

# Options Trading Simulator - Architecture

## Team 1 --- The Big Team

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

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

## October 18, 2019

# 1. High Level Design Architecture

This project is a web application that performs backtesting on options trading. The “Options Trading Simulator” is a prototype for the ODDS Online platform. The basic function of the program is to take in a set of data and evaluate a portfolio based on the performance of the selected investment strategy. This section discusses the high level design of the program.

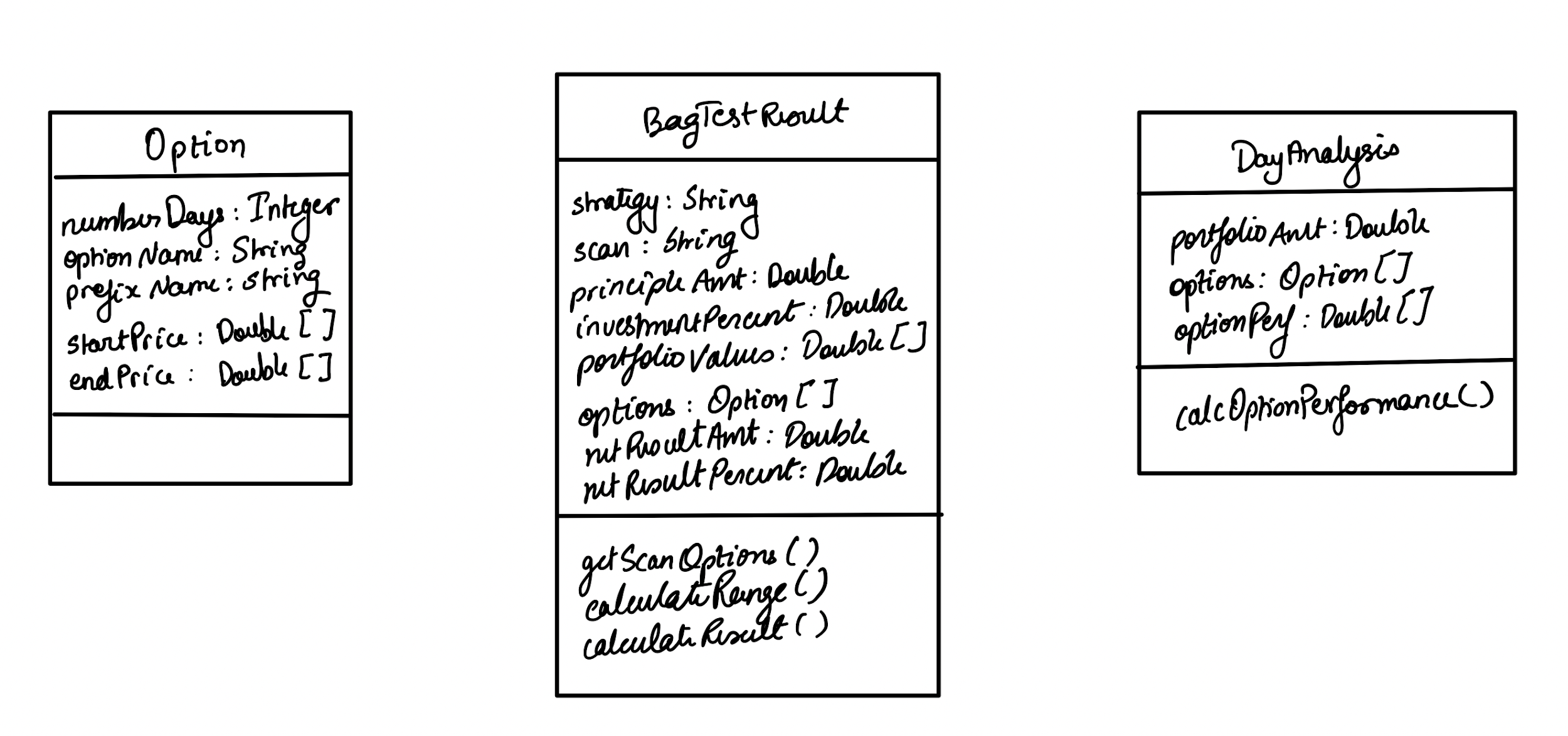
Below is the diagram for the high level design. The architecture is made up of both front-end and back-end components that work to provide the functionality of the backtest. The UI consists of an input form where users can set-up their portfolio. Once submitted, this information is sent to the backend where a scan program that outputs a list of options is run based on the data submitted to the form. The scan programs are provided by our client Don Fishback, in the form of a command line program. The output of the scan is a list of options that are sent to the backend processor. The backend processor will trim this list to be the top 5 options. The backend processor then queries the database to get the relevant data for each option. With the options data, the backend processor performs the necessary calculations for the back test. The calculations are then sent back to the UI to be displayed.



# 2. Detailed Design

### 2.1 Class Diagram

Our project will be object oriented, containing three separate classes. The BackTestResult class contains the appropriate members that will be used to store all the information. This class also contains three functions that uses the stored variables. With that information, the correct quantity of options and specific options are selected, the OVS calendar range is determined, and the calculations of the options over the range are performed. The Option class contains data types that describes what makes up an option. The DayAnalysis class is made up of information on what should be shown whenever the user wants to look at how their options are performing on any given day. This information is calculated by the calcOptionsPerformance() function which takes in the option’s start and end price for a certain day. These classes combined will make up our entire Backtest.



*The diagram above shows the three classes that will be used to implement the backtest*

### 

### 

### 2.2 User Interface Design

For our Options Simulator, we plan on having a design that enable the user to have control over what they select and see the results in a clean fashion. We plan on incorporating a graphical representation of the actual portfolio growth over a given amount of time. This way the user can have an idea about how the portfolio-value increased or decreased over time and when was a sudden difference in the trend. This way a good correlation can be identified between actual events in the past and the effect it had on the value of the portfolio.

When the WebApp first begins, the user is taken to the initial page, where he/she needs to input values for certain fields. This way the portfolio and the investment strategy can be tailored for each specific instance that the user uses the WebApp. The user has to choose the strategy that they wish to test for, followed by the scan option, the Principle amount that they wish to invest and test the simulator for, the percent amount they wish to invest from the principle amount and finally the start and the end date. After all of those have been selected, the user clicks “Submit” button to send all these inputs to the backend process.

Post submission of the inputs, the user is rerouted to a different page that is responsible for displaying the growth graph of the portfolio and the net gain or loss, for the provided date range. This second page is where the net results are displayed to show how the portfolio would have actually performed over the given date range, in the past. The graph is necessary to show how the portfolio would have performed in the past. Visualizing the growth or decline can help make informed inferences which could be helpful to evaluate investments in similar kinds of scenarios, which may arise in the future.

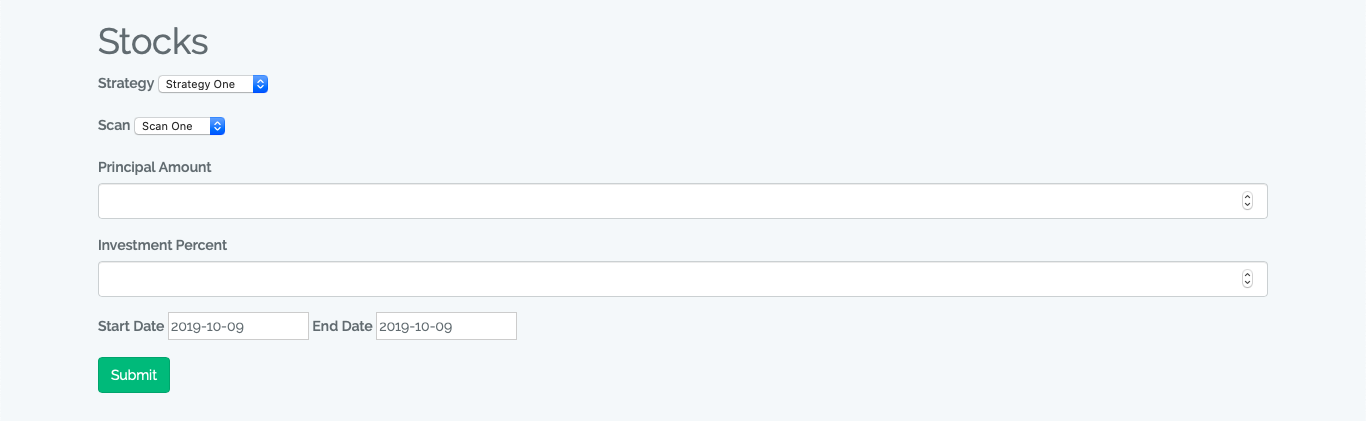
If the user wishes to see the performance of each option and the entire portfolio, then he/she can click on the graph. This would open up a pop-up window. This way the user can choose to see the portfolio performance and the respective options’ performance that specific day.

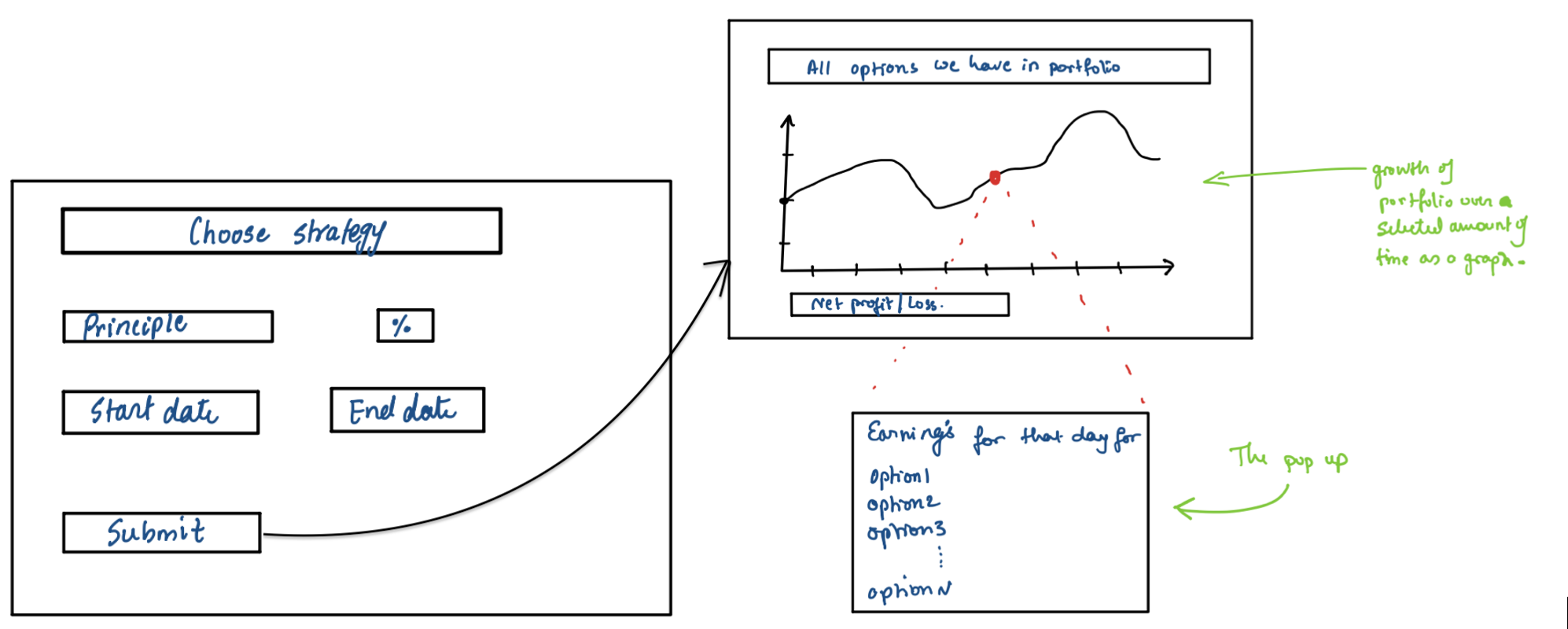
For instance if there was a major event that took place, we can look at the market trend for that day and find correlations between that event and the market trend. This can be very helpful in making inferences for future investments.

The user interface will be designed using the open source toolkit Bootstrap 4. In Bootstrap we are able to use predefined CSS classes that will give the site a uniform feel. Bootstrap’s layouts generally consist of containers. Inside those containers are rows. Finally inside those rows are columns. Bootstrap classes use CSS media queries to select which styles to apply depending on the width of the screen. That means that the website will be able to adapt to different screen sizes so you will have a good user experience whether you using the website with a large desktop monitor or a small iPhone 5.

We will have various places where a popup modal will need to be implemented. Well will be using Bootstrap’s javascript modal plugin to show the various pieces of information throughout the website.

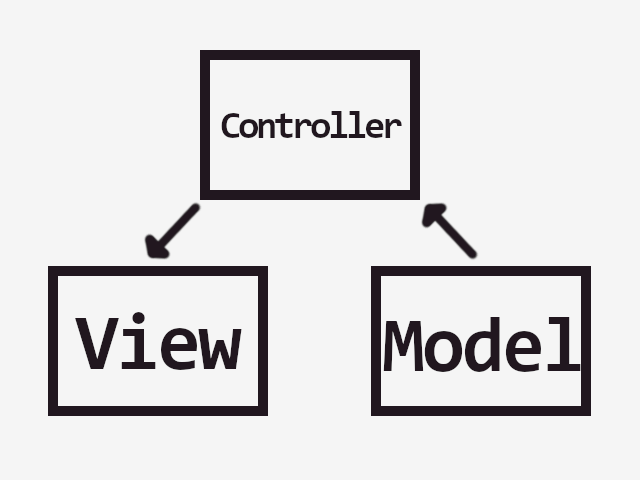
There will have to be tables of information that get displayed. We will use Bootstrap’s table classes to give a good experience on desktop and mobile. For tables with a lot of information, the mobile view of the website will be displayed differently. There will be expandable regions or modals to declutter the small screens.





*Depicts the three UI plans that the WebApp is going to consist of*

### 2.3 Design Patterns

This website will make heavy use of the Model View Controller (MVC) design pattern. The Model in this design pattern is a data structure manages data. The Model has getters which return pieces of data and setters which set pieces of data. For example a user model would have getters and setters for username, password hash, date created. The View is the part that the user will see. The view is made of HTML, CSS, Javascript, and any other front end components. The Controller handles all the logic in between the model and the view. For example logging into the website would need to get the information from the username model, verify passwords, and then send relevant information to the view to be displayed. 

# 

# 3. Testing

For our Options Trading Simulator; we are planning to use Selenium for the User Interface and PHPUnit tests for the backend processor. For every controller class, we plan on having a test class to verify that the values being returned from those class values are valid.

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Test Case Description** | **Test Data/Variable** | **Test Results** |
| get\_scan\_options() | Tests to see if our program gets the right options from the client-provided scan program | Scan type and date.  Example - “technology sector” &  “01/05/2015” | Correct quantity and predetermined options |
| calc\_range() | Tests to see if the date range matches with the actual date range, based off of Stock Market calendar (provided to us). | Start & end date.  Example - “12/01/2014” & “01/05/15” | The date range computed through the client-provided calendar is the same as ours. |
| calc\_netvalue() | Tests the net portfolio gain or loss after the end date | Start portfolio amount  Example - “100,000” | The correct net gain or loss. Which can be tested against the excel sheet already produced by the client. |
| calc\_opt\_perf() | Tests to see if we are querying the option performance from the database correctly. | Option name & Date  Example -  “Microsoft” & “01/03/2014” | The result should match the performance data from the database. |

# 4. Quality Review

### 4.1 High Level

After reviewing the high level design, we realize that the scan program portion could better fit as part of the backend process, instead of being called at submit. The backend process is acting as the controller and therefore it makes sense for this process to do the invocation of the scan program.

### 4.2 Detailed Design

#### 4.2.1 Classes

For the project, we are using a calendar provided by the client that only has market days. The current project plan consists of the user being able to choose the date, and the backend process invalidating the date based on the market-day-calendar. To improve this functionality, we can add a method that would not allow the user to choose and invalid date in the first place. The method can validate and disallow the user from choosing the date, so the process doesn’t get to submission and then terminated due to the error.

#### 4.2.2 User Interface

Looking over the UI we have implemented so far, we realize we need to change everywhere it says “stock” to “options”, since we are trading options. Also, once we determine the names of the strategies and scans, we will change them from the placeholder names that are present now.

An important piece to mention also is the graph of the options in each profile. We plan to implement a section in which the progress of the data can be easily viewed by the user in graph form. This will allow the user to asses on their own if each option is either being recommended or suggested to be switched for a better choice.

The user interface is lower on the priorities at the current moment as we implement the functionality, but plans to become a higher priority shortly after correct data can be verified through our program. In the future it would be nice to have a design that is adaptable to different devices such as mobile and improve overall aesthetics.

### 4.3 Testing

Considering we haven’t implemented much of the backend logic, we feel as though the test cases we have now will suffice. As we implement more of the backend code there will be more need for testing.

# 5. Metrics

# 5.1 Size Estimate:

|  |  |
| --- | --- |
| **User Story** | **Story Points** |
| Portfolio Setup | 5 |
| Date Range Option | 3 |
| Daily Portfolio Performance | 21 |
| Strategy Selection | 13 |
| Net Gain or Loss | 3 |
| Graphical Portfolio Trend | 13 |
| **Total** | **58** |

We have six different user stories, all with varying story points for a total of 58.

### 

### 5.2 Product Size

* Number of User Stories: 6
* Number of Unit Test Cases: 4
* Number of Classes: 3
* Lines of code:

108 text files.

102 unique files.

70 files ignored.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Language | files | blank | comment | code |
| CSS | 7 | 1458 | 45 | 12616 |
| Javascript | 8 | 2024 | 1751 | 7780 |
| PHP | 58 | 407 | 1125 | 114 |
| XML | 1 | 1 | 0 | 30 |
| SASS | 2 | 11 | 11 | 25 |
| ASP.Net | 1 | 0 | 0 | 33 |
| SUM: | 77 | 3901 | 2932 | 21618 |

### 5.3 Product Effort

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

* Tej Patel: 19 hours 10 minutes
* Justin Luttrell: 19 hours 10 minutes
* Josh Luttrell: 19 hours 10 minutes
* Blake Sweet: 15 hours 50 minutes
* Tom Busby: >20:52:20

### 5.4 Defects

Currently, the scan and strategy choices are not appropriately labeled. Right now there are generic input options. A potential defect that may appear is inaccurate calculations reported to the user. If the graph displays inaccurate results, it may lead the user to become unsure of the given decision to keep or change the option it follows. For this reason, hand calculations will be made and compared against the results calculated by the program.

# 6. Developer Notebook

### 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>

# 7. Word Count

Tej Patel: 505

Justin Luttrell: 381

Joshua Luttrell: 327

Blake Sweet: 179

Tom Busby: 314