Red Trade Investments

REPORT 2

Fort Hays state University

CSCI 441 VA Software Engineering

Professor Hieu Vu

March 15th, 2020

Taylor Pressley – Team Lead

Jayesh Nair

Gideon Lal

Hyunsook Shin

Michael Nguyen

Repository: <https://github.com/jnair1993/CSCI-441-Software-Engineer-project-RedTrade>

Shared Google Drive: <https://drive.google.com/drive/folders/1P2a4BUPlheVONdbzUB5T61HSl6d4yJso?usp=sharing>

Table of Contents

[1. Interaction Diagrams 3](#_Toc35205228)

[1.1 Introduction 3](#_Toc35205229)

[1.2 Diagrams 3](#_Toc35205230)

[1.2.1 Use Case 1 3](#_Toc35205231)

[1.2.2 Use Case 2 4](#_Toc35205232)

[1.2.3 Use Case 3 5](#_Toc35205233)

[1.2.4 Use Case 4 5](#_Toc35205234)

[1.2.5 Use Case 5 6](#_Toc35205235)

[1.2.6 Achievements Sub 7](#_Toc35205236)

[1.2.7 Alternate Solution Diagramming 7](#_Toc35205237)

[1.2.8 Authentication Sequence 9](#_Toc35205238)

[1.2.9 Place Market Order 10](#_Toc35205239)

[1.2.10 Show Manage Portfolio 11](#_Toc35205240)

[1.2.11 Show Market Data 12](#_Toc35205241)

[1.2.12 Design Pattern 12](#_Toc35205242)

[2. Class Diagrams and Interface Specification 14](#_Toc35205243)

[2.1 Class Diagram 14](#_Toc35205244)

[2.2 Data Types and Operation Signatures 14](#_Toc35205245)

[2.3 Tracebility Matrix 19](#_Toc35205246)

[3. System Architecture and System Design 19](#_Toc35205247)

[3.1 Architectural Styles 19](#_Toc35205248)

[3.2 Identifying Subsystems 19](#_Toc35205249)

[3.3 Mapping Subsystems to Hardware 21](#_Toc35205250)

[3.4 Persistent Data Storage 21](#_Toc35205251)

[3.5 Network Protocol 21](#_Toc35205252)

[3.6 Global Control Flow 22](#_Toc35205253)

[3.6.1 Execution Orderness 22](#_Toc35205254)

[3.6.2 Time Dependency 23](#_Toc35205255)

[3.6.3 Concurrency 23](#_Toc35205256)

[3.7 Hardware Requirements 23](#_Toc35205257)

[4. Algorithms and Data Structure 24](#_Toc35205258)

[4.1 Algorithms and Data Structures 24](#_Toc35205259)

[5. User Interface Design and Implementation 25](#_Toc35205260)

[6. Design of Tests 25](#_Toc35205261)

[6.1.1 Test Cases 25](#_Toc35205262)

[6.1.1.1 Test Case 1 – Check to make sure that a new account can be created with valid data 25](#_Toc35205263)

[6.1.1.2 Test Case 2 – Check to make sure that a new account cannot be created if data is invalid or incomplete. 25](#_Toc35205264)

[6.1.1.3 Test Case 3 – Check customer login with valid data. 25](#_Toc35205265)

[6.1.1.4 Test Case 4 – Check customer login with invalid data. 25](#_Toc35205266)

[6.1.1.5 Test Case 5 – Check that client can make trade with valid data. 25](#_Toc35205267)

[6.1.1.6 Test Case 6 – Check that client cannot make a trade if data is invalid or incomplete. 26](#_Toc35205268)

[6.1.1.7 Test Case 7 – Check that client can update profile information. 26](#_Toc35205269)

[6.1.1.8 Test Case 8 – Check that client can access video tutorials and training on the educational page. 26](#_Toc35205270)

[6.1.2 Test Coverage 26](#_Toc35205271)

[6.1.3 Integration Testing Strategy 26](#_Toc35205272)

[7. Plan of Work 26](#_Toc35205273)

[8. Project Management 30](#_Toc35205274)

[8.1 Project Management 30](#_Toc35205275)

[9. References 30](#_Toc35205276)

# 1. Interaction Diagrams

## Introduction

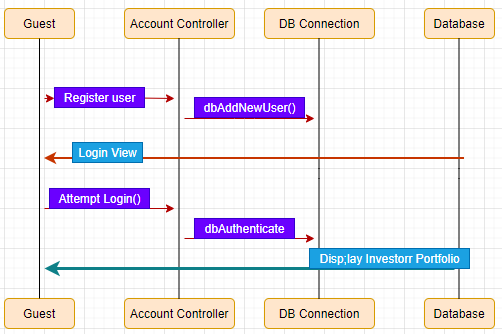
An interaction diagram will be used to outline the system interactions, an important part of our software. Each use case that is used, we will give an outline of how each part interacts with each other as well as analyze how the Use Systems will handle different types of scenarios. The diagrams will showcase how the systems can handle failure and success conditions within the following scenarios: Database, controller, and API. Due to the fact that this is a web-based and data based project, the database and controller play a hefty role within the system interactions/ Users will be promoted to log in and constantly access data pulled form an API to have a refreshed and more up to date information.

## Diagrams

### Use Case 1

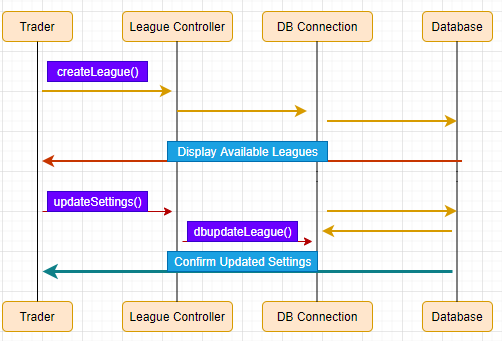
UC-1 shows the user two options: Login or Register. If the user wishes to make a new account, an Account Controller is activated with the user’s personal information. The Account Controller can try to check and make sure no other duplicate information exists within the database. The database will not store the guest user’s info but like other account creations, a confirmation email will be sent to the user in which the Account Controller will update along with the Login View.

When the user wants to log in, the account controller will attempt to access the log in details fetched form the database. If the details match accordingly, the Account Controller will send the guest into investor mode and display the user profile in which they can update the information they wish to be displayed to the public.



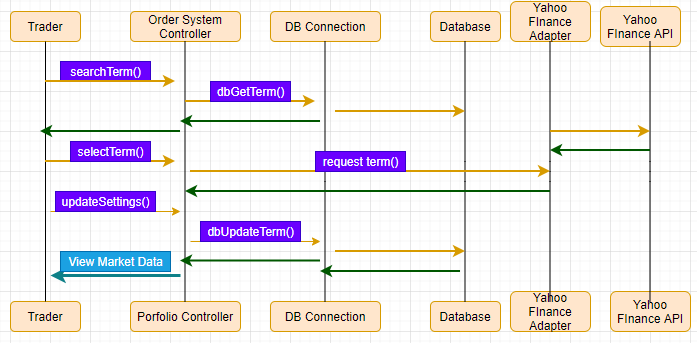
### Use Case 2

When the user wants to view or update their Player Profile, there will be an updateSettings() function in which will be processed through the database and allow settings to be updated for the league. To join the league is a blunt process; a league controller will be put in place and show the available teams the player or user and join as they see fit.



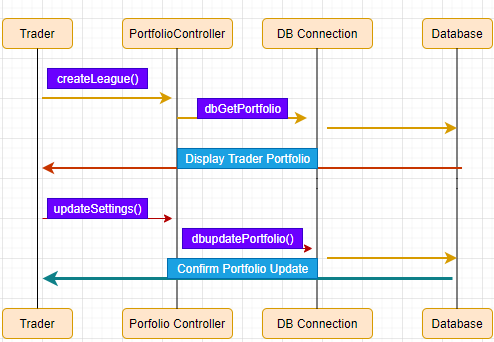
### Use Case 3

When the user wants to view market data, and Order System Controller is put into place. The system will fetch matches from the database and display them to the player. The investor (user) will choose a match and the Order System Controller takes the chosen term and request data.



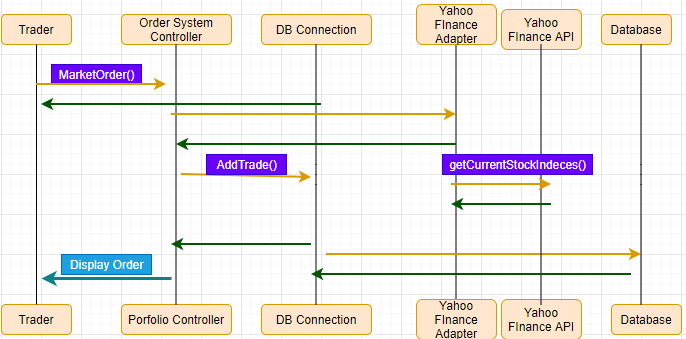
### Use Case 4

In most cases, investors and users will want to update and make changes to their profile. When the user clicks on portfolio or profile, the portfolio controller will fetch the investors stocks from the database. Alongside that, they can update to their choosing.



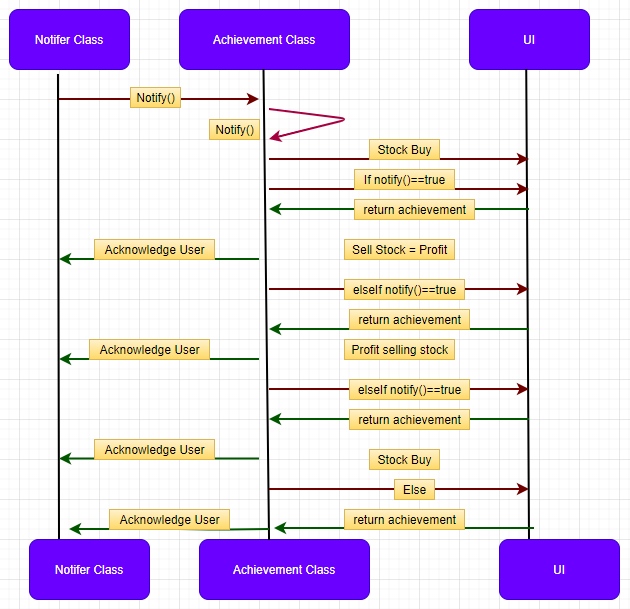
### Use Case 5

As an investor, you want to be able to place orders. When an order is placed, the Order System Controller will retrieve the price of stock that it is currently going for. When a price is found, the Order System Controller must be confirmed through the database and make sure the investor has enough funds to make the purchase. When confirmed, the information about the purchase will be stored in the Database.



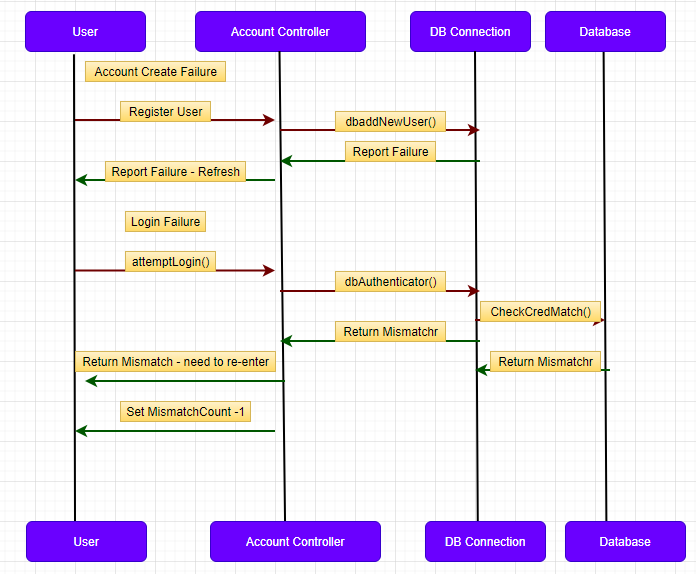
### Achievements Sub

When a player earns or receives an achievement reward, a subscription or publisher type pattern is used to handle these achievements in which will be updated in the database for the user. Instead of handling it in one location, the spreading of the logic code is used throughout making the achievements addition be controlled with more ease.

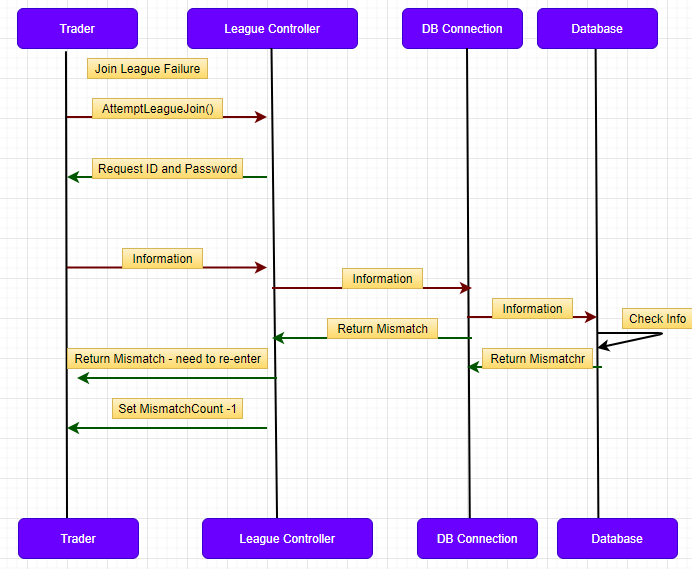


### Alternate Solution Diagramming

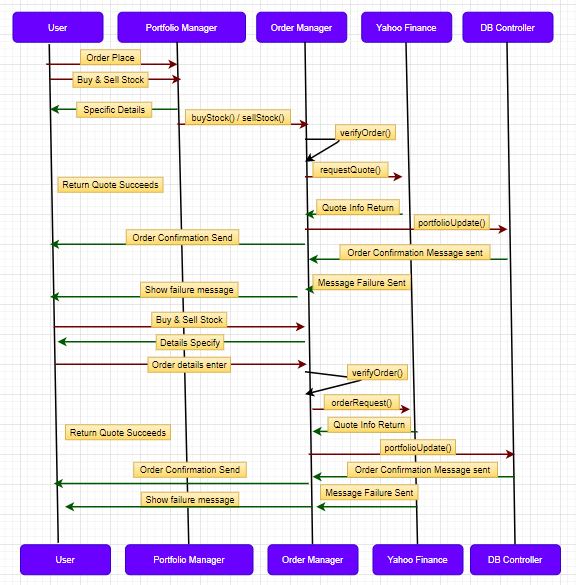
When it comes to Software design, it should be about alternative solutions to your first and picking the best one at hand to base your criteria from.



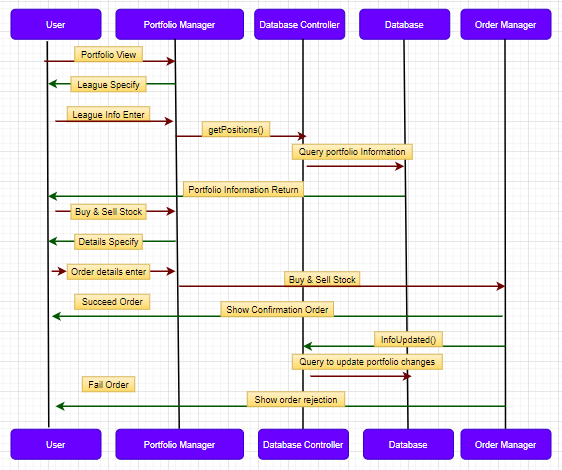
### Authentication Sequence



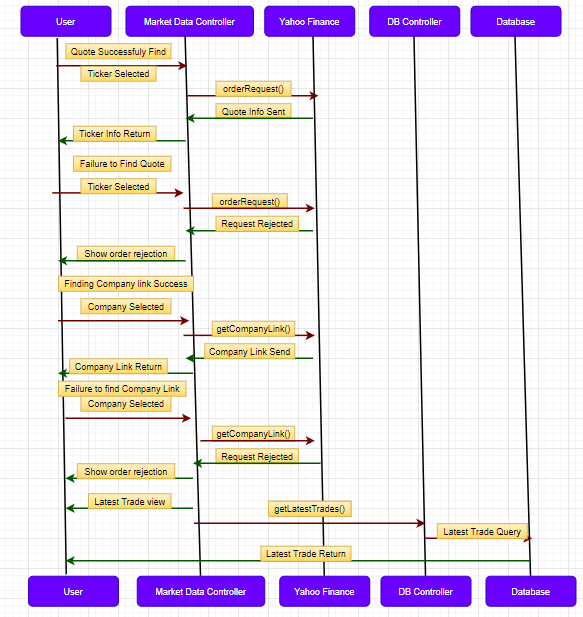
### Place Market Order



### Show Manage Portfolio



### Show Market Data



### Design Pattern

Different variations of standard and non-standard designs were made to provide functionality for different functions such as authentication.

* + - 1. Model View Controller

The Model-View-Controller (MVC) design was vigorously utilized all through the framework to appropriately arrange model rationale, business rationale and introduction rationale. This exceptionally instinctive example permitted the group to handily appoint chip away at various degrees of the framework. Every now and again, a determination of group individuals would create front-end usefulness which required just the perspectives to be modified, while different individuals actualized backend usefulness which was done either in controllers or models. Specifically, the MVC design approaches assets just when they are really required which forestalls superfluous overhead. For instance, strategies created to be called as it were automatically don't endeavor to show a view which brings about quicker reactions.

* + - 1. Publisher / Subscribe

The Publisher/Subscribe design permits us to effortlessly decouple our treatment of business rationale from that of our game rationale, most remarkably, our accomplishments framework. This permits us to no problem at all change business rationale without perusing code devoted to accomplishment dealing with. We likewise accept that we can utilize the Publish/Subscribe model later to stretch out usefulness to restrict and stop orders, permitting entertainers to be refreshed when a request has been finished.

* + - 1. Object Relational Model Pattern

The Object Relational Model (ORM) design, a smart execution of a database get to configuration design, was utilized only to collaborate with constant stockpiling innovations utilized in the framework. This example offered the significant bit of leeway of not expecting to hard code any database specific questions. All solicitations made through the ORM are meant the as of now utilized DB framework's language and information is returned in legitimately its item structure. The absence of need to compose direct inquiries likewise lead to an extraordinary reaction, specifically database rationalism which permitted different database executions to be tried during various phases of improvement. During improvement, SQLite was utilized for its lightweight impression on the designer’s machine, at that point for creation MySQL was utilized as it is increasingly proficient when managing bigger measures of information. This plan unquestionably improved upon advancement by endless long periods of improvement time.

* + - 1. RESTful Design

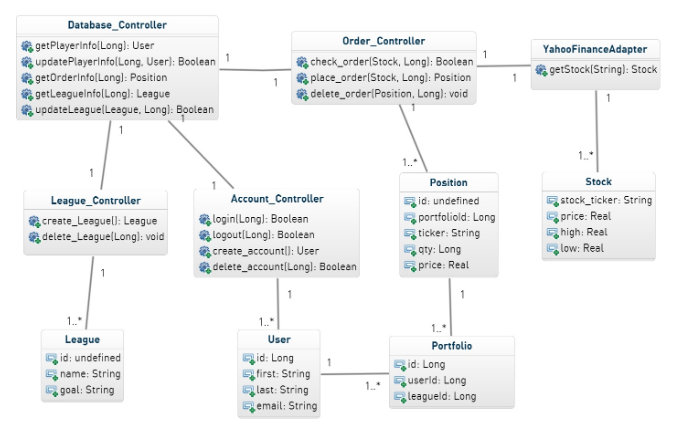
The RESTful arrangement configuration being used progressively more now on the web allowed us to execute our unique solicitation getting ready structure. The RESTful structure of some internal convenience allowed it to be found a good pace securely through a fundamental API. As RESTful organizations are at the center of the Play Framework, it didn't require a lot of effort to reveal some inside value without making huge security openings. Future accentuation will continue contingent upon the stateful correspondence that our RESTful API offers.

* + - 1. Responsive UI Pattern

The Bootstrap UI structure actualized a plan design totally isolating visual introduction from substance and client experience. This gave a wonderful responsive plan which adjusted to various customer gadgets going from work areas to shrewd devices. The example exploits the adaptable markup of HTML5 to tweak it on the fly when the page is rendering in the program utilizing JavaScript and CSS. This permitted our group to focus on the quickly developing versatile clients absent a lot of additional usage exertion. It has created a quicker client experience since significant preparation is done during the beginning of page rendering and for the most part once the page is visible to the client, we effectively strived to accomplish both objectives.

# 2. Class Diagrams and Interface Specification

## 2.1 Class Diagram



## 2.2 Data Types and Operation Signatures

Our database administrator plays out the limit of managing the database. This can mean anything from including customer information into the database, recuperating information from the database, reviving information in the database, regardless of whether the information oversees customers/records, unions and requests.

**Methods**

**+ get\_player\_info(in user id : long) : class User**

Retrieves information for specific player

**+ update\_player\_info(in user id : int, in upd: class user) : bool**

Updates user information

**+ get\_order\_info(in transaction id : int) : class transaction**

Stores the transaction ID and returns information related to the transaction

**+ update\_league(in league id : int, in leagueInfo : class league) : bool**

Returns the last updates made in the league

**+ return\_league\_updates(in league id : int) : class league**

Used when league needs to be updated with most updated information

Order Manager

Our supervisor class is liable for taking care of the considerable amount of errands identified with orders/exchanges. It is liable for putting in the request in the framework and for moving old requests to the document exchanges table

**Methods**

**+ Check\_order(in symbols: class Order) : bool**

Used to check and make sure that when orders are placed, they can be processed as well as used to check user’s balance.

**+ place\_order(in symbols: class Order) : bool**

Used to place orders with info from Stock Class

**+ delete order(in symbols: transaction id): bool**

Cancels order if a mistake was made if it’s not processed. If it has, then it will return a false value.

**+ Execute order(in transaction id : int) : bool**

Responsible for Stock info.

**League Manager**

This class is for dealing with all the groups in the framework. It has the power to make associations, erase classes, and alter groups as it is to do as such.

**Methods**

**+ Create league () : Class league**

Allows user to create their own league

**+ Delete leagues(in league id : int) : bool**

Allows user to delete their league

**+ change league name( in league id : int) : bool**

Allows user/leader of league to change their team/league name

**+ Change league manager (in league id : int, in usr : class User) : bool**

Allows user to give leadership of league too a difference player in the league

**+ add rules(in league id : int) : bool**

Allows user to add rules to their league if they choose

**+ Delete rules(in league id : int) : bool**

If they don’t want rules, they can just delete

Account Controller

Takes care of any functions that relate to the account or profile of user.

**Methods**

**+ Login(in user id : int) : bool**

Lets users log in.

**+ logout(in user id : int) : bool**

Lets user log out

**+ Verify User(in User id : int) : bool**

Confirmation to show that however is logging in is the correct person

**+ Create account() : class User**

Allows user to create an account

**+ delete account(in suser id : int) : bool**

Allows user to delete their account

Stock

Represent stock and who holds highest bid, etc.

**Attribute Methods**

**+ String:stock ticker**

Ticker Symbol

**+ double:price**

Current value of stock in relation to ticker symbol

**+ double:high**

Current High price of stock

**+ double:low**

Low price of stock

User

Responsible for user such as holding names, email and userid.

+ long:id

Distinguishes a user ID from one to another

+ String:first

First name

+ String:last

Last Name

+ double:low

Email address

Position

Represents who owns what stock

+ long:id

Distinguish different users from one another.

+ long:portfolioId

Distinguish different users from one another.

+ String:ticker

Holds the first name of the user.

+ long:qty

Quantity of the certain position.

+ double:price

Price that the position was purchased at.

Portfolio

Represents set of stocks user owns.

+ long:id

distinguish different portfolios from one another.

+ long:userId

distinguish who owns the portfolio.

+ String:leagueId

distinguish what league this portfolio is a part of.

League

Responsible for lead in which people belong to and can compete against each other.

+ long:id

A unique ID to distinguish different portfolios from one another.

+ String:name

This attribute holds the name of the league.

+ String:goal

Holds the necessary requirement for a person to be declared the winner of a league.

## 2.3 Tracebility Matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Class** | **Order Controller** | **Stock** | **User** | **League** | **DB Manager** |
| **Order Controller** |  | x | x |  | x |
| **League Controller** |  |  |  | x | x |
| **Database Manager** | x | x | x |  | x |
| **Yahoo Finance Adapter** | x | x |  |  |  |
| **Account Controller** |  |  | x |  | x |

# 3. System Architecture and System Design

## 3.1 Architectural Styles

To make the most use of our product, we will couple a few known programming devices and standards into our Architectural structure. The follow design types will be extended in detail to reflect general utility, yet additionally to reflect usefulness of the product all in all. Each will assume a critical job in the achievement of our product and will be to a great extent inferred from the necessities of the product. That being stated, structural frameworks will be incorporated (and may be developed later) the Model View Controller, Data-Centric Design, Client-Server get to, and RESTful plan.

## 3.2 Identifying Subsystems

Model-View-Controller

The Model View Controller is a User Interface use procedure which will seclude the programming into 3 express social events; that is: the model, view, and controller subsections. The view is normally limited to UI yield, for instance a site page with stock information. That being expressed, the model remains the middle section of the MVC procedure which holds the whole of the data, limits, and gadgets. The controller just takes the data and changes over it into a request for either the model or the view.

The MVC procedure is ideal for this programming since it allows the arrangement to be separated into tinier sub-issues. By separating into 3 segments, we can disconnect UI limits, from database works, and have all of them dealt with finally by the controller. Subsequently, to the extent smoothness of the structure, including the MVC allows each to be and mulls over the programming to be made far more straightforward.

Data-Centric Design

The Model View Controller is a User Interface usage technique which will isolate the programming into 3 explicit gatherings; that is: the model, view, and controller subsections. The view classification is regularly restricted to UI explicit yield, for example a website page with stock data. That being stated, the model remains the center segment of the MVC technique which holds the entirety of the information, capacities, and devices. The controller just takes the information and changes over it into an order for either the model or the view.

The MVC technique is perfect for this specific programming since it permits the plan to be broken down into littler sub-issues. By parting into 3 sections, we can isolate UI capacities from database works, and have every one of them took care of at last by the controller. As far as smoothness of the structure, including the MVC permits each to be and takes into consideration the programming to be made far simpler.

Client-Server Access

The client will be continually associating with the interface. The entirety of the collaborations is happening, hence, on a customer server premise. The client remains the VIP customer, and in that capacity, continually should collaborate with different subsystems. The entirety of the framework gave by Red Trade Investments should be gotten to by the client. This guarantees a smooth correspondence between each portion of the MVC and among customer and foundation. Furthermore, the foundation gave by Central speculations will have the option to get to foundation of non-acquainted frameworks.

Representational State Transfer

As a product actualizing a customer server Access framework, a REST framework is additionally characteristically inferred. The RESTful plan standards express that notwithstanding having a Client-Server Access framework, the framework has a versatility of parts, that the interface is uniform, stateless, and cacheable. Utilizing this technique will utilize a smooth, secluded arrangement of code. Utilizing the interface details inside the RESTful blueprint permits both the client and the fashioners to have streamline cooperation’s with the interface. That is the client knows plainly what the individual in question is doing when state a connection is tapped on a website page. The solicitation is changed over and conveyed to the controller.

## 3.3 Mapping Subsystems to Hardware

The Red Trade Investments League is contained on a MySQL database server which is put away on one machine. Be that as it may, the framework in general is spread over a few machines. The framework to be is partitioned into two separate areas: a front-end side that is run on the customers internet browser of decision, and a backend that spikes in demand for the server side of the database. The front-end is the principle graphical UI (GUI) between the framework and the customer. The front-end is liable for correspondence between the GUI and the database for example, affirming market orders and refreshing a speculators portfolio. These adjustments in the front-end are reflected in the back-end side of the server.

## 3.4 Persistent Data Storage

The arrangement for information stockpiling exists at the center of Red Trade Investments. Since such a large amount of our programming relies upon appropriately created and refreshed information, it is the very pinnacle of significant that our database blueprint speaks to precisely all items included. That is, the information should precisely (at all occasions) mirror all important client information, stock data, ticker factors, alliance settings, accomplishments, leaderboards, and all other pertinent articles. Central Investments will utilize the social database MySQL. Social databases are unquestionably progressively down to earth for the necessities of this specific programming.

That is, social databases comprise of a few ordered tables loaded up with different item traits. As can be seen in the class graphs on the past page, this is fundamental for the huge number of items which will be available in the product. Tables should exist not just for client information and settings such as sign in and class profiles, yet additionally for stock and portfolio data. Further, these databases should be continually composed and revised to guarantee continually refreshed and precise data. Things, for example, leaderboards, and data which will have the option to be seen on every client portfolio need to continually reflect exact information.

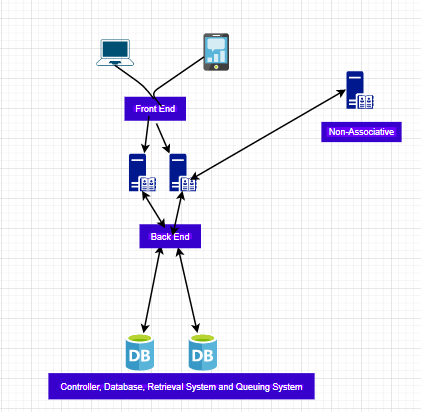
## 3.5 Network Protocol

As a standard for programming for this sort, Red Trade Investments will utilize the standard Hypertext Transfer Protocol (HTTP). HTTP by organizing content which is utilizes hyperlinks to convey messages through content between hubs. While not extraordinary or specific to our circumstance, it is yet critical to take note of that this will be the essential convention among client and programming interface. More significantly, the HTTP convention will be utilized on electronic gadgets as well as on Android and iOS gadgets also. From any of these mediums, the clients can get to different site pages and connections from the Red Trade Investment site. They will have the option to access, through this convention, all important stock, portfolio, and pertinent data through these pages and by utilizing the HTTP

## 3.6 Global Control Flow

### 3.6.1 Execution Orderness

As a rule, the execution of the framework at Red Trade Investments is generally, occasion driven. All the highlights that the framework brings to the table must be activated by some element, regardless of whether it be simply the client or some other piece of the framework. Most of the event driven trademark originates from the client end of the framework. A significant number of the functionalities (stock exchanging, portfolio seeing, alliance joining, and so on.) must be activated by the client. There are in any case, some occasion driven usefulness that are started by the framework. At the point when the client places a request, it is handled and included into the database.



### 3.6.2 Time Dependency

All in all, the framework at Red Trade Investments is especially an ongoing framework, however there are highlights that don't rely upon time. The ongoing framework is extremely dependent on the securities exchange, which itself has certain seasons of activity. As the client is perusing the site, there are ongoing clocks that help the framework procedure data that it is accepting

### 3.6.3 Concurrency

There are sub-frameworks in our primary framework which must be painstakingly thought of because of simultaneousness. The greatest of these is the lining subsystem. This creates a simultaneousness issue because we need to ensure that close to 1 request is being embedded into the line at some random time. Moreover, we additionally need to ensure that close to 1 request is being dequeued from guaranteed line. Other than this our framework truly needn't bother with any synchronization. Be that as it may, this may change as we are actualizing our framework.

## 3.7 Hardware Requirements

The equipment prerequisites on the server side are the principle commitment to the activity of Red Trade Investments League, leaving the customer side with insignificant prerequisites. The just prerequisite of a customer will that it runs a program that is equipped for running a cutting-edge web browser.

Internet Connection

All together for Red Trade Investments League to utilize any of its center capacities (exchanging stocks, refreshing client portfolio, following managerial activities, and so forth.), a web association is required.

Since the greater part of the information being moved is content (executable directions), a low band of recurrence is required. Note that a total adaptable examination has not been performed on the framework, so a low band of recurrence is dependent on the requirements of the present site. For perfect execution, higher transfer speeds of recurrence ought to be utilized to lessen any overhead. A system association between the server and the Yahoo Finance API is important during exchange hours (9:30am - 4:00pm Monday through Friday), in any case, no financial specialists can play out an exchange Disk Space.

The server must have enough hard drive space to have the alternative to store the total of the database information. All data being taken care of is the entire of all program headings for the system. 10 GB of limit space should be sufficient for the structure.

System Memory

Since this structure is in powerful improvement, there is obliged strong evidence that reinforces the all-around execution of the system. The structure will stack copies of database set away information in solicitation to work over it. For better throughput, the memory should be supervised using a Least Recently Used arrangement (LRU) to keep the system memory populated with significant information.

An LRU plan will release any bits of memory that haven't been found a workable pace some time, and it will replace it with information that is used even more much of the time. Similarly, any assignments used on stacked information will moreover experience system memory. In any event 512 MB should be used for testing our system. In addition, as our customer based broadens, plainly the system memory will moreover need to create with it.

Client-side Hardware Requirements

The center equipment prerequisite on the customer side of the framework will be a web association. This is basic for the customer to have the option to remotely associate with the server to get to the database. Without a web association, no customer will have the option to utilize an internet browser to visit the Red Trade Investments League site. Notwithstanding a web association, and for a well-disposed client experience, anybody on the customer side ought to have a practical mouse and console, just as a realistic presentation to see their portfolio. To show the Red Trade Investments League site, a screen with a base goal of 800x600 pixels is enough.

# 4. Algorithms and Data Structure

## 4.1 Algorithms and Data Structures

All our algorithms and data structures come from the financial API that we will have linked to the site. We will not be creating algorithms ourselves.

# 5. User Interface Design and Implementation

No changes were made since the first mockup.

# 6. Design of Tests

## 6.1.1 Test Cases

### 6.1.1.1 Test Case 1 – Check to make sure that a new account can be created with valid data

This test case will check to make sure that a new account can be created with all fields filled out properly and with data that has not been used for a previous account. This will allow us to check to make sure that the database is connected properly, that the database stores the new information properly, and that the website is functioning properly.

### 6.1.1.2 Test Case 2 – Check to make sure that a new account cannot be created if data is invalid or incomplete.

This test case will check to make sure that a new account is not created if any of the data entered has been used for a previous account. This will allow us to check to make sure that the database is connected properly, that the database checks the information properly, and that the website is functioning properly.

### 6.1.1.3 Test Case 3 – Check customer login with valid data.

This test case will check to make sure that an account is valid and allow entry if valid. This will allow us to check to make sure that the database can be queried and is connected properly. It will also allow us to know if the website is functioning properly.

### 6.1.1.4 Test Case 4 – Check customer login with invalid data.

This test case will check to deny a user entry if the login information in the database is incorrect. This will allow us to check to make sure that the database can be queried and is connected properly. It will also allow us to know if the website is functioning properly.

### 6.1.1.5 Test Case 5 – Check that client can make trade with valid data.

This test case will check to make sure that a client can make a trade if all the information entered is valid. It will allow us to check to make sure that the financial API is working for current stock prices. It will also allow us to make sure that the database is connected and storing trade information and calculating balances.

### 6.1.1.6 Test Case 6 – Check that client cannot make a trade if data is invalid or incomplete.

This test case will check to make sure that a client cannot make a trade if any information in the fields are incorrect or invalid due to low balance, improper trading techniques or etc. This will allow us to test to make sure that the financial API is working for current stock prices. It will also allow us to make sure that the database is connected and can be queried for current balances etc.

### 6.1.1.7 Test Case 7 – Check that client can update profile information.

This test case will check to make sure that a client can make an update to their profile like email, password etc. This will allow us to test to make sure that the database is connected and can update and overwrite with new information and store it properly.

### 6.1.1.8 Test Case 8 – Check that client can access video tutorials and training on the educational page.

This test case will check to make sure that a client can access the video tutorials via a link or an embedded video. This will allow us to test for valid links and that the webpage is working properly.

## 6.1.2 Test Coverage

All the test cases above fully cover what we are planning to implement in the first demo for our project. It will test all functionality that is required to make the website work. We are looking for functionality and not necessarily beauty in the first demo.

## 6.1.3 Integration Testing Strategy

Because there are so many different people writing code, we are planning to complete the coding for the first demo by March 25th so that we can successfully run the test cases to ensure that integration is working. We have one team member writing the SQL database, one writing the integration using PHP with the Database, and two team members writing the front end. Once these are coded out, we will integrate them in one of our weekly meetings and use the test cases to confirm that it works successfully. The test cases are inclusive of all integrations and should help with integration testing.

# 7. Plan of Work

Below, Figure 7.1 and 7.2 display the Gantt Chart for the breakdown of the work. After this report, we will continue coding the requirements that we have set for ourselves for demo 1 and write report 2. Once these are completed, we will move on to coding for demo 2 and completing report 3 and the project essay. We will continue to have project meetings every Tuesday at 8:30pm EST to stay on track.

Figure 7.1

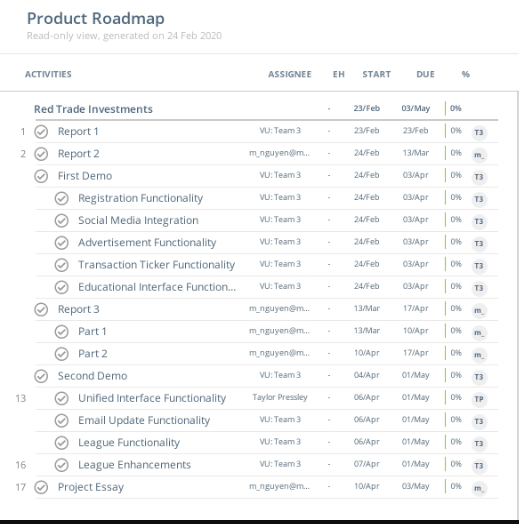
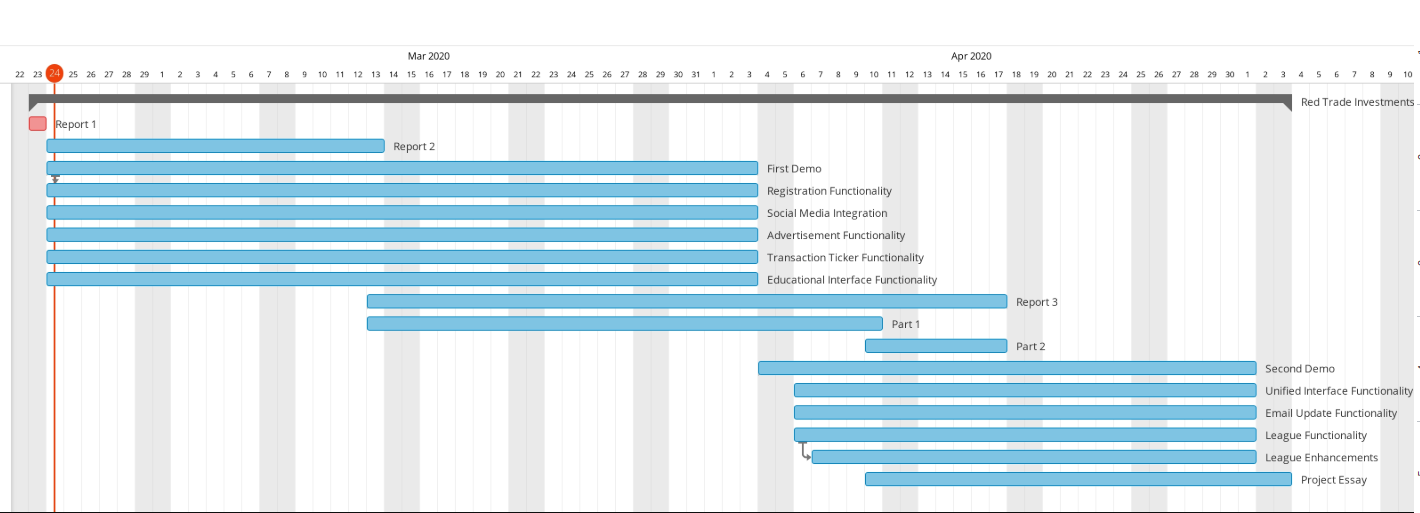


Figure 7.2



This project consists of multiple core components to make the application work. As a team, we are all responsible to make sure that the Project #5: Stock Market Investment Fantasy League game works for the end users. We have manual and automated testing to make sure the software works flawlessly.

Registration Functionality

Front End: Jayesh Nair

Back End: Hyunsook Shin

Documentation: Michael Nguyen

Social Media Integration Functionality

Front End: Gideon Lal/Taylor Pressley

Back End: Gideon Lal

Documentation: Michael Nguyen

Transaction Ticker Functionality

Front End: Taylor Pressley

Documentation: Michael Nguyen

Unified Interface Functionality

Front End: Taylor Pressley

Documentation: Michael Nguyen

Portfolio Management Functionality

Front End: Jayesh Nair

Back End: Hyunsook Shin/Gideon Lal

Documentation: Michael Nguyen

Email Update Functionality

Front End: Jayesh Nair

Back End: Gideon Lal

Documentation: Michael Nguyen

Educational Interface Functionality

Front End: Taylor Pressley

Documentation: Michael Nguyen

League Functionality

Front End: Jayesh Nair

Back End: Hyunsook Shin/Gideon Lal

Documentation: Michael Nguyen

Advertisement Functionality

Front End: Jayesh Nair

Documentation: Michael Nguyen

# 8. Project Management

## 8.1 Project Management

As this project has evolved, our contributions and interactions are more complex and intertwined. It would be impossible for us to break down the work and contributions that everyone did. It is for these reasons that we feel as a team it is safe to say that we equally contributed to the project and therefore request that equal contributions be applied for all members of this project. We have weekly meetings where we discuss the overall projects, edit the report documentation, and divide work. Additionally, we all share a Google drive account and GitHub repository where we all always make edits to the documents and scripting and changes are hard to track.

# 9. References

<https://www.investopedia.com/terms/n/nasdaq.asp>

<https://smartasset.com/investing/what-is-an-investment-portfolio>

<https://www.investopedia.com/terms/s/stockmarket.asp>

<https://betoclock.com/fantasy-sport-league/>

<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/>

<https://medium.com/@olegchursin/a-brief-introduction-to-domain-modeling-862a30b38353>

<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/>

<https://www.tutorialspoint.com/estimation_techniques/estimation_techniques_use_case_points.htm>