COM3504 Intelligent Web Report

Introduction

The aim of this assignment was to build a program which would enable a user to create queries that would retrieve data from the social web and web of data. A user can search for tweets published to twitter based on a number of different criteria and retrieve in-depth information about two football teams.

The query page makes use of twitter's SEARCH API to allow users to write queries that can be built up in many different ways. Users may search using the following terms: team, players, hashtags or keywords [1]. Two buttons are provided on the interface which allow the user to choose whether they would like to search twitter for ALL of these terms or ANY of these terms.

If successful, tweets will be retrieved and displayed on the screen in a format very similar to that of twitter. In addition to the tweets retrieved pane, a statistics pane is displayed, this contains the statistics listed in the assignment handout in section 1.1 [2]. Finally, a google map entered on the UK is displayed which shows all geo located tweets if any have been found.

Information retrieved from these queries is stored in a web accessible database. The database is utilised as a cache to reduce the number of queries sent to the twitter API by providing data that has been found as a result of past queries. Alternatively, it is possible to query the database on its own by selecting this option from the interface.

The journalist page allows a user to enter two teams and a date. The results page shows, for each team: a description of the team, a list of all of the players on the team along with their position and DOB, the manager - with a photograph and description and finally the stadium with a photograph and description. Players are displayed in a side scrolling bar as tiles, clicking a tile will open a profile pane for that player which shows more detailed information than that on the tile including their career history. Journalist pages contain RDFa so that generated pages can be read by a web data crawler.

Our interface is implemented using HTML and Javascript and is served by a node.js server [3]. All interaction with the social web is carried out through this server.

One additional feature has been implemented. The journalist results page displays a map for each team that the user has searched for. The map displays the birthplace of each player on the team with markers that expand to show player details when clicked.

1.1 Querying the Social Web

1.1.1 Issues

The task for this section was to provide an interface which would allow a user to query the social web and display the results. The interface had to allow users to build complex queries using several terms, which could be used with twitter's SEARCH API to retrieve relevant tweets from twitter. Relevant tweets needed to be displayed with some statistics calculated and their locations shown on a map (if applicable). The main challenges faced when working on this section were: calculating statistic information efficiently, validating user input to make sure valid data was sent to the server and ensuring that an appropriate number of tweets were retrieved.

1.1.2 Design Choices

The interface is designed as four text fields with buttons to search using ANY or ALL of the entered terms as described in the introduction. A google font was used throughout the system [4].

The ways through which user input is validated are described below. It should be noted that a user is free to leave any field blank, but they must enter information into at least one field before validations will be started. The entered team must exist within the database, a choice of available teams will be provided in an autocomplete drop down (the drop down was implemented with the jQuery autocomplete plugin) [5]. Players must be entered separated by commas. Each player must exist within the database and, as before, suggestions for players will be provided in an auto complete drop down. Hashtags must be entered separated by commas and are validated according to twitter's rules for a valid hashtag [6]. Keywords must be entered separated by commas and are not validated - they are designed to give the user the freedom to enter any term.

When data is retrieved from the server the tweets are parsed through once. The advantage of this solution is that unnecessary looping over large amounts of data is prevented. It was found that this could result in wait times of a few seconds and so a loading bar is displayed so that the user does not mistake this for a broken or unresponsive page. The loading bar image was generated as a gif using an online tool [7].

A recursive call has been used to make sure that 300 tweets are returned for each query. A call is made to twitter to retrieve the first 100 relevant tweets, if these exist, a recursive call is made to collect the next 100 and so on up to 300. If at any stage fewer than 100 tweets are returned then the recursion stops as all relevant tweets have been retrieved.

1.1.3 Requirements

To the best of our knowledge, all requirements for this section have been met.

1.1.4 Limitations

All work to calculate statistics is done in one parse of the data, this means that there is a small delay before any information is displayed. It may have been faster to display just the tweets as soon as they are retrieved and calculate the statistics separately.

1.2 Storing Information

1.2.1 Issues

The task for this section was to design and create a database capable of holding relevant information collected from queries. The database needed to be used both as a source of data in its own right (with the option to search only the database provided on the interface) as well as a cache to reduce the number of queries sent to the twitter API. The main challenges associated with this section were designing an effective database schema and implementing caching.

1.2.2 Design Choices

The database was designed with tweets as the focal point. All the tweets are associated with the user who wrote them and the remaining data associated with the tweet (media, hashtags etc.). As a result, the schema is broken down into the elements of a tweet. This provided a way to reduce the total amount of data stored as repeated content is only stored once in a linking table. This further extended the ability of the system to pull specific collections of data efficiently.

The names and twitter handles of the teams and players currently in the premiership league have been stored in the database. The only accepted values at validation stage are those provided in the database. This makes the range of acceptable values clear to the user and ensures that the program can only be used for football related queries. When a query is submitted, the twitter handle or dbpedia link is submitted as part of an Ajax call as appropriate whilst the user is able to enter the team name in a human readable format.

1.2.3 Requirements

To the best of our knowledge, all requirements for this section have been met.

1.2.4 Limitations

Teams and players that have been added to the database for validation purposes must be added manually and continuously researched so that they are correct and up to date.

1.3 Producing a tool for sports journalists

1.3.1 Issues

The task for this section was to provide an interface which would allow a user to enter two teams as well as a date for the purpose of providing a tool for sports journalists. The query had to be implemented as both a twitter API and database search in the same fashion as the page described in section 1.1. The tool needed to provide detailed information about each of the entered teams as collected from DBPedia. The information provided to the user is described in the introduction. The main challenge for this section was querying the web of data.

1.3.2 Design Choices

The interface has been implemented as three text fields which allow a user to enter two teams and a date. The valid choices for teams are those which exist in the database and are suggested to the user through an autocomplete drop down as described in section 1.1.2. The date entered must be a valid date that is in the future.

SPARQL has been used to query DBPedia, the query terms were constructed using the RDF types from pre selected teams and players as examples [8]. The SPARQL client plugin has been used to generate query requests as described in the lecture slides [9].

1.3.3 Requirements

To the best of our knowledge, all requirements for this section have been met although our solution could be improved to reduce the limitation described below.

1.3.4 Limitations

In order to determine the types of resources to be retrieved the DBPedia page for the Manchester United Football Club was examined. These types have been used for all queries which introduces a key limitation - if a different football team has resources stored under different types these will not be returned. For example, if a user attempts to search Manchester City Football Club some resources are missing.

1.4 Producing Data for the Web of Data

1.4.1 Issues

The task for this section was to make sure that all pages generated as part of the journalist tool (the initial query results page and the player profile pages) contained relevant RDFa so that they could be found by a Web of Data crawler. Relevant RDFa was the subjects, relations and ranges for each piece of information extracted from DBPedia. The main challenge associated with this section was including the RDFa information on dynamically generated content.

1.4.2 Design Choices

The relevant RDFa for each resource is retrieved from DBPedia alongside the resource itself. The RDFa is then dynamically generated on the page.

1.4.3 Requirements

To the best of our knowledge, all requirements for this section have been met.

1.4.4 Limitations

Extensive RDFa has not been provided, only those terms as requested on the assignment hand out in section 1.4. Furthermore, RDFa is not provided for the twitter query page or any of the static content. By adding this additional RDFa the site could be used more extensively by Web of Data crawlers.

1.5 Quality of the Solution

1.5.1 Issues

The task for this section was to produce a solution of the highest quality possible. The main challenges associated with this task were to produce an interface that displayed relevant information and the storing large amounts of complex data in an efficient way.

1.5.2 Design Choices

Steps have been made to improve the overall efficiency of the system. Javascript is used to perform client side validations. Calls are made to the server as Ajax requests which means that the client can remain active whilst the server fetches the results. Information is then sent to the client as a JSON object [10].

We have broken complex data down into its basic information components to ensure minimal data repetition in the database.

1.5.3 Requirements

To the best of our knowledge, all requirements for this section have been met.

1.5.4 Limitations

The google map is implemented but hardly used. It would be beneficial if further ways to utilise this map were implemented in the future.

1.6 Additional Features

1.6.1 Issues

The task for this section was to design and implement an additional feature that would enhance the program results or ease its use. The main challenge associated with this task was choosing a feature to implement, and several options were considered.

1.6.2 Design Choices

The team results panels on the journalist brief page each display a map which shows the birth place of each player on the team. Each marker on the map is interactive and can be clicked to display the players name, date of birth and photograph.

1.6.3 Requirements

There were no specific requirements for this section. We believe the feature that we have implemented has been done well although there is room for natural expansion to the system with further features such as a weather display at each stadium on the entered date.

1.6.4 Limitations

The google maps implemented are a fairly simple feature.

Conclusions

We think that splitting the work into a client and a server side worked well as we were each able to focus on different areas of the project. If we were to repeat the project we would explore different ways of displaying tweet information instead of replicating the tweet display style found on twitter.

Division of Work

We both agree that there has been a 50/50 division of work. As the assignment had a client and a server side this seemed to be the most natural way to divide the work.

James worked on the server side, writing the code for the node.js server and querying both twitter and the web of data. He also produced the database. James retrieved RDFa for pages.

Nicola worked on the client side, writing the code to display results from both twitter and journalist queries, calculate statistics and display a google map. Nicola stored RDFa on pages.

Documentation was produced by us both.

Extra Information

No additional information is required to run the code.

Bibliography

[1] Twitter SEARCH API, https://dev.twitter.com/rest/public/search

[2] Assignment Handout, https://vle.shef.ac.uk/bbcswebdav/pid-2243684-dt-content-rid-4974567_1/courses/COM3504.A.154617/AssignmentFinal%20%202015-16.pdf

[3] Lecture Week 2, https://vle.shef.ac.uk/bbcswebdav/pid-2205917-dt-content-rid-4890265_1/courses/COM3504.A.154617/week%202%20accessing%20the%20Web%284%29.pdf

[4] Google Font, https://www.google.com/fonts/specimen/Oswald

[5] JQuery AutoComplete Plugin, http://jqueryui.com/autocomplete/

[6] Twitter Hashtag Rules, https://www.hashtags.org/featured/what-characters-can-a-hashtag-include/

[7] Loader Icon Generator, http://www.ajaxload.info/#preview

[8] SPARQL, http://dbpedia.org/page/Sparkle

[9] Lecture Week 8, https://vle.shef.ac.uk/bbcswebdav/pid-2244503-dt-content-rid-4978115_1/courses/COM3504.A.154617/Week%208%20RDF%20and%20SPARQL%281%29.pdf

[10] Lecture Week 5, https://vle.shef.ac.uk/bbcswebdav/pid-2222544-dt-content-rid-4916571_1/courses/COM3504.A.154617/Week%205%20socket.io%20WebRTC.pdf