International Journal of Scientific Research and Engineering Development-– Volume 4 Issue 3, May -June 2021

Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Test automation for chatbot middleware application

Priyangshu Pal*

*Inf0rmati0n Science, R.V. C0llege 0f Engineering, Bengaluru

Email: priyangshupal.is17@rvce.edu.in

----------**********----------

Abstract:

Testing is an indispensable phase of software development life cycle. It is the primary way through which quality of software is improved. Unit testing has a key role in developing high quality software and therefore is widely used. Unit testing means testing the smallest separate components in the system in isolation from each other. When a project is under development, errors may occur in any part of the software development life cycle. Thus, the importance of quality assurance/testing cannot be overlooked. Any nontrivial program contains some errors in the source code thusthere could be high chances that the final code has errors in functionality or design. For the identification of errors before their occurrence in production environment, it is essential to performing testing of software.

Keywords **—Unit testing, automation framework, middleware, API endpoint, code coverage, mocha, chai.**

----------**********----------

I. INTRODUCTION

In a survey conducted by Runeson, the results have found that the main reason for developers to include unit testing in their applications was external requirements. The unit testing framework could later be used as a technical specification. Unit testing helps scrutinize the system design and code changes. It has been proven that Test-Driven Development approach finds a greater number of errors than Test-After approach, but it is still not clear whether this is an effect of better code coverage by TDD or from unit testing itself.

A RESTful web service will provide information via an API over the network using HTTP, interacting with databases and other downstream APIs if required. Testing a RESTful API is difficult, as inputs/outputs trigger HTTP requests/responses to a remote server. So, many testing approaches resort to black box testing, as the tested API is a remote service whose code is not available. In this paper, we consider testing from the point of view of the developers, who have full access to the code that they are writing. Tests are judged based on code coverage and fault-finding capacity.

II. ARCHITECTURE

A. Test automation architecture

The testing process will not be carried out on the actual server which is live on public internet. Instead, a mock server with the exact same functionalities as the original one will be created, to which all the requests will be sent. The mock server will service requests just like the live server and will respond back with appropriate information. The actual testing architecture diagram has been shown in Fig 1.

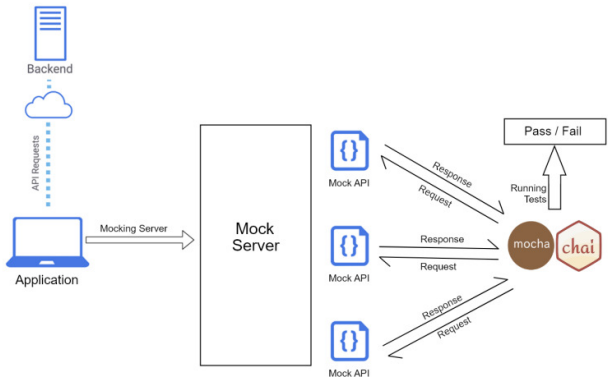

Fig. 1Test automation architecture

Any downstream APIs which the server requests information from also need to be mocked.

III. METHODOLOGY

The mock server files have to be loaded as soon as the test framework is run. These also include files for adding mock endpoints to the server. Once the mock server starts, it will have the mock API endpoints already loaded to which requests will be sent using Mocha and Chai. All in all, the steps followed for testing process are:

A. Server Mocking

Create a mock server imitating the real server for testing the endpoints (this is the server to which all our requests will be sent and be serviced from).

B. Endpoints Mocking

Determine the endpoints required to be tested and analyzethe flow of each of the handlers. Once the flow has been analyzed, mock the downstream APIs (if any).

C. Testing the application

Send requests to each of the endpoints and run tests on the responses from the server (the tests may report errors, response and status codes and ensure correct working of components/endpoints of chatbot).

Once the mocha and chai framework receive back the response from mock server, test cases are run on the response and they are categorized as pass/fail.

IV. RESULTS

A snapshot of the mock server loading all the endpoints and starting up is shown in Fig. 2.

Fig. 2Starting mock server

The mock server loads all the mock API endpoints' files. When they have been loaded, the mock server is started and requests are sent to it using mocha and chai framework.

After all the tests have been run a code coverage report is generated using *nyc* reporter as shown in Fig. 3.

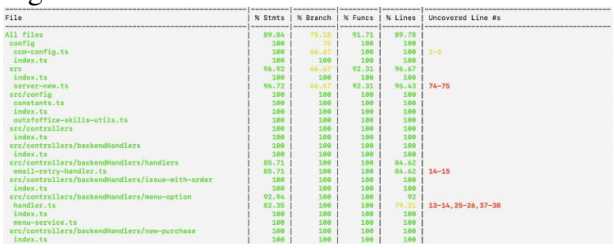

Fig. 3Code coverage report

Since a lot of requests are sent to the test server while testing, it also generates a lot of log messages. So, the tests have been organized into a lot cleaner format using mochawesome reporter which helps the developer scan through all the test cases and filter out any test cases in just a jiffy. A view of mochawesome reporter report is shown in Fig. 4.

Fig. 4Report generated by mochawesome reporter

V. CONCLUSIONS AND FUTURE WORK

The project was carried out to develop a test automation framework for an existing chatbot application. While developing the application it was

found that unit testing improves manageability (skills of planning and execution of a particular project) in terms of visibility, reporting, control, correction, efficiency, speed, planning and customer satisfaction. Also, unit testing is very important for learning to develop complex software products. There could be lesser chances of production bug occurrence.

Some changes can still be incorporated to the project to increase its efficacy:

- Automating the user input*:* The messages being sent from user end can be automated to obviate manual end to end testing. It will save a lot of time and energy.
- Adding tests for future bugs:As the application grows, newer bugs will come to light. The tests for those bugs need to be added too.

Being able to test live servers:Removing mock servers will help us run a more comprehensive check on the server running and also let us deal with any runtime issues in QA/Production environment.

REFERENCES

[1] J. -W. Lin, N. Salehnamadi and S. Malek, "Test Automation in Open-Source Android Apps: A Large-Scale Empirical Study," 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2020, pp. 1078-1089.

[2] Fraser G., Rojas J.M. (2019) Software Testing. In: Cha S., Taylor R., Kang K. (eds) Handbook of Software Engineering. Springer, Cham. doi: 10.1007/978-3-030-00262-6_4.

[3] P. Runeson, "A survey of unit testing practices," in IEEE Software, vol. 23, no. 4, pp. 22-29, July-Aug. 2006, doi: 10.1109/MS.2006.91.

[4] Maximilien, E.M. & Williams, L. (2003). Assessing test-driven development at IBM. Proceedings - International Conference on Software Engineering. Vol. 6. 564- 569. 10.1109/ICSE.2003.1201238.

[5] A. Arcuri, "RESTful API Automated Test Case Generation," 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), 2017, pp. 9-20, doi: 10.1109/QRS.2017.11.

