David Reider & Tristan Staso

*Widener University*

Computer Science Department

*Trumptweet.io* Project Report

Friday May 1st, 2020

TrumpTweet.io: An Interactive Web Application

**Abstract**

TrumpTweet.io is an interactive web application that is designed to analyze the times that President Donald Trump tweets on Twitter.com and overlay it on a stock market graph. The web application will add indicators that show significant changes in stock price, acting as a signal that President Trump may have influenced a change in the stock market with a tweet. Then, hovering over an indicator will show tweets from the marked date, to show what may have caused a change.

**Introduction**

Our growing interest in stock trading and following various markets like cryptocurrencies, futures, and option contracts paired with our critically acclaimed president brought this project to surface. The main goal of this application was to correlate tweets posted by Donald Trump to major movements in the stock market, allowing a user or someone using the tool to draw a conclusion on whether or not that supposed tweet made an impact on the market. To put it simply, a tweet will be shown on a timeline, which essentially is an overlay on top of the stock market, mainly ETFs like the S&P 500 (SPY) or the DOW Jones (DJI). To expand on this even further, another main goal for the project was to make the tool work with any stock ticker, which was accomplished. Any ticker on the NASDAQ can be input and the tool with graphs and overlay the tweets accordingly.

A project like this was tackled because we wanted to work on something that was relevant to our own interests but also practical. We had previously proposed to work on a video game, but quickly turned our heads at it given that it could be more work for a years time frame and that would fit into an oversaturated pool of other senior projects.

**Background**

The idea being TrumpTweet.io: An Interactive Web Application was brought about due to the interest in the stock market that my partner and I shared when we met in university. As freshmen in a new environment, in our first computer science class, we met since we sat next to each other. As the friendship grew, we learned more about our interests and found out that we shared an interest in the stock market. This interest was not strongly brought up until junior year when we became roommates and eventually both talked about investing.

After researching the stock market, we found information claiming that President Donald Trump heavily influenced the stock market simply by using his Twitter. We both found this aspect intriguing, and decided to do more digging. When senior year rolled around, we thought of ideas for our senior project. For the project we had the choice to make any computer science related project, such as a video game, a mobile application, or anything else that we could imagine. At first, we were considering a game, but in the end, we decided to rekindle our interest in President Donald Trump and his influence over the stock market by tweeting.

After deciding to explore the correlation between his tweets and the stock market, we had to figure out how to show the connection in our own way. We also wanted others to be able to see the relationship that we were beginning to explore. Due to the ‘want’ of other people to be able to see a connection, we narrowed it down to creating a downloadable program that would need to be saved on a machine and run by the user or hosting an online web application.

We did not have knowledge of creating a web application, and we only knew how to write a program to show the correlation, but could not think of ways to share it. We also thought that creating a downloadable program would be too complicated and not create enough interest to entice the user to download it to see. This led us to choose in creating a web application, despite not having prior experience creating and hosting an online website.

With no knowledge in what it took to make a website, research into what needed to be learned had to occur before anything else in our project. Also, we are both not front-end developers that focus on creating a good user interface, so we had to create an appealing design that would look good and research good color schemes and design techniques to make the website look as best as it could with minimal prior knowledge.

**Main (Body)**

TrumpTweet.io is an interactive web application that applies the common languages and practices of web development. These programming languages include Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), and Javascript (JS) with some accompanying libraries and tools. The web application also requires a database to host tweet information, which we used an online database called Firebase. Finally, an application programming interface (API) is needed to extract stock market data that is manipulatable to our charts and other uses.

To create the elements that appear on the web application on the screen of the user, we had to use a language called HTML (HyperText Markup Language). HTML is considered the ‘building blocks’ of the web. It creates the structure of the web page by adding elements such as containers or paragraphs to a page. For example, the search bar appears in the top left of our web application due to HTML, and the functionality is assisted by other languages. HTML is generally assisted by other languages for further functionality or style by languages such as Cascading Style Sheets and Javascript. We both learned about this programming language and were able to create a basic but functional web page for this project.

Cascading Style Sheets (CSS) is a language that describes the style of HTML elements. It is considered the ‘adjectives’ of a web page. For example, our search bar and the submit button are the size that they are due to CSS containing them to their current sizes. Another example would be to consider the font of the header “TrumpTweet.io”. It would be the default text of HTML if it were not for CSS describing the font and changing it. We use some CSS to style the page so that the text and colors are how we like, such as the blue of the chart matching the twitter blue. We also use the styling to size some of the elements to all fit onto the screen.

Javascript is another language that assists HTML. Javascript can be a front-end language that does work that the user of a web application would see, but it also can be a back-end language that does work behind the scenes of a web page, such as math or other analytical processes. For example, our project uses a Javascript library called Charts.js that creates a chart on the screen for the user to see. This is a front-end use of the language. A back-end or functional use of the language would be when the user types into the search bar and clicks submit, Javascript will take the string that the user entered and submitted, create a new string that calls and retrieves the stock information from AlphaVantage for Charts.js to then display. We use Javascript for this purpose, as well as for our analysis behind the scenes. The analysis includes finding the percent difference between two days, and seeing if it is greater than a predetermined threshold value that is based on the price range of the stock. Then, if it is a greater percent change than the threshold, an indicator is added to the chart. To get the annotation holding a tweet to appear on the screen upon hovering over an indicator, that is also a mix of HTML and Javascript.

Storing data was something that was initially glossed over at the beginning of the project's development. It was assumed that there would not have to be any data stored given that this will be an online application working with real-time data sets and no sorts of localized user base. This was something that, if given the funds and additional time, would be hosted on a server and call other API’s and databases to display information in our application. This idea of not having to store data was debunked about halfway into our project. We realized that one of our initially planned implementations, the Twitter API, would not work with our current project specifications. To successfully call the API and have it return the tweet data that was crucial to our application, we would need to have a backend script written in Node.js running on a backend server, which unfortunately we did not have at the time. One of the reasons for this is because Twitter had made changes to their API and introduced OAuth, which would not let us make calls to the API from the client side, most likely for privacy purposes and to deter potential abuse of the system.

Enter FireBase, a free online tool curated by Google that would allow us to host data in JSON format gave us the ability to call that data internally from our web app. One of the main reasons that FireBase was selected to use is that it allows the client side code to reference the stored data on FireBase directly. So whatever large amount of data that was stored we could easily use to implement onto our application. The main reason for the initially proposed Twitter API was to call it to retrieve Donald Trump's tweets in real time, which was one of the main goals of the project initially. As mentioned in previous parts of this report, this would have to be moved to future work and development given its complexity thanks to Twitter.

We implemented FireBase by first dumping a whole lot of tweets found from an online archive (trumptwitterarchive.com) into the callable database format that FireBase provided. The meta-data included with the actual tweet worked out in our favor because it included the actual content of the tweet as well as the tweet id. An even better functionality from the archive is that it seperated actual tweets from retweets, which is ideal in this case because we wanted to include original tweets only. From these stored tweets, we can index them and retrieve the ones we need based on the date. The only problem with this method of implementation is that there is no real time retrieval of tweets, the only tweets available are the ones that we manually push to FireBase. Knowing this, we still decided to move forward with it for practicality’s sake.

For our charting library, we ultimately decided to use Chart.js because of its easy implementation onto our webpage and its plethora of features that we could manipulate to achieve the project goals, the most important being the custom points marked on the graph. Before this, however, there was a previous library called LightweightCharts provided for free use by tradingview.com. The LightweightCharts.js library is built around charting financial changes for stock market prices. This fact paired with our own familiarity with it charting trends and using it to make investment plans as a side hobby, gave us the incentive to use it for our project. We soon found out that LightweightCharts would not support the feature that we were looking for, specifically plotting a custom point to use as an indicator, and also give us a hover action that would display the tweet when a user moused over it.

With our dilemma at hand, we researched charts.js and made an almost effortless switch implementing it into our page. The new library allowed us to plot the Donald Trump head (which is visible in our most recent screenshot) which signifies where one of his tweets shows up on the stock price timeline. The chart functionality is almost identical to that of LightweightCharts, being that it is a simple line chart, and we were able to easily plot the data from AlphaVantage onto the chart container. One of the default features of LightweightCharts was scalability, allowing a user to zoom into parts of the chart to enlarge them and take a closer look. Charts.js did not give us this out of the box, and further research is required to find out how this can be made possible with the new library. This is elaborated on in our future work and development section.

One of the main data sources for this project was AlphaVantage. They are an online database for real time stock prices, mainly for use for embedding into other API’s or applications. This is exactly what our project called for so it was decided that AlphaVantage would be our source for stock prices. These prices would be directly referenced by our charting library, Charts.js. One of the advantages of AlphaVantage was that their data can be accessed by a simple HTML request. Their API structures the stock data in JSON format, which is perfect for our implementation into charts.js. AlphaVantage gave us an API key for their databases, they limit us to only ten thousand calls a day, which is more than enough for our educational purposes. To make a call to this API, we build a URL containing our API key, the stock ticker that we want prices for, and the timeframe that we want the prices to be laid out across. For simplicity’s sake while developing this project, we used the default time frame of yearly to test and advance our progress.

To interface the JSON that AlphaVantage provides us when we execute an API call, we use an extension of JavaScript called jQuery. jQuery lets us read the JSON straight from our client side JavaScript code and manipulate it as we wish. So after a call to AlphaVantages API is executed using a URL we can then read the stock prices easily using list comprehensions. For example, we use the first price returned by the API in the timeframe specified to be our first point on the leftmost side of the chart, and fill the chart accordingly. Depending on which kind of timeframe we specify, AlphaVantage will give us the stock prices on that interval. These values returned are locally stored on the client side, as this project in its current state has runs only on the host computer, so everytime you make a new call by entering a ticker, the old data is overwritten and you have now used two calls to the API. Ideally, what we would want to have is that data from the first call is cached once a new one is made, so that you are not making redundant calls. Given more time, it might be possible to have this section of our code executed on the server side, saving resources on the computer that access the page, and then cache that data so that future users are not causing the API to be called more than once for that ticker.

After the researching and learning of the aforementioned languages and tools, we began the development process. This process began with the learning, and led into a design process. To come up with a design, we originally sketched out and debated multiple ideas of how we would like the project to look and function. This process was done multiple times throughout the duration of the project and will likely continue as new aspects are added and refined. We had many changes over time and will likely continuously change the design because there are always ways to improve or be more efficient, so it requires constant evolution.

The next aspect of the development process was to create a timeline of what we would like to accomplish by certain dates. Of course, not everything always goes according to plan when developing and creating a project that we have not done before, so the deadlines were not truly deadlines and were flexible, but were ideal marks to hit along the way. After the timeline was set, we were able to see a rough idea of how the project would come together. From afar, our timeline was fluid, we ran into obstacles that required us to scratch a whole library or idea and then reevaluate our options, pick something that would work, and then implement it to work with the rest of the project. One example of this, that is later described in detail, was the chart that we needed to display on the webpage. The initial chart did not support the functions that we needed it to have, and this was not found out until we needed to implement that function. So, we scrapped the first chart that we had worked on and implemented, and then researched and found a new one that would hopefully do what we wanted it to do. Thankfully, it did, but it set us back a little bit in our timeline, but from then on things went smoothly.

In the first few months after the project was assigned, most of our time was used to research functionalities of web applications and languages that we would need to use to achieve the main goals of the project. Lots of experimenting was done to determine what kind of code libraries would work in our project. We also had to research API’s for our source of data. This was not too difficult, as we had some experience with getting data from an outside source and scraping data off of other websites. We were lucky to have our one API, AlphaVantage, the source for our stock data, work from all the way in the beginning of the project to what the current state of the project is in now. There were almost no complications whatsoever when dealing with this one, mostly because its function is simple, but nonetheless this worked out in our favor.

After the design and timeline was determined, we began on what we had laid out to do on the timeline. The first part of the development was set to just get a functioning web application on the screen, and then to worry about polishing it up and making it look more appealing later.

Once the ‘prototype’ was functional, we began to polish the look of the application. We applied the earlier conceptualized design and attempted to scheme the project to be similar to the design, while considering functionality as well as the user interface.

Throughout the project, we worked close together to help each other whenever we were stuck. We worked on the project when we had time, and since we were roommates, we had easy access to each other as a reference. Often, we bounced ideas off of each other and developed the project in a free-flowing manner.

One of the resources we used to share code and keep the project up to date and allow for us to work simultaneously together was through the use of GitHub.com, a website designed to host repositories of projects. The website allows for a quick upload of updated work, and a quick download of the latest version of a project on the website. Also, the website allows for ‘branches’ of a project, where multiple ideas can be worked on at once and then merged together at a later time. This allows for control over what makes it to the master version.

When working on a larger project that incorporates many different components, libraries, programming languages, etc. there are bound to be some challenges and changes that occur along the development cycle. Some of the challenges during development include a struggle with Twitter API, not hosting the website on a server, not being able to use a local database and some other smaller challenges that were resolved along the way.

Most of our challenges were due to not actively hosting the project on a server at the moment. For example, we could not make Twitter API calls from the browser, because Twitter API requires a server. Due to this, we had to pre-download the tweet information we are using and store it into a database. To access this database, we then had to host it online using a free service, Firebase. We couldn’t host the database locally because browsers do not allow access to local databases for safety reasons. We also couldn’t host the database ourselves as we did not have a server set up.

We also faced some other challenges that we resolved with some critical thinking and problem-solving skills. This includes a problem we had with our charting library. At first, we had implemented a library that worked with our Stock API and allowed us to chart the stock prices as we wanted, but we later found out some limitations with the library that did not allow us to easily create indicators on the stock. This caused us to have to change charting libraries, which took a good amount of time as we needed to get used to the new documentation and manipulate it to work within our code. There were some other challenges that we encountered along the way, but were not too difficult to solve together.

**Future Work**

While implementing new features into our application, we came to the realization that our project was not truly a “web-application” if it is not hosted somewhere and available from any computer. The decision to implement a true backend came about when we crossed paths with the inability to call the Twitter API straight from the browser. We needed a backend server to make these calls for us which we did not have. About 3 months before the project was due, we decided not to begin to set up a backend server. Instead, we decided that we needed to have the application purely client side while using data that was either provided from our own databases or from an API that allows us to cross-script to return information. This was the case with AlphaVantage, our stock price API that we used. There were no problems involved with retrieving data directly through our application code, however with Twitter it was a different story because of some of Twitter’s restrictions and limitations when calling their API.

This brings our project to another stage. If there had more time to fully develop it into a full-scaled web application, a server would be set up for the back-end API calls to reference Twitter’s API and even implement other parts of our code that do not necessarily need to be on the client side. For example, the API keys to AlphaVantage are currently exposed in the client-side code simply because there is nowhere else to put them. Obviously having a personal API key exposed is not ideal given that someone could take it and abuse the system. In a nutshell, the future work on this project would involve cutting up the current code to put it on a server and implementing Twitter’s API so that the application could work in real time without having to manually update a custom database with the current tweets on the backend server side. Essentially, we would be taking the current build and making it more practical.

Other options to improve our project down the road would include a ton of quality of life updates. As of now, our chart does not scale correctly when resizing the window nor does it scale when you use a browser’s zoom function. We would also like to improve the options for viewing on the chart, for example right now the chart shows the past year and that's it. There is no zooming to a certain week or a certain month, which would be nice to be able to do. Also, we would like to change the time period between price points from daily, which is the current setting, to a user-choice of their own time frame, such as minutely, every 5 minutes, every hour, etc. This would likely not be hard to implement, but it is currently on the list for things to add and improve. The same goes for the other direction, it would be useful to be able to view the chart on a multiple year basis, given that Donald Trump has been in office for four years.

Finally, an additional update that we would like to implement would be an analysis of the content that Donald Trump tweets about. For example, we would pull all of the dates that got indicators, then we would collect the content of all of the tweets created on that date and analyze the words. The study of the words would include seeing which words are included most, to see if there are certain ‘triggers’ that cause the market to move. Some examples of ‘trigger’ words could include: ‘China’, ‘Tariffs’, ‘Impeachment’, ‘COVID-19’, and ‘Economy’. Also, we would consider the quantity of tweets per day, to see if higher frequency dates cause more or less change.

**Conclusion**

Overall, TrumpTweet.io has turned out as intended thus far. The timeline was accurate along the way, even with any of the problems or challenges we faced along the way. Intentionally, we built in time for any problems or challenges, as to not completely stress if something went wrong or had to be changed or redone.

TrumpTweet.io in its current state is a functioning web application that will have live stock data displayed on the screen, allow the user to input any stock ticker they would like to see, and populate the screen with indicators for big changes to display tweets on. This is the intended functionality of the project; however, as seen in the Future Works section, there are still some improvements to be added to the project, such as actually hosting it for anyone to use, displaying all tweets on a specific day instead of a random tweet from the day, and getting Twitter API to work so we do not have to update the database of tweets manually.

This project has taught us a lot of valuable lessons and skills. First of all, the project required cooperation and teamwork. Working as a team on a project is a valuable skill because in the real-life workplace, often a computer science employee will be a part of a team to begin, so they will need to be able to work as a team. Teamwork includes skills such as clear and concise communication, the ability to project ideas, and cooperation. Cooperation is valuable because it allows for negotiations to occur to decide and make final decisions on different aspects of the project. Throughout this project, we had to cooperate and negotiate over different aspects of the project, such as the color scheme or layout of the web application.

Also, the project has taught us both the development cycle of a project. Beginning with planning and research, we have learned it is important to lay out what the project should be, starting with the idea and intended features, to the timeline of working on the project and milestones to reach along the way, to finally implementing the project. This is important with any project and translates to the real workforce. A final aspect once the project is ‘finished’ will be maintaining the website and keeping it up to date and functioning.

Another thing to note while developing this project, is that there will almost always be some kind of rough patch in the process. That being said, we encountered a few different ones while trying to tie everything together. The first being the chart, we had to scrap the initial one and rebuild our code onto another, which was not difficult but still set us back. The second was more intricate. We learned that we could not use the Twitter API with the current setup, so we either had to build a whole new component to the project (i.e. a web server) or make the project less intuitive but keep its functionality. Given our limited time frame, we opted to downgrade what the application could do while retaining its functionality. These types of dilemmas were a huge part of our learning process and taught us how to work past these kinds of things while making a compromise.

Finally, the project has taught us the ability to use some new programming languages. Beginning with researching the languages and technologies required to implement the project the way we intended, to actually putting the languages into practice. We mainly have learned some basic HTML, CSS and Javascript, along with some libraries with Javascript. Since we did not know the languages previously, we had to learn about reading documentation and how to use and manipulate the documentation to do what we wanted instead of the ‘cookie-cutter’ basic implementation. Learning and implementing a new language can be important when entering the workforce as we may be asked to learn a new stack of languages to use for the employer’s projects, and this project has taught us how to do that.

**Sources**

<https://github.com/>

<https://developer.mozilla.org/en-US/docs/Web/HTML>

<https://www.w3schools.com/html/>

<https://www.w3schools.com/css/>

<https://www.javascript.com/learn/strings>

<https://www.w3schools.com/js/>

<https://www.chartjs.org/>

<https://api.jquery.com/>

<https://www.alphavantage.co/>

<https://www.youtube.com/>

<https://stackoverflow.com/>

<https://www.howtogeek.com/>