# Doubleclick
# Project Part 3: Design

University of Nevada Reno

Computer Science and Engineering

---

CS 425 - Software Engineering

Team 30: Colin Comstock, Eugene Eom, Loren Parvin, Nicholas Rinehart, Henry Shi

Dr. Sergiu Dascalu, Vinh Le, Devrin Lee, Mitchell Martinez, Yifan Zhang

External Advisor: Michael Isaac

November 17th, 2021

# 1. Table of Contents

# Doubleclick

---

## 2. Abstract:

Our project aims to simplify and streamline the investment process. Previous forms of investing software are built by individuals who understand economics and trading but not simplicity and UX design. Our project will be elegantly suggesting stocks to an individual in an intuitive way. The user will input parameters in the form of risk tolerance, geographic location, and much more; providing them with the right stock for them. The application will only suggest one potential trade at a time with some information about the stock. By interpreting previous data sets and other stock exchange news the program will be able to provide users with an informative and simple hands on experience.

---

## 3. Introduction:

With this project we aim to increase the accessibility of investing/trading by simplifying the user experience to an almost extreme degree. Instead of the classic approach taken by traditional investment firms where tons of information is thrown at the user with no regard for how it is presented, our application will show the user only what is necessary to them. The application will only be usable when the user is logged in, and their account must be connected to their ThinkorSwim account. Once logged in, the user will also be able to input their preferences for trading. These preferences include information about risk tolerance, market sector, industry practices, etc. In order to effectively streamline and tailor the process to each user, all of these factors will be considered when Doubleclick pushes a possible trade. All

of the information about the user will also be stored into a database, and our project will be using Firebase to provide such functionality. The user will also be able to request a new trade with the click of a button that will take into account their previously inputted preferences and suggest a similar trade option, and the previously suggested trade will be discarded. Our project also aims to have a brief summary of the stock being suggested, and a graph indicating recent trends. The stock information may be collected via the ThinkorSwim api or by other means such as web scraping. Our front end will be using Vue.js as our javascript framework. In essence, the main goals of Doubleclick are to provide a clean and user friendly experience, create an accessible way for newer investors to start trading, and smartly suggest trades for users to buy and sell through API.

Our team is slowly but surely making progress on this project. We have nearly everything mapped out about our system architecture and user interface design. Our github repository has also been created and we are all familiarizing ourselves with Vue.js, our javascript framework of choice. A developer account for TD Ameritrade with an app has been created for accessing the API. Firebase has been set up with a Realtime Database for information storage and access. We have not made any significant changes to the project requirements between this report and our previous report (Specification).

---

## 4. High Level and Medium Level Design:

The context diagram in Fig.1 shows the high level system components. The User Settings System controls the user accounts and their settings. The Security System deals account storage, authentication, and security. The Stock Database is the database containing relevant stock data. The Stock Display System controls the stock display, description, visual data/charts for a given stock. The User Transaction History System allows the user to view all the stock they have either traded throughout their account's history, or their current stock's and manage/sell them. The Unique Recommended Stock Generation

System generates pseudo random companies to invest into based on the user's settings and preferences. The Statistics system calculates the stock's value at time of purchase, quantity and displays the stock's history/visual data relating to the timespan the stock was owned for and possible stock's value at time of sales and quantity sold.

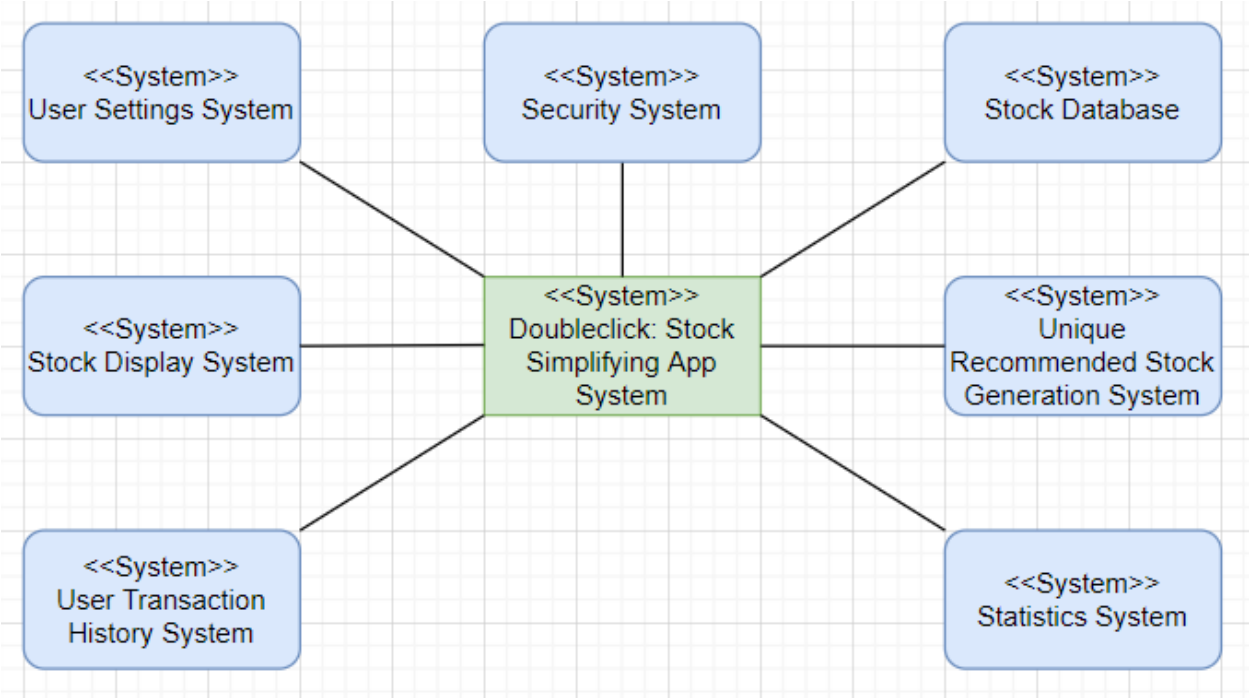


Figure 1. Context model of Doubleclick: Stock Simplifying App system.

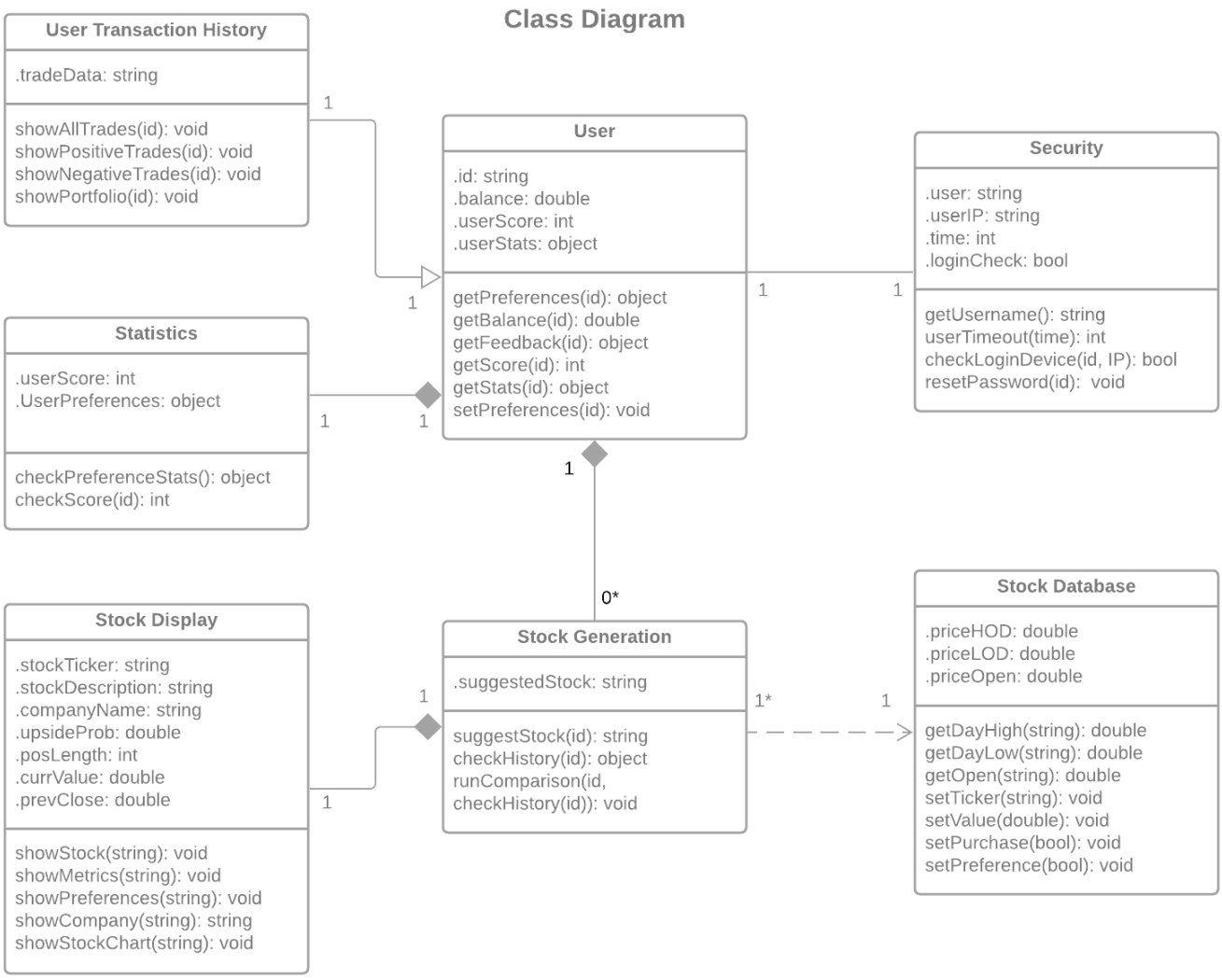


Figure 2: This class diagram shows the methods with relationships between classes.

Table 1: User class methods

| User | Description |
|---|---|
| getPreferences(id) | Passes in the User ID and returns an object with data associated with their preferences. If user preferences are not set on that user this method will return a null value. |
| getBalance(id) | Passes in the User ID and returns a double of the account balance associated with that user. This will return 0 if a valid ThinkorSwim account is not associated with that user. |
| getFeedback(id) | Passes in the User ID and returns an object with feedback about how user preferences have performed over time in their account. This method will |

| | |
|---|---|
| | return a null value if there have been no trades placed by this account. |
| getScore(id) | Passes in the User ID and calculates the score for that user compared to other users in the database. This method returns a value based on percentage gains/losses, not monetary value. This method will return a null value if there have been no trades placed by this account. |
| getStats(id) | Passes in the User ID and returns an object containing information about how previous trades have performed over time. It will return null if there have been no trades placed for that user. |
| setPreferences() | This method will be a member of the preferences object class and will set the preferences for the user based on input. This method will be called when the user hits save after editing their preferences. |

Table 2: Stock Display class methods

| Stock Display | Description |
|---|---|
| showStock(string) | This method will display the stock ticker and description information. It takes in a string of the stock ticker being displayed. |
| showMetrics(string) | This method will display the likelihood of upside, recommended position length, current stock value, and yesterday EOD value. It takes in the stock ticker as a parameter. |
| showPreference(string) | This method will display the preferences that were associated with that suggested trade, taking the stock ticker as parameter. |
| showCompany(string) | This method will display the company name, taking the associated stock ticker as parameter. |
| showStockChart(string) | This method will display the stock chart, taking the ticker for the chart as a parameter. |

Table 3: User Transaction History class methods

| User Transaction History. | Description |
|---|---|
| showAllTrades(id) | This method will display the history of all trades made by the user. This will take id as a parameter. |
| showPositiveTrades(id) | This method will be similar to the above but only show positive trades. This will take id as a parameter. |
| showNegativeTrades(id) | This method will do the opposite as the above. This will take the id as a parameter. |
| showPortfolio(id) | This method will show the overall portfolio of an individual. This will take the |

| | |
|---|---|
| | id as a parameter. |

Table 4: Statistics class methods

| Statistics | Description |
|---|---|
| checkPreferenceStatistics() | This method will take in no parameters and show the user how often their preferences are used by other users. |
| checkScore(id) | This method will take in the user's id and show them how they compare to other users in the system. |

Table 5: Stock Generation class methods

| Stock Generation | Description |
|---|---|
| suggestStock(id) | This method will take in the id of the user and return a string with the stock suggestion based on the preferences of that user. If there are no preferences associated with that user, this method will return a string with an error message. |
| runComparison(id, checkHistory(id)) | This method will compare preferences with a user's history. It will take in id and checkHistory(id) as a parameter. It will modify a queue by removing stocks similar to previous ones that resulted in a loss. It will return nothing. |
| checkHistory(id) | This method will view if preferences in past trades gained a positive result. It will return an object with preferences and how often they lead to a positive result. |

Table 6: Stock Database class methods

| Stock Database | Description |
|---|---|
| getDayHigh(string) | Passes in the stock ticker as a string and returns the high of day price. |
| getDayLow(string) | Passes in the stock ticker as a string and returns the low of day price. |
| getOpen(string) | Passes in the stock ticker as a string and returns the price of open for the day. |
| setTicker(double) | Passes in current value for a ticker and returns the current price of the stock |
| setPurchase() | Passes in 1 or 0 depending on if the trade was made by the user and returns the true or false equivalent. |

Table 7: Security class methods

| Security | Description |
|---|---|
| getUsername(id) | This method will call the firebase method to get the first and last name of the user, and return it as a string. |

| | |
|---|---|
| userTimeOut(time) | This method will take in time in seconds as a parameter and will be used to determine how long a user can be inactive before a log out will occur. |
| checkLoginDevice(id, ip) | This method will compare if a user’s device is the same as its usual login. It will take in the user's id and their ip. It will return a bool to determine true or false. |
| resetPassword(id) | This method will take in the user's id and send them an email to change password. It will return nothing. |

There are a few main data structures that will be utilized in our project. Our database is Firebase and we will be using the Realtime database. This means that there are no tables being used, but data is represented as a JSON tree. When records are added to the tree, a new node is created with an associated key to reference that data. On the Javascript side we will also be using queues to store data associated with trades that have been generated. When the user requests a new trade, the next trade in the queue will be displayed to the user. We will also be making use of arrays and objects to structure data. Additionally, many get methods that we would create will be handled by the API that connects to the brokerage, somewhat simplifying the amount of get methods required.

---

## 5. Detailed Design

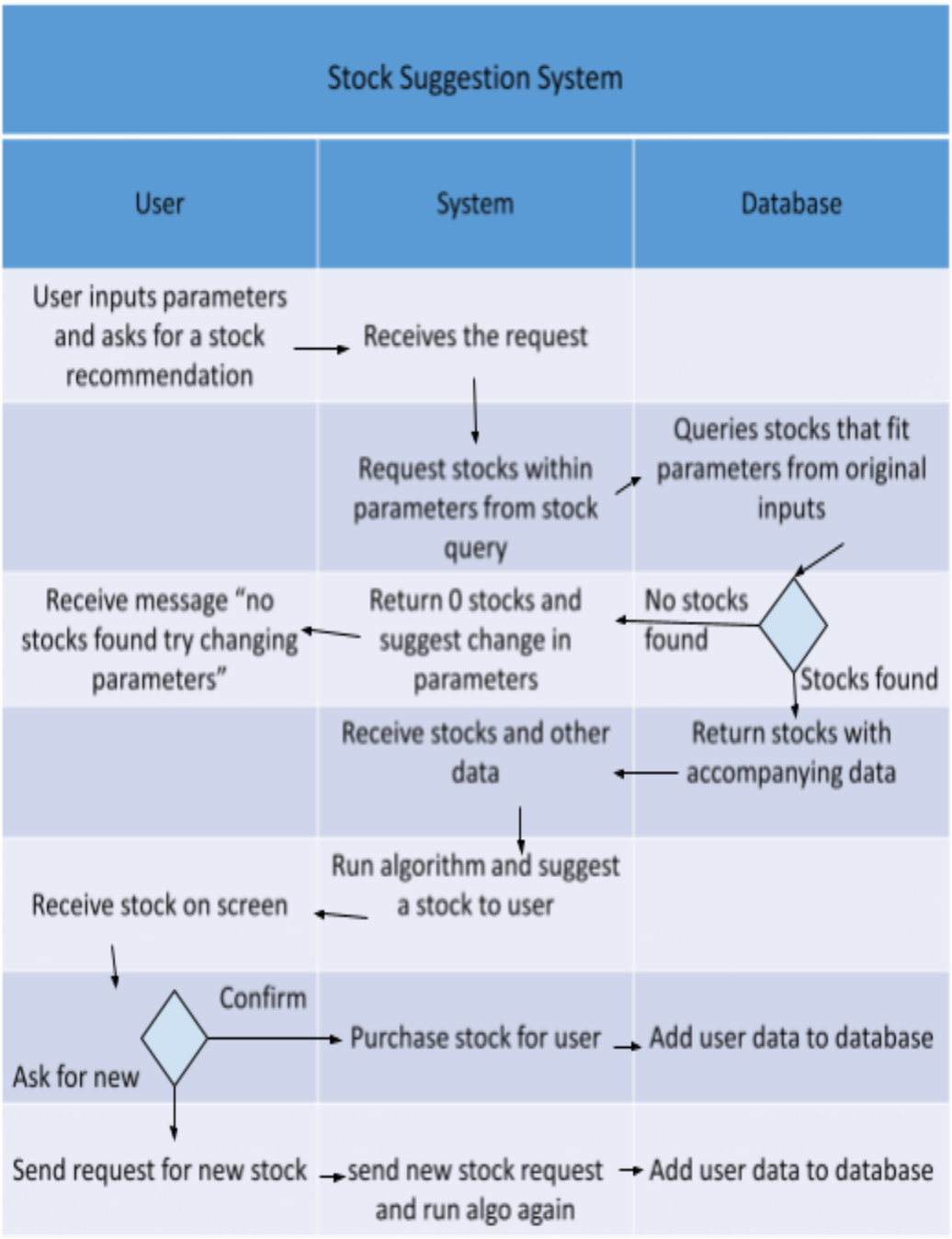


Figure 3: This activity chart shows the basic functionality of the software.

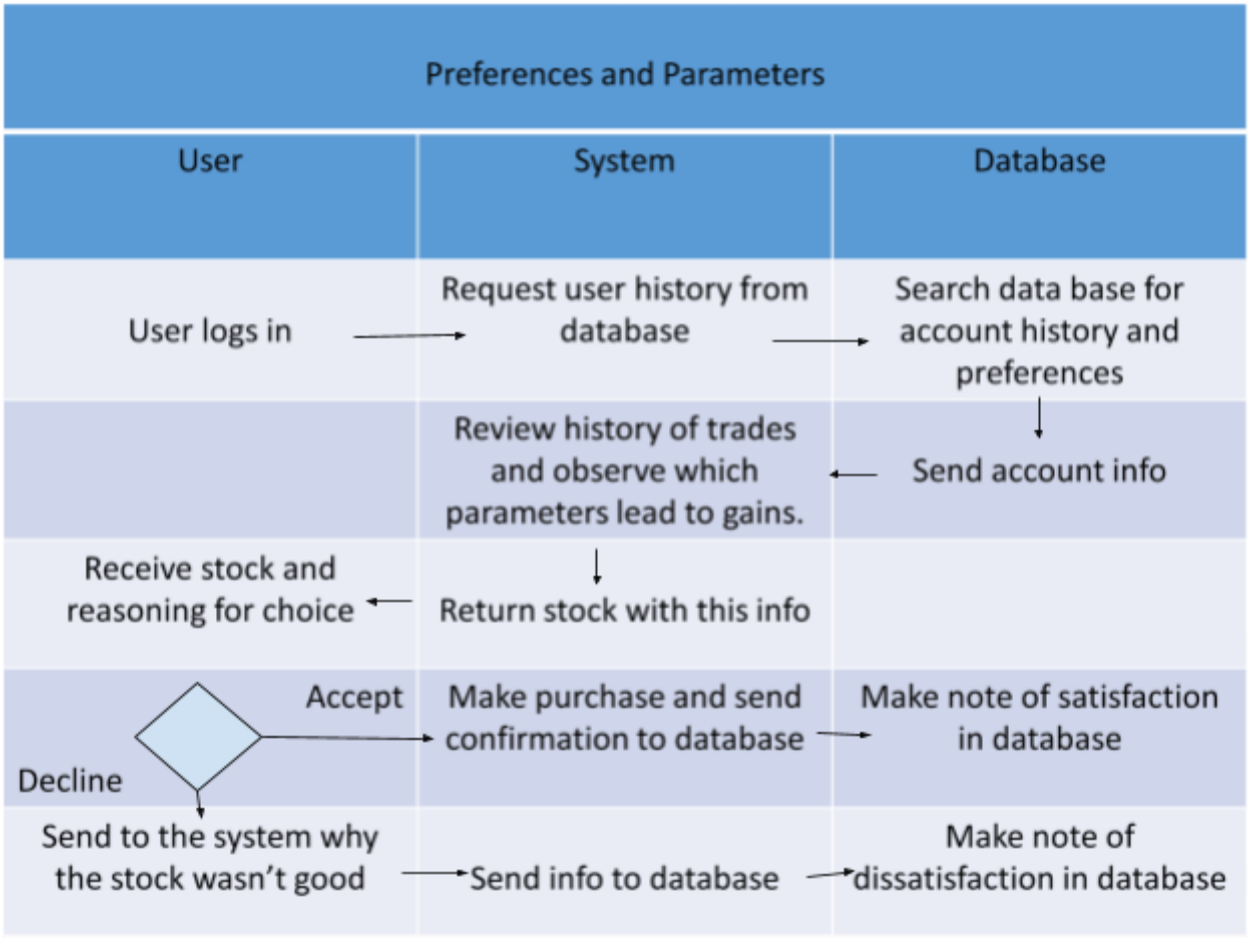


Figure 4: This activity diagram shows the effect of history on our original process.

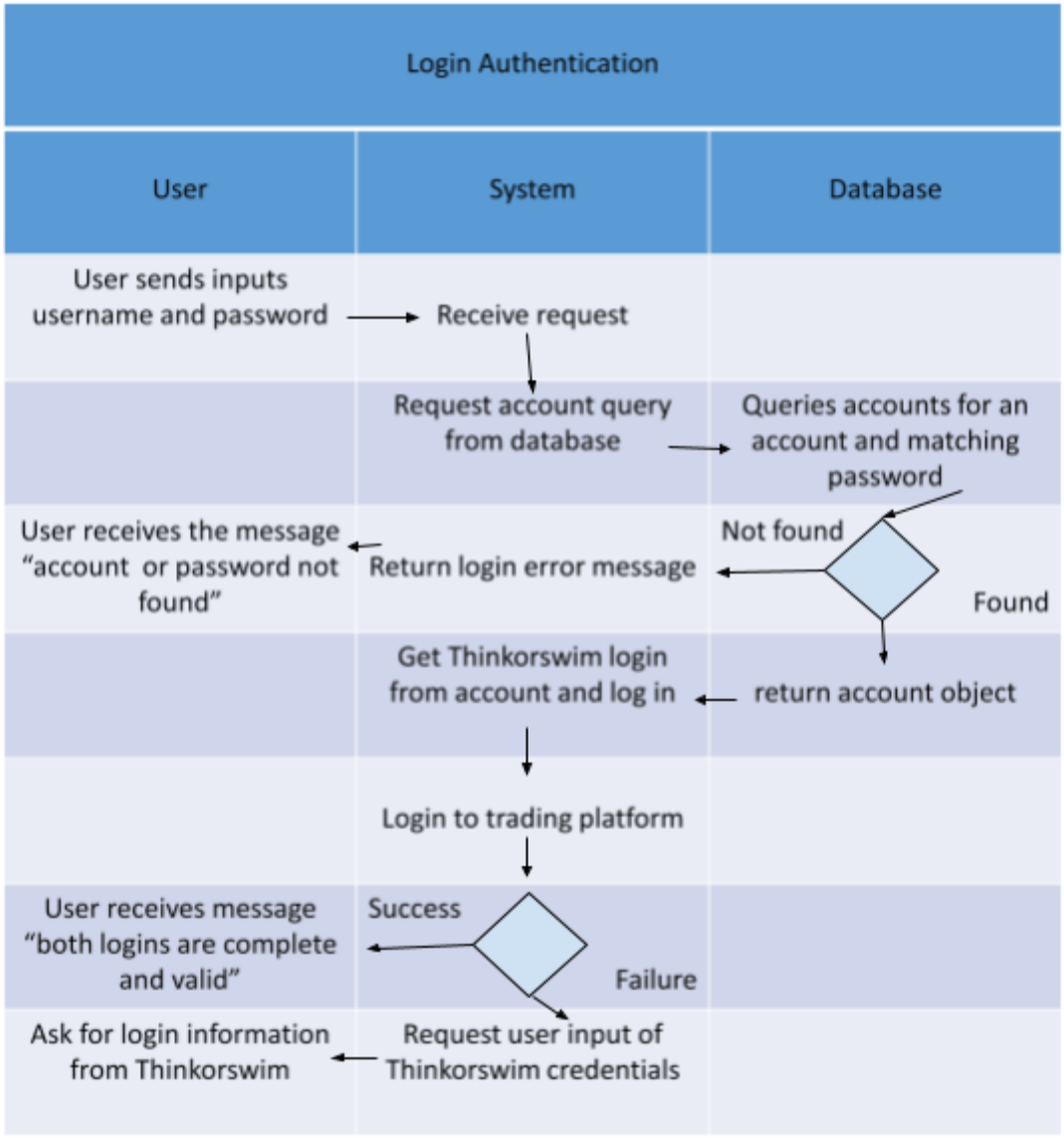


Figure 5: This activity diagram shows how both passwords and usernames interact within the system.

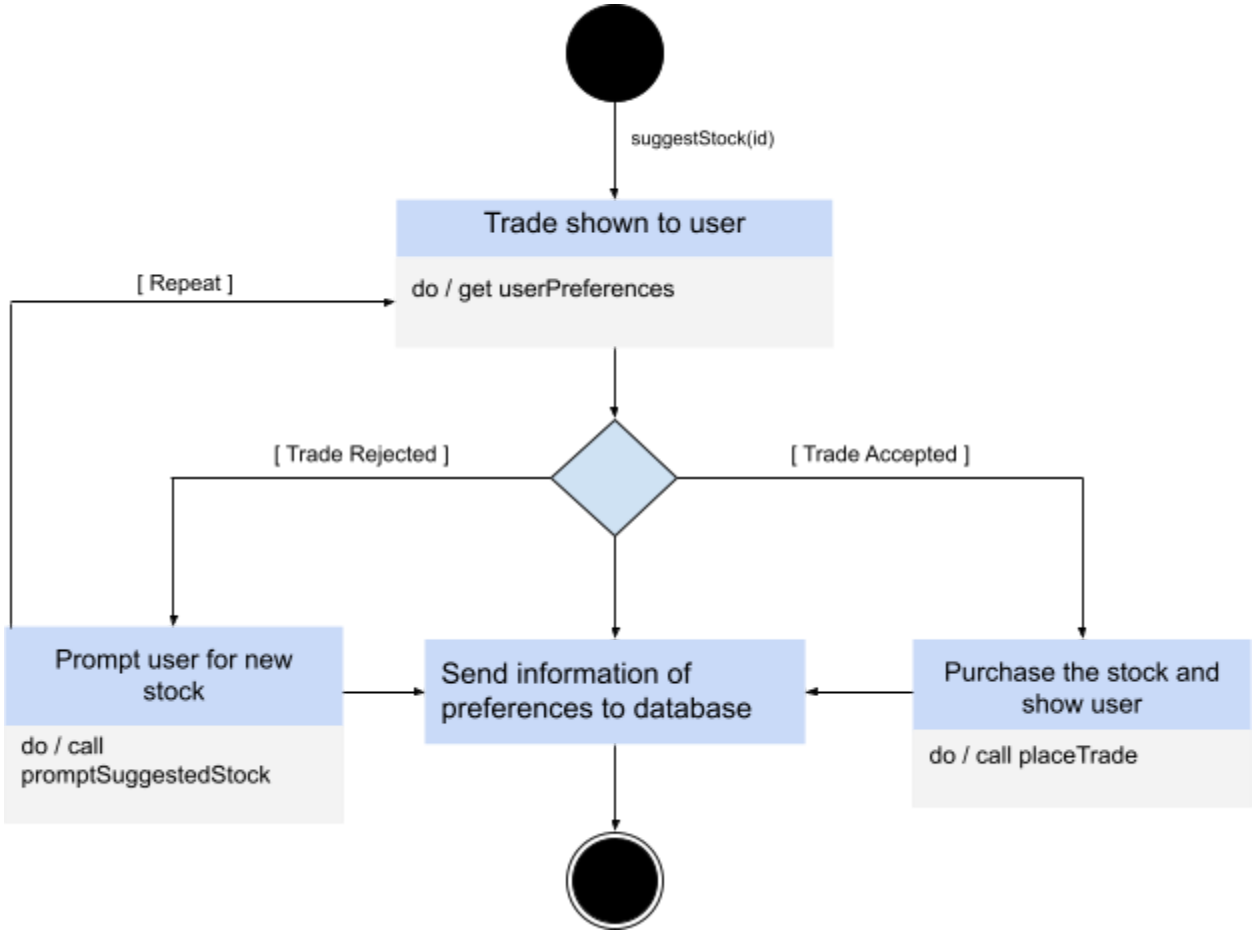


Figure 6: This state diagram shows the user's process for placing trades while running the program.

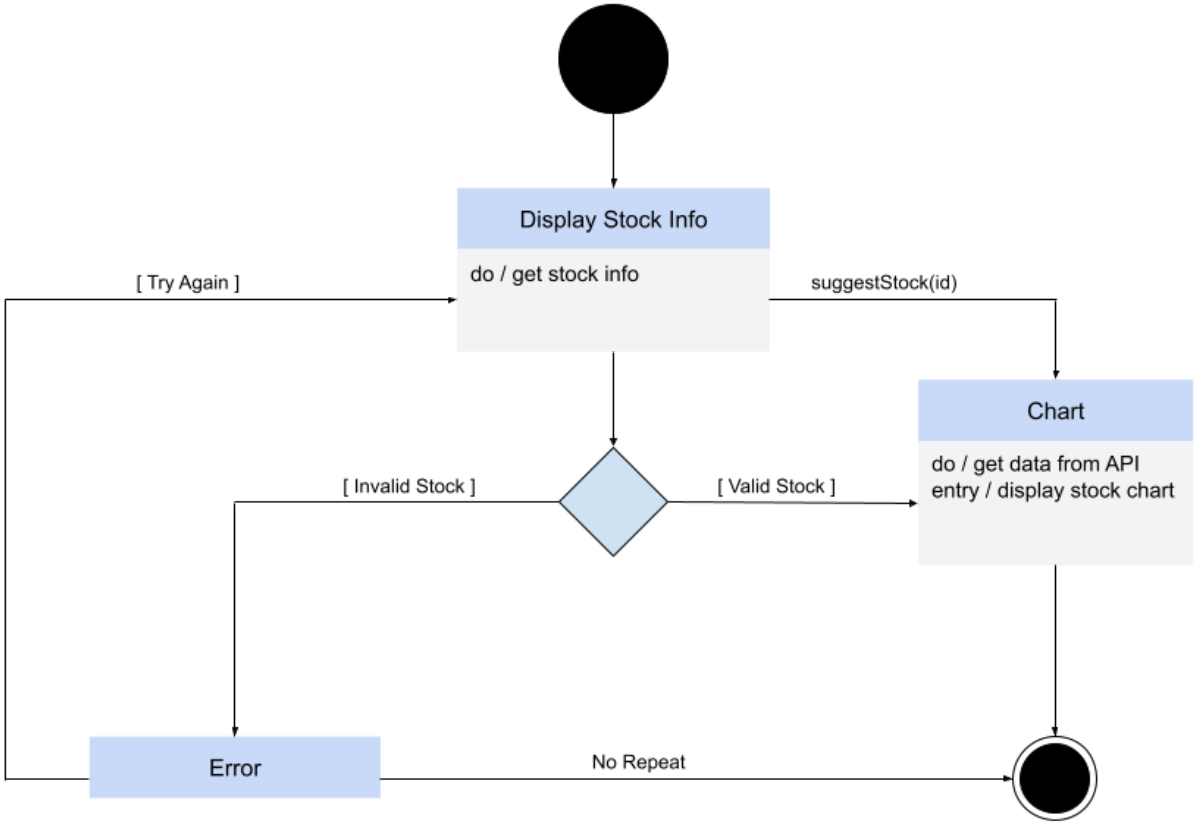


Figure 7: This state diagram demonstrates the system process for displaying stock data and information.

---

## 6. Initial Hardware Design:

Our project is not related to hardware in any way. The only hardware requirements are a computer and internet access.

---

## 7. User Interface Design:

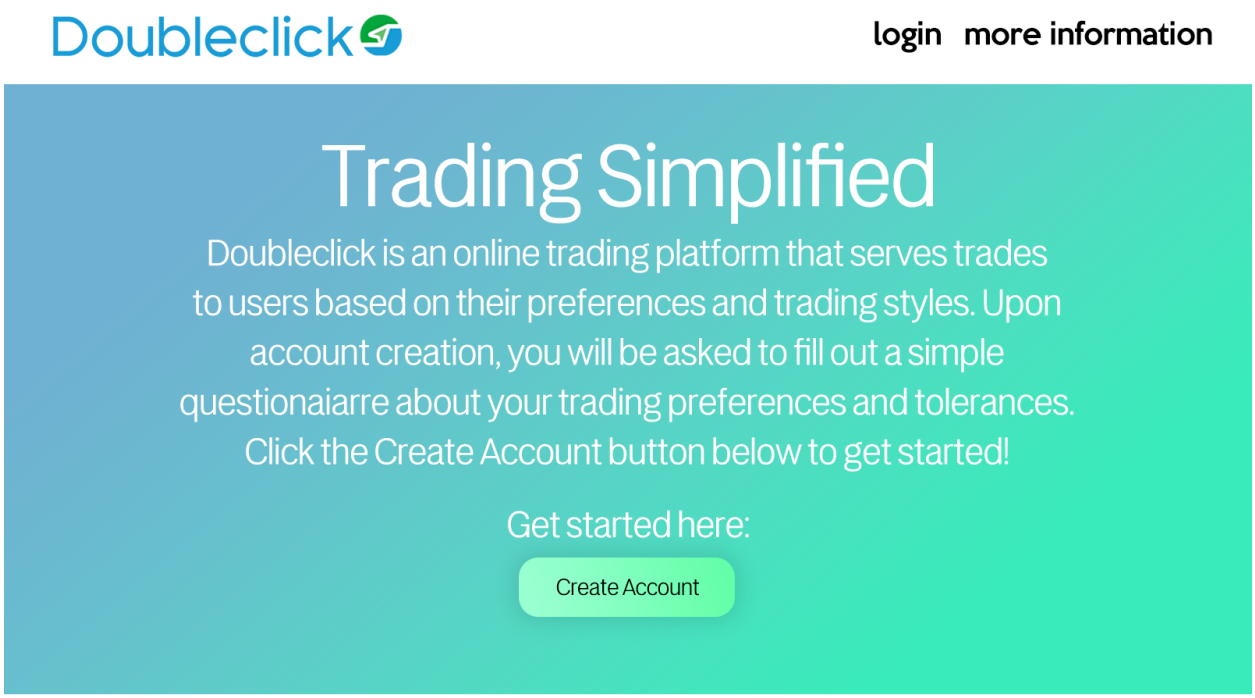


Figure 8: Listed above is the welcome screen for all users of the web service. The page has a large basic description of the site with buttons to allow login, more info, and account creation.

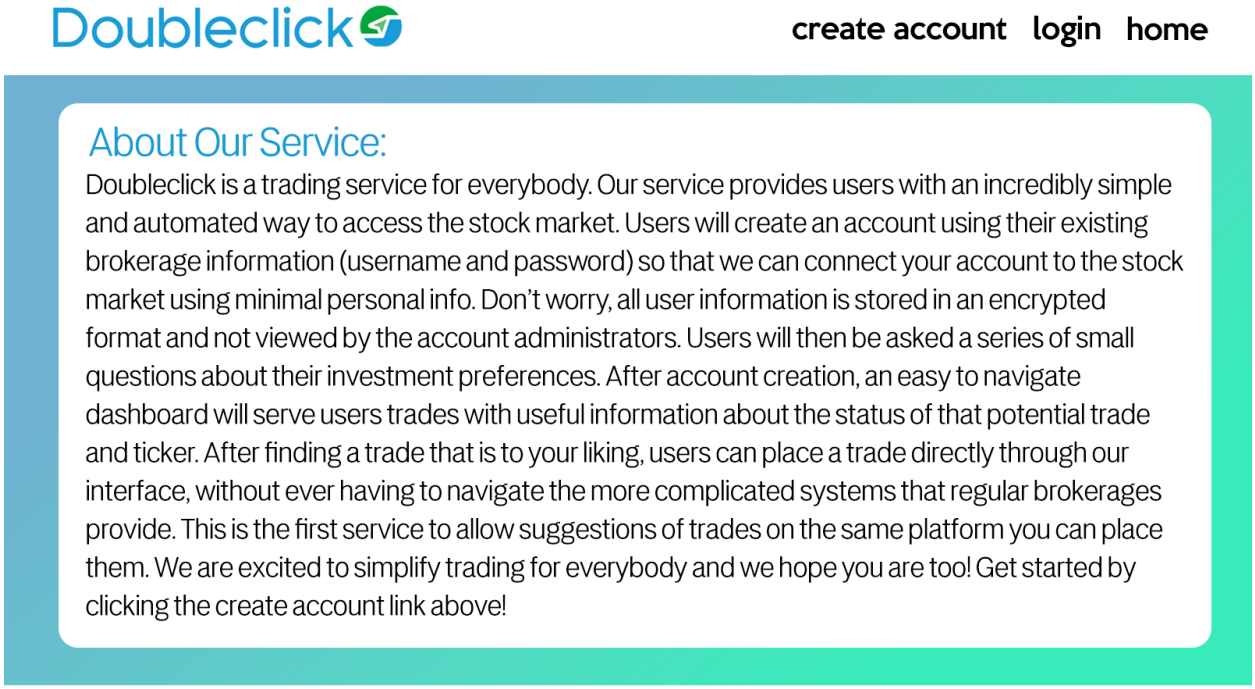

Figure 9: This is the more information screen. This section of the website tells potential users more about what the service offers in greater detail than the landing page. From here users can create their account, login, or go back to the landing page.

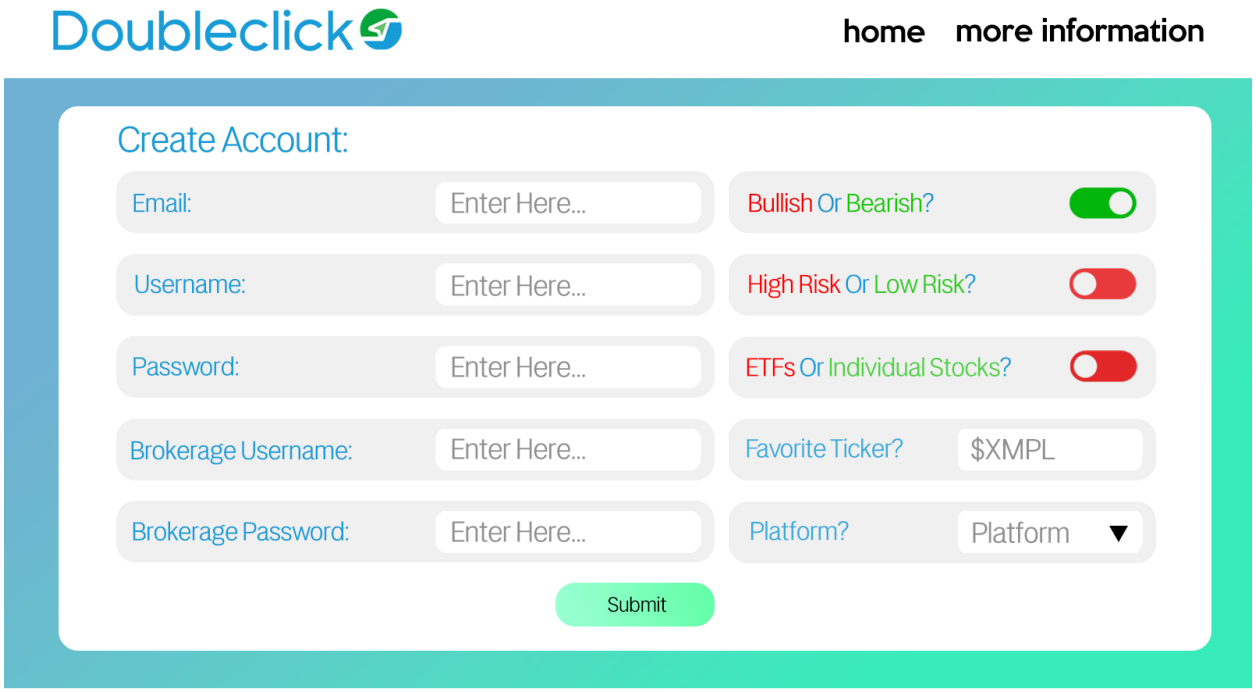


Figure 10: This is the account creation screen. This asks the user for all of the relevant information to create their account as well as presents them with the questionnaire that will determine the trades that will be served to them.

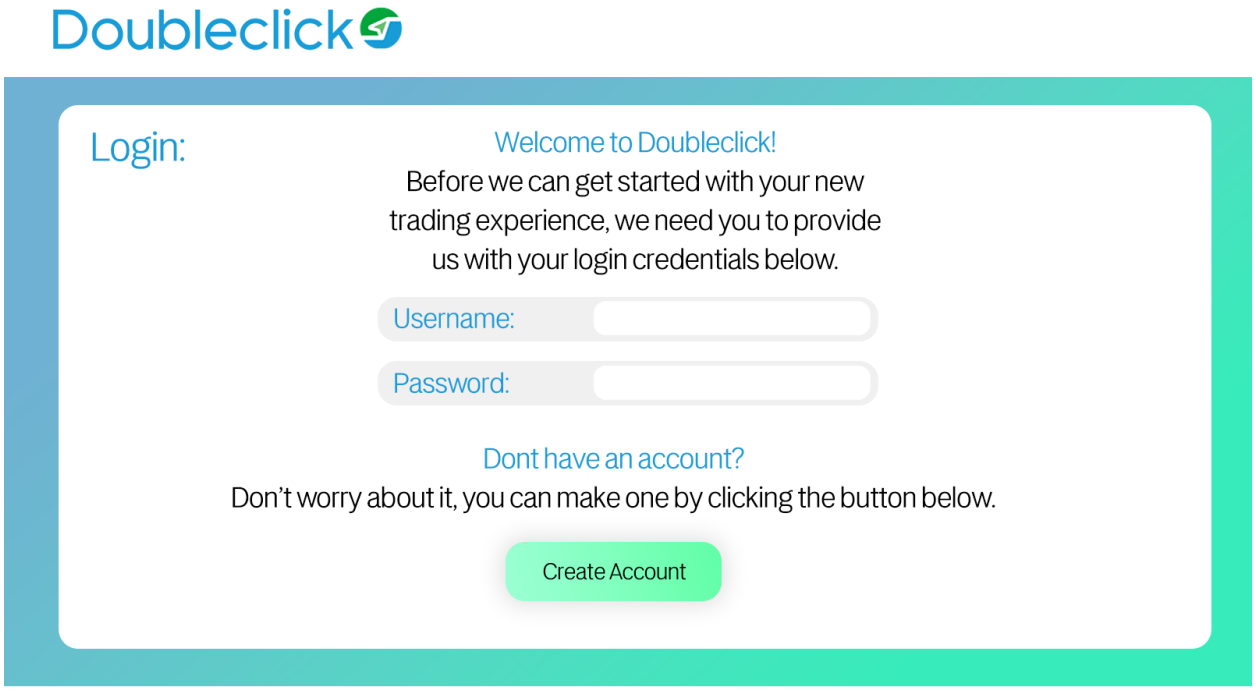


Figure 11: Above is the login screen for our users. This allows users to enter their username and password so that their credentials can be authenticated. If the user doesn't have an account they can click the create account button from here to be directed to that section of the service.

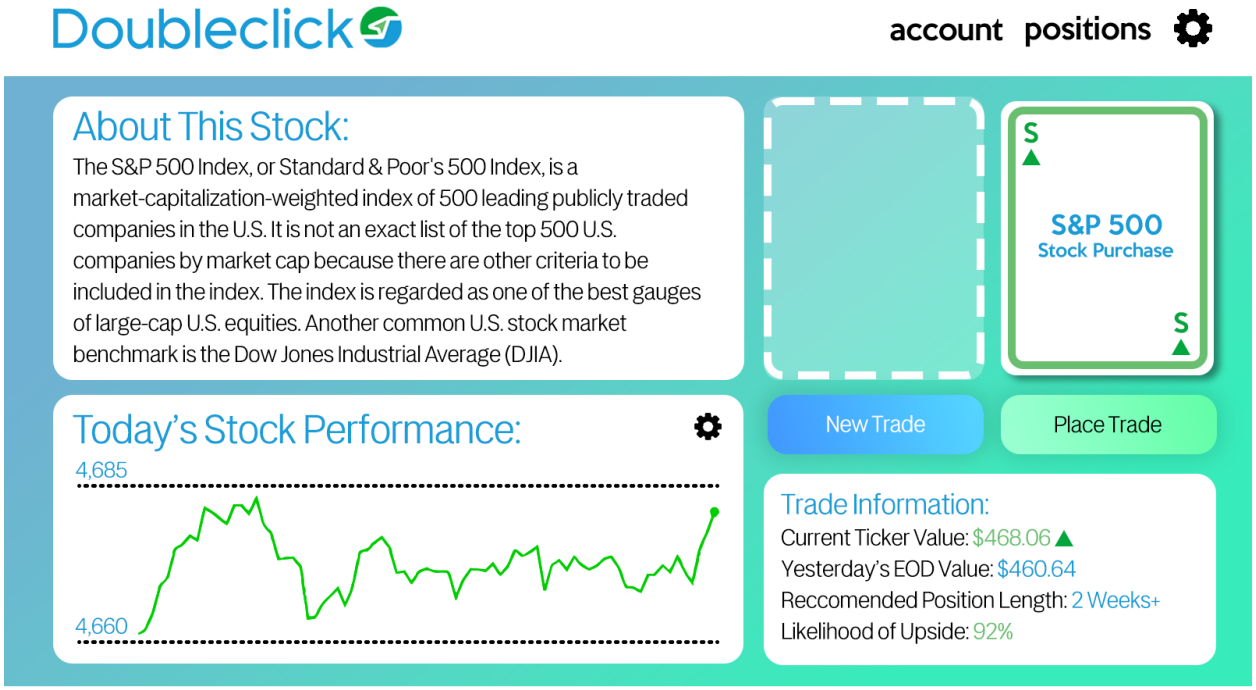

Figure 12: Listed Above is the Main UI for our Project. This features stock information, stock performance, potential trades, ability to place trades, and find new trades. You can also navigate to the account, positions, settings, and graph settings page.

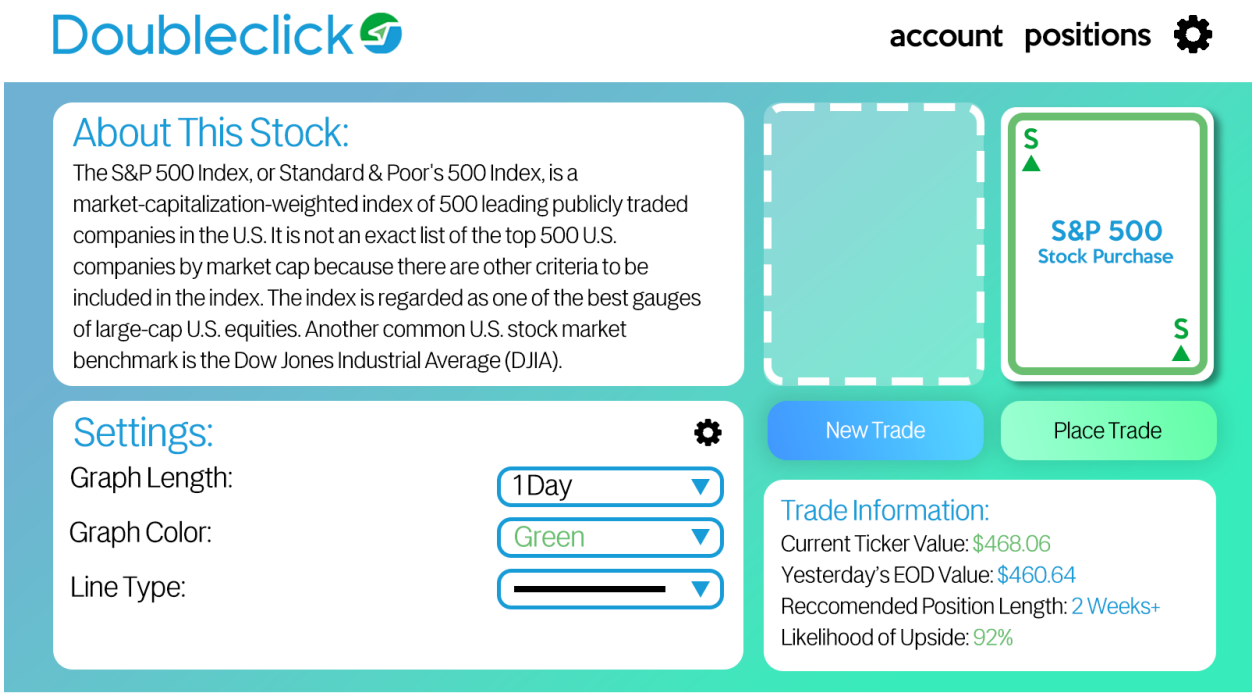


Figure 13: This screenshot demonstrates the look of the graph settings page. The graph settings allow you to adjust the period it displays, the color of the graph, and the line type.

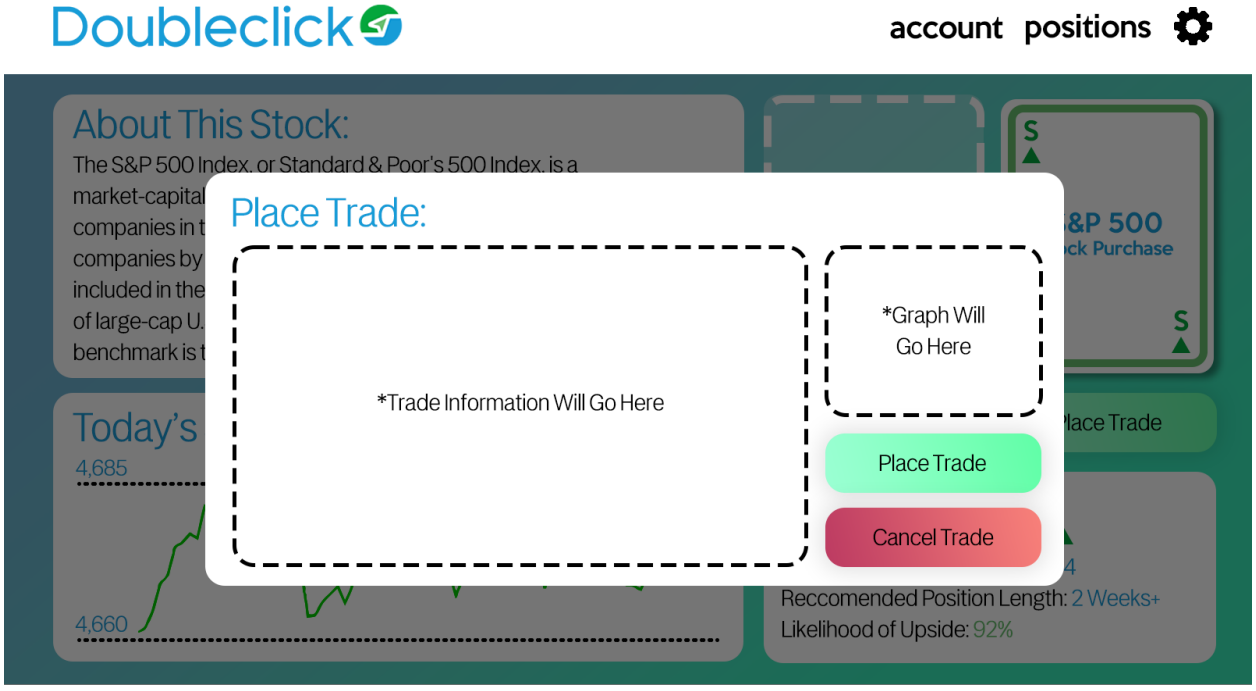

Figure 14: This screenshot demonstrates what it looks like to place a trade on our platform. UI is simple but communicates the necessary info. The user can place a trade or back out from this screen.

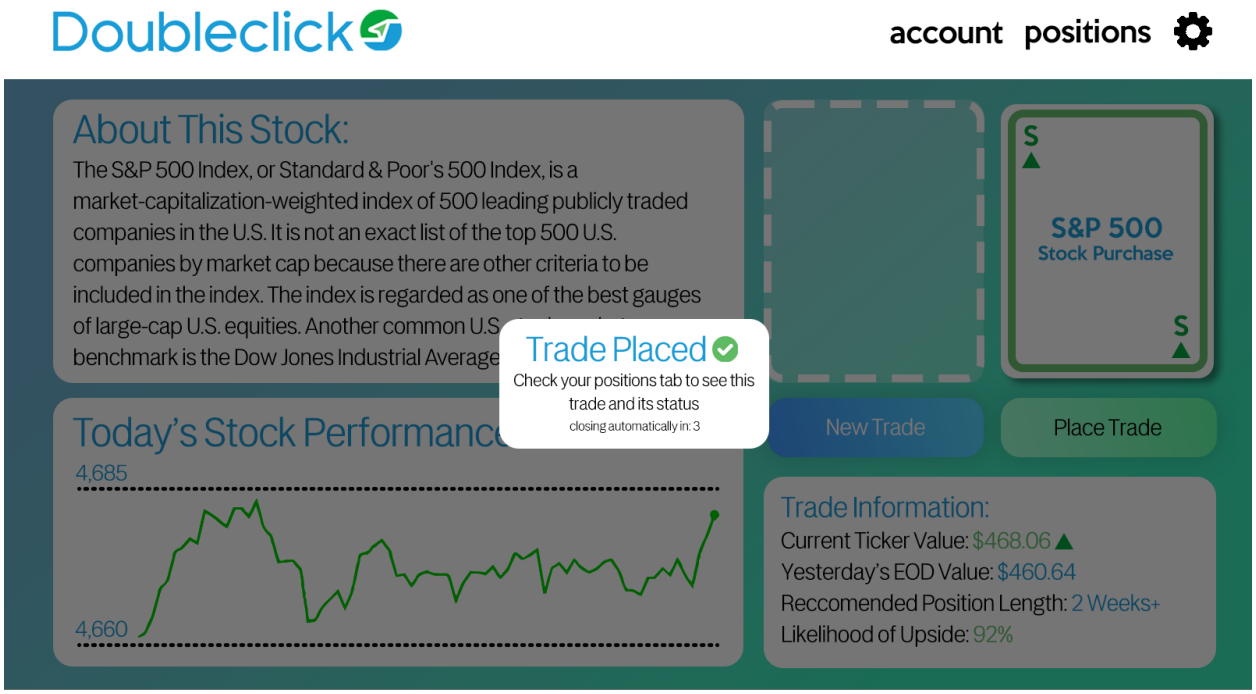


Figure 15: This is the message that is displayed to users once a successful trade has been placed. It informs users that they can open the positions tab to see information about the trade. The window will automatically close after some time and a countdown is displayed.

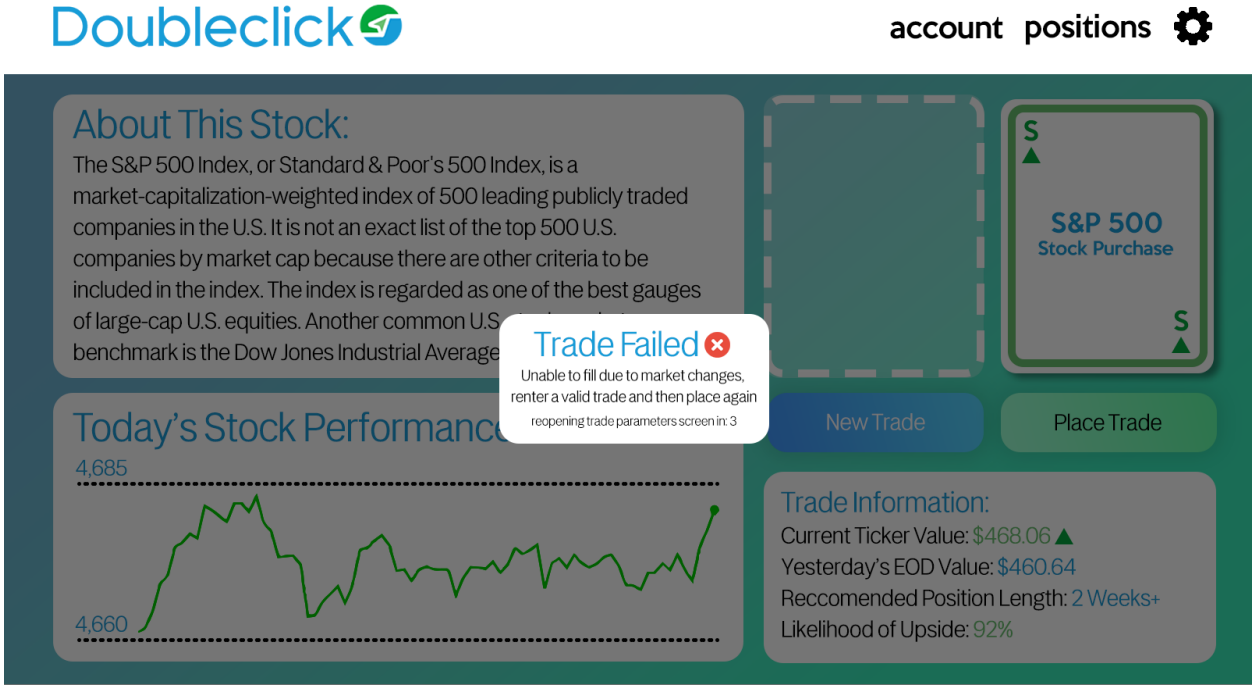

Figure 16: This is a message that is displayed to users once an unsuccessful trade is attempted to be placed. It informs users why it was unable to be placed. The window will automatically close after some time and a countdown is displayed.

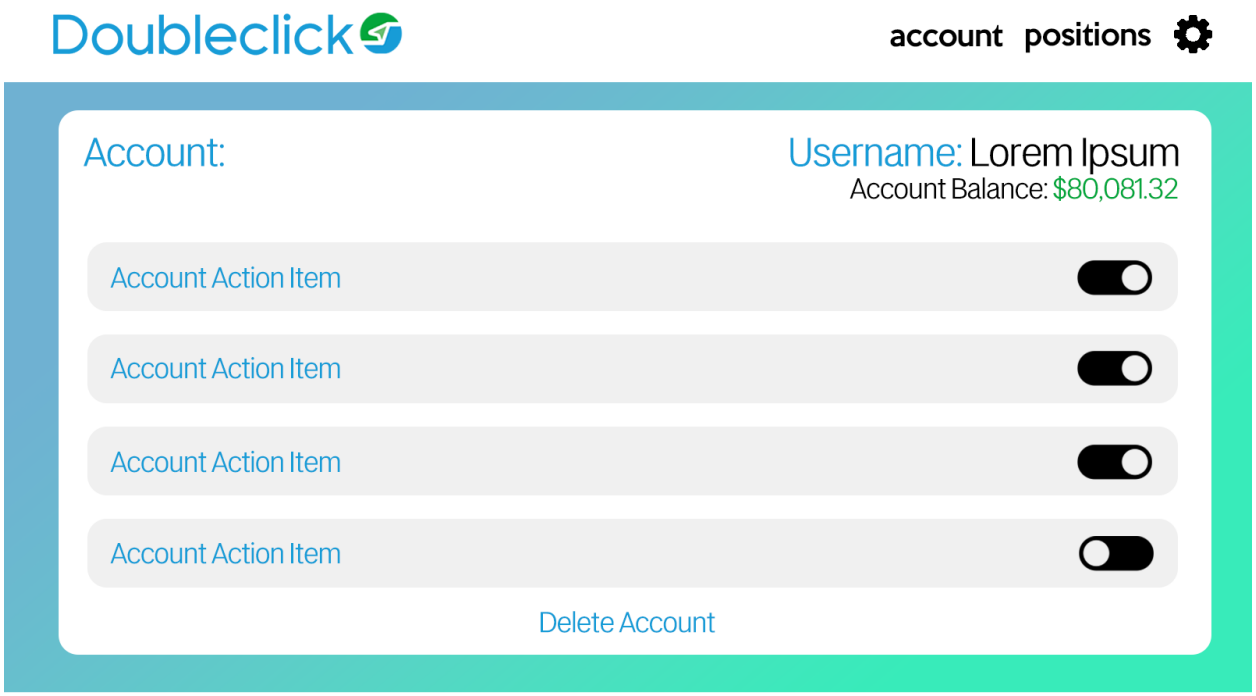


Figure 17: This screenshot shows the account management screen. The user will be able to change various factors relating to their account or delete it.

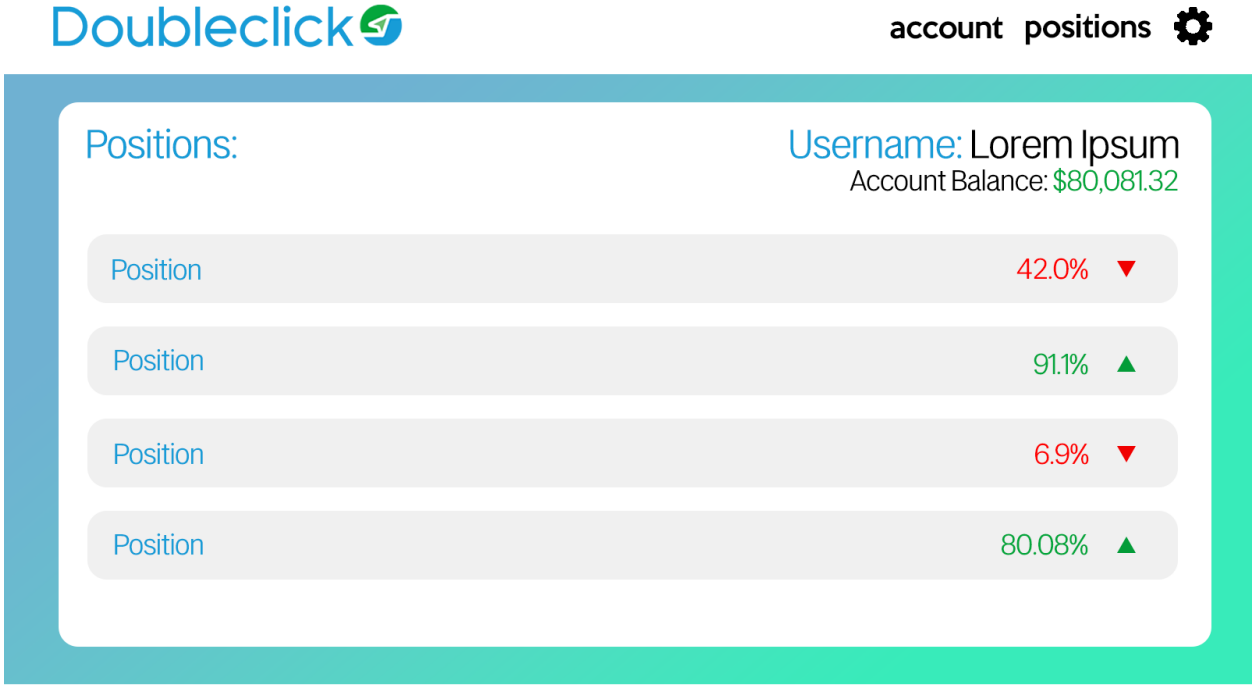

Figure 18: This screenshot demos the positions screen. This is a simple page that shows all the users current positions and their relative performance.

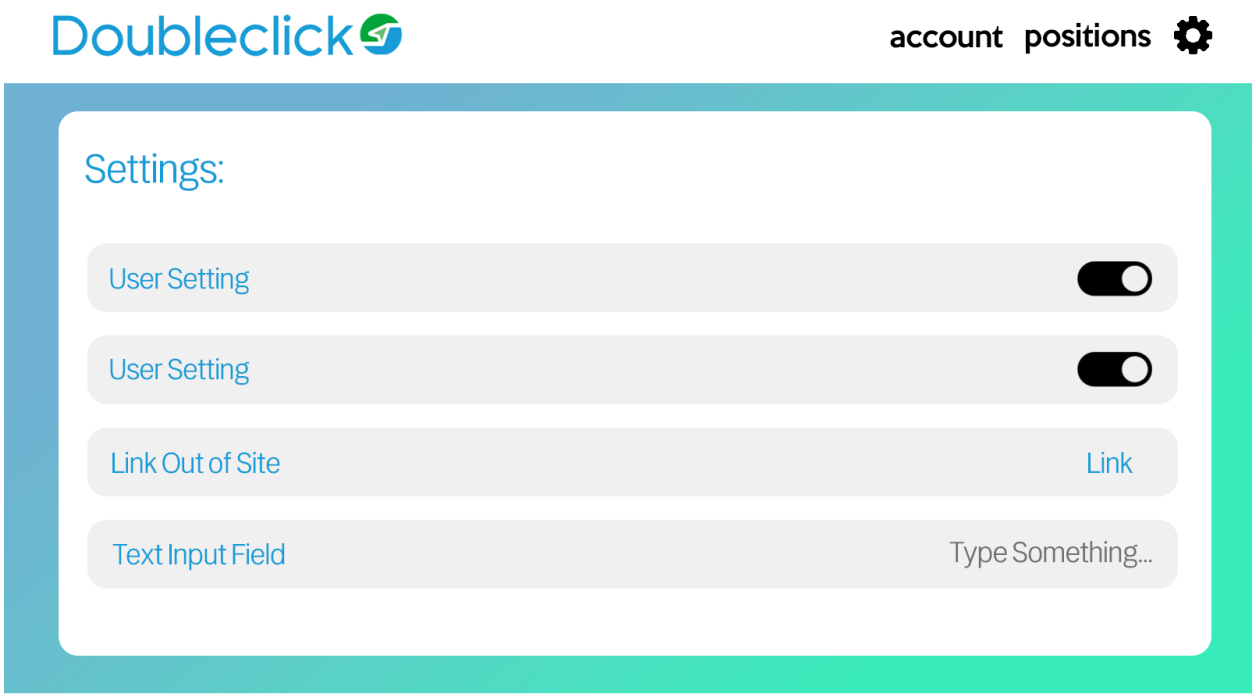


Figure 19: This screenshot shows the settings management screen. The user will be able to change various factors relating to the service and tailor their user experience specifically to them.

---

## 8. Version Control and Change Management System:

The link to our public repository on Github can be accessed via:

https://github.com/shi-nry/DoubleClick_T30

---

## 9. Contributions of Members:

Table 10: Time worked distribution between team members

| Member | Time (hrs) | Details |
|---|---|---|
| Colin Comstock | 6 | ● User Interface design, User Class Methods |
| Loren Parvin | 6 | ● Abstract, Introduction, User Class Methods, Data Structures |
| Nicholas Rinehart | 6 | ● Detailed design and methods |
| Eugene Eom | 2 | ● High level and medium level design |
| Henry Shi | 6 | ● High level and medium level design, Detailed design, Version control and change management system |