API Validation Testing Using Karate

Shashwati Jha[1], Srinivasula Katigari[2], S.B. Prapulla[3] and Ramakanth Kumar[4]

[1]Student, Department of Computer Science, RV College of Engineering, India

[2]Applications Tech Lead, LBrands Mast Global, India

[3]Assistant Professor, Department of Computer Science, RV College of Engineering, India

[4] Professor & HOD, Department of Computer Science, RV College of Engineering, Bangalore, India

Email: [1]shashwatijha.cs17@rvce.edu.in

***Abstract* —** Functional Automation is a widely adopted domain in the evergreen field of testing and is poised to grow at an average growth rate of more than 15% from 2020 to 2027. As testing is the most time and cost intensive phase of a project, there is an active effort to develop a framework which can reduce associated costs and increase time efficiency of both, development, and deployment. Karate is one such open source automation framework developed in 2017 which can be used for API testing, performance testing and UI automation. It combines the advantages of a neutral language that's easy to understand by even a non-developer with powerful assertions and inbuilt multithreading. In this paper, we detail the development of a Karate framework for API testing and analyze its performance in sequential and parallel execution and compare and contrast Karate with the popular Cucumber BDD (Behavior Driven Development) framework.

***Keywords*—** API (Application Programming Interface), Automation Testing, Cucumber, Feature File, Functional Validation, Karate, Scenario.

I. INTRODUCTION

In an Agile Project Development Environment, testing of the project at hand to validate its functionality, performance and stability is a necessary albeit time consuming task prone to Budgetary and Time constraints as well as human operator error. Successive development cycles necessitate execution of similar testcases and performing validations on common APIs (Application Programming Interface). Using a test automation framework along with DevOps Tools, it's possible to Script a test suite and execute it at regular timeframes.

With a growing adoption of microservice architecture, the development and use of API's are more commonplace. API's or Application Programming Interfaces serve as an interface between a backend data store such as a database or cloud and a front-end software where data is requested and manipulated by an end-user. To assure system reliability and functionality, we need to ensure an API works as per requirement and does not send invalid or corrupted data to the user.

In this paper, we detail the development of an API testing framework using Karate Domain Specific Language (DSL). The organization of this paper is as follows; In Section 3, relevant information about the framework is provided for clearer understanding. In section 4, an overview of API automation testing and Karate DSL is supplied; Section 5 provides system architecture followed by section 6 which describes the functioning of each module in system architecture. Section 7 details the methodology of the framework; In Section 8, the results and analysis of framework is detailed and Finally, Section 9 carries the conclusion.

The following section provides a brief description of previous work in the domain of automation testing.

II. STATE OF ART

In the paper [1] it is conveyed that while manual testing had helped to refine the requirements as per user stories, one aspect it didn't perform well in was regression testing as Manual testing via a graphical user interface also missed these bugs as it was conducted for newer features and testing regressively was neglected. Here, the process of automating user stories was also documented, and a cost vs effort estimation was made for automation.

Automation testing is adopted by industries as testing is a cost and resource intensive process. In The research article [2] the authors discuss proposed software's time to market, by measuring cost of project in terms of the COCOMO model (Constructive Cost Model) cost in terms of effort per person/month and as a majority of software's cost lies in testing, automation of such tests can give good return in the long term. Ramler et al in the paper [3] documented the benefits of automation testing. Similar to paper [2], automation and manual testing were compared to reduce costs and a cost-based model was discussed to reduce overall costs of development.

These Proposed improvements over time have persuaded organizations to further employ automation. To enable automation, Various tools exist with most popular one being Cucumber Framework.

In the article [4], the Authors discuss a prosed solution of automation using Cucumber and Junit, in back end

automation which explores the viability of Cucumber framework as an automation tool. As The project aimed to use Karate which has the feature of multithreading to speed up execution, in the research [5] the author details a method to execute A BDD framework in parallel by using selenium grid, which allows flexibility in choosing tests to execute, and reducing batch processing time. It utilizes the Cucumber framework to make test cases readable by even a domain expert who may not be technically skilled.

In the paper [6] the authors have discussed a way to implement performance tests on a microservice, where the aim was to develop a microservice which be integrated without difficulty making minor changes to application code.

In [7], the author discusses the various tools available for automation testing such as Selenium, Test Complete etc. thus providing a good overview of the advantages of each tool. Finally, Test automation has made significant process over time from using record and play techniques to now using tools such as Cucumber and Junit that support developing automation scripts at code level, the paper [8] highlights these improvements and also discusses a technique to make automation testing intelligent via the use of AI to simulate rules and create a knowledge base of an agent.

Automation testing has wide scope and reduces costs while significantly increasing performance. The studied papers reflect the innovations to this domain and provide an insight into various techniques to carry out automation. However, no work has been done to carry out a review of Automation processes using Karate DSL, a newer tool which can reduce development time while still maintaining powerful assertion capabilities of java.

III. RELEVANT INFORMATION

Cucumber: Cucumber is a Behavior Driven Development (BDD) tool which utilizes natural language syntax to create test scripts that are easy for a product owner or business analyst to understand. Scripts are created in an easy given, when, then syntax which follows a natural language. However, for most step definitions, Back end code must be supplied in Java to automate these steps.

Karate: Karate is bult on top of the cucumber framework and this has the language neutrality syntax of cucumber, however it does not require any step definitions to be written in java unlike cucumber and has ease of compatibility with third party software. Which makes using this framework to do API validation tests simple, powerful, and flexible.

Maven: Maven is often used from the command line and it serves to make the building process easier by shielding developers from knowing about underlying mechanism. It also provides runtime information about the build, thus increasing quality by creating change logs, creating reports, and installing dependencies. Dependencies are specified in a pom.xml file which is built before execution.

API (Application Programming Interface): API's are software tools that are used to pass messages between two applications and to interact with these systems.

API status code: Every API upon execution returns a Status code. Status codes in the range 200-299 depict a successful request. Status codes in 400-499 and 500-599 represent client and server-side errors respectively.

IV. OVERVIEW OF API TESTING USING KARATE

APIs or Endpoints serve as a message transfer link between back-end and front-end systems. When using APIs over a server, we need to ensure that the API is functioning as per contract.

This entails checking an APIs return status which must be 200 for success and between the range of 400-500 for Failure. Also, we need to ensure that the response we obtain from the API in a positive scenario must always match a contract.

A contract is a binding schema of the API that must always be followed. For example, if we specify that in a response body with fields Name, ID, and Date, that the name and ID must be in string format, and the Date must be within a specified range and that the ID is a mandatory field.

Then each response must contain an ID field that has a string format, and whenever name and Date are passed, they too must be passed in the specified format.

Organizations need to regularly check the performance and functionality of APIs managed by them.

Using Manual testing, it would be cumbersome to verify and validate multiple endpoints with each having complex multi-lined responses. Here, automation testing serves as the best approach to testing.

Karate is a modern framework that has the capability to assert complex schema matching and perform Endpoint testing in a short span of time using multiple threads.

Karate has a simple-to-use syntax that allows developers to write test cases faster.

This speedup coupled with the speedup in execution due to parallelism makes Karate a lightweight framework to deploy cases over a cloud regularly.

V. SYSTEM DESIGN

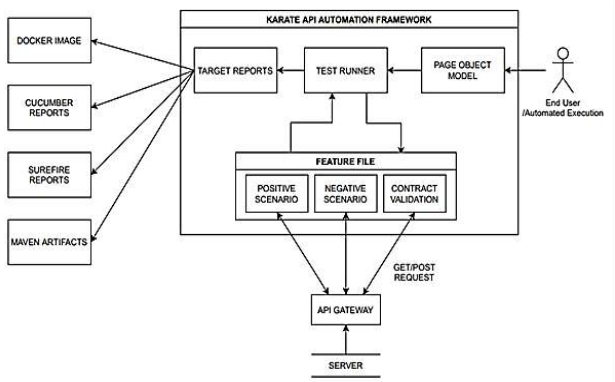

Fig.1: System Design of Karate framework

Figure 1 illustrates the system design of a Karate Framework for API Test Automation; the following section describes the functions of each module.

VI. MODULE DESCRIPTION

```
@Timezone
Scenario: Verify Functionality of API
  Given url 'http://worldtimeapi.org/api/timezone/Asia/Kolkata'
  When method get
  Then assert responseStatus == 200
```

Fig.2: Sample code snippet for API Validation using Karate syntax

1. *End User/ Automated Execution*: An end user or a pipeline starts executing the program by specifying tags and number of threads. Tags depict the function of the scenario. These tags can later be used to selectively order a scenario or feature file for execution.

2. *Page Object Model:* The page Object model file, which contains all the dependencies required by the program, installs these dependencies, and equips local system for execution.

3. *Test runner*: Test Runner is a java file that is akin to a main file. It is used to specify execution order using Hooks. It calls the feature file or group of feature files as specified by tags, and once execution has returned from feature files, it calls the report generator using a '@after' hook which generates the reports into the target folder.

4. *Feature Files*: Feature files consist of multiple scenarios. These are used to combine those scenarios of positive, negative and contract validation which call a common API. These also contain some background steps which are executed before each scenario.

5. *Scenarios:* Scenarios contain steps in Given, when, then format as shown in Fig. 2. These are of positive, negative and contract valuation format.

6. *API gateway:* API gateway serves as an interface between a client store and the front-end system at hand. It transfers requests to and from a data store or a server.

7. *Target Reports:* When Execution has completed, Reports are generated by test runner which are stored in target. Reports are created in various formats, the most common ones being Docker Images, Cucumber reports and Surefire reports. Docker Images contain code, dependencies etc. and are used to make an executable image, Cucumber reports and Surefire reports contain documentation of each step.

8. *Docker Images*: Docker creates an image of the entire project and is useful for execution over integrated systems and cloud systems.

9. Additionally, a Maven jar file is created on execution of jar file which is used to reduce code duplication.

VII. METHODOLOGY

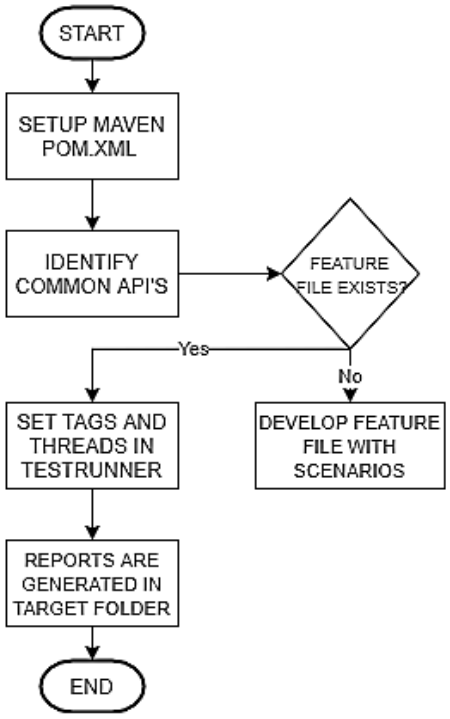

Fig.3 Flow of Development and Execution of Karate Framework

Initially framework dependencies are identified and a Maven POM.xml (Page Object Model) is set up. This contains all dependencies used by the system. Upon execution the POM.xml file will execute first and will install any dependencies and configurations required by the program.

Post Setup of the Karate framework the APIs were identified and chosen. Similar endpoints which have a common access pattern form an API collection and one such collection was chosen for the project.

For Each API within the collection a feature file was created. Each feature file should contain positive validation scenarios, negative validation scenarios and a schema validation scenario as well as background information.

The background information will contain common information that is executed before each scenario. For the positive validation scenarios, the application will match a 200-status code when the correct configurations of path and query parameters and payload are sent. In the negative validation scenario, the scenario will match a status code in the range of 400 to 500 on passing wrong or empty parameters.

However, if a 200-status code is returned, the program must check if the response payload is empty. This will also constitute a negative validation case. In the schema validation case, the program matches the obtained responses schema against the required schema known as contract. For better documentation and understanding, both scenario and feature files are named according to their purpose.

Also, tags can be given to each scenario and feature file which specify the scenarios function and make execution easier. Tag '@Timezone' has been shown in figure 2, the tag describes the scenario which is carrying out positive validation for the world time API.

Finally, upon development of feature files a TestRunner.java file is created This file serves as a central point of execution. It is in this file that the user can specify which feature file or scenario must be run or not run by using tags. A user can also specify a collection of feature files to be run.

For example, if the user wishes to not execute any scenario marked with '@ignore', they must specify '~@ignore' in the TestRunner.java. Tags follow logical operators hence when tags are supplied in the following pattern:

@KarateOptions(tags = {"@Test1,@test2","@Positive","~@ignore"})

Follows the logical sequence of:
((@Test1 OR @Test2) AND @Positive AND NOT @ignore)

- "@Test1,@Test2": Any combination of tags within double quotes ("*")run in OR configuration. That means any scenario with either tag @Test1 or @Test2 will be executed.

- "@Test1,@Test2","@Positive":Any combina-tion of tags separated by double quotes and a comma(,) run in AND configuration. Here, any scenario marked with @Test1 OR @Test2 AND @positive will be run.

- "~@ignore": Any scenario having associated ~(NOT) Tag will Not be executed.

Further, Parallelism is inbuilt into Karate and the number of tags can be specified ranging from 1 which is sequential execution to multiple.

A limit on the number of threads will be user's system processor and ability to simultaneously multithread. Other software and hardware factors will also limit the degree of parallelism.

Upon execution, Target Reports are built in Cucumber and Surefire formats. These reports detail each step-in execution, the time taken to execute each step and if any step has passed, failed or been skipped.

They also show the response that was received after execution of an API.

VIII. RESULTS AND ANALYSIS

Table. I: Tabular data of Time vs Feature files

No. of feature files	Time (sec) to Complete Execution			
	Threads = 1	**Threads = 2**	**Threads = 4**	**Threads = 6**
1	10.97	6.8	5.49	6.54
2	14.42	8.83	6.67	7.01
3	18.83	11.09	8.76	7.64
5	30.36	18.25	11.76	10.58
10	58.12	31.78	18.38	14.57
20	103.75	52.1	29.6	20.65
40	199.26	100.11	53.29	36.68

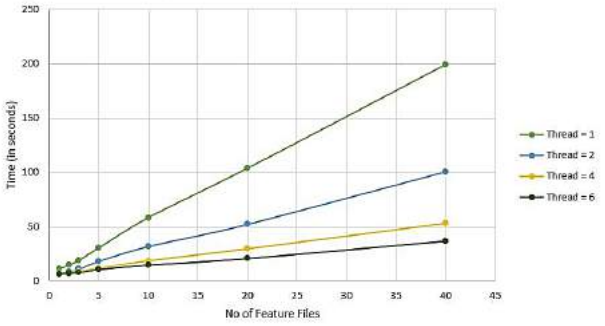

Chart.1: Linear graph of No. of Feature Files vs. Time(sec)

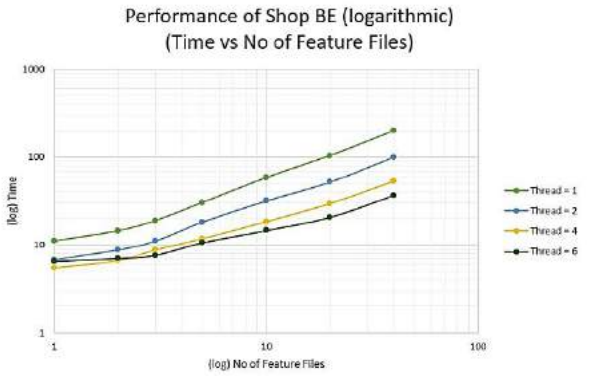

Chart.2: Log_{10} Graph of No. of Feature Files vs. Time(sec)

The Framework was run with varying number of Feature files and Threads. The following section contains the data used to execute and its time performance.

Chart.1 describes a graph plotted with data from Table.1. It is Observed that utilizing 2 threads decreases the run time by half of using 1 thread. It is also observed that using 6 threads on a 4-core system (intel i5) resulted in a 5-factor speedup.

Chart.2 Describes the same data but with logarithmic scale. Here. Difference in execution of 1 feature file can be visualized. Running 1 feature file with 6 threads took more time that running 1 feature file with 4 threads. This is due to added latency by 2 extra threads on a 4-core system.

Some notable advantages of Karate framework exist over Cucumber Framework; these have been detailed below:

- *Parallelism:* Karate supported inbuilt parallelism whereas Cucumber framework requires 3rd party tools to support threading. Overall, Karate is a flexible and lightweight tool to implement faster development

- *Step Definition*: Karate requires step definitions only in neutral language whereas Cucumber supports neutral language but requires corresponding steps to be written in Java. This reduces code complexity and development time.

- *Assertions:* To match schemas, Karate has powerful inbuilt assertion which can match complex schemas. In cucumber, assertions have to be done through parsing '*.yaml' and '*.json' files.

IX. CONCLUSIONS

The primary motive for automation testing is to reduce time in and increase software output. Both these goals were achieved as well as increasing software quality and reliability by automating tests that evaluate these services.

In Conclusion, Karate DSL demonstrated good performance when analyzed with varying number of test cases. The reports Karate generated contain execution details of each step as well as the overall percentage of test cases that have passed vs failed which helps in gauging a quick idea of how many services are up and running. it also had the advantage of being easy to use which allowed us to rapidly automate and test each endpoints functionality and Schema and hence decreases both, test production time as well as test deployment time.

REFERENCES

[1] C. Klammer and R. Ramler, "A Journey from Manual Testing to Automated Test Generation in an Industry Project," 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2017, pp. 591-592, doi: 10.1109/QRS-C.2017.108.

[2] Divya Kumar, K.K. Mishra, "The Impacts of Test Automation on Software's Cost, Quality and Time to Market", Procedia Computer Science, Volume 79,2016,Pages 8-15,ISSN 1877-0509,https://doi.org/10.1016/j.procs.2016.03.003.

[3] Rudolf Ramler and Klaus Wolfmaier. 2006. "Economic perspectives in test automation: balancing automated and manual testing with opportunity cost". In Proceedings of the 2006 international workshop on Automation of software test (AST '06). Association for Computing Machinery, New York, NY, USA, 85–91. DOI: https://doi.org/10.1145/1138929.1138946

[4] Mr Tarik Sheth , Ms. Priyanka Bugade , Ms. Sneha, Pokharkar, Analysis Of Code Coverage Through Gui Test Automation And Back End Test Automation, IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 3 Issue 3, March 2016. ISSN 2348 – 7968

[5] R. Anand, ArulPrakash Ma, Business driven automation testing framework, March 2018, International Journal of Engineering & Technology 7(2.8):345, DOI: 10.14419/ijet.v7i2.8.10438

[6] André de Camargo, Ivan Salvadori,Ronaldo dos Santos Mello, Frank Siqueira, An architecture to automate performance tests on microservices, iiWAS '16: Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services November 2016 Pages 422–429 https://doi.org/10.1145/-3011141.3011179

[7] Neha Bhateja, A Study on Various Software Automation Testing Tools, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 5, Issue 6, June 2015, ISSN: 2277 128X

[8] Li, J...J., Ulrich, A., Bai, X. et al. Advances in test automation for software with special focus on artificial intelligence and machine learning. Software Qual J 28, 245–248 (2020). https://doi.org/10.1007/s11219-019-09472-3

[9] Contan, C. Dehelean and L. Miclea, "Test automation pyramid from theory to practice," 2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR),2018,pp.1-5,doi: 10.1109/AQTR.2018-.8402699.

[10] Z. Sun, Y. Zhang and Y. Yan, "A Web Testing Platform Based on Hybrid Automated Testing Framework," 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), 2019, pp. 689-692, doi: 10.1109/IAEAC47372.2019.8997684.

[11] M. Iyama, H. Kirinuki, H. Tanno and T. Kurabayashi, "Automatically Generating Test Scripts for GUI Testing," 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2018, pp. 146-150, doi: 10.1109/ICSTW.2018.00043.

[12] V. H. Kiranagi and G. K. Shyam, "Feature Driven Hybrid Test Automation Framework (FDHTAF) for web based or cloud based application testing," 2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon), 2017, pp. 1555-1559, doi: 10.1109/SmartTechCon.2017.8358626.

[13] K. Sneha and G. M. Malle, "Research on software testing techniques and software automation testing tools," 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), 2017, pp. 77-81, doi: 10.1109/ICECDS.2017.8389562