Diamond Hands Investment Fantasy League

Report #2

Software Engineering

14:332:452

Repository Link

Group 3:

Aarushi Pandey

Aarushi Satish

Apoorva Goel

Christine Mathews

David Lau

Jacques Scheire

Jawad Jamal

Krishna Prajapati

Riya Tayal

Sahitya Gande

Yati Patel

Yatri Patel

Table of Contents

1 Contribution Breakdown **5**

2 Analysis & Domain Modeling **5**

2.1 Conceptual Model 5

2.1.1 Concept Definitions 5

2.1.2 Association Definitions 7

2.1.3 Attribute Definitions 10

2.1.4 Traceability Matrix 11

2.2 System Operation Contracts 13

2.2.1 UC - 4: Add AI Players 13

2.2.2 UC - 5: View Portfolio Status & Information 13

2.2.3 UC - 7: View Leagues & Rankings 13

2.2.4 UC - 9: View Equity Information 14

2.2.5 UC - 10: Make Trades 14

2.3 Data Model & Persistent Data Storage 15

2.3.1 User Account Schema 15

2.3.2 League Schema 16

2.3.3 Equity Schema 16

2.3.4 Portfolio Schema 19

2.3.5 Tooltips Schema 20

2.4 Mathematical Model 20

2.4.1 Mean Reversion 21

2.4.2 Momentum Trading 21

2.4.3 Japanese Candlesticks 21

3 Interaction Diagrams **23**

3.1 Use Case UC - 4: Add AI Players 23

3.2 Use Case UC - 5: View Portfolio Status & Information 24

3.3 Use Case UC - 7: View League & Rankings 25

3.4 Use Case UC - 9: View Equity Information 26

3.5 Use Case UC - 10: Make Trades 27

4 Class Diagram and Interface Specification **27**

4.1 Class Diagram 27

4.2 Data Types and Operation Signatures 28

4.2.1 CD-1: AiServiceHandler 28

4.2.2 CD-2: Authenticator 29

4.2.3 CD-3: CacheUpdater 29

4.2.4 CD-4: Controller 30
4.2.5 CD-5: DatabaseManager 31
4.2.6 CD-6: EntityRetriever 32
4.2.7 CD-7: Equity 32
4.2.8 CD-8: League 33
4.2.9 CD-9: LeagueHandler 34
4.2.10 CD-10: News 35
4.2.11 CD-11: Order 35
4.2.12 CD-12: Page 36
4.2.13 CD-13: Portfolio 37
4.2.14 CD-14: TradingHandler 37
4.2.15 CD-15: User 38
4.2.16 CD-16: UserInput 39
4.2.17 CD-17: Tooltip 40
4.3 Class Diagram Traceability Matrix 41

5 Algorithms 42
5.1 Algorithms 42
5.1.1 Bollinger Bands 44
5.1.2 Simple Moving Average 44
5.1.3 Standard Deviation 44
5.1.4 Relative Strength Index 45
5.1.5 Rate of Change 45
5.1.6 MACD 46
5.2 Data Structures 46
5.3 Concurrency 46

6 User Interface Design & Implementation 46
6.1 Page Design Updates 46
6.2 Page Efficiency 47

7 Design of Tests 47
7.1 View Equity Information 47
7.2 Login and Registration 48
7.3 Trading 49
7.4 Stock Symbol Lookup 50
7.5 Create League 50
7.6 Join League Page 51
7.7 Navigation Bar 52

8 Project Management & Plan of Work **53**
8.1 Merging Contributions from Individual Team Members 53
8.1.1 Version Mismatching 53
8.1.2 Setting up file structure and boilerplate code 54
8.1.3 Setting up the linter 54
8.2 Project Coordination & Progress Report 54
8.2.1 UC-1: Login 54
8.2.2 UC - 2: Create League 55
8.2.3 UC - 3: Join League 55
8.2.4 UC - 4: Add AI player 55
8.2.5 UC - 8: Read News 55
8.2.6 UC - 9: Research equity information 56
8.2.7 UC - 10: Trade 56
8.3 Plan of Work 56
8.4 Breakdown of Responsibilities 57
8.4.1 Aarushi Pandey 57
8.4.2 Aarushi Satish 57
8.4.3 Apoorva Goel 57
8.4.4 Christine Mathews 57
8.4.5 David Lau 58
8.4.6 Jacques Scheire 58
8.4.7 Jawad Jamal 58
8.4.8 Krishna Prajapati 58
8.4.9 Riya Tayal 59
8.4.10 Sahitya Gande 59
8.4.11 Yati Patel 59
8.4.12 Yatri Patel 60

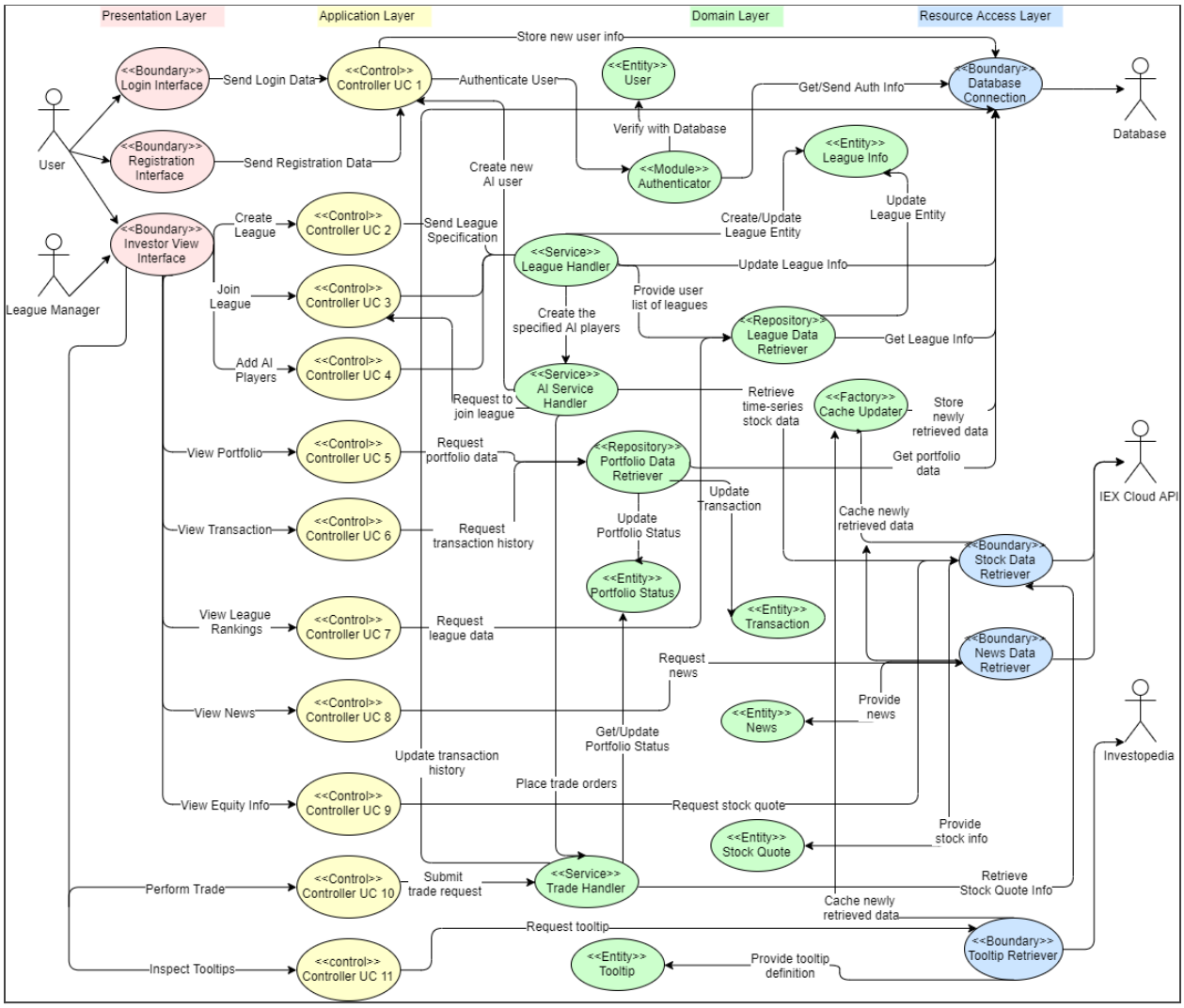
9 References **60**

1 Contribution Breakdown

All team members contributed equally.

2 Analysis & Domain Modeling

2.1 Conceptual Model



2.1.1 Concept Definitions

D = Doing, K = Knowing, N = Neither

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

Container for definitions of financial terms	K	Tooltip
Make trading decisions supported by mathematical models in all leagues containing AIs	D	AI Handler
Updating/Adding stock quote data and news data to database cache	D	Cache Updater

2.1.2 Association Definitions

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

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

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

2.1.3 Attribute Definitions

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

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

2.1.4 Traceability Matrix

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

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

2.2 System Operation Contracts

2.2.1 UC - 4: Add AI Players

Preconditions

- User is logged in
- User is creating new league
- Set how many AI players league manager wants in the league

Postconditions

- AI bot was created and users will be able to view the AI bot's portfolio and transaction history
- All this information will be saved and updated regularly in the database by AI Handler

2.2.2 UC - 5: View Portfolio Status & Information

Preconditions

- User logs into Diamond hands Investment League
- User is a member of the current league

Postconditions

- User's portfolio for this league will be displayed

2.2.3 UC - 7: View Leagues & Rankings

Preconditions

- Investor must be logged in
- User must be registered in a league to be able to access league information

Postconditions

- User's current & past leagues will be displayed
- User's ranking in each individual league will be displayed on the corresponding league page

2.2.4 UC - 9: View Equity Information

Preconditions:

- Investor must be logged in
- Search the equity symbol in symbol lookup

- User entered a valid equity symbol

Postconditions:

- Information of the equity will be displayed

2.2.5 UC - 10: Make Trades

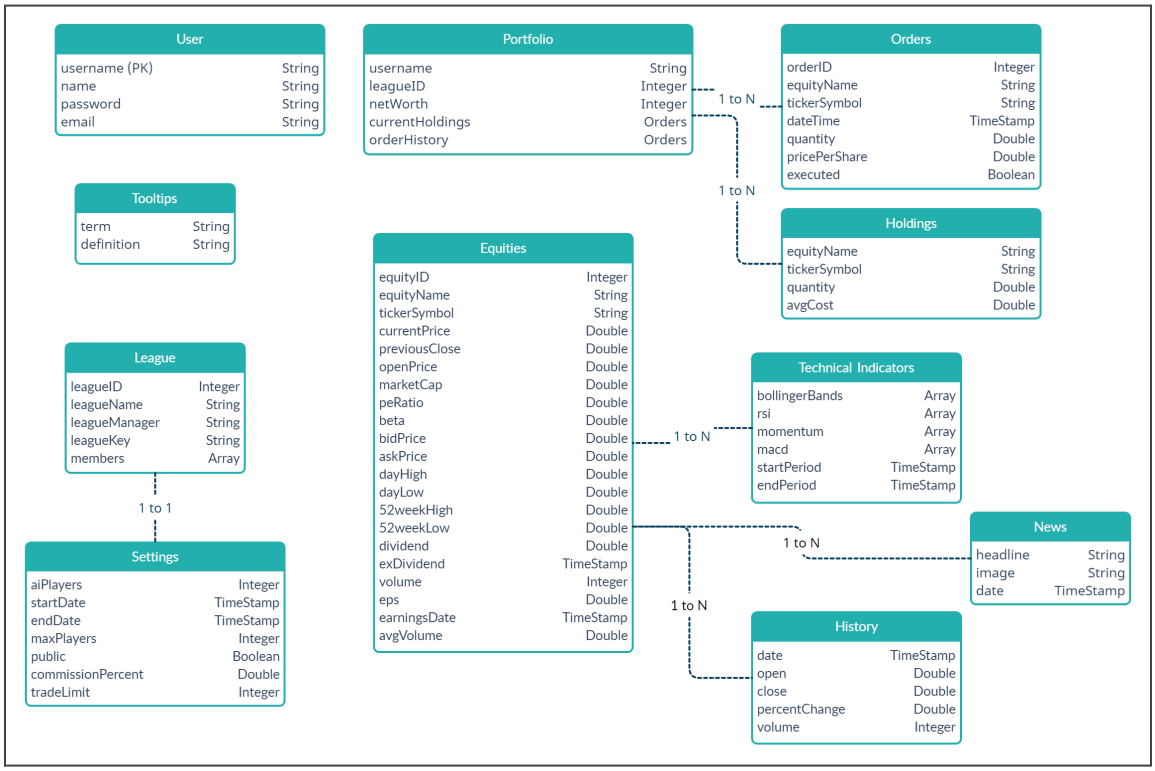
Preconditions:

- Investor must be logged in
- User must be a part of a league
- User must choose the (ongoing) league the trade is being made for
- User must possess sufficient capital/holdings to make the trade
- User must submit the request during regular trading hours

Postconditions:

- Investor profile will reflect these changes in the portfolio after trade is made
- Current trade request will be appended to the portfolio's transaction history
- All portfolio adjustments made will be saved in the database

2.3 Data Model & Persistent Data Storage



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

2.3.1 User Account Schema

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

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

2.3.2 League Schema

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

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

2.3.3 Equity Schema

```
Equity = {
        “equityName”: Integer ,
        “tickerSymbol”: String,
        “currentPrice”: Double,
        “previousClose”: Double,
        “openPrice”: Double,
        “marketCap”: Double,
        “peRatio”: Double,
        “beta”: Double,
        “bidPrice”: Double,
        “askPrice”: Double,
        “dayHigh”: Double,
```

```
“dayLow”: Double,
“week52High”: Double,
“week52Low”: Double,
“dividend”: Double,
“exDividend”: TimeStamp,
“volume”: Integer,
“eps”: Double,
“earningsDate”: TimeStamp,
“avgVolume”: Double ,
“news”: [
{
        “headline”: String,
        “image”: String,
        “date”: TimeStamp,
}
]
“historicalPrices”: [
        “fiveday”: [
                {
                        “date”: TimeStamp,
                        “open”: Double,
                        “close”: Double,
                        “percentChange”: Double ,
                        “volume”: Integer,
                }
        ],
        “onemonth”: [
                {
                        “date”: TimeStamp,
                        “open”: Double,
                        “close”: Double,
                        “percentChange”: Double ,
                        “volume”: Integer,
                }
        ],
        “sixmonth”: [
                {
                        “date”: TimeStamp,
                        “open”: Double,
                        “close”: Double,
```

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

```
        }
        ]
        “technicalIndicators” : [
        {
                “bollingerBands” : Array,
                “rsi” : Array,
                “momentum”: Array,
                “macd”: Array,
                “startPeriod”: TimeStamp,
                “endPeriod”: TimeStamp,
        }
        ]
}
```

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

2.3.4 Portfolio Schema

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

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

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

2.3.5 Tooltips Schema

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

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

2.4 Mathematical Model

Our algorithmic trading bots will utilize the following algorithmic strategies to execute trades and generate returns. These bots will be limited to trading only a selected number of equities and will identify the best possible price action for a good return on investment. The mathematical models are outlined below.

2.4.1 Mean Reversion

Mean reversion is a mathematical method of trading implemented by many other successful quantitative hedge funds and value investors. It is rooted in the belief that the prices will eventually revert to the mean of the historic data and even out over time. This can be applied to the strategy of 'buying low and selling high'. When implementing this strategy, investors are attracted to prices that stray away from historical averages. For example, if the price of an equity dropped significantly due to the resignation of a CEO, investors would buy into the equity with the beliefs that the stock will return to its historical average in time.

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

2.4.2 Momentum Trading

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

There are many ways in which investors apply the momentum trading strategy. We plan to make use of technical indicators such as moving average convergence divergence (MACD), volume, rate of change (ROC) and a movement indicator. These indicators help us identify which of these spikes in prices are real and supported and help us stay away from reversals.

2.4.3 Japanese Candlesticks

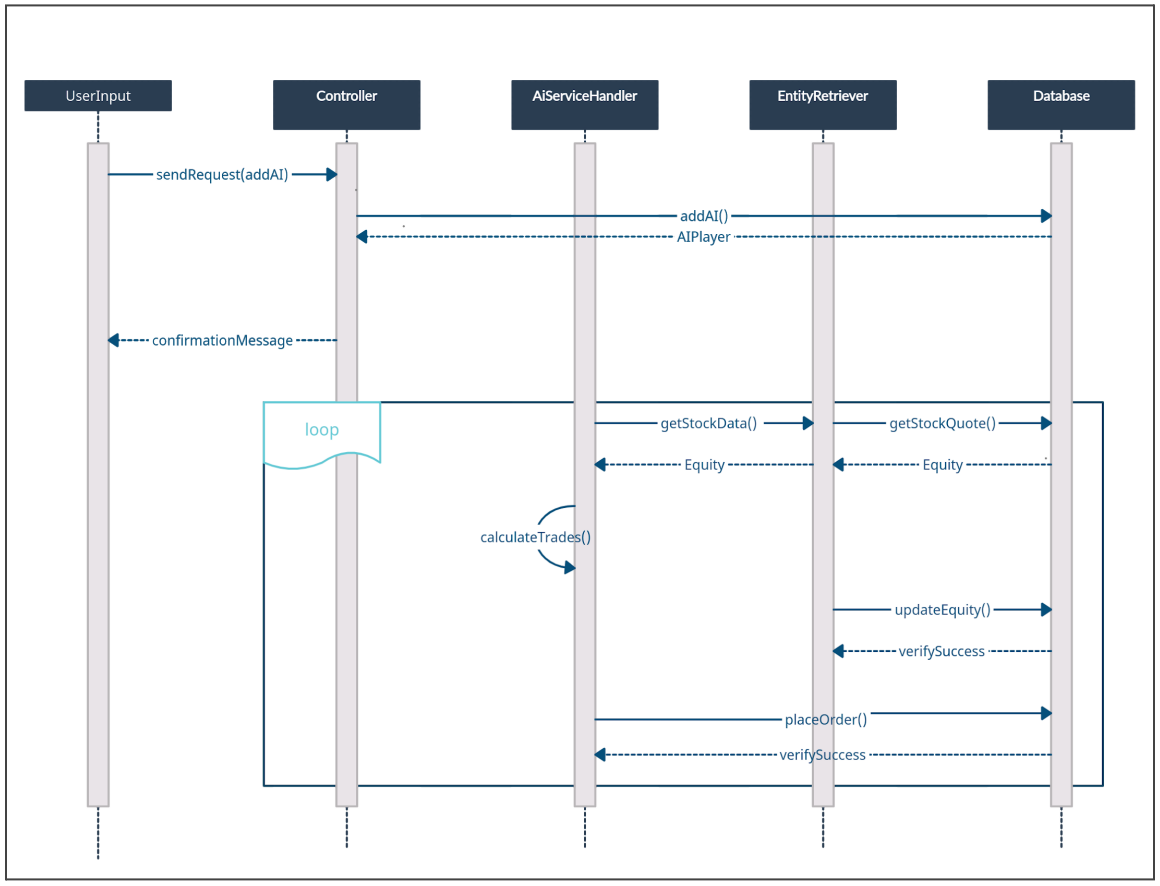
Japanese candlesticks are technical analysis charts that are used by investors to analyze the price movements of equities. They provide a graphical representation of the activity of the particular equity in terms of demand and supply over a set period of time. The candlesticks each have a body, which shows the distance between the open and close prices of the particular equity. They also keep track of the lows and highs of each time period. The patterns made by the candlesticks

provide a visual perspective for predicting future market trends which help investors more accurately plan their entries and exits for profit maximization and loss reduction.

We plan on choosing a set number of candlesticks and trying to classify them under one of the five candlestick patterns. The patterns we will identify include the engulfing, hammer & shooting star, dragonfly & gravestone doji, morning & evening star, and tweezer top & bottom patterns. Each pattern can give us more insight about the market and can help us realize market reversals. These reversal patterns will then be used as an entry trigger to either buy or sell equities. We will be making our judgements by following the TAE framework which stands for Trend, Area of Value, and Entry Trigger. The trend criteria can be based off of the moving average and will let us know if we have a long or short bias. The area of value criteria takes into consideration the support/resistance of the market and the trendline. And the entry trigger will be any reversal patterns we can identify from the aforementioned candlestick patterns. Using all of the criteria together, we can make an informed decision when placing orders.

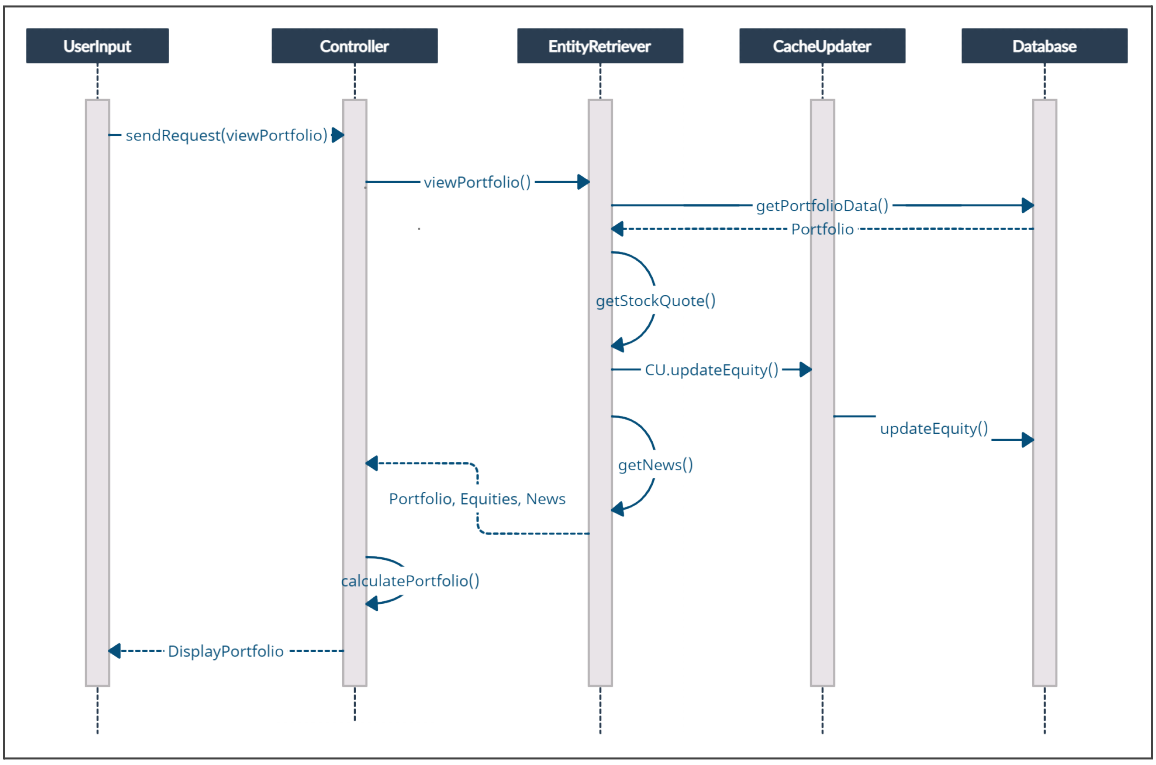
3 Interaction Diagrams

3.1 Use Case UC - 4: Add AI Players



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

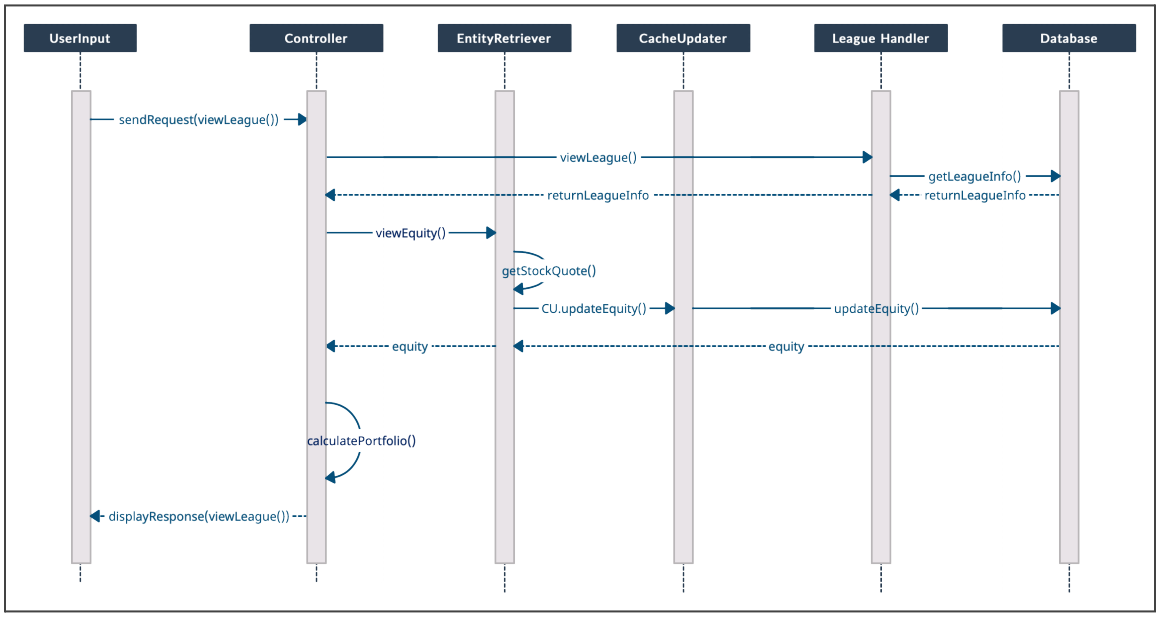
3.2 Use Case UC - 5: View Portfolio Status & Information



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

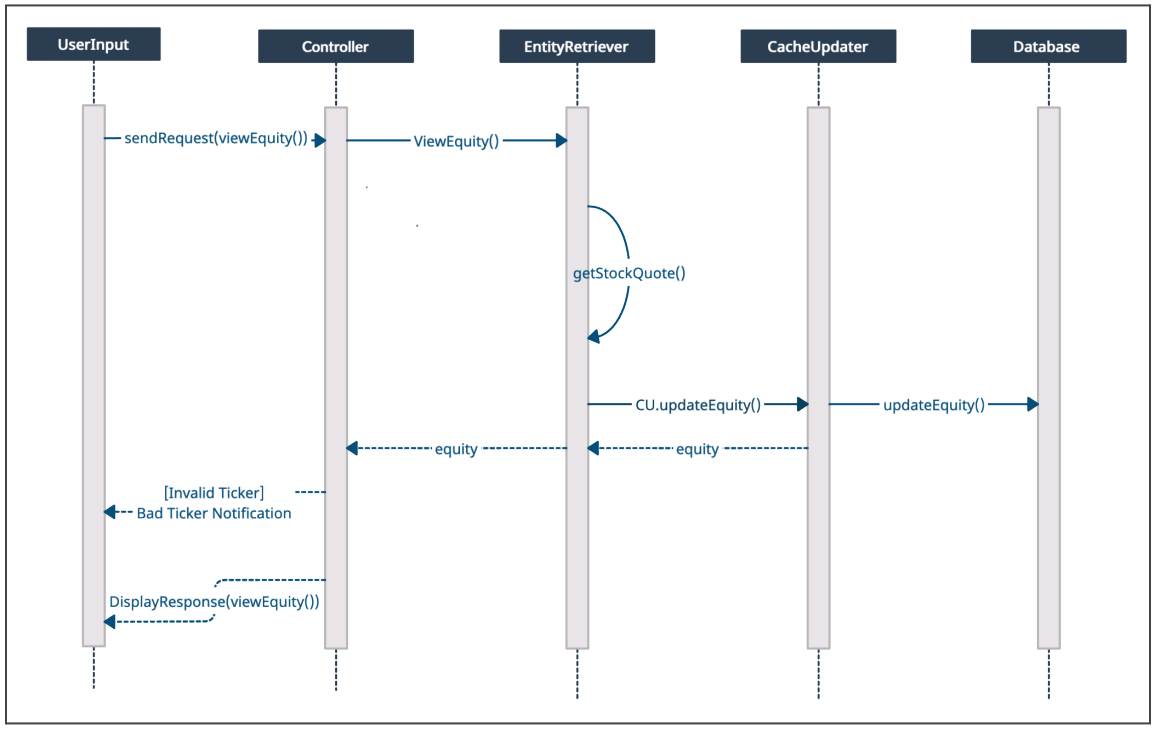
The design principles employed are the **High Cohesion principle** and the **Expert Doer principle**. The High Cohesion principle is exhibited in all the actors which have specific tasks relevant to their functionalities, which limits the number of computational responsibilities. The Expert Doer principle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

3.3 Use Case UC - 7: View League & Rankings



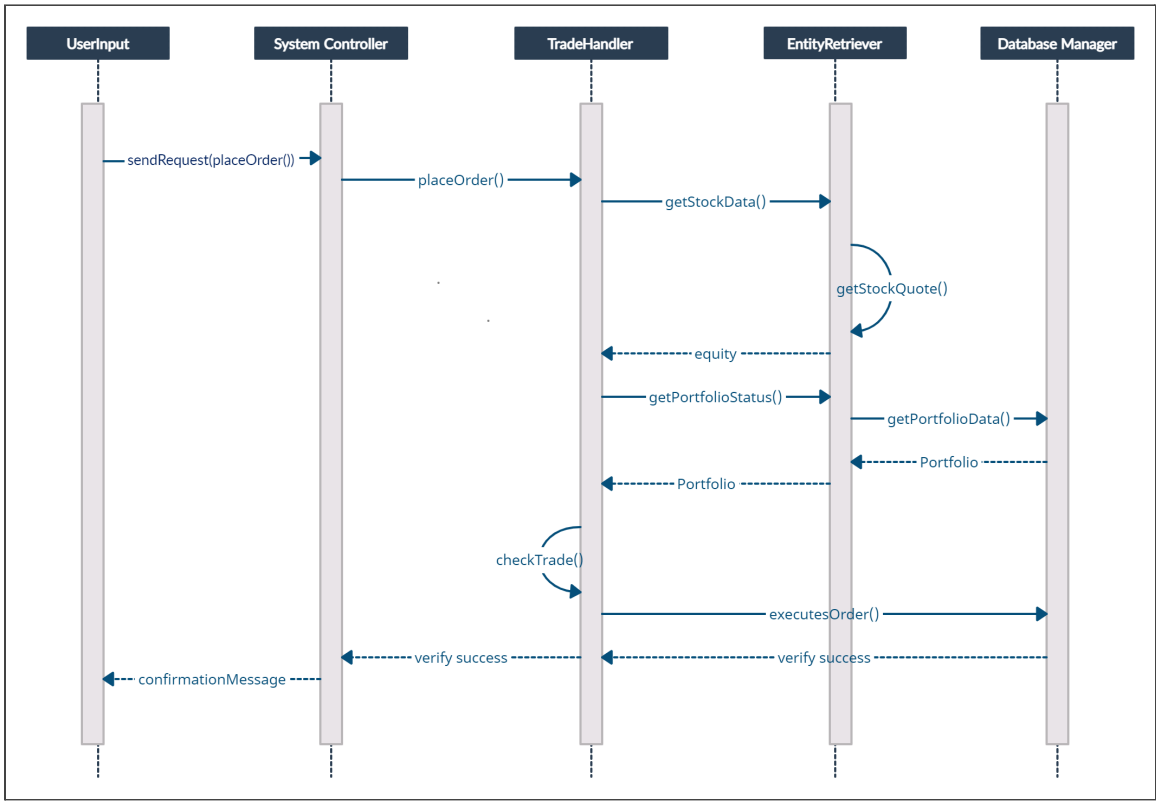
This diagram shows the interactions the actor will encounter in 'UC-7:View League & Rankings'. This will allow the actor to view their rank in their current leagues as well as their league information. Once the user is logged in and selects a certain league the **Controller** requests the league information.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

3.4 Use Case UC - 9: View Equity Information



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

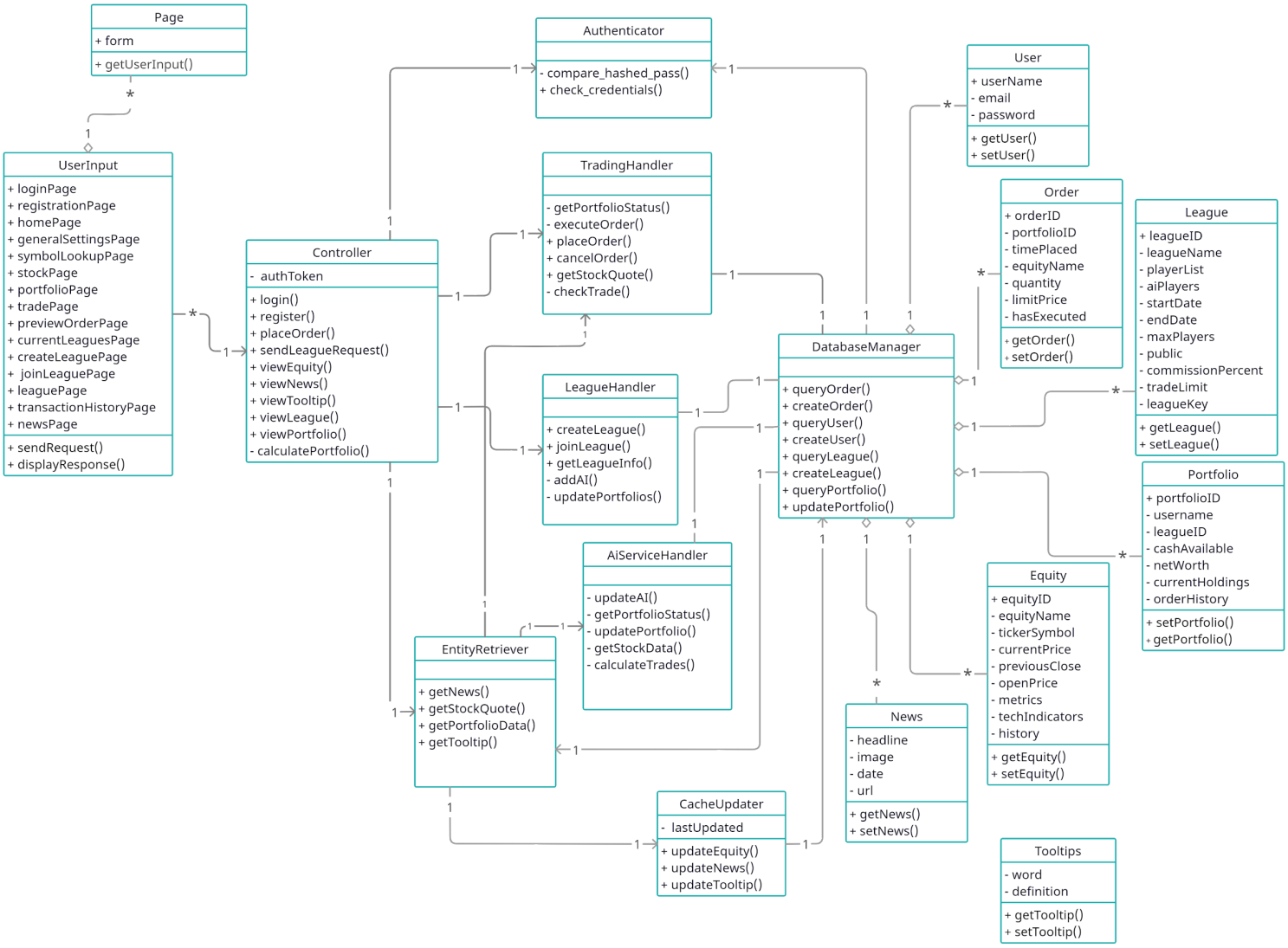
3.5 Use Case UC - 10: Make Trades



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

4 Class Diagram and Interface Specification

4.1 Class Diagram



4.2 Data Types and Operation Signatures

4.2.1 CD-1: AiServiceHandler

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

Operations:

- updateAI(leagueName: String): Object

Updates AI settings and information for the league.

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

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

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

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

4.2.2 CD-2: Authenticator

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

Operations:

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

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

4.2.3 CD-3: CacheUpdater

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

Attributes:

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

Operations:

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

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

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

4.2.4 CD-4: Controller

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

Attributes:

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

Operations:

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

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

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

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

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

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

+ viewTooltip(tooltipID: Integer): Object

Get the definition of a term to display on the tooltip.

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

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

4.2.5 CD-5: DatabaseManager

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

Operations:

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

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

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

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

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

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

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

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

4.2.6 CD-6: EntityRetriever

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

Operations:

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

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

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

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

4.2.7 CD-7: Equity

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

Attributes:

+ equityID: Integer
  The unique identifier for this equity.

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

- tickerSymbol: String
  The ticker symbol of the equity.

- currentPrice: Double
  The current value of the equity.

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

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

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

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

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

Operations:

+ getEquity(): Object
  Get the equity

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

4.2.8 CD-8: League

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

Attributes:

+ leagueID: Integer
  The unique identifier for this league.

- leagueName: String
  The name of the league.

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

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

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

- endDate: Date
  The date when this league ends.

- maxPlayers: Integer

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

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

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

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

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

Operations:

+ getLeague(): Object
  Get the league

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

4.2.9 CD-9: LeagueHandler

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

Operations:

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

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

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

+ addAI(leagueName: String): Boolean

Creates and adds AI players to league

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

4.2.10 CD-10: News

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

Attributes:

- Headline: String
  Title of news article

- Image: String
  Link to image associated with news article

- Date: DateTime
  Date and time news article was published

- URL: String
  Link to news article webpage

Operations:

+ getNews(): Object
  Get the news

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

4.2.11 CD-11: Order

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

Attributes:

+ orderID: Integer
  A unique identifier for this order.

- portfolioID: Integer

The portfolio of which this order is placed for.

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

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

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

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

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

Operations:

+ getOrder(): Object
  Get the order

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

4.2.12 CD-12: Page

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

Attributes:

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

Operations:

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

4.2.13 CD-13: Portfolio

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

Attributes:

+ portfolioID: Integer
  The unique identifier for this portfolio.

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

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

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

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

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

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

Operations:

+ getPortfolio(): Object
  Get the portfolio

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

4.2.14 CD-14: TradingHandler

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

Operations:

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

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

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

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

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

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

4.2.15 CD-15: User

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

Attributes:

+ userName: String
  The username of the user.

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

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

Operations:

+ getUser(): Object
  Get the user

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

4.2.16 CD-16: UserInput

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

Attributes:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Operations:

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

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

4.2.17 CD-17: Tooltip

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

Attributes:

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

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

Operations:

+ getTooltip(): Object

Get the tooltip entity from database

\+ setTooltip(Tooltip: Object): Boolean

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

4.3 Class Diagram Traceability Matrix

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

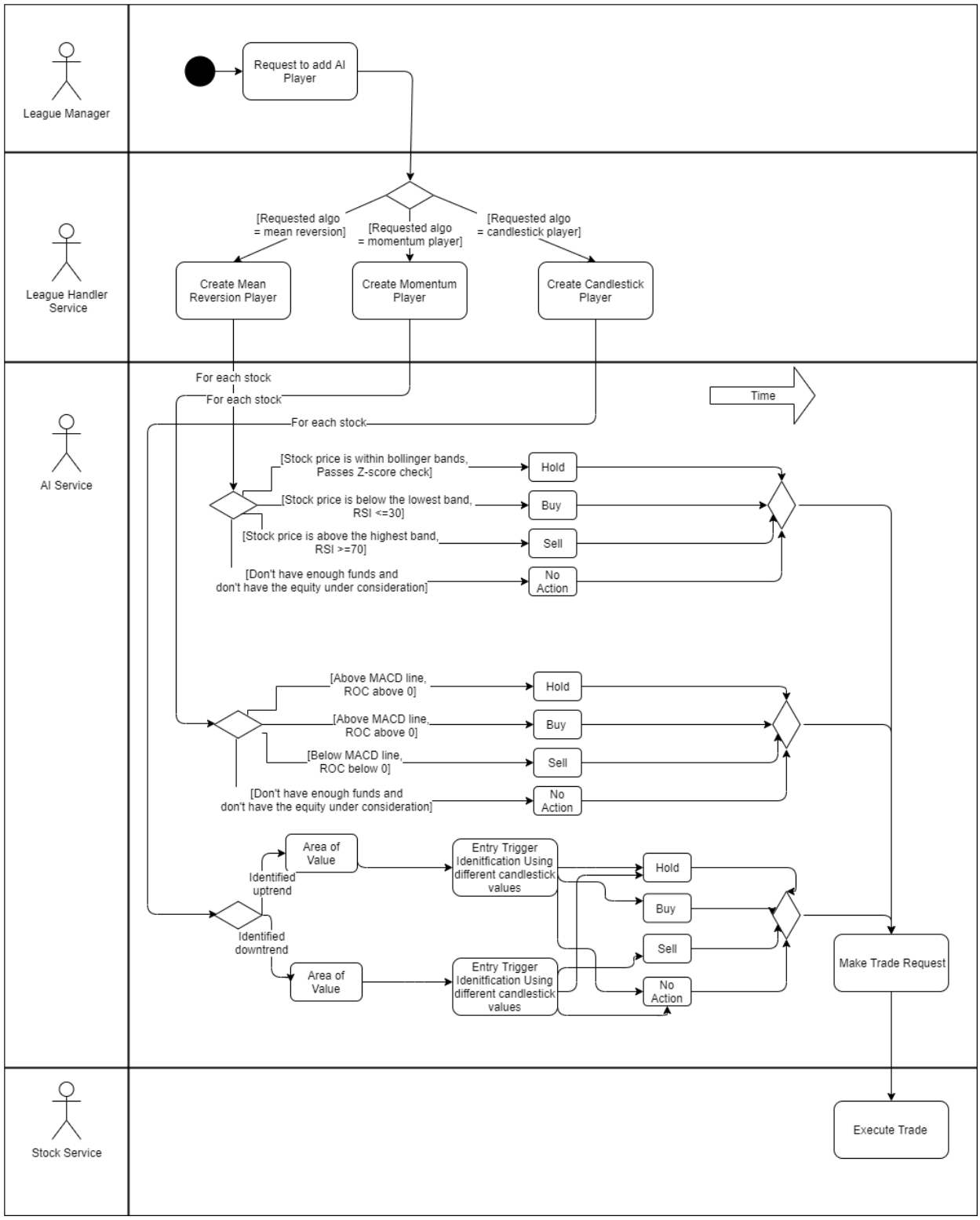
	CD-1	CD-2	CD-3	CD-4	CD-5	CD-6	CD-7	CD-8	CD-9	CD-10	CD-11	CD-12	CD-13	CD-14	CD-15	CD-16	CD-17
Controller (UC 1-11)				X													
Database Connection					X												
Login Interface													X				
Registration Interface													X				
Investor View Interface													X				
League Handler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

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

5 Algorithms

5.1 Algorithms

The following technical indicators can be acquired through the IEX Cloud API. Our stock service aims to request the API for this data and store it in our database often. When this information is needed, it can easily be pulled from our database and used to determine the best time to trade an equity. These indicators cannot be used alone to solely determine the price direction of an equity. However, when used together, they work to create a somewhat reliable basis for placing trades.


League Manager
Request to add AI Player
League Handler Service
[Requested algo = mean reversion]
[Requested algo = momentum player]
[Requested algo = candlestick player]
Create Mean Reversion Player
Create Momentum Player
Create Candlestick Player
AI Service
For each stock
For each stock
For each stock
Time
[Stock price is within bollinger bands, Passes Z-score check]
Hold
[Stock price is below the lowest band, RSI <=30]
Buy
[Stock price is above the highest band, RSI >=70]
Sell
[Don't have enough funds and don't have the equity under consideration]
No Action
[Above MACD line, ROC above 0]
Hold
[Above MACD line, ROC above 0]
Buy
[Below MACD line, ROC below 0]
Sell
[Don't have enough funds and don't have the equity under consideration]
No Action
Area of Value
Identified uptrend
Entry Trigger Idenitfication Using different candlestick values
Hold
Buy
Sell
No Action
Identified downtrend
Area of Value
Entry Trigger Idenitfication Using different candlestick values
Make Trade Request
Stock Service
Execute Trade

5.1.1 Bollinger Bands

The Bollinger Bands are a technical analysis tool that comprises of three trend lines: a lower band, a middle band and an upper band. The middle band represents the Simple Moving Average (see 5.1.2) over a 20 day period, while the upper and lower bands are plotted two standard deviations (see 5.1.3) away from the SMA, both positively and negatively. These bands helped to identify oversold and overbought conditions. The upper and lower support lines help give traders confidence that the price is moving as expected. If the prices continually touch the upper band, it means the equity is being overbought, and if it continually touches the lower band, it means it is being oversold. The majority of the price action should occur between 2 of the bands. If prices breakout of this region, it is considered a major event. The bands expand if the price of the equity becomes volatile and will contract if the equity displays a tight trading pattern.

5.1.2 Simple Moving Average

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

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

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

5.1.3 Standard Deviation

The standard deviation is a statistic that helps measure the volatility of an equity. It measures the amount of variation relative to the average price. If the prices are further away from the mean, this means that there is a larger spread of prices and thus a higher standard deviation. Higher standard deviations indicate volatility in equity price action. Equities with low standard deviation are much more stable and generally have lower risk.

It can calculated as the square root of variance:

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

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

5.1.4 Relative Strength Index

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

RSI can be calculated using the following formula:

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

where RS = average gain / average loss

5.1.5 Rate of Change

The rate of change (ROC) is a technical momentum indicator that measures the percent change between the current price and the price taken before a specified time period. A ROC that is increasing and above 0 generally indicates that the equity is in an uptrend, while a decreasing ROC that is below 0 indicates a downtrend. This helps to determine the momentum trends of an equity.

ROC can be calculated using the following formula:

$$ROC = \frac{Closing\ Price_C - Closing\ Price_{C-N}}{Closing\ Price_{C-N}}$$

where $Closing\ Price_C$ = current closing price

and

$Closing\ Price_{C-N}$ = closing price N periods before current period

5.1.6 MACD

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

5.2 Data Structures

Our system is written almost entirely in Javascript and makes use of Javascript objects to handle most of our data structure requirements. Javascript objects behave similarly to hash tables with key-value pairs but are implemented differently in each Javascript engine. This structure is both highly flexible and performant. Every entity in our system can be stored as an object with easily readable and accessible fields.

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

5.3 Concurrency

Since we are running a Node application, everything runs on a single thread.

6 User Interface Design & Implementation

6.1 Page Design Updates

At this point in the project there have been no significant changes to the mock-ups made for the application. We are also in the early stages of the project, so maybe there may be some changes to the overall setup of each page. In this current stage, we are also more concerned with matching the pages as closely to the mock-ups as possible before looking at the implemented design objectively. When we do additional testing in the future, there may be some significant changes that are made to the setup of specific pages.

6.2 Page Efficiency

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

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

We are also trying to ensure that we can guarantee the user an enjoyable viewing experience, regardless of the device that they are using. Currently, we are trying to focus on the viewing experience that a user would have on a laptop or desktop computer. So we have taken great care in checking the look of each page on all the different devices that members in our group use. It has been difficult, but by trying to maintain this type of consistency early on, we hope to give ourselves time to focus on some of the more difficult implementation aspects of the application.

7 Design of Tests

7.1 View Equity Information

Test Case Identifier:	TC-1
Use Case Tested:	UC-5, UC-7, UC-8, UC-9, UC-10
Pass/fail Criteria:	This integration test passes if the user is able to access all equity data, including prices, metrics, and news
Input Data:	Equity ticker
Test Procedure:	**Expected Result:**

1. Request equity prices from an invalid ticker	System returns error message of “invalid ticker”
2. Request equity prices from a valid ticker	System gathers recently cached prices or retrieves new prices and caches them; System displays prices to user
3. Request equity metrics from an invalid ticker	System returns error message of “invalid ticker”
4. Request equity prices from a valid ticker	System gathers recently cached metrics or retrieves new metrics and caches them; System displays metrics to user
5. Request equity news from an invalid ticker	System returns error message of “invalid ticker”
6. Request equity news from a valid ticker	System gathers recently cached news or retrieves new news and caches them; System displays news to user

7.2 Login and Registration

Test Case Identifier:	TC-2
Use Case Tested:	UC-1
Pass/fail Criteria:	The test passes if the user enters a valid username along with the correct associated password and/or creates an account with the appropriate credentials
Input Data:	Username, password
Test Procedure:	**Expected Result:**
1. Logging in with valid username and incorrect password	System returns error message of “invalid username or password” and is prompted to try again
2. Logging in with invalid username and some password	System returns error message of “invalid username or password” and is prompted to try again
3. Type in too short of a username or password	System returns error message of “username must be minimum 3 characters” or

	"password must be minimum 6 characters" and is prompted to try again
4. Registering with a username that is already taken	System returns error message "username already in use" and is prompted to try again
5. Logging in with valid username with correct password	System returns success message and user is able to access restricted parts of the application

7.3 Trading

Test Case Identifier:	TC-3
Use Case Tested:	UC-10
Pass/fail Criteria:	This case tests whether the user is able to buy or sell a particular equity. The test passes if the transaction is executed and the user is able to view this purchase in their portfolio.
Input Data:	League Name, Equity Symbol, Transaction Type, Quantity and Price

Test Procedure:	**Expected Result:**
1. Select the particular league name from the dropdown menu	System returns an error message of "Cannot preview order without league name" and is prompted to try again
2. Type the equity name.	System returns an error message of "Cannot preview order without existing stock symbol" and is prompted to try again
3. Select the transaction type from the dropdown menu	System returns an error message of "Cannot preview order without transaction type" and is prompted to try again
4. Type in the quantity.	System returns an error message of "Cannot preview order without a valid quantity" and is prompted to try again.
5. Select the price type	System returns an error message of "Cannot preview order without selecting a price

6. If price type selected is either limit or stop, type in the limit price or stop price accordingly	type" and is prompted to try again. System returns an error message of "Cannot preview order without a limit price or a stop stop price" and is prompted to try again.

7.4 Stock Symbol Lookup

Test Case Identifier:	TC-4
Use Case Tested:	UC- 9
Pass/fail Criteria:	This case tests whether the user is able to look-up information on a particular equity. The test passes if the information is displayed.
Input Data:	Equity symbol

Test Procedure:	**Expected Result:**
1. Type in a stock symbol	System returns error message of "invalid stock symbol" and is prompted to try again

7.5 Create League

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

Test Procedure:	**Expected Result:**
1. User enters a league name that is already in use.	System returns error message of "already taken league name, please enter a new name"

2. User tries to put an invalid start or end date.	System returns error message of “Please provide a valid [start/end] date”
3. User tries to input a commission percentage of below 0% or above 100%	System returns error message of “Please provide a valid comm percentage up to 2 decimal places”
4. User tries to leave required inputs blank.	System returns error message “Please provide a [appropriate value]”
5. User types in valid inputs and successfully hits the “Create League” button.	System stores new league information in the database, and the user becomes the league manager.
6. User tries to access create league page without proper authentication	System redirects user to login page and prompts them for authentication

7.6 Join League Page

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

Test Procedure:	**Expected Result:**
1. User enters a league name that does not exist	System returns error message of “league not found”
2. User tries to join a private league without specifying a league key	System returns error message of “Invalid league key”
3. User tries to join a private league with an incorrect league key	System returns error message of “Invalid league key”
4. User tries to join an existing public league	System stores user as member of the league and initializes a blank portfolio for the new member and returns a success message

5. User tries to join an existing private league with a valid league key	System stores user as member of the league and initializes a blank portfolio for the new member and returns a success message
6. User tries to access join league page without proper authentication	System redirects user to login page and prompts them for authentication

7.7 Navigation Bar

Test Case Identifier:	TC-7
Use Case Tested:	UC-2, UC-3, UC-5, UC-6, UC-7, UC-8, UC-9, UC-10
Pass/fail Criteria:	This case tests whether the user is able to navigate to each page within the application via the navigation bar.
Input Data:	No input data
Test Procedure:	**Expected Result:**
1. Click Home	This should route the user to the Home Page
2. Click Portfolio	This should route the user to the Portfolio Page
3. Click Trade	This should route the user to the Trade Page
4. Click Leagues	This should be compressed when the page loads. Then it should open up when it is clicked once and then close when clicked again.
6. Current Leagues	This should route the user to the Current

	Leagues Page
7. Join League	This should route the user to the Join Leagues page
8. Create League	This should route the user to the Create Leagues page
9. Transaction History	This should route the user to the Transaction History page
10. Symbol Lookup	This should route the user to the Symbol Lookup page
11. News	This should route the user to the News page
12. Settings	This should route the user to the Settings page

8 Project Management & Plan of Work

8.1 Merging Contributions from Individual Team Members

8.1.1 Version Mismatching

One of the version issues that we ran into was when we were setting up npm for the project. At the time everyone had the same version number, but because of an update, one group member's npm version was different from everyone else's. As a result, many lines of code kept getting deleted when members tried to open pull requests for their changes. We finally realized the version difference after some deep diving in the code and were able to make the necessary changes on each person's system to fix the recurring issue.

Another situation of mismatching was when downloading all the different dependencies. At times there could be compatibility issues because certain dependency versions could not be managed by other dependency versions. So

there was a lot of testing dependency version compatibility to ensure that all the necessary dependencies were functioning in an efficient manner.

8.1.2 Setting up file structure and boilerplate code

Some may argue that this is not necessarily an issue, but we felt it was a prerequisite step that we needed to take as a group to minimize merge conflicts and facilitate individual contributions to the code base. In order to do this, considerable research was done in order to outline our frontend and backend file structure properly to make it easier to delegate work when implementing each feature of our application. For some important files, we included boilerplate code in order to provide more guidance and to keep our implementations more consistent across our application.

Additionally, taking these steps made it easier to create issues on Github, which are quite helpful for delegating tasks and responsibilities while minimizing chances for conflicts.

8.1.3 Setting up the linter

Setting up a linter was a challenging step since many of us were not informed of proper code conventions and style, especially in Javascript. Once again, we spent some time researching certain popular Javascript style guides we wanted to make use of for our codebase and enforced them with ESLint. Additionally, we added lint rules we felt would make our code base appear more consistent and uniform across the board.

8.2 Project Coordination & Progress Report

8.2.1 UC-1: Login

A registered user will be able to log in using their credentials. This allows users to access their secured account. We have created the user interface for the login page and the new user registration page. We have also implemented authentication for the login page that makes use of JSON Web Tokens, a popular token based authentication method. When a user logs in, a unique token will be stored in the browser and assigned to the user. When the user makes a request to the stock service or league service, they will pass said token in order to authorize these requests.

8.2.2 UC - 2: Create League

A user can become a league manager by creating their own league. This allows users to customize the way that they play. League managers are given a wide variety of settings to play around with, such as the number of AI player bots, the start and end dates of the league, and commission fees. The backend portion of league creation is almost finished, as API requests can be sent to create leagues in the database with the specified parameters. The frontend framework is present, but the form still needs to be made. The backend of the league creation is finished and tested to be working properly. However, more tests will be required when the frontend is complete.

8.2.3 UC - 3: Join League

A user can join a league by clicking on one of the "join league" buttons in the list of public leagues. They can also enter a private key given by someone and join a private league. This allows users to play with their own league portfolio and compete against other players in the league. We have almost completed the UI portion of this page. Currently, the back end portion of joining leagues is complete. Provided that the user can make an API request, a user can successfully join and leave leagues as they please. The backend current has basic error checking, but more will be added as we integrate the front and back end components.

8.2.4 UC - 4: Add AI player

A league manager can choose to include up to three different AI Trading Bots in a league. The AI Trading Bots each implement a different algorithm to optimize returns. This will allow other players in the league to be challenged and compete with more advanced players while also learning more about the different techniques that the AI trading bots use. These trading bots will have a select set of equities to choose from when placing trade orders and they will check the data periodically.

8.2.5 UC - 8: Read News

The user will be automatically presented with news on various equities relevant to their portfolios. The articles from API have been pulled from news sites that mention the company name. Additionally, users will have access to a catered news feed, which compiles the news of all stocks in a user's portfolio and presents articles in a list based on date and the user's positions (i.e. the user's top holdings will be prioritized). Users will then be able to click on any article and have it open up in a new browser tab.

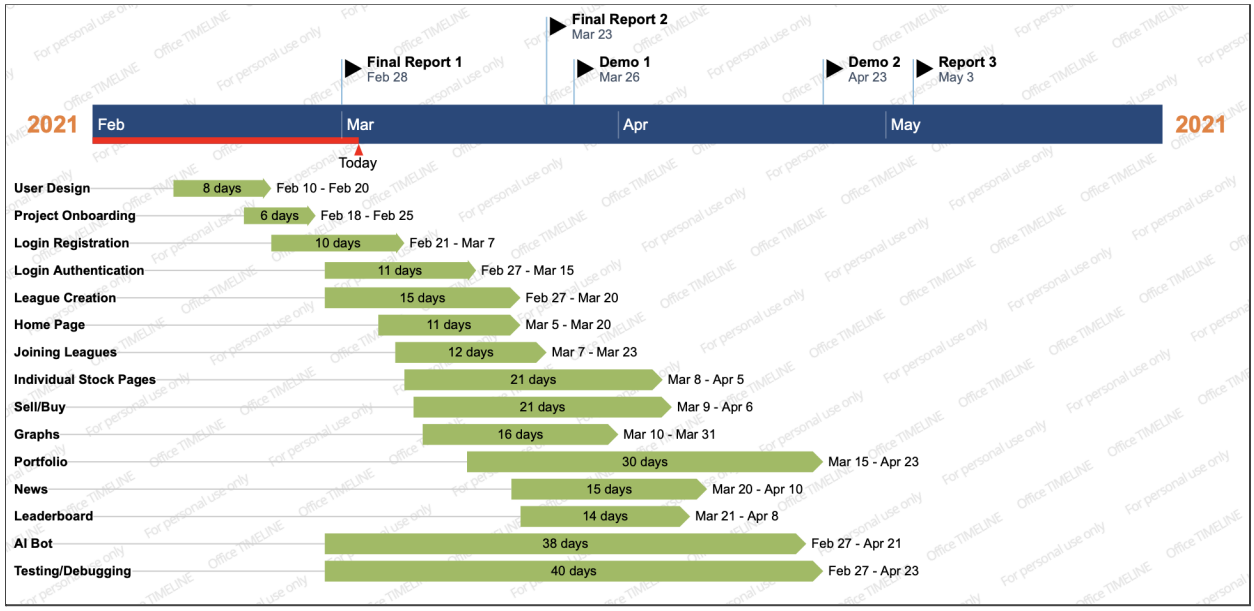
8.2.6 UC - 9: Research equity information

A user will be able to search a stock symbol using a search bar. A detailed set of information will show up on the individual equity page to help the user make an informed trade. We are in the process of finishing the UI portion of this page. Our stock service that provides all of the information to the application is mostly completed and functional. Current, intraday, and historical prices are all available. Additionally, additional equity statistics, such as market cap and 52 week high.

8.2.7 UC - 10: Trade

The user will be able to make trades during market hours for any available stock. They can place market or limit orders and decide if they want to sell or buy the stock. The user can also cancel an order before it is executed. We are in the process of creating the UI elements for the buy/sell page. There are a lot of elements that will be incorporated such as which league the user is in to purchase or sell the stock, stock symbol, the quantity, the price, the duration, and the account details of the user. There is currently functionality through our league service API for users to complete market orders - buying or selling an equity at the current market price. We are currently in the process of developing the UI elements for users to access these features.

8.3 Plan of Work



8.4 Breakdown of Responsibilities

8.4.1 Aarushi Pandey

Currently I am collaborating with my partners Aarushi S and Sahitya G and are working on the equity research page which will display the current information of the specific equity. This page will display a graph of the equity compared to s&p 500 as well as other equity information. I have also finished working on the trade page with my partners. This page will allow the user to buy and sell stock. In this section the player will also be able to view information only pertaining to the equity they are currently purchasing as well as their account details. In addition to this I have also finished the equity look up user interface page with my partners which will lead to the equity research page.

8.4.2 Aarushi Satish

Currently working on creating the user interface portion of the trading equities component of the application with Aarushi P and Sahitya G. This entails creating components for placing orders on equities. More specifically, I will be working on implementing the order stock segment of this page. This segment allows users to trade stocks within their league with specifications such as duration and price. I will also be working on the search bar component which allows users to search more detailed information about a specific equity.

8.4.3 Apoorva Goel

Currently working on the user interface portion of the league creation page with Yatri. This will allow a user to become the league manager by creating a league. The league manager can also make the league public or private, and initialize the settings of the league. Also currently working on the join league page with Yatri, which will allow users to join public and private leagues.

8.4.4 Christine Mathews

Currently working on an endpoint that serves portfolio information for each user in any league that they participate in. This includes data such as current net worth, holdings and a graph of progress over time. Additionally, I am working on planning out the logistics of the AI players. I worked on creating a stock statistics endpoint which serves to grab financial information about individual stocks from the third-party API and store it in our database. Examples of these statistics include market capitalization, PE ratio, earning's dates etc. This service extracts the desired metrics from the API response that are required by our application.

8.4.5 David Lau

Developed a historical and intraday stock price endpoint to serve all price information to the application. This service aims to remove the dependency on the third-party API that we pull stock information from by acting as cache storage. Completed trading functionality with Jacques to allow users to place market orders on equities. Currently working on further developing trading to allow limit orders, develop endpoints to provide portfolio information, and implement tools for the league service to accurately depict portfolio value.

8.4.6 Jacques Scheire

Created user interfaces for login and registration as well as implemented their respective functionalities on the backend. Also finalized implementation of user authentication and authorization with use of passport.js and JWTs. Thus, this has added a layer of security to our application such that only authenticated users will be able to access certain parts. Users will now need to pass a bearer token into the authorization header of a request in order to access certain protected routes across the application. I have also implemented the backend league service, which serves to allow authenticated users to create and join leagues in which they can trade equities. Additionally, I have implemented portfolio functionalities that allow users to manage their personal portfolio in every league that they are a part of. Currently working on integrating stock service functionality with the league service to have a user's individual portfolio reflect the actions they take in a given league.

8.4.7 Jawad Jamal

Worked on creating the basic functionalities of news endpoints which can pull news tailored to a defined list of securities. Currently working on expanding the functionalities of the news endpoint. It will be able to present a curated list of articles to users based on the stocks in the user's portfolio. Additionally, I will also be working on creating the chart component which graphs data pulled from the historical equity data stored on the database. I will also assist in adding functionality to the home page of the project.

8.4.8 Krishna Prajapati

Worked with Jacques and finished the backend league endpoints that allow a user to join a league, create a league, disband a league, and query for leagues. Also tested and debugged these endpoints to confirm that they are working correctly, but I will perform more testing as Apoorva, Yatri, and I finish up the frontend of the createLeague and joinLeague pages. Additionally, I will work on making

more validations for this portion of the web application after finishing a minimum viable product. Currently working on the frontend of the joinLeague page, and will be working on connecting the front and back ends of the createLeague and joinLeague.

8.4.9 Riya Tayal

After Jacques finished creating the login and registration page I helped to fix the overall setup of the page so that it matched our original vision for what these pages would look like. I finished working on the navigation bar and it successfully sits on the left hand side of the screen and allows the user to navigate between the different pages of the application. Currently there are not many pages to navigate to, but once the pages are created all that needs to be done is to add the routing ability. I have now begun working on the home page and hope to have the front-end set up and aspects of the backend functioning as well. The home page will include information about the user, similar to the way a user profile would look but will include snapshots of information from the other pages in the application.

8.4.10 Sahitya Gande

Currently working on creating the user interface components for the individual stock page with Aarushi S and Aarushi P. This page displays all the information of the equity that the user searched for. This includes different graphs of the stock and other important values the user will be interested in. I finished the frontend portion of the trading equities page and the symbol lookup page. For the trading equities use case, I will be working on the account details portion of the page. This will show the user how much account value they have, their buying power, and the amount of cash they have.

8.4.11 Yati Patel

Currently working on an endpoint that serves portfolio information for each user in any league that they participate in. This includes data such as current net worth, holdings and a graph of progress over time. Additionally, I am working on planning out the logistics of the AI players. I worked on creating a stock statistics endpoint which serves to grab financial information about individual stocks from the third-party API and store it in our database. Examples of these statistics include market capitalization, PE ratio, earning’s dates etc. This service extracts the desired metrics from the API response that are required by our application.

8.4.12 Yatri Patel

Currently working on the user interface portion of the league creation page and league joining page with Apoorva, which will allow a user to become the league manager by creating a league or join a league, respectively. The creation page will allow the league manager to fill out a form to set the initial settings for the league. The join league page displays a table of public leagues to join, as well as a box to enter the code for a private league. Also, I am currently working with Apoorva on improving the user experience with league creation and joining by preventing certain bad inputs from being submitted.

9 References

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

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

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

[4] Rayner, "The Essential Guide to Momentum Trading", *TradingWithRayner*, 08-Oct-2020. [Online]. Available: https://www.tradingwithrayner.com/the-essential-guide-to-momentum-trading. [Accessed: 04-Mar-2021]

[5] R. Cameron, "Momentum Day Trading Strategies for Beginners: A Step by Step Guide", *Warrior Trading*, 31-Mar-2015. [Online]. Available: https://www.warriortrading.com/momentum-day-trading-strategy. [Accessed 04-Mar-2021]

[6] J. Fernando, "Moving Average Convergence Divergence (MACD) Definition", *Investopedia*, 05-Jan-2021. [Online]. Available: https://www.investopedia.com/terms/m/macd.asp. [Accessed 04-Mar-2021]

[7] C. Mitchell, “Price Rate Of Change Indicator (ROC)”, *Investopedia*, 29-May-2020. [Online]. Available: https://www.investopedia.com/terms/p/pricerateofchange.asp. [Accessed 04-Mar-2021]