**University of Leeds School of Computing**

**COMP3011, 2023-2024**

**Web Services and Web Data**

A RESTful API for

News Aggregation

By

Abhinav Ramakrishnan

201517997 sc21a2r@leeds.ac.uk

**Date:** 18th March 2024

# Introduction

I have been able to implement all the coursework. On the server side, this includes user authentication, allowing authors to log in and log out, meaning only authorized authors can access other functionalities. I have also implemented the ability for users to post stories, get all stories or stories based on filters, and the ability to delete stories based on a key they provide.

On the client side I have also bee able to implmenat

I have uploaded the Django server code to pythonanwhere.com. The client application is written in python 3.11.5 and has been thoroughly tested.

**URL**

sc21a2r.pythonanywhere.com

**Superuser Credentials**

username: ammar

password: ammar123

# The Database

The Django database model for the news service comprises two main tables: Author and Story.

The Author table represents the authors who can post news stories. This table is linked to Django’s built-in User model via a one-to-one relationship, enabling seamless integration with Django’s authentication system. It allows the author field to utilize the username and password fields from the User model, along with an additional 'name' field for storing the author's name.

The Story table stores the news stories posted on the service. It includes fields for the headline, category, region, author, date, time, and story details. The category and region fields are implemented as choices to restrict inputs to predefined options. For the category field, the options are: politics (pol), art, technology (tech), and trivia. For the region field, the options are: uk (United Kingdom), eu (Europe), and w (World). The author field establishes a many-to-one relationship with the Author table, linking each story to its respective author. One author can write many stories, but each story can only be written by one author. The date field stores the date of the story, while the time field records the timestamp of when the story was posted. Finally, the details field provides information about the story, limited to 128 characters.

# The APIs

**Log In**

The /api/login endpoint handles POST requests for user authentication. It extracts the username and password from the request payload and attempts authentication using Django's authenticate function. If successful, the user is logged in; otherwise, appropriate error responses are returned. Error handling ensures correct HTTP status codes and error messages are returned according to the specifications.

**Log Out**

The logout API in Django is implemented at `/api/logout`. It handles POST requests, checks the user's authentication status, and logs out the user if authenticated using Django's logout function. Successful logout returns a `200 OK` response with a goodbye message, while failed logout attempts return a `403 Forbidden` response.

**Post a Story**

The API endpoint /api/stories handles POST requests for posting stories. It checks user authentication, processes incoming JSON data for story details, and stores them in the database. Error handling manages exceptions during database operations, providing appropriate responses. Successful postings return HTTP status 201 (CREATED), while errors return status 503 (Service Unavailable).

The APIs are implemented using Django, a high-level Python web framework, which provides built-in tools for handling HTTP requests, routing, authentication, and database management. Here's how each API is implemented:

1. **Login API (Login):**
   * The API endpoint is decorated with **@csrf\_exempt** to bypass CSRF protection.
   * It checks the request method; only POST requests are allowed.
   * It retrieves the username and password from the request body.
   * Authentication is performed using Django's **authenticate** function, which verifies the credentials against the database.
   * If authentication is successful, the user is logged in using the **login** function.
   * HTTP responses are returned based on the authentication result.
2. **Logout API (Logout):**
   * Similar to the login API, it's decorated with **@csrf\_exempt** to bypass CSRF protection.
   * Only POST requests are allowed.
   * It checks if the user is authenticated and logs them out using the **logout** function.
   * HTTP responses are returned based on the authentication status.
3. **Stories API (Stories):**
   * The API is also decorated with **@csrf\_exempt** to bypass CSRF protection.
   * Handles both POST and GET requests.
   * For POST requests, it checks if the user is authenticated, reads JSON payload containing story data, creates a new **Story** object, and saves it to the database.
   * For GET requests, it filters stories based on provided parameters (category, region, date), retrieves them from the database, processes the data, and returns them as a JSON response.
4. **Delete API (Delete):**
   * Decorated with **@csrf\_exempt** to bypass CSRF protection.
   * Only DELETE requests are allowed.
   * Checks if the user is authenticated and retrieves the story object by its key.
   * Deletes the story object from the database.

Throughout the implementation, Django's built-in functionalities for handling authentication, request processing, database operations, and response generation are utilized. Proper error handling is also incorporated to handle various scenarios, ensuring the API's robustness and reliability. Additionally, Django's ORM (Object-Relational Mapping) simplifies database interactions, making CRUD operations straightforward and efficient.

The Login API (**Login**) within the Django application serves the purpose of authenticating users. This API endpoint is configured to accept only POST requests, utilizing the **@csrf\_exempt** decorator to bypass Cross-Site Request Forgery (CSRF) protection. Upon receiving a POST request, the API extracts the username and password from the request body. It then leverages Django's built-in authentication system, specifically the **authenticate** function, to verify the user's credentials against the database. If the authentication is successful, indicating valid credentials, the user is logged in using the **login** function. The API returns HTTP responses accordingly, notifying the client whether the login attempt was successful or unsuccessful due to incorrect credentials or other issues.

The Logout API (**Logout**) provides functionality for users to terminate their session and log out of the system. Similar to the Login API, it is also decorated with **@csrf\_exempt** to bypass CSRF protection and accepts only POST requests. When a POST request is received, the API checks if the user is currently authenticated. If so, the **logout** function is invoked to terminate the user's session. Subsequently, an appropriate HTTP response is returned, informing the client of the successful logout or indicating that the user was not logged in to begin with.

The Stories API (**Stories**) facilitates both the posting and retrieval of stories within the application. This API endpoint handles both POST and GET requests. For POST requests, the API ensures that the user is authenticated before proceeding. Upon authentication, it parses the JSON payload containing the details of the story, creates a new **Story** object, and saves it to the database. On the other hand, for GET requests, the API filters stories based on provided parameters such as category, region, and date. It retrieves the relevant stories from the database, processes the data to format it appropriately, and returns the results as a JSON response to the client.

Lastly, the Delete API (**Delete**) is responsible for deleting stories from the system. This API endpoint only accepts DELETE requests. After verifying the user's authentication status, it retrieves the targeted story object using the provided key. The API then proceeds to delete the story from the database. Depending on the success or failure of this operation, the API returns an appropriate HTTP response to notify the client of the outcome, ensuring proper handling of the deletion process.

Clearly yet briefly explain how you implemented your APIs.

(Restrict this section to a maximum of one page)

# The Client

The client implementation provided is a Python script designed to interact with a news agency system via a RESTful API. Here's a concise explanation of how it works:

1. **Imports**: The script imports necessary libraries such as **requests** for making HTTP requests and **json** for handling JSON data. It also imports **tabulate** for displaying data in a tabular format.
2. **Session Management**: The script utilizes **requests.Session()** to manage a session for making HTTP requests. This allows for persisting parameters across requests, such as cookies for authentication.
3. **Authentication**: The **login()** function prompts the user for a username and password and sends a POST request to the login API endpoint with the provided credentials. If successful (status code 200), it sets the **authenticated** flag to **True**.
4. **Story Operations**: Functions are provided for posting, retrieving, and deleting stories. These functions interact with the respective API endpoints using appropriate HTTP methods (POST, GET, DELETE). The **post\_story()** function constructs a JSON object with story details and sends it as the payload of a POST request.
5. **Agency Operations**: Functions are provided for retrieving a list of agencies and fetching news from agencies. The **get\_agencies()** function sends a GET request to the agencies API endpoint and returns the JSON response. The **all\_news()** function iterates over agencies, retrieves news from each agency using **get\_stories()**, and displays the results.
6. **Command Line Interface (CLI)**: The **main()** function serves as the entry point. It presents a CLI interface where users can input commands and arguments. Depending on the command provided, it calls the corresponding function.
7. **Input Parsing**: The script parses user input to extract the command and its arguments. It handles different scenarios such as login, listing agencies, fetching news, posting stories, deleting stories, and logging out.
8. **Error Handling**: The script includes basic error handling to deal with exceptions that may occur during HTTP requests, JSON parsing, or invalid user input.
9. **Session Closure**: After the main loop exits, the script closes the session using **session.close()** to release any resources and terminate the session.

Overall, the client implementation provides a simple and intuitive way for users to interact with the news agency system, enabling actions such as authentication, story management, and fetching news from various agencies.