University of Leeds **School of Computing**

COMP3011, 2023-2024

Web Services and Web Data

A RESTful API for News Aggregation

By

Thomas Harling

201513413 – sc21twh@leeds.ac.uk

Date: 22/03/2024

1. Introduction

I have been able to implement all 5 views of the API specifications, with the working database and all can be run successfully locally on my own web service. I have also been able to implement a client application that works for all my views with the commands correctly implemented.

I have not been able to implement any of the Directory API and the client application elements that relate to this Directory API (the list command).

Created a superuser – username – ‘ammar’ – password – ‘instructor’ in my local database and in the Author table

I was unable to deploy my web service on pythonanywhere.

2. The Database

My database for this project consists of 2 tables, an Author table and a Story table to be able to store the information required for the successful execution of my web service.

The Author table consists of 2 fields, one for the author’s name stored as a character field with a maximum length of 100 characters, this is to store the full name of the author and then we have the second field, which is the user field. This is a foreign key field so that there can be a one-to-one relationship between a user and an author, this is also useful to utilise Django’s built in User as this allows for sessions in the application, which is vital to keep track of a user being logged in and allows only logged in users to access certain views in the web service. Additionally, the built in Django User allows for secure authentication of a user and the storage of their private credentials (passwords) using advanced hashing techniques.

The Story table is used to create and store the stories posted by authors in the web service, there are 8 fields, and an id field that is created when each story is posted and added to the database. The id field is what I have used for the ‘key’ in terms of identifying a story by its unique ‘key’ as the id is already implemented and very useful. The region and category fields can only be one of a small possibility of choices, that are set in the models file and the rest of the fields are used in storing the information about each story that is created. The author field is like the user field from the Author table and is a foreign key to ensure that each story can only be written by one author and enables this relationship to occur.

Both user and author fields are deleted if the admin user is deleted as this means that your author privilege is revoked if your User has been deleted and then this in turn deletes all stories made by the author as their author status no longer exists.

3. The APIs

The first API is the login API, this utilises Django’s built-in sessions functions. The first step after ensuring the correct request method was used, is to gather the guessed username and password of a potential user to the system. We then attempt to authenticate the user with the built-in authentication function to confirm whether a user is who they say there are or not. If the user has been successfully authenticated, we use the login function to log in the user which keeps track of the user across their status of being logged in and save the session status sent in by the request. We then provide a suitable HTTP response based on the result, a 200 code with a welcome message for a successful login and messages with corresponding codes for unsuccessful attempts.

For the logout API, we first check for an existing logged in user from the session sent in by the client in the headers. If there is no user, there can be no logout if there wasn’t a login, and correct response is sent. If there is a user, the logout function is used to logout the user and send a 200 code with a successful logout message.

The post story API requires a user being logged in before posting a story, if a user isn’t they are unable to post a story and receive a response about having to be logged in to post a story, I then locate the id of the user to be able to query and attach the author to the story. Once this has completed, I attempt to read all the story data and if successful, I set the date equal to today’s date and then create the story with all the story data from the request, the author’s id and the current date. This results in a successful post with code 201, and a failure to post a story results in a code 503.

Next is the get stories API, a user isn’t required for this API, so no session manipulation is involved to find a user. The criteria for the story filtering are inputted in the request, once this has been resolved, I begin to query the stories table. Initially, I query the whole table for all stories, then filter further if the date, region or category have been entered. I had to manipulate the inputted string date to match the stored format in the database to be able to check for dates after or on the inputted date. Then for every story that has been filtered, I created a JSON version of each story to append to the outputted JSON payload, I changed the stored regions and categories to a more human readable form for the users. If the length of the payload is 0 then I return a 404 code with no stories being found, but if this isn’t the case then I return the JSON payload of all the stories with a code 200.

The final API is for the delete story API, once again a user must be logged in to access this, so we perform the relevant session checks to find the user and messages, and 503 codes sent if the user isn’t logged in. However, before this we check for the additional parameter for this API which is the key of the story to delete, this is passed in through the URL in the client application, this key is then used to query the Story table for a story with id being equal to the key. If this cannot be found, an appropriate message is sent to say the story cannot be found in the database. If all checks are passed and the story can be found, it is then deleted followed by a successful completion message and a status code 200.

4. The Client

I implemented the client in my folders in ‘sc21twh_cwk1/newsAggregator/client.py’. When this file is run, you are entered into the main loop which requests a user enters commands, which can be ‘login *url*’, ‘logout’, ‘news [-cat=] [-reg=] [-date=]’, ‘post’ and ‘delete *key*’. These commands all relate to their own functions, this loop doesn’t end until the ‘exit’ command is inputted.

A session is created when the file is run and is set up for the user once they log in.

The only function that can be accessed without logging in that isn’t login is ‘news …’, this function is used in my scenario to select all stories from the Story table that match all the criteria of category, region and date. These criteria can be left blank, which means that this will not be filtered down based on this section. The first check is to check through all the sections of the input that follow ‘news’ and I check that all are valid and match the specification. If not, then no request is made as input is incorrect. If all are correct, the split-up input values are sent into the request with the URL, if criteria are left blank on the command line, they are sent as ‘*’ to the view function.

The client provides a URL to the login client function for the site for the user to attempt to log in to, they are then prompted for the username and password for the website, which are then used in the post request to the specified URL if it is one of the valid URLs. The response of the login API is then relayed to the user on the command line.

For the logout function, the session is checked to ensure a user is logged in before being able to log out as this wouldn’t achieve anything. I then obtain the cookies and csrf-token for the session to be able to send in the headers of the post request to be able to logout and stop the session. In the logout function, the session created before is set to None to indicate the session no longer exists. The user is then informed of the result of the request.

For the client post story function, the user is prompted for all the required inputs to post a story, if any of the inputs are invalid then no request is made, and the user is informed of this. The required cookies and csrf-token are once again sent in the headers of the post request to inform the view function of the existing session and the user that is in session. After completion of the request, the user is informed of whether the story was posted or not.

The final implemented client function is the delete story function, a session is checked for once again and if this exists, the key is checked to ensure it is a number as otherwise no story can be deleted. A valid key can be added to the URL of the delete request, as well as the headers containing the cookies and csrf-token again and the user can be informed on the outcome of their delete story request.

References

W3Schools – Django Tutorial

Django Tutorial. https://www.w3schools.com/django/index.php. Accessed 22 Mar. 2024.