Rutgers University

School of Engineering

Department of Electrical and Computer Engineering

StockHop: The Stock Market Fantasy League Game

REPORT #1

URL: http://www.thestockhop.com

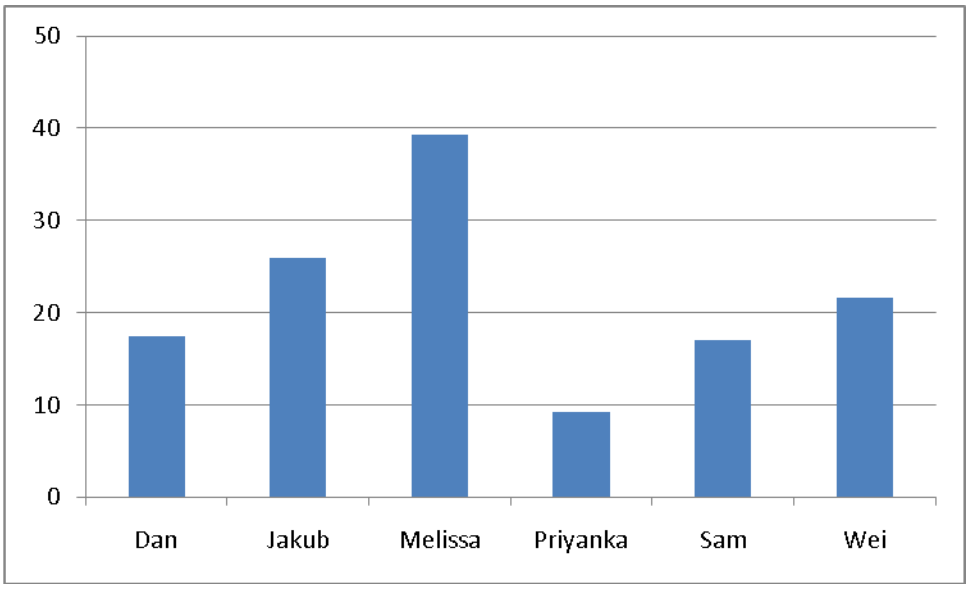
Group #2

Priyanka Kale
Jakub Kolodziejski
Dan Marzullo
Wei Shen
Sam Ramezanli
Melissa Romanus

I. Individual Contributions Breakdown

		Team Member					
		Dan	Jakub	Melissa	Priyanka	Sam	Wei
Responsibility Matrix	Project Management (8 pts)	5%	25%	70%			
	Software Development Introduction (3 pts)			100%			
	Interaction Diagrams (20 pts)						100%
	Class Diagram and Interface Specification (24 pts)	50%		50%			
	System Architecture and System Design (24 pts)		47.5%	47.5%	5%		
	Algorithms and Data Structures (10 pts)		15%	35%	30%		20%
	User Interface Design and Implementation (10 pts)					100%	
	Progress Report and Plan of Work (3 pts)	10%	10%	80%			
	References (1 pt)			100%			

Responsibility Matrix



II. Table of Contents

III. Software Development Introduction

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. 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.

g. 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.

IV. Interaction Diagrams

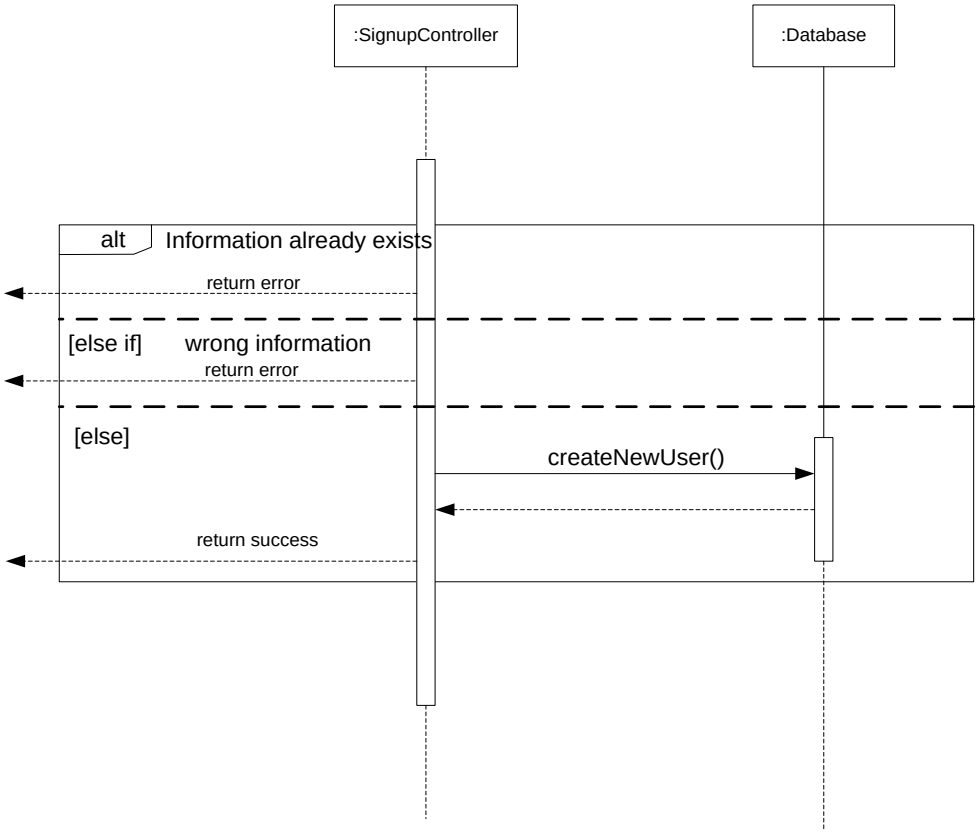
a. *Summary of interaction diagrams*

The following explain the interaction diagrams for the use cases covered in Demo 1.

i. Registration
ii. View Homepage
iii. Buy Stocks
iv. Sell Stocks
v. Research Stocks
vi. View Portfolio
vii. Send Notifications
viii. View Transaction History
ix. Preferences

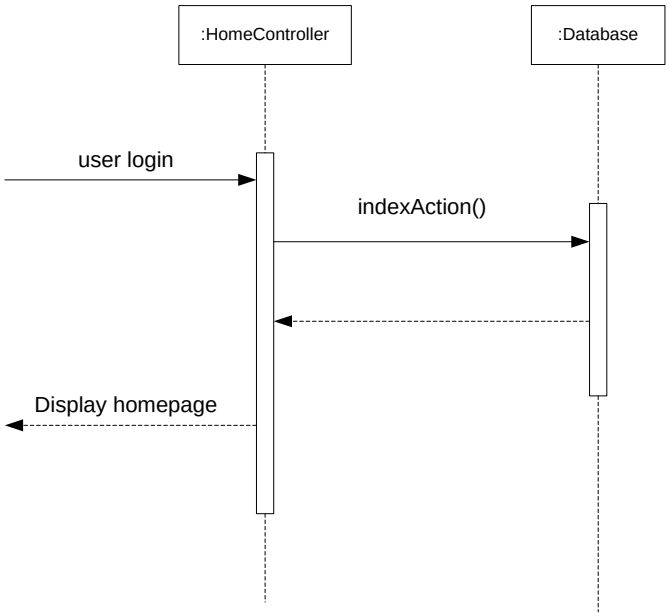
b. Interface Diagrams

Registration



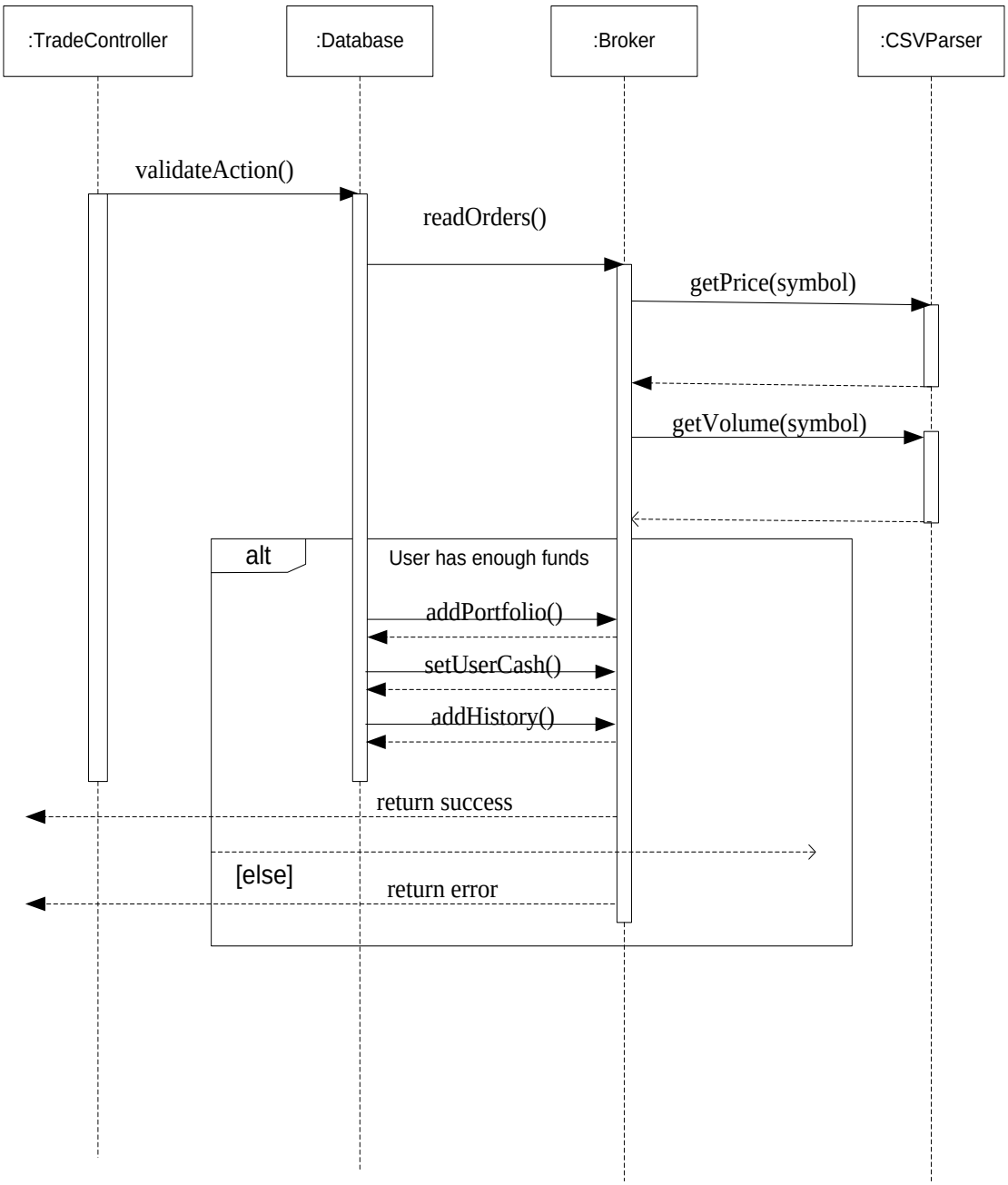
The above diagram shows Registration use case. When an user goes to the registration page, the system asks for username and password that the user wants to create. After that, system will request information from database and check information to see whether this username is already taken or not and 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 an user is logged into the system, the HomeController would ask information from the database and show the home page to the current user.

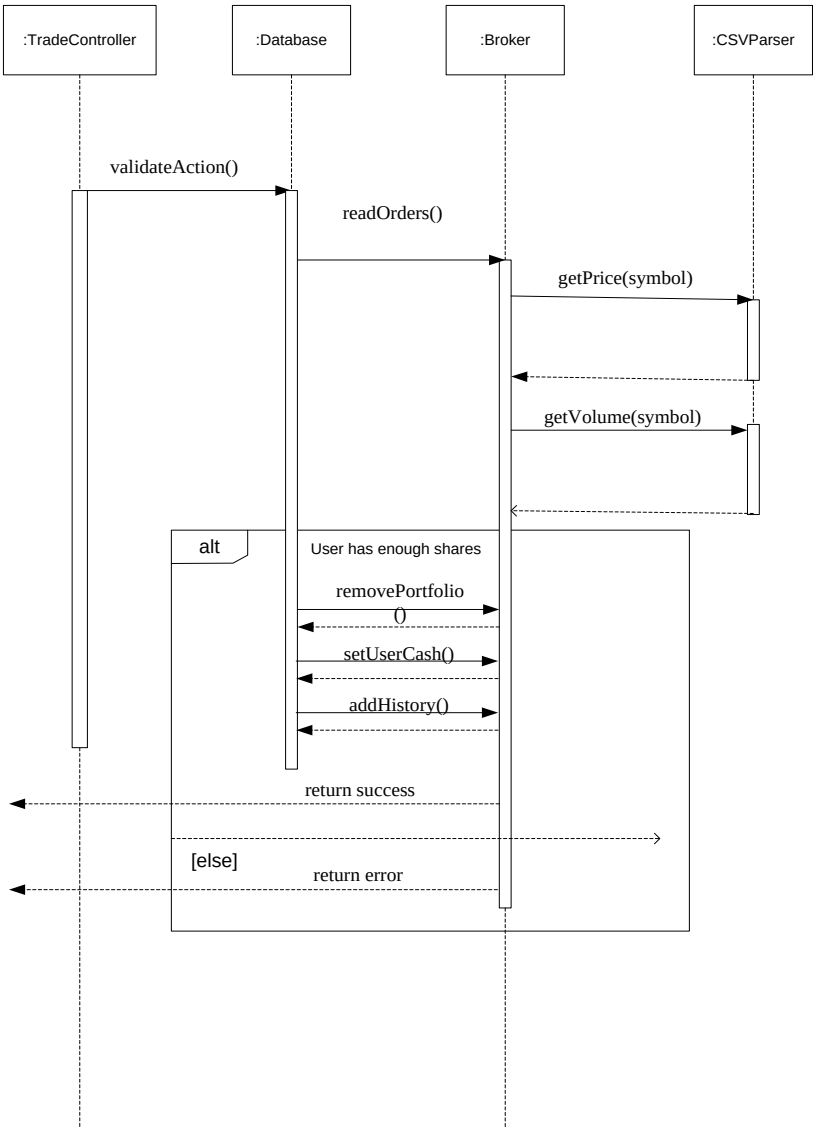
Buy Stocks



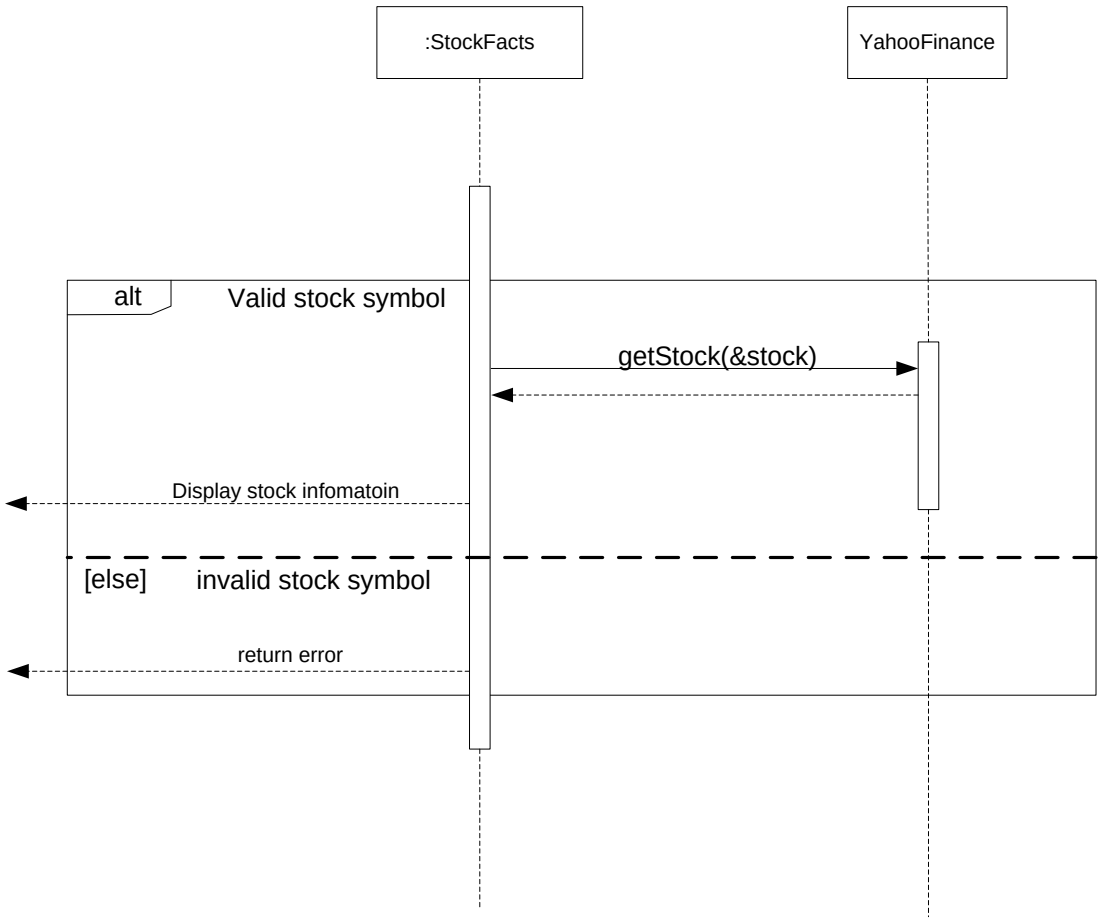
The use case of Buy Stock is represented in the above diagram. An user at the front end first initiates a buy action and specifies the type(market, limit, or stop), price, and volume of the order he wants. Then the backend fetch the order through database and query about the stock information from Yahoo 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. If the order is executed successfully, the system would notify user that the transaction has been completed.

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, or 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. If the order is executed successfully, the system would notify user that the transaction has been completed.

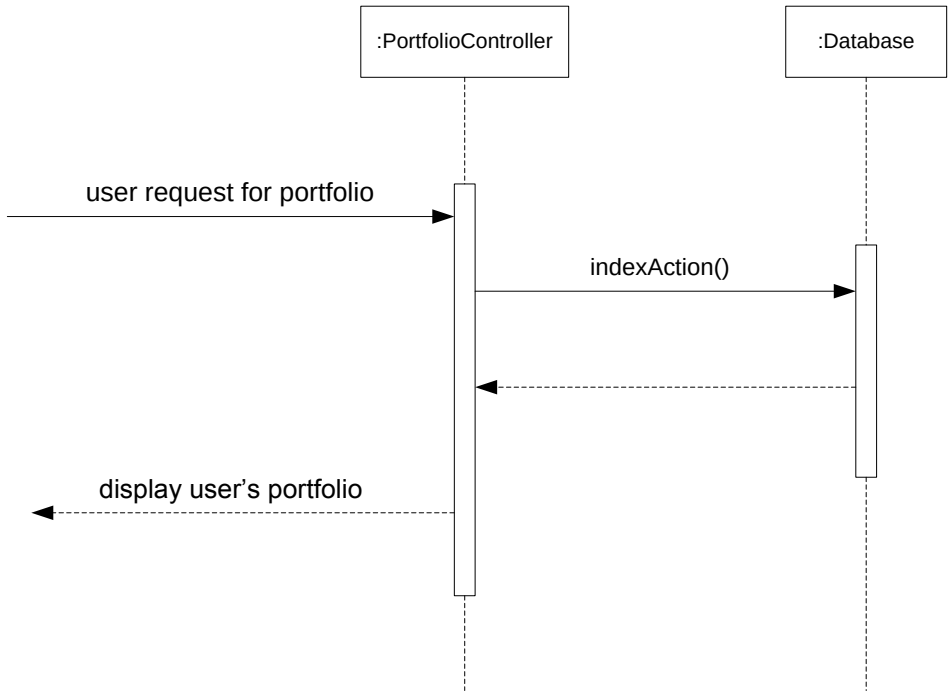


Research Stocks



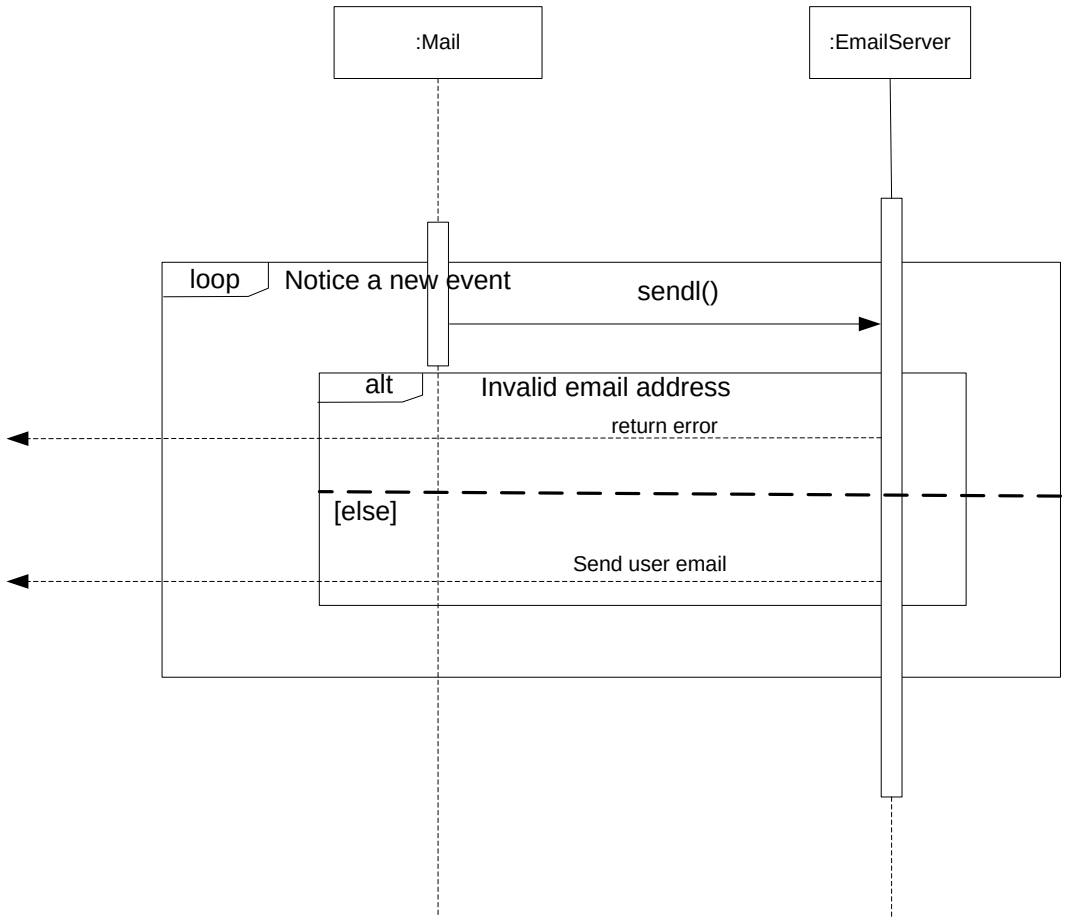
The above diagram shows Research Stocks use case. First user enters the desired stock symbol. If the symbol is valid, StockFacts queries Yahoo Finance for current price per share of the entered stock symbol as well as any available history about the stock, and then returns requested information to StockFacts.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

View Portfolio



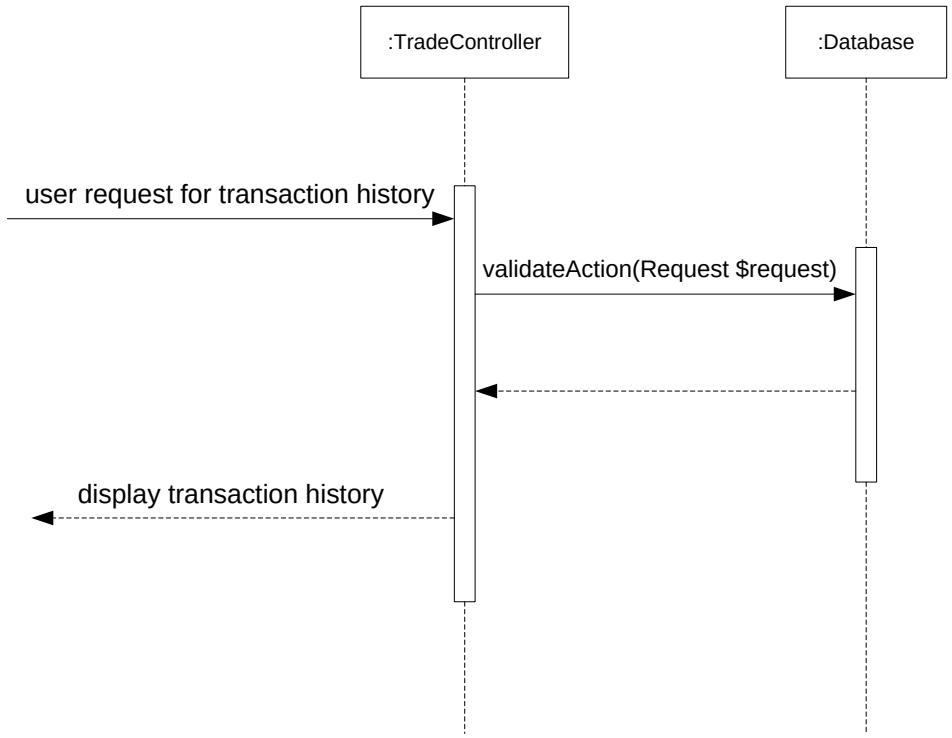
The above diagram shows the View Portfolio use case. After an user is logged into the system and asks for his/her portfolio, the PortfolioController would request information from the database and display the corresponding portfolio of the user.

Send Notifications



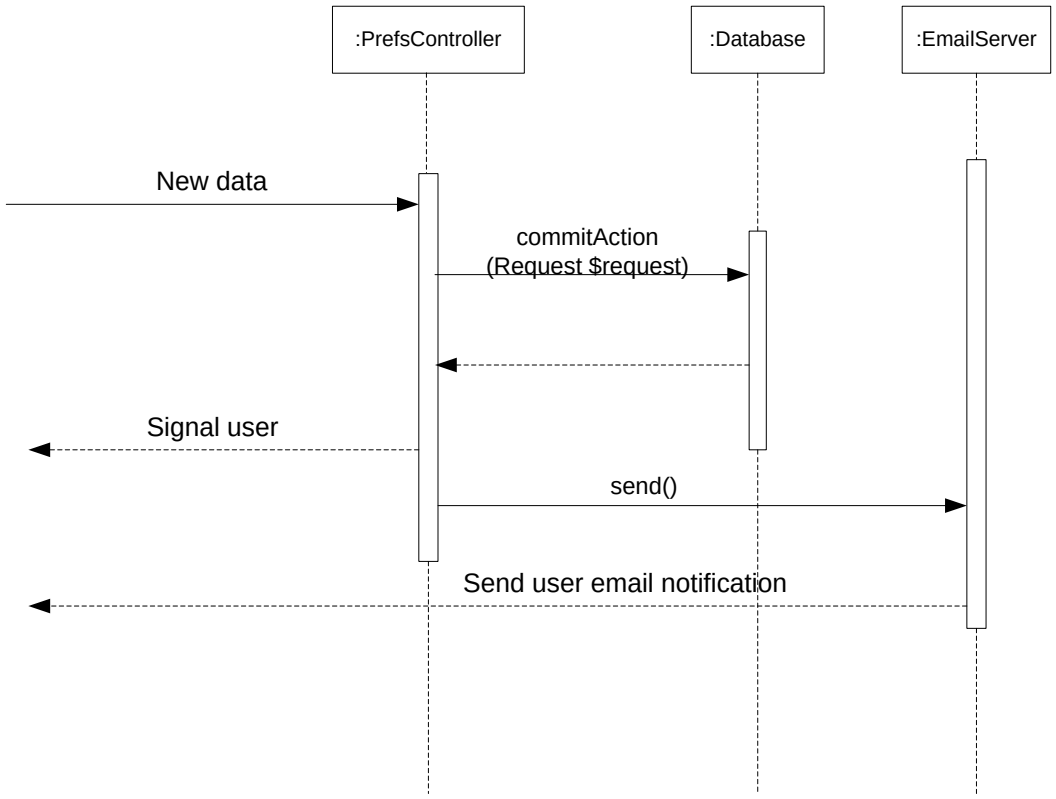
Send Notifications use case is showed as the above diagram. A loop is constantly waiting for a new event. After a new event is noticed, system signals email server (or SMS server) for sending notification to the user. If the email address is correct, then the user will be aware of what has happened.

View Transaction History



The above diagram shows the use case of View Transaction History. After an 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.

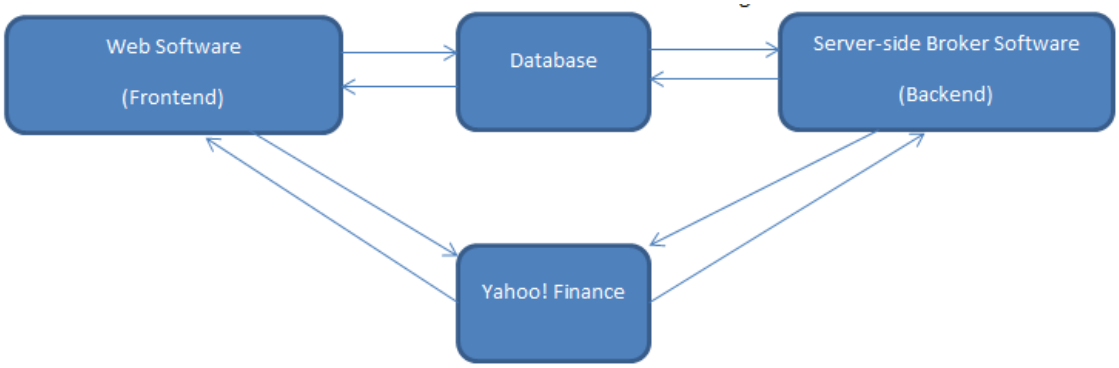
Preferences



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.

V. 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.

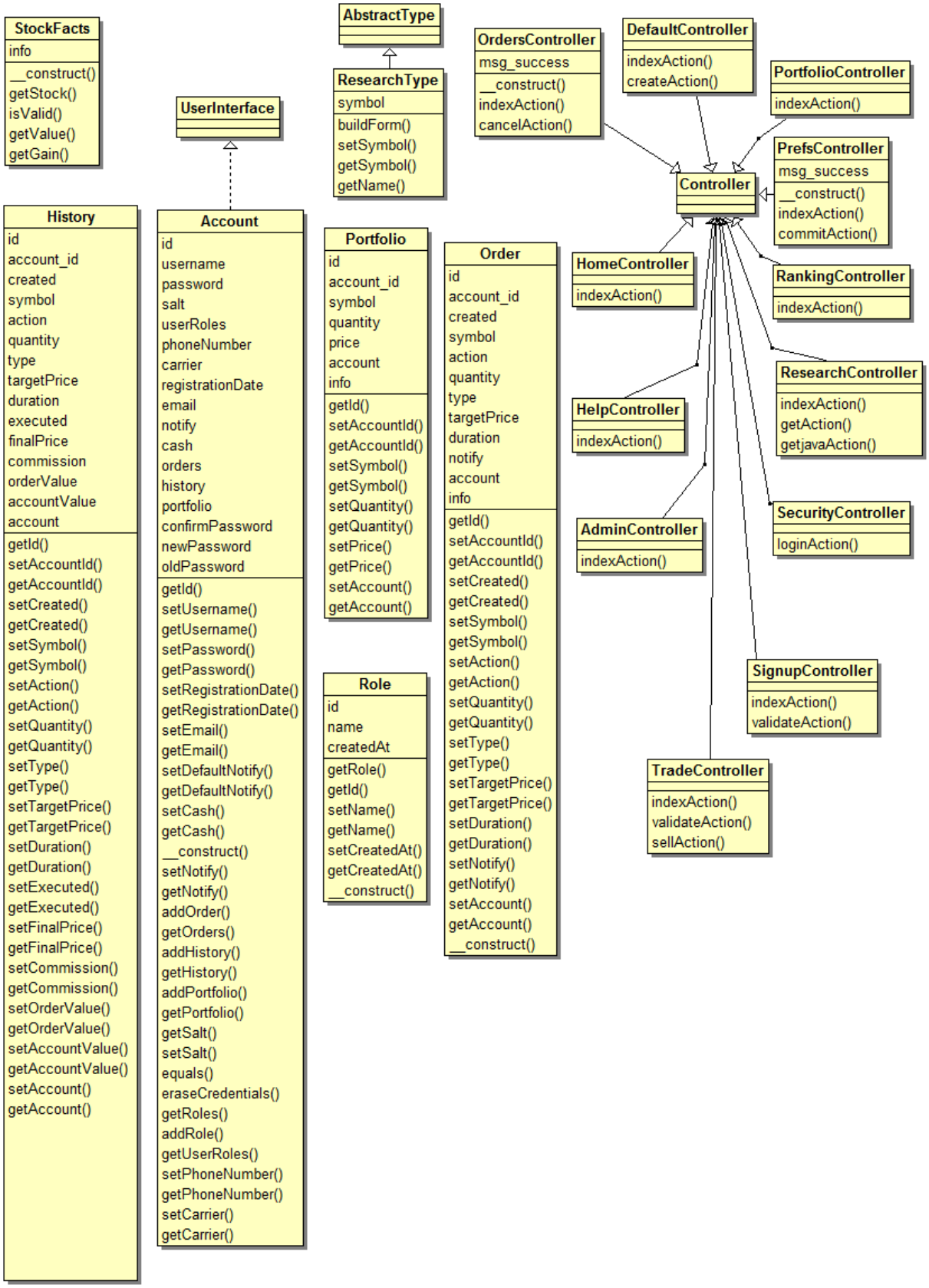


h. Class Diagrams

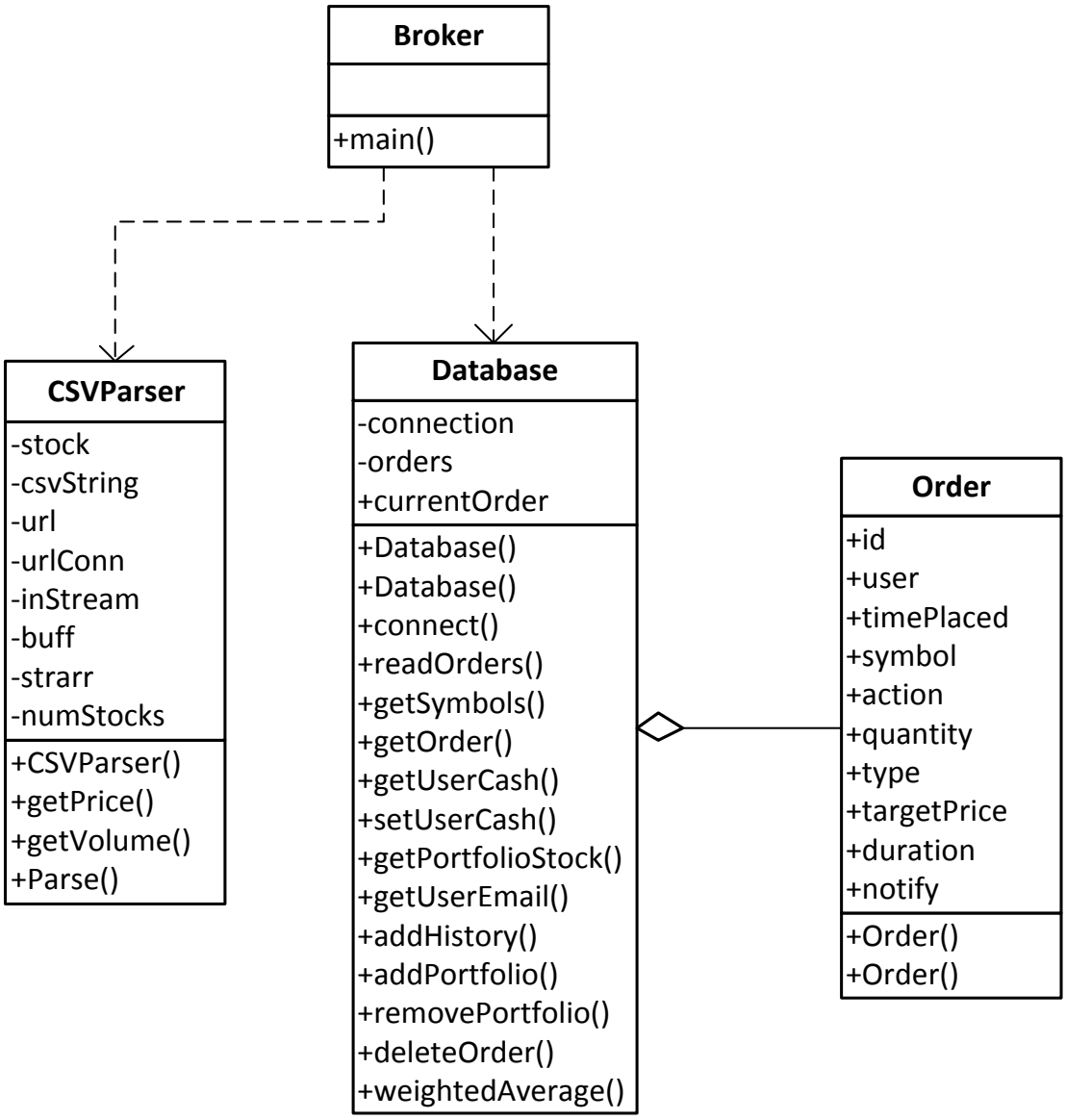
Frontend Class Diagram

In this UML 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.


StockFacts
info
__construct()
getStock()
isValid()
getValue()
getGain()
AbstractType
ResearchType
symbol
buildForm()
setSymbol()
getSymbol()
getName()
OrdersController
msg_success
__construct()
indexAction()
cancelAction()
DefaultController
indexAction()
createAction()
PortfolioController
indexAction()
PrefsController
msg_success
__construct()
indexAction()
commitAction()
Controller
UserInterface
History
id
account_id
created
symbol
action
quantity
type
targetPrice
duration
executed
finalPrice
commission
orderValue
accountValue
account
getId()
setAccountId()
getAccountId()
setCreated()
getCreated()
setSymbol()
getSymbol()
setAction()
getAction()
setQuantity()
getQuantity()
setType()
getType()
setTargetPrice()
getTargetPrice()
setDuration()
getDuration()
setExecuted()
getExecuted()
setFinalPrice()
getFinalPrice()
setCommission()
getCommission()
setOrderValue()
getOrderValue()
setAccountValue()
getAccountValue()
setAccount()
getAccount()
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()
Role
id
name
createdAt
getRole()
getId()
setName()
getName()
setCreatedAt()
getCreatedAt()
__construct()
Order
id
account_id
created
symbol
action
quantity
type
targetPrice
duration
notify
account
info
getId()
setAccountId()
getAccountId()
setCreated()
getCreated()
setSymbol()
getSymbol()
setAction()
getAction()
setQuantity()
getQuantity()
setType()
getType()
setTargetPrice()
getTargetPrice()
setDuration()
getDuration()
setNotify()
getNotify()
setAccount()
getAccount()
__construct()
HomeController
indexAction()
RankingController
indexAction()
ResearchController
indexAction()
getAction()
getjavaAction()
HelpController
indexAction()
AdminController
indexAction()
SecurityController
loginAction()
SignupController
indexAction()
validateAction()
TradeController
indexAction()
validateAction()
sellAction()

Backend Class Diagram



i. 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.

Entity Definitions

Entity Account:

Operations

+getId()
+setUsername($username)
+getUsername()
+setPassword($password)
+getPassword()
+setRegistrationDate($registrationDate)
+getRegistrationDate()
+setEmail($email)
+getEmail()
+setDefaultNotify($defaultNotify)
+getDefaultNotify()
+setCash($cash)
+getCash()
+__construct()
+setNotify($notify)
+getNotify()
+addOrder(\RU\TheStockHopBundle\Entity\Order $orders)
+getOrders()
+addHistory(\RU\TheStockHopBundle\Entity\History $history)
+getHistory()
+addPortfolio(\RU\TheStockHopBundle\Entity\Portfolio $portfolio)
+getPortfolio()
+getSalt()
+setSalt($value)
+equals(UserInterface $user)
+eraseCredentials()
+getRoles()
+addRole(\RU\TheStockHopBundle\Entity\Role $userRoles)
+getUserRoles()
+setPhoneNumber($phoneNumber)
+getPhoneNumber()
+setCarrier($carrier)
+getCarrier()

Attributes

$id;
$username;
$password;

$salt;
$userRoles;
$phoneNumber;
$carrier;
$registrationDate;
$email;
$notify;
$cash;
$orders;
$history;
$portfolio;
$confirmPassword;
$newPassword;
$oldPassword;

Entity Role:
Operations
+getRole()
+getId()
+setName($name)
+getName()
+setCreatedAt($createdAt)
+getCreatedAt()
+__construct()
Attributes
$id;
$name;
$createdAt;

Entity Orders:
Operations
+getId()
+setAccountId($accountId)
+getAccountId()
+setCreated($created)
+getCreated()
+setSymbol($symbol)
+getSymbol()
+setAction($action)
+getAction()
+setQuantity($quantity)
+getQuantity()

+setType($type)
+getType()
+setTargetPrice($targetPrice)
+getTargetPrice()
+setDuration($duration)
+getDuration()
+setNotify($notify)
+getNotify()
+setAccount(\RU\TheStockHopBundle\Entity\Account $account)
+getAccount()
+__construct()

Attributes
$id;
$account_id;
$created;
$symbol;
$action;
$quantity;
$type;
$targetPrice;
$duration;
$notify;
$account;
$info;

Entity History:

Operations
+getId()
+setAccountId($accountId)
+getAccountId()
+setCreated($created)
+getCreated()
+setSymbol($symbol)
+getSymbol()
+setAction($action)
+getAction()
+setQuantity($quantity)
+getQuantity()
+setType($type)
+getType()

+setTargetPrice($targetPrice)
+getTargetPrice()
+setDuration($duration)
+getDuration()
+setExecuted($executed)
+getExecuted()
+setFinalPrice($finalPrice)
+getFinalPrice()
+setCommission($commission)
+getCommission()
+setOrderValue($orderValue)
+getOrderValue()
+setAccountValue($accountValue)
+getAccountValue()
+setAccount(\RU\TheStockHopBundle\Entity\Account $account)
+getAccount()

Attributes
$id;
$account_id;
$created;
$symbol;
$action;
$quantity;
$type;
$targetPrice;
$duration;
$executed;
$finalPrice;
$commission;
$orderValue;
$accountValue;
$account;

Entity Portfolio:

Operations
+getId()
+setAccountId($accountId)
+getAccountId()
+setSymbol($symbol)
+getSymbol()

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

Attributes
$id;
$account_id;
$symbol;
$quantity;
$price;
$account;
$info;

Controller Definitions

PrefsController
Operators
+__construct()
+indexAction(Request $request)
+commitAction(Request $request)
+saveAction(Request $request)
Attributes
$msg_success
$msg_failure
$msg_other

ResearchController
Operators
+indexAction()
+getAction(Request $request)
+getjavaAction($name = "")
Attributes
$msg_success
$msg_failure
$msg_other

HelpController
Operators
+indexAction()

Attributes
$msg_success
$msg_failure
$msg_other

PortfolioController
Operators
+indexAction(Request $request)
Attributes
$msg_success
$msg_failure
$msg_other

SecurityController
Operators
+loginAction()
Attributes
$msg_success
$msg_failure
$msg_other

OrdersController
Operators
+__construct()
+indexAction()
+cancelAction($id)
Attributes
$msg_success
$msg_failure
$msg_other

HomeController
Operators
+indexAction()
Attributes
$msg_success
$msg_failure
$msg_other

RankingController
Operators
+indexAction(Request $request)

Attributes
$msg_success
$msg_failure
$msg_other

AdminController
Operators
+indexAction()

SignupController
Operations
+indexAction(Request $request)
+validateAction(Request $request)
Attributes
$msg_success
$msg_failure
$msg_other

TradeController
Operations
+indexAction(Request $request)
+validateAction(Request $request)
+buyAction(Request $request)
+sellAction(Request $request)
Attributes
$msg_success
$msg_failure
$msg_other

Backend

Broker
Operations
+ main(): void

CSVParser
Attributes
- stock : String
- csvString : String

- url : URL
- urlConn : URLConnection
- inStream : InputStreamReader
- buff : BufferedReader
- strarr[][] : String
- numStocks : int

Operations

+ 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 (user : String) : BigDecimal
+ setUserCash (user : String, cashMoney : BigDecimal) : void
+ getPortfolioStock (user : String, symbol : String) : long
+ getUserEmail (user : String) : String
+ addHistory (user : String, timePlaced : Timestamp, symbol : String, action : String, quantity : long, type : String, targetPrice : BigDecimal, duration : String, price : BigDecimal, commission : BigDecimal, orderValue : BigDecimal, accountValue : BigDecimal) : void
+ addPortfolio (user : String, symbol : String, quantity : long, price : BigDecimal) : void
+ removePortfolio (user : String, symbol : String, quantity : long) : void
+ deleteOrder (id : long) : void
- weightedAverage (price1 : BigDecimal, quantity1Long : long, price2 : BigDecimal, quantity2long : long) : BigDecimal

Order

Attributes

+ id : long
+ user : String
+ timePlaced : Timestamp

+ symbol : String
+ action : String
+ quantity : long
+ type : String
+ targetPrice : BigDecimal
+ duration : String
+ notify : String

Operations

+ Order ()
+ Order (id : long, user : String, timePlaced : Timestamp, symbol : String, action : String, quantity : long, type : String, targetPrice : BigDecimal, duration : String, notify : String)

VI. System Architecture and System Design

a. Architectural Styles

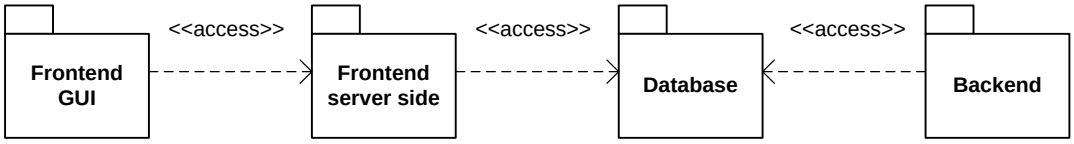
Before speaking of the kinds of architectural styles we used in thestockhop, it is useful to provide a definition of an architectural style. “An architectural style … defines a family of systems in terms or a pattern of structural organization. More specifically, an architectural style determines the 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



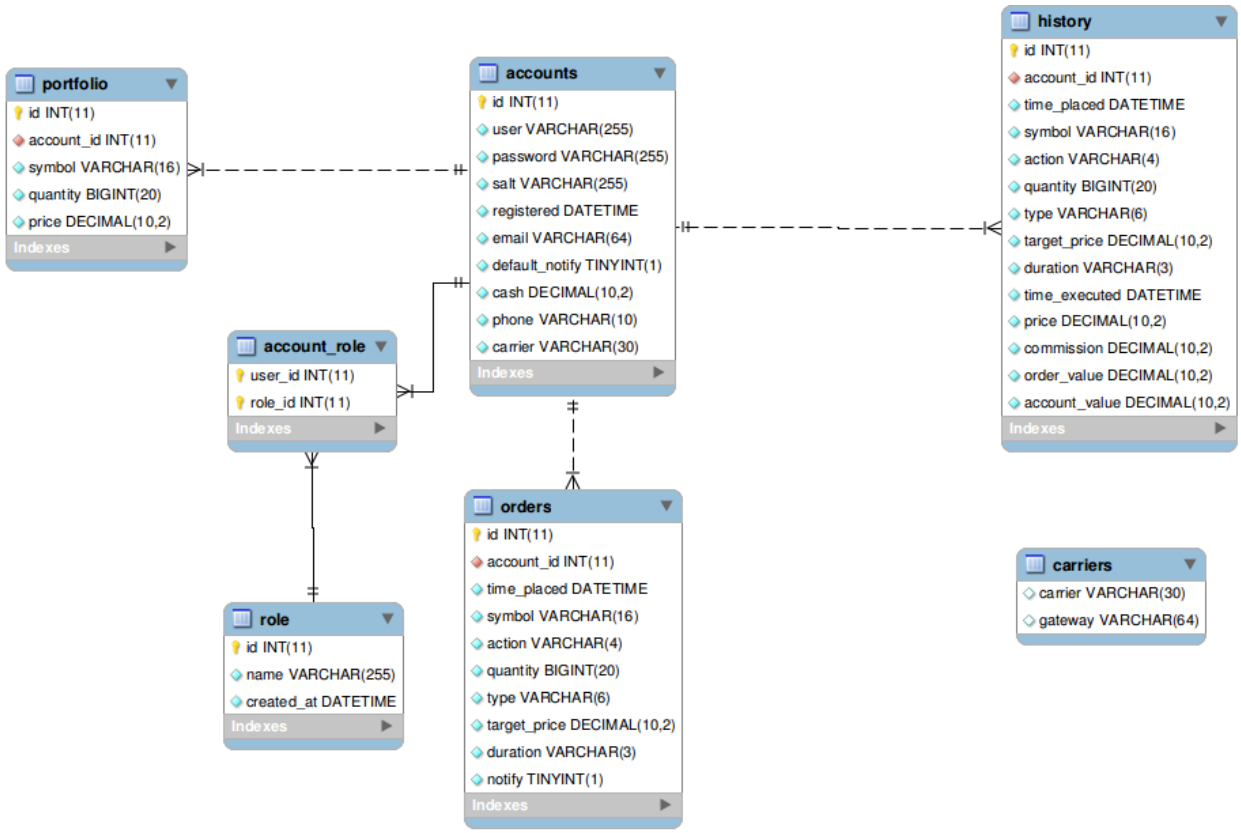
The frontend GUI provides the interface by which a user can interact with the system and view data from the system. The front end server side retrieves data from the database and Yahoo Finance and presents it to the user via the frontend GUI. It also stores new stock orders in the database. The database 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 seven tables within the MySQL database entitled thestockhop. These tables are as follows: accounts, account_role, role, orders, history, portfolio, 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.

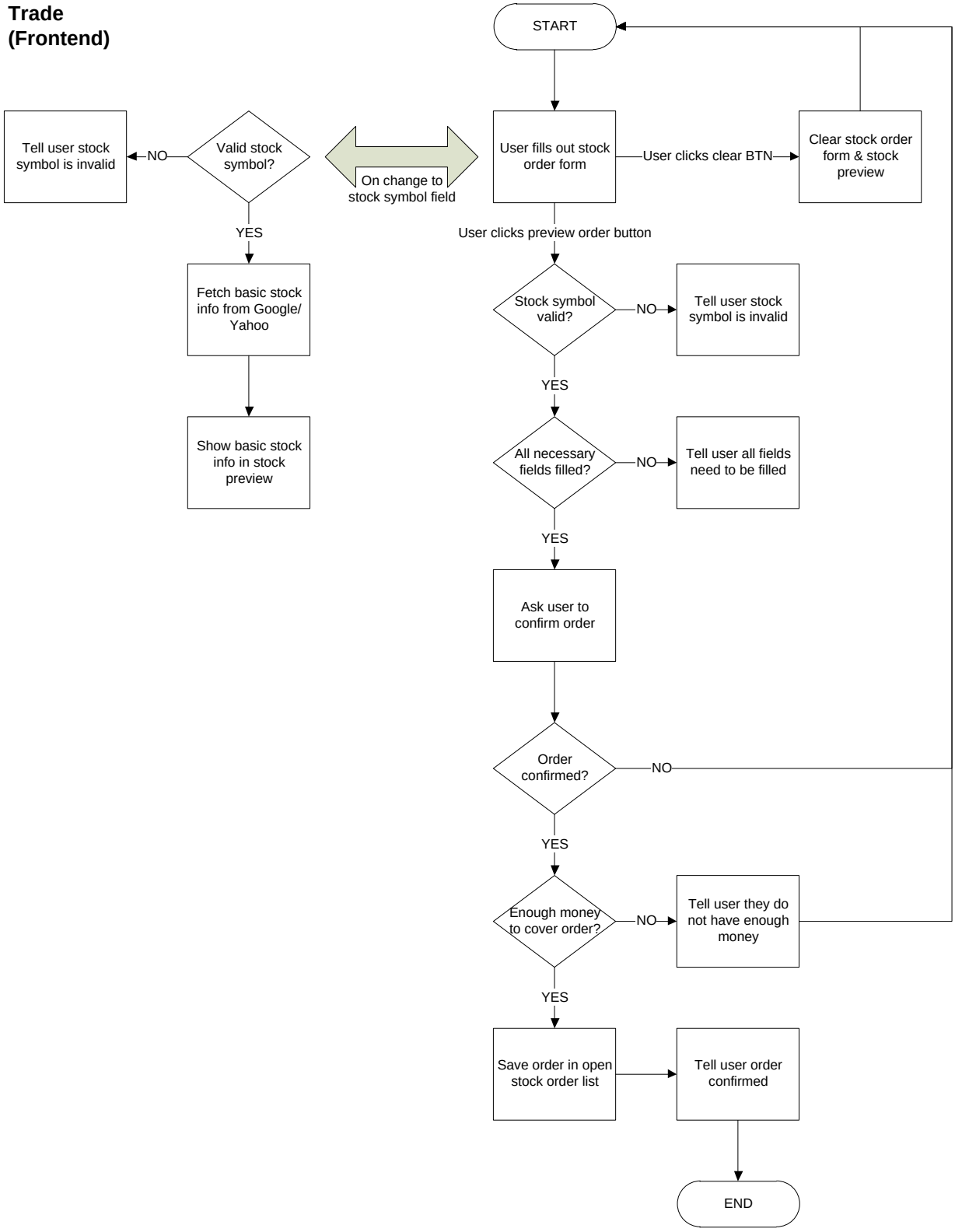
Client-Side

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).

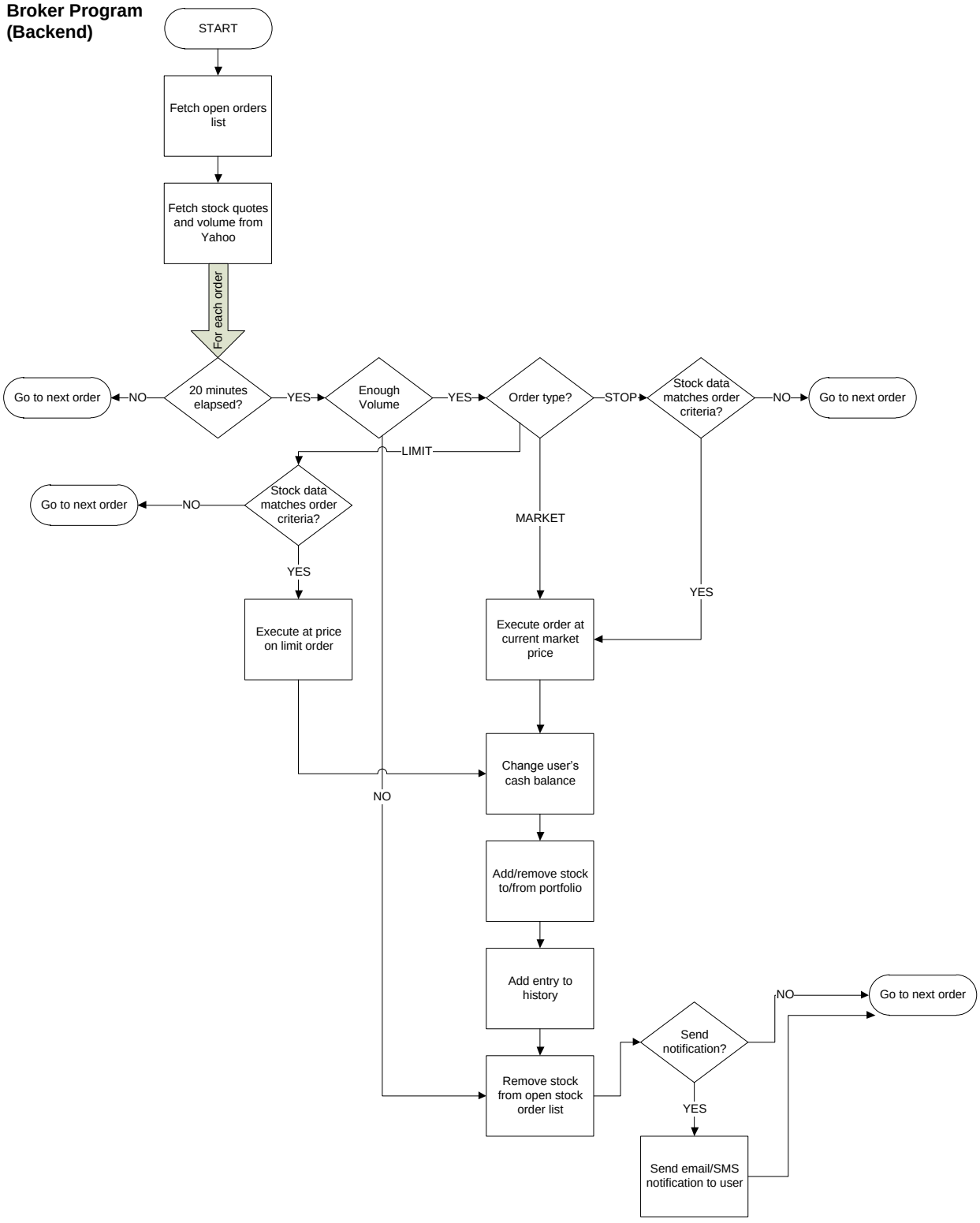
h. Program Flow Diagrams

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)



VII. Algorithms and Data Structures

a. Algorithms

The original incarnations of TheStockHop do not have complex algorithms but they do follow a well-defined Program Flow (See Broker Backend Program). It is planned for Demo 2 to implement more complex stock analysis algorithms in order to separate TheStockHop from existing stock market fantasy league games.

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, or stop order
targetPrice	BigDecimal	Target price to determine execution (applies only to limit and 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

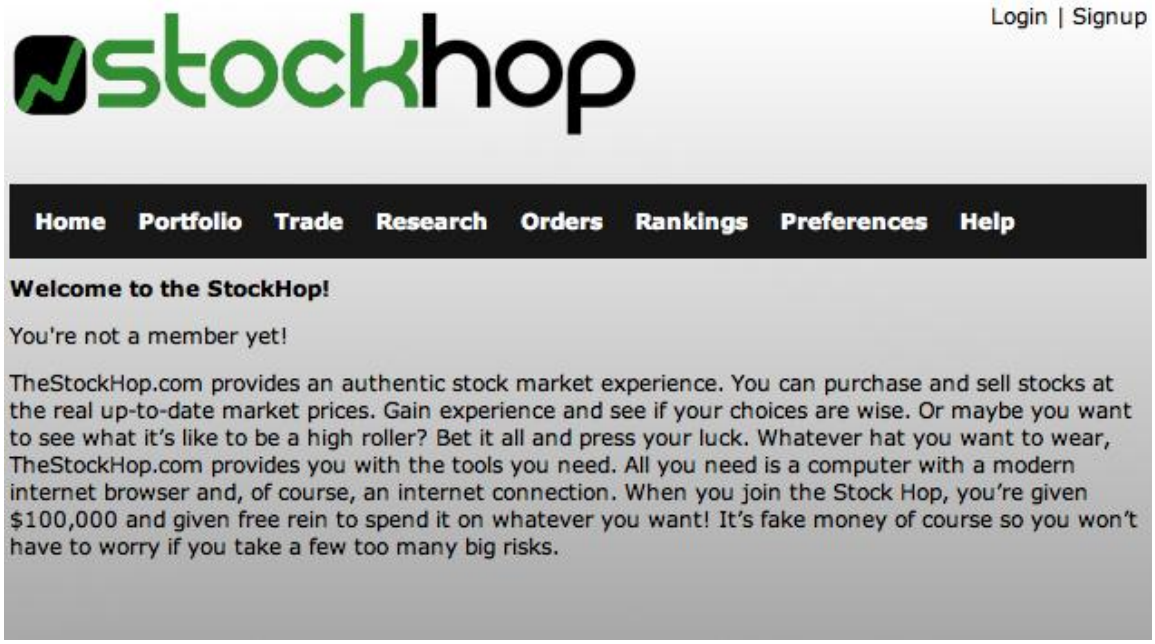
manipulate data within this object. Finally, the object can be posted to the database using the PHP PDO connections described above. In the instance where data is being collected from a particular table, the $user type (an Account entity) can be dereferenced: $user->getPortfolio() accesses the Entity to obtain the entries in the portfolio table associated with a given user.

VII. User Interface Design and Implementation

We have stuck somewhat close to our original screen mock-ups. Aside from some color changes, and the presence of a logo, the site is essentially the same. Major differences include the removal of the data 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.

Introduction:

The first thing a visitor will notice about StockHop, is how simple everything is; the main goal of website creators is to avoid complexity and make everything as easy as possible.



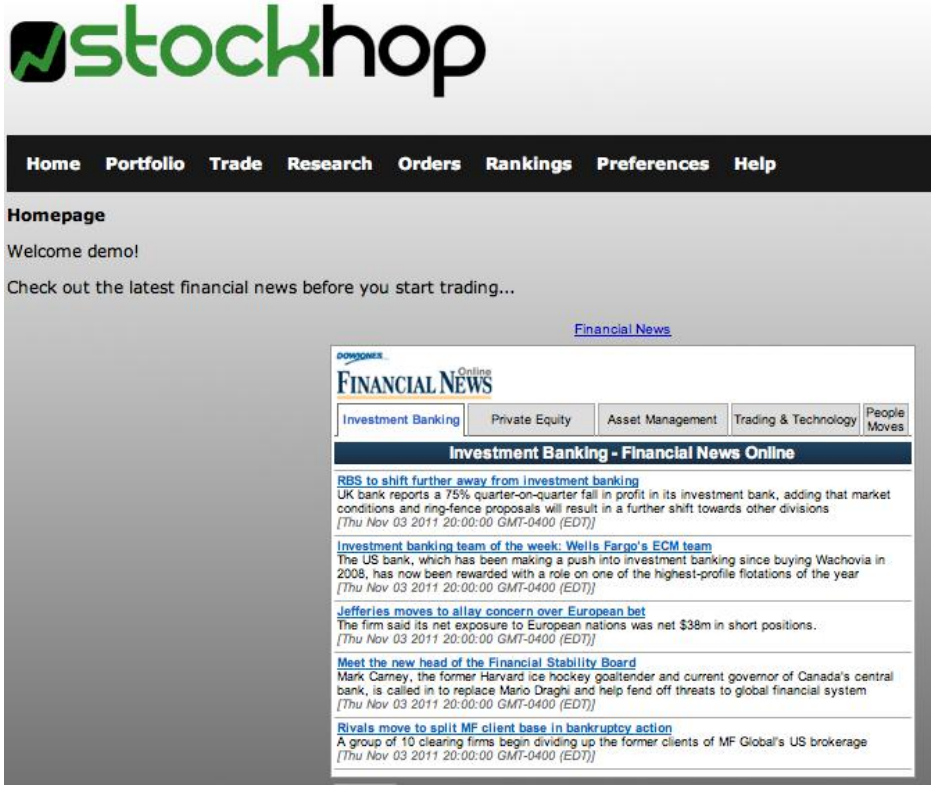
Sign up and Login:

Now, sign up and login to your StockHop profile could not be simpler. StockHop avoids requiring sensitive and detailed information from users. They can sign up and make a profile in just few seconds.

Homepage:

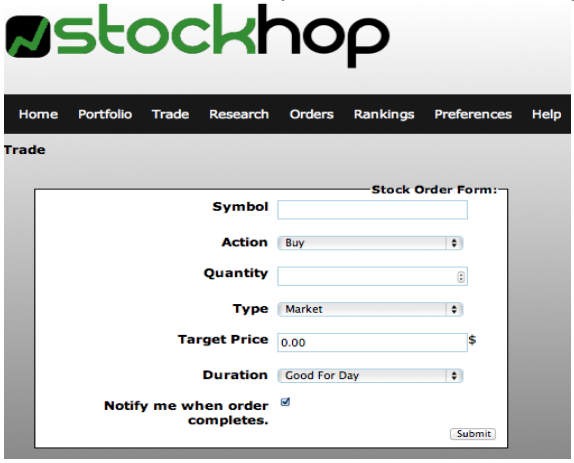
After signing on, the user is presented with the latest finance news and events in his/her profile with no need to have different tabs open in your browsers for the latest stock news; it's all there in the profile homepage.

In addition, after signing on, you have access to every section of the website, the tabs at the top will help you navigate the various sections of the site.



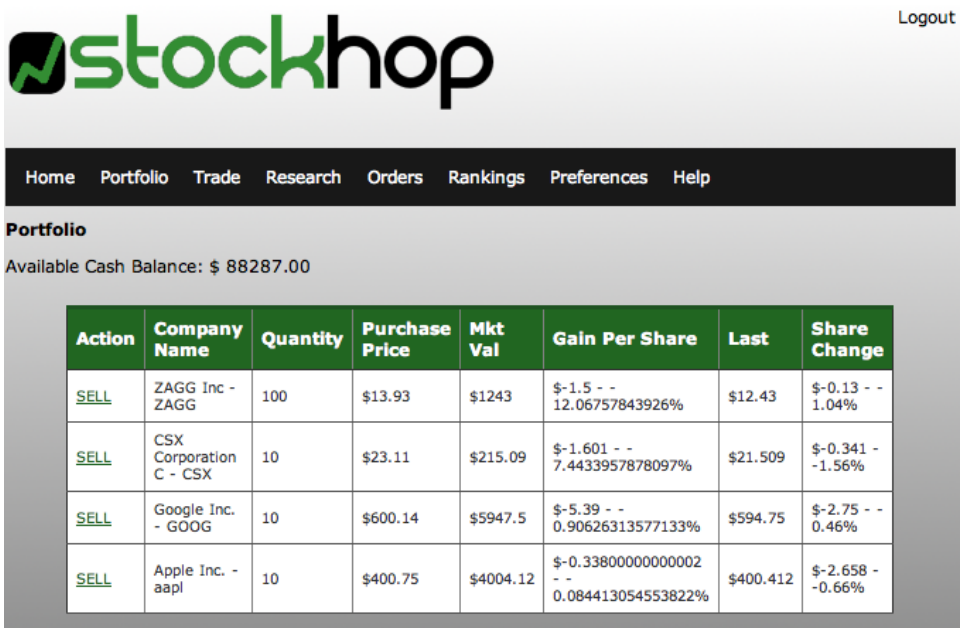
Trading:

We understand that when a user want to buy or sell stocks, It may be useful to double check such a decision at the last minute; StockHop will populate the trade screen with relevant information to the stock you wish to trade. The relevant information, similar to what the research screen would provide, will immediately appear just right of the trade form..



Portfolio:

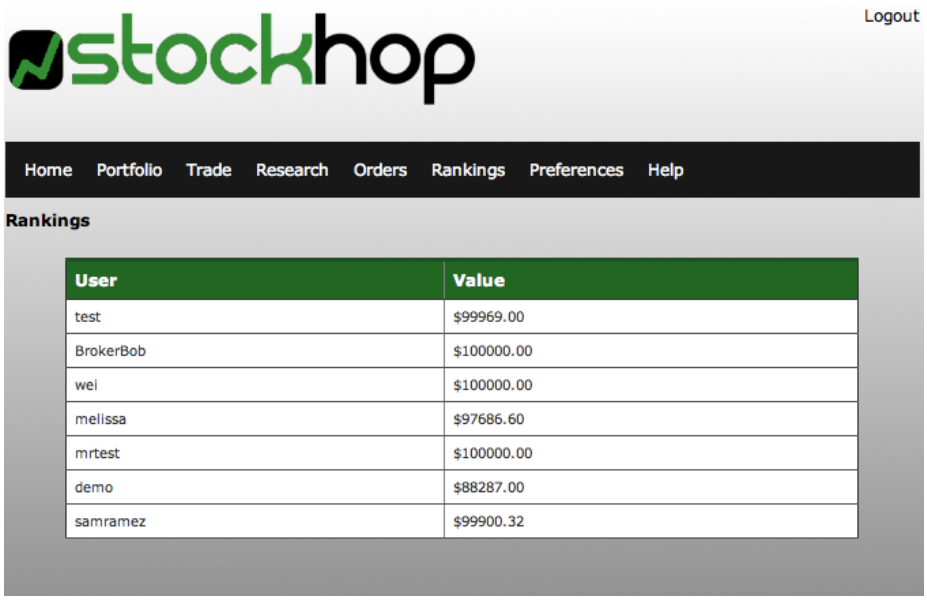
We let users to easily check the performance of the stocks they own in the portfolio section. Necessary information about owned stocks, such as stock quantity, current market value, amount of money gained per share, and Share change are provided in portfolio section. In the event you decide it is a favorable time to drop your position in a certain stock, there is a quick link sell option that will take you to the trade form and populate it with information about the stock you may desire to sell.



Action	Company Name	Quantity	Purchase Price	Mkt Val	Gain Per Share	Last	Share Change
SELL	ZAGG Inc - ZAGG	100	$13.93	$1243	$-1.5 - - 12.06757843926%	$12.43	$-0.13 - - 1.04%
SELL	CSX Corporation C - CSX	10	$23.11	$215.09	$-1.601 - - 7.4433957878097%	$21.509	$-0.341 - -1.56%
SELL	Google Inc. - GOOG	10	$600.14	$5947.5	$-5.39 - - 0.90626313577133%	$594.75	$-2.75 - - 0.46%
SELL	Apple Inc. - aapl	10	$400.75	$4004.12	$-0.33800000000002 - - 0.084413054553822%	$400.412	$-2.658 - -0.66%

Rankings:

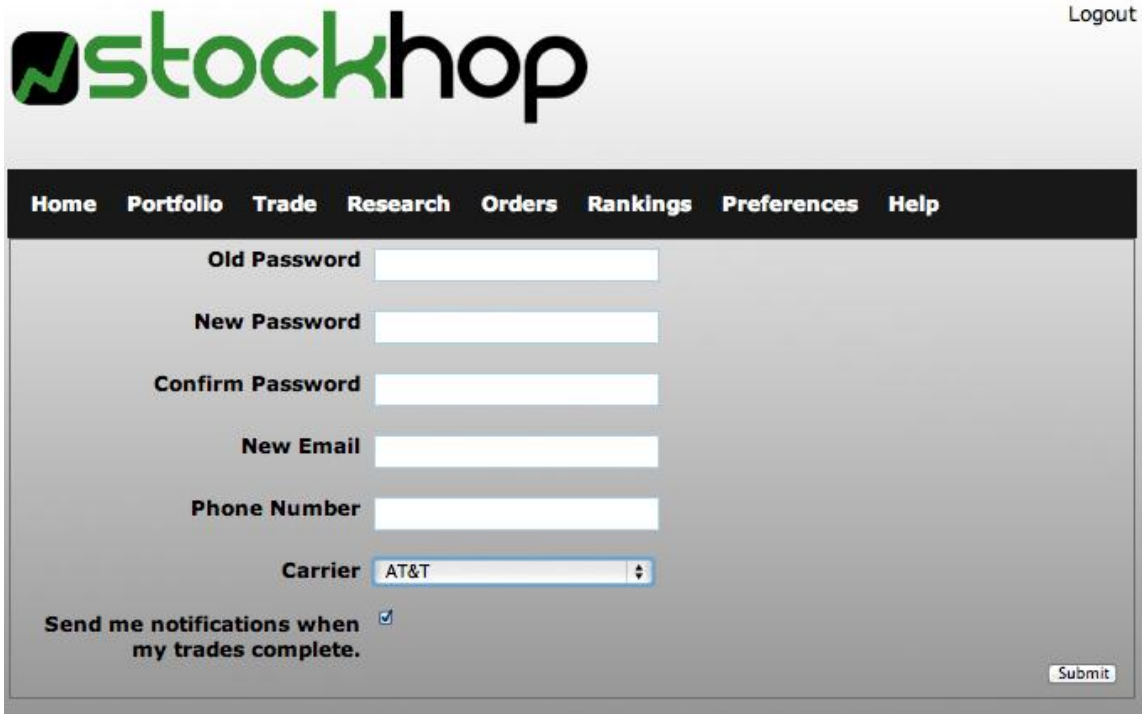
Are you interested in knowing your success in trading among others? If yes, you can go to Rankings section of the website and find it out yourself.



Preferences:
If you are not satisfied with your account information, you can always go to Preference Section and change them. Email address, password, notification default, and mobile information can all be modified here.

Notifications:

In order to make TheStockHop more convenient, users have the ability to add mobile notification in addition to email notification; in this case, you will be notified about your transaction status when you're on the move.



IX. Progress Report and Plan of Work

a. Progress Report

The table below shows the percentage done of code for each use case.

Use Case Number	Use Case Name	Percentage Completed
1	Registration	100%
2	Sign-In	100%
3	View Home Page	100%
4	Buy Stocks	50%
5	Sell Stocks	25%
6	Research Stocks	75%
7	Send Notifications	100%
8	View Rankings	25%
9	User sign-out	100%
10	View Help	50%
11	View Transaction History	100%
12	View Pending Transaction History	100%
13	View Portfolio	100%
14	User Preferences	100%
15	Manage Advertisements	0%
16	Maintain Website	25%
17	Update Stock Prices	100%

Currently, we are debugging some backend features that do not fully work. In addition, the frontend is getting extensive error checking and data validation functions for all the forms. The CSS is also being modified. The frontend sell features also need to be changed.

Over the next week, the team will start implementing the advertiser interface and then will add the advanced stock analysis to the Research page.

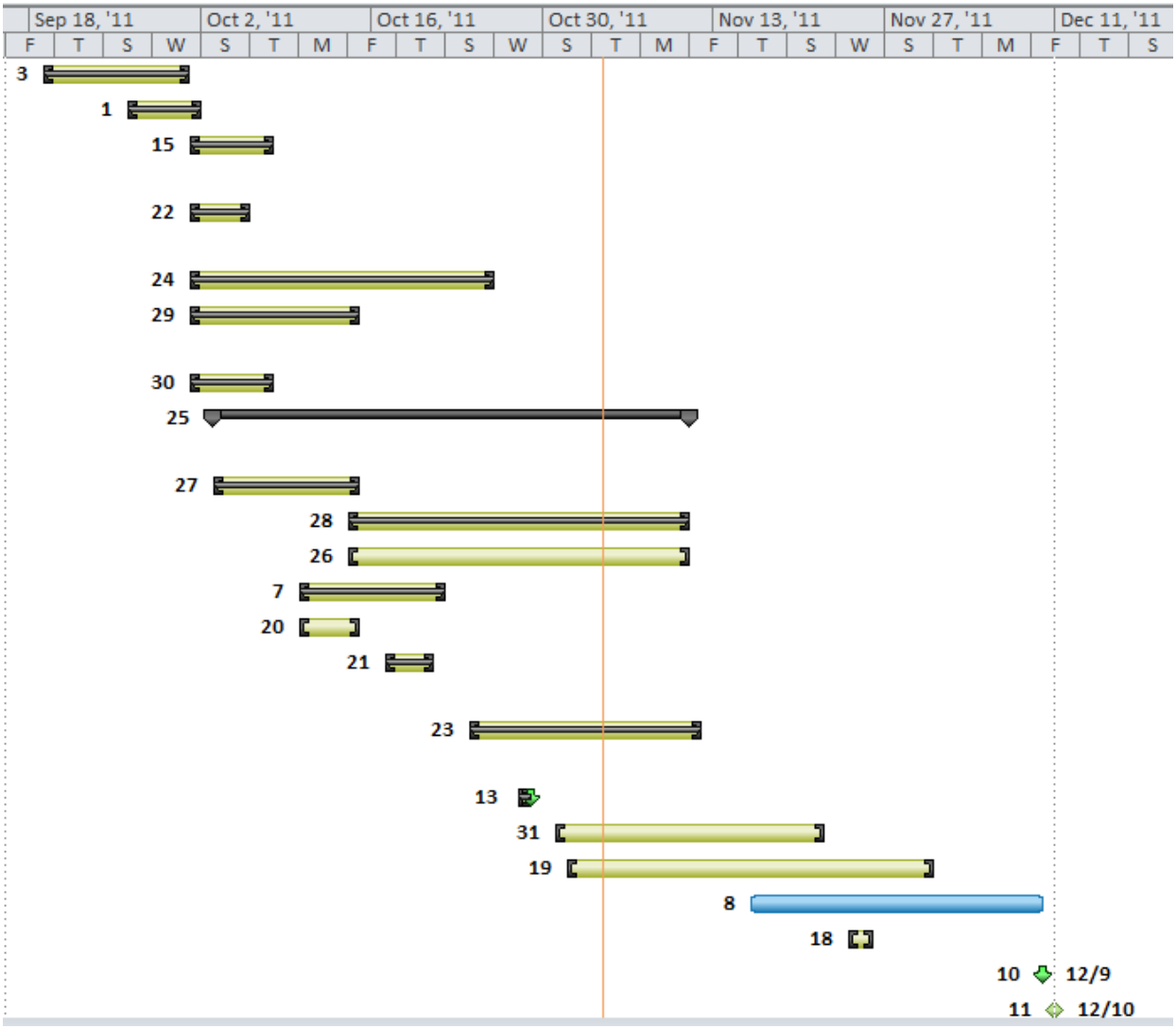
b. Plan of Work

The unique ID is used to show the display of each task in the Gantt Chart.

Completed	Unique ID	Task Name	Duration	Start	Finish
100%	**8**	**Project Proposal**	**1 day**	**Fri 9/16/11**	**Fri 9/16/11**
100%	**2**	**Report 1**	**11 days**	**Mon 9/19/11**	**Fri 9/30/11**
100%	1	Server Setup	6 days	Mon 9/26/11	Sat 10/1/11
100%	9	Code: Registration, Login, Logout	6 days	Sat 10/1/11	Fri 10/7/11
100%	14	Code: Transaction History	4 days	Sat 10/1/11	Wed 10/5/11
100%	16	Code: Trade Screen	18 days	Sat 10/1/11	Tue 10/25/11
100%	21	Code: Yahoo Finance Data Gatherer	11 days	Sat 10/1/11	Fri 10/14/11
100%	22	Code: Portfolio	6 days	Sat 10/1/11	Fri 10/7/11
50%	**17**	**Code: Buy and Sell Features**	**29 days**	**Mon 10/3/11**	**Thu 11/10/11**
100%	18	Code: Market Order	10 days	Mon 10/3/11	Fri 10/14/11
50%	19	Code: Stop Order	20 days	Fri 10/14/11	Thu 11/10/11
50%	20	Code: Limit Order	20 days	Fri 10/14/11	Thu 11/10/11
100%	**3**	**Report 2**	**10 days**	**Mon 10/10/11**	**Fri 11/4/11**
0%	12	Code: Advertisements	5 days	Mon 10/10/11	Fri 10/14/11
100%	13	Code: Email and SMS Notifications	4 days	Mon 10/17/11	Thu 10/20/11
50%	15	Code: Pending Transactions	15 days	Mon 10/24/11	Fri 11/11/11
100%	**5**	**Demo 1**	**1 day**	**Fri 10/28/11**	**Fri 10/28/11**
10%	23	Advanced Stock Analysis	16 days	Mon 10/31/11	Mon 11/21/11
0%	11	Code: Mobile Site	22 days	Tue 11/1/11	Wed 11/30/11
0%	4	Report 3	18 days	Wed 11/16/11	Fri 12/9/11
0%	10	Code: RSS Feeds	2 days	Thu 11/24/11	Fri 11/25/11
N/A	**6**	**Demo 2**	**1 day**	**Fri 12/9/11**	**Fri 12/9/11**
0%	7	Electronic Project Archive	1 day	Sat 12/10/11	Sat 12/10/11

Gantt Chart:

Green or blue bars with a black line going through them indicate that that part of the task has reached completion.



c. Breakdown of Responsibilities

Team Member	Responsibilities
Dan M.	• Use Case Design • Partial System Design • Backend Programming • Backend Code Integration • Class Diagram and Interface Specification (Backend) • System functional structure (frontend/backend and their substructures)

Jakub K.	• System functional structure (frontend/backend and their substructures) • GUI design • Database and table design • Wrote database interaction wrapper class and associated "Order" data structure for backend • Wrote email and SMS notification classes and implementation code for the backend • Testing and debugging of the backend "broker" functional code • Program Flow Diagrams
Melissa R.	• Project Leader (organize meeting times, allocate resources for particular tasks, project schedule/planning) • Server Administration (built backend server from scratch, implemented the mail server, database server, and all necessary programs) • System functional structure (frontend/backend and their substructures) • Coded entire frontend – including all pages in Symfony2 framework, created Doctrine entity classes to interact with controllers [all classes/modules] • HTML/CSS/Twig templating to provide styling for the website • FURPS, Functional Requirement Development • Frontend/backend/database integration testing
Priyanka K.	• Domain Analysis, Report 1 • PHP Function to Grab Stock Data from Yahoo! Finance for frontend • PHP Function to message users to confirm sign-up • System functional structure (frontend/backend and their substructures) • Researched architectural styles • Researched stock analysis algorithms • Website Testing
Sam R.	• Use Case Development • User Interface Design, Report 2 • Human Factors Analysis • Incorporated reviewer comments from Report 1 into Report 2 • Research Java code for sending email and text messages • System functional structure (frontend/backend and their substructures) • Website Testing
Wei S.	• Domain Analysis, Report 1 • Coded backend functionality to grab data from Yahoo! Finance • Wrote Java code (CSVParse.java) to parse this data, and put it in an array that was used by the backend to facilitate trading

	functionality • System functional structure (frontend/backend and their substructures) • Website Testing • Interaction Diagrams, Report 2

X. 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.

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
```

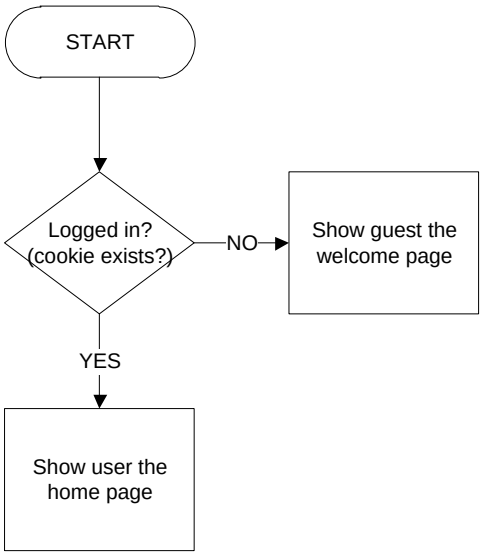
```
          vendor: Linux 2.6.18-274.3.1.el5 ehci_hcd
          physical id: 1
          bus info: usb@1
          logical name: usb1
          version: 2.06
          capabilities: usb-2.00
          configuration: driver=hub slots=8 speed=480Mbit/s
     *-pci:3
          description: PCI bridge
          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
```

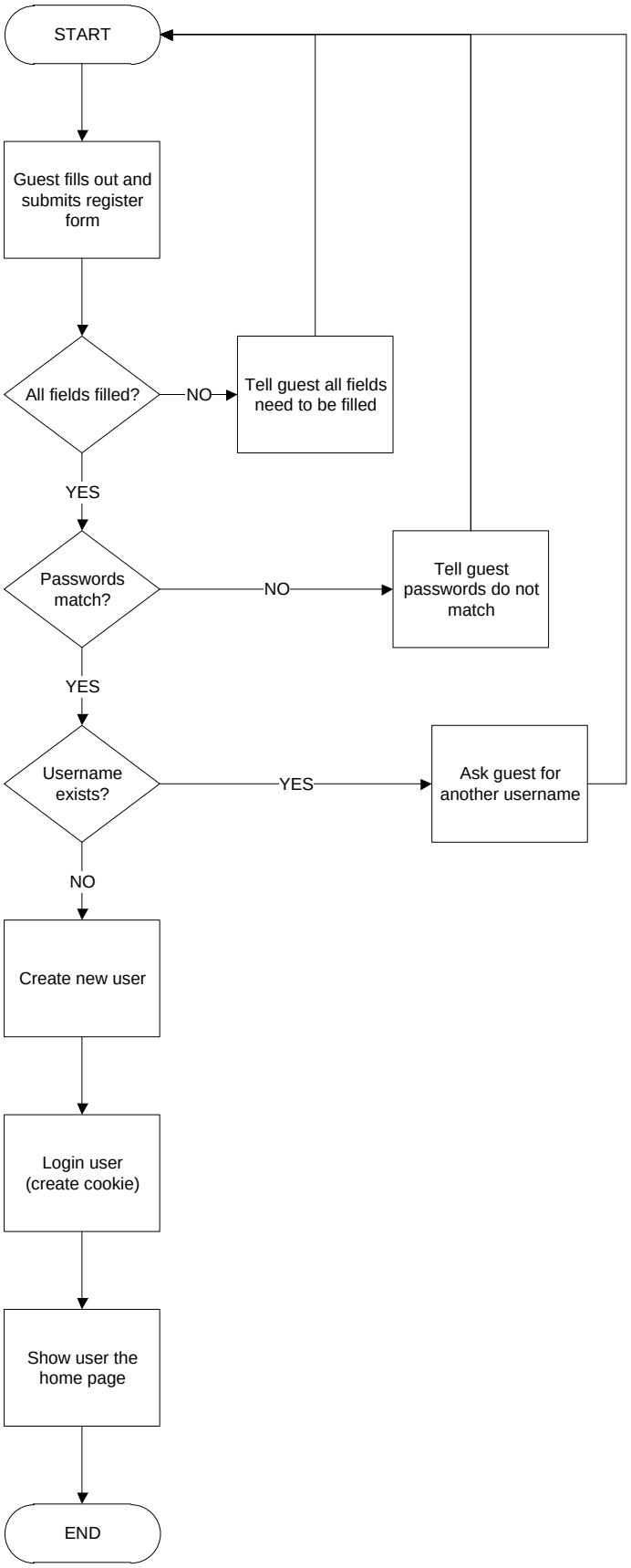
```
version: 01
width: 32 bits
clock: 33MHz
configuration: driver=i801_smbus latency=0
resources: irq:185 ioport:2000(size=32)
```

A.2 Complete Flow Chart Program Flow

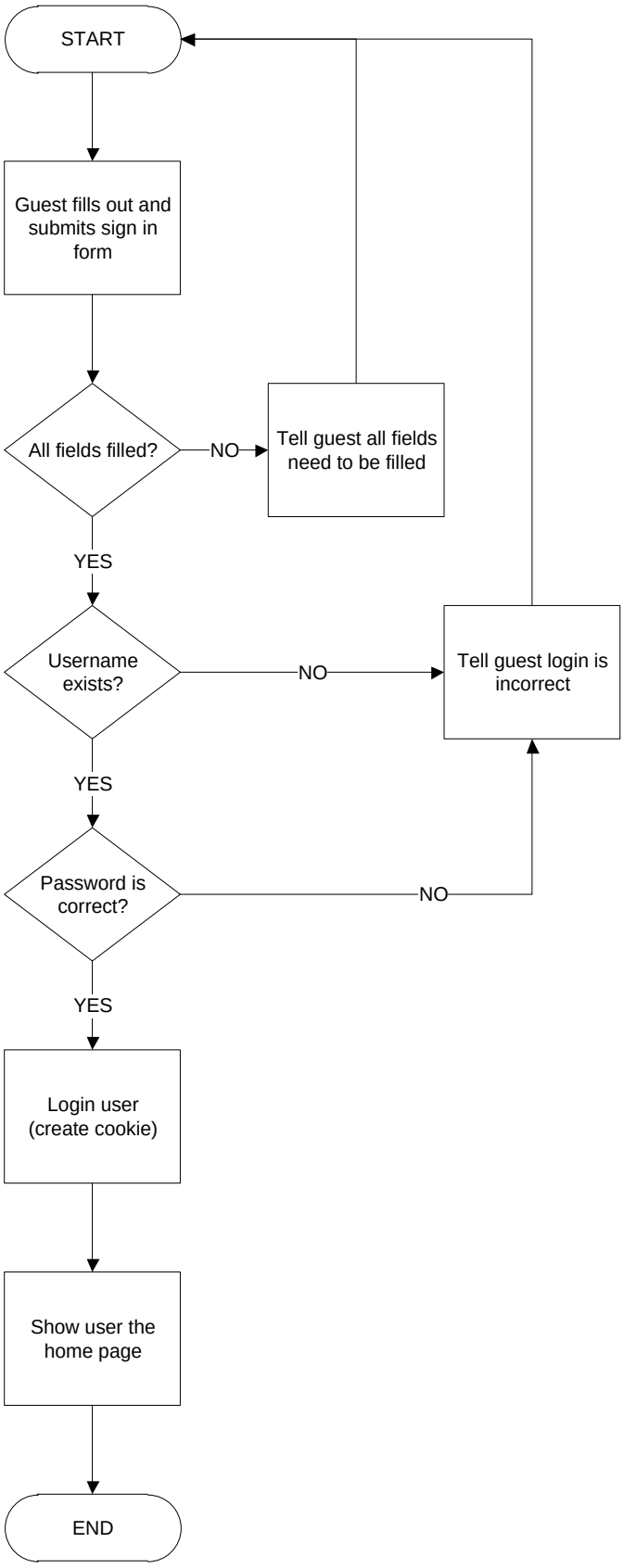
Check Login


START
Logged in?
(cookie exists?)
NO
Show guest the
welcome page
YES
Show user the
home page

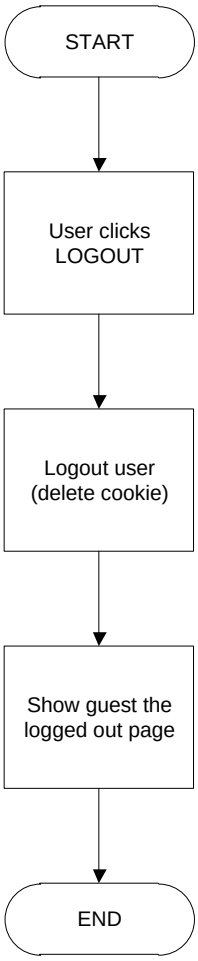
Register


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

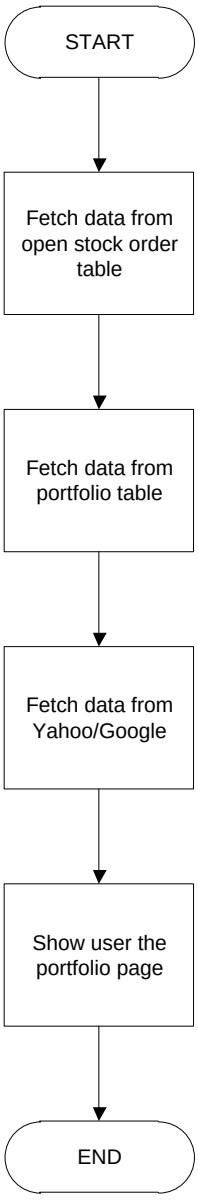
Sign In


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

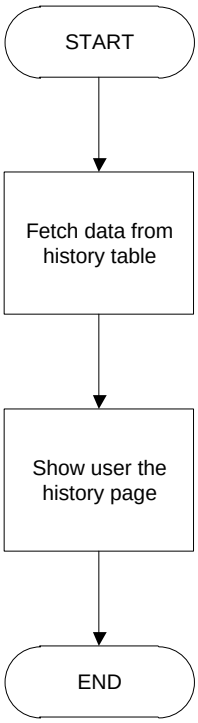
Sign Out


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

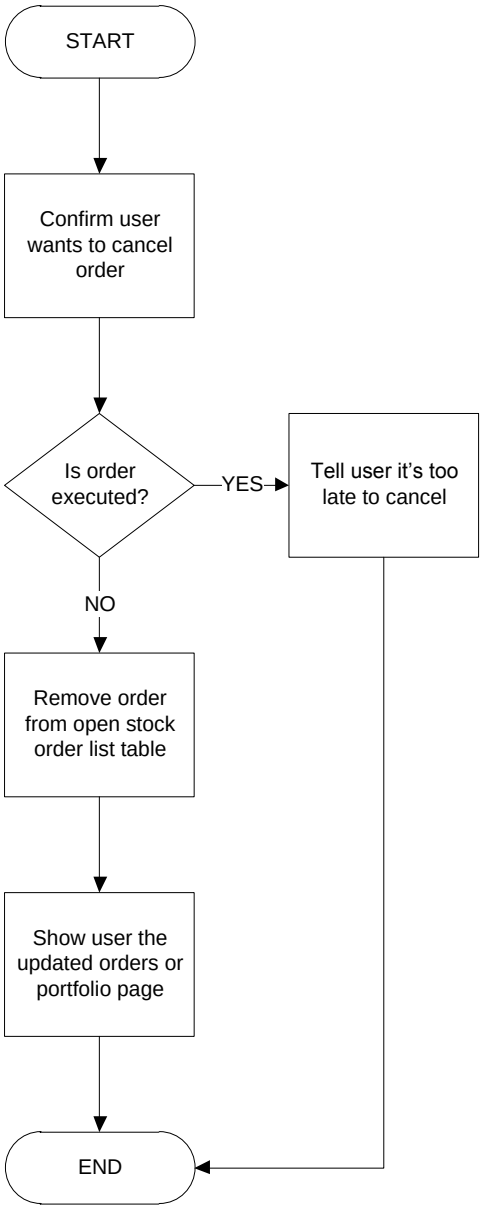
View Portfolio



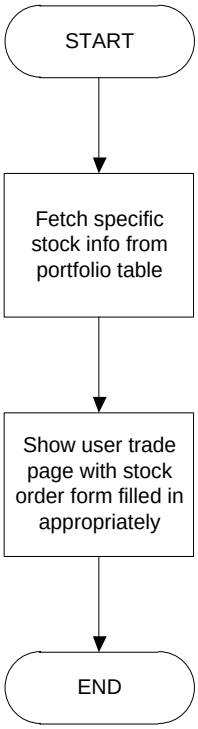
View History



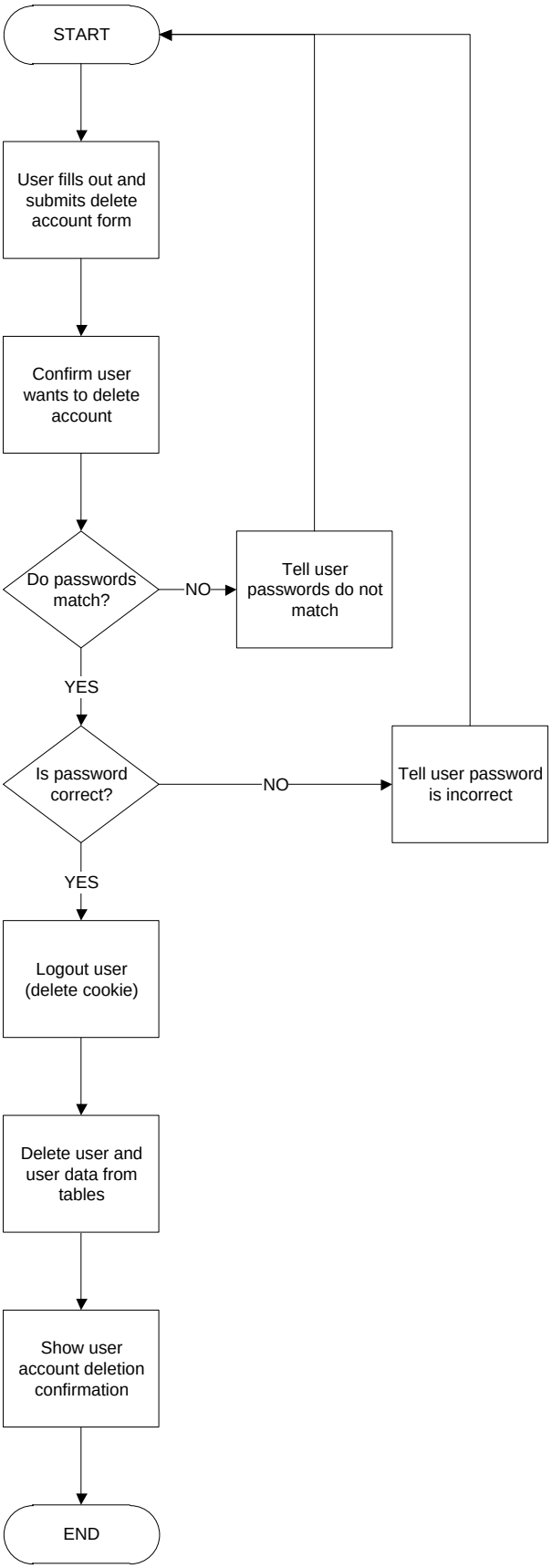
Cancel Order LNK



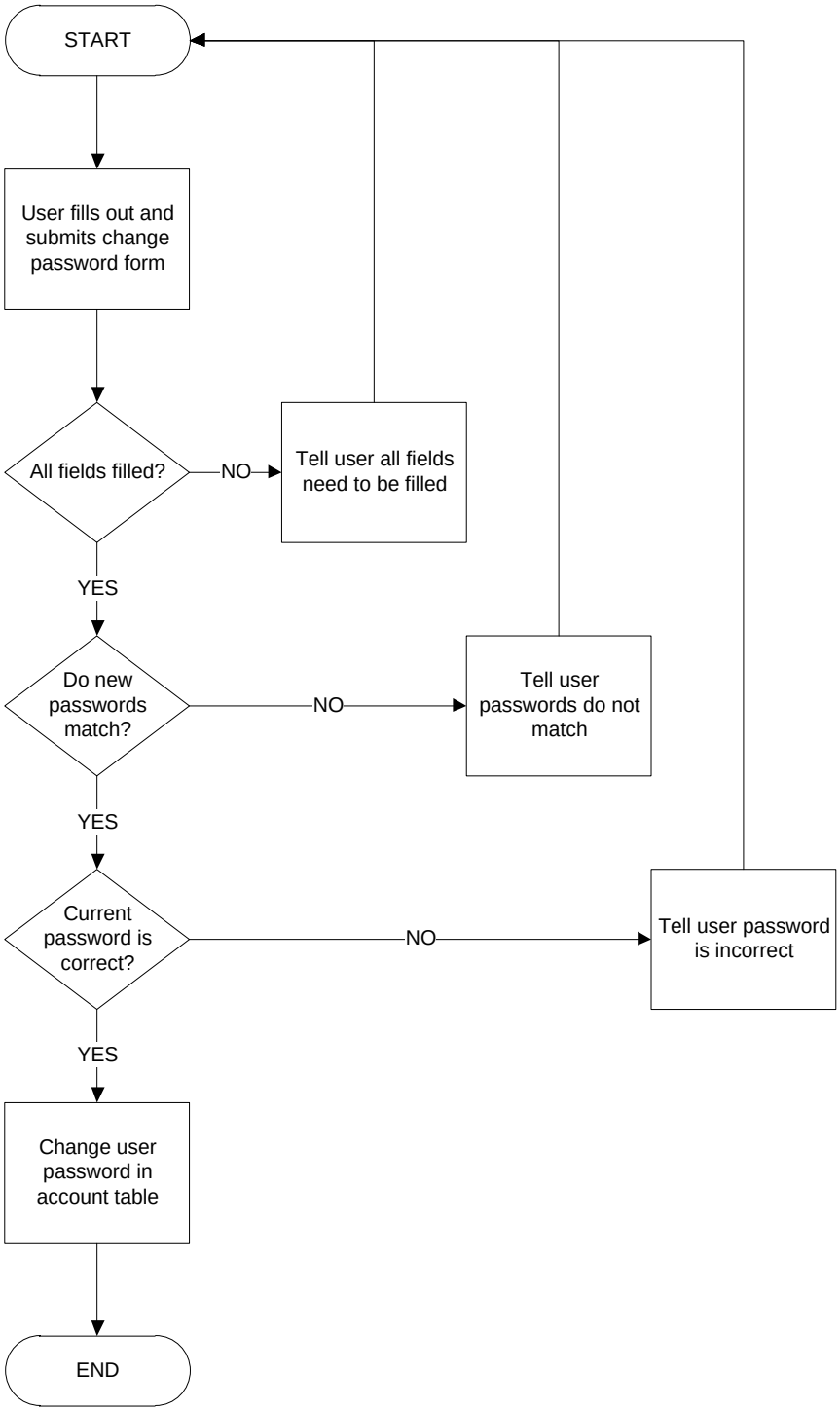
Sell Stock LNK



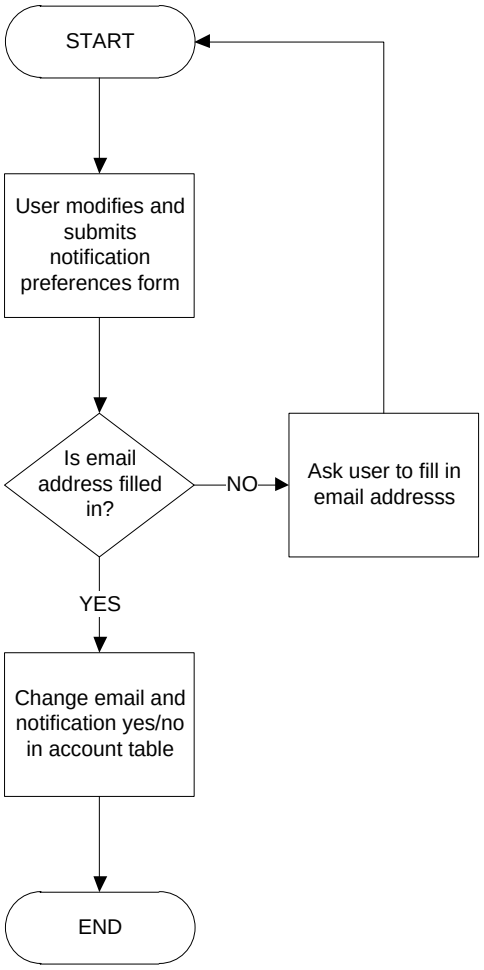
Delete Account



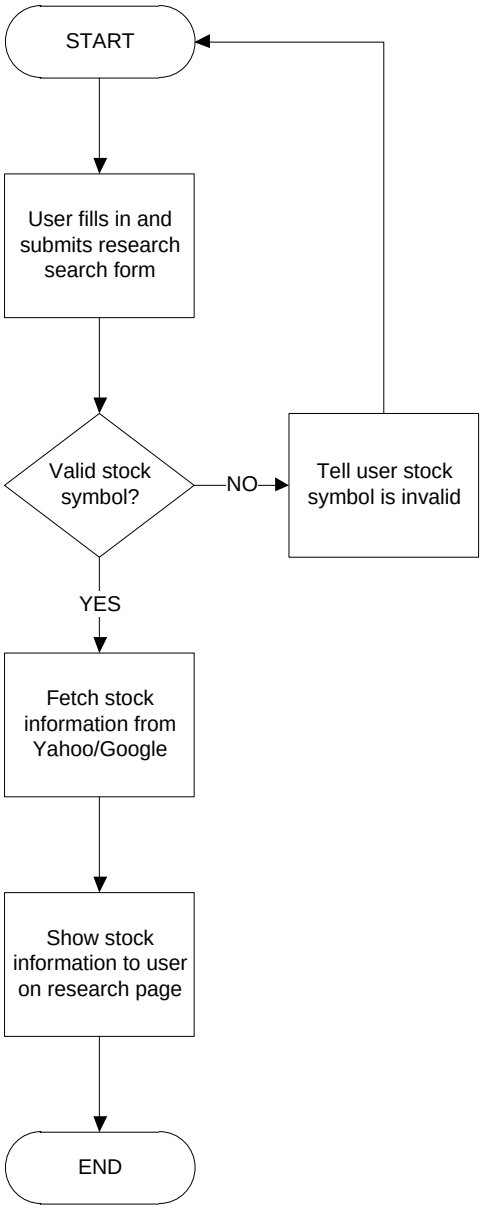
Change Password



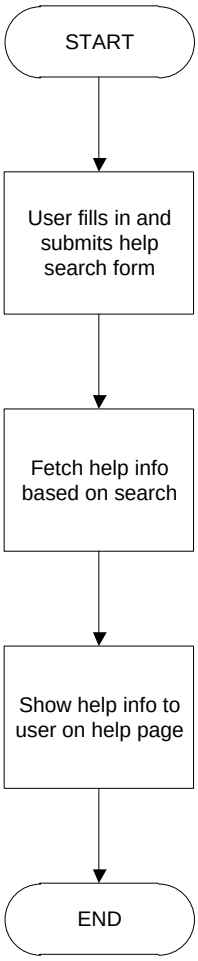
Notification Changes



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