

# Options Trading Simulator - Coding Assignment

## Team 1 --- The Big Team

### Tom Busby, Blake Sweet, Justin Luttrell, Josh Luttrell, Tej Patel

## University of Kentucky -- CS499 -- Fall 2019

## November 24, 2019

Introduction

In our project, we created a web application that acts as an options trading simulator which monitors the portfolio performance of invested options over a period of time. Users will set up a portfolio and select an investment strategy and will then be able to observe the performance of their strategy. The portfolio performance is displayed graphically to the user where they can then view daily investments by selecting a day on the graph. This document will outline the implementation and testing of our project. From here, we will present these results to our client and peers.

1- Implementation

1.1- Source Code

The source code can be found on our Github repository link below. Our client requested that we use CodeCommit as our version control system but we have pushed our code onto the Github repo as well.

Link: <https://github.com/TejPatel98/cs_499_project>

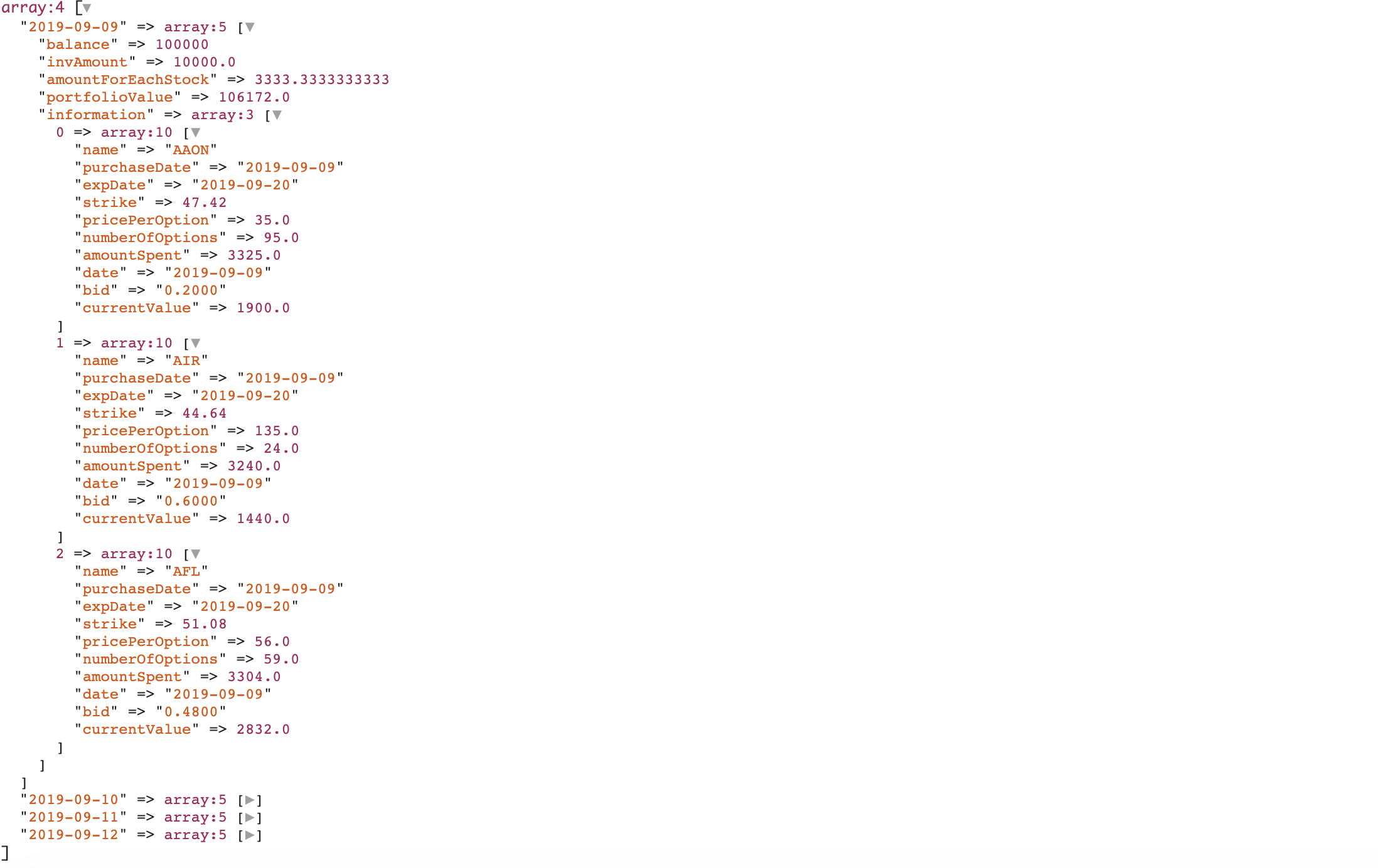
We have tried to closely follow the Architecture developed during the start of the semester. The architecture was developed by the team to make the workflow as smooth and streamlined as possible.

The first thing that the software does is to take user inputs. The user is prompted to provide their choice/input for the type of strategy, the scan option, Principle amount, Allocation Percentage, Maximum number of Trades, Minimum and Maximum Trade Length and the start & end dates.

The UI takes this values and feeds it to the backend. The Backend processes are mainly handled by the app/Http/Controllers/stocks.php. The stocks.php first chooses a list of stocks that are to be chosen from, based on the date. These stocks are derived from a scan program that our client developed. The scan returned would depend on the type of scan the client chooses to use. Once the scan program returns the list stocks, we choose stocks upto max number of trades that the user wants to have. That way we will have an upper limit of the max number of stocks that the user chose.

Those stock results are then used to generate a list of options that each stock corresponds to. The best trade for each option is selected by the code based on the max allocation per stock, the min-max trade length. This gives us a list of trades with an upper limit of maxTrades and a lower limit of 0. These trades are chosen for each day accordingly. This process is repeated n number of times, where n is the number of trading days in the provided date range. Hence if the start and end date had 12 trading days between them, we would end up with an associative array of length 12, with each element being a nested associative array, holding all the information for each trade.

All the data for the options is stored in a nested associative array indexed by the date. Certain rearrangements are done to make sure that there are no redundancies and all the data needed is being provided. During this process of rearrangements, certain values are removed, used for portfolio-value calculations and new values derived from them are inserted into the data structure.

 *That’s how the raw data, being fed by the backend looks.*

After all the calculations regarding the portfolio values and the cash balances, the entire data structure is returned to the front end to be converted into the graph form. The data provided to the front end is also a nested associative array and is indexed by date. Which would make it easier for the UI to display it with minimal code, since all it has to do is to go look up a date and grab all the data out of it.

The receiving of data from the backend and converting it into the graph form, along with showing the dropdown for each day is mainly handled by resource/views/results/results.blade.php. The front end is responsible for storing and working with all the data once the data has been worked with by the back end. It displays an interactive graph, the balance for each day, the net gain or loss over the chosen period and a dropdown when clicked on a specific day in the graph.

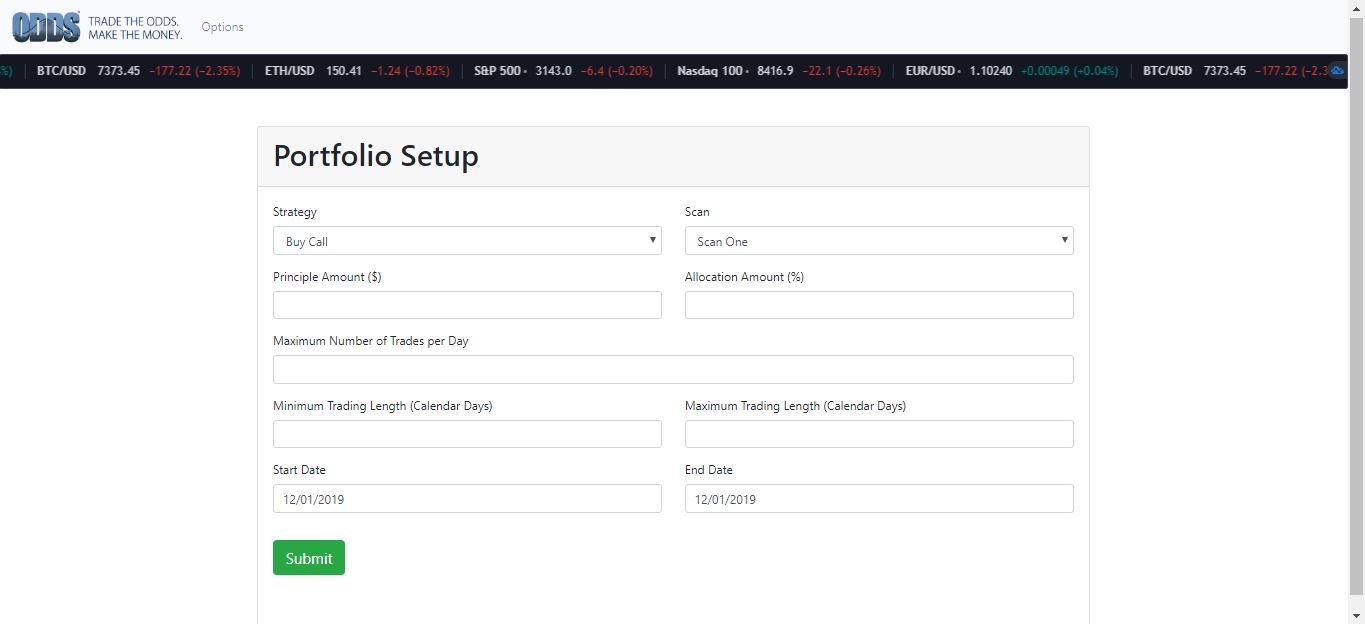
1.2- Quality Review

Once a majority of development was done, our team decided to hold a quality review to assess the functionality of our project and the structure of our code. We discovered that our code needs to be refactored. There were a couple of methods that contained all the code. From there we refactored the front end to modularize files to separate markdown from JavaScript. Once this was done, our front end code looked cleaner and was much easier to interpret. On the backend, our main file, stocks.php, contained only one large function. This was difficult to read so we broke the code into 4 separate methods that performed their own specific tasks.

Initially, we developed a skeletal portfolio setup page that was rather bare and unprofessional looking. During the quality review, our team decided we need to update this page. We implemented a stock ticker, added company branding and colors, reorganized the layout of the page, and update the navigation bar. Now, the setup page looks professional and is appealing to the user. Also, the results page intended to show the daily portfolio performance (by clicking on a day in the graph) through a modal. However, we changed this to display results at the bottom of the results page. Once a day is clicked on in the graph, the page automatically scrolls down to the results. This decision was made because a modal would not allow the user to view the graph once daily results were displayed. This results in a bad user experience, so by making these decisions in our quality review, our program is now more user friendly.

1.3- User’s Manual

The web application was intended to be a simple, user friendly, options trading simulator. As a user, you will first land on the portfolio setup page. From there, you will first specify your investment strategy. Our program supports Buy Call and Buy Put strategies. The user will need to select one of these. Next, the user will select which proprietary scan they would like to use. There are 3 provided to the user. Now, the user will specify a principle amount which is how much money is in your portfolio and an allocation amount which is the percentage of your principle amount that you would like to invest. Then, clarify the maximum number of trades per day which is the amount of options you wish to buy in a day and the maximum trading length (in Calendar Days). In other words, the maximum trading length is the longest amount of time your contract can be exercised before expiring. Finally, the user will select a start and end date that will act as the range of time investing will occur. Once the user has specified all this information, they will submit this form and will be directed to the results page.



Above is an image of the portfolio setup page. The user will fill in the form and submit that information.

If there is an invalid input on submit then an error message will prompt the user for correction.

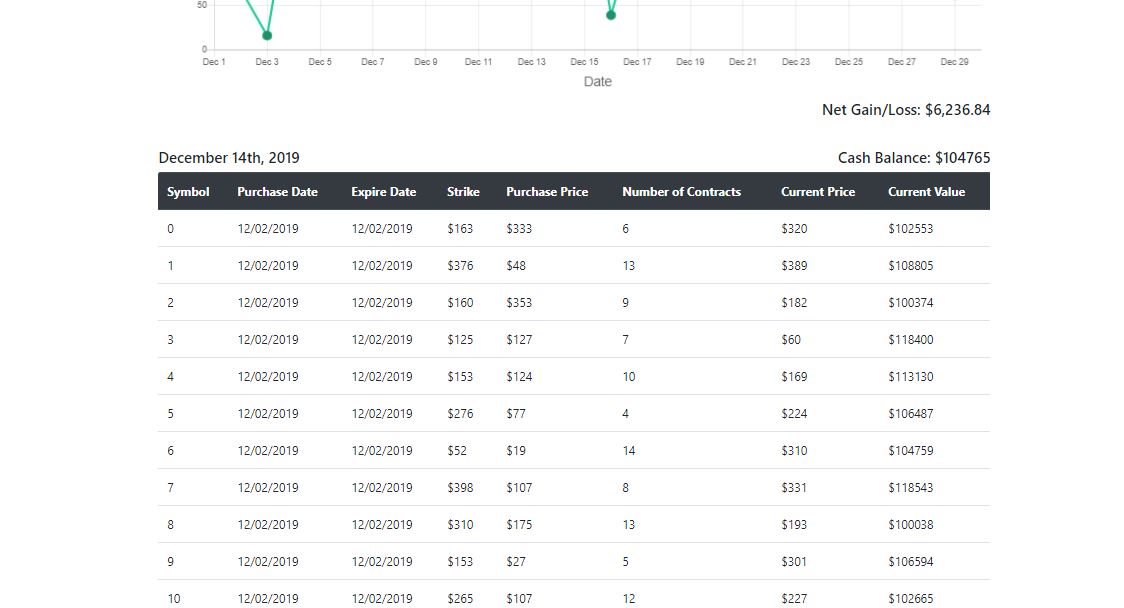
Once directed to the results page, users will see a graphical representation of their portfolio’s performance. The user can select any day within the graph to see the day’s breakdown of investments. When selecting a point, the page automatically scrolls down so the user can see their data.



The image above shows the results page that the user will see once they submit their form. The user

can select a point in the graph which correlates to daily investments.

In order to navigate back to the home page, the user can select the Options tab on the navigation bar.



Above is an image of the daily investment data that is seen whenever the user selects a day on the graph.

Our program is simple. We intended to make it as user friendly as possible. With this information, users should be able to easily navigate our web application.

1.4- Administrator’s Manual

**Server Requirements**

The application was built using the Laravel framework, version 5.5. Laravel 5.5 has these following requirements for PHP:

PHP >= 7.0.0

OpenSSL PHP Extension

PDO PHP Extension

Mbstring PHP Extension

Tokenizer PHP Extension

XML PHP Extension

Composer is required to install Laravel and its dependencies. To make everything simple composer should be installed so that is accessible system wide.

The live website uses Apache to process and serve web pages so the document root for Apache needs to be set to the public folder inside the Laravel project. Apache modrewrite needs to be enabled to allow pretty URLs. The Apache user needs to have write permissions to the storage directory and the bootstrap/cache directory otherwise the Laravel web application won’t run.

More in depth documentation for Laravel server installation can be found in their documentation:

https://laravel.com/docs/5.5#server-requirements

**Local Development Setup**

All of these requirements are fulfilled in the Vagrant Homestead virtual machine, which we highly recommend using for development.

On Windows hardware visualization is required as the virtual machine may not run otherwise.

Software that is required to setup the local development environment is:

* Git
* Vagrant
* Virtualbox
* OpenSSL

First Homestead needs to be clone to your home folder using the following git command:

git clone https://github.com/laravel/homestead.git ~/Homestead

Then Homestead needs to be configured, to do that open up a terminal (On Windows Git Bash is the easiest to use) and go instead the newly created Homestead directory. Then run

./init.sh

And this will do the first time setup required for Homestead. Next you will need to configure the Homestead.yaml file.

On Mac & Linux the Homestead.yaml file should look something like:

---

ip: "192.168.10.10"

memory: 2048

cpus: 2

provider: virtualbox

authorize: ~/.ssh/id\_rsa.pub

keys:

- ~/.ssh/id\_rsa

folders:

- map: /home/[YOUR USERNAME]/code

to: /home/vagrant/code

sites:

- map: example.local

to: /home/vagrant/code/project/public

databases:

- homestead

# ports:

# - send: 50000

# to: 5000

# - send: 7777

# to: 777

# protocol: udp

On Windows the Homestead.yaml file should look like:

---

ip: "192.168.10.10"

memory: 2048

cpus: 2

provider: virtualbox

authorize: ~/.ssh/id\_rsa.pub

keys:

- ~/.ssh/id\_rsa

folders:

- map: C:\Users\[YOUR USERNAME]\code

to: /home/vagrant/code

sites:

- map: project.local

to: /home/vagrant/code/project/public

databases:

- homestead

# ports:

# - send: 50000

# to: 5000

# - send: 7777

# to: 777

# protocol: udp

The two most important parts of the Homestead.yaml file is the folders and the sites part. The folders maps a folder on the current machine to a folder in the virtual machine. This allows you to edit code on the virtual machine without having to SSH into your virtual machine, allowing you to use your favorite editor.

Next you will need to generate SSH keys if you don’t already have them. You can use the command

ssh-keygen

Then just follow the prompts and the ssh keys will be generated. To use the default options you can just leave the inputs blank (This will work for most cases).

More in depth instructions can be found in the Laravel Homestead documentation:

https://laravel.com/docs/5.5/homestead

Vagrant should be ready to launch. In the terminal inside the Homestead folder in your home directory, type the following command:

vagrant up

Afterwards ssh into the virtual machine using the command:

vagrant ssh

Navigate to code/project and run the command

php artisan key:generate

This will generate a key for your application that will be used for user sessions and encrypted data.

If you encounter any other errors you can consult the documentation above.

**Project Specific Instructions**

These instructions are specific only to this project.

Because the database being used can only be accessed via localhost on the clients server, we setup an ssh tunnel from the vagrant virtual machine. From t

he vagrant virtual machine you will need to generate ssh keys and copy them to the clients server if there aren’t already ssh keys. You can use the same command as before:

ssh-keygen

You will then need to copy the public ssh key of your vagrant virtual machine to the clients server. You can use the command:

ssh-copy-id

If for some reason you cannot use ssh-copy-id you can manually copy the the public ssh key to the authorized\_keys file on the clients server.

Once the ssh keys are setup you can run this command:

ssh -i /home/vagrant/.ssh/id\_rsa -f ukfl2019@ukfl2019.donfishback.com -L 33060:127.0.0.1:3306 -N

You may need to change the command around to fit your credentials if they are different.

Next the web application uses the clients stock scanning application to select stocks to track and display. Because we cannot copy the application to our local environments (The application is an industry secret and should not be copied), we created a bash script that calls the actual program over ssh and gets the results of that program. The program is very simple and should be placed in vagrant virtual machines home directory.

#!/bin/sh

ssh -i ~/.ssh/id\_rsa ukfl2019@ukfl2019.donfishback.com './FishbackStockScanner '$\*

The program needs to be marked as executable with chmod otherwise the web application will fail.

Next the environment file needs to be setup. The project comes with a file called .env.example. Copy that file and call it:

.env

You will need the following entries

These are for the student database

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=[YOUR DATABASE]

DB\_USERNAME=[YOUR DB USERNAME]

DB\_PASSWORD=[YOUR DB PASSWORD]

These are for the OVS database

OVS\_DB\_CONNECTION=mysql

OVS\_DB\_HOST=127.0.0.1

OVS\_DB\_PORT=33060

OVS\_DB\_DATABASE=ovs

OVS\_DB\_USERNAME=[YOUR DB USERNAME]

OVS\_DB\_PASSWORD=[YOUR DB PASSWORD]

This is the path to the program we put in the vagrant home directory.

SCAN\_PROGRAM=/home/vagrant/FishbackStockScannerSSH

**Adding New Functionality**

Adding new functionality is pretty simple using Laravel’s command line interface artisan.

Making new controller is pretty simple. Just use the command:

php artisan make:controller YourNewController

And the controller will automatically be generated and put in the app/Http/Controllers folder. More information about controllers can be found here:

https://laravel.com/docs/5.5/controllers

Web routes are defined in the routes/web.php file. More information about routes can be found here:

https://laravel.com/docs/5.5/routing

Views are found in the resources/views folder. More information on views can be found here:

https://laravel.com/docs/5.5/views

The database is handled specially because the client provided raw SQL queries for the project. We used the DB facade to run raw queries.

More information on raw queries can be found in the documentation:

https://laravel.com/docs/5.5/database#running-queries

**Maintenance**

Since PHP 7.0 is not being supported anymore, eventually the framework will need to be upgraded. However the client is still running PHP 7.0 on their servers. When the time comes to upgrade Laravel you can follow the upgrade guide found here:

https://laravel.com/docs/5.6/upgrade

2- Testing

Our testing procedure involves two different levels which include unit testing and acceptance testing. Laravel provides an environment “out of the box” for exactly these purposes. It is also important to note that, Laravel refers to acceptance testing as “feature” testing.

2.1 - Unit Testing

Our unit tests consisted of four programs from the essential five in our program. The fifth function was “submitSelection” which is not viable for unit testing. Instead it will be referenced during the acceptance testing section.

The first of the four functions subjected to unit testing was the “getPriceHistory” function. This function was used in order to create the graph and tracking the history of the option. Given an option to reference, the start date, and end date as parameters of the function, this function should return an array that is a reference array. This refers to the elements in the resulting array being named by a value, in this case the stock reference number. For this reason it was essential to ensure that given a reference number, the correct number of data points for the price history were returned from the function. In addition, each of those data points should also be the correct values for the history. This resulted in the two unit tests for this function “testGetPriceHistoryNumDays” and “testGetPriceHistoryUsingCorrectDay”.

The next function used for unit testing was “getPortfolioValues”. Another essential function to our program, which returned the overall value of the portfolio containing multiple options. This function returned the combined asset value of current options as well as cash, also known as “buying power”.

The final function tested was “getSpecificDayValue”. This function takes a portfolio value from the previous mentioned function and a date, then provides a value of the overall assets for the day. This is a core mechanic of our project, since the tracking of this number over time allows us to achieve desired behavior.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Case** **#** | **Tested Function** | **Test Case** | **Pass Condition** | **Fail Condition** | **Results** |
| 1 | getPriceHistory | Returns number of days for a given option | Returns correct number of days for a given option | Returns incorrect number of days for a given option | Pass  ✅ |
| 2 | getPriceHistory | Return days for history | Return correct days of history for a given option | Return incorrect days of history or error | Pass  ✅ |
| 3 | getPortfolioValues | Checking portfolio value | Value is correct and expected | Returned null or incorrect data | Pass  ✅ |
| 4 | getSpecificDayValues | Get portfolio value for a specific day | Portfolio value is correct and expected | Returns null or incorrect data | Pass  ✅ |

2.2 - Acceptance Testing

Our acceptance testing was performed using a method widely known as black box testing. In this method of testing, the tester acts as if they are the user with no inside knowledge of how the program operates or what should be returned. Using this method, testing was performed on the submitSelection function of our program. This program is run when the user clicks on the “submit” button at the bottom of the portfolio setup page. This method consisted of 6 simple pass/fail conditions. For example, during the first test if the user was allowed to fill out the “portfolio setup” form and submit incorrect data and an error message was shown as a result, the test case was a success and documented in the table below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case** **#** | **Test Case** | **Pass Condition** | **Fail Condition** | **Results** |
| 1 | Incorrect data passed into  submitSelection($data) | Gives error message | Allows submission of incorrect data | Pass ✅ |
| 2 | Options navigation tab takes you to portfolio setup page | After clicking button, user is redirected to setup page | After clicking button, user is not redirected to setup page | Pass ✅ |
| 3 | User inputs calendar dates to begin a scan before 02/01/2015 | Gives error message to user | Allows user to enter incorrect date and submit scan | Pass ✅ |
| 4 | Submit button takes user to results page | Results page shown with graph and correct information | Error message shown from incorrect code | Pass ✅ |
| 5 | Hovering over a point on the graph displays the date | Correct date is displayed | Incorrect date is displayed | Pass ✅ |
| 6 | Clicking on a point scrolls the page down to display the daily data | Page scrolls down to the data | Page does not scroll, crashes, or gives error message | Pass ✅ |

2.3 - Testing Quality Review

Upon inspection for quality review, it should be noted that there are potential issues that should be fixed if chosen to continue with this project. A current issue with the project is that there is no indication of the format needed for the date. The program currently relies on google chrome to be used in order to automatically format the date in such a way that it can be submitted into the functions properly. A potential solution to this issue would be to implement a user interface that allows the user to click on a date (potentially with a pop up calendar) followed by the date being automatically entered into the field. This would avoid potential confusion for the user and ensure that the correct date format is entered.

3- Metrics

3.1- Size Estimate

|  |  |
| --- | --- |
| **User Story** | **Story Points** |
| Portfolio Setup | 13 |
| Date Range Option | 3 |
| Daily Portfolio Performance | 8 |
| Strategy Selection | 13 |
| Net Gain or Loss | 3 |
| Graphical Portfolio Trend | 8 |
| **Total** | **48** |

We have six different user stories, all with varying story points for a total of 48. Previously, our total was 58, but after completion of the project, we more accurately scored our story points.

3.2- Lines of Code

100 files

116 text files.

classified 116 files

Duplicate file check 116 files (105 known unique)

Unique: 100 files

110 unique files.

Counting: 100

28 files ignored.

github.com/AlDanial/cloc v 1.74 T=3.17 s (28.1 files/s, 9306.3 lines/s)

-------------------------------------------------------------------------------

Language files blank comment code

-------------------------------------------------------------------------------

CSS 7 1458 45 12616

JavaScript 12 2068 1760 8027

PHP 58 471 1185 1375

Blade 6 47 12 223

JSON 2 0 0 79

XML 1 1 0 30

Sass 2 11 11 25

Vuejs Component 1 2 0 21

-------------------------------------------------------------------------------

SUM: 89 4058 3013 22396

-------------------------------------------------------------------------------

3.3- Product Size

* Number of User Stories: 6
* Number of Unit Test Cases: 4
* Number of Classes: 2

3.4- Product Effort

See <https://github.com/TejPatel98/cs_499_project/wiki> for team meetings

* Tej Patel: 53 hours
* Justin Luttrell: 53 hours
* Josh Luttrell: 51 hours
* Blake Sweet: 40 hours
* Tom Busby: 55 hours

3.5- Defects

Currently, our program is functioning as we intended. There are only a few minimal defects at the moment and none of these hinder the complete functionality of the project. The first defect occurring is when we are displaying the daily options data. Right now, the data is only showing options that were bought on that day. However, we want to also display all options in the portfolio from the previous days as well. This is a very minor issue and will likely be taken care of before presentations.

To continue, our web application is not mobile responsive. Although all the data is there, the daily data chart is not organized neatly when sizing down to mobile. Even though this is a defect, it is not an issue that we think should be addressed since the program would only be intended for use on a desktop. Another minimal defect that occurs is when the user clicks a point on the graph for the first time. We intended for the page to scroll down to the daily data chart but this does not occur. This issue is not worth addressing because it only occurs for the first point that is clicked. All the user has to do is double click the first point. Because of this, we do not plan to address the issue.

4- Developer Notes

### Github Link

Project Link: <https://github.com/TejPatel98/cs_499_project>

Logs:

* Justin Luttrell:
  + <https://github.com/TejPatel98/cs_499_project/wiki/Developer-Logs:-Justin-Luttrell>
* Josh Luttrell:
  + <https://github.com/TejPatel98/cs_499_project/wiki/Developer-Logs:-Josh-Luttrell>
* Tej Patel
  + <https://github.com/TejPatel98/cs_499_project/wiki/Developer-Log-:-Tej-Patel>
* Tom Busby
  + <https://github.com/TejPatel98/cs_499_project/wiki/Developer-Log-:-Tom-Busby>
* Blake Sweet
  + <https://github.com/TejPatel98/cs_499_project/wiki/Developer-Log-:-Blake-Sweet>

5- Word Count

Tej Patel: 536

Justin Luttrell: 222

Joshua Luttrell: 948

Blake Sweet: 844

Tom Busby: 1067