

Plutocrat

Connor L. Byrne

C16485224

Submitted in partial fulfilment of the requirements for the degree of:

B.Sc. in Business Computing

Technology University Dublin

May 2020

Supervised by - Neil O’Connor

# Acknowledgments

I would like to extend my gratitude towards all my lecturer and mentors that I have had throughout my course, contributing knowledge that made this project possible.

I would like to give special thanks to my project supervisor Neil for overseeing my development process and steering the project in valuable new directions throughout the process.

I would also like to thank Jenny Munnelly, for being the projects second reader, offering valuable advice and sanity checks, and also for teaching the courses distributed systems module which ended up being very applicable to Plutocrat’s early stages of development.

**Declaration**

This is an original work. All assistance is acknowledged

Signed: Connor Luke Byrne

Date: 23/04/2020

# Introduction

Plutocrat is a web-based machine learning system application which utilised web scraping to gather sentiment data concerning companies on the Nasdaq 100 listing. It analyses this data using Stanford Natural Language Processing, before passing its findings to a TensorFlow based neural network which finds the correlation between what online users are saying about a company, to its stocks, and the change in value that can be expected from those stocks.

I found it pertinent to find a project with a strong business application, that I also personally find interesting. I was able to apply an understanding of equity trading that I gathered from books such as ‘The Intelligent Investor’ which lead me to an understanding of how heavily the publics opinion of a company could influence the value of its stocks and shares. On top of this, from frequenting business new sources such as Forbes and Business Insider, I was able to observe the value of companies stocks rise and fall along with the publics reactions to the organisations actions and scandals, lending credence to my theory that if these falls and rises could be observed and their effects predicted, it would lead to lucrative opportunities for those interested in the world of stock trading.

A large amount of the knowledge I gathered in the course of the program has been applicable to the project, such as web scrapping, database manipulation, web stack development, etc. However, the development of Plutocrat required proponent of self-education on top of what I already knew. First of I need too self-educate from a bare bone beginning in the realms of sentiment analysis, and machine learning, two proponents of the project which I had absolutely zero experience with before beginning the project.

I believe that these extra skills and experiences that I have garnished on to of the already impressive plethora of abilities I gained during my degree, will assist me in my future career, and stand as testimony to my progression as developer and as an individual. Already I can see how Plutocrat can be improved and expanded in the future, how it’s processes can be improved, and how it’s predictions can be made more accurate.

# Objectives of Plutocrat

The primary objective of Plutocrat is to build a web service that that can act as a guide to the chaotic world of stock trading. Of course, the system can never hope to become infallible. It is basing its prediction on one aspect of influence in a complex system of stimulus that dictates the ebbs and flows of the stock market. However, I do believe that the effect that sentiment has over the value of equities is significant enough that it can offer. The front page will display a list of all the companies listed in the Nasdaq 100. Clicking into any of the companies will reveal specific details about each company, including the predicted change in that stock price within the next 24 hours.

The overt operations of the website are sparse in comparison to its backend processes, the webservice will primarily work to display the work of the backend sentiment analysis and neural network.

# Business Case of Plutocrat

The idea for Plutocrat was formed when, in my research, I learned that since the 2008 financial crisis investor interest had shifted from the use of mutual funds (in which fund managers pick stocks with the aim of beating the market) to index funds(which replicate established stock indices e.g. S&P 500, Nasdaq 100, etc). I also learned that this sector is under the control of a power oligarchy referred too as the “Big 3”. The Big 3 is made up of 3 asset managers: BlackRock, Vanguard, and State Street, which combined manage nearly $11 trillion in assets. The part that we are most interested with, however, is the fact that the Big 3 firms, along with other investment firms, have consistently failed to beat the market in previous years. This means that the increase in value of these funds has been, in best case scenarios equal to the amount the value of the general market has increased, but in most cases has been considerably lower. Similar phenomena are mentioned in the Benjamin Graham text “The Intelligent Investor” indicating that this is not a new problem, but has been happening for decades, and has instead been exacerbated by modern market uncertainty. Even Warren Buffet has said in interviews that he has found it difficult to make profitable investments in recent years due to market volatility.

This is where the value of Plutocrat becomes apparent. The fact that entrusting assets to a firm of investing professionals appears to be slightly less effective than investing randomly or flipping a coin to choose stocks. This lack of certainty when desiring returns on one’s investment indicates the dire need for some sort guiding agent, to inform invertors, both beginners and seasoned sector veterans.

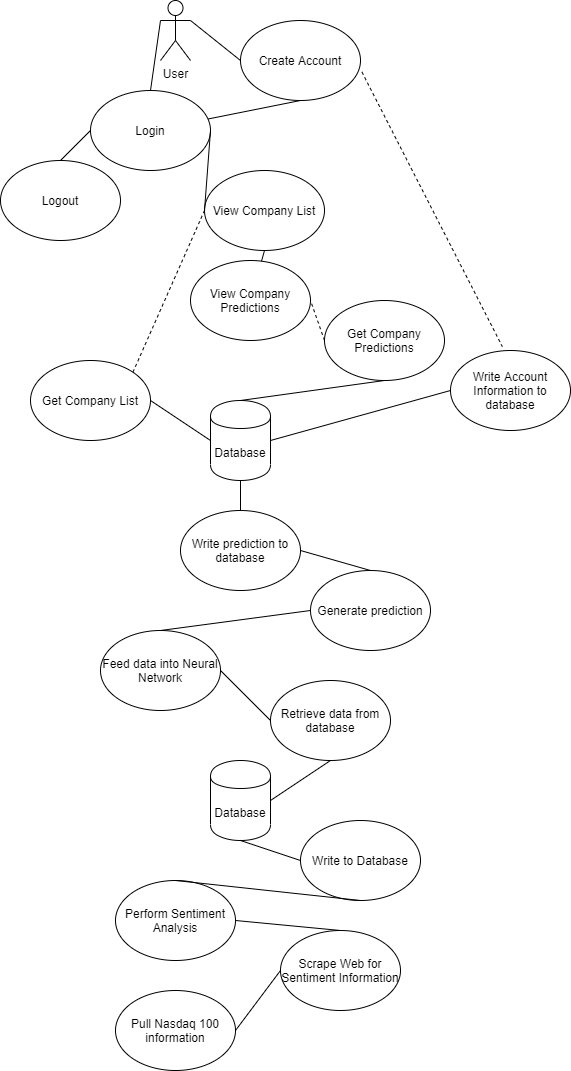
The main commodity that Plutocrat deals in is information. Its current functionality can be expanded in the future to allow users to more heavily interact with the service. Perhaps facilitating trading and information transfer between users. Informing and, perhaps, even automating trading in the same manner as other platforms such as Etoro and Robinhood. Such recommendation and trading would take place through the framework of sentiment-based predictions but could be expanded as the service evolves. Currently the service would subsist on selling subscriptions for access to predictions. It is easy to analyse the cause of a certain economic event after it has already occurred, after all opportunity to profit from it had evaporated. If Plutocrat can provide any clarity in predicting these events before they happen, it’ll certainly become a lucrative venture for all of its users and owners.

# Requirements Capture Analysis

The only business actor is the user.

Everyone that visits the website in its current form, or signs up for a subscription in future versions, takes on a user role. All users can perform the same actions and view the same data. Studying investment texts and news sites has provided a clear idea of what information would be valuable to the end user. All pointed to the Adjusted Close price of a share to be the most valuable piece of information over time, as it compensates for the apparent “drop” in value that shares experience when a company decides to perform stock splitting. Any other metric may cause the system to become confused and recommend that users sell their ownership when a spit is eminent, when this usually occurs when a company is doing quite well.

# User Use Case Diagram



# Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Section/Layer** | **Title** | **Requirement** | **Justification** |
| Web layer | User Login | User must have valid account | Security. Subscriptions is how profit would be made in future versions |
| Web layer | Create Account | Allow the user to create a new account on the website | Allows a new user to subscribe to the website and contribute to its traffic. |
| Web layer | Logout | Allow the user to logout when their session is over. | Reduces the amount of inactive connections to the server at any one time. |
| Web layer | Show Companies | Allow the user to view all the companies (Currently all companies on the Nasdaq 100 listing) | Enables the user to engage with the website, choosing exactly what data they are interested in. Improved convenience. |
| Web layer | Show Company | Allow the user to click into any company and view specific predictive data for that company | This is where the real valued data from the Plutocrat neural network will be displayed for the user |
| Web layer | Pull data from database | Query the database for the company listing, and all data relevant to each company | Enables all of the data to be gathered and displayed to the user. |
| Machine Learning Layer | Pull sentiment information from database | Bring data into program to be analysed | Enables for information from a lower section of the system to be brought into the machine learning layer and fed into the neural network |
| Machine Learning Layer | Get stock price | Get the historic price for sentiments whose dates have already passed | Brings historic stock prices into the system for the sake of training the neural network, and for users to visually compare how accurate the predictions were on the front-end webpage. |
| Machine Learning  Layer | Construct/train neural network | Assemble the structure of the neural network, layer by layer, and trains it by passing in training data. | The foundation of the project, and Plutocrat’s value. Enables the system to make predictions about value changes in stock prices. |
| Machine Learning Layer | Make Predictions | Pass sentient data from the database into the model to produce a predicted value change | Outputs a predicted value change in sentiment relevant companies to guide investments |
| Machine Learning Layer | Write to database | Write the prediction data back into the database | Allows the predictive data to be retrieved from the machine learning layer and to be displayed to the user |
| Web-scraping layer | Gather Nasdaq 100 listing | Use JSOUP to scrape the web for the current Nasdaq 100 listing | Allows the system to gather the names and information of relevant companies, for which it will gather sentiment information |
| Web-scraping layer | Gather Sentiment data | Use JSOUP to scrape sentiment data from sources | Gathers data to be analysed by Stanford NLP |
| Web-scraping layer | Analyse sentiment data | Use Stanford NLP to generate an average positivity rating for the piece of sentiment. | Generates information that can later be fed into the Machine learning layer’s neural network to train, and to make predictions |
| Web-scraping layer | Write to database | Writes the sentiment, and company data, into the database. | Makes this data available for use by the Machine Learning layer and the Web layer |

# Non-Functional Requirements

## Usability

The website is minimalistic to allow users a sleek and quick experience. Currently the website consists of two pages; one for searching for and selecting the company you are interested in, and the other for viewing specific information about the company you have chosen. All of the complicated mechanisms of the web-app’s functionality are effectively hidden from the user. The data that is displayed is simple enough to be understood by a layman unfamiliar with the world of stock trading yet is gathered and processed in a way that even experienced traders should fine pertinent to their success.

## Reliability

The success and future of the service relies heavily on the reliability of the predictions that the service can offer, and how reliably users can access this information. To this end, we should be constantly aware of ways that we can intelligently steer the learning of the neural network. Both by altering the data being fed to the network and the networks over all structure itself. The purpose of the app is too of course finding patterns that humans would not be able to see, using data that is far too vast for humans to process. But this does not mean that all of the work has been done, and that the algorithm will take it from here. Instead we should us our human intelligence to gauge the effectiveness of the service and lightly steer operations when the need arises.

## Availability

The web app will be ready for use instantly with no download or installation required by the user. The service will be continuously available at all times that the servers are active and will require no input from the user.

# Design

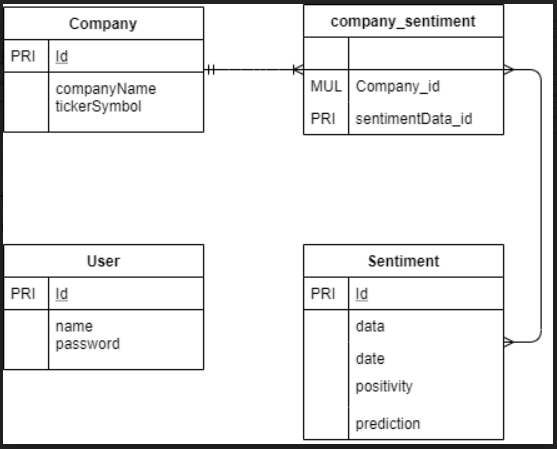
## Technologies Utilized

A large plethora of technologies were utilized in the development of Plutocrat. The main technologies used are listed below. The main languages utilized were Java and Python, with an amount of HTML and XML for webservice functionality, database persistence, and maven dependency management.

|  |  |
| --- | --- |
| How to import a simple Java project into the Eclipse IDE - Algol.dev | **Eclipse IDE and JEE for Java developers**  Two of Plutocrat’s 3 layers are coded in java. The layers pertaining to web scrapping with **JSOUP**, sentiment analysis with **Stanford NLP**, and saving this data to a **MySQL** database using **Hibernate ORM**, were created with Eclipse IDE.  The webservice layer which utilised a **Spring MVC** framework to retrieve the data from the **MySQL** database, again, using **Hibernate**, utilised Eclipse JEE to create a dynamic web project on an **Apache Tomcat** server. |
| Top VS Code Extensions to Improve Your Productivity | SyntaxSofts ... | **Visual Studio Code**  While the top and bottom layers of Plutocrat were created using **Java Eclipse** the intermediate layer was coded in Python. This layer utilised **Yahoo Finance API** for retrieving financial values pertaining to each company. This information was then passed to a **TensorFlow** neural network along with sentiment analysis data retrieved from the database, to create the machine learning algorithm at the core of Plutocrat’s functionality. |
| C:\Users\i352770\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\DC4168DF.tmp | **Version Control GIT/GitHub**  I used GIT and GitHub to keep a repository of the process that was being made throughout the course of the project. It was useful to have access to a technology that not only prevented loss of progress (which unfortunately happened once in the project, before I started using GIT, when windows automatic back-up failed to save my files when my laptop needed to be re-imaged). But also, to have a means to test new features before integrating them into the main project. |
| Jsoup Iterate all elements of HTML example - Java Code Examples | **JSOUP API**  One of the simpler technologies in the project, JSOUP allowed me to access Nasdaq listing without a complicated API, but also allowed me to scrape sentiment information from various news and social media sites. |
| C:\Users\i352770\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\EAD7CC61.tmp | **Stanford Natural Language Processing API**  I used Stanford NLP to perform the sentiment analysis aspect of my project. Stanford NLP has many abilities regarding the processing of English such as entity/keyword recognition, and sentence tokenisation. However, for this project I was only interested in its sentiment analysis capabilities. When data is fed into the API, it will breakdown the data into its individual sentences, and return a Positive, Negative, or Neutral rating.  The API performed its duties with reasonable effectiveness. When reading back over the result, versus the data it was fed, it is possible to see instances where the technology perhaps misunderstood the intention of a statement or misinterpreted a sarcastic remark as positive. This is too be expected as language is a complicated phenomenon, that the human mind has evolved to be able to interpret. It is possible that access to more resources would afford usage of a more effective system. This is one way that Plutocrat can be improved in the future |
| Hibernate (@Hibernate) | Twitter | **Hibernate Object Relationship Manager**  I used Hibernate ORM to automate the task of creating and managing relational tables in the underlaying **MySQL** Database. The use of Hibernate was more of a convenience and a timesaver than a necessity, as the project could be completed without it, but it did save me from having the execute raw sql statements within my Java classes. |
|  |  |
|  |  |
| C:\Users\i352770\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\F82F01D9.tmp | **TensorFlow**  I used TensorFlow to construct a neural network that could be fed data from the bottom layer of the project; generated with **JSOUP**, and **Stanford NLP**, retrieved from the **MySQL** database; alongside financial information pulled from the **Yahoo Finance API**. This data is used by the network to construct a learning algorithm that will make predictions concerning price changes when sentiment information is passed into it in the future.  From all the technologies involved in this project, TensorFlow provided the steepest learning curve by far. Not only was it such an integral part of the core concept of the project that its presence could not be circumvented, but I was also the technology that was the furthest from anything I had encountered in the past. I found TensorFlow distinctly alien compared to all my past knowledge of other technologies. On top of this, neural networks are an incredibly obtuse subject matter, that even experienced developers have issues wrapping their heads around.  In the end, I feel as though I utilised TensorFlow to an acceptable amount. The system that I have in place can certainly be improved in the future, and the combination of machine learning and sentiment analysis has not come close |
|  | **Yahoo Finance API**  Yahoo Finance API is an incredibly simple technology that only took minutes to learn how to use. You simply pass it a companies ticker symbol and a date, and the API will return all the financial data relevant to that company on the specified date in the form of a Data frame, which can be dissected further to gain the specific information you are looking for. |
| Spring | Home | **Spring MVC**  I used Spring MVC to construct the website to display the data generated from the rest of the project. I created I dynamic web project in **Eclipse JEE**, which I then converted into a maven project to handle all of my spring dependencies. I was able to use this to create a quick and simple website hosted on an **Apache Tomcat** server. |
| C:\Users\i352770\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\4476A835.tmp | **Apache Tomcat**  Finally, I hosted my **Spring MVC** web application on an Apache Tomcat server, allowing for a simple creation and hosting of the service. |
| C:\Users\i352770\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\2104A953.tmp | **MySQL**  MySQL is a basic run-of-the-mill database. It provided exactly what the project needed, in regard to the fact that the project only required basic database functions, such as data to be saved and retrieved across multiple sections of the project. |
|  | **Bootstrap**  Bootstrap is a collection of CSS and JavaScript libraries that I used to style and organise the website. |
|  | **Google Charts**  Google charts is a JavaScript API from google that allows developers to create interactive charts on their websites. I used it for the sole purpose of creating material line graphs for the company’s individual pages. |

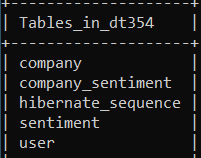
# Data Model

## Data model Diagram

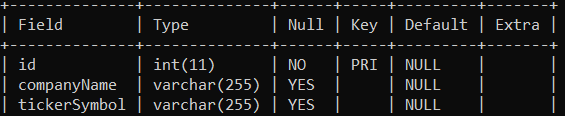


## Database Object Summery

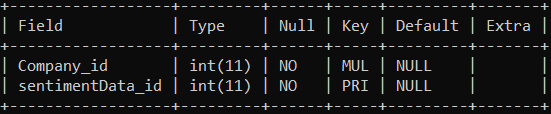
Database



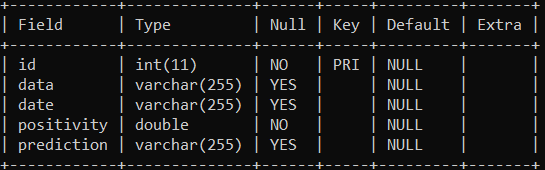
Company Table



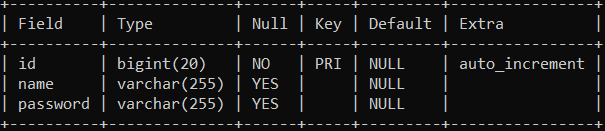
Company/Sentiment joint table



Sentiment table

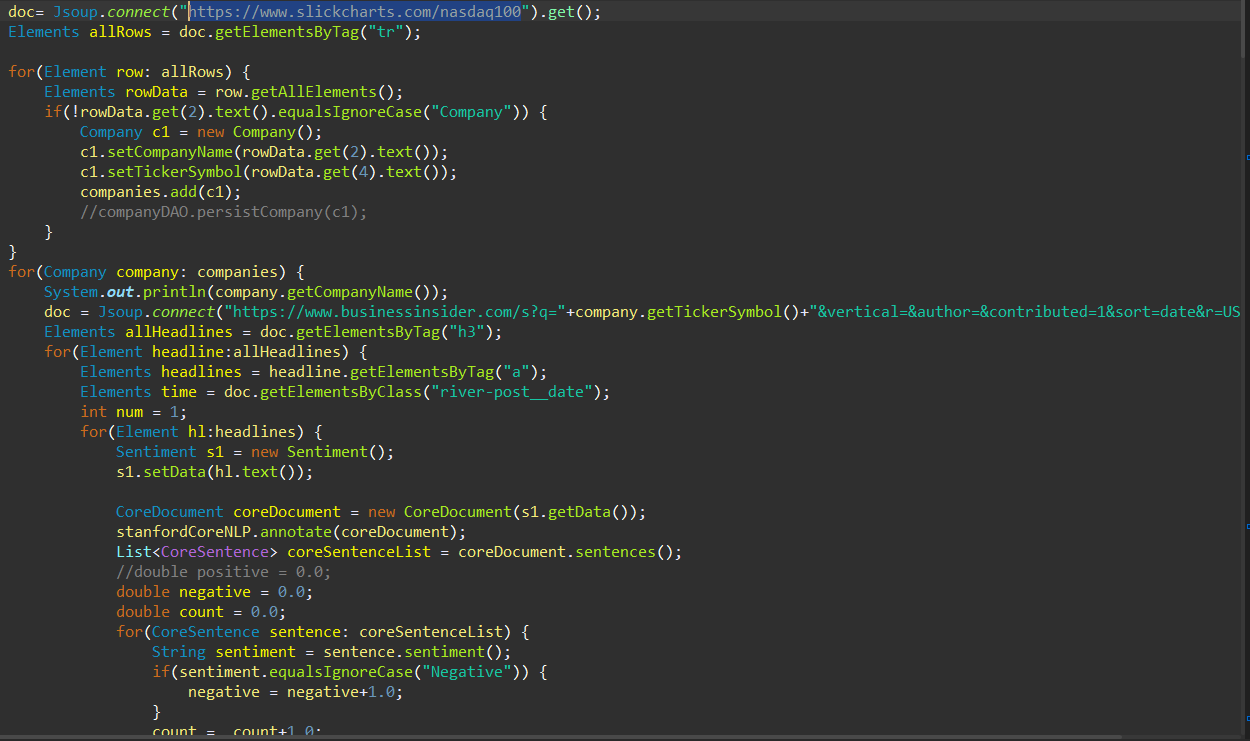


User Table



# Implementation

## Web scrapping



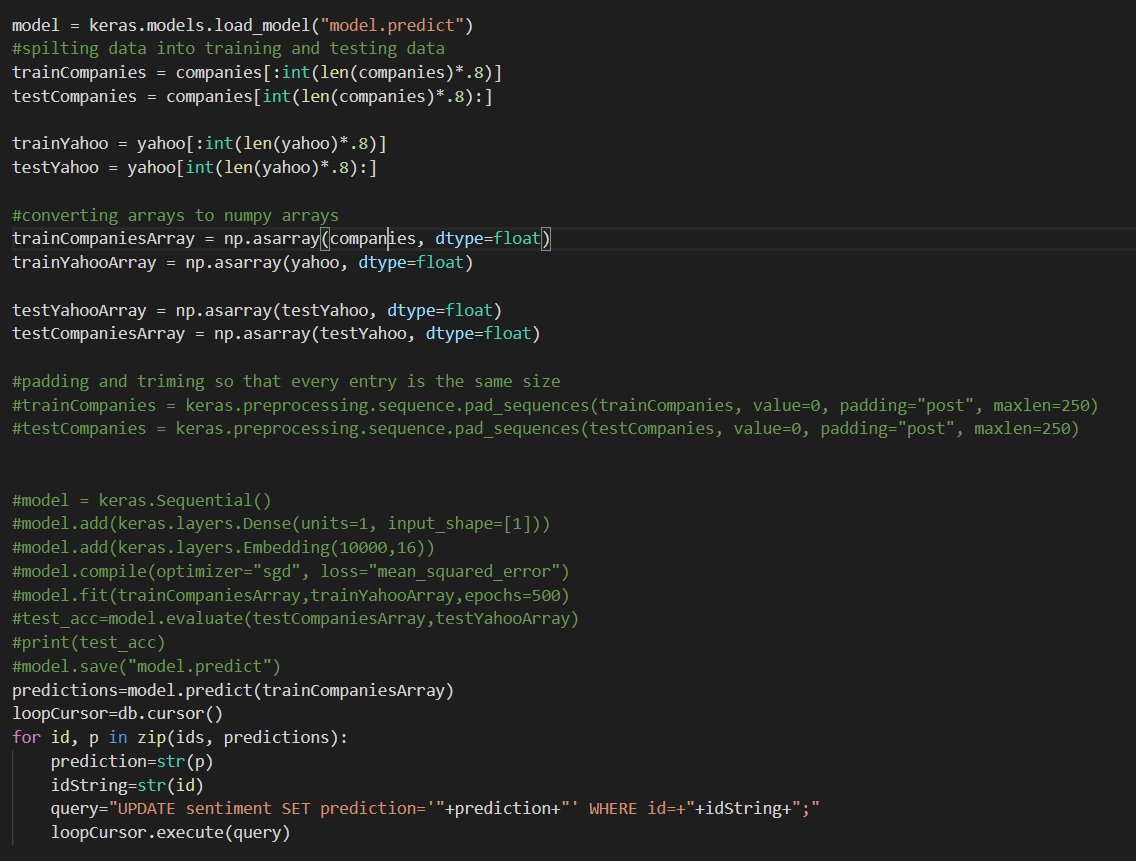
The first action that the system performs is getting the Nasdaq 100 listing from slick charts, using the information gathered to create company objects, and persist those objects to the database with hibernate ORM. It then slots the name of the company and its ticker symbol into the URL of any sit that we wish to scrape for sentiment data. Currently the site being used for gathering information from bussinessinsider.com.

After trials with many different potential sites, such as; twitter, Forbes, and reddit. business insider was chosen for a number of reasons. Firstly, the site is very stable in comparison to sites such as twitter, which change their structure and function so often that the system would need bi-weekly maintenance to keep functioning. I learned this the hard way in the early days of the project, when a previously functioning section of the systems ceased to function after twitter got an update.

Secondly, the language on the site is plainly written and informative, rather than the more esoteric stylings of other sites, which the Stanford NLP seems to find more conjunctive towards generating accurate readings of sentiment from the site. Thirdly, the site has a simple layout that is more co-operative when specifying what data is to be scraped, in comparison to more complicated websites such as Forbes. Fourthly, when the data is passed to the neural network it appears to generate much more accurate predictions than data pulled from investment concerned sections of reddit. Business Insider can be searched much more accurately, with more recent results than reddit.

A final note to make about the current functionality of the system, is the current use of Business Insider as the single source of data for the current version of the project. Having a single source for data makes the system more stable and maintainable for testing purposes. While the project is in it’s early proof-of-concept stages, it’s best to have a stable source of test data to run through the system and demonstrate its capabilities, particularly considering that certain JSOUP and Hibernate functionality are not thread-safe, making a system that draws from multiple sources unstable and needing much more upkeep. When the project goes into full time production, other sources of data can be reintroduced.

## Machine Learning



The intermediate layer of Plutocrat is a Python program concerned with the machine learning aspect of the project. It was implemented by pulling data from the underlaying MySQL database. Each piece of sentiment data was retrieved, along with the date that it was generated, and the ticker symbol of the relevant company. The date and the ticker symbol are then passed into the Yahoo Finance API to find what the companies adjusted close price was the day that this piece of sentiment was created. The date is then incremented, so that the adjusted closing price for the companies next day can be found. One is retracted from the other to find the value that the companies stock changed by in the 24 hours subsequent to the creation of the piece of sentiment.

When this is complete, what we are left with is one list containing all of the sentiment data generated by the bottom java layer of the project, utilising JSOUP and Stanford NLP, and another list containing the amount that the value of the companies with which this sentiment data is concerned changes by in the 24 hours succeeding the publishing of this data. The above snippet of code illustrates how these lists are split into training data and testing data, and converted into NumPy arrays, which are the only types of arrays that can be converted into tensors inside of a neural network.

The sections of code that actually construct and train the neural network are commented out in the screen capture above, the reason for this is that the model only needs to be built and trained once, it can then be saved, where it can be loaded to save time in the future. The line ‘keras.model.load\_model(“model.predict”)’ can be seen at the top of the capture, this is where the model is being loaded.

In the commented-out section, you can see that the neurons are being defined. The structure of the neural network has gone through many iterations and versions throughout the lifetime of this project. Initially, I structured the neural net work very similarly to that of a practice project I built while teaching myself the functionalities of TensorFlow. Specifically the initial structure of the neural network consisted of an embedding layer, which would take a number input from the developer (10,000 for example), and would only take the most common 10,000 words that appear in all the sentiment data passed to it into consideration, and disregards all other words weighting it’s predictions. The reason that I took this out is that was no practical way to make this work with the rest of the network, or in regard to the application in general. An embedding layer essentially requires the developer to provide a custom dictionary to define all the possible words that could appear in a piece of data, so that it may decide which words are relevant to its predictions. In the practice project that I created during my self-education, this dictionary was provided by keras itself, as it was an in-built project for to teach the user how to develop a text classification algorithm for ranking movie reviews. I had neither the time nor, frankly the know-how to develop a custom word dictionary, due to the fact that I could not find a tutorial online to instruct on how to construct a dictionary of reasonable quality.

Another neuron that was originally in the network was the GlobalAveragePooling1D layer, which worked in tandem with the embedding layer, to average out the results of the preceding layer and distribute these average results across the neurons in the following layer. This layer served no function without the embedding layer, so I decided to remove it alongside the embedding layer. When these layers were included in the network, the implementation of the project at that time involved converting entire pieces of sentiment data into decimal representations of themselves to be passed into the algorithm. The removal of these layers prompted the change of having the results generated from the Stanford NLP API into the network instead.

In the end I opted for simplicity. Taking direction from TensorFlow official guides and documentation, I constructed the network from layers of Dense neurons, which are the simplest type to implement. Dense layers are used for pattern matching between two NumPy arrays of numbers. I used these layers to find the patterns in price changes corresponding to the positivity of sentiment data published in the preceding 24 hours.

## Plutocrat Web



The web proponent of this project handles retrieving the information generated from the rest of the project and displaying it for the user. This is achieved by creating a web-application with the Spring MVC framework. This top-level of the project reverts back to Java as its language.

After pulling the information that was written to the database in the bottom level of the project, along with the prediction data that was supplemented in the intermediate layer. The framework accomplishes the construction of the website by implementing a number of classes:

#### WebAppInitializer:

Performs the actions needed to be performed when the server is started. Such as creating servlets, defining mapping, and registering the config files.

#### WebMVCConfig:

This file creates a view resolver, which informs the application of the file path that webpages can be found on, and what suffix to expect: .jsp, .html, etc. Which informs the sever of what file types that it is dealing with and allows them to be located more accurately.

Also, in this case, I have added a resource handler, which points to the location that the bootstrap resources and jar files are hosted at, within the maven dependencies, so that the application may make use of them

#### JpaConfig:

This class defines that java beans used in this project.

It helps create the Entity Manager Factory in accordance to the details provided in the persistence.xml file in the META-INF folder. Since I made this webapp using hibernate ORM the app needs this to enable it to manage its interactions it the underlaying database in regard to the present entities

#### CompanyRepository:

A repository is an interface that can be used to handle interactions with the webapps database. Many queries can be performed automatically without needing to write sql (such as save) while query’s can also be written manually if needed with the @Query annotation. This repository Is set to implement a CrudRepository interface and is set up to specifically deal with Company Entities.

#### CompanyService

The product service class allows for decoupling between the controller class and the repositories. It also allows for all objects to be handled through one class rather than needing to have a perfect knowledge of what capabilities each repository has when handlining several different entities.

#### CompanyController

The controller class is where all the URL mapping of the webapp is performed. When a certain URL is linked too in a view with a button or a link, a method in the product controller will be invoked using the @RequestMapping annotation. This method can perform any actions that are necessary and return a view with data passed to it, or a redirection to another page in the form of a string.

#### Company

This class holds all the information for the Companies within the webapp, such as name, ticker symbol, and a list of sentiment objects which contains all the sentiment data and predictions that are relevant to that company, this list of sentiment objects is implemented as a join table within the database

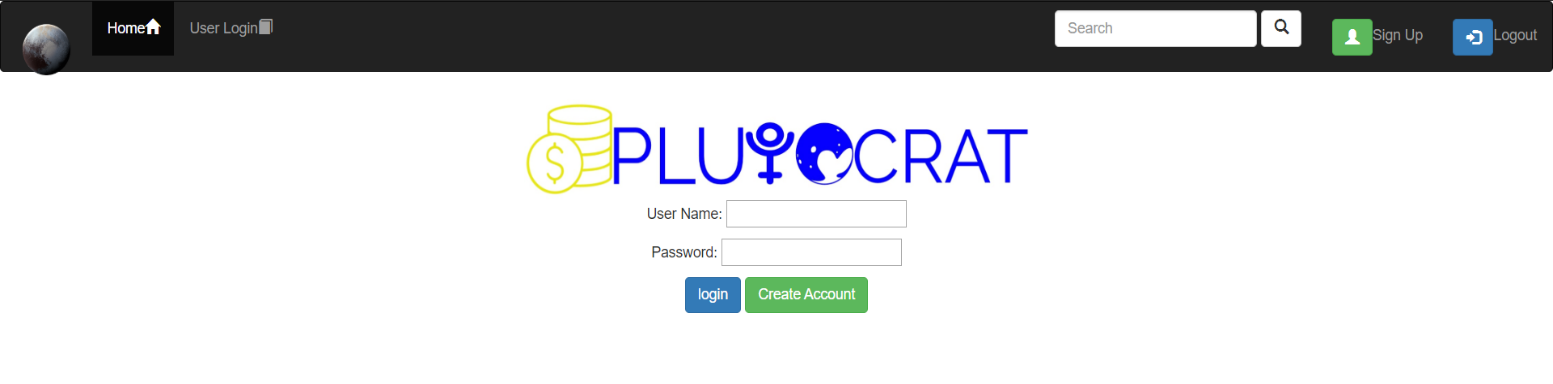
#### Sentiment:

This class holds all the information for the sentiment data within the webapp, such as their the date it was created, the change in the company’s value change that the neural network believes this sentiment will cause in the company’s value, and of course, the sentiment data itself; such as the tweets and article that have been analysed.

#### User:

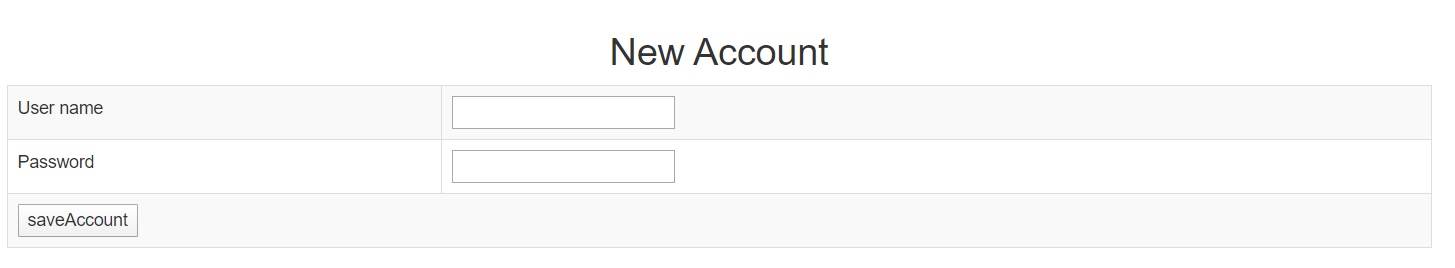
This object class represents the User who creates their account and interacts with the system through the webpage. The only information this class currently holds is the users name and password. The class also contains a list of companies, to facilitate a user following the companies that they are most interested in and adding these companies to a list so that they do not have to search for them every time they visit. This functionality is unimplemented at the current time; however, it acts as a place holder for a wide range of functionality that can be added to the system in the future.

## User Experience

When a new user first encounters the site, they will be met with this page:

Where they will be prompted to create an account or login if they have one

If they click on the “Create Account” button they will be redirected to this page:



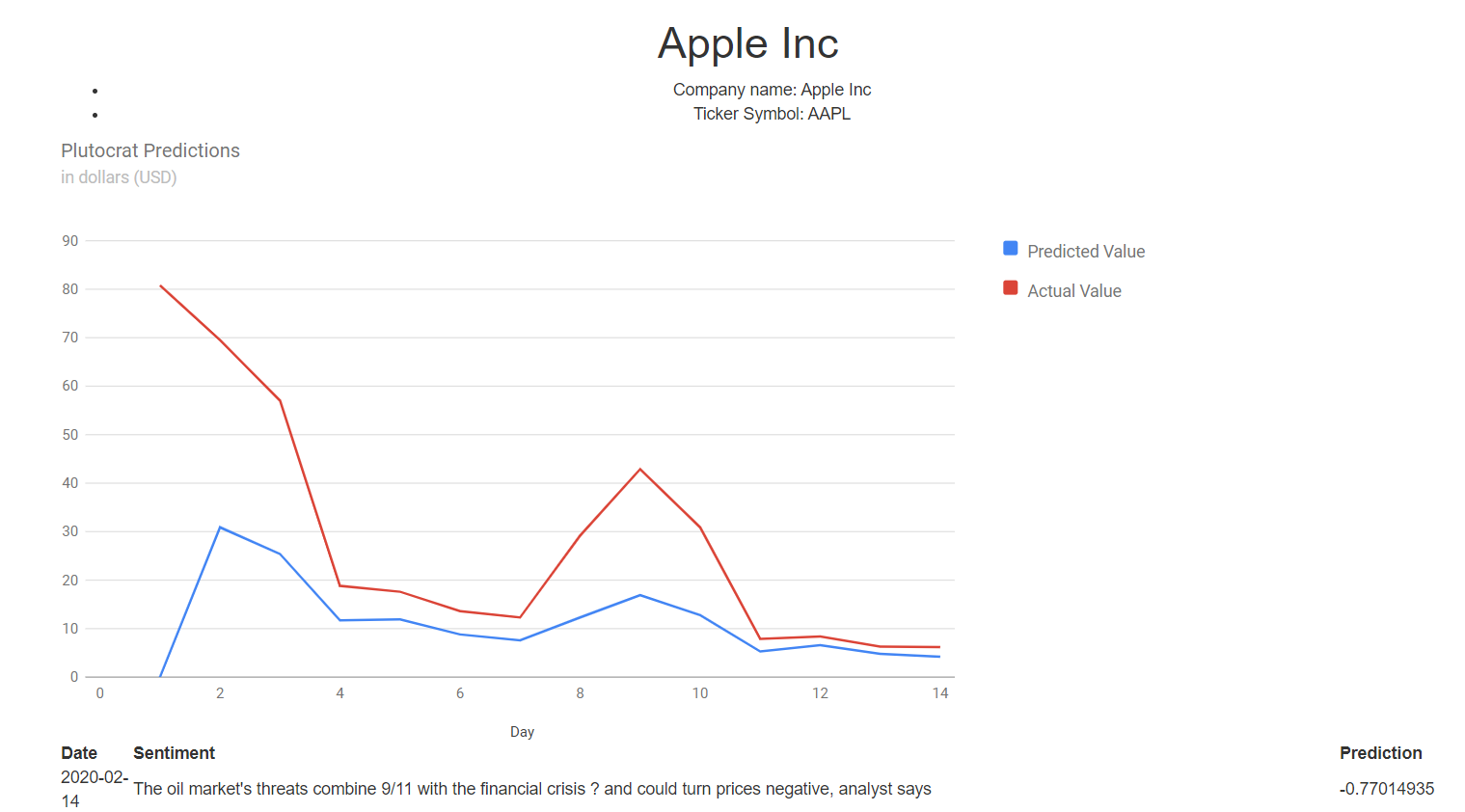
Where they can enter a username and password that will be stored in the systems database.

If the user already has an account, and choose to login instead, they will be brought to the company index page:

# 

Where they can view all available companies, and search for any in particular.

From here they can click into any company that they are interested in, and be brought to a page with that companies’ details:



As mentioned above, the design of the website is minimalistic by choice, so as to not overload the user with information. Newcomers to the world of trading are usually put-off by the shear amount of information that is thrown at them by other similar services, and how little they comprehend, or have the ability to analyse this information.

Plutocrat instead, offers a sleek experience, where people of all experience levels can seek straight forward, and easy to understand guidance concerning their investment strategies.

# Obstacles and setbacks

This project was very ambitious from the beginning, perhaps overly so. I wanted to create something that was not only genuinely useful, but also challenged me. The largest obstacle in the creation of Plutocrat was the sheer amount of self-education that I needed to perform in order to create a project even close to my original vision. Though there were elements of Plutocrats implementation that I was able to complete by expanding directly on what I learned during my course, such as; JSOUP and Hibernate ORM; a huge amount of the functionality of the project was built completely from scratch, without any prior knowledge of the technologies, and with comparatively little input from outside help.

The most difficult section of the project to implement was the neural network aspect. TensorFlow is completely removed, in both concept and practicality, from anything I have encountered thus far, and consequentially took months of trial and error to result in any acceptable result. The Plutocrat’s neural network went through dozens of iterations before resulting in the simple version that it now has. Even following TensorFlow official tutorials produced example neural networks that had little relevance

# The future of Plutocrat

As mentioned, many times throughout this documentation, let this project act as proof of concept on a system that has not realised its full potential yet. There are many new pieces of functionality that may be added to the system in the future. Such as:

* An ability for users to follow companies that they are most interested in, and receive notifications when Plutocrat predicts that there is going to be a substantial shift in that companies’ value
* The ability to trade stock within the system similar to Etoro, Robinhood, and other services in this sector. With Plutocrats predictive elements setting it apart from competition in a similar manner to how Etoro invites new investors with the ability to set their trades to automatically mimic more experienced traders.
* The refinement of Plutocrat’s underlaying neural network, not only to increase the accuracy of its predictions, but also the scope of information that the system can provide. Perhaps the system can be trained to predict changes far further into the future. It’s possible that the system can be taught to gain an understanding of companies as a whole, and base predictions off of all elements that may effect their success, such as new of regulation changes, or buying trends, or competitors actions, rather than just news that is directly related to the company itself.

With more time, experience, and resources, Plutocrat can be grown into a real asset. One which can add valuable direction and guidance to traders at any level or experience.