**University of Leeds School of Computing**

**COMP3011, 2023-2024**

**Web Services and Web Data**

A RESTful API for

News Aggregation

By

Robert Helps

Sc21rh – sc21rh@leeds.ac.uk

**Date:** 29/02/2024

# Introduction

In this coursework, we have made a successful News API with an integrated database, as well as with a command line client program that communicates with the API. My API was deployed to <https://sc21rh.pythonanywhere.com>. The superuser name is sc21rh and a password of monkey123. The client program was made using Python 3.12 and both the client and server were thoroughly tested to a high standard in line with the specifications given.

# The Database

In this Web Service, I decided to implement a simple Database using Django.db models. This database was named Story to best represent the stories that the database would hold. In the Database we have the fields of: key, headline, region, category, author, date and details. All these fields are set as CharFields in order to hold a string, with a suitable max\_length property. However, they key of this database was set as an AutoField as this was intended to be automatically filled when inserting into the database model. The key is also set as the primary\_key, which other models could use if we were to expand the database.

# The APIs

The API of this web service was made in a way to perform a set of commands that a client could use to access, post and delete news stories held on a centralised database as well as accessing news stories from other publishers. This could be seen to follow <https://www.newsalliance.org/>, which the European news agencies are members of. In this web services I have implemented a view for: login, logout, posting stories, viewing stories as well as deleting stories.

In the login view, we first prepare a HttpResponseBadRequest which is returned in the event the request method to the Django server is not a POST. In the case we have a POST request, we get the username and password from the application/x-www-form-urlencoded payload which is authenticated with Django’s authentication method. In the event we have a successful authentication, a welcome message with an OK status code is returned, else we return a failure message with an appropriate 503 status code.

In the logout view we once again prepare a bad request response in the case the request method is not POST. If we do receive a POST request, the user is logged out using Django’s logout method and we return a goodbye message with a 200 OK status code. This method requires the user to be logged in with the @login\_required tag.

The next view is when the server receives a GET or POST request to the URL /api/stories. In this URL, the server decides if we should post a story when we receive a POST request or if we should get a story from the database if we receive a GET request. In the event of a POST request the view post\_stories is used. We first try and load the Json body request and get the required data from the Json object received. With this data, we attempt to write to the database with the Json data and then save to the database. If an exception occurs, this is caught and an appropriate HttpResponse is returned, else we post the story successfully as inform the user.

If the /api/stories instead receives a GET request we instead use the get\_stories view. In this view, we try and get the data from the application/x-www-form-urlencoded payload. With this data we query the database suing a Q() object to construct a query. In the event one of the parameters is left empty, we instead query using all (\*). With the query to the database, we must check if the returned list is of size 0 indicating no stories meet the parameters given. If a story is returned, we prepare a JSON response and append the models’ fields to the JSON response. This is then returned to the user as a JsonResponse with an appropriate status code. If an exception occurs, we inform the user with an appropriate message and a 503-status code.

The final view is a delete function which takes a key at /api/stories URL. With this key we query the database and if the user is authenticated delete this story. In the case that the story could not be deleted, an appropriate message and status code 503 is returned to the user.

All these functions were tested using the POSTMAN application. For this we tested using the URL of the local web server with appropriate request methods and data to make sure the server worked as intended.

# The Client

The goal of the client program was to communicate with the API Django server which we created. It is a command like program which takes a command followed a potential list of parameters. Firstly, the program gets which command we have and decides which functionality it should trigger. We also have an exit command and a catch for all invalid commands.

The first of these functions is to login the user. This command also takes the URL for which we will login into. Upon entering the command, the user is prompted to enter the username and password and if successful are saved as global variables. With these details we send a Session request post to the URL given and check if we have a successful response indicated by status code 200. In the case of failure, we return the status code with a failed response message.

The next function is logout. Firstly, as with the rest of the methods below, we check if we have a session id indicating a user is logged in and authenticate this with the global session object. The client then sends a post to the URL and waits for a response which is successful or not. An appropriate message is returned to the user shortly after.

Another function implemented is a post story method. The user is prompted to enter a: headline, category, region and details. We create a Json data format which is posted to the URL. Once again, we get the response status code and return an appropriate message to the user.

Similarly to post, the news command takes a list of parameters as an input. The client program then decides which are present, else we give the parameter as all (\*). We then create the object for the server and wait for the response code. A 200-response code causes the client to get the Json object returned stories and prints all the fields in a clean manner on the command line. In the case of failure, an appropriate message is once again returned to the user.

The list command requests to the URL given by the documentation and prints the first 20 agencies which the client could connect to. With the Json object returned in a successful connection, we print the Agency Name, URL and Agency Code to the command line.

Finally, the delete command deletes a certain story in the server’s database by the key given in the delete command. We once again wait for a status code to be returned and print a response which is dependent on the code returned.

To test this client, we made sure to test all the commands with data that it should take as well as parameters which are not required by the specification. Overall, the client performed all the tests successfully and is sufficient to communicate with the News API.