Introduction

This project is to design and implement a website to allow users to track and analyse discussion relating to football teams and players on Twitter. With the exception of additional features we have fulfilled the current requirements in the assignment description This includes an interface to track discussion, see analysis of it and see the locations too.

1.1: Querying the social Web

Issues

The main issues faced when performing the analysis of the returned tweets included processing the tweet itself. Some processing of the text was required to allow the processing of keywords such as @ManUtd to be the same as @manutd and @ManUtd: or .@ManUtd. A further issue was how to display the data and the design choices involved with that. Another issue was the question of how to store the data and what information was necessary to keep.

An issue that we faced with the implementation of a map showing locations of tweets was the fact that very few tweets had information about their location attached. By default when a Twitter account is created it will be set to not contain data about where the tweet was created. Due to this when 300 tweets are returned generally only a couple of contain the required information to plot them on a map.

Design Choices

The analysis works by making a “wordCount” for each tweet as we process the data. This wordCount is a javascript object working like a dictionary where for each word within that dictionary we are given a value that is the number of occurrences the word has within the text. By adding the keys and values into an ‘overall’ word count we can produce an array of the words contained within the collection and slice the highest 20 values. Along the way we can also record for each tweet each User, and for each User we can retrieve their personal wordCount, their username and the number of tweets they have written.

Our maps implementation has been done by making use of the freely available Google Maps API. This allowed for an interactive Google Map to be embedded into the analysis tab on our site and for markers to be shown at the locations of retrieved tweets. When these markers are hovered over with the cursor they display the given tweets content. The locations of the tweets were found by using the ‘place.full_name’ information returned with tweets and by using geolocator, which is included in the maps API, a plottable longitude and latitude could be returned.

Some design choices made along the way. We implemented a “trending hashtags” feature, where in addition to being presented in the top 20 keywords, the top 5 popular hashtags will also be displayed. Secondly, if a twitter user has only tweeted once, and they are in the top ten active users slice, they will not be displayed. This is simply because a user tweeting once does not have any significance.

This design meets the requirements within the specification. Exceptional situations where the code might struggle would be where the user's input characters that have not been seen before. While this would not be enough to crash the whole program it would mean tweets containing these non-characters would not be analysed and their contents added to the analytical results. The design is quite adaptable and simply relies on two things, a tweet's text and the user who tweeted it.

1.2: Storing Information

Issues

The main issue presented with storing information was designing the database. We needed to make it fast to insert and retrieve data, but also keep the memory overhead low and keep it simple so the queries didn't get too complex. We came across an issue where the database was taking a long time to store anything, after some time I realised it was due to a bottleneck of only one connection to the database, I therefore changed to a pooled connect, and while this works, it was important to make sure that I closed connections properly as due to the async nature of NodeJS it was easy to reach the connection limit. The last issue we had was trying to handle errors gracefully while still letting the user know that the database couldn't be connected to.

Design Choices

We decided on using MySQL for our database, via the DCS server (this means it will only work on the University Network or when VPN'd in), rather than a local storage solution for portability. We decided to use several tables linked via foreign keys for storage. The tables are as follows: users, tweets, tweet_search_link, searches, media, teams. We decided to do it like this rather than having one table with all the data because it gave less memory overhead for queries returning the same tweets and users. The cost of doing it like this is a rather complex SELECT function to retrieve data, and inserting takes a fairly long time - this isn't as important due to the async nature of the code the server is still responsive while inserting.

The database is queried in the same way you query twitter, through the route /search. When a twitter query is received in search.js it will first check the database to see if the database is storing any tweets for that query, if it is then the maximum tweet id is provided the the twitter call, so as to only retrieve tweets not in the database from twitter. The data from the database and twitter is then combined and returned. After this the new tweets will be saved into the database, so as to not make the user wait for insertion.

Limitations:

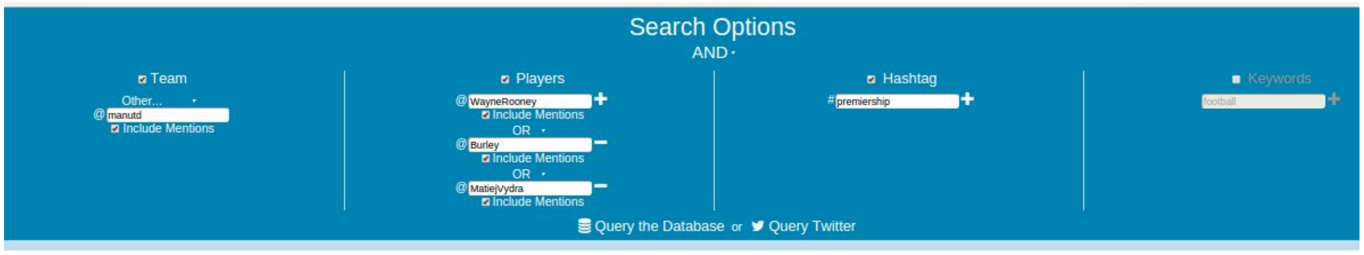
We make use of the database where possible returning a mix of database tweets and twitter tweets in a cached way, given a bit more time we would improve this functionality using search metadata to give a more complete coverage of tweets.
Currently our database will only return tweets which were searched for originally given the exact same query is provided, it does not have the function to query itself, however, due to how we implemented our database design; it will be easy to add this functionality later.

1.3: Web interface

Issues

The job of the web interface is to provide an intuitive way for users to search for and view returned tweets. A major challenge was to provide an interface which allowed for complex heavily customisable searches to be made whilst also keeping it simple to use and intuitive.

Design Choices



The design choice that we made for our search interface was to modularize it into the different searchable categories specified in the requirements. Each of these can be either included or excluded from the query by use of checkboxes placed next to the title of each category. The user is then able to specify whether they would like for the tweets returned to be ones containing all of the search elements selected or any tweet containing at least one of these elements. This choice is made by choosing either ‘and’ or ‘or’ from the dropdown menu at the center top. If the user would like to include multiple players, hashtags or keywords they are able to do so by using the ‘+’ and ‘-’ signs to add and remove additional text fields.

Once the user has finished their search, results are shown below the ‘Search Options’ box. These are presented down the center of page with two selectable tabs to change what is being shown. The default tab, ‘Tweets’, shows a feed of every tweet returned and the ‘Analysis’ shows information of the frequencies of keywords, the top 10 users for the search and an interactive map showing the locations of tweets.

We also added a drop down box to pick a pre-selection of teams which we are storing in our database, we plan on extending this function to the players section before the final deadline.

1.4: Quality of the Solution

Issues

The JSON data returned from the twitter API did not always contain the same data for each tweet, for example many tweets did not contain location data.

Design Choices

As described in the ‘Quality of the Solution’ section of the assignment description our solution makes use of javascript to check user inputs, ajax to handle updating information when searches are made (allowing the site to function from a single page) and data is handled and exchanged using JSON.

1.5: Additional Features

Due to time constraints our project currently has no additional features beyond what is specified in the requirements. We will however be adding some in time for the next hand-in.

Conclusions

A lesson that we learnt as a team was to pull together our different interpretations of the sometimes vague specification. By offering our different idea’s about what we thought was expected we managed to create a much more thorough plan of what we wanted the system to be.

Learning to write asynchronous code was also a lesson well learnt as it required a different way of thinking compared to most coding that team members had done before. It took a bit of time to adjust to the new style of writing but by learning it together we all got the hang of it fairly quickly.

Division of Work

Work for individual team members was decided at meetings scheduled (roughly) weekly to discuss to progress and plans for the following week.

Initial setup of the node server was completed as a team during a meeting.

From there individual members worked on the following topics;
Paul - Database design and implementation, queries made to the Twitter API, search interface
Alex - The analysis tab
Ross - Implementation of the map & location tracker, search interface design

Extra Information

You must be on the uni network for the database to work.
Run via `node solution/bin/www’ or `npm run`