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; a user can also perform a query on the web of data about two football teams.

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.

A search about two named football teams can be performed against DBPedia, this query will return the team name, description, manager, stadium, and players. More details can be loaded about a specific player such as their career history.

# 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.

# Producing a tool for sports journalists

## Issues

This web tool allows you to enter the names of two football teams, it then searches the web of data for the basic details about the two teams, and the players. More details about a player, including their career history can be accessed by clicking on a player’s name.

There were a few issues that were encountered when developing this section; first the user must enter the name of the football teams exactly for them to be found, there is no fuzzy searching, due to the complexity of this it was not added. For players and managers there are multiple possible names and positions returned for them.

## Design Choices

There is a basic form in the interface where a user can input the date and the name of the two teams they want to search for. Pressing the search button then makes an AJAX request to the server with the search criteria; for each of the teams: the team name is inserted into a pre-written SPARQL query that searches for the basic data about the team, manager, stadium, and players, if this search succeeds then the returned data is formatted into an object, another SPARQL query is performed that returns the basic information about each of the players in the team, this is then also formatted into an object and inserted into the team data, once searches for both the teams are complete, the two objects are merged and sent to the client as JSON. This data is then formatted on the screen using Angular. Clicking on a link next to a player name you can retrieve more information about the user, this is done by sending the player URI to the server using AJAX, which then performs another SPARQL query that gets the basic details of the player, and their career history, which is sent back to the user as JSON and formatted again using Angular as a modal on the screen. Each type of SPARQL search is defined in a function that takes the search parameter and a callback function, this callback is then called with the search results. Using this allows the same search to be performed in multiple places and the stacking of searches.

All the requirements have been met: can search for two teams; returns team name and description; manager name, description, and image; stadium name, description, and image; players name, image, date of birth, position, and number. Everything is linked back to their corresponding DBPedia pages. Extended player data including name, dob, position, number, description, and career history can be viewed.

There are some limitations, an exact team name must be entered for it to find anything, acronyms and misspellings return no data. This got be fixed by implementing fuzzy searching for the team name and let the user pick from a list if multiple possible teams are returned.

# Producing Data for the Web of Data

## Issues

The page which displays the team and player data should be readable by a web crawler so that it can understand the information being shown. There were no issues encountered.

## Design Choices

This is simply done by having RDFa attributes attached to the HTML elements in the page which describe the data being shown on the screen. These attributes have been simply added to the Angular HTML template and does not affect how a human user uses the system, as it is not visible to them. Again the usage of a template system was an advantage in doing this as it took no time to simply added these attributes to the existing template.

The implementation meets all the requirements as subjects, relations, and ranges are added into the page using RDFa.

Only limitation that as this system uses AJAX to fill in the page there is no direct URL that can be loaded to get this RDFa data, the form must be submitted.

# Additional Features

No additional features have been implemented in the social searching section. Career history of a player has been added in the journalists’ tool.

# Conclusion

There is still some work that could be done, some improvements could be made to the user interface to make it easier to use, and not many additional features were implemented. The user interface and the backend API were done separately for the social searching section, 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, and for the journalists’ tool section we prevented this problem with better communication and planning.

# Division of Work

The tasks for this piece of work were evenly distributed. Aaron provided the user interface for the social and journalist tool in Angular, Kranti implemented the querying of the Twitter API, and the storage of the Tweet cache was done together in a pair programming environment. Aaron implemented the SPARQL querying for the team and basic player details, the extended player details with career history was done in a pair programming environment as well.

# Extra Information

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