SENG 3011 - Testing Report

“Weekly Cri Sesh”

Madeline Dobbie (z5112961), Nathan Driscoll (z5204935), Shabrina Yusri (z5205391), Sisil Harry Gunasekera (z5113599), Sumeet Charan (z5160224).

Testing processes

Our API was tested through Pytest unit tests, inspecting output to ensure they were expected, and automatic testing through a JavaScript script used in Postman.

Unit testing using Pytest

Unit testing involved testing that the API acts appropriately when given varying inputs. The API takes 4 JSON inputs: startDate, endDate, keywords, location. startDate and endDate are required whereas keywords and location are optional. Testing involved passing valid/invalid/well-formed/malformed versions of these inputs.

The tests are run using pytest. Pytest is a simple Python testing framework which is used to assist our testing. For the tests, each test is defined as a separate function with name beginning with test_. Each test calls the API with particular JSON inputs and asserts that the response given equals the response expected. Running the command “pytest” will execute every test case and give a clear indication of the results of all tests.

In the case that a test fails, pytest will alert us to this failure and point of the actual result compared to the expected result.

The test cases are:

- invalid dates: no dates
- invald dates: no endDate
- invald dates: no startDate
- invalid time in startDate, valid endDate
- invalid day in startDate, valid endDate
- invalid month in startDate, valid endDate
- invalid year in startDate, valid endDate
- invalid dates: startDate before endDate
- invalid dates: both dates in the future
- invalid startDate - not a date
- invalid startDate - doesn't have a time
- valid dates - successful API call
- valid dates, empty keywords - successful API call
- valid dates, empty location - successful API call
- valid dates, empty location and keywords - successful API call
- valid dates, and location, empty keywords - successful API call
- valid dates, and keywords, empty location - successful API call
- valid dates, another input which is invalid is ignored - successful API call

The unit tests are in unit_test.py inside the PHASE_1/TestScripts folder.

When in the PHASE_1/TestScripts folder, typing: “pytest” on the command line will run the tests. The output of this command is in the file: “pytest_output.txt”

<u>Testing Outputs</u>
Testing outputs involved calling the API and visually checking that it is producing the correct outputs.

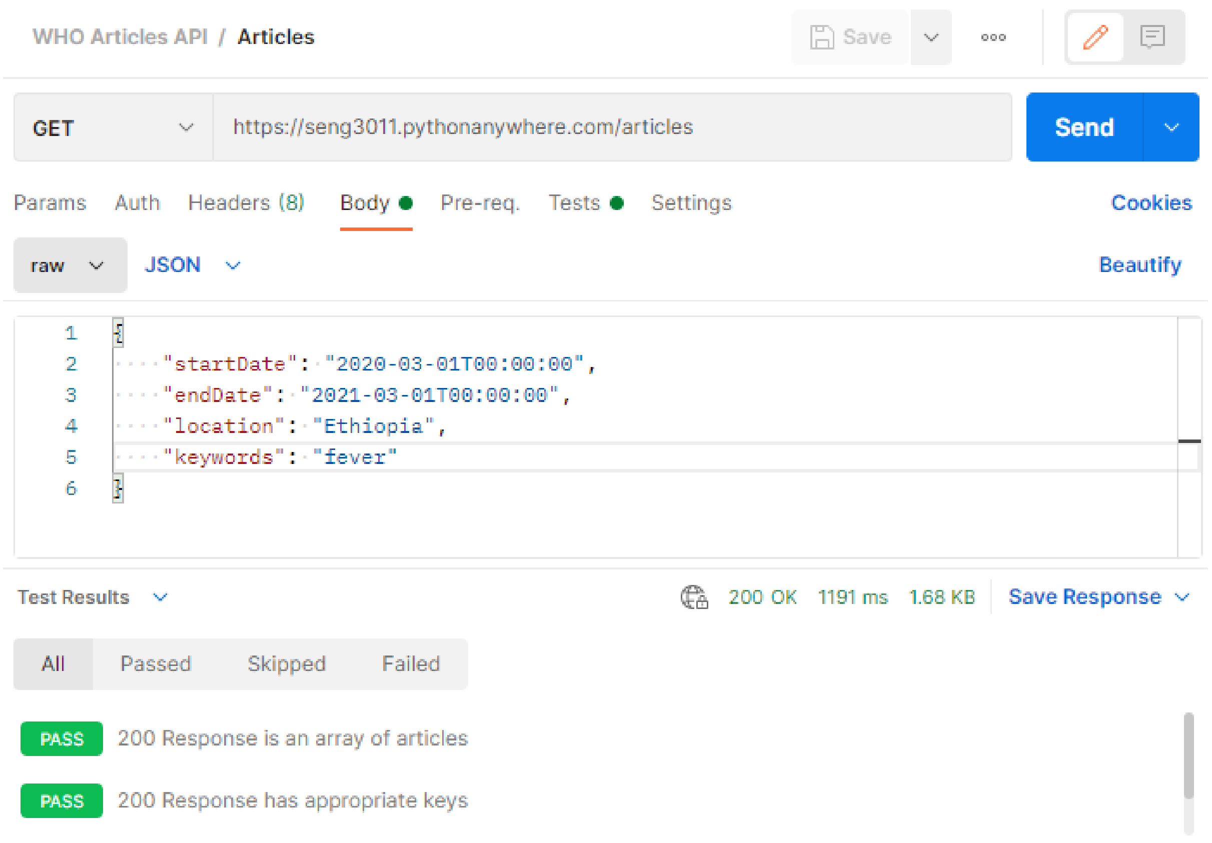
The generation of API calls is done in the file: “api_outputs.py” in the PHASE_1/TestScripts folder. This file generates some calls to the API and prints the results. Some calls involve invalid inputs which given appropriate error messages from the API and the other API calls are valid and JSON reports are returned. Valid calls involve using keywords involving symptoms and diseases as well as different locations such as China. The output of these API calls is in the file: api_output.txt

<u>Automated testing with JavaScript in Postman</u>
Postman, which hosts our API documentation and allows users to try our API through an easy to use interface, also provides the ability for every request to be tested, provided with a script written in JavaScript (JS) with the help of the *pm* library and Chai Assertion Library BDD syntax.

In this script, we test:

- That the response codes of all requests are either 200, 400 or 404 as we determined
- That responses are appropriately formatted according to what they are supposed to return
    - E.g. 200 responses return an array, appropriate keys are in the response
    - All 404 responses respond with the same message.

These tests run every time a request is made to our API using Postman, and the results are shown in a separate Testing tab, as shown below.



GET https://seng3011.pythonanywhere.com/articles

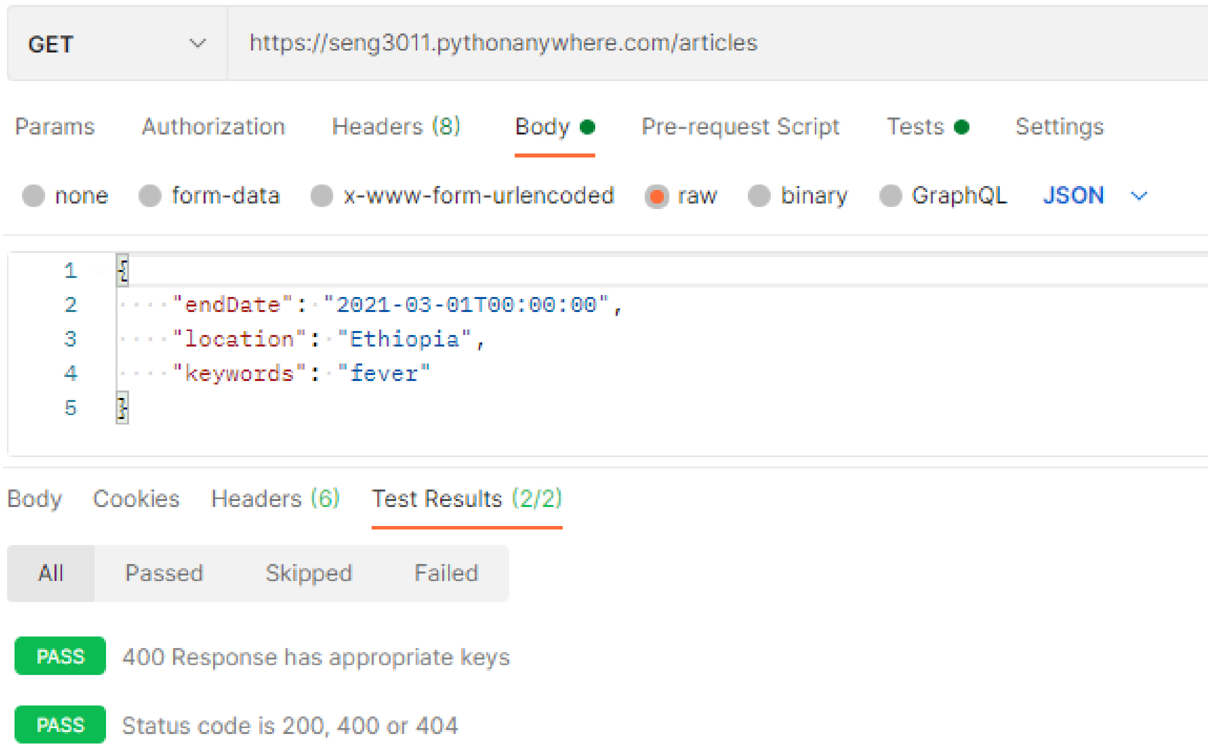
```
{
    "startDate": "2020-03-01T00:00:00",
    "endDate": "2021-03-01T00:00:00",
    "location": "Ethiopia",
    "keywords": "fever"
}
```

200 OK 1191 ms 1.68 KB

PASS 200 Response is an array of articles

PASS 200 Response has appropriate keys

Figure 1 - Tests that would run and pass during a request that returned 200



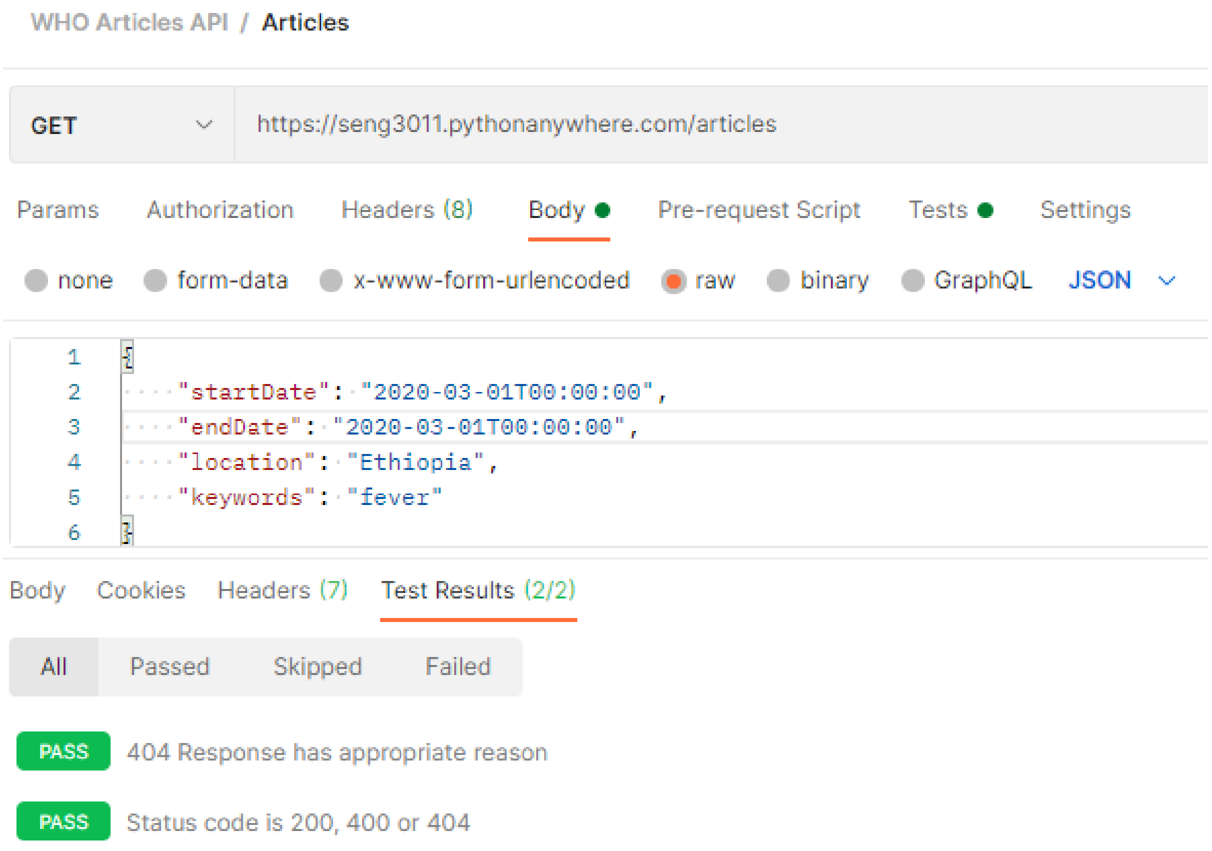
GET https://seng3011.pythonanywhere.com/articles

```
{
    "endDate": "2021-03-01T00:00:00",
    "location": "Ethiopia",
    "keywords": "fever"
}
```

PASS 400 Response has appropriate keys

PASS Status code is 200, 400 or 404

Figure 2 - Tests that would run and pass during a request that returned 400



https://seng3011.pythonanywhere.com/articles

```
{
    "startDate": "2020-03-01T00:00:00",
    "endDate": "2020-03-01T00:00:00",
    "location": "Ethiopia",
    "keywords": "fever"
}
```

PASS 404 Response has appropriate reason

PASS Status code is 200, 400 or 404

Figure 3 - Tests that would run and pass during a request that returned 404

Testing Feedback

Through this testing, bugs were found when incorrect inputs were given. The API would fail when given invalid dates, for example given a time with 70 seconds or a date with day 40 - error 500 would be returned. These issues were identified from unit testing and fixed. Now when given these inputs, the API responds with a reason why these calls failed (Start date in incorrect format.).

There was also a problem where if one of keywords and location was filled in but not the other, the API would return a 500 error. This issue was picked up during testing and fixed.

For the automated testing, the library was mostly optimised for JSON objects that started off with '{}'. Since our 200 responses were arrays of JSON objects, it was difficult to use this library to its full capability (i.e. making use of key-value pairs). However, this was not too much of an issue as our unit tests already covered the individual details, and this automated testing was more useful in providing a general overview of whether our API was returning the right format of responses.