Rutgers University

School of Engineering

Department of Electrical and Computer Engineering

StockHop: The Stock Market Fantasy League Game

REPORT #3

URL: http://www.thestockhop.com

Group #2

Priyanka Kale
JakubKolodziejski
Dan Marzullo
Sam Ramezanli
Melissa Romanus
Wei Shen

Individual Contributions Breakdown

For Report 3, all team members contributed equally. This report involved collating our old reports and updating sections based on recent updates.

2. Table of Contents

3. Summary of Changes

Going further in the project helped us to understand better meaning of the earlier concepts we had in our reports. In addition, reading our classmates' comments about our report made us review our report and correct and update some parts.

The concepts of Trailing Stop and Stop-Limit orders were not previously mentioned in Reports 1 and 2. This is due to the fact that we ended up with a decent product one week early and thought about what extra order types we could add to enhance our website. If we had not finished the basic functionality early, we would not have had the time to envision and come up with this plan. Reports 1 and 2 also did not have fully-dressed descriptions and Sequence Diagrams for all of the Use Cases. Since we were able to realize all of our use cases, we needed to update the use cases and system sequence diagrams.

Several Use cases have been updated. Some of them had simple errors such as arrow directions or font variations that we notice them after reading comments. The main changes in Use Cases was use case alternate scenario and Flow of events that we gained a better understanding of after we started coding the real task. So, some lines have been added or removed from earlier report.

We also had some changes in our domain model. The central idea for the domain model remains the same. We have a central controller concept, which communicates with various other concepts. In the updated model we have two such controllers, Broker, which did most of the backend control, and UI Manager, which did most of the frontend control. A few concepts are renamed though their roles are unchanged and a few concepts are added new like controllers for each of research, ranking and preferences page. Few concepts like limit order, stop order are combined into single concept order, as during implementation we realized we could have a single concept order and the broker concept would internally handle individual orders.

A discussion on OCL was added, and we also modified the explanation of Design Patterns in our code as we learned more about them. A cohesion analysis of the backend code was added. The class diagrams had to be redone from Reports 1 and 2, since code was both added and enhanced since the first Demo. Additionally, we added the Use Case Effort Estimation section.

Bulleted Summary of Report Changes

Domain Analysis

- Domain Model
- Concepts and Attributes

Class Diagrams (Frontend and Backend)

Design Patterns

OCL Contracts

Use Cases

- Fully Dressed Descriptions
- System Sequence Diagrams
- Effort Estimation

Interaction Diagrams

User Interface

- Mockups and Explanation of Navigational Paths

4. Customer Statement of Requirements

To Whom It May Concern:

Bulls and Bills, LLC has been serving the everyday stock market investor for over 50 years. With the recent economic downturn, we have seen a decrease in the number of users on our site. We want to expand our customer base as much as possible and reach people that may have never considered the prospect of stock market trading. In the past, trading in the stock market was thought to be limited to only those with a strong financial educational background. As the internet became more and more popular, it became possible for the everyday person to become involved in stock market trading through the use of online web services. Our customer base lacks novice investors, whom we think may be too intimidated to gamble with their money. We are hoping to change all of that with a stock market investment fantasy game.

The game will simulate the basic functionality and provide some of the same features as the US stock market, but instead of real money, users of the website will be able to invest with virtual money. This virtual money should be in the amount of $100,000. The stocks in which they can invest should be real-world stocks. The site should display current information about these stocks when someone attempts to make a trade with them (we realize this may not be real-time since data from real-time stock market information is expensive and we do not have the licensing to share ours with you). We do not want to provide stock forecasting, since that is something we offer on our real website and costs money.

Each user should be able to have his or her own account. These accounts should be secure. When a user logs in to the site, they should have access to their portfolio. The portfolio should contain all of the stocks that they have purchased. It is also important for users to access their transaction history, so they can keep track of where their money is going. An open transactions page will also be important to show users that have placed stop or limit orders that their transactions are still valid. Open transactions also include those orders placed after the close of the stock market. Please remember the stock market hours are Monday through Friday from 9:30AM to 4:00PM. Remember that it is also closed on Holidays such as Thanksgiving and the day after Christmas. It is a requirement to only let users trade during those times, however, we are flexible if you want to implement Pre-Market Trading or After-Hours Trading. Please notify our office before implementing these in the game.

We would like this to be a website that is as user-friendly and informative as possible. There are many existing stock market simulators on the market today. We want to set ourselves apart with superior tutorials and accessibility. It is important that it is easy for users to both learn about the stock market and easily use the website. It should have simplistic style pages but also equip users with most of the functionality they would be able to get from the real market. Since it is a game, we also want to

make this a competition. Users should be able to view their rank in the competition that is based off of their net worth. We hope to attract new customers this way as well as keep our old customers loyal to the company even if they do not currently have the money to invest in the real market.

When a user makes a trade, we would like them to have a single page in which they can buy and sell. Please make sure to implement market, limit, and stop orders. Any additional orders you add are not required but definitely help us set ourselves apart (and may be attached to some Achievement Bonuses for your team). Users should be able to enter the symbol of the company that they want to purchase, the quantity of shares, whether they want to Buy or Sell, and if the transaction should be good for the day or good until cancelled (i.e. for stop/limit orders). The system should confirm this amount with the user before executing the order.

We would like to make stock research as easy as possible for the user. They should be able to view the stock's performance over the course of the day, as well as information like the price of the last trade, time of the last trade, change, previous close price, opening price, bid and asking price, the volume, and the market cap. They should also be able to view the stock information for the company they select from the trade page.

We would like the site to offer advertising, so that we can make money from advertisers, and also so that we can advertise our real investment website. The advertising interface will need to have a private login. An advertiser's login should have a clean and simple interface where they can Add a New Ad. The ad and edit functions should be able to take uploaded pictures and text. When the advertiser creates a new ad, they should be able to choose a duration and will be billed for this duration. After this duration is up, the ad should be removed from the page. We would like to have any ads that go through this interface allowed to be deleted by a future system administrator at our company, so please allow a manual delete by someone with privileged access.

We did an independent survey for our own investment website and found that people preferred the site to update them on current business news. The approved company source for this news is CNN. We would like this to be displayed somewhere on the game's site. If users want to add their own favorites to this feed, that will be okay.

In order to appeal to a younger market, we are also hoping to send out text messages or emails to customers that elect to be messaged whenever a trade is processed or if they have set an alert for a stock. We would like you to have the entire website ready to launch by December 2011. We look forward to working together in the future.

Sincerely,

Frank “Red” Johnson

Requirements from Red’s Letter

REQ1	**System shall be a stock market fantasy game.**
REQ2	**System shall have separate user accounts and will allow users to change preferences.**
REQ3	**System shall give user $100,000 virtual money.**
REQ4	**System shall allow users to virtually buy and sell stock.** **System shall allow users to choose the type of orders (Market, Limit, Stop, Stop-limit, Trailing stop) and the order duration (Good for Day, Good Until Canceled)**
REQ5	**System shall provide tutorials that explain** **a. How to Use the Website** **b. Basics of Investing**
REQ6	**System shall allow companies and individuals to purchase advertising space on the website.**
REQ7	**System shall contain portfolio, transaction history, and pending transactions.**
REQ8	**System shall obtain stock data from a given website.**
REQ9	**System shall obtain stock news from a given website.**
REQ10	**System shall allow the user to ask for notification via email or text message.**

Note that there are more requirements from Red’s Letter, but these are the most important.

5. Glossary of Terms

• Actors – External players of a system.

• Attributes - properties of concepts, are usually for storage/accounting purposes or state information

• Domain Model – a conceptual model of a system or piece of software

• FURPS+ – an acronym meaning Functionality, Usability, Reliability, Performance, and Reliability, which describe nonfunctional requirements

• Gantt Chart- project planning chart that shows time and the duration of tasks

• MySQL – an open-source database system

• Portfolio –The collection of all stocks that a user owns and their current value

• System Administrator –Responsible for maintaining the server, including setup, security, user accounts, etc.

• UML – (Unified Modeling language) - Software Engineering modeling language

• Use Case – shows a certain scenario of system and the actors that take place in it

• User - Any individual visiting the website

6. Functional Requirements Specification

- Stakeholders
    a. Bulls and Bills, LLC – The customer and sponsor of this project is interested in this project to attract customers to their company and to make revenue off of the game website.
    b. Advertisers – Other financial institutions may want to advertise on the website, but really any appropriate advertisements are approved. Therefore, potential advertisers can come from a variety of backgrounds (e.g. software, job websites, investment websites, banks, etc.).
    c. Users
        - Teachers and Students – Educators can use the website to assist with a business or economics course.
        - Novice Users – Novice users will need the most guidance through the tutorials.
        - Intermediate/Expert Users – These users may be focused in riskier portfolios and also in being at the top of the rankings to increase their chances of winning prizes.
        - Investors – Investors may be interested in how efficiently this website models the real stock market.
    d. Internet game fanatics – Some people love competition and love to play games. These users of the website are interested in winning monthly and yearly games.
    e. Development Team – The website developers are concerned with developing a reliable product in the given amount of time.
    f. System Administrator – The system administrator is concerned with the maintainability and security of this website.
    g. Testers – The site testers, a group that will be able to use the site in demo mode before it goes to full production, will be able to give feedback.

- Actors and Goals

- **Actor 1: New Player**
  i. **Type:** Initiating actor
  ii. **Goals:**
    - To create new account.
    - Learn about game.
    - Play the game.

- **Actor 2: User**
  i. **Type:** Initiating actor
  ii. **Goals:**
    - Access his/her account (Login).
    - View cash balance and transaction history at Homepage.
    - Buy a specified number of shares of a company's stock.
    - Sell a specified number of shares of a company's stock.
    - View current and past information about the price at Research section.
    - View a user's stock account performance relative to other users' performance.
    - View Transaction History
    - View Portfolio
    - Edit Preferences

- **Actor 3: Advertiser**
  i. **Type:** Initiating actor
  ii. **Goals:**
    - Access his/her account (Login).
    - Manage advertisements on the website.
    - Attract users to find out about his/her service.

- **Actor 4: Database**
  i. **Type:** Participating actor
  ii. **Goals:**
    - Information is saved in database for new User.
    - Data is being checked to prevent duplicating for usernames.
    - Transaction History is saved and kept in database.
    - Users' portfolio is saved and kept in database
    - Stock market updates are saved in database.

- Information is deleted when there is a request from System.

- **Actor 5: Email Server**
  i. **Type:** Participating Actor
  ii. **Goals:**
    - Send email to contacts when it receives request from System.

- **Actor 6:Yahoo Finance**
  i. **Type:** Participating
  ii. **Goals:**
    - Provide up to date information regarding Stock market.
    - Whenever there is a request, information is sent to System.

- **Actor 7: Timer (now Poll-er)**
  i. **Type:** Initiating
  ii. **Goals:**
    - Prompt system for requesting new Stock market data from Yahoo Finance.
    - Since Report 1, the timer has evolved into a Poll-er. Rather than periodically request Yahoo data at fixed intervals (Timer mode), the backend software does not actually request any data from Yahoo! Finance until there are orders in the Open Orders database. This reduces the number of calls to Yahoo! Finance.
- **Actor 8: CNN Money Markets RSS Feed**
  i. **Type:** Participating
  ii. **Goals:**
    - Provide up to date information regarding stock market news.
    - Data is requested each time a new page is loaded and information is sent to the System.

- Use cases

i. Casual Description

UC-1: Registration: All visitors to website are able to create free accounts by choosing user name and password and giving some basic information of themselves such as birthday.

UC-2:Sign in: A user is able to sign in to his/her profile from the login page. Once logged in to an account, the user can then access details about the account and make changes.

UC-3: View Home Page: A user is able to view portfolio summary (best/worst stocks), cash balance, overall rank, and other personalized information from profile's homepage.

UC-4: Buy Stocks: Users can search for available stocks as well as select a type of stock order and the number of shares they wish to purchase. If the user has enough cash to cover the cost, the order can be placed, and will then be added to the pending transaction list.

UC-5: Sell Stocks: A user can sell his/her owned shares of stock by selecting the company, number of shares, and type of order. The order will then be added to the pending transaction list.

UC-6: Research Stocks: Users can request and view information about a company's current stock price and price history. By entering a company's stock symbol, the user will be presented with the company's stock information.

UC-7: Send Notifications: The user can receive an E-Mail or SMS message when an important event happens in an account. When a pending buy or sell order is completed or cancelled, a notification will be sent to user.

UC-8: View Rankings: Users can view their ranking in terms of investment performance among other users of the system. The user can view the top ranking players as well as sort the list based on a variety of criteria.

UC-9: User sign out: A user can click the "log out" button to restrict access to their account by unauthorized users. After logging out, a user will need to reenter a valid username and password pair to regain access to the account.

UC-10: View Help: The user can access a variety of documentation about how to use the Stock Hop site as well as basic tutorials on the stock market.

UC-11: View Transaction History: Users can view all of the previous transactions on an account. All completed orders will appear with the details of the transaction.

UC-12: View Pending Transactions: Users can view all of the pending transactions in an account. If an order has yet to be completed or is submitted during off-hours, it will be viewable on the pending transaction page

UC-13: View Portfolio: A user can view the portfolio associated with an account. By navigating to the portfolio page, the user can see the stocks that are owned and details about them such as the number of owned shares and the profits of each.

UC-14: User Preferences: A user can change various settings about his or her account by navigating to the preferences page. There a user can set up E-Mail and SMS messages, change the preferred email contact, change his or her password, or delete the account entirely.

UC-15: Manage Advertisements: The advertiser can log in to the website, add advertisements, and remove their advertisements.

UC-16: Maintain Website: The system administrator can access the backend for periodic maintenance and upgrades. A team of maintenance coders can add or remove features or bugs as needed.

UC-17: Poll for Stock Prices: The Poll-er will query Yahoo! Finance in order to execute orders when there is data in the Open Orders Database table.

UC-18: View Current Stock News: The user can view the current stock market news as obtained by CNN Money Markets at the top of each page. This information is exchanged from the external source CNN every time a page is loaded.

The login use cases are not usually shown as use cases in software engineering development, because a user does not want (or request) to login. The reason that it is included in this report is that the login functionality does interact with the database, the user's browser, and the system. Additionally, the way StockHop is written, a user only sees a registration page and login option when he or she visits the site (user cannot browse the site without logging in). This makes login a necessary step in being able to view the features of the site, and thus, facilitates its need to be included in the Use Cases.

The team was able to implement all of the use cases that came up with from the proposal. After completing these, the team came up with some ideas for future work. These ideas for future work are shown in the Conclusions section. More Use Cases would be developed as the future work ideas were developed.

ii. Fully Dressed Descriptions

The following in depth use cases reflect those features that will be ready by Demo 1.

Use Case UC-1: Registration

- Initiating Actor: New User or Advertiser
- Participating Actors: Database
- Actor's Goal: Create an account
- Pre-condition: having a computer, which is connected to Internet.
- Post-condition: An account is created and user is able to login to the system.
- Flow of Events for main success scenario:
  - → User goes to start page of our website.
  - ← System asks for username and password.
  - → User choose arbitrary username and password
  - ← System request information from Database.
  - → Database send information.
  - ← System check information to see whether this username is already taken or not.
  - → User gives additional information such as Birthday and sex**.**
  - ← System save the information in database.
  - ← System show confirmation note for the user.
- Flow of Events for Extensions (Alternate Scenario):
  - → User enter a used username.
    ← System request information from Database.
    → Database sends information to System.
    ← System verify and compare information.
    ← System would return to main page and shows message: "This username is taken"

Use Case UC-2: User Sign In

- Initiating Actor: User
- Participating Actors: Database

- Actor's Goal: To login into his/her account.
- Pre-condition: User or advertiser has a previously established account.
- Post Condition: The system displays the user's personalized home page.
- Flow of Events for main success scenario:
    - → User visits StockHop webpage.
    - ← System prompts user to enter username and password or to create a new account.
    - → User enters valid username and password pair.
    - ← System authenticates username and password with **database.**
    - ← System login User and take User to Home page.
- Flow of Events for Extensions (Alternate Scenario):
    - ← System detects invalid username and password with **database**
    - ← System informs **user** of the error.
    - ← System prompts the **user** to try entering information again or to create a new account.

Use Case UC-3: View Homepage

- Initiating Actor: Users
- Participating Actors: Yahoo Finance, Database
- Actor's Goal: To view portfolios, cash balance, transaction history and other personalized information.
- Pre-condition: User already is registered and has a profile.
- Post-condition: none worth mentioning.
- Flow of Events for main success scenario:
    - → User login to profile.
    - ← System request information from Yahoo Finance.
    - → Yahoo Finance sends information to System.
    - ← System compare the incoming information with User's portfolio
    - ← System updates Database.
    - ← System displays Homepage.
    - → User is informed about recent changes and update in his/her account.

Use Case UC-4: Buy Stocks

- Initiating Actor: Users

- Participating Actors: Database, Yahoo Finance
- Actor's Goal: To buy a specified number of shares of a company's stock
- Pre-condition: User is logged in.
- Post-condition: (1) The purchased number of shares appears in the user's account. (2) The purchase price is deducted from the user's available cash
- Flow of Events for main success scenario:
    - → User request buy section of the StockHop webpage.
    - ← System prompts user to enter stock name/symbol and amount of stocks.
    - → User enters valid stock name/symbol.
    - ← System queries Yahoo Finance for current price per share of the entered stock symbol.
    - → Yahoo Finance finds a match for the stock name/symbol and return the price.
    - ← System compute total cost and add the commission cost to it.
    - ← System verify that total cost plus commission is less than user's cash balance.
    - ← System asks user to enter type of transaction (Market, Limit, Stop, Trailing Stop, Stop-Limit).
    - → User choose type of price he/she wish to have.
    - ← System send the user's request to Yahoo Finance.
    - ← System send information to database and update user's portfolio.
    - ← System notifies user that the transaction has been completed.
- Flow of events for Extensions (Alternate Scenarios) :

    1) → User enters invalid stock symbol

    ← System queries Yahoo Finance for current price per share of the entered stock symbol.

    → System notifies the User that the stock symbol is invalid.

    →System prompts for the User to enter a different stock symbol.

    2) → System determines that the User does not have enough cash to make the desired purchase.

    → **System** notifies the User that there is not enough cash to make the purchase.

    → System prompts the User to enter new purchase information.

Use Case UC-5 : Sell Stocks

- Initiating Actor: Users and Advertisers
- Participating Actors: Database, Yahoo Finance, Email server.
- Actor's Goal: Sell stocks to investors
- Pre-condition: User is logged in, User owns at least one amount of share.
- Post-condition: Transaction is done and user's portfolio is updated with stock removed from it.
- Flow of Events for main success scenario:
    - → User request sell section of the StockHop webpage.
    - ← System prompts user to enter stock name/symbol and amount of stocks.
    - → User enters valid stock name/symbol and amount of stocks he wish to sell.
    - ← System request database for user's portfolio information.
    - → Database sends information.
    - ← System verifies that user own the stock and the amounts of share he wish to sell.
    - ← System queries Yahoo Finance for current price per share of the entered stock symbol.
    - → Yahoo Finance finds a match for the stock name/symbol and return the price.
    - ← System compute total income and subtracts the commission cost from it.
    - ← System asks user to enter type of price (Market, Limit, Stop, Trailing Stop, Stop-Limit).
    - → User choose type of price he/she wish to have.
    - ← System sends the offer to Yahoo Finance.
    - ← System sends information to database to update user's portfolio and removes sold shares from portfolio and.
    - → Database confirms changes to System.
    - ← System notifies user that the transaction has been completed.
- Flow of Events for Extensions (Alternate Scenario):

    1. → User enters wrong amount of share (User does not have enough money to buy these amount of shares).

       ← System check database and asks user to enter true amount of money.

    2. → our share will not be bought by the Stock investors because of high price or type of share.

    ← System sends user email that transaction is not done.

We received feedback that we should break up the Buy and Sell use cases into separate use cases based on each type of transaction. The team considered this, but ultimately decided to expand upon the details within each Buy and Sell Use Case. We chose to do this because the same general execution flow is the same. The only changes made would be on how the system handles the type of order on the backend. These are more logic-based if/else statements and did not pertain to an overall difference in functionality.

Use Case UC-6: Research Stocks

- Initiating Actor: Users
- Participating Actors: Yahoo Finance.
- Actor's Goal: To view current and past information about the price of a company's stock.
- Pre-condition: User or advertiser has a previously established account.
- Post-condition: The system will display information about the current and historical prices of a company's stock to the user.
- Flow of Events for main success scenario:
  - → User request the research section of the webpage
  - ← System prompts user to enter stock symbol
  - → User enters valid stock symbol
  - ← System queries Yahoo Finance for current price per share of the entered stock symbol as well as any available history about the stock.
  - → Yahoo Finance (a) returns requested information to the System
  - ← System (a) displays retrieved information to the User (b) provides an area for the User to enter another stock symbol to look up another stock
- Flow of Events for Extensions (Alternate Scenario):
  - → User enter invalid stock symbol.
  - ← System queries Yahoo Finance for current price per share of the entered stock symbol as well as any available history about the stock
  - → Yahoo Finance notifies the System that the stock symbol is invalid
  - ← System notifies the User that the stock symbol is invalid and
  - ← System prompts User to enter a different stock symbol.

Use Case UC-7 : Send Notifications

- Initiating Actor: Poll-er
- Participating Actors: User, Advertiser, Database and Yahoo Finance.
- Actor's Goal: Send notifications in order to inform profile owner about events happened in the profile
- Pre-condition: User has an account, User has activity in his/her account., User has chosen to receive SMS or email for notification.
- Post-condition: Email would be sent to User successfully.
- Flow of Events for main success scenario:
    - → Timer prompts system to check for new event.
    - ← System request information from Yahoo Finance.
    - → Yahoo Finance sends information to system.
    - ← System notice an important new event.
    - ← System signals email server (or SMS server) for sending notification to User.
    - → User checks his/her email/mobile and will aware of what has happened.
- Flow of Events for Extensions (Alternate Scenario):
    - → User enters wrong email address.
    - ← System sends email to wrong address

Use Case UC-8 : View Rankings

- Initiating Actor: User
- Participating Actors: User System, Database
- Actor's Goal: To view a user's stock account performance relative to other users' performance
- Pre-condition: 1) User or advertiser has a previously established account 2) User is currently logged into an account
- Post-condition: Email The system will display information about the user's ranking based on certain criteria
- Flow of Events for main success scenario:
    - → User visits the rankings section of the webpage.
    - ← System requests account information from the Account Database and sorts users by certain criteria
    - ← System signals a) displays a default view of the User's ranking and (b) provides clickable options to the User to alter the display or view top ranking users.

Use Case UC-9: User Sign Out

- Initiating Actor: User
- Participating Actors: System, Database.
- Actor's Goal: To protect account from unauthorized access and to delete account.
- Pre-condition: 1) User or advertiser has a previously established account 2) User is currently logged into an account
- Post-condition: The user's account is not immediately accessible from the current computer
- Flow of Events for main success scenario:
  - → User is logged into an account on the webpage.
  - → User clicks the "log out" button on the webpage
  - ← System sends all unsaved data to the Database for storage
  - ← System (a) displays "successfully logged out" message to user and (b) disables immediate continued access of the user's account.

Use Case UC-11: View Transaction History

- Initiating Actor: User
- Participating Actors: Yahoo Finance, Database, User.
- Actor's Goal: to view all of the previous transactions on an account and all completed and pending orders will appear with the details of the transaction.
- Pre-condition: User should have at least one transaction since he/she has joined the website
- Post-condition: User is aware of history of transactions and their current status.
- Flow of Events for main success scenario:
  - → User Login.
  - → User choose transaction history from the Homepage.
  - ← System request information from Yahoo Finance
  - → Yahoo Finance sends current market updates.
  - ← System sends new updates to database and request database for user's transaction information
  - → Database sends information
  - ← System verify information from Yahoo finance and Database.
  - ← System signal User about his transaction history.

Use Case UC-12: Help

- Initiating Actor: User
- Participating Actors: Database
- Actor's Goal: To learn about how the system as a whole or a particular function of the system works
- Pre-condition: (1) User or advertiser has a previously established account (2) User is currently logged into an account
- Post-condition: The user is more informed about how to use the system.
- Flow of Events for main success scenario:
  - → User is logged into an account on the website
  - → User navigates to the "Help" section on the webpage
  - ← System displays all clickable help topics to the User
  - → User clicks to request a help topic from the System
  - ← System sends new data to database.
  - ← System the help content for the requested topic as well as an option to return to the previous list of help topics.

Use Case UC-13: View Portfolio

- Initiating Actor: User
- Participating Actors: Yahoo Finance, Database, and User.
- Actor's Goal: A user can see the stocks that are owned and details about them such as the number of owned shares, profits of each, current price and recent changes.
- Pre-condition: User should have at least one share since he/she has joined the website
- Post-condition: User is aware of history of his/her stock values and information.
- Flow of Events for main success scenario:
  - → User Login.
  - → User choose View Portfolio from the Homepage.
  - ← System request database for user's transaction information
  - → Database sends information
  - ← System request information from Yahoo Finance
  - → Yahoo Finance sends current market updates.
  - ← System verify information
  - ← System signal User about his transaction history.
- Flow of Events for Extensions (Alternate Scenario):
  - **User** does not have any shares in portfolio:
    Portfolio is shown with zero number of shares.

Use Case UC-14 : Preferences

- Initiating Actor: User
- Participating Actors: Database, Email Server
- Actor's Goal: Delete account, change email, change password.
- Pre-condition: User has an account.
- Post-condition: Profile preference will be changed.
- Flow of Events for main success scenario:
  - → User goes to preference page and select the setting he/she wish to change.
  - ← System asks for new data.
  - → User enter new data.
  - ← System sends new data to database.
  - ← System prompt email server to notify the user of successful operation.
  - → Email server sends user notification about operation.

Use Case UC-15: Manage Advertisements

- Initiating Actor: Advertiser, System Admin
- Participating Actors: Database, email server, Users.
- Actor's Goal: display and manage advertisements on the website.
- Pre-condition: Advertiser is logged in. Advertiser has the right from webpage instructors to post his advertisements on user profiles.
- Post-condition: Advertisements will be posted on Users profiles. Every time any user clicks on one of the links, amount of money would be sent to advertiser's bank account.
- Flow of Events for main success scenario:
  - → Advertiser log in to his/her profile.
  - → Advertiser choose a picture (or motion picture) for his product.
  - ← System add the picture to database and post the picture on advertisement section on the website.
- Flow of Events for Extensions (Alternate Scenario):
  1. → agreements between user and web instructor is expired.
     ← System would delete the pictures from the website.
  2. → Advertiser post inappropriate picture on the website!
     ← System (web instructor) would delete the advertisement from the website.

Use Case UC-16: Maintain Website

- Initiating Actor: System Administrator
- Participating Actors: System Administrator, Email Servers, Database, User
- Actor's Goal: Perform maintenance on the website, including adding/deleting user
- Pre-condition: Varies. System Administrator can modify or delete user accounts
- Post-condition: Varies. For user deletion, Database and Email Servers remove user from it.
- Special Note: Maintain website is a broad use case. For brevity, I have shown the steps for removing a user. Maintaining a website may vary over time. Group 2 thought about this, and decided that perhaps this use case could be broken up if more maintenance activities were needed.
- Flow of Events for main success scenario:
    - → System Administrator connects to Database and Email Servers.
    - → System Administrator selects a particular User from Database and deletes it.
    - ← System removes User from Database.
    - → System confirms User Deletion.

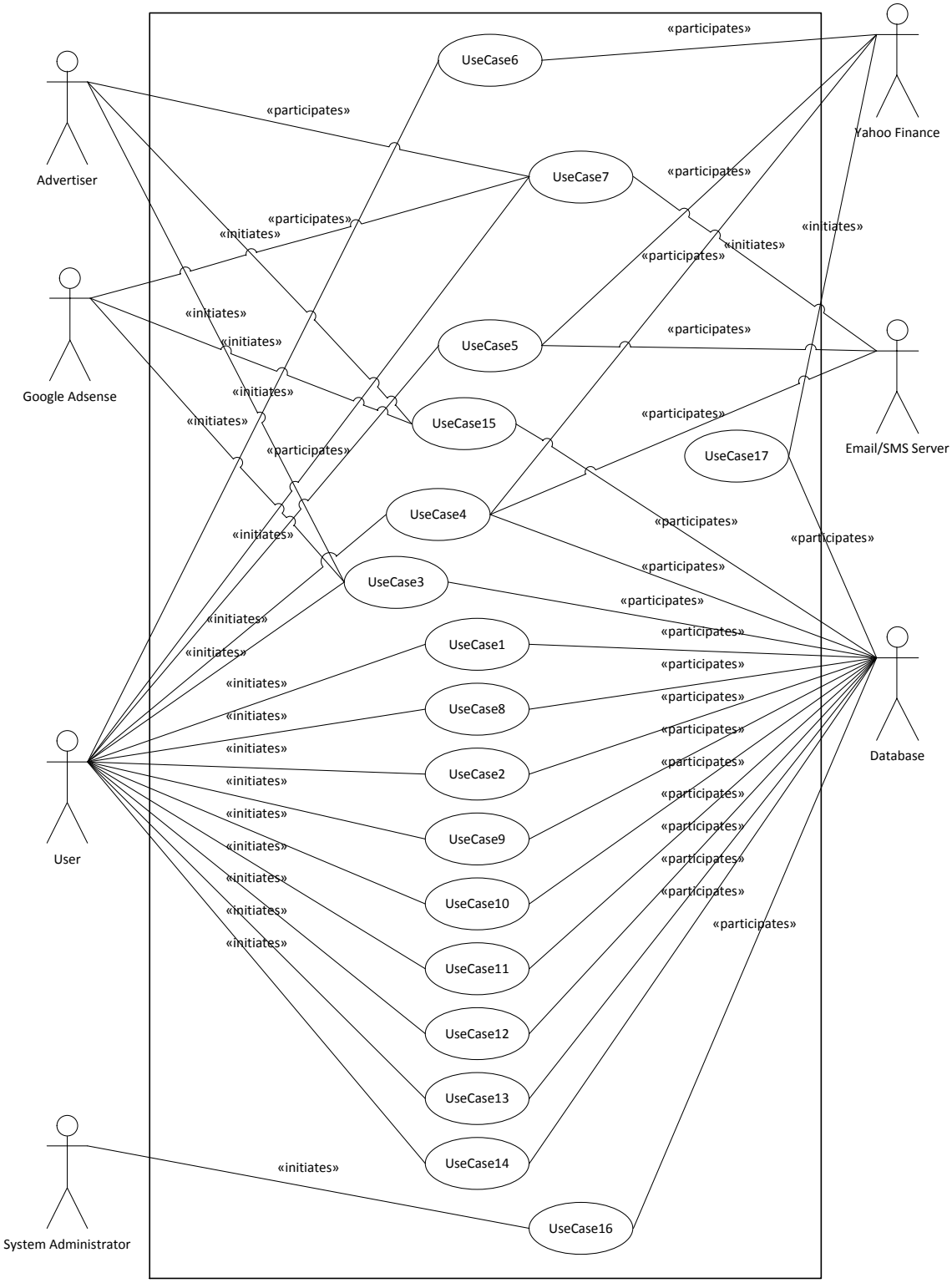
Use Case UC-17: Poll for Stock Prices

- Initiating Actor: The Poll
- Participating Actors: Database, Yahoo Finance.
- Actor's Goal: The Poller will query Yahoo! Finance in order to execute orders when there is data in the Open Orders Database table.
- Pre-condition: There should be at least one order in Open Order Database.
- Post-condition: Order will be executed by Yahoo Finance
- Flow of Events for main success scenario:
  - → Poll sends request to the system.
  - ← System ask database to send current list of orders.
  - → Database sends requested information.
  - ← System verify information from database to see whether there are any orders.
  - ← System send request to Yahoo Finance to execute current Orders.
- Flow of Events for Extensions (Alternate Scenario):
  1. There are no Orders to be executed.

Use Case UC-18: View Current Stock News

- Initiating Actor: User
- Participating Actors: CNN Money Market RSS Feed
- Actor's Goal: User has an access to latest Stock news by logging in to his/her profile.
- Pre-condition: User should have an account in Stock Hop.
- Post-condition: User is aware of latest Stock news.
- Flow of Events for main success scenario:
  - → User choose Dashboard from the Homepage.
  - ← System request CNN Money Market RSS Feed latest stock news.
  - → CNN Money Market RSS Feed sends current stock news and updates.
  - ← System display latest Stock news on the Dashboard section of the profile.

iii. Use Case Diagram



iv. System Requirements – Use Case Traceability Matrix

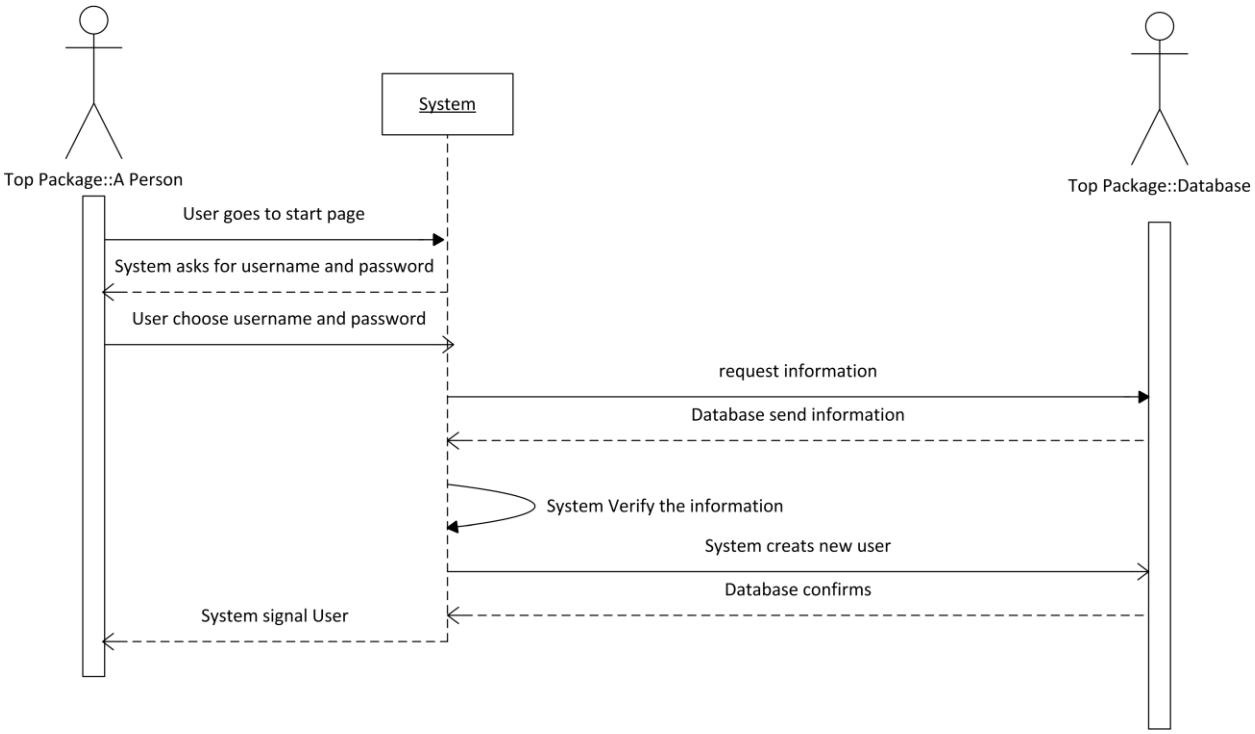
Requirement Number	Requirement	Use Case Traceability
1	**System shall be a stock market fantasy game.**	All Use Cases
2	**System shall have separate user accounts and will allow users to change preferences.**	1,2,9,14
3	**System shall give user $100,000 virtual money.**	1,3,4,5
4	**System shall allow users to virtually buy and sell stock.** **System shall allow users to choose the type of orders (Market, Limit, Stop, Stop-limit, Trailing stop) and the order duration (Good for Day, Good Until Canceled)** **System shall let the user to ask for notification**	4,5
5	**System shall provide tutorials that explain** **How to Use the Website** **Basics of Investing**	10

6	**System shall allow companies and individuals to purchase advertising space on the website.**	15
7	**System shall contain portfolio, transaction history, and pending transactions.**	13,12,11,10
8	**System shall obtain stock data from a given website.**	17

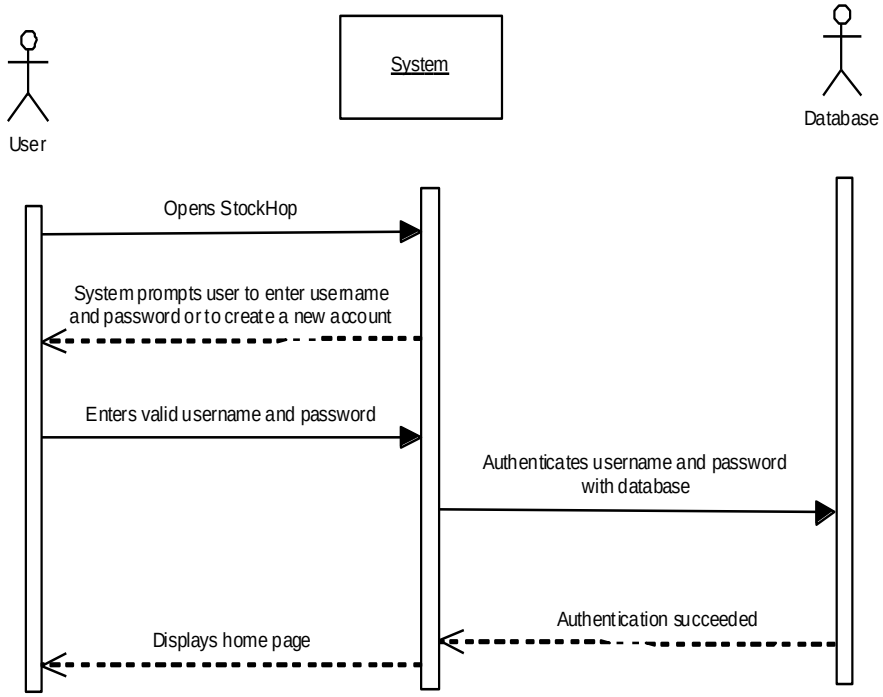
d. System Sequence Diagrams

These diagrams are explained in the above Use Case descriptions. Please see the above descriptions for in depth explanations of each diagram. They are organized by Use Case, where UC-1 corresponds to Use Case 1, etc. This was done in the interest of not being repetitive.

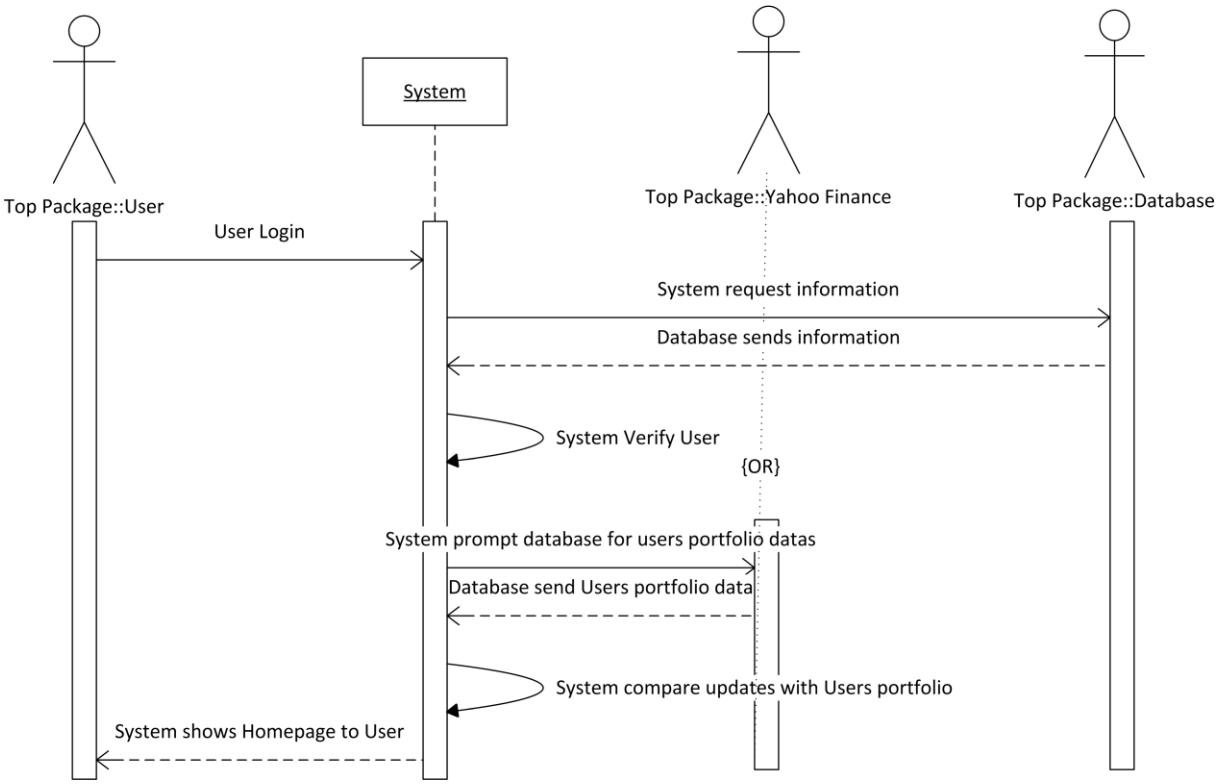
UC-1 Registration:



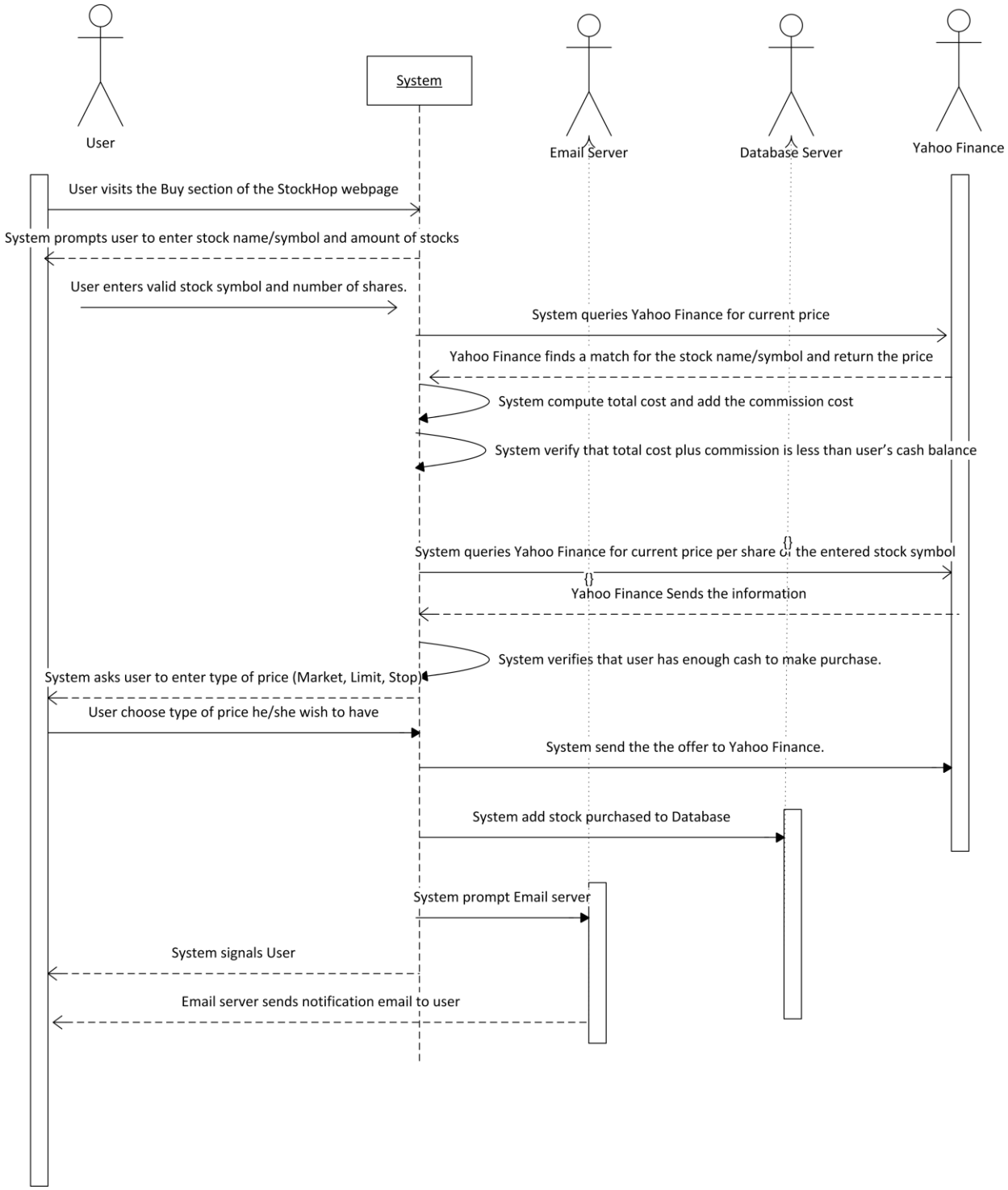
UC-2: Login



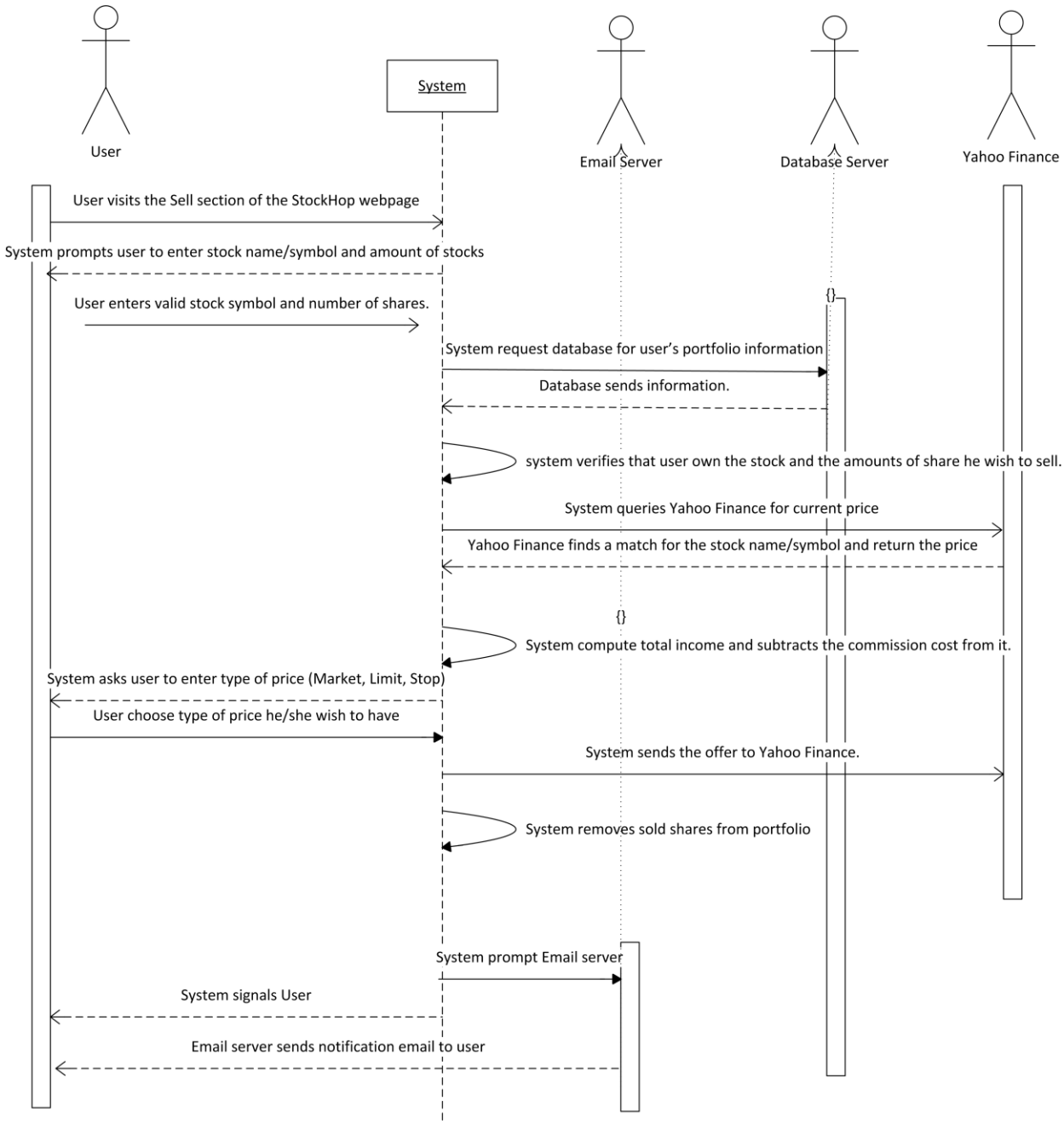
UC-3 View Homepage:



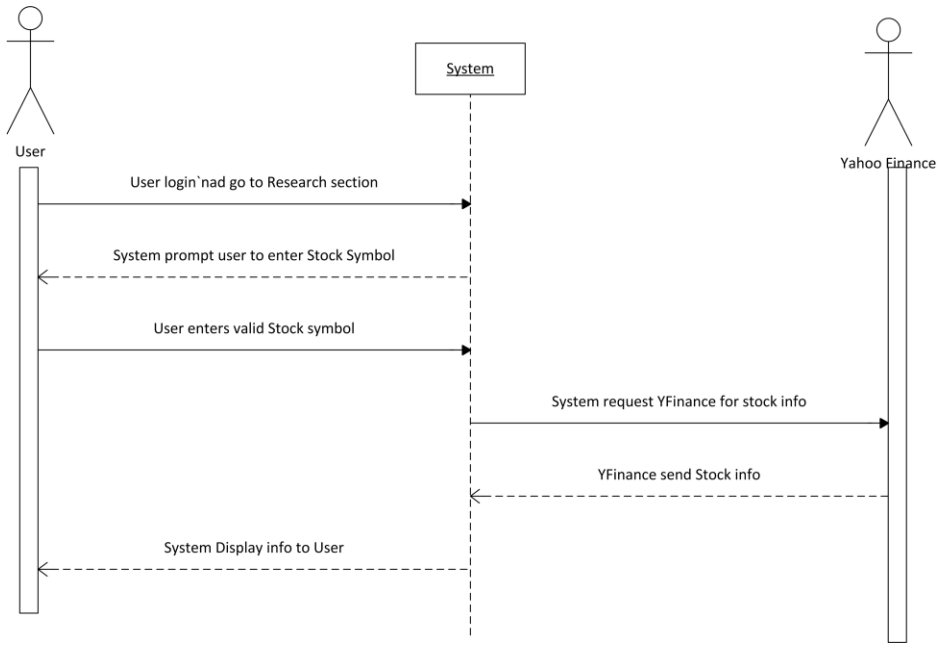
UC-4 Buy Stock:



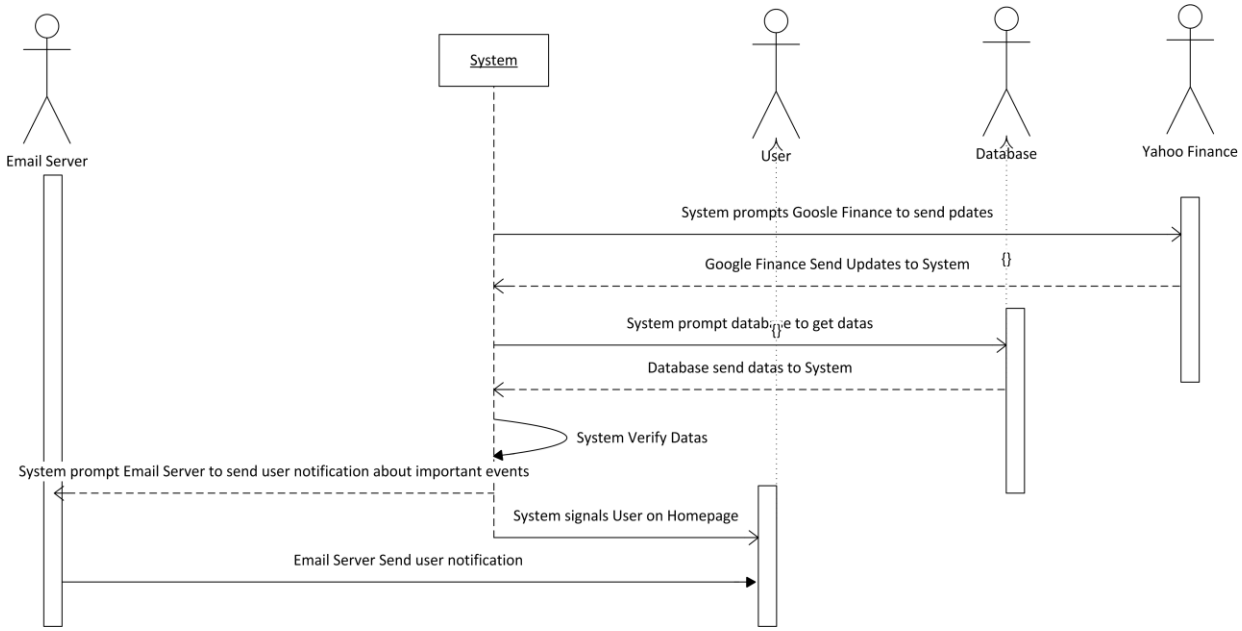
UC-5 Sell Stocks:


User
System
Email Server
Database Server
Yahoo Finance
User visits the Sell section of the StockHop webpage
System prompts user to enter stock name/symbol and amount of stocks
User enters valid stock symbol and number of shares.
{}
System request database for user's portfolio information
Database sends information.
system verifies that user own the stock and the amounts of share he wish to sell.
System queries Yahoo Finance for current price
Yahoo Finance finds a match for the stock name/symbol and return the price
{}
System compute total income and subtracts the commission cost from it.
System asks user to enter type of price (Market, Limit, Stop)
User choose type of price he/she wish to have
System sends the offer to Yahoo Finance.
System removes sold shares from portfolio
System prompt Email server
System signals User
Email server sends notification email to user

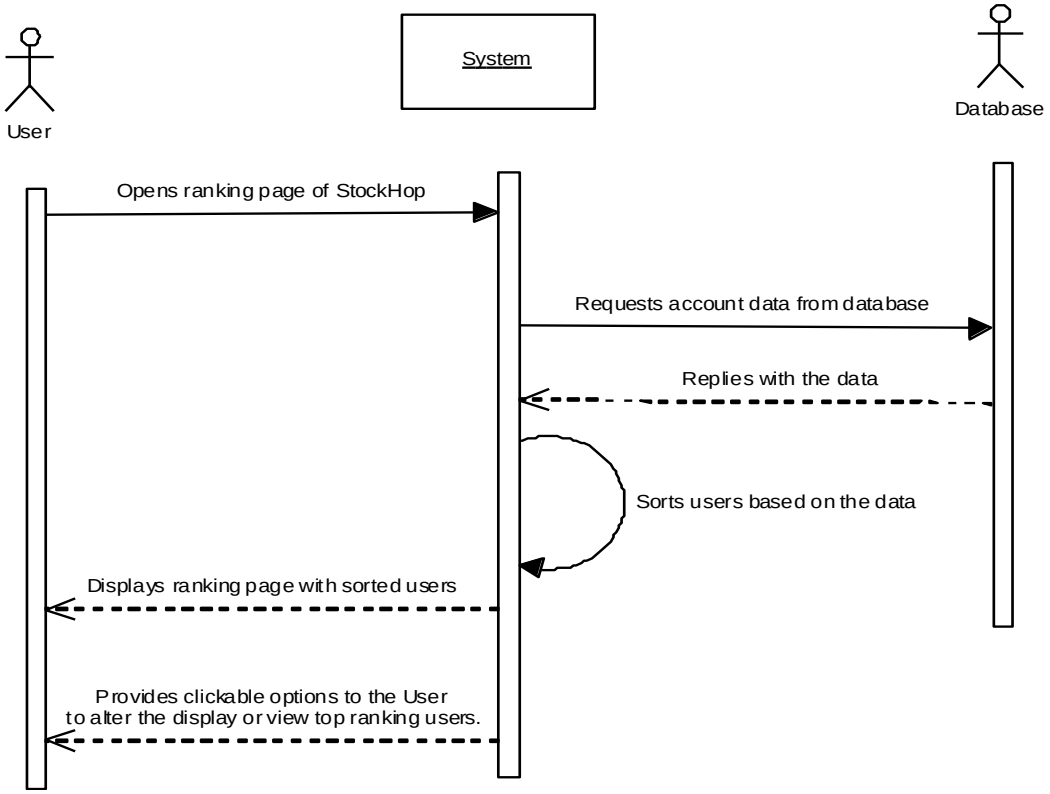
UC-6 Research Stocks



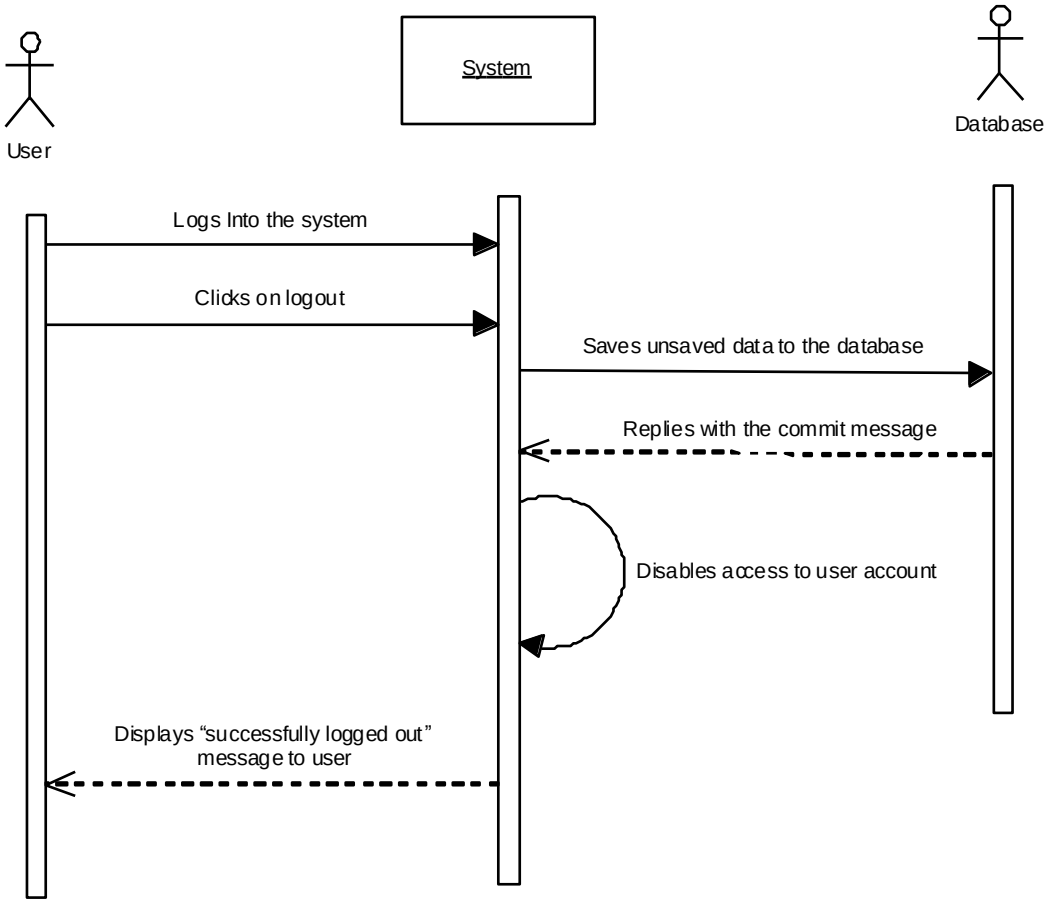
UC-7 Send Notifications:



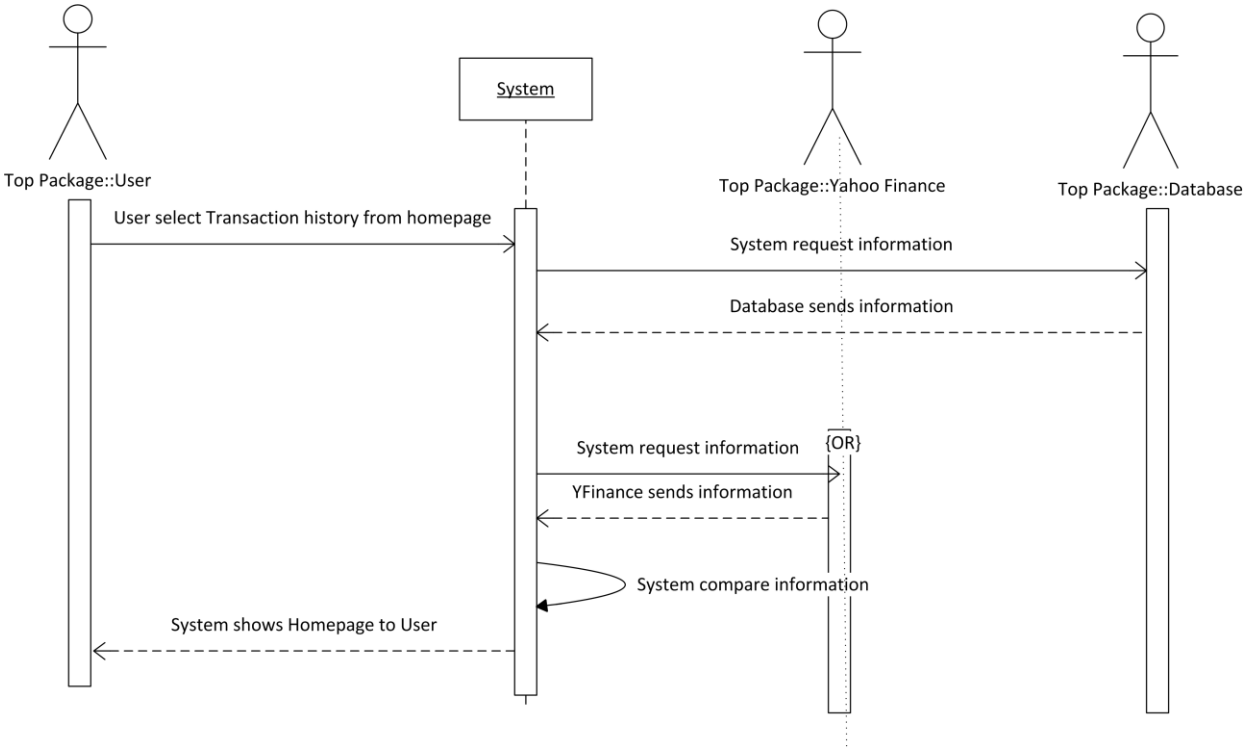
UC-8: View Rankings:


User
System
Database
Opens ranking page of StockHop
Requests account data from database
Replies with the data
Sorts users based on the data
Displays ranking page with sorted users
Provides clickable options to the User
to alter the display or view top ranking users.

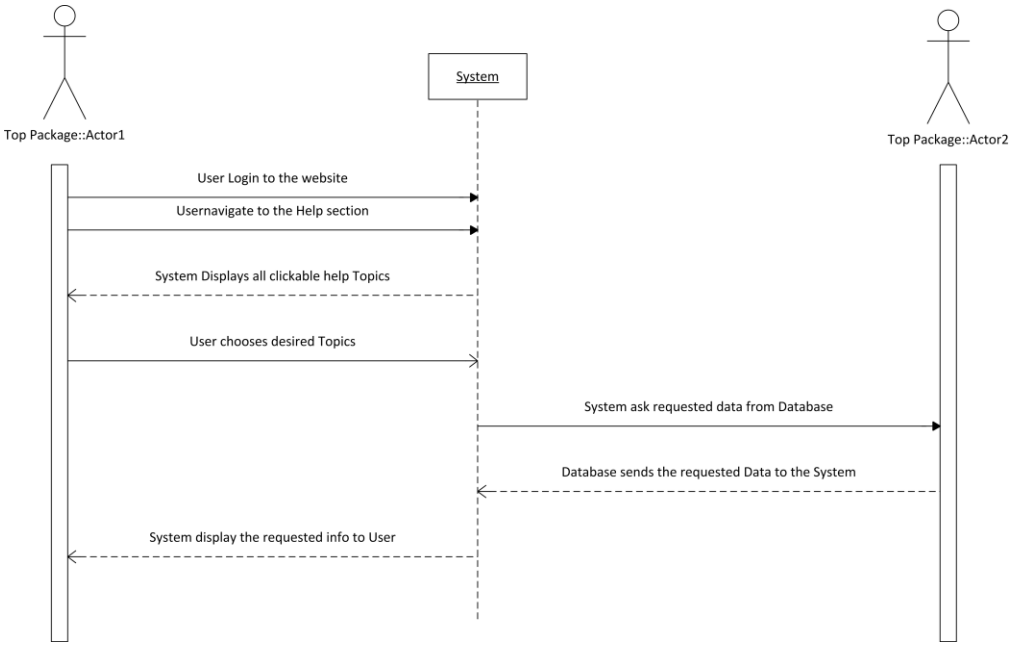
UC-9: User Sign out



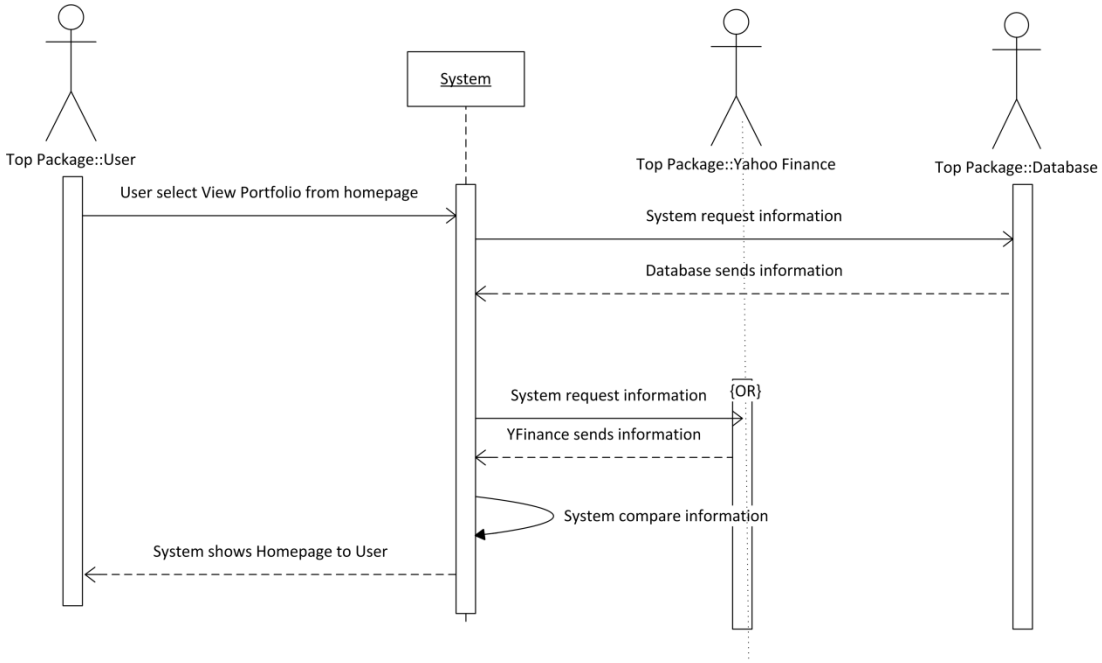
UC-11 View Transactions History:



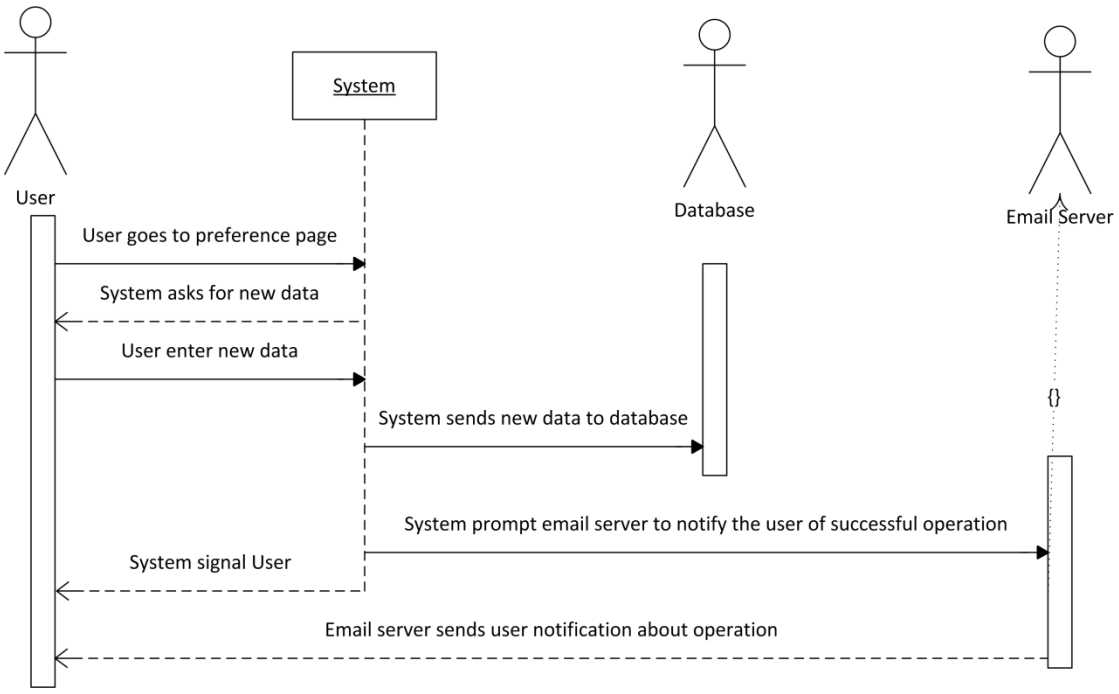
UC-12 Help



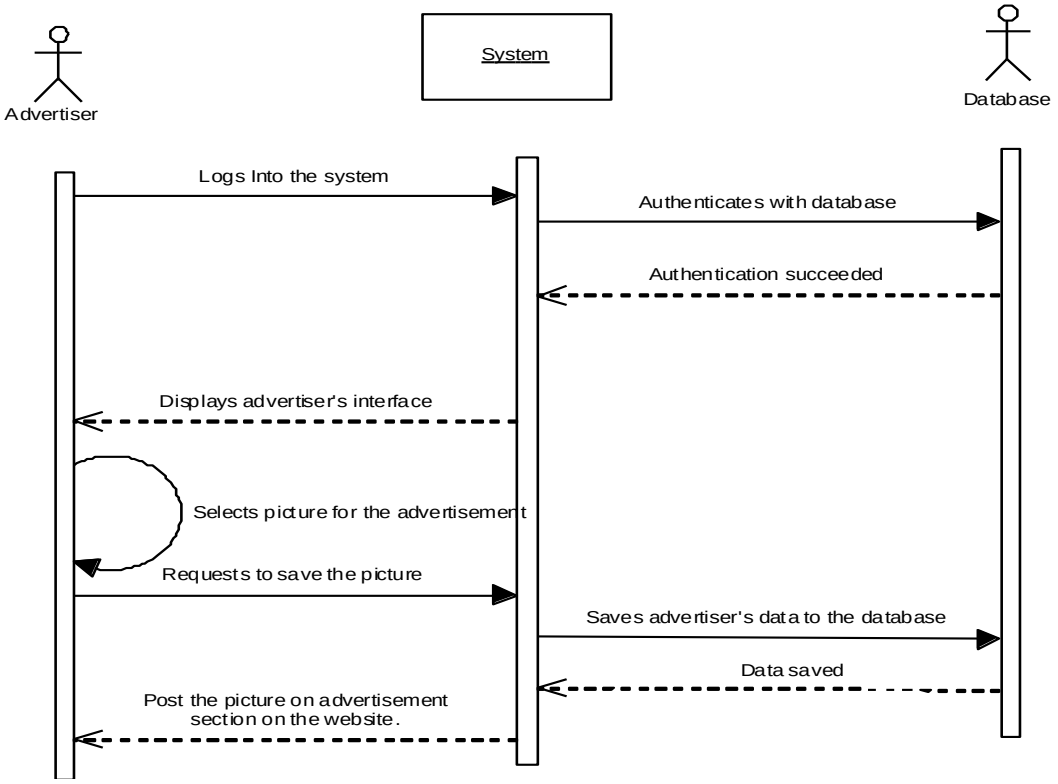
UC-13 View Portfolio:



UC-14 Preferences:



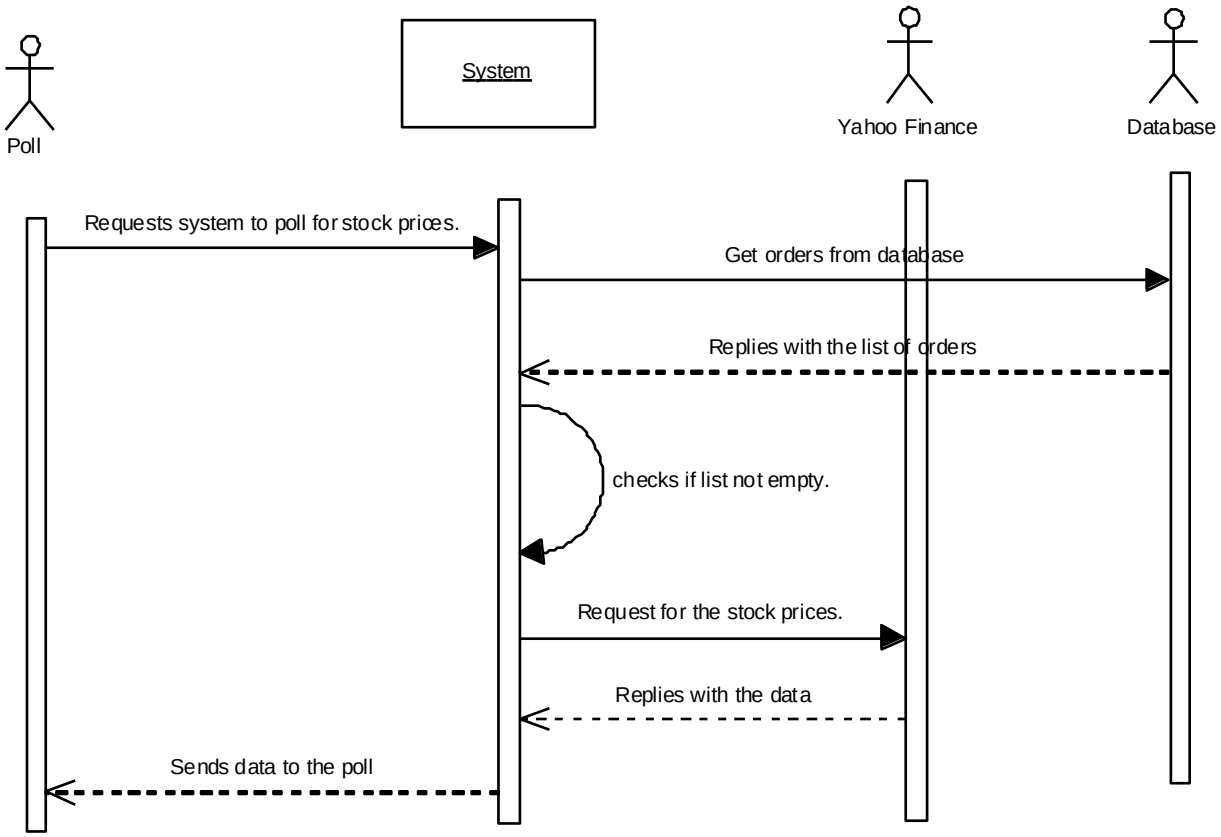
UC-15: Manage Advertisements



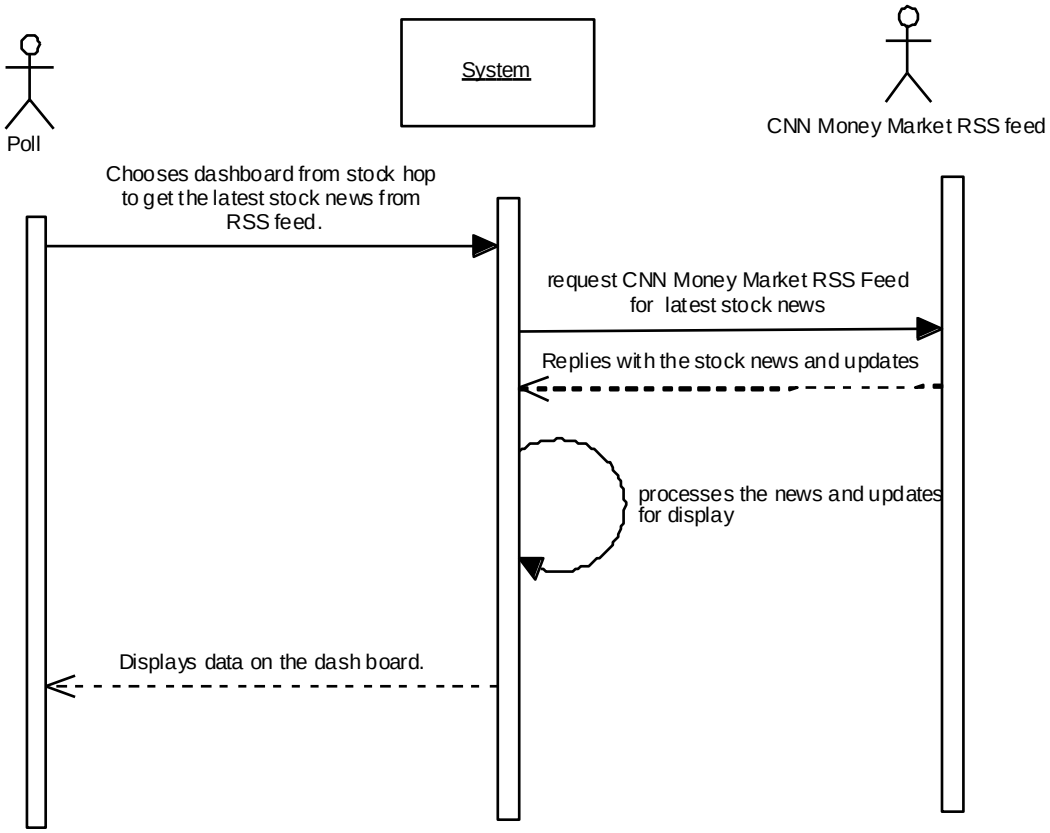
The team had many discussions on how to implement the stock queries from Yahoo Finance. Rather than periodically dumping all information of all possible stocks to a database (memory and hard drive intensive), the team decided that the queries to get information from Yahoo Finance would be done at the time that the user is querying. The group decided that stop, limit, stop-limit, and trailing stop orders would spawn a server-side polling process that would periodically check Yahoo Finance and create an event if any of its criteria is met.

UC-17: Poll for Stock Reports

The Poller actor interacts by seeing if there are orders in the Database. If there are Orders in the Database, the Poller asks Yahoo Finance for the data needed. Once this information is back, it follows with the Execute Buy / Execute Sell logic.



UC-18:



7. Nonfunctional Requirements (FURPS)

- **F**unctionality

  Feature set
  - System shall be able to display stock information on website in easily navigable tables
  - System shall be able to send both email and text message alerts on transaction completion and individual stock alerts
  - System shall be able to display a business News RSS Feed
  - System shall be user-friendly and minimize time to completion of orders
  - System shall display scrolling stock ticker at the top of trading page

  Capabilities
  - System shall be able to model stock trading -- market orders, limit orders, stop orders, stop-limit orders, and trailing stop orders
  - System shall be able to run on multiple browsers, including mobile web

  Security

Logins to the website will be password protected so that users cannot view other users' portfolios. The login forms will take extra security measures to prevent against MySQL injection attacks (i.e. an attack where the user enters malicious code in a PHP form that accesses a MySQL database to try to take over the server or tables where sensitive information is stored). The MySQL database will store passwords in an encrypted format.

- **U**sability

  The website should be easy to navigate and operate in a realistic manner to the actual stock market. Each page on the website will have the same top bar, navigation, and color scheme so that users feel a consistency as they visit all of the pages within the site. In addition, the website should include easy to use navigability and minimization of the time it would take a user to complete transactions. The website should also offer easy-to-understand tutorials about how to use the website.

- **R**eliability

  The website shall be as accurate as possible by taking information from Yahoo Finance. These stock prices are delayed by 15 minutes, but should still be consistent with the information that would be displayed on the original websites (i.e. stock prices will not be made up).

  The website host server requires a Verizon FIOS internet connection to work and runs on PSE&G power. If this connection were to go out due to problems from either company, this would cause the website to be affected. These external factors are not controllable, but the server itself is attached to a UBS backup power unit to reduce its downtime in the event of a power failure.

  Additionally, the system shall periodically backup the user data and MySQL database tables in order to reduce the time to get the website back up and running should it become compromised in any way. This also reduces the time to recover if the primary server storage fails.

- **P**erformance

  The user should be not experience slow-down when the server is gathering data from Yahoo Finance or when other users are trying to make transactions. The actual server hardware is lightweight but is easily upgradable. The website and server-side applications will be written in a way that is not intensive to the

server hardware and also that provides users with the quickest possible response time.

- **S**upportability
  The website shall be written to be as modular as possible to ease any code maintenance that needs to take place in the future and to account for additional features to be added. Code shall be well-commented for future maintainability. It should be written such that it is expandable in the future to run on upgraded server hardware. Since the users of the site are so diverse, the overall site aims to have compatibility with as many different browsers as possible, including Google Chrome, Mozilla Firefox, Internet Explorer, Safari, as well as mobile browsers.

8. Effort Estimation using Use Case Points

To estimate the effort needed for the development of the system, we need to first find the UCP, or Use Case Points of the system. To do this, we begin by finding the Unadjusted Use Case Points (UUCP), which is made up of the sum of the Unadjusted Actor Weight (UAW) and Unadjusted Use Case Weight (UUCW).

Unadjusted Actor Weight (UAW)

Actor Name	Description of relevant characteristics	Complexity	Weight
New Player	New Player is interacting with the system via a Graphical User Interface	Complex	3
User	User is interacting with the system via a Graphical User Interface	Complex	3
Advertiser	Advertiser is interacting with the system via a Graphical User Interface (when placing advertisements)	Complex	3
Database	Database is another system which interacts with the system through a defined API.	Simple	1
Email Server	Same as Database.	Simple	1
Yahoo Finance	Same as Database.	Simple	1
Timer	Same as Database.	Simple	1

UAW(Stockhop) = 4 x Simple + 0 x Average + 2 x Complex = 4 x 1 + 0 x 2 + 2 x 3 = 10

Unadjusted Use Case Weight (UUCW)

Use Case	Description	Category	Weight
UC-1:Registration	Complex User Interface. 9 steps for the main success scenario. 1 participating actor (Database)	Average	10

UC-2:Sign in	Moderate interface design. 5 steps for the main success scenario. 1 participating actor (Database)	Average	10
UC-3: View Home Page	Complex User Interface. 7 steps for the main success scenario. 2 participating actors (Yahoo Finance, Database)	Complex	15
UC-4: Buy Stocks	Complex User Interface. 12 steps for the main success scenario. 2 participating actors (Yahoo Finance, Database)	Complex	15
UC-5: Sell Stocks	Complex User Interface. 15 steps for the main success scenario. 3 participating actors (Yahoo Finance, Database, Email Server)	Complex	15
UC-6: Research Stocks	Complex User Interface. 6 steps for the main success scenario. 1 participating actor (Yahoo Finance)	Average	10
UC-7: Send Notifications	Simple User Interface. 6 steps for the main success scenario. 4 participating actors (User, Advertiser, Database, Yahoo Finance)	Average	10
UC-8: View Rankings	Simple User Interface. 3 steps for the main success scenario. 2 participating actors (User, Database)	Simple	5
UC-9: User sign out	Simple User Interface. 4 steps for the main success scenario. 1 participating actor (Database)	Simple	5
UC-10: View Help	Simple User Interface. 1 step for the main success scenario. 1 participating actor (Database)	Simple	5
UC-11: View Transaction History	Moderate interface design. 8 steps for the main success scenario. 3 participating actors (Database, Yahoo Finance, User)	Average	10
UC-12: View Pending Transactions	Moderate interface design. 8 steps for the main success scenario. 3 participating actors (Database, Yahoo Finance, User)	Average	10

UC-13: View Portfolio	Moderate interface design. 8 steps for the main success scenario. 3 participating actors (Database, Yahoo Finance, User)	Average	10
UC-14: User Preferences	Moderate interface design. 6 steps for the main success scenario. 2 participating actors (Database, Email Server)	Average	10
UC-15: Manage Advertisements	Complex User Interface. 3 steps for the main success scenario. 3 participating actors (Database, Email Server, Users)	Complex	15
UC-16: Maintain Website	Complex interface design. The length of the success scenario cannot be determined, since it is a system administration job that varies based on what site maintenance needs to be done. 1 required participating actor is System Administrator. Other participating actors depend on what is broken or needs maintenance and could include the Database, Email Server, etc.	Complex	N/A
UC-17: Poll for Stock Prices	Complex interface design. 8 steps for the main success scenario. 3 participating actors. (Database, Yahoo Finance, User)	Complex	15
UC-18: View Current Stock News	Moderate interface design. 5 steps for the main success scenario. 2 participating actors (User, Database, CNN Money Market RSS Feed)	Average	10

UUCW (StockHop) = 3 x Simple + 8 x Average + 5 x Complex = 3 x 5 + 9 x 10 + 5 x 15 = 180

UUCP = UAW + UUCW = 10 + 170 = 190

Next, we need to find the Technical Complexity Factor (TCF).

Technical Factor	Description	Weight
T1	Distributed system (running on multiple machines)	2
T2	Performance objectives (are response time and throughput performance critical?)	1
T3	End-user efficiency	1
T4	Complex internal processing	1
T5	Reusable design or code	1
T6	Easy to install (are automated conversion and installation included in the system?)	0.5
T7	Easy to use (including operations such as backup, startup, and recovery)	0.5
T8	Portable	2
T9	Easy to change (to add new features or modify existing ones)	1
T10	Concurrent use (by multiple users)	1
T11	Special security features	1
T12	Provides direct access for third parties (the system will be used from multiple sites in different organizations)	1
T13	Special user training facilities are required	1

Technical Factor	Description	Weight	Perceived Complexity	Calculated Factor (Weighted x Perceived Complexity)
T1	Distributed, web-based system	2	4	2 x 4 = 8
T2	Users expect good performance but there are inherent delays in the simulation	1	2	1 x 2 = 2

T3	End-user expects efficiency but there are no exceptional demands	1	3	1 x 3 = 3
T4	Moderate internal processing	1	3	1 x 3 = 3
T5	No requirement for reusability	1	0	1 x 0 = 0
T6	Ease of install is not very important as it is run on a single server for many users (currently)	0.5	1	0.5 x 1 = 0.5
T7	Ease of backup, startup, and recovery are very important	0.5	5	0.5 x 5 = 2.5
T8	No portability concerns beyond a desire to keep database vendor options open	2	2	2 x 2 = 4
T9	May need to add or modify features	1	3	1 x 3 = 3
T10	Concurrent use is required	1	4	1 x 4 = 4
T11	Security of server and database is a moderate concern	1	3	1 x 3 = 3
T12	Provides direct access for advertisers	1	3	1 x 3 = 3
T13	No unique training needs	1	0	1 x 0 = 0
			Technical Factor Total:	36

Constant-1 (C1) = 0.6

Constant-2 (C2) = 0.01

TCF = Constant 1 + Constant 2 x Technical Factor Total = 0.6 + 0.01 x 36 = 0.96

Finally, we need to find the Environment Complexity Factor (ECF).

Technical Factor	Description	Weight
E1	Familiar with the development process (e.g., UML-based)	2
E2	Application problem experience	1
E3	Paradigm experience (e.g., object-oriented approach)	1
E4	Lead analyst capability	1
E5	Motivation	1
E6	Stable requirements	0.5
E7	Part-time staff	0.5
E8	Difficult programming language	2

Technical Factor	Description	Weight	Perceived Complexity	Calculated Factor (Weighted x Perceived Complexity)
E1	Beginner familiarity with the UML based development	1.5	1	1.5 x 1 = 1.5
E2	Some familiarity with application problem	0.5	2	0.5 x 2 = 1
E3	Moderate knowledge of object-oriented approach	1	3	1 x 3 = 3
E4	Beginner lead analyst	0.5	1	0.5 x 1 = 3
E5	Highly motivated	1	5	1 x 5 = 5
E6	Stable requirements expected	2	5	2 x 5 = 10
E7	All of the staff is part-time	-1	5	-1 x 5 = -5
E8	Programming language of average difficulty will be used	-1	3	-1 x 3 = -3

Technical Factor Total:	15.5

Constant-1 (C1) = 1.4

Constant-2 (C2) = -0.03

ECF = Constant 1 + Constant 2 x Environmental Factor Total = 1.4 - 0.03 x 15.5 = 0.935

Now that we have the UUCP, TCF, and ECF, we can find the Use Case Points (UCP) by the following equation.

UCP = UUCP x TCF x ECF = 190 x 0.96 x 0.935 = 170.544

Finally, to find out the duration of development time for the system, we can use the provide Productivity Factor (PF) of 28 hours per use case point in the following equation.

Duration = UCP x PF = 170.544 x 28 = 4775.232 person-hours

Evaluation of this equation results in 4775.232 person-hours needed to develop this system. Of course this is only an estimation, but it is based on a logical process that helps to give a base expectation. Group 2 acknowledges that this number is very high. There were 6 of us working on this project, so that would have to equate to ~750 hours (or roughly 30 days of 24 hour work per person). This is obviously an impractical calculation. The person-hours serve to show us an estimate. Since there was no historical data on what constitutes a Simple, Average, or Complex task for the 6 of us at our current skill level and no historical data on what the weights should be, we obtained an unreasonably high value for the person-hours. In the future, we would look at previous data, such as the actual hours spent on coding this project to come up with more reasonable estimates.

9. Software Implementation Design Choices

This section describes why Group 2 made decisions to use certain programming languages, frameworks, or abstraction layers (and what, specifically, they are) in order to code this project.

a. PHP 5.3.8 - PHP stands for PHP:Hypertext Preprocessor. PHP is a web development server-side scripting language. It is used to create websites with dynamic content and the ability for user interaction. PHP is often embedded with HTML to develop web applications. PHP was chosen by Group 2 due to a familiarity with it between the students. PHP 5.3 was the latest major release of PHP and incorporates many new features that older PHP does not provide, mostly to do with the ability to use namespaces. These namespaces allow the code to be organized in a cleaner fashion.

b. Symfony2 [PHP Framework] – Symfony2 is a PHP based framework implementing a MVC (Model/View/Controller) architecture with the goal of rapidly developing a web application. MVC has become a very popular and widely used architecture for web applications. Symfony2 utilizes the namespace functionality of PHP 5.3. Symfony2 makes an attempt to provide the effective structure and functionality that most web applications will require, which is advantageous since the developer does not have to reconstruct code on his own that is already provided (saving time, production cost, etc). It integrates other outside projects, Doctrine and Twig (see below), to further ease the burden of initial project set-up.

   Symfony2's base framework is designed to allow the developers to effortlessly design a "Representational state transfer", or REST, interface to both regular web users and to other applications if needed. The REST API was advantageous to Group 2 because of the benefit of allowing us to conceal a lot of the complicated data from the user. The user does not need to see large complicated query strings, and we, as developers, do not need to do large complicated mod_rewrite apache expressions to 'clean up' our query strings. Group 2 chose symfony2 because it is a new framework that incorporates all of the features of the latest MVC technologies. When tested against various benchmarks, the Symfony2 framework was up to 100x faster than other frameworks such as CakePHP and up to 10x faster than the popular Zend Framework. Group 2 knew that they wanted to implement the MVC method when writing their code, in order to increase the readability of the code and also to make a clearer separation between the “web development” and “web design” for future maintenance.

c. Twig [PHP Templating Engine] – Twig is a template engine for PHP. A template engine is “software that is designed to process web templates and content information to produce output web documents (“Template Engine (web)”, Wikipedia)”. Symfony2 effectively uses Twig to separate the presentation logic from the internal logic (for instance, HTML from complex PHP functions). Symfony 2 overall uses a very object oriented approach; its base functionality is all provided by via a series of objects. Symfony2’s integration with Twig further helps to separate and speed up the development process. By using Twig, the developers do not need to write overly complicated PHP mixed in with the HTML. Twig functionality provides just enough logic to make simple decisions, include extra content, or iterate through arrays. Twig is not responsible for, nor could it handle, any actual logic relevant to the tasks such as form validation, user verification, or database interactions.

d. Doctrine Object Relational Mapper (ORM) – Doctrine is a Free and Open Source Software (FOSS) project to provide stable and consistent Object Relational Mapping (ORM) functionality to the PHP language. Its function as an ORM is to allow persistent database records to be exposed to the program as standard PHP classes. These classes obfuscate the underlying database data types and expose a very convenient set of accessor and modify functions. The classes in Doctrine are allowed to inherit parent classes and implement interfaces, giving the abstracted database objects the ability to take advantage of the flexibility and functionality of OO concepts. Another benefit of Doctrine is its use of PHP's PDO (PHP:Data Objects) database abstraction layer. This abstraction layer allows PHP, and thus Doctrine, to support many types of database servers beyond MySQL. Group 2 chose Doctrine over other ORMs due to its integration with Symfony2. The abstraction layer allows setting/getting of fields in the databases through function calls, rather than having to write many queries.

e. jQuery – jQuery is used for client-side javascript scripting. It is the most widely used javascript library that includes many useful functions. It works across multiple browsers that have javascript enabled. Group 2 chose jQuery over straight javascript due to the ease of use of jQuery and its included functions. It is primarily used in thestockhop to load data from Yahoo! Finance and parse this data without having to reload the page.

f. PHPUnit – Unit testing framework for PHP. Allows a user to write tests for their PHP functions. Can execute its own requests as a mock ‘user’ and will execute the PHP code without having to go through the GUI. This testing method

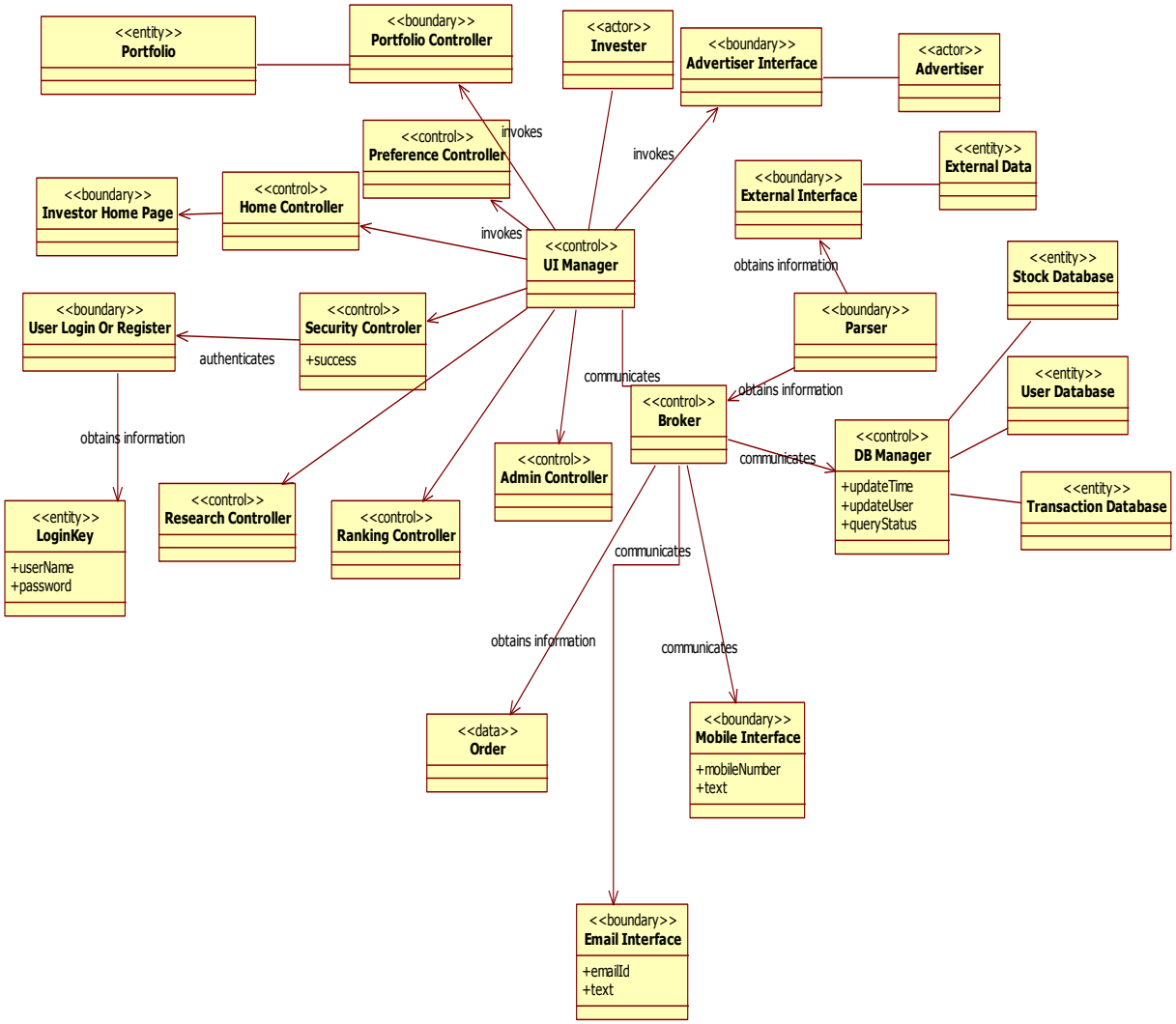
works best in practices where the test is written before the code, so the specific goals that each function is supposed to achieve are clearer.

g. Java Programming Language [Backend Server-Side Program] – Java is a high-level object oriented programming language. Before choosing a language, Group 2 knew they wanted to use an object-oriented language for the backend software. Group 2 ultimately chose Java over C/C++ due to its ease of integration with web applications – many websites already use Java on the backend, so libraries were already written for connecting to the database, sending mail via a mail server, etc. Another key feature of Java is, once compiled, a java.class file (machine bytecode) can typically run on any machine that has java installed, independent of the computer architecture. This increases the portability of our software.

h. Other Decisions – MySQL was chosen as the relational database. MySQL was chosen because it is easy to deploy on the backend server and Group 2 teammates had the most functionality with this database. The Apache web server was used because it was easy to deploy and is highly configurable. Apache was chosen over something like Microsoft IIS because we were writing our application in PHP and not something like Microsoft's ASP.NET. We also set up an email server POSIX and routed through a gmail account in order to get around constraints imposed due to firewall issues with Verizon FIOS.

10. Domain Analysis

a. Revised Domain Model



This is a revised Domain Model from Report 1. The central idea for the domain model remains the same, however, minor edits were made to align it with the current status. When we first created our domain model, we had not decided on the model-view-controller framework for the frontend, so we did not define a central controller the same way.

We have a central controller concept which communicates with various other concepts. In the updated model we have two such controllers, Broker which did most of the backend control and UI Manager which did most of the frontend control. A few concepts are renamed though their roles are unchanged and a few

concepts are added new like controllers for each of research, ranking and preferences page. Few concepts like limit order, stop order are combined into single concept order, as during implementation, we realized we could have a single concept order and the broker concept would internally handle individual orders

i. Concept Definitions

Concept Name	Type	Concept Description
Administrator Interface	D	Interface for system administrator actor to perform various operations like maintaining user accounts, configuration changes, data manipulation, etc.
UserLoginOrRegister	D	Displays functions for users to log in into the system or for new users to register themselves.
Login Key	K	Stores the username and password information.
Place An Order	K	Stores information about a transaction initiated by an investor.
Investor Home Page	D	Displays information after users log in.
Trade Page	D	Provides interface for user to enter data to buy and sell stocks. Once a symbol is selected, interacts with Yahoo finance to display the stock information.
Buy A Stock	D	Stores data for the order placed to buy a stock. Uses functions provided by Check Order Valid to verify the validity of the order. Adds data to open orders database table. For valid orders based on the order type invokes appropriate transaction i.e. market, limit, stop.

Sell A Stock	D	Stores data for the order placed to buy a stock. Uses functions provided by Check Order Valid to verify the validity of the order. For valid orders based on the order type invokes appropriate transaction i.e. market, limit, stop.
Check Order Valid	D	Provides functionality to check the validity of the order placed. Checks if the symbol is valid, user has enough cash to perform the transaction etc.
Place Market Order	D	Provides functionality to complete a market order transaction. Checks to see if 20 minutes has passed. Checks if user has enough cash to perform the transaction. If yes, execute order. Update information in database to reflect purchase (decrement user's cash, update transaction history and portfolio, etc). Notify user if user preferences are set.
Place Limit Order	D	Provides functionality to complete a limit order transaction. Gets the stock prices and volumes for all open orders. Monitors data from yahoo to check if the limit price has reached. If yes, further processes the order by purchasing or selling the shares at limit price and following the same path as the market order.
Place Stop Order	D	Provides functionality to complete a stop order transaction. Gets the

		stock prices and volumes for all open orders. Monitors data from yahoo to check if the price has reached. If yes, further processes the order by transitioning the order from limit order to market order.
View Portfolio	K	Provides interface for user to view his portfolio. Interacts with database to obtain the portfolio information.
Stock Data Observer	D	Keeps observing the up to date stock prices and information. Gets data from the controller through the external interface
UI Manager	D	Manages the various user interfaces. Interacts with the controller for information from database, to obtain external data etc.
Authenticator	D	Responsible for authenticating the user based on the login information provided.
Error Interface	D	Displays appropriate error messages to the users.
Advertiser Interface	D	Interface for advertiser actor.
Controller	D	Coordinates information between concepts and events based on use cases. It instructs the database, obtains information from the database and gives it to the

		different interfaces.
External Interface	D	Interacts with the external systems like Yahoo to get the data.
External Database	K	External data from Yahoo or any other external system
Email Interface	D	System Interface for email system
Mobile Interface	D	System Interface for SMS system
DB Manager	D	Interface to database used to retrieve and store data
Stock Market Database	K	Stores stock market information
Transaction Database	K	Stores information about transactions
User Database	K	Stores information about the users

Type D: Doing, Type K: Knowing

ii. Association Definitions

Concept	**Relation [direction: ->]**	**Concept**
Controller	Communicates	UI Manager
Controller	Communicates	Email Interface
Controller	Communicates	Mobile Interface
Controller	obtains information	Portfolio
Controller	Instructs	DB Manager
UI Manager	Invokes	User Interface
UI Manager	Invokes	Admin Interface
UI Manager	Invokes	Advertiser Interface
UI Manager	Invokes	External Interface
UI Manager	Communicates	Controller
Stock Data Observer	obtains information	Controller

Authenticator	obtains information	Controller
User Interface	invokes	Login Page
User Interface	invokes	Buy/Sell Page
User Interface	invokes	Home Page
Admin Interface	invokes	Login Page
DatabaseManager	forwardsRequest	RetrieveStockData
DatabaseManager	forwardsRequest	RetrieveStockChar
Login Page	obtains information	Login
Login Page	authenticates using	Authenticator
Trade Page	Performs	Place An Order
Place An Order	Performs	Buy A Stock
Place An Order	Performs	Sell A Stock
Buy A Stock	Validates	Is Order Valid
Sell A Stock	Validates	Is Order Valid
Buy A Stock	Performs	Place Market Order
Buy A Stock	Performs	Place Limit Order
Buy A Stock	Performs	Place Stop Order
Sell A Stock	Performs	Place Market Order
Sell A Stock	Performs	Place Limit Order
Sell A Stock	Performs	Place Stop Order
Authenticator	Report Error	Error Interface
Buy A Stock	Report Error	Error Interface
Sell A Stock	Report Error	Error Interface
Email Interface	Report Error	Error Interface
Mobile Interface	Report Error	Error Interface

iii. Attribute Definitions

Concept	Attribute/ Definition
Administrator Interface	
UserLogin	
LoginKey	1. username – Username provided by the users 2. password – Password provided by the user
Invester Home page	
Trade Page	
PlaceOrder	1. orderType – Type of order that is either buy or sell 2. noOfShares – number of the shares to be bought or sold

	3. status - Indicates whether the buy/sell was successful
BuyAStock	1 status – indicates whether buy was successful (the same is communicated in upward direction)
SellAStock	1 status – indicates whether sell was successful(the same is communicated in upward direction)
ViewPortfolio	1. user – user for whom the portfolio is created 2. portfolioData - other data pertaining to portfolio
Check Order Valid	
Place Market Order	
Place Limit Order	
Place Stop Order	
UIManager	
Authenticator	1. success – set to 1 if the authentication succeeded
Error Interface	1. error – holds the error message
Advertiser Interface	
Controller	
External Interface	
Email Interface	1. emailId – email id of the user 2. text – the text to be emailed
Mobile interface	1. mobileNumber – mobile number 2. text – text to be messaged
DB Manager	1. updateTime – time when the update was performed 2. updateUser – user who performed the update 3. updateStatus – Indicated if the update was successful

b. System Operation Contracts

Operation:	Create New Account
Preconditions:	➢ The account name does not exist
Postconditions:	➢ A new user account created ➢ An amount of virtual money is added to the new account

Operation:	Buy Stocks
Preconditions:	➢ An investor is already logged in ➢ The investor has sufficient funds ➢ Validate the trading volume of the day
Postconditions:	➢ Update the investor's portfolio

Operation:	Sell Stocks
Preconditions:	➢ An investor is already logged in ➢ The investor has sufficient number of shares of the stock
Postconditions:	➢ Update the investor's portfolio

Operation:	View Portfolio
Preconditions:	➢ An investor is already logged in
Postconditions:	➢ Display investor's portfolio information

Operation:	View Stock Detail Information
Preconditions:	➢ An investor is already logged in ➢ The entered content matches a stock name in database

Postconditions:	➢ Display the desired stock information

Operation:	Manage User Accounts
Preconditions:	➢ A system administrator is already logged in ➢ The desired account information exists
Postconditions:	➢ The account reflects the change based on the administrator's operation

Operation:	Manage Webpage
Preconditions:	➢ A system administrator is already logged in
Postconditions:	➢ The webpage reflects the change based on the administrator's operation

Operation:	Email Notification
Preconditions:	➢ An operation or an event is triggered
Postconditions:	➢ An email notification is sent to the corresponding user based on an event or operation

Operation:	Mobile Device Notification
Preconditions:	➢ An operation or an event is triggered

Postconditions:	➢ A text message is sent to the corresponding user's mobile device based on an event or operation

Operation:	View Leaderboard
Preconditions:	➢ An investor is already logged in
Postconditions:	➢ Display the leaderboard

11. Interaction Diagrams

a. *Summary of interaction diagrams*

i. Registration

ii. View Homepage

iii. Buy Stocks

iv. Execute Buy

v. Sell Stocks

vi. Execute Sell

vii. Research Stocks

viii. View Portfolio

ix. Send Notifications

x. View Transaction History

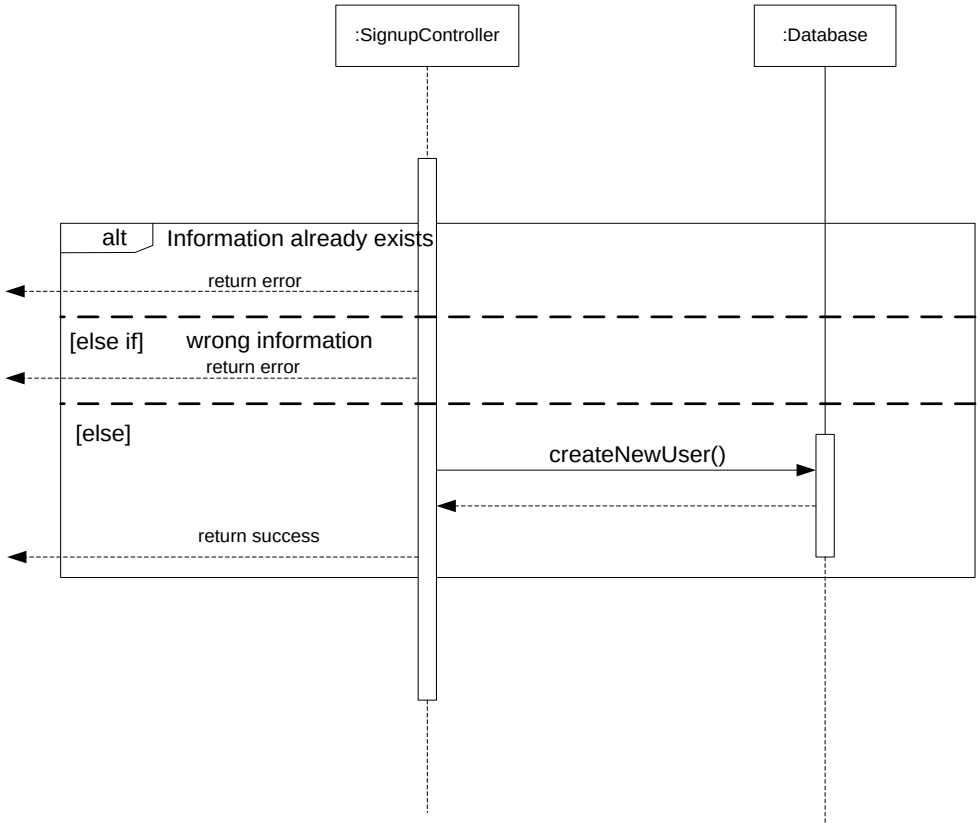
xi. Preferences

As a general note, the frontend is using the Model-View-Controller design pattern. Almost everything about the Symfony2 frontend leverages well-defined Java design patterns. Any time the word "Controller" is shown in the interaction diagrams, it is representing the Front End Controller design pattern ("Front Controller Pattern," Wikipedia). The Frontend interaction diagrams show the use of the Model-View-Controller whenever the word "Controller" is used.

Please note: The Design Patterns are sufficiently described in the section 12C (Design Patterns). The Interaction Diagrams Execute Buy, Buy, Execute Sell, and Sell were created and show the backend design patterns in use. The actual explanation of these Design Patterns is provided in 12C and was not repeated in this section.

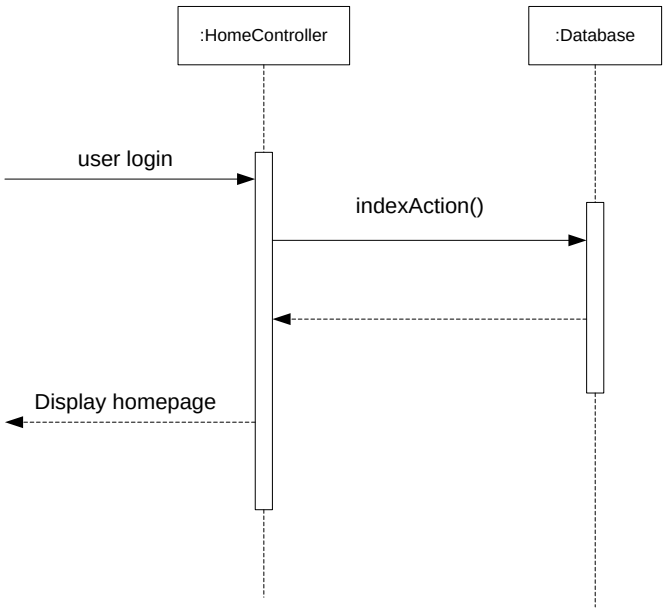
b. Interface Diagrams

Registration



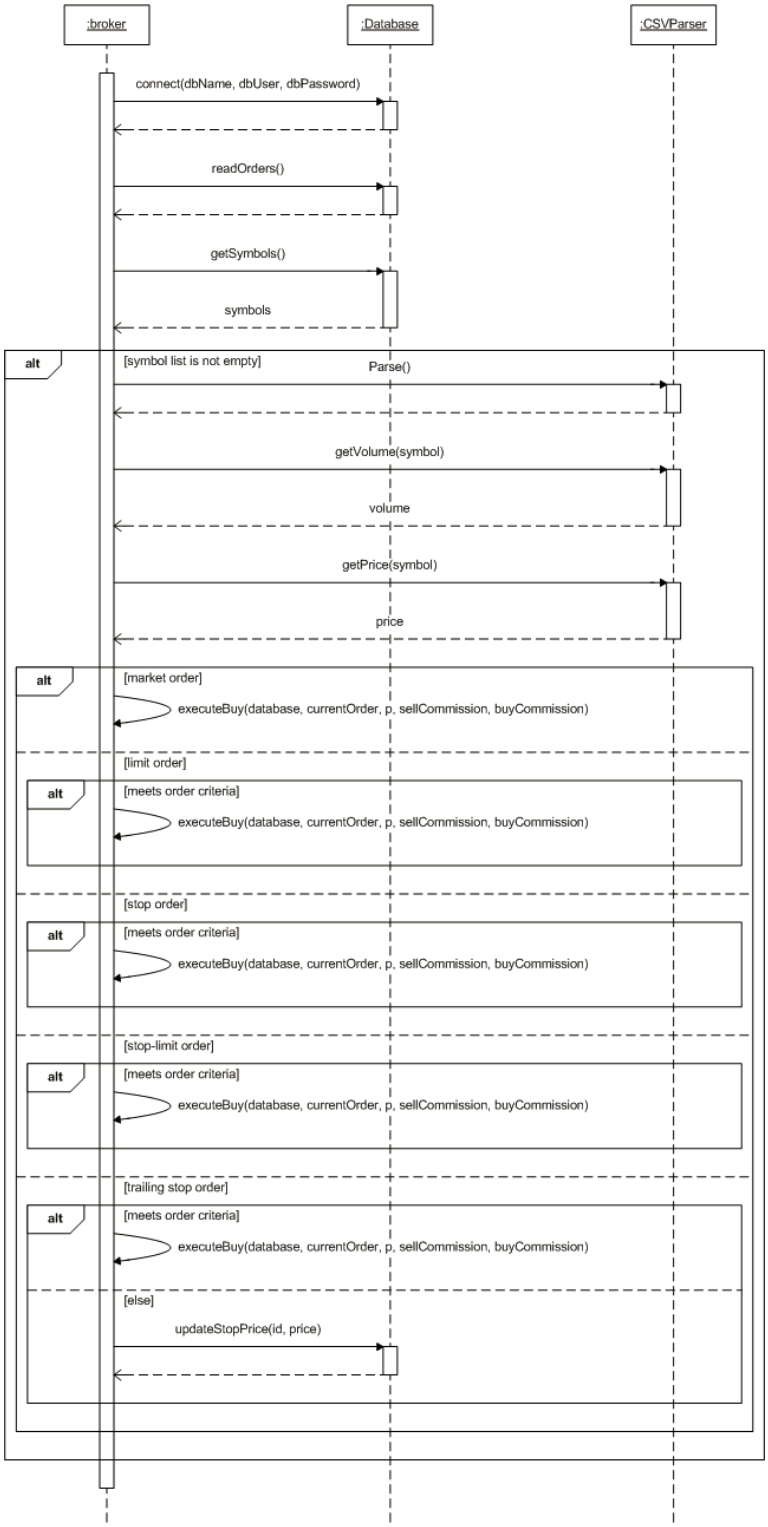
The above diagram shows Registration use case. When a user goes to the registration page, the system asks for username and password that the user wants to create. After that, system will request information from database and check information to see whether this username is already taken or not and if the information is valid. If there is nothing wrong with the information, the database is updated and show confirmation note for the user.

View Homepage



The above diagram shows View Homepage use case. After a user is logged into the system, the HomeController would ask information from the database and show the home page to the current user.

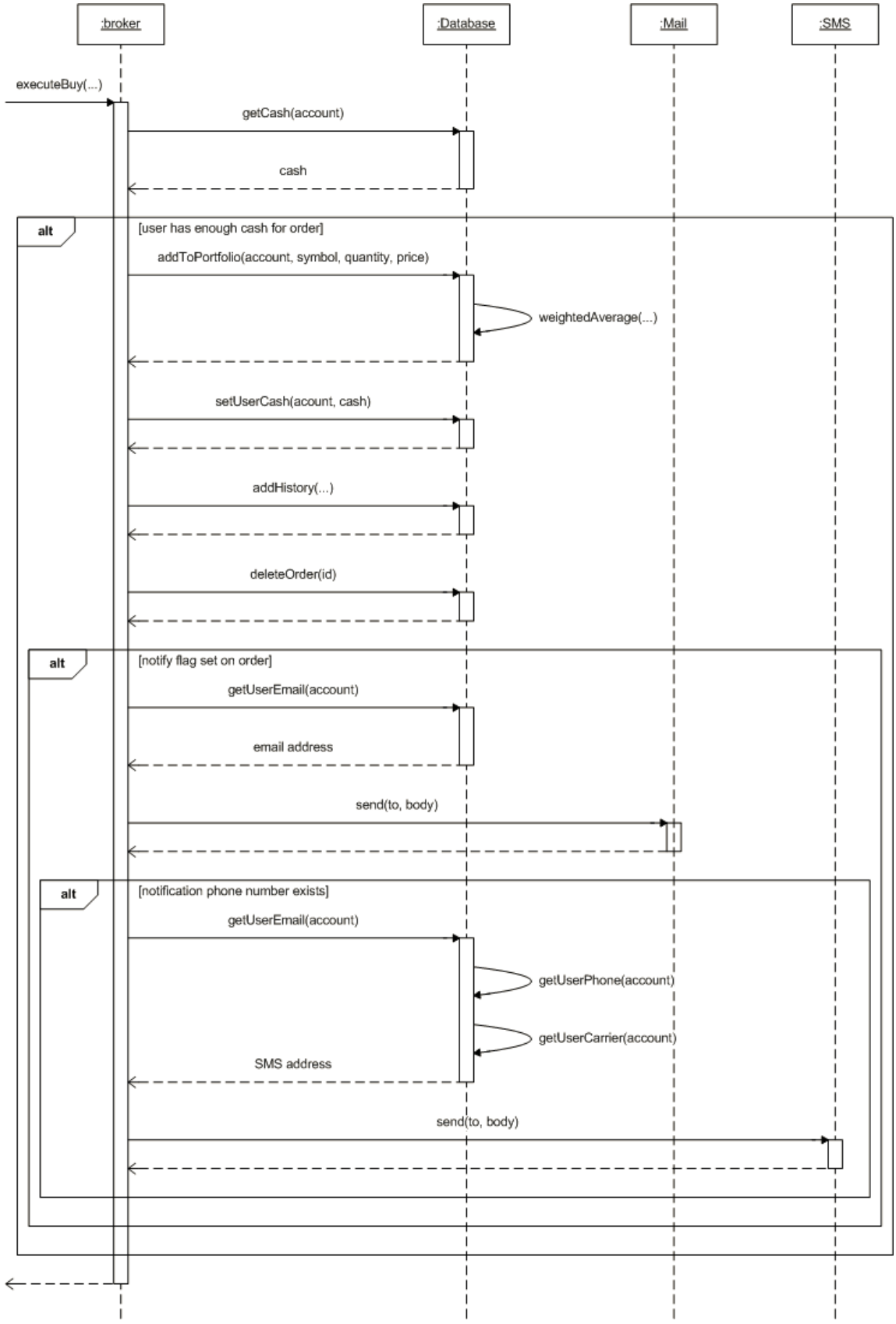
Buy Stocks


:broker
:Database
:CSVParser
connect(dbName, dbUser, dbPassword)
readOrders()
getSymbols()
symbols
alt
[symbol list is not empty]
Parse()
getVolume(symbol)
volume
getPrice(symbol)
price
alt
[market order]
executeBuy(database, currentOrder, p, sellCommission, buyCommission)
[limit order]
alt
[meets order criteria]
executeBuy(database, currentOrder, p, sellCommission, buyCommission)
[stop order]
alt
[meets order criteria]
executeBuy(database, currentOrder, p, sellCommission, buyCommission)
[stop-limit order]
alt
[meets order criteria]
executeBuy(database, currentOrder, p, sellCommission, buyCommission)
[trailing stop order]
alt
[meets order criteria]
executeBuy(database, currentOrder, p, sellCommission, buyCommission)
[else]
updateStopPrice(id, price)

The use case of Buy Stock is represented in the above diagram. An user at the front end first initiates a buy action and specifies the type(market, limit, stop, limit, or trailing stop), price, and volume of the order he wants. Then the backend fetch the order through database and query about the stock information from Yahoo Finance. After getting the price and volume information, the system verify that total cost less than user's cash balance and send information to database and update user's portfolio.

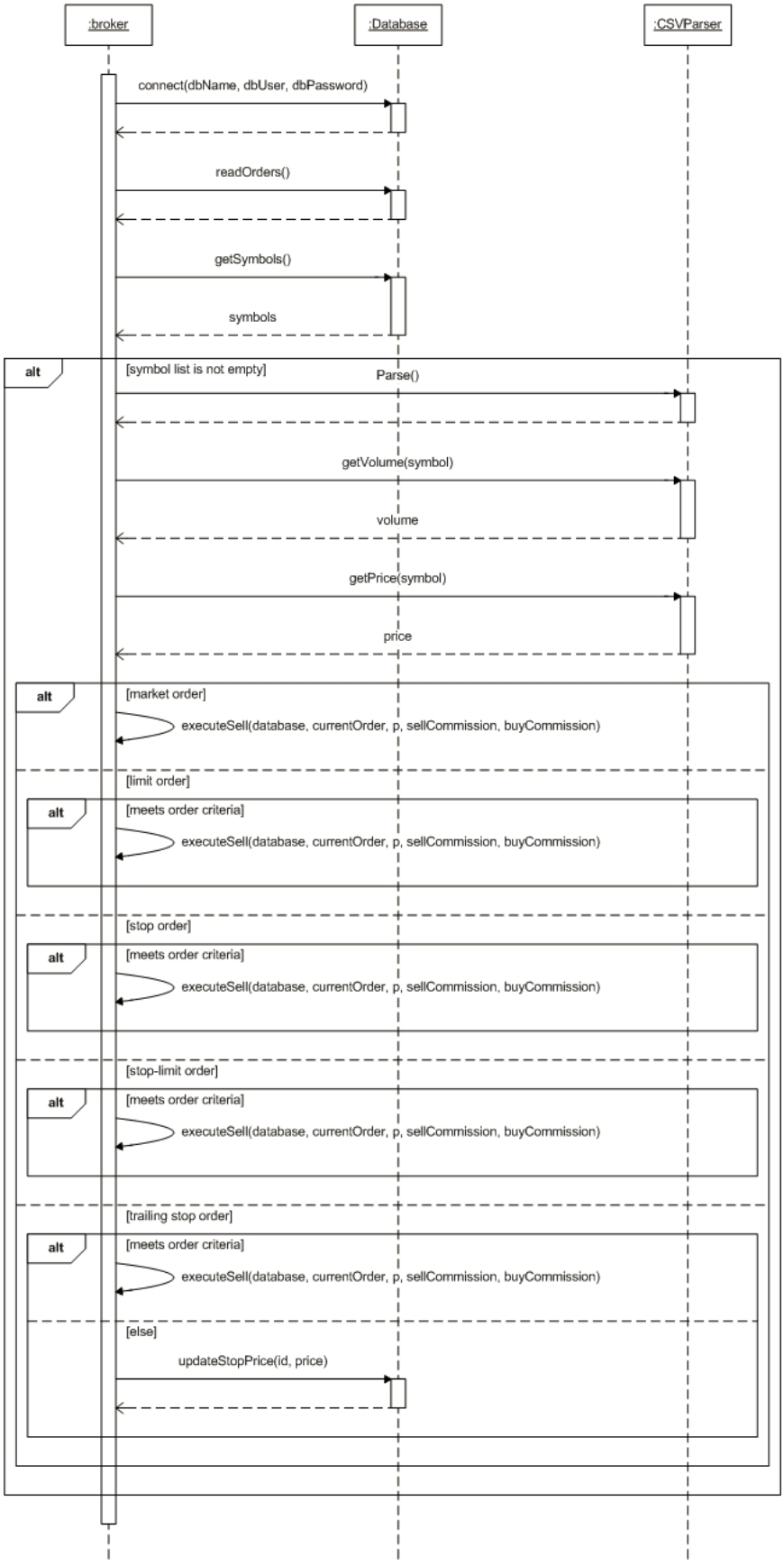
Execute Buy

Execute Buy differs from Buy Stocks in that it describes the behavior that happens on the backend after a stock is successfully deemed to be able to execute and is a 'Buy' action. If the order is executed successfully, the system would notify user that the transaction has been completed.


:broker
:Database
:Mail
:SMS
executeBuy(...)
getCash(account)
cash
alt
[user has enough cash for order]
addToPortfolio(account, symbol, quantity, price)
weightedAverage(...)
setUserCash(acount, cash)
addHistory(...)
deleteOrder(id)
alt
[notify flag set on order]
getUserEmail(account)
email address
send(to, body)
alt
[notification phone number exists]
getUserEmail(account)
getUserPhone(account)
getUserCarrier(account)
SMS address
send(to, body)

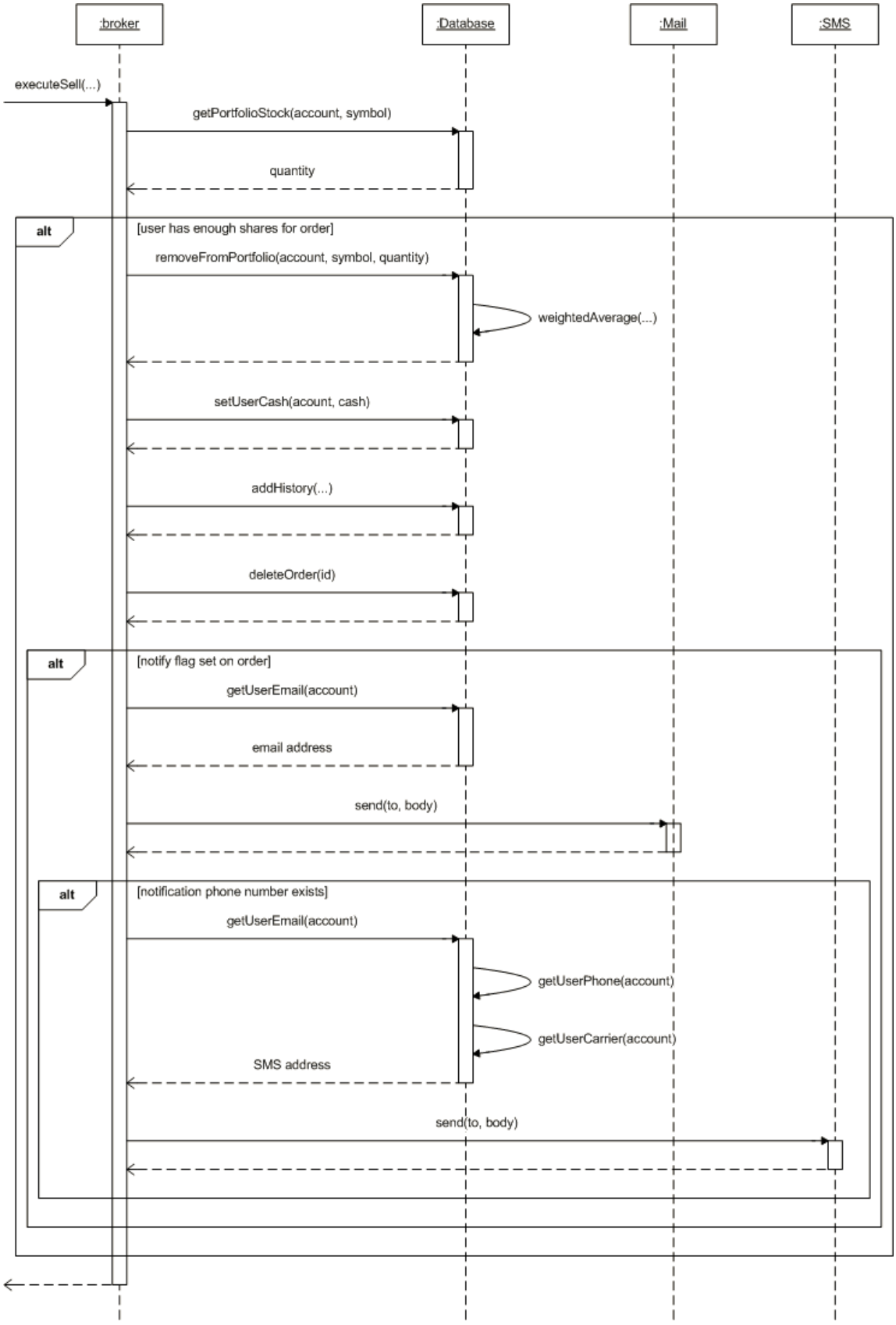
Sell Stocks

The diagram below is to show the use case of Sell Stock. An user at the front end first initiates a sell action and specifies the type(market, limit, stop, stop-limit, or trailing stop), price, and volume of the order he wants. Then the backend fetch the order through database and query about the stock information from Yahoo Finance. After getting the price and volume information, the system verify that user owns the stock and the amounts of share he wish to sell and send information to database and update user's portfolio.

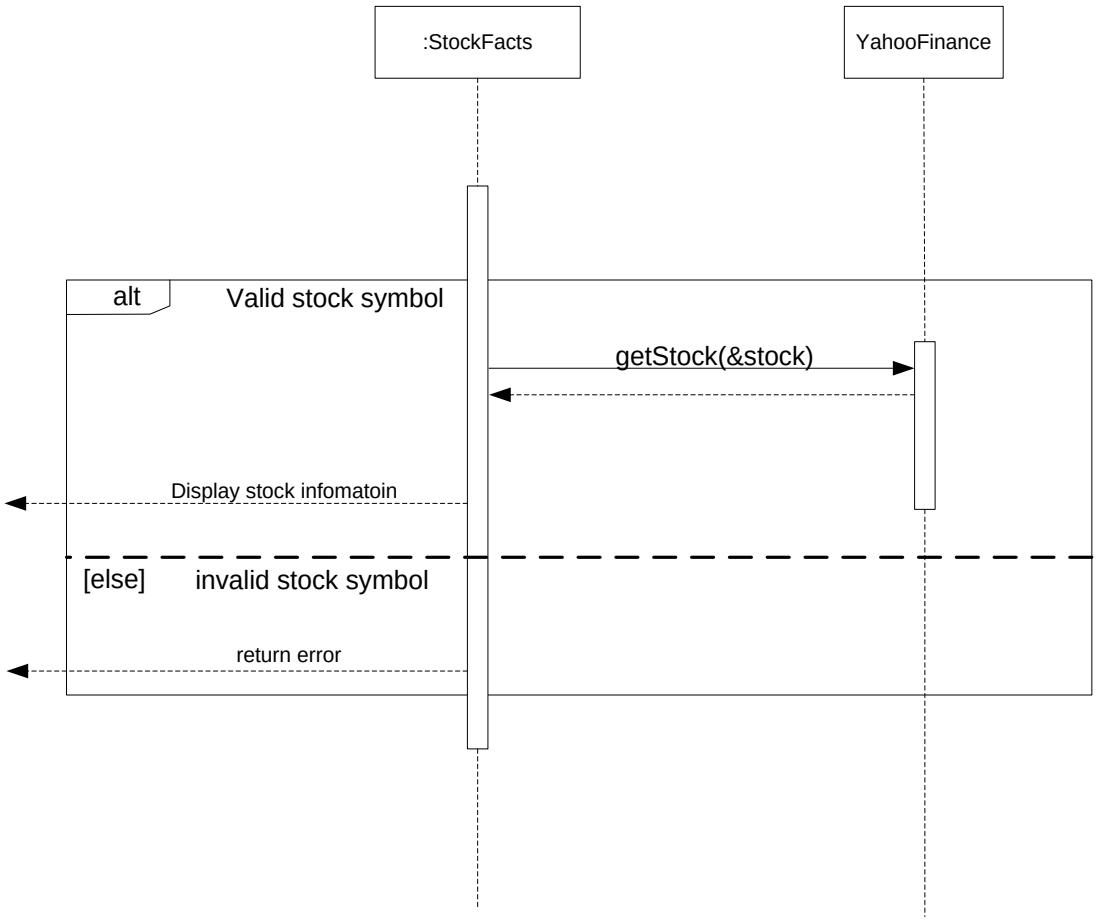

:broker
:Database
:CSVParser
connect(dbName, dbUser, dbPassword)
readOrders()
getSymbols()
symbols
alt
[symbol list is not empty]
Parse()
getVolume(symbol)
volume
getPrice(symbol)
price
alt
[market order]
executeSell(database, currentOrder, p, sellCommission, buyCommission)
[limit order]
alt
[meets order criteria]
executeSell(database, currentOrder, p, sellCommission, buyCommission)
[stop order]
alt
[meets order criteria]
executeSell(database, currentOrder, p, sellCommission, buyCommission)
[stop-limit order]
alt
[meets order criteria]
executeSell(database, currentOrder, p, sellCommission, buyCommission)
[trailing stop order]
alt
[meets order criteria]
executeSell(database, currentOrder, p, sellCommission, buyCommission)
[else]
updateStopPrice(id, price)

Execute Sell

Execute Sell differs from Sell Stocks in that it describes the behavior that happens on the backend after a stock is successfully deemed to be able to execute and is a “Sell” action. If the order is executed successfully, the system would notify user that the transaction has been completed.

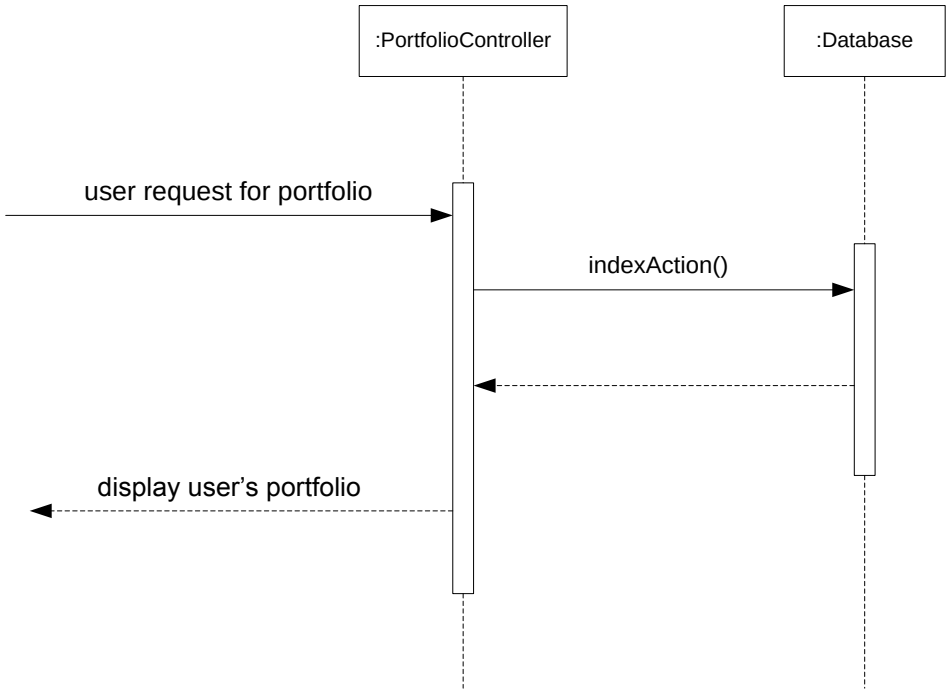

:broker
:Database
:Mail
:SMS
executeSell(...)
getPortfolioStock(account, symbol)
quantity
alt
[user has enough shares for order]
removeFromPortfolio(account, symbol, quantity)
weightedAverage(...)
setUserCash(acount, cash)
addHistory(...)
deleteOrder(id)
alt
[notify flag set on order]
getUserEmail(account)
email address
send(to, body)
alt
[notification phone number exists]
getUserEmail(account)
getUserPhone(account)
getUserCarrier(account)
SMS address
send(to, body)

Research Stocks



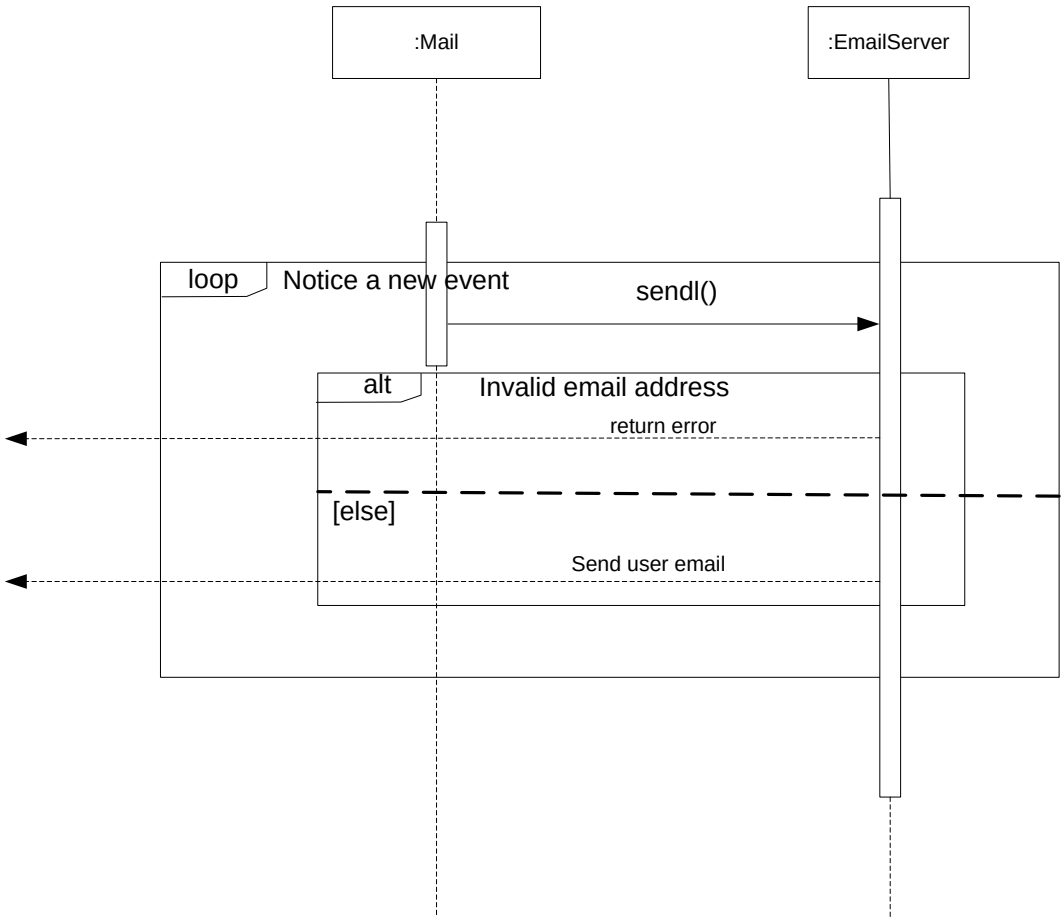
The above diagram shows Research Stocks use case. First user enters the desired stock symbol. If the symbol is valid, StockFacts queries Yahoo Finance for current price per share of the entered stock symbol as well as any available history about the stock, and then returns requested information to StockFacts. Finally, the system displays retrieved information to the user.

View Portfolio



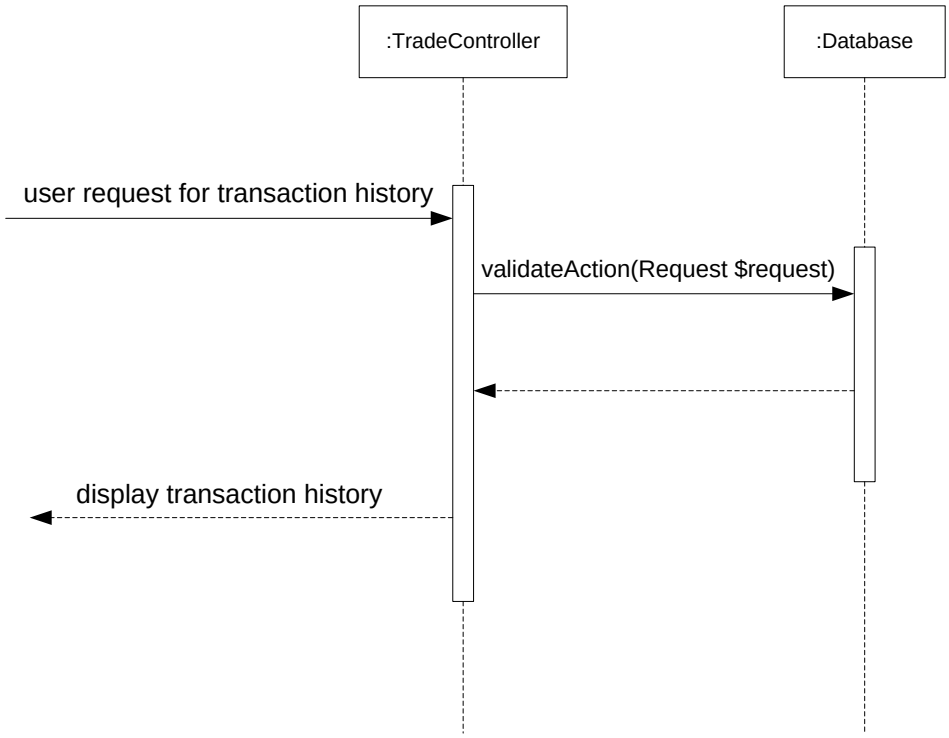
The above diagram shows the View Portfolio use case. After an user is logged into the system and asks for his/her portfolio, the PortfolioController would request information from the database and display the corresponding portfolio of the user.

Send Notifications



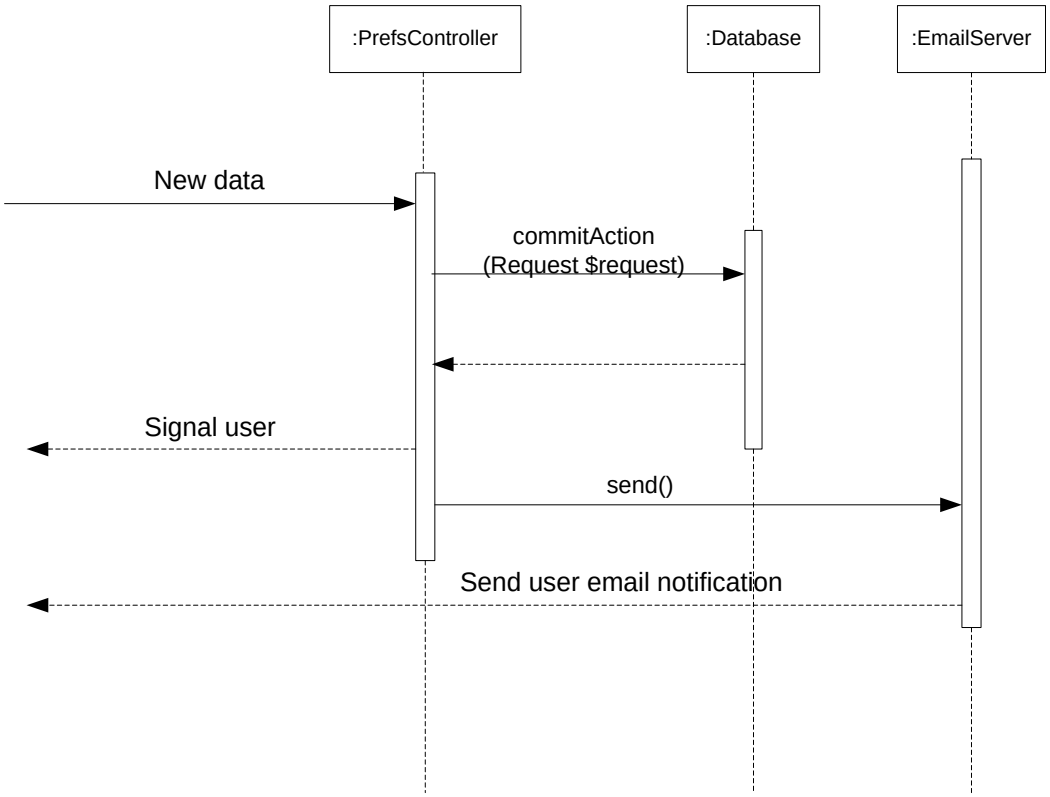
Send Notifications use case is showed as the above diagram. A loop is constantly waiting for a new event. After a new event is noticed, system signals email server (or SMS server) for sending notification to the user. If the email address is correct, then the user will be aware of what has happened.

View Transaction History



The above diagram shows the use case of View Transaction History. After a user is logged into the system and asks for his/her transaction history, the TradeController would request information from the database and show the corresponding transaction information to the user.

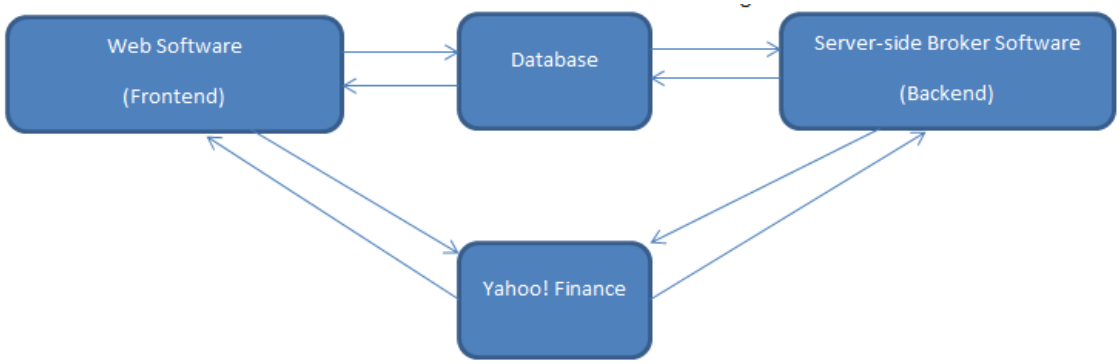
Preferences



The use case of Preferences is represented in the above diagram. After a user goes to preference page and enters the setting he/she wish to change, the PrefsController would send the new data to database. Besides, the email server will send an email to notify the user of successful operation.

12. Class Diagram and Interface Specification

The class diagrams and interface specifications were split up between the backend and the frontend. The rationale for doing this is that the front end and the backend never directly collaborate with one another. The frontend (i.e. client side, web side) inserts and extracts data from the database and also pulls data from Yahoo! Finance. The backend (i.e. server side, polling program) inserts and extracts data from the database and pulls data from Yahoo! Finance independent of the frontend. In this way, the database works as somewhat of a middleware between the two bodies of code as shown in the figure below.



a. Class Diagrams

The class diagrams have evolved since Report 2. This is because in between Demo 1 and Demo 2, code was redone to make it run more efficiently. Frontend code underwent an entire UI design, and as more was learned about Symfony2's functionality, the team tried to stick to the best practices of Symfony2 and the code often required revision. Also, testing was added in between Demo 1 and Demo 2, so the additional testing classes are shown below.

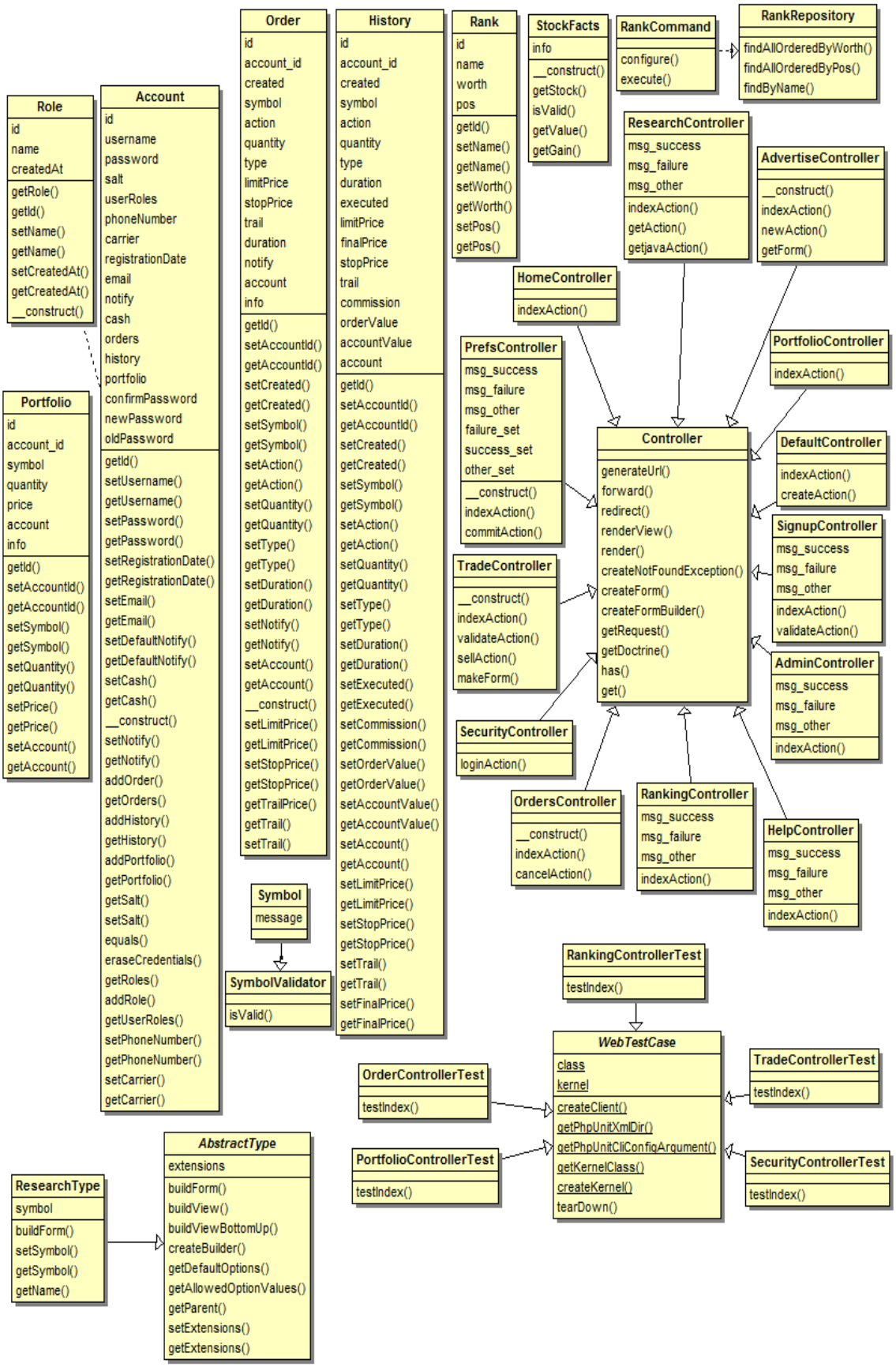
On the backend, new order types were added. Also, at the time of Demo 1, some errors were found in the way the code was operating together. The java code was restructured to run smoothly and be more loosely-coupled and highly cohesive.

Frontend Class Diagram

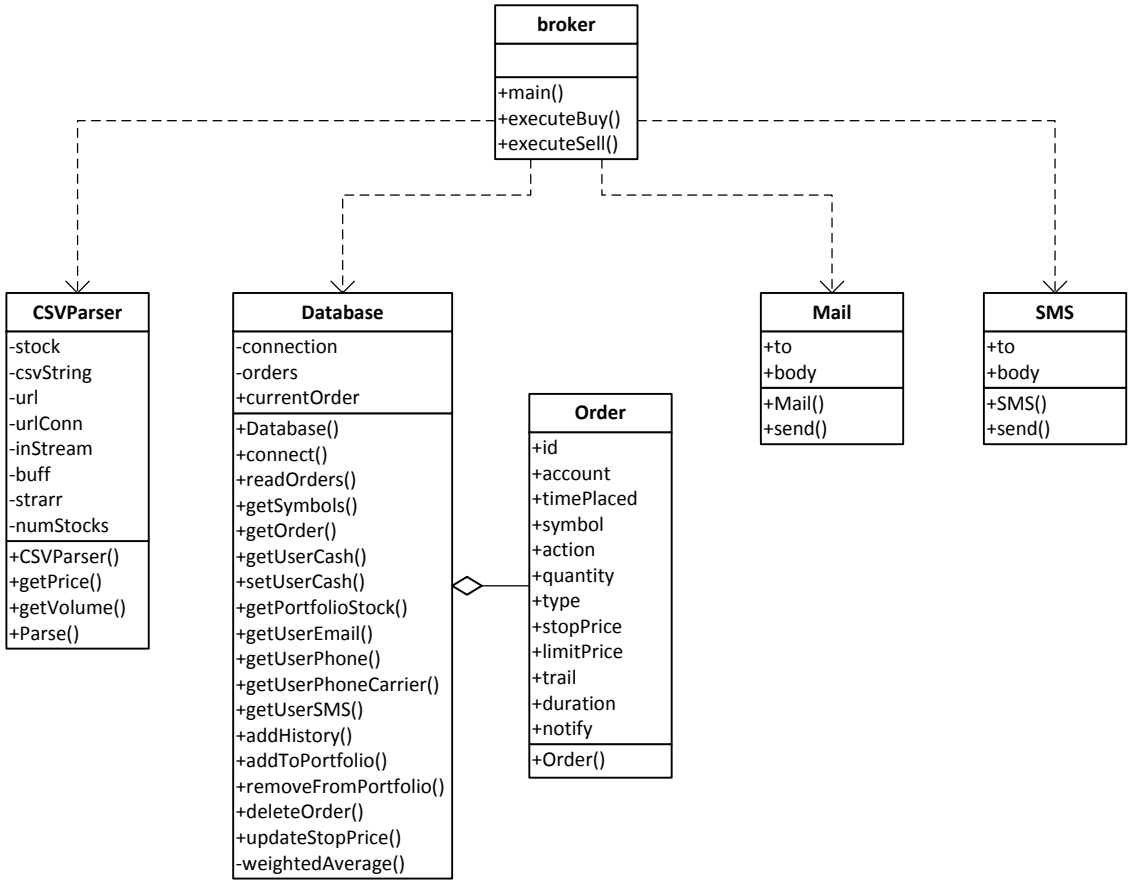
The entities are shown on the left of the Frontend Class Diagram. Each entity corresponds to a table in the database and has set/get methods to retrieve the corresponding fields of the database table. The relationship between entities and controllers is explained in Section VII.b.iii below. All controllers inherit from the base controller class. For the entities, only the Account class inherits from the UserInterface base class. Note that the 'indexAction()', 'retrieveAction()', functions are the basic building blocks that Symfony2 looks for to communicate with the database. As such, the function names are not as descriptive as the actual controller names. This is by design, to maintain consistency with the Symfony2 framework.

On the right of the Frontend Class Diagram are the base controller classes, which house all of the more complex PHP logic. Please note that while Symfony2 uses OO principles, making class diagrams for the different entities and controllers does not show as much of a program flow as simple flowcharts can show. The program flow diagrams are shown in Appendix A.2.

On the bottom right of the Frontend Class Diagram are the TestUseCase classes, which house all of the PHPUnit logic that tests the code. Please note that this is different than the validation scripts, which are not shown below since validation is done by Symfony2 code that was not written by Group 2. The testing code purposefully injects data into the Controllers and expects them to behave a certain way. A report is generated as to whether or not the tests behave as expected. The test writer defines what "expected" behavior is and can use assert statements or manually inspect the data that is inserted into the database. Please note, after generating this graphic, I actually realized that I forgot the class to implement the removal of orders at the end of each day that were "Good for the Day" orders. The graphic misses this class, however, it is listed in the function names below it. It is the class EndOfDayCommand. It has variables $input and $ouput, and has the functions configure() and execute(). I apologize for omitting this in the diagram.


Role
id
name
createdAt
getRole()
getId()
setName()
getName()
setCreatedAt()
getCreatedAt()
__construct()
Account
id
username
password
salt
userRoles
phoneNumber
carrier
registrationDate
email
notify
cash
orders
history
portfolio
confirmPassword
newPassword
oldPassword
getId()
setUsername()
getUsername()
setPassword()
getPassword()
setRegistrationDate()
getRegistrationDate()
setEmail()
getEmail()
setDefaultNotify()
getDefaultNotify()
setCash()
getCash()
__construct()
setNotify()
getNotify()
addOrder()
getOrders()
addHistory()
getHistory()
addPortfolio()
getPortfolio()
getSalt()
setSalt()
equals()
eraseCredentials()
getRoles()
addRole()
getUserRoles()
setPhoneNumber()
getPhoneNumber()
setCarrier()
getCarrier()
Portfolio
id
account_id
symbol
quantity
price
account
info
getId()
setAccountId()
getAccountId()
setSymbol()
getSymbol()
setQuantity()
getQuantity()
setPrice()
getPrice()
setAccount()
getAccount()
Order
id
account_id
created
symbol
action
quantity
type
limitPrice
stopPrice
trail
duration
notify
account
info
getId()
setAccountId()
getAccountId()
setCreated()
getCreated()
setSymbol()
getSymbol()
setAction()
getAction()
setQuantity()
getQuantity()
setType()
getType()
setDuration()
getDuration()
setNotify()
getNotify()
setAccount()
getAccount()
__construct()
setLimitPrice()
getLimitPrice()
setStopPrice()
getStopPrice()
getTrailPrice()
getTrail()
setTrail()
History
id
account_id
created
symbol
action
quantity
type
duration
executed
limitPrice
finalPrice
stopPrice
trail
commission
orderValue
accountValue
account
getId()
setAccountId()
getAccountId()
setCreated()
getCreated()
setSymbol()
getSymbol()
setAction()
getAction()
setQuantity()
getQuantity()
setType()
getType()
setDuration()
getDuration()
setExecuted()
getExecuted()
setCommission()
getCommission()
setOrderValue()
getOrderValue()
setAccountValue()
getAccountValue()
setAccount()
getAccount()
setLimitPrice()
getLimitPrice()
setStopPrice()
getStopPrice()
setTrail()
getTrail()
setFinalPrice()
getFinalPrice()
Rank
id
name
worth
pos
getId()
setName()
getName()
setWorth()
getWorth()
setPos()
getPos()
StockFacts
info
__construct()
getStock()
isValid()
getValue()
getGain()
RankCommand
configure()
execute()
RankRepository
findAllOrderedByWorth()
findAllOrderedByPos()
findByName()
ResearchController
msg_success
msg_failure
msg_other
indexAction()
getAction()
getjavaAction()
AdvertiseController
__construct()
indexAction()
newAction()
getForm()
HomeController
indexAction()
PortfolioController
indexAction()
PrefsController
msg_success
msg_failure
msg_other
failure_set
success_set
other_set
__construct()
indexAction()
commitAction()
Controller
generateUrl()
forward()
redirect()
renderView()
render()
createNotFoundException()
createForm()
createFormBuilder()
getRequest()
getDoctrine()
has()
get()
DefaultController
indexAction()
createAction()
SignupController
msg_success
msg_failure
msg_other
indexAction()
validateAction()
AdminController
msg_success
msg_failure
msg_other
indexAction()
TradeController
__construct()
indexAction()
validateAction()
sellAction()
makeForm()
SecurityController
loginAction()
OrdersController
__construct()
indexAction()
cancelAction()
RankingController
msg_success
msg_failure
msg_other
indexAction()
HelpController
msg_success
msg_failure
msg_other
indexAction()
Symbol
message
SymbolValidator
isValid()
RankingControllerTest
testIndex()
WebTestCase
class
kernel
createClient()
getPhpUnitXmlDir()
getPhpUnitCliConfigArgument()
getKernelClass()
createKernel()
tearDown()
OrderControllerTest
testIndex()
PortfolioControllerTest
testIndex()
TradeControllerTest
testIndex()
SecurityControllerTest
testIndex()
AbstractType
extensions
buildForm()
buildView()
buildViewBottomUp()
createBuilder()
getDefaultOptions()
getAllowedOptionValues()
getParent()
setExtensions()
getExtensions()
ResearchType
symbol
buildForm()
setSymbol()
getSymbol()
getName()

Backend Class Diagram



b. Data Types and Operation Signatures

Frontend

It is difficult to state the data types for PHP-based software. PHP is loosely-typed. This means that I can use a variable named $someData to represent an integer, a string, a double, or a Boolean. Therefore, it is not easy to identify each $variable as integer, string, etc. Instead, the variable names will be listed without specific types. On that same note, functions are also not represented by specific types. Instead, the word “function” usually proceeds a function in PHP.

Please note, this front end list has changed from Report 2. As the code evolved and more was learned about Symfony2, the function and variable names changed a bit. This list also includes the new Test classes which were implemented to use PHPUnit unit testing.

CLASS: PortfolioControllerTest
FUNCTION: testIndex()
CLASS: TradeControllerTest
FUNCTION: testIndex()
CLASS: OrderControllerTest
FUNCTION: testIndex()
CLASS: RankingControllerTest
FUNCTION: testIndex()
CLASS: SecurityControllerTest
FUNCTION: testIndex()
CLASS: TheStockHopExtension
FUNCTION: load(array$configs,ContainerBuilder$container)
CLASS: Configuration
FUNCTION: getConfigTreeBuilder()
CLASS: Portfolio
VARIABLE: $id
VARIABLE: $account_id
VARIABLE: $symbol
VARIABLE: $quantity
VARIABLE: $price
VARIABLE: $account
VARIABLE: $info
FUNCTION: getId()
FUNCTION: setAccountId($accountId)
FUNCTION: getAccountId()
FUNCTION: setSymbol($symbol)
FUNCTION: getSymbol()
FUNCTION: setQuantity($quantity)
FUNCTION: getQuantity()
FUNCTION: setPrice($price)
FUNCTION: getPrice()
FUNCTION:
setAccount(\RU\TheStockHopBundle\Entity\Account$account)
FUNCTION: getAccount()
CLASS: Role
VARIABLE: $id
VARIABLE: $name
VARIABLE: $createdAt
FUNCTION: getRole()
FUNCTION: getId()
FUNCTION: setName($name)

FUNCTION: getName()
FUNCTION: setCreatedAt($createdAt)
FUNCTION: getCreatedAt()
FUNCTION: __construct()
CLASS: Order
VARIABLE: $id
VARIABLE: $account_id
VARIABLE: $created
VARIABLE: $symbol
VARIABLE: $action
VARIABLE: $quantity
VARIABLE: $type
VARIABLE: $limitPrice
VARIABLE: $stopPrice
VARIABLE: $trail
VARIABLE: $duration
VARIABLE: $notify
VARIABLE: $account
VARIABLE: $info
FUNCTION: getId()
FUNCTION: setAccountId($accountId)
FUNCTION: getAccountId()
FUNCTION: setCreated($created)
FUNCTION: getCreated()
FUNCTION: setSymbol($symbol)
FUNCTION: getSymbol()
FUNCTION: setAction($action)
FUNCTION: getAction()
FUNCTION: setQuantity($quantity)
FUNCTION: getQuantity()
FUNCTION: setType($type)
FUNCTION: getType()
FUNCTION: setDuration($duration)
FUNCTION: getDuration()
FUNCTION: setNotify($notify)
FUNCTION: getNotify()
FUNCTION:
setAccount(\RU\TheStockHopBundle\Entity\Account$account)
FUNCTION: getAccount()
FUNCTION: __construct()
FUNCTION: setLimitPrice($limitPrice)
FUNCTION: getLimitPrice()

FUNCTION: setStopPrice($stopPrice)
FUNCTION: getStopPrice()
FUNCTION: getTrailPrice()
FUNCTION: getTrail()
FUNCTION: setTrail($trail)

CLASS: StockCache
VARIABLE: $id
VARIABLE: $symbol
VARIABLE: $name
VARIABLE: $sector
VARIABLE: $industry
VARIABLE: $price
VARIABLE: $updated
FUNCTION: getId()
FUNCTION: setSymbol($symbol)
FUNCTION: getSymbol()
FUNCTION: setName($name)
FUNCTION: getName()
FUNCTION: setSector($sector)
FUNCTION: getSector()
FUNCTION: setIndustry($industry)
FUNCTION: getIndustry()
FUNCTION: setPrice($price)
FUNCTION: getPrice()
FUNCTION: setUpdated($updated)
FUNCTION: getUpdated()

CLASS: Rank
VARIABLE: $id
VARIABLE: $name
VARIABLE: $worth
VARIABLE: $pos
FUNCTION: getId()
FUNCTION: setName($name)
FUNCTION: getName()
FUNCTION: setWorth($worth)
FUNCTION: getWorth()
FUNCTION: setPos($pos)
FUNCTION: getPos()

CLASS: History
VARIABLE: $id
VARIABLE: $account_id
VARIABLE: $created

VARIABLE: $symbol
VARIABLE: $action
VARIABLE: $quantity
VARIABLE: $type
VARIABLE: $duration
VARIABLE: $executed
VARIABLE: $limitPrice
VARIABLE: $finalPrice
VARIABLE: $stopPrice
VARIABLE: $trail
VARIABLE: $commission
VARIABLE: $orderValue
VARIABLE: $accountValue
VARIABLE: $account
FUNCTION: getId()
FUNCTION: setAccountId($accountId)
FUNCTION: getAccountId()
FUNCTION: setCreated($created)
FUNCTION: getCreated()
FUNCTION: setSymbol($symbol)
FUNCTION: getSymbol()
FUNCTION: setAction($action)
FUNCTION: getAction()
FUNCTION: setQuantity($quantity)
FUNCTION: getQuantity()
FUNCTION: setType($type)
FUNCTION: getType()
FUNCTION: setDuration($duration)
FUNCTION: getDuration()
FUNCTION: setExecuted($executed)
FUNCTION: getExecuted()
FUNCTION: setCommission($commission)
FUNCTION: getCommission()
FUNCTION: setOrderValue($orderValue)
FUNCTION: getOrderValue()
FUNCTION: setAccountValue($accountValue)
FUNCTION: getAccountValue()
FUNCTION:
setAccount(\RU\TheStockHopBundle\Entity\Account$account)
FUNCTION: getAccount()
FUNCTION: setLimitPrice($limitPrice)
FUNCTION: getLimitPrice()

FUNCTION: setStopPrice($stopPrice)
FUNCTION: getStopPrice()
FUNCTION: setTrail(\int$trail)
FUNCTION: getTrail()
FUNCTION: setFinalPrice($finalPrice)
FUNCTION: getFinalPrice()

CLASS: StockCache
VARIABLE: $id
VARIABLE: $symbol
VARIABLE: $name
VARIABLE: $sector
VARIABLE: $industry
VARIABLE: $price
VARIABLE: $updated
FUNCTION: getId()
FUNCTION: setSymbol($symbol)
FUNCTION: getSymbol()
FUNCTION: setName($name)
FUNCTION: getName()
FUNCTION: setSector($sector)
FUNCTION: getSector()
FUNCTION: setIndustry($industry)
FUNCTION: getIndustry()
FUNCTION: setPrice($price)
FUNCTION: getPrice()
FUNCTION: setUpdated($updated)
FUNCTION: getUpdated()

CLASS: Role
VARIABLE: $id
VARIABLE: $name
VARIABLE: $createdAt
FUNCTION: getRole()
FUNCTION: getId()
FUNCTION: setName($name)
FUNCTION: getName()
FUNCTION: setCreatedAt($createdAt)
FUNCTION: getCreatedAt()
FUNCTION: __construct()

CLASS: Account
VARIABLE: $id
VARIABLE: $username
VARIABLE: $password

VARIABLE: $salt
VARIABLE: $userRoles
VARIABLE: $phoneNumber
VARIABLE: $carrier
VARIABLE: $registrationDate
VARIABLE: $email
VARIABLE: $notify
VARIABLE: $cash
VARIABLE: $orders
VARIABLE: $history
VARIABLE: $portfolio
VARIABLE: $confirmPassword
VARIABLE: $newPassword
VARIABLE: $oldPassword
FUNCTION: getId()
FUNCTION: setUsername($username)
FUNCTION: getUsername()
FUNCTION: setPassword($password)
FUNCTION: getPassword()
FUNCTION: setRegistrationDate($registrationDate)
FUNCTION: getRegistrationDate()
FUNCTION: setEmail($email)
FUNCTION: getEmail()
FUNCTION: setDefaultNotify($defaultNotify)
FUNCTION: getDefaultNotify()
FUNCTION: setCash($cash)
FUNCTION: getCash()
FUNCTION: __construct()
FUNCTION: setNotify($notify)
FUNCTION: getNotify()
FUNCTION: addOrder(\RU\TheStockHopBundle\Entity\Order$orders)
FUNCTION: getOrders()
FUNCTION:
addHistory(\RU\TheStockHopBundle\Entity\History$history)
FUNCTION: getHistory()
FUNCTION:
addPortfolio(\RU\TheStockHopBundle\Entity\Portfolio$portfolio)
FUNCTION: getPortfolio()
FUNCTION: getSalt()
FUNCTION: setSalt($value)
FUNCTION: equals(UserInterface$user)
FUNCTION: eraseCredentials()

FUNCTION: getRoles()
FUNCTION: addRole(\RU\TheStockHopBundle\Entity\Role$userRoles)
FUNCTION: getUserRoles()
FUNCTION: setPhoneNumber($phoneNumber)
FUNCTION: getPhoneNumber()
FUNCTION: setCarrier($carrier)
FUNCTION: getCarrier()

CLASS: Account

VARIABLE: $id
VARIABLE: $username
VARIABLE: $password
VARIABLE: $salt
VARIABLE: $userRoles
VARIABLE: $phoneNumber
VARIABLE: $carrier
VARIABLE: $registrationDate
VARIABLE: $email
VARIABLE: $notify
VARIABLE: $cash
VARIABLE: $orders
VARIABLE: $history
VARIABLE: $portfolio
VARIABLE: $confirmPassword
VARIABLE: $newPassword
VARIABLE: $oldPassword
FUNCTION: getId()
FUNCTION: setUsername($username)
FUNCTION: getUsername()
FUNCTION: setPassword($password)
FUNCTION: getPassword()
FUNCTION: setRegistrationDate($registrationDate)
FUNCTION: getRegistrationDate()
FUNCTION: setEmail($email)
FUNCTION: getEmail()
FUNCTION: setDefaultNotify($defaultNotify)
FUNCTION: getDefaultNotify()
FUNCTION: setCash($cash)
FUNCTION: getCash()
FUNCTION: __construct()
FUNCTION: setNotify($notify)
FUNCTION: getNotify()
FUNCTION: addOrder(\RU\TheStockHopBundle\Entity\Order$orders)

FUNCTION: getOrders()
FUNCTION: addHistory(\RU\TheStockHopBundle\Entity\History$history)
FUNCTION: getHistory()
FUNCTION: addPortfolio(\RU\TheStockHopBundle\Entity\Portfolio$portfolio)
FUNCTION: getPortfolio()
FUNCTION: getSalt()
FUNCTION: setSalt($value)
FUNCTION: equals(UserInterface$user)
FUNCTION: eraseCredentials()
FUNCTION: getRoles()
FUNCTION: addRole(\RU\TheStockHopBundle\Entity\Role$userRoles)
FUNCTION: getUserRoles()
FUNCTION: setPhoneNumber($phoneNumber)
FUNCTION: getPhoneNumber()
FUNCTION: setCarrier($carrier)
FUNCTION: getCarrier()

CLASS: History
VARIABLE: $id
VARIABLE: $account_id
VARIABLE: $created
VARIABLE: $symbol
VARIABLE: $action
VARIABLE: $quantity
VARIABLE: $type
VARIABLE: $duration
VARIABLE: $executed
VARIABLE: $limitPrice
VARIABLE: $finalPrice
VARIABLE: $stopPrice
VARIABLE: $trail
VARIABLE: $commission
VARIABLE: $orderValue
VARIABLE: $accountValue
VARIABLE: $account
FUNCTION: getId()
FUNCTION: setAccountId($accountId)
FUNCTION: getAccountId()
FUNCTION: setCreated($created)
FUNCTION: getCreated()
FUNCTION: setSymbol($symbol)

FUNCTION: getSymbol()
FUNCTION: setAction($action)
FUNCTION: getAction()
FUNCTION: setQuantity($quantity)
FUNCTION: getQuantity()
FUNCTION: setType($type)
FUNCTION: getType()
FUNCTION: setDuration($duration)
FUNCTION: getDuration()
FUNCTION: setExecuted($executed)
FUNCTION: getExecuted()
FUNCTION: setCommission($commission)
FUNCTION: getCommission()
FUNCTION: setOrderValue($orderValue)
FUNCTION: getOrderValue()
FUNCTION: setAccountValue($accountValue)
FUNCTION: getAccountValue()
FUNCTION:
setAccount(\RU\TheStockHopBundle\Entity\Account$account)
FUNCTION: getAccount()
FUNCTION: setLimitPrice($limitPrice)
FUNCTION: getLimitPrice()
FUNCTION: setStopPrice($stopPrice)
FUNCTION: getStopPrice()
FUNCTION: setTrail(\int$trail)
FUNCTION: getTrail()
FUNCTION: setFinalPrice($finalPrice)
FUNCTION: getFinalPrice()
CLASS: Order
VARIABLE: $id
VARIABLE: $account_id
VARIABLE: $created
VARIABLE: $symbol
VARIABLE: $action
VARIABLE: $quantity
VARIABLE: $type
VARIABLE: $limitPrice
VARIABLE: $stopPrice
VARIABLE: $trail
VARIABLE: $duration
VARIABLE: $notify
VARIABLE: $account

VARIABLE: $info
FUNCTION: getId()
FUNCTION: setAccountId($accountId)
FUNCTION: getAccountId()
FUNCTION: setCreated($created)
FUNCTION: getCreated()
FUNCTION: setSymbol($symbol)
FUNCTION: getSymbol()
FUNCTION: setAction($action)
FUNCTION: getAction()
FUNCTION: setQuantity($quantity)
FUNCTION: getQuantity()
FUNCTION: setType($type)
FUNCTION: getType()
FUNCTION: setDuration($duration)
FUNCTION: getDuration()
FUNCTION: setNotify($notify)
FUNCTION: getNotify()
FUNCTION: setAccount(\RU\TheStockHopBundle\Entity\Account$account)
FUNCTION: getAccount()
FUNCTION: __construct()
FUNCTION: setLimitPrice($limitPrice)
FUNCTION: getLimitPrice()
FUNCTION: setStopPrice($stopPrice)
FUNCTION: getStopPrice()
FUNCTION: getTrailPrice()
FUNCTION: getTrail()
FUNCTION: setTrail($trail)

CLASS: Portfolio
VARIABLE: $id
VARIABLE: $account_id
VARIABLE: $symbol
VARIABLE: $quantity
VARIABLE: $price
VARIABLE: $account
VARIABLE: $info
FUNCTION: getId()
FUNCTION: setAccountId($accountId)
FUNCTION: getAccountId()
FUNCTION: setSymbol($symbol)
FUNCTION: getSymbol()

FUNCTION: setQuantity($quantity)
FUNCTION: getQuantity()
FUNCTION: setPrice($price)
FUNCTION: getPrice()
FUNCTION: setAccount(\RU\TheStockHopBundle\Entity\Account$account)
FUNCTION: getAccount()

CLASS: Rank
VARIABLE: $id
VARIABLE: $name
VARIABLE: $worth
VARIABLE: $pos
FUNCTION: getId()
FUNCTION: setName($name)
FUNCTION: getName()
FUNCTION: setWorth($worth)
FUNCTION: getWorth()
FUNCTION: setPos($pos)
FUNCTION: getPos()

CLASS: AdvertiseController
FUNCTION: __construct()
FUNCTION: indexAction(Request$request)
FUNCTION: newAction(Request$request)
FUNCTION: getForm()

CLASS: PrefsController
VARIABLE: $msg_success
VARIABLE: $msg_failure
VARIABLE: $msg_other
VARIABLE: $failure_set
VARIABLE: $success_set
VARIABLE: $other_set
FUNCTION: __construct()
FUNCTION: indexAction(Request$request)
FUNCTION: commitAction(Request$request)

CLASS: ResearchController
VARIABLE: $msg_success
VARIABLE: $msg_failure
VARIABLE: $msg_other
FUNCTION: indexAction()
FUNCTION: getAction(Request$request)
FUNCTION: getjavaAction($name="")

CLASS: HelpController

VARIABLE: $msg_success
VARIABLE: $msg_failure
VARIABLE: $msg_other
FUNCTION: indexAction()
CLASS: PortfolioController
FUNCTION: indexAction(Request$request)
CLASS: DefaultController
FUNCTION: indexAction()
FUNCTION: createAction()
CLASS: SecurityController
FUNCTION: loginAction()
CLASS: OrdersController
FUNCTION: __construct()
FUNCTION: indexAction()
FUNCTION: cancelAction($id)
CLASS: TradeController
VARIABLE: $msg_success
VARIABLE: $msg_failure
VARIABLE: $msg_other
VARIABLE: $failure_set
VARIABLE: $success_set
VARIABLE: $other_set
FUNCTION: __construct()
FUNCTION: indexAction(Request$request)
FUNCTION: validateAction(Request$request)
FUNCTION: sellAction(Request$request)
CLASS: HomeController
FUNCTION: indexAction()
CLASS: RankingController
VARIABLE: $msg_success
VARIABLE: $msg_failure
VARIABLE: $msg_other
FUNCTION: indexAction(Request$request)
CLASS: AdminController
VARIABLE: $msg_success
VARIABLE: $msg_failure
VARIABLE: $msg_other
FUNCTION: indexAction()
CLASS: SignupController
VARIABLE: $msg_success
VARIABLE: $msg_failure
VARIABLE: $msg_other

FUNCTION: indexAction(Request$request)
FUNCTION: validateAction(Request$request)
CLASS: TradeController
FUNCTION: __construct()
FUNCTION: indexAction(Request$request)
FUNCTION: validateAction(Request$request)
FUNCTION: sellAction(Request$request)
FUNCTION: makeForm($order)
CLASS: Symbol
VARIABLE: $message
CLASS: SymbolValidator
FUNCTION: isValid($value,Constraint$constraint)
CLASS: ResearchType
VARIABLE: $symbol
FUNCTION: buildForm(FormBuilder$builder,array$options)
FUNCTION: setSymbol($data="")
FUNCTION: getSymbol()
FUNCTION: getName()
CLASS: OrderActionType
FUNCTION: buildForm(FormBuilder$builder,array$options)
FUNCTION: getDefaultOptions(array$options)
FUNCTION: getName()
CLASS: FixtureLoader
FUNCTION: load($manager)
CLASS: RankRepository
FUNCTION: findAllOrderedByWorth()
FUNCTION: findAllOrderedByPos()
FUNCTION: findByName($user)
CLASS: StockFacts
VARIABLE: $info
FUNCTION: __construct()
FUNCTION: getStock($stock)
FUNCTION: isValid()
FUNCTION: getValue($qty=1)
FUNCTION: getGain($cost=1)
CLASS: TheStockHopBundle
CLASS: RankCommand
FUNCTION: configure()
FUNCTION: execute(InputInterface$input,OutputInterface$output)
CLASS: EndOfDayCommand
FUNCTION: configure()
FUNCTION: execute(InputInterface$input,OutputInterface$output)

Backend

Operations
+ main() : void
+executeBuy(database : Database, currentOrder : Order, p : CSVParser, sellComission : double, buyCommission : double) : void
+executeSell(database : Database, currentOrder : Order, p : CSVParser, sellComission : double, buyCommission : double) : void

CSVParser
Attributes
- stock : String
- csvString : String
- url : URL
- urlConn : URLConnection
- inStream : InputStreamReader
- buff : BufferedReader
- strarr[][] : String
- numStocks : int
Operations
+ CSVParser() : void
+ CSVParser(st : String) : void
+ getPrice (symbol : String) : double
+ getVolume (symbol : String) : double
+ Parse () : void

Database
Attributes
- connection : Connection
- orders : ResultSet
+ currentOrder : Order
Operations
+ Database ()
+ Database (dbName : String, dbUser : String, dbPassword : String)
+ connect (dbName : String, dbUser : String, dbPassword : String) : void
+ readOrders () : void

+ getSymbols () : String
+ getOrder () : Order
+ getUserCash (account : int) : BigDecimal
+ setUserCash (account : int, cashMoney : BigDecimal) : void
+ getPortfolioStock (account : int, symbol : String) : long
+ getUserEmail (account : int) : String
+ getUserPhone(account : int) : String
+ getUserPhoneCarrier(account : int) : String
+ getUserSMS(account : int) : String
+ addHistory (account : int, timePlaced : Timestamp, symbol : String, action : String, quantity : long, type : String, stopPrice : BigDecimal, limitPrice : BigDecimal, trail : BigDecimal, duration : String, price : BigDecimal, commission : BigDecimal, orderValue : BigDecimal, accountValue : BigDecimal) : void
+ addToPortfolio (account : int, symbol : String, quantity : long, price : BigDecimal) : void
+ removeFromPortfolio (account : int, symbol : String, quantity : long) : void
+ deleteOrder (id : long) : void
+ updateStopPrice(id : int, stopPrice : BigDecimal) : void
- weightedAverage (price1 : BigDecimal, quantity1Long : long, price2 : BigDecimal, quantity2long : long) : BigDecimal

Order

Attributes

+ id : long
+ account : int
+ timePlaced : Timestamp
+ symbol : String
+ action : String
+ quantity : long
+ type : String
+ stopPrice : BigDecimal
+ limitPrice : BigDecimal
+ trail : BigDecimal
+ duration : String
+ notify : byte

Operations

+ Order ()
+ Order (id : long, account : int, timePlaced : Timestamp, symbol : String, action : String, quantity : long, type : String, stopPrice : BigDecimal, limitPrice : BigDecimal, trail : BigDecimal, duration : String, notify : String)

Mail

Attributes
+ to : String
+body : String
Operations
+ Mail()
+ Mail(mailto : String, mailBody : String)
+ send () : void

SMS
Attributes
+ to : String
+body : String
Operations
+ SMS()
+ SMS(mailto : String, mailBody : String)
+ send () : void

c. Design Patterns

Symfony2 utilizes the Model-View-Controller design pattern, and the front-end controller design pattern. The frontend also utilized inheritance through a testing class. The testing class used the builder method to facilitate easy testing by injection of test objects into the PHP code. In addition, Symfony2 was used to add an abstraction layer to the database. In common day web coding, this abstraction layer (or ORM) is known as an ORM Design Pattern. Though these do not follow the classical design patterns, they do follow the new idea of web-based design patterns.

The backend of the system uses the proxy design pattern extensively to shield the core logic and calculation portions from the interaction with the database, Yahoo Finance, and notifications. There are effectively four separate proxy style modules that we developed and incorporated into the design. The primary one is the Database module which handles all calls and interaction between the “broker” program and the MySQL database. The database interaction code could have been hard coded into the main broker class; however, we decided to separate this code into its own class and have the broker program interact with it via our own “standard” methods would facilitate in easier coding and maintenance of the broker program. This became very convenient when we decided to incorporate additional stock order types which necessitated some significant changes to the orders and history tables of the database. Because the broker code was written to call our own custom methods in the Database class, the heavy code changes needed to interact with the modified tables were kept within the Database class. As such, all the code in the broker class for handling the original order

types remained unchanged, and therefore was guaranteed to still work as expected. Only additional code had to be written to handle the new order types. The other classes that were designed in a proxy style were the CSVParser (for interacting with Yahoo Finance), Mail (for sending email notifications), and SMS (for sending text messages) classes. In a similar manner to the Database class, any changes we needed to make to improve our system interaction with the outside world or fix newly discovered issues did not affect the core function of the backend software.

By nature of the problem, the backend uses state based design patterns as well. All the different order types are stored with the same sequence of data types in the MySQL database. Each order is retrieved in the same manner, regardless of whether it is a "market buy" order, "limit sell" order, or any other type of order. However, once the type of order is known, the backend enters the appropriate state to process the order according to the correct rules it needs to. This state is naturally determined by the current order type. The state determines whether the order is going to follow a buy or sell route, as well as the conditions which must be satisfied to process the order. Once the order has been executed accordingly, the backend returns to normal operation, saving the order completion information in the history table and removing the order from the orders table before the next order is fetched.

d. Object Constraint Language (Contracts)

Group 2 feels that OCL is most useful before you actually code an entire project. It is difficult to reverse engineer OCL contracts from already written code. OCL is meant as extra constraints that are not captured by UML class diagrams. They should be required for Report 2, prior to the students coding up the entire project. It was difficult to write up constraints at the end of the project.

Additionally, Group 2 frontend code does not interact with the database in the same way that one might expect, due to the abstraction level that we added. The database abstraction layer enforces its own contracts by using set and get methods for everything in the database. Instead of writing the OCL Contracts for the frontend, a sampling of the OCL contracts were instead written for the backend code that interacts with the database.

context Database::readOrders() : void
pre: connection.createStatement() != NULL
post: result = stmt.executeQuery("SELECT * FROM orders ORDER BY time_placed");
result.isEmpty()

context Database::getSymbols () : String
pre: orders.next()
let entries: symbols.indexOf(orders.getString("symbol")) > 0 **then**
    continue;
if symbols != “” **then**
     symbols += “+”
post: result += orders.getString(“symbol”)

context Database::getOrder() : void
pre: connection.createStatement() != NULL
post: result = stmt.executeQuery("SELECT * FROM orders ORDER BY time_placed");
          result.isEmpty()

context Database::getUserEmail(int account) : String
pre: connection.prepareStatement() != NULL
post: result = connection.prepareStatement("SELECT email FROM accounts WHERE id=?");
      result.isEmpty()

context Database::addHistory(int account, Timestamp timePlaced, String symbol, String action, long quantity, String type, BigDecimal stopPrice, BigDecimal limitPrice, BigDecimal trail, String duration, BigDecimal price, BigDecimal commission, BigDecimal orderValue, BigDecimal accountValue) : void
post: prepStmt = connection.prepareStatement("INSERT INTO history (account_id,time_placed,symbol,action,quantity,type,stop_price,limit_price,trail,duration,price,commission,order_value,account_value,time_executed) VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)");

13. System Architecture and System Design

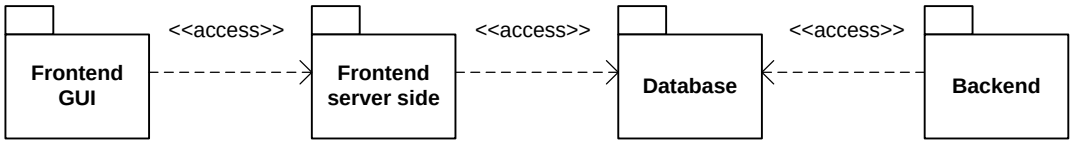
a. Architectural Styles

Before speaking of the kinds of architectural styles we used in thestockhop, it is useful to provide a definition of an architectural style. “An architectural style … defines a family of systems in terms or a pattern of structural organization. More specifically, an architectural style determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be combined (Garlan and Shaw, 6).”

Web applications usually combine a mix of architectural styles. The frontend design uses the Model-View-Controller architecture, which is considered to be a separated-presentation architectural style. The separated-presentation architectural style describes the separation of presentation code from internal logic code, which is exactly what has been done with the use of Symfony2 and doctrine. It is also using the client/server architectural style. All of the data to run the application is stored centrally on thestockhop server, but many clients can access the web application from different places around the world through many different Internet browsers.

Both the frontend and the backend are using the component-based architectural style by using design and development languages that allows them to be run independent of the platform they are on. In this way, the code gets great reusability and allows for growth and scalability. Both ends are also using the object-oriented architectural style to increase code modularity and readability. Sometimes, the Representational State Transfer (REST) (described in Section III.b) is described as an architectural style when it is thought of as being comprised of a uniform interface and a layered architectural style. In this way, the use of Symfony2 in a conventional manner also uses the REST architectural style.

b. Identifying Subsystems



The frontend GUI provides the interface by which a user can interact with the system and view data from the system. The front end server side retrieves data

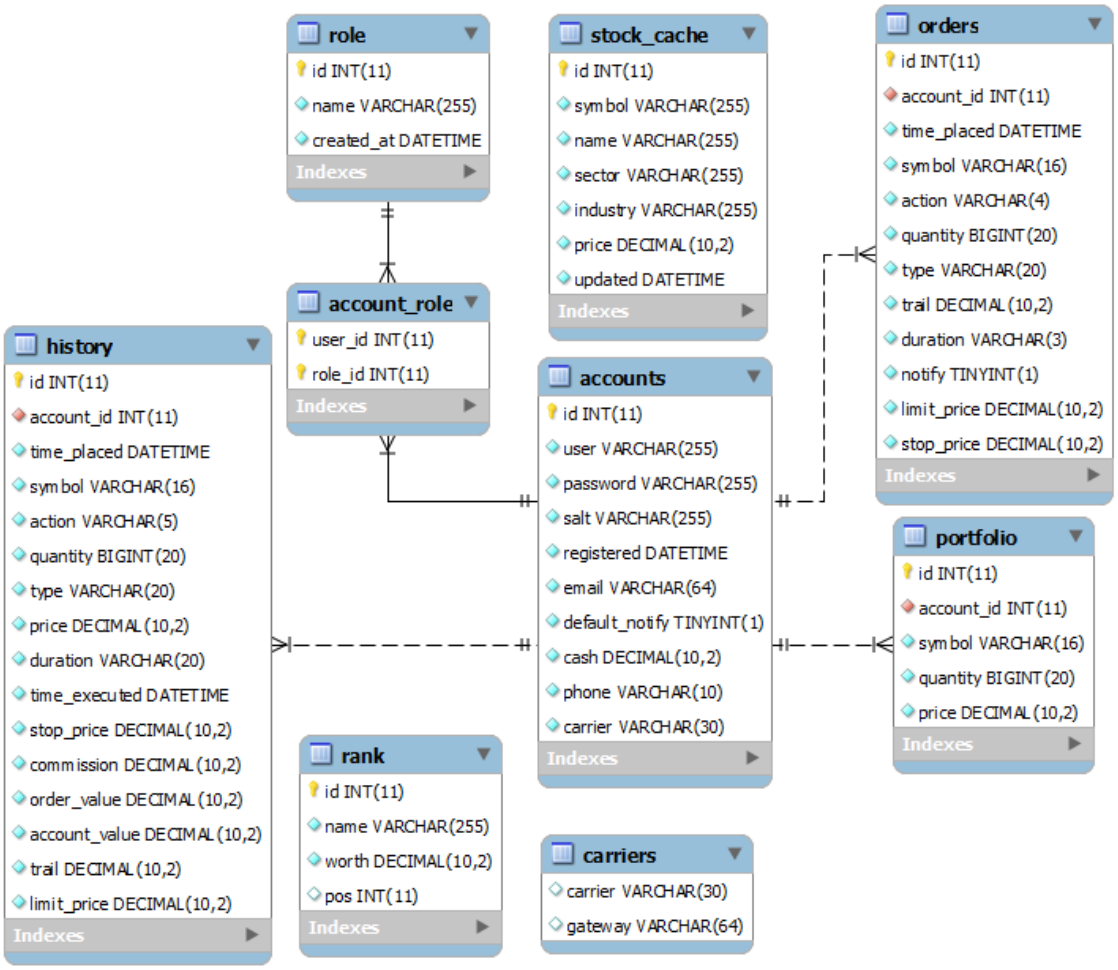
from the database and Yahoo Finance and presents it to the user via the frontend GUI. It also stores new stock orders in the database. The database holds all the persistent information such as user accounts, portfolios, the transaction history, and the list of open stock orders to be processed. The backend handles open order execution, user notification of completed orders via email and/or SMS, and updates account information, portfolios, and transaction histories appropriately.

c. Mapping Subsystems to Hardware

While the server is contained to one machine, the system as a whole is spread across different machines. The system is effectively split into two separate and fairly independent sections, a frontend and a backend, with the database (residing on the server) acting as the intermediary between the two. The frontend is further subdivided into a GUI component which runs within a web browser on a client's computer (or realistically, many clients' computers) providing a rich interactive experience and the server side portion runs within the web server process on the server. The server side frontend handles interaction between the GUI and the database, such as retrieving portfolio contents and ensuring orders are properly and legitimately entered into the database. On the backend, there is only one process we call the "broker" which handles proper order execution and user notification of completed orders. This process runs on the server alongside the database and the server side half of the frontend.

d. Persistent Data Storage

A MySQL relational database is used for persistent data storage. There are nine tables within the MySQL database entitled thestockhop. These tables are as follows: accounts, account_role, role, orders, history, portfolio, rank, stock_cache, and carriers. The figure below shows the database schema mapping including field names and types.



The main table is the accounts table. This stores the user's information such as username, password (encrypted), cash balance, etc. In the figure, the field ‘salt’ is used for the advanced encryption algorithm for the user passwords. This accounts table also holds the primary key based on user. Each user has a unique ‘id’ associated with their account. This id is the primary key. All subsequent tables use ‘account_id’ which is a foreign key that refers to the accounts table id. Therefore, an entry in the “Portfolio” table is associated with a particular user by the use of the foreign key (which relates to the accounts table primary key to get the user data). The decision was made not to repeat the username and data in every table to reduce redundancy and also to follow closely with Codd’s 12 Rules for Relational Databases (“Codd’s 12 Rules”, Wikipedia).

Additionally, the account_role and role tables are used as a Many-to-Many relationship with the accounts table, because many users can have many roles, whereas in something like the Orders table, one order can only have one user

associated with it. The purpose of these roles tables is to record the different roles of users. For example, “username: group2” could have the role of “user” *AND* the role of “administrator”. Or, more likely, a user might have the role of both “user” and “advertiser”. This particular person needs to be able to have access to the stock trading game and also to the interface to upload their advertisements.

The orders table is used to store open orders for all users. Each individual entry is traced back to a particular user using the foreign key. The system has a 20 minute delay before it processes any orders (Market, Stop, or Limit) due to the fact that Yahoo! Finance data is 20 minute delayed. If users had access to real-time stock market data, they would be able to cheat at thestockhop.com. The frontend software inserts the information filled out from the trade form into the open orders database, and the backend software constantly polls the open orders database to see if new data has appeared.

The history table stores transaction history for all users. The portfolio table stores the stocks owned by particular users and their purchase price. Similar to the other tables, a user is identified through the use of foreign keys to index the accounts table.

The carriers table is actually not used in connection with other databases and does not follow Codd’s 12 Rules. It is instead used as a matter of convenience to globally store an array that can be seen by all the software. It is used specifically for SMS text messaging. In order to text message users, the software emails their number at the correct carrier relay, i.e. a Verizon customer with the phone number 555-555-5555 can be text messaged by emailing 5555555555@vtext.com.

e. Network Protocol

TheStockHop is self-contained to one server. Running on the localhost, there is a MySQL database server, a POSTFIX mail server, and an Apache web server. There are only 4 different external connections needed for the server.

First, the website is accessed via incoming HTTP requests on port 80, this is an Internet standard, all user interactions happen over the frontend web interface.

Both the backend and frontend make outgoing HTTP requests to Yahoo! Finance in order to gather the necessary financial data. The reason Group 2 used HTTP is because that it is what Yahoo! Finance provides for sessionless one time queries of its data, and there was no further interaction with the Yahoo! Finance website than that.

The email notification uses SSL encryption on outgoing port 587 for the simple mail transport protocol (SMTP). SMTP is an internet standard for sending mail. The reason we use port 587 and encrypt the outgoing mail connection is due to the fact that the current ISP the server is on blocks outgoing mail connections on port 25. We relay our mail through Google Mail servers using the port 587 method.

Internally, there are different connections needed for application communication. The frontend relies on PHP's internal drivers to connect to a MYSQL database for access to the persistent data. The backend uses the JDBC driver to connect to the MySQL database and the Javamail driver to connect to the mail server.

f. Global Control Flow

Our system is event-driven in that it waits for users' input and acts according to it. User interaction is required in frontend features like logging in, placing a trade, etc. The system does not log anyone in until the initial event is driven by an external request.

Also, we have a timer run in our backend program to fetch user orders and stock information. Since the information on Yahoo Finance is twenty minutes delayed for most of people, the timer in our system is also used to make user orders are executed twenty minutes delay for users to play fair.

For time dependency, the backend of our system periodically processes pending orders and checks the preconditions for different trades. The system has multiple threads to support multiple users operating at the same time and periodically fetch orders.

g. Hardware Requirements

Server-Side

Note that a complete scalability analysis has not been performed, so the server-side hardware requirements are based on the needs of the current website. The website requires an external hard drive. This hard drive must be capable of storing the database data. An internet connection is required, so necessary hardware includes a network card.

The hardware profile for the current thestockhop.com server is shown in the Appendix as A.1. This hardware profile is everything needed for server maintenance and administration on the current server.

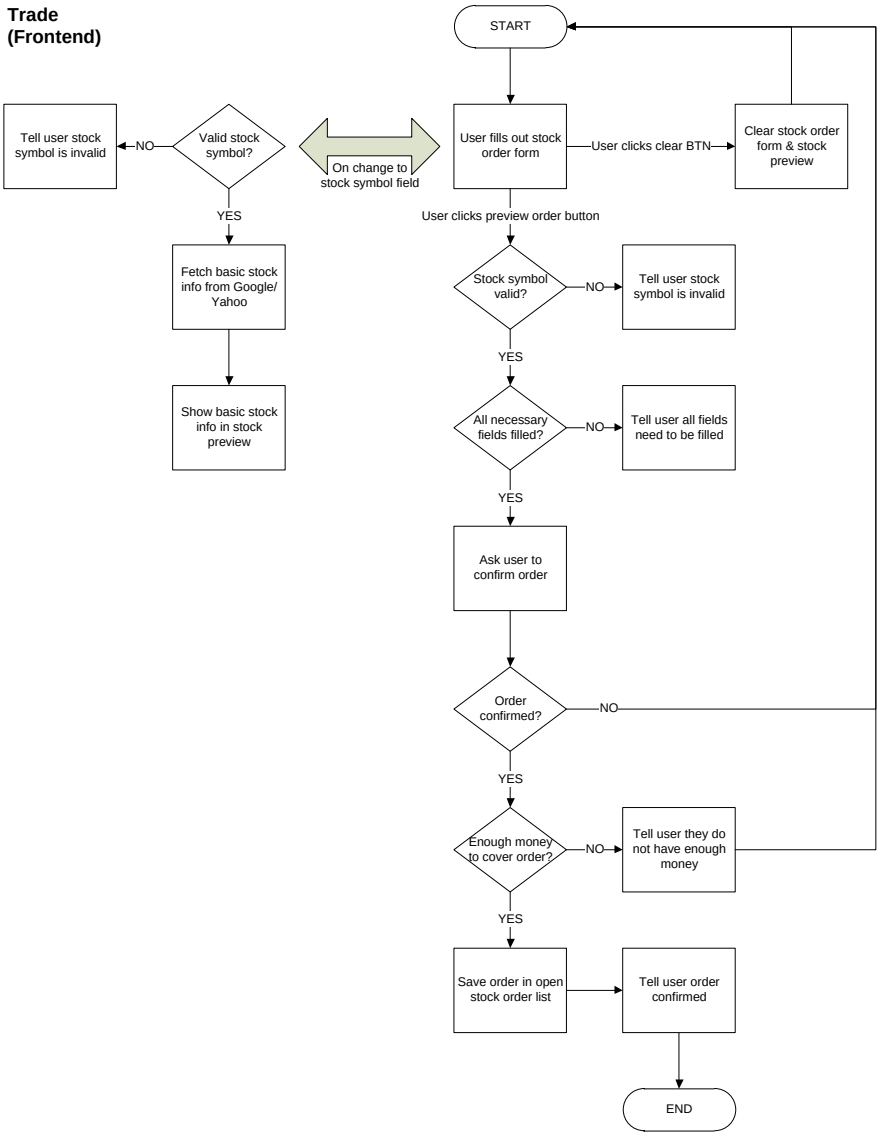
Client-Side

On the client side, the client needs to have a monitor to view the GUI Display. In addition, since a web browser is required, the other hardware that comprises a computer is required such as CPU, Graphics Card, RAM, keyboard, mouse, etc. A web connection is also required, so the computer must contain a network card (whether wireless or Ethernet).

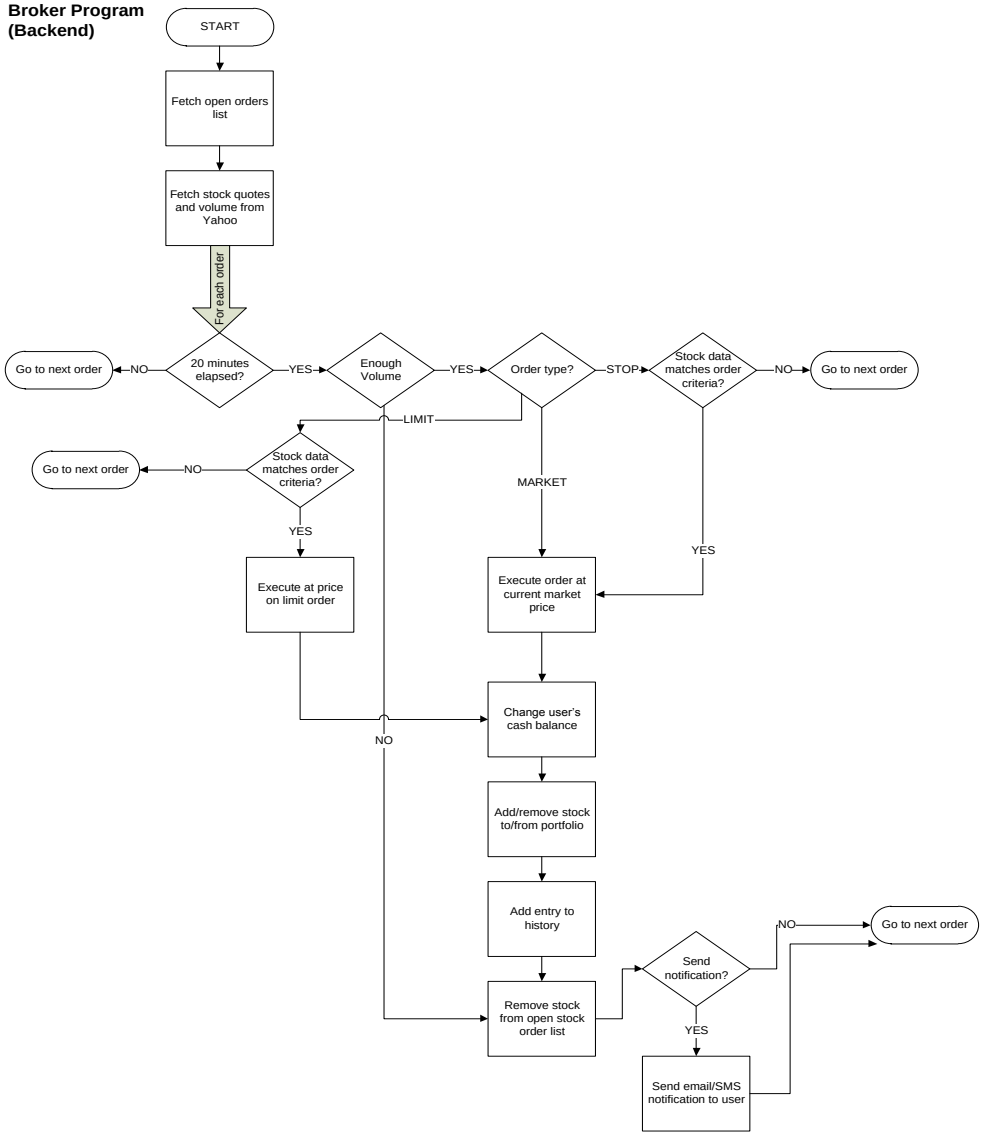
h. Program Flow Diagrams

Before constructing class diagrams or interaction diagrams, Group 2 wished to capture the basic Program Flow using simple flowcharts. It was decided to incorporate these as part of Report 2, since it was part of our design process. Within the documentation, we will show the logic for the front end trade screen and then the backend “Broker” program since these flowcharts explain the most important functionality to TheStockHop. There are more flowcharts for other cases in Appendix A.2.

Trade
(Frontend)



Broker Program (Backend)



14. Algorithms and Data Structures

a. Algorithms

TheStockHop does not currently use many complex algorithms. In future implementations, moving average analysis will be done server-side using more complex data mapping tools. One interesting aspect that is almost algorithmic is the way the trailing stop order is treated. In a trailing stop, the stop order is recomputed as the market price moves. Upon periodic queries to Yahoo! Finance, the Open Orders table that contains the trailing stop orders slides the "Stop Target Price" by the trail amount (by slides, we mean that it takes the new Market Price, applies the trail amount, and obtains a new 'stop target price').

b. Data Structures

i. "Order" Data Type [Backend]

The Order data structure is used to convey the relevant data of the currently in-process stock order between the MySQL interaction wrapper code and the rest of the "broker" program. It is comprised of the following data types, each of which is used by the broker program to determine whether an order is to be processed, how to process the order, and to make necessary changes to a user's account, portfolio, and transaction history.

Name	**Data Type**	**Description**
Id	Int	Unique "key" used to identify the order
Account	Int	Account ID of the user who placed the order
timePlaced	Timestamp	Date and time the order was placed
Symbol	String	Stock symbol for this order
Action	String	Indicates whether this is a buy or sell order
Quantity	Long	Number of shares to process
Type	String	Indicates whether this is a market, limit, stop-limit, trailing stop, or stop order
Target_stop	BigDecimal	Target price to determine execution of stop orders (can be used in combination with target_limit for stop-limit orders)
Target_limit	BigDecimal	Target price to determine execution of limit orders (can be used in combination with target_stop for stop-limit orders)
Trail	BigDecimal	The trailing amount for Trailing Stop Orders
Duration	String	How long this order is good for (day or good until cancelled)
Notify	Byte	Indicates whether to notify the user when the order is executed

ii. Stock Information Array [Backend]

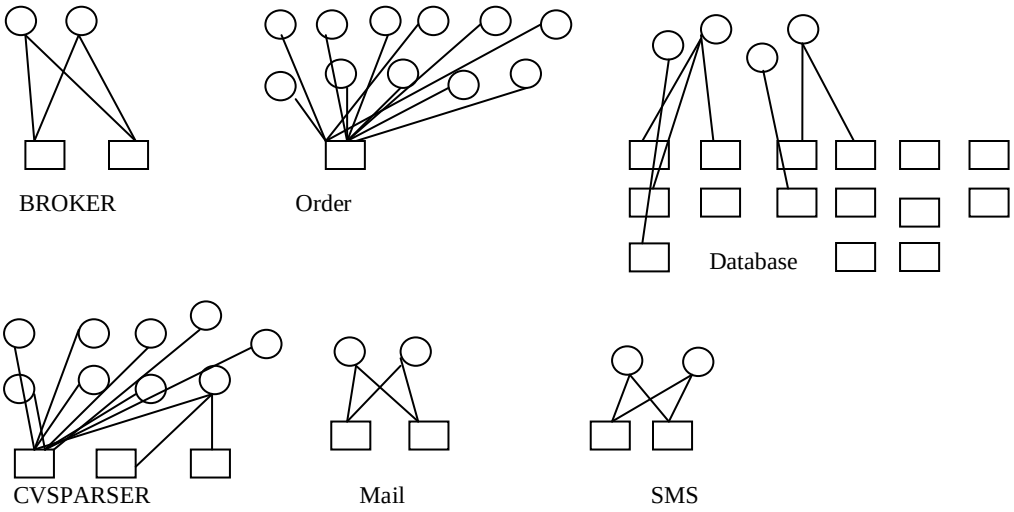
After parsing from Yahoo Finance, the program uses a two dimensional array to store stock information. The columns in the array are symbol, price and volume, and the rows are different stocks that parsed from Yahoo Finance.

iii. Entity Classes and Controller Classes [Frontend]
The entity classes and controller classes on the frontend can be described in a general way that applies to all of them.

An entity class directly correlates to a table within the database. For example, there is a class called Account. Each field in the accounts table is then a protected data type within this class. The entity classes contain setter and getter methods to access their protected data types. The controller class instantiate an entity the same way one would instantiate an object in other high level programming languages, i.e. an account object is created in the controller using 'new Account();'. The controller classes then use these objects to manipulate data within this object. Finally, the object can be posted to the database using the PHP PDO connections described above. In the instance where data is being collected from a particular table, the $user type (an Account entity) can be dereferenced: $user->getPortfolio() accesses the Entity to obtain the entries in the portfolio table associated with a given user.

15. Cohesion Analysis

COHESION MODEL FOR BACKEND



Above are the cohesion models for the different classes used in our backend. The circles represent the attributes, the rectangles represent the methods and the lines

indicate what methods use what attributes. From the diagram it can be observed that most of the classes are fairly balanced in their computations except for the Order class. We tried to make the classes such that no class methods had to do too many computations, however the order class was mostly a state class which held all the data for the particular order and hence it was highly cohesive. Below are the cohesion values for SCOM, CC, LSCC and CAMC. The values are generated using a plugin of java , metrics analysis. An XML with the generated dependency values is attached.

The calculations are based on the below rules:

SCOM: The SCOM for two components is calculated using product of intersection of the two components and the union of the two divided by minimum of the two and the number of attributes.

CC: CC is simply the ratio of the intersection of the two components and the union of the two components.

LSCC: It's a measure of the number of attributes, the number of methods, and the number of methods that reference the attribute.
CAMC: was calculated as *a/kl,* where *l* is the number of distinct parameter types, *k* is the number of methods, and *a* is the summation of the number of distinct parameter types of each method in the class.

Class/Attribute/Method	SCOM	CC	LSCC	CAMC
Broker	1	1	1	1
ExecuterOrderSell	1	1	1	1
ExecuteOrderBuy	1	1	1	1
Cash	1	1	1	1
Order	1	1	1	1
CSVParser	0.8	.778	.8	0
GetVolume	.1	.1	0	0
GetPrice	.1	.1	0	0

Parse	1	1	.5	.16
stock	.3	.6	.4	0
url	.3	1	.7	0
urlConn	.3	1	.3	.33
inStream	.3	.8	.156	0
Buff	.3	.7	.67	0
strarr	1	1	.9	0
numStocks	.3	.8	.5	0
Order	1	1	1	1
Id	1	1	1	1
account	1	1	1	1
timePlaced	1	1	1	1
symbol	1	1	1	1
action	1	1	1	1
type	1	1	1	1
quantity	1	1	1	1
targetPrice	1	1	1	1
stopPrice	1	1	1	1
limitPrice	1	1	1	1
trail	1	1	1	1
duration	1	1	1	1
notify	1	1	1	1
Order	1	1	1	1

Mail	1	1	1	1
to	1	1	1	1
body	1	1	1	1
mail	1	1	1	1
send	1	1	1	1
SMS	1	1	1	1
to	1	1	1	1
body	1	1	1	1
SMS	1	1	1	1
send	1	1	1	1
Database	.3	.5	.5	.3
connection	0	.1	.3	.5
order	0	.3	1	.667
debug	0	1	1	.4
currentOrder	0	.1	.3	0
connect	.5	.66	.667	0
readOrders	.3	.4	.3	0
getSymbols	0	0	0	.146
getOrder	0	0	0	1
getUserCash	0	0	0	1
setUserCash	0	0	0	1
getPortfolioStock	0	0	0	1
getUserEmail	0	0	0	1

addHistory	0	0	0	1
getUserPhone	0	0	0	.133
getUserPhoneCarrier	0	0	0	1
getUserSMS	0	0	0	1
addToPortfolio	0	0	0	1
removeFromPortfolio	0	0	0	1
deleteOrder	0	0	0	.5
updateStopPrice	0	0	0	1
weightedAverage	0	0	0	1
debug	.25	.3	.2	1

A report on the unit testing for the front end is still being created.

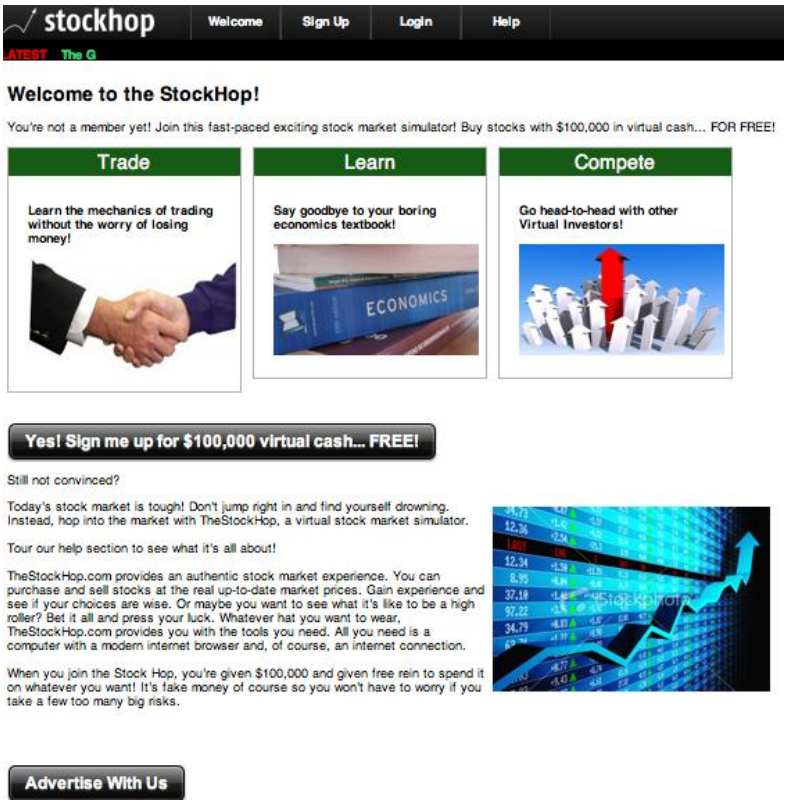
14. User Interface

A focus for thestockhop.com from Demo1 to Demo2 was to undergo a complete UI redesign. After sitting down to think about the usability from the perspective of new users, we realized that our website was not as friendly, inviting, and 'fun-looking' as many of the most used websites on the Internet these days (Facebook, Twitter, etc). The overall functionality of the original user interface remains unchanged, but a heavy emphasis was placed on color schemes and user-friendliness. Major differences include the removal of the data ticker in favor of a news-only ticker, a switch in the visual style of the tabs, and the location of the "Login | Signup" and "Logout" actions. Below we provide a walkthrough of how users will experience the site.

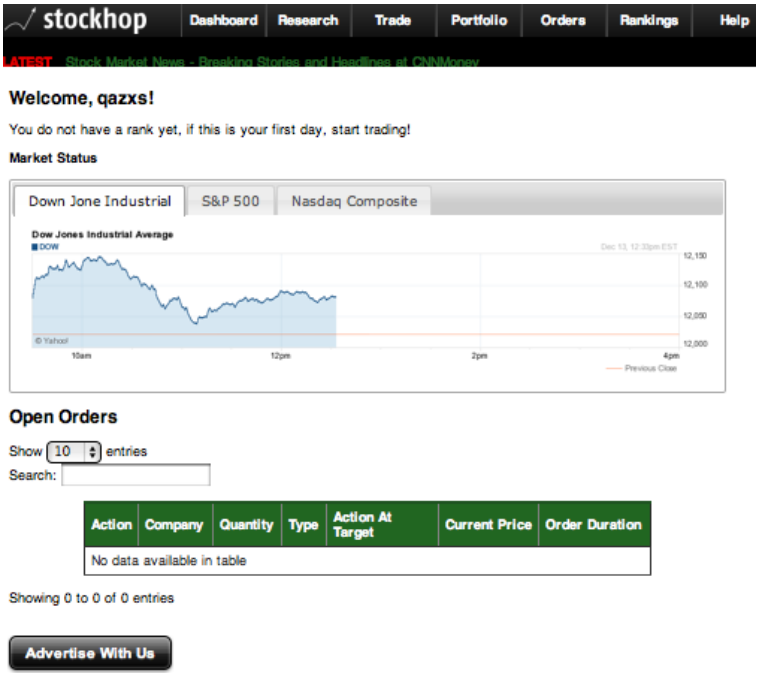
We also considered the fact that our application was data-driven and that there needed to be an effective way of displaying and searching through all of this data. We implemented new searchable tables that are super fast for searching for data or display only a certain number of entries per page.

The site is comprised of the following pages:

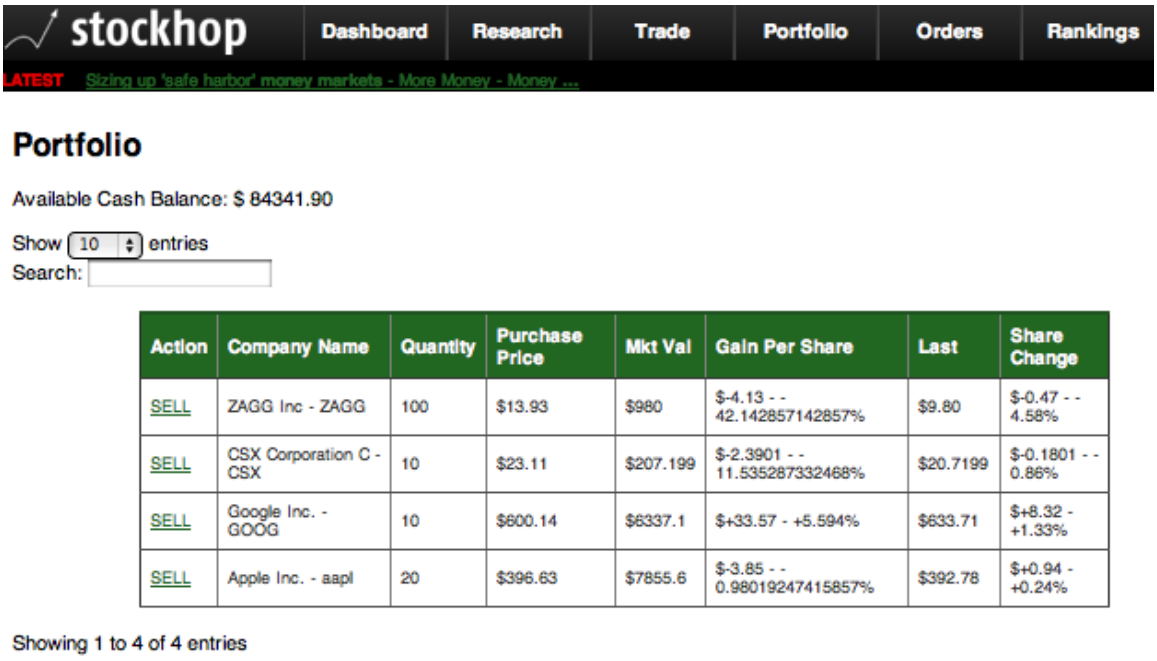
1. Welcome – default page seen when a new or not yet logged in existing user accesses the website at http://www.stockhop.com. Here a new user may register or an existing user may sign in.



1. Home – individual user's home screen containing the user's current rank, a brief performance summary of their portfolio showing the best/worst stocks they currently own, and a news feed showing current business news. Clicking on a news feed title will take the user to a third-party website that has the full article.

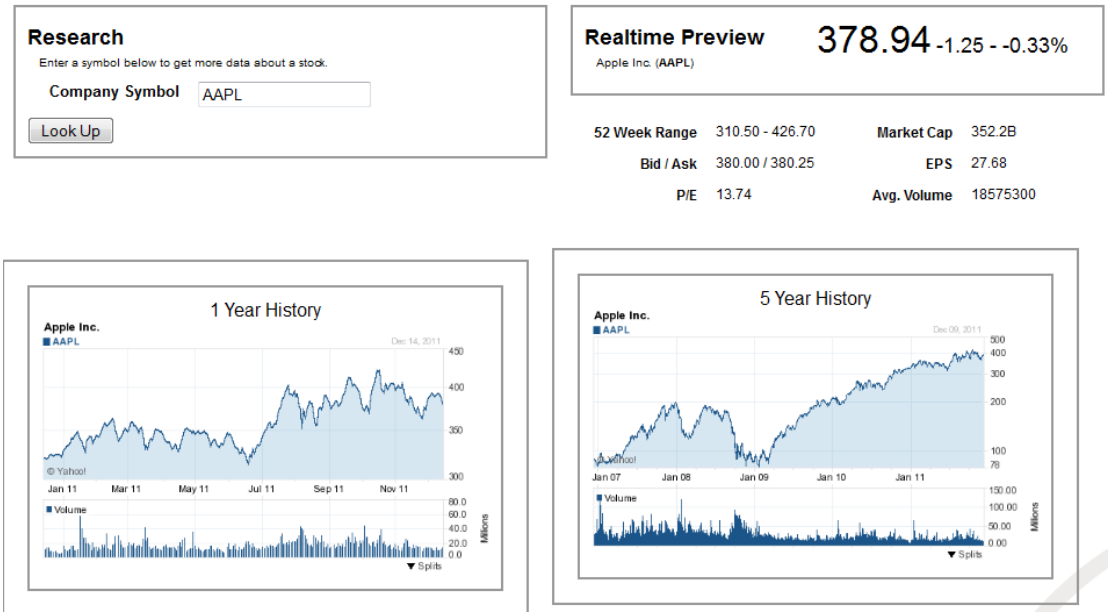


2. Portfolio – shows the pending transactions and the stocks the user currently owns. The user can cancel or edit individual pending transactions from this screen. The user can also click “Sell” next to a currently owned stock to automatically fill out a sell order.



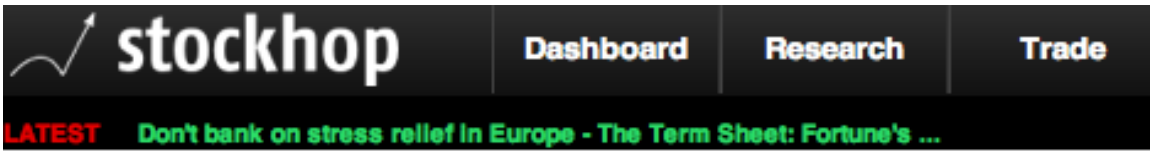
3. Research – used for researching and viewing detailed information about a particular stock. Stock information is searched by stock symbol.

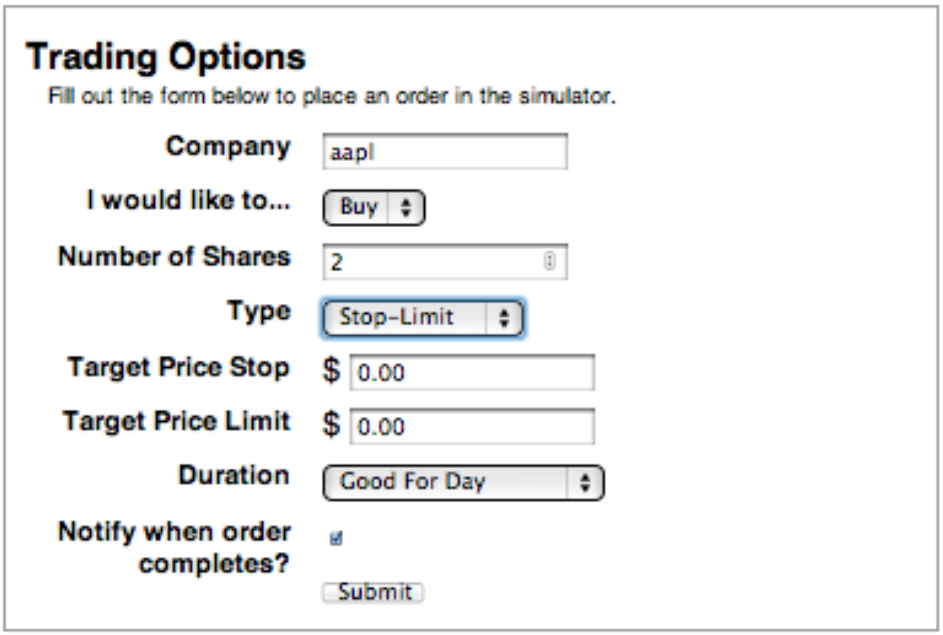
Research



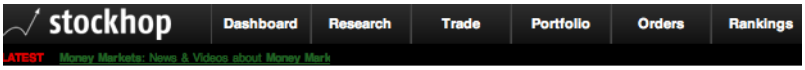
Note: Navigation bar was omitted from the above image in order to show both history tables but is actually present on the research page.

4. Trade – buy/sell orders are made from this screen. They are then previewed and confirmed via a popup dialog.





5. Orders – shows the currently pending buy/sell orders that have not yet been executed. The user may cancel an order from this screen.



Open Orders

Show 10 entries
Search:

Action	Company	Quantity	Type	Action At Target	Current Price	Order Duration
No data available in table						

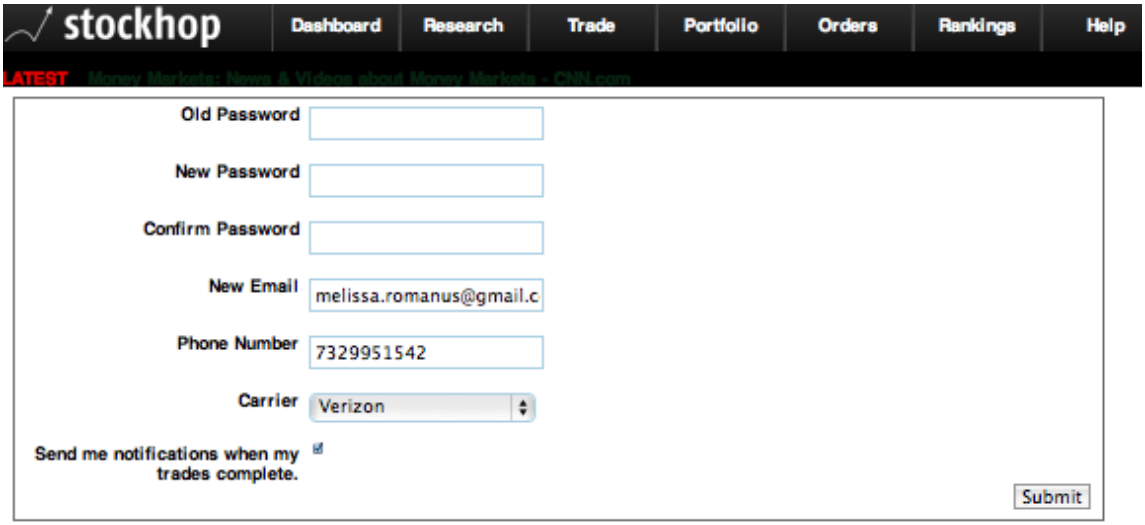
Showing 0 to 0 of 0 entries

Transaction History

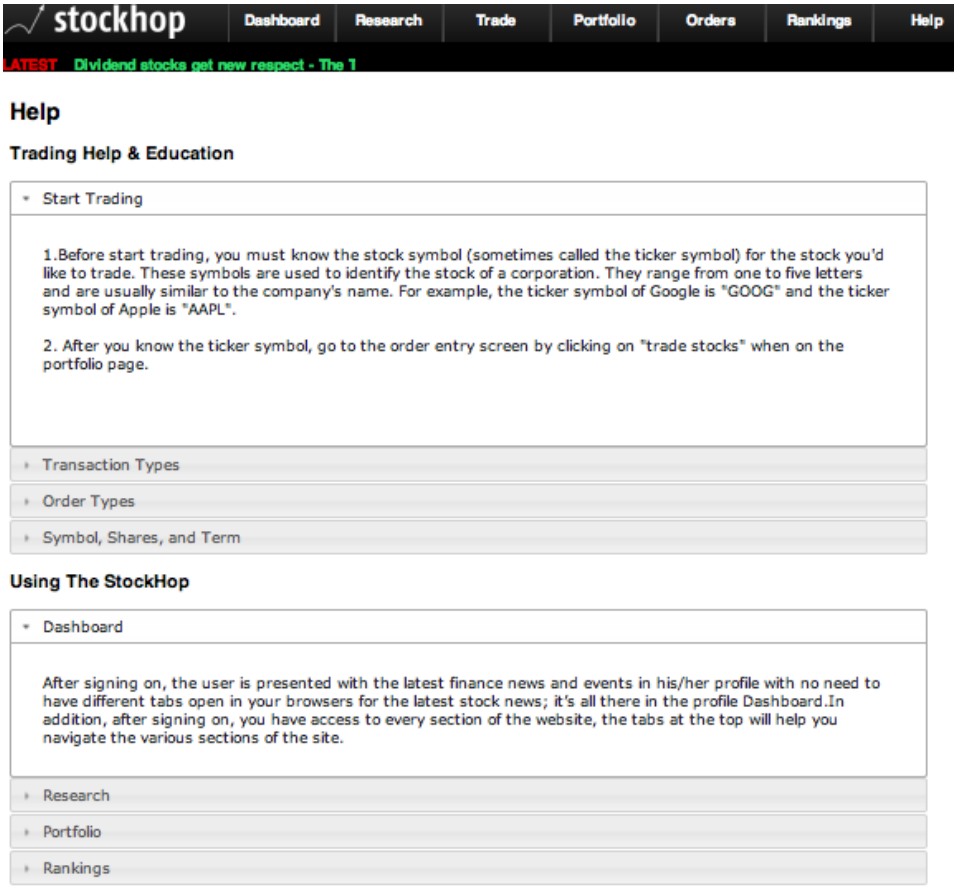
Show 10 entries
Search:

Date Executed	Transaction Details	Company	Quantity	Target	Purchase Price	Order Value	Account Value
October 28, 2011 16:48	Buy - Market	ZAGG	100	Market	$0.00	$1393.00	$98587.00
October 28, 2011 16:49	Buy - Market	CSX	10	Market	$0.00	$231.10	$98335.90
October 28, 2011 17:17	Buy - Market	GOOG	10	Market	$0.00	$6001.40	$92314.50
November 4, 2011 12:33	Buy - Market	aapl	10	Market	$0.00	$4007.50	$88287.00
December 9, 2011 11:54	Buy - Market	aapl	10	Market	$392.51	$3925.10	$84341.90

6. Preferences – used to edit the notification email address, enable/disable notifications about stock order completions, change the user's password, and permanently delete the user's account.



7. Help – contains help and tutorial information on stock market basics and how to use theStockHop website



15. History of Work & Current Status of Implementation

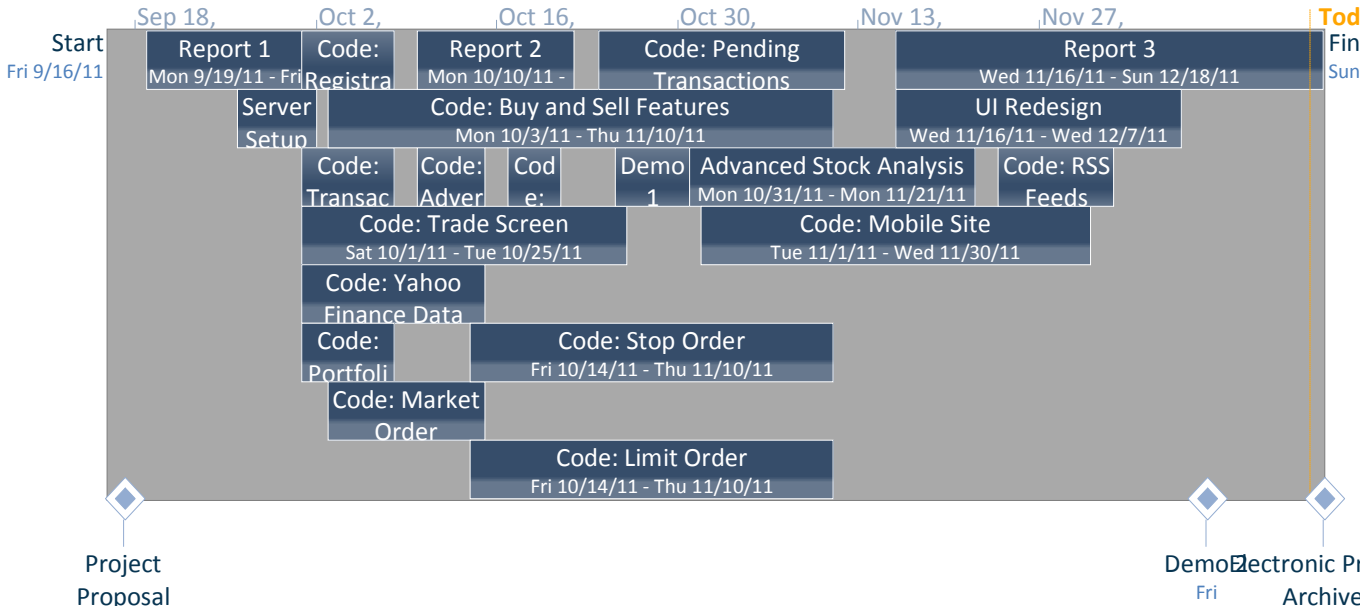
TheStockHop programming team managed to achieve almost all of their programming goals this semester. The following chart summarizes the work completed by the group. After Group 2 was satisfied with the results of the UI Design, they decided to add in the Stop-Limit and Trailing Stop orders.

Task Name	Duration	Start	Finish
Project Proposal	**1 day**	**Fri 9/16/11**	**Fri 9/16/11**
Report 1	**11 days**	**Mon 9/19/11**	**Fri 9/30/11**
Server Setup	6 days	Mon 9/26/11	Sat 10/1/11
Code: Registration, Login, Logout	6 days	Sat 10/1/11	Fri 10/7/11
Code: Transaction History	4 days	Sat 10/1/11	Wed 10/5/11
Code: Trade Screen	18 days	Sat 10/1/11	Tue 10/25/11
Code: Yahoo Finance Data Gatherer	11 days	Sat 10/1/11	Fri 10/14/11
Code: Portfolio	6 days	Sat 10/1/11	Fri 10/7/11
Code: Buy and Sell Features	**29 days**	**Mon 10/3/11**	**Thu 11/10/11**
Code: Market Order	10 days	Mon 10/3/11	Fri 10/14/11
Code: Stop Order	20 days	Fri 10/14/11	Thu 11/10/11
Code: Limit Order	20 days	Fri 10/14/11	Thu 11/10/11
Code: Trailing Stop	10 days	Mon 11/28/11	Wed 12/7/11
Code: Stop-Limit	10 days	Mon 11/28/11	Wed 12/7/11
Report 2	**10 days**	**Mon 10/10/11**	**Fri 10/21/11**
Code: Advertisements	5 days	Mon 10/10/11	Fri 10/14/11
Code: Email and SMS Notifications	4 days	Mon 10/17/11	Thu 10/20/11
Code: Pending Transactions	15 days	Mon 10/24/11	Fri 11/11/11
Demo 1	**1 day**	**Fri 10/28/11**	**Fri 10/28/11**
Advanced Stock Analysis	16 days	Mon 10/31/11	Mon 11/21/11
Code: Mobile Site	22 days	Tue 11/1/11	Wed 11/30/11
Report 3	23 days	Wed 11/16/11	Sun 12/18/11
Code: RSS Feeds	2 days	Thu 11/24/11	Fri 11/25/11
Demo 2	**1 day**	**Fri 12/9/11**	**Fri 12/9/11**
Electronic Project Archive	7 days	Sat 12/10/11	Sun 12/18/11
UI Redesign	16 days	Wed 11/16/11	Wed 12/7/11

We did not end up implementing a mobile site, however, our current website was tested on the default Android browser and it does work. The only thing that we did not do was optimize it for mobile web.

Initially, we forgot to account for Testing and Debugging of our code. This meant that there were a lot of difficulties in our first Demo that we were trying to work out in more of a last minute fashion. We left extra time for debugging in time for Demo 2, and also implemented PHPUnit tests of our frontend code. We also used the testing methods of having friends and family use the system that had not previously had experience with stock trading or our interface.

We were also able to implement the 'basic functionality' ahead of schedule. This was due to the fact that we wanted to have a very functional Demo 1. After we had implemented the functionality, we used some of the ahead of schedule time that we had acquired in order to look at the project and assess the areas that needed the most improvement. We added a UI Redesign, which was not part of the initial plan. We also tried out code for a cache of sorts. It did not show a speed-up at a small scale, but instead, we found that we could streamline our calls to Yahoo Finance. Instead of making multiple calls for each stock symbol, we were able to make just one call and get information for many symbols back.



Key Accomplishments

- Set-up web server and database server
- Deployed a website that is live to the Internet (not running locally)
- User Interface Design
- Data interactions with external website
- Database interactions between web client and also server-side programs

- Order Types: Buy or Sell for Market, Limit, Stop, Trailing Stop, and Stop-Limit
- RSS Feed
- Views of Portfolio, Open Orders, and Transaction History
- Email & Cell Phone Notifications
- Historical Stock Research
- Fast searchable and sortable data tables
- Tutorials
- Advertising Interface Form that allows file uploads to the server
- Nightly ranking system organized by user net worth
- After-Hours Deletion of “Good for the Day” stocks

16. Conclusions and Future Work

TheStockHop.com has been tremendous undertaking given the allotted development time and small development team. Additionally, some of the staff were very inexperienced in building a large software package so there were training requirements to handle as well. TheStockHop.com team tried to find contributions for both the inexperienced and experienced members of the team – where some members were more concerned with adding code and other members were concerned with research and human factors.

From the beginning, TheStockHop.com set itself apart from other members in the class in that it was implemented as a live website on a home server. The server was also not previously running, so we started from essentially scratch by installing a Linux Operating System, and configuring it to be a web server. We then focused on installing all of the features and plugins needed to create our project. The result is the live URL: http://www.thestockhop.com.

One of the biggest initial challenges faced before Demo 1 was the fact that we forgot to schedule in testing mechanisms and time. This did not allow us to show all of the functionality we coded in for Demo 1. In order to tackle this challenge, we added unit testing in to the PHP and additionally, leveraged friends and family to test out the system and report any bugs found.

For the frontend, learning a new framework, Symfony2 proved to be very difficult in the beginning. It really took a few weeks for the team to get up to speed with the project in its entirety. Initially, the code for the frontend did not leverage all the

possible features of Symfony2. After learning more about design patterns in Software Engineering, the students learned more about the intricacies of the Model-View-Controller pattern and even went back to fix some of their code. Another software engineering idea learned was the idea of separating presentation logic from business logic in web-based applications through the use of Twig (explained above).

Prior to doing this project, the object-oriented aspects of PHP had not been explored by the students, so leveraging object-oriented PHP allowed them to really take into account scalability and expandability. Refactoring was another important factor in the design process. After taking an agile approach to the user interface design of Demo 1, the team was able to reevaluate their progress. Before implementing the new UI, the site was mocked up more thoroughly in inkscape, and css was written to match the same grid that was used to make the image. This really made the design look much cleaner. In a future iteration, the jQuery and javascript code that was written would be organized into separate files. Some integration issues occurred when trying to move the javascript files a few levels up into a separate directory, so most of the java script is actually implemented in-line with the HTML Twig files.

TheStockHop.com allows the user to place a wide variety of order types. Currently, these types include market, limit, stop, stop-limit, and trailing stop orders. However, there are many more order types that can be added to the system to add to its completeness and to allow it to excel as a teaching tool. In time, it is our hope that TheStockHop.com

From a computational standpoint, the processing required to handle a customer's requests at TheStockHop.com is not very high. However, if the number of users of the system grows significantly larger, the aggregate of the requests from all the users could tax the server and potentially result in slow performance or, in the worst case, refusal of service. Solutions to this problem have already begun to be researched and a caching system for stock prices has been tested to prevent redundant calls to the price provider.

Currently, TheStockHop.com relies on the stock prices being provided by Yahoo Finance. This has so far proved reliable and fast. However, when using Yahoo Finances services, TheStockHop.com is legally restricted from making money. Ideally, we would like to change providers to one that has fewer restrictions. More than that, we would also like to have backup providers if Yahoo Finance ever were to go down. Also, the current server is very minimalistic hardware running off of a normal Verizon FIOS connection. If TheStockHop were to move into a for-profit sector, better hardware and a Small Business grade connection would be required.

Finally, because TheStockHop.com is primarily a teaching tool, we would like to further engage the user beyond simply placing orders. Some sort of analysis application or predictive tool that the user can interact with would tremendously enhance the experience and raise learning potential. We also hope to implement ranking prizes. Again, this was not something we realized in the first iteration in order to avoid violating any of the terms of service of Yahoo Finance.

Using the design principles learned in software engineering, the majority of the core functionality was indeed implemented and received a level of polish, making TheStockHop.com a worthy competitor to other fantasy stock trading websites. There is still plenty of room for TheStockHop.com to grow in the future, and the tools that the team used make it easy to scale and modify this project.

17. References

"Template Engine (web)." *Wikipedia, the Free Encyclopedia*. Web. 04 Nov. 2011. <http://en.wikipedia.org/wiki/Template_engine_(web)>.

"Codd's 12 Rules." *Wikipedia, the Free Encyclopedia*. Web. 04 Nov. 2011. <http://en.wikipedia.org/wiki/Codd's_12_rules>.

Garlan, David, and Mary Shaw. *An Introduction to Software Architecture*. Tech. no. CMU-CS-94-166. New Jersey: World Scientific, 1993. Print.

"Front Controller Pattern." *Wikipedia, the Free Encyclopedia*. Web. 15 Dec. 2011. <http://en.wikipedia.org/wiki/Front_Controller_pattern>.

APPENDIX

A.1 Hardware Profile

```
stockhop-01.fc2112.net
   description: Desktop Computer
   product: ()
   width: 32 bits
   capabilities: smbios-2.4 dmi-2.4 smp-1.4 smp
   configuration: boot=normal chassis=desktop cpus=1 uuid=98C158E4-2334-11DD-AE72-
0011113186F6
 *-core
      description: Motherboard
      product: D945GCLF
      vendor: Intel Corporation
      physical id: 0
      version: AAE27042-302
      serial: AZLF82100ACB
      slot: Base Board Chassis Location
    *-cpu
         description: CPU
         product: Intel(R) Atom(TM) CPU  230   @ 1.60GHz
         vendor: Intel Corp.
         physical id: 0
         bus info: cpu@0
         version: 6.12.2
         serial: 0001-06C2-0000-0000-0000-0000
         slot: U1PR
         size: 1600MHz
         capacity: 4GHz
         width: 64 bits
         clock: 133MHz
         capabilities: boot fpu fpu_exception wp vme de pse tsc msr pae mce cx8 apic mtrr pge mca
cmov pat clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe nx x86-64 constant_tsc pni monitor ds_cpl
tm2 ssse3 cx16 xtpr lahf_lm
         configuration: id=1
       *-cache:0
            description: L2 cache
            physical id: 1
            slot: Unknown
            size: 512KiB
            capacity: 512KiB
            capabilities: asynchronous internal write-back unified
       *-cache:1
            description: L1 cache
            physical id: 2
            slot: Unknown
            size: 32KiB
            capacity: 32KiB
            capabilities: asynchronous internal write-back instruction
       *-logicalcpu:0
            description: Logical CPU
```

```
          physical id: 1.1
          width: 64 bits
          capabilities: logical
       *-logicalcpu:1
          description: Logical CPU
          physical id: 1.2
          width: 64 bits
          capabilities: logical
    *-firmware
       description: BIOS
       vendor: Intel Corp.
       physical id: 3
       version: LF94510J.86A.0182.2009.0528.2014
       date: 05/28/2009
       size: 64KiB
       capacity: 448KiB
       capabilities: pci upgrade shadowing cdboot bootselect edd int9keyboard int14serial
int17printer int10video acpi usb zipboot biosbootspecification netboot
    *-memory
       description: System Memory
       physical id: 10
       slot: System board or motherboard
       size: 2GiB
       capacity: 2GiB
       *-bank
          description: DIMM DDR2 Synchronous 533 MHz (1.9 ns)
          product: 0x393931353535202839393635353529000000
          vendor: 0x7F7F7F9400000000
          physical id: 0
          serial: 0x00000000
          slot: J1MY
          size: 2GiB
          width: 64 bits
          clock: 533MHz (1.9ns)
    *-pci
       description: Host bridge
       product: 82945G/GZ/P/PL Memory Controller Hub
       vendor: Intel Corporation
       physical id: 100
       bus info: pci@0000:00:00.0
       version: 02
       width: 32 bits
       clock: 33MHz
       configuration: driver=agpgart-intel
       resources: irq:0
       *-display UNCLAIMED
          description: VGA compatible controller
          product: 82945G/GZ Integrated Graphics Controller
          vendor: Intel Corporation
          physical id: 2
```

```
          bus info: pci@0000:00:02.0
          version: 02
          width: 32 bits
          clock: 33MHz
          capabilities: msi pm vga_controller bus_master cap_list
          configuration: latency=0
          resources: memory:88200000-8827ffff ioport:20c0(size=8) memory:80000000-
87ffffff(prefetchable) memory:88280000-8829ffff
       *-pci:0
          description: PCI bridge
          product: N10/ICH 7 Family PCI Express Port 1
          vendor: Intel Corporation
          physical id: 1c
          bus info: pci@0000:00:1c.0
          version: 01
          width: 32 bits
          clock: 33MHz
          capabilities: pci pciexpress msi pm normal_decode bus_master cap_list
          configuration: driver=pcieport-driver
          resources: irq:201 ioport:1000(size=4096) memory:88100000-881fffff
ioport:88000000(size=1048576)
           *-network
              description: Ethernet interface
              product: RTL8101E/RTL8102E PCI Express Fast Ethernet controller
              vendor: Realtek Semiconductor Co., Ltd.
              physical id: 0
              bus info: pci@0000:01:00.0
              logical name: eth0
              version: 02
              serial: 00:1c:c0:45:da:d1
              size: 100Mbit/s
              capacity: 100Mbit/s
              width: 64 bits
              clock: 33MHz
              capabilities: pm msi pciexpress msix vpd bus_master cap_list rom ethernet physical tp mii
10bt 10bt-fd 100bt 100bt-fd autonegotiation
              configuration: autonegotiation=on broadcast=yes driver=r8169 driverversion=2.3LK-1-
NAPI duplex=full ip=172.16.2.10 latency=0 link=yes multicast=yes port=MII speed=100Mbit/s
              resources: irq:50 ioport:1000(size=256) memory:88100000-88100fff memory:88000000-
8800ffff(prefetchable) memory:88020000-8803ffff(prefetchable)
       *-pci:1
          description: PCI bridge
          product: N10/ICH 7 Family PCI Express Port 3
          vendor: Intel Corporation
          physical id: 1c.2
          bus info: pci@0000:00:1c.2
          version: 01
          width: 32 bits
          clock: 33MHz
          capabilities: pci pciexpress msi pm normal_decode bus_master cap_list
```

```
        configuration: driver=pcieport-driver
        resources: irq:209
    *-pci:2
        description: PCI bridge
        product: N10/ICH 7 Family PCI Express Port 4
        vendor: Intel Corporation
        physical id: 1c.3
        bus info: pci@0000:00:1c.3
        version: 01
        width: 32 bits
        clock: 33MHz
        capabilities: pci pciexpress msi pm normal_decode bus_master cap_list
        configuration: driver=pcieport-driver
        resources: irq:217
    *-usb:0
        description: USB Controller
        product: N10/ICH 7 Family USB UHCI Controller #1
        vendor: Intel Corporation
        physical id: 1d
        bus info: pci@0000:00:1d.0
        version: 01
        width: 32 bits
        clock: 33MHz
        capabilities: uhci bus_master
        configuration: driver=uhci_hcd latency=0
        resources: irq:225 ioport:2060(size=32)
      *-usbhost
          product: UHCI Host Controller
          vendor: Linux 2.6.18-274.3.1.el5 uhci_hcd
          physical id: 1
          bus info: usb@2
          logical name: usb2
          version: 2.06
          capabilities: usb-1.10
          configuration: driver=hub slots=2 speed=12Mbit/s
    *-usb:1
        description: USB Controller
        product: N10/ICH 7 Family USB UHCI Controller #2
        vendor: Intel Corporation
        physical id: 1d.1
        bus info: pci@0000:00:1d.1
        version: 01
        width: 32 bits
        clock: 33MHz
        capabilities: uhci bus_master
        configuration: driver=uhci_hcd latency=0
        resources: irq:185 ioport:2040(size=32)
      *-usbhost
          product: UHCI Host Controller
          vendor: Linux 2.6.18-274.3.1.el5 uhci_hcd
```

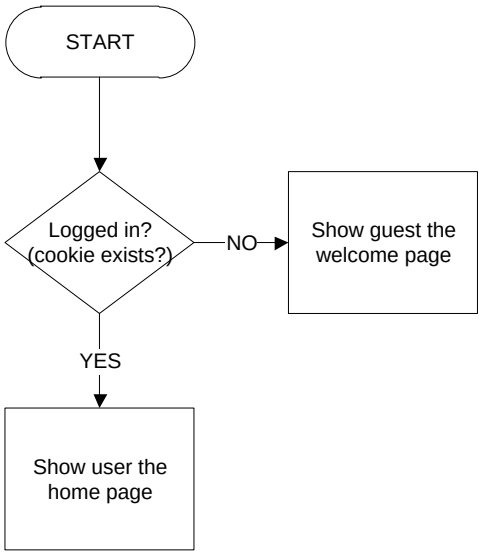
```
          physical id: 1
          bus info: usb@3
          logical name: usb3
          version: 2.06
          capabilities: usb-1.10
          configuration: driver=hub slots=2 speed=12Mbit/s
     *-usb:2
          description: USB Controller
          product: N10/ICH 7 Family USB UHCI Controller #4
          vendor: Intel Corporation
          physical id: 1d.3
          bus info: pci@0000:00:1d.3
          version: 01
          width: 32 bits
          clock: 33MHz
          capabilities: uhci bus_master
          configuration: driver=uhci_hcd latency=0
          resources: irq:233 ioport:2020(size=32)
        *-usbhost
          product: UHCI Host Controller
          vendor: Linux 2.6.18-274.3.1.el5 uhci_hcd
          physical id: 1
          bus info: usb@4
          logical name: usb4
          version: 2.06
          capabilities: usb-1.10
          configuration: driver=hub slots=2 speed=12Mbit/s
     *-usb:3
          description: USB Controller
          product: N10/ICH 7 Family USB2 EHCI Controller
          vendor: Intel Corporation
          physical id: 1d.7
          bus info: pci@0000:00:1d.7
          version: 01
          width: 32 bits
          clock: 33MHz
          capabilities: pm debug ehci bus_master cap_list
          configuration: driver=ehci_hcd latency=0
          resources: irq:225 memory:882a0000-882a03ff
        *-usbhost
          product: EHCI Host Controller
          vendor: Linux 2.6.18-274.3.1.el5 ehci_hcd
          physical id: 1
          bus info: usb@1
          logical name: usb1
          version: 2.06
          capabilities: usb-2.00
          configuration: driver=hub slots=8 speed=480Mbit/s
     *-pci:3
          description: PCI bridge
```

```
          product: 82801 PCI Bridge
          vendor: Intel Corporation
          physical id: 1e
          bus info: pci@0000:00:1e.0
          version: e1
          width: 32 bits
          clock: 33MHz
          capabilities: pci subtractive_decode bus_master cap_list
       *-isa
          description: ISA bridge
          product: 82801GB/GR (ICH7 Family) LPC Interface Bridge
          vendor: Intel Corporation
          physical id: 1f
          bus info: pci@0000:00:1f.0
          version: 01
          width: 32 bits
          clock: 33MHz
          capabilities: isa bus_master cap_list
          configuration: latency=0
       *-ide:0
          description: IDE interface
          product: 82801G (ICH7 Family) IDE Controller
          vendor: Intel Corporation
          physical id: 1f.1
          bus info: pci@0000:00:1f.1
          version: 01
          width: 32 bits
          clock: 33MHz
          capabilities: ide bus_master
          configuration: driver=PIIX_IDE latency=0
          resources: irq:177 ioport:2090(size=16)
       *-ide:1
          description: IDE interface
          product: N10/ICH7 Family SATA IDE Controller
          vendor: Intel Corporation
          physical id: 1f.2
          bus info: pci@0000:00:1f.2
          logical name: scsi0
          version: 01
          width: 32 bits
          clock: 66MHz
          capabilities: ide pm bus_master cap_list emulated
          configuration: driver=ata_piix latency=0
          resources: irq:185 ioport:20a8(size=8) ioport:20cc(size=4) ioport:20a0(size=8)
ioport:20c8(size=4) ioport:2080(size=16)
        *-disk
           description: ATA Disk
           product: WDC WD1600AAJS-6
           vendor: Western Digital
           physical id: 0.0.0
```

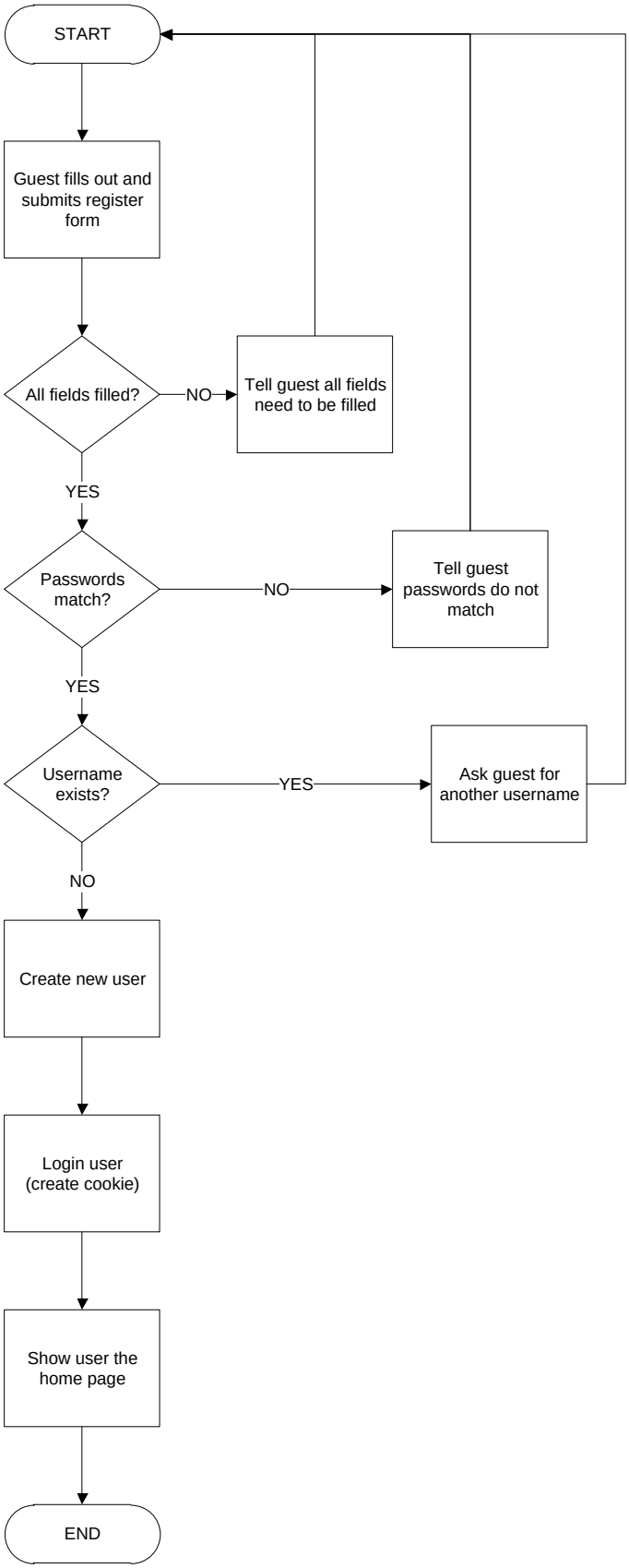
```
           bus info: scsi@0:0.0.0
           logical name: /dev/sda
           version: 21.1
           serial: WD-WCAP92555587
           size: 149GiB (160GB)
           capabilities: partitioned partitioned:dos
           configuration: ansiversion=5 signature=9c879c87
         *-volume:0
              description: EXT3 volume
              vendor: Linux
              physical id: 1
              bus info: scsi@0:0.0.0,1
              logical name: /dev/sda1
              logical name: /boot
              version: 1.0
              serial: a00bef24-b7f4-4c46-b334-7e47542ad398
              size: 101MiB
              capacity: 101MiB
              capabilities: primary bootable journaled extended_attributes recover ext3 ext2 initialized
              configuration: created=2006-10-02 15:23:32 filesystem=ext3 label=/boot
modified=2006-10-13 17:29:54 mount.fstype=ext3 mount.options=rw,data=ordered
mounted=2006-10-13 17:29:54 state=mounted
         *-volume:1
              description: Linux LVM Physical Volume partition
              physical id: 2
              bus info: scsi@0:0.0.0,2
              logical name: /dev/sda2
              serial: gq7jTA-n0Y4-Q0G4-ALUr-mg2r-ADaf-Dk1DHZ
              size: 148GiB
              capacity: 148GiB
              capabilities: primary multi lvm2
      *-serial
         description: SMBus
         product: N10/ICH 7 Family SMBus Controller
         vendor: Intel Corporation
         physical id: 1f.3
         bus info: pci@0000:00:1f.3
         version: 01
         width: 32 bits
         clock: 33MHz
         configuration: driver=i801_smbus latency=0
         resources: irq:185 ioport:2000(size=32)
```

A.2 Complete Flow Chart Program Flow

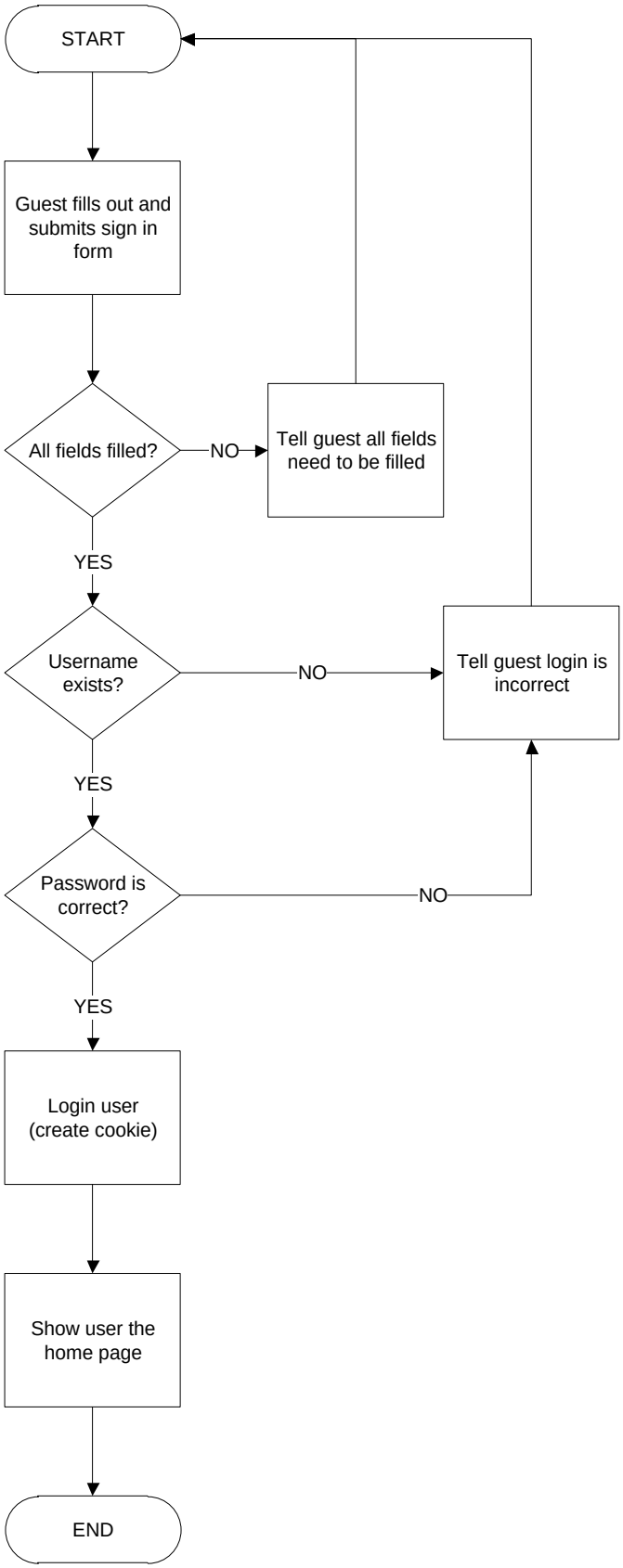
Check Login



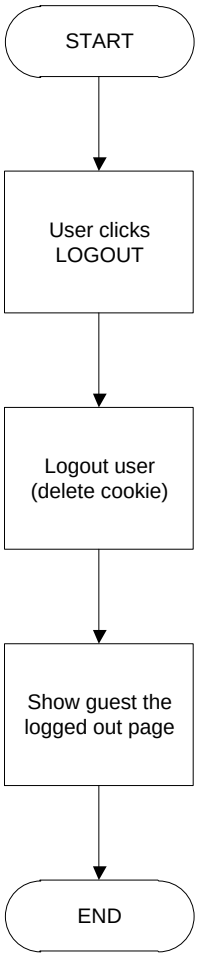
Register


START
Guest fills out and submits register form
All fields filled?
NO
Tell guest all fields need to be filled
YES
Passwords match?
NO
Tell guest passwords do not match
YES
Username exists?
YES
Ask guest for another username
NO
Create new user
Login user (create cookie)
Show user the home page
END

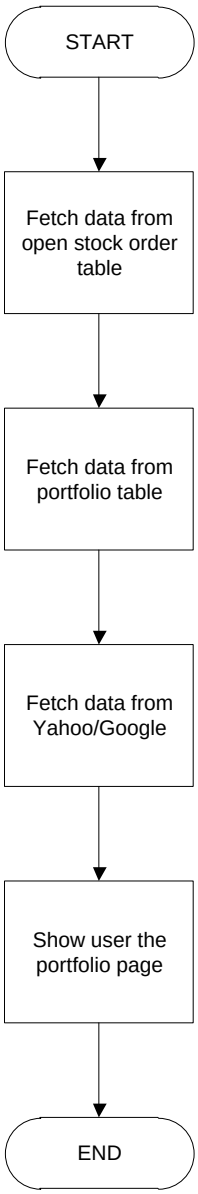
Sign In


START
Guest fills out and submits sign in form
All fields filled?
NO
Tell guest all fields need to be filled
YES
Username exists?
NO
Tell guest login is incorrect
YES
Password is correct?
NO
YES
Login user (create cookie)
Show user the home page
END

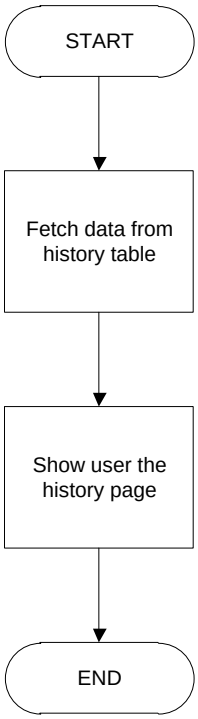
Sign Out


START
User clicks LOGOUT
Logout user (delete cookie)
Show guest the logged out page
END

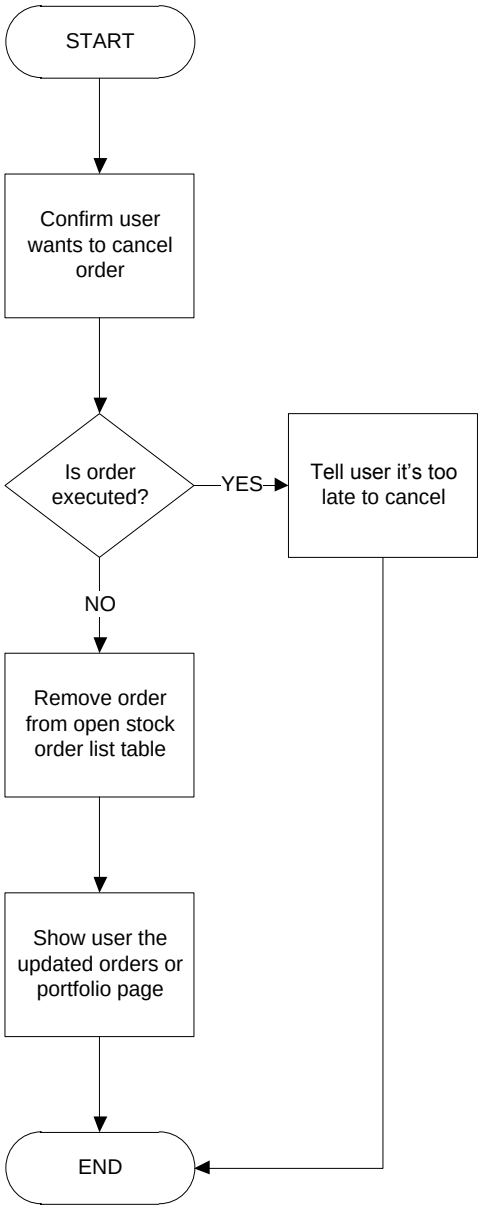
View Portfolio



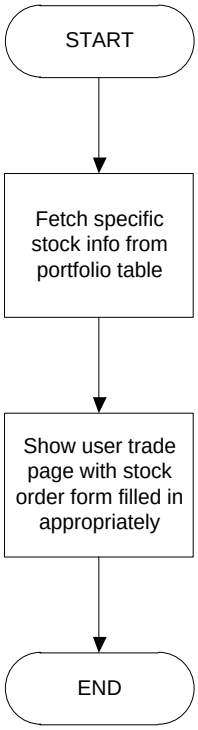
View History



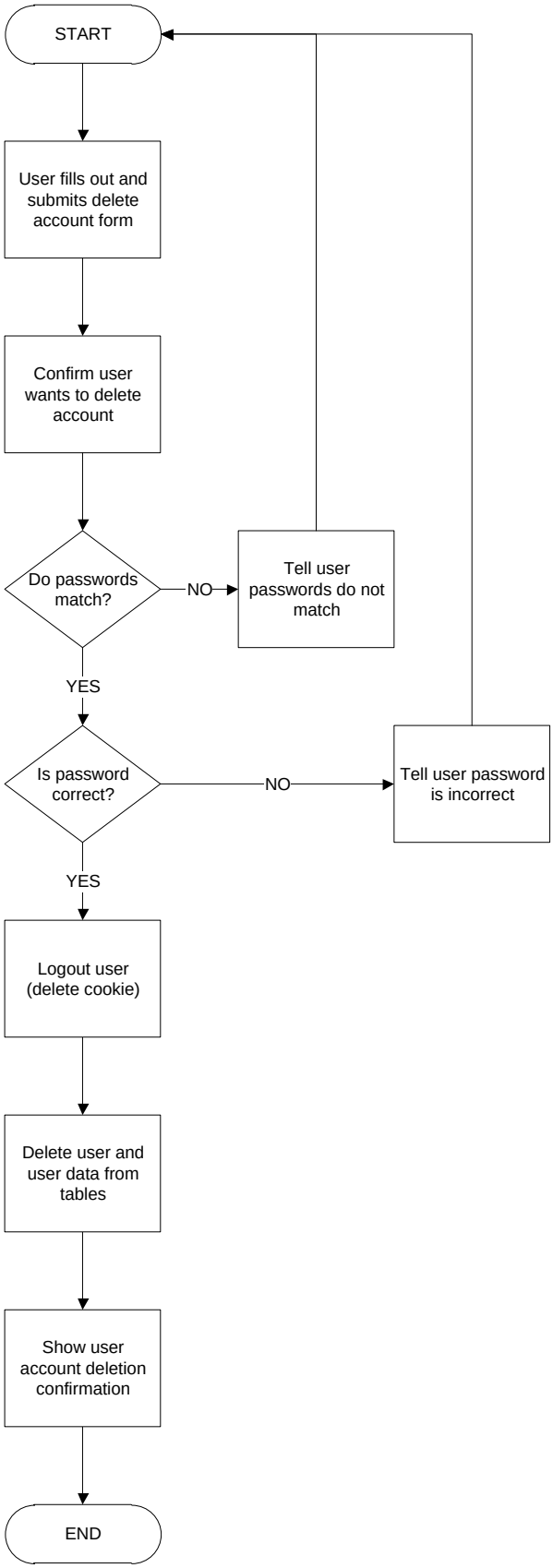
Cancel Order
LNK



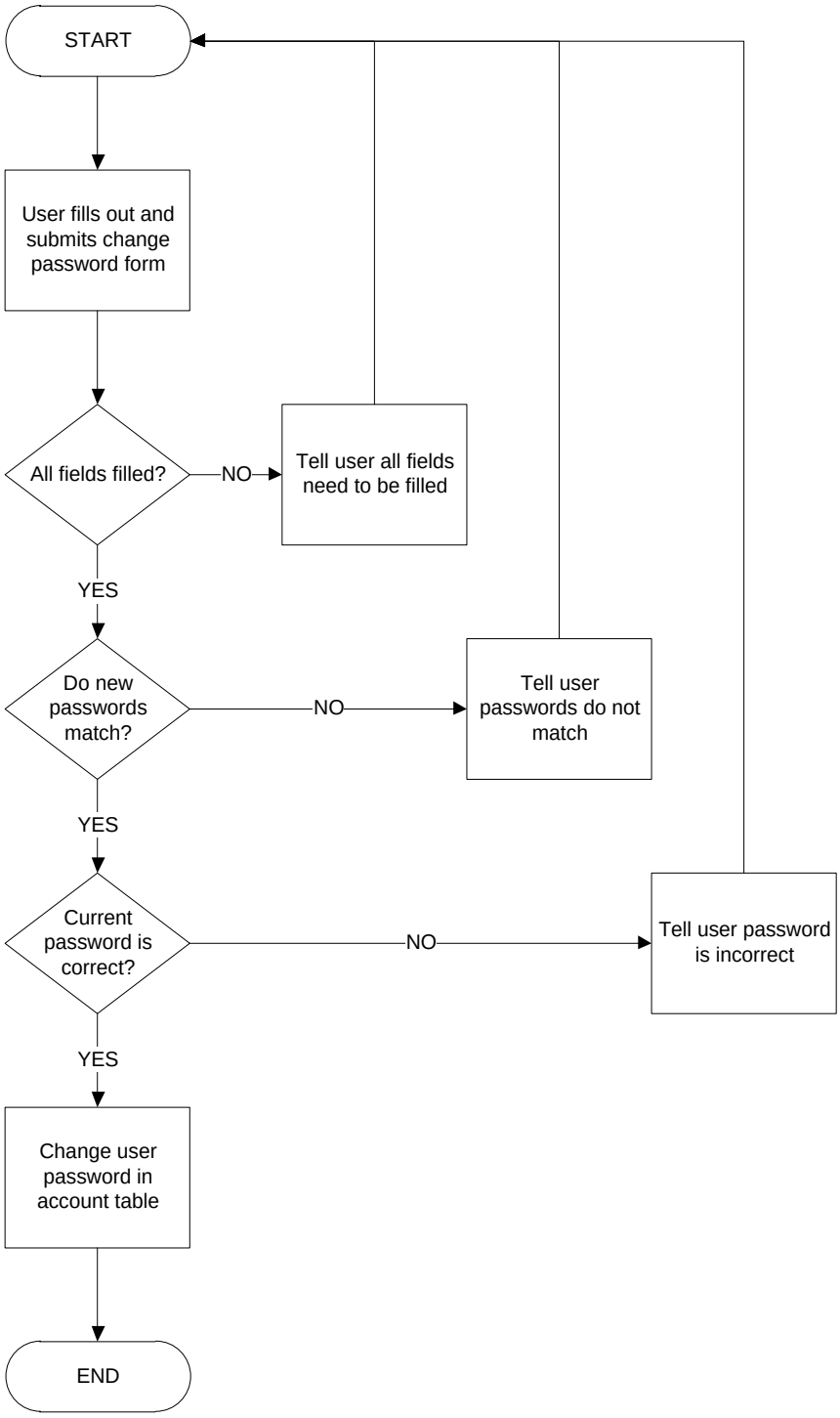
Sell Stock LNK



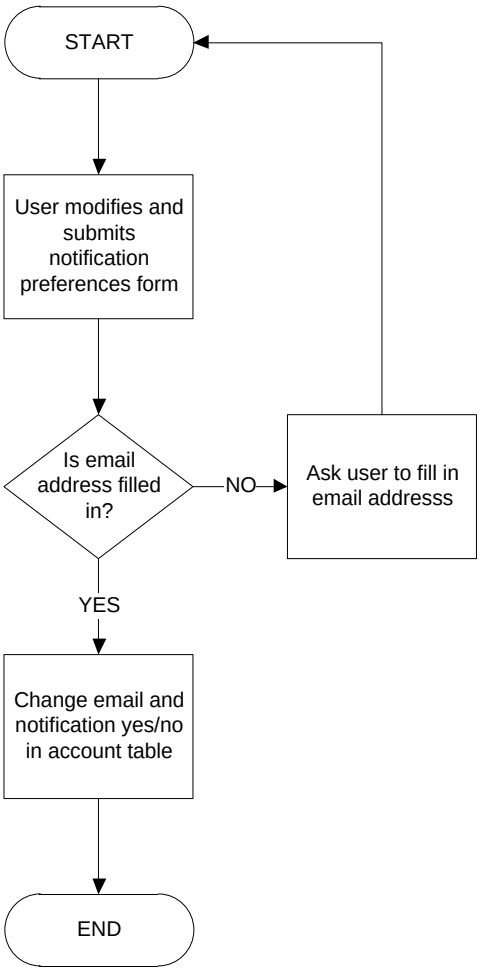
Delete Account



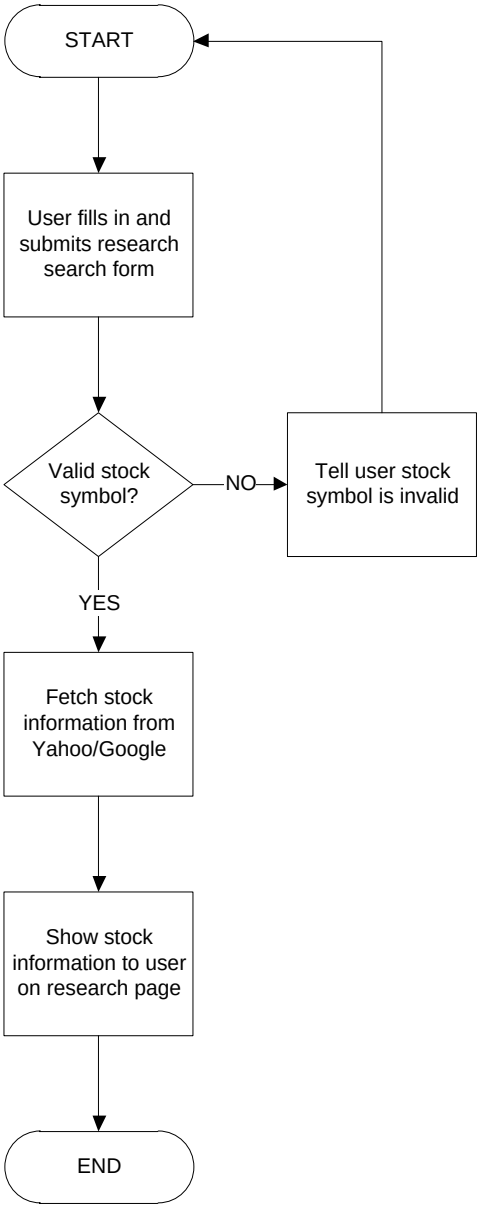
Change Password



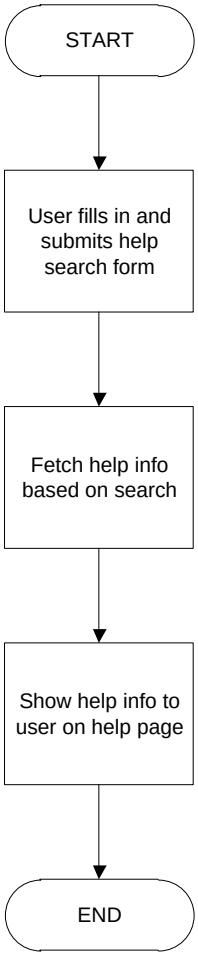
Notification Changes



Research Search



Help Search


START
User fills in and submits help search form
Fetch help info based on search
Show help info to user on help page
END