VISION : A Business Insights Tool based on Twitter Sentiment Analysis

Jagadeeshwaran Raja Umashankar
SID: 40126184
Master of Engineering
Electrical and Computer Engineering
j_rajaum@encs.concordia.ca

Nivedha Ramesh
SID: 40128111
Master of Engineering
Electrical and Computer Engineering
r_nivedh@encs.concordia.ca

Jad Saad
SID: 40084673
Master of Engineering
Electrical and Computer Engineering
ja_saa@encs.concordia.ca

I. PROBLEM STATEMENT

The popularity of Twitter as a micro-blogging social platform has increased tremendously in the last couple of years. With more than 500 million tweets a day exchanged by about 152 million users [1], Twitter has become one of the most used micro-blogging social medium on the web. These tweets regularly contain information about a product of brand, which might influence the opinion of others. Understanding how consumers perceive brands is fundamental to much of marketing strategy. From a data analysis point of view, tons of information that expresses people's feelings, opinions, feedback and thinking are published and exchanged among users every day through Twitter.

Consumer insight sentiment based on social media sources may impact reputation in both positive or negative ways can be an effective tool for an organization's strategy, plans and marketing enhancement schemes. We propose a solution for business clients, to continuously track all relevant Twitter content about their brand and products in real-time, analyze the data, and visualize the sentiments relating to products or sentiments, reputation monitoring, result prediction and decision making. By monitoring brand mentions on Twitter, businesses could inform engagement and deliver better experiences for their customers across the world.

II. PROJECT GOALS

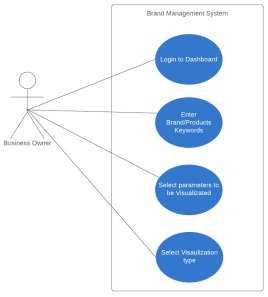

Fig. 1. VISION ICDE Usecase Diagram.

A. User Stories

1) As a new user, i shall be able to create a new account with management access and sign in to view my dashboard.
2) As a user with manager access i should be able to create a user account for an employee I'm managing and give them access to the dashboard (e.g. senior sales member).
3) The user shall also be able to select the type of the visualization charts and select the metrics that I wants to visualize, like the sentiment of the twitter users, the follower trend, the retweet count, word cloud and many more.
4) As a user i shall see the charts update every certain duration so they stay up-to-date.

B. User Requirements

1) The user should be able to sign in or register to authenticate to his dashboard
2) The Forgot Password feature should allow the user to reset his password as the need maybe
3) As the user enters the subject of interest (hashtags related to the products or the brand), the predictive text feature suggests popular keywords or keywords from his previous searches
4) The user should be able to select from a predefined list of chart types and the respective tweet metric that he wants to be able to visualize
5) The user preferences need to be saved and automatically reloaded on his next log in session

III. ARCHITECTURAL DESIGN AND SOFTWARE PROCESSES

A. High level architecture

Based on the user stories and the documented user and system requirements, a high level architecture diagram was created. Fig 2 shows an abstract model of the architecture of our twitter based brand insights app. Its uses a data grabbing module to stream tweets from twitter and the processing module analyzes the customer sentiment. The back-end server stores the user credentials and the processed data in the database and also fetches data to be displayed on the front end system.

This high level model explains the distribution of the different modules and the relationship between the different systems.

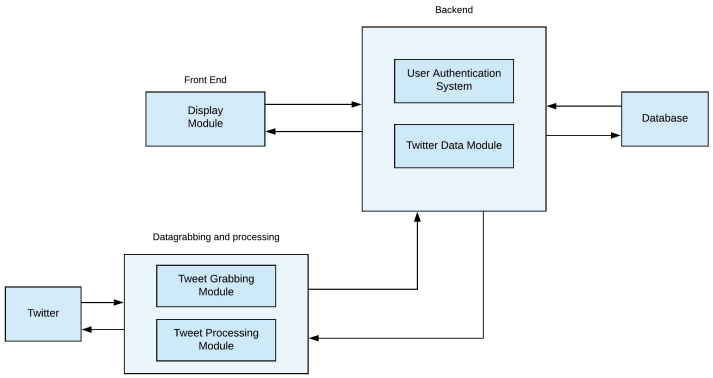

Fig. 2. Architecture of the system

B. Architectural Design Decisions

After modeling an abstract view of the system architecture, the team had to make certain architectural decisions to proceed with designing each component.

The problem was thoroughly discussed with the stakeholder and solution was analysed to be feasible and a lower cost of implementation. Our design decisions were documented.

Some of the important design decisions we had to make over the course are as follows:

1) **Problem Statement:** How will the system be distributed across hardware cores or processors?

   Solution: The application being data intensive we also had to account for the Performance of the system together with the functional requirements. Hence it was sensible to not run our front-end an back-end code on a single server. We have three different server, one hosting the frontend application, the second consisting of the data grabbing system and finally the mongoDB database is hosted on the cloud.

   Pros: Efficient management of different systems and better performance

   Cons: Increased Interfacing complexity

2) **Problem Statement:** Efficiently implementing the Twitter data component

   Solution: We chose to use tweepy, a python library, to access twitter's official api. It provides option for both streaming and searching for tweets. Our other choice was to use web scraper or other python libraries which did not use official twitter api.

   Pros: Reduced redundant coding when used instead of web scraping tools. Allowed reuse of existing library. Hence it made the code modular and reduced implementation time.

   Cons: The twitter official api has a rate limit of 15 requests per window per access token. The number of tweets that can be fetched is limited to 100.

3) **Problem Statement:** Software stack for efficiently implementing Frontend.

   Solution: This particular decision was crucial as we had to factor in ways of interfacing the front end with the back end. Any refactoring at a later stage would disrupt the entire system because changes would have to be made in both the frontend and backend parts. After careful evaluation, we decided to use React.js, Node.js along with HTML and CSS to implement the webpages and forms.

   Pros: Efficiently tailored for frontend implementation, unlike python flask which was our other alternative. Easier to integrate with third party visualization tools.

C. Architectural Design

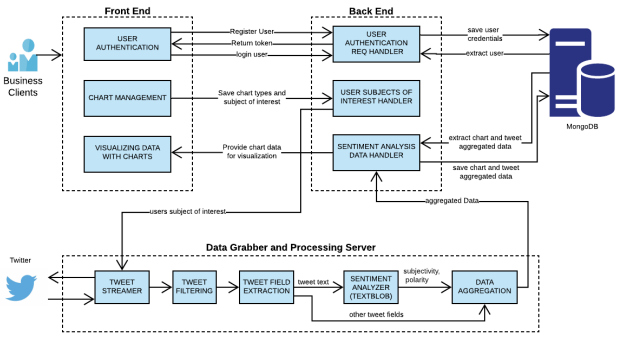

Fig. 3. VISION ICDE System Level Architecture.

Before dwelling in to the granularity of each component, it was important to decide how to decompose the components into sub-components.

The tweet cleaning and the sentiment analysis is crucial to the system. To design our data grabbing and processing system, we have used Tweepy, a python library to access the official Twitter API. So the grabbing module will create an API object, that authenticates with Twitter. The keywords required to search Twitter is received from the back-end server and the API object fetches data from Twitter using this. Further the obtained Tweet objects are sent to the Processing module, which comprises of a data cleaning and sentiment analyzer. The Tweet object is broken down to extract the text and other relevant data. Any emoji's or URLs or other irrelevant content in the text is filtered and the filtered data is given to the

sentiment analyzer. To analyze the sentiment we use Textblob, a python library that can quantify text as subjective or objective and gauge the sentiment. The processed data is sent to the back-end server.

On the front-end server, we've majorly used React.js, Node.js and Express.js frameworks to design our dashboard. Separate pages are created to incorporate the User sign-in or sign-up functionality and the dashboard page to visualize our data. The user credentials entered during registration will be sent to the back-end server and then validated during sign-in. The keywords entered by a specific user is sent to the back-end server, which renders the data to be visualized. The charts for visualization have been embedded from Recharts, a charting library built with React and D3.

The back-end server handles all the communication between the front-end system / data grabbing system and the database. It consists of the user authentication handle that stores the user details in the database and later retrieves the same upon request by the front-end system. It also updates the database with the subjects of interest of a specific user. The sentiment processing handle invoked by the data grabbing system, updates the database with processed values, which is then rendered to the front-end system, upon request.

To further describe our architectural views, we have used class and state diagrams for the logical view, sequence and activity diagrams for the process view.

D. Revisions to Architectural Designs

We have carefully scrutinized all our design decisions and have decided upon the system architecture as seen in Fig 3. This design meets our current functional and non-functional requirements and hence we will not be making any amends to it.

But to account for evolution, and scaling up existing functionalities, we could use dockers to enhance tweet grabbing and use elastic search to efficiently handle large amount of data.

E. MVC Model and Layered Architecture

1) MVC Model: As seen in Fig. 4, adopting the MVC architecture pattern separates our data model, presentation information and control information. The data model mainly consists of user credentials, user preferences and the aggregated twitter data. The View involves visualizing the twitter data according to a specific user's choice of keyword. The Controller deals with authenticating the user and processing other http requests such as updating the database with the user choice of subjects.

Adopting this pattern enabled us to work in parallel on different components efficiently.

F. Layered Architecture

The layered architectural pattern adopted for our application is as seen in figure 5.

Our architecture consists of three layers:

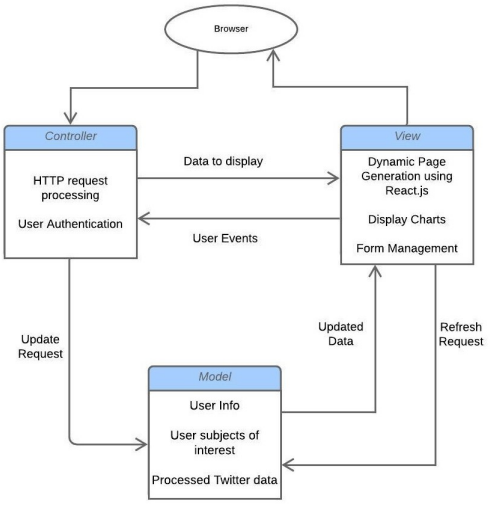

Fig. 4. Model View Controller(MVC) model.

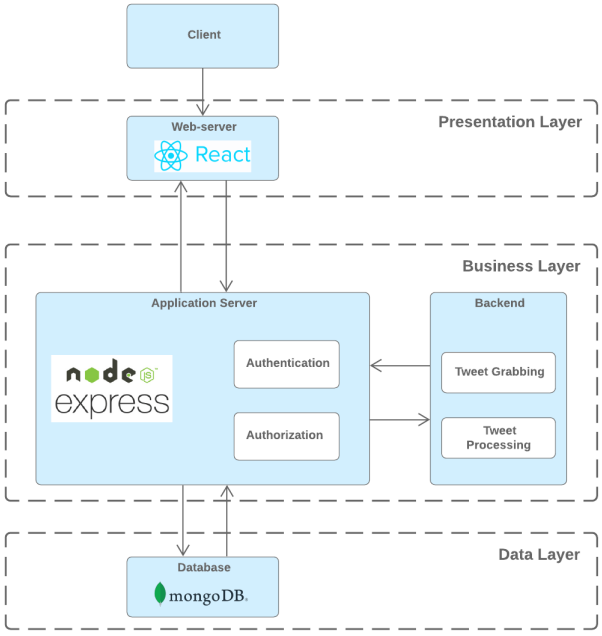

Fig. 5. Layered Architecture

- **Presentation Layer**: Majorly designed using React library displaying the pages to the user.
- **Business Layer**: This layer consists of the application module and data grabbing module.
  - The application module deals with authenticating users into their respective dashboards and authorizes them to view and modify charts.
  - The data grabbing module authenticates with the official twitter API and stream tweets related to the user's subject of interest. It also deals with processing the twitter data to extract sentiment from

TABLE I
SOFTWARE METRICS AND GRANULARITY OF COMPONENTS

Task Name	LOC	Component Granularity Level	Number of Unit
Tweet grabbing	90	Tweepy Python Library	1
Tweet Sentiment Analysis	156	TextBlob Python library	1
Setup - Back-end Server Application	38	JavaScript Functions using ExpressJS Library	13
User Authentication - Front-end View Display	206	JSX Functions Using React Library	2
User Authentication - Front-end Handle User Actions and Events	108	JavaScript Functions	4
User Authentication - Front-end Data Management	61	JavaScript Functions with Redux Library	1
User Authentication - Define Schema of User Data	24	JavaScript Functions with Mongoose Library	1
User Authentication - Back-end APIs	160	JavaScript Functions with ExpressJS in NodeJS	3
Dashboard - Define Schema of Chart Data	48	JavaScript Functions with Mongoose Library	1
Dashboard - Handle HTTP requests for Chart Data	212	JavaScript Functions with ExpressJS Library	6
Dashboard - Viewing Charts Front-end	215	JSX Functions using React Library	4
Dashboard - Manage User Inputs and Handle Events	124	JavaScript Functions	4

it.

- **Data Layer**: It consists of a mongoDB database designed to store the user credentials and processed data.

Adopting the layered architecture enabled us to efficiently perform component testing by validating the functionality of each layer before moving to the higher layer of abstraction.

IV. SOFTWARE METRICS AND GRANULARITY OF COMPONENTS

A statistical count of software metrics from all the tasks of our projects is shown in Table 1. It includes all classes/objects, packages, libraries, frameworks and platforms used in our code.

REFERENCES

[1] https://www.omnicoreagency.com/twitter-statistics/
[2] Ian Gorton, "Essential Software Architecture, Second edition"
[3] Ian Sommerville, "Software Engineering, Tenth Edition"
[4] Steve Y Yang, Sheung Yin Kevin Mo, and Anqi Liu. "Twitter financial community sentiment and its predictive relationship to stockmarket movement". In:Quantitative Finance15.10 (2015), pp. 1637–1656
[5] Python Natural Language Toolkit(NLTK). 2020. URL: http://www.nltk.org/
[6] TextBlob. 2020. URL: https://textblob.readthedocs.io/en/dev/
[7] Twitter API. 2020. URL:https://twitter.com/rest/public
[8] Martin Kleppmann - "Designing Data-Intensive Applications. The Big Ideas Behind Reliable, Scalable and Maintainable Systems", O'Reilly (2017)