COM3504 Assignment

# Introduction

This project is a system that allows a user to easily query Twitter for Tweets relating to football, by building a search query based on text in the tweet, hashtag, and users involved. The querying and storing of Tweets in the database has been implemented, and a user interface for this has been created. The interface at the moment is only a generic Twitter searcher, and does not have anything to allow you to do football specific searches, such as easily searching for anything related to a certain team. The additional features section has not been implemented.

# Querying

## Issues

Querying Twitter using its API is the key component for this assignment. We faced no issues with this part as the Twitter API is well documented and easy to use, and the NodeJS library provided a simple interface.

## Design Choices

The search is done in two steps, first the database is search to see if the query has been done before, if it hasn’t it moves to the next step, if it has, the previous Tweets for that query are retrieved, and then it moves onto the next step. Searching Twitter is then the next step (unless it’s a database only search), the query and last ID of the Tweet previously retrieved from Twitter for this query is sent to the Twitter API and the results and appended to the Tweets retrieved from Twitter, as well as being stored in the database, and the last Tweet ID for this query is updated. Once the Tweets have been retrieved they are parsed, where the frequent words and active users are extracted. The list of Tweets and frequent words and people are sent to the client as a JSON object. This database search first prevents unnecessary data from the Twitter API.

# Storing

## Issues

Previous queries and the returned Tweets are stored in a database, this prevents unnecessary data from being retrieved from Twitter, and allows Tweets to be returned past the 7 day limit of the Twitter API.

## Design Choices

The tweet cache is stored in a MySQL database with two tables. A query table has the Twitter query string as a primary key and the ID of the last Tweet matching that query which was retrieved from Twitter, this allows us to search for new Tweets since that ID. Another table contains all the cached Tweets, with the query string as an identifier and the entire JSON encoded Tweet. The advantage of this is that it allows you to easily get the cached Tweets for a specific query, saving on calls to the Twitter API. But as the Tweet is just stored as a JSON encoded string, no other search operations on the details of the Tweet can be done.

# Web Interface

## Issues

In web interface allows the user to generate arbitrary length queries. Making this interface provided had some challenges, as to allow these arbitrary length queries, a user must be able to add extra search parameters, and delete them.

## Design Choices

We used Bootstrap for the user interface, this allowed us to quickly create and style elements on the page without having to spend much time on the CSS and browser differences. For the JavaScript interactivity the Angular framework was used, this framework provided a simple interface for extracting data from forms on the screen, submitting the data from the form to the server, and then using its template system, the returned Tweet data from the server can be easily formatted and styled, without having to build the HTML in the JavaScript.

This interface fulfils the interface requirements as a user can build a query with an arbitrary amount of search parameters. A limitation of the current solution is that there is no pagination, so all the Tweets are displayed on the current page in one large list, which could be hard for a user to navigate; also a query can only either be entirely AND or entirely OR, can’t have AND between some search terms, and OR for the others.

# Additional Features

Currently no additional features have been implemented.

# Conclusion

There is still some work to be done, some improvements need to be made to the user interface to make it easier to use, and the additional features need to be implemented. The user interface and the backend API were done separately, which meant that there was some difference in how we both though the data would be organised, this would’ve been avoided with some better planning, but was easily overcome.

# Division of Work

The tasks for this piece of work were evenly distributed. Aaron provided the user interface in Angular, Kranti implemented the querying of the Twitter API, and the storage of the Tweet cache was done together.

# Extra Information

Run the server.js file and the application should be accessible on port 4000.