Testing Documentation

Testing Process

Throughout the development of the API, one form of testing was completed to ensure the API endpoints were continually working. Before the API began development, four unit testing files were created for each of the endpoints, search, article, report and live. Tests were created in these files that mapped our sample API responses from design details. Additional tests were created to test the functionality of these endpoints as the design of the API changed and more functionality was added.

Whilst the team was not able to set up a testing environment due to time constraints, plans to create one were in the works. This environment was to be attached to the Github via Github Actions along with continuous integration and deployment. Once this was to be completed, the parser would automatically be tested against our created unit tests and then deployed onto AWS via the AWS command line tool, SAM.

Testing Scripts

Each of the unit tests were a Pytest script that would call endpoints of the API and match the response against expected data. We chose to usePytest to create the tests due to the team's previous knowledge using pytest in the past, meaning the group would be able to create and modify these scripts with ease compared to if another testing library was to be used. Additionally, Pytest allows for the creation of tests to be much simpler and easier through its additional features.

The Unittest library was also considered for testing as there were some advantages to it when compared to Pytest. It is part of the standard Python library, allowing us to easily test wherever, including on Github Actions. However, as the team had never used it before, the time constraints to create the testing files were too strict and the team's time would be much better spent on completing the API.

Within the scripts themselves, the following was tested. Each file would have a simple test to determine if a correct status code was returned if the API was called with data found in the database. If this test failed, then the corresponding endpoint was checked to determine if there was an error in the parser script, specifically either the sql statement passed into psycopg2 or if the data received via the request was incorrectly formatted or did not handle bad data.
All testing scripts excluding live also included a large test that would test variable data and determine if the parser was able to handle this data correctly. This includes providing null values in the request, incorrect data types and data that was not present within the database. The objective of this test was to determine if the input handling of the parsers were correct and it

would throw errors if it was received or if the query did not return data from the database. If any of these tests were incorrect, then the following actions would be taken to determine the cause of the error:

1. The database was checked to determine if the response would correctly return a query if it was manually inputted
2. The input handling sections of the parsers were examined and manually tested with the incorrect input to determine if the error was from this area
3. Finally, the sql statement was checked as sometimes the data would be incorrectly formatted when placed within the query string and an error would be thrown only when psycopg2 would attempt to search the database with this string

We were unable to test some areas of the API due to time constraints, this included testing time zones and some additional edge cases from which the API would return incorrect data. These areas will be tested as the functionality of the API increases and we start to integrate more testing phases, such as blackbox and whitebox testing when time permits.