**Team 3**

**Software Engineering Final Project**

**Nicole A. Cill**

**Yuwei Zhou**

**Keonta S. Hafley**

**Keith Estrella**

**Brian Yao**

**11th December 2019**

**Written by Nicole A. Cill**

**Version: (3) Created: December 2nd, 2019**

**Table of Contents**

1. **Introduction**

**1.1 Purpose**

* 1. **Scope**

**1.3 Definitions, Acronyms, & Abbreviations**

**1.3.1 Table**

**1.3.2 Definitions**

1. **Overall Description**

**2.1 Product Perspective**

1. **Requirement Engineering**

**3.1 - Software Requirements Specifications**

1. **System Modeling**
2. **Architectural Design**
3. **Software Design and Development**

**6.1 Daily Update Routine**

**6.2 Team Features**

**6.2.1 Team Feature 1 – Statistical Graph**

**6.2.2 Team Feature 2 – Betting Game**

1. **Database Design**
2. **Project Management**

**8.1 Progress Log**

**9. Testing Plan**

1. **Introduction**

Initially, Team 3 struggled to finalize an idea for our final project. We regrouped and reprogrammed multiple times until we had an ideal and fully functional system. The system interface will retrieve stock information from a database. After collecting the data, we will use a variety of features to display and interpret the information.

* 1. **Purpose**

This project allowed us all to work in a collaborative environment in a software engineering team. Together, we were able to have a better understanding of the IEX Cloud API and how to utilize it. Learning how to work with the command calls of IEX Cloud API, made us more comfortable with using RESTful API’s as well. Our team was able to precisely and clearly present the data for our [hypothetical] clients.

* 1. **Scope**

The software we developed will use IEX Cloud API to obtain the stock information. This information will be recollected/ updated on a daily basis. We will be receiving data from three stock symbols: (1) rfem, (2) googl, (3) aple. Our database will collect about twenty parameters, such as: basic company info, a description of the company, which stock exchange the company uses, etc. The software will also report calculations of the prices from the data searched and retrieved from the database. Based on the data collected over a 30-day time span, we will also provide statistical graphs and a betting game for the users.

* 1. **Definitions, Acronyms, & Abbreviations**

All of the definitions, acronyms, and abbreviations that will be seen throughout the document will be listed here.

(\*) label means the feature was not tested but is being defined for clarification.

**1.3.1 Table**

|  |  |  |
| --- | --- | --- |
| Acronyms | Abbreviations | Definitions |
|  | MVC |  |
|  | JSPs |  |
|  | API |  |

**1.3.2 Definitions**

* Acceptance testing - level of software testing where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.
* Agile method iterative development methodology, where requirements evolve through collaboration between the customer and self-organizing teams and agile aligns development with customer needs.
* Bug triage - evaluate, prioritize and assign the resolution of defects. The team needs to validate severities of the defect, make changes as per need, finalize resolution of the defects, and assign resources.
* Database Server - refer to the back-end system of a database application using client/server architecture. The back-end, sometimes called a database server, performs tasks such as data analysis, storage, data manipulation, archiving, and other non-user specific tasks.
* \*Hardware Interface - an architecture used to interconnect two devices together. It includes the design of the plug and socket, the type, number and purpose of the wires and the electrical signals that are passed across them.
* IEX Cloud API - allows you to get access to data quickly so you can focus on building the features your users need. a platform that makes financial data and services accessible to everyone.
* Integration Testing / API Testing - a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.
* JavaServer Pages - type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSPactions and commands.
* MVC Model-View-Controller (MVC) - framework is an architectural pattern that separates an application into three main logical components Model, View, and Controller. Hence the abbreviation MVC. Each architecture component is built to handle specific development aspect of an application.
* MySQL - provides an easy to use interface for performing the many tasks involved when working with databases. It integrates SQL development, administration, database design, creation and maintenance into one visual integrated development environment.
* RESTful API - REST is an architectural style, and RESTful is the interpretation of it. That is, if your back-end server has REST API and you make client-side requests (from a website/application) to this API, then your client is RESTful.
* \*Software Interface - are the languages, codes and messages that programs use to communicate with each other and to the hardware. Examples are the Windows, Mac and Linux operating systems, SMTP email, IP network protocols and the software drivers that activate the peripheral devices.
* System testing - a level of software testing where a complete and integrated software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.
* Unit Testing - a level of software testing where individual units/ components of a software are tested. ... A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. In procedural programming, a unit may be an individual program, function, procedure, etc.
* \*User Interface - the means by which the user and a computer system interact,
* Webserver - a computer that stores web server software and a website's component files (e.g. HTML documents, images, CSS stylesheets, and JavaScript files).
* \*Website Security - protecting website or web application by detecting, preventing and responding to attacks.

**1.4 References**

Definitions from: <https://developer.mozilla.org/en-US/docs/Learn/>

1. **Overall Description**

Overall, we wanted to create a software that would make analyzing the stock information of a 30-day time period as easy and clear as possible. We wanted the user to be able to select any day [or range of days] to display and compute several operations to the data. The user will be able to: (1) simply list the data, (2) view the changes in data overtime, (3) compute the mean/min./max./median. With the use of graphs and tables, all of the information will be plainly displayed for users to view.

**2.1 Product Perspective**

Our system is not one of a kind. There are several software’s that have already been developed and can do the same functions ours can do. Similarly, to how we produce this system ourselves, many people can do it too. However, after carefully constructing the best layout and program to be able to do the basic functions we wanted, we finally created one that works. Stock trading websites/apps such as: Robinhood, Acorn, Stash, etc. all can do the things our software can do. Therefore, we like to think of ours as the simplest form of stocking information.

**3. Requirement Engineering**

**3.1 - Software Requirements Specifications**

Before we could deliver a final product, we had to perform several tests to ensure the functionality of the program. In order to guarantee the program would work on a regular basis, we had to test our search and retrieval programs. After composing tests using the agile method, and making a few minor adjustments, we were able to consistently obtain the data from the database. Without a constant flow of data collection from the database, we would not be able to perform the later – added functions.

All of the tests performed must fulfill no less than 100% for the pass and run rate. If our tests were unable to meet the criteria for successful completion of a test phrase, adjustments were made. Once the tests were up and running, they were set to a daily routine check to continue to reassure the progress.

**4. System Modeling**

**5. Architectural Design**

1. The user makes a request to the web server and will receive a response using JavaServer Pages (JSPs).
2. The web server will host the applications various layers which will conform with MVC:
   1. Presentation layer will allow users to interact with the application via HTTP requests and responses rendered in a browser.
   2. Application layer manages the flow of the application, implements business logic and liaises with the data layer to process requests from users

and their responses. It is open-source, and third-party products reside here.

Figure 1, architectural design model

* 1. Data layer handles domain data and provides persistence and retrieval services for the database.

1. The database is where the data is persisted and retrieved.
2. The web services allow for interaction with other applications.

**6. Software Design and Development**

**6.1 Daily Update Routine**

By using Linux crontab, we are able to schedule the daily update routing “iextry.py” to kick off daily at 7:00 a.m.

“A program was written to check if the database is updating daily at 7:00AM. The program creates two text files, storing the current database as "today" and renaming the previous existing file to "yesterday". The program checks both lists and finds any entry that does not exist on both lists and therefore attempts to find if the database was updated. As of 12/5/2019 it has been working as intended” posted by Keith, our software tester.

**6.2 Team Features**

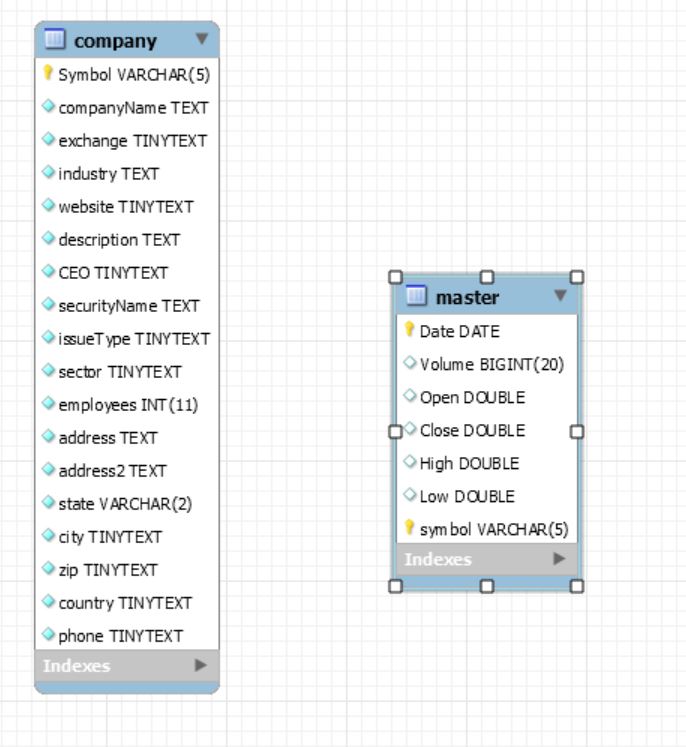
**6.2.1 Team Feature 1 – Statistical Graph**

Using the user system, the user will be able to input information. Based on the input, the user can adjust and draw graphs**.**

**6.2.2 Team Feature 2 – Betting Game**

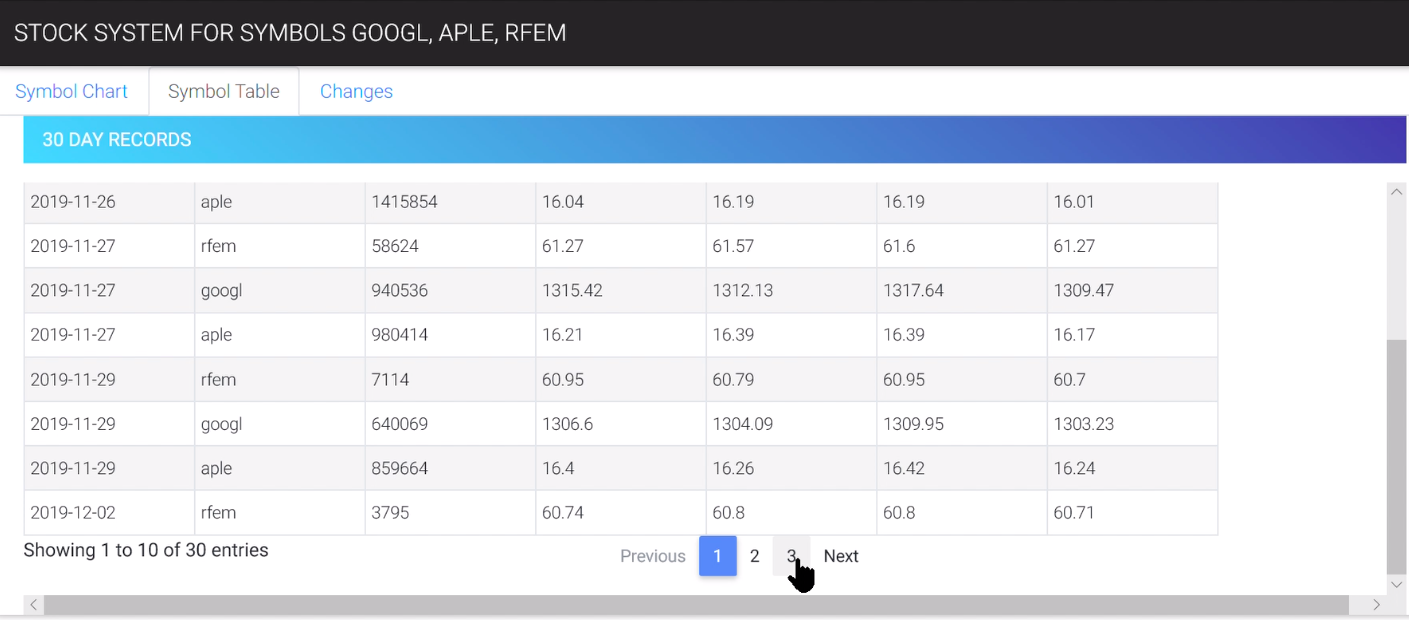
Using the user system, the user will be able to place a bet on any of the three stocks. Based on which stock had the highest entry in the database, the interface will send an email telling the user if their stock won or lost.

**7. Database Design**



Our database is simple but effective. There are two indexes: (1) company and (2) master. With the master index pointing/ collecting information from the company index, we are able to list out the basic but necessary information for each of the companies. The information the database lists out, but is not limited to,

1. The CEO’s name
2. The company’s symbol/sigil
3. The company’s address
   1. The street
   2. The city
   3. The zip codes
   4. The state
   5. The country
4. The company’s name
5. 3-4 sentence description of the company
6. The amount of employee’s the company has
7. The stock exchange used for the company
8. The company’s industry
9. The issue types
10. The company’s phone number
11. The sector
12. The securities name
13. The company’s website link

****

*Figure 2, A display of the stock system for symbols googl, aple, rfem.*

**8. Project Management**

Our group experiences several setbacks. Technically, we did not officially begin to make progress until November 17th. After completing our group and receiving some guidance from Dr. Wong we were able to start.

**8.1 Progress Log**

|  |  |
| --- | --- |
| September 25th | Teams were assigned.  Team leader: Yuwei  Team members: Nicole C., Keith E., Austin S. |
| October 4th | Austin S. was removed from Team 3 and replaced with Brian Yao. |
| October 5th | All email addresses were obtained, and Nicole C. sent out a group email. |
| October 8th | Yuwei, Keith, and Nicole had brief (5 minute) meeting to discuss the start of the project. |
| October 15th | Yuwei, Keith, and Nicole had meeting to discuss the project and the approach the team will take. |
| October 23rd | Team meeting with Yuwei, Brian, and Nicole.  We further discussed the approach we would  take. Clarified the purpose and extent of the  project. |
| November 4th | Keonta H. was added to Team 3. |
| November 17th | Dr. Wong provided examples, models, and  templates [of System diagrams,  Architecture Layer diagrams, and  Software Stack diagrams] to help us get  started. As well as, scheduled a group phone  call with us. |
| November 18th | Team 3 had a Skype phone call with Dr. Wai-Tak Wong. We were given job roles, and task. We uploaded all the initial information on Trello. |
| November 25th | Team 3 had a follow up phone call with Dr.  Wong to make sure we were staying on track  and making progress.  Brian created the architectural design for the program.  Keonta created the database design for the program.  Keith began his testing plan. |
| November 27th | Dr. Wong put a cron schedule to update  Keonta's python program to update the  daily data. |
| December 3rd | Brief team meeting to discuss the remainder of  the project. Also, gathered more information to  add to the document. |
| December 5th | Keith did his first test case for the daily  database update and process at  approximately 2:30 p.m. All tests passed. |
| December 9th | Team 3 had a final follow up call with Dr.  Wong to discuss the presentation. |
| December 11th | Team 3 had a Skype call to discuss added  features and to schedule a team meeting for to  finalize the presentation.  Keith ran second test on the front-end user  interface at approximately 10:40 p.m. All tests  passed. |

**9. Testing Plan**

\*\* Please refer to Keith E.’s testing plan document. Can be found on GitHub: Team3 > Testing folder > Testing Plan – Software Engineering.