

**PROJECT REPORT**

**Twitter Search Application**

**“Chirp Search”**

**Prepared by**

Vishnuram Jatin Bangaru(vb440)

Ankeeta Priyam(ap2213)

Noopur Singh(ns1482)

Ganesh Raj Kyatham(grk62)

Table of Contents

[**INTRODUCTION** 3](#_Toc132218008)

[**PROJECT ARCHITECTURE** 4](#_Toc132218009)

[**PROCESSING TWEETS FOR STORING IN DATASTORES**](#_Toc132218010) 5

[**SEARCH APPLICATION DESIGN** 7](#_Toc132218011)

[**RESULTS** 9](#_Toc132218012)

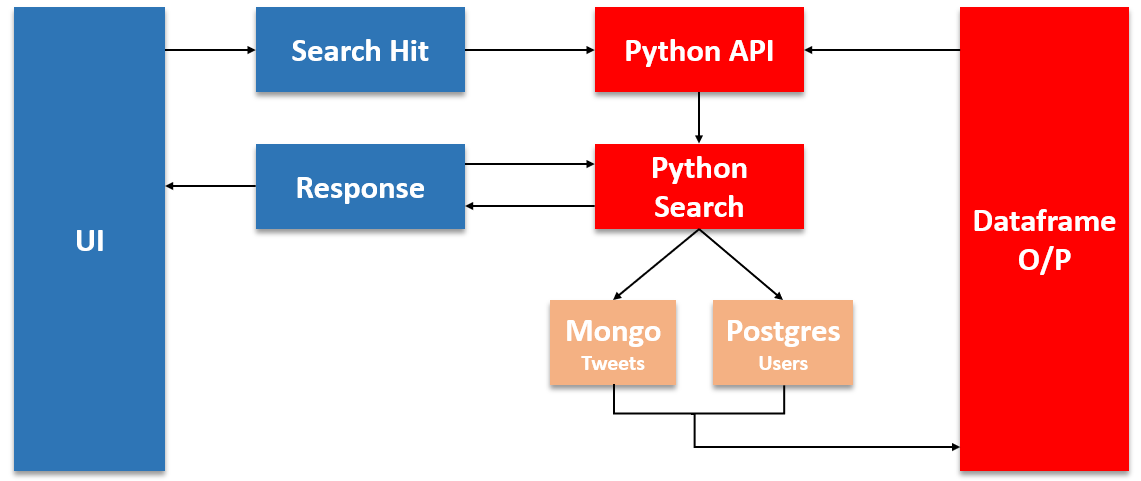
[**REFERENCES** 11](#_Toc132218013)

# **Introduction**

Twitter is a popular social media platform that enables users to share short messages, known as tweets, with a wide audience. With over 330 million monthly active users, Twitter has become a critical communication tool for individuals, businesses, and organizations worldwide. The platform's real-time nature and broad reach makes it a valuable source of information for breaking news, trending topics, and public opinion. Overall, Twitter plays a significant role in shaping public discourse and connecting people from diverse backgrounds and perspectives.

As part of the MSc Data Science program at Rutgers University, State University of New Jersey, course 954:694:01, the project is started that involves developing a search application utilizing Twitter data. The objective is to design and store the information in multiple data stores to enable rapid access via Python. Additionally, the project involves implementing a caching system for the top search results to minimize the need for repeated visits to the data stores.

# **Project Architecture**



*Fig. 1*

The project architecture displayed above depicts the flow of information. Firstly, the UI receives input from the user and sends it to the Python API.

The Python API then directs the control to the search function in the Python source code. Within the search function, a query string is formulated based on the input received from the user, and it is sent to the mongo dB context, where the Tweets table contains the relevant tweets.

The output of this search is returned to the Search Function, which generates an output Data frame that includes the tweet, username, and the number of retweets for that tweet. The Python API then receives this result and sends it back to the UI, which displays the response to the user.

If the user clicks on the username or number of tweets, the Python API is called again, and depending on the selection, either the Users table in PostgreSQL (for user click) or the Tweets table in MongoDB (for tweet click) is queried. If the user clicks on the username, the details of the user are returned. If the user clicks on the retweet count, the retweeted tweets are returned.

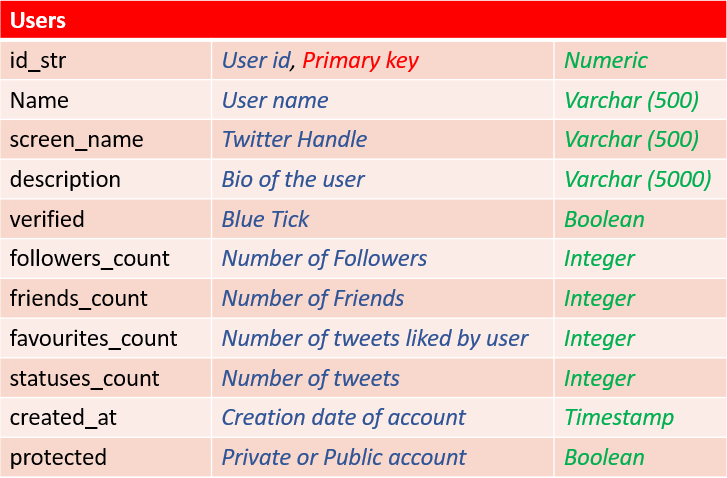
**Persisted Data Model and DataStores**

The project incorporates two different types of databases - Relational and Non-Relational. For Relational database management, PostgreSQL is employed, and for Non-Relational database management, MongoDB is utilized.

The project have connected both of these databases to python and performed insert, delete, query and other basic operations.

The tweet dataset is distributed between these databases. User level details are stored in the Users table in PostgreSQL, while tweet-related details are stored in the Tweets table in MongoDB.

The Users table's architecture and schema are displayed in the figure 2.

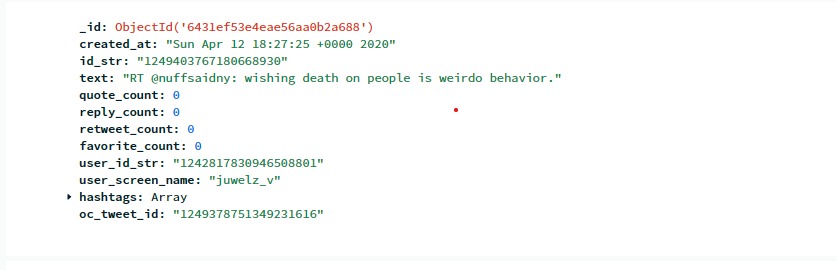


*Fig. 2*

PostgreSQL is a popular open-source relational database management system that uses SQL to store and manage data in tables.

The schema is designed such that the data is stored and accessed efficiently. Each tweet is processed, and user detail is extracted from each and inserted one by one using the command “Insert Into” in the Users table.

The Tweets table's architecture and schema are illustrated in the figure 3.



*Fig. 3*

MongoDB is a popular NoSQL database that stores data as JSON-like documents in collections. To insert the tweets, we have made use of dictionary collection. Formulated the required data into key-value pair and inserted into the table.

The Fields in the Mongo DB tweets collections:

\_id : It is an default Object ID which MongoDB assigned when a collection is inserted.

created\_at : It is the timestamp at which the tweet was made by the user

Id\_str : ID of the tweet

Text: The tweet of the user

quote\_count : Number of quoted status

reply\_count: replies on the tweet

Retweet\_count: the number of times it was retweeted.

Favorite\_count: number of likes on the tweet.

User\_id\_str: id of the user who made the tweet.

User\_screen\_name: screen name of the user.

Hashtags: The hashtags made on the tweet.

Oc\_tweet\_id : Original tweet id. Also indicates if the tweet is a retweet or not.

**SEARCH APPLICATION DESIGN**

The project has allowed for basic searches where a user can search based on user\_name, a tweet string or hashtag. The search would retrieve information about the tweet made, when it was made, the retweet\_count, the user details (no\_of\_followers, user\_name). The project will expand the search results functionality after the minimum queries have been handled.

The project expands by providing drill downs by clicking on the retweet count, user\_name to further display details of the retweet or the other tweet details made by the user.

Suppose the search is based on a string, we would clean the string i.e split words into a list, remove stop words and trigger a python function which would retrieve the tweet details containing all the words in the list using the find operator and regex operator from MongoDB. We would then retrieve the user details from the User Table using the foreign-key alike user\_id\_str and merge to get the final output. This is a bare bound approach which we are working for the search queries.

We are planning on using a number of factors to order the search results. The factors include retweet\_count, quote\_count, reply\_count.

Cache Implementation Design :

The input search string will be cleaned before its passed to be queried. It is planned to store string in a dictionary with key being the string and value being the number of times this string has been searched. Every time a new string is searched, the string is compared with the dictionary's keys and the count is incremented to the key that matches. If not it is appended to the dictionary with count 1. The counts are sorted and a top n number of records in the dictionary are used to query the tweets and stored to be accessed. This process is repeated each time as new string is searched. We are yet to come up with a better key for dictionary.

# **Team Work And GitHub**

# The work has been split between the team members:

# Ganesh Raj – He is working non-relational database to store the tweet information on MongoDB. He will work on the basic Search Engine UI.

# Vishnu - He is working on the Query Processing and will contribute towards the Cache functionality.

# Noopur - She is working on Python API processing and will contribute towards the Cache functionality.

# Ankeeta- She is working on the relational database to store the user details and will work on the query processing.

# GitHub Link - <https://github.com/ankeetapriyam/694-2023-7>

# GitHub usernames –

a/ ankeetapriyam (Ankeeta Priyam)

b/ vishnuramj10(Vishnuram Jatin Bangaru)

c/ Noopur665(Noopur Singh))

d/ ganeshraj-k( Ganesh Raj K)

# **RESULTS**

Show the results of each type of query.

Timings of your test search queries (make sure you are hitting cached and non cached data)

**CONCLUSIONS**

# **REFERENCES**