The Paramount Investments League

Report 2
Software Engineering
14:332:452

Team 1:

David Patrzeba
Eric Jacob
Evan Arbeitman
Christopher Mancuso
David Karivalis
Jesse Ziegler

March 9, 2014

Hyperlinks:

Webapp Link
Project Repository
Reports Repository

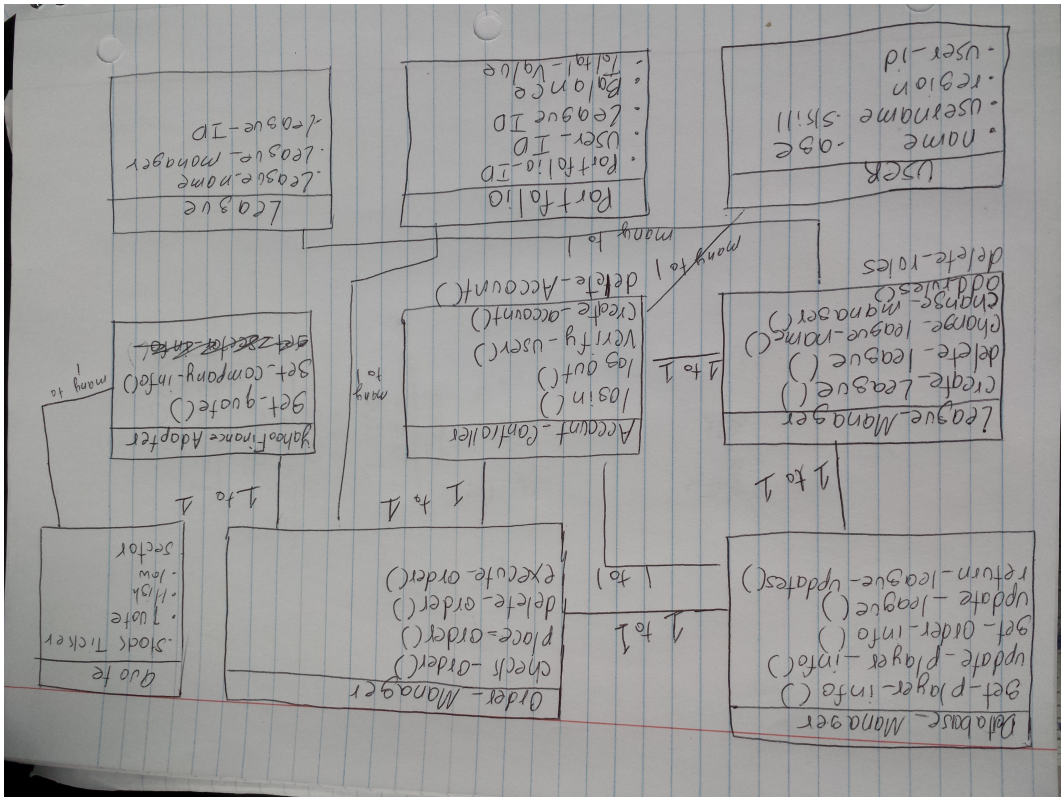
Revision History:

Version No.	Date of Revision
v.2.1	3/2/2014
v.2.2	3/9/2014

Contents

1 Class Diagrams and Interface Specifications

1.1 Class Diagram



1.2 Class Data Types and Operation Signatures

Database Manager

Attributes

Our database manager performs the function of managing the database. This can mean anything from adding user information into the database, retrieving information from the database and updating information in the database, regardless of whether the information deals with users/accounts, leagues, orders.

Methods

+ get_player_info(in user_id : int) : class User

This method is used when information needs to be retrieved for a specific player

+ update_player_info(in user_id : int, in upd: class user) : bool

This method is used to update a players information, whether it be administrative or game related

+ get_order_info(in transaction_id : int) : class transaction

This method takes in a transaction_id and returns the information associated with that specific transaction

+ update_league(in league_id : int, in leagueInfo : class league) : bool

This method is used when a league needs to be updated with the newest information provided in the league in the input

+ return_league_updates(in league_id : int) : class league

This method returns the latest updates in the league

Order Manager

Attributes

Our order manager class is responsible for handling all the tasks related to orders/transactions. It is responsible for placing the order in the system and for moving old orders to the archive_transactions table.

Methods

+ Check_order(in symbols: class Order) : bool

This method is simple used to check and make sure that the input order can be processed. It will check the users balance, etc.

+ place_order(in symbols: class Order) : bool

As the name suggests, this method is used to place an order in the system, with the information given by the input Order class

+ delete_order(in symbols: transaction_id): bool
As the name suggests, this method deletes an order from the system, assuming that it hasnt already been processed. If it has then this function will return a false value.

+ Execute_order(in transaction_id : int) : bool
This method is responsible for actually getting the stock information from Yahoo Finance API and then changing the account/portfolios to reflect it accordingly.

League Manager

Attributes This class is responsible for managing all the leagues in the system. It has the authority to create leagues, delete leagues, and modify leagues as it is instruction to do so.

Methods

+ Create_league () : Class league
This function is used to create a league from scratch so that the user can create a league.

+ Delete_leagues(in league_id : int) : bool
As the name suggests, this method will delete the league matching the input

+ change_league_name(in league_id : int) : bool
This method is here solely for the purpose implied by its name. Its only function is to change the name of the league.

+ Change_league_manager (in league_id : int, in usr : class User) : bool
This function replaces the current league manager stored in the input league with the user specified in the input.

+ add_rules(in league_id : int) : bool
This function is here for the reason its name suggests. It is here just to add rules to a given league.

+ Delete_rules(in league_id : int) : bool
This method exists just to delete the rules in a league.

Account_Controller

Attributes
This class exists to take care of any function that relates to accounts. This can mean creating an account, modifying an account, or even deleting an account.

Methods

+ Login(in user_id : int) : bool
This function is used by the user to log into the system
+ logout(in user_id : int) : bool
This function is the opposite to the one above it, it is used by the User to log out of the system.

+ Verify_User(in User_id : int) : bool
Method to make sure that the person logging in or that the person who is logged in is not an imposter/fake.

+ Create_account() : class User
Creates an account with the current user

+ delete_account(in suser_id : int) : bool
Used to delete an account from the database/ system.

Yahoo Finanace Adapter

Attributes
This class is responsible for obtaining market data from Yahoo Finance API. It consists of 3 functions to get quotes, get company information, and to get sector information.
Methods

+ Get_quote(in stock_ticker_id : string) : class quote
As the name suggests, this method is responsible for obtaining quote information about a given stock_ticker

+ get_company_info(in stock_ticker_id : string) : class Company
This method is responsible for getting market information about a specified company.

2 System Architecture and System Design

2.1 Architectural Styles

In order to make the most efficient use of our software, we will couple several known software tools and principles into our design. The follow architecture types will be expanded in detail to not only reflect general functionality, but also to reflect functionality of the software as a whole. As explained, each will play a crucial role in the success of our software and will be largely derived from the necessities of the software. That being said, architectural systems will include (and may be expanded upon in the future) the Model View Controller, Data-Centric Design, Client-Server access, and RESTful design, with each architecture serving a small part of the whole result.

Model-View-Controller

The Model View Controller is a User Interface implementation method which will separate the software into 3 specific groups; that is: the model, view, and controller subsections. The view category is typically limited to UI specific output, i.e. a webpage with stock information. That being said, the model remains the core component of the MVC method which holds all of the data, functions, and tools. The controller simply takes the input and converts it into a command for either the model or the view.

The MVC method is ideal for this particular software because it allows the design to be broken down into smaller sub-problems. By splitting into 3 parts, we can separate UI functions, from database functions, and have all of them handled ultimately by the controller. Thus in terms of fluidity of the design, adding in the MVC allows each to be distinct and allows for the programming to be made far easier.

Data-Centric Design

Data is the fundamental backbone of Paramount investments. Stored within our database, will be numerous bouts of data, which will be necessary for all aspects of the software. The database needs to contain not only data pulled from the Yahoo! Finance API, but more importantly user specific data. Whenever the user logs in, they need to have access to a personal host of their own data. That includes but is not limited to complete portfolio, leagues, achievements, leaderboard, and settings. More importantly, the data needs to be stored in a way that it can be accessed by multiple subsystems whenever necessary. So in using this method, we can keep the data specific parts in the software abstract and easily accessible.

Client-Server Access

The user will be constantly interacting with the interface. All of the interactions are occurring, thus, on a client server basis. The user remains the primary client, and as such, constantly must interact with the other subsystems. All of the infrastructure provided by Paramount Investments will need to be accessed by the user. This ensures a smooth communication between each of the parts of the MVC and between client and infrastructure. Further, the infrastructure provided by Paramount investments will be able to access infrastructure of non-associative systems.

Representational State Transfer

As a software implementing a client server Access system, a REST system is also inherently implied. The RESTful design principles state that in addition to having a Client-Server Access system, the system has a scalability of components, that the interface is uniform, stateless, and cacheable. Using this method will employ a smooth, modular set of code. Using the interface specifications within the RESTful outline allows both the user and the designers to have streamline interactions with the interface. That is the user knows quite clearly what he or she is doing when say a link is clicked on a web page. The request is converted and sent out to the controller.

Importantly, the RESTful implementation can be implemented on multiple levels. And as is desired, this system will be able to work on Android and iOS as well as through standard web interfaces. Thus a smooth transition between these mediums is incredibly important. Thus whether a user places an order on his cell phone or online, he should be able to experience a uniform experience across all mediums. Using the RESTful system will help in this process.

2.2 Identifying Subsystems

Paramount investments aims to set its platform on multiple interfaces. As such, subsystem identification becomes an integral part of initial analysis of the software. On a thick layer, our platform exists with a front-end system and a back-end system. But on a much deeper level, we can see that, each of these subsystems can be broken down into still greater detail. Front end systems typically involve user interface, and object interactions with the user. Back-end will refer to all database schema, implementation and interactions with relevant hardware. Also included are non-associative items which are necessary to the success of our system.

Front-end systems are formally plain. The user interface which displays views and specific data to the user on multiple platform is included here. That is, it will contain different mappings and specific implementations for iOS and Android as well as natively for the Web. The front-end system will have to maintain constant communication with the back-end system to maintain consistency and retrieve data regularly. It needs to be able to successfully communicate information from commands given by the user and communicate them to the back end. The back end system will retrieve necessary data and information and return the data to the UI and user to project the page or information requested.

Our back end system will be broken down further and is easily considered the most important part of our infrastructure. Since we are using the MVC framework, the back end system is to be broken down into controller and database subsystems. Additionally, we will have the financial

retrieval system and queuing systems as previously outline. Thus, the bulk of the command processing is handled by our back-end subsystem. The back-end system must not only communicate among the subsystems within itself, but it must also communicate with the front-end UI system to respond to commands and also communicate with the non-associate systems as well.

Breaking down the subsystem further, we highlight the importance of the financial retrieval system, and the queue system. The financial retrieval system will communicate with Yahoo! Finance to retrieve relevant information as requested by the controller (whenever the controller receives an input from the front-end user). The queuing system will handle other processes and largely communication with non-associative systems. It will also be involved in queuing and handling all back-end processes and monitors to ensure that the correct commands are processed at the correct time. The success of these modules, the success of the entire back-end system, and the success of communication amongst the systems will be crucial for the overall success of the software.

2.3 Mapping Hardware to Subsystems

The Paramount Investments League is contained on a MySQL database server, which is stored on one machine. However, the system as a whole is spread across several machines. The system to be is divided into two separate sections: a front-end side that is run on the clients web browser of choice, and a back-end that runs on the server side of the database. The front-end is the main graphical user interface (GUI) between the system and the client. The front-end is responsible for communication between the GUI and the database for purposes such as confirming market orders and updating an investors portfolio. These changes in the front-end are reflected in the back-end side of the server. The back-end will handle proper execution of market orders and will updates users on each of their transactions.

2.4 Persistent Data Storage

The plan for data storage exists at the core of Paramount Investments. Since so much of our software depends on properly developed and updated data, it is of the utmost important that our database schema represent accurately all objects involved. That is, the data must accurately (at all times) reflect all relevant user data, stock information, ticker variables, league settings, achievements, leaderboards, and all other relevant objects.

Paramount Investments will make heavy use of the relational database MySQL. Relational databases are far more practical for the needs of this particular software. That is, relational databases consist of several indexed tables filled with various object attributes. As can be viewed in the class diagrams on the previous page, this is necessary for the large quantity of objects which will be present in the software. Tables will need to exist not only for user data and settings such as log in and league profiles, but also for stock and portfolio information. Further, these databases need to be constantly written and rewritten to ensure constantly updated and accurate information. Items such as leaderboards, and information which will be able to be viewed on each users portfolio need to constantly reflect accurate data.

The data will be retrieved from the respective database table in the form of a query. When a user inputs a command to retrieve data, a query must be placed, the table searched, and the eventual correct data value (or values) returned. For example if a user requests his or her settings,

it can query currently selected settings and return those values to the UI and to the user. If the user elects to make a change this will be sent back to the database, updated and saved for further access later. The same process can be mirrored and applied to all facets of the software. Several tables will be used for varying data as has been outlined in the diagrams above. The success of the software is dependent on the values being returned accurately and in the most updated form at all times. Because of that, the database must receive a regular feed from the Yahoo! Finance API in order to constantly update and reflect data when queries are placed. In doing so, users will have constantly accurate views of their portfolio performance, leaderboards, achievements, stock tickers, and recent trades going on throughout the league and entire user base. It is in this way that the Paramount Investment software will distinguish itself from others and retain functionality and efficient realization of its ultimate goals and requirements.

2.5 Network Protocol

As is standard for software of this type, Paramount Investments will uses the standard Hypertext Transfer Protocal (HTTP). HTTP acts by structuring text which is uses hyperlinks to communicate messages through text between nodes. While not necessarily unique or particular to our situation, it is still important to note that this will be the primary protocol between user and software interface. More importantly, the HTTP protocol will be used not only on web-based devices but also on Android and iOS devices as well. From any of these mediums, the users can access various webpages and links from the Paramount Investment website. They will be able to access, through this protocol, all relevant stock, portfolio, and relevant information through these pages and by using the HTTP protocol.

2.6 Global Control Flow

Execution Order

In general, the implementation of the system at Paramount Investments is for the most part, event-driven. All the features that the system has to offer must be triggered by some entity, whether it be the user themselves or some other part of the system. Overall, most of the event-driven characteristic comes from the user end of the system. Many of the functionalities (stock trading, portfolio viewing, league joining, etc..) can only be triggered by the user. There are however, some event-driven functionality that are initiated by the system. When the user places an order, it is processed and added into the database. From here, the system initiates the process of checking the order and then it uses Yahoo Finance API to retrieve market information about the stock, obtain a quote and then actually process the order.

There is some functionality that have to be executed in a defined order. Before the user starts investing, they must a few steps:

- Registration/creating an account: any user must register within our website before joining a league.
- Join a league: any user must first join a league before they can start investing.
- Achievements: any user must first complete the required criteria before they can be awarded with the achievement trophy.

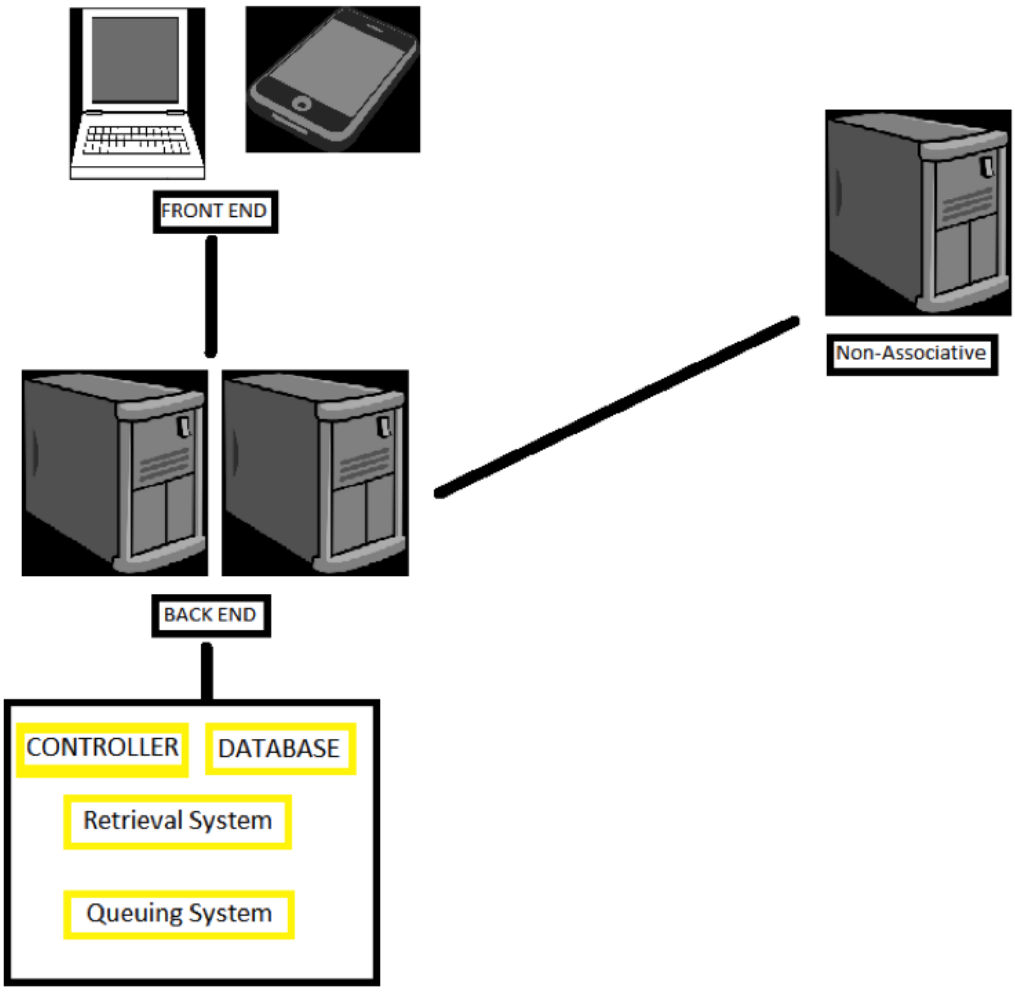


Figure 2.1: Diagram of the network protocol.

However, on the whole, our system is still definitively an event-driven one.

Time Dependency

In general, the system at Paramount Investments is very much a real-time system, but there are features that do not depend on time. The real-time system is very reliant on the stock market, which itself has certain times of operation. As the user is browsing the website, there are real-time timers that help the system process information that it is receiving.

- Achievement Timer: This timer is used at the end of the day to check for achievement specs for all the users. Achievements/ rewards will then be dished out accordingly.
- Stock Market open and close: The stock market has a time interval between when its open and when it is closed.
- Queuing system: the orders placed by users are placed into a queue. Depending on market conditions, this can place high loads on the server. To balance the server load, we must split the orders effectively. The timer in this system helps to check for unexecuted/ outstanding orders and then processes them.

Concurrency

There are sub-systems in our main system which have to be carefully thought of due to concurrency. The biggest of these is the queuing subsystem. This produces a concurrency issue because we have to make sure that no more than 1 order is being inserted into the queue at any given time. Likewise, we also have to make sure that no more than 1 order is being dequeued from a given queue. Other than this our system really does not need any synchronization. However, this may change as we are implementing our system.

2.7 Hardware Requirements

The hardware requirements on the server side are the main contribution to the operation of Paramount Investments League, leaving the client-side with minimal requirements. In fact, the only requirement of a client will that it runs a browser that is capable of running a modern web browswer.

Internet Connection

In order for Paramount Investments League to use any of its core functions (trading stocks, updating user portfolio, tracking administrative actions, etc.), an internet connection is required. Since most of the data being transferred is text (executable instructions), a low band of frequency is required. Note that a complete scalable analysis has not been performed on the system, so a low band of frequency is based off of the needs of the current website. For ideal performance, higher bandwidths of frequency should be used in order to reduce any overhead. A network connection between the server and the Yahoo Finance API is necessary during trade hours (9:30am - 4:00pm Monday through Friday), otherwise, no investors can perform a transaction.

Disk Space

The server must have adequate hard drive space to be able to store all of the database information. All data being stored is the sum of all program instructions for the system. 10 GB of storage space should be sufficient for the system.

System Memory

Since this system is in active development, there is limited concrete evidence that supports the overall performance of the system. The system will load copies of database stored information in order to operate over it. For better throughput, the memory should be managed using a Least

Recently Used scheme (LRU) in order to keep the system memory populated with useful information. A LRU scheme will release any bits of memory that havent been accessed in a long time, and it will replace it with information that is used more often. Also, any operations used on loaded information will also use up system memory. A minimum of 512 MB should be used for testing our system. In addition, as our user based expands, it is obvious that the system memory will also have to grow with it.

Client-side Hardware Requirements

The core hardware requirement on the client-side of the system will be an internet connection. This is essential for the client to be able to remotely connect to the server in order to access the database. Without an internet connection, no client will be able to use a web browser to visit the Paramount Investments League website. In addition to an internet connection, and for a friendly user experience, anyone on the client-side should have a functional mouse and keyboard, as well as a graphic display to see their portfolio. To display the Paramount Investments League website, a screen with a minimum resolution of 800x600 pixels is adequate.

3 Plan of Work

3.1 Development and Report Milestones

Illustrated on the next page is a chart reflecting our goals relative to the project dead-lines. It incorporates both core development and report items. For our initial stages we focus on environment and platform set-up (eg: deploying a development webserver) and the initial, core code implementation. At the same time we will finalize the details of our final product via the report milestones.

Development milestones have been spread out following the completion of Report 1 on 23 February 2014. It begins with deploying our development environment and server through Digital Ocean[1]. We concurrently will roll out developer images, the Play Framework[2], and develop database schema. Implementing user registration/login will follow shortly along with deploying a solution to use the Yahoo! Finance API. The development milestone finishes up with the implementation of user portfolios along with basic market operations and basic achievements.

Report milestones are also set concurrently. As we begin to initialize our development environment, we will also build on top of and expand on previous reports to expand upon and fully realize the details of *P*aramount Investment League.

Core goals leading up to Demo 1 include establishing all core functionality for *P*aramount Investment League. This includes the following:

- **Play Framework deployment :** This includes basic site navigation, user login/registration, and Twitter Bootstrap deployment.
- **Setting a foundation for the database:** Schema should be built to be extensible to support future enhancements.
- **Implement the Yahoo! Finance API**
- **A functional user interface:** The user interface should function across multiple platforms with a focus on experience and expectations.

3.2 Breakdown of Responsibilities Introduciton

Contributions leading up to the completion of this report are covered in the "Contributions" in Chapter 7. For the future division of labor, we all plan on subdividing aspects of both the next reports as well as the development of the *P*aramount Investment League Demo 1.

3.3 Breakdown of Responsibilities

Core server deployment will be the repsonsibility of David Patrzeba. Eric Jacob will be responsible for the database rollout. David Patrzeba will also be responsible for the core software rollout on the server including git, Play Framework, nginx, and other core libraries and software. David Karivalis will be repsonsible for integrating Twitter Bootstrap into Play Framework.

Routing will be headed by Eric Jacob and assisted by Chris Mancuso and Evan Arbeitman.

User Interface will be done by David Karivalis and Jesse Ziegler and they will integrate the REST API[3] to facilitate dynamic views.

The rest of the development workload will be divied up based around the Model, View, Controller design pattern. David Patrzeba and Eric Jacob will focus on the controllers, David Karivalis and Jesse Ziegler will focus on the Views, and Evan Arbeitman and Chris Mancuso will focus on models. David P., David K., and Eric will be made available for technical advising.

David Patrzeba will be responsible for formatting the report. David Karivalis will be responsible for digitization of paper diagramming for all reports. Report duties will be divied up based on percieved strengths of the team and availability.

Overall project success will be decided with how well the MVC[4] component teams communicate and work with each other, as *P*aramount Investment League will rely on the interactivity between the Model, Views, and Controller portions of the architecture.

3.4 Projected Milestones

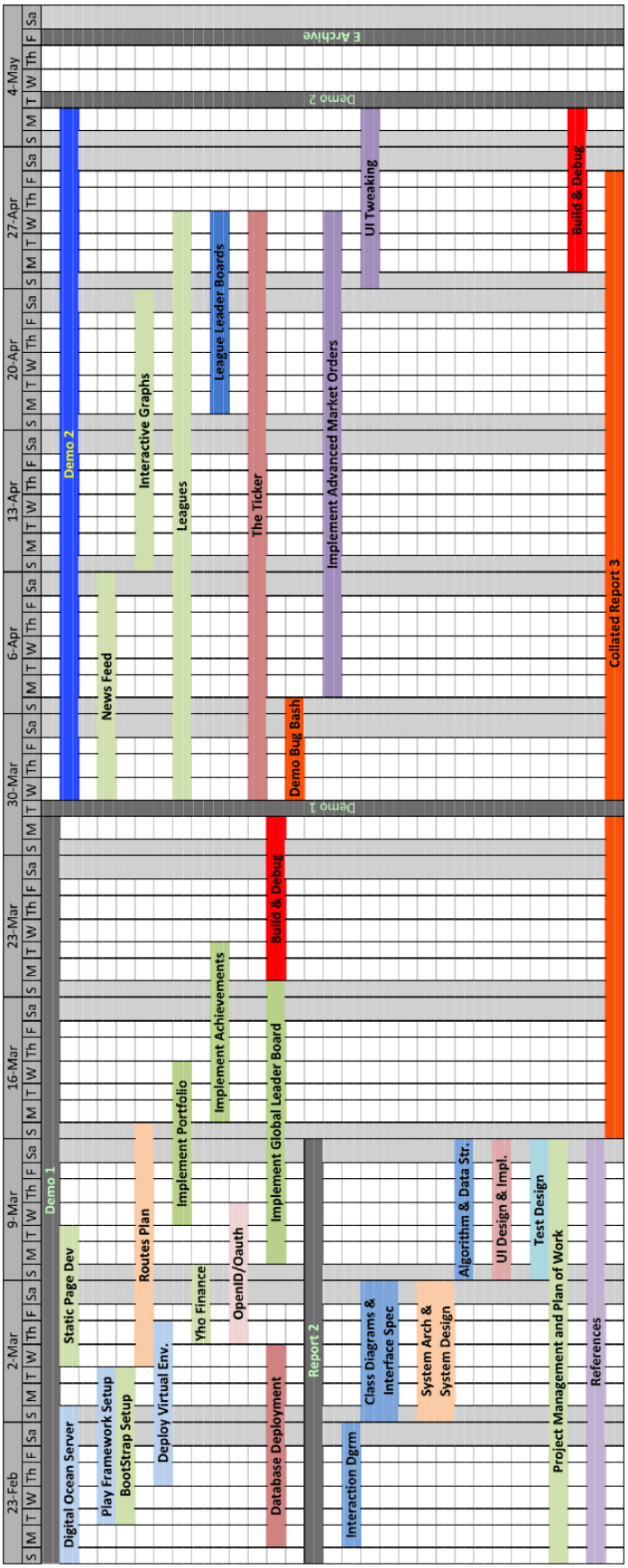


Figure 3.1: This chart is the roadmap to meeting all our milestones.

References

[1] Wikipedia, "Digital ocean." http://en.wikipedia.org/wiki/DigitalOcean. [Online; accessed 23 February 2014].

[2] Wikipedia, "Play framework." http://en.wikipedia.org/wiki/Play_Framework. [Online; accessed 23 February 2014].

[3] Wikipedia, "Representational state transfer - Wikipedia, the free encyclopedia." http://en.wikipedia.org/wiki/Representational_state_transfer. [Online; accessed 23 February 2013].

[4] Wikipedia, "Model-view-controller - Wikipedia, the free encyclopedia." http://en.wikipedia.org/wiki/Model-view-controller. [Online; accessed 23 February 2014].