2017

Final Report

SOFE3700 FINAL PROJECT

JASINDAN RASALINGAM – 100584595
DARRON SINGH – 100584624
KAUSHAL PATEL – 100586212
ASHWIN KAMALAKANNAN - 100584423

Link to Github repository: https://github.com/AshwinK97/SOFE3700-FinalProject

Abstract

By successively completing and implementing our stock analysis tool our group has created the following report that summarizes and outlines what we have accomplished. The beginning of the report states our objective and goals that we aimed to achieve with the stock analysis tool and how it was related to the course. The report then goes on to state the programs and software that were used in creating the site for the stock analysis tool and how the API was created. Screenshots of the ER Diagram are shown along with visualizations of the site itself, highlighting its various features. Finally, the conclusion summarizes what our group has done and the contributions that were made while stating our brief thoughts about any future work.

Introduction

Our project came about from the realization that most financial analysis site are cluttered. We created a site that consists of a simple to use interface, with the basic's a new investor would appreciate. The site contains data on 13 companies from the past 10 years, there is room to add more in the future.

Objective

Buying and selling stocks is an imperative part of our economy. Investing in stocks comes with many risks and doing so without prior knowledge can result in loss of money. Most online financial helpers lack the features that people need to do complex analysis, or they hide these features behind paywalls. In addition to this, most other tools have crowded interfaces that can daunting and hard to navigate. Our goal is to build a financial data analysis tool that is both powerful and easy to use by both new and experienced investors.

Relation to the course

This project required us to implement concepts that we learned both in lecture and in our labs. The stock market is based around extremely fast data transfer, and a lot of it. The project attempts to utilize this data with organization learned from the course. Various types of queries had to be used to retrieve the necessary data in tabular form to be displayed for the user. Data cleaning also helped with organizing the data to be displayed in charts for visualizing ticker prices.

Overview

We have built a web application that allows users to view and analyze an archive of financial data for various companies. This data can be used to compare and contrast stocks, look up a company's historical data and predict future trends.

Goals

- API: Download Price data (filtered i.e. date range, only volume, only close, etc.)
- Up to date price information (Daily)
- View a variety of tickers from the NASDAQ

- Compare two tickers using any attribute (Open, High, Low, Close, and Vol.)

Features

The app is capable of letting the user:

- Download price data from any of the 13 stocks
- View the price data of an individual ticker
- View the volume of tickers to show trends in the stock
- Compare two tickers
- View current price of stock as well as prices from up to 10 years ago
- Able to zoom in on specific date ranges on the graph for more accurate viewing
- Can take snapshots of the graph that can be downloaded and saved for later use
- Able to edit the plot data using plotly cloud

Technologies used

Programming Languages

Python

Python is a general purpose programming language with high level data structures and effective object-oriented programming. It is commonly used in data science applications because it is fast and easy to use when implementing complex functions. Python is compatible with many third-party frameworks and libraries such as pandas and flask. This makes development quick and it keeps our application lightweight.

Flask

Flask is a web framework written in python that uses an MVC architecture. Flask can be used for building compound, database directed websites. Just like python, it is easy to learn and use, allowing the developer to create web applications in a short period of time. Flask can also be called a micro-framework due to being directed at small projects.

Jinja

Jinja is a full feature template engine for Python and it is the default template engine when using flask. Jinja is used to pass derived data from the server to templates which are served to the client. This allows our pages to be loaded dynamically rather than containing static content.

Data Management

SQLite

SQLite is a lightweight, serverless, cross platform database engine. The database itself will be stored in an encrypted '.db' file and can be accessed and updated by a database driver. Since we used Python, we chose pythons native sqlite3 driver because it is simple and easy to use, and it provides us with all the necessary features such as querying and setting up transactions.

NumPy

NumPy is an open source python library for scientific computing written in python. Numpy provides multidimensional arrays and matrices which are faster than python lists, but not as flexible (only able to store same data type in each column).

Pandas

Written for python, Pandas is a library that can manipulate and analyse data. It provides data structures and operations for manipulating numerical tables and time series. Pandas allows us to easily convert between different data types which are required by the different views and plots that we implemented.

External Resources

Quandl API

Quandl is a financial data distribution site. They provide various APIs that allow people to access up to date stock information, as well as do complex analysis on them. We used Quandl only as a source of raw data, so that we could populate Provided us with the financial data to populate our database initially.

Plotly

Plotly is a data analysis and graphing tool. Plotly models can be shared and edited online and the data is accessible from the graph. The plots generated contain a wide variety of manipulation such as zooming, panning, and even advanced features like lasso select. The plots generated by plotly only need to be drawn once, after that the user can manipulate them and even export them as image files.

Development Process

Setup Environment

In order to start the server, it is necessary to have Python 2.7, as well as a recent version of pip (Python package manager) installed. When the server is started, it runs the ‘setup.py’ script which performs initial startup checks and operations. First, the setup will check if the necessary python dependencies are installed. If any packages are missing, the script will automatically install them using pip. After the required dependencies are checked, the setup will search for a database file under ‘./db/database.db’. This file is required for the website to run correctly and if it is missing, ‘setup_db.py’ will attempt to create a new one. This script will make the necessary calls to the Quandl API and populate a database with all of the required data. This new data will be up to date as it would be newly updated. This part of the setup can take up to 30 minutes depending on the user’s internet connection, so it is not advised to delete the provided ‘database.db’ file. Once the setup has been completed successfully, the server will start. By default, the website will be hosted on your local host, usually ‘http://127.0.0.1’ through port 8080. This can be changed later if necessary.

An in depth setup and installation guide can be found in our **README.md** file

Server

We used the Python Flask framework to create our server on which the web application will run. Flask uses an MVC architectural pattern to do navigation, build views and pass data. This means that our server will act as the controller, where all the data is processed, and requests are made and fulfilled. When a user requests a page, the server will look for the corresponding route which contains a function implementation that dictates how the input data is handled. The function is called, which will then perform the necessary database queries and calculations to prepare the values that should be sent to the template. The function then returns a call of the 'render_template()' function, which is built into flask's render templates module. This function takes a template name and data parameters as arguments, and looks for the requested template in the './templates' folder. The server will then execute the Jinja blocks found in the template, replacing them with the necessary data and return an HTML formatted webpage, which is then served to the client. If the user ever requests a page that does not exist, or they provide a query string that cannot be processed by the server, the '404: not found' page will be served instead.

Database

For the database we chose to use SQLite because it does not require a server to perform transactions and it is very lightweight. We simply install the sqlite3 Python driver and we can perform secure transactions without the need of a dedicated server. When writing our queries, we pay careful attention to the types of inputs that can be set as parameters to the database. We use prepared statements to protect against SQL injection attacks that can potentially compromise our database. Since we essentially only have 2 tables, the scale of our project is not very big, so SQLite serves well for us.

Views

When a client requests a URL for a page on the site, the server will use the route information to serve them the correct templates. For the front end we used html paired with the Jinja view engine. We use the html to build and structure our templates and style it with the Pure.css library. The data that we generate from our server is then passed to each template using Jinja blocks. For standalone data, we can use the '{{}}' notation to indicate that we are inserting data at that point, so the server can replace it before it is served to the client. If we want to programmatically display content, such as looping through a dataset or displaying a visualization, we can use the '{% %}' notation to indicate that we want the server to execute this block as code before it is served to the client.

- ('/'):
  - Displays a list of all tickers available to be viewed
- ('/about'):
  - Shows the project 'about' info
- ('/about/api'):
  - Shows instructions on how to use the project api. Used to pull price data
- ('/api'):
  - Endpoint for JSON api of the project
  - Explained more in the API section
- ('/stock'):
  - Displays the ticker price table selected

- The table contains the first 50 prices starting from the latest date
- The page has pagination, which allows users to go through each page by clicking a button or changing the page number in the URL

➔ (‘/graph’):
- Displays graph of selected ticker
- A volume indicator is available, if selected a volume graph will appear below the chart
- A date range option is also given, this allows for custom range charts
- Examples of charts **here**

➔ (‘/compare’):
- Two tickers and an attribute can be chosen to compare the price action

API

Our public API provides users with the ability to request data from our database. These requests are handled as routes in our server.. Before any queries are executed in the database, the query string that the user typed in will be parsed and checked for specific parameters. The database will only be queried if the necessary parameters are found. This ensures that we do not accidently process any malicious requests. When we query the database, we use a transaction system using prepared statements. This ensures that we do not execute any queries with injected SQL statements that could potentially destroy our data. Once we have retrieved the requested data, we check the contents of the returned object. If the object is empty, we return the 404 page because this means that the requested data did not exist in our database. If the returned object is not empty, we use Python's native json library functions to dump the data out as a JSON object. Finally, we return this object to the client so it can be further processed by the user.

When the user requests data, it will be in the form of a query string after typing the ‘/api/<ticker>’ URL. For our API, the user can request any one or combination of the following parameters:

➔ Rows
- Specifies the maximum number of rows returned

➔ Col
- Specifies which column should be returned

➔ Date-start
- Specifies the start date for the query

➔ Date-end
- Specifies the end date for the query.

ER Diagram

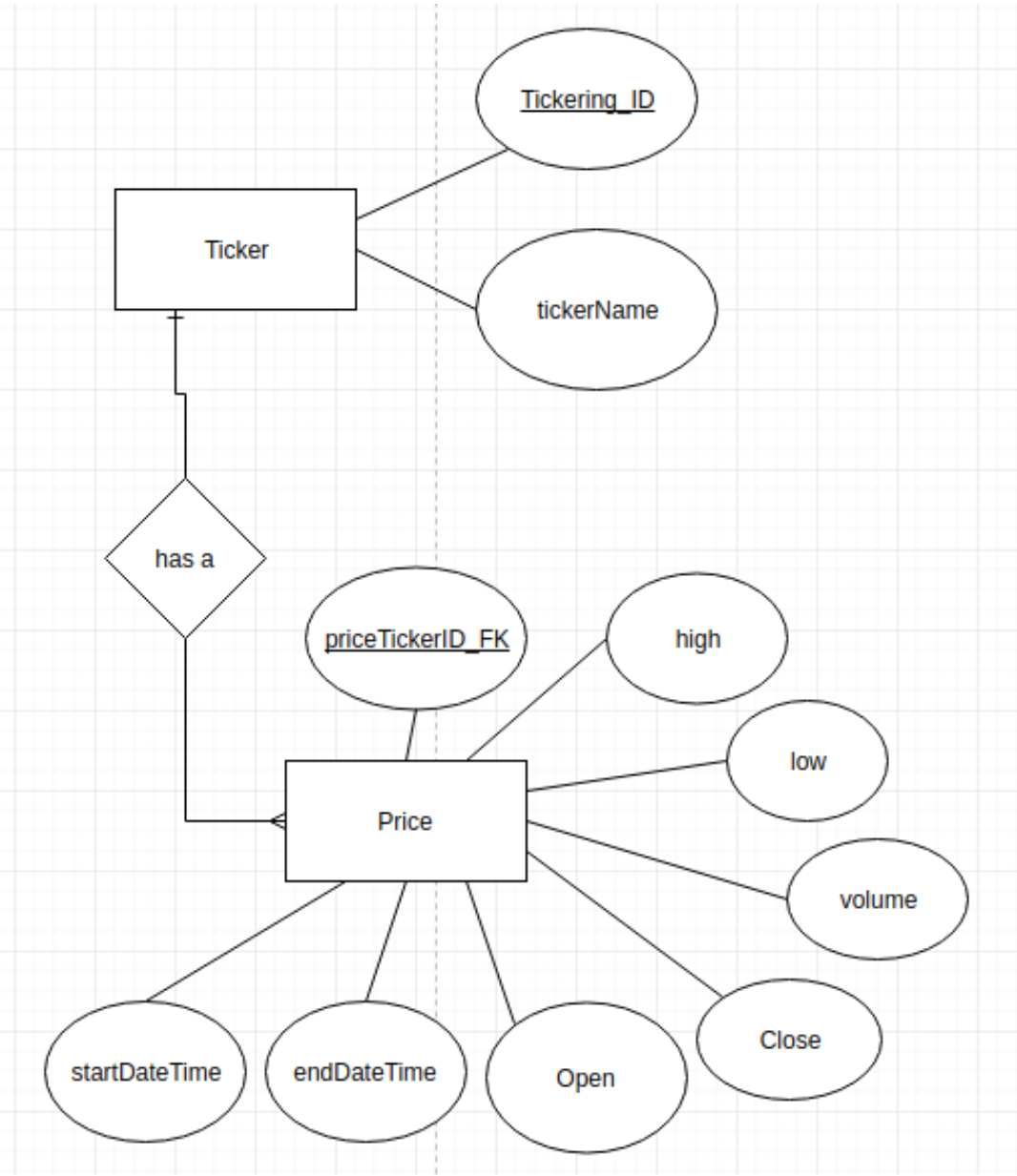


Image 1: Required only two tables since multiple values can be derived from the ticker table and price table

Tables

Tickers

	id	name	company
	Filter	Filter	Filter
1	1	GOOGL	Alphabet Inc Class A
2	2	MSFT	Microsoft Corporation
3	3	AAPL	Apple Inc.
4	4	AMZN	Amazon.com, Inc.
5	5	NVDA	NVIDIA Corporation
6	6	TSLA	Tesla Inc
7	7	INTC	Intel Corporation
8	8	IBM	IBM Common Stock
9	9	CSCO	Cisco Systems, Inc.
10	10	AMD	Advanced Micro Devices, Inc.
11	11	ORCL	Oracle Corporation
12	12	QCOM	QUALCOMM, Inc.
13	13	HPQ	Hewlett-Packard Inc.

Image 2: Ticker table contains “tickers” which also have associated ID’s and the full name of the company that the tickers are associated with

Prices

	price_id	ticker_id	date	open	high	low	close	volume
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	19370	1	2017-11-24	1054.39	1060.07	1051.92	1056.52	825342
2	21370	2	2017-11-24	83.01	83.43	82.78	83.26	7425503
3	23370	3	2017-11-24	175.1	175.5	174.6459	174.97	14026519
4	25370	4	2017-11-24	1160.7	1186.84	1160.7	1186	3526582
5	27370	5	2017-11-24	215.59	217	214.6	216.96	4517982
6	29370	6	2017-11-24	313.79	316.41	311	315.55	3242220
7	31236	7	2017-11-24	44.55	44.775	44.43	44.75	6465615
8	33236	8	2017-11-24	151.95	152.2	151.33	151.84	1193025
9	35236	9	2017-11-24	36.41	36.57	36.32	36.49	6155294
10	37236	10	2017-11-24	11.38	11.42	11.3	11.38	11033178
11	39236	11	2017-11-24	48.88	49.13	48.69	49.01	6008333
12	41236	12	2017-11-24	68.51	69.28	68.45	68.91	9233010
13	43236	13	2017-11-24	21.44	21.52	20.99	21.24	9657949
14	19371	1	2017-11-22	1051.16	1055.43	1047.25	1051.92	721498
15	21371	2	2017-11-22	83.83	83.9	83.04	83.11	20213704
16	23371	3	2017-11-22	173.36	175	173.05	174.96	24997274
17	25371	4	2017-11-22	1141	1160.27	1141	1156.16	3516336
18	27371	5	2017-11-22	217	217	213.61	214.93	8766814
19	29371	6	2017-11-22	316.77	317.42	311.84	312.6	4890091
20	31237	7	2017-11-22	44.94	44.95	44.535	44.65	19191241
21	33237	8	2017-11-22	152	152.39	151.33	151.77	3125416
22	35237	9	2017-11-22	36.7	36.715	36.36	36.45	16650130
23	37237	10	2017-11-22	11.41	11.49	11.3	11.37	23686100
24	39237	11	2017-11-22	48.56	48.83	48.42	48.58	10618868
25	41237	12	2017-11-22	66.37	68.26	66.24	68.13	15165652
26	43237	13	2017-11-22	21	21.53	20.59	21.34	28411338
27	19372	1	2017-11-21	1040.04	1050.39	1039.14	1050.3	1075568
28	21372	2	2017-11-21	82.74	83.84	82.74	83.72	21033981
29	23372	3	2017-11-21	170.78	173.7	170.78	173.14	24875471
30	25372	4	2017-11-21	1132.86	1140	1128.2	1139.49	2449503

Image 3: Price table contains financial data in which the tickers are sorted by date. They also contain the associated ID and the full name that is represented by the ticker

Queries

1. Retrieves 500 rows of relevant financial data for the ORCL ticker, which is the Oracle stock, ordered by price id. This query was used to get the data necessary for the graph view.

```
SELECT p.date, p.open, p.close, p.high, p.low, p.volume
FROM Prices AS p JOIN Tickers AS t ON t.id = p.ticker_id
WHERE p.ticker_id = 11 ORDER BY p.price_id ASC LIMIT 500
```

2. Gets 50 rows after a 200 row offset of relevant data for the NVDA ticker, which is the NVIDIA stock, as well as the corresponding row number, ordered by price_id. This is used to get rows of stock data for each stock page. When the next button is pressed, it executes this query again with a new offset. This way we can display 50 lines at a time.

```
SELECT (SELECT count(*) FROM prices AS t2 WHERE t2.price_id) AS t1.row,
t1.date, t1.open, t1.high, t1.low, t1.close, t1.volume FROM Prices AS t1
WHERE t1.ticker_id = 5 ORDER BY t1.price_id ASC LIMIT 50 OFFSET 200
```

3. Gets all rows of relevant data for QCOM ticker, which is the Qualcomm stock. This query is the default query used when a standard call is made to our public API.

```
SELECT p.date, p.open, p.high, p.low, p.close, p.volume FROM Prices AS p
WHERE p.ticker_id = (SELECT t.id FROM Tickers AS t WHERE t.name = “QCOM”)
```

4. Gets 400 rows of volume data after June 25, 2016 for the INTC ticker, which is the Intel Corporation stock. This is a modified version of the above query, where a user has made a call to our API specifying the number of rows, columns and the start date for the data.

```
SELECT p.volume FROM Prices AS p WHERE p.ticker_id =
 (SELECT t.id FROM Tickers AS t WHERE t.name = "INTC")
WHERE p.date >= "2016-25-11" LIMIT 400
```

API

Our api allows users to access the raw data from our database. You can specify how much data you want by changing start and end dates, as well as specifying the row count. You can also specify which column of data you want by giving a column parameter. This api allows users to take their analysis one step further and query entire datasets to use for calculations.

1. Requests 250 closes of Microsoft stock after November 10, 2015.

```
/api/MSFT?col=close&date-start=2015-11-10&rows=250
```

2. Requests all volumes of Tesla stock between February 1, 2016 and February 1, 2017.

```
/api/TSLA?col=volume&date-start=2016-2-1&date-end=2017-2-1
```

3. Requests 1000 rows of AMD stock before July 6, 2014.

```
/api/TSLA?date-end=2014-7-5&rows=1000
```

Visualization

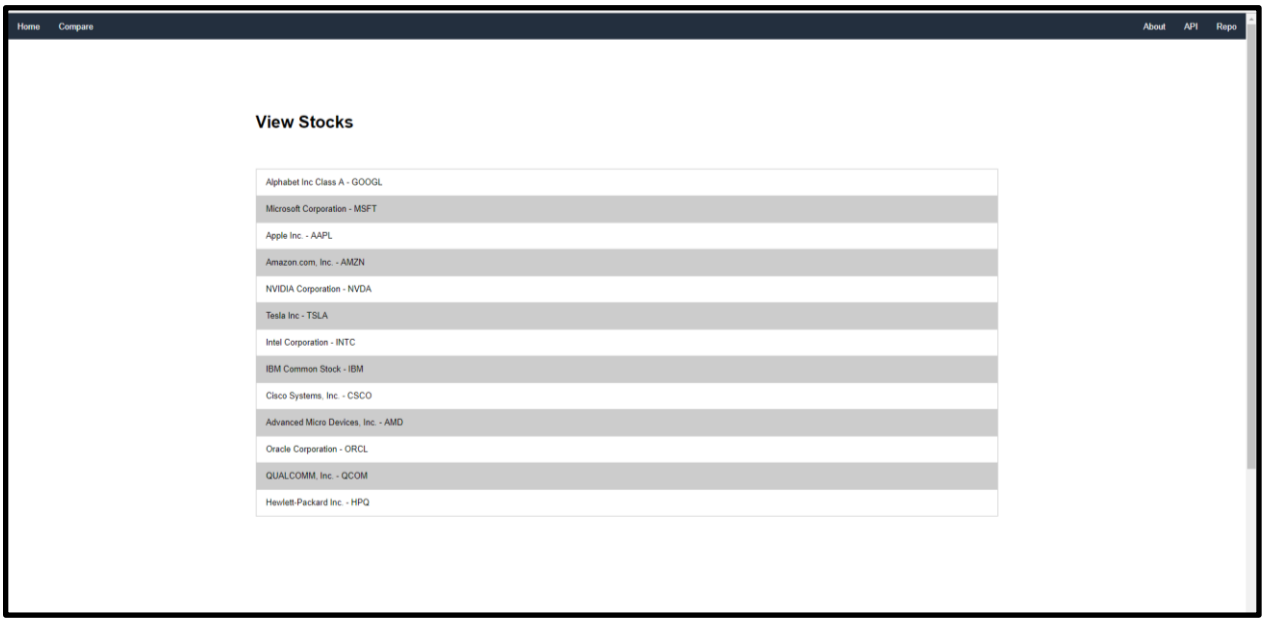
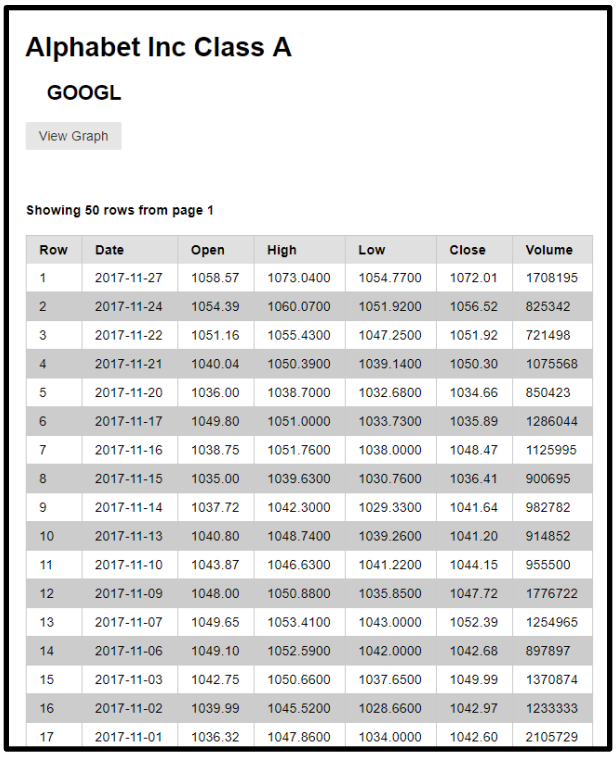


Image 4: Homepage of the site shows all the stocks that can be queried from the database



Alphabet Inc Class A

GOOGL

View Graph

Showing 50 rows from page 1

Row	Date	Open	High	Low	Close	Volume
1	2017-11-27	1058.57	1073.0400	1054.7700	1072.01	1708195
2	2017-11-24	1054.39	1060.0700	1051.9200	1056.52	825342
3	2017-11-22	1051.16	1055.4300	1047.2500	1051.92	721498
4	2017-11-21	1040.04	1050.3900	1039.1400	1050.30	1075568
5	2017-11-20	1036.00	1038.7000	1032.6800	1034.66	850423
6	2017-11-17	1049.80	1051.0000	1033.7300	1035.89	1286044
7	2017-11-16	1038.75	1051.7600	1038.0000	1048.47	1125995
8	2017-11-15	1035.00	1039.6300	1030.7600	1036.41	900695
9	2017-11-14	1037.72	1042.3000	1029.3300	1041.64	982782
10	2017-11-13	1040.80	1048.7400	1039.2600	1041.20	914852
11	2017-11-10	1043.87	1046.6300	1041.2200	1044.15	955500
12	2017-11-09	1048.00	1050.8800	1035.8500	1047.72	1776722
13	2017-11-07	1049.65	1053.4100	1043.0000	1052.39	1254965
14	2017-11-06	1049.10	1052.5900	1042.0000	1042.68	897897
15	2017-11-03	1042.75	1050.6600	1037.6500	1049.99	1370874
16	2017-11-02	1039.99	1045.5200	1028.6600	1042.97	1233333
17	2017-11-01	1036.32	1047.8600	1034.0000	1042.60	2105729

Image 5: Once a stock is selected, the user is shown the 50 rows of data in table format. If the user presses the next button, the next 50 rows will be loaded.

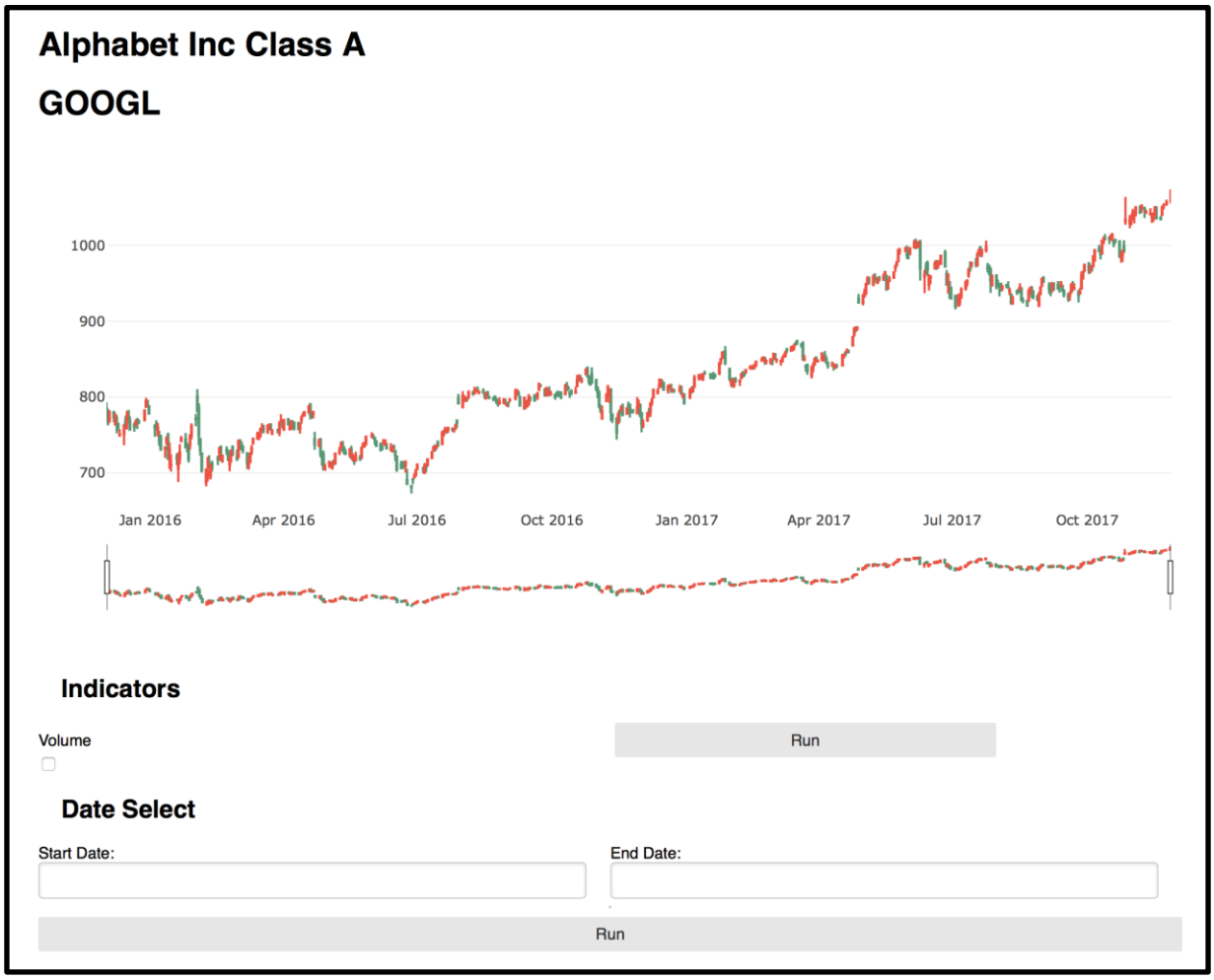


Image 6: Users can view a graph of the selected company showing the change in stock data in the past year

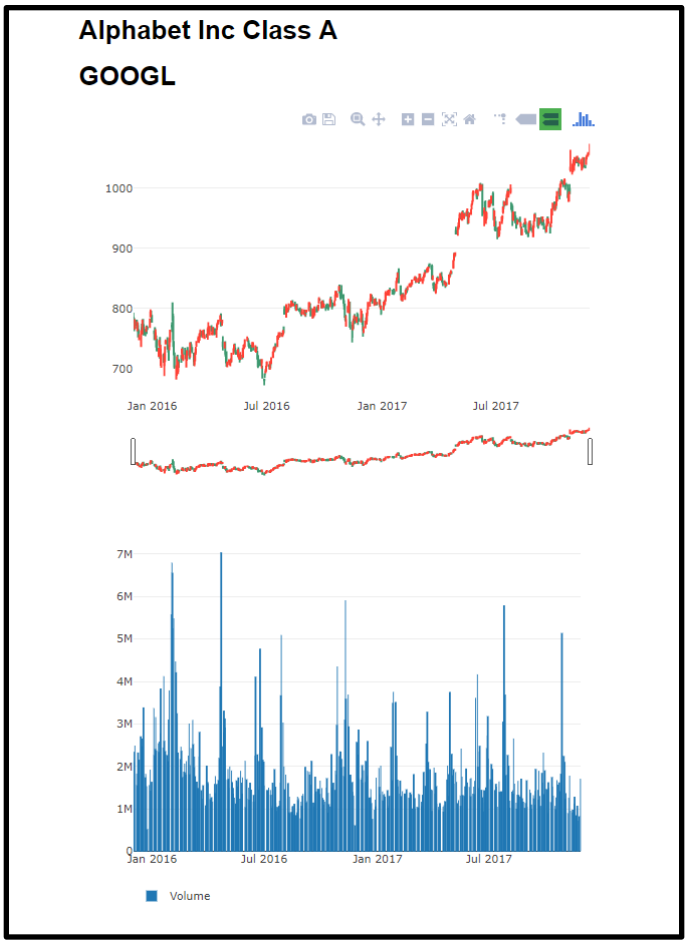


Image 7: Users can use the volume indicator to get a look at the volume of stock that is being bought/sold.

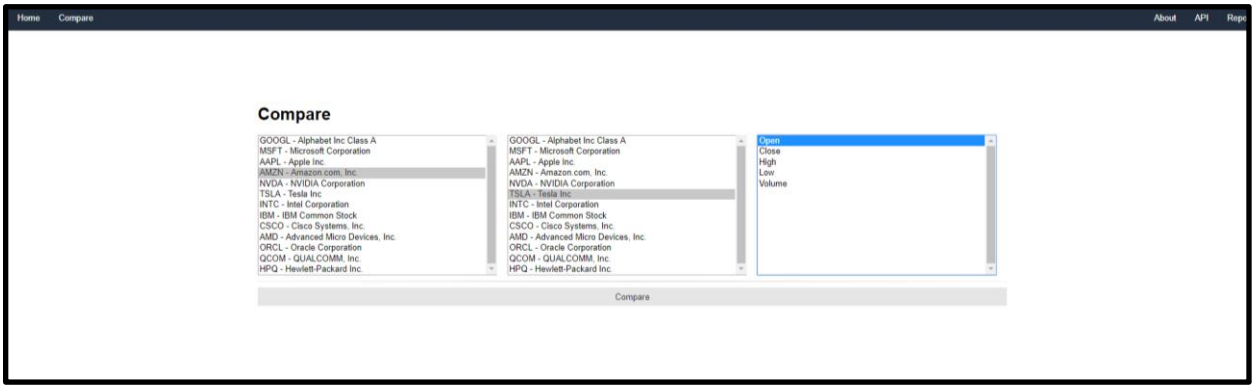


Image 8: Users can select the compare button at the top left of the site and are given the option to select the two companies they wish to compare and what views they wish to see.

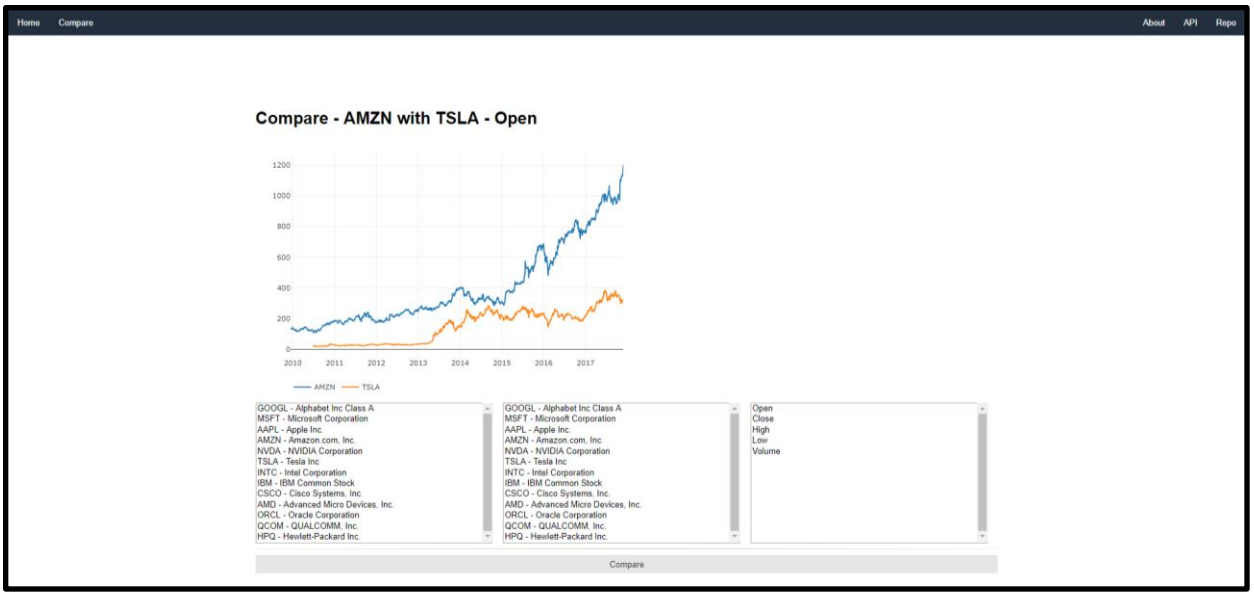


Image 9: Users are then given a graph that displays both companies that they selected to compare. They are also given the option to continue comparing stocks without having to go back to the previous page or re-selecting the compare button.

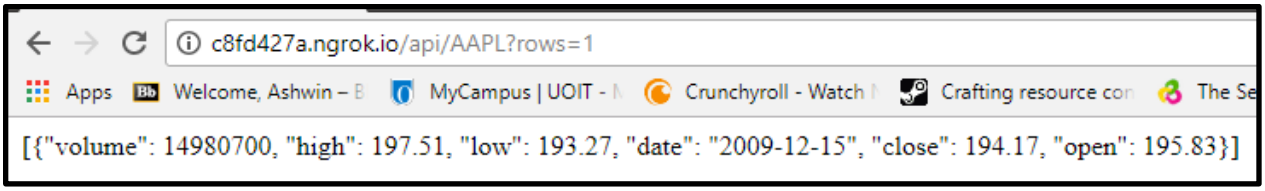


Image 10: Example of our API being used to retrieve some JSON data

Future Work

In the future, some basic additions to the project that we can make are indicators, and real time data. We foresee the implementation of a Machine Learning library to analyze the price data to forecast values for company reports, and potential price moves. This can also be fitted to price action to “predict” future price with a probability score.

Conclusion

In conclusion, our project was fairly successful. The project consisted of a majority of the goals we set out to achieve. It is capable of outputting ticker charts, and has a dedicated api for users to use for their own purpose. The course helped a lot in designing the backend of the site, as well as error checking the sql queries.

Some goals were not met such as multiple indicators, and a real-time data feed. There is room to implement these functions. The project is set up in a very modular fashion, allowing future implementation. Another major function that is highly likely to be implemented in the future is a machine learning based system, that allows for price data analysis. This would be neat when trying to visualize potential future trends.

Individual Contributions

Name	Contribution	Student Number
Kaushal Patel	- Charting - Layout and UI - Data cleansing	100586212
Ashwin Kamalakannan	- Setup (Flask, DB) - Layout and UI - API	100584423
Darron Singh	- Making queries - Report - Presentation	100584624
Jasindan Rasalingam	- Getting data from external API - Report - Presentation	100584595

References

1. Quandl API: https://www.quandl.com/tools/api
2. Plotly: https://plot.ly
3. Github: https://github.com/AshwinK97/SOFE3700-FinalProject