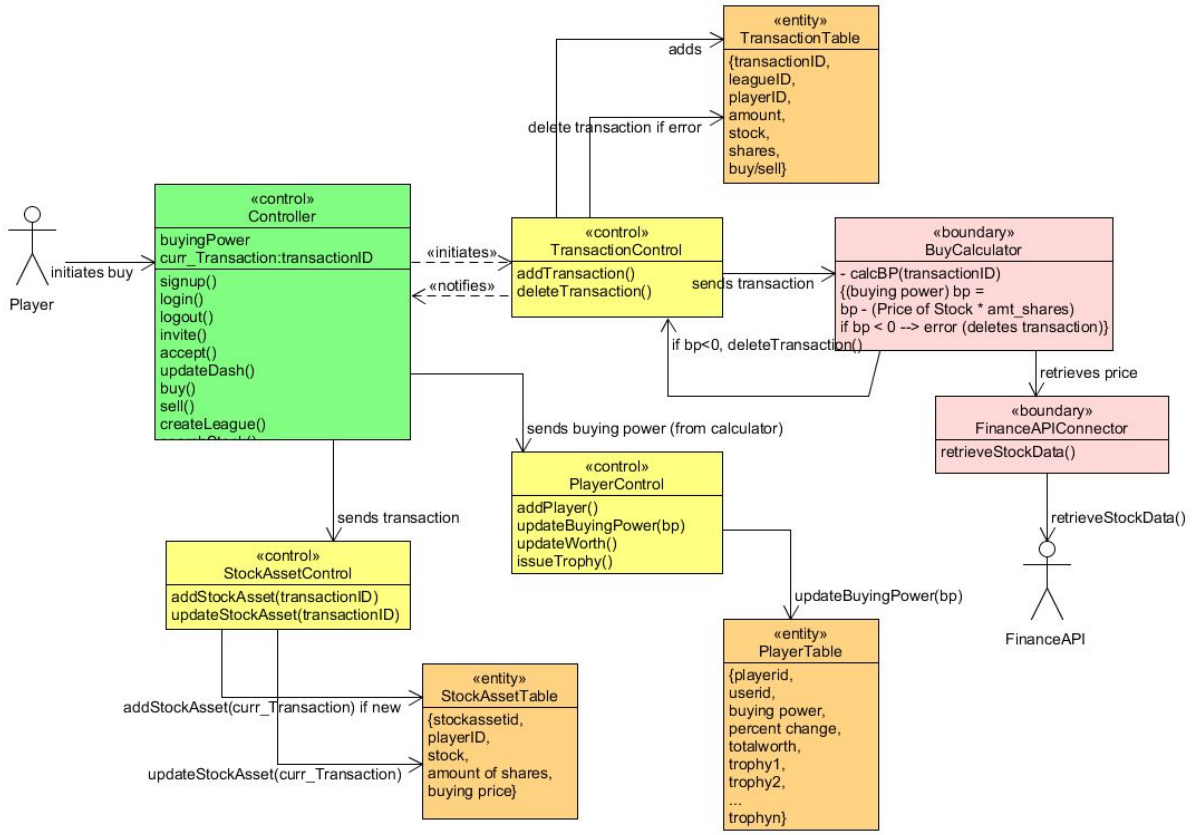
- The next domain concept is the User Table. This is used to keep track of user data, thus it is used in the use cases where account creation, login, and logout are used.
- We also have a login manager and a signup manager, which actually control when accounts are created and used. Login manager is mapped to use cases 3 and 4, which are login and logout. Signup manager is mapped to use case 2, which is sign up. These managers access information from the user table.
- The next domain concept is a Ranking Calculator, which is used to determine how players rank within specific leagues. Depending on their ranking, players can unlock trophies, thus this domain concept is mapped to UC-13.
- Next, we have the setting table, which is used to store information about private leagues. Because of this, it is mapped to UC-1 where private leagues are actually created.
- The next domain concept is a player table. Players are associated to different users and leagues, and the player table is used for UC-9 where the dashboard and leaderboards are viewed. Data about an individual's performance in a league is stored in the player table, thus this domain concept is mapped to UC-9.
- Next, we have the player control domain concept. This serves to control all player activities, thus it is mapped to UC-1,7,8, and 11. UC-7,8,11 are associated with the purchase of stocks and since each purchase is associated to a player, these use cases map to the player control domain concept.
- Next, we have the league table domain concept. The league table stores all the information for the private leagues, thus it is associated with league creation and viewing (UC1 and 9 respectively).
- We also have a league control domain concept because, like the previous table concepts, we want to have a controller associated to handle all the functions of these objects. The league control is mapped to all use cases involving league creation and data, thus it is mapped to UC-1 and UC-5 (create league and invite user to league).
- Our next domain concept is the Financial API Connector. This domain concept is the API we use to get real time stock and cryptocurrency data, thus it is associated with the use cases where these prices and statistics are used (UC-6,7,8,11,16).
- Next we have the Transaction Table and Transaction Control domain concepts. Like our other table-control concept pairs, the table stores information while the control handles the functions. These domain concepts are mapped to UC-7,8,16,17, which are the use cases that handle buying and selling stock/cryptocurrency and handling the receipts associated to the transactions.
- Our next pair of domain concepts are also a table-control pair that handle the actual stock assets themselves. The asset table records all the information for the stocks, while the control handles the functions associated with them, which are buying and selling thus UC- 7,8,16.
- The buy calculator is a domain concept that calculates how much it costs to buy a stock, thus it is mapped to UC-7 buy a stock. The sell calculator calculates how much a player can sell a stock for, thus it is mapped to UC-8 sell a stock.
- Next, we have another table-control pair for our AI. The AI Table stores the information of our AI (AI is like a player) and the control handles the functions associated with it.

These domain concepts are associated with the use cases that handle buying and selling stock, as well as the use case for buying and competing against an AI, thus it is mapped to UC-7,8,14.

- We also have a shop control and shop table domain concept pair, where the shop table stores the information of what is available in our shop and the shop control handles the functions associated with interacting with the shop. These domain concepts are mapped to UC14,15,16,18, which are purchasing items from our shop and connecting your account to PayPal.
- Our final domain concept is the Inventory Table, which stores user IDs and which items from the shop the user has. This domain concept is mapped to the use cases associated with shop purchases (UC-14,15,16,18).

Buy Stock Use Case - UC7

The following is an example of how domain model concepts will interact while executing Use Case 7 - Buying a Stock:



7.2 System Operation Contracts

UC-1 - Create a League

- Preconditions:

    - User is involved in fewer leagues than the league limit
- Postconditions:
    - User is a league manager for the new league. User does not exceed the league limit.

UC-4 - Logout

- Preconditions:
    - User is already logged on to their account on the website
- Postconditions:
    - User successfully logged out and returned to the visitor homepage

UC-5 - Invite User to League

- Preconditions:
    - League Manager invites a user that is not already in their private league
- Postconditions:
    - User has been successfully added to the private league. The dashboard updates to reflect the league addition

UC-6 - View Market Statistics

- Preconditions:
    - User/Guest is on any page of the website
- Postconditions:
    - User/Guest is on the market statistics page (homepage)

UC-7 - Buying a Stock

- Preconditions:
    - User is on the League Dashboard page
- Postconditions:
    - User has bought the desired stock and the Database reflects the purchase

UC-8 - Selling a Stock

- Preconditions:
    - User is on the League Dashboard page
- Postconditions:
    - User has sold the desired stock and the Database reflects the sale

UC-9 - Viewing Dashboard and League Leaderboards

- Preconditions:
    - User is on any page of the website
- Postconditions:
    - User is successfully presented information on current league rankings

UC-10 - Accept/Decline Invitation

- Preconditions:
  - User is viewing his profile page
- Postconditions:
  - User may be entered into the given league depending on the choice made, and the database reflects the league changes made

UC-11 - Search Stock

- Preconditions:
  - Player is viewing the buy page for some league
- Postconditions:
  - None

UC-12 - View Instructions/FAQ

- Preconditions:
  - Player is on any page of the website
- Postconditions:
  - Player is viewing the Instructions/FAQ page

UC-14 - Buying and Competing Against an AI

- Preconditions:
  - Player is on any page of the website.
- Postconditions:
  - Player has bought an AI and linked it with a private league.

UC-15 - Purchasing and Link Text Compatibility

- Preconditions:
  - Player is on any page of the website.
- Postconditions:
  - Player has bought an Text compatibility and linked it with a phone number.

UC-16 - Buying a stock using Text Compatibility

- Preconditions:
  - User has a phone that is linked with Text Compatibility
- Postconditions:
  - Player has bought a stock using Text Compatibility.

7.3 Economic and Mathematical Models

Cryptocurrency versus Stocks

In Titan Trading, there are two league types to which you can join: 1) a public equities league, or 2) a cryptocurrency league. Within our game, you cannot create a league where you

can jointly trade stocks and cryptocurrency. This is due to the fact that stocks and cryptocurrency operate under very different economic models and assumptions, which require different investment strategies.

For the public equities league, we assume that stocks operate under the efficient market hypothesis (EMH). The efficient market hypothesis is an investment theory that states that stocks will *always* trade at their fair value, so that investors will never be able to purchase underpriced stocks or sell stocks at an inflated price. (“Efficient Market Hypothesis - EMH”). In our current market, investors are typically rational and are only willing to pay what an equity is actually worth.

In comparison, for the cryptocurrency league, cryptocurrency does not abide by any economic models or government regulations, since its origins only date back to 2009 (Marr, Bernard). Unlike public equity, cryptocurrencies - like Bitcoin - drastically fluctuate in price and are extremely volatile for a variety of reasons. Cryptocurrency is affected by uncertainty in its store of value, which is the function of an asset’s usefulness with some predictability. In addition, its current market is driven by extreme press about security breaches, rapid positive gains, or illegalization of the currency (Barker, Jonathan Todd) . Because crypto is influenced by its lack of certainty and its inconsistent media perception, cryptocurrency investors trade irrationally, thus contradicting the efficient market hypothesis and causing volatility in the crypto market.

In order to stay consistent with these two economic models, Titan Trading separates its two types of leagues, to accurately portray the separate universes in which these markets exist. This gives Titan Traders the opportunity to change their investment strategies based on if they are trading stocks or cryptocurrencies.

Perfect Competition

In Titan Trading, we assume that all users are playing in a market with perfect competition. This principle states that no single participant has enough resources/power to control the market. (Fontinelle, Amy) In order to apply perfect competition to our fantasy league, the players will follow the following rules:

- Not one player will control the market or industries
- Every individual will have access to the same information as other investors
- One user can represent multiple players, whose assets are separate from the others
- All investors within a league start with the same amount of money so no one person has more power than everyone else
- Insider trading will not be present because stock information is uniform throughout the website

The financial APIs we will be using are representative of current market data. This, however, leads to an inconsistency in our economic model since Titan Traders will be playing in a universe of perfect competition, while the current stock and current cryptocurrency market do not exhibit this concept. However, if Titan Trading did not follow the rules of perfect

competition, each player would have some authority to change the stock data in non-traditional ways, making the fantasy league even more unrealistic from real stock/cryptocurrency trading.

Buy and Sell Orders

All market prices are retrieved from two separate APIs. One API will retrieve public equity market prices, while the second API will retrieve cryptocurrency prices. All information is accessible by every player within Titan Trading.

Once a buy order is submitted, the following mathematical model will be used to calculate the trader's new total worth and buying power.

$$\text{buying power} = \text{current buying power} - (\#\text{ of assets bought}) \times \text{market price of asset}$$
$$\text{total worth} = \text{total worth} + (\#\text{ of assets bought}) \times \text{market price of asset}$$

Similarly to buy orders, sell orders will use the following mathematical model.

$$\text{buying power} = \text{current buying power} + (\#\text{ of assets sold}) \times \text{market price of asset}$$
$$\text{total worth} = \text{total worth} - (\#\text{ of assets sold}) \times \text{market price of asset}$$

Total worth is variable to change based on the current market price of an asset, while buying power will stay constant.

Achievements

Once a user has met the requirements for a certain achievement, they user will be given a reward. Most mathematical models for receiving achievements will be defined like the following example.

$$\text{if } (\#\text{ of participating leagues}) == 3, \text{ get a reward}$$

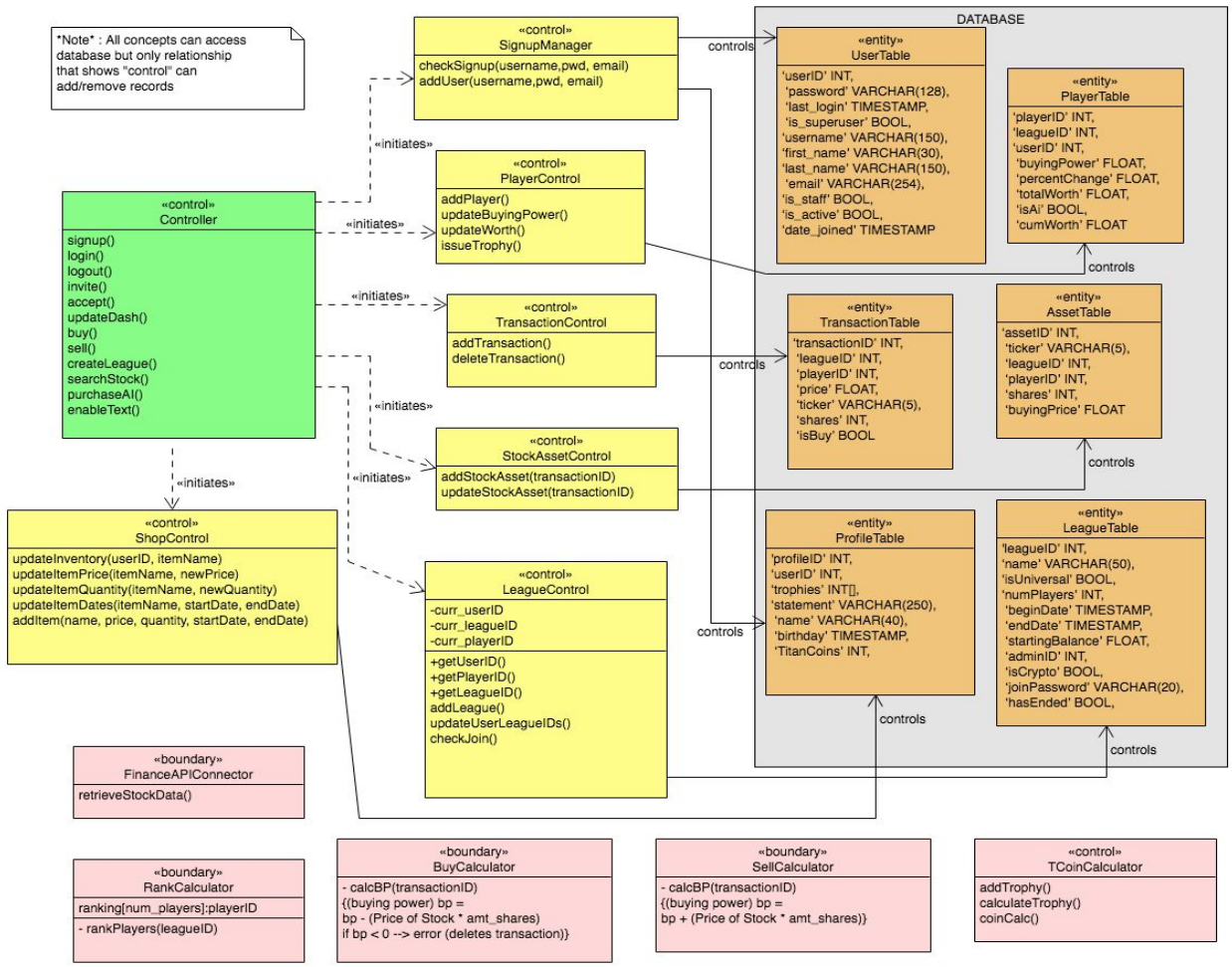
8 SYSTEM INTERACTION DIAGRAMS

8.1 Introduction

Titan Trading is a web app, built with Django and Python and will interface with a PostgreSQL database. The Financial API we will be using is AlphaVantage, which can obtain both stock and cryptocurrency data. Following is an in-depth analysis of use cases and system interaction between domain concepts.

8.2 Domain Model

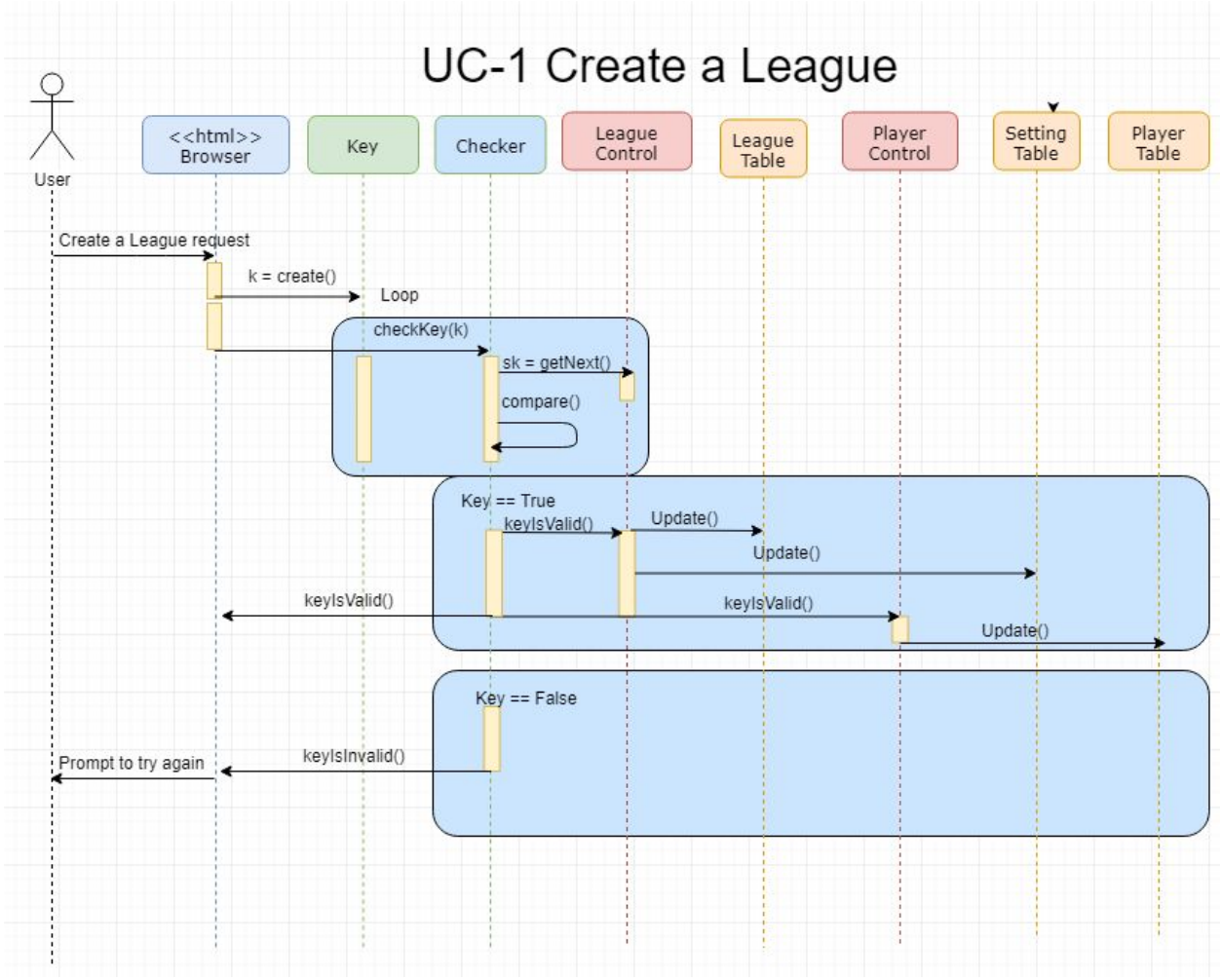
The following is our domain model. The controller is the main interface between the system and the browser with which the user interacts with. The controller allocates responsibilities to the various controls, which respectively update the database.



8.3 System Sequence Diagrams

The following diagrams show the system processes of each fully dressed use case. These diagrams showcase the interaction of our domain concepts and how all user data is managed in a database.

UC-1 Create a League

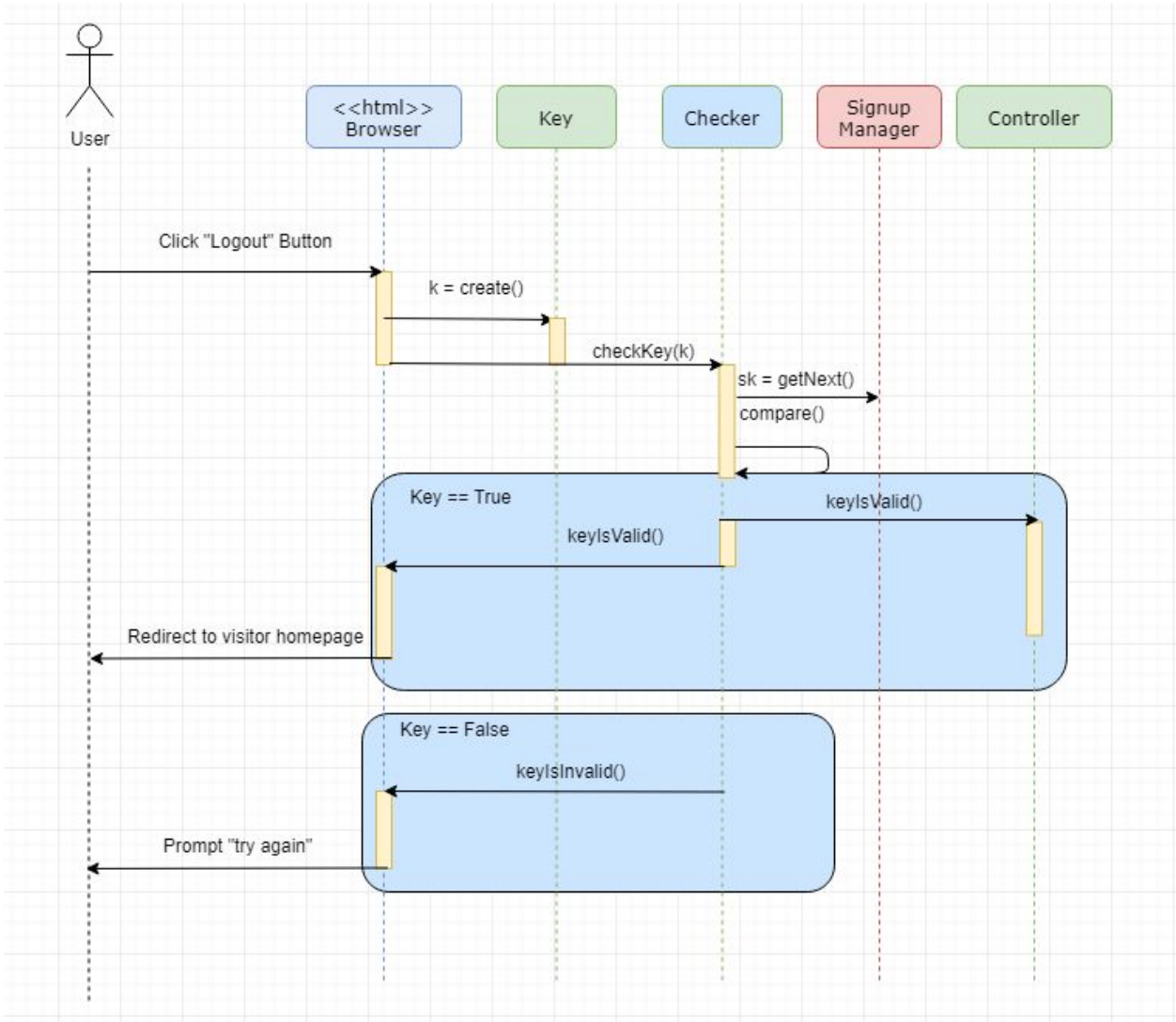


To create a league, a user enters all required data into a form. The controller keeps track of this data and sends it to the League Control and the Player Control. The League Control takes all data associated with the new league and new settings and updates the two league and setting tables in the database. The admin is now the first player, so that data needs to be updated in the player table as well. The player control will create a new player record. This record will now be associated with the prior created league and settings records.

Design Pattern: This use case implements the publisher-designer pattern. When a league is created, the key for the league is checked against the database. If it exists in the database, then

the user is denied the league creation and is requested to try again. Otherwise, the league creation process occurs as stated above.

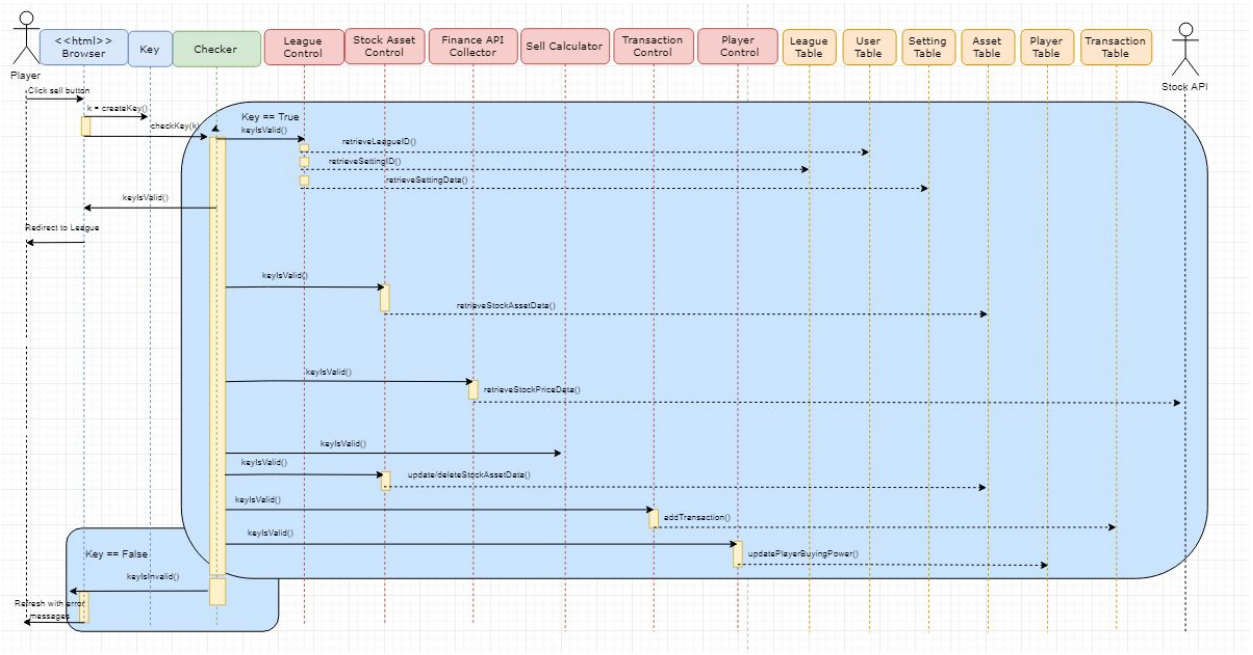
UC-4 - Logout



An essential feature is to distinguish visitors from registered users. Users become visitors by logging out and may proceed to initiate this process by clicking the logout button. The framework of the website allows all user data to be saved in the database when the user logs out and that state will be preserved until the user logs in again.

Design pattern: This implements the publisher-designer pattern. By passing keys to verify whether the user is logged in or logged out, we are able to pass a valid and invalid function to our publishers. By doing this, it will be easier to add more software concepts if we so choose to have additional things occur when logging out happens.

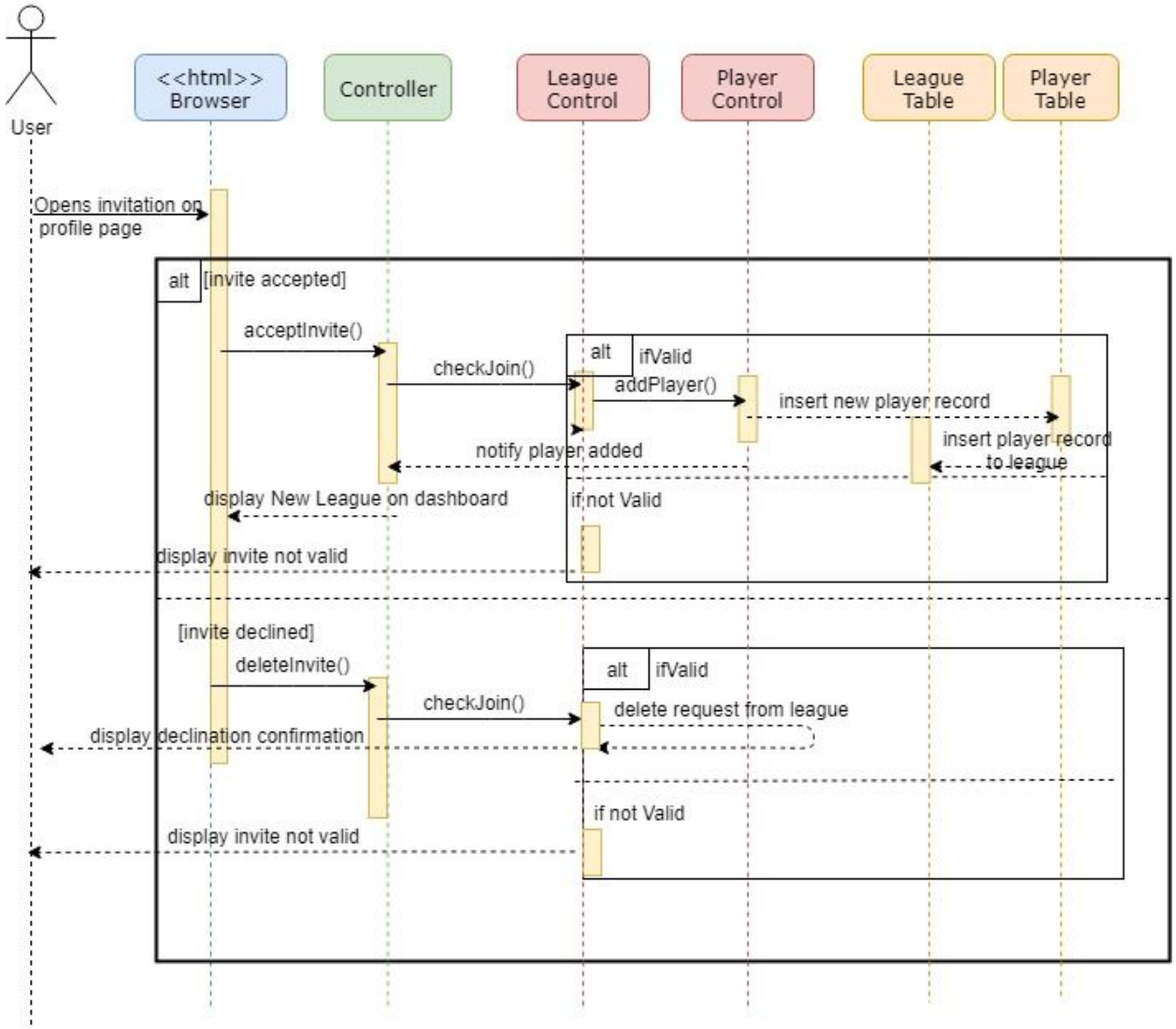
UC-8 - Selling a Stock



The above system sequence diagram shows the interactions of a player with the system to sell a stock. The player will start at the league dashboard page and will have to click a league where the system will redirect them to the individual league page of the league the player clicked. The redirection process requires the system controller to obtain all the necessary information about the individual league from the League Control. After being redirected to the individual league page with the displayed League Data, the player will have to click the sell button to declare that he wants to sell a stock from the current individual league. When the player clicks the sell button, the system controller will communicate to the stock asset control to obtain all the stocks the current player has within the individual league. The player will click a stock to sell and the system will retrieve the current market price of the stock from the Stock API and prompt the player with a sales form. The player will fill out the sales form with necessary transaction information and the system controller will have to communicate with the Sell Calculator to calculate the sales transaction, Stock Asset Control to updateStockAsset, Transaction Control to create a transaction record, and the player control to update the player's buying power.

Design Pattern: This use case uses a publish-subscribe pattern. The controller publishes the verification request to all the necessary controls, which check the validity of the transaction. If any subscriber finds evidence that violates the integrity of the game, then the transaction will be rejected.

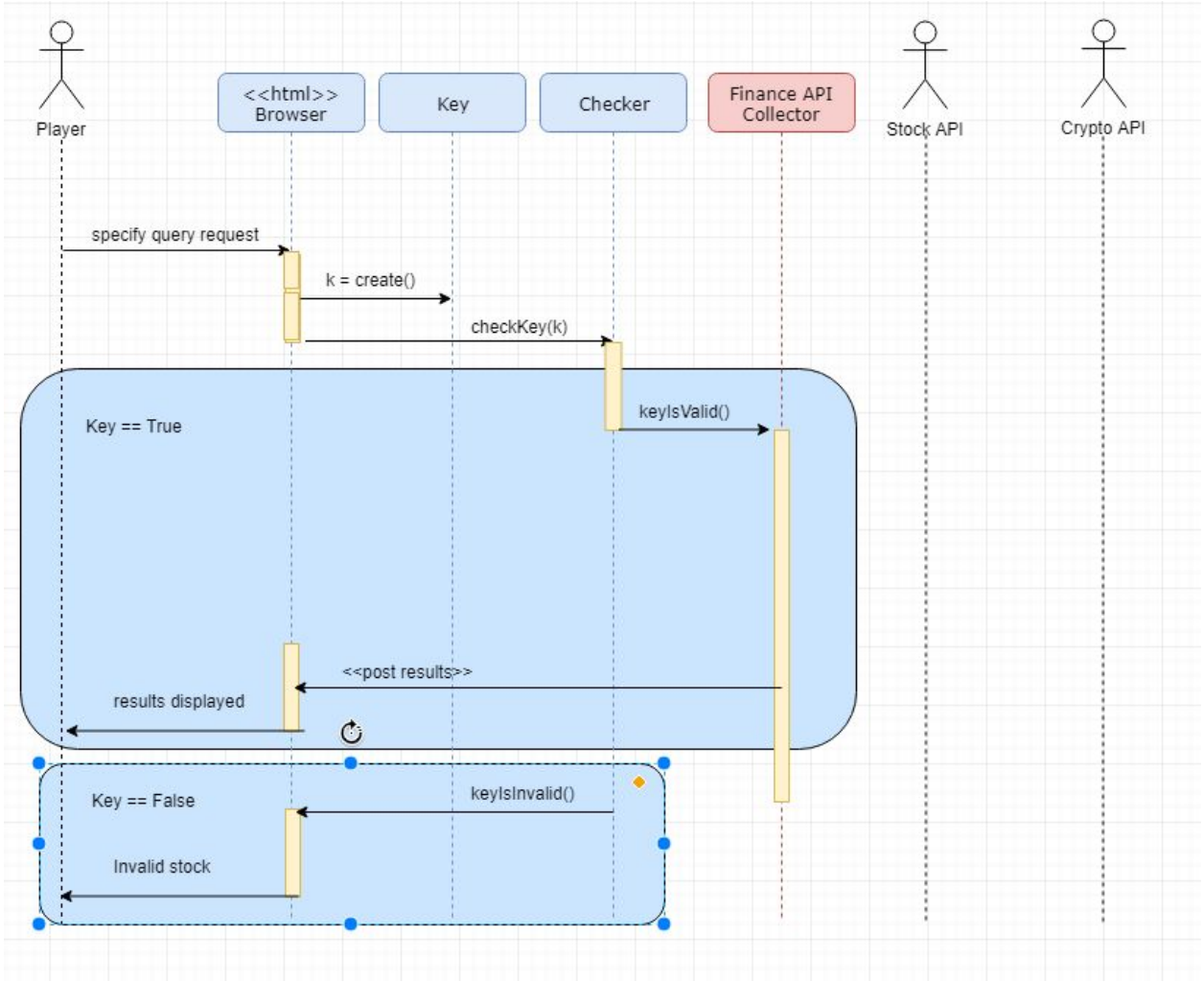
UC 10 - Accept/Decline Invitation



This use case controls whether or not a user invited to a private league accepts or declines the invitation. If they accept the invitation, the controller will communicate with the league control, league table, player control and player tables to create new player profiles and store the new data into the system. If they decline the invitation, the controller will communicate with the league controller to delete this request from the system. In both cases, the league control must check that the invitation request is valid.

Design Pattern: This use case implements a state pattern. It separates state-dependent event-handling functions from each other and allows easy additions of ew states and events.

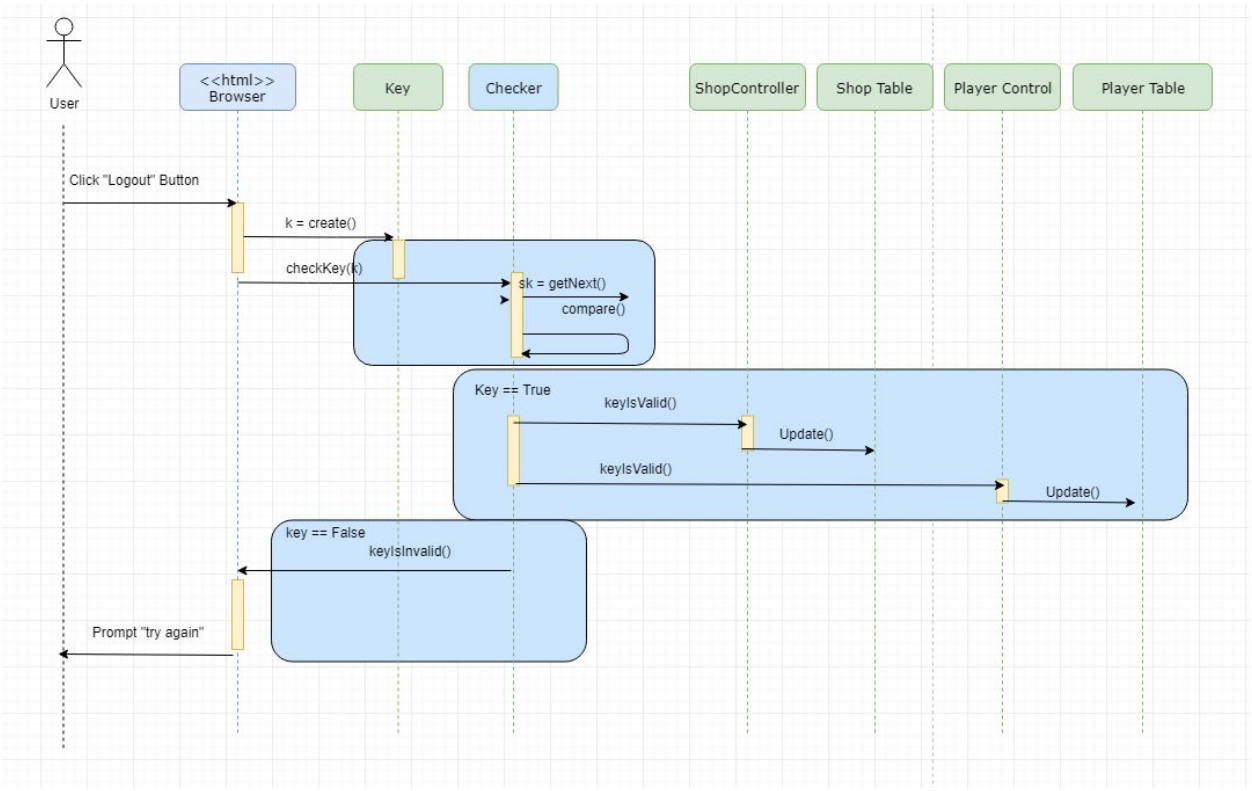
UC-11 - Search Stock



A user may want to search for a stock or cryptocurrency in order to find information about its current market value before deciding to buy or sell their assets. When a player is searching for a stock, they initiate a query request which is sent to the Browser. The Browser rocesses the query request and redirects it to the Finance API Collector to search for most updated market information, which is updated by the minute. A boolean isStock determines if the Finance API Collector will retrieve records for the Stock API or the Crypto API. The result is then passed back to the Browser and displayed to the Player.

Design Pattern: This uses a publish-subscribe pattern for the same reasons. If we choose to add functionality to our search stock use case, it will be easy to pass functions to new software objects and allow new members to quickly become familiar with this use case.

UC-14 -Buying and Competing Against an AI



A user may want to buy an AI to either play in their league as another player, or they can use the AI to play for them while they take a break. The user must first buy the AI from the shop using the browser, which causes a database check before the purchase may go through. This is done using the publish-subscribe pattern detailed below. Once an AI transaction has met integrity standards, the user is assigned the AI in the database and receives a purchase receipt. To use the AI for the league, the user visits their profile in the browser and selects the desired option for the function of the AI. The user is granted their respective AI action, and the database changes to reflect this.

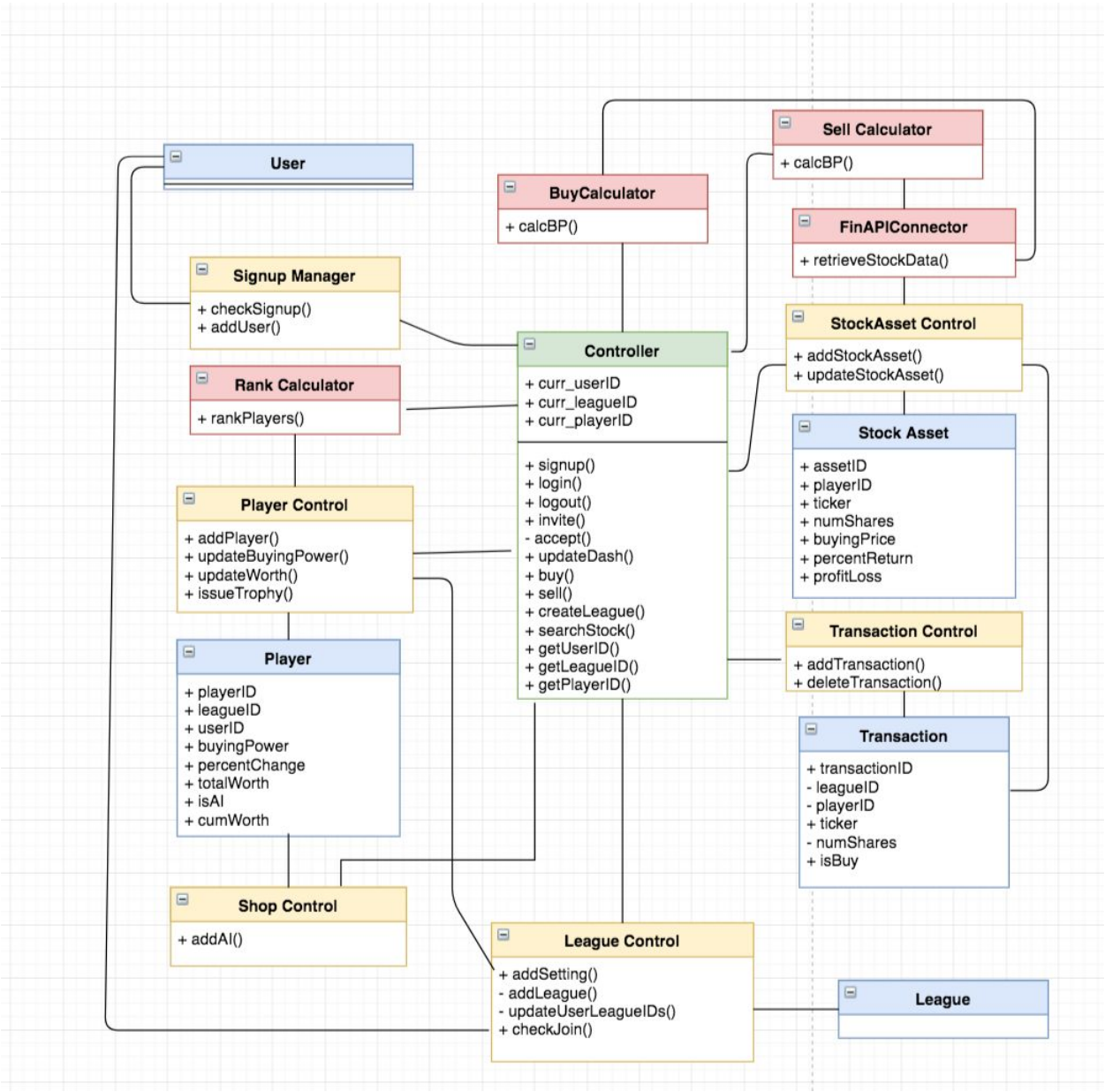
Design pattern: This uses the publish-subscribe pattern. The key is checked when purchasing the AI. If the transaction is deemed to be valid according to the user's current inventory and the AI's status in the shop, then the transaction will occur as detailed above. Otherwise, the user will be informed that their transaction is invalid and to try again.

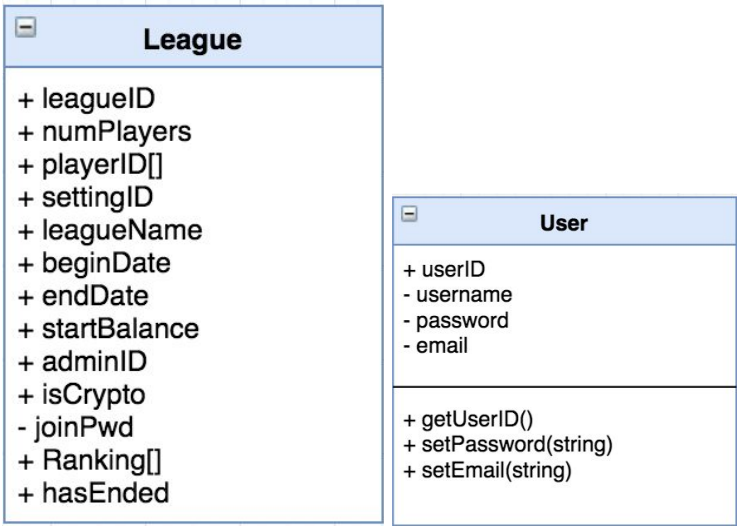
9 CLASS DIAGRAM AND INTERFACE SPECIFICATION

9.1 Introduction

The class diagram has a visualization of all classes and variables that will be used to execute the core functionalities of *Titan Trading*. We employ the use of a primary controller to oversee all activity, and then individual class controllers to separate responsibilities.

9.2 Class Diagram





9.3 Data Types and Operation Signatures

Controller

The controller is responsible for partitioning user actions into a sequence of steps involving the other classes of the system. The controller is the primary control point for all other control classes, and its methods are a precursor to other class actions.

Attributes

- **curr_userID : int**
  Indicates current user initiating an action
- **curr_leagueID : int**
  Indicates current league that is participating in the action
- **curr_playerID : int**
  Indicates current player that is participating in the action

Methods

+ **signup(username : String, pwd : String, email : String) : bool**
  This method is used to signup a new user to become a member of Titan Trading.
  User data signup data is stored in the *User Table* of the database.
+ **login(username : String, pwd : String) : bool**
  This method is used to log the user in to Titan Trading.
  User data entered will be checked with the data in the User Table of the database.
+ **logout() : bool**
  This method is used for the user to log out of their account.
+ **invite(email : String)**
  This method is used for a league admin to invite new players.

- **accept()**
  This method is used for new players to accept invitations to a league.'
+ **updateDash(c_userID : int)**
  This method is used to update the users dashboard while logged in.
+ **buy(c_leagueID : int, c_playerID : int, ticker : String, numShares : int)**
  This method is for the user to buy shares while in a league.
+ **sell(c_leagueID : int, c_playerID : int, ticker : String, numShares : int)**
  This method is for the user to sell shares from their portfolio within a league.
+ **createLeague(c_userID : int)**
  This method is used to create a new league.
+ **searchStock(ticker : String)**
  This method is used to search for a specific stock
+ **getUserID() : int**
  This method is used to retrieve the UserID for a specific user.
+ **getLeagueID(): int**
  This method is used to retrieve the LeagueID for a specific league.
+ **getPlayerID() : int**
  This method is used to retrieve the PlayerID for a specific player.

SignupManager

The signup manager is responsible for all signup and login activity. It is the primary interface to check username, passwords, and emails.

Methods

+ **checkSignup(username : String, pwd : String, email : String) : bool**
  This method checks whether the data given for a new account is valid. This will contain checking for a valid and unique email, unique username, and valid password.
+ **addUser(username : String, pwd : String, email : String) : bool**
  This method is used to add a new user's data to the User Table in the Database

User

This class represents a user and contains user login information and participating leagues. We currently allow the user to be part of the Universal League and an additional 3 private leagues.

Attributes

+ **userID : int**
  Identifies a unique user.

- **password : string**
  Passcode for a unique user to log-in to their account.
- **email : string**
  Unique email for a user to log-in to their account.

Player

This class represents a player in a particular league.

Attributes

+ **playerID : int**
  Unique ID associated with a unique player.
+ **leagueID : int**
  LeagueID associated with this player entry.
+ **userID : int**
  UserID who is running the player.
+ **buyingPower : float**
  Indicates how much money a player has in order to buy stock or cryptocurrency.
+ **percentChange : float**
  Indicates the percent change a player has made on a daily basis.
+ **totalWorth : floats**
  Indicates a player's total worth, which is the sum of their buying power and assets.

StockAsset

This class represents a specific stock and all of the data/information associated with it.

Attributes

+ **assetID : int**
  Unique integer identifying the asset a player has.
+ **playerID : int**
  Identifies which player owns the asset.
+ **ticker : String**
  Identifies the ticker symbol for the asset (i.e., BTC, APL).
+ **numShares : int**
  Identifies how many shares the player bought.
+ **buyingPrice : float**
  Identifies for how much the player bought the asset.
+ **percentReturn : float**
  Identifies the current return on investment in terms of a percentage.
+ **profitLoss : float**
  Identifies the total value of profit or loss on the original purchase.

StockAssetControl

The StockAssetControl class manages the Stock Assets and can perform actions such as adding and updating them.

Methods

+ **addStockAsset(curr_transaction : Transaction)**
  Adds a stock or holding to the StockAsset Table in the database. Each holding is associated with a league player.
+ **updateStockAsset(curr_transaction : Transaction)**
  Updates a holding in the StockAsset Table when a player buys or sells shares of a stock he/she already owns.

Transaction

This class represents a transaction, or a buy/sell operation. Each transaction is associated with the league, player, and specific stock asset.

Attributes

+ **transactionID : int**
  Identifies current transaction.
+ **leagueID : int**
  Identifies the unique league under which the transaction falls.
+ **playerID : int**
  Identifies the unique player that submitted the order for the transaction.
+ **ticker : String**
  Identifies the ticker symbol associated with the specific buy or sell order.
+ **numShares : int**
  Identifies how many shares were bought at the time of the transaction.
+ **isBuy : bool**
  Identifies whether the transaction is a buy or sell order. If isBuy is true, it is a buy order; else, it is a sell order.
+ **buysellPrice : int**
  Indicates the current price of the stock, retrieved from the API. This the latest stock price we will be using for all asset management

TransactionControl

The transaction control manages all the operations pertaining to the transaction table of the database.

Methods

+ **addTransaction(curr_tr : Transaction)**
  Adds a transaction to the database.
+ **deleteTransaction(curr_tr : Transaction)**
  Deletes a transaction from the database.

PlayerControl

The player control handles all player activity, which includes updating buying power, worth, and percent change whenever a player makes a transaction. The player control also keeps track of player trophies.

Methods

+ **addPlayer(curr_user : User, curr_league : League)**
  This method updates a league with any new players to be added
+ **updateBuyingPower(curr_tr : Transaction)**
  This method updates a player's buying power after a transaction
+ **updateWorth(curr_tr : Transaction)**
  This method updates a user's net worth at any point
+ **issueTrophy(trophyName : String)**
  This method gives a trophy to the winning player of a league

League

This class represents all the information for a league. It helps to complete track of which players are competing in the league, who the league manager is, and other league settings.

Attributes

+ **leagueID : int**
  Integer to identify the unique league.
+ **numPlayers : int**
  Indicates how many players are playing in this league.
+ **playerIDs[20] : int**
  Array of playerIDs to identify the players in this league. We currently allow for a maximum of 20 players in each league. (With the exception of the Universal League)
+ **settingID : int**
+ **leagueName : String**
  Indicates the name of the league.
+ **beginDate : datetime**
  Indicates the date and time at which the league was created.
+ **endDate : datetime**
  Indicates the date and time at which the league competition will end.
+ **startBalance : float**

Indicates the starting balance each player in the league will begin at.

+ **adminID : int**
  Indicates the user ID of the league manager.
+ **isCrypto : bool**
  Indicates whether the league will be investing in cryptocurrency or stocks.
+ **joinPwd : String**
  Indicates the password players must enter in to join the league.
+ **Ranking[num_players] : intRanking[num_players] : int (playerID int)**
  Current ranking of all players in terms of the league's player IDs.

LeagueControl

The LeagueControl class manages each specific league and is used whenever players are added or settings are edited.

Methods

+ **addSetting(curr_league : League)**
  Add setting record to database.
+ **addLeague(curr_league : League)**
  Add league record to database.
+ **updateUserLeagueIDs(curr_leagueID : int, curr_userID : int)**
  Update list of the current user's active leagues when adding a player to a league.
+ **checkJoin(c_leagueID : int, c_playerID : int, c_userID : int) : bool**
  Notifies whether a player has successfully joined a particular league.

ShopControl

The ShopControl class handles the purchases of artificial intelligence and TitanCoins by users.

Methods

+ **addAI(leagueID : int, userID: int, aiDifficulty: int)**
  Add purchase of AI record to database.

FinAPIConnector

The FinAPIConnector class uses the Finance API to gather stock data to be used.

Methods

+ **retrieveStockData(ticker : String) : String**

Retrieves all stock data from the Finance API including the timestamp, open, high, low, close, and volume of the particular stock. Returns a string in JSON format.

RankCalculator

The RankCalculator class will order the players in a league by their ranking based on total earnings.

Methods

+ **rankPlayers(leagueID : int) : int ***
  Ranks players in a particular league based on totalWorth. Returns a pointer to the sorted array of players.

BuyCalculator

The BuyCalculator class will update a player's buying power based on the old buying power and and the

Methods

+ **calcBP(c_playerID : int, transactionID : int) : int**
  Calculates buying power for a player based on the input transaction. For a buy, subtracts price of stock * amount of shares from current buying power.

SellCalculator

The SellCalculator class will calculate the buying power after a sell order has been made.

Methods

+ **calcBP(c_playerID : int, transactionID : int) : int**
  Calculates buying power for a player based on the input transaction. For a sell order, the calculator will add the market price of the asset * amount of shares to current buying power.

9.4 Traceability Matrix

	Domain Concepts																					
Class	Controller	User Table	Login Manager	Signup Manager	Ranking Calculator	Setting Table	Player Table	Player Control	League Table	League Control	Financial API Connector	Transaction Table	Transaction Control	Asset Table	Stock Asset Control	Buy Calculator	Sell Calculator	AI Control	AI Table	Shop Control	Shop Table	Inventory Table
Controller	x																					
SignupManager			x	x																		
User		x	x																			
Player							x	x										x	x			
StockAsset														x								
StockAssetControl														x	x							
Transaction												x										
TransactionControl												x	x									
PlayerControl			x				x	x										x	x			
League									x													
LeagueControl						x			x	x												
FinAPIConnector											x											
RankCalculator					x																	
BuyCalculator																x						
SellCalculator																	x					
ShopControl																				x	x	x

- The Controller, RankingCalculator, BuyCalculator, and SellCalculator all have classes with the same name as their domain concept. For each of these classes, we decided that the domain concept name was suitable for one class and so we kept the same names.
- The controller class will be the central control point for all other classes that make up the game.
- The RankCalculator, BuyCalculator, and SellCalculator are all named for their function, thus they were the only classes derived from their respective domain concepts.
- The SignupManager will keep track of all signup and login activity, thus it is derived from the LoginManager and the SignupManager.
- The FinAPIConnector comes from the Financial API Connector domain concept. We chose to shorten the name to FinAPIConnector so it wouldn't be as long.
- From the User Table and Login Manager domain concepts, we created the User class, which will create the different users in the game and keep track of them.
- Next, we created the Player class from the Player Table and PlayerControl concepts, which will create the players represented in a particular league. With the addition of the AI Table and AI Control, we felt that the Player class is suitable to hold the information

for a player. While the AI trades by itself, its information and assets can be stored the way a player's would.

- From the LoginManager, PlayerTable and PlayerControl domain concepts, we decided to create a PlayerControl class. This class will keep track of all player activity and each players trophies. We also created StockAsset and StockAssetControl classes derived from the asset table and stock asset control domain concepts to create the actual stock objects, keep track of any information associated with it, and manage the stock assets in the game. Again, the AI Table and AI Control also fit under this class, as the AI would be considered another "player" in the league even though the way the AI comes up with its trades is different.

- Next, we created a Transaction class from the Transaction Table domain concept to create the individual transactions that take place.

- The TransactionControl class, derived from the Transaction Table and Transaction Control domain concepts will keep track of all the transactions that occur in the game, and what league, player, and stock asset the transaction is associated with.

- Next, we have the League class, which comes from the LeagueTable domain concept.

- The League class will create the individual leagues and their settings, while the LeagueControl class, derived from the LeagueControl, SettingTable, and League Table domain concepts, will be the central point of management for all the different leagues in the game.

- The final class we created is the ShopControl class. This is derived from the Shop Control, Shop Table, and Inventory Table domain concepts associated with our in game shop idea. The domain concepts outline how we have a shop and items in the shop, as well as how we determine whether or not a user has certain items. From these concepts, we came up with the ShopControl class, which will take care of the functions of buying and selling from the shop, as well as updating the user's inventory table when they make a purchase from the shop.

9.5 Design Patterns

Publish-subscribe pattern:

Creating a league implements publish-subscribe due to the database verifications needed. When creating a league, the league and setting attributes need to not violate the integrity of the database. The methods used for LeagueControl to check this are addLeague(), addSetting(), and checkJoin(). Therefore, the respective controls are subscribers that receive the message upon the league creation form submission.

Logout implements this pattern by verifying that no database operations respective to the user are in progress when logout is requested. This operation is done by the SignupManager with checkSignup().

To sell a stock, the Checker publishes an integrity violation check to all the necessary subscribers. Each control runs their respective functions to handle the transaction. If anything is found to violate the transaction, the transaction is rejected.

When searching for a stock, the Finance API Collector is the subscriber to the messages of the Browser. Upon a search query, the Collector receives the message and returns the correct stock ticker if valid. Otherwise, the query is found to be invalid and the user is given an error.

To buy and create an AI, verification is needed with the user's current purchases in the database. The addAI() function is a subscriber to the browser. Upon notification of an AI purchase, the database is checked with the userID and the AI difficulty to confirm that this purchase is valid. If not, then the purchase is dismissed.

State Pattern:

To accept or decline an invitation, the event handler object is the Controller. Without knowledge of the state of the controls, the Controller requests them to check whether the player's join is valid. If invalid, the join is rejected.

9.6 Object Constraint Language Contracts

Object Constraint Language

To compartmentalize program functionality, we separate our domain components in to separate controller classes.

Main Controller (linked to other controllers)

Controls primary functionality for all use cases. See views.py in home directory for explicit listing of functions associated with specific urls. For example, we have a function for *UC-7: "buy stock"* use case and a function for *UC-8: "sell stock"*.

Preconditions: None

Postconditions: Completed desired function and redirects to appropriate page

Signup Manager

The signup manager allows users to create a custom username and enter a secure password. Passwords must be at least 8 characters long. Passwords Ensures proper authentication using Django's PAP (Password Authentication Protocol). The signup manager also makes an associated user profile linked to entries in Django's auth_user table in the database.

Preconditions: User has entered username and valid password in signup form

```
form = SignUpForm(request.POST)
If form.is_valid(): ...
```

Postconditions: User entry has been populated in database, user is authenticated and logged in

```
...
user = authenticate(username=username, password=raw_password)
user.save()
auth_login(request, user)
```

Player Control

The player control manages buys and sells for each player in a league, and updates player attributes accordingly

Preconditions: User has already been redirected to an individual league page, so playerID and leagueID are known

```
submitBuy(request,league_id,player_id): ...
selldash(shares,player_id, league_id, asset_id): ...
```

Postconditions: Player buying power, assets, and totalWorth are updated with every transaction

```
...
player.buyingPower += sellTotal
player.totalAssets -= sellTotal
player.cumWorth = player.buyingPower + player.totalAssets
player.save()
```

Shop Control

The shop control allows users to buy packages of TitanCoins or purchase an AI to compete against in a private league. We use a Django one-to-one model to link user profiles to user models.

Preconditions: User must have a Paypal account to purchase TitanCoins or enough TitanCoins to purchase an AI. Alternatively, a player can receive TitanCoins by winning games, beating the AI, or making a certain number of trades.

```
if (endDate < presentDate): # league has ended, redirect to leaderboard.html
             if not (league.hasEnded): # need to handle trophies
                    league.hasEnded = True
                    current_user.profile.trophies[3] += 1 # increment for game
played
```

```
                    if rank < 4: # top 3 = win
                          current_user.profile.trophies[2] += 1 # increment for win

                    current_user.profile.trophies[4] =
current_user.profile.trophies[4] + numAIbeat

                    if admin.id == request.user.id: # this user is admin
                          if current_user.profile.trophies[5] < count: # new record
for # ppl managed
                                current_user.profile.trophies[5] = count
                    current_user.save()
```

Postconditions: A player has successfully added TitanCoins to his/her account or successfully added an AI to his/her private league.

```
def submitShop(request,item):
      if item == 1:
            request.user.profile.TitanCoins = request.user.profile.TitanCoins +
100
      elif item == 2:
            request.user.profile.TitanCoins = request.user.profile.TitanCoins +
200
      elif item == 3:
            request.user.profile.TitanCoins = request.user.profile.TitanCoins +
300
      elif item == 4:
            if (request.user.profile.TitanCoins<100):
                  return HttpResponseRedirect('/dashboard')
            request.user.profile.TitanCoins = request.user.profile.TitanCoins +
300
```

StockAsset Control

The StockAsset control updates user assets periodically in order display an accurate player worth and buying power. Asset prices need to be pulled asynchronously from the API and update the value of all the shares the user owns.

Preconditions: User is authenticated and owns shares of one or more stock

```
      assets = Asset.objects.filter(playerID = currPlayer.id)
```

Postconditions: Asset objects are updated with accurate price to update player worth and buying power

```
players = Player.objects.filter(leagueID = league).order_by('-cumWorth')
      for i in players:
            worth = 0
            assets = Asset.objects.filter(playerID=i.id)
            for a in assets:
                  marketPrice = getPriceFromAPI(a.ticker, league.isCrypto)
```

```
                    worth+= (marketPrice*a.shares)
            i.totalWorth = worth
            i.cumWorth = i.totalWorth + i.buyingPower
            i.save()
```

Transaction Control

The Transaction Control stores player transactions in order to provide past purchase information to the user and the capability to print purchase receipts.

Preconditions: The player has purchased a stock

```
new_transaction = Transaction(leagueID = league, playerID = player.id, price =
tmpPrice, ticker = ticker, shares = shares, isBuy = True)
```

Postconditions: The player receives a receipt of his/her transaction

```
def transactionReceipt(request,transaction_id):
      lastTransaction = Transaction.objects.get(pk=transaction_id)
      league = lastTransaction.leagueID
      price = lastTransaction.price
      ticker = lastTransaction.ticker
      shares = lastTransaction.shares
      return render(request, 'receipt.html', {'price': price, 'ticker': ticker,
'shares': shares})
```

10 SYSTEM ARCHITECTURE AND SYSTEM DESIGN

10.1 Architecture Styles

For our project we will be using various Software Architecture styles to help create a reliable and fast product. Many of these styles are well known and used across all different types of software projects, making them well documented and easy to implement in our project. The styles we are using in our project include Representational State Transfer (REST), Data-Centric, Plug-ins, and Client-Server styles.

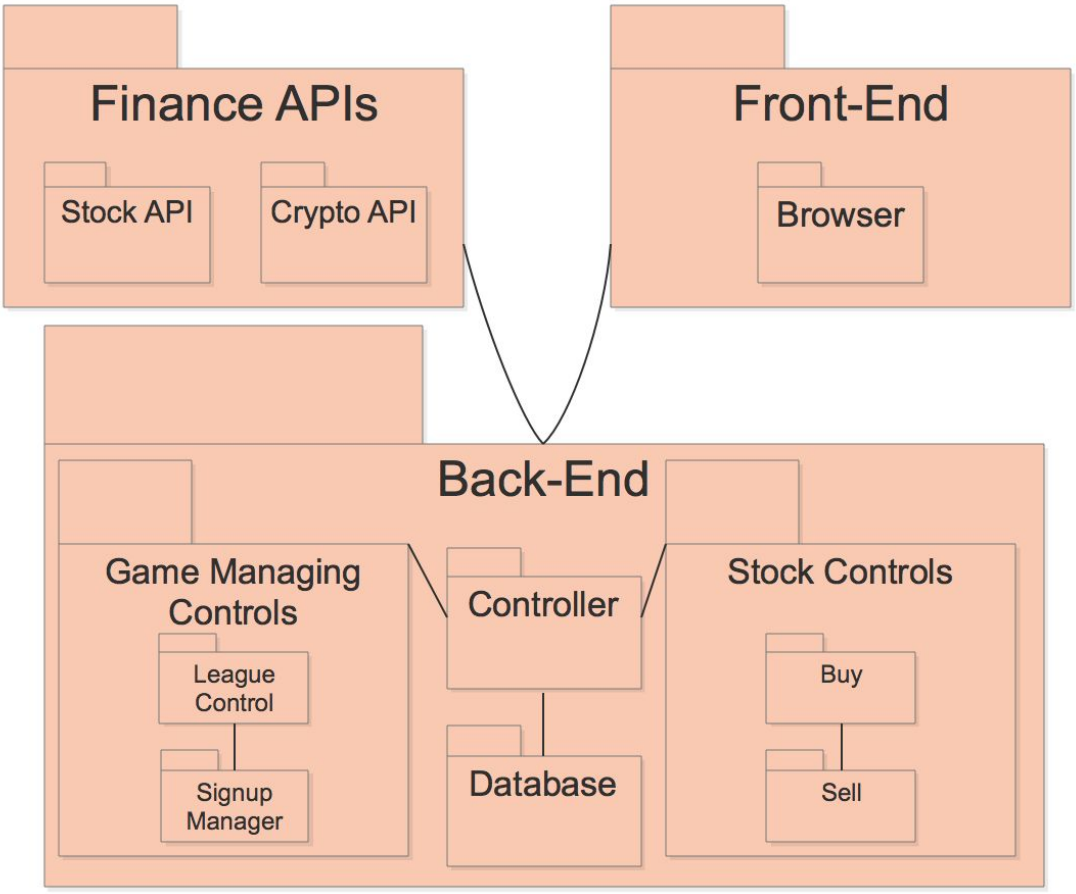
Representational State Transfer:
Representational State Transfer (REST) type APIs make it very simple to update database information from client side. In our project will be storing a lot of information inside a database, including stocks held and information about sales. When a stock transaction is performed, a REST type API will be used by the client to put this information in the database. Setting this up is streamlined due to the fact that we are using the Django framework with Python.

Data-Centric:
Our application is data-centric by its very nature. We need to keep track of not just what stocks that the user is holding, but the live price of the stock, league standings information. Additionally, the client must be given all of this data in an understandable and easy to digest way. To do this we will be leveraging the REST API system and a SQL database in conjunction with various other stock information APIs to gather price information for the clients. This information will be stored in the databases, where the user can access it by means of a transaction or information query.

Plugins:
The user will not interact directly with our plug-in system, but will reap the benefits of it when using our website. Python and Django are both plug-in heavy frameworks. Many of the difficult challenges of other programs can be solved with a package available on the pip repository. This will allow for us to have a speedy development time while being able to concentrate on the end user experience.

Client-Server:
The Client-Server style ties directly in with the Data-Centric and REST system styles. The client will be interacting with the server a lot when browsing and using the website. Almost all of the content we will be serving is dynamic and needs to be updated both by the user and outside data. When it is updated, the user must be able to request and see this new information.

10.2 Identifying Subsystems



The project incorporates three main subsystems: the front-end, the back-end, and the finance APIs.

The front-end encompasses everything the user interacts with. It includes the browser subsystem, which lets the user browse the various web pages of the website and interact with the web widgets. When the user interacts with the web page that requires the system to need to consult the database or finance APIs, the front-end subsystem will communicate the necessary tasks to the back-end system, which handles all the work. The front-end subsystem simply waits for the desired response from the back-end system and presents the updated information to the user.

The finance APIs are used to retrieve real-time stock market data and news. The stock API subsystem gives information relating to stock prices and sector performances while the crypto API does the same for cryptocurrencies. The only subsystem that communicates with the Finance API subsystem is the back-end subsystem, which requests the API subsystem for the required information.

The back-end subsystem handles the majority work in the project. It first holds the controller, which manages communication between the various subsystems. The database subsystem allows for additional, removal, and access of the data in the application. Other subsystems that need use of the database submit their request to the controller, which notifies the database subsystem to query the requested information.

The game managing controls subsystem handles the managerial portion of the league system. It contains the league control subsystem, which handles league creation, settings, players, and invites.

The signup manager is in charge of user accounts on the application. When a user creates an account, it interacts with the database to make sure that the creation request complies with any existing constraints.

The stock controls subsystem checks that any stock operations preserve the integrity of the game. It calculates the purchase/sale, verifies the asset prices with the API, and checks the user balance with the database.

10.3 Persistent Data Storage

Titan Trading is a platform that provides Users the ability to create accounts, join leagues, and store stock and crypto assets. Persistent Data Storage is a key function that must be incorporated into the backend to keep track of player data and provide the most realistic trading experience possible.

For the backend, we will be using PostgreSQL, an enhanced version of SQL, to create relational database tables that store specific information. The persistent tables that will be made include: User Table, Setting Table, Player Table, League Table, Transaction Table, and Asset Table.
SQL Database Schema Query with a short description of each table's use of Persistent Data:

```
CREATE TABLE 'auth_user'(
        'userID' INT,
        'password' VARCHAR(128),
        'last_login' TIMESTAMP,
        'is_superuser' BOOL,
        'username' VARCHAR(150),
        'first_name' VARCHAR(30),
        'last_name' VARCHAR(150),
        'email' VARCHAR(254),
        'is_staff' BOOL,
        'is_active' BOOL,
        'date_joined' TIMESTAMP,
        PRIMARY KEY('userID')
)
```

Description: The auth_user table will hold the general Titan Trading account information. Fields such as username,password, and email must be persistent because a user's account information should definitely still exist after a single execution. Originally, the leagueIDs of a user would be stored in this table, but we shifted to a more dynamic approach which will be explained in greater detail in the following tables.

```
CREATE TABLE 'home_player'(
        'playerID' INT,
        'buyingPower' FLOAT,
        'percentChange' FLOAT,
        'totalWorth' FLOAT,
        'userID' INT,
        'leagueID' INT,
        FOREIGN KEY('userID') REFERENCES 'auth_user'('userID'),
        FOREIGN KEY('leagueID') REFERENCES 'home_league'('leagueID'),
        PRIMARY KEY('playerID')
)
```

Description: The home_player table will hold the information on how a user is doing in a particular league. Fields such as buyingPower, percentChange, and totalWorth must be persistent so that the progress of the players can be saved.

```
CREATE TABLE 'home_league'(
        'leagueID' INT,
        'name' VARCHAR(50),
        'isUniversal' BOOL,
        'numPlayers' INT,
        'beginDate' TIMESTAMP,
        'endDate' TIMESTAMP,
        'startingBalance' FLOAT,
        'adminID' INT,
        'isCrypto' BOOL,
        'joinPassword' VARCHAR(20),
        FOREIGN KEY('adminID') REFERENCES 'auth_user'('userID'),
        PRIMARY KEY('leagueID')
)
```

Description: The league table contains the various settings necessary when a league is created. The leagueID is used to relate players to leagues. The settings were originally stored in a separate table, but we found it neater to be contained in one table. The adminID is the userID of the user that created the league, which is necessary to see who has certain privileges such as inviting.

```
CREATE TABLE 'home_transaction'(
        'transactionID' INT,
```

```
    'leagueID' INT,
    'playerID' INT,
    'price' FLOAT,
    'ticker' VARCHAR(5),
    'shares' INT,
    'isBuy' BOOL,
    FOREIGN KEY('leagueID') REFERENCES 'home_league'('leagueID'),
    FOREIGN KEY('playerID') REFERENCES 'home_player'('playerID'),
    PRIMARY KEY('transactionID')
)
```

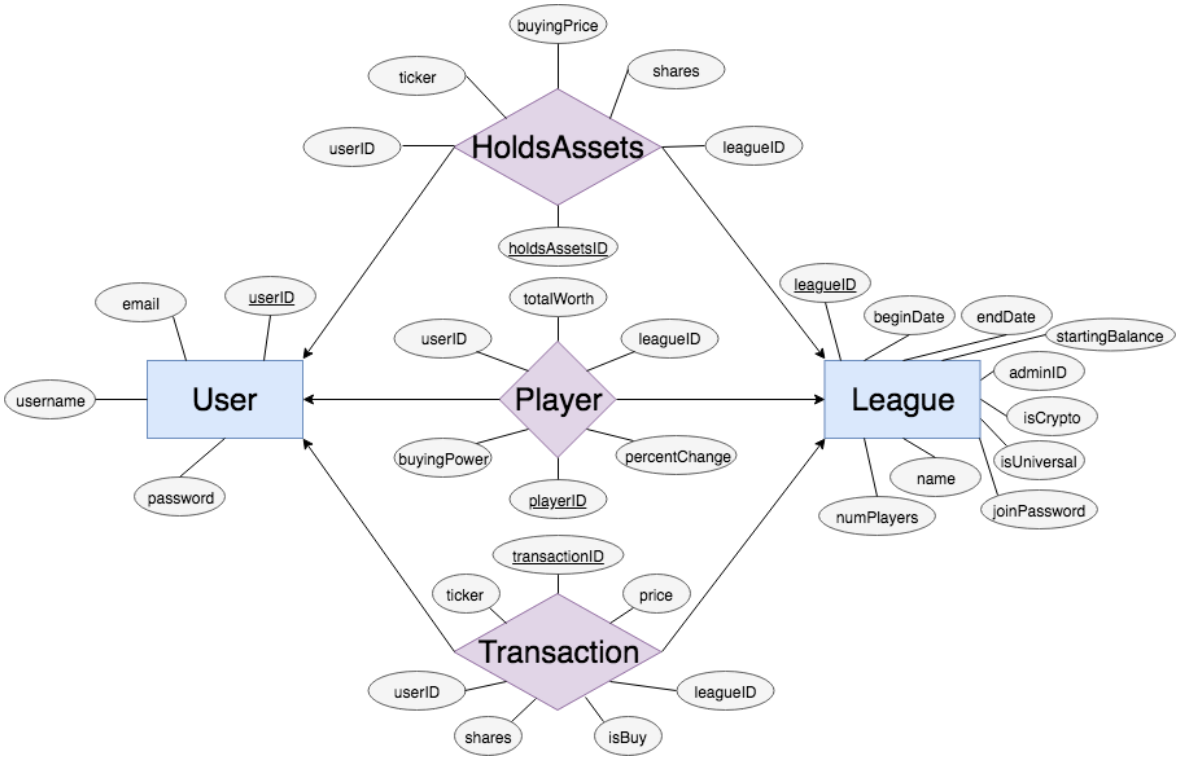
Description: The transaction table will be a record of all the transactions that a player takes within a league. This must be persistent so that the players can see the history of their transactions.

```
CREATE TABLE 'home_asset'(
    'assetID' INT,
    'ticker' VARCHAR(5),
    'leagueID' INT,
    'playerID' INT,
    'shares' INT,
    'buyingPrice' FLOAT,
    FOREIGN KEY('playerID') REFERENCES 'home_player'('playerID'),
    FOREIGN KEY('leagueID') REFERENCES 'home_league'('leagueID'),
    PRIMARY KEY('assetID')
)
```

Description: The asset table stores information about the users current stock assets. This information must be kept persistent so that the user won't lose his assets upon log out.

```
CREATE TABLE 'home_profile'(
    'profileID' INT,
    'trophies' INT[],
    'statement' VARCHAR(250),
    'name' VARCHAR(40),
    'userID' INT,
    'birthday' TIMESTAMP,
    FOREIGN KEY('userID') REFERENCES 'auth_user'('userID'),
    PRIMARY KEY('profileID')
)
```

Description: The profile table stores information about each specific user. This information is kept persistent in case any user views the profile page.



Here is the entity relationship diagram to describe our design for the database.

10.4 Network Protocol

The communication protocol Titan Trading has picked is Hypertext Transfer Protocol (HTTP). The main goal of our project is to create a functional stock market game that can communicate with users through a website. HTTP is the communication protocol used by the World Wide Web and is how web browsers and web servers communicate with each other the internet. Our servers will be able to communicate with the user's browser and respond to user actions with the appropriate commands instantly. HTTP will also enable our application to seamlessly update real-time stock prices and communicate these to our users through our website. Through this protocol, users will be able to access different web pages through hyperlinks by interacting with our unique interface on our website and have the full Titan Trading experience.

10.5 Global Control Flow

Execution Order

Titan Trading is a primarily event-driven system because our website is mostly based on user actions. Most of our functionalities and use cases are initiated by the user. As shown in the user interface specifications, there are multiple decisions that a user can decide to take when navigating throughout our website. Therefore, every user will not go through the same steps and web pages every time. Rather, the system will wait in a loop until a user makes a decision to

navigate to participate in any of the functionalities Titan Trading offers. For example, after signing in, user1 may decide to go to the Glossary page to read a definition about trading; user1 may then proceed to his League Dashboard and enter the Universal League to view the leaderboard. In contrast, after signing in, user2 may decide to go straight to the League Dashboard and create a new league. User2 will then begin to invite players to join her league.

Although Titan Trading is a primarily event-driven system, some system processes are strictly linear, as shown by the system sequence diagrams. The linearity is present in the interactions between our domain concepts and players. For example, in Use Case 4 (Logout), the user will take initiate an event by clicking the "Logout" button. Then, for every user, control will shift to the Browser, followed by the Controller, followed by the Browser, and back to the user. Other instances of linearity are present for certain events. For example, if a guest wants to begin investing, the guest must first register with Titan Trading, join a league, and traverse to the league's specific buy page in order to start investing.

Time Dependency

Titan Trading is a time-dependent system. Titan Trading uses real-time market prices from our finance API, which can be updated every minute. In addition, our website displays a host of widgets, such as graphs and tickers, that are also dependent on real-time data. Titan Trading utilizes system wide timers, in order to correctly process actions.

For buy and sell orders, there is a Finance API timer and Stock Market Open/Close timer. These two timers are utilized to update real-time market data, which can be described as follows.

- Finance API timer - This timer will notify the system after a minute has passed. After each minute passed, the market data will be updated. This timer is particularly useful for buy and sell orders, so that the market price at the exact time the order was submitted will be used to update critical player parameters, such as total worth and buying power.
- Stock Market timer - This timer will notify the system once the stock market open and closes. Between these intervals of time, any orders submitted will not be accepted by Titan Trading and will notify the user that the market has closed.
- Percent Change timer - This timer will notify the system when to update the percent change for all players. The percent change for each player will be recalculated on a daily basis.

Concurrency

Concurrency constructs are necessary for websites that are managing multiple users. Python's Django already provides a framework for multiprocessing and data safety with synchronization constructs. Within Django, multiprocessing is necessary to process and execute the concurrent actions that concurrent users may decide to take. Synchronization constructs are necessary for data safety, so that at any point in time, all databases used within Titan Trading have consistent data. Specifically, Django uses synchronization locks in order to maintain data consistency (Benita).

10.6 Hardware Requirements

Overview:

As a whole, the majority of computational resources are used on the server-side, with the use of Django, a PostgreSQL database, and a python backend that handles front-end interaction. This minimizes the need for large hardware on the user side, and allows as many devices as possible to be supported.

Internet Connection:

The system relies on the use of APIs and widgets that are available for use all over the web. As such, a persistent internet connection is required to use the system to pull data and verify the legitimacy of transactions. This also requires a 99%+ uptime for the server infrastructure, in order to maintain consistency of use for all users. For user connections, a broadband connection should provide enough bandwidth. Since most data transmissions will be done using HTTP and TCP protocols, more bandwidth will result in faster performance.

Server Space and Size:

Thanks to the proliferation of Amazon AWS, Google Cloud, Microsoft Azure and other cloud service providers, the servers that we select are easily scalable and adjustable. Meeting demand and matching required disk space is simple using the modern interfaces of these cloud services. This scaling in size can occur over any period of time. Our PostgreSQL database can be scaled as user count grows and our space usage during a given day can also be scaled to match demand during the busy trading hours every day.

User Device:

Users will be able to access the site through any modern browser that support Javascript, HTML5, and CSS3. This includes, but is not limited to:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Apple Safari
- Opera

The site is optimized to run in a desktop orientation and is best accessed from a laptop or desktop. This does not rule out the use on mobile devices, however the experience may be different and unoptimised for the screen size. As such a 1024 × 768 is the specified minimum screen resolution.

11 ALGORITHMS AND DATA STRUCTURES

11.1 Algorithms

Algorithm Design

We are using a few different algorithms for this project. Both the backend and the front end require algorithms in order to provide a favorable user experience. In the back end we will need to use various performance algorithms in order to see what the performance of our program is and how we can improve on it. In the backend we will be mostly using a mathematical model to calculate and process how much buying power the player has, along with their total worth. The calculation for these values after a buy order are below:

$$\text{buying power} = \text{current buying power} - (\#\text{ of assets bought}) \times \text{market price of asset}$$
$$\text{total worth} = \text{total worth} + (\#\text{ of assets bought}) \times \text{market price of asset}$$

Additionally the calculation for these values after a sell order are below:

$$\text{buying power} = \text{current buying power} + (\#\text{ of assets sold}) \times \text{market price of asset}$$
$$\text{total worth} = \text{total worth} - (\#\text{ of assets sold}) \times \text{market price of asset}$$

In the front end we will be using a mathematical algorithm to decide league rules and if a user is breaking them or not, in addition to giving the user achievements. Some of the models for these achievements would be if the user is in a certain number of leagues. This calculation is pictured below.

$$\text{if } (\#\text{ of participating leagues}) == 3, \text{ get a reward}$$

Artificial Intelligence Algorithm

Players may compete against several difficulty levels of the artificial intelligence.

Medium and Hard Difficulty

The medium difficulty level utilizes a common machine learning algorithm known as linear regression. The medium level difficulty artificial intelligence will run two different linear regression. The first session will take in monthly market stock and time data, in which it will similarly find linear weights and biases to fit a linear graph onto the data. In the model, we will use a loss function called square loss. The model will run through 100 epochs of training, where the Gradient Descent algorithm will attempt to minimize the squared error between predicted Y values and the true Y data.

A loss function in linear regression is a mathematical function that calculates how far the predicted values are away from the true data. It initially guesses the weights and biases to equal zero. During each epoch and iteration of the [x,y], the Gradient Descent algorithm will guess which direction to approximate the next prediction.

The hard AI differs slightly. In addition to linear regression, we will be scraping the Internet for both tweets and news articles about the stock or cryptocurrency in question. Sentiment analysis will be used to analyze either the tweet or news article has a positive connotation, negative tone, or neutral.

Therefore, we use three different layers to our analysis in order to make a decision. A monthly regression analysis will allow the artificial intelligence to view long term and average trends. Twitter sentiment will inform the AI if the stock is in the favor of the public people, while news sentiment will inform the AI if the stock is in favor of professional stock and cryptocurrency analysts.

In each decision, the AI will be scanning the web for top 25 stocks of the day in order to analyze. Each of the 25 stocks will run through this process and achieve a certain amount of reward for each positive contribution the stock will be making. The amount to buy or sell will be a percentage of the reward amount.

The pseudocode is as follows:

```
def decideBuyAndSell():
	top25 = findTop25(isCrypto);
	for i in range(25):
		equity= top25(i)
		isDaily = true
		[dailyData, timeData] = getFromAPI(isDaily, equity)
		[wd, bd] = runlinreg(dailyData, timeData)
		[monthlyData, timeData] = getFromAPI(false, equity)
		[wm, bm] = runlinreg(montlyData, timeData)

		perPos, perNeg, perNeut, numTweets = getTwitterSentiment(equity)
		perPosN, perNegN, perNeutN, numArticles = getNewsSentiment(equity)

		rewards[i] = calculateReward([wd, bd], [wm, bm], [perPos, perNeg, perNeut,
	numTweets],    [perPosN, perNegN, perNeutN, numArticles]
	end

	[numRewardBuy, optimalBuy] = max(rewards)
	[numRewardSell, optimalSell] = min(rewards)

	return optimalBuy, numRewardBuy, optimalSell, numRewardSell
```

Algorithm for Future Use: Challenging Difficulty

The challenging difficulty operates very similarly to the medium difficulty; however, the linear regression model, which is typically categorized under the machine learning realm, will be replaced with a convolutional neural network. This deep learning will predict more accurately the direction of the stocks than a simple linear regression model. This deep learning network will be modeled after Alexandr Honchar's two-layer convolutional neural network ("Neural networks for algorithmic trading"). Both layers are combinations of convolution and max-pooling layers with a rectified linear unit (ReLU) as the activation function. In this model, a binary cross entropy loss function and Adam Optimizer will be used, which contrast the linear regression simple squares loss function and Gradient Descent Optimizer. Using this model, we can train the specific dataset to a stock's particular data. Then, we can use the trained convolutional network to predict future values and past values to determine what the price difference would be, using dynamic programming. Afterwards, it will follow the same strategy as above to look at sentiments from tweets and news articles. The predicted price and sentiment analysis will yield the reward to find optimal buy and sell.

11.2 Data Structures

Database/Hash-Table

A database can be considered its own data structure because it has several constructs and methods of accessing, retrieving, and updating data. Databases have a structure similar to hash-tables, where data is retrieved and added to the table in O(1) time because it is mapped by a key, or index calculated by a hash function. Here are some of the key terms of databases:

Bucket: a bucket is a unit of storage and can contain one or more database records.

Hash function: maps sets of search keys to the address of buckets that contain valid records in the database.

Insertion: When we want to add a record to the database, we find the correct bucket address by calling the hash function. For search key **k** and hash function **F(),** we would call **bucket address = F(k).** For optimizing insertion, we can use methods of chaining or linear probing which you can read more about here.

Search/Query: We can use the same hash function to search for a record, provided a valid key.

Delete: Same as search operation followed by a delete.

Update: Perform a query and update the record.

Structure:

A database can contain multiple tables called relations. Each relational table contains multiple records, each identified by a primary key. In our model, we have named our primary keys clearly, with names like userID, leagueID, playerID, etc. Each record contains several attributes as well. See Persistent Data Storage for more information.

Queue

For this project we plan to put all our background processes and multiple user requests in a queue structure to synchronize application activity. We plan to use Django-rq which is a queue structure that follows a “first-in, first-out” (FIFO) methodology. We will be using the queue to handle requests from multiple users and execute commands in the proper order.

Enqueue adds an item to the queue at the end of the list.

Dequeue removes an item from the queue at the from the front of the list.

12 USER INTERFACE DESIGN AND IMPLEMENTATION

User Interface Design and Implementation

Implementation

Implementation of all web pages were used in html and css, and Javascript. As of right now, our web pages can be run on a local server if you have the source code. To implement these webpages, we used Bootstrap, which is an open-source, front-end framework for developing with html, css, and Javascript. All webpages were routed together and can be launched, using Python's Django. The following code is an example of connecting the written html for each page to the actual web server. Finally, all background images used are open-source.

```
from django.shortcuts import render
from django.template import loader
from django.http import HttpResponse

def index(request):
    template = loader.get_template('greet.html')
    return HttpResponse(template.render({},request))
def signup(request):
    template = loader.get_template('signup.html')
    return HttpResponse(template.render({},request))
def aboutus(request):
    template = loader.get_template('aboutus.html')
    return HttpResponse(template.render({},request))
def home(request):
    template = loader.get_template('home.html')
    return HttpResponse(template.render({},request))
def dashboard(request):
    template = loader.get_template('dashboard.html')
    return HttpResponse(template.render({},request))
def createleague(request):
    template = loader.get_template('createleague.html')
    return HttpResponse(template.render({},request))
def universal(request):
    return HttpResponse("This is the universal league page.")
def league1(request):
    return HttpResponse("This is the league1 page.")
def league2(request):
    return HttpResponse("This is the league2 page.")
def league3(request):
    return HttpResponse("This is the league3 page.")
def profile(request):
    return HttpResponse("This is the profile page.")
# Create your views here.
```

Code Using Python's Django to Launch Html Web Page

Page by Page Modifications and Implementations

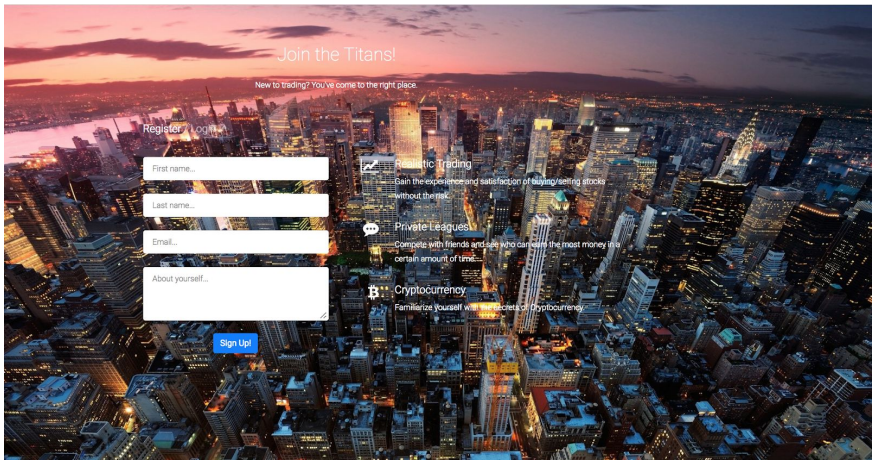
Greeting Page

Modification/Implementation:

Our current design relies heavily on the use of the NYC aesthetics to build a brand that can be commonly associated with finance, trading and modernity. However future considerations for the website may call for less complicated backgrounds that don't hide the UI and elements in the complicated mess.

Implementation of this greeting page is simple using our chosen frameworks and easy ability to create webpages and elements. This is the first page a non-logged in user would reach, and thus must be simple, inviting, and leave a good visual impression, more so than others pages on the website. The buttons, title, background image etc. were put together using Bootstrap as the front-end framework. All images used are open source, allowing us to set the desired aesthetic.

Sign Up/Login Page

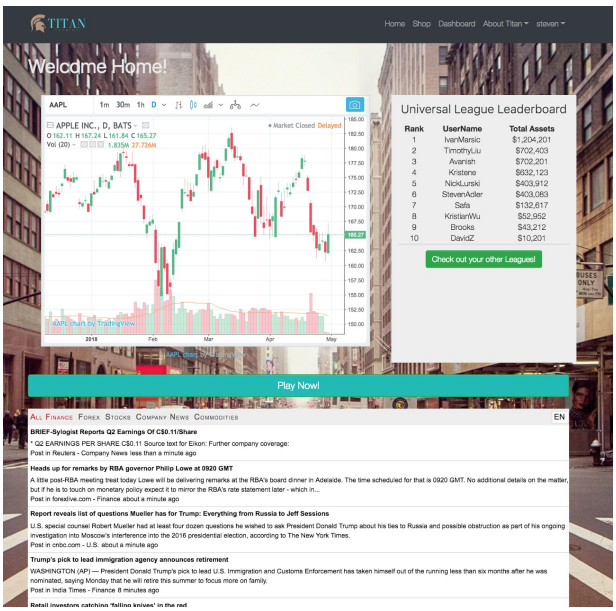


Modification/Implementation:

The sign up/login page is one of the most seen pages on the site, as every user will be thrown this page at some point or another during their visit. Signing up for an account should be as simple as possible, with very few questions that need to be filled. That way, users don't see signing up as a burden, rather a simple necessity. The implementation again relies on simplicity and and ease of use.

The sign-in page was flushed out from the simple gray bars and lines into a page that shows actually content and displays real interactive buttons and text boxes. This gives the page a more legitimate look and will allow us to easily integrate with the soon to be constructed backend.

Home Page



Modification/Implementation:

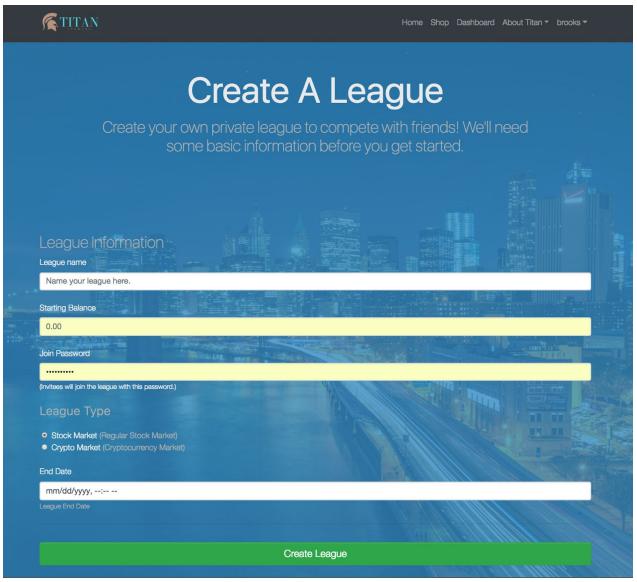
The home page is again one of the most viewed pages on the Titan Trading website. The home page was designed to be viewed as a hub of information, provided by external news sources, regarding cryptocurrency and stocks. This gives users and guests the experience of living as a real-life stock market trader, who makes decisions on previous data analytics and public news. All external sources will be clickable and will redirect the user to the original article.

In addition to all these news sources, there are two instances of Titan Trading specific UI tools. This is the "Play Now!" button and the universal league leaderboard. These two buttons are used to entice the player into playing the Titan Trading game.

With a similar theme to the sign-in page, we decided on a more simplistic look that was functional and clean. Square boxes were used to modularize information, and it makes it easily

readable to the audience. Compared to our initial mock-up of the home page, the functionality and layout of the page are almost identical, which reflects our initial intentions to provide the user with a simple and informative layout.

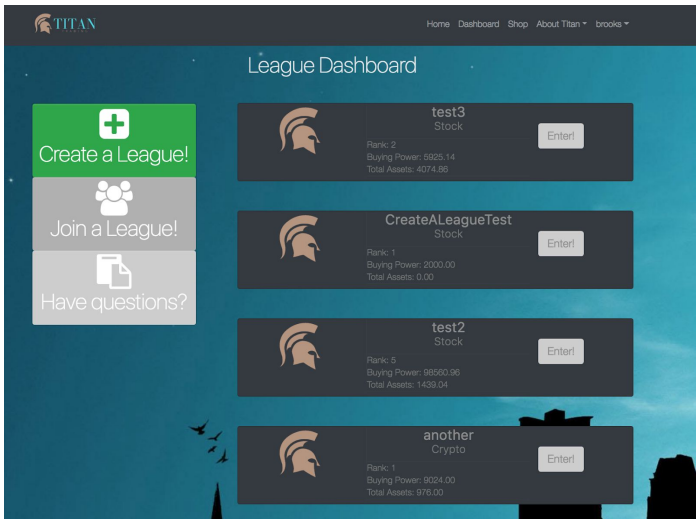
Create a League Page



Modification/Implementation:

The create a league page shown above prompts the aspiring league manager to put in necessary information such as league name, starting balance, league type, etc. There were minimal modifications in this section. The only clear modification is the background color, but content of the form stays the same.

League Dashboard

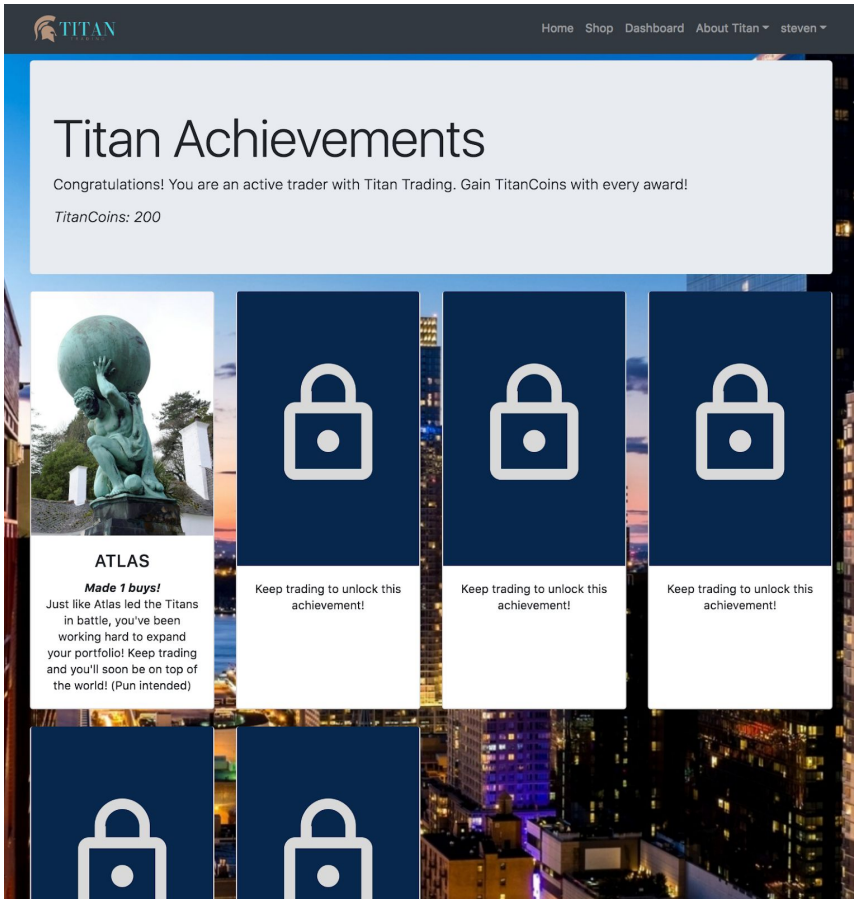


Modification/Implementation:

The League Dashboard page is another critical page in Titan Trading. This page can be viewed as the portal of entry into a given user's leagues. From the original mockups, the league dashboard has some undergone several important revisions. Initially, our dashboard page was in a chart format, which would have condensed important information in a small space. In our revision, we decided instead to change the dashboard into a picture gallery format.

A picture gallery format is easier for the user to visualize and will allow the user to make a more deliberate decision "to enter the league." This gallery format allows for a more personalized experience for users, since they can decide the photo they would like as their league photo. Any unused slots will be create or join a private league button.

Achievements Page



Modification/Implementation:

The trophies page displays user awards. This page is updated at the termination of every league, to reflect accurate TitanCoin amounts and reward the user for making smart trading choices.

We have 6 awards, each represented by a Titan:

1. Made several "buy" transactions - Atlas
2. Made several "sell" transactions - Metis

3. Top 3 in a league - Cronus
4. Winner of a league - Prometheus
5. Beat an AI - Hyperion
6. Managed a league - Uranus

Ease-of-Use Page by Page

Titan Trading's GUI provides an intuitive and easy navigation and participation experience. Through the use of conveniently placed and smartly named Buttons, the interface allows the user smoothing routing through the pages. When handling transactions, Titan Trading will generate a form for the specific transaction that will prompt the user for information.

Global Navigation Pane (All Pages)

Located at the top of each page, the Global Navigation Pane is a key feature that allows a user to navigate through the main pages of the website. This navigation bar gives users the benefit of conveniently transitioning to a desired page from any other page on the website.

Homepage

The homepage is the very first page that a user will visit when logged in. The homepage has a clean and simple design that includes a stock market widget that can potentially inform users about current stock trends, a Universal League leaderboard, and recent stock news articles. Under the Universal League leaderboard and stock widget, there is a "Play Now" and "See Leagues" button that will allow the user to transition to the dashboard page.

Dashboard

The dashboard page will display general information on leagues the user is a part of and acts as a link to specific league information. A user can click a league they are a part of to view detailed information on that leagues individual league page, or if they are not part of a private league, a "Join League" and "Create League" button will show up instead.

FAQ/Instruction/Glossary

The FAQ/Instruction/Glossary pages will answer key questions on how to get started with trading that new traders might have, instructions on how to buy/sell a stock and create/join a league, and definitions to key trading terms. The user can always view these pages to understand the function of each page and how to navigate through the GUI.

Create a League/Join a League/Buy a Stock/Sell a Stock

Each of these pages will have a form generated that will facilitate the process of each the pages main functions. The create a league and join a league page will generate forms asking for only necessary information such as league name, join password, date league ends, etc... The buy/sell a stock form will show the full name as well as the acronym of the stock, the league name that is being acted on, the amount of shares, price, etc... and will prompt a confirmation message with the transaction information to confirm the purchase/sale.

Preventing User Errors by Process

While Ease-of-Use is important to GUI design, it is also essential, especially when involving assets, that a GUI is designed to Prevent User Error. In the context of GUI design, User Error are actions that the user does unintentionally that might lead to unfavorable consequences. The processes in our stock market fantasy league most prone to User Error are buying a stock and selling a stock.

Preventing a User from Buying/Selling on the Wrong League

In our stock market fantasy league, a user can be part of four leagues, a universal league and three private leagues of choice. When Buying/Selling a stock it is essential that the user know which League is being affected. Thus, to ensure that the user knows which league is currently being acted on, a user must first be on an individual page for that specific league before buy and sell options are available.

Prevent a User from Selling the Wrong Stock

When choosing to sell a stock for a specific league, the GUI for the individual league sell page will provide a chart of which stocks the player owns and how many shares in that particular league. The player will only have the option of selling stocks from the chart so the player won't try to sell any stock they do not own.

13 DESIGN OF TESTS

Test Cases:

Player Control:

Test Case Identifier: **TC-1.1**

Function Tested: **addPlayer()**

Success/Fail Criteria: A successful test will send an invite to a player for the corresponding league.

Test Procedure:	Expected Results
Call Function (Success)	User receives an invite to the respective league on their user profile page. Function returns true.
Call Function (Failure)	No invite is sent to any user and the league membership is not changed. Function returns false.

Test Case Identifier: **TC-1.2**

Function Tested: **updateBuyingPower()**

Success/Fail Criteria: A successful test will properly update the player's buying power for that specific league.

Test Procedure:	Expected Results
Call Function (Success)	The buying power for a player associated with a specific league is updated.
Call Function (Failure)	The buying power for a player is updated for a different league.

Test Case Identifier: **TC-1.3**

Function Tested: **updateWorth()**

Success/Fail Criteria: A successful test will properly update the player's net worth in that specific league.

Test Procedure:	Expected Results
Call Function (Success)	The worth of a player in a specific league is updated.
Call Function (Failure)	The worth of a player is updated in a different league.

Test Case Identifier: **TC-1.4**

Function Tested: **issueTrophy()**

Success/Fail Criteria: A successful test will properly display the new trophy on the user's profile.

Test Procedure:	Expected Results
Call Function (Success)	The new trophy can be viewed from the user's profile page.
Call Function (Failure)	The new trophy cannot be found on the user's profile page.

League Control:

Test Case Identifier: **TC-2.1**

Function Tested: **addSetting()**

Success/Fail Criteria: A successful test will add the desired setting to the new league.

Test Procedure:	Expected Results
Call Function (Success)	The league will reflect the new setting in its description. Function returns true.
Call Function (Failure)	The league's description does not contain the new setting. Function returns false.

Test Case Identifier: **TC-2.2**

Function Tested: **addLeague()**

Success/Fail Criteria: A successful test will add a league to the database and make the creating user the league manager.

Test Procedure:	Expected Results
Call Function (Success)	The league is recorded in the database and the user is made the league manager. Function returns true.
Call Function (Failure)	No league is added to the database and the user is not made a manager of any league. Function returns false.

Test Case Identifier: **TC-2.3**

Function Tested: **updateUserLeagueIDs()**

Success/Fail Criteria: A successful test will update one of the user's League ID fields to match the new league they are joining.

Test Procedure:	Expected Results
Call Function (Success)	One of the user's League ID fields will be updated to reflect the new league they are joining. Function returns true.
Call Function (Failure)	The user does not have a League ID assigned for the league they joined. Function returns false.

Test Case Identifier: **TC-2.4**

Function Tested: **checkJoin()**

Success/Fail Criteria: A successful test will verify that the user was issued an invite to join the league.

Test Procedure:	Expected Results
Call Function (Success)	The user's invitation matches one of the ones issued for that league. Function returns true.
Call Function (Failure)	No invitation matches the user's invitation. Function returns false.

Test Case Identifier: **TC-2.5**

Function Tested: **sendInvite()**

Success/Fail Criteria: A successful test will verify that an email invite was sent to the specified recipient.

Test Procedure:	Expected Results
Call Function (Success)	Email invitation sent out to proper address, allowing new user to join.
Call Function (Failure)	No invite was sent out, function returns false.

Browser Functionality:

Test Case Identifier: **TC-3.1**

Function Tested: **updateDash()**

Success/Fail Criteria: A successful test will display the homepage with up-to-date market trends.

Test Procedure:	Expected Results
Call Function (Success)	User is redirected to homepage which properly displays data retrieved from Finance API. Function returns true.
Call Function (Failure)	User is redirected to homepage which displays either out-of-date or no data. Function returns false.

Test Case Identifier: **TC-3.2**

Function Tested: **searchStock()**

Success/Fail Criteria: A successful test will return the queried stock results to the user after consulting the Finance API using the given stock search terms.

Test Procedure:	Expected Results
Call Function (Success)	User is shown the stocks matching the search terms in the buy widget. Function returns true.
Call Function (Failure)	User is given an error response depending on what caused the issue - invalid text input or inability to reach Finance API. Function returns false.

Test Case Identifier: **TC-3.3**
Function Tested: **buy()**

Success/Fail Criteria: A successful test will record the transaction in the database and assign the user the correct assets while subtracting from the user's balance.

Test Procedure:	Expected Results
Call Function (Success)	Transaction changes are recorded in the database and browser reflects these changes. Function returns true.
Call Function (Failure)	User is given an error response depending on what caused the issue and no changes are made to the database. Function returns false.

Test Case Identifier: **TC-3.4**

Function Tested: **sell()**

Success/Fail Criteria: A successful test will record the transaction in the database and increase the user's balance while removing the corresponding assets from the user.

Test Procedure:	Expected Results
Call Function (Success)	Transaction changes are recorded in the database and browser reflects these changes. Function returns true.
Call Function (Failure)	User is given an error response depending on what caused the issue and no changes are made to the database. Function returns false.

Test Case Identifier: **TC-3.5**

Function Tested: **login()**

Success/Fail Criteria: Visitor is logged into their account and status changed to User.

Test Procedure:	Expected Results
Call Function (Success)	Visitor's web page is redirected to personalized dashboard. The visitor status is updated to user.

Call Function (Failure)	Visitor's web page is redirected to the incorrect web page or the visitor status is not updated properly to user.

Test Case Identifier: **TC-3.6**

Function Tested: **logout()**

Success/Fail Criteria: User is logged out their account and status changed to Visitor.

Test Procedure:	Expected Results
Call Function (Success)	User's web page is redirected to the visitor homepage. The user status is updated to visitor to reflect the logout.
Call Function (Failure)	User's web page is redirected to the incorrect web page or the user status is not updated properly to visitor.

Transaction Control:

Test Case Identifier: **TC-4.1**

Function Tested: **addTransaction()**

Success/Fail Criteria: A successful test will add the transaction in the database with the corresponding changes.

Test Procedure:	Expected Results
Call Function (Success)	Transaction is successfully recorded in the database. Function returns true.
Call Function (Failure)	No changes are made to the database. Function returns false.

Test Case Identifier: **TC-4.2**

Function Tested: **deleteTransaction()**

Success/Fail Criteria: A successful test will delete the transaction in the database with the corresponding changes.

Test Procedure:	Expected Results

Call Function (Success)	Transaction is successfully removed from the database. Function returns true.
Call Function (Failure)	No changes are made to the database. Function returns false.

Signup Manager:

Test Case Identifier: **TC-5.1**

Function Tested: **addUser()**

Success/Fail Criteria: A successful test test will add a unique user to the user table with all the information filled out on the signup form.

Test Procedure:	Expected Results
Call Function (Success)	A new user entry is added to the user table with all required information entered such as username, password, email, etc. Return true.
Call Function (Failure)	A new user entry is not inputted successfully. Failures could include no new user entry, lack of required information, or an overwritten entry. Return false.

StockAsset Control:

Test Case Identifier: **TC-6.1**

Function Tested: **addStockAsset()**

Success/Fail Criteria: A successful test will add a stock asset entry to the stock asset table.

Test Procedure:	Expected Results
Call Function (Success)	A new stock asset entry is recorded in the stock asset table with the number of shares and the value. Return true.
Call Function (Failure)	A new stock asset entry is not inputted successfully. Failures could include no new stock asset entry, invalid inputs (ex. negative number of stocks), or an overwritten entry. Return false.

Test Case Identifier: **TC-6.2**

Function Tested: **updateStockAsset()**

Success/Fail Criteria: A stock asset entry is updated correctly with the appropriate information.

Test Procedure:	Expected Results
Call Function (Success)	Stock asset entry is updated with the new information inputted. Return true.
Call Function (Failure)	Stock asset entry is not update correctly. Failures could include no update or an incorrect update. Return false.

User:

Test Case Identifier: **TC-7.1**

Function Tested: **setPassword()**

Success/Fail Criteria: A new password is updated for the User.

Test Procedure:	Expected Results
Call Function (Success)	Old password is overwritten and updated with a new password. User is now able to login with the new password and only the new password. Return true.
Call Function (Failure)	User is unable to login with the new password. The user's password has not been updated properly. Return false.

Test Case Identifier: **TC-7.2**

Function Tested: **setLeague()**

Success/Fail Criteria: A new league is added under the user object.

Test Procedure:	Expected Results
Call Function (Success)	A new league is added sequentially under the four available league spots for each user. Return true.
Call Function (Failure)	A new league is unable to be added. The user is already a part of four leagues and needs to leave a league to join another one. Return false.

Test Case Identifier: **TC-7.3**

Function Tested: **setEmail()**

Success/Fail Criteria: A new password is updates for the User.

Test Procedure:	Expected Results
Call Function (Success)	Old email is overwritten and updated with a new unique email. User is now able to login with the new email and only the new email. Return true.
Call Function (Failure)	Either the email inputted was not unique or the user is unable to login with the new email. Return false.

Game Upgrades:

Test Case Identifier: **TC-8.1**

Function Tested: **purchaseTitanCoins()**

Success/Fail Criteria: Money is given by the user in exchange for TitanCoins, TitanCoins should be added to the user account.

Test Procedure:	Expected Results
Call Function (Success)	TitanCoins are added to the user account, and a PayPal transaction is processed.
Call Function (Failure)	A users PayPal transaction fails, TitanCoins will not be added to a users account since they did not pay for them. Return false.

Test Case Identifier: **TC-8.2**

Function Tested: **purchaseGameUpgrade()**

Success/Fail Criteria: Money is given by the user in exchange for TitanCoins, TitanCoins should be added to the user account.

Test Procedure:	Expected Results

Call Function (Success)	TitanCoins are deducted from a user account and an upgrade is applied to their account.
Call Function (Failure)	User does not have enough TitanCoins to cover the purchase or they already have the purchase active on their account.

Artifical Intelligence Opponent:

Test Case Identifier: **TC-9.1**

Function Tested: **enableAI()**

Success/Fail Criteria: Set isAi to be true for a new player and add to the league you are enabling AI for.

Test Procedure:	Expected Results
Call Function (Success)	TitanCoins are added to the user account, and a PayPal transaction is processed.
Call Function (Failure)	A users PayPal transaction fails, TitanCoins will not be added to a users account since they did not pay for them. Return false.

Test Case Identifier: **TC-9.2**

Function Tested: **upgradeAI()**

Success/Fail Criteria: Set the AI difficulty for a league to be a different difficulty from the one it currently is.

Test Procedure:	Expected Results
Call Function (Success)	AI difficulty is set to a different setting than the one it is currently set for. Return true.
Call Function (Failure)	Difficulty attempted to set to is the same as current difficulty or user does not have a AI enabled for their league. Return false.

Test Case Identifier: **TC-9.3**

Function Tested: **diasbleAI()**

Success/Fail Criteria: Turn off the AI player for a league.

Test Procedure:	Expected Results
Call Function (Success)	AI player is disabled for a league.
Call Function (Failure)	User does not have a AI enabled for their league. Return false.

Communication:

Test Case Identifier: **TC-10.1**

Function Tested: **registerNumber()**

Success/Fail Criteria: Associate a phone number for text communication with a user. Fails if number is already associated with another user account.

Test Procedure:	Expected Results
Call Function (Success)	Number is added to profile entry in database, user can now buy and sell shares through text.
Call Function (Failure)	Number is already associated with a different user. Function returns false.

Test Case Identifier: **TC-10.2**

Function Tested: **phoneTransaction()**

Success/Fail Criteria: Execute transaction that user sent through text, texts back error message if transaction is not valid.

Test Procedure:	Expected Results
Call Function (Success)	Assets or funds are added to the users account, a confirmation receipt is sent back to the user.
Call Function (Failure)	Transaction is invalid, return false.

Test Coverage:

Our tests aim to cover all parts of the system from front end user interaction to the backend data handling. The main parts of the system to test are user account creation, league creation and changes, database modification, and the accessibility of browser pages. The test cases may involve multiple subsystems to confirm that the communication between the subsystems are working as expected. For most of the test cases, we are testing to see that the functions updates database correctly and that any changes are properly reflected in the game. By testing each individual component of the game separately, we are able to more specifically pinpoint any errors that may occur and in which method isn't performing properly.

Integration Testing Strategy:

Before assembling the different subsystems in the project, the test cases will be tested by the individual coding subgroups on their respective portions of the project. Then, after internal test cases are passed, the subsystems will be combined. The formal test cases will then be tested to confirm that integration has worked successfully. Many test cases encompass multiple subsystems, so they are proper tests for the integration of the system. Here are some test cases and the subsystems they test:

TC-2.2: addLeague() - Uses Browser, Database
TC-3.1: updateDash() - Uses Browser, Finance API
TC-3.2: searchStock() - Uses Browser, Finance API, Database

14 HISTORY OF WORK

Since the beginning of the semester there has been a significant amount of progress made towards the project through the accomplishment of various milestones and reports. This section details the history of that work including milestone accomplishment dates and commits. Note that this section does not encompass the entire project's work yet, but will once the full project has been completed. This section entails a broad summary, as well as a listing of milestones completed:

Below is a summary of our major accomplishments in the project so far as well as a discussion of future plans to take this project:

- Successfully set up a fully consistent user interface for a website from scratch. This includes all of the custom design work on things such as:
    - Wireframes of the interface
    - Color scheme and logo
    - UI design schemes
    - Navigation maps
    - Other design schemes found earlier in this report

- Develop, train and integrate fully functional AI opponents that can successfully execute any of the operations a typical user can. This includes:
    - Researching typical machine learning concepts and methods
    - Reading over the Tensorflow documentation
    - Creating a connection between our finance API, the AI, and the platform to allow for real trades
    - Training our machine learning algorithms against real Twitter and news data to make trades

- Implement Django as our main framework with which we can connect the frontend and backend components of the website. This includes
    - Researching dependencies
    - Setting up front end UI elements to query database on all pages
    - Properly follow HTTP web protocols to send JSON formatted data to APIs and widgets that appear on the website

- Construct and plan a full PostgreSQL database that can hold data related to the user and site as a whole. This accomplishment includes:
    - Drawing out and wireframing a database scheme including users, roles, and tables
    - Setup a testing environment so that all developers have access to their own database to test on separately, ensuring comprehensive testing of the system

    - Connect database to ElephantSQL so that all queries are live in a cloud interface

- Construct key features using Django to interface with our backend. These fully implemented features thought out in our user stories including:
    - A sign-in and login system for users including full authentication
    - Creating a new league with invitation code
    - Joining a currently operating league with an invitation code
    - Buying stocks and cryptocurrency available on the market
    - Selling stocks and cryptocurrency available in a user's portfolio
    - A user profile page page to display relevant information

- Integrating Twilio with our Django framework to give users another option for buying stocks, instead of restricting the user to using a single way of purchasing.
    - Setting up a Twilio developer account and setup guide
    - Follow documentation to properly integrate the buy functionality to be received from a SMS url setup by Twilio
    - Tie the integration to a valid US phone number

- Create a functional shop interface where users can spend real money on in-game items that give real impact to the game experience
    - Creating a fresh interface that clearly displays the various options and price points
    - Integrating Paypal as a payment method for a safe and secure transactions
    - Creating a virtual currency, TitanCoins, as is common in most games that use in-game purchases

- Research and implement multiple plugins and widgets to give the user more experience. These widgets include:
    - A non-tailored general financial news widget
    - A tailored financial news widget that will adjust to a user's portfolio
    - A stock lookup widget that allows users to search for any stock or cryptocurrency and see a chart of its performance.

- Creating a Heroku instance that can host and deliver content related to Titan Trading.
    - Create a hosting environment on a valid Heroku app to host all data including front-end templates, Python scripts and the Django modules
    - Register a valid domain that serves all content related to the website
    - Handle requests over a valid TCP connection, instead of a localhost connection over port 8000

- Additional small User Experience (UX) choices to better enhance a given user's ability to use the website such as:
    - Receipt printing of all transactions made
    - Navigation bar to easily switch between relevant pages
    - Well placed frames and buttons to display what is needed of the user

- Input validation to ensure that the user puts in valid data as input for the forms

What follows are the full listing of milestones that have been completed:

Milestone #1 - Front End implementation of pages
Due 3/11/18

The report does not call for implementation, but the pages that are wireframed should nonetheless be implemented in some front end capacity. There may be no real data that is drawn, but have a working front end is the first step to using Django and bringing our concepts into a reality.

Milestone #2 - Configuration and setup of database
Due 3/16/18

The backend will be a PostgreSQL database and requires some configuration and setup before implementation can begin. Once this is constructed, the different aspects of the project that rely on this shared infrastructure can be implemented.

Milestone #3 - Writing up a FAQ and About Titan Page
Due 3/11/18

The front end pages such as the FAQ are simple and only serve to enhance the presentation of the demo. They are not a major requirement as they present no real functional aspects, but they are needed for good presentation.

Milestone #4 - User authentication and logging in/logging off
Due 3/18/18

User authentication is directly tied to the backend and getting data from the back end. User authentication is an important first step in constructing a shared infrastructure. Milestone #5 is closely tied to this milestone. See ST-1, ST-2 and the associated acceptance tests AT-1, AT-2 for the tasks that must be completed and tests.

Milestone #5 - Connecting account page front end and user data
Due 3/21/18

A user should be able to go to their account page and change the values for basic preferences and their profile picture. This must be accomplished after the shared database infrastructure is completed. The changes to profile picture, username, and other basic front-facing account information should be able to be changed, with the results being displayed on the front end.

Milestone #6 - Connecting the buy page to the finance API
Due 3/21/18

Displaying real data on the buy page, in a simple non-visual context, is the first step to getting our product to getting real-time information. This can visually be simple but so long as the API can properly be used to get correct numbers, the milestone can be accomplished.

Milestone #7 - Connect third-party widgets for search and company information
Due 3/23/18

The ability for a user to search and get information using third-party widgets is an important functionality that comes with the buy and sell page. This includes both:

- Seeing the stock prices for buying
- Seeing the result of a transaction

Milestone #8 - Connecting the Universal League to a user's portfolio
Due 3/23/18

A given user's data should also include their portfolio. In other words the list of assets that are owned by a user should be showing on the dashboard page for that particular user. At this point there will be no individual leagues implemented yet, so these will be a single portfolio in the single universal league.

Milestone #9 - Connecting the Sell page to a user's portfolio
Due 3/23/18

In the opposite direction, a user should be able to sell their stocks back to the market and have money move into their buying power. This is the dual of the buy page and moves money and assets in the opposite direction. This is a key functionality that will be extended to any league and is a key feature.

Milestone #10 - Create a League form connects to back end and saves values
Due 3/29/18

Once the buy and sell functionality is built in, users now have the features needed to form their own leagues. This starts with ensuring that the form that users fill out is not just a basic UI form.

It should save the preferences that a user wants set to the back end. These won't have any effect yet, but the values for that league should be kept in the back end.

Milestone #11 - 'Join a League' invitation interface

Due 4/12/18

As user must be able to send out invites to their friends in order for them to join the league. The first milestone related with this would involve sending an email to the users that the league administrator (user who created the league) wants to invite. This milestone involves connecting our backend to some email service/API, which is important for other aspects later and merits its own milstone.

Milestone #12 - Ensure buy/sell functionalities display to the user and reflect database changes

Due 4/12/18

An important addition that was desired was the addition of cryptocurrency to the typical stock trading game. At this point the project already has transaction infrastructure and the ability to well display information. The addition of cryptocurrencies complicates the product more horizontal than vertically. This means that current systems simply have to be expanded to include this new dynamic, rather than producing entire new systems from scratch.

Milestone #13 - Allow users to print out transaction receipts to PDF files

Due 4/18/18

An important addition that was desired was the addition of cryptocurrency to the typical stock trading game. At this point the project already has transaction infrastructure and the ability to well display information. The addition of cryptocurrencies complicates the product more horizontal than vertically. This means that current systems simply have to be expanded to include this new dynamic, rather than producing entire new systems from scratch.

Milestone #14 - Add cryptocurrency data with "Crypto-leagues"

Due 4/22/18

An important addition that was desired was the addition of cryptocurrency to the typical stock trading game. At this point the project already has transaction infrastructure and the ability to well display information. The addition of cryptocurrencies complicates the product more horizontal than vertically. This means that current systems simply have to be expanded to include this new dynamic, rather than producing entire new systems from scratch.

Milestone #15 - Tensorflow Machine Learning and AI

Due 4/30/18

The use of artificial intelligence and machine learning is central to our product's ability to properly educate users and deliver a meaningful trading simulation experience. The idea of having computer opponents has been common in video games over the past few decades. But now with advances in artificial intelligence and machine learning we are looking to apply this to real stock data and building multiple competitive AI players.

Milestone #16 - In-game shop

Due 4/30/18

Most modern games have some sort of store or marketplace where users can use real money to buy additional content for the games that they play. Since our product is a free-to-play game, wherein no payment is required to get started, we needed a viable way of enhancing the game's experience while also presenting a realistic avenue for making money. Our goal is to build a shop where users can buy boosts, customization options, and even different AIs to compete against.

Milestone #17 - Targeted News

Due 4/30/18

As we learned in our first demo, targeted news is an important feature that can deliver a much more comprehensive experience for all users. Our goal is to look at a given user's portfolio and delivered a tailored news experience based on the assets that they hold. This could mean news about not only the assets that they do hold, but also the stocks that are similar, whether that be competitors, similar priced stocks, or stocks in the same industry.

Milestone #18 - Asset Listings

Due 4/30/18

Portfolios are an important part of any trader's life, and as such we need to not simply list a given user's assets, but also display them in a simple and meaningful way. This includes the amount of stock, the price bought at, the current price, and other important metrics for them to notice and use during their trading times.

Milestone #19 - Trophies and Achievements

Due 4/30/18

Rewarding the user for good practices is a game mechanic that serves to enhance the overall experience of the user, and keep them coming back to play the game. The user should get rewarded for goals on a lifetime basis, as well as in individual leagues. These achievements and rewards enhance the game aspects and give users a reason to keep coming back and breaking their records.

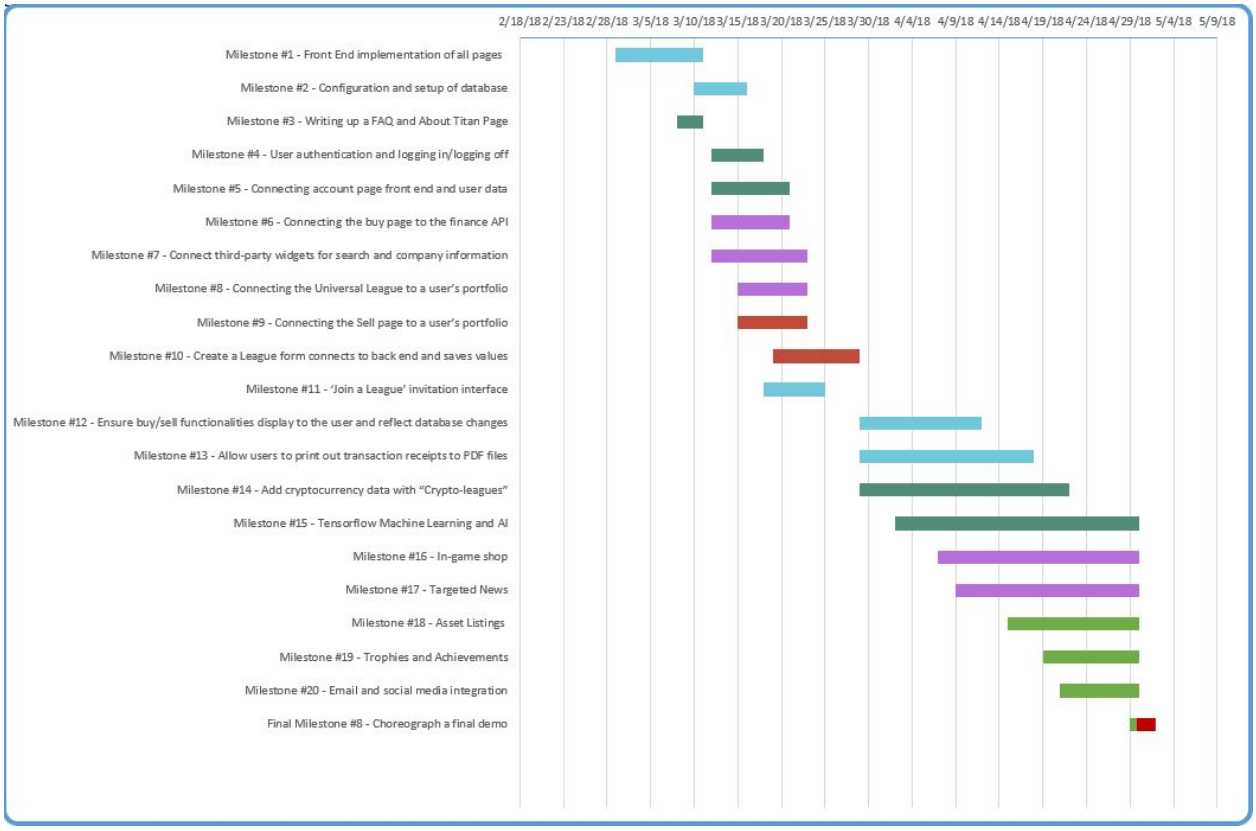
Milestone #20 - Email and social media integration

Due 4/30/18

The use of the league level password to join the league, needs to work in tandem with inviting new users to the league, both upon creation as well as while the league is running. This can also be extended to social media, such as Facebook, where users can make a post on facebook to their timelines inviting people to sign up and join. All that invited users need to do is make a profile and use the attached league password.

14.7 Gantt chart

Task Name	Start Date	End Date	Duration (Days)	Days Complete	Days Remaining	Percent Complete
Milestone #1 - Front End implementation of all pages	3/1/2018	3/11/2018	10	10.00	0.00	100%
Milestone #2 - Configuration and setup of database	3/10/2018	3/16/2018	6	6.00	0.00	100%
Milestone #3 - Writing up a FAQ and About Titan Page	3/8/2018	3/11/2018	3	3.00	0.00	100%
Milestone #4 - User authentication and logging in/logg	3/12/2018	3/18/2018	6	6.00	0.00	100%
Milestone #5 - Connecting account page front end and	3/12/2018	3/21/2018	9	9.00	0.00	100%
Milestone #6 - Connecting the buy page to the finance	3/12/2018	3/21/2018	9	9.00	0.00	100%
Milestone #7 - Connect third-party widgets for search	3/12/2018	3/23/2018	11	11.00	0.00	100%
Milestone #8 - Connecting the Universal League to a u	3/15/2018	3/23/2018	8	8.00	0.00	100%
Milestone #9 - Connecting the Sell page to a user's por	3/15/2018	3/23/2018	8	8.00	0.00	100%
Milestone #10 - Create a League form connects to bac	3/19/2018	3/29/2018	10	10.00	0.00	100%
Milestone #11 - 'Join a League' invitation interface	3/18/2018	3/25/2018	7	7.00	0.00	100%
Milestone #12 - Ensure buy/sell functionalities display t	3/29/2018	4/12/2018	14	14.00	0.00	100%
Milestone #13 - Allow users to print out transaction re	3/29/2018	4/18/2018	20	20.00	0.00	100%
Milestone #14 - Add cryptocurrency data with "Crypto	3/29/2018	4/22/2018	24	24.00	0.00	100%
Milestone #15 - Tensorflow Machine Learning and AI	4/2/2018	4/30/2018	28	28.00	0.00	100%
Milestone #16 - In-game shop	4/7/2018	4/30/2018	23	23.00	0.00	100%
Milestone #17 - Targeted News	4/9/2018	4/30/2018	21	21.00	0.00	100%
Milestone #18 - Asset Listings	4/15/2018	4/30/2018	15	15.00	0.00	100%
Milestone #19 - Trophies and Achievements	4/19/2018	4/30/2018	11	11.00	0.00	100%
Milestone #20 - Email and social media integration	4/21/2018	4/30/2018	9	9.00	0.00	100%
Final Milestone #8 - Choreograph a final demo	4/29/2018	5/2/2018	3	0.75	2.25	25%



15 FUTURE PLANS

While we were able to successfully implement a lot of the goals that we set out for the project, there are still more features that can be added and paths to take in future development. We essentially wanted to answer the question “What would we do if we had another semester?” Some of these plans are large and haven’t really been thought through, while others are smaller things that we couldn’t fit into our final demo without proper testing and development. These future plans are thus categorized into three categories based on the size and scope of the plan and its impact on the project: Minor, Moderate, and Major

Minor Plans

- More Social Media Integration - While the project does use social media as a way of sharing league and other information, there could always be more social media integration with other aspects of the site. Attaching social media to various parts of the site would allow users to stay more engaged with their actions and ultimately keep them more engaged with the game.
- More Machine Learning Algorithms and AI - More is always better! Developing more computer opponents can only give the experience more depth and add more possibilities for the user. This all comes down to researching more algorithms and training. One extension that we would like to explore is using convolutional neural networks and deep learning for AI prediction.
- Optimization of Machine Learning Algorithms - Currently the machine learning algorithm run for a moderately long time. We would like to be able to optimize these algorithms to evaluate stocks with a more breadth in less time.
- Profile Customization Options

Moderate Plans

- Data Visualization - Displaying information to users is a key to any good user interface. And while our interface does this for assets in a nice row formatted way, there is no avenue for displaying this information in the form of a graph or other data visualization scheme.
- Mentor-to-Student Chats and Forums - We would like to make this website more social. Although friends can interact through social media and can invite each other to leagues, we would like to introduce ways on our site for friends to interact. Whether this be a chat room or forum, we would like to increase the communication among users. More specifically, an opportunity for advanced traders to interact with novice traders would bring a new aspect to educating our users.

Major Plans

- Accurate Prediction of Stocks - We had initially planned to include some form of predictive graphing interface that the user could use to more accurately predict the price of a given asset. However with more careful and thorough research we realized that this

is a problem that multi-million dollar firms with far more resources are tackling on their own, and to get involved with that process we simply do not have the resources or time to focus on that. Instead of projecting potentially false or misleading information to the user, we decided to let them choose where they get their finance predictions from.

- Mobile Application - As our product stands, there is a solid web based solution, but it would be great to refactor this code into a native mobile application for Android and iOS devices. This would allow users to be engaged more through all hours of the day and, if integrated properly with the backend, can provide a seamless and continuous experience across multiple platforms.

16 REFERENCES

"Choosing a Business Model." Apple.com. https://developer.apple.com/app-store/business-models/

Benita, Haki. "How to Manage Concurrency in Django Models – Haki Benita – Medium."*Medium*, Medium, 7 July 2017, medium.com/@hakibenita/how-to-manage-concurrency-in-django-models-b240fed4ee2.

Barker, Jonathan Todd. "Why Is Bitcoin's Value So Volatile?" *Investopedia*, 2 Jan. 2018, www.investopedia.com/articles/investing/052014/why-bitcoins-value-so-volatile.asp. [Online, accessed 17 February, 2018]

Fontinelle, Amy. "Perfect Competition." *Investopedia*, 29 Jan. 2018, www.investopedia.com/terms/p/perfectcompetition.asp. [Online, accessed 17 February, 2018]

I. Marsic, Software Engineering. New Brunswick, USA: Ivan Marsic, 2012

Investopedia. "Efficient Market Hypothesis - EMH." *Investopedia*, 28 Sept. 2016, www.investopedia.com/terms/e/efficientmarkethypothesis.asp. [Online, accessed 17 February, 2018]

Marr, Bernard. "A Short History Of Bitcoin And Crypto Currency Everyone Should Read." *Forbes*, Forbes Magazine, 6 Dec. 2017, www.forbes.com/sites/bernardmarr/2017/12/06/a-short-history-of-bitcoin-and-crypto-currency-everyone-should-read/#35cfdb633f27. [Online, accessed 17 February, 2018]

"The Paramount Investments League." *Rutgers University*, 8 May. 2014, http://www.ece.rutgers.edu/~marsic/books/SE/projects/TradingLeague/2014-g1-report3.pdf. [Online, accessed 17 February, 2018]

"Rutgers University Investing." *Rutgers University*, 12 May. 2014, http://www.ece.rutgers.edu/~marsic/books/SE/projects/TradingLeague/2014-g2-report3.pdf. [Online, accessed 17 February, 2018]

"User & Design Specifications for "Capital Games." *Rutgers University*, 3 May. 2013, http://www.ece.rutgers.edu/~marsic/books/SE/projects/TradingLeague/2013-g2-report3.pdf. [Online, accessed 17 February, 2018]

"Tweepy Documentation". Docs.tweepy.org. http://docs.tweepy.org/en/v3.5.0/getting_started.html#introduction

"TextBlob: Simplified Text Processing." Textblob. http://textblob.readthedocs.io/en/dev/

"NewsAPI Documentation." NewsAPI. https://newsapi.org/docs

"Loss Function." Wikipedia, Wikimedia Foundation, 21 Nov. 2017, https://www.en.wikipedia.org/wiki/Loss_function

"Machine Learning Blog & Software Development News." Datumbox, https://www.blog.datumbox.com/tuning-the-learning-rate-in-gradient-descent/

Ruder, Sebastian. "An Overview of Gradient Descent Optimization Algorithms." Sebastian Ruder, Sebastian Ruder, 23 Nov. 2017, https://www.ruder.io/optimizing-gradient-descent/

"TensorFlow Mechanics 101 | TensorFlow." TensorFlow, www.tensorflow.org/get_started/mnist/mechanics

Alexandr Honchar. "Neural networks for algorithmic trading." Medium.com. https://medium.com/machine-learning-world/neural-networks-for-algorithmic-trading-part-one-simple-time-series-forecasting-f992daa1045a

"Convolutional Neural Network." Wikipedia. https://en.wikipedia.org/wiki/Convolutional_neural_network

"Bootstrap: Getting Started." W3Schools. https://www.w3schools.com/bootstrap/bootstrap_get_started.asp