***STONKS***

A project realised by:

Hamza Shahid

Julian Ngong

Lucian-Gabriel Rusu

Matthew Nagy

Stefanos Charakidis

The whole reddit meme community

**Overview**

**The client**

Our client, CSS, is the Computer Science society at the University of Bristol. They help making being a student fun by hosting many events/activities you can engage in alongside your course. Over the years, they have successfully hosted many competitions such as Game Jams, Hackathons and Capture The Flags. These events are often sponsored and catered with drinks/snacks, even breakfast as they can last up to 24 hrs.

Our main contact is Daniel Jones, the webmaster for CSS. We have discussed with him about the needs and requirements for the program we’re making.

**The Application Domain**

As mentioned earlier, there are currently no events held that help students delve into their

Interest in algorithmic trading. This event would allow students to get involved in trading and

do so in a manner in which they can directly compete with their peers and have some fun

alongside.

**The Problem**

Due to the coronavirus outbreak, there are severe limitations to events that can be held in person according to the university’s guidelines. Though currently there are events that help students nurture their skills in a vast range of subjects, there is nothing to do with algorithmic trading.

Our program would allow users (members of CSS) to take part in designing and updating on simulated stock markets.

**Our vision for the product**

We aim to create a system that is easy to use for both the pupils, who will be creating/updating the trading algorithm and carrying out the trades, and the CSS committee members who will be managing the program and analyse player performance to determine winners.

Since the two main users here would be the pupils and the committee members, they will both have a different set of requirements and features. We aim to create a system that gives easy access of relevant features to the relevant user.

**Requirements**

In order to make sure we provide our client with a final product which best matches their criteria, it is highly important we make sure we meet all necessary requirements. We must identify all the different stakeholders of the system and what they need from the program so we can then specify the requirements for the system.

User Stakeholders

The CSS

The Computer Science Society is hosting the entire event therefore they will be using the system to manage the program and analyse the performance of the players to determine the winners of the event. Their key interests would be the ability to:

* Add students to the game as players
* Access all the players accounts so they can track all their earnings/score/trading history to ensure a winner of the game can be determined.
* Have access to a leader board clearly showing all students positions against one another
* Let the program run on a clear and suitable website which allows the stock market to assessable to all players from any of their devices.
* Have the program simulate stocks in the most realistic way possible via having the stocks affected by a news feed, other players trades, computer bots etc.
* Access information about the programmed bots to allow the control of bots which will make automatic trades to fluctuate the stock market

Pupils

The pupils are the main users of the program. They will be the ones creating accounts to complete the trading of stocks based on information given to them by the news feed which effects the relevant stocks. Their key interests would be the ability to:

* Have a clear and easy to use UI to which they can complete all necessary actions to play the game.
* Create an account on the website to which they can login to play the game whenever they want whilst saving all their trading progress
* Have access to an easy to use news feed which is constantly updating and delivering new pieces of information on the Stonks companies which they can use to influence their trading decisions.
* Access a list of all the Stonks companies available to trade their stocks with
* Easily access and view all their previous transactions and current balances to gage how well they are doing in the game
* Access an easy to read the leader board showing the ranking of all the other players in the game to compare their success against others.
* Access information about each companies’ stocks in the form of a graph for each companies showing their stocks history.

User stories

The use of user stories will enable us to more deeply understand how players will be using the system. We have created a 3 user stories for each of the stake holders, focusing on a range of different aspects of our system, allowing us to have a definitive understanding on how the program will be used.

Css :

* As the leader of the CSS, It's important that the system can be accessed by all members of the CSS via a nicely designed and easy to use website.
* As the leader of the CSS, I should be able to view the information on all the players including their trading histories, current balance, position on the leader board. Information such as all players trading histories should only be viewable to me and not the other players.

Player :

* As a player of the game I should be able to create an account which I can login too at any time to play the game and complete any necessary changes
* As a player of the game I should be able to easy manoeuvre around the website and complete any tasks that need to be done.
* As a player of the game I should be able to have access to the news feed and view the graphs if the stocks for all the companies at easy so I can come to a justified decision before completing any trades.

In the below table we have listed all the functional requirements this project will have, along with the reason they are important and their level of priority. These requirements denote the requirements that are important to ensure the platforms success.

|  |  |  |  |
| --- | --- | --- | --- |
| Req No. | Description | Justification | Priority (1-10) |
| 1 | Web Based Application | To allow the students to access the game, it must be developed as a web application so multiple users to play live at the same time. | 10 |
| 2 | Players have access to view their earnings | To allow the students to know how much money they have so they know the appropriate trades they can undertake. | 9 |
| 3 | Players have access to view their trading history | To allow students to see what trades they have previously done ensuring they can come to an informed decision on their next trades |  |
| 4 | Players each have a score | To allow a ranking between all the players of the success of their trading so far so a leader board can be created, and winner can be determined. This will be determined via the metric of their balance and the value of their stonks. |  |
| 5 | Players have access to the leader board | To allow students to see how they are doing compared to their competition, so they know how much more they need to achieve to win. |  |
| 6 | Bots are utilised to complete trades automatically | To allow the stock market simulation to be more realistic, programming bots to complete trades keeps each stock more volatile and less predictable, increasing effectiveness. |  |
| 7 | Stonks value changes | To allow each Stonk to change its value. This will be done by assigning it with a length of time a change will happen in and changing its value incrementally over this period until reaching the intended value. |  |
| 8 | The Stonk market contains a list of all the stonks imported from a csv file | To allow for a data base of stonks and their associated information to be easily edited and changed before being imported into the program for it to be utilised. |  |
| 9 | There is a news feed providing players with information about each company | To allow for a more accurate representation of a stock market with information being provided on events happening to each company which could affect its associated stonks positively or negatively. |  |
| 10 | Each player can create a secure account to access their account | To allow for secure login into their account so no other third party can access. This ensures they can make changes, save their progress, and come back later to do more. |  |
| 11 | Stock market is constantly being updated | To allow for a more accurate representation of the ever changing nature of the real world stock market. |  |
| 12 |  |  |  |
|  |  |  |  |
|  |  |  |  |

In the below table we have listed all the non-functional requirements along with the reason they are important and their level of priority. These requirements denote the requirements which help improve the programs effectiveness but are not vital for the program to work.

|  |  |  |  |
| --- | --- | --- | --- |
| Req No. | Description | Justification | Priority (1-10) |
| 1 | Clean and simplistic website design | To ensure lack of confusion when navigating through the website and improving enjoyability of playing the game through an aesthetically pleasing design. |  |
| 2 | Website elements scale correctly for all display sizes | To ensure the layout still appears clearly for the large range of different screen sizes each player may have on their computer. |  |
| 3 | Stonk changes including random fluctuation | To allow the volatile semi unpredictable nature of the real life stock market to be more closely imitated by the program by using randomised values. |  |
| 4 | Help menu/ initial tutorial | So each player understands how the stock market works and how to play the game, ensuring enjoyability and lack of confusion when using the program |  |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |
| 8 |  |  |  |
| 9 |  |  |  |
| 10 |  |  |  |

**Ethics and Security**

Data collection

Stonks Project does not collect any sensitive user data as it does not require any personal or important information to access our virtual stock market.

There are no cookies required to access Stonks. Data protection is in compliance with GDPR\* and with University Ethics regulation.

Any action that is against the UE regulation and the GDPR is not acceptable.

The data we collect are the following:

1. A username and a password that each team needs to send to the bots in order to be confirmed as participants, which are safely stored and are not shared to any third parties outside of the Stonks project.
2. There might be a feedback section which we will collect data from users and use them for future corrections and improvements.

Consider, also the following:

* The names of the companies of the “Stonks Market” do not reflect reality and are used only for entertainment purposes only.
* Any name, image or currency is fictitious and should not be treated seriously.
* Any events in newsfeed is fictional and they do not have any impact on reality.

\*GDPR = General Data Protection Regulation

**Architecture**

The client wants a simulated stock market that can be run on some server. Multiple teams should be able to write and run trading programs (bots) that can interact with the server using HTTP over the internet, with the purpose to buy and sell stock, thus increasing the teams score (The total value of the assets they have at hand). The end goal of each team is to have the highest finishing score. To aid with this, there should also be a website where members of CSS Bristol overseeing the event and the various teams can see graphs representing the history of the stocks on offer, a leaderboard that shows the current positions and scores of all of the teams, and also a history of transactions made by the team’s bots.

The stock market will be written in Java using the SpringBoot framework, which will allow us to use other plug ins like Thymeleaf (to interact with/ act as the website) and REST, which will make it easier for the student written bots to interact with the server. The application will receive incoming HTTP requests and use those to update the state of the program, thus allowing the bots to send and receive information to/from the server.

The website will be written using HTML/CSS/Javascript, using JSReact to show graphs showing a stock price over time, with the javascript dynamically updating what graph is being showed depending on how the user interacts with it. The Javascript can also interact with the RESTfull elements of the server to get up to date information both on what score each team currently has, and also the state of the stock market at that point in time.

The stock market will be made up as a single object, containing multiple stock objects it shall create by reading in a csv file. This will allow the exact stocks used run to run to be easily varied, and each stock be updated on its own. The stock markets prices will be changed by three events. A buy, a sell, and news. The news system will generate news about various sectors and will alter the prices of stocks based of what sector they were stated to be in as part of their entry in the csv. Buy and sell events will raise and lower the stock price accordingly. To add an element of randomness to the stock market (because even seasoned economists are surprised by the way it acts sometimes) we will create a number of java based bots behind the scenes to buy and sell stock pseudo-randomly. Certain bots will favour certain stock, with weights attached to buying and selling. This will allow predictable trends to emerge in the market for team’s bots to notice and exploit. This will also stop the situation from arising where completely random bots cancel out to a net 0 influence on all stock, as each stock is equally likely to be bought as sold and thus over a period of time no change will occur.

**Development Testing**

We are going to use the test class that is already created for our project by SpringBoot. We will make tests for every class, taking into consideration the most common, as well as the more unusual or marginal cases.

The main class of this project (and the most complex one) is going to be Stonk\_Market. It has functions that are going to control the market and return information about it. If all the test pass here we can say that also all the other classes are working great.

The functions that we will test are:

- updateMarket(): Should update all the stonks increasing or decreasing their value (In the test function we should verify if the values of the stonks are not negative, that the number of stonks in the list have not changed).

- getStonkWithName(string): Should return the stonk with the name given as input and should throw an exception if there is no stonk with that specific name. -table 1-

- createStonksFromCSV(file): Should populate the allStonks list with the information given in the CSV file (the test function should verify if all the stonks have been added and if their value is the one written in the CSV file). -table 2-

In order to make the explanation easier we will consider the list of stonks (allStonks) =

[Apple, Google, Microsoft, Sony] for this table:

|  |  |  |
| --- | --- | --- |
| Function | Input | Output |
| getStonkWithName(string) | "Apple" | Apple |
|  | "Sony" | Sony |
|  | "GGoogle" | throw exception |
|  | "Microsoftaaaaaaa" | throw exception |
|  | "Appple" | throw exception |

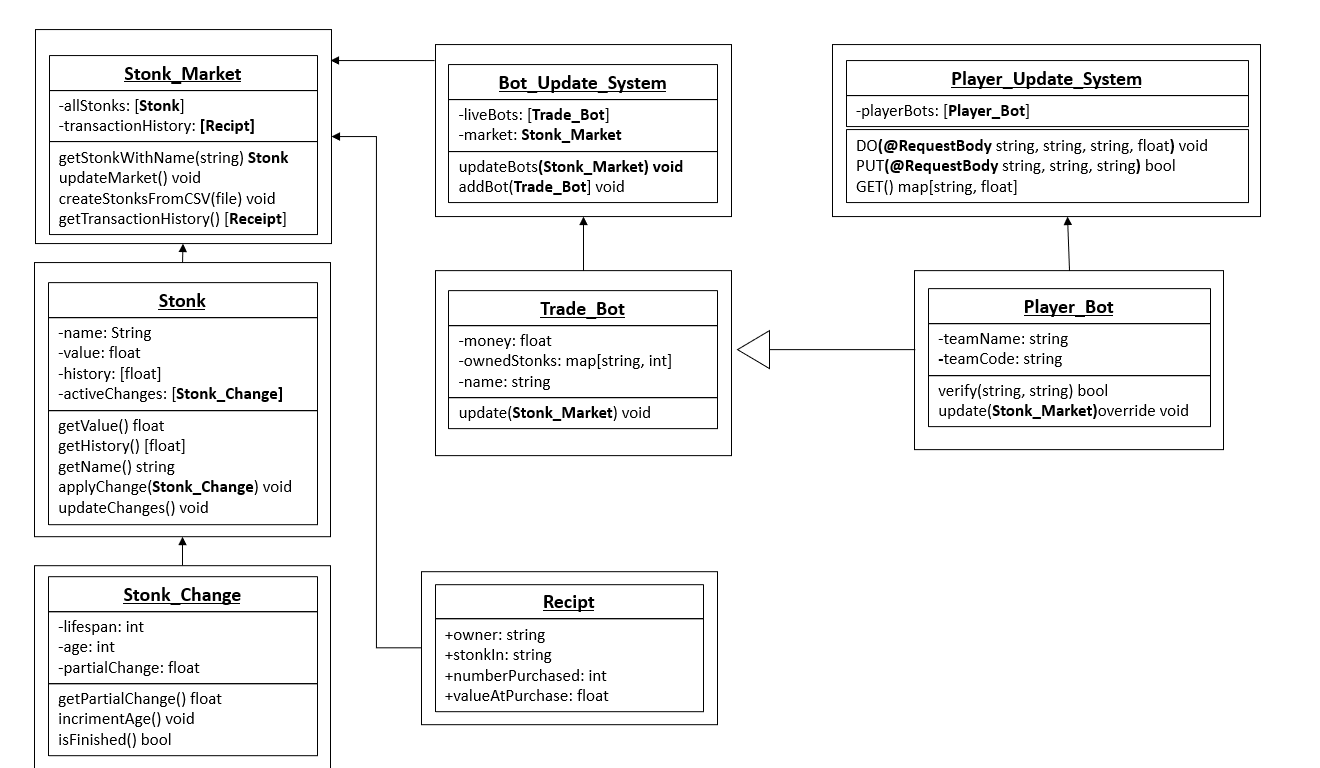
|  |  |  |
| --- | --- | --- |
| Function | Input (a CSV file -for simplicity we are going to write it here as list-) | Output (allStonks == L ) |
| createStonksFromCSV(file) | [("Apple",4)] | for L = [("Apple",4)] => true |
|  | [("Apple",4)] | for L = [("Apple",12)] => false |
|  | [("Apple",1),("Google",2)] | for L = [("Apple",1), ("Google",2)] => true |
|  | [("Apple",1),("Google",2)] | for L = [("Apple",1)] => false |
|  | [("Apple",1),("Google",2)] | for L = [("Google",2)] => false |

**Release testing**

For release testing, we will partially be using the test classes provided by spring, but may also have to write some custom code ourselves. In the process to creating the product we will be creating some Java based bots to interact randomly with the stock market, and these bots will have been tested as part of a unit test, therefore can be assumed to work. The environment any of these release tests will be running in will be the stock market itself running on an AWS instance, and then multiple bots connecting to that instance and attempting to trade with the stock market. The exact conditions will change from test to test, as different parts of the integrated system will be tested.

|  |  |
| --- | --- |
| Stress test | CSS Bristol has run many events in the past and estimates that an event utilising the Stonks project will attract around twenty teams to come and join the competition. By running a simulation of the event using thirty bots (an extra 50%) running multiple requests per second (random, so much faster than expected condition), we can see if the program would be able to withstand abnormal operating conditions. |
| Anti-cheat test | As all the competitors will be computer scientists studying both programming and security, there is the very real possibility of teams attempting to attack/game either the system or each other in an attempt to gain an edge. This test will check our guards against such attacks. This will involve bots that attempt to PUT multiple more bots onto the system, bots impersonating other bots with fake data, multiple requests simultaneously in an attempt to force undocumented behaviour, and any other possible weak spots identified over the course of development |
| Regular use test | Over the course of general use the server is not expected to have to deal with many outlying conditions. This test will involve connecting a small simple python-based bots (the language we expect competitors will use) to the server and running a full length 10 hour session of the stonk market. During the course of this, the website will be regularly checked, and the bots will keep track of what they expect their score to be at the end. After the simulation is over, these expected scores will be compared to the scores the system gives them to check if the system is running as expected. |

There is the possibility for one more test. Assuming a beta version of the stink market is finished by March, CSS Bristol has expressed their interest in running a short small scale event to gather data about what extra features could be added to the final release. This real world test of the program could serve as a test of the system as a whole up until that point Of course, all other tests will have been run on the product beforehand to minimise the chance of an error occurring during actual use, however such real world conditions may highlight any previously unnoticed flaws in the system.

**OO Design & UML**

This diagram is an example of how we see the MVP being built, with a fleet of java trade bots behind the scenes, buying and selling set stocks according to a sin curve, causing the pricing of the stocks to fluctuate accordingly due to a Stonk\_Change being passed to a Stonk every time a buy or sell action occurs. The Stonk prices are then all updated at set time intervals, and a snapshot of the previous value saved.

As SpringBoot is still new to all the members of the team it has been more challenging to visualise how the classes above will interface with Spring. An example of the progress made towards this can be seen in the Player\_Bot (PB) and Player\_Update\_System (PUS) classes. We believe that each team should have a team username and password that will be passed to the PUS, and then by checking the registered player bots the PUS can figure out which PB to update. This decreases the possibility for attacks between teams, where nonsensical commands can be sent to another teams bot as a form of sabotage, while also allowing the system to identify incoming instructions. The DO command would tell a PB to interact with a certain number of a certain stock. Negative would be sell, positive would buy. The PUT command would generate a new Player\_Bot and register it with the username and password passed as arguments. The GET command would return the player rankings, with the Bot name (Not the team username!) and their current net worth. This can then be sorted and displayed as a leaderboard by our website.

When any bot buys or sells on the Stonk Market it will call the respective methods (not shown above) to see if A: It can buy (Does it have enough money, and in a later release if there is a buyer waiting) and B: If it has enough money for the number of stocks it wants. If this succeeds, a Receipt will be made, which will both be passed back to the Bot and also added to the transaction history. The Stonk market will then create a new Stonk\_Change based on the value of the transaction, and pass it to the respective Stonk. This will cause the price to change; and increase over time if it was just bought and a decrease over time if it was just sold.

The main bulk of the backend has been modelled with very little Spring based interaction, because there are not many complicated actions we predict the system to have to handle. Any additional GET functions needed can be implemented by supplementary Spring classes the simply call the already available getter methods, and all complicated decision logic should be left to the bots the teams have programmed themselves. The only interaction they need with the project should be a simple “buy/ sell” command. By focusing on an easily accessible system with inbuilt flexibility (An overridable Update in Trade\_Bot and the freeness of the Stonk\_Change struct) it will be easy to adapt to any additional web based needs, which is a must considering our teams lack of web based experience. This will let us easily rectify mistakes we may make due to inexperience.