Haskell Twitter API App

Temesgen Daniel-Teklebrhan

linkedin.com/temesgendaniel

Introduction:

Twitter API app is a command-line programme written in Haskell, it allows retrieving information relating to user and tweets using opensource Twitter API endpoints. and saves the information in an “SQLite” database. Twitter API offers different methods of authentication, however, for this use case, I decided to use the Bearer Token, which requires the application to pass an “Authorization” header alongside each call made to the API. This method of authentication is part of Twitter’s OAuth 2.0 implementation. It also means that I had to obtain an API key - in the form of a Bearer Token. I achieved this by registering an application on Twitter’s developer website. I chose twitter as website of interest since it presented an exciting challenge of having to parse dynamic real-time data with different formats, which meant that I had to implement multiple parsers to handle data returned by Twitter API. With further privileges granted the app could be extended to deal with authenticated user details like network of users and followers/following list.

What the App Does:

Once running the application displays several interactive options, the first option enables searching user by username to grasp information like name, bio, verification status, date of creation and public user metrics including following, followers and tweet counts. Following option is similar to Twitter’s own search function, it allows user to search for a specific topic keyword search query (e.g., “crypto”) and returns the latest 10 tweets and their corresponding metrics (likes, retweets, replies and quotes) along with a successive internal request that returns the user details of the tweet author. Alternatively, options to search user and tweets by Id is also given as separate functionality. There are also two functionalities that inner join query the database and checks if a user/tweet exists in the database. At the end of the program there is an option to export the database contents to a JSON files is provided to the user.

The corresponding JSON files returned from the above requests are then parsed to fit into custom User and Tweet data types and saved in an SQLite database that comprise of two tables, Users and Tweets. Those tables have one to many relations where then

primary key (user_id) of the users table is a foreign key of the tweets table as shown in the ER schema diagram below.

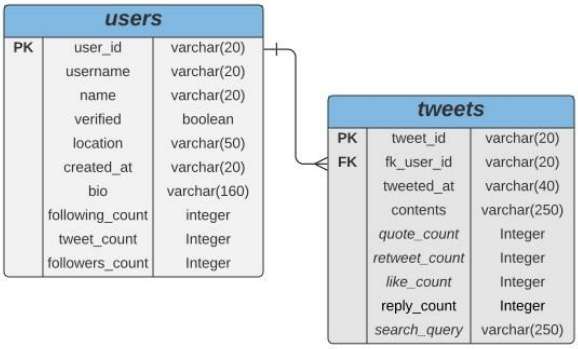

Figure 1: Database Schema

Error Handling:

Several Error handing actions were performed by the app to prevent crashing and unexpected outputs, staring with the fetching process the app returns an HTTP Exception error and exits the program if the user is not connected to the internet, likewise bad response resulting from invalid requests are handled by confirming the response status code to be 200("**OK**") before continuing to return the response output. User input validation is also performed at every step of the program, validating usernames by constraining it to be alphabetic between 4 and 15 characters long, and Tweet ID to be only numeric of alike length, prompting incorrect entry and restarting otherwise. Moreover, error returns by twitter were parsed separately to handle empty search results, suspended users or non-existent user as well as and deleted tweets.

Extra Features:

The programme utilizes Twitter API as its data source, this is not a single source of data, in fact, the application queries multiple endpoints to retrieve, parse and store data.

When searching for users using the Twitter API, the "author_id" field (i.e., the user ID of the person who authored the tweet) is returned, however no further information such as their name, whether they are verified, etc. is given. In order to display more useful information to the user This information can then be more easily analysed within the "SQLite" database, for example, by querying tweets with the highest "like_count" for a topic.

This programme provides a base for extracting information from Twitter and storing it in an “sqlite” flat-file database. This information can then be more easily analysed within the “sqlite” database, for example, by querying tweets with the highest “like_count” for a topic.

Lastly, Twitter API allows us to query multiple data sources by providing us access to several endpoints that all return different data, thus increasing the variety of data this application handles.

Challenges and Limitations:

- Unfortunately, the application is limited to only a handful of queries. This limitation is imposed by Twitter, as they offer three distinct tiers of API access. Attempting to access any resources or endpoints outside of the scope of access will simply result in an error message stating that we require additional privileges (scope). Hence out app is limited to 900 user and tweet lookup for 15 minutes window which was enough for this implementation.
- Other major limitation that could arise with time since I am parsing real-time dynamic data the JSON file returned could change any time at the hands of twitter, this may render the parsing module useless and in turn the application as well.

Prerequisites

- Stack - https://docs.haskellstack.org/en/stable/README/
- Other modules are used in the project, however by following the instructions below these can be downloaded and compiled to work for the project.

How to Run the app:

1. Extract the archive into your chosen directory
2. Enter the Haskell-project directory and enter the following command. This may take a while as it will download and build any missing modules and dependencies needed to run the programme: ▪ **`stack build`**
3. To run the application, enter:

    ▪ **`stack run`**

4. Follow the on-screen instructions: