2021

Graphical user interface, text, application

Description automatically generated

Twitter Enalyst

CAB432

Assignment 2

Dac Duy Anh Nguyen n10603280

Do Le Hoang Minh n00000000

Contents

[Introduction 2](#_Toc86622804)

[Purpose & description 2](#_Toc86622805)

[Services used 2](#_Toc86622806)

[Search Tweets: Standard API v.1.1 2](#_Toc86622807)

[Sentiment 2](#_Toc86622808)

[Google Trend 2](#_Toc86622809)

[Use cases 3](#_Toc86622810)

[User story 1 3](#_Toc86622811)

[User story 2 3](#_Toc86622812)

[User story 3 4](#_Toc86622813)

[User story 4 5](#_Toc86622814)

[Technical breakdown 7](#_Toc86622815)

[Architecture 7](#_Toc86622816)

[Process flow diagram 8](#_Toc86622817)

[Data object 10](#_Toc86622818)

[Scaling and Performance 10](#_Toc86622819)

[Test plan 10](#_Toc86622820)

[Difficulties 12](#_Toc86622821)

[Extensions 12](#_Toc86622822)

[User guide 12](#_Toc86622823)

[References 13](#_Toc86622824)

[Appendices 14](#_Toc86622825)

# Introduction

## Purpose & description

Twitter Enalyst is an amazing application for anyone interested in discovering and analyzing sentiment factor of certain topics. And Twitter Enalyst allows users to search for a keyword and to receive sentiment score of the most recent posts containing that keyword. The app also suggests trending keywords on Google trends at that moment which users can test their sentiment score on our app. On the server side, this app utilizes Sentiment module to perform sentiment analysis on input Twitter posts that are queried from Twitter API.

Chart

Description automatically generated

Figure 1 Sample result of the app

## Services used

#### Search Tweets: Standard API v.1.1

Returns a collection of relevant Tweets matching a specified query – can also be filtered based on time, location, language, geography, etc.

Endpoint: <https://api.twitter.com/1.1/search/tweets.json>

Docs: <https://developer.twitter.com/en/docs/twitter-api/v1/tweets/search/api-reference/get-search-tweets>

#### Sentiment

A node.js module that uses the AFINN-165 wordlist and Emoji Sentiment Ranking to perform sentiment analysis on arbitrary blocks of input text.

Endpoint: using from library to communicate with Sentiment

Docs: <https://www.npmjs.com/package/sentiment>

#### Google Trend

A node.js module that providing trending keywords from

Endpoint: using from library to communicate with Google Trend API

Docs: <https://www.npmjs.com/package/google-trends-api>

### Use cases

#### User story 1

|  |  |
| --- | --- |
| As a | User |
| I want | To see what keywords are trending right now in Australia on google search engine |
| So that | I may use them as input for sentiment analysis |

For this user story, we implement the Google Trend library to get trending keywords in Australia. Each time a new user access our app, a request is trigger to route /googleTrend. Then, googleTrend.js will use Google Trend library to get trending keywords and send them to client side.

Text

Description automatically generated

Figure 2 getTrends() used for requesting trending keywords in Australia

Text

Description automatically generated with medium confidence

Figure 3 Google trending keywords on client side

#### User story 2

|  |  |
| --- | --- |
| As a | User |
| I want | To search for a keyword |
| So that | The app can return the sentiment score of posts containing that keyword within the last 7 days |

On the client side, users can type in their desired keyword to analyze sentiment score. Once they click ‘Search’, that keyword will be sent to the server via route /twitter and twitter.js will start querying Redis or DynamoDB or sending requests to Twitter API.

Graphical user interface, text, application, website

Description automatically generated

Figure 4 Search bar

A screenshot of a computer

Description automatically generated with medium confidence

Figure 5 Code logic to query Tweets: Redis -> DynamoDB -> Twitter API

|  |  |
| --- | --- |
| With the tweets result, the server will analyze responses’ text using get Sentiment from ‘module/sentiment.js’ .  Text  Description automatically generated  Figure 6 Get sentiment score from sentiment.js and send result to client | Text  Description automatically generated  Figure 7 Inside setiment.js |

#### User story 3

|  |  |
| --- | --- |
| As a | User |
| I want | The app to display a chart of sentiment score summary (total positive /negative scores and posts) |
| So that | The result is more intuitive to users |

As said in the previous user story, the analysis results received by client side is handled TotalSearchTweet.js (utilizing react-chartjs-2) to create chart with visualized result summary, namely Total positive/negative score, Number of positive/ negative /neutral posts.

Text, chat or text message

Description automatically generated

Figure 8 Calculating items for chart and then send html components

Chart

Description automatically generated

Figure 9 Sentiment scores and summary

#### User story 4

|  |  |
| --- | --- |
| As a | User |
| I want | The post analyzed to be displayed on the app |
| So that | I can know what those posts says |

|  |  |
| --- | --- |
| Graphical user interface, text, application, chat or text message  Description automatically generated  Figure 10 Tweets found | The app simply uses tweets ID from server response to make <TweetEmbed> components. Client side will use the parameters it got (tweetID) to get full tweets content from Twitter to display.  A screenshot of a computer  Description automatically generated with medium confidence  Figure 11 TweetEmbed component |

# Technical breakdown

## Architecture

Diagram, engineering drawing

Description automatically generated

Figure 12 Architecture diagram

The diagram above describes the high-level overview of architecture and operation of the application.

The whole Twitter Enalyst application is run directly on the EC2 T2 Micro instances. Server is built on Express framework of Nodejs and responsible for handling client request, responding client, interacting with API endpoint as well as DynamoDB and Redis. Each server is run on an EC2 T2 instance of AWS. On the server, Google Trend library is utilized to get trending keywords and Sentiment library is utilized to analyze tweets and give out their sentiment score.

Its client side is built with React to provide users with a modern and interactive interface. Standing between client and server, Elastic load balancer helps to balance load between EC2 T2 instances in the scaling group and deploy new instances to handle an increasing load if it satisfies the scaling policy.

A screenshot of a computer

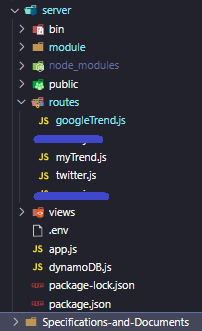
Description automatically generated with medium confidenceThere are 3 main routes in server for client-side to communicate to:

Figure 13 App structure

* /**googleTrend**: utilizes google trend library to get trending keywords on Google
* /**myTrend**: queries database and get the trending keywords are being searched on this app
* /**twitter**: queries tweets containing the requested keyword from Twitter API

In Front-End, we use the React Hooks – useRef and useState to manage the state and useContext hook to pass the data through the component tree. The new React Hook help to update state separately that make the UI flow more reusability, readability, and testability.

## Process flow diagram

Note from the process flow diagram that if the data is found in DynamoDB for the keyword, then that data will be stored in Redis as well. And if the data is queried from Twitter API, then that data will be stored in DynamoDB and Redis for later repeated request.

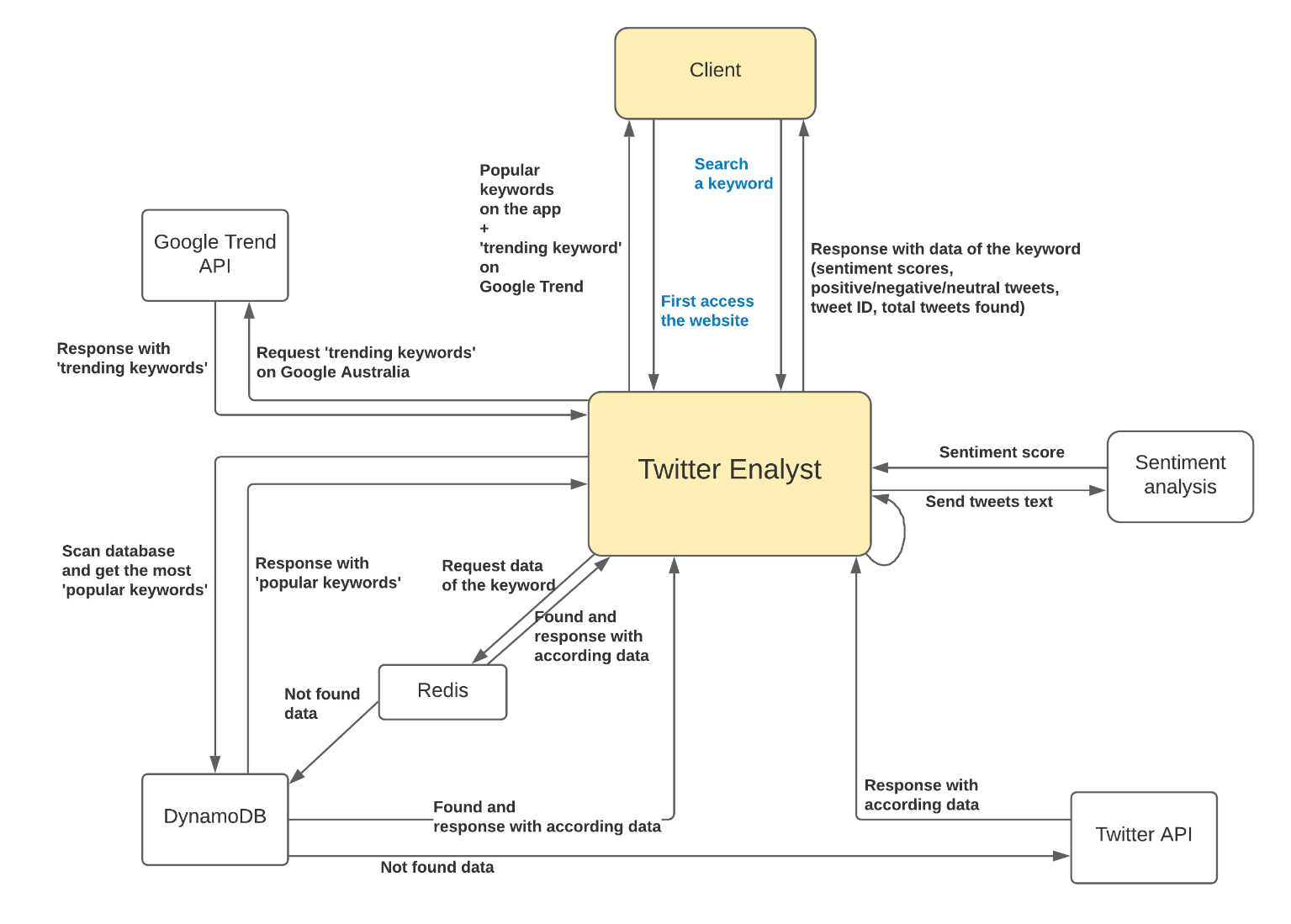


Figure 14 Process flow diagram

Diagram

Description automatically generated

Figure 15 The flow of the state in the client-side.

## Data object

## Scaling and Performance



Figure 16 In service instances snapshot

# Test plan

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Expected outcome | Result | Screenshot/s |
| Data stored in Redis | New Tweet result is stored in Redis | Pass |  |
| Data stored in DynamoDB | New Tweet result is stored in DynamoDB | Pass |  |
| Retrieve data from DynamoDB | Result of repeated keyword within 24 hours is retrieved from DynamoDB | Pass |  |
| Retrieve data from Redis | Result of repeated keyword within 24 hours is retrieved from Redis | Pass |  |
| Get data from Twitter API | New Tweet result is queried directly from Twitter API | Pass |  |
| Trending keyword is got from Google Trending API | New trending keywords are displayed on client side | Pass |  |
| Check old data from DynamoDB/Redis and get new data from Twitter API | Result in DynamoDB/Redis more than 24 hours old is disqualified and new Tweet post are queried from Twitter API | Pass |  |
| Trending keyword on website is got from DynamoDB | Client receives trending keyword on this website response | Pass |  |
| Background image is loaded | Page displaying background image | Pass |  |
| Search button is functional | Search button directs user to tweets summary chart | Pass |  |
| Notification appears in result page | Notification displays basic result information | Pass |  |
| Client receives the results | Chart display tweet summary with sentiment score and tweet statistic | Pass |  |
| Tweet posts analyzed are displayed at the page bottom | Tweet’s content automatically loads at the page bottom | Pass |  |
| Search for unpopular/invalid keywords | Notification prompts invalid keyword/ no post found | Pass |  |
| Reload button is functional | Reload button brings user back to search page | Pass |  |
| Scale server when load increases | New instance is deployed | Pass |  |

# Difficulties

At the beginning, we used the twitter API endpoint for streaming data real time through Web Socket with the component socket.io and search endpoint for 100 recent twitter posts at the same time. However, we encountered a problem: Twitter API streaming only accepts one connection endpoint at a time which prevents multiple users to use our application when we try to scale the app (more users). The second problem is that using a web socket will make the application become a stateful application, which does not meet the requirement of the assignment.

After technical analysis, we found 2 solutions for the first problem that serve the socket io as an independence with the sentiment score analytics server or move to the twitter API search endpoint and HTTP request. Because of the time limit, we took the advice of a tutor - Michael Esteban to decide to change to using the API search endpoint and HTTP request. Sadly, we also had to remove some features of the application such as real time sentiment score chart and the summary of real-time score summary.

Variable type: This is a pretty basic mistake as we did not declare type of a variable (outside an if statement) and that make the operation inside an if could not access it. Although it was a small error but takes us a while to find it. After that, we managed to fix it by adding ‘var’ type.

# Extensions

There are some further improvements for this app:

**App functionalities**:

In order to increase the precision of the sentiment scores, we could increase the number of posts and the time range queried from API and. At the moment, due to the limitation of free tier access to the Twitter API, we can only query 100 posts that lies with 7 days period. With a small number of posts in a short time range being analyzed, the sentimental scores do not reflect exactly all the twitter posts in the past 7 days from the moment of query.

**Persistence**:

The trending keyword from both Google trend and Twitter Enalyst should be store in DynamoDB and Redis and be updated every hour in order to save network load as well as reduce interaction with database.

There is a section to display Tweets that are used in sentiment analysis. Their contents could also be stored in DynamoDB and Redis to quickly send and display Tweets on client-side.

# User guide

# References

# Appendices