Maciej Zak
Rafael Casabianca
Andrew Cook

Report: Team 64

Introduction

This report outlines the construction of the assignment code and the design motivations behind our web application, following the guidelines and requirements detailed in the assignment specification. Our solution uses the Express.js framework, runs on a Node server, and renders HTML views using the JADE template engine. Our styling is composed of basic Bootstrap elements, making our solution compatible with the majority of current browsers. Our code, at first glance, differs stylistically from code presented in the lectures - partly due to our choice of technology and partly due to our frequent and direct consultation of the Twitter API documentation and JavaScript documentation.

Section 1.1

Solution

Express is the web framework we decided to implement our solution in. This his provides the ease of transition into the second part of the project, as Express allows easier integration of login and the implementation of sessions. It also provides us with the ability to include string patterns in routing paths, which can be useful in the later implementation, once our app will consist of a larger network of URLs.

Express also provided a number of advantages for this part of the project, highlighting again why this was our technology of choice. Express generator was used to set up the initial configuration of the app, from which we could build upon. This provided a quick and simple setup of all the initial files and basic structure and skeleton of the app, allowing us to focus on the advanced features that we wanted to implement. Routing was a feature of Express which proved very easily understandable by the programmers, and also robust in usage for our application - the use of methods, paths and handlers was intuitive and proved to work well with our desired application functionality. This included GET methods for the Twitter tweets search functionality, as well as POST methods for obtaining parameters for the search. These were then converted into appropriate OR and AND queries depending on the inputted fields. Upon selection, the queries can be executed on the database, which is possible through the simple database connection, being another advantage of the implementation with Express.

Issues

The main issue we encountered while implementing the search function was the integration of the Jade template language and the Bootstrap framework for the user interface, not the search function itself. Since both are designed mainly for the use with pure HTML, some code conversions proved difficult. This made us choose appropriate elements on the interface more carefully, and sometimes redesign the ordering in order to get the layout and functionality we needed.

Maciej Zak
Rafael Casabianca
Andrew Cook

Requirements

Our solution utilises the Twitter REST API, allowing the querying of search terms and the returning of these tweets as a JSON object. This is made possible through our Node.js server **app.js** and runs on most modern web browsers. Users may query by player names, screen names, appropriate hashtags, team names, and tweet authors. Users may choose to query the API or the Database. The user may query using Boolean operators as described in the specification. The AND operator is implicit between the HTML text fields, but the OR operator must be entered between search terms inside the text field, as outlined in the API documentation.
The system outputs tweets as a list in the correct format. 300 tweets are returned if they are available, however, the API only returns tweets made in the last seven days. Appropriate hyperlinks are implemented in the returned list to both the tweet author and the original tweet itself.
The chart tracking the frequency of tweets over time can be found at the bottom of the page.

Limitations

Our solution incorporates the Express.js framework, with the file structure of the solution being generated by the **express-generator** node module. Extensibility in terms of multiple route scripts and views being added should raise no problems. However, our server-side application could be modularised more - as all server side functionality is contained within **app.js**. If the database becomes too large, this may raise performance issues with the GET requests. It may be a more practical solution to consider splitting the returned results into pages with a maximum of 50 tweets, and options to navigate between pages.

Section 1.2

Solution

All the information queried by the Twitter API is stored in a web accessible mysql database. The program consists of two search options: API Search and Database Search. Users are free to choose whichever search mode they want by selecting the corresponding radio button on the UI search bar.

The first option is to use the Twitter API to fetch new tweets. This option returns all the new tweets that fit the corresponding parameters directly from Twitter. To do this, the ‘since_id’ option from the API is implemented in the search function. This allows the query to only fetch tweets that have an id greater than the selected one. This parameter is calculated by getting all tweets that have the searched team and player names and returning the id of the newest tweet. This value allows the system to be more efficient by reducing the number of queries. Once the tweets are fetched, they are sent directly to the UI and then stored in the database.

The second option is to search on the database only. By selecting this option, the system returns the values that correspond with the search parameters directly from the database. There is no call to the Twitter API. If there are no tweets matching the search parameters, then the function will not return an empty list.

The database stores the player name, the team name searched with each input, as well as all the important values returned by the API. These include the tweet id, user, date, time and text of each tweet. This design makes it easier to search for data, as it allows the user to filter the information by player and team, to see how likely it is for a player to move to said club. But it also allows users to see the rumors of a specific player and all the potential clubs.

The main advantage of this solution is its efficiency. This allows the system to fetch and display information quicker. It not only is time efficient, but memory efficient as well, as it avoids duplicate items to be stored in the database.

Issues

Information storing is crucial for the software and it has done in the most efficient way possible, avoiding duplication and reducing the number of queries to a minimum. During this implementation, some issues were encountered. Due to all the different searching options (e.g hashtags, user mentions and keywords), the system might return the same tweet but with a different value on the player or team name, making it a different entry. To avoid this, the tweet id was marked as a unique key. This allowed the program to use the mysql INSERT IGNORE command, to avoid entries that contain a duplicate key when inserting data into the database.

Requirements

Our design allows the user to search using Twitter API and the Database. The database is accessible using the same web interface used in Tasks 1.1. As described in the “Solution” subsection, searching via the API fetches only new results, with the database acting as a cache. The user may also opt to search exclusively to the database (as a source of information), which makes no call using the API.

Limitations

A bug that remains in our code is searching empty strings twice, which raises an error. The database has potential to become very large after some use, so searching the database may cause performance issues. A function to delete old and uninformative tweets to counteract this would be necessary.

Division of work

Maciej and Andrew split the work on Section 1.1.
Rafael worked on Section 1.2, helping with functionality on Section 1.1.1 and GUI.

Extra Information

The application may be run by navigating into the project folder and running “node app.js”
No other extra configuration is required.