Final Group Report

* Prologue
  + Team Name: Nird
  + Team Slogan: Nirds of a feather
  + Team Members: Patrick Gallagher, Spencer Russell, Tianyi Zhang, and Hank Zhu
  + Project Name: Stock Ticker

Sprint 1 Report:

1.1 Teammates Roles

a. Scrum Master: Patrick Gallagher

b. Product Owner: Tianyi Zhang

c. Team Members: Spencer Russell and Hank Zhu

d. Combined Hours on the job: ~15 hours

1.2 User Stories

The first sprint we worked on the connection of the API and the GUI for the program. The API that we are using is AlphaVantage and the project is wrapped in Gradle to help us connect the API with AlphaVantage. The GUI has three frames that connects to each other when searching for a stock, clicking on the searched stock, and adding the stock buttons. The user stories that we chose were:

· As a programmer, I want the program to be in a simple GUI interface. So that the program can be easily used by everyone. – 1pt.

· . As a programmer, I want the project to have an API to disconnect and connect automatically. So that the application can display the stocks in real time. – 8pts.

We have connection with the API, also the GUI can pop up all three frames when clicking on the buttons that revolves around the other frames.

1.3 Design, Requirements, Test Plan (GUI User Story)

The GUI user story’s design had three frames, one for the menu, a frame for the searched stocks, and the third frame about the description of the selected stock. The requirements for the GUI were the panels, buttons, and a few of different functions for the frames, like the buttons going to different frames when clicked. The menu frame has only one requirement that when the user hits enter it goes to the second frame. The second frame has an array and a for loop that makes buttons for each item in the array. When clicking on one of the buttons, the third frame will show up with the description of the stock and an option of adding the stock. The testing in this sprint was testing for what each frame looks like and the testing on for each button that connects to another frame.

1.3 Implementation details and issues (GUI User Story)

We have implemented frames, labels, buttons, panels, and other java applications in make the frame. The issues when we implement the java.awt’s is the positions of where the panels go into the frame. The other issue is that when we do implement the ActionListener, some classes will not co-operate with each other and we had to fix some of the code.

1.3 Statement of outcomes (GUI User Story)

· The outcome after testing is that all the frames work and finally connect between each other.

1.3 Names of Implementer(s) of that story (GUI User Story)

· Patrick Gallagher has worked on the menu frame, the first frame, Hank Zhu worked on the second frame, and Spencer Russell worked on the third frame.

1.3 Design, Requirements, Test Plan (API User Story)

· The second user story was working on the connection of the API. The design for the API is that it calls the stock numbers for the day. Also, the API grabs the stocks information and the stock name. The requirement for the user story is that we just want to have a connection for the API. TEST CASES

1.3 Implementation details and issues (API User Story)

· The implementations on API is using patriques and we had to wrap the project in Gradle to get Project and External Dependencies jar files. Wrapping the project in Gradle will make project able to connect to AlphaVantage. The only issue is that the project needs to be wrapped in a Gradle project to make the project to work.

1.3 Statement of outcomes and Names of Implementer(s) (API User Story)

· The outcome of the testing is that the description of the stocks and how high or low the stock numbers. Tianyi Zhang implemented the API to the project.

1.4 Integration Testing

· There is only one integration test that the project had was the connection between all three frames. All three classes used ActionListener, but only two classes use the other classes frames names to call the class frame. The third class, stockName, uses a constructor in the display class called display.run() to run the display frame. The quality of the product is that all the buttons makes a new frame from a different class shows up. Everything works in the project that is supposed to work and there are no errors.

1.5 Scrum 1 Retrospective

· Hanks’ perspective: I made a panel, which services as a container for favorite stocks that user added personally. I created string list that contains all the stock names that user added, and I made a “for” loop to show these stock names in the panel vertically. My plan is to make a favorite stocks container that user can either add or remove the stock, when user click on the stock, different box pops up with detailed information about the stock that user clicked. For elaboration, I want user had a preview of what their favorite looks like without click on it, and user can also customize the stocks order by clicking and dragging it.

· Spencer’s Perspective: My panel has been designed and cannot currently be improved due to the lack of API. This panel is supposed to help with the one user story involving understanding the history of stocks by displaying all relevant information. Another possible story relation may be to the given functionality of adding stocks to a personalized list. The panel I designed is going to be implemented as window for presents information related to the stock including the stock listing abbreviation, full business name, a brief description of the company, and a graph that displays the rise and fall in stock value for the past day, week, month, and year of information, and a button for adding the stock to the user’s collection. Currently, the window is lacking functionality as the API has yet to be implemented so no information is being displayed. I need to better apply my time, possibly by assisting others or seeking assistance for myself.

· Tianyi’s Perspective: In this Sprint, I oversaw working with the API. I spent most of the time try to understand how to get the JSON data become Java beans. I found this API service of stocks called “Alphavantage” and a wrapper called “alphavantage4j”, with them I can have all kinds of stock information. Then I spent around three hours making the daily graph of a single stock. It can display the price and volume of a stock in one day from 10am to 16pm in every 60 minutes. This graph can be called with the command StockHourlyGraph.run(Lday,Lday.get(0).getDateTime().toString()), where Lday is a list that contains information objects of a single stock

· Patrick’s Perspective: In this sprint I have made the menu frame. The frame has four different panels. One panel for the user information, the user name and other information. The second panel is the search function that allows the user to search what stock they want to look for. The third panel is all the added stocks that the user has bought or favorited, so far, we put random items in the array to show that the array works in the early version of this project. The last panel is the information of an added stock that the user has clicked on. The search bar, when hitting enter, Hank’s frame will show up.

1.5 Product Owner Statement

· By the end of the Sprint I, the team members have finished the design of the GUI of the Stock Ticker. The frames are runnable and are linked with each other. But they cannot work with the functionality of saving the stock preference or display the graphs, which will be the goal to achieve in next Sprint. Also, the GUI does not look aesthetic, we expect to improve its design after having it works functionally. The team has the first draft of the one-day graph of a stock made, it successfully presents the key information, but the team will need a discussion on the standard of all the graphs. The API works fine with displaying the stock data within a day. But sometimes the API does not connect until several runs. The cause of the error is still unknown.

1.5 Scrum Master Statement

· Everyone has worked on this project around 15 hours combined in this sprint. We would have the API connected to the entire project instead of having just a connection that does not work along with the GUI. There are going to be arrays that have all the items in the API and another array that has the stocks name that the user has searched. Also, the group will start making graphs and descriptions for stocks.

1.5 Set up for Sprint 2:

· As a user, I want multiple data displays when I am looking at a stock. So that I can have an easier time looking at a stock. – 4pts.

· As a user, I want a search function when looking for stocks to add, so that I can find stocks easier. – 2pts.

· As an investor, I want to keep hold of my personal shares. So that I can monitor my own financial gain. – 4pts

Sprint 2 Report:

2.1 Teammates Roles:

* Scrum Master: Hank Zhu
* Product Owner: Patrick Gallagher
* Team Members: Spencer Russell and Tianyi Zhang
* Combined Hours on the job: ~20 hours

2.2 User Stories

* After the connection of the API and made the GUI we started to add functionality and data into the project. The functions in the project is allowing the user to buy the stocks after looking at the data of the graph. Also, the project includes the data of all the stocks in the API. Including the data for the users allows them to have a good understanding of the stock. The user stories that the team have worked on were:
  + As a user, I want multiple data displays when I am looking at a stock. So that I can have an easier time looking at a stock. – 4pts.
  + As a user, I want a search function when looking for stocks to add, so that I can find stocks easier. – 2pts.
  + As an investor, I want to keep hold of my personal shares. So that I can monitor my own financial gain. – 4pts

2.3 Design, Requirements, Test Plan (Data User Story)

* In Sprint 2, we plan to have another graph that presents the data in the past seven trading days, in the same format as the DailyGraph we developed in the last Sprint. We will revise the graph classes to panel classes so that they can be added to the frames. But we will still have method that can run the graph individually for the convenience of testing them.

2.3 Implementation details and issues (Data User Story)

* The team worked out a new java class called “GetStock.java”. It wrapps the API functions in three method, “getEverHour”, “getEverDay”, “getEveryMonth”. These methods return a object of List<StockData> at every given time nodes. It provides convenience for every team members in future development. Because the programmers can simply create a instance of object of GetStock and call the method to get the list of stock data they need.

2.3 Statement of outcomes (Data User Story)

* The Daily and Weekly graphs works fine.

2.3 Names of Implementer(s) (Data User Story)

* Tianyi Zhang, Spencer Russell

2.3 Design, Requirements, Test Plan (Search User Story)

* The search function is a pop-up window that allows users to enter the keywords that they can think about, even though it is not as powerful as Google search, but I managed to ignore the case-sensitive to allow users having more abstract searches.

2.3 Implementation details and issues (Search User Story)

* The implementation part is nearly perfect, we built a user-friendly search bar and nice display layout, with scroll bar on the both right and bottom that allows users have more flexible way of checking their search results. One of the issues we have right now is displaying the searching result as desired, since the database file is not perfectly organized. Another issues is the layout, when trying to implement the favorite function to the search results that allows users to add the stock to their favorites where the buttons do not line-up perfectly as desired.

2.3 Statement of outcomes and Names of Implementer(s) (Search User Story)

* The search function allows user to search repeatedly if the result was not desired, in order to minimize the latency of searching time, I wrote the code the read the file from the local database instead of online API, the file will only contain names of the stock, acronym of stock, and type of the stock, which are the only information required in order to proceed the next step, the step is get the detail information of the stock by reading the stock’s acronym.

2.3 Names of Implementer(s) (Search User Story)

* Hank Zhu

2.3 Design, Requirements, Test Plan (Add Stock User Story)

* During the second sprint, the team wanted to allow the users to add the stocks after they saw the data for the stock. They thought it was a user friendly action, the programmers did not want to fool the user into buying the stock first time searching the stock, but after they saw the information of the stock is when the users can buy the stocks. The requirement for the story is that the users will always have the information of the stock that they searched for before making the decision of the buying the stocks. The testing in this story was when the user hit the add button in the display class. Once the add button was hit, the stock would be written in a text file and be read in the Stocks class.

2.3 Implementation Details and Issues (Add Stock User Story)

* The group implemented text files into the project because they thought it would be easier to implement instead of a database. So far the project can only add the facebook stock, but in the next sprint the project will add the correct stock that the user inputted. The biggest issue of adding the stocks is that once the program comes back to the main menu, the list does not update itself and will not show the newly added stock. The user has to exit out of the program and run it again.

2.3 Statement of outcomes (Add Stock User Story)

* The outcome of the story is that after the user adds their stock, the stock name will be copied to a text file. The text file shows how much the user has bought and prints out to user in a form of buttons. The buttons will go to the display frame, displaying the information.

2.3 Names of Implementer(s) (Add Stock User Story)

* Patrick Gallagher, Spencer Russell

2.4 Integration Testing

* The integration testing that went into this sprint was that the graphs that were implemented into the display graph. After the testing of the graphs being a separate class, the program started to implement the graphs into the display frame. The second integration test was after testing the display frame writing stock names into a text file, the stock frame started to read from the text file and printing the buttons from the text file.

2.5 Scrum 2 Retrospective

* Spencer’s Perspective: In this sprint, I worked with Tianyi to display his graphs in the stock information window and to paint non-numeric data along the side of the graph. There were a few issues with populating the fields as well as the inability to switch between graphs. I am still working towards the ability to record new stocks to the user’s list which when completely implemented would allow us to begin working towards non-preset examples. I am working towards improving my java skills to no longer hinder my teammates. Team work has significantly improved from the first sprint, which has helped us with completing our tasks.
* Tianyi’s Perspective: In this sprint, my responsibility was to make the monthly graph. During the process, we encountered some difficulties such has having the stock information frame and the graph connected, and we still having difficulty reading data from the API. The GetStock.java class helps solving that trouble. The graphs can be improved in some way, for instance, having more time nodes so the trend lines will look more smooth. But it requires a lot of work. The team is not decided on whether we should implement this change.
* Patrick’s Perspective: In this sprint, I was working on the functionality of the add button in the display frame. I implemented a text file to keep track of the all of the added stocks that the user has bought. I am still working on how to update the stock frame after the user has bought the stock, showing the newly added stock button. The team have worked a lot in this sprint than last sprint.
* Hank’s Perspective: In this sprint, what I did looks pretty easy, because it just a search function that linked to the stock database, all cI did is creating a “for” loop that keep checking the user input, and tells whether it matches. I found a method that ignore the case-sensitive, which allows users to have more abstract search when users do not what exactly they wants. Secondly, when I trying to implement the “favorite” button to the search function, the display layout becomes the problems, I could not find the perfect way of lining up all the buttons. But I will fix it at the next spring.

2.5 Product Owner Statement:

* The end of this sprint the program can now print out some of the data of the stock, even though the program can only print a predetermined stock, but they are currently working on a way to have a connection between their two frames. The display frame also has a predetermined stock, but it also displays the daily graph and the current price of closing and the stock name. There are some choppy parts in this project that needs to be fixed like the ability to update the stock frame to correct number of stocks without closing and rerunning the application.

2.5 Scrum Master Statement:

* Everyone has worked on this project around 20 hours combined in this sprint, which demonstrates the commitment that everyone putted in this project.
* There are few details that we are focusing on during this sprint.
  + Connecting API to the every file of the project.
  + Making the user-interface looks better.
  + Displaying the graph based on hours, days, and months.
  + Integrating the search function to the detailed stock information panel.
* Overall, everyone did a wonderful job, especially Tianyi who did an excellent job of helping his teammates for the API and the stock graph.

2.5 Set up for Sprint 3:

* As a user, I want a help function for the application. So that I know exactly what the program is doing. – 8pts.
* As a user, I want to make a password for the application, so that I can protect my stocks. – 16pts.
* As a user, I want to compare the same stock from two different years. So that I can see how the stock does in long term. – 4pts.
* Also starting to debug and make the program look nicer than it is now

Sprint 3 Report:

3.1 Teammates Roles:

* Scrum Master: Tianyi Zhang
* Product Owner: Spencer Russell
* Team Members: Patrick Gallagher and Hank Zhu

Hours:

Patrick Gallagher: 10 hours

Spencer Russell: 6 hours

Tianyi Zhang: ` 10 hours

Hank Zhu: 10 hours

* Combined Hours on the job: 36 hours

3.2 User Stories

* As a user, I want a help function for the application. So that I know, exactly what the program is doing - 8pts
* As a user, I want to make a password for the application, so that I can protect my stocks. -16 pts
* As a user, I want to have data graphs for different time periods, so that I can see the trends more conclusively - 4 pts
* As an investor, I want to keep hold of my personal shares. So that I can monitor my own financial gain.-4 pts

3.3 Design, Requirements, Test Plan (Data User Story)

* The AlphaVantage API has not been willing to cooperate. For everytime we retrieving the data for one stock, it fails for random number of times before it works. and it cost from 30 secs to 5 mins. As we approaching the end of the project development, the team do not feel like switching to another API. Therefore, we came up with a idea that can waive the waiting time in a way. We decided to preload all the saved stocks data into a class variable, so when we search for their data, it will retrieve from the saved list instead of calling the api. It will take a while opening the software, but we don’t have to wait after all the saved stocks data are preloaded.

3.3 Implementation details and issues (Data User Story)

* When the software is opened, the preload method in Get Stock class read the txt file “Added Stock” and recognized all the stock names the file contains. The method will store the data in three different time periods which are hourly, daily, and monthly in a 2D list : List<List<Stock Data>>. Get Stock class also has functions that can return the list of stock data when user is searching for a stock. When user search for a stock that was not stored, the method will call the api to search for the data.

3.3 Statement of outcomes (Data User Story)

* Depending on the performance of the API, it took from 30 secs to 3 mins to preload all the saved stocks, the monthly data usually takes longer. This issue is really not under our control. The group made the best effort we could.

3.3 Names of Implementer(s) (Data User Story)

* Tianyi Zhang

3.3 Design, Requirements, Test Plan (Login and Registration User Story)

* I designed both login and registration function for stock ticker in order to protect user’s data in a better level. My user interface is well neat and simple with two columns in the main page (username and password), and three columns in the registration page (username, password and confirm password). If the user doesn’t have a username and password, all they need to do is go to registration page where they can get a new one.

3.3 Implementation details and issues (Login and Registration User Story)

* Implementation process was painful and fun, since I had to write an algorithm to ask registration page validation between “password” and “confirm password” column, if these two doesn’t match, the program will reject the registration and ask them to try it again. If the registration completed, it will direct user to the main page where they can login in their username and password that they just registered. On the background, the program will write the username and password to the local file, so the main login page is able to compare the information with the user input in the main page and decides whether user typed right password or not.
* The issue I have right now is the individual profile for every unique user. Basically, there is only one profile for whoever login in the program.

3.3 Statement of outcomes (Login and Registration User Story)

* The “Login and Registration” function works perfectly and smooth, I debugged dozen of times, and test every possibility that might cause the program compromise. Turns out, everything works fine.

3.3 Names of Implementer(s) (Login and Registration User Story)

* Hank Zhu

3.3 Design, Requirements, Test Plan (Help Window User Story)

* Build a panel that would list helpful information for anyone new to the product. Provide paragraphs for each frame to explain functionalities. This should be contained to a separate frame and should be closable by the user. It is implemented by a button on the homepage that creates a frame to populate.
* clearly organize points to allow future adjustments to the help window
* Minimal functionality suggests that the only desire is to have the window pop up and close without incorrectly displaying text or impacting the program on its close.

3.3 Implementation Details and Issues (Help Window User Story)

* While working on the frame, the panel was not visible from all machines. This was corrected by sharing the file with teammates to have them view it; while not the most efficient method it allowed the team to give their input on the design. The panel now effectively populates the pop-up frame consistently. A grid layout way used as a vertical box layout after the latter wouldn’t correctly compile with the rest of the product.

3.3 Statement of outcomes (Help Window User Story)

* The window does display information about the other panels and their functionality in the product. The code is clean enough to add more information sections in possible future iterations.

3.3 Names of Implementer(s) (Help Window User Story)

* Spencer Russell

3.3 Design, Requirements, Test Plan (Save User Story)

* All of the saved stocks are posted for the user to see in the main menu, under all of the user information. The design for the save function, is allowing the users to add the stocks that they like and also seeing all of the stocks that they have bought. All of the bought stocks are stored in a text file, once the user pushes the add button with the stock information the program will write to the text file with all useful stock information. In the main frame of the project the users can see all of the added stocks. The program reads the text file that has all of the bought stocks. The program reads the text file and makes the stock names into buttons that can be clicked to go to the stock graph page. The design of the buttons has the stock name, how many stocks the user has bought, and the percentage increase or decrease from when the user has bought the stock. If the percentage is above zero then there would be a green arrow and a red arrow if the percentage is less than zero.
* The requirements for the save function is that the program needs a way to save all the stocks that the user has bought. The group found that a text file costs less time than using a database. Another requirement is that reads the text file and prints out stock information for the user. When printing out the stocks into buttons, the team thought that the user wanted to have more information about the bought stocks. The buttons had the percentage change and how many stocks the user has bought the same stocks.
* The test plan for the save function is testing everytime the code has changed. There was a test plan for writing the stocks, reading the stocks, placing stock names into the buttons, and showing the correct information of the stock into the button for the user.

3.3 Implementation and Issues (Save User Story)

* The group implemented a text file instead of using a database. Most of the implementation was quick and easy, until the stock button had to display the percentage information for the user. At first, the program wrote the stock name then the price of the stock when bought. This caused trouble down the road because the when reading the stocks name first, the program executed the code that revolved around the stock information. That information dealt with the percentage change, so the code needed the price first. The price was equal to zero at the time, so the first stock that was read had a 100% decrease. The user would not like that error because they spent money on the stock and they do not want to be given false information. This issue was solved by printing the price of the stock first, then the name because the price of the stock would be the correct information and give the user the correct information.

3.3 Statement of Outcomes (Save User Story)

* Saving works fine, the bought stocks gets written into the text file with no errors. Reading the stocks and displaying the information works fine. The only thing that still needs to be worked on, is when adding the stocks the program comes back to the main menu but the frame has not been refreshed, so the user can not see the recently added stocks. The only way to fix that so far is quitting the application and rerunning it.

3.3 Names of Implementer(s)

* Patrick Gallagher

3.4 Integration Testing

* The testing that went into this sprint was having the password function as the main class that runs first and also after implementing to everyone’s code to see if there were any errors. The second integration testing was how the save function would work with the bought stocks and the search stocks. There were some errors in the search stock, but after debugging the code everything started to work fine now.

3.5 Scrum 3 Retrospective

* Spencer’s Perspective:
  + I didn’t have much free time during the final sprint, luckily my teammates covered the more challenging user stories. I primarily focused on the help window which populated the frame implemented by the button Patrick put on the main menu. We, the Nirds, improved our communication during the final sprint. Emails were sent more frequently, and meeting times were more relaxed to allow everyone to work at their own pace. While I am still not the strongest coder on the team, I am pleased with the product we created as a team.
* Tianyi’s Perspective:

The situation was a little messy when we started thi Sprint. We had

unfinished user stories from previous Sprints and we have difficult new functionalities to work on. For me, I had met three major obstacles.

First, It took too long every time we tried to get stock data from the API It became so annoying that I came up the method that preload the stock data, which fixed the defect on some level. Second, we couldn’t

successfully switch between different graphs, we solved it by changing the graphic components from instance variables to local variables.

The last problem was the buttons generated by the search function can only call the graph for the last stock of the result list. We fixed it by using ActionEvent e.getSource(), to recognized the button and find it’s equivalent stock.

* Patrick’s Perspective:
  + In this Sprint, I worked mostly on the save function. The save function was a step up to the last sprint user story, the user story was adding the function of reading the stock and displaying the graph when the button has been clicked. After completing this sprint, I am very proud how the save function turned out to be. Even though I would like to add some new functions into the save function, like the remove function. But we did not have enough time to implement the the function. Our team worked the hardest in this sprint and we communicated much more often than other sprints. I think that everyone worked on something very important in this sprint and they helped each other when there was a problem during the sprint.
* Hank’s Perspective:
  + In this Sprint, I worked on the password & registration function, which help user protect their personal information into a new level than ever before. It tooks me tremendous of time to figure out how to synchronize the data from password panel and registration panel, because password field is encoded text file, which can’t be written or read by other function. I end up put the two function into the same class and solved issue.
  + I also worked on modifying the search function, fixed bugs when connecting to different panels.

3.5 Product Owner Statement:

* This final sprint did not cover all of the user stories originally outlined by the Nirds. However, this program does include most of the user stories, as well as all of the outlined functionality. There was mentioning of a comparison tool to see differences between stocks, however that could still be done manually by the user whereas collecting the information requires the automation. Considering the previous owner statements during the course of this assignment, this sprint wrapped up loose ends to create a satisfactory product. I am also pleased to see the product devoid of advertisements as briefly mentioned some of the team members.

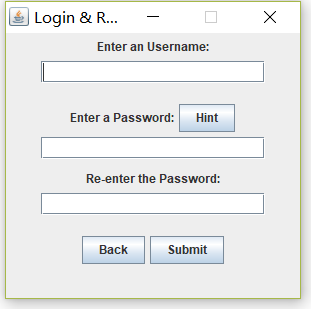
3.5 Scrum Master Statement:

* In this Sprint,we have finalized most of functionalities, including the login/register interface, the help window, the trading function and the multiple graphs. During the process of development, we have encountered some bugs caused by bad coding skill and habit, Specifically, the password verification system was not working because of the if statement logic was wrong; the graphs didn’t show up when clicking on the button on different time period, because we initialized the graphic variables in the wrong place, and the price trends line was drew on the wrong coordinates because the coefficient was miscalculated. Fortunately, we fixed the errors by revalidating and refactoring our code together. I think every team member has learned a lot in this group work.

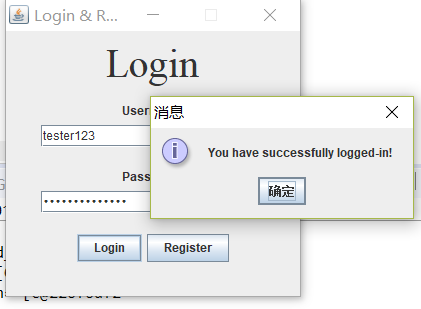
Final Report:

Description of Demonstration with screenshots.

* Register: This function is very helpful for people who want more privacy when using this stock ticker, for simplicity purpose, I did minimizing for the register requirement, which means you can register with any kind of password combinations. In this case, user has option to decide whether to set up a easy password or difficult password as the way they want. Take my self as example, for some accounts I use frequently, I hate to memorized a hard password, but the registration part did not give me a option to set up the password the way I want. After the password was registered, it will store into a local database file, which is not very secure. For the future improvement, I could connect the password field to online database, where user can login to their account to any computer that has this stock ticker program and connected to the Internet.



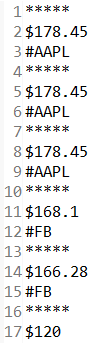
* Login: login function is relative easy to set up, all I need to do is create an action listener for “login” button, and tells program to read a local file and compare the data with user input when user click on in. I am glad there is a “JPasswordField” exist in Java application. Because I was creating my own function that reads user input, and mask it with the “black dots”.



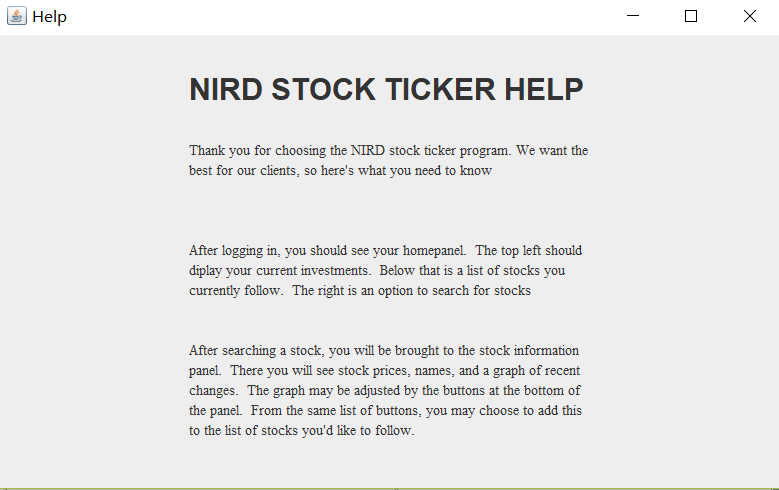
* Search: search function is the function I spent majority of my time of this project. I first created layout in a separate pop-up panel, which was looks so professional and neat, but for coherence purposes, I have to allow other members of my group to insert the search function inside the main panel, which is not prettiest looking panel, but it does the job. When comes to how the search function works, there are so many to talks about. Firstly, I did the ignore the case-sensitive function, which helps user to have more abstract search. Secondly, it's the filtered search, the stock name database is not as organized as we can see below, I created a function that only pick the important word for each stock name shows up, give the user just enough information to decide which one to pick, you can search by acronym of the stock or full name of the stock, if you don’t know how to spell the stock name, don’t worry, you can just type what you remember, the search function is going to find all the stocks that related to the information you entered. Thirdly, it is the scrollbar, is it nice to have a scrollbar in the panel? Lastly, it is the local file that generated by API, if I directly get information from API server, it will take forever each time you search something, but put the stock name database in the local file, it will improve the speed of searching thousand times faster, which helps user use the stock ticker more efficiently.



* Save: The way how we saved the user stocks was once the add button was clicked on the program writes into a text file with the stock name and the current price when user bought that stock. The picture below shows the text file when the user buys the stock. The program writes the current price when bought and the stock name. Once the text file is written, the file gets read again with all of the added stocks information. The main menu, the picture above, has the buttons under the Total Stocks. Those buttons are added when the text file has been read, the buttons shows the user what stock they have bought. The number next to the stock name is how many stocks they own, instead of printing one button for each stock, the program prints only one button but tells the user how many stocks with the same name they own. The percentage is calculated with the current price and the price when bought. Then the user is shown by a green arrow or a red arrow if the percentage is positive or negative represectively.

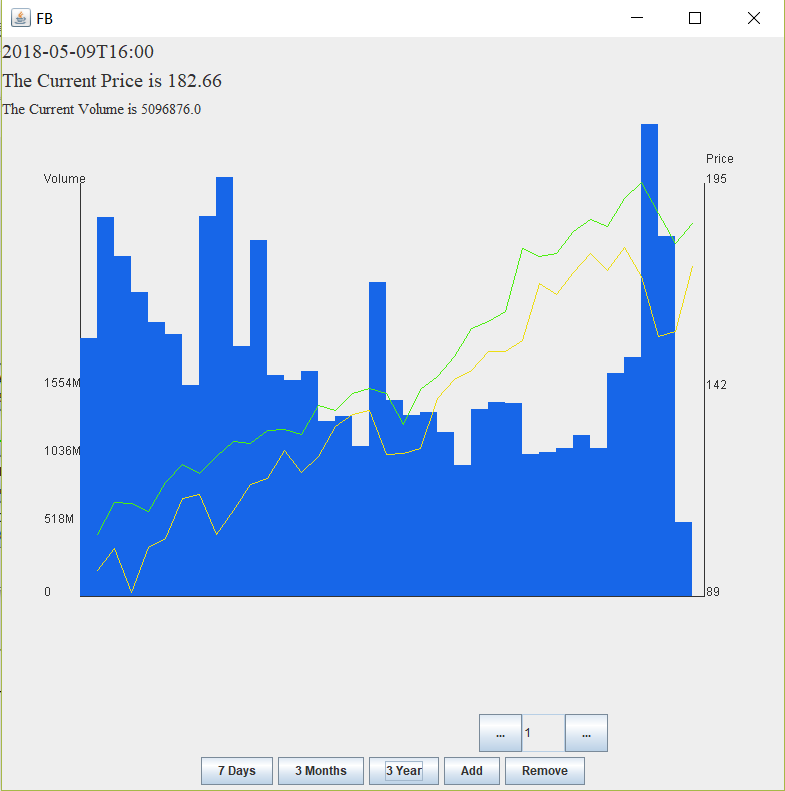


* Help:



A simple frame filled with a JPanel of information about the other panels and their functionalities. Only one of these will ever exist at a time, so there is less concern with monitor cluttering. It is set up to easily add new sections with the addition of new content.

* Stock Information Frame:



The current time, price and trading volume are displayed on the top left corner. The Graph shows the price and volume trend of Facebook over the past three years. The blue bars indicate the volume at each time nodes, the green line indicates the high price, and the yellow line indicates the low price. On the bottom you can switch to other graphs of time periods of past seven days and past three months. To save and purchase this stock, first select how many shares, then hit add. The remove function is not implemented yet. The user can have multiple stock frames pop up at the same time.

Further Development:

* There are two major user stories that we didn’t accomplish, the remove function and the compare the same stock from two different years. During the sprints, we made trade off to focus working on more important functions. These two functions are not the key feature of this software, but we will implement them if time allows.

Project Retrospective:

* As a group, the most important thing we have learned is that communication is the key to successful teamwork. In the beginning of a Sprint, the Scrum Master should properly allocated the jobs to the team members. Individuals should try their best to finish their job, and if one can’t solve their problem, it is better to consult with the group than delay it to the last day before it is due. It is also necessary in order to keep everyone on the same page. It’s the Scrum Master’s responsibility to check the progress of each team member during the development to make sure the group will not be behind schedule. When the group feels like can’t finish all the goals, the members should discuss together to make a trade off. The product owner is also not a position to take lightly, it is more than just stating whether or not a user story is met, but if it is a usable and desirable product.

GitHub Link:

<https://github.com/PGallagher12/StockTicker>