SE Stock League

Report 3
Software Engineering
14:332:452

Group 18

Joseph Coleman
Murali Gunti
Matthew Jackson
Himateja Madala
Andrew Marfitsin
Akshaykumar Patil
Paul Stanik
David Wang

May 5, 2019

Contributions Breakdown

[This contribution breakdown relates to the writing of report 3 (combination of both reports 1 and 2).]

Joseph Coleman (GUI Management):

-On-Screen Appearance Requirements - 70%
-User Interface Specification - 33%
-Effort Estimation - 100%
-Interaction Diagrams(Description) - 33%
-System Architecture and Design - 20%

Murali Gunti (Product Functionality)

-User Interface Specification - 33%
-Plan of Work - 100%

Matthew Jackson (Website Operator and Advertiser Interaction)

-Problem Statement - 33%
-Glossary of Terms - 80%
-User Stories - 100%
-Nonfunctional Requirements - 50%
-On-Screen Appearance Requirements - 30%
-Functional Requirements Specification[Stakeholders, Actors and Goals, Use Cases(Use Case Diagram, Traceability Matrix, Fully-Dressed Description), System Sequence Diagrams] - 40%
-Domain Analysis - 100%
-Interaction Diagrams(Diagrams) - 33%
-Class Diagram and Interface Specification - 100%
-Data Structures - 100%
-Design of Tests - 100%

-References - 100%

Himateja Madala (Portfolio and Market Insight Tools, Achievements)

-Plan of Work - 33%

Andrew Marfitsin (Portfolio and Market Insight Tools, Achievements)

-Problem Statement - 33%

-Nonfunctional Requirements - 50%

-Functional Requirements Specification[Use Cases(Casual Description] - 30%

-Plan of Work - 33%

-Glossary of Terms - 10%

Akshaykumar Patil (GUI Management)

-Interaction Diagrams(Description) - 33%

-System Architecture and Design - 40%

Paul Stanik (Website Operator and Advertiser Interaction/Product Functionality)

-Problem Statement - 33%

-Plan of Work - 33%

-User Interface Specification - 33%

David Wang (Product Functionality)

-Functional Requirements Specification[Use Cases] - 30%

-Glossary of Terms - 10%

-System Architecture and Design - 40%

Summary of Changes

+ Changed the contribution breakdown to better reflect the updated team oriented setup.
+ Added references.
+ Added early gantt chart.
+ Updated Glossary of Terms to include “security” as a term.
+ Updated Use Cases (1), (3), (4), and (10) for clarification.
+ Updated section 3.4 Use Case (1) system sequence diagram to be more consistent.
+ Added definitions.
+ Added candlestick charts to 14.3 Plan of Action.
+ Site has been changed from https://matthew0405.github.io/SE_Stock_League/ to https://sestockleague.epizy.com. This new website is hosted under InfinityFree which supports built in PHP databases through mySQL and phpMyAdmin. This move needed to be made in order to allow users to register an account, login, join leagues, and begin investing.
+ Project Size has been updated and renamed to Effort Estimation.
+ Updated Breakdown of Responsibilities.
+ Added Design Patterns under Interface Diagrams

Contents

1. Customer Statement of Requirements

1.1 Problem Statement

Investing can be very intimidating for anyone starting out, especially when real money is involved. In order to prevent investment failures from discouraging a newcomer from continuing in the field, we will provide a more comfortable and fun investment platform that gets rid of any investing anxieties and bridges the gap between higher and lower level investing. A good way to introduce a novice to the world of investing is through a simulation that features true-to-life trading mechanics, but involves fantasy money. This way, the investor-to-be has the opportunity to learn from mistakes and explore market trends without putting real money on the line.

To further simplify and de-stress the experience, the user will be offered a tutorial and walkthrough of the features offered by the application, and how to use them. This will include pointers on navigating the application interface and where to find functions such as the user's own stock portfolio, the user's group or league membership, and other views. Depending on the user's comfort level and knowledge, additional, more complex functions will be (optionally) offered to make the experience as engaging to a seasoned investor as it would be to a newcomer. These features, such as the mutual fund vs. portfolio graphs, "shorting" stock, and otherwise would be offered incrementally, with the passage of time and the user's advancement in the game, and not included in the basic first tutorial so as not to confuse a novice player.

To make the experience more engaging and exciting, in-game achievements will be implemented, rewarding the user for successful investing strategies. These achievements could range in difficulty from something like "first week with a net portfolio value gain" to "first 5% profit". The user would also be shown achievements that have not yet been reached, and the requirements for them, giving the user concrete goals to work towards. Achievements could vary for groups, to give leagues of players more challenging goals to work towards together.

Users of the application will have the option to join groups and communicate with each other, to make the experience more social if the user so chooses. Additionally, "Leagues" can be formed among players. The purpose of the league is to introduce competition among players by ranking the players within the league based on a parameter such as percentage of profit. League

settings/rules can also be changed by the league manager(the user that creates the league), and the league manager would have the option of adding/removing players from the league as well. Messaging will be implemented between individual users and within leagues, such that users can effectively communicate their ideas on investment strategy and observations of the market at large. Each league will have a public message board that can be seen by all the users within the league, which would allow for league updates to be made and viewed on a regular basis. A "friend" system would be put in place, such that if two users have agreed to become friends within the game, they can see each other's portfolios and coordinate future investments. This is a useful step in making the social aspect of the game a clear option to the user, accommodating users who would want the social functionality and those who would prefer a more "single player" experience.

The user would also be offered insightful, advanced tools to aid in stock buying decisions, including per-stock price charts, portfolio comparison charts, and a historic value chart for the user's portfolio. These functions would expand the user's experience with the application to encompass more of the functions available to investors in the real world, and ease the user into familiarity with them. The user could compare their own portfolio against ETFs or index funds using the mentioned comparison charts, to observe the behavior of the market at large and evaluate how their trading decisions compare. These statistics could show the user whether or not professionally managed investment products (such as mutual funds) are really worth the price of entry.

Experience with data visualizations could help investors decide whether they prefer long-term planning, riskier short term, high payoff investments, or another trading strategy. This can be accomplished through the use of bar graphs, pie charts, box-and-whisker chart, and ext. Knowing this and having a general concept of the profit margins the user can expect that their strategy could later help in the investor's choice of brokerage account or investment product. In general, these are common tools offered by brokerage accounts to their users, and a functionality that the investor should become comfortable using.

2 Glossary of Terms

League - Users can join different leagues that each have their own rules such as start dates, end dates, starting cash, commission values, etc. This is where users will invest.

- **Public -** Any investor can join a public league. A public league can be made by a user who will then become the league manager.
- **Private -** A private league can only be entered by a request or through a unique code. A private league can be created a user.

League Manager - A league manager has the ability to make a league and decide whether it will be public or private. League managers set the rules for the league.
Site Administrator - A site administrator monitors the whole web application to make sure it is operating properly and can remove leagues.
Achievement - Goals set that reward users for good investments.
Marketable Securities - Financial instruments that can be converted into cash.
Investment Vehicle - A product used by investors for gaining positive returns.

- **Mutual Fund -** An investment vehicle that involves multiple investors pitching in money to invest into securities.
- **Exchange-Traded Fund(ETF) -** A type of security that tracks a stock index.

Security - A fungible, negotiable financial instrument that holds some type of monetary value/An ownership position in a publicly-traded corporation (via stock), a creditor relationship with a governmental body or a corporation (represented by owning that entity's bond), or rights to ownership as represented by an option.
Stock - A type of security that represents ownership of a company.

- **Ask Price -** Price at which a trader is willing to sell a stock.
- **Bid Price -** Price at which a trader is willing to buy a stock.
- **Bid-Ask Spread** - The amount by which the ask price exceeds the bid price. The difference between the highest price that a buyer is willing to pay for an asset and the lowest price that a seller is willing to accept to sell it.
- **Commission -** A charge due to a broker handling the purchase or sale of a security.

Order - An investor must place an order for the purchase or sale of a stock.

- **Market Order -** An order to buy or sell a security immediately at the best available price.
- **Limit Order -** An order to buy or sell a security by setting a maximum or minimum price.
- **Stop Order -** An order to buy or sell a security once the price of the stock reaches a specific price. Different from a market order or a limit order because a stop order is not active until the stock reaches that specific price. Once the stock reaches that price, the stop order changes to a market order.
- **All or None(AON) Order -** A type of order that guarantees an investor gets the entire amount of stocks they requested or none at all.
- **Good 'Til Canceled(GTC) -** A time restriction placed on an order that remain active until an investor cancels it.

Rights - Stockholders can buy new shares issued by a company at a predetermined price based on how many shares the stockholder already owns.
SSO - Single sign-on. A user can authenticate once and have access to all parts of the web application.
Stock Symbol (Ticker) - A unique set of letters that represent a security for trading purposes.
Symbol List - A list of the market's stock symbols.
Portfolio - A grouping of financial assets, owned in this case by the Investor.

3. System Requirements

3.1 User Stories

Identifier	User Story	Weight
ST-1	As a user, I can create an account through the web application using my email.	10 pts
ST-2	As a user, I can join as many leagues as I desire.	6 pts
ST-3	As a user, I may opt to create a league and become a league manager so I may have my own personal league.	10 pts
ST-4	As a user, I can search for a company by symbol or company name.	6 pts
ST-5	As a user, I can report another user who demonstrates negative behavior.	2 pts
ST-6	As a user, I will have access to a tutorial that introduces me to the web application by explaining different mechanics and tabs of the web app.	4 pts
ST-7	As a user, I will have access to a forum where I can talk to other users about different topics.	4 pts
ST-8	As a user, I can manage my portfolio to track investments.	8 pts
ST-9	As a user, I can buy or sell stocks so that I may build a portfolio.	10 pts

ST-10	As a user, I can browse a companies profile and view the performance data over a configurable span of time so that I may determine whether or not I want to invest in them.	6 pts
ST-11	As a user, I can see trades being made by all other users in real-time so I can have a quick overview of current trends	3 pts
ST-12	As a user, I can visually track my finances via graphs and charts so I may more easily manage my portfolio	4 pts
ST-13	As a user, I can view a portfolio leaderboard so I may have a summary of my performance in comparison to other users.	2 pt
ST-14	As a user, I can add other users and see their online status.	2 pts
ST-15	As a user, I can message other users individually or in a league.	2 pts
ST-16	As a user, I can recover or change my password so I may always have access to my own account.	5 pts
ST-17	As a user, I can access my profile and settings on a dashboard on the top of every page within the site.	8 pts
ST-18	As a user, I can view the stock prices in a candlestick chart.	8 pts
ST-19	As a league manager, I can add league rules, a league name, and a league logo to personalize my league.	8 pts
ST-20	As a league manager, I may manage players within the league so I may invite players I want to join, ban players that are being abusive, and assign other league managers.	8 pts
ST-21	As a league manager, I can moderate and delete comments in the league page.	4 pts

ST-22	As a league manager, I can create league announcements.	3 pts
ST-23	As a site administrator, I can view reports of users.	2 pts
ST-24	As a site administrator, I can delete abusive/offensive comments and ban users or IP addresses.	6 pts
ST-25	As a site administrator, I may post front page news or announcements.	3 pts
ST-26	As a site administrator, I may have access to a user count, number of active leagues, total leagues, quantity of daily transactions, the most/least popular stocks, and newly created or banned users so I may have reliable site statistics.	9 pts

3.2 Nonfunctional Requirements

Functionality

Additional features for security will be enabled through the use of a SQLite database API. A SQLite database is not very resource-intensive and provides more space than Web storage. SQLite allows for select and sort actions along with transactions that affect the state of the database, and is easily implemented with the webapp infrastructure. SQLite will also be used to store hashed passwords, provide recovery options for users that have forgotten their password, and store investment transactions. Locally hashed passwords improve speed of use and simplicity of implementing an SSO (single sign-on) system for the whole application and its sub-services. Alpha Vantage's developer toolset will be implemented to obtain stock data, both historic and current. The syntax is similar to that of SQL, and scripted queries can be used to serve data per the user's input, or navigation of the UI. This way, the application will have up-to-date and real security prices to make the experience authentic. Using an HTML and CSS security checklist, the web application will follow best coding practices to ensure sound security.

Usability

The web application should be designed in a way that is welcoming to new users but can offer more complex functions for veteran users. Using the CSS framework Bootstrap, we will be able to main a high quality web application with a clean and sleek UI. Through CSS, every page will have a header and navigation bar that offers access to all other pages of the web application.

Portfolio and stock price views can be implemented graphically using graph options built into Bootstrap, and ready to implement with a connection to the SQLite database and Alpha Vantage services for current and historic stock price visualizations. These views should be able to adapt per user query, relaying the request to Alpha Vantage, fetching the requested data, and displaying it. Javascript will be used to provide a higher level of responsiveness and it will be the main framework of the welcoming tutorial.

Reliability

In order to avoid inconsistency with a user's account, all transactions must be confirmed before being registered on the server. This is done so that in the case of internet or server failure, no changes to the account are made until after this confirmation, and inconsistent data is not retained by the server. As a result, the user's portfolio will remain in a consistent state and will be restored when the user is able to log back in. A user that leaves the application without logging out and returns later will remain logged in. The effects of server failure should also be mitigated by keeping backups of user data. In case of an incomplete set of database transactions or an error state, the last valid backup should be reverted to, in avoidance of completely stopping the user's experience with the application. It should also be noted to a user when a particular stock source is not available.

Performance

Using CSS and Javascript, the aim is to keep the web application running smoothly while keeping hardware demands to a minimum on both the client and server sides. The web application should be able to complete any task initiated by the user in a timely manner without many errors or stalls. The web server should be able to serve concurrent requests to handle when a large number of users are logged in. The use of SQLite reduces the use of computing resources and will offer a faster operating experience. SQLite only offers the core functions of SQL, thereby reducing its footprint and improving its responsiveness. Additionally, locally hashing passwords will remove the need to contact external SSO services, and thus reduce latency and increase the speed of authentication. To most simply and efficiently implement communication within the application between its users, ad-hoc server based messaging will be leveraged. This choice is made due to the small resource impact of the method and the small data size it will use, which will reduce network resource use and improve the speed of communication.

Supportability

The web application will be made with desktop users as the focus. Despite this, it will still be possible for users to login to the app and use it on their phones using a mobile web browser. This will be achieved by using dynamic webpage practices, which allow the on-screen interface to adapt to any screen size it is viewed on and retain usability. This would offer the greatest possible user reach and convenience in use of the application. Locally hashed passwords allow greater flexibility in implementing new security standards, such as a required password change or update, requirements towards the password length or complexity, and so on. Without

the need to contact an external SSO provider, authentication can be easily maintained and updated. The web application should be coded in a way that makes it easy to extend or update any server components and push improved versions of modules which can be installed by administrators. Preparation will be made to include an additional number of servers to achieve load balancing. The system should be backed up to a remote server for version experimentation and a rollback in the case that a new version creates a glitch.

3.3 On-Screen Appearance Requirements

Identifier	Requirement
OSR-1	The login page will display the web application's logo and present the user with a login/registration page.
OSR-2	Every page except for the login page will have a scrolling ticker at the bottom to update the users on stock prices.
OSR-3	The webapp will have text in the top right corner that shows the user's username and options to check their account settings, leagues they are a part of, and a search bar for leagues, stocks, companies, and currencies.
OSR-4	Each user will have an option to report them next to comments they leave on certain things.
OSR-5	New users will have a popup asking them if they would like to do the tutorial after their first login.
OSR-6	Leagues will display a leaderboard as well as stocks being bought and sold in that league and whether the league is private/public.

Figure 2.1: A mockup for a login screen

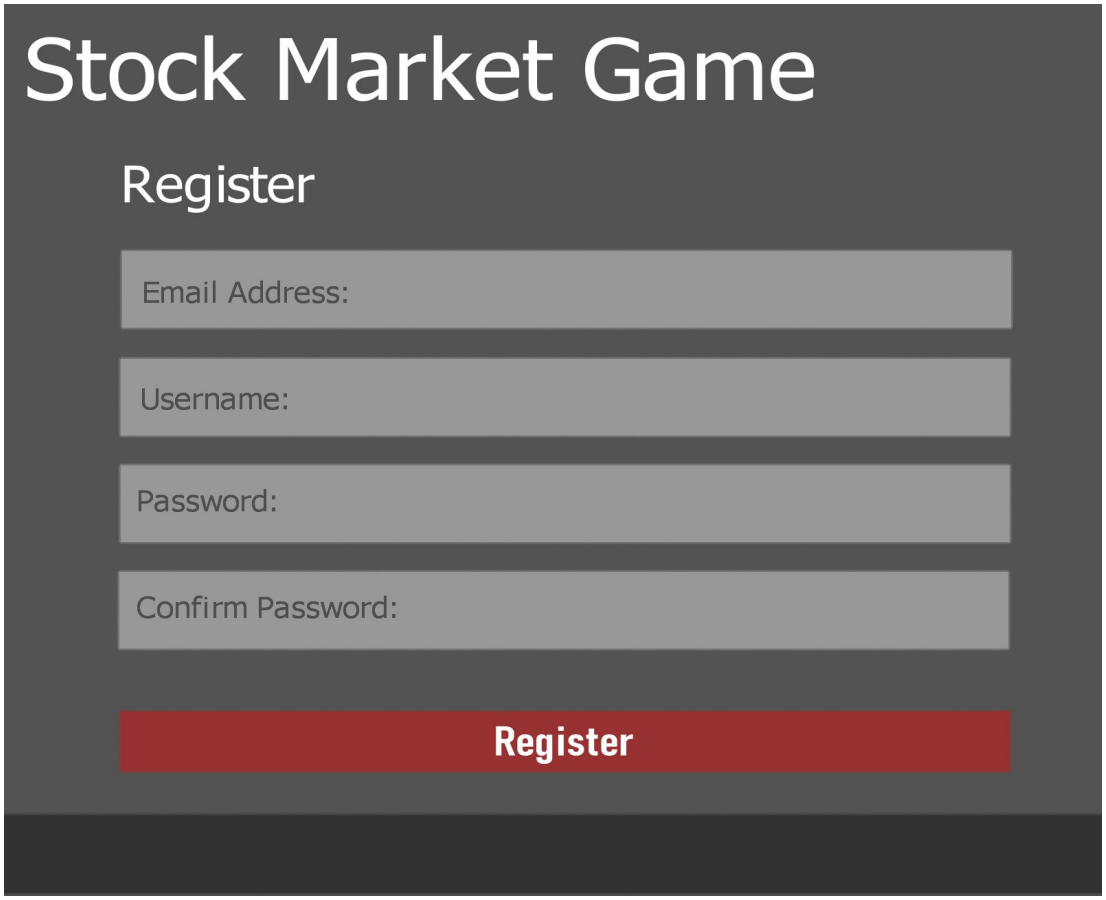


Figure 2.2: A mockup of the registration page

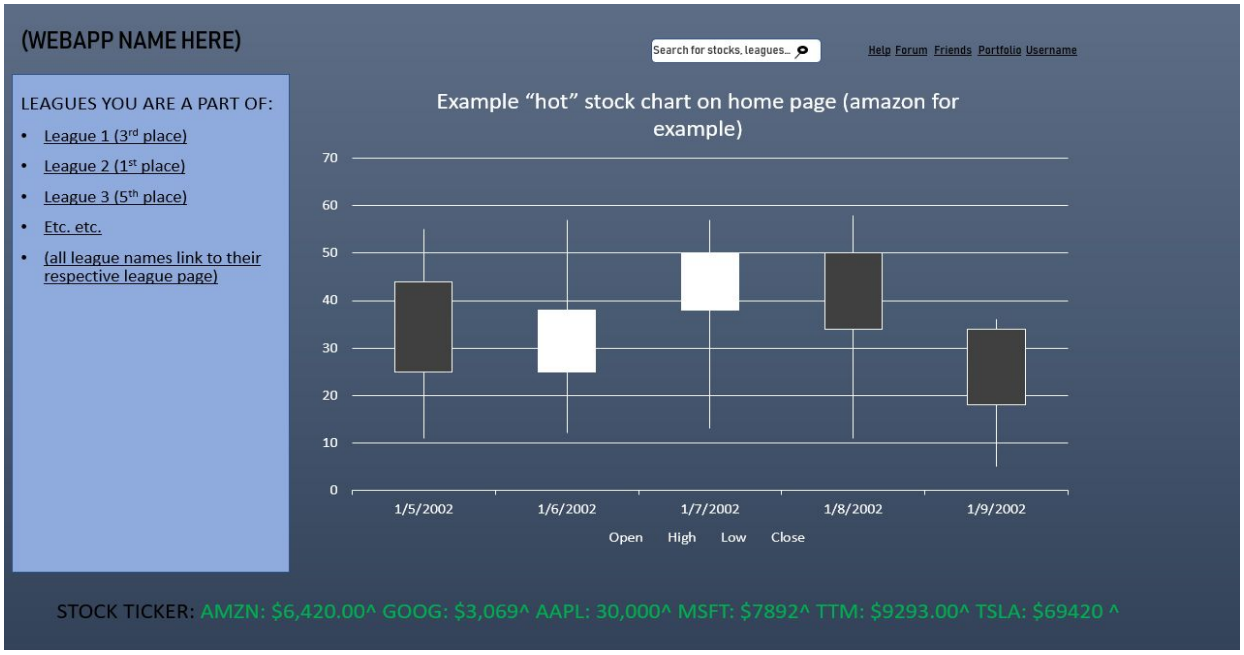


Figure 2.3: A mockup for the "home page"

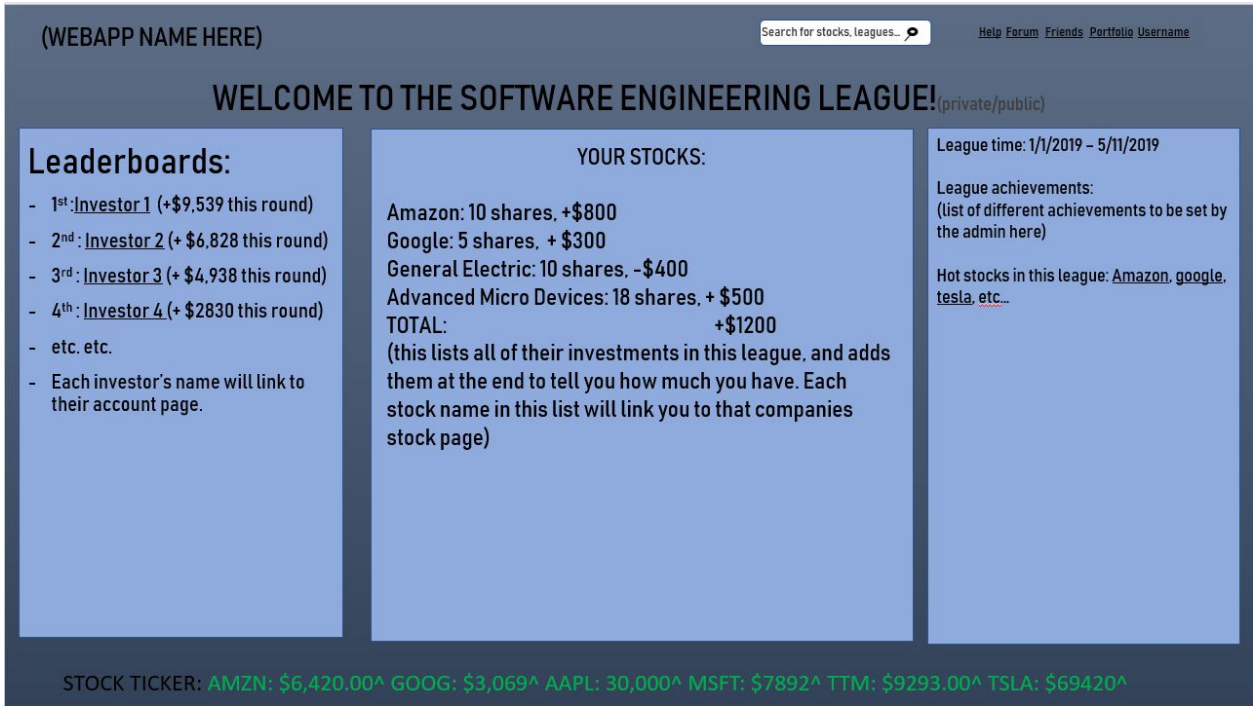


Figure 2.4: A mockup of a "league page"

4. Functional Requirements Specification

4.1 Stakeholders

Although the web application can be used by anyone interested in investing, one potential target of the web application would be students. High school students can be introduced to investing through the software to get a head start on learning investing tactics. High school students looking to major in economics would benefit from experimenting with the software and develop a heightened interest in the stock market. This same thing applies to college students with an economics major. Just like high school student, university students can use the software to learn financial concepts. Unlike high school students, college students are more likely to partake in real world investing, and so they can more seriously experiment with investing techniques through the web application and bring that knowledge over with them when engaging in real investing. As mentioned earlier, the software can be used casually by anyone looking to dip their toes into investing and is not limited to students.

The web application will be a free service but include advertisements placed throughout the webpage. This means that companies looking to promote their products or services can contact us to discuss marketing. These advertisements would be used to support the web application and allow for continual development of the platform. The use of advertisers would allow for the web application to remain free which will allow it to broadcast to the largest number of customers.

4.2 Actors and Goals

Guest

A user of the web application who has either not logged in or not registered.

- Create an account.
- View leaderboard and stock prices.

Investor

A user who has registered and logged into an account.

- Join and create leagues.
- View portfolio.
- View leaderboard and stock prices.
- Buy or sell securities.

League Manager

A user who has created or is in control of a league.

- Create a league competition by setting the rules.
- In charge of adding/removing players participating in the league
- Can set league to public or private.
- Edit league settings.

Site Administrator

Manages the whole web application.

- Remove users who exhibit inappropriate behavior.
- Make front page announcements and post news.

Database

Holds the information of all users' accounts.

- Retrieve information as soon as a user account is created.
- Store new data about users or events, including user portfolios.

Alpha Vantage API

Holds information about current financial statistics

- Retrieves information about stock prices and sends it to the database or live view in application.

4.3 Use Cases

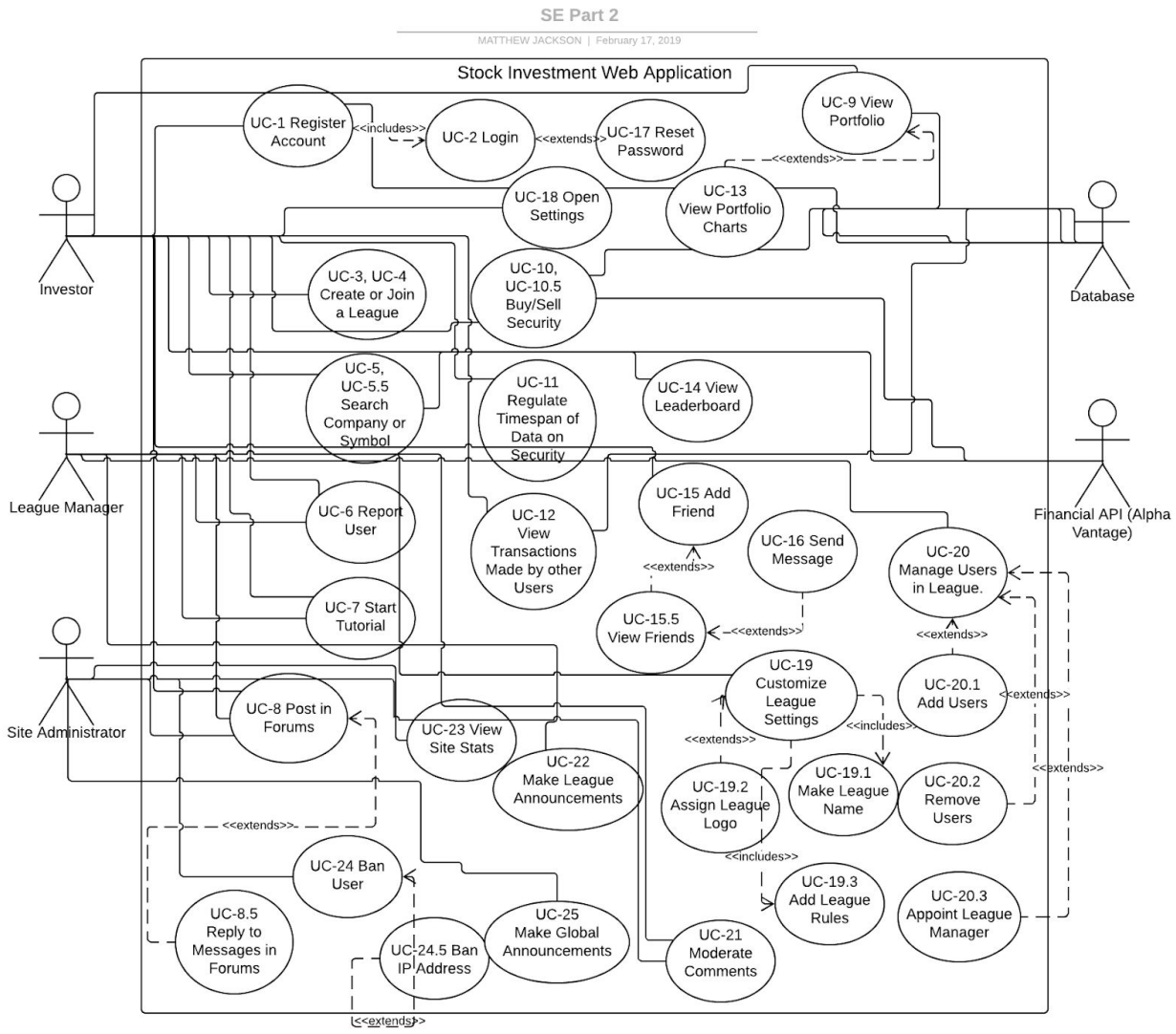
i. Casual Description

Use Case	Use Case Name	User Story	Actor	Actor's Goal
UC-1	CreateAccount	ST-1	Investor	To create an account using an email.
UC-2	Login	ST-1	Investor	To login to user account.
UC-3	JoinLeague	ST-2	Investor	To join any/all leagues desired.
UC-4	CreateLeague	ST-3	Investor/League Mgr	To create a league and become its manager.
UC-5	SearchCompany	ST-4	Investor	To search for company data using symbol or name.
UC-5.5	SearchSymbol	ST-4	Investor	Search non-company security using its symbol.

UC-6	ReportUser	ST-5	Investor	To report other abusive users.
UC-7	StartTutorial	ST-6	Investor	To be led through the basics of the application, and shown its functions.
UC-8	ForumPost	ST-7	Investor	To post a message in the user forum.
UC-8.5	ForumReply	ST-7	Investor	To reply to a posted message in the forum.
UC-9	PortfolioView	ST-8	Investor	To assess the status of all investments using a portfolio.
UC-10	BuySearched	ST-9	Investor	To purchase a found security.
UC-10.5	SellOwned	ST-9	Investor	To sell owned security.
UC-11	SetTimespan	ST-10	Investor	To regulate the timespan of data provided on a chosen security.
UC-12	ViewOtherTransactions	ST-11	Investor	To see transactions made by other users.
UC-13	PortfolioChartView	ST-12	Investor	To assess one's own portfolio's status via data visualizations such as charts.
UC-14	LeaderboardView	ST-13	Investor	To compare one's own profits to those of other users in a leaderboard format.
UC-15	AddFriend	ST-14	Investor	To "add" other users as friends.
UC-15.5	ViewFriend	ST-14	Investor	To view added users' online status.
UC-16	SendMessage	ST-15	Investor	To send messages to friends and/or league members.
UC-17	ResetPwd	ST-16	Investor	To recover or change password if need be.
UC-18	OpenPortfolio	ST-17	Investor	To access own profile/portfolio from any active page.
UC-18.5	OpenSettings	ST-17	Investor	To access settings from any active page.
UC-19	CustomizeLeague	ST-18	League Mgr	To customize league settings.
UC-19.1	LeagueName	ST-18	League Mgr	To customize league name.
UC-19.2	LeagueLogo	ST-18	League Mgr	To customize league logo.
UC-19.3	AddLeagueRule	ST-18	League Mgr	To customize league rules.
UC-20	ManageUser	ST-19	League Mgr	To manage user membership in a league.

UC-20.1	AddLeagueUser	ST-19	League Mgr	To manage user membership: add league users.
UC-20.2	DelLeagueUser	ST-19	League Mgr	To manage user membership: remove league users.
UC-20.3	AppointMgr	ST-19	League Mgr	To appoint other league managers.
UC-21	ModComment	ST-20 /23	League Mgr/Site Admin	To moderate/remove league page comments.
UC-22	LeagueAnnounce	ST-21	League Mgr	To issue announcements to owned league.
UC-23	SiteStats	ST-22 /ST-25	Site Admin	To view reports on users activity, including user count, active and total league number, daily transactions, favored stocks, created/banned users.
UC-24	BanUser	ST-23	Site Admin	To issue user bans.
UC-24.5	BanIP	ST-23	Site Admin	To issue IP bans.
UC-25	GlobalAnnounce	ST-24	Site Admin	To issue news or announcements directly on the application site.
UC-26	GetCompanyInfo	ST-10	Investor	To view historical stock data on a certain company to aid investors in buying decisions

ii. Use Case Diagram



iii. Traceability Matrix

User Story	PW	UC-1	UC-3	UC-5	UC-7	UC-8	UC-9	UC-10	UC-19	UC-23
ST-1	10	X								
ST-2	6		X							
ST-3	10		X							

ST-4	6			X						
ST-5	2									X
ST-6	4				X					
ST-7	4					X				
ST-8	8						X			
ST-9	10							X		
ST-10	6			X			X			
ST-11	3						X			
ST-12	4						X			
ST-13	2						X			
ST-14	2									
ST-15	2									
ST-16	5	X								
ST-17	8									
ST-18	8								X	
ST-19	8								X	
ST-20	4								X	
ST-21	3								X	
ST-22	2									X
ST-23	6									X
ST-24	3									X
ST-25	9									X
Total Priority		15	16	12	4	4	23	10	23	20

iv. Fully-Dressed Description

Use Case UC-1	Register an Account
Related Requirements:	ST-1
Initiating Actors:	Guest
Actor's Goal:	Register with our servers to enter the database.
Participating Actors:	Guest, Database
Preconditions:	-The guest must not be a registered user. f
Postconditions:	-The Database gets updated with information about the user and the guest becomes an Investor.
Flow of Events for Main Success Scenario	
→ ← ← →	1. Guest visits the web application and attempts to register. 2. Database is accessed and checks that an investor is not found. 3. Guest is registered as an investor in the database. 4. Registration is confirmed to the user and start portfolio is displayed.
Flow of Events for Extensions	
→ ← ←	1. Guest visits the web application and attempts to register. 2. Database is accessed and checks that investor is not already registered. 3. Message signaling that the account already exists is displayed to the user.

Use Case UC-3,4	Create/Join a League
Related Requirements:	ST-2, ST-3, ST-18, ST-19, ST-20, ST-21
Initiating Actor:	Investor
Actor's Goal:	Create or participate in a league that will hold competitions. No limitations on amount of leagues created/joined.

Participating Actors:	Database, Investors
Preconditions:	-User is logged in. -League is not already created. -Investor is not already in league.
Postconditions:	-The league is created with all settings filled in. -The Investor joins the league. -Database is updated about new league and investors in it.
Flow of Events for Main Success Scenario	
→ ← → ←	1. Investor navigates to leagues page. 2. All public leagues are displayed to the Investor. 3. Investor selects to join a league. 4. Investor is registered into the league and the Database is updated.
Flow of Events for Extensions	
3a. The investor selects to create a league.	
→ ← ←	1. Investor inputs league name and settings. 2. The league is created and updated in the Database. 3. Investor is turned into the League Manager of the league.
4a. Investor attempts to join or create a league without permission.	
←	1. The request is rejected and the reason for the rejection is displayed.

Use Case UC-5	Browse Companies
Related Requirements:	ST-4, ST-10
Initiating Actor:	Investor
Actor's Goal:	View information on securities and companies.
Participating Actors:	Database, Alpha Vantage
Preconditions:	-Investor is logged in. -Alpha Vantage is up and running appropriately.
Postconditions:	-Correct companies and security prices must be displayed.

Flow of Events for Main Success Scenario	
→ ← → ←	1. Investor begins searching for a company. 2. Investor is given suggestions based on the characters inserted. 3. Investor enters their own search or clicks on a suggestion. 4. Information is retrieved from Alpha Vantage and is displayed to the Investor.
Flow of Events for Extensions	
3a. Investor's search is invalid.	
←	1. A message is displayed that the entered company does not exist and similarly named companies are displayed.

Use Case UC-10	Place Market Order
Related Requirements:	ST-9
Initiating Actor:	Investor
Actor's Goal:	To purchase or sell a security through a market, limit, or stop order.
Participating Actors:	Database, Alpha Vantage
Preconditions:	-Investor is logged in. -Alpha Vantage is up and running appropriately. -Investor is a member in a league. -Time is between 9:30 AM ET to 4:30 PM ET (Normal trading day hours)
Postconditions:	-Investor's portfolio is updated based on order.
Flow of Events for Main Success Scenario	
→ ← → ←	1. The investor places an order from inside a league. 2. Prompt for order type, stock amount, and company are displayed. 3. Investor fills out the prompt and the order is placed. 4. Market price is received from Alpha Vantage and places the order into the Database.
Flow of Events for Extensions	

1a. The Investor places an order from a company's profile and not from inside a league.	
→ ←	1. The Investor selects which league to place the order in. 2. The same prompt explained in Step 2 above is displayed to the investor.
4a. The Investor does not have enough funds or margin to place an order.	
←	1. A message alerting the Investor that they do not have enough funds or margin is displayed and the order is prevented from completing.

Use Case UC-9	Manage Portfolio
Related Requirements:	ST-8, ST-12, ST-13
Initiating Actor:	Investor
Actor's Goal:	View current standings and securities.
Participating Actors:	Database
Preconditions:	-Investor is logged in. -Investor is a member of a league.
Postconditions:	-Correct portfolio information must be displayed.
Flow of Events for Main Success Scenario	
→ ←	1. Investor opens up portfolio. 2. Portfolio information is retrieved from the Database and is displayed to the Investor.

Use Case UC-7	Start Tutorial
Related Requirements:	ST-6
Initiating Actor:	Investor
Actor's Goal:	To learn how to navigate the web application while learning about investing.
Participating Actors:	None
Preconditions:	-Investor is logged in.

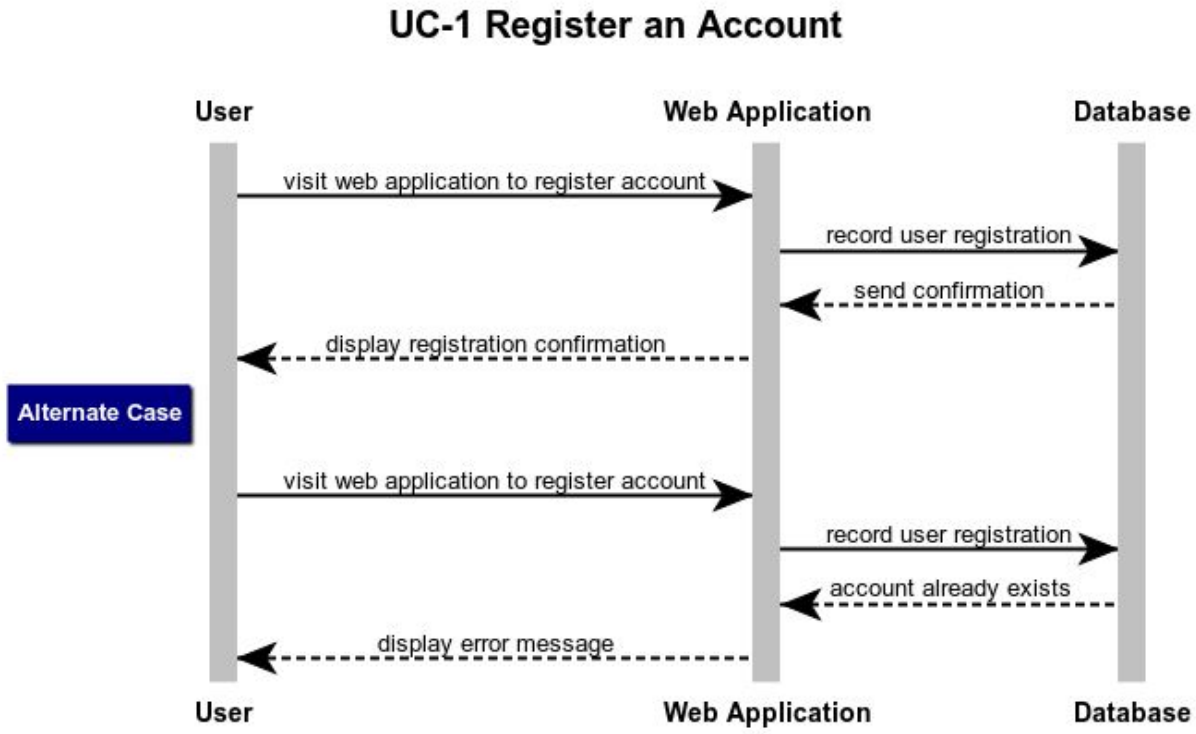
Postconditions:	-It will be marked that the user has completed the tutorial.
Flow of Events for Main Success Scenario	
→ ← ←	1. Investor selects tutorial option. 2. What will be taught in the tutorial is displayed. 3. The investor is taken through a step-by-step tutorial on the workings of the web application.

Use Case UC-23	Take Administrative Actions
Related Requirements:	ST-22, ST-23, ST-24, ST-25
Initiating Actor:	Site Administrator
Actor's Goal:	Manage the website and the Database
Participating Actors:	Database, Investors, League Managers
Preconditions:	-User is the Site Administrator.
Postconditions:	-The Database is updated to reflect any actions taken by the Site Administrator.
Flow of Events for Main Success Scenario	
→ ←	1. The Site Administrator requests logs from the Database. 2. Database sends the data requested by the Site Administrator to the Site Administrator.

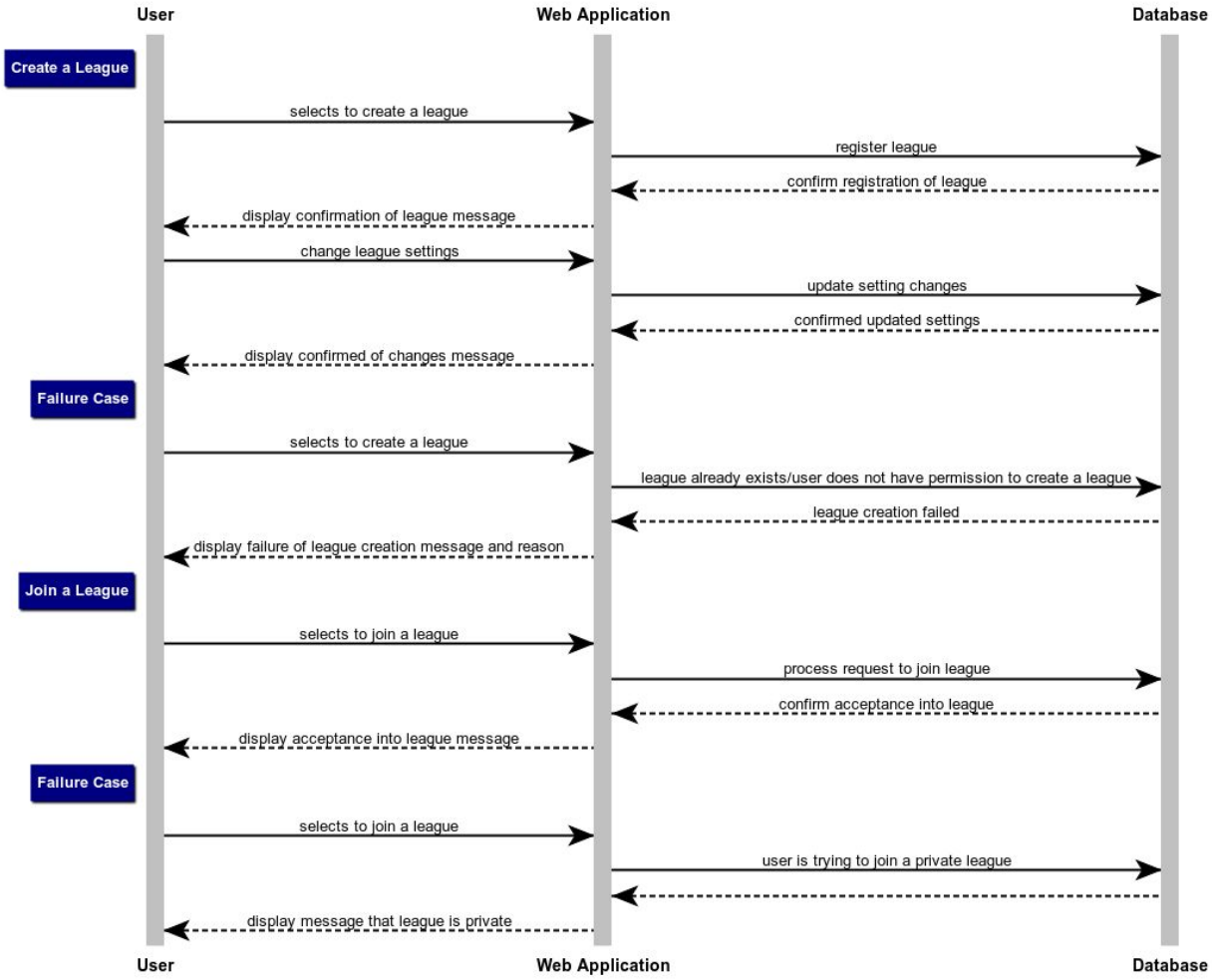
Use Case UC-19	Manage League Settings
Related Requirements:	ST-18, ST-19, ST-20, ST-21
Initiating Actor:	League Manager
Actor's Goal:	Set/Change league settings
Participating Actors:	Database, Investors
Preconditions:	-User is a League Manager.

Postconditions:	-Database is updated to reflect the changes made by the League Manager.
Flow of Events for Main Success Scenario	
→ ← ←	1. League Manager makes adjustments to league settings. 2. Database is updated with the setting changes. 3. The updated changes are displayed to the League Manager.

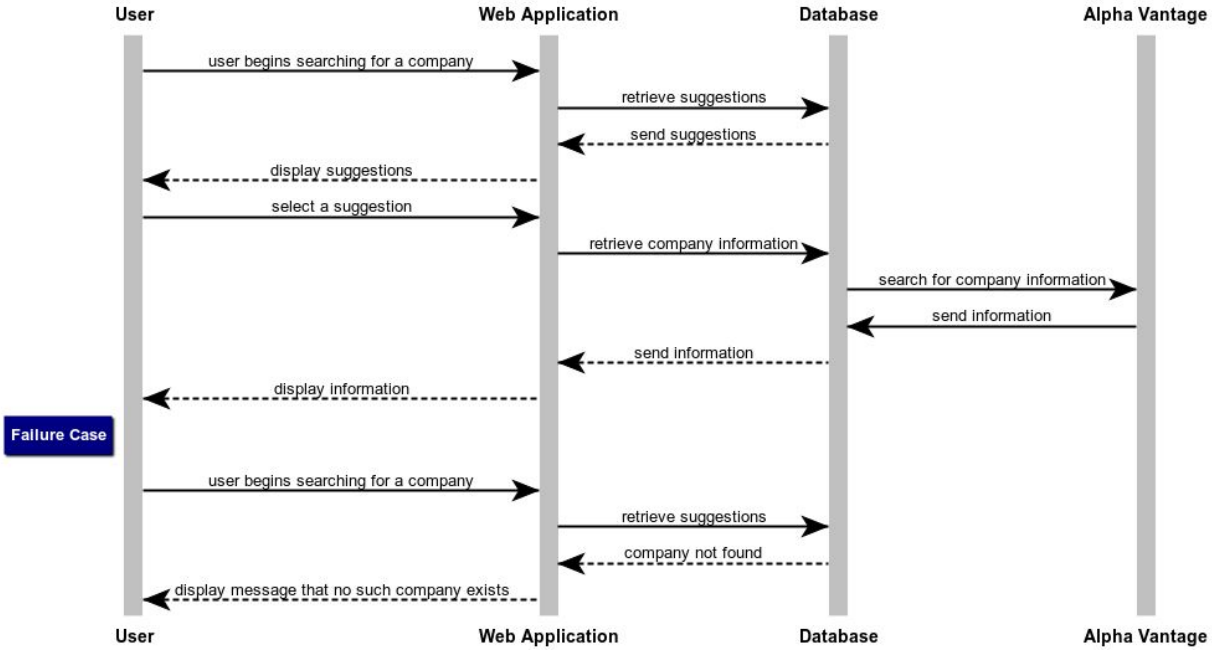
4.4 System Sequence Diagrams



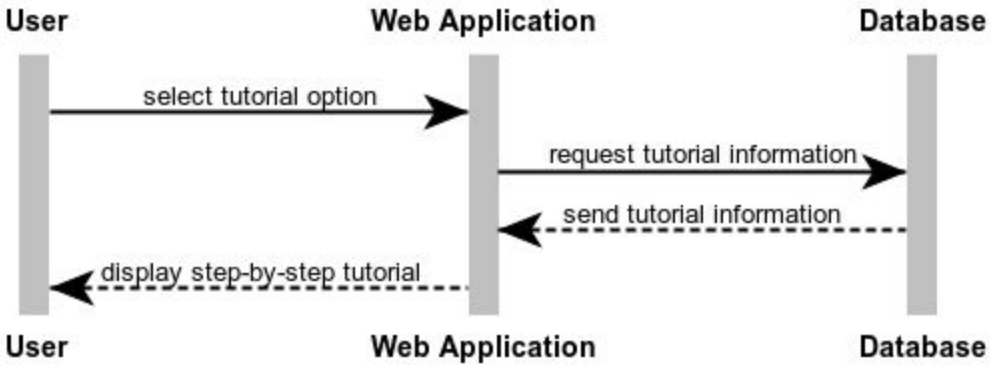
UC-3,4 Create/Join a League



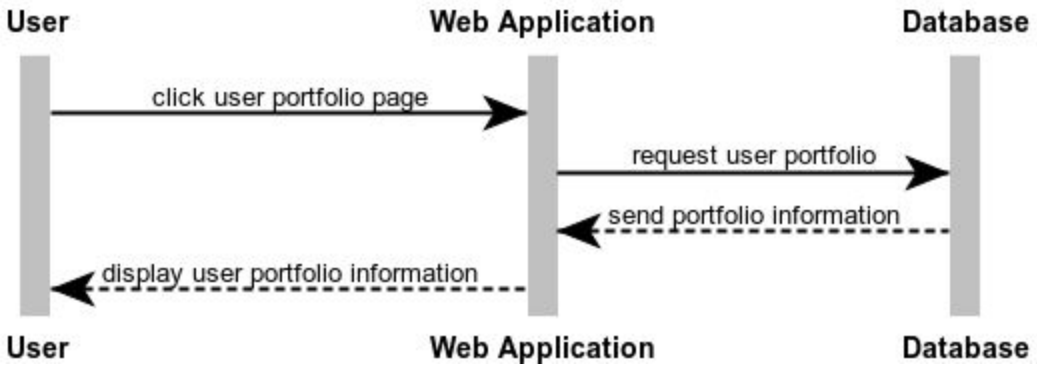
UC-5 Browse Companies



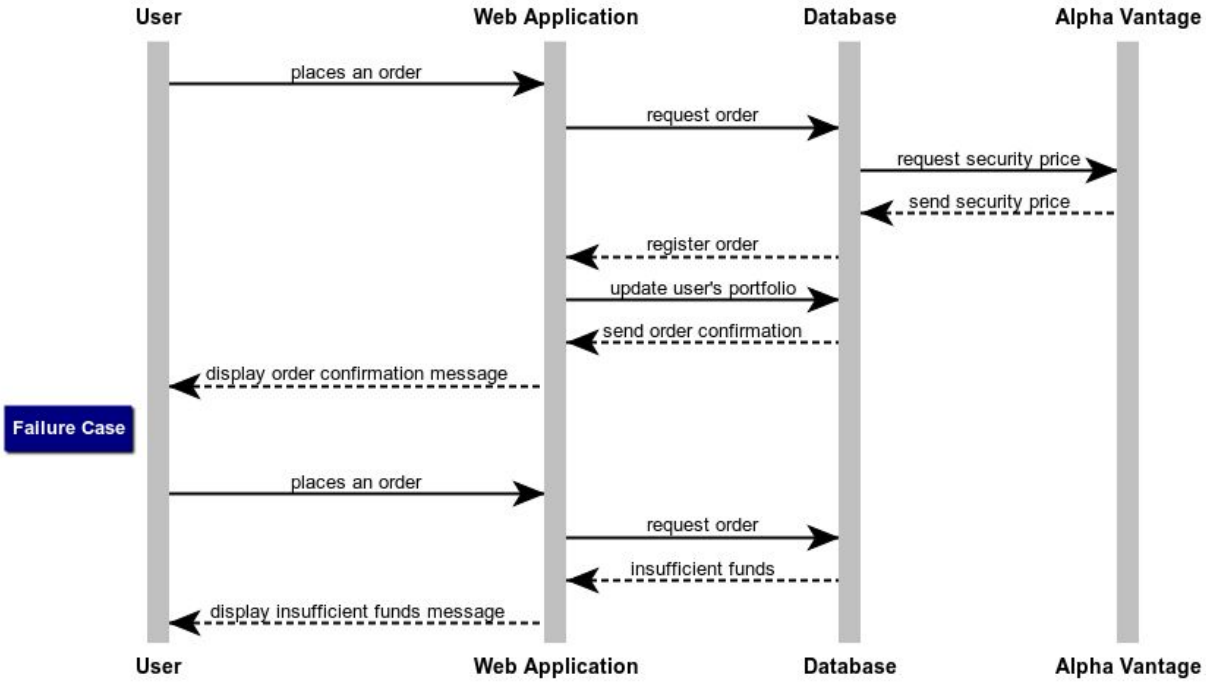
UC-7 Start Tutorial



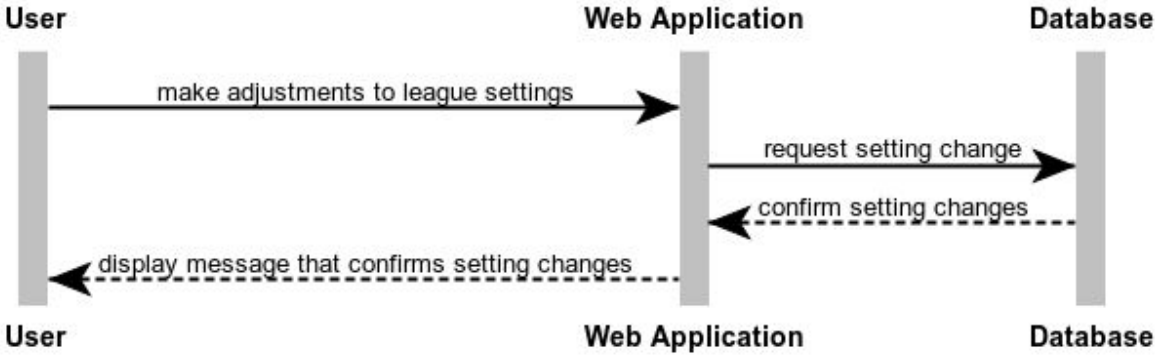
UC-9 Manage Portfolio



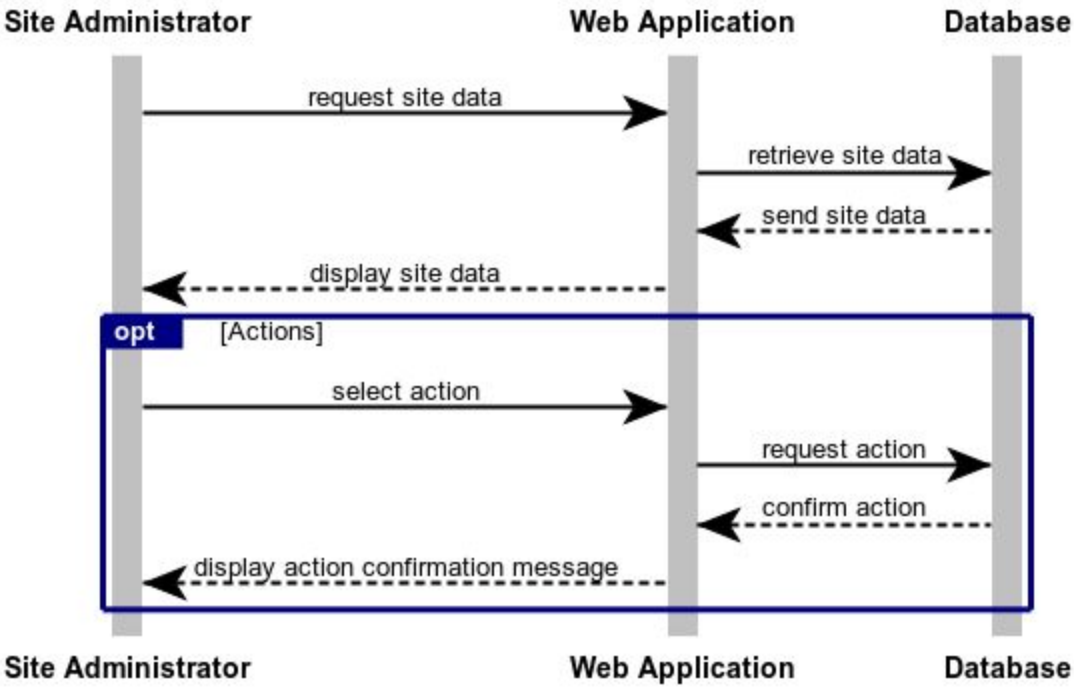
UC-10 Place Market Order



UC-19 Manage League Settings



UC-23 Take Administrative Actions



5. Effort Estimation

The "Use Case Points" system of estimating the effort necessary to create the system is what we will use to estimate the effort done. This is motivated by a need to have a metric on the complexity of the design of the system in order to properly motivate resource allocations, accepted that any created metric will be necessarily subjective and arbitrary.

5.1 Background of UCP

The effort estimation is a factor represented by the product of sums of many different weighing factors. It can also be used to estimate the number of man hours required to complete the project. The factor is represented as such:

$$\text{UCP} = \text{UUCP}*\text{TCF}*\text{ECF}$$

UUCP = UAW + UUCW represents the unadjusted use case weight which is a sum of the unadjusted actor weight (the complexity of actor involvement), and the Unadjusted Use Case Weight (weighted complexity of the use cases of the system). There are two total complexity factors, technical and environmental:

$$CF = C_1 + C_2 \sum_{i=1}^{13} W_i F_i$$

TCF, the Technical Complexity Factor, is a heuristic index representing the challenges posed in implementing nonfunctional requirements of a system and is specified by interviews with experienced developers. C1 = .6, C2 = .01, $W_i \in \{.5,\ 1,\ 2\}$, and $F_i \in [0, 5]$. ECF, the Environmental Complexity Factor, is another heuristic index representing miscellaneous factors including experience and staffing C1 = 1.4, C2 = -0:03, $W_i \in \{-1,\ .5,\ 1,\ 1.5,\ 2\}$, and

$F_i \in [0, 5]$. The UCP can be interpreted as weighted count of the various requirements and specifications needed to implement a system. Therefore, the duration of a project can be estimated by multiplying the UCP by a productivity factor PF representing the average development man-hour needed per use case point.

5.2 Unadjusted Use Case Points

Actor Name	**Relevant characteristics**	**complexity**	**weight**
investor	Interacts with finance system through a graphical user interface in order to look at stocks and manage leagues	complex	3
League manager	Manages leagues through the gui interface on the finance site	complex	3
Site admin	Interacts with the site through text based means in order to administer the site	average	3
database	Is accessed by the users through a dedicated api that we develop	simple	1
Finance api (Alpha Vantage)	Pulls finance data from outside sources using a set api	simple	1
guest	Makes new account on site, only happens if account not already made for user	average	2

Unadjusted actor weight: 2*3 + 2*1 + 2*2 = 12

Use Cases and Weights

Use Case	**description**	**category**	**weight**
UC-1	Guest registers for an account in our database, 2 actors (investor, database) <5 steps for success with <5 accessory steps	average	10
UC-2	User logs into our database, 2 actors (investor, database) but <5 steps for success	simple	5
UC-3	Investor attempts to join a league that already exists in our database, 2 actors (investor, database)	average	10
UC-4	Investor attempts to create a league to hold competitions	average	10

	(investor, database)		
UC-5	Investor searches for information on securities and/or companies, 3 actors (investor, database, alpha vantage). Average work because 4 events for success and 1 extension event	average	10
UC-7	Investor selects the start tutorial link. 1 actor (investor) <5 events for success	Simple	5
UC-9	Investor wants to manage his stock portfolio, 2 actors (investor, database) <5 events for success	average	10
UC-10	Place market order, 3 actors (investor, database, alpha vantage) <10 total events	Complex	15
UC-19	Manage league settings, 3 total actors (league manager, investor, Database) with <5 events total	average	10
UC-23	Take administrative actions, be able to make changes to site and database, 4 total actors (database, investors, league managers, site administrator) with <5 events for completion but with many different options	complex	15

Unaltered Use Case Weight: 2x5 + 6*10 + 2*15 = 100

5.3 Technical Complexity Factors

Technical factor	Description	Weight	Perceived complexity	Calculated weight factor
T1	Users expect good performance	1	3	3
T2	Web based system to be used on multiple machines	2	3	6
T3	Easy to modify features of leagues and profiles	1	3	3
T4	Easy enough to change from a site administrator standpoint	1	2	2

T5	Security is very important to protect users finances	1	4	4
T6	Some unique training required through the tutorial but should be easy to use	.5	1	.5
T7	Response times critical to keep up with stock changes	2	2	4
T8	Internal processing should not be too complex	1	2	2
T9	Concurrent use of the system is required as there are multiple users	2	2	4
T10	System needs to remain up to date to follow stock prices every day	3	1	3
T11	User pages need to have multiple areas of information to track different leagues	3	2	6
T12	Easy to “install” and use as the app is web based	.5	1	.5
T13	Has some direct access from third party finance API’s (alpha vantage)	2	1	2
	Technical Factor totals:			**40**

5.2 Environmental Complexity factors

Environmental factor	**Description**	**Weight**	**Perceived impact**
Development experience	Some beginners With UML based development and construction	1.5	1.5

Application Experience	Some beginners to the finance world, some slightly more experienced	.5	0
Resource Experience	Beginner to intermediate on the use of databases and web frameworks	1	1.5
Lead Capabilities	Some leadership experience, but not much	.5	0
Motivation	Motivation is high but fluctuates throughout the semester with other projects and exams	1	3
Stable requirements	Requirements are approximate but well known	2	2
Part time	Developers working with very few hours a week, very busy otherwise	-1	4
Language	Developers are using multiple different languages and frameworks, some are familiar but some are alien	-1	2
		Total:	15

5.5 Calculations

$$UUCP = 3 \times 3 + 2 \times 1 + 1 \times 2 + 2x5 + 6 \times 10 + 2 \times 15 = 113$$

$$TCF = 0.6 + .01 \times (40) = 1$$

$$ECF = 1.4 - 0.03(5.75) = 1.23$$

$$UCP = 113 \times 1 \times 1.23 = 138.99$$

$$Duration = UCP \times PF = 138.99 \times 28 = 3892$$

6. Domain Analysis

6a. Domain Model

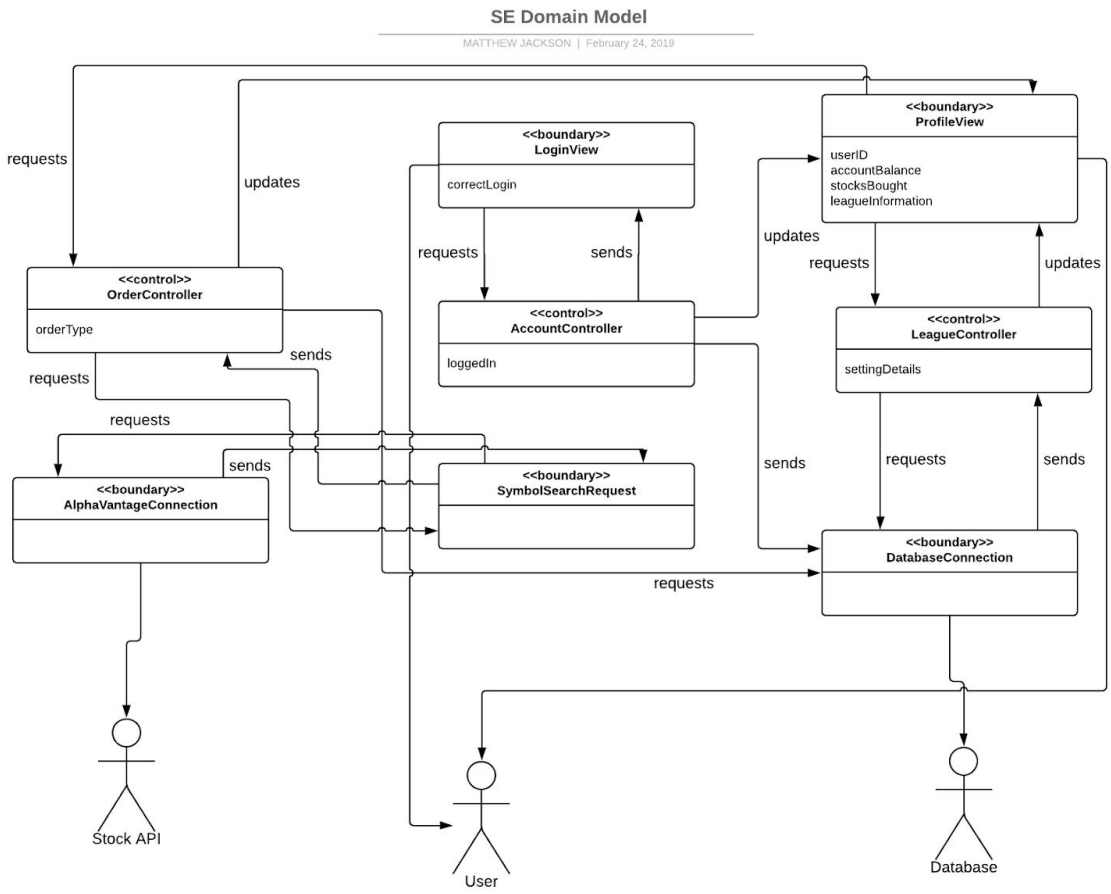


Figure 5.1: Domain Model

i. Concept Definitions

Responsibility Description	Type	Concept Name
Rs1. Controls information regarding to a new user's account registration and an existing user's login.	D	Account Controller

Rs2. Controls information related to an Investor purchasing a security through a certain order type.	D	Order Controller
Rs3. Controls information regarding league managers, league members, and league settings.	D	League Controller
Rs4. Database query that sends information to and from the database that can revolve around league, portfolio, or security information.	D	Database Connection
Rs5. Specifies search parameters for retrieving company or stock information from the Alpha Vantage API.	K	Symbol Search Request
Rs6. API that contains information on stocks, trades, and companies.	K	Alpha Vantage Connection
Rs7. HTML document that displays a user's profile, which contains information related to their own securities.	K	Profile View
Rs8. HTML document that displays the login and account registration screen.	K	Login View

Account Controller

The Account Controller works in conjunction with Login View, Profile View, and Database Connection to handle all the information related to a user's account. Once a user logs in and registers an account, the Account Controller takes this data and sends it to Database Connection. When the user clicks to view their profile, Account Controller changes the view from Login View to Profile View.

Order Controller

The Order Controller updates the Profile View when a user makes an order of any security. Order Controller works with Symbol Search Request to get the price of the security to be purchased or sold when a user makes an order.

League Controller

The League Controller updates the Profile View when a change is made to any league. This includes the creation or removal of a league, changes made to a league's settings, and the addition or removal of an investor to or from the league.

Database Connection

Database Connection takes on the big job of sending and retrieving data to the database from the rest of the model. Database Connection mainly works in conjunction with the controllers to handle all data related to a user or a league. It is in charge of saving new data and retrieving existing data pertaining to users' information.

Symbol Search Request

Symbol Search Request is in charge of making suggestions to the user when the user beings entering letters to search for a stock or a company. Symbol Search Request works with Alpha Vantage Connection to get the list of companies and stocks that a user can invest in.

Alpha Vantage Connection

Alpha Vantage Connection is used to send information from the finance API to the rest of the model. This contains data related to stock prices and other securities along with companies.

Profile View

The Profile View is where a user will see all information regarding their account. When a user is curious about their portfolio or account balance, they can navigate to their profile. Profile View is updated through the Account Controller which has access to Database Connection.

Login View

The Login View is displayed to a new user or a user who is not logged in. The user can navigate to this page manually, but upon trying to get access to any features that require registration, they will be forcefully redirected to this page. Login View works with Account Controller to send data to the database through Database Connection that revolves around account login information.

ii. Association Definitions

Concept Pair	Association Description	Association Name
Login View ↔ Account Controller	Login View sends a request to create an account or login to one. Account Controller sends back an error if something goes wrong.	requests, sends
Account Controller ↔ Database Connection	Account Controller sends login or registration information. Database connection confirms account registration.	sends
Account Controller ↔ Profile View	Account Controller changes view to Profile View.	updates
Profile View ↔ League Controller	Profile View sends a request to join, leave, or alter a league. League Controller updates Profile View with this new information.	requests, updates
League Controller ↔ Database Connection	League Controller requests league information from Database Connection. Database Connection sends information.	requests, sends
Profile View ↔ Order	Profile requests security order information from	requests, updates

Controller	Order Controller. Order Controller updates Profile View with this new information.	
Order Controller ↔ Database Connection	Order Controller requests Database Connection to store orders into the database.	requests
Symbol Search Request ↔ Alpha Vantage Connection	Symbol Search Request requests information about securities and companies. Alpha Vantage Connection returns the information.	requests, sends
Order Controller ↔ Symbol Search Request	Order Controller requests information about symbols and companies for when a user goes to make an order.	requests, sends

iii. Attribute Definitions

Concept	Attributes	Attribute Description
Account Controller	logged in	Used to determine if the user is logged in
Profile View	portfolio information	Holds information related to a user's portfolio which is the collection of all the user's investments.
	user's identity	The user's account name and password.
	account balance	Holds information related to the user's funds.
	stocks bought	Holds information regarding what stocks the user has invested in.
	league information	Contains information related the leagues a user is in.
Order Controller	order type	Holds information relating to whether the user has requested a market, limit, or stop order.

Login View	correct login	Checks if the user has input the correct account information when logging in.
League Controller	setting details	All the information regarding a league's rules.

iv. Traceability Matrix

		Domain Concepts							
Use Case	**PW**	Account Controller	League Controller	Order Controller	Login View	Profile View	Symbol Search Request	Database Connection	Alpha Vantage Connection
UC-1	10	X			X			X	
UC-3,4	6		X			X		X	
UC-5	3						X		X
UC-7	2	X							
UC-9	5	X				X		X	X
UC-10	6			X				X	X
UC-19	5		X					X	
UC-23	4	X						X	

6b. System Operation Contracts

UC-1 Register/Create an Account

- *Preconditions*
  - If a new user is visiting the web application, they must first register before joining/creating a League.
- *Postconditions*
  - After registration, the database is updated and the new user is now acknowledged as being an investor.

UC-3,4 Create/Join League

- *Preconditions*
  - User is logged into their account.
  - There cannot already exist a League with the same name.
  - User has not joined a League yet.
- *Postconditions*
  - User has joined a League.
  - Database has been updated.
  - League has been set with selected settings.

UC-5 Browse Companies

- *Preconditions*
  - User is logged into their account.
  - Alpha Vantage is accepting inquiries.
- *Postconditions*
  - None

UC-7 Start Tutorial

- *Preconditions*
  - User is logged into their account.
- *Postconditions*
  - User is marked as having completed the tutorial, which is required for any user to become a league manager.
  - The tutorial is not closed to a user who has completed it. It can be repeated as many times as desired.

UC-9 Manage Portfolio

- *Preconditions*
  - User is logged into their account.
  - Alpha Vantage is accepting inquiries.
- *Postconditions*
  - Any adjustments made to the investors portfolio have been updated in the database.

UC-10 Place a Market Order

- *Preconditions*
  - User is logged into their account.
  - Investor has enough funds in their account to place a market order.
  - Alpha Vantage is accepting inquiries.
  - Time is between 9:30 AM ET to 4:30 PM ET (Normal trading day hours)
- *Postconditions*
  - User profile is reflected with any change to funds or position.

- Database has been updated with these changes.

UC-19 Manage League Setting

- *Preconditions*
  - Initiating actor is the League Manager.
  - League Manager is logged into their account.
- *Postconditions*
  - Database is updated to reflect any changes made to their account.
  - All users are notified of any changes made in their League.

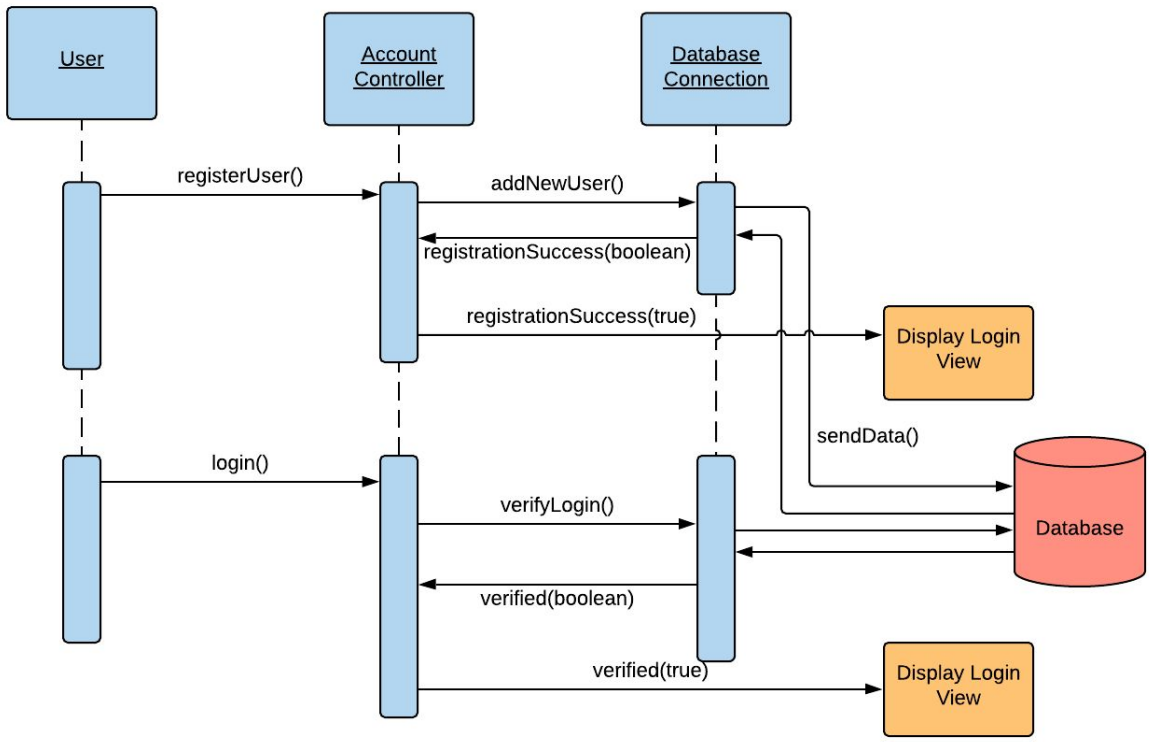
UC-23 Take Administrative Actions

- *Preconditions*
  - User is the Site Administrator.
  - There exists an issue that needs to be resolved.
  - There are outstanding abuse reports.
- *Postconditions*
  - Conflicts/Issues have been resolved.
  - The reported user has been notified of any actions taken against them.

7. Interaction Diagrams

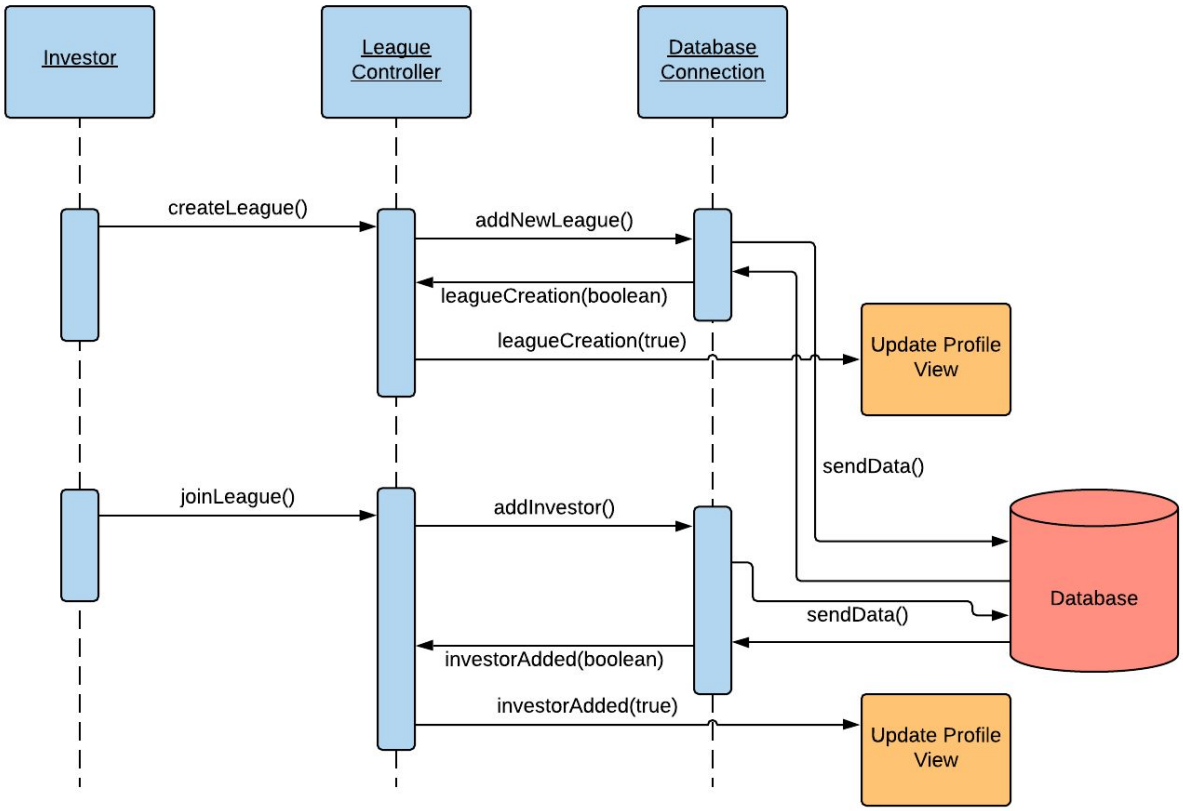
7.1 Use Case Diagrams

Use Case - 1: Register an Account



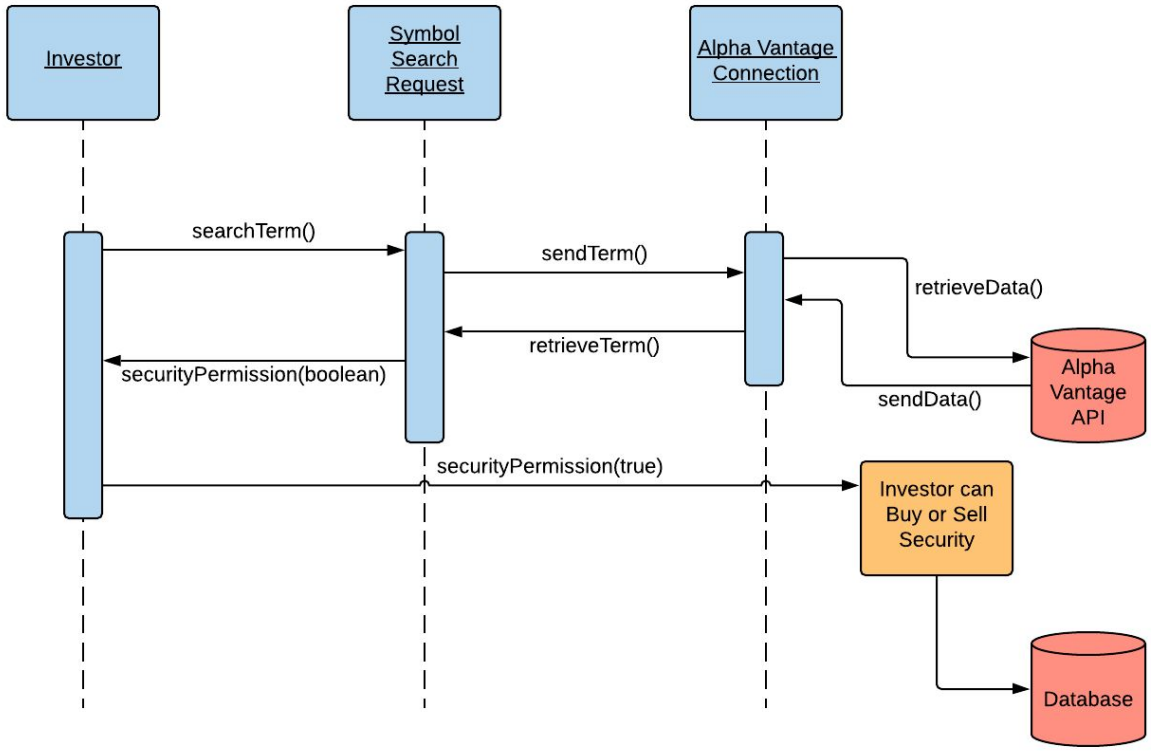
The Interaction Diagram shown above is what the user will encounter first when they start using this software. The user can either login to a previously created account or they can register a new one if they are a new user. The Account Controller(A.C.) will receive the user's request and perform its tasks automatically. If the user is new and requesting a new account, then the A.C. will determine if the user already exists in the database or not. If the database confirms the user is new, then the A.C. will allow for a new user to be created. The second job that the A.C. has in this situation is to verify the login credentials; verifyLogin() will authenticate by comparing user input with the database and return a boolean. If boolean return true, the user is verified as an Investor.

Use Case - 3/4: Create/Join a League



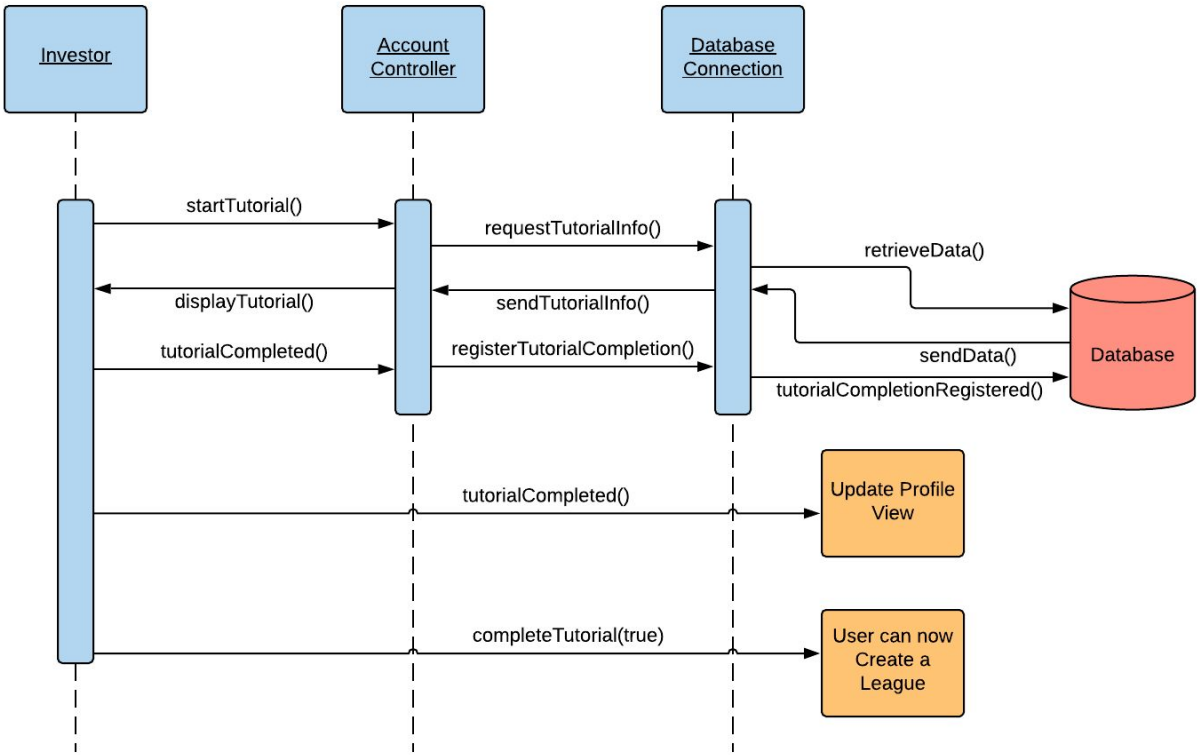
The Interaction Diagram shown above is what the user will encounter if they decide to create or join an Investment League. If the Investor decides to create a new league, the League Controller will connect to the database. The database connection is used to send user information and league settings to the database afterwards. The control is then given back to the League Controller which then displays the Login View. The Investor can also enter a previously existing league by looking at available leagues, and join if wanted. The user will be entered into the database and it will be reflected in their profile.

Use Case - 5: Browse Companies



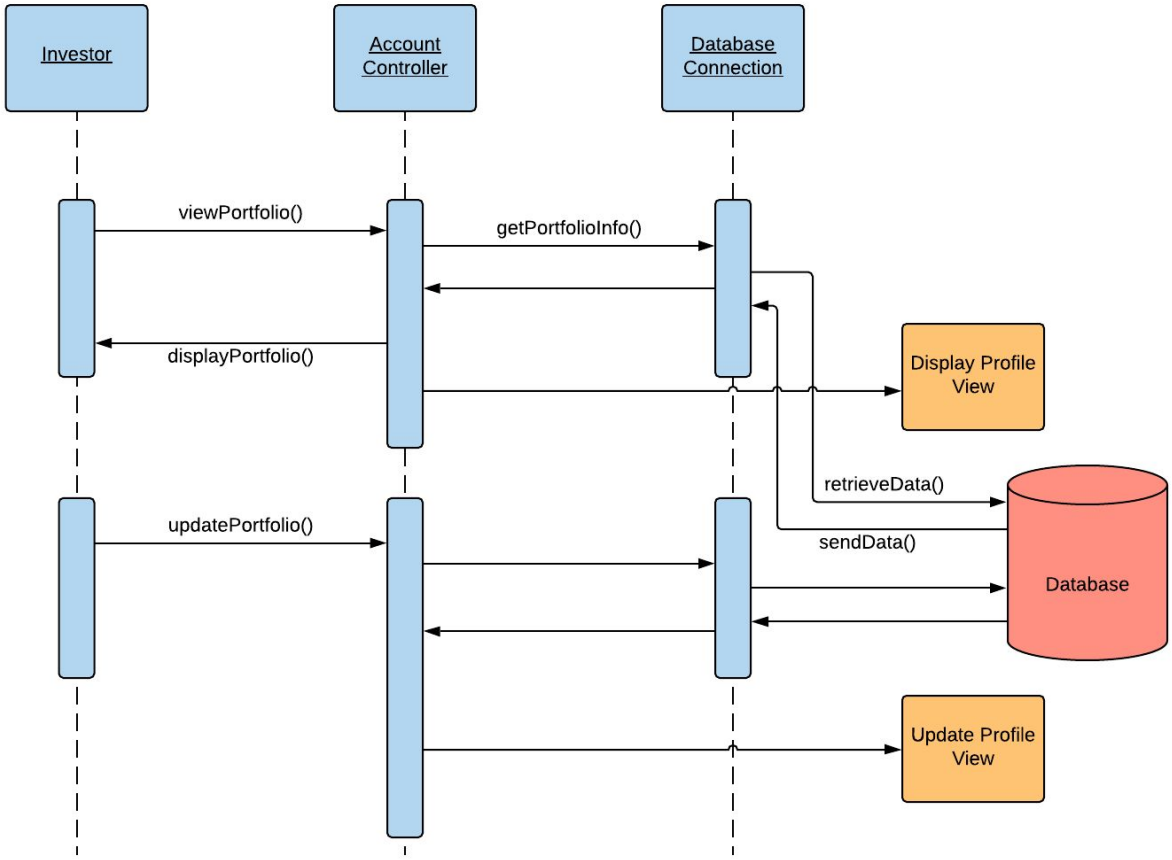
The Interaction Diagram shown above is what the user will encounter when searching for Securities. The term being searched is the input into the Search Symbol Request. The information for current financial statistics is saved in the Alpha Vantage API, so it must be accessed, and all possible query results returned. When the control is returned back to the investor, they can view the security. The information from the API is sent to the database for use, therefore it must be accessed in order to buy or sell.

Use Case - 7: Start Tutorial



The Interaction Diagram shown above is what the new Investor will encounter when learning to navigate through the web application while learning about investing. The Investor must initiate this by sending a request to the Account Controller. The pre-made information for the tutorial is saved in the database, therefore the Account Controller must send the requestTutorialInfo() request to the Database Connection and retrieve the proper information back to the Account Controller. This information is then available for the user to use in order to perform the tutorial and learn the basic concepts of Investing. Afterwards, the database is notified that the user has completed the tutorial and the user can perform operations such as investing or creating a new league.

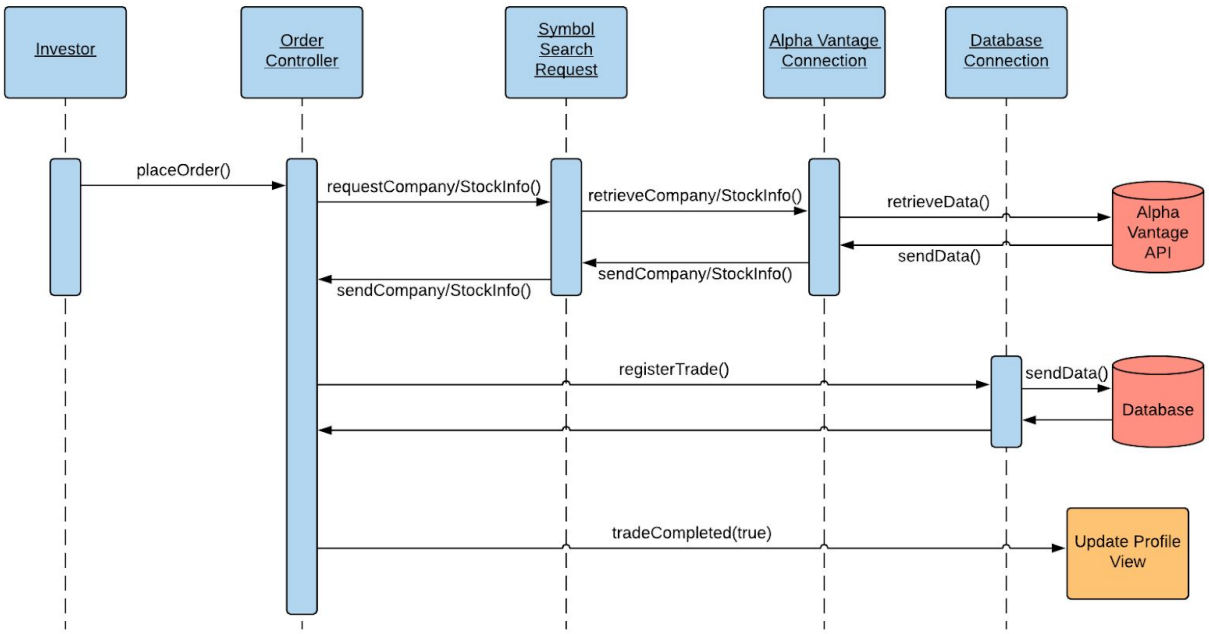
Use Case - 9: Manage Portfolio



This interaction diagram dictates what the actions taken by the Investor attempting to manage his portfolio will call from the Account Controller and the Database Connection. When the investor wishes to view his portfolio the viewPortfolio() command must first be sent to the account controller, which in turn can ask the database connection for the information to populate the portfolio page through getPortfolioInfo(). The database controller sends retreiveData() to the database and that in turn sends the information back to the Database controller with sendData(), which sends the portfolio information to the account controller, and finally the investor may see his portfolio data with displayPortfolio(). The investor has the option to see what his profile looks like to others by using their own profile page link which goes to the display profile view path. The investor can also update his own portfolio by sending different updatePortfolio() commands to the account controller depending on what they wish to change. This will in turn update how the profile looks to others through the Update Profile View path. This also updates

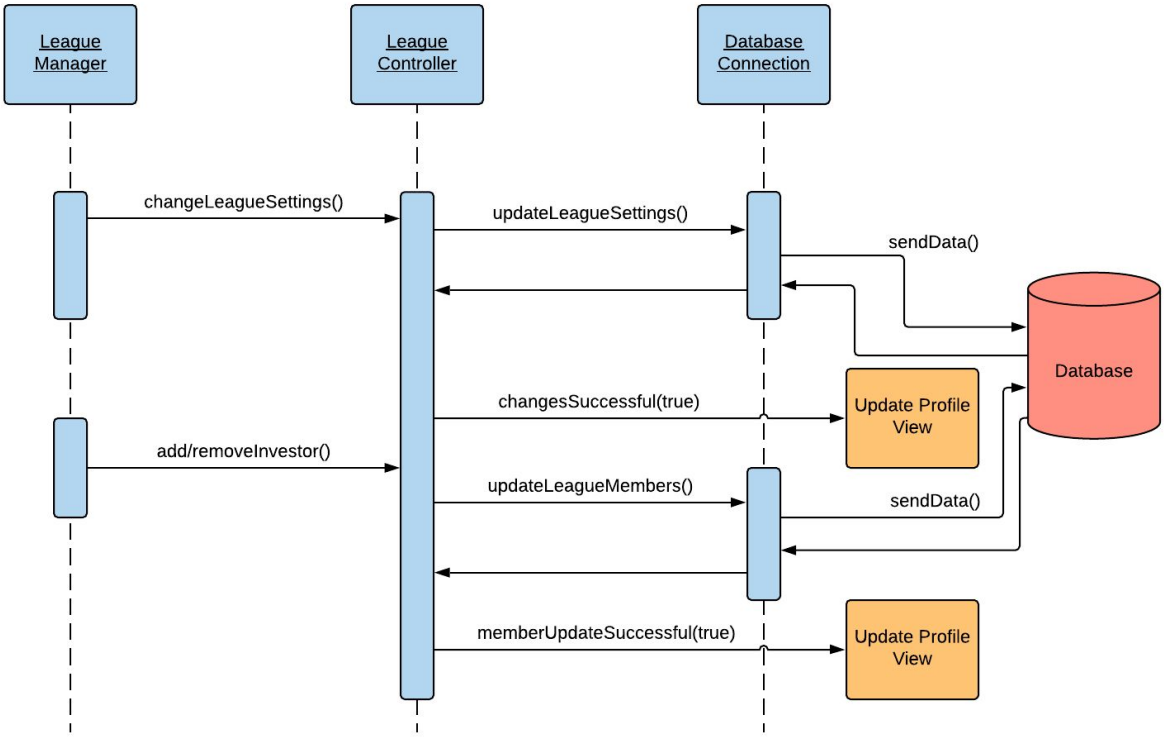
the database with the changes done by the investor after the account controller tells the database connection about the changes done to the investor's portfolio.

Use Case - 10: Place Market Order



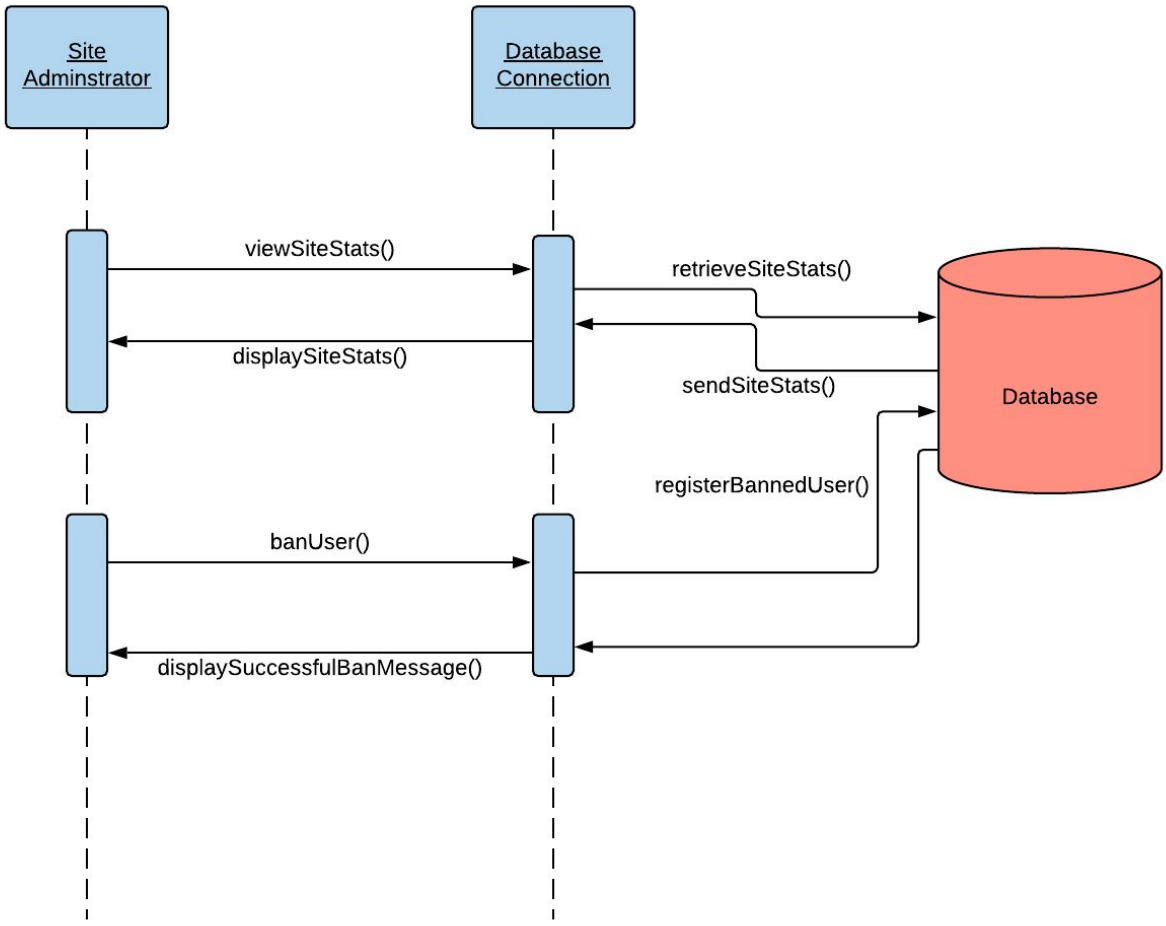
The Interaction Diagram shown above is what the user will encounter when placing a market order. The Investor sends an order request to the Order Controller. The request asks for order type, stock amount and company. This information is accessible via the Alpha Vantage API, so a retrieveCompanyStockInfo() request is sent to the API and then accessed through the database connection. All the data pertaining to market price and the amount is received from the Alpha Vantage API and the order is placed into the database for use. When the control is returned to the Order Controller, a registerTrade() request is made to the database, and the purchased security is then updated in the Investor's portfolio. There are limitations for requests; the user must be in a league and can only operate during normal trading hours.

Use Case - 19: Manage League Settings



This interaction diagram illustrates the different data paths and commands sent between the League Manager, League Controller, Database Connection, and Database itself when the League manager would like to change league settings. For a generic settings change (like league lifespan, League name, etc.) the changeLeagueSettings() command is sent to the league controller, which then communicates with the Database through the Database connection in order to keep the league information current using updateLeagueSettings(). If the changes are successfully implemented it will reflect on the league through changesSuccessful(true) and the Update Profile View path. The league manager will send a separate command, add/removeInvestor(), when he wishes to change the league's user base. After the command is sent, the League controller sends updateLeagueMembers() to the database through the established connection, and the changes (if successful, with a memberUpdateSuccessful(true) command) then are reflected in the update Profile View path sent from the league controller. This will allow the League Manager full control over their league.

Use Case - 23: Take Administrative Actions

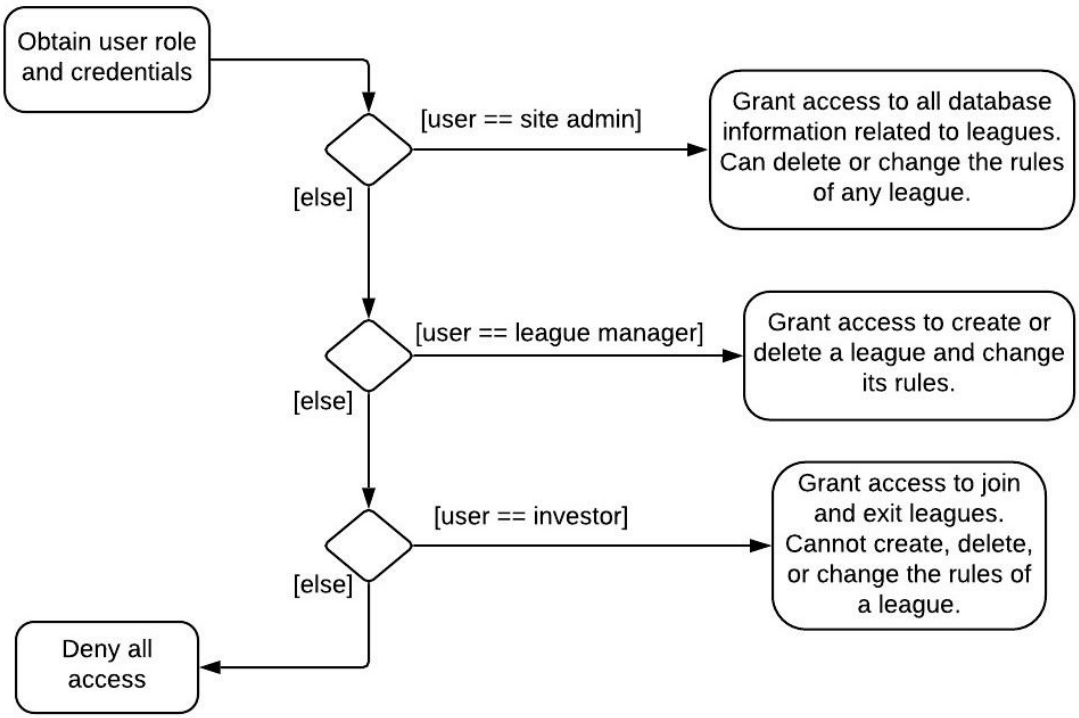


This short interaction diagram describes the commands used by the site administrator in order to keep the site in order (view stats) and to ban those that are not conducive to conversations or break league rules. viewSiteStats() is sent to the database connection which then queries the database for a list of stats it stores, which is sent back through the connection to the site administrator through displaySiteStats(), these stats are already kept in a certain format on the database and just need to be queried. The other command that the site administrator finds quite handy is the banUser() command, which tells the database, through its established connection, that this certain user is banned, which will prevent them from logging into the site and continuing their bannable offenses. After the database registers the user is banned it sends a message back to the site admin through the database connection that displays that the user was successfully banned (displaySuccessfulBanMessage()).

7.2 Design Patterns

Proxy Pattern for Leagues

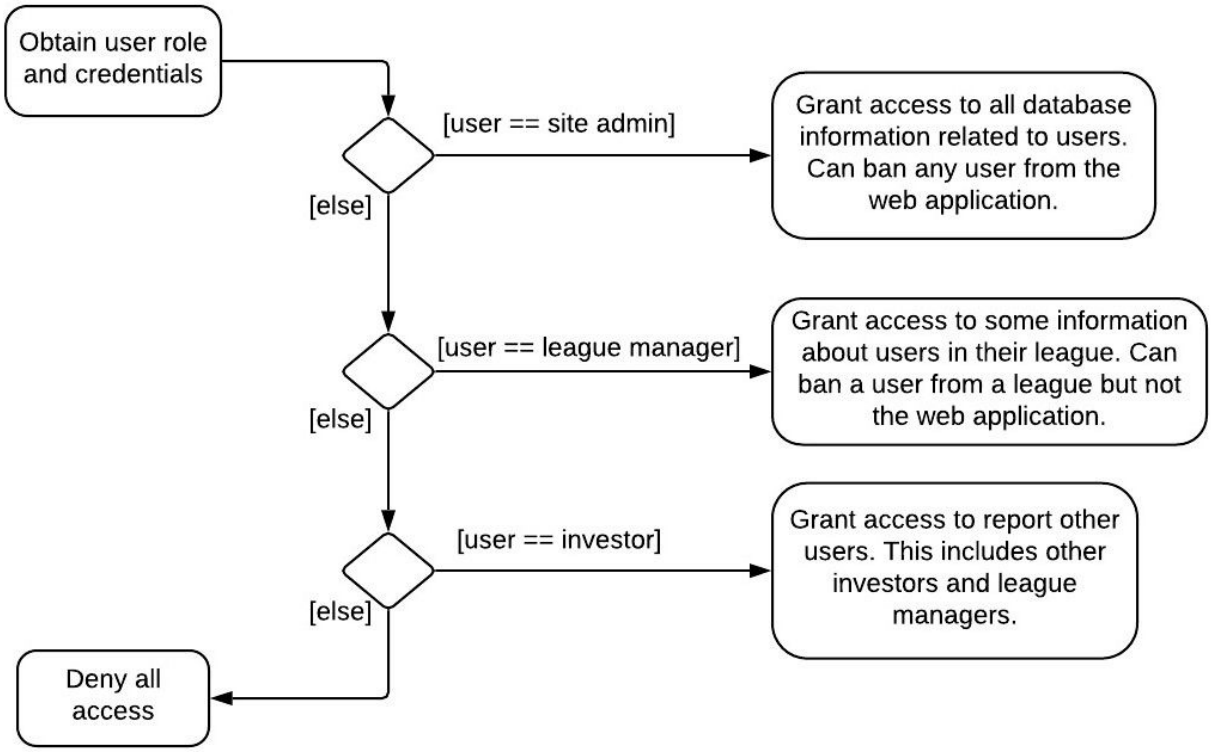
The diagram below shows the privileged access to information related to leagues based on whether a user is a site administrator, league manager, or user. If the user is a site administrator, they have complete control over all leagues and can view all information related to any league. The site administrator's role in this situation is to ban any problematic leagues and make sure that any duplicate leagues are removed. This will make it so that the amount of leagues that can be created is monitored to avoid an overabundance of leagues. The site administrator shall not abuse their ability to manipulate league settings and should leave that to the league manager. If a league manager is abusing their ability to kick members out of a league without good reason, the site administrator can remove that league manager from their position. If the user is a league manager, they have control over the creation and deletion of the league along with changing the league's rules. League managers can kick members of a league out if they feel a specific member is behaving inappropriately. If the user is an investor, they cannot delete a league or change its rules. An investor can only participate in the buying or selling or securities done in the league.



Proxy Pattern for User Actions

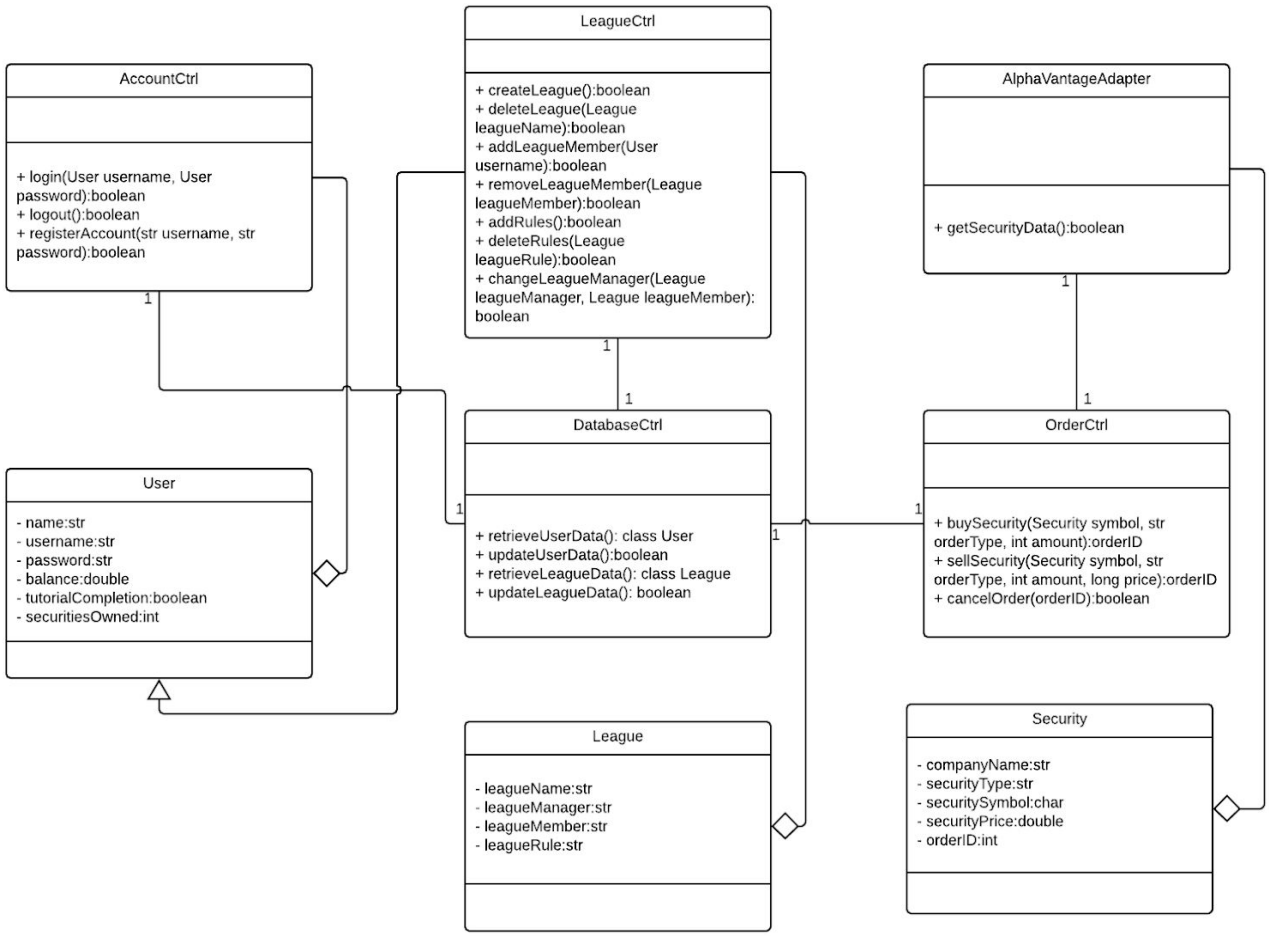
The diagram below shows the privileged access to information related to other users based on whether a user is a site administrator, league manager, or user. If the user is a site

administrator, they have complete control over lower authority users and can view all information related to any of these users. The site administrator's role in this situation is to ban or suspend any problematic users. When a user is banned by a site administrator, they will not be allowed to use the web application again. A suspension by a site administrator temporarily bans a user for a certain period of time. After this time period has elapsed, the user will be allowed access to the web application again. The site administrator shall not abuse their ability to ban other users and should only ban users that have displayed clear forms of misconduct. If the user is a league manager, they have control other users inside of their league. League managers can kick/ban members inside of their league but cannot ban them from the web application. If the user is an investor, they cannot ban any other user. An investor can only report another user that they believe is behaving inappropriately.



8. Class Diagram and Interface Specification

8a. Class Diagram



8b. Data Types and Operation Signatures

AccountCtrl

AccountCtrl works with the User and DatabaseCtrl classes to store the user's username, password, and other information related to their account.

Methods

+ login(class User username, class User password) : boolean
The login method takes in both the user's username and password and returns a boolean value based on whether the user has entered a valid username and password.

+ logout() : boolean
The logout method has no parameters. It will mainly return true when a user clicks to logout. A false return value would only result from a fault in the server.

+ registerAccount(string username, string password) : boolean
The registerAccount method takes in the parameters username and password and assigns it to the attributes located in the User class.

LeagueCtrl

The LeagueCtrl class works with the User, League, and DatabaseCtrl classes to configure all the data related to a league. This includes creating a league, deleting a league, changing league rules, etc.

Methods

+ createLeague(class League LeagueName) : boolean
The createLeague method takes no parameters and just returns a boolean value depending on whether the creation of the league is successful or not.

+ deleteLeague(class League LeagueName) : boolean
The deleteLeague method takes the League class's LeagueName attribute as a parameter and returns a boolean value depending on whether the deletion of the league is successful or not.

+ addLeagueMember(class User username) : boolean
This method takes in the User class's username attribute as a parameter to add an investor to the league. It does not take the League class's leagueMember as a parameter since the investor is not a league member until they are added to the league. The investor must first be added to the league through their username to become a league member.

+ removeLeagueMember(class League leagueMember) : boolean
The removeLeagueMember method takes the League class's leagueMember attribute as a parameter to remove an existing league member from a league. If the member is successfully removed, the method returns true. If the removal process is not successful, the method returns false.

+ addRules() : boolean
This method is called when a league manager wishes to add new rules to a league. If the rule(s) are successfully added, the method returns true. If not, the method returns false.

+ deleteRules(class League LeagueRule) : boolean
The deleteRules method takes in a league rule as a parameter and removes it from the league. Returns true if the rule is successfully removed and false if not.

+ changeLeagueManager(League leagueManager, League leagueMember) : boolean
This method is used to change the manager of a league by taking in the existing manager of the league and the manager-to-be as parameters. If the league manager is successfully changed, the method returns true. If not, the method returns false.

OrderCtrl

The OrderCtrl class works with the AlphaVantageAdapter and DatabaseCtrl classes to configure all the data related to an investor buying and selling securities.

Methods

+ buySecurity(Security symbol, string orderType, integer amount) : orderID
The buySecurity method takes in the security symbol, the order type(market, limit, stop, etc), and the amount of the security the investor wishes to purchase. The order gets assigned an ID which is returned by the method.

+ sellSecurity(Security symbol, string orderType, integer amount, double price) : orderID
This method takes in the security symbol, the order type(market, limit, stop, etc), the amount of the security the investor wishes to sell, and the price they wish to sell it at. The order gets assigned an ID which is returned by the method.

+ cancelOrder(class Security orderID) : boolean
The cancelOrder method takes in the ID of the order as a parameter to cancel the completion of an order.

DatabaseCtrl

The DatabaseCtrl works with the AccountCtrl, LeagueCtrl, and OrderCtrl classes to transfer all data to and from the database.

Methods

+ retrieveUserData(): class User

This method retrieves all data from the User class.

+ updateUserData(): boolean
The updateUserData method is used to update all the data in the User class. The method returns true if the data is successfully updated and false if not.

+ retrieveLeagueData(): class League
This method retrieves all data from the League class.

+ updateLeagueData(): boolean
The updateLeagueData method is used to update all the data in the League class. The method returns true if the data is successfully updated and false if not.

AlphaVantageAdapter

The AlphaVantageAdapter class works with the OrderCtrl and Security classes to handle all data related to securities.

Methods

+ getSecurityData() : boolean
This sole method in the AlphaVantageAdapter class retrieves all information about a security from the AlphaVantage API and stores them in the Security class.

User

The User class contains all private attributes related to a user's data.

Attributes

- name : string
This attribute holds on to the user's real name.

- username : string
This attribute holds on to the user's unique username which is used for login.

- password : string
This attribute holds on to the user's password which is used for login.

- balance : double
The balance attribute contains the user's current account balance which is used when buying or selling securities.

- tutorialCompletion : boolean
The tutorialCompletion attribute holds true or false depending on whether the user has completed the tutorial or not.

- securitiesOwned : integer
The securitiesOwned attribute is an integer which contains the number of securities that a particular user owns.

League
The League class contains all private attributes related to a league's data.

Attributes
- leagueName : string
The leagueName attribute is a string which holds the league name inputted by the league manager.

- leagueManager : string
The leagueManager attribute is a string which holds the username of the league manager.

- leagueMember : string
The leagueMember attribute is a string which holds the username of a league member.

- leagueRule : string
The leagueRule attribute is a string which holds the name of a rule to be added to or removed from a league by the league manager.

Security
The Security class contains all private attributes related to a security's data.

Attributes
- companyName : string
The companyName attribute is a string that holds the name of the company that has a security.

- securityType : string
The securityType attribute is a string that holds the type of a security such as a mutual fund, ETF, or stock.

- securitySymbol : character
The securitySymbol attribute is a character that holds the symbol for a certain security under a company.

- securityPrice : double
The securityPrice attribute is of type double and contains the current price of a particular security.

- orderID : integer
The orderID attribute is an integer that contains a unique ID relating to an order made by an investor.

8c. Traceability Matrix

	Software Classes						
Domain Concepts	Account Ctrl	League Ctrl	Order Ctrl	Database Ctrl	AlphaVantage Adapter	User	League
AccountController	X					X	
LeagueController		X					X
OrderController			X				
LoginView	X						
ProfileView	X	X					
SymbolSearchRequest			X		X		
DatabaseConnection				X			
AlphaVantageConnection					X		

The AccountCtrl, LeagueCtrl, and OrderCtrl classes are modified names for the AccountController, LeagueController, and OrderController domain concepts, respectively. The AccountCtrl class now accounts for the user logging in, logging out, and registering an account rather than only checking if the user is logged in. The LeagueCtrl class has been made more detailed. In the domain concept LeagueController, the attribute was kept general, but in the

LeagueCtrl class, the settings described in the domain concept have been expanded upon. The LeagueCtrl class controls the league name, league rules, league manager, and league members. The OrderCtrl class now has methods that take in the OrderController domain concept's orderType attribute as a parameter. The orderType attribute has been moved under a class called Security. The DatabaseCtrl class is an evolved form of the domain concept DatabaseConnection. DatabaseConnection originally had no attributes, but in the DatabaseCtrl class there are methods that update and retrieve information related to a user or a league. The AlphaVantageAdapter is an evolved form of the domain concept AlphaVantageConnection. The AlphaVantageConnection domain concept had no attributes but the AlphaVantageAdapter has one method that retrieves information related to a security. Three new classes have been created that are not an evolved form of any domain concept. These classes are User, League, and Security. The User class contains variables related to a user's account. The League class contains variables related a league. The Security class contains variables related to a security.

8d. Design Patterns

The use of design patterns clarifies what users have access to what privileges. The proxy pattern was used in this case to help identify the what control specific users have other certain functions of the web application. The proxy pattern makes it more clear as to how we will go about coding the actions that the site administrators, league managers, and investors can perform. The details of the design patterns are described in detail above under the Interface Diagrams section.

8e. Object Constraint Language Constraints

AccountCtrl

In order for the user to be successfully logged in, they should not already be logged in. If the user is already logged in, the option to login should not be available. The below constraint will be used to remove the ability for a logged in user to login.

context: AccountCtrl::login(class User username, class User password) : boolean **pre:** login() != true

The same reasoning follows for a user attempting to logout. The option for a user to logout should be not available unless the user is logged in.

context: AccountCtrl::logout() : boolean **pre:** logout != true

OrderCtrl

In order for an investor to purchase a security, their balance should be greater than or equal to the buying price of that security. This constraint makes sure that the purchase does not go through unless the investor has the appropriate balance.

context: OrderCtrl::buySecurity(Security symbol, string orderType, integer amount) : orderID **pre:** balance >= price

In order for an investor to sell a security, they should have the amount of the security that they are trying to sell. If the investor has less of the security than they are trying to sell, they should be restricted. This constraint ensures that an investor will be unable to sell a security unless they enter an amount less than or equal to the amount they currently own.

context: OrderCtrl::sellSecurity(Security symbol, string orderType, integer amount, double price) : orderID **pre:** self.amount >= amount

9. System Architecture and System Design

9a. Architectural Styles

In order to develop an efficient application, we will combine the use of multiple types of architecture.

Model View Controller:

The Model View Controller type interface is as its name suggests, a three part implementation method. Its parts are analogous to a typical function, with an input and an output.

-The controller aspect will be considered the input. Users can manipulate the view and the model by using the controller. Some parts of the controllers include html forms like text boxes and buttons.

-The view aspect will be the output of our software, namely the webpage. The webpage will vary based on the data received from the model.

-The model aspect will be the function. It will manipulate data based on controller input and is what ultimately changes the user view.

Data Driven Design:

Considering multiple users will be using our application, each one should have specific data pertaining to their account. Stocks they have invested in, achievements they have completed, and personal settings they have selected are just some examples of user specific data which will need to be saved. In addition to this type of data, the database will also need to hold data from Alpha Vantage in order to allow users to view historical stock data. As mentioned before, this will be solved by using the SQLite database. SQLite is not very resource intensive and has significantly more storage than web storage.

Client/Server Architecture:

Since the user (client) will need to access data frequently, this type of architecture will also be pivotal in our application. This is a network architecture in which each computer on the network is either a client or a server. Clients are users that will run the application, while servers will provide resources and computations. By using this architecture we hope to ensure a smooth communication between each part of our Model View Controller.

Representational State Transfer Architecture:

Since the our project is inherently a web-based application, it will naturally use a Representational State Transfer architecture. This type of architecture is ideal due to its simplicity and the fact that it's based off already existing systems and features of the internet's HTTP. Although our current focus is developing the app for desktop users, if we decide to later develop a mobile version, this architecture will be compatible with many other systems such as Android and iOS.

9b. Identifying Subsystems

We can initially divide the application into two sections, a front-end section and a back-end section. As described earlier, the front-end will primarily be the user interface. It is essentially the webpage and contains both the controller and the view. The back-end on the other hand is our model. More specifically, this model refers to anything related with database schema and implementation of relative hardwares.

The front end system is pretty simple in the sense that it only needs to display a webpage. This webpage will be user specific as the data for each user will almost always be unique. One important function of the front-end will need to be reliable communication with the back-end. Each webpage access will most definitely need to have user specific data loaded in. In addition, data will constantly be needed to be sent in to the back-end as the goal of our application is to allow investments to be made.

The back end system will handle all functionality of our application. Data sent in from the front-end needs to be processed and interpreted. This data will determine how the front-end will look later on. In addition, during day to day operations, prices of stocks will change so the database will constantly need to be updated. The back end system needs to handle these changes so the users may have a real-time update of current prices.

9c. Mapping Subsystems to Hardware

The front end can be accessed by the user's internet and web browser. It is the bridge between the client and the server. The back end will primarily be the server side with the SQLite database. However, some of the backend will be contained in the browser as well. Consider the functions which operate on the data from the database, like a purchase option. Based on inputs from the user, the code must send requests to the database. The data received from the database will determine if the user made a valid purchase or if it was invalid. But the data retrieved must be interpreted which means inevitably the backend will be contained in the webpage code.

9d. Persistent Data Storage

In order to keep with with the volatile and ever changing nature of the data that we are using, it is imperative that we have a proper database scheme that can properly represents the objects used. This means that all information such as the User's personal information, portfolio, League settings, and achievements are updated and shown.

This web app will use the help of SQLite because of its reliability. This type of database is quite popular and very easy to use. We plan to make great use of this relational schema because of its practicality in what we are trying to achieve. There are many variables in play here, and keeping track of them can be a hassle. If one were to look at our Class Diagrams, they will see that there are many objects being used and constantly updated.

If the user wants to retrieve a piece of information, a query is created. This query allows for the tables to be searched and the corresponding values returned to the user. If the user decides to look at his/her portfolio, then query is formed and executed. At the end, the information to be returned is formatted in a manner such that that the user can see it. If the user wants make a change to something, the database is accessed and specific parts rewritten to the user's request. This is saved so that it can be looked at later when user wishes. It is important that our software return accurate and updated information

9e. Network Protocol

We will be using HTTP(Hypertext Transfer Protocol). This system works by using hyperlinks for communication in order to transfer information between nodes. This type of protocol will be useful because of its flexibility; it can be used in iOS and Android devices in addition to web based devices. This will allow the user more freedom to access their accounts when they are not at a stationary computer without sacrificing any amenities such as portfolio access, stock information, and trade requests.

9f. Global Control Flow

Execution Orderness:

Our site executes in a linear fashion, with every step needing to be taken every time in order to do an action. You must always navigate to the same pages in order to make changes to things such as your portfolio, your stock league settings, and other profile settings. The timings for this linear setup can be found in the project size area of this report, where each number of keystrokes for each part that executes being about the same every time as this is a navigable website.

Time Dependency:

This system is a real time system as it must update data through Alpha Vantage constantly as stocks change in price, and this periodic timer will refresh on a real-time basis as Alpha Vantage updates their own stock numbers. The site itself will update other settings as they are changed by either the site admin or a stock league manager but all of the important data (stocks, companies, and standings in leagues) will be updated real time as new data from Alpha Vantage comes from their API.

Concurrency:

While the client side website will not need multiple threads as it will take up one thread as a tab of the browser being used, the web server will need to simultaneously host many different user instances at once and will therefore require multiple threads to keep things running smoothly as most of the work is being offloaded to the web server. One thread on the server will be tasked with being updated by Alpha Vantage and to send this data to the client side, while the other threads will handle the user logins and other database requirements.

9g. Hardware Requirements

The only requirement that the user has in this operation is having a modern web browser such as google chrome and a high speed internet connection in order to simulate the most realistic trading environment. All other processes and functions will be handled by the server side so that the client can focus on trading.

Internet Connection:

In order for Traders to be efficient in their work, they need the right tools for the job. The transfer of data from the server to the client requires a fast internet connection. If the connection speed is low, the client would receive old information about stock prices and this can potentially have a negative effect on their decision to purchase or sell a security. The client needs a connection that has high upload and download speeds. Upload speed is needed for placing order on time, or else the client can face delays in their purchase. Download speed is important because it informs the client about the latest stock prices so that they can make a more informed decision. It it recommended that the User have a minimum of 1-MB and 2-MB upload and download speed.

Disk Space:

In order for the server to store all database information safely and securely, it must have an abundant amount of hard disk space available. It is recommended that at least 10GB of space be used for storage.

System Memory:

In order to keep the system running in an optimal manner it is necessary to use a LRU (Least Recently Used) System. This will ensure that all information that is old be wiped from the memory in order to make space for new pertinent data. It is recommended that at least 512 MegaBytes be used for testing purposes.

Client Side Hardware Requirements:

As aforementioned, the only hardware requirement the client needs to have is a fast and secure internet connection. This is needed because without it the client cannot login and use all available functions. The client also needs to have the rest of the basic amenities such as a mouse, keyboard, and basic monitor with at least the standard 800X600 resolution.

10. Data Structures

10.2 Data Structures

Queue

The queue data structure will be used to track all orders made by an investor. The reason a queue was chosen for this task is due to the fact that it follows the FIFO (first in first out) property, which will provide a realistic representation of performing an order of security. Orders made amongst investors will be processed in the order in which they are received. If one investor places a market order before the other, that first investor's order will be executed first. If two investors were to place a limit order at the same limit price, the investor who made the limit order first will get their order completed first.

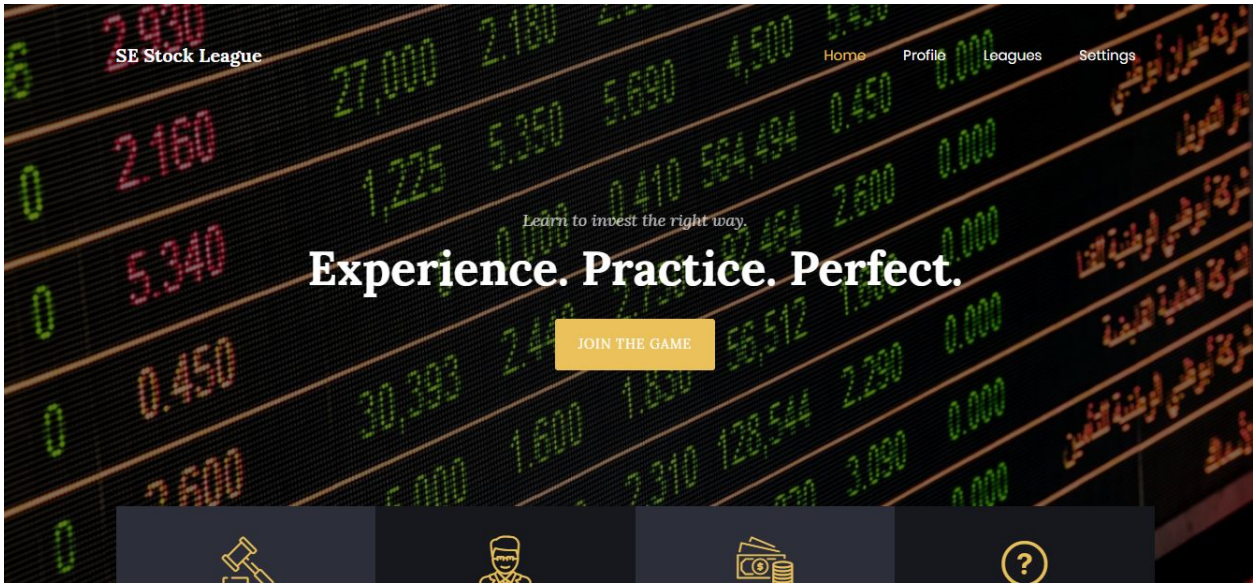
Hash Table

1. The first use of the hash table data structure will be used to store information regarding the purchasing and selling of a security done by an investor. When an investor makes an order, the investor's username and the symbol of the security need to be tracked. A hash table is the optimal data structure for this task since it can allow for easy insertion of all the necessary data along with quick searching of all the data inside of it. This is helpful when finding information related to a given investor's order history. By using the get() function on a certain username, it will be shown what securities the investor under that username has invested in. In this case, the key will be the username and the value will be the security's symbol.
2. The second use of the hash table will be to retrieve information related to the price of a security. After the database has retrieved data regarding security prices and their symbols, the hash table will be used to get the price of a security upon being given the symbol of that security. In this case, the key will be the security's symbol and the value will be the security's price.

11. User Interface Design and Implementation

11.1 Updated Pages

In the original mockup of the home page, there was a good deal of information related to the user, leagues, and stocks. The home page has been changed to mainly display information related to the web application rather than a user. The updated home page is displayed in Figure 11.1.



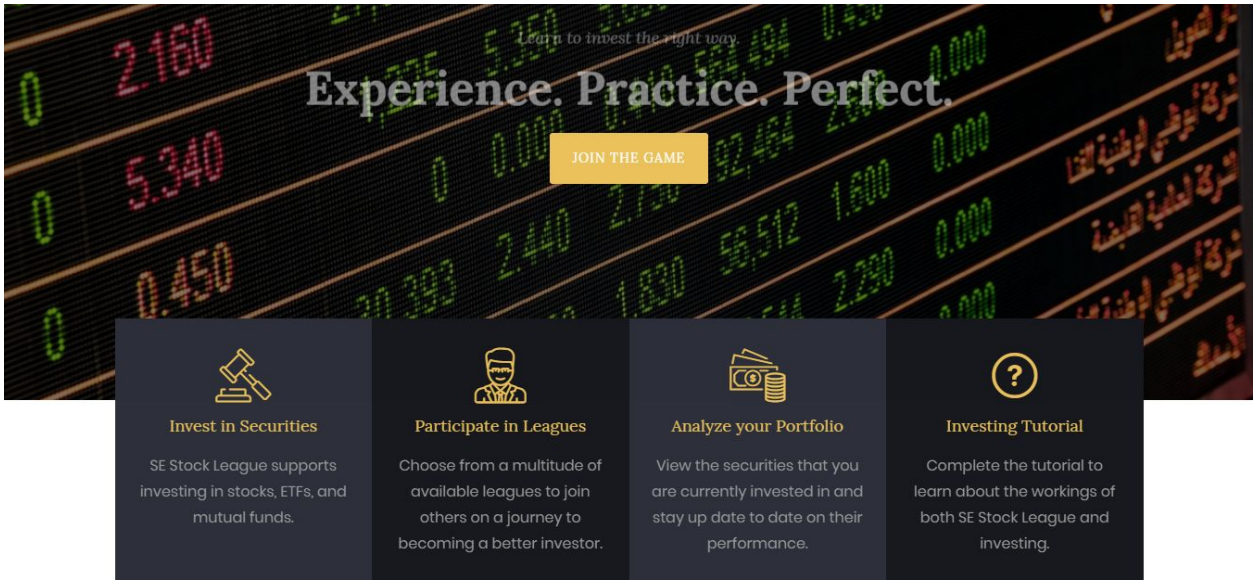


Figure 11.1: Homepage

11.2 Efficiency of Views

The first thing on the list of priorities for this site is to ensure that it is fast and responsive for every user no matter the device or connection they use. This can be done by caching certain elements client side in order to minimize the amount of data that needs to be queried from the database. One way to do this is to separate the header and ticker from the rest of the site content. Both of these things will be consistent throughout the entire site, and therefore do not need to be loaded in for every change in web content, and instead can be loaded once and kept client side.

We will be implementing the site using an HTML/CSS template and a Git repository in order to make changes quickly and easily throughout the development process. The HTML, CSS, and Javascript files that are uploaded to Git are also on InfinityFree which is a free website hosting application. For the content of the pages we will be using HTML5 and CSS3 to create our UI, relying mostly on text and links for the meat of the site with all photos being relegated to backgrounds and our logo.

Our focus will be desktop users so the site will be created with that format in mind. However, users will still be able to use mobile browsers in order to access the site, and therefore it will be built to be mobile responsive by compressing our CSS and making the buttons and text large enough to be seen on mobile but not too big as to clutter the site using PureCSS as a style basis.

Unfortunately due to our modern style templates and such, much older browsers and equipment will not be able to display or parse our HTML5/CSS3 or won't be compliant with modern web standards. This should not cause us many issues as our target audience is university students and other young adults who should all have access one way or another to computers

with connections and hardware capable of supporting our site, so this being an issue should be an exceptionally rare occurrence.

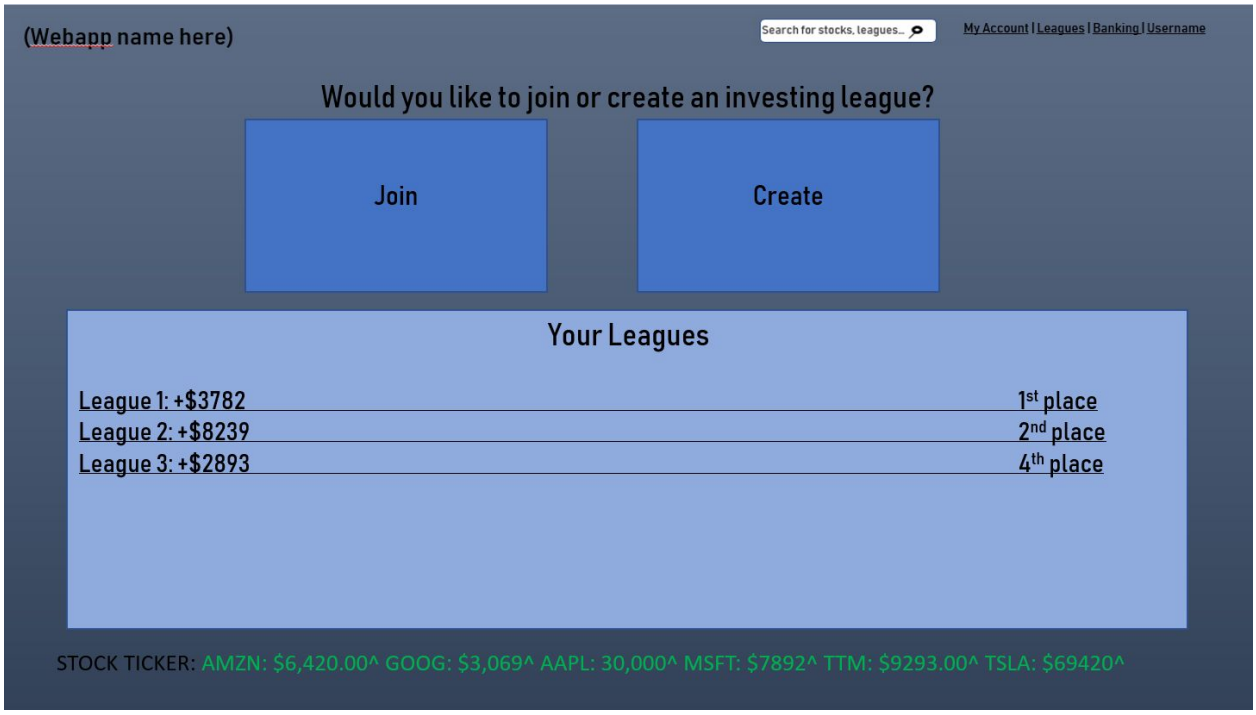
11.3 Early Design

Our user interface will behave as the main area our users will see when investing. It will display security prices, securities that are available, current portfolio, their value, global rank, a dash to perform market orders, a news feed, and a graphic to provide a quick view of security performance. We hope to add different views, making this not the only place a user can see certain information.

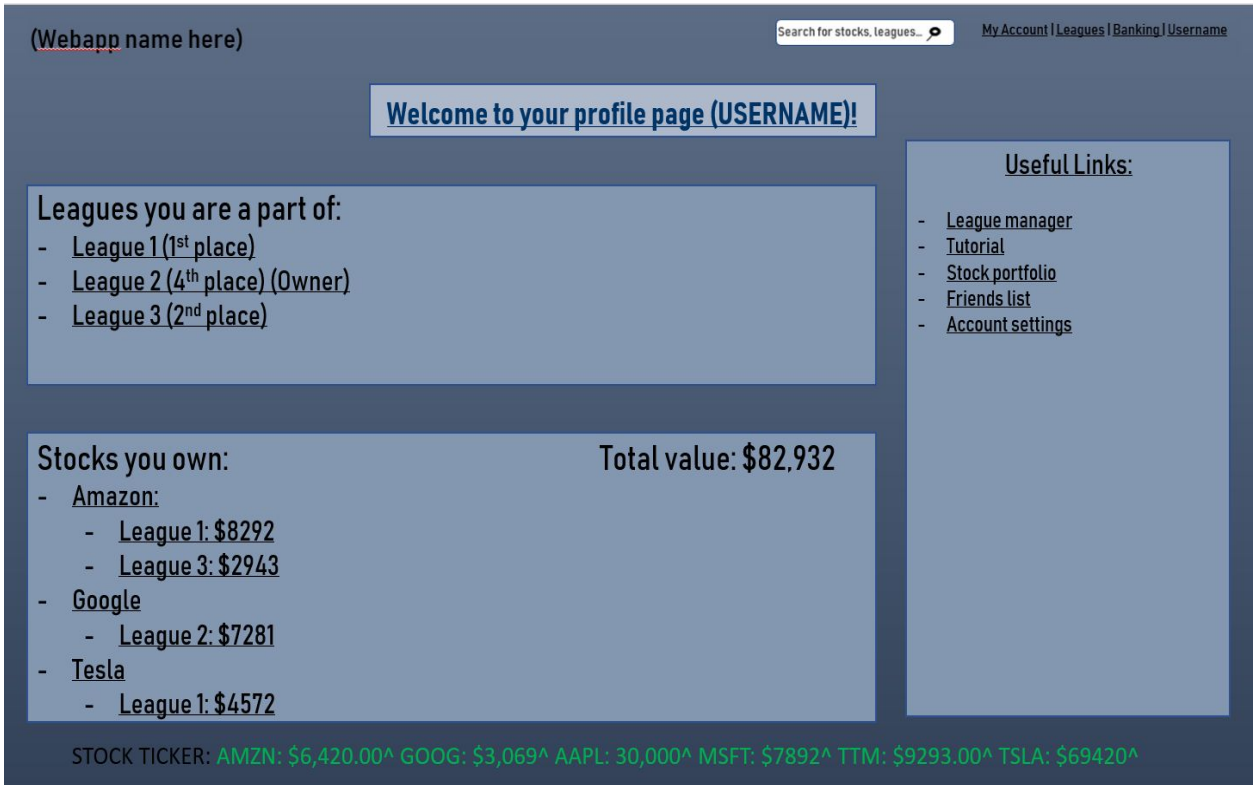
Intro Page/Screen will have 4 elements. The first element at the top is the title of the app. The second element is the logo, which will take up the left half of the page. The right half is for the user to sign in or sign up. The fourth element is at the bottom where we have live ticker updates for the use to view. If the use selects to create a new account, the layout of the page does not change, just the right part of the page will display a new box with details regarding sign-up details. After this page the use will be displayed the home page.

The home page has the top reserved for the logo and menu. The menu has options to direct the use to My Account | My Leagues | Banking | Search Stock. Below the menu, the page is divided into 3 vertical sections,the sections on the left is ¼ size of the page, middle is 1/2 , right is 1/4. The section on the left is used to list all information regarding league updates, league messages, league manager, league news. The middle section displays current portfolio and/or has stocks the user has interest in. The section on the right has news related to the stocks of interest, or any important updates/notification sent my admins/managers. Once a stock is selected from the middle screen, it will navigate to a different page, which is the transaction page. This page has the logo on the top along with the menu described above. Below the page displays various option to buy the stock: Market Order | Limit Order | Stop Loss Order. Once the option is selected, the page displays a message when the order has been executed and immediately the page updates to shows the charts, total gain and total return on the stocks.

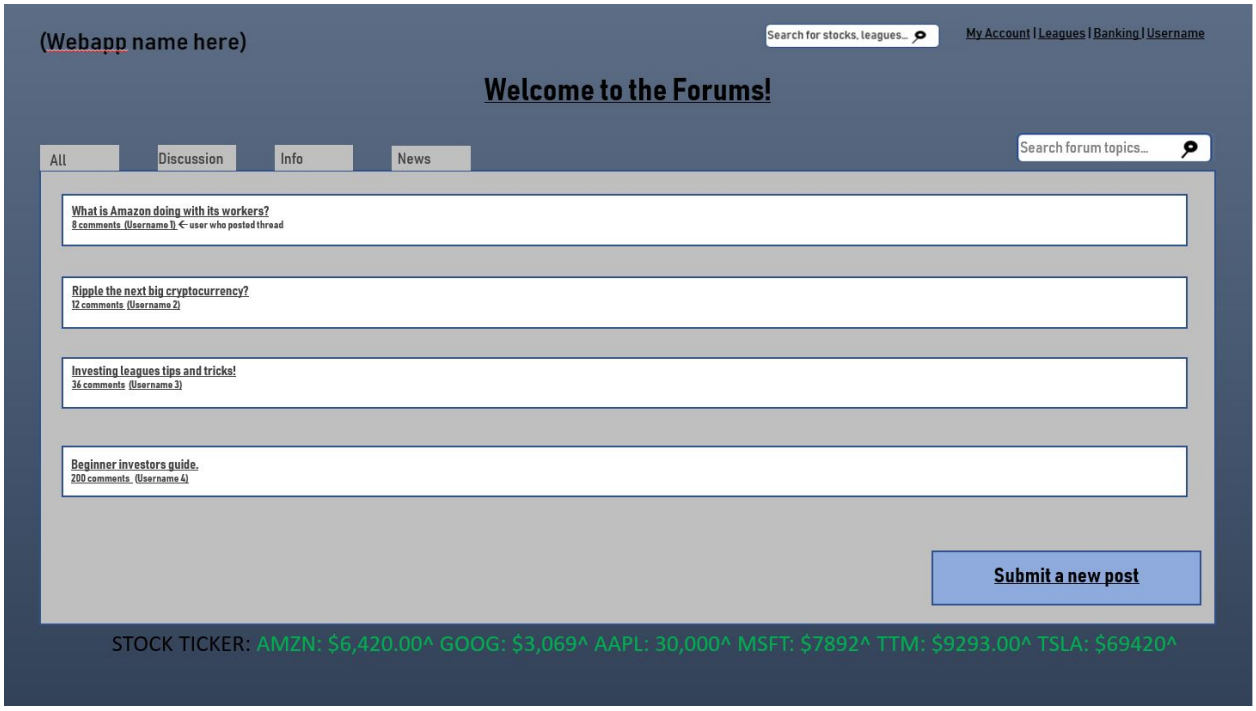
Example stock buy page (note that all numbers are placeholders)



Example “leagues” tab. Can join/create leagues and see ones user is already a part of.



Example profile page



Example Forums home/all page

11.4 User Effort

We hope that the user does not have to put too much effort into navigating through our app, to facilitate this, we will have clearly labeled buttons that the user can read. To execute an order, we will make sure this button is obvious and clear. In terms of how many clicks it takes to access each specific part of the website are minimal.

- From login page to list of leagues (2 clicks)
    - Login page
    - Leagues hotbar at the top right of the screen
- From login to any security, company, or league (3 clicks & some keystrokes)
    - Login page
    - search bar search for whatever is desired by the user (with any number of keystrokes as a search)
    - select desired query
- For a specific forum post (4 clicks & some keystrokes)
    - Login
    - Forum hyperlink in top right corner
    - forum search bar (any # of keystrokes)
    - Desired forum post
- To access a fellow league member's profile (3 clicks or 3 clicks and keystrokes)
    - Login
    - League hotbar
    - Investor username in list of league members
    - **OR**
    - Login
    - Search bar searching for profile name (any # of keystrokes)
    - Select desired profile
- To access your own profile (2 clicks):
    - Login
    - Profile tab in the top right
- Find the tutorial (2 clicks):
    - Login
    - Investing tutorial in the right middle of the page

This app is being designed so that each page is rich in links and features in order to swiftly navigate to any part of the site desired within 5 clicks or so with an occasional set of query keystrokes in between.

12. Design of Tests

12.1 Test Cases

Test-Case Identifier:	TC-1
Function Tested:	Account Registration
Use Case Tested:	UC-1
Pass/Fail Criteria:	The test passes if the user can successfully create an account.
Input Data:	Username, Password
Test Procedure:	**Expected Result:**
Step 1. Type in an invalid username or one that is already taken. Step 2. Type in an invalid password. Step 3. Type in a valid username that is not already taken and a valid password.	System returns that the username contains an invalid character or that the username is already taken. System returns that the password contains an invalid character. System returns a message stating that the account has been successfully created and the account data is saved into the database.

Test-Case Identifier:	TC-2
Function Tested:	Login
Use Case Tested:	UC-2
Pass/Fail Criteria:	The test passes if the user can successfully login to their account.
Input Data:	Username, Password
Test Procedure:	**Expected Result:**
Step 1. Type in an invalid username. Step 2. Type in an invalid password.	System returns a message that states that the username is incorrect. System returns a message that states that the password is invalid.

Step 3. Type in a valid username and password.	User is logged into their account and can perform actions such as investing and viewing their profile.

Test-Case Identifier:	TC-3
Function Tested:	League Creation
Use Case Tested:	UC-4
Pass/Fail Criteria:	The test passes if the league is created.
Input Data:	League Name
Test Procedure:	**Expected Result:**
Step 1. An investor enters an invalid league name or one that is already taken.	The system returns an error message that states that the league could not be created.
Step 2. An investor enters a valid league name that is available.	The system returns a message stating that the league has successfully been created and directs the user to the league's settings.

Test-Case Identifier:	TC-4
Function Tested:	League Deletion
Use Case Tested:	UC-4
Pass/Fail Criteria:	The test passes if the league is deleted.
Input Data:	League Name
Test Procedure:	**Expected Result:**
Step 1. A league manager attempts to delete a league without proper permission.	The system returns a message stating that the league cannot be deleted.
Step 2. A league manager attempts to delete a league with proper permission.	The system returns a message stating that the league has successfully been deleted and the league is removed from the "Leagues" section of the web application.

Test-Case Identifier:	TC-5
Function Tested:	Add or Remove an Investor from a League
Use Case Tested:	UC-20, UC-20.1, UC-20.2

Pass/Fail Criteria: The test passes if the investor is successfully added or removed from the league. **Input Data:** Username	
Test Procedure:	**Expected Result:**
Step 1. An investor attempts to join a league that is either private or full. Step 2. An investor attempts to join a league that is public and not full.	System displays a message that states that the league cannot be joined. System displays a message that the investor has joined the league and the investor is taken to the league's page.

Test-Case Identifier: TC-6 **Function Tested:** Search Company/Security **Use Case Tested:** UC-5, UC-5.5 **Pass/Fail Criteria:** The test passes if the user enters a valid company name or security symbol. **Input Data:** Company Name or Security Symbol	
Test Procedure:	**Expected Result:**
Step 1. Investor searches for a company name or security symbol that does not exist. Step 2. Investor searches for a company name or security symbol that does exist.	The system displays a message that states that no companies or securities with that name or symbol exist. The system displays all the securities and companies that have the symbol or name inserted.

Test-Case Identifier: TC-7 **Function Tested:** Place an Order **Use Case Tested:** UC-10, UC-10.5 **Pass/Fail Criteria:** The test passes if the user can successfully place an order **Input Data:** Symbol, Order Type, Amount, Price	
Test Procedure:	**Expected Result:**
Step 1. Investor has an inefficient balance, symbol does not exist, or	The system displays a message that the order was unsuccessful.

order type does not exist. Step 2. Investor has a sufficient balance, the symbol exists, and the order type exists.	 The system displays a message that the order was successful and the investor's profile is updated to reflect the effect of this order.

12.2 Test Coverage

Rather than beginning with more complicated tests, we have decided to stick with testing the fundamental properties of the investment simulator for this first demonstration. These fundamental properties include registering an account, logging in, creating a league, deleting a league, joining a league, searching for a company or security, and placing an order.
With these fundamental functions of the investment simulator being tested and confirmed as operating correctly, the more intricate aspects of the simulator can be fully focused on for the second demonstration without any errors being a result from these fundamental properties. When an error occurs in the future, we can be assured that it is not due to one of these functions that have already been tested. This makes it easier for future builds of the web application to be debugged.

12.3 Integration Testing

By using Github and creating multiple branches, we will be able to create multiple builds of the web application that can each be individually tested and debugged. Creating multiple branches allows for more experimentation without the worry of changing the web application in a way that could create a glitch in the software. By having multiple branches, we can minimize the amount of times we would have to revert a commit on the master branch. An excessive amount of commit reverts could create confusion amongst the group in terms of what build is currently being worked on. When the master branch has an error, the files in the master branch can be copied over to a different branch and debugged there. These branches are the key to us being able to pinpoint certain problems when testing our functions. Different members will be testing different functions on their local systems and updating the group on whether the functions are working properly or not.

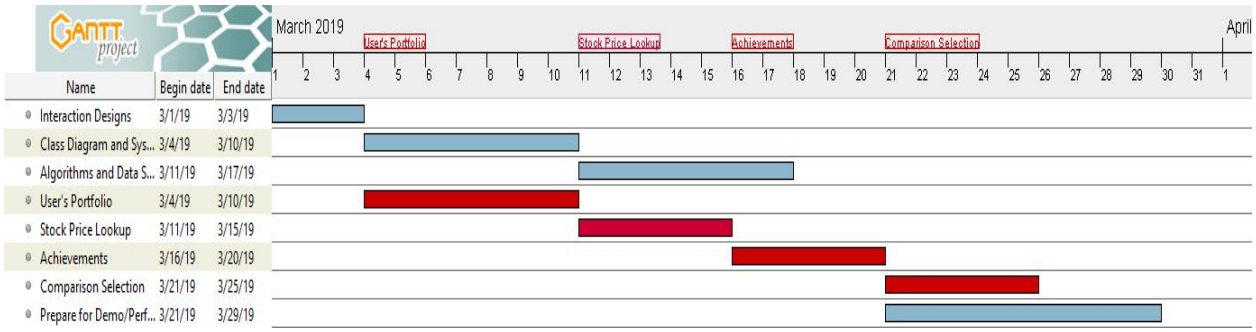
13. History of Work, Current Status, and Future Work

13.1 Development and Report Milestones

The key accomplishments of our project include the things listed below that have been developed over the last few months.

Portfolio and Market Insight Tools, Achievements: Andrew Marfitsin, Himateja Madala

- Create user's portfolio - 3/10
    - Simple stock list view that provides basic information about the user's portfolio
- Stock price lookup - 3/15
    - Allow to search for a stock by ticker
    - Display the current price of the stock
    - Display historic price chart
- Achievements - 3/20
    - Create achievements page that shows completed and incomplete achievements
    - Show requirements for incomplete achievements
- Comparison Section - 3/25
    - Create comparisons page housing dynamic view chart
    - Allow user to compare portfolio to stock or other charts
- Account Registration/Login - 4/16
    - Allow for the creation and login of an account
- Leagues - 4/21
    - Allow for the creation and deletion of a league
    - Allow users to join and exit leagues
- Market Orders - 4/28
    - Allow for users to buy and sell securities through orders.



13.2 Breakdown of Responsibilities

Portfolio and Market Insight Tools, Achievements: Andrew Marfitsin, Himateja Madala

- Stock price lookup with historic price chart
- Own portfolio value chart
- Own portfolio vs. stock/ETF/index/mutual fund chart
- User's list of achievements, accomplished and incomplete
- Making sure achievements are triggered

Website operator: Matthew Jackson

- Manage the webapp
- Allow account registration and account login
- Keep track of all users through a database

Product Functionality: David Wang, Paul Stanik, Joseph Coleman

- Buying and selling stocks
- Creating/deleting a portfolio
- Detecting high/low price, executing sell/buy based on said prices
- Creating and deleting leagues

Charts and Additional Functionality: Murali Gunti, Akshaykumar Patil

- Create financial charts
- Assist in the creation of databases for leagues.

13.3 Future Work

Before the first demo, we mainly focused on the front-end design of the web application without including much functionality to it. After demo 1, we have begun and will continue to work at developing the back-end of the web application by allowing users to register accounts, login to them, and then proceed to join leagues and buy and sell securities.

14. References

- Information about ETFS
https://www.investopedia.com/terms/e/etf.asp
- Information about Mutual Funds
https://www.investopedia.com/terms/m/mutualfund.asp
- Information about Stocks
https://www.investopedia.com/terms/s/stock.asp
- Information about Orders
https://www.investopedia.com/terms/o/order.asp
- Information about Portfolios
https://www.investopedia.com/terms/p/portfolio.asp
- Stock API
https://www.alphavantage.co/
- A previous report from group 1 during the spring 2018 semester
https://www.ece.rutgers.edu/~marsic/books/SE/projects/TradingLeague/2014-g1-report3.pdf
- A previous report from group 2 during the spring 2013 semester
https://www.ece.rutgers.edu/~marsic/books/SE/projects/TradingLeague/2013-g2-report3.pdf
-Investopedia's Stock Simulator was used as a guide for how an investment simulator should function
https://www.investopedia.com/simulator