Final Report

Software Engineering

14:332:452

Team Members:

Keeyan Haghshenas

Umer Qazi

Joe Gordon

Dunbar Birnie

Ludwig Randazzo

Colter Mooney

Bohdan Kryzh

Ashish Motyala

**Table of Contents:**

# 1 Customer Problem Statement

# 1.1: Problem Statement……………………………………………….……………….…3

# 2 Glossary of Terms……………………………………………………......................................5

# 3 System Requirements

# 3.1: Functional Requirements……………………………….……….…………………..7

# 3.2: Non Functional Requirements………………...……………………………….......9

# 3.3: On-Screen Appearance Requirements…………………………………………….10

**4 Functional Requirements Specification**

**4.1: Stakeholders………………………………………………………………………....12**

**4.2: Actors and Goals……………………………………………………………………12**

**4.3: Use Cases…………………………………………………………...........................13**

**4.3.1: Casual Description………………………………………………………..13**

**4.3.2: Use Case Diagram………………………………………………………..14  
 4.3.3: Traceability Matrix.……………………………………………………….15**

**4.3.4: Fully-Dressed Description.………………………………………………15**

**4.4: System Sequence Diagrams………………………………………………………..18**

**5 Effort Estimation using Use Case Points……………………….……….………………….21**

**6 Domain Analysis**

**6.1: Domain Model.…………………………………………………………………......23**

**6.1.1: Concept Definitions.……………………………………………………...23**

**6.1.2: Association Definitions.…………………………………………………..26**

**6.1.3: Attribute Definitions……………………………………………………...27**

**6.1.4: Traceability Matrix.……………………………………………………….28**

**6.2: System Operation Contracts.…………………...…………………….……………28**

**7: Interaction Diagrams**

**7.1: Introduction……………………………………………………………………….. 30**

**7.2: Diagrams……………………… …………………………………………………...30**

**8: Class Diagram and Interface Specification**

**8.1: Class Diagram……………………………………………………………………....33**

**8.2: Data Types and Operation Signatures…………………………………………….34**

**8.3: Traceability Matrix………………………………….……………………………...38**

**8.4: User Interface Specification…………………….………………………………….38**

**9: System Architecture and System Design**

**9.1: Architectural Styles…………………………………………………………………42**

**9.1.1: Model/View/Controller(MVC)…………………………………………..42**

**9.1.2: Entity Framework………………………………………………………...42**

**9.1.3 Front-End and Back-End Specific Architecture…………………………42**

**9.1.4 Event Driven Architecture………………………………….…………......43**

**9.2: Identifying Subsystems…………………………………………………………….43**

**9.3: Mapping Subsystems to Hardware………………………………….…………….43**

**9.4: Persistent Data Storage………………………………….…………………………44**

**9.5: Network Protocol………………………………….………………………………..45**

**9.6: Global Control Flow………………………………….…………………………….46**

**9.6.1: Execution Orderliness………………………………... …………………46**

**9.6.2: Time Dependency………………………………….…………………….46**

**9.6.3: Concurrency………………………………….…………………………...46**

**9.7: Hardware Requirements………………………………….………………………..46**

**10: Algorithms and Data Structures**

**10.1: Algorithms………………………………….………………………………….………………..47**

**10.2: Data Structures………………………………….………………………………....48**

**11: User Interface Design and Implementation**

**11.1: Updated Pages………………………………….………………………………….49**

**11.2: Efficiency of the Views…………………………………………………………....49**

**12: Design of Tests**

**12.1: Test Cases………………………………………………………………………......51**

**12.2: Test Coverage……………………………………………………………………...56**

**12.3: Integration Testing Strategy………………………………………………….…..56**

**13: History of Work, Current Status and Future Work………………………………...……...57**

**13.1: Current Status………………………………………………………………...........57**

**13.2: Key Accomplishments…………………………………………………………….64**

**13.3: Future Work………………………………………………………………………..65**

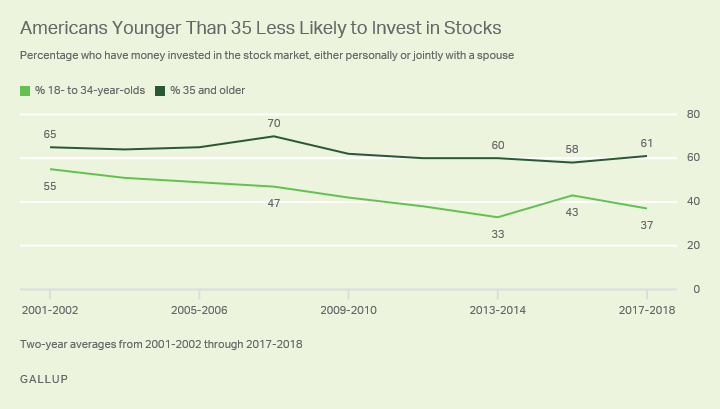
**14: References…………………………………………………………………………………...65**

**15: Responsibility Matrix……………………………………………………………………… .66**

**1: Customer Problem Statement:**

1.1: Problem Statement:

Investopedia defines the stock market as a collection of markets and exchanges where the regular activities of buying, selling and issuance of shares of publicly held companies take place. (<https://www.investopedia.com/terms/s/stockmarket.asp>). With exchanges like the New York Stock Exchange, the Bombay Stock Exchange, and Nasdaq, the stock market is an ever growing and necessary portion of the global economy. Due to its size, stature, and volatility the stock market has developed a reputation as an increasingly complex and convoluted entity. Though with more and more companies arising each day it is clear that an understanding of the markets is becoming increasingly more essential for people who want to not only live well but also prosper.

According to a recent Gallup Poll, people under thirty five are investing in stocks at the lowest rate in decades (*fig. 1*). This dramatic decline in young investors is partially due to fear of market volatility but even more so due to the lack of youth exposure in early schooling to the stock market and the opportunities available. Stock trading, if performed right, is a great way to invest in ones future. The trouble is that most people don't recognize the importance of stock investments and never learn how to get involved in the industry. As shown below, the percentage of people investing at ages 18-34 has dropped from 55% to 37% in the last 17 years. This drop in investing is due to the fact that many young people are beginning their job hunt with little to no money. Student loans and various debts are hindering millennials from investing in their future. 

That is why we the consumers are interested in working with Ducklab Developments. We believe their product Ducklab Investments seeks to remedy this issue.

Ducklab Investments creates a fun and educational way for all people to experiment with stock trading and investing with zero real risk. This zero risk platform will remove the main barriers of entry and allow anyone to begin investing regardless of their monetary status. As well as provide a educational and enticing method of introduction to the stock market.

Ducklab investments provides not only educational and exciting environments to learn and complete but also an easy method of sign up and sharing with friends. Their social media integration as well their security of user information guarantees that all playing will be risk free and enjoyed in tandem with friends. Though it does not stop there, because the games are based around factual stock data, due to the implementation of the IEX API, Ducklab Investments push notification system allows us as user to stay up to date with everything that is happening in leagues that we are participating in. With a required tutorial for each new user, no one feels as if they are lost in the game despite the complexity of the information.

The primary feature of Ducklab Investments is the competitive nature of the game. We as humans love to compete and be the best at what we do and we believe Ducklab Investments leaderboard and achievement features provide that whole heartedly. They are able to not only produce a fun and competitive environment similar to most games of gaming consoles but also educate the user as well. With rewards for top competitors and an achievement motivated ladder system the game provides incentive to continue playing and learning adhering to the users competitive nature. These rewards will also help decorate each players profolio, as well as advance in skill trees leading to play in more competitive pools. Though as users we are also always given the option to play with friends.

Access of the information is also integral for better play and Ducklab Investments provides numerous methods in which players can stay up to date for better completing. With push notification on both mobile and desktop devices all leagues that a player participates in will not be neglected. Finally, Ducklab Investments provides two hardware aspects for us the consumers to be up to date with each league. The first of which is a desktop ticker that provides actual data from each league and real time stock data, to ensure that us the user are up to date with all play. As well as providing Amazon Alexa integration, ease of controls and informative play are key aspects of Ducklab Investments.

The experience that Ducklab Investments provide is not limited to solely desktop competitors. There is a unified infrastructure amongst all, mobile, tablet, and desktop users to aid in the access of the Ducklab Investment Experience. In reality all a user needs to access the wonders of Ducklab Investments is simply an internet connection.

**2: Glossary of Terms**

The following terms provide an overview of some of the terminologies which are important for fully understanding the purpose of this software's design.   
   
**Achievement**: An achievement is defined as a goal which is reached by an investor. Certain Achievements will be set by the user and their progress will be tracked by the system.  
   
**Transaction Ticker:** The Transaction Ticker will constantly update and scroll through the most recent trades across the market or game. The users can observe market trends from global equities which may or may not already be in their portfolio.  
   
**Leaderboard:** League based ranking system determined by overall net worth of player.  
   
**Security:** A security is a certain type of tradable asset. A security can include debts, equities, or derivatives. For out game, we will be dealing with equities.  
   
**Dividend:** A dividend is a payment made by a corporation to its shareholders. Dividends are generally paid as a distribution of profit. It is usually distributed as fixed percent of shareholder value.  
   
**Exchanges:** A system or market in which commercial transactions involving currency, shares, commodities, etc., can be carried out within or between countries.

**Derivative**: A derivative is any financial contract which derives its value from another asset or index.  
   
**Millennials:** A person reaching young adulthood in the early 21st century.

**Option:** The game will give the user option to buy or sell an asset at a special price on or before a given date. The buyer and seller are both obligated to fulfill the transaction on the given date if the option is taken.  
   
**Future**: A future allows the user to buy an asset at its current price and pay for it at that price in the future. The buyer and seller are both obligated to fulfill the transaction on the given date if the future is agreed upon.  
   
**Forward**: A forward allows the user to buy an asset at its current price and pay for it at that price in the future. A forward is a private agreement between buyer and seller not necessarily based around market equity. The buyer and seller are both obligated to fulfill the transaction on the given date if the future is taken.  
   
**League**: A market simulation with a pre-determined rule set for teams and several investors with a common goal to determine a winner. Goals can vary across leagues as determined by league managers. Investors can choose to opt into a private league, public league, or no league at all.  
   
**Portfolio**: A detailed account of assets associated with a particular investor in a given league. Portfolios are unique to each user and will contain specific details such as earnings, losses, performance, averages, as well as detailed asset performances of equities within the given portfolio.  
   
**League manager**: The league manager will have the responsibility of deciding number of teams in a league, adding and/or removing investors from the league. League managers control settings, and victory conditions for a particular league. League managers maintain their manager status only for the league in which they have created.  
   
**Order**: An investor must place an order for the purchase or sale of an asset on or before a given date.  
   
**Stock**: A type of asset that represents equity in a company.  
   
**Sell Price**: The price at which a user is willing to sell a stock.

**Buy Price:**The price at which user is willing to pay for a stock.  
   
**Ticker Symbol:** An abbreviation used to uniquely identify publicly traded shares of a particular stock on a particular stock market.  
   
**Market Order:** Any order placed for immediate market transaction.  
   
**Buy**: The user has decided to purchase a particular stock and has placed a bid to buy that stock.

**Market Volatility**: a statistical measure of the dispersion of returns for a given security

**Sell**: The user has elected to sell a particular stock and has posted an ask price for it.  
  **Short**: Short selling occurs when an investor borrows a security, sells it on the open market and expects to buy it back later at a cheaper price. Short sellers may be hoping to profit from the decline in price or their goal could be to hedge their risk in a related security or asset class.  
  
**Limit**: A limit is an investment which will only take place at a given price. An investor placing a buy limit will place a maximum amount they'll pay and an investor placing a sell limit will place a minimum value for which the stock will be sold. Limit orders are not guaranteed to ever process, and only do when the particular limit is reached.  
   
**Stop**: Orders which are activated if a particular stock falls below or rises above a particular price. It is used to minimize gains and losses for the investor.  
   
**Share**: A share is a small percentage of a given company which can purchased or sold from other traders.

**3: System Requirements:**

3.1: Functional Requirements

The functional requirements written and defined below along with the priority order serves as the main requirements for the system functionality. These functional requirements will be explained in details and referenced on later in the document. Requirements mentioned below convey a general idea about different functionalities of the system.

|  |  |  |
| --- | --- | --- |
| **Identifier** | **Priority** | **Requirement** |
| Req-1 | 1 | The system shall allow users to register using their email and participate in Duck Lab investments. |
| Req-2 | 1 | The system shall be accessible across multiple devices / platforms. |
| Req-3 | 2 | The system shall allow users to recover their account in case user forget his / her password. |
| Req-4 | 1 | The system shall provide updated and real time stats about different leagues on main screen. |
| Req-5 | 1 | The system shall allow users to buy and sell stocks. Users should be able to stop trading whenever required. |
| Req-6 | 1 | The system shall allow users to join any / multiple league(s). |
| Req-7 | 3 | The system shall present user a tutorial to understand the basic knowledge of fantasy league investments |
| Req-8 | 3 | The system shall allow users to invite other people to participate in Duck Lab Investments. |
| Req-9 | 2 | The system shall allow users to see all their investments on one screen. |
| Req-10 | 1 | The system shall provide stock data over specific span of time of different leagues for its users. |
| Req-11 | 1 | The system shall allow users to create and manage their own league. |
| Req-12 | 1 | The system shall allow users to ask Amazon Alexa about their stock investments and present standings. |
| Req-13 | 1 | The system shall be available in mobile application and send periodic email updates to users.. |

The requirements catalogued above provide a general priority based list of what we intend on being the core functionality of our software application. Our goal is to create a learning environment based on games and competition. We are taking raw stock market data and using it to allow users to play long term games with real time data. We are also creating scenarios in the form of mini games to simulate trading out of a deficit and other real life situations. Our new functionality including alexa integration and real life personalized stock tickers add to user experience and further differentiate us from the competition. We look forward to also adding new functionality in the future, but right now the core is strong. Core functionality is critical to the success of this application.

3.1.1: Core requirements

These are the requirements which are needed for the system to work properly and cannot be overlooked.

Duck Lab Investments will use basic authentication system for users to register and login into their account or recover account in case any user forget his / her password. Users will have access to their portfolios and investments. Users will be able to buy / sell stocks and stop trading. ( Req-1, Req-3, Req-5)

The system will be accessible across multiple platform / devices. Duck Lab Investments will provide real time updates / stats to its users and participants will be able to join any / multiple league(s) or even create their own league. Manager of the league will be able to decide number of teams in a league and rules for its league. Users will be able to access and see all their investments on one screen. Stock history of any league would be available to the users over a specific span of time. (Req-2, Req-4, Req-6, Req-11, Req-14)

To maintain the idea of perfect competition and unbiased market prices, the most important step is acquiring accurate and real time data to simulate an authentic experience for our consumers. Our plan to acquire this data is as follows, we plan to first research all places where this information is available, there are a variety of sources in which stock market information is available, then compile a list of these sources and see which sources provide the most accurate. Once completed, each source that we deem accurate and reliable is then intended to be averaged together and combine to work in tandem with API that we plan to use. The API we plan to program in was developed by IEX and is called the IEX Development Platform. A platform entrusted by some of the largest companies involved with the stock market today including but not limited to the, Wall Street Journal, Bloomberg Company and Fast Company. This software is an educational tool and not intended for actual trading. Instead users will learn how to trade and gain knowledge about how markets work.

3.1.2: Supplemental Requirements

The system will present new users with the short tutorial (optional) to understand basics of fantasy league investments. Users will have access to share their investments and invite friends for rewards. (Req-8 , Req-9, Req-10)

For users to have best and easy environment, system will also be available in mobile application (Android) for users to access it easily and not going through the effort of opening web browser every time. Users will also be able to ask basic questions from Amazon Alexa about their stock investments and trade standings. (Req-17, Req-18)

3.2: Nonfunctional Requirements

|  |  |  |
| --- | --- | --- |
| **Identifier** | **Priority** | **Requirement** |
| Req-15 | 2 | The system shall support expected transaction traffic with the quantity TBD. |
| Req-16 | 3 | The system shall support an expected number of active users. |
| Req-17 | 4 | The system shall update stock prices in real-time |
| Req-18 | 3 | The system shall encrypt user passwords |
| Req-19 | 2 | The mobile app shall have comparable latency to the web-based application. |
| Req-20 | 1 | The system shall only allow users to create a finite number of active leagues to prevent wasting space on the server. |
| Req-21 | 3 | The system shall have <10ms trading latency |

Functionality:

Having the platform on a website allows it to be accessible from anywhere with internet service. Having the platform available as an mobile app would provide easy access for the users, and compatibility with Alexa would allow for extensive portability and interoperability. The platform will have easy integration of groups varying in sizes. With size adaptive features, the platform will have capability to handle an influx of members with no latency or user exclusion (Ex. full servers). The platform will have account creation with encrypted password protection for profile security. This will ensure users have access to their personal accounts and protection from exploits.

Usability:

A large portion of what we are working on implementing involves variation in the ways a user can interact with the application ranging from an in depth feature rich yet concise and organized web-site, to a streamlined mobile app, to mere voice commands. As mentioned in REQ-2, we are supplying many platforms of varying complexity for the users, usability is really only limited to where the user is most comfortable. In regards to usability of the program itself, as mentioned in REQ-8, a tutorial is included on the web-based application going over how to actually get started playing.

Reliability:

In the case of server failure, the problem will be remedied by the host of the platform. Incase of shutdowns, backups of user data will be utilized to ensure continuation of the servers after recovery. Transactions will require confirmation to provide transaction individuality as well as integration into the stock database. User logins will be managed based on user preference. The system will provide choice to stay logged in for easier access and portable synchronization.

Performance:

Due to the time sensitive nature of stock trading, latency could pose a problem. To mitigate this we desire a lightweight and efficient website. In the case of a modulating-fixed-price gamemode, current value would be tied to the transaction. For hardware sensitive versions such as the mobile-app, the reduction of features such as chat and tutorial is possible to yield a more streamlined performance.

Supportability:

The web-based application will be mobile-browser friendly allowing for a normal experience on smaller touch screens. With the high level of accessibility, the platform will have a management system monitoring and maintaining the servers to provide an optimal user experience. To adapt to newer versions of web services, new patches would be developed and tested in a backed up version of the platform as to mitigate system errors.

3.3 On-Screen Appearance Requirements

|  |  |
| --- | --- |
| **Identifier** | **Requirement** |
| OSR-1 | The login page will show two textboxes for the user to enter its username and password and below that, buttons to the user to sign up for DuckLab Investments. |
| OSR-2 | The user will be redirected to the login page if they have yet to login. |
| OSR-3 | An error page will be displayed whenever a user tries accessing a page that does not exists or does not have the permissions for. |
| OSR-4 | Every page except the login page has a toolbar which has shortcuts to all of the main pages of the application. |
| OSR-5 | The Portfolio screen will display information on the games the user is involved in. |
| OSR-6 | On the game-info page, a leaderboard will be shown. |
| OSR-6 | The Portfolio page will display all public information about the user, including the games the user is involved in. |
| OSR-7 | The my-account page will have inputs for all of the users private account information and app-settings. |

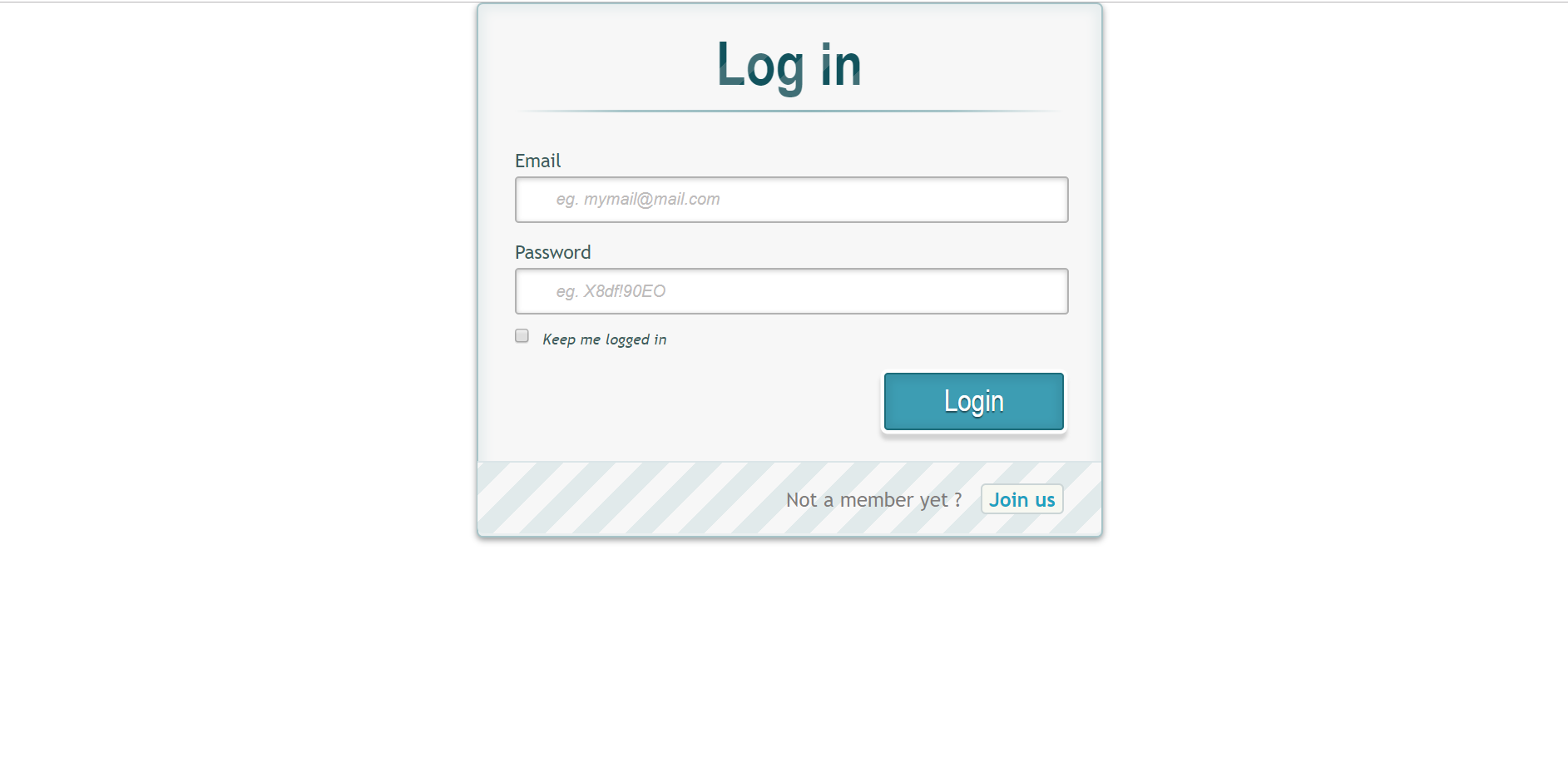


Image above provides an idea of the homepage appearance.

**4: Functional Requirements Specification**

4.1: Stakeholders:

**Internal stakeholders**

Owners: The people responsible for creating the platform

Managers: the people responsible for streamlining the web service and databases

**External stakeholders**

Users: individuals registered to the platform interested in investing in stocks

Advertisers: businesses interested in potential customers from users of the platform

4.2: Actors and Goals:

Initiating:

**Guest**

Individual who visits the website has access to the homepage

Goals: View homepage, create an account.

**User**

Individual logged into the system allowed to interact with the website

Goals: Check portfolio, create/join league, buy/sell shares, monitor stocks

**League Administrator**

User that created a league with the ability to manipulate its rules for other participants

Goals: Set league rules, add/kick users, monitor session

Participating:

**Database**

Storage for all data for the website including stock and user information

**Financial API (IEX)**

Provides real time stock prices for the database

**Email Server**

System that send users notifications over email

**Alexa**

Home system that communicates with user to provide updates

**Web Server**

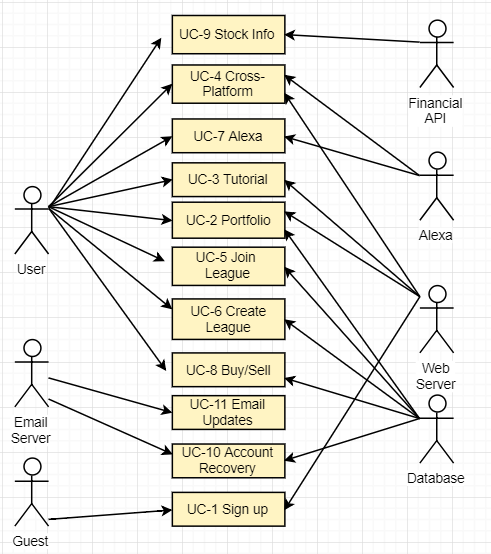
Virtual location of the website where the users interact

4.3: Use Cases:

**4.3.1: Casual Description**

|  |  |  |
| --- | --- | --- |
| **Use Case** | **Description** | **Reqs** |
| **UC-1** | A guest will be able to register an account | **Req-1** |
| **UC-2** | A user can view their own investments as well as others across multiple leagues. | **Req-3**  **Req-8**  **Req-9** |
| **UC-3** | A user can access a tutorial that will cover the mechanics of the application to learn fundamental strategies of stock trading. | **Req-7** |
| **UC-4** | A user will be able to access it across multiple platforms including a mobile application (Android). | **Req-2** |
| **UC-5** | A user will be able to join multiple leagues. | **Req-6** |
| **UC-6** | A user will be able to create and manage their own league. | **Req-11** |
| **UC-7** | A user will be able to ask Amazon Alexa about their standing stock information. | **Req-12** |
| **UC-8** | A user will be able to buy and sell stocks. | **Req-5** |
| **UC-9** | The stock API will supply stock data over specific span of time. | **Req-10** |
| **UC-10** | If a user has forgotten his/her password he/she can request to recover account. | **Req-4** |
| **UC-11** | The email server will periodically email the user the latest trends. | **Req-13** |

**4.3.2: Use Case Diagram**

****

**4.3.3: Traceability Matrix**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Req’s | PW | UC-1 | UC-2 | UC-3 | UC-4 | UC-5 | UC-6 | UC-7 | UC-8 | UC-9 | UC-10 | UC-11 |
| Req-1 | 4 | X |  |  |  |  |  |  |  |  |  |  |
| Req-2 | 4 |  |  |  | X |  |  |  |  |  |  |  |
| Req-3 | 2 |  | X |  |  |  |  |  |  |  |  |  |
| Req-4 | 4 |  |  |  |  |  |  |  |  |  | X |  |
| Req-5 | 4 |  |  |  |  |  |  |  | X |  |  |  |
| Req-6 | 4 |  |  |  |  | X |  |  |  |  |  |  |
| Req-7 | 3 |  |  | X |  |  |  |  |  |  |  |  |
| Req-8 | 3 |  | X |  |  |  |  |  |  |  |  |  |
| Req-9 | 2 |  | X |  |  |  |  |  |  |  |  |  |
| Req-10 | 4 |  |  |  |  |  |  |  |  | X |  |  |
| Req-11 | 4 |  |  |  |  |  | X |  |  |  |  |  |
| Req-12 | 4 |  |  |  |  |  |  | X |  |  |  |  |
| Req-13 | 4 |  |  |  |  |  |  |  |  |  |  | X |
| Max PW | | 4 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| Total PW | | 4 | 7 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |

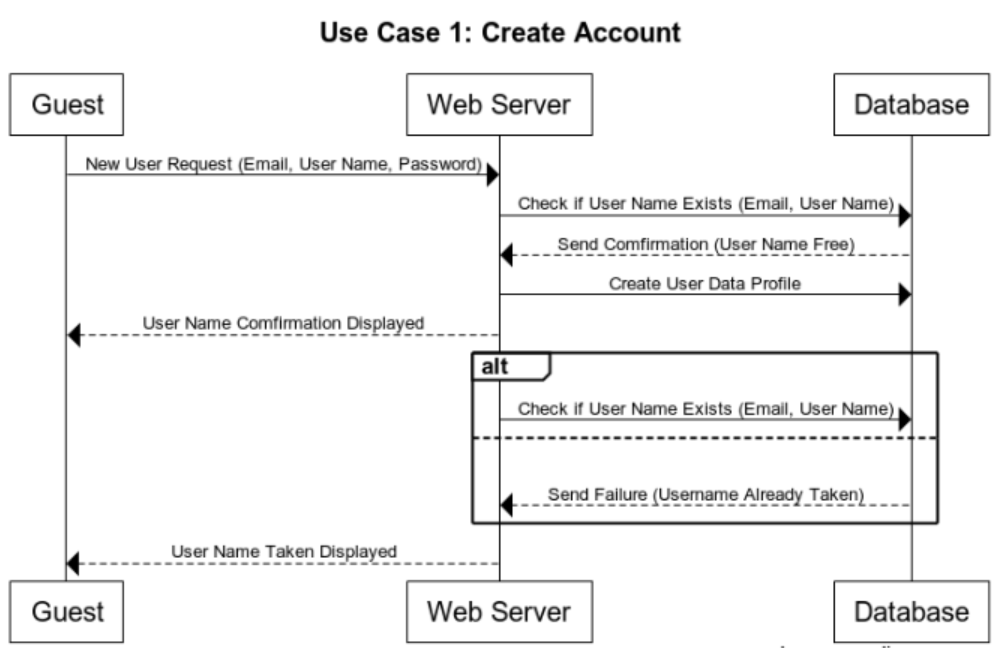
**4.3.4: Fully-Dressed Description**

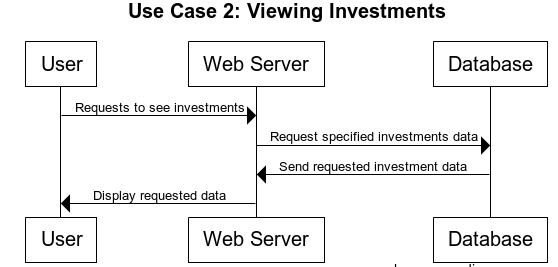
|  |  |
| --- | --- |
| **Use Case UC-2:** Viewing-Investments  **Related Requirements:** Req-3, Req-8, Req-9  **Initiating Actors:** System  **Actor’s Goal:** To display the users investments as well as others across multiple leagues.  **Participating Actors:** Database, Financial API  **Pre Conditions:**   * User has a registered account   **Post Conditions:**   * The investments will be displayed for the user. | |
|
|
|
|
|
| Flow of Events for Main Success Scenario: | |
| 1. User logs in 2. User clicks on Portfolio tab 3. Web server fetches user data and current stock prices from the Database 4. Website displays the users investments | |
| Flow of Events for Alternate Scenarios | |
| 1. User logs in 2. User enters his league 3. User clicks on another user name 4. Web server fetches user data and current stock prices from Database 5. Website displays selected users investments | |

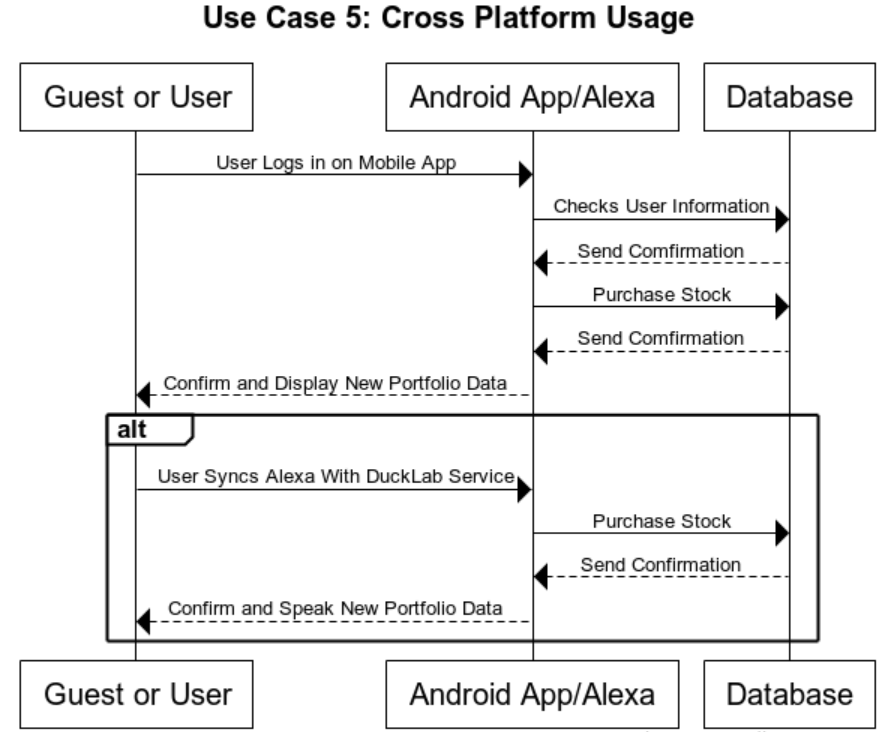
|  |  |
| --- | --- |
| **Use Case UC-4:** Cross-Platform Usage  **Related Requirements:** Req-2, Req-18  **Initiating Actors:** Any of: User, Guest  **Actor’s Goal:** To access and the application across multiple platforms including a mobile app.  **Participating Actors:** Web Server, Alexa  **Pre Conditions:**   * Device is synced in with web server   **Post Conditions:**   * User logs into web server and performs an action from compatible device | |
|
|
|
|
|
| Flow of Events for Main Success Scenario: | |
| 1. User downloads app on their phone 2. User enters app on their phone 3. User is shown mobile version of website homepage 4. User buys a share of a stock 5. Transaction goes through | |
| Flow of Events for Alternate Scenarios | |
| 1. User syncs web server with Alexa 2. User wakes up Alexa 3. User asks Alexa for stock price 4. Alexa provides price of stock asked | |

|  |  |
| --- | --- |
| **Use Case UC-1:** Create-Account  **Related Requirements:** Req-1  **Initiating Actors:** Guest  **Actor’s Goal:** To create an account with DUCKLAB investments using a valid email, username, and password.  **Participating Actors:** Web Server, Database  **Pre Conditions:**   * Guest is currently not logged in. * Email is valid.   **Post Conditions:**   * Guest will become a registered user and will be able to participate in leagues. | |
|
|
|
|
|
| Flow of Events for Main Success Scenario: | |
| 1. User clicks on “create account” on the web page. 2. Web server loads the appropriate web-page. 3. User enters Username, password, and email and clicks “create account”. 4. Web Server checks the validity of username/password. Database checks if username or password are already registered. 5. Users information is entered into the database and they are logged in as a user. | |
| Flow of Events for Alternate Scenarios | |
| 1. User clicks on “create account” on the web page. 2. Web server loads the appropriate web-page. 3. User enters invalid/already registered Username and email and/or invalid password and clicks “create account”. 4. Web Server checks the validity of username/password. Database checks if username or password are already registered. 5. Message is displayed for the guest user clarifying the nature of the invalid input. | |

4.4: System Sequence Diagrams:









**5: User Effort Estimation**

|  |  |  |
| --- | --- | --- |
| Usage Scenario | Clicks | Keystrokes |
|  |  |  |
| Regular Login | 3 | 10-50 |
|  |  |  |
| Register Account | 5 | 10-50 |
|  |  |  |
| Join A League | 4 | 0-40 |
|  |  |  |
| Create a League | 5-6 | 14-60 |
|  |  |  |
| View Leaderboard | 3 | 0 |
|  |  |  |

**Regular Login:**

Assume user has reached site with intention of logging in and has previously registered. Navigation:  
1. Click on field to enter email and enter email, 1 click, email length keystrokes

2. Click on field to enter password and enter password, 1 click, password length keystrokes  
3. Click on login, 1 click

**Register Account:**

Assume user has reached site with intention of registering account.

•Navigation:  
1. Click Register An Account, 1 click   
2. Click on field to enter email and enter email, 1 click, email length keystrokes

3. Click on field to enter password and enter password, 1 click, password length keystrokes

4. Click on field to re-enter password and re-enter password, 1 click, password length keystrokes  
4. Click on Register, 1 click

**Join a League**  
Assume User is logged in and has intention on joining league  
• Navigation:  
1. Click on Leagues tab, 1 click  
2. Click on Join a League tab, 1 click  
• Data Entry:  
1. Click on desired league if displayed, or enter league name, 1 click,up to 40 keystrokes  
2. Click on confirmation tab after pop up occurs, 1 click

**Create a League**  
Assume user has logged in and has intentions of creating a league  
• Navigation:  
1. Click on Leagues tab, 1 click  
2. Click on Create a League tab, 1 click  
• Data Entry:  
1. Enter name of league, 4 - 40 keystrokes  
2. Select one of given game modes from pop up menu, 1 clicks  
3. Set parameters to user preference, 1-2 clicks and 10-20 keystrokes  
4. Click on confirmation pop up, 1 click

**View Leaderboard**  
Assume that the user has logged in and has intention of viewing leaderboard  
• Navigation:  
1. Click on Leaderboard tab, 1 click  
2. Click on click drop down menu, 1 click

3. Select League or Global, 1 click

# 

# 

# 6: Domain Analysis

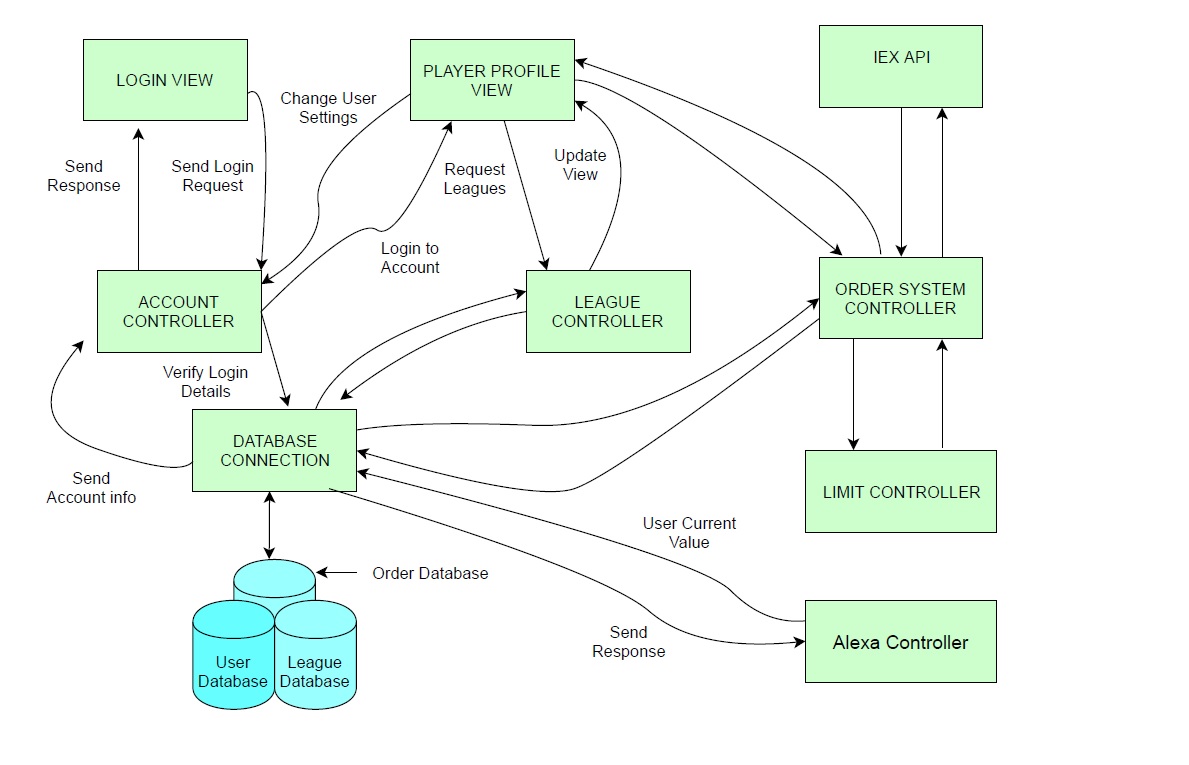
**6.1: Domain Model**

6.1.1: Concept Definitions

The Domain Model Concepts are derived from responsibilities contained in the Use Cases from Chapter 3.

|  |  |  |
| --- | --- | --- |
| Responsibility | Type | Concept |
| R1: Allow existing users to login and new users to create an account and participate in stock trading games. | D | Account Controller |
| R2: Make new accounts with already set fixed virtual amount to play different game modes. | D | Account Controller |
| R3: Check if user is in a league or not. | K | League Controller |
| R4: Update new leagues at set interval of time. (to be decided) | K | League Controller |
| R5: Retrieve information about Stocks of different companies. | K | IEX Finance Adapter |
| R6: Display information about users Stocks, Trades and Leagues. | K | Player Profile View |
| R7: Record and Execute Buy/Sell Stock trades | D | Order System Controller |
| R8: Display Welcome screen to create an account or sign up. | K | Login View |
| R9: Provides information about user current standings | D | Alexa Controller |
| R10: Limit the user investments to stop at specific point. | D | Limit Controller |

Domain Diagram



Account Controller

Home screen will provide user to login or sign up into the system. To sign up, account controller will check for the email if it already exists, if not account controller will make the new account and store the details in Users Database.

Player Profile View

The Player Profile View will show the users portfolio and investments through account controller. Users will be able to see updates investments on their portfolio through IEX Adapter. Users will be able to search for new stocks / companies through the portfolio.

Login View

The Login View displays a UI to allow the user to login into the system through their credentials which will be passed to account controller to match with Users Database. If succeeded, users will be navigated to user profile page. Otherwise will receive the error message to enter credentials again or sign up.

League Controller

League Controller will keep up to date display of rankings of every player in each league. These will be fetched from a League database and displayed in the Player Profile. League controller will allow Investors to join open leagues that they are not already in. This will require querying the database and updating if necessary. League Controller may give out achievements based on certain accomplishments within the leagues.

IEX API Adapter

IEX API provides the almost real time stock data that our application is dependent on. IEX Finance API Adaptor serves as a translation between the CSV le that IEX produces through their API, and our application. Our Adaptor will take the spreadsheet IEX Produces and convert the data into syntax that our application can understand. This adaptor is modular in order to allow multiple subsystems to make queries for stock updates.

Order System Controller

Any order placed by an investor will go through the Order System which will sum the cost of the transaction and check that the account have enough credit available. This will require communication with the IEX Finance API Adaptor to get the current price of the target stock to be purchased/sold. If the order is a limit trade, system will check if the current price of the stock matches the limit price and execute the order.

Database Connection

Database Connection will maintain three sub databases. Users Database will store the user’s credentials and help Account Controller to verify login / sign up. League Database will store the information for all the leagues created by league managers. Order Database will keep track of the orders asked by the users.

Alexa Controller

Once the user has updated the profile settings in their Alexa device, Alexa controller will be able to provide current status of the user stocks to the user through Alexa.

# 

# 6.1.2: Association Definitions

|  |  |  |
| --- | --- | --- |
| Concept Pair | Association description | Association name |
| Login View <> Account Controller | Login View ask the controller to verify login details. | Sends |
| Account Controller <> Database Connection | Account Controller verify the details from Database asked by Login View. | Sends |
| Account Controller <> Player Profile View | Account Controller updates the Player Profile View by making changes in Database. | Updates |
| Player Profile View <> League Controller | Users can join / leave the league. League Controller will update the Player Profile View accordingly. | Update / Sends |
| League Controller <> Database Connections | League Controller ask the Database to update the user league settings on user input. | Updates / sends |
| Player Profile View <> Order system Controller | Player Profile View ask the Order System Controller to update Order Database on users input. | Sends |
| Order System Controller <> Database Connections | Order System Controller ask Database Connections to update Order database on user inputs / orders. | Update / Sends |
| Order system controller <> IEX Adapter | Order System Controller asks for latest stock value from IEX Adapter before making the order. | Sends |
| Order System Controller <> Limit Controller | Order System Controller checks with Limit Controller if any limit has been put on the order by the user. | Sends |
| Alexa Controller <> database Connection | Alexa Controller ask Database Connections to retrieve user order status from Order Database and response back. | Sends |

The associations of domain concepts are evolved from the table above. The Account Controller gets the information from a Login View which user inputs and this information is sent to Database Connection to check if user already exists or not. From the Player Profile View a user can access account settings, leagues, and portfolio details. The league details are managed by the League Controller, which can allow requests to create/join a certain league. The Order System Controller allows the user to search market data, and attempt buy / sell stocks. The Order System Controller must communicate to see if the user has funds available. Limit trade may not be executed until the stock price reaches a certain value. The Order System Controller communicates with the IEX API Adaptor to retrieve current value for stocks. IEX Finance Servers must be online for this to work correctly.

6.1.3: Attribute Definitions

|  |  |  |
| --- | --- | --- |
| **Responsibility** | **Attribute** | **Concept** |
| R11: If user login failed | Login Failed | Login View |
| R12: Username | Username | Player Profile View |
| R13: User Account Balance | AccountSummary | Player Profile View |
| R14: Stocks user own | UserStocks | Player Profile View |
| R15: Leagues and rankings | League | Player Profile View |
| R16: If user is logged in | isLoggedIn | Account Controller |
| R17: If league is created | isCreatingLeague | League Controller |

6.1.4: Traceability Matrix

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Domain Concepts** | | | | | | | | | |
| Req’s | PW | Account Controller | Player Profile View | Login View | League Controller | IEX API Adapter | Order System Controller | Database Connection | Alexa Controller |
| UC-1 | 10 |  |  |  |  |  |  | X |  |
| UC-2 | 4 |  | X |  | X |  |  |  |  |
| UC-3 | 5 | X |  |  |  |  |  |  |  |
| UC-4 | 10 | X | X |  |  |  |  | X |  |
| UC-5 | 7 |  |  | X | X |  |  |  |  |
| UC-6 | 7 |  |  |  | X |  |  |  |  |
| UC-7 | 8 |  |  |  |  |  |  |  | X |
| UC-8 | 9 |  |  |  |  |  | X |  |  |
| UC-9 | 6 |  |  |  |  | X |  |  |  |
| UC-10 | 2 |  |  |  |  |  |  | X |  |
| UC-11 | 2 | X |  |  |  |  |  |  |  |
| Max PW | | 10 | 10 | 7 | 7 | 6 | 9 | 10 | 8 |
| Total PW | | 15 | 14 | 7 | 14 | 6 | 9 | 22 | 8 |

6.2: System Operation Contracts

**UC-1 Create Account**

Preconditions

* User is not registered
* User has valid email

Postconditions

* User account is added to the database
* User has access to membership actions

**UC-5 Create / Join League**

Preconditions

* User is logged into Duck Lab Investment System
* No leagues can have same name
* User is not already a participant of a league

Postconditions

* Manager of the league has applied settings
* League Database is updated

**UC-2 Viewing Investments**

Preconditions

* User is registered in the database
* User is part of a league

Postconditions

* Users investments are reflected from the database

**UC-4 Cross Platform Usage**

Preconditions

* User is registered in the database
* User has access to multiple devices with internet
* User has multiple devices synchronized with the database

Postconditions

* Database synchronizes users account on each device
* User has access to the database from synchronized devices

Mathematical Model:

The math that we will use in this project will be limited to basic arithmetic operations (add, subtract etc.). Therefore we will not have a mathematical model.

**7: Interaction Diagrams:**

7.1 Introduction

The interaction diagrams in the following section will show how sub parts of the system will interact with each other. For each particular use case, we will outline the interactions among the subsystems and databases. Further, we will analyze multiple cases in which the systems will handle different scenarios. That is, it will show how the system handles both failure and success conditions. Users will need to log in and constantly access data from database to have constantly refreshed and updated information.The diagrams below will accurately detail how this will be accomplished within the system.

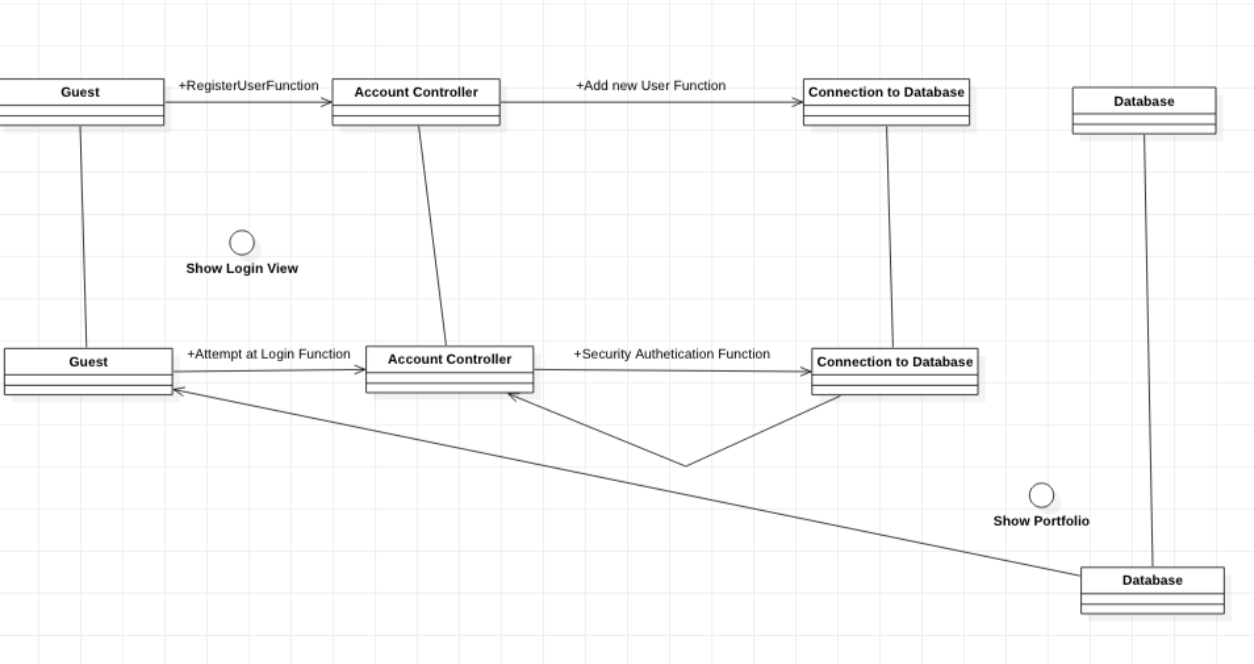
7.2: Diagrams

**Use Case 1**

In the sequence diagram for our first use case, we display two options that occur when a guest user is operating our product. Separated by two series of boxes they’re connected by non-arrowed lines to show how they are the same instance but two different scenarios are occuring. Between the first four icons shows an interface circle, which would appear to both situations hence why it is positioned in the middle.

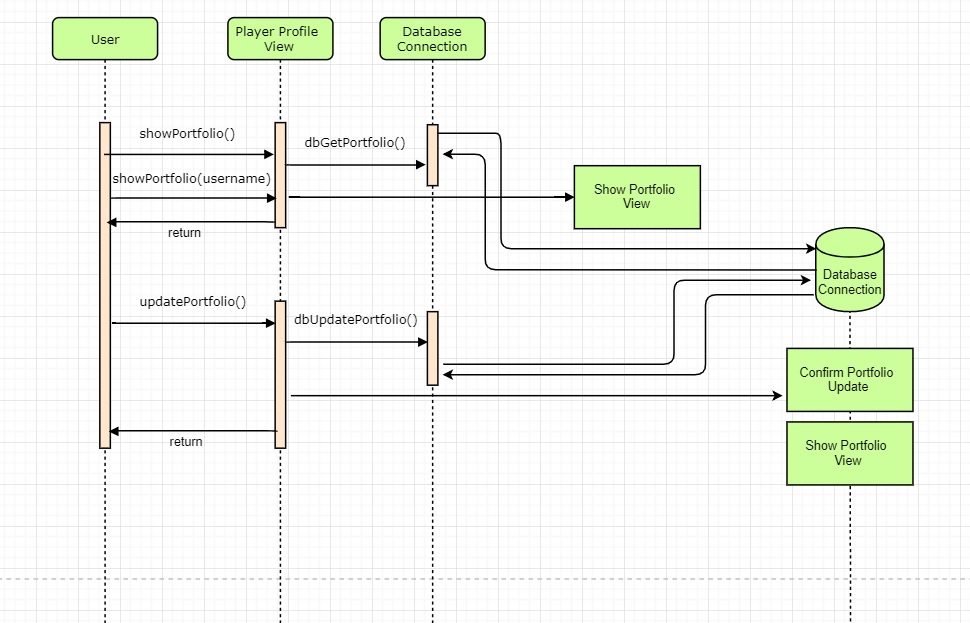
The first scenario is when a guest visits the login page and wishes to register an account. The first instance that occurs is that the register new account function contacts the account controller with the information that was taken in by the registration user intake fields, that were prompted. The account controller then calls the add new user function which contacts the database to make sure there is no duplicate data (email, first, last name etc). If no data duplicate is there, it stores the data in the database.

Once registered, if a guest tries to login with the correct login information, the account controller checks the information with the database, if correct, the program brings the guest to the Portfolio page. If incorrect the user is kept at the login page a prompted to try again, with either a new password attempt or a new email attempt.



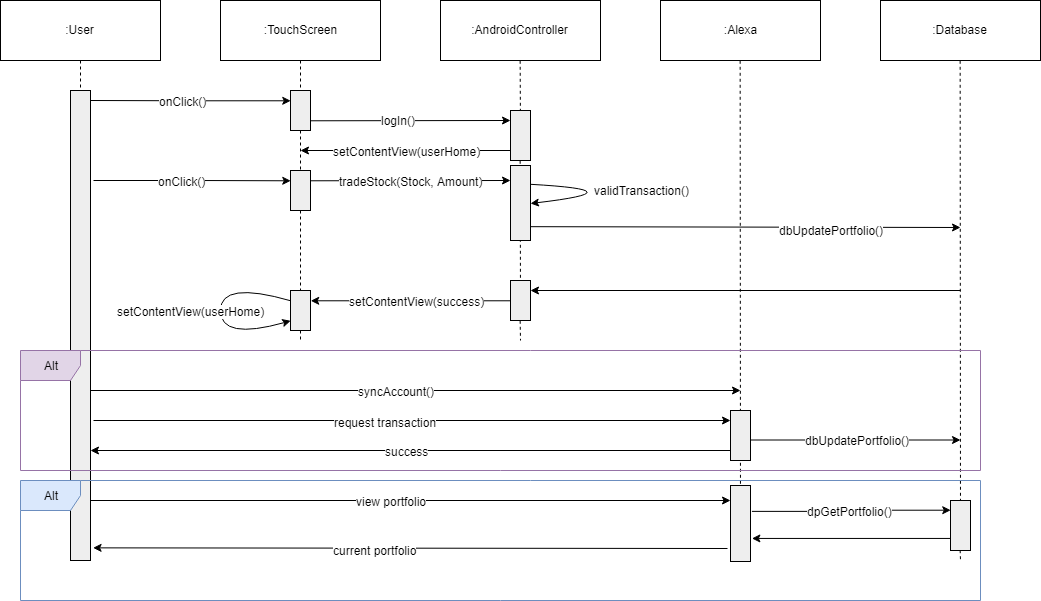
**Use Case 2**

The user should be able to view and make changes to their Portfolio View or view other players Portfolio with limited access . When the user clicks to show portfolio, the Player Profile View Controller will fetch the user portfolio stocks from the database via the DB Connection module. The user can also update their view of the portfolio and other settings. When ever order is made by the user, Portfolio will get updated simultaneously.



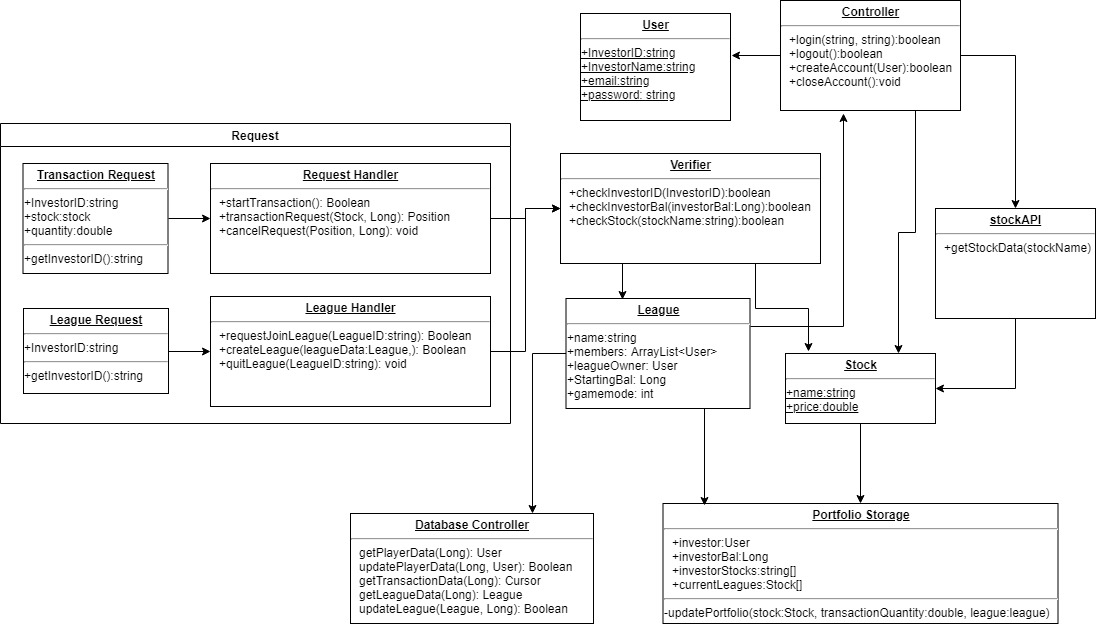
**Use Case 5**

For the case of cross platform usage, the user should be able to interact with the web server through an android application or Alexa. Through the touch screen the user can to send commands to the AndroidController. With this capability, user should be able to log into the application with their account information and have access to standard user actions. For an alternate situation, the user should be able to synchronize Alexa with the DuckLab server. After synchronization the user should be able to interact with the web server through Alexa through commands such as: “buy/sell stocks” and “view portfolio”. All transactions done on any platform will be updated on the database and in turn other synchronized devices.

****

**8. Class Diagram and Interface Specification**

8.1 Class Diagram



8.2 Data Types and Operation Signatures

Database Controller

The database controller is responsible for managing the database and the processes using it. The functions available to the controller allow it to manipulate player data, league data, and perform order transactions.

**Methods**

* getPlayerData(in userID: long): class User

Used when a player data retrieval request is placed

* updatePlayerData(in userID: int, in update: class user): bool

Used to replace old player data with updated player information and statistics

* getTransactionData(in transactionID: int): class transaction

Used to retrieve stock data for a specific transaction

* getLeagueData(in leagueIDLint, in leagueData: class league): bool

Used to retrieve league information, statistics and rankings

* updateLeague(in leagueID: int): class league

Updates league data with real-time statistics and rankings

Request Handler

The request handler takes care of processing orders by starting transactions, requesting data from the stock API, confirming the order and canceling started transactions.

**Methods**

* startTransaction(in symbols: class Order) : bool

Used to begin orders of specific stock shares

* transactionRequest(in symbols: class Order) : bool

Used to request stock data from the API and process into the order

* cancelRequest(in symbols: transactionID): bool

Used to stop and terminate ongoing transactions

Portfolio Storage

The portfolio storage is responsible for player and league data availability to the users. It hold player names, balances, stocks and current leagues.

**Methods**

* investor:User

Used to retrieve the investor username

* investorBal:Long

Used to retrieve the player league balances

* investorStocks:string[]

Used to retrieve the players statistics such as stock name and information

* currentLeagues: Stock[]

Used to retrieve league data for the player portfolio

**Controller**

* login(string, string):boolean  
  User’s login attempt.
* logout():boolean  
  Logs user out.
* createAccount(User):boolean  
  Creates account for user, specific to the inputted User object.
* closeAccount():boolean  
  Deletes user account.

**Stock**

* name:string  
  Responsible for the name of the stock
* Price:double  
  Responsible for the price of a unit of the specified stock

**League**

* name: string

Responsible for the name of the league

* members: ArrayList<User>

Creates an array list of the users in the specific league

* leagueOwner: User

Allows a certain user league managing capabilities.

* startingBal: Long

Responsible for providing starting balance to each user in the league

* gamemode: int

Sets the specified game mode rules for the league.

**StockAPI**

* getStockData(stockName)  
  Takes in a string type stockName to retrieve stock data

**User**

* investorID: string

Responsible for a string of the users ID

* investorName: string

Responsible for the string of the users name

* email: string

Contains a string of the users email address

* password: string

Contains an encrypted string of the users password

**Verifier**

* checkInvestorID(investorID): boolean

Compares users ID to the database for validity.

* checkInvestorBal(investorID): boolean

Checks users balance for validity

* checkStock(stockName: string): boolean

Checks the string of the stock name for availability

**League Handler**

* requestJoinLeague(leagueID:string): boolean

Request to add user to league array

* createLeague(leagueData: league): boolean

Request for new league by user

* quitLeague(leagueID: string): void

Request by user to be removed from specified league

**League Request**

* investorID: string

Responsible for holding the users ID

* getInvestorID(): string

Retrieved users ID for a league request

**Transaction Request**

* investorID: string

Specifies users ID

* Stock: stock

Specifies the stock to be used in the transaction

* quantity: double

Specifies the number of shares requested for the transaction

* getInvestorID(): string

Retrieves users ID for the transaction

8.3: Traceability Matrix

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Domain Concepts** | | | | | | | | | |
| Req’s | PW | Account Controller | Player Profile View | Login View | League Controller | IEX API Adapter | Order System Controller | Database Connection | Alexa Controller |
| UC-1 | 10 |  |  |  |  |  |  | X |  |
| UC-2 | 4 |  | X |  | X |  |  |  |  |
| UC-3 | 5 | X |  |  |  |  |  |  |  |
| UC-4 | 10 | X | X |  |  |  |  | X |  |
| UC-5 | 7 |  |  | X | X |  |  |  |  |
| UC-6 | 7 |  |  |  | X |  |  |  |  |
| UC-7 | 8 |  |  |  |  |  |  |  | X |
| UC-8 | 9 |  |  |  |  |  | X |  |  |
| UC-9 | 6 |  |  |  |  | X |  |  |  |
| UC-10 | 2 |  |  |  |  |  |  | X |  |
| UC-11 | 2 | X |  |  |  |  |  |  |  |
| Max PW | | 10 | 10 | 7 | 7 | 6 | 9 | 10 | 8 |
| Total PW | | 15 | 14 | 7 | 14 | 6 | 9 | 22 | 8 |

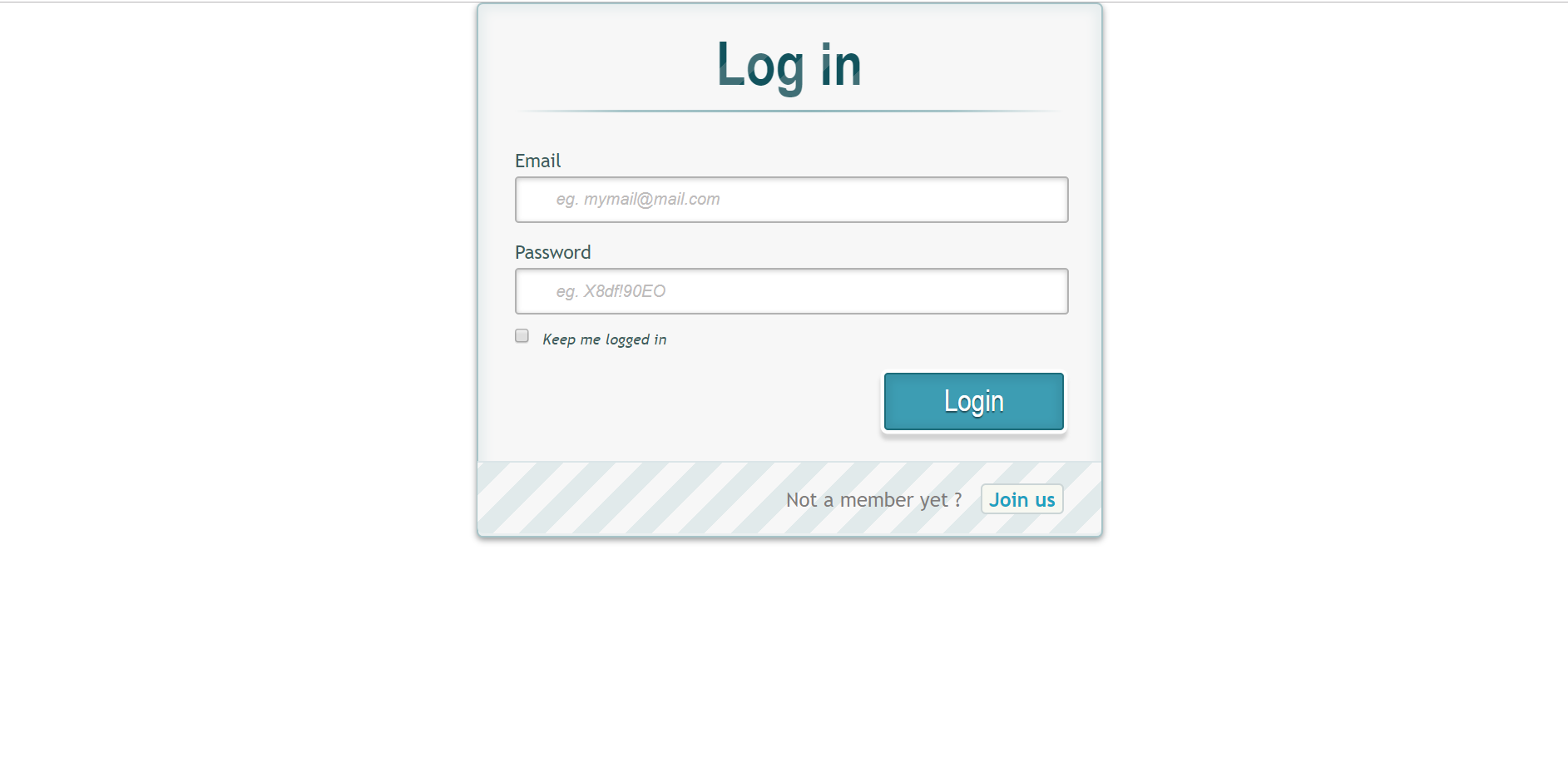
8.4: User Interface Specification

We, at Ducklab, wanted to design the user interface to act as a smooth simple design which will provide the user with all of the games that they are currently part of. The design will consist of their current portfolio, leagues they are a part of, and conduct research on potential orders. The dashboard will consist of all the graphs that the user selects that shows particular stocks, a specific portfolio or a specific game. There will also be an “all games” section; this will show all of the users current games and will inform the progress and the amount of money that was invested in each game. There will also be the leaderboard section. This will contain the leaderboard of all of the games the user is a part of as well.

The UI will be built on top of Twitter's open source Bootstrap CSS framework to help make easy and well crafted visuals for our wer application. The content will be accessible to the three target platforms, desktop, mobile, and tablet. Bootstrap provides a mobile rst design philosophy, but can be customized to target specific platforms.

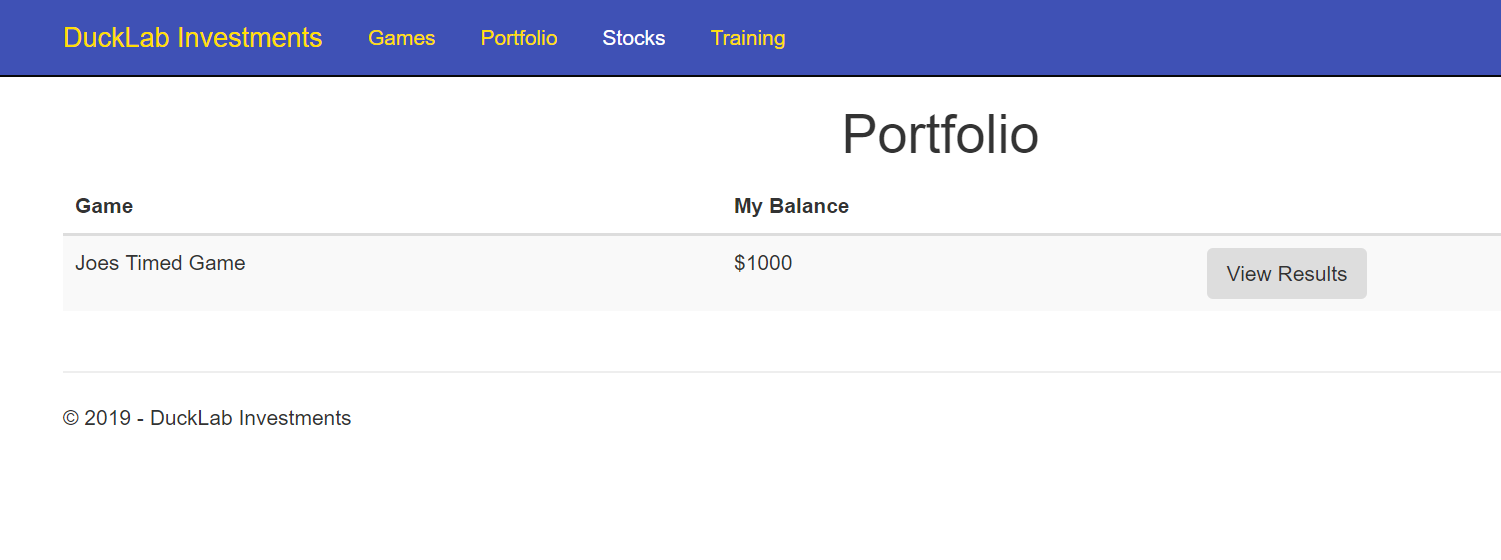
Login page

The login page will have the easy ways for new users to register and for existing users to login. We will focus on adding new members with an easy login option. We need to allow the users to sign in with facebook, twitter, google etc. This will make new users more likely to use our service for the ease of access. The login page will have a simple color scheme and will focus on informing the user of our storing points and the reasons they should play our game.



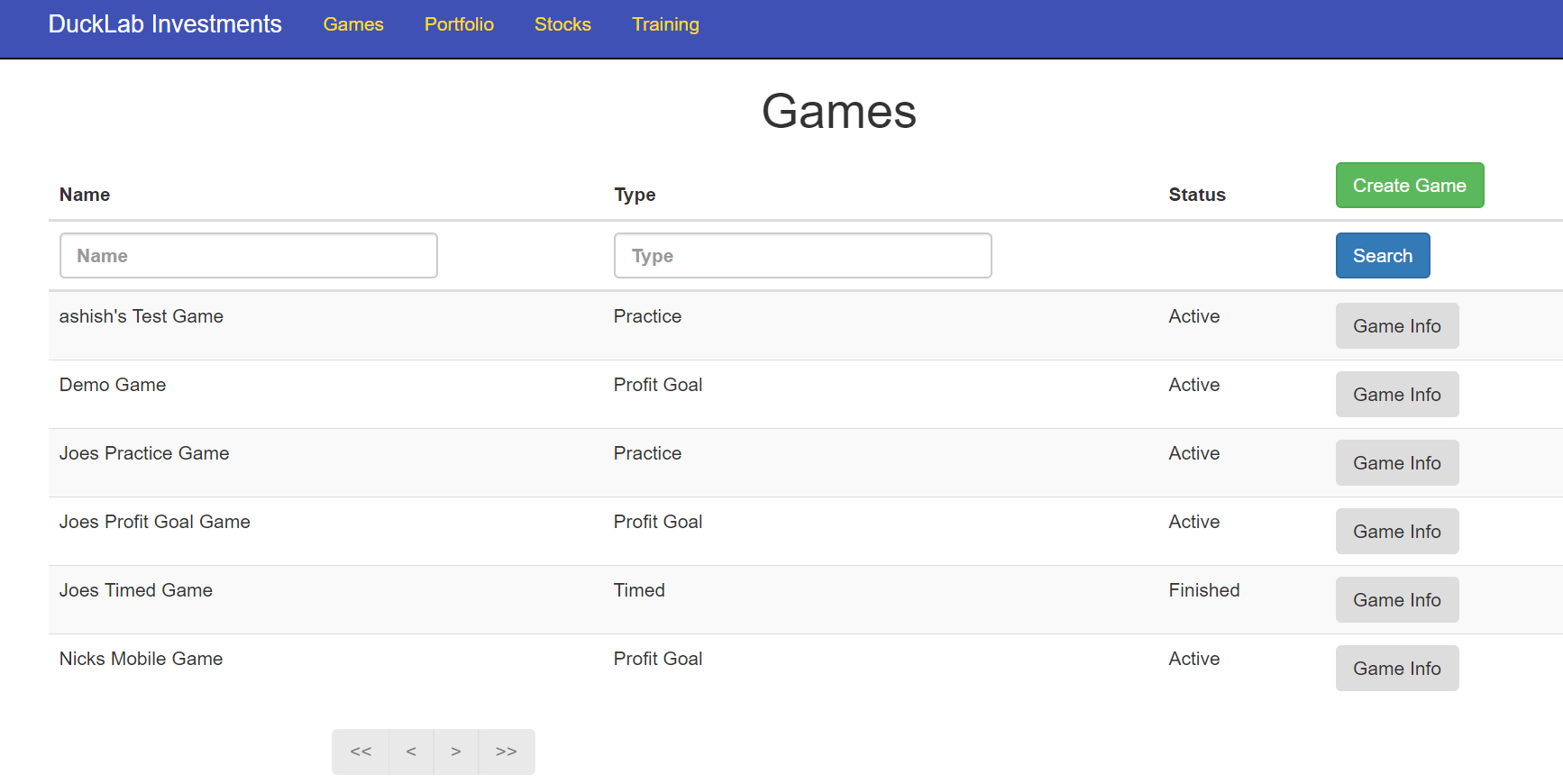
Portfolio

The toolbar will have easy access to the dashboard, all the games the user is a part of, the leaderboards and a place to start a new game. The toolbar will be present at all of the pages except for the login page. This will allow all of the users to have an easy simple way to move around our app.



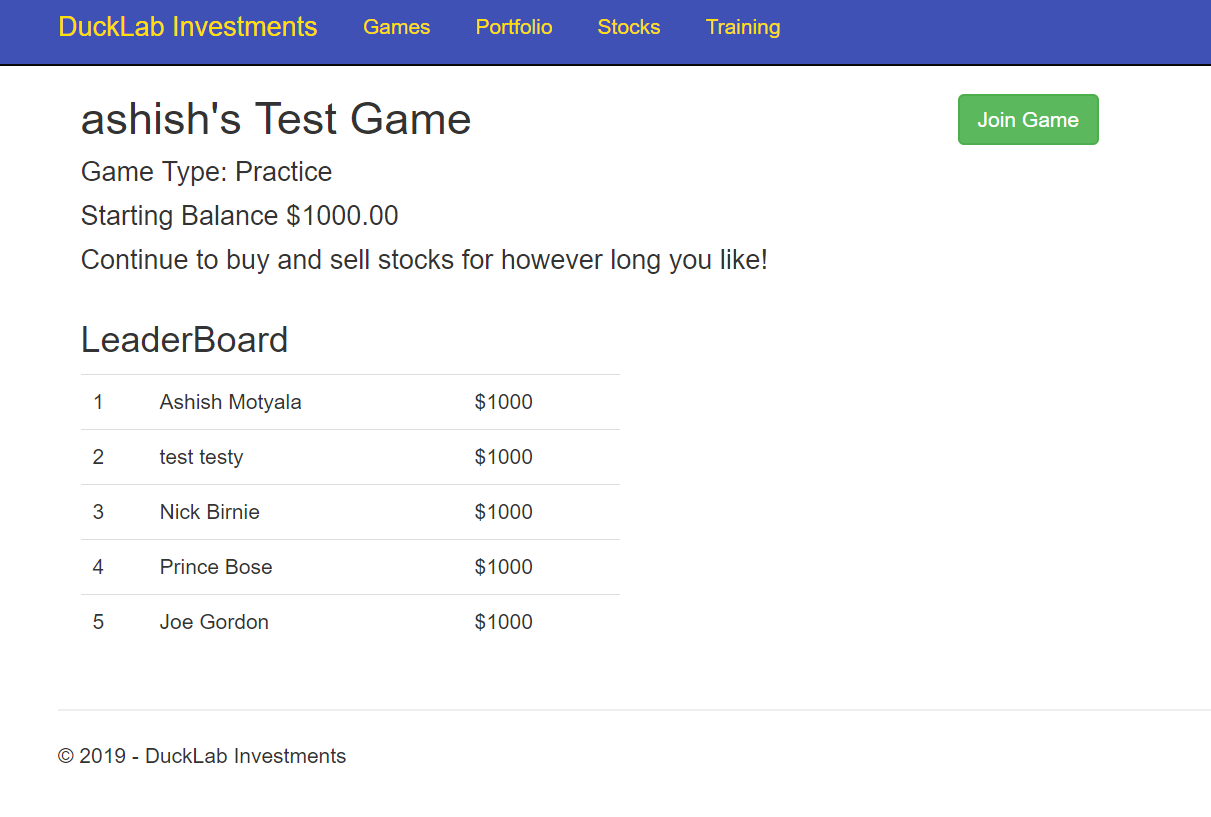
Games

The games tab will have all of the current games that the user is apart of. This will also show the progress of each game and the total money invested.The games tab is an ideal way to show the user the way all of their games are going at once and will allow all of the users .



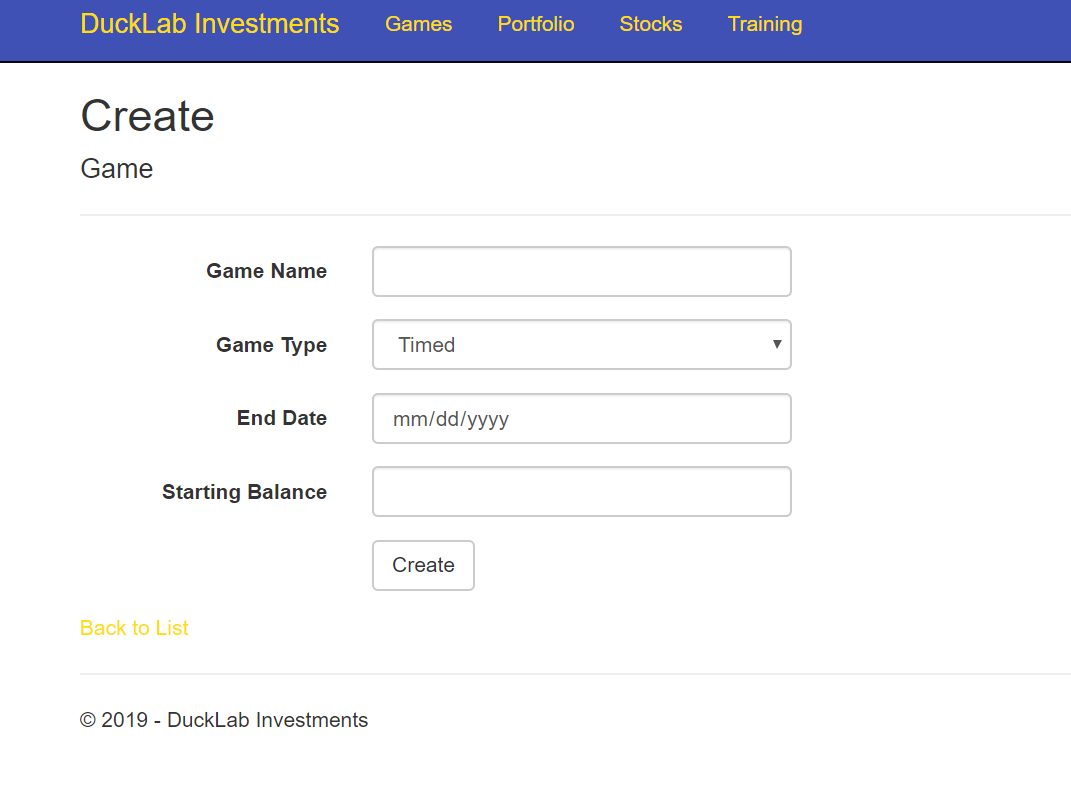
Leaderboards

This is still in working progress, however there will be a global leaderboard and each user will have their own friends leaderboard. This is the place where users can come and see their progress based on other peoples rank and place in their games. It also provides a sense of connectivity between friends.



Create Game

This will be the place for users to easily create new games. This will be the place were the users will be able to send requests and set the duration of the games. There will also be tutorials of the way each game mode is played.



**9. System Architecture and System Design**

9.1 Architectural Styles

Ducklab Investments will use multiple different architectural styles to give the user a seamless experience. The main focus will be with the model/view/controller architectural style as well as the Entity Framework structure. The .NET Web Application will be built around these architectures. Here are our styles:

9.1.1 Model/View/Controller (MVC)

The MVC architecture divides the workload of the application into three different types of objects. The Model represents all of the data the application uses. The model will be connected to our SQL database. The Controller objects handle the logic of our application. In our application, our views are coded in the C# language. Controllers can communicate to the model and retrieve the data it requests. Controllers can pass data to View objects. View objects are the front end of the application. They our coded with HTML and can also retrieve data from the controller and can utilize C# as well to use logic to display the data.

9.1.2 Entity Framework

In our application we utilize Entity Framework to take the tables the Model gets from our SQL database and convert them into classes. This is extremely helpful when we are handling data in our controllers. It allows us to retrieve, update and delete records in our SQL tables.

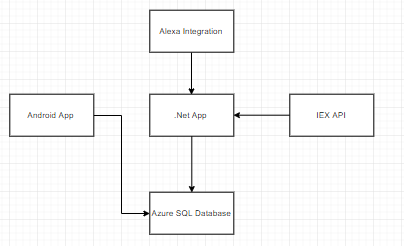
9.1.3 Front-End and Back-End Specific Architecture

Since our goal is many different types of applications, integration to one seamless system will be difficult. To solve this we will be using a well defined front-end and back-end architecture. We will be using the web browser to show our web application using HTML and CSS to display the app to the user. For the Android app we will be using Java and display the same user to the user as the Web application. For the alexa integration we will be using python and hosting on the AWS lambda tool for our hosting. However, in order for these three front end architectures to synchronize and provide the user with a seamless experience; we need to use one backend database and synchronise our applications to it. This backend will be hosted on Microsoft Azure and will be a MySQL database. This database will then communicate with our other applications and allow the user to access their account from multiple locations.

9.1.4 Event Driven Architecture

For our Stock Management app to be realistic we need to provide a wide variety of stocks for our user. In order for us to do this we incorporated all the stocks we have in an SQL database. But if we update the entire list when a user asks for a stock, the average wait time is 3 minutes. Therefore when a user asks for a stock, that specific stock will be updated and this will take less than a second.

9.2 Identifying Subsystems



9.3 Mapping Subsystems to Hardware

Our subsystems (what subsystems) will be run on remote cloud servers. We chose to host the application using Microsoft Azure tools. This will allow us to do maintenance and updates with relative freedom. We manage our application by first testing locally using visual studio and Forking protocols with Git. We then host the updated application to Microsoft Azure using their user-friendly gui. The system can be managed by any of the administrators and changes and bug fixes can be applied with ease. We also have our SQL Database hosted on Azure so integrating our software backend will be simple. We then can host the android application on the Google Play store using the Google Servers and where we will also host the app while it is in development for demo purposes.

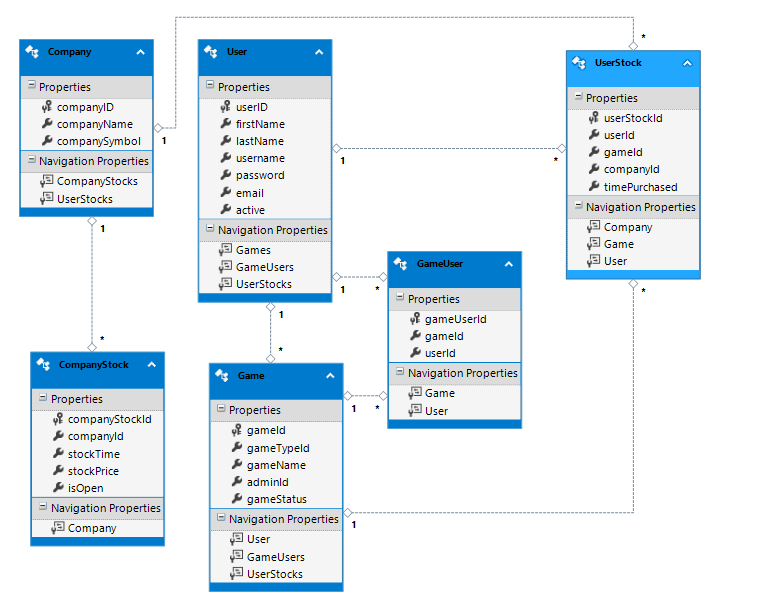
9.4 Persistent Data Storage

For our project it is infeasible to pull all the data we need every time someone logs into the game. Hence, we need to store all of the data in a database so that we don't have an incredibly slow system. We need to store all of the stock prices and other stock data in a relational database. Specifically an SQL database hosted on the Azure cloud.

Relational databases are far more practical for the needs of this particular software. That is, relational databases consist of several indexed tables labeled with various object attributes. 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. Users will be able to update their password in the database but username and email cannot be updated once its set. In case of forgotten password, verify email will be sent on user email to set a new password and that will overwrite the current saved password.

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(s) 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. 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 IEX 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 Duck Lab system will distinguish itself from others and retain functionality and efficient realization of its ultimate goals and requirements.

****

9.5 Network Protocol

We chose for the web app to use HTTP because it is the easiest to implement with a .NET application. 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 system interface. More importantly, the HTTP protocol is supported across multiple devices. From any of these mediums, the users can access various web pages and links from the Duck Lab 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.

9.6 Global Control Flow

9.6.1 Execution Orderliness

The events are independent for every user. Meaning, one user can be creating a game while another is just checking their stocks and another is just browsing the site to explore our functionality. Thus, the system is event driven at times but not always. The games will update scores in an event driven manner.

9.6.2 Time Dependency

We will run a script to periodically refresh all of the data once every day at a specific time. This will have one timer. Also the games will each have a timer specifying when the game has started and when it will end. This will be different for most games.

9.6.3 Concurrency

The Android app will use multiple threads to run all of the touch listeners but all of this implementation is abstracted by the Android developing suite.

9.7 Hardware Requirements

The system we have developed needs only a web browser. Any screen resolution that works for a web browser will do. For the Android app part of our system any phone running Android Marshmallow 6.0 or later will work.

**10. Algorithms and Data Structures**

10.1: Algorithms

As it is not feasible to show all the algorithms for our website, we are giving one example for how we will get individual company stock value when requested by the user.

public PartialViewResult \_Companies(string name = "", string symbol = "", int lastPage = 0, string page = "")

{

int numPage;

if (page == "" || page == "Search")

numPage = 1;

else if (page == "<")

numPage = lastPage - 1;

else if (page == ">")

numPage = lastPage + 1;

else

numPage = Convert.ToInt32(page);

ViewBag.page = numPage;

ViewBag.name = name;

ViewBag.symbol = symbol;

ViewBag.pageCount = db.Companies.Where(x => x.companyName.StartsWith(name) && x.companySymbol.StartsWith(symbol)).Count();

if(ViewBag.pageCount % 25 == 0)

ViewBag.pageCount = ViewBag.pageCount / 25;

else

ViewBag.pageCount = (ViewBag.pageCount / 25) + 1;

numPage--;

return PartialView(db.Companies.Where(x => x.companyName.StartsWith(name) && x.companySymbol.StartsWith(symbol)).OrderBy(x => x.companyName).Skip(numPage \* 25).Take(25).ToList());

}

10.2: Data Structures

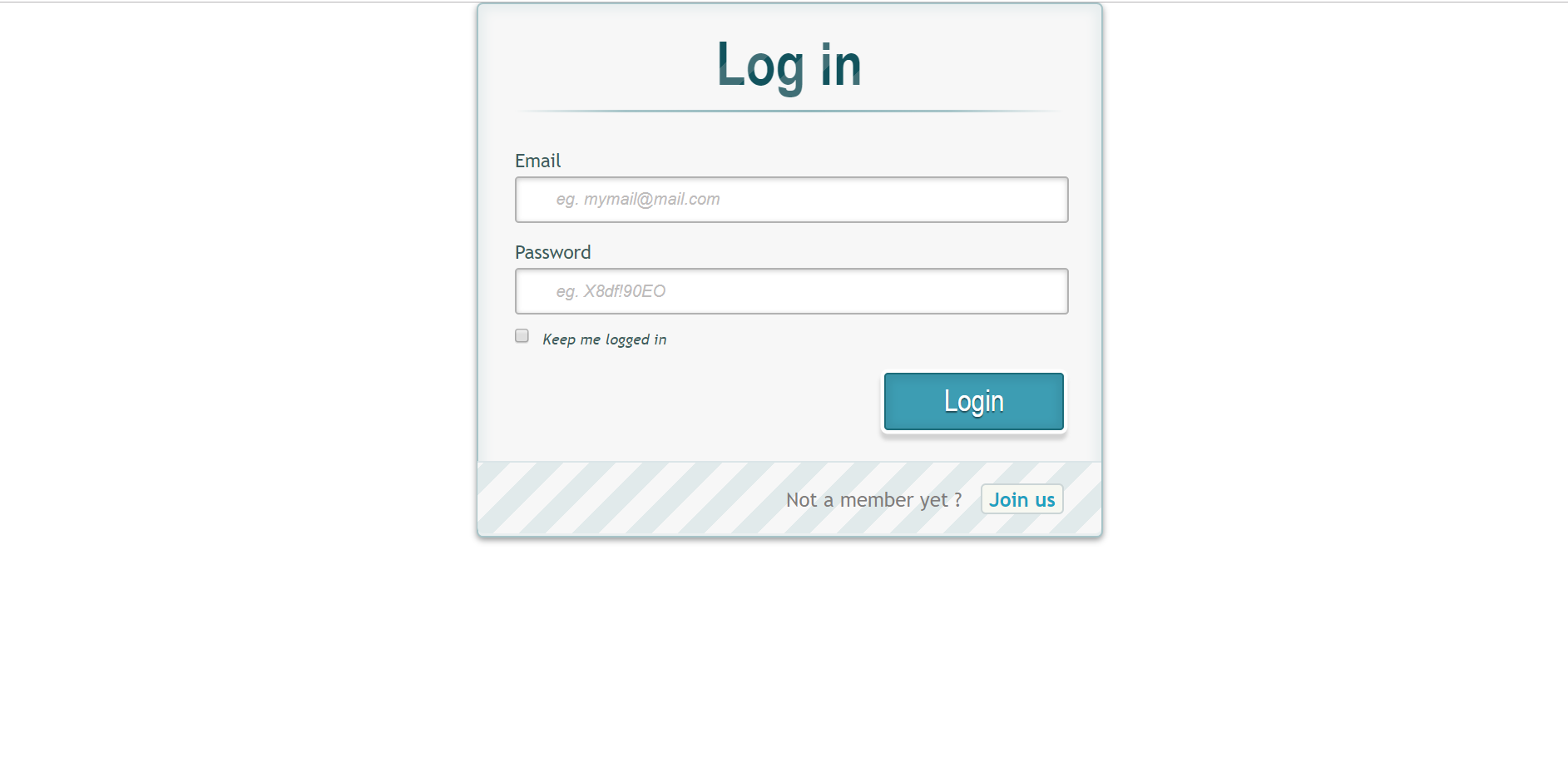
The system uses many types of data structures to accomplish many distinct goals.

For main data storage we are using an SQL database. This will contain all information about leagues, current stock data, and user profiles. Arrays, ArrayLists, and objects will be utilized heavily throughout the project.

**11. User Interface Design and Implementation**

11.1 Updated Pages

Work is still in progress but at the time, we are working on homepage, registration page and stock data page for our website. When a user enters the URL for our website, he/she will be navigated to homepage where an option will be provided to the user to enter login credentials. If a user is already a member of Duck Lab investments, he/she can input the credentials and it will navigate the user to their portfolio but if a user is not a member already, by clicking the sign up button they will be navigated to registration page where they can register for Duck Lab investments by entering some basic information.



11.2 Efficiency of the Views

Duck Lab investments is making sure that the system provide fast and smooth navigation for all of its users. We are using AJAX (Asynchronous JavaScript and XML) to design user interfaces that provide an interactive experience while collecting data from users and dynamically creating new page elements.

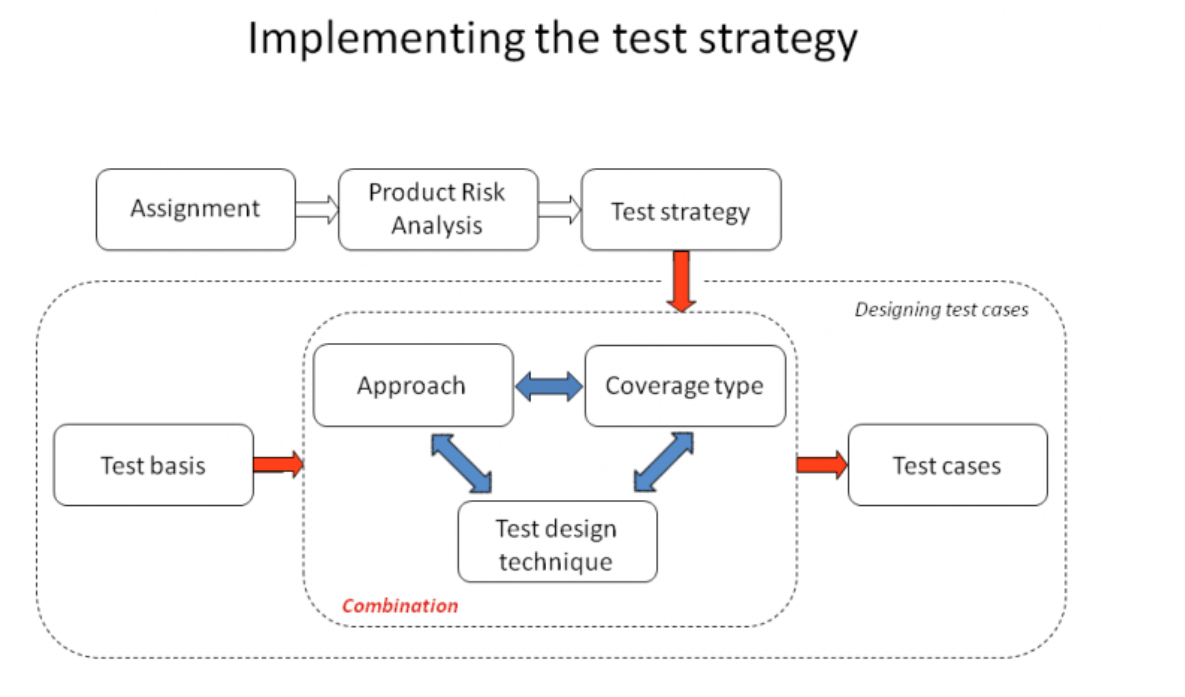
AJAX provides a way for web applications to asynchronously send and receive data from a server without the need to refresh the page or interfere with its appearance. By decoupling the presentational layer from the data exchange layer, it enables web applications to dynamically change content without reloading the entire page.

Ajax has enabled many enhancements to website user interfaces as support for Ajax has become more extensive in modern web browsers. Support for Ajax in the widely-used jQuery JavaScript library has simplified Ajax implementation. All modern devices support modern web browsers which will allow us to use AJAX technology without impacting our system efficiency.

**12. Design of Tests**

One of the most important goals of testing is a clear advise on quality and risk in such a way that all the parties involved gain confidence in the product. To be able to do this, a tester has to gather information on system behavior. One of the main tools in gathering information is the executing [test cases](http://www.tmap.net/wiki/test-cases). The results of those cases give information on the system behavior. The main questions are: Which [test cases](http://www.tmap.net/wiki/test-cases)? How many? And how do we get those cases? In answering those questions test design is indispensable.

Designing the right set of [test cases](http://www.tmap.net/wiki/test-cases) is the essential link between the [test strategy](http://www.tmap.net/building-blocks/test-strategy) and the implementation of the [test strategy](http://www.tmap.net/building-blocks/test-strategy) - the tests that are executed. See the picture below for the link between the relevant terms in test design:



The terms and conditions of the [assignment](http://www.tmap.net/building-blocks/assignment) determine the time and the effort available for testing. In the [Product Risk Analysis (PRA)](http://www.tmap.net/building-blocks/product-risk-analysis-pra) is determined which parts of the system are most important to test and should be tested more thoroughly. In the [test strategy](http://www.tmap.net/building-blocks/test-strategy) an overview is made of the whole test and how the testing effort is divided between different test varieties to cover all the risks most efficiently. The parts of the test object that are under test and the thoroughness that is used when testing those parts determine the coverage over the test.

Accordingly, we first define the features and tests we plan on developing around, proceed to analyze the coverage offered by these tests, and then briefly discuss how we intend to test the integration of the components.

12.1: Test Cases

Duck Lab Investments is using a ASP.Net Framework to develop our web application system. The main reason for choosing this framework is that it is one of the latest framework, very user friendly, easy to implement and also supports big databases. Also, most of the team members are proficient in C# and ASP.Net supports C#.

We are using SQL database to store all the data and website and SQL will be interacting for updating / fetching any data. We will run some interaction tests to see if we get the accurate results.

**Test Case # 1: Registration**

Test to create an account on the website

|  |  |  |
| --- | --- | --- |
| Procedure | Result | Action |
| Pass input for user name | Pass | User entered valid string |
| Fail | User entered a non-string value |
| Pass input for email | Pass | User entered existing email |
| Fail | User entered a non existing email |
| Pass input for password | Pass | User entered a string value 8 characters or longer |
| Fail | User entered non-string value or string value shorter than 8 character |
| Activate create account | Pass | User presses “Create Account” button |
| Fail | User doesn’t press “Create Account” button |

**Test Case # 2: Login**

Test to see how login will behave towards invalid inputs.

|  |  |  |
| --- | --- | --- |
| Procedure | Result | Action |
| Valid input for username and password | Pass | User entered valid string for username and password. Matched string values with database values |
| Fail | User entered invalid string. Return error |
| Invalid input for username and password | Pass | User entered invalid string. Return error |
| Fail | Do not show error |
| Valid input for username but not password | Pass | Return error |
| Fail | Login the user and navigate to user portfolio |

**Test Case # 3: Log out**

Test to check how system will react when user clicks logout

|  |  |  |
| --- | --- | --- |
| Procedure | Result | Action |
| User clicks logout | Pass | User gets navigated to homepage.. |
| Fail | User gets navigated to some other page or no action. |

**Test Case # 4: Purchasing Stock**

Test to purchase a stock

|  |  |  |
| --- | --- | --- |
| Procedure | Result | Action |
| Request to buy stock | Pass | User entered valid stock and quantity |
| Fail | User entered invalid stock value or quantity |
| Confirmation of transaction | Pass | User has enough balance to purchase stock. Transaction is confirmed |
| Fail | User doesn’t have enough balance for the transaction. Transaction canceled |

**Test Case # 5: Selling Stock**

Test to sell a stock

|  |  |  |
| --- | --- | --- |
| Procedure | Result | Action |
| Request to sell stock | Pass | User entered valid stock and quantity |
| Fail | User entered invalid stock value or quantity |
| Confirmation of transaction | Pass | User has more or equal stock shares compared to amount requested |
| Fail | User doesn’t have enough stock shares for the transaction. Transaction canceled |

**Test Case # 6: View Portfolio / User Data**

A successful test is one that retrieves information about the requested player.

|  |  |  |
| --- | --- | --- |
| Procedure | Result | Action |
| User enters valid input to login and gets navigated to portfolio. | Pass | Information requested matches the player data. |
| Fail | Information requested does not match the player data. |

**Test Case # 7: Create League**

Test to create a league

|  |  |  |
| --- | --- | --- |
| Procedure | Result | Action |
| Click “Create League” button | Pass | User clicked the correct button and is given league owner privileges |
| Fail | User reached limit of leagues allowed to create or system has an error. |
| Enter League settings | Pass | User entered correct values in settings fields. League is created with specified settings |
| Fail | User entered invalid values for settings fields. Error is shown. |

**Test Case # 8: Join League**

Test to join a league

|  |  |  |
| --- | --- | --- |
| Procedure | Result | Action |
| Find a League | Pass | User entered a valid value for an existing league ID and is shown the specified league |
| Fail | User entered a non-string value or a valid value for a non-existing league ID |
| Click “Join League” | Pass | User clicked “Join League” button and is added to the roster |
| Fail | User clicked the wrong button and is transferred to different page or user is denied due to limitations of specified league |

**Test Case # 9: Update Portfolio / User Data**

A successful test is one that updates a player's information, whether it be an administrative action or game related.

|  |  |  |
| --- | --- | --- |
| Procedure | Result | Action |
| User updates user settings or make a purchase/sell order | Pass | User portfolio is updated with new information. Return true |
| Fail | User portfolio is not updated with new information. Return false |

**Test Case # 10: Get Company Stock Data**

A successful test will return the current stock value for the company user requested.

|  |  |  |
| --- | --- | --- |
| Procedure | Result | Action |
| User is logged in and clicks on company’s name | Pass | Company stock information is presented to the user. System can access company stock information from database |
| Fail | Company stock information is not presented to the user. System cannot access company stock information from database |

**Test Case # 11: Add Rules by League Manager**

A successful test will add a new rule to the list of league rules already established.

|  |  |  |
| --- | --- | --- |
| Procedure | Result | Action |
| User is a league admin and trying to add a new rule to the existing set of rules | Pass | The newly added rule is reflected in the database. |
| Fail | The newly added rule is not reflected in the database or no action |

12.2: Test Coverage

The ideal test coverage would be to have a test that covers every edge case of every method. This is not just not feasible, it is impossible since it is not possible to actually know all the edge cases. Because of this we plan to test core functionality to provide a core amount of testing. Then through the use of alpha and beta build interactions with end users, we will be able to identify ways that user interact with the system that were not foreseen. We can then add additional testing to cover these new edge and use cases which will also help debug and prevent regression in the future.

12.3: Integration Testing Strategy

Integration testing will be done on a local developer machine by emulating the server environment. The system may not go live until the current system works in the integration environment. We accomplish this by having two branches of source code, master and developer. Developer is the branch that all new work will be done on. From there, it will be pulled down into the local integration machine, tested and debugged. Once the system has been debugged, the source code will be pushed to master. Once pushed to master, any system config changes will be made on the production server in order to accommodate the new branch. Once those changes are made, master will be pulled into the production machine and a second round of integration testing will begin by launching the service on a developer port. If it passes all the tests, then the developer port will be shut down, and the system will relaunch the website on the normal http port.

**13. History of Work, Current Status, & Future Work**

During the complete course of semester we have worked on this project and accomplished many small task keeping in view the timelines provided to our team.

We as a team worked on the project and submitted our Report 1 on the given timeline.

Besides working on Report 1, we were developing our website side by side so we get ready for first Demonstration. During our first demonstration, we showed the working model of our website along with some key functions of our Android application and hardware ticker.

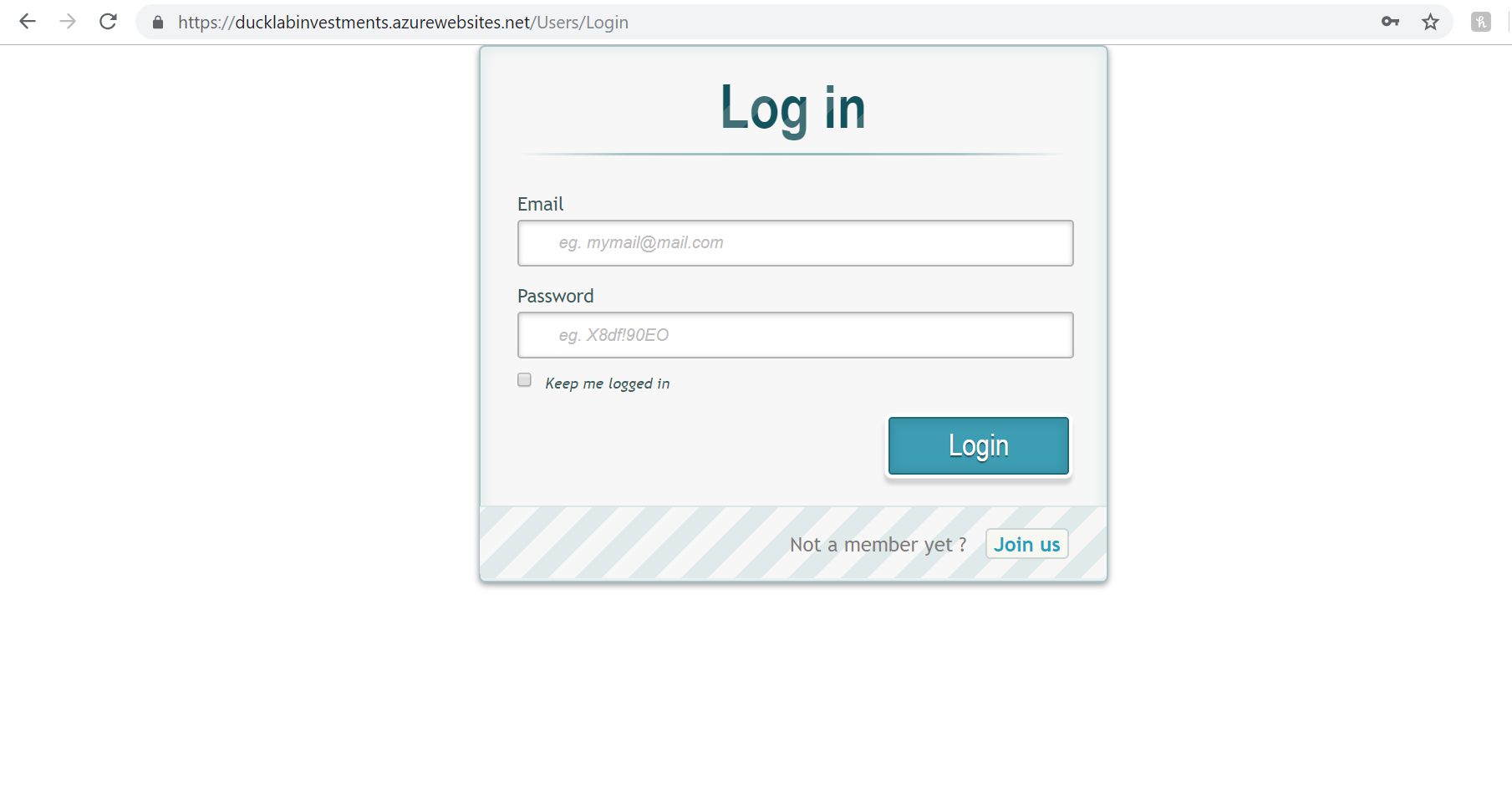
` Our second report was submitted on time and after completing all the functionalities of our website, we focused on our Android application to get it complete by the second Demonstration and besides that we started to write API for our Alexa integration.

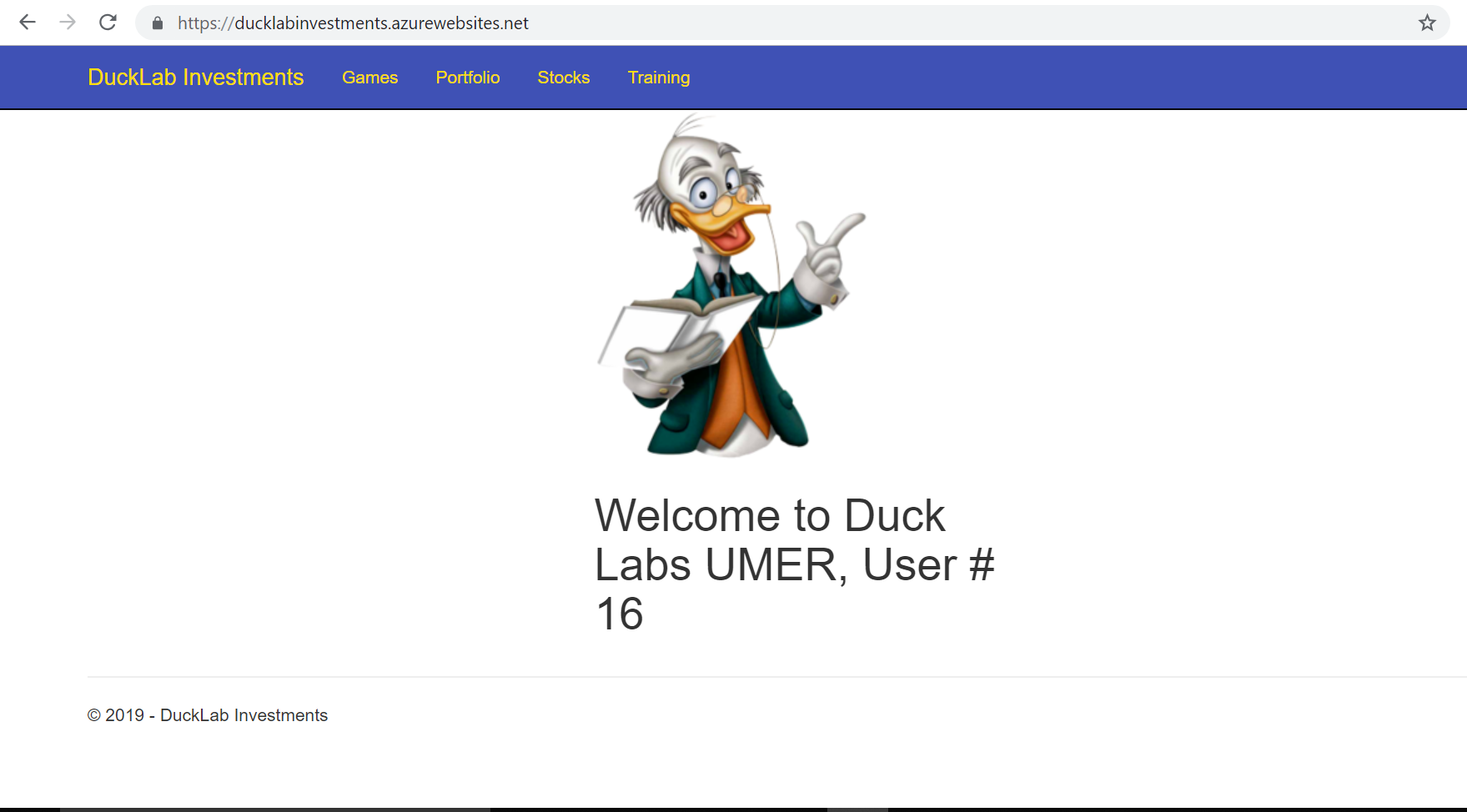
During second Demonstration, we had already completed our website functionality and Android Application. We demonstrated the key functions of our website and Android application along with Amazon Alexa skills we incorporated into our system. We displayed our hardware ticker during second demonstration.

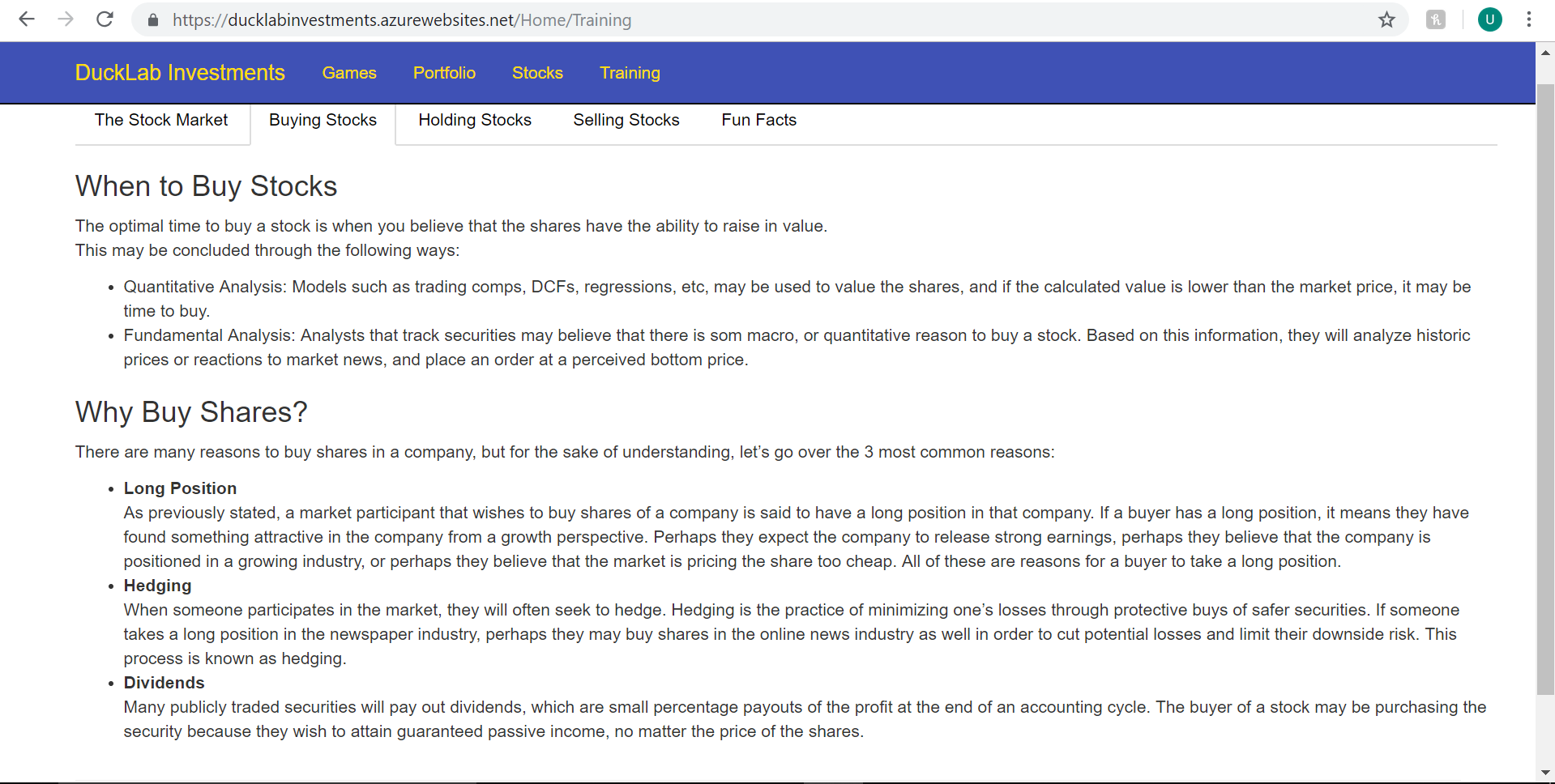
13.1 Current Status

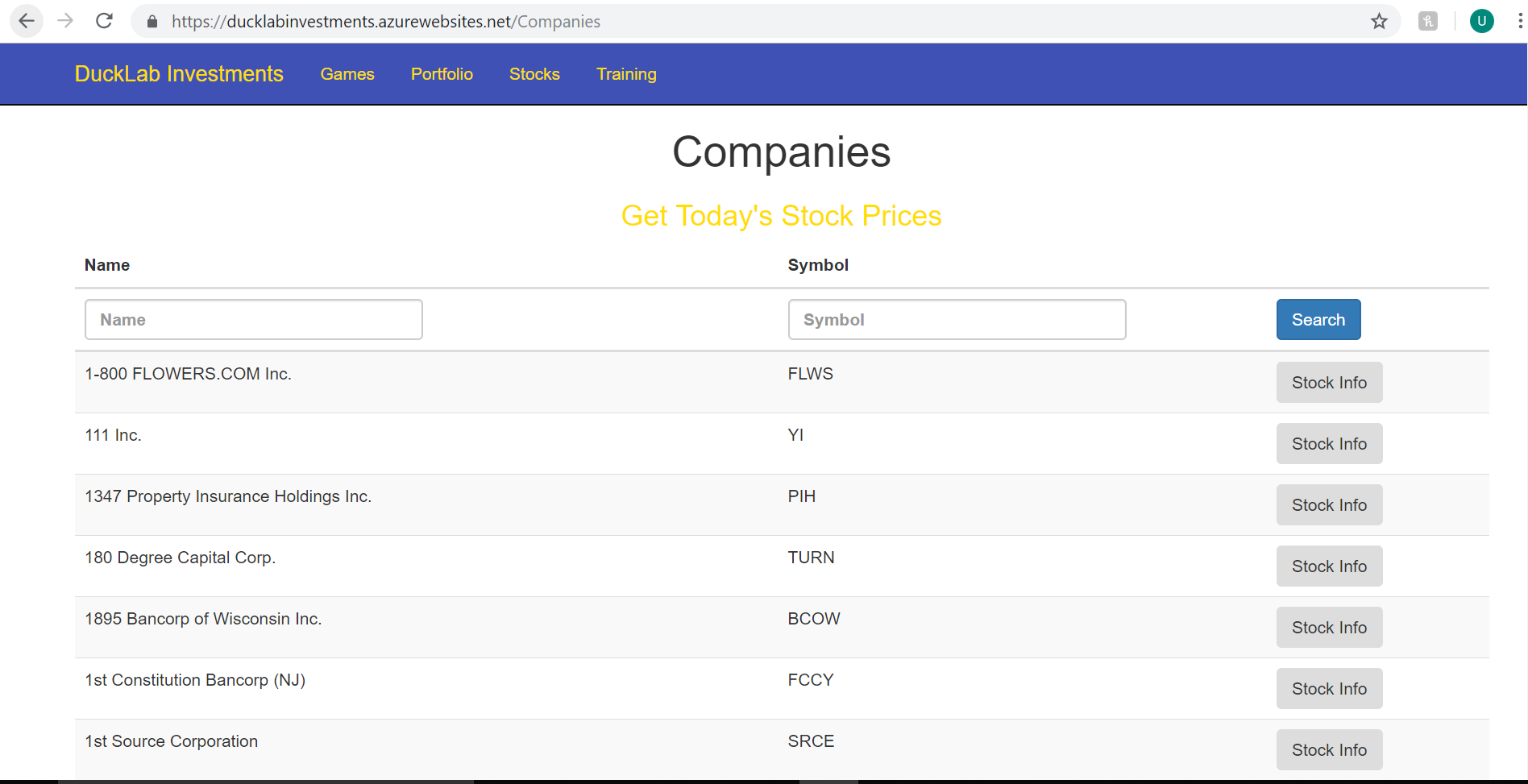
Currently, DuckLab Investments League is a functioning web application along with Android application and Alexa Integration. Our website can be accessed at [https://ducklabinvestments.azurewebsites.net](https://ducklabinvestments.azurewebsites.net/). Our website features working orders, leagues, portfolios, and user systems with a fully-functional and responsive UI that can be used in tablets and phones in addition to personal computer to be user friendly for anyone on a smart device. Our application can be downloaded at Play store under the name DuckLab Investments. Most core functionalities have been deployed and the current status is debugging and optimizing our website to address orders in an asynchronous fashion to maximize the efficiency of our application.

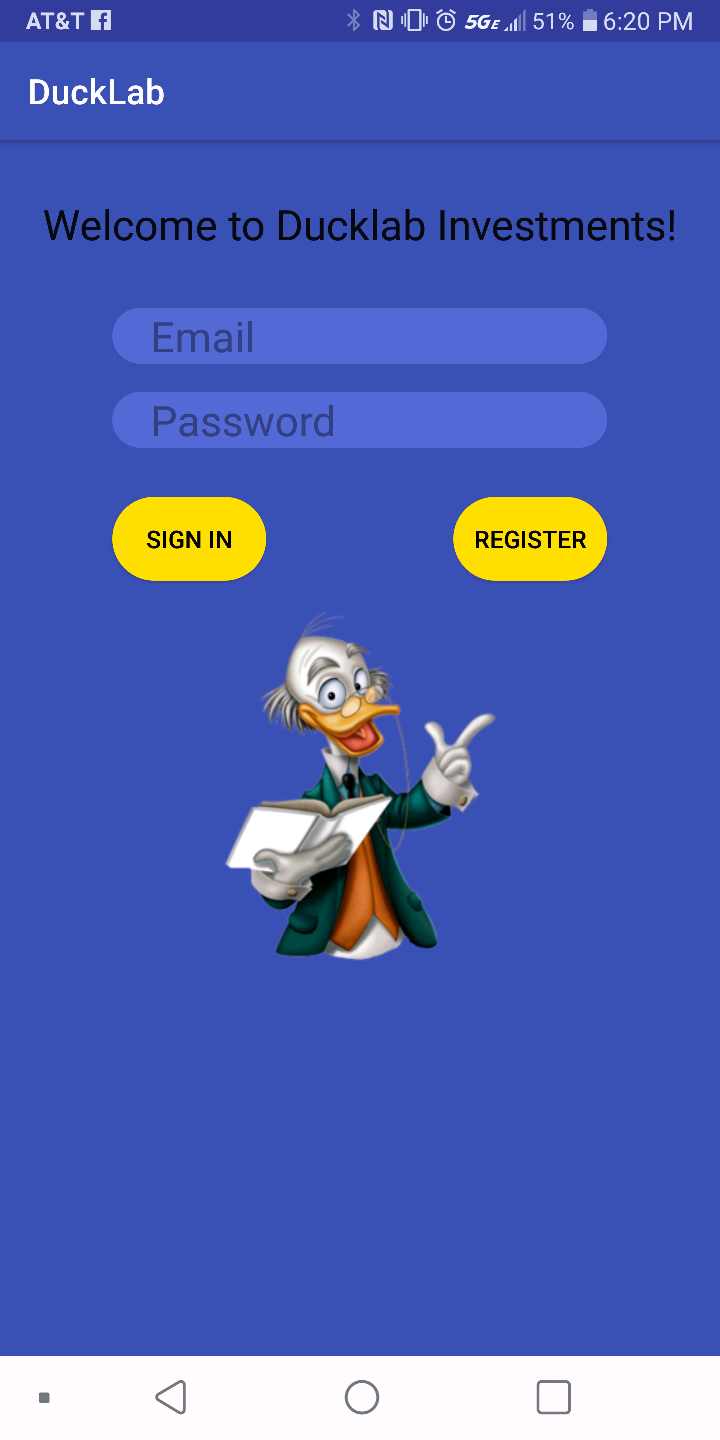
Some screen shots from our working model are displayed below.

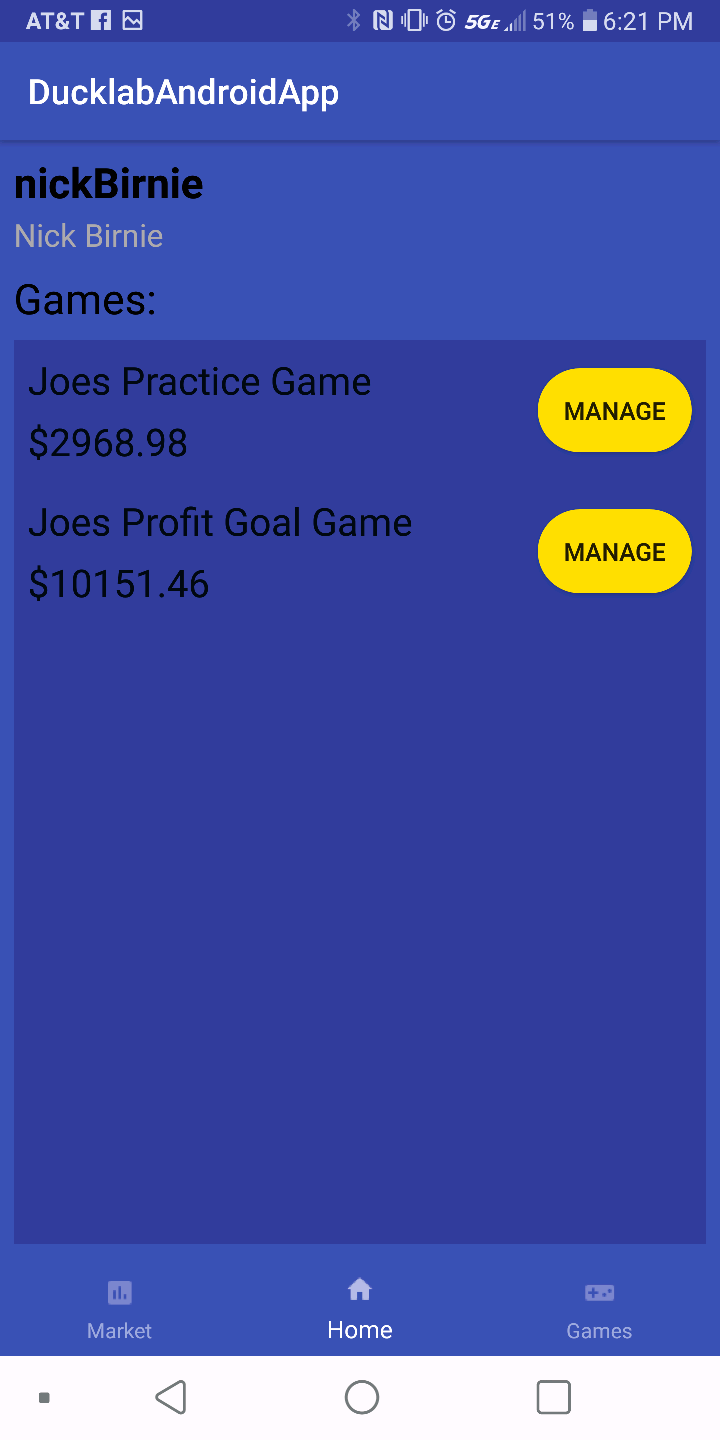


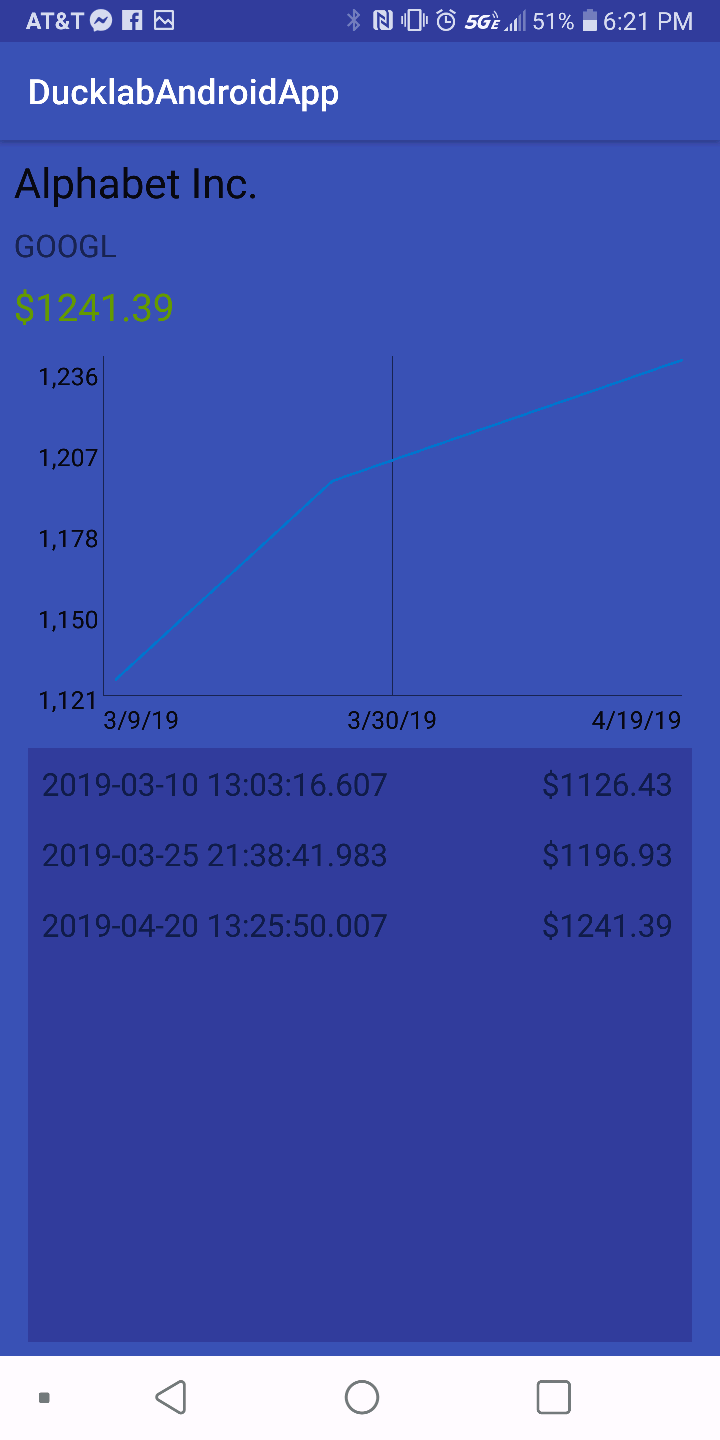


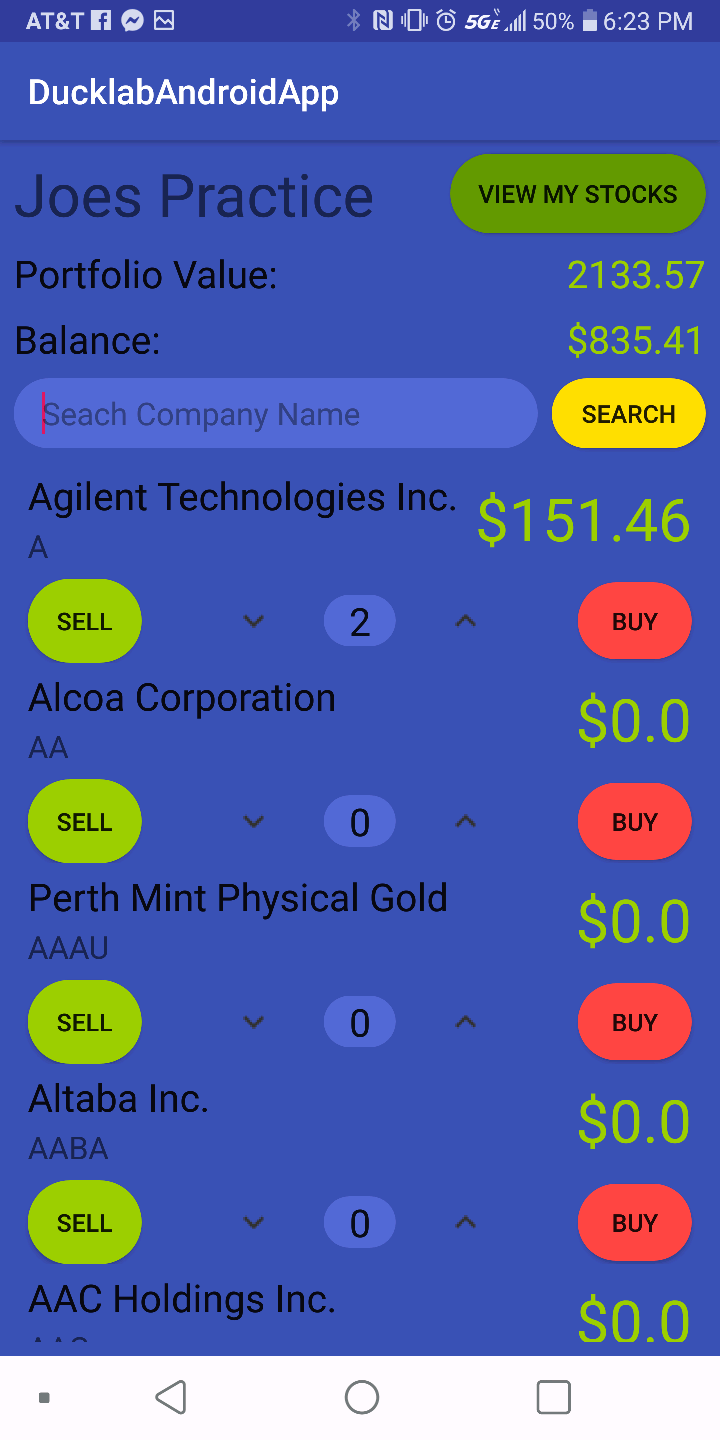












Our website can be visited at [https://ducklabinvestments.azurewebsites.net](https://ducklabinvestments.azurewebsites.net/) to checkout other important functionalities we have implemented in our system.

**13.2 Key Accomplishments**

The DuckLab Investment League has implemented the following functionalities:

* Easy registration
* Accessible across multiple platforms
* Separate Android application for user convenience
* Alexa integration for user comfort
* Different Game modes
* Real time stock data
* League creation

The DuckLab Investment League has also satisfied the following requirements:

* Req-1
* Req-2
* Req-3
* Req-4
* Req-5
* Req-6
* Req-7
* Req-8
* Req-9
* Req-10
* Req-11
* Req-12
* Req-13
* Req-14

The DuckLab Investment League also implemented the following use cases:

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

**13.3 Future Work**

As major portion of the project is already implemented and requirements for the project mentioned have been fulfilled, our focus is to implement the hardware ticker and designing of the device. This device will become the part of user home and will remind the user for being the part of DuckLab investments and keep playing.

We will also focus on polishing Alexa skills to answer complex questions asked by users and even able to do trading for user by using voice commands.

Finally, to maximize user retention, another future work goal would have been to implement social media and Ez login options so users can be reminded to check and update their portfolios and leagues on a regular basis.

**14. References**

<https://docs.microsoft.com/en-us/azure/>

<https://docs.microsoft.com/en-us/sql/?view=sql-server-2017>

<https://docs.aws.amazon.com/index.html#lang/en_us>

**http://www.tmap.net/building-blocks/test-design**

[https://docs.microsoft.com/en-us/azurehelp/](https://docs.microsoft.com/en-us/azure/)

[https://docs.microsoft.com/en-us/sqlhelpFAQ/?view=sql-server-2017](https://docs.microsoft.com/en-us/sql/?view=sql-server-2017)

[https://docs.aws.amazon.com/index.html#langlambdalang/en\_us](https://docs.aws.amazon.com/index.html#lang/en_us)

**15: Responsibility Matrix:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Section** | **Team Members** | | | | | | | |
| Keeyan Haghshenas | Umer Qazi | Joe Gordon | Dunbar Birnie | Ludwig Randazzo | Colter Mooney | Bohdan Kryzh | Ashish Motyala |
| UML Diagrams  (10 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Prose descr. of diag  (10 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Alt. solution descr.  (10 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Class Diag & descr.  (5 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Signatures  (5 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Styles  (5 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Package Diagram  (2 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Map hardware  (2 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Database  (3 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Other  (3 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Alg’s & data struct  (4 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Appearance  (6 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Prose. descr.  (5 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Testing design  (12 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Doc merge  (11 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Project coordination  (5 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Plan of work  (2 points) | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |

Contribution Calculation

Σ(Section weight)i\*(contribution percentage)i

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Keeyan Haghshenas | Umer  Qazi | Joseph Gordon | Dunbar Birnie | Ludwig Randazzo | Colter Mooney | Bohdan Kryzh | Ashish Motyala |
| 12.5 | 12.5 | 12.5 | 12.5 | 12.5 | 12.5 | 12.5 | 12.5 |