



StockOverflow

Members:

Ambrish Parekh: parekha@usc.edu

Gwenyth Portillo-Wightman: gportill@usc.edu

Tiancheng (Sean) Gao: tiancheg@usc.edu

Hridee Patel: hrideepa@usc.edu

Jiho Kim: jihok@usc.edu

Prayaas Aggarwal: prayaasa@usc.edu

Table of Contents

Project Proposal.....	Page 3
Meeting Times.....	Page 3
Project Overview.....	Page 3
High-Level Requirements.....	Page 4
Specifications.....	Page 4
Changes.....	Page 5
Technical Specifications.....	Page 6
Web Interface.....	Page 6
Sign In (Landing Page).....	Page 6
Register New User.....	Page 6
Home.....	Page 6
Stocks.....	Page 7
Rankings.....	Page 7
Login/Register Tab.....	Page 7
My Money (Logged In Functionality)	Page 7
Database.....	Page 8
Stock/Profit Calculation.....	Page 9
Ranking.....	Page 9
Stock Selling/Purchasing.....	Page 9
Changes.....	Page 10
Detailed Design.....	Page 11
Web Interface.....	Page 11
Log-In Page.....	Page 11
Register Page.....	Page 12
Home Page.....	Page 13
Rankings Page.....	Page 14
Stocks Page.....	Page 15
User Profile Page.....	Page 16
Bank Class.....	Page 17
User Class.....	Page 17
Database.....	Page 17
User Table.....	Page 18
Investment Table.....	Page 19
Updating Investments.....	Page 19
Stocks Table.....	Page 20
Rankings Table.....	Page 20

Searching.....	Page 20
Stock/Profit Calculation.....	Page 21
Ranking.....	Page 21
Stock Selling/Purchasing.....	Page 21
Data.....	Page 23
Graphs.....	Page 23
Design.....	Page 23
Multithreading and Networking.....	Page 23
Data Structures.....	Page 23
Tools and Software Used.....	Page 24
Changes.....	Page 24
Testing Document.....	Page 25
Black-Box Testing.....	Page 25
Login Page.....	Page 25
Register Page.....	Page 26
Home Page.....	Page 28
Rankings Page.....	Page 30
Stocks Page.....	Page 30
User Profile Page.....	Page 31
White-Box Testing.....	Page 34
Deployment Document.....	Page 38
Requirements.....	Page 38
Deployment Steps.....	Page 38

Project Proposal

Meeting Times

Group Work Time: **Tuesdays 8:00 PM - 10:00 PM**

Meeting Location: **Lorenzo**

CP Meeting Time: **10:00 PM - 10:30 PM**

CP Meeting Location: **Lorenzo**

Project Overview

The project will be a web application that allows users to track changes in the stock market. More specifically, users will be able to chart the real-time prices of the stocks they select on the same graph. The web application will also be a simulation game, in which users can invest fake money in stocks. Each user will then be ranked against all the other users based on how much total money each user has, creating a leaderboard. To allow the users to customize which stocks they want to invest in and view on the graph, there will be a page where users can see live prices of every stock that is available in the database, along with some extra information about each stock.

The GUI aspect of the web application will come from HTML/CSS/Javascript. Multithreading will be used to update the chart of live stock prices, as well as to assist in the addition and removal of stocks from the graph. Networking will be used to make calls to the stock API to retrieve the live price update data, and this will allow multiple users to simultaneously access the data as well.

High-Level Documentation

Specifications

When users first enter the site, they will be sent to a landing page where they can enter their credentials to log in, register as a new user, or continue as a guest.

If a user continues as a guest, they will enter the main page, which will have a navigation bar at the top with multiple tabs: Home, Stocks, and Rankings, and Login/Register. The Home tab (open by default) will display a graph with some default stock(s). Users will see their stock prices updated live on the same graph. In the Stocks tab, users can view a list of all stocks available on the market and the live, updated prices of the stocks. When a user clicks a button next to the name and price of specific stock, a small blurb will appear and show the stock's information. Users can use the search bar at the top of the Stocks page to look up a specific stock. The third tab will lead users to a Rankings page. This page will show a numbered list of the top fifty registered users who have made the greatest profits from their hypothetical investments in the application (see the description of logged in users below for more detail on the process of investing). In one column, the names of the top ranking users will be displayed, and, in another, the profits of each user. By clicking the fourth tab, Login/Register, the user will be taken to a page which allows them to register as a new user.

If a user chooses to register on the landing page, they will be directed to another page that asks them to enter some basic profile information, along with a username and password. Once they have submitted their account information, they will be logged in and arrive at the Home tab. The graph on the main home page will display the default stock(s). Registered users will have the ability to select which stocks to view on the graph, and their stock preferences will be remembered across sessions. Additionally, the newly-created user's profile will be initialized with a default amount of pretend money that they can use to begin investing. The money used is not tied to any real world money as this part of the site is only a simulation of real investing. Logged in users will see the Stocks and Rankings tabs just as guest users do, but now the new user will also be entered into the rankings database, like all registered users. Because the user is validly logged in, a new My Money tab will be enabled next to the Home, Stocks, and Rankings tabs in the navigation bar, replacing the Login/Register tab that guest users see. The My Money tab will display the user's ranking, the amount of money the user has available for investing, and an interface for investing and selling stocks. In the interface, the user will see a list of stocks they've chosen to follow on the Stocks page, and there will be two buttons next to each stock to Buy or Sell. To buy a stock, the user will enter how

many shares they want and click the Buy button, which will validate their purchase and update their balance accordingly. To sell their stocks, the user will similarly update how many shares they want to sell. The application will then determine their new balance based on the profit or loss from the stock they have sold.

If a user chooses to log in with an already existing account, they will not be greeted by the default stock on the graph that guest users and newly registered users see when they reach the main home tab. Rather, the graph will display the stocks they were viewing from their previous session. The rest of the experience for logged in users will be similar to that of newly registered users.

Changes

All pages:

- Guest users see an additional, fourth tab called Login/Register. Upon clicking this tab, users are redirected to a page where they can register as a new user.

Home page:

- Users are not able to follow or unfollow stocks on the Home page; they must do this on the Stocks page.
- Users are not able to search for stocks on the Home page; they must do this on the stocks page.

Stocks page:

- The information blurbs are not available by clicking on the name of the stock but rather a button beside the name and price of the stock.

Profile page:

- This page is now called the My Money page.
- Users do not see the amount of money they have made in profits/losses from each stock they've invested in, just the balance of their bank.
- There is not a search bar on the My Money page. Users can find stocks they want to invest in by searching and following stocks on the Stocks page.

Technical Specifications Document

Web Interface

Sign In (Landing Page): 4 hours

- The page will have a form with a username field, a password field, and a submit button.
- Upon verification, users will be directed to Home Page. If the entered information does not match the records, the application will display an appropriate error message and allow the user to attempt to sign in again.
- Two links below the sign in form will allow users to continue to the Home page as Guest or register as a new user by redirecting them to the Register New User page.

Register New User: 4 hours

- The page will have a form that prompts users for a username, a password, and a confirm password. There will be a submit button to create the new account.
- If the username already exists in the database of users, the new user will be asked to select a different username. The application will perform validation checks to ensure that the username and password are acceptable.
- After the user has successfully created an account, they will be logged in by default and directed to the Home Page.

Home: 6 hours

- The page will display a navigation bar that lists the available pages in the application: Home, Stocks, and Rankings. Guest users will see a Login/Register tab. Logged in users will also see the My Money page in the navigation bar.
- Clicking on the name of a page in the navigation bar will redirect the user to the requested page.
- The Home page will contain a graph that displays multiple stocks within the same view and provides live updates about the prices of the stocks. There will also be a graph that displays historical data about the stocks.
- A list of the stocks currently displayed on the graph will be available by scrolling over the lines on the graphs.
- Users will start off with a default set of stocks that they can modify by adding and removing stocks to display on the Stocks page. The user's preferences

will be saved in the database and will roam across sessions for logged-in users.

- Multithreading will be used to concurrently update the graph with live data of stock prices and accommodate the concurrent addition and removal of stocks from graph based on user interaction with the list of displayed stocks next to graph.

Stocks: 4 hours

- The page will have the same navigation bar as the Homepage, with links to navigate to Home, Stocks, Rankings, and (for guest users) Login/Register or (for logged in users) My Money.
- The page will provide a list of all of the stocks available in the database, split across pages so that only 25 results appear per page.
- When a user clicks on the name of a stock, a blurb will appear, displaying information about the stock, including its name, its current price, and how much its stock value has changed.
- The page will provide a search bar that allows users to search for a specific stock in the database of stocks.
- Upon submitting a search term in the search bar, the application will look for matches by text and display information for the specific stock that the user requested.

Rankings: 4 hours

- The page will have the navigation bar with links to navigate to Home, Stocks, Rankings, and Login/Register (for guest users) or My Money (for logged in users).
- The page will display a numbered list of the fifty top-ranked users who have made the greatest profits from their investments and current balance within the application.
- The list will display the name of the user and their total money.
- The list will update at the end of the stock market day.

Login/Register tab: 10 minutes

- This tab will simply redirect users to the page where they can register as a new user.

My Money (Logged In Functionality): 7 hours

- This page will be available only to logged in users.

- The page will have the navigation bar, with links to navigate to Home, Stocks, Rankings, and My Money.
- The user's name will be displayed along with information about how much money is available for them to invest, the amount of profit or loss acquired from the stocks they've invested in, and a list of their current investments.
- There will be an interface on this page for investing and selling stocks (See "Stock Selling/Purchasing" below for details).

Database: 5 Hours

- The User table will be used for validating users. It will consist of userID, username, password, first name, last name, and balance, which is the amount of virtual money the user has in the bank.
- Buying stocks will decrease the value of the user's balance, while selling stocks will increase their balance according to the number of stocks and each of their prices.
- When a user signs up, a new row in User table will be created with the information that the user provided on the Register New User page. The value of balance of the row will be initialized to a fixed, default amount.
- The Investment table will store the information about all of the users' investments. It will consist of investmentID, userID, stockID (a unique ID for each stock that can be used to get the stock's information from a stocks API), amount (the number of stocks that the user has right now), and price (the price of the stock at the time of investment).
- The Investment table will be used to calculate how much money a user has in stocks at a particular time. It will also be used to calculate the amount of money user gained or lost since the time of the investment.
- Both the User table and the Investment table will be used to determine the rankings of users in the Rankings tab. Users will be displayed in descending order based on the sum of user's money in bank and their holdings in stocks, so the ranking algorithm will query the two tables to get this value.
- The Rankings table will store the rankings of users to be displayed in the Rankings tab. This table will be updated at the end of the market day (See "Rankings" below).
- Networking will be used whenever a user wants to access the database of stock information, since multiple users may want to retrieve data at the same time.

Stock/Profit Calculation: 2 Hours

- Information from Investment Table in database will be used to make spontaneous calculations of the user's profit/loss of money from their investments. The calculations will be made on the server when the My Money page is requested.
- A call to a stocks API will give current information for the prices of each of the stocks that the user has invested in. There will be a function to calculate the gain or loss of money based on the amount that the user has invested in each stock and the price of that stock at time that the profit is calculated.

Ranking: 2 Hours

- Rankings will be based off of the sum of each user's money in bank and their money in stocks.
- A list on the page will display the users in order of greatest total money made, displaying the name of each user and their total money.
- Rankings will be updated at the end of the market day.

Stock Selling/Purchasing

- The buying and selling of stocks will take place through an interface on the My Money page.

Selling: 2 Hours

- To allow for the selling functionality, users will see a list of stocks they currently hold investments in.
- Each stock in this list will have a field next to it where users can enter how many shares of the stock they would like to sell.
- When the user clicks the submit button next to the specific stock, the amount they would like to sell will be validated against their current investment in the stock.
- If the amount the user would like to sell is within the bounds of their current investment, the sale will go through, and the sale price will be added to the total sum of money that they have available for investing in the database.
- The user's list of investments on the My Money page will be updated to account for the sale of their investment.
- If the amount the user would like to sell is not within the bounds of their current investment, an error message will be displayed, and the user can attempt to make another sale by using the same field and submit button.

Purchasing: 2 Hours

- To allow for the purchasing functionality, users will look at the list of stocks they are following on the My Money page. Next to each of these stocks will be a field where the user enters how many shares they want to sell, and they can click the button next to the field to sell those shares.
- Clicking the Sell button will verify that the user has enough money to make the investment.
- If validation is successful, the purchase will go through, and the user's new investment will be visible on the My Money page.
- The user's investments will be updated in the database, and their balance will be adjusted as well.
- If validation is unsuccessful, an error message will be displayed and the user can attempt to make another purchase by using the search bar and the same form.

Changes

All pages:

- Guest users will see a Login/Register tab in place of the My Money tab that logged in users see.
- When clicked on, the Login/Register tab will redirect users to a page where they can register as a new user.

Home page:

- Instead of just one graph that displays recent data about the stocks, there is also a graph of historical data. The recent data is updated live each minute.
- Instead of the list of stocks on the graph being displayed next to the graph, the list is available by scrolling over the lines on the graphs.
- There is no button to remove stocks from the graph on the Home page. Following and unfollowing stocks is done on the Stocks page.
- There is no search bar on the Home page.

Stocks page:

- The list of stocks is split across pages so only up to 25 stocks appear on the page at a time.
- The prices of stocks update live on the Stocks page.

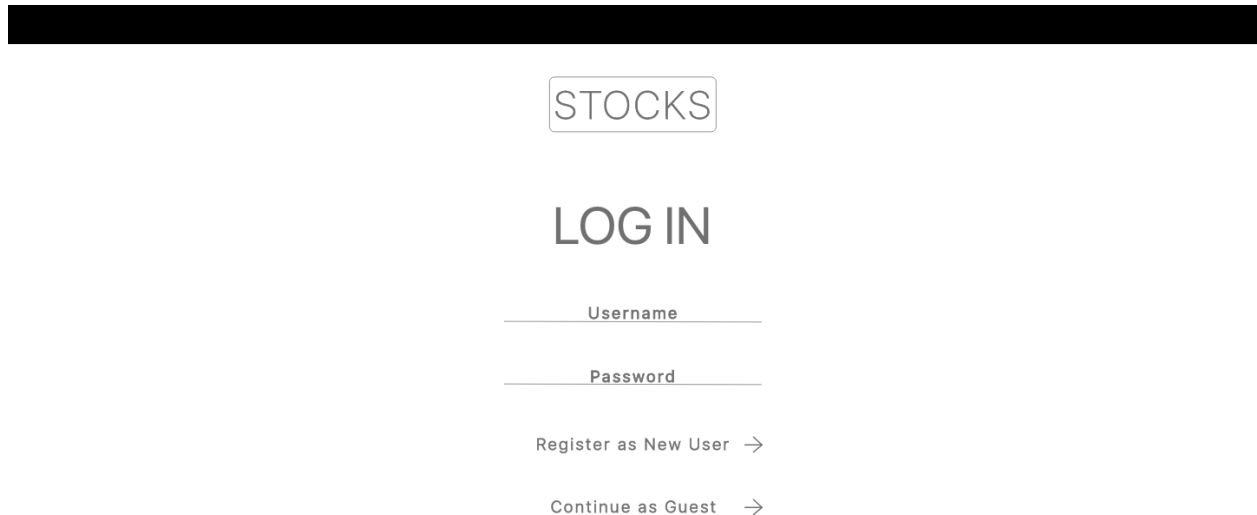
Buying/Selling interface:

- Users can only buy and sell stocks if they are already following those stocks. New stocks can be followed on the Stocks page.

Detailed Design Document

Web Interface

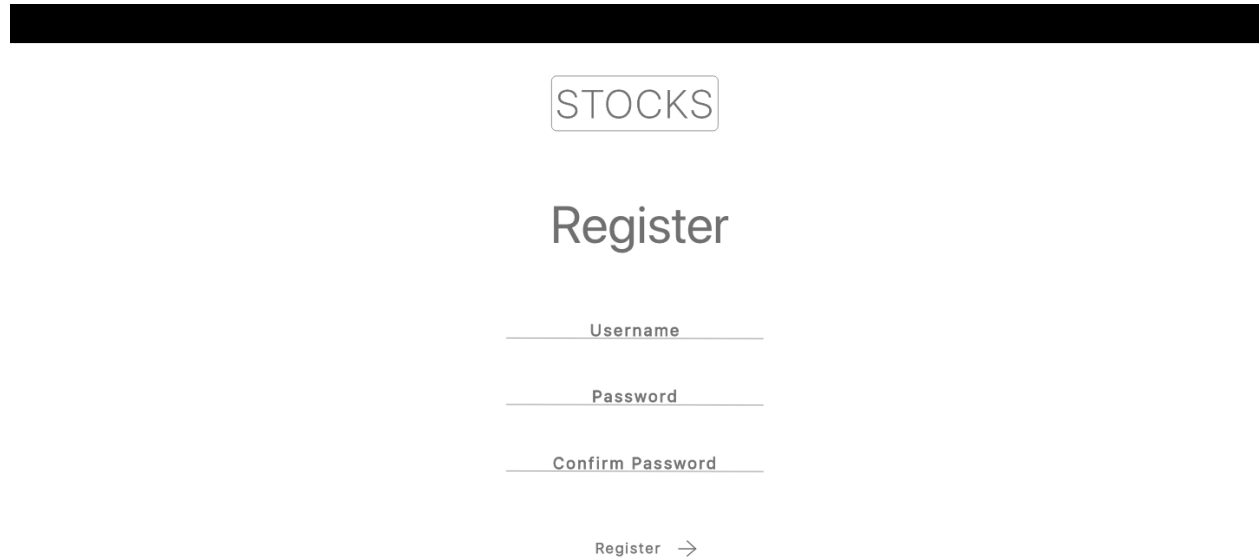
Log-in page:



The mockup shows a login page for an application titled 'STOCKS'. At the top, the word 'STOCKS' is enclosed in a rounded rectangular box. Below this, the text 'LOG IN' is centered in a large, bold, sans-serif font. Underneath 'LOG IN', there are two input fields: the first is labeled 'Username' and the second is labeled 'Password', both in a smaller, lighter font. Below the password field, there are two links: 'Register as New User' followed by a right-pointing arrow, and 'Continue as Guest' followed by a right-pointing arrow. The entire form is centered on a white background.

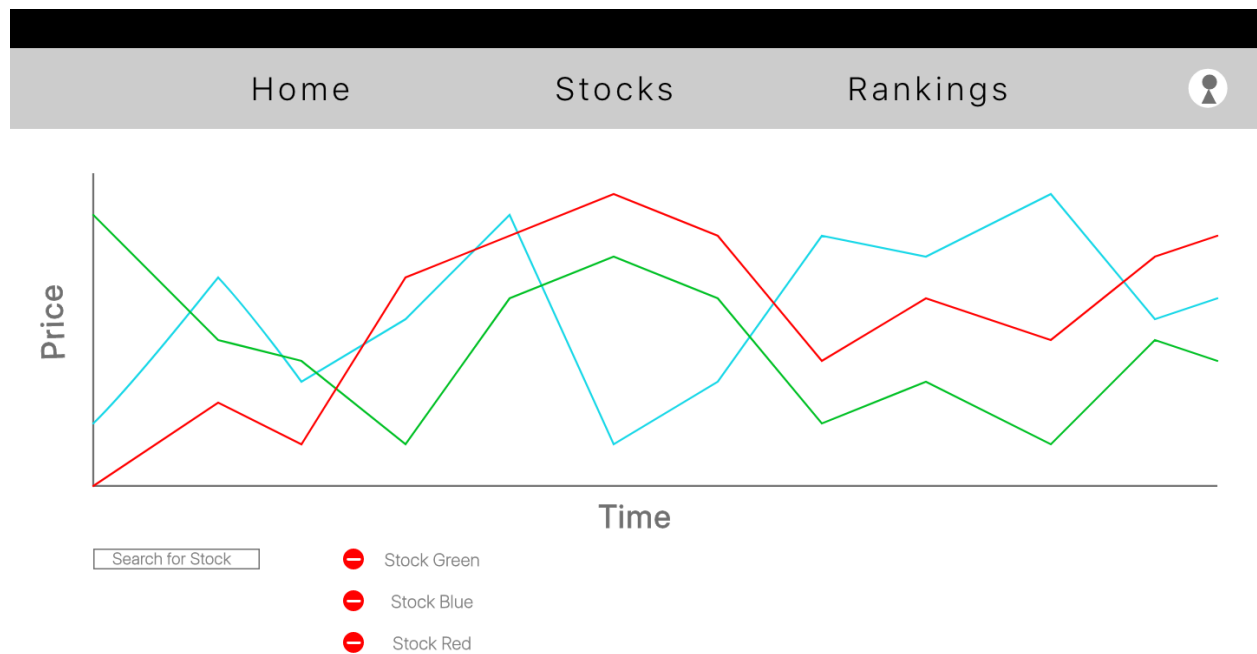
- The page will have a form with a username field, a password field, and a submit button.
- Upon verification, users will be directed to the Home Page. If the information entered does not match records from the database, the application will display an appropriate error message and allow the user to attempt to sign in again.
- Two links below the sign in form will allow users to continue to the Home page as a Guest user or register as a new user by redirecting them to the Register New User page.

Register page:

A mockup of a registration page. At the top, there is a black horizontal bar. Below it, the word "STOCKS" is centered in a light gray box. The word "Register" is centered in a large, bold, dark gray font. Below "Register" are three input fields, each with a light gray border and a dark gray label: "Username", "Password", and "Confirm Password". At the bottom, there is a "Register" button with a right-pointing arrow.


- The page will have a form that prompts users for a username, a password, and a confirm password. They can click the Submit button underneath create the new account.
- If the username already exists in the database of users, the new user will be asked to select a different username. The application will perform validation checks to ensure that the username and password are acceptable.
- After the user has successfully created an account, they will be logged in by default and directed to the Home Page.

Home page:



- The page will display a navigation bar that lists the available pages in the application: Home, Stocks, and Rankings. Logged in users will also see the My Money page in the navigation bar. Guest users will see a Login/Register tab.
- Clicking on the name of a page in the navigation bar will redirect the user to the requested page.
- The Home page will contain two graphs that displays multiple stocks within the same view.
- The first graph, displayed above the second, will display the stocks and their prices with by-the-minute price updates. Each minute, the graph will load an additional data point, corresponding to the price of the stock in the previous minute.
- The second graph, displayed below the first, will show the historical data for the stocks, up to 5 years from the current time.
- The user can see what stocks are being displayed by mousing over the lines on the graph.
- Multithreading will allow the application to concurrently update the graph with live data of stock prices and accommodate the concurrent addition and removal of stocks from graph based on user interaction with the list of displayed stocks next to graph.

Rankings page:

Home	Stocks	Rankings	
------	--------	----------	---

Rankings

Rank	Username	Total Earnings
1	ambro	\$999999999
2	Captainhoji	\$9001
3	gwen	\$7777
4	damsel in distress	\$6996
5	notHotdog	\$5000

- The page will have the navigation bar with links to navigate to Home, Stocks, Rankings, and (for guest users) Login/Register or (for logged in users) My Money.
- The page will display a numbered list of the fifty top-ranked users who have made the greatest profits from their investments within the application.
- The list will display the name of the user and their total money.

Stocks page:

HomeStocksRankings

Search

Symbol	Name	Price	%change
AAPL	Apple Inc.	216.30	-1.59%
GOOG	Alphabet Inc Class C	1,071.47	-2.20%
FB	Facebook, Inc. Common Stock	145.37	-3.70%
SNAP	Snap Inc	6.28	-10.16%
TWTR	Twitter Inc	32.36	+1.76%
TSLA	Tesla Inc	330.90	+5.09%
AMZN	Amazon.com, Inc.	1,642.81	-7.82%

HomeStocksRankings

Search

Symbol
Company Name

Price of StockPrice ChangeAvg. PriceDividends


CEO:

Type:

Description:

- The page will have the same navigation bar as the Homepage, with links to navigate to Home, Stocks, Rankings, and (for logged in users) My Money.
- The page will provide a list of all of the stocks available in the database.
- When a user clicks on the button next to the name and price of a stock, a blurb will appear, displaying information about the stock, including its name, its price, brief description about the company, its CEO, and how much its stock value has changed.
- The page will provide a search bar that allows users to search for a specific stock in the database of stocks.
- Upon submitting a search term in the search bar, the application will look for matches by text and display information for the specific stock that the user requested.

User profile page:







Home
Stocks
Rankings



Username's Profile

Money in Bank: \$8913.63

Money in Stocks: \$8969.45

My Stocks

Symbol	Name	Current Price	# owned				
AAPL	Apple Inc.	216.30	7	0 	Buy	0 	Sell
NFLX	Netflix, Inc.	299.83	5	0 	Buy	0 	Sell
TSLA	Tesla Inc	330.90	18	0 	Buy	0 	Sell

 Follow New Stock

- This page will be available only to logged in users.
- The page will have the navigation bar, with links to navigate to Home, Stocks, Rankings, and My Money.
- The user's name will be displayed along with information about how much money is available for them to invest and a list of their current investments.
- There will be an interface on this page for investing and selling stocks (See "Stock Selling/Purchasing" below for details).

Bank	User	
-balance: int -profit: int -stocks: Set<String>	-bank: Bank -username: String -password: String	
+getBalance(): int +getProfit(): int +setBalance(): void +setProfit(): void	+getUsername(): String +getPassword(): String +setUsername(): void +setPassword(): void +updateAccount(): void	Calculation
		+calculateProfit(int, double): void

Bank Class

- The Bank class stores an integer representing the remaining amount of money that the user has left to invest and an integer that represents the profit that the user has made, calculated with the calculateProfit() method in the Calculation class.
- The Bank will store a set of Strings that are the stock symbols that the user has invested.
- There will be a public getters and setters for balance and profit.

User Class

- The user class will have username, hashed password, and a Bank object which will allow the application to access the user's profit and balance using the Bank getters and setters.
- The getters and setters in the User class will be used in combination with validateUsernameChange() and validatePasswordChange() in the Validation class (See below for detail).

Database

- We will use MySQL for the database.
- Most queries and operations on the tables will use PHP and JDBC.

User

userID	username	password	balance

Investment

investmentID	userID	stockSymbol	amount

User table:

- userID is the primary key for the table and is auto-incremented.
- username is the unique username that the user created when they registered for the application.
- password is the hashed password (hashed according to SHA1) that was created based on the user's selected password when they registered for the application.
- balance is the amount of money that the user is the bank at a given time, which will be updated at the end of each day using the getBalance() function in the Bank object corresponding to each user and a PHP call that will update the balance in the database. The time scheduler package in node.js will update the data at the same time each day.

Validation	LoginServlet	RegisterServlet
+validateUser(): boolean +validateUsernameChange(): boolean +validatePasswordChange(): boolean	+validate(): boolean -query(String): ResultSet	+validate(): boolean -query(String): ResultSet

- The Validation class will be used in the LoginServlet in order to verify that a user exists in the database.
 - validateUser() will take a String password as parameters. The function will make a call to the Login Servlet, passing the password to the backend. The Login Servlet will then connect to the database and search for a matching entry using an instance of the Validation class. in the User table that contains this username and hashed password. If it finds a match, it will return true. Otherwise, it will return false.

- validateUsernameChange() will assist in letting a user change their username. It will make a call to the UpdateAccount Servlet, where the original username will be found in the database with a JDBC call and updated to the new, desired username. As long as the new username is not already in use by another user, the function will return true. Otherwise, it will return false.
- validatePasswordChange() will assist in letting a user change their password. It will first hash the new password and then make a call to the UpdateAccount Servlet, where the user will be found in the database with a JDBC call, and their old hashed password entry will be replaced with the new one.
- When a new user would like to register for the application, the RegisterServlet will be used. The Validation class will make sure that the username does not already exist in the database, using the validateUser() method. If validateUser() returns false, then the new user can successfully be registered. The information that the user fills out on the Register page (username and password) will be passed to the servlet. The password will be hashed within the servlet, and a JDBC call will insert a new entry into the User table with the parameters being the desired username, hashed password, and default initial balance of \$1,000.

Investment table:

- investmentID is the primary key for the table and is auto-incremented.
- userID is a reference to the userID variable of the User table, connecting an investment with the user who made the investment.
- stockSymbol is the symbol for that stock that the user has chosen to invest in.
- amount is the number of stocks that the user holds for a specific stock.
- The methods addInvestment() and sellInvestment() will update the rows by adding new rows to the table when a user makes a purchase, removing rows if the user chooses to sell the full amount of stocks that they own for a particular investment, or otherwise updating the amount of stocks that a user owns in a particular investment. See "Stock Purchasing/Selling" below for more detail.

Updating Investments:

- When a user invests in specified number of stocks using the Buy interface form, the addInvestment() method will be used to modify the investment table using PHP. If the row containing user's id and the stock's symbol does not exist, new row will be created. The addInvestment() takes an integer

and a string as parameters. They are the number of stocks the user would like to purchase, which will be added to the investment table under the “amount” column, and the symbol of that stock. If the user has already invested in the desired stock in the past and would like to purchase more of the same stock, the amount in the row in the table corresponding to the investment will be incremented to match the additional number of stocks the user wants to buy.

- When a user chooses to sell an investment using the Sell interface, the `sellInvestment()` method will be used to update a row in the investment table using PHP. The `sellInvestment()` method will take an integer and a string parameter that the user specified within the Sell interface form corresponding to the number of stocks they want to sell and the symbol for that stock. When `sellInvestment()` is called with this parameter, the row of the table will be updated to subtract that many stocks from the row corresponding to the appropriate investment. If the amount of the stock for the user becomes zero, the row will be deleted.

Stocks Table:

- The stocks table will store all the information about each stock that the application supports. This table will be populated before the service is provided, by data from IEX Trading API (More on Data and Searching below).

Rankings Table:

- It will store the username and his or her total money. This table will be updated at every end of the market day (More on Ranking below).

Searching

- All information about stocks will be retrieved from the symbols reference data and company data of IEX Trading API and will be stored in the database beforehand for efficient searching.
- The search function will look for the company names and symbols that contain the search string. The wildcard characters will be used to query the database.
- The search result will change dynamically as user types in each letter for the search term. The browser will make asynchronous queries to the database using PHP as the user is typing.
- All the searching functions in the application (in Home tab, Stocks tab, and My Money tab) will use the same searching algorithm to guarantee consistency.

Stock/Profit Calculation

Calculation
+calculateProfit(int, double): void

- Information from Investment Table in database will be used to make calculations of the user's profit/loss of money from their investments. The calculations will be made on the server when the My Money page is requested.
- The Calculation class (in Java) will have a calculateProfit() method that calculates the total profit for each investment the user has made. The integer parameter is how many of the specific stock that the user owns, and the double is the current price of the stock, determined through a call to the IEX Trading API. Multiplying the values, the function will determine how much of a profit the user has made.

Ranking

- Rankings will be calculated once a day at the end of the market day using the Node.js time scheduler and will be stored to the Rankings table in the database. This will allow the application to find the top fifty users and list them on the Rankings page.
- Rankings will be based off of the sum of each user's money in bank and their money in stocks, calculated by accessing the User and Investment tables.
- A list on the page will display the users in order of greatest total money made, displaying the username of each user and their total money.
- Users will be displayed in descending order based on the sum of user's money in bank and their holdings in stocks, so the ranking algorithm will query the two tables to get this value.
- The Rankings table will store the rankings of users to be displayed in the Rankings tab with their username, name, and amount of money. This table will be referenced whenever a user requests for the Rankings page.

Stock Selling/Purchasing

- The buying and selling of stocks will take place through an interface on the My Money page.

- Users need to follow a stock before investing in it. The “follow a new stock” button will allow users to search for a new stock to follow and add it to “my stocks” list.
- Users can also unfollow a stock if the number of stocks the user currently has in that company is zero.

Selling:

- All stocks that user is holding will be displayed in the My Stocks list.
- Each stock in this list will have a field next to it where users can enter how much of the stock they would like to buy or sell.
- When the user clicks the sell button next to a specific stock, the amount they would like to sell will be validated against their current investment in the stock.
- If the amount the user would like to sell is within the bounds of their current investment, the sale will go through, and the sale price will be added to the total sum of money that they have available for investing in the database. The corresponding entry in the investment table will be modified accordingly.
- If the amount the user would like to sell is not within the bounds of their current investment, an error message will be displayed, and the user can attempt to make another sale by using the same field and submit button.

Purchasing:

- Users must follow a stock from the Stocks page before they can buy more shares of that stock.
- Users can enter the number of shares of that stock they would like to buy.
- Clicking the submit button will verify that the user has enough money to make the investment.
- If validation is successful, the purchase will go through, and the user's new investment will be visible on the My Money page.
- The user's investments will be updated in the database (either new entry will be created or existing entry will be modified), and their balance will be adjusted as well, according to the current price of the stock retrieved from the IEX Trading API.
- If validation is unsuccessful, an error message will be displayed and the user can attempt to make another purchase by using the search bar and the same form.

Data

- Data will come from the IEX Trading API and the Web sockets wrapper which uses Socket.io. These APIs provide updates about the price and standing of many stocks. The application will take the 1-minute interval data and work with JSONP to update the charts and buying/selling prices of the stocks.
- The Node.js time scheduler package will be used to update the data on the graph in 1-minute intervals.
- The information about every stock will be retrieved from the IEX API and will be stored in the database beforehand to increase searching efficiency. This data will be updated each day, when the market is closed.

Graphs

- The graph display on the Home page will be created using High Charts API.
- As mentioned in the Data section, the graph will be updated at set intervals of 1 minute using the Node.js time scheduler.
- The update function will query the IEX Trading API every minute on all the stocks that the user specified (maximum of 5 different stocks).

Design

- Wireframing: Adobe XD.
- Front-End: SASS framework, Flexbox, Bootstrap, and Anime.js.
- Logo: Adobe Photoshop, Adobe Illustrator.

Multithreading and Networking

- Each individual stock will be multithreaded, which will allow the stocks to simultaneously update on the graph.
- Additionally, multithreading and networking will be used to manage all concurrent users. Multiple users can log into the system at the same time and this will result in simultaneous queries. Multithreading and networking will make sure that these simultaneous database queries to the server result in the correct data to be retrieved from the database and API.

Data Structures

- Vectors are used as a thread-safe way to retrieve a list of users from the database using JDBC.

- Hashmaps are used to associate users with their list of investments and store usernames with their associated, hashed passwords.
- Heaps are used to determine the rankings of users, using a max heap that prioritizes users with a greater total money value.
- Sets are used to hold the list of valid stock symbols.
- Tries are used to aid with the searching functionality, where users can begin typing in the name of a stock symbol, and the search bar autofills with the name of stock symbols that begin with the letters that the user has typed in so far.

Tools and Software Used

- IDE: Eclipse, Visual Studio
- Text Editors: Atom Text, Notepad, Sublime
- Socket.io and Node.js

Changes

All pages:

- In the navigation bar of every page, guest users see a Login/Register tab that, if clicked, directs them to a page where they can create an account.

Home page:

- The Home page contains two graphs, one for recent data from the past hour and one for historical data up to five years ago.
- The Home page does not have the option to search for stocks or follow/unfollow stocks. Users can do this on the Stocks page.

Stocks page:

- The user must click a button next to the name and price of the stocks in order for the information blurb to appear.

My Money page:

- This page does not display the money in investments that the user has. It displays only their balance, which is the amount of money they have left to invest.

Graphs:

- Only the Highcharts library was used to create charts, not chart.js.

General changes to this document:

- Added Data Structures section.
- Added Tools and Software Used section.

Testing Document

Black-Box Testing

Login Page:

- For these test cases, assume the database contains these (username, password) pairs: (username1, PassWord!), (usernameTwo, p@ssWORD), (username333, MyPassWord@).
- Test case: The username and password given by the user are valid (They exist in the database, and the password exactly corresponds to the given username).
 - Input: Username - username1; Password - PassWord!
 - Expected Output: Valid
- Test case: The username and password fields are empty when the user submits the login form.
 - Input: Username - "", Password - ""
 - Expected Output: Invalid
- Test case: The username field is empty and password field is filled in when the user submits the login form.
 - Input: Username - "", Password - PassWord!
 - Expected Output: Invalid
- Test case: The username field is filled in and the password field is empty when the user submits the login form.
 - Input: Username - username1, Password - ""
 - Expected Output: Invalid
- Test case: The username given by the user is not a valid username because it does not exist in the database.
 - Input: Username - userDNExist
 - Expected Output: Invalid
- Test case: The username given by the user is valid (it exists in the database), but the password given is wrong.
 - Input: Username - username1; Password - PassWo#rd
 - Expected Output: Invalid
- Test case: The username is 21 characters long, exceeding the 20 character username limit.
 - Input: thisusername\$istoOlong
 - Expected Output: Invalid

- Test case: The password is 21 characters long, exceeding the 20 character password limit.
 - Input: thispasswordistoolong
 - Expected Output: Invalid

Register Page:

- *For these test cases, assume the database contains these (username, password) pairs: (username1, PassWord!), (usernameTwo, p@ssWORD), (username333, MyPAssWord@).*
- *Allowed special characters are @\$!%*?&.*
- Test case: The username is 7 characters long.
 - Input: Username - usernam; Password - PassWord!; Confirm Password - PassWord!
 - Expected Output: Invalid
- Test case: The username is 8 characters long.
 - Input: Username - username; Password - PassWord!; Confirm Password - PassWord!
 - Expected Output: Valid
- Test case: The username is 20 characters long.
 - Input: Username - thisisalongusernamee; Password - PassWord!; Confirm Password - PassWord!
 - Expected Output: Valid
- Test case: The username is 21 characters long.
 - Input: Username - thisisalongusernameee; Password - PassWord!; Confirm Password - PassWord!
 - Expected Output: Invalid
- Test case: The password is 7 characters long.
 - Input: Username - username1; Password - Pas\$Wor; Confirm Password - Pas\$Wor
 - Expected Output: Invalid
- Test case: The password is 8 characters long.
 - Input: Username - username1; Password - Pas\$Word; Confirm Password - Pas\$Word
 - Expected Output: Valid
- Test case: The password is 20 characters long.
 - Input: Username - username1; Password - PassWord!VeryLong%^&; Confirm Password - PassWord!VeryLong%^&
 - Expected Output: Valid
- Test case: The password is 21 characters long.

- Input: Username - username1; Password - PassWord!VeryLong%^&a;
Confirm Password - PassWord!VeryLong%^&a
 - Expected Output: Invalid
- Test case: The password contains 0 special characters and 0 uppercase letters.
 - Input: Username - username1; Password - password, Confirm Password - password
 - Expected Output: Invalid
- Test case: The password contains 0 special characters and 1 uppercase letter.
 - Input: Username - username1; Password - Password; Confirm Password - Password
 - Expected Output: Invalid
- Test case: The password contains 0 special characters and 2 uppercase letters.
 - Input: Username - username1; Password - PaSsword; Confirm Password - PaSsword
 - Expected Output: Invalid
- Test case: The password contains 1 special character and 0 uppercase letters.
 - Input: Username - username1; Password - passwo%d; Confirm Password - passwo%d
 - Expected Output: Invalid
- Test case: The password contains 2 special characters and 0 uppercase letters.
 - Input: Username - username1; Password - p\$%sword; Confirm Password - p\$%sword
 - Expected Output: Invalid
- Test case: The password contains 1 special character and 1 uppercase letter.
 - Input: Username - username1; Password - Newpassw!rd; Confirm Password - Newpassw!rd
 - Expected Output: Valid
- Test case: The password contains 1 special character and 2 uppercase letters.
 - Input: Username - username1; Password - newpassw!rd; Confirm Password - newpassw!rd
 - Expected Output: Valid
- Test case: The password contains 2 special characters and 1 uppercase letter.
 - Input: Username - username1; Password - Newp%ssw!rd; Confirm Password - Newp%ssw!rd
 - Expected Output: Valid
- Test case: The password contains 2 special characters and 2 uppercase letters.

- Input: Username - username1; Password - NewP%ssw!rd; Confirm Password - NewP%ssw!rd
 - Expected Output: Valid
- Test case: The username field is left empty when the form is submitted.
 - Input: Username - ""; Password - Newpassw!rd; Confirm Password - Newpassw!rd
 - Expected Output: Invalid
- Test case: The password field is left empty when the form is submitted.
 - Input: Username - username1; Password - ""; Confirm Password - ""
 - Expected Output: Invalid
- Test case: The confirm password field is left empty when the form is submitted.
 - Input: Username - username1; Password - NewP%ssw!rd; Confirm Password - ""
 - Expected Output: Invalid
- Test case: The content of the confirm password field does not exactly match the content of the password field when the form is submitted.
 - Input: Username - username1; Password - NewP%ssw!rd; Confirm Password - NewPassw!rd
 - Expected Output: Invalid
- Test case: The username given already exists in the database.
 - Input: Username - username1; Password - NewP%ssw!rd; Confirm Password - NewPassw!rd
 - Expected Output: Invalid

Home Page:

- Test case: For logged in users, upon login, the home page displays the stocks that the user was viewing in their last session.
 - Input: None (This should be an automatic behavior).
 - Expected Output: The stocks in the graph should match the stocks in the set<String> stocks member variable of the User object corresponding to the current user.
- Test case: For guest users, upon login, the home page displays the default set of stocks.
 - Input: None (This should be an automatic behavior).
 - Expected Output: The home page graph should display the four default stocks: AAPL, FB, MSFT, and GOOG. The buttons to add a new stock or remove a stock are not enabled.

- Test case: For newly registered users who use the application for the first time, the home page displays the default set of stocks but displays the options to customize the graph by adding and deleting stocks.
 - Input: None (This should be automatic behavior).
 - Expected Output: The home page graph should display the default three stocks: AAPL, FB, and GOOG. The buttons to add and delete stocks are enabled.
- For registered users who have used the application before, the home page displays the set of stocks that they were viewing in their last session and displays the options to customize the graph by adding and deleting stocks.
 - Input: None (This should be automatic behavior).
 - Expected Output: The home page displays the stocks that the user was viewing in their last session and has options to add and delete stocks.
- Test case: There are no stocks displayed in the graph, and the user tries to add a stock.
 - Input: The user uses the search bar to look up a valid stock symbol, like "AAPL" and clicks to add it.
 - Expected Output: A line is added to the graph for the selected stock, AAPL, and that stock is also now listed in the list of stocks beneath the graph.
- Test case: There are five stocks being displayed on the graph, and the user tries to add another.
 - Input: The user uses the search bar to look up a valid stock symbol, like "AMZN," which is not already on the graph.
 - Expected Output: Option to add new stocks is disabled.
- Test case: As the user types additional letters into the search bar, the search results are dynamically updated.
 - Input: User types "A," then "A," then "P," then "L."
 - Expected Output: Each new letter narrows down the stocks that are available. With the first A, the user now sees only stock symbols that begin with A. When they enter the second A, the user now sees stock symbols that begin with AA.
- Test case: The user searches for a stock symbol that does not exist.
 - Input: The user searches "XJK."
 - Expected Output: The search bar displays "No Results."
- Test case: The user clicks the minus button next to the name of the stock in the list.
 - Input: The user clicks the minus button next to the name of a stock.
 - Expected Output: The stock is removed from the graph and from the list of stocks beneath the graph.

Rankings Page:

- Test case: The Rankings page displays the fifty users with greatest total balances in decreasing order in accordance with the values in the database.
 - Input: None (This should be automatic behavior).
 - Expected Output: The Rankings Page displays the fifty top users.
- Test case: We artificially inflate one user's balance to see if the page updates properly.
 - Input: Modify balance for a specific user to move them up into the leaderboards.
 - Expected Output: The Rankings Page reloads or updates (within a minute) to include this user.

Stocks Page:

- Test case: The stocks table updates every minute to update the price and percentage in change columns of the table.
 - Input: None (This should be an automatic behavior).
 - Expected Output: After one minute elapses, the stocks should automatically update.
- Test case: As the user types additional letters into the search bar, the search results are dynamically updated.
 - Input: User types "A," then "A," then "P," then "L."
 - Expected Output: Each new letter narrows down the stocks that are available. With the first A, the user now sees only stock symbols that begin with A. When they enter the second A, the user now sees stock symbols that begin with AA.
- Test case: The user searches for a stock symbol that does not exist.
 - Input: The user searches "XJK."
 - Expected Output: The search bar displays "No Results."
- Test case: The user clicks on the stock symbol for a specific stock in the list of stocks.
 - Input: User clicks on the stock symbol for a specific stock in the list of stocks.
 - Expected Output: An information blurb will appear on the page and report the information for the stock corresponding to the stock symbol that the user clicked on.
- Test case: The user clicks on the name of a specific stock in the list of stocks.
 - Input: User clicks on the name of a specific stock in the list of stocks.

- Expected Output: An information blurb will appear on the page and report the information for the stock corresponding to the stock name that the user clicked on.
- Test case: The user clicks on the price for a specific stock in the list of stocks.
 - Input: User clicks on the price for a specific stock in the list of stocks.
 - Expected Output: An information blurb will appear on the page and report the information for the stock corresponding to the stock price that the user clicked on.
- Test case: The user clicks on the percentage change for a specific stock in the list of stocks.
 - Input: User clicks on the percentage change for a specific stock in the list of stocks.
 - Expected Output: An information blurb will appear on the page and report the information for the stock corresponding to the percentage change that the user clicked on.
- Test case: The user, already viewing the information blurb, clicks on the exit button within the information blurb.
 - Input: User clicks on the exit button.
 - Expected Output: The information blurb will disappear to reveal the normal stocks page, with the list of stocks.

User Profile Page:

- Test case: When a guest user uses the application, the option to access the User Profile is not available.
 - Input: None (This should be an automatic behavior).
 - Expected Output: The user profile button in the top right corner should not be available for guest users.
- Test case: The User Profile page displays the name, money in bank, money in stocks, and investments corresponding to the user who is currently logged in.
 - Input: None (This should be an automatic behavior).
 - Expected Output: The correct name, balances, and investments should be displayed on the User Profile page.
- Test case: The user has no investments currently.
 - Input: None.
 - Expected Output: The My Stocks table reports "No investments" and displays the option to "Follow New Stock."
- Test case: The user chooses to follow a new stock and enters a valid stock symbol into the search bar.
 - Input: The user searches for a valid stock symbol, like "AAPL."

- Expected Output: A new row is added at the bottom of the My Stocks table, and the "# owned" is initialized to zero. The database of investments is updated to include an additional row for this user, where the number owned is also initialized to zero.
- Test case: The user chooses to follow a new stock and enters an invalid stock symbol into the search bar.
 - Input: The user searches for an invalid stock symbol, like "XJK."
 - Expected Output: The search bar should display "No Results."
- Test case: When the user selects to "Follow New Stock," as they type additional letters into the search bar, the search results are dynamically updated.
 - Input: User types "A," then "A," then "P," then "L."
 - Expected Output: Each new letter narrows down the stocks that are available. With the first A, the user now sees only stock symbols that begin with A. When they enter the second A, the user now sees stock symbols that begin with AA.
- Test case: The user enters 0 into the Buy field for a specific stock and clicks Buy.
 - Input: User enters 0 into the Buy field.
 - Expected Output: The user's balance and "# owned" for that stock should remain unchanged.
- Test case: The user enters 0 into the Sell field for a specific stock and clicks Sell.
 - Input: User enters 0 into the Sell field.
 - Expected Output: The user's balance and "# owned" for that stock should remain unchanged.
- *For the next several test cases, assume the user has invested in AAPL and owns 4 total shares in AAPL. Assume they have \$200.00 and that the current price of an AAPL share is \$207.10.*
- Test case: The user enters a valid number of stocks to Buy. That is, the number of stocks the user selects to buy times the price of that stock is less than or equal to the amount of money they have in the bank.
 - Input: The user enters 2 into the Buy field next to AAPL.
 - Expected Output: The purchase is successful. The user's total money in bank (displayed on the page) is decreased by $2 * (\text{current price of an AAPL share})$, and their total money in stocks is increased by $2 * (\text{current price of an AAPL share})$. The "# owned" column for AAPL is incremented by 2. The row in the Investments table in the database is updated to 6, which is the previous balance of 4 plus two additional

shares that they just bought. The balance in the User table of the database is changed to the user's new money in bank value.

- Test case: The user enters an invalid number of stocks to Buy. That is, the number of stocks the user selects to buy times the price of that stock is greater than the amount of money they have in the bank.
 - Input: The user enters 1 into the buy field next to AAPL.
 - Expected Output: Invalid. The purchase will not go through because the user's current money in bank, \$200.00, is less than the cost of (# to buy) * (current price of AAPL) = \$ 207.10.
- Test case: The user enters a valid number (less than or equal to the number that they own of that stock) of stocks and clicks Sell.
 - Input: The user enters a 2 into the Sell field next to AAPL.
 - Expected Output: The purchase is successful. The user's total money in bank (displayed on the page) is increased by 2 * (current price of an AAPL share), and their total money in stocks is decreased by 2 * (current price of an AAPL share). The "# owned" column for AAPL is decreased by 2. The row in the Investments table in the database is updated to 2, which is the previous balance of 4 minus the two shares that they just sold. The balance in the User table of the database is changed to the user's new money in bank value.
- Test case: The user enters an invalid number (greater than the number that they own of that stock) of stocks and clicks to Sell.
 - Input: The user enters 5 into the Sell field next to AAPL.
 - Expected Output: Invalid. The purchase will not go through because the user's number of shares in AAPL, 4, is less than the number of shares the user wants to sell, 5.
- Test case: The user chooses to update their username.
 - The same test cases performed on the Register page above will be performed in order to make sure that the new username meets the length requirements and that the new username does not already exist in the database. If the `validateUsernameChange()` returns true, then the username will be updated in the database (See below for details). The change can be verified by getting the username of the User with the `getUsername()` function in the User class, and that should return a value that matches the new username.
- Test case: The user chooses to update their password.
 - The same test cases performed on the Register page above will be performed in order to make sure that the new password meets the length and character requirements. If `validatePasswordChange()`

returns false, then the username will be updated in the database (See below for details).

White Box Testing

- Test case: The User class getUsername() function returns the user's username. Call getUsername() on a User object.
 - Input: None.
 - Expected Output: The function should return the String username for this User object.
- Test case: The User class setUsername() function updates the User database appropriately. A new String username is passed into the function, and we check that the member variable username was updated properly by using getUsername().
 - Input: A String username.
 - Expected Output: The call to getUsername() should return the updated String username. The database should be updated by this point from the call to setUsername().
- Test case: The User class setPassword() function updates the User database appropriately. A new String password is passed into the function.
 - Input: A String password.
 - Expected Output: The database should be updated by this point from the call to setPassword().
- Test case: The Calculation class calculateProfit() function produces the correct result.
 - Input: The int numberOfStocks and double priceOfStock should be passed in. For example, we give the input number of Stocks = 4 and priceOfStock = 78.21.
 - Expected Output: For Stocks = 4 and priceOfStock = 78.21, the output should be 312.84.
- Test case: The Bank class getBalance() function returns the user's balance.
 - Input: None.
 - Expected Output: If the member variable balance is set to 654.23, the return value of getBalance() should be 654.23.
- Test case: The Bank class getProfit() function returns the user's profit.
 - Input: None.
 - Expected Output: If the member variable profit is set to 654.23, the return value of getProfit() should be 654.23.

- Test case: The Bank class setBalance() function updates the user's balance. We call setBalance(654.23) and then use getBalance() to see if the member variable balance was updated.
 - Input: A new double balance should be passed in.
 - Expected Output: The call to getBalance(654.23) should return 654.23.
- Test case: The Bank class setProfit() function updates the user's profit. We call setProfit(654.23) and then use getProfit() to see if the member variable balance was updated.
 - Input: A new double profit should be passed in.
 - Expected Output: the call to getProfit() should return 654.23.
- Test case: The Validation class validateUser() function correctly returns whether or not a user with that username and password exists in the database.
 - For these test cases, assume the database contains these (username, password) pairs: (username1, PassWord!), (usernameTwo, p@ssWORD), (username333, MyPAssWord@).
 - Test case 1: A valid username and password are given.
 - Input: (username1, PassWord!)
 - Expected Output: Valid
 - Test case 2: A valid username and invalid password are given.
 - Input: (username1, PassWord!!!)
 - Expected Output: Invalid
 - Test case 3: An invalid username is given.
 - Input: (username112312312, PassWord!)
 - Expected Output: Invalid
- Test case: The Validation class validateUsernameChange() function correctly returns whether or not the user has selected a username that meets the length requirements and does not already exist in the database.
 - Test case 1: The username is 7 characters long.
 - Input: Username - usernam; Password - PassWord!; Confirm Password - PassWord!
 - Expected Output: Invalid. The function will return false.
 - Test case 2: The username is 8 characters long.
 - Input: Username - username; Password - PassWord!; Confirm Password - PassWord!
 - Expected Output: Valid. The function will return true.
 - Test case 3: The username is 20 characters long.
 - Input: Username - thisisalongusernamee; Password - PassWord!; Confirm Password - PassWord!
 - Expected Output: Valid. The function will return true.

- Test case 4: The username is 21 characters long.
 - Input: Username - thisisalongusernameeee; Password - PassWord!; Confirm Password - PassWord!
 - Expected Output: Invalid. The function will return false.
- Test case 5: The username given already exists in the database. *For this test case, assume the (username, password) pair (username1, NewP%ssw!rd) already exists in the database.*
 - Input: Username - username1; Password - NewP%ssw!rd; Confirm Password - NewPassw!rd
 - Expected Output: Invalid. The function will return false.
- Test case 6: The new username desired is the empty string.
 - Input: The new username is "".
 - Expected Output: False.
- Test case 7: The new password desired is the empty string.
 - Input: The new password is "".
 - Expected Output: False.
- Test case: The Validation class validatePasswordChange() function correctly returns whether or not the user has selected a username that meets the length and character requirements.
 - The function will perform the same length and character validation that was done on the Register page above. When the register page password check outputted Invalid, the validatePasswordChange() function will return false. When the Register page password check outputted Valid, the validatePasswordChange() function will return true.
 - Additional test case: The new password desired is the empty string.
 - Input: The new password is "".
 - Expected Output: False.
- Test case: The LoginServlet validate() function correctly returns whether the values submitted in the username and password field of the Login page correspond to a user in the database.
 - For these test cases, assume the database contains these (username, password) pairs: (username1, PassWord!), (usernameTwo, p@ssWORD), (username333, MyPAssWord@).
 - Test case 1: A valid username and password are given.
 - Input: (username1, PassWord!)
 - Expected Output: Valid
 - Test case 2: A valid username and invalid password are given.
 - Input: (username1, PassWord!!!)
 - Expected Output: Invalid

- Test case 3: An invalid username is given.
 - Input: (username112312312, PassWord!)
 - Expected Output: Invalid
- Test case: The Register Servlet validate() function correctly returns whether the values submitted in the username, password, and confirm password field of the Register page are satisfactory.
 - The same length checks for the username that were performed on the Register page will be performed here. The same length and character checks for the password that were performed on the Register page will be performed here. The validate() function will return false for the invalid cases (when either username or password were found to be invalid), and it will return true for the valid cases (when both username and password were found to be valid).
 - Test case 1: The values of the password field and confirm password field do not match.
 - Input: Username - username1; Password - PassWord!; Confirm Password - PassWord!!
 - Expected Output: Invalid.

Deployment Document

Requirements

- A computer
- Access to a modern browser with HTML5 and CSS3 support

Deployment Steps

- Download FileZilla, or similar FTPS Client application.
- Connect to 303.itpwebdev.com server with specific user credentials on FTPS Client application. This step may differ for each user, depending on which PHP supported server they will be hosting to. It is suggested to use WordPress Hosting if you don't have access to the ITP 303 server.
- Upload all files to server in same configuration of filenames and folder names.
- Make sure the site behaves as expected. Using the tests in the Testing Document will help to determine successful deployment to the server.
- Once site behavior is validated, send out the URL to all users.