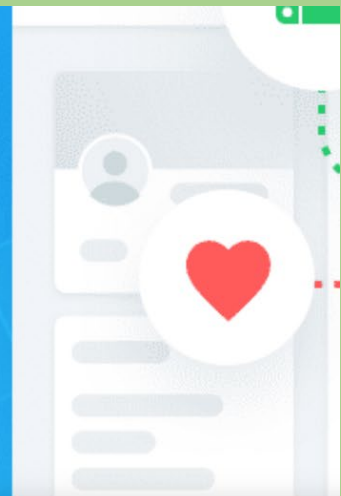


2021

# Twitter Analyst

## Sentiment Analysis of Twitter



CAB432

Assignment 2

Dac Duy Anh Nguyen n10603280

Do Le Hoang Minh n10570926

## Contents

Introduction.....	2
Purpose & description .....	2
Services used .....	2
Search Tweets: Standard API v.1.1 .....	2
Sentiment .....	2
Google Trend .....	2
Use cases .....	3
User story 1.....	3
User story 2.....	3
User story 3.....	4
User story 4.....	5
Technical breakdown.....	7
Architecture .....	7
Process flow diagram.....	8
Data object.....	10
Scaling and Performance .....	11
Test plan.....	13
Difficulties.....	14
Extensions.....	15
User guide .....	16
References.....	19
Appendices.....	19

# Introduction

## Purpose & description

Twitter Analyst is an amazing application for anyone interested in discovering and analyzing sentiment factor of certain topics. And Twitter Analyst 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.

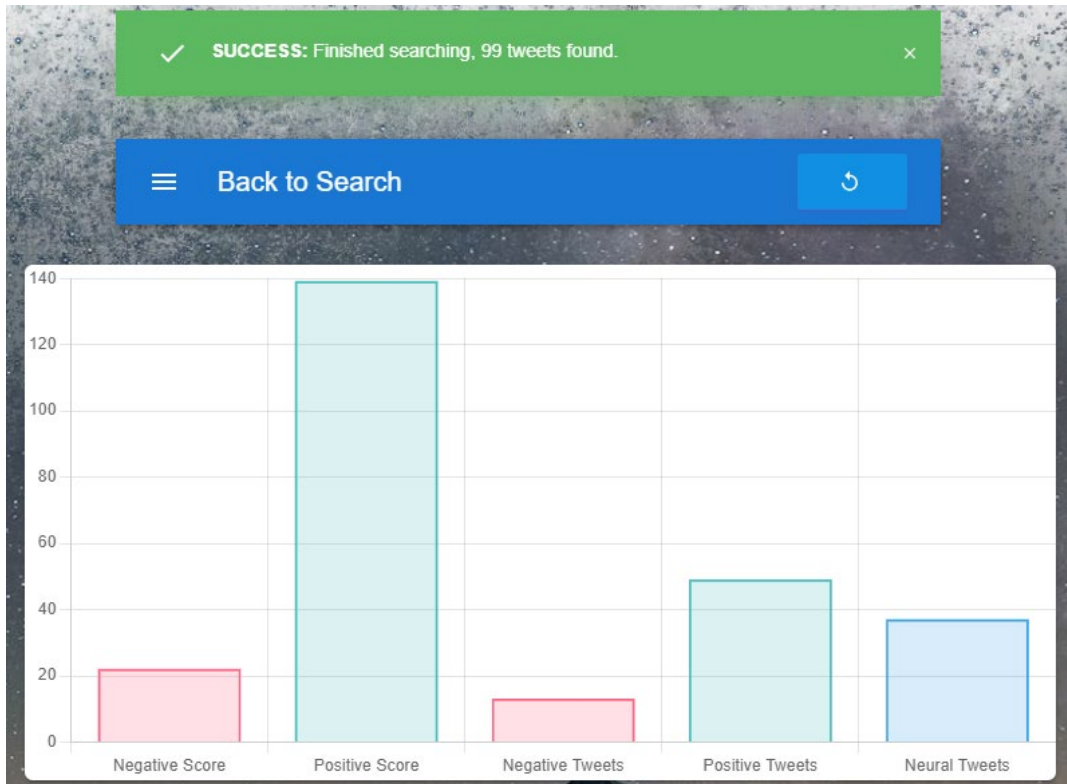


Figure 1 Sample result of the app

## 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.

```
7
8  const getTrends = async () => {
9    return await googleTrends.realTimeTrends({
10     geo: "AU",
11     category: "all",
12   });
13 }
```

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



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.

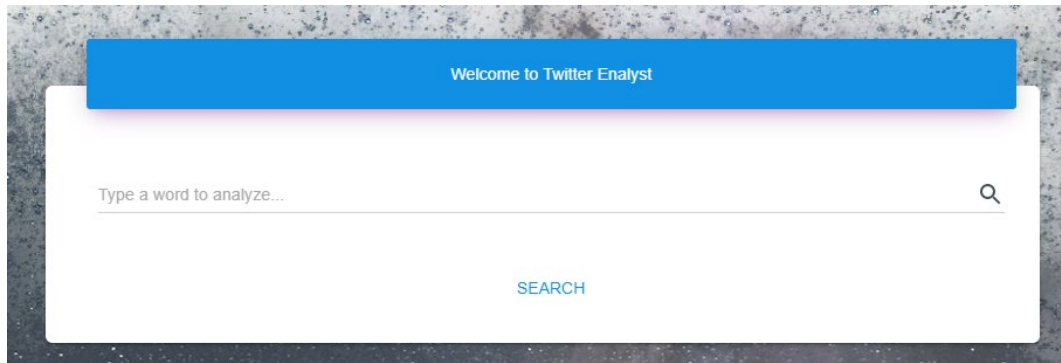


Figure 4 Search bar

```

122
123 console.log("Persistence -----> Check data in Redis");
124 redisClient.get(`TwitterEnalyst:${keyword}`, (err, result) => {
125 >   if (result && checkData(JSON.parse(result), count)) { ...
132   }
133   else {
134     console.log("Persistence -----> Not found in Redis");
135     console.log("Persistence -----> Check data in DynamoDB");
136     readDynamo(keyword).then((data) => {
137 >       if (checkData(data.Item, count) !== 0) { ...
148       } else {
149         console.log("Persistence -----> Not found in Dynamo");
150         console.log("Persistence -----> Using Twitter API");
151         clientTwitter.get(
152           "search/tweets",
153           { q: keyword, lang: "en", count: "100" },
154           function (error, tweets) {

```

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'.

```

tweets.statuses.forEach(function (tweet) {
  result.push(getSentiment(tweet));
})
res
  .status(200)
  .json({ error: false, data: result });

```

Figure 6 Get sentiment score from sentiment.js and send result to client

```

5 //Send Tweet Text to Sentiment Analysis
6 sentiment.getSentiment = (tweet) => {
7   const sentimentScore = sentiment.analyze(tweet.text);
8   numScore = sentimentScore.score;
9   if (numScore < 0) {
10     score = "negative";
11   } else if (numScore > 0) {
12     score = "positive";
13   } else {
14     score = "neutral";
15   }
16   return sentiment.appendSentiment(tweet, score, numScore);
17 };
18
19 //Send sentiment score of tweet to Client
20 sentiment.appendSentiment = (tweet, sentiment, numScore) => {
21   var scoreTweet = {
22     sentiment: sentiment,
23     num_score: numScore,
24     created_at: tweet.created_at,
25     timestamp_ms: tweet.timestamp_ms,
26     id_str: tweet.id_str,
27     text: tweet.text,
28   };
29   return scoreTweet;
30 };

```

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.

```

62 | // Sum up total positive / negative scores and posts
63 | const TotalSearchChart = () => {
64 |   const { scoreSearchTweet, setAchirveScore, summary100PostScore } = useContext(TweetContext);
65 |   const [negativeScore, setNegativeScore] = useState(0);
66 |   const [positivePost, setPositivePost] = useState(0);
67 |   const [negativePost, setNegativePost] = useState(0);
68 |   const [positiveScore, setPositiveScore] = useState(0);
69 |   const [countNeural, setCountNeural] = useState(0);
70 |
71 |   useEffect(() => {...
86 |   }, [scoreSearchTweet, summary100PostScore]);
87 |
88 |   useEffect(() => {...
95 |   }, [scoreSearchTweet, negativeScore, positiveScore, setAchirveScore]);
96 |
97 |   const data = [negativeScore, positiveScore, negativePost, positivePost, countNeural];
98 |   return (
99 |     <div>...
120 |     </div>
121 |   );
122 | };

```

Figure 8 Calculating items for chart and then send html components

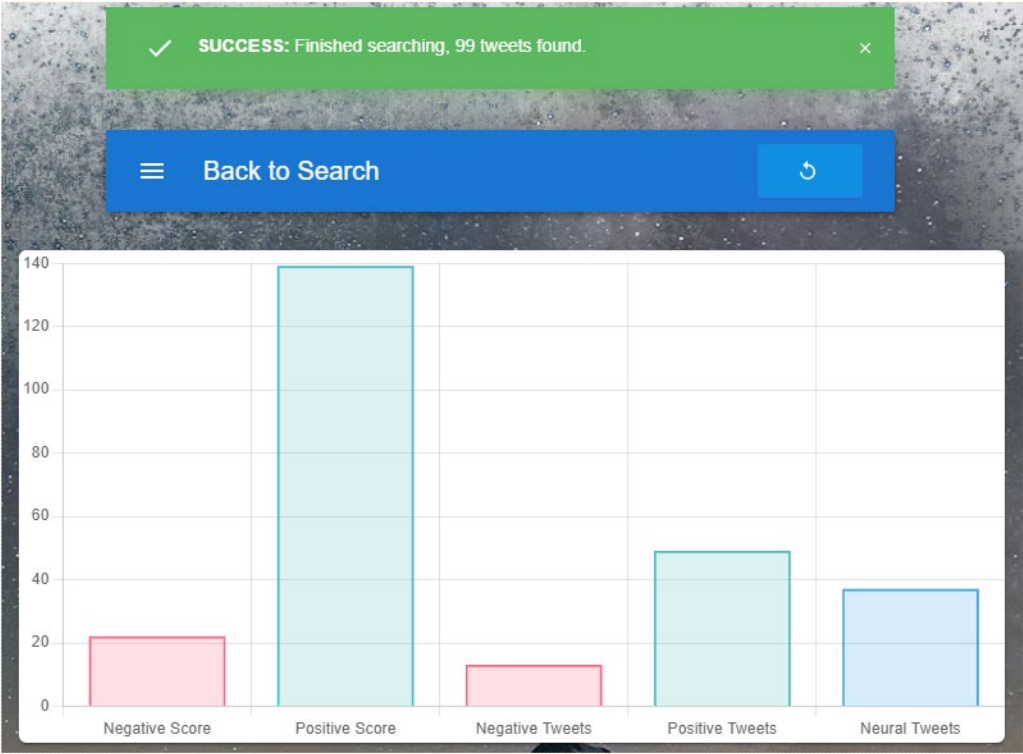


Figure 9 Sentiment scores and summary

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



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.

```

{idTweet.length ? (
  idTweet.map((id, key) => {
    if (key < 30) {
      return (
        <TweetEmbed
          id={id}
          placeholder={"loading"}
          className={classes.tweet}
        />
      );
    }
    return null;
  })
)

```

Figure 11 TweetEmbed component



# Technical breakdown

## Architecture

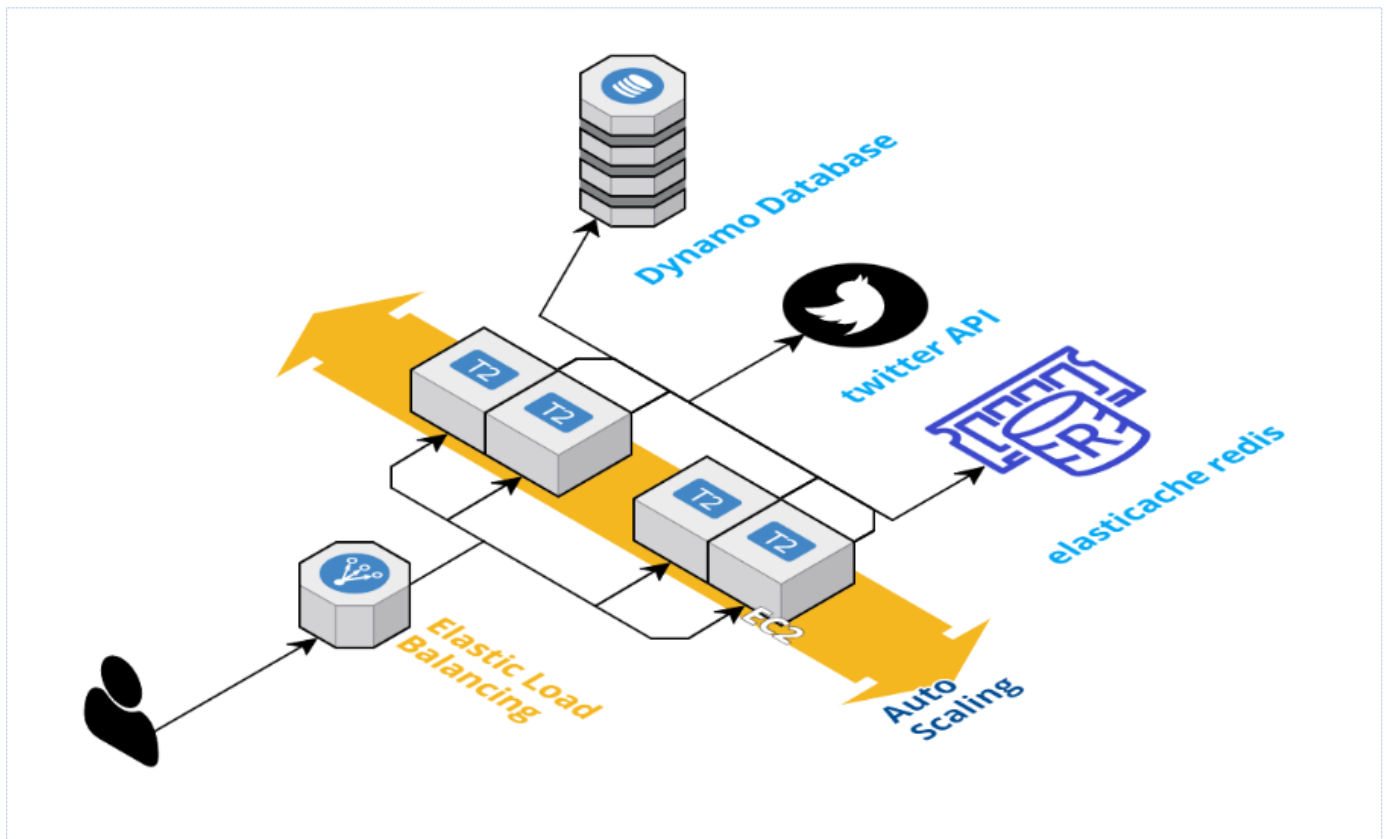


Figure 12 Architecture diagram

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

The whole Twitter Analyst 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.



There are 3 main routes in server for client-side to communicate to:

- /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

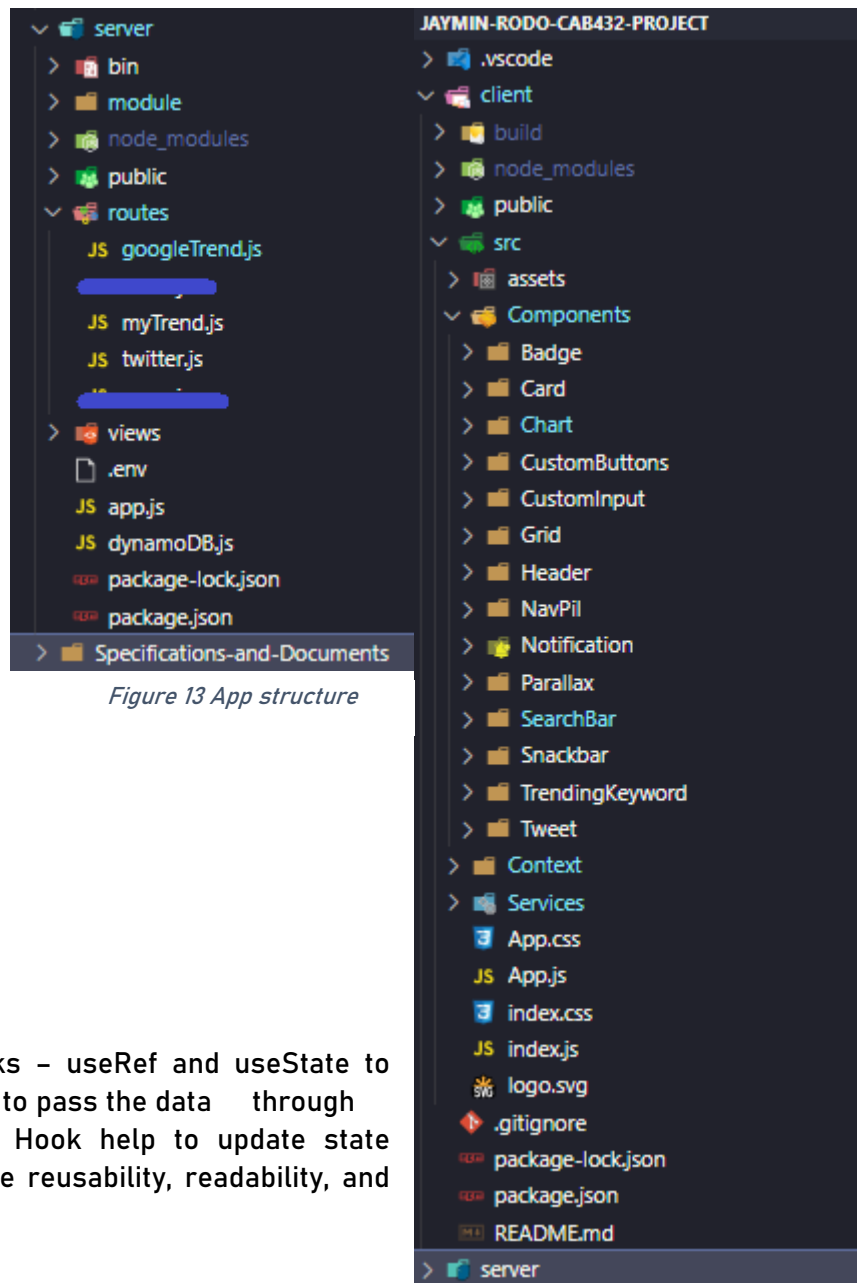


Figure 13 App structure

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.

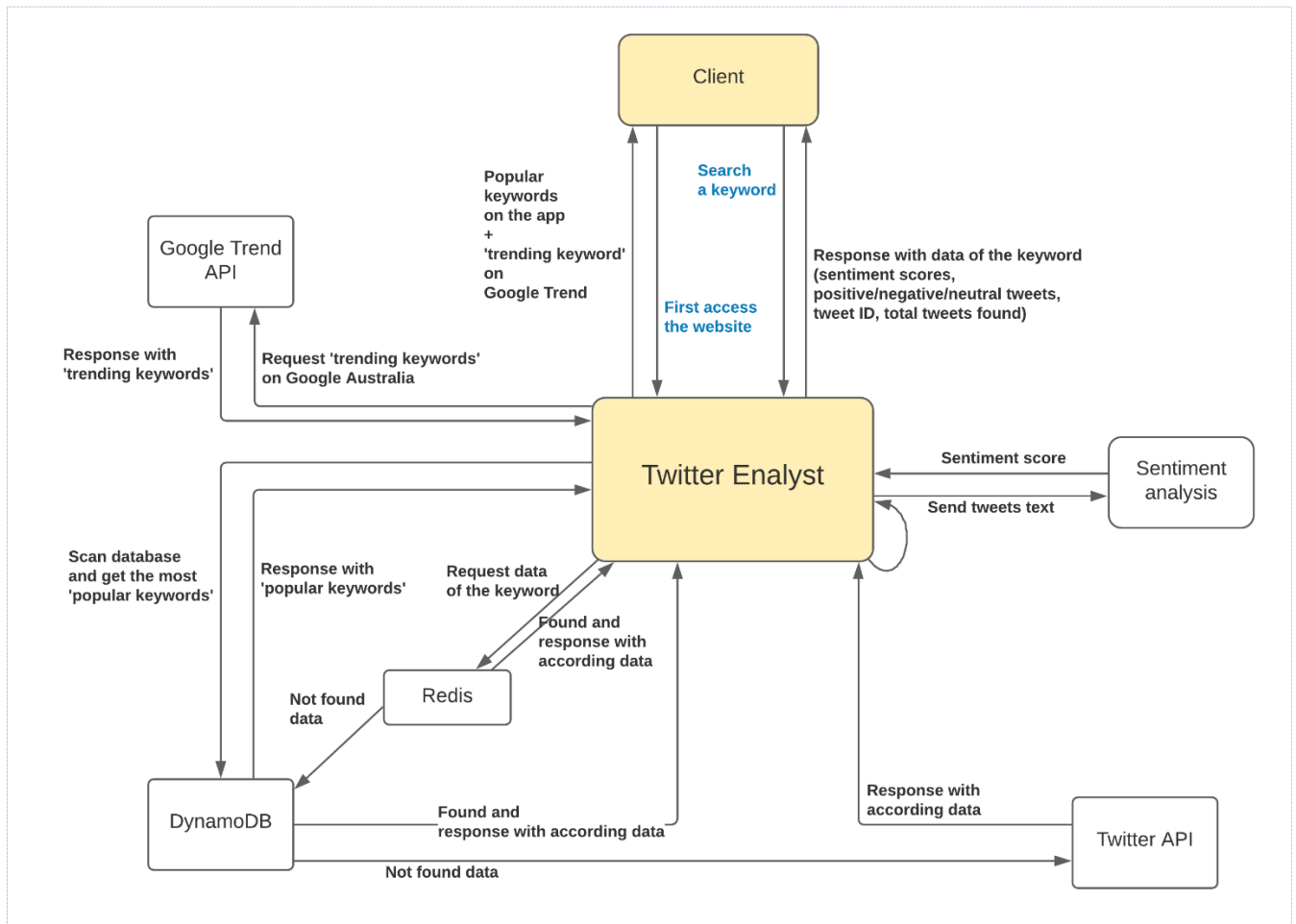


Figure 14 Process flow diagram

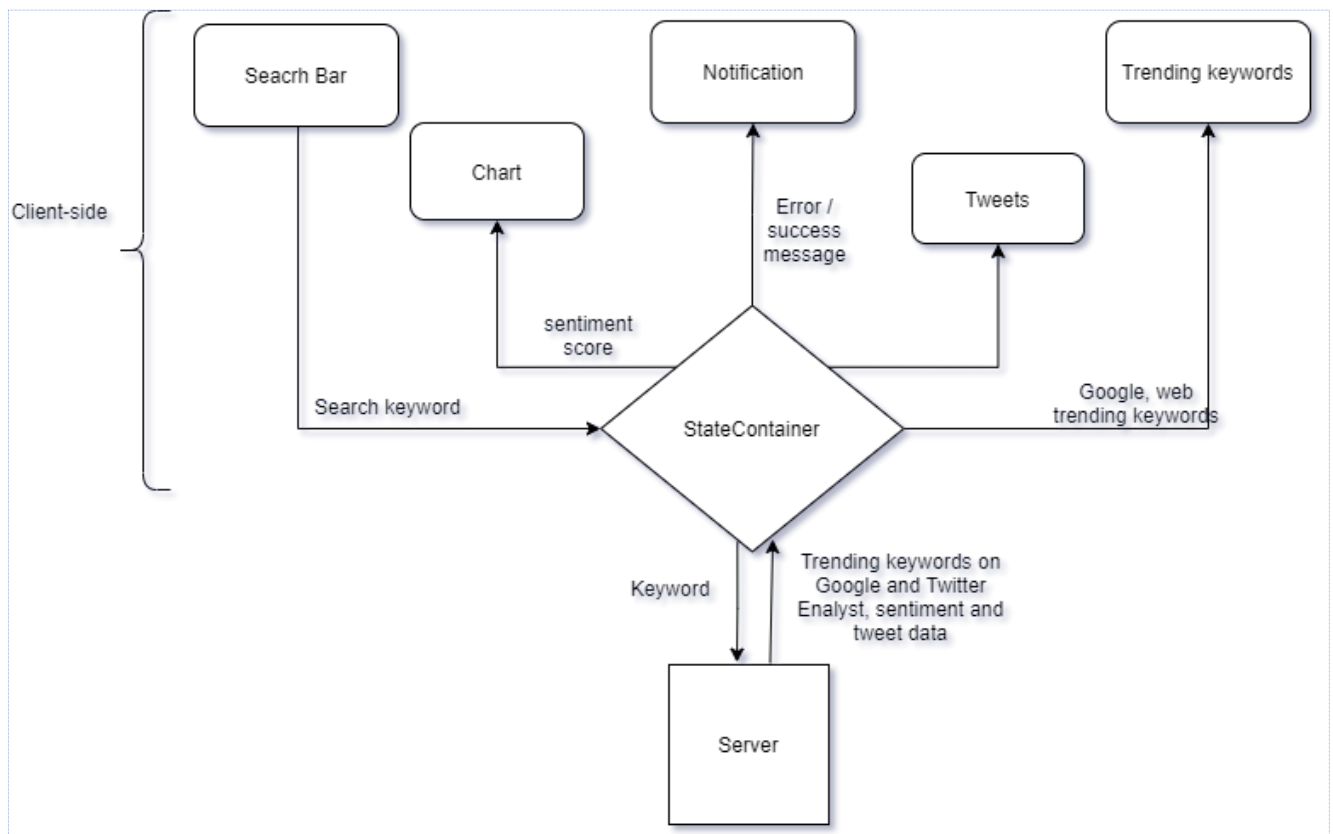





Figure 15 The flow of the state in the client-side.


GET  http://localhost:3000/twitter/?keyword=apple


Params  Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	keyword	apple	
	Key	Value	Description


Body Cookies Headers (8) Test Results  200 OK 1436 ms 24.81 KB

GET  http://localhost:3000/twitter/?keyword=apple

Params  Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	keyword	apple	
	Key	Value	Description

Body Cookies Headers (8) Test Results  200 OK 20 ms 24.81 KB

As we can see from the latencies of 2 response (Upper: directly from Twitter API, bottom: from Redis), Redis helps to reduce the response time significantly from 1400 ms to 20 ms.

## Data object

These are the 2 JSON data packages the server sends to client, 'Trending keywords on Google' and 'Trending keywords on this website'.

```

1  {
2    "error": false,
3    "data": [
4      {
5        "keyword": "NFL"
6      },
7      {
8        "keyword": "NFL"
9      },
10     {
11       "keyword": "Tinder"
12     },
13     ...,
14     {
15       "keyword": "Rare"
16     },
17     {
18       "keyword": "Extreme-weather"
19     },
20     {
21       "keyword": "Cancer-vaccine"
22     },
23     {
24       "keyword": "NASCAR-Cup-Series"
25     }
26   ]
27 }
```

Figure 16 Sample data returned from /googleTrend

```

1  {
2    "error": false,
3    "data": [
4      "Anfield",
5      "ANFIELDENGLAND",
6      "rodo",
7      "dtt",
8      "2000",
9      ...,
10     "Australia",
11     "bad",
12     "oawihfoishosdovnosnpvioabh",
13     "Kurnell",
14     "Beetlejuice"
15   ]
16 }
```

Figure 17 Sample data returned from /myTrend

```

1  {
2    "error": false,
3    "data": [
4      {
5        "sentiment": "positive",
6        "num_score": 2,
7        "created_at": "Mon Nov 01 03:11:23 +0000 2021",
8        "id_str": "1455009545529536515",
9        "text": "RT @tante02: Earning a #passiveincome while you sl
10     },
11     {
12       "sentiment": "neutral",
13       "num_score": 0,
14       "created_at": "Mon Nov 01 03:11:04 +0000 2021",
15       "id_str": "1455009467221909509",
16       "text": "RT @cryptobakerone: Hey you. Yeah you.\nStop looki
17     },
18     ...
19     {
20       "sentiment": "neutral",
21       "num_score": 0,
22       "created_at": "Mon Nov 01 03:10:44 +0000 2021",
23       "id_str": "1455009382534631427",
24       "text": "RT @Loday43760804: #DeFiChain will bring #cashflow
25     },
26     {
27       "sentiment": "neutral",
28       "num_score": 0,
29       "created_at": "Mon Nov 01 03:10:27 +0000 2021",
30       "id_str": "1455009314159071235",
31       "text": "RT @stefjeff91: #DeFiChain will bring #cashflow to
32     }
33   ]
34 }

```

Figure 18 Sample data returned from /twitter

As users search for a keyword, after querying from database/twitter API and analyzing sentiment score, route /twitter from server will return a JSON package to client side including sentiment, score, time created, tweet id, and its text content. From this data, client side will sum up the number of scores and posts according to each category.

The tweet ID will later be used to fetch <TweetEmbed> content at the page bottom.

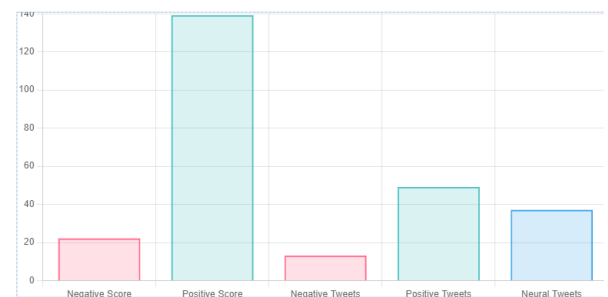


Figure 19 Summary chart from /twitter data

## Scaling and Performance

For the scaling, this application uses AWS Load balancer as the single point of contact for clients and Load Balancer is responsible for directing clients' requests to AWS Target Group. And AWS Auto scaling group can automatically increase or decrease number of VM instances in response to the Dynamic scaling policies defined (This app's scaling policy has been set according to Figure 20 on the right).

Overall, the app has scaled out and in seamlessly without latency increase in client-side as observed in the monitoring metrics below.

Target Tracking Policy

Policy type:

Target tracking scaling

Enabled or disabled?

Enabled

Execute policy when:

As required to maintain Average CPU utilization at 20

Take the action:

Add or remove capacity units as required

Instances need:

15 seconds to warm up before including in metric

Scale in:

Enabled

Figure 20 Scaling Policy



Figure 21 In service instances snapshot

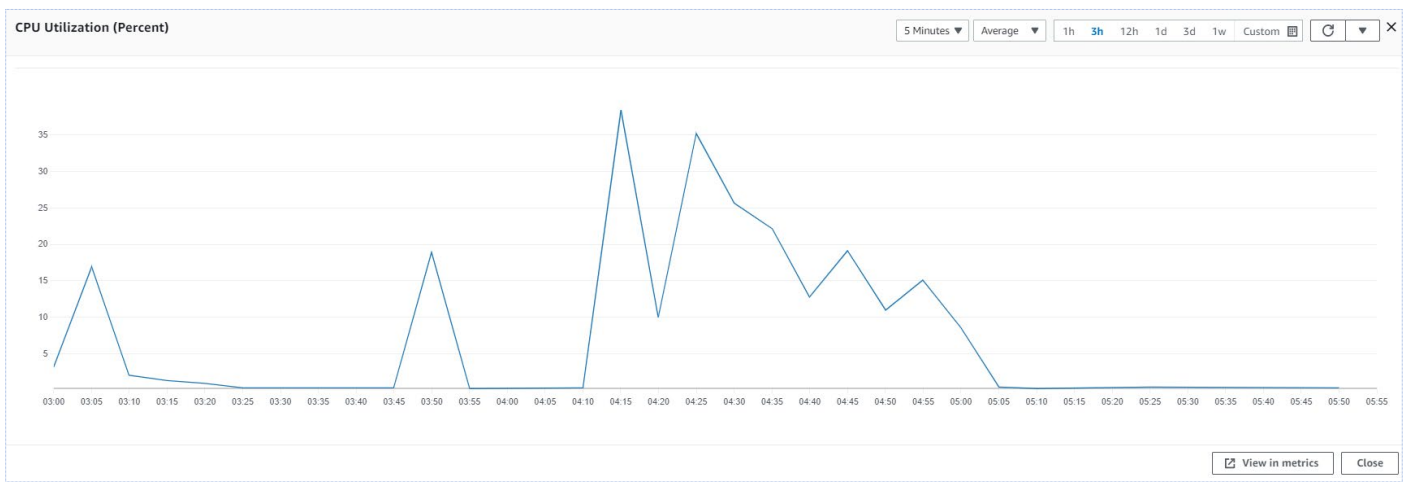


Figure 22 CPU utilization %



Figure 23 Other monitoring metrics of EC2

With large amount of network traffic generated from Postman, AWS Auto Scaling proved to successfully scale out to 5 instances. During that time, the latencies of receiving responses were not affected (as can be seen in the screenshot below).

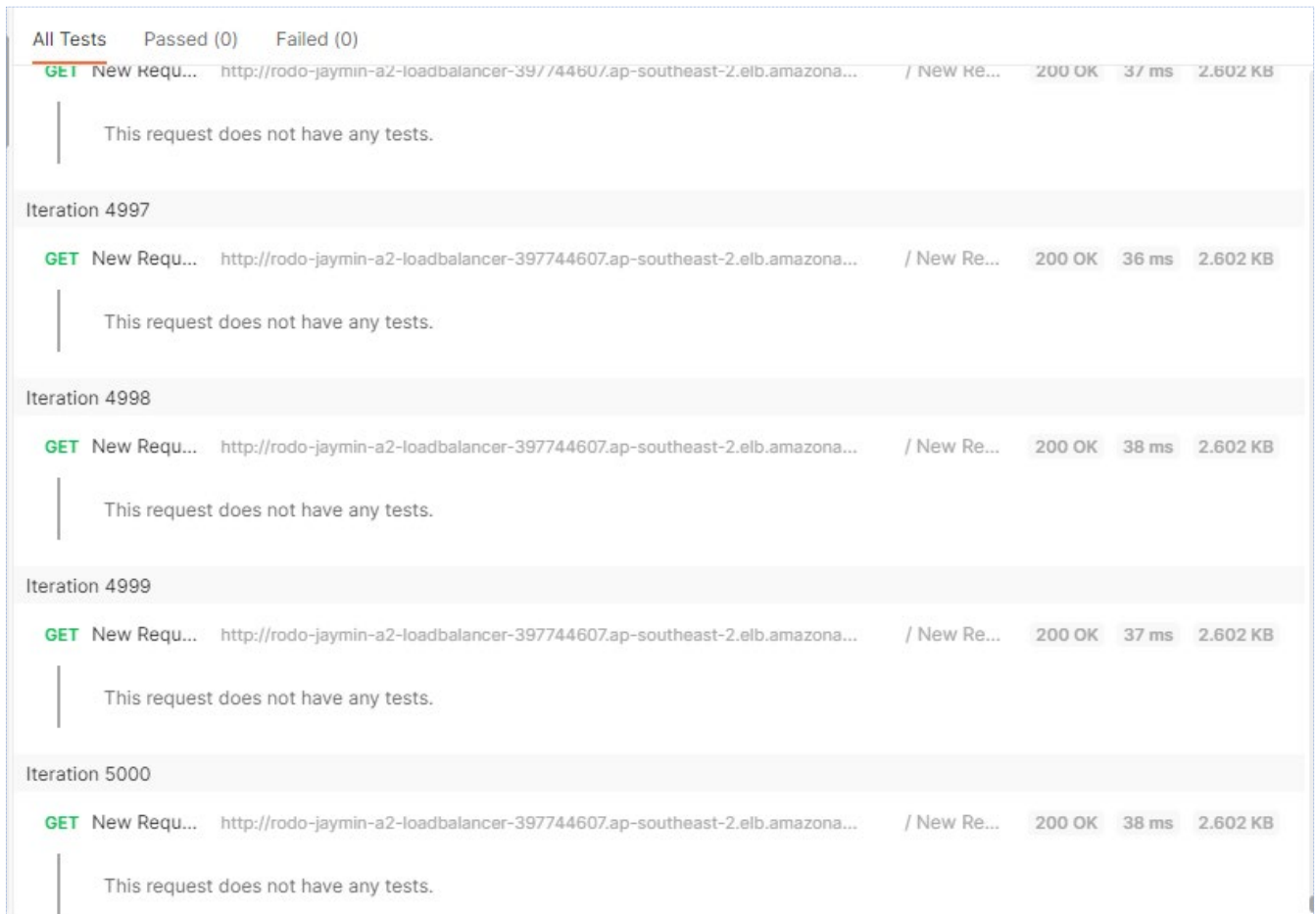


Figure 24 Response latencies during scale out period

## Test plan

Task	Expected outcome	Result	Screenshot/s
Data stored in Redis after query from Twitter API	New Tweet result is stored in Redis	Pass	Appendices/1
Data stored in DynamoDB after query from Twitter API	New Tweet result is stored in DynamoDB	Pass	Appendices/1
Retrieve data from DynamoDB	Result of repeated keyword within 24 hours is retrieved from DynamoDB	Pass	Appendices/2
Retrieve data from Redis	Result of repeated keyword within 24 hours is retrieved from Redis	Pass	Appendices/3
Get data from Twitter API	New Tweet result is queried directly from Twitter API	Pass	Appendices/1
Trending keyword is got from Google Trending API	New trending keywords are displayed on client side	Pass	Appendices/2,4
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	Appendices/1
Trending keyword on website is got from DynamoDB via route /myTrend	Client receives trending keyword on this website response from route /myTrend	Pass	Appendices/5
Background image is loaded	Page displaying background image	Pass	Appendices/

			6,7,8
Search button is functional	Search button directs user to tweets summary chart	Pass	
Notification appears in result page	Notification displays basic result information	Pass	Appendices/ 7,9
Client receives the results	Chart display tweet summary with sentiment score and tweet statistic	Pass	Appendices/ 7,9
Tweet posts analyzed are displayed at the page bottom	Tweet's content automatically loads at the page bottom	Pass	Appendices/10
Search for unpopular/invalid keywords	Notification prompts invalid keyword/ no post found	Pass	Appendices/9
Reload button is functional	Reload button brings user back to search page	Pass	
Scale server when load increases	New instance is deployed	Pass	Figure 20 In service instances snapshot

## 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 that can solve for both problems. The first solution is serving the streaming API with socket io as an independence server and the other server that scale with the app is sentiment score analytics. Another solution is moving 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.



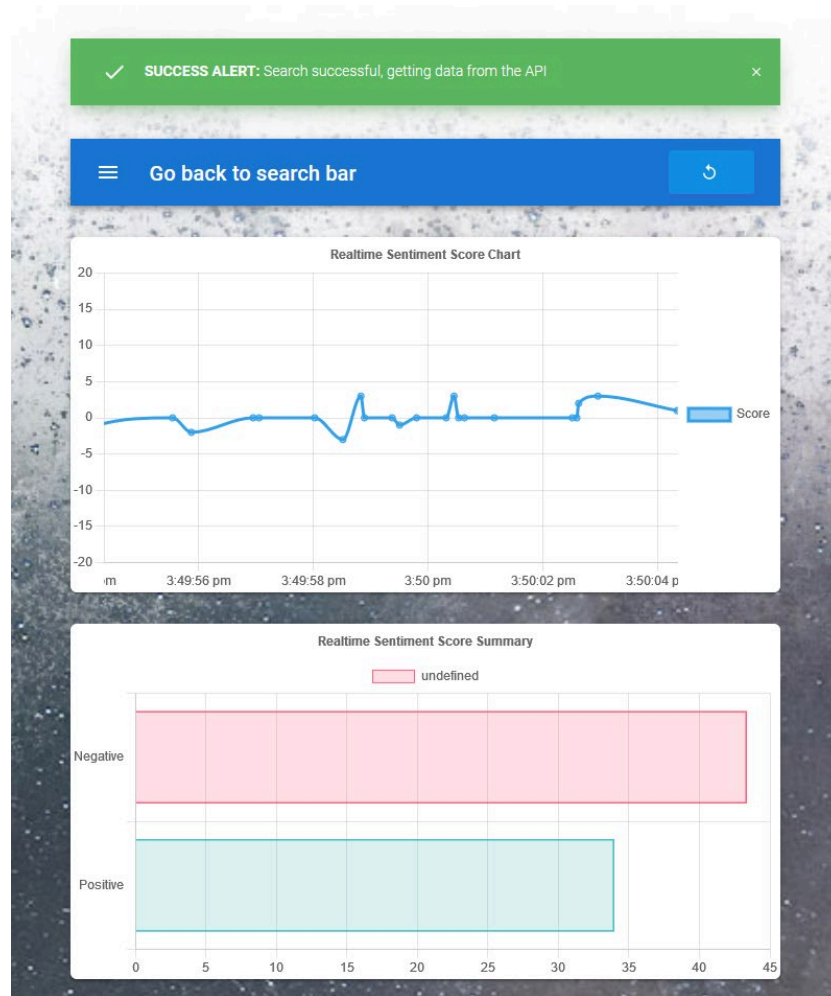


Figure 25 Streaming analysis feature (removed)

**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 we hope to implement in 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.

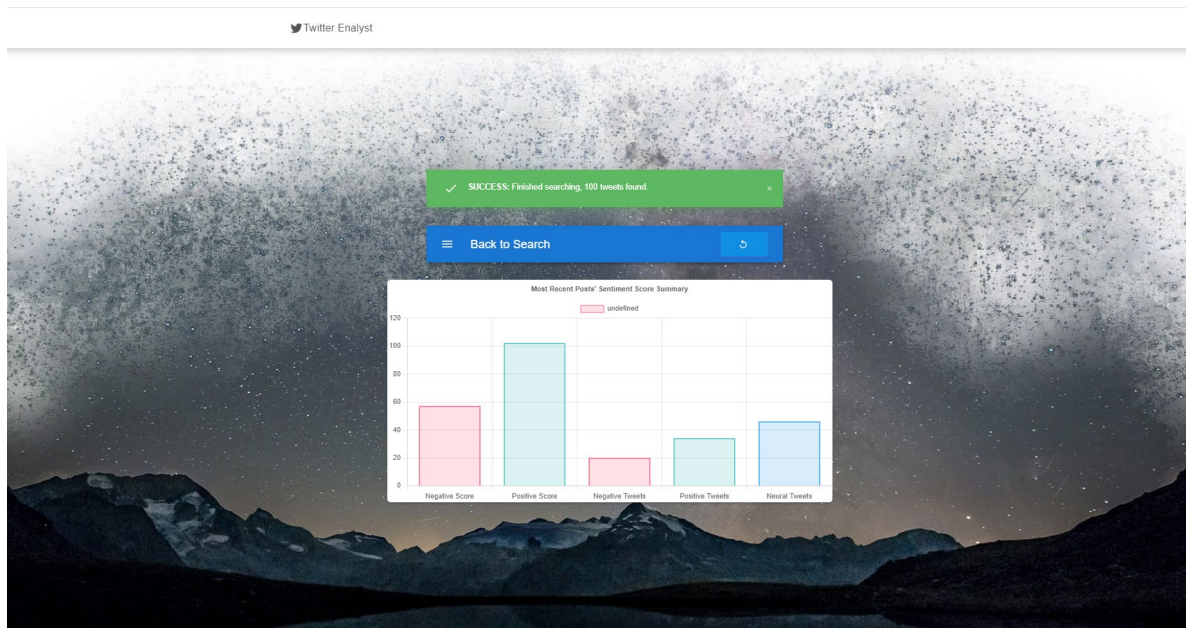
## User guide

1. Firstly , users can access the Twitter Analyst through the DNS of the load balancer at port 3000 and this is the first view they will see:



2. From here, users can view the trending keywords that are searched by google users. Or, click on "Trending keywords on the website" to view popular search on Twitter Analyst. They can advance to search for a keyword of their choice and Click "Search".

3. After approximately 1-3 seconds, users should see the sentiment result of Tweets containing that keyword shown in a column chart.



4. Scroll down, they can see the Tweet contents which have been used to analyze sentiment.

## Twitter Posts Found

**Sandy** 🇺🇸 🇯🇵 🇺🇸  
@Sandy\_L\_K

Let's Go Brandon! 🇺🇸

**Fox News** 🇺🇸 @FoxNews

Biden apologizes for being late to G20 press conference: 'We were playing with elevators' foxnews.com/politics/biden...

12:05 PM · Nov 1, 2021

👍 💬 🔗 Copy link to Tweet

[Tweet your reply](#)

**@uppermoony**

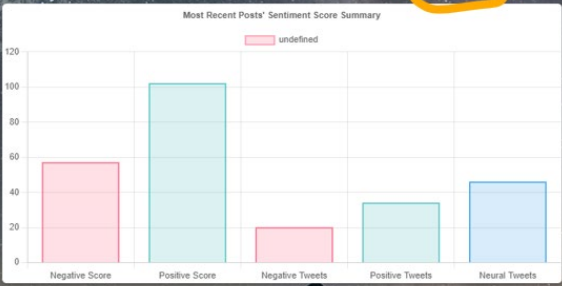
"Light Yagami, L, Kira. You have lost." #deathnote



5. To start searching and analyzing a new keyword, users can click Reload button above the chart.

✓ SUCCESS: Finished searching, 100 tweets found. ✕

☰ Back to Search 🔁





## Appendices

1.

```
GET /myTrend 304 633.327 ms - -
GET /googleTrend 304 903.360 ms - -
GET /favicon.ico 304 3.209 ms - -
GET /manifest.json 304 2.660 ms - -
request heythere undefined
Persistence -----> Check data in Redis
Persistence -----> Not found in Redis
Persistence -----> Check data in DynamoDB
data undefined or empty
Persistence -----> Not found in Dynamo
Persistence -----> Using Twitter API
'result' is collected
Persistence -----> Writing result to Dynamo
Persistence -----> Writing result to Redis
Persistence -----> writeRedis: "heythere"
GET /twitter?keyword=heythere 200 1489.577 ms - 19618
Persistence -----> writeDynamo:
[]
```

2.

```
GET /googleTrend 200 2477.256 ms - 477
request lol undefined
Persistence -----> Check data in Redis
Persistence -----> Not found in Redis
Persistence -----> Check data in DynamoDB
string
2021-10-31T08:18:20
checkData::timestamp 2021-10-30T22:18:20.000Z
Persistence -----> Found in DynamoDB
Persistence -----> Add this data to Redis
Persistence -----> writeRedis: lol
GET /twitter?keyword=lol 200 175.985 ms - 21773
[]
```

3.

```
request lol undefined
Persistence -----> Check data in Redis
string
2021-10-31T16:12:50
checkData::timestamp 2021-10-31T06:12:50.000Z
Persistence -----> Found in Redis
GET /twitter?keyword=lol 304 4.765 ms - -
[]
```

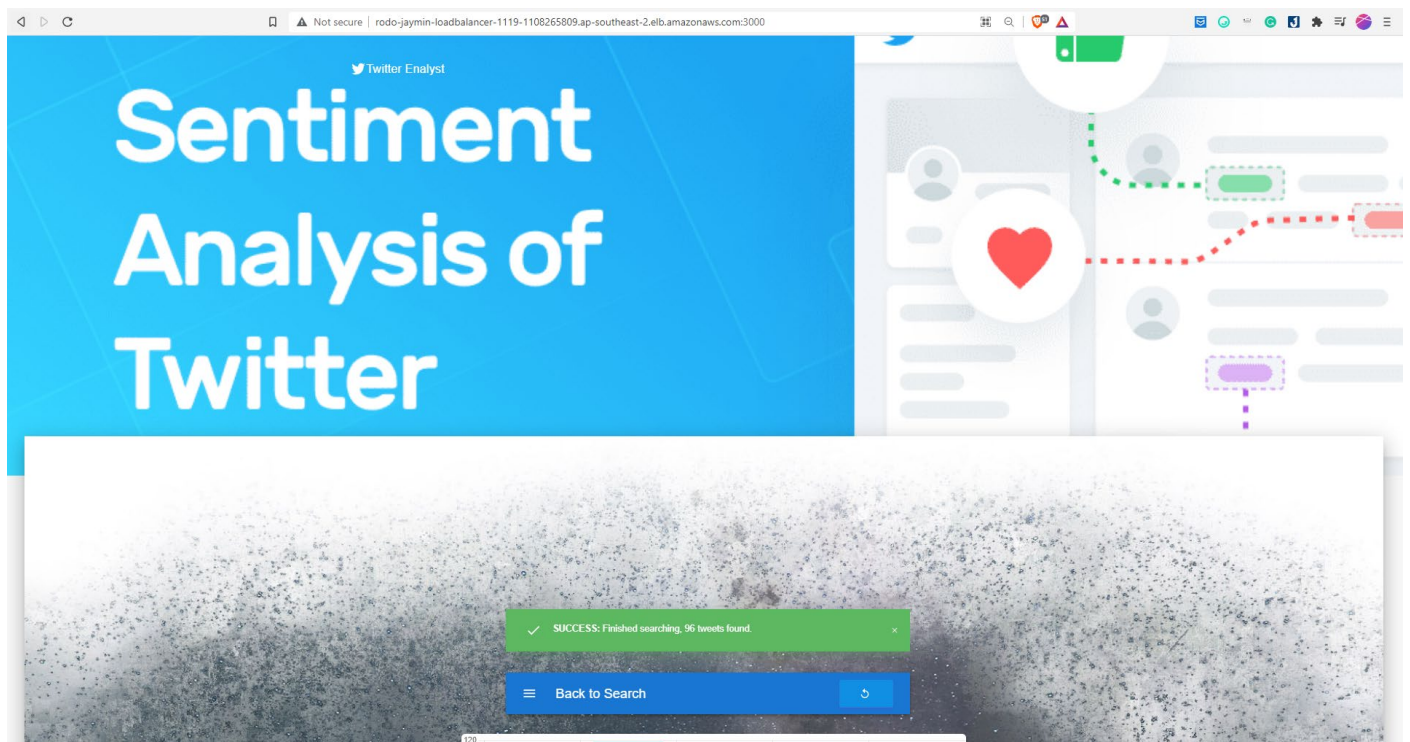
4.

```
1  {
2    "error": false,
3    "data": [
4      {
5        "keyword": "NFL"
6      },
7      {
8        "keyword": "NFL"
9      },
10     {
11       "keyword": "Tinder"
12     },
13     ...
14     {
15       "keyword": "Rare"
16     },
17     {
18       "keyword": "Extreme-weather"
19     },
20     {
21       "keyword": "Cancer-vaccine"
22     },
23     {
24       "keyword": "NASCAR-Cup-Series"
25     }
26   ]
27 }
```

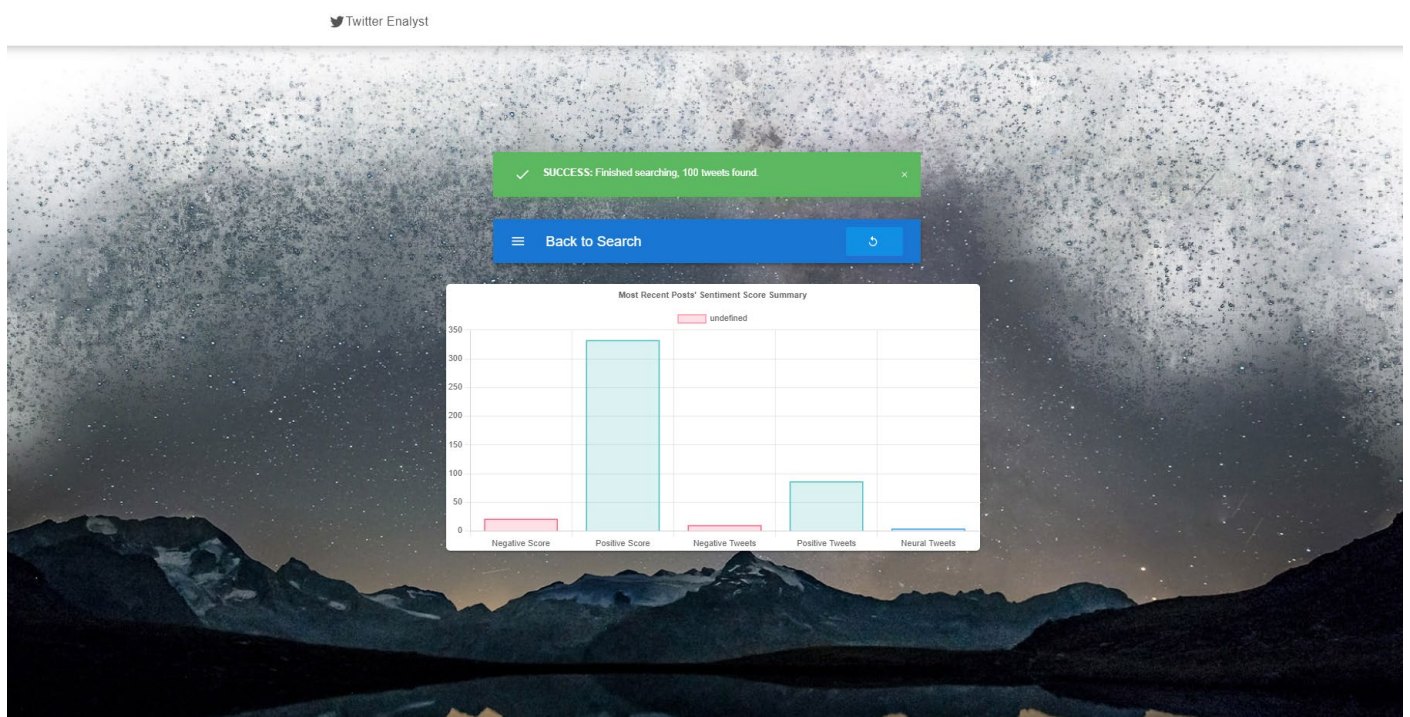
5.

```
/myTrend - Scan succeeded.
[
  'Anfield',      'ANFIELDENGLAND', 'rodo',
  'dtt',          '2000',         'cat',
  'werwe',        'lol',           'abcdef',
  'doge',         'crypto',        'ore-i9New-York',
  'qwer21',       'btc',           'ANFIELD',
  'NFL',          'Web',           'England',
  'ca',           'hello',         'casd',
  'raw',          'Flemington',    'taylor-swift',
  '2020',         'Doggo',         'dog',
  'Doge',         'csd',           'BTC',
  'Australia',   'bad',           'Kurnell',
  'Beetlejuice'
]
GET /myTrend 304 563.234 ms - -
GET /manifest.json 304 3.659 ms - -
GET /favicon.ico 304 4.316 ms - -
GET /googleTrend 304 883.506 ms - -
[]
```

6.

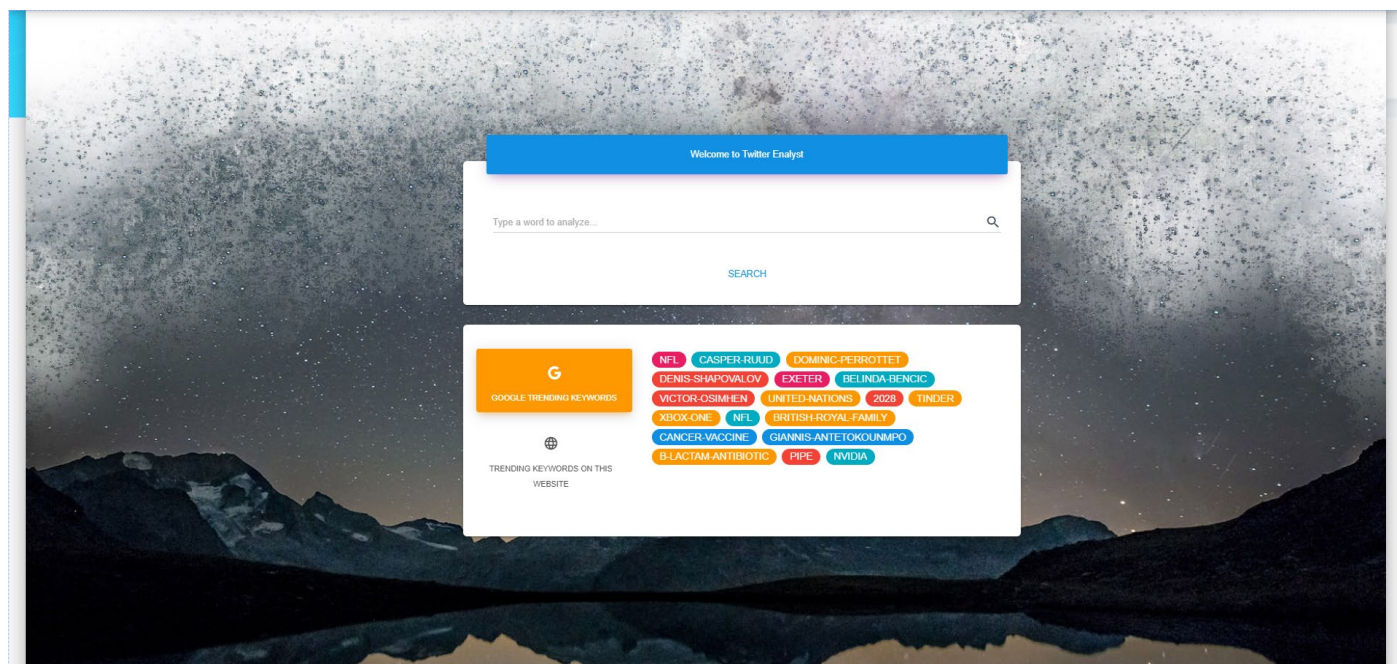


7.

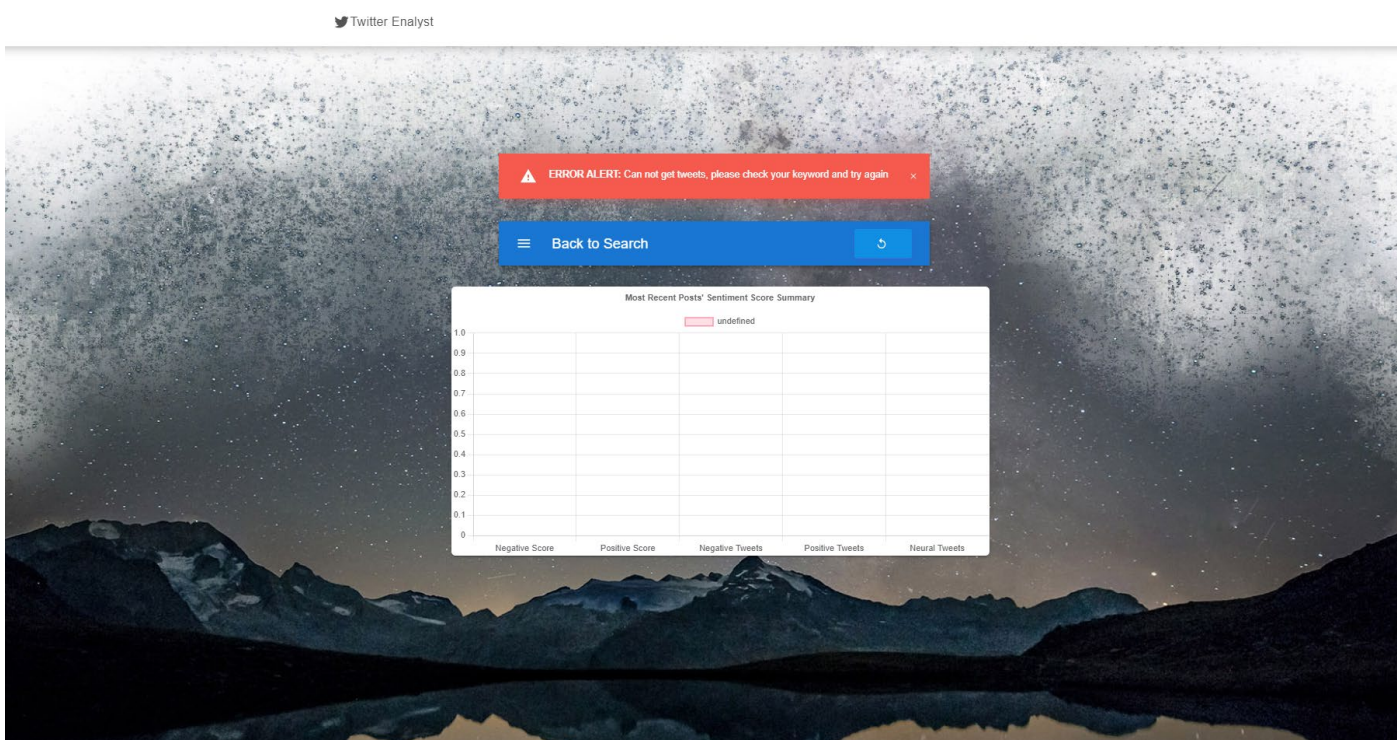




8.




9.




10.

## Twitter Posts Found




**Sandy** 🇺🇸🇵🇸🇺🇸  
@Sandy\_L\_K

Let's Go Brandon! 🇺🇸🇵🇸



**Fox News** @FoxNews

Biden apologizes for being late to G20 press conference: 'We were playing with elevators' foxnews.com/politics/biden...

12:05 PM · Nov 1, 2021

 Copy link to Tweet

[Tweet your reply](#)

**@uppermoony**

"Light Yagami, L, Kira. You have lost." #deathnote

