International Journal of Scientific & Engineering Research, "Volume 14 issue 3, March 2023 Edition - IJSER"
ISSN 2229-5518

Micro Service Testing with Rest Assured Technology

Asit Adhikari, Tech Mahindra Limited, Kolkata

Abstract— With the increase popularity of RESTful services, there is a need for fast and lightweight tool for REST web services testing automation. One of the most popular choices is Rest-Assured framework. It introduces simplicity of testing web services from dynamic languages like groovy or ruby to java. REST Assured supports the familiar Given/When/Then syntax from behavior-driven development (BDD), resulting in a test that is easy to read and takes care of everything (setup, execution, and verification) with just a single line of code.

1 INTRODUCTION

- Micro service is a software development technique and it is a variant of SOA(service oriented architecture).
- This application is collection of loosely coupled services.
- It is breakdown of a single service into chucks.
- The benefit of decomposing an application into different smaller services is that it improves modularity.
- Due to micro-services the application is easier to understand, test, and develop.
- Parallel development of a single service is possible as it is divided into smaller services, and each service an then be developed, tested individually by small autonomous teams.
- Micro services-based architectures enable continuous delivery and deployment.

2 API

API or "Application Programming Interface" is an interface between two software systems, which provides interaction and data sharing. The basic principle of API is to send requests to a server and receive a response, indicating to the sender whether the request was successful or not. For example, if there are endpoints which are responsible for a store's shopping cart and the ability to populate and empty it, a request to add to the composition of the basket is sent, which in turn returns either a positive or a negative response – whether the request was successful or not. Likewise, it is possible to make a request in which the contents of the basket are removed from it, therefore, the user has performed the basic function of the shopping cart within seconds and checked its ability to work without going through the UI [1].

2.1 When is API Testing Useful?

One of the API's advantages is that testing can start even before the creation of a UI, allowing for early testing, as the tester does not have to wait until the programmer creates UI elements for each functionality. API tests both manually and x automated run much faster because the tester physically does not need to go through the UI, repost test data. All it takes is a single, or multiple, API calls to be performed to successfully test functionality.[1]

2.2 API call methods

API call methods define what type of a request will be sent. Whether for it to be a new addition of data to a platform, or an update to already existing data. Just like with response codes, there's a multitude of call methods, with the most popular being:

- **GET method** in API tests is probably the most used, because it obtains information from the corresponding endpoint without requiring input data. This method can receive just a response code, or a response code and response body, containing data that was asked to be returned.
- **POST method** is used to send new information to the server, or to update already existing information, by entering the requested information in the request body before sending it. Most often, this method is used to create new entries in an existing database
- **PUT method** is used only to update existing data and returns a positive result only if the system contains already existing and updatable data
- **PATCH method** is similar to PUT, in that it is used for updating existing data, but differs from it in so that to use it, it does not necessarily indicate that the user has to input all data of the database record. It is enough to specify the desired data which needs to be updated
- **DELETE method** is self-explanatory. It's used to delete existing data from the system. This option is non-reversible.[1]

3. TYPES OF TESTING

3.1 Contract Testing

Contract testing is integral to micro services testing, and it can be of two types and the right method can be decided based on the end purpose that the micro service would cater to and how the interface with the consumer would be defined.

- **Integration contract testing:** Testing is carried out using a test double (mock or stub) that replicates a service that is to be consumed. The testing with the test double is documented and this set needs to be periodically verified with the real service to ensure that there are no changes to the service that is exposed by the provider.

- **Consumer-driven contract testing**: In this case, consumers define the way in which they would consume the service via consumer contracts that can be in a mutually agreed schema and language. Here, the provider of the service is entrusted with copies of the individual contracts from all the consumers. The provider can then test the service against these contracts to ensure that there is no confusion in the expectations in case changes are made to the service.

3.2 End to End Testing

It is usually advised that the top layer of testing be a minimal set, since a failure is not expected at this point. Locating a point of failure from an end-to-end testing of a micro services architecture can be very difficult and expensive to debug.

4. Approaches of Automation Testing

4.1 Rest Template

The Rest Template is the basic Spring class for simultaneous client-side HTTP access. It simplifies the interaction with HTTP servers and enforces RESTful systems. It is much related to the JdbcTemplate, JmsTemplate and the various other templates.

Rest Template library was not selected because we need to write additional code for handily the Security Certificate for API.

4.2 Rest Assured

Rest Assured is a Java based DSL (Domain Specific Language) which is most commonly used to test out REST based services. Rest-Assured presents a great advantage because it supports multiple HTTP requests and can validate and/or verify the responses of these requests. These multiple requests include GET, POST, PUT, DELETE methods. it supports these various requests, it also allows for them to be constructed with multiple parameters, headers, and their respective body, as well as validating response's status code, headers, cookies and response time, we will be using JSON based requests and responses (application/Json). Not only is it easy to use and get started with but it's also built to scale to more advanced use cases using detailed configuration, filters, specifications and the like. There are many Java libraries that allow us to write a REST client. It is also possible to use a simple HTTP client library and manually (de)serialize JSON data from/to the server. We could use a client library like Jersey or the Spring REST Template to write REST unit tests. After all, it is very logical to use the same Java library both on the client and on the server side for the sake of simplicity. However, REST Assured has an additional testing DSL on top of its REST client that follows the BDD paradigm, making tests very readable. Behavior-driven development (BDD) is about minimizing the feedback loop. It is a logical step forward in the evolution of the practice of software development. This article explains the concept and its origins. [3]

5. Advantages of REST Assured

5.1 Easy HTTP Request Buinding and Execution

Requesting building comprises of defining many things like query params, cookies, headers, path params and request body. Using its DSL, REST-assured hides away the complexity of building and executing an HTTP request behind a fluent interface. We will see this in action in the next section.

5.2 Response Assertion

REST-assured also handles, within its library, the parsing of responses. It provides many constructs for making assertions on cookies, response headers and response body. For doing assertions on response body it provides JSONPath for assertions on JSON responses and XmlPath for assertions on XML responses. It also uses Java Hamcrest Matchersto make readable assertions.

5.3 Ability to write clean code

The real value of automated tests is realized when they are easy to understand and takes very less effort to maintain them. In the dictionary of REST-assured's DSL, you can also find constructs for writing in Given-When-Then style. Using this style helps in specifying pre-conditions under the Given section, behavior under test under the When section and verifications under the Then section. This helps in maintaining a clear separation of concerns within a test, thus leading to a very readable test. [2]

6. Framework for END-TO-END Testing

In this Framework we have testing end to end seniors with multiple API, where output from one API is given as input to other.

6.1 Maven Framework

Maven framework was used as it lets you get your package dependencies easily, helps reusing the code also provides scalability (lower level of additional info/code to add a new step to the build process).

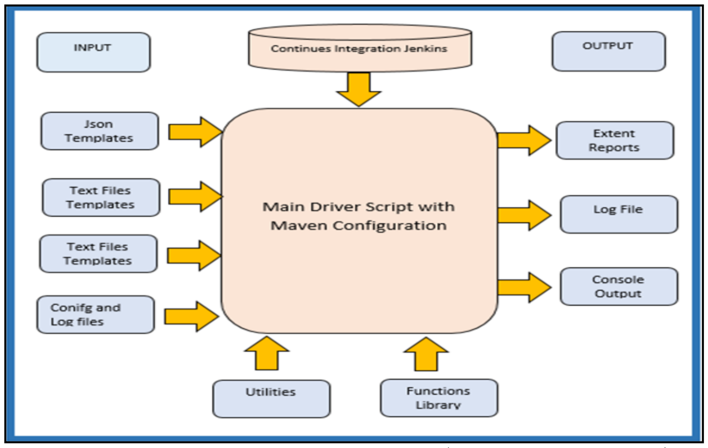
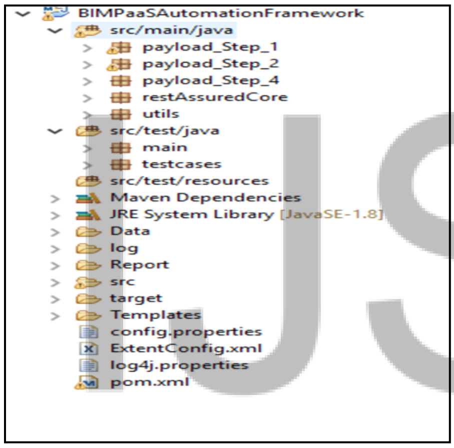


Fig: BIMPaaS Automation Structure (End to End Testing)



```
BIMPaaSAutomationFramework
  src/main/java
    payload_Step_1
    payload_Step_2
    payload_Step_4
    restAssuredCore
    utils
  src/test/java
    main
    testcases
  src/test/resources
  Maven Dependencies
  JRE System Library [JavaSE-1.8]
  Data
  log
  Report
  src
  target
  Templates
  config.properties
  ExtentConfig.xml
  log4j.properties
  pom.xml
```

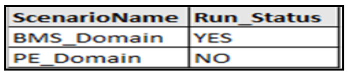
Fig: Folder Structure for End to End testing

6.2 Framework Details

- **main Package**: Main package consists of main class. Main class helps to read the excel file as well as the properties file and call the method in other packages according to the test cases and test data
- **testcase Package**: This package consists of different steps and the call of that steps into a single class.
  - **testCaseCreation Class**: This class is created to call the different functions from TestSteps class
  - **testSteps Class**: This class consists of different functions for various scenarios. This class calls different functions from all the other packages to implement the logic for testing the API.
- **restAssuredCore Package**: This package consist of a RESTCALLS class. That is written to call different services of REST ASSURED methods such as Get, Post, Put, and Delete. This class performs operations using json file as well as json string.
- **Payload_Step_1 package**: This package consists of different classes. This package is used to generate the desired json file that will be given as input to an API, with the help of the output of some other API. This package creates a Model for extracting the useful data from the response of another API.
- **Payload_Step_2 package**: This package consists of different classes. This package is used to generate the desired json file that will be given as input to an API, with the help of the output of some other API. This package creates a Model for extracting the useful data from the response of another API.
- **Payload_Step_4 package**: This package consists of different classes. This package is used to generate the desired json file that will be given as input to an API, with the help of the output of some other API. This package creates a Model for extracting the useful data from the response of another API.
- **utils package**: This package consists of different class that consists multiple functions for multiple different operations and logic.
  - **URL Class**: This class helps to create the URL of API, which will be given as input to the RESTCALLS class. This class combines the input from config.properties file and string and concates it and returns a complete URL.
  - **JsonCreation Class**: This class is used to create the Json file with the help of data provided to it and the template file. Also this class consists of a function that will user to convert a string format to json format.
  - **CommonValidations Class**: This class consists of multiple functions for performing different validations on the response of different API.
  - **ValidateDMN class**: This class is used to validate the API using different test cases. With different data provided to this class.
  - **StausCodeCheck class**: This class helps to check the status code of the response.

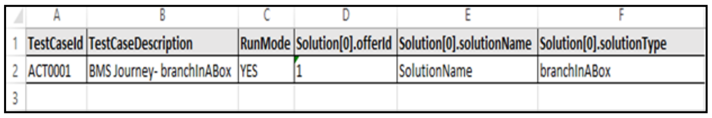
6.3 Input data

Main Input Data-Sheet: -



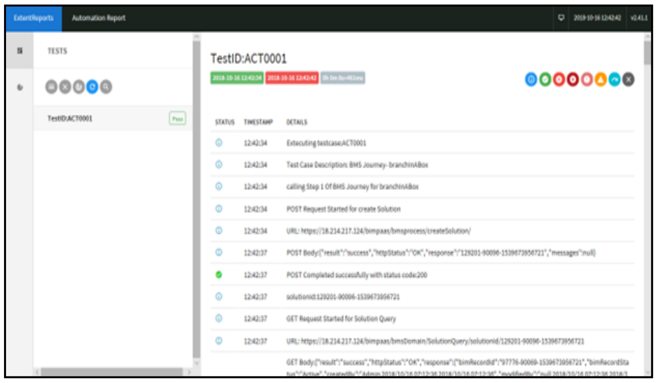
ScenarioName	Run_Status
BMS_Domain	YES
PE_Domain	NO

Input Data-Sheet: -



	A	B	C	D	E	F
1	TestCaseId	TestCaseDescription	RunMode	Solution[0].offerId	Solution[0].solutionName	Solution[0].solutionType
2	ACT0001	BMS Journey- branchInABox	YES	1	SolutionName	branchInABox
3						

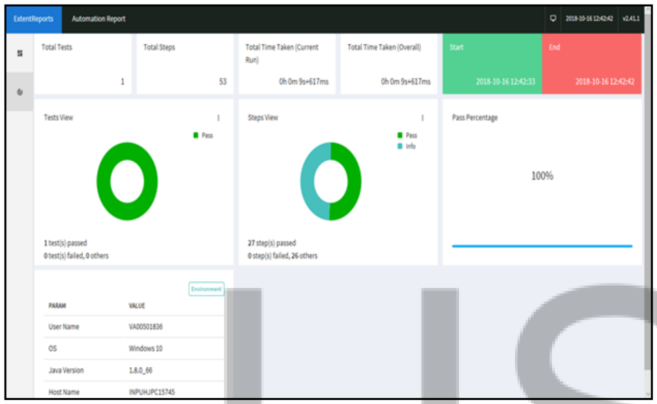
6.4 Extent Reports



TestID:ACT0001

2018-10-16 12:42:34 2018-10-16 12:42:42

STATUS	TIMESTAMP	DETAILS
ⓘ	12:42:34	Executing testcase:ACT0001
ⓘ	12:42:34	Test Case Description: BMS Journey- branchInABox
ⓘ	12:42:34	calling Step 1 Of BMS Journey for branchInABox
ⓘ	12:42:34	POST Request Started for create Solution
ⓘ	12:42:34	URL: https://18.214.217.124/bimpaas/bmsprocess/createSolution/
ⓘ	12:42:37	POST Body:{"result":"success","httpStatus":"OK","response":"129201-90096-1539673956721","messages":null}
✓	12:42:37	POST Completed successfully with status code:200
ⓘ	12:42:37	solutionid:129201-90096-1539673956721
ⓘ	12:42:37	GET Request Started for Solution Query
ⓘ	12:42:37	URL: https://18.214.217.124/bimpaas/bmsDomain/SolutionQuery/solutionid/129201-90096-1539673956721
		GET Body:{"result":"success","httpStatus":"OK","response":{"bimRecordId":"97776-90069-1539673956721","bimRecordStatus":"Active","createdBy":"Admin 2018/10/16 07:12:36 2018/10/16 07:12:36","modifiedBy":"null 2018/10/16 07:12:36 2018/1



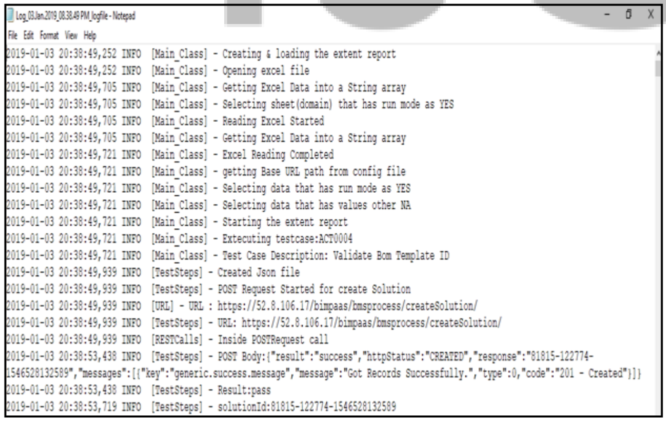
Total Tests	Total Steps	Total Time Taken (Current Run)	Total Time Taken (Overall)	Start	End
1	53	0h 0m 9s+617ms	0h 0m 9s+617ms	2018-10-16 12:42:33	2018-10-16 12:42:42

1 test(s) passed
0 test(s) failed, 0 others

27 step(s) passed
0 step(s) failed, 26 others

PARAM	VALUE
User Name	VA00501836
OS	Windows 10
Java Version	1.8.0_66
Host Name	INPUHJPC15745

6.5 Logs


Log_03Jan2019_08.38.49 PM_logfile - Notepad
File Edit Format View Help

```
2019-01-03 20:38:49,252 INFO  [Main_Class] - Creating & loading the extent report
2019-01-03 20:38:49,252 INFO  [Main_Class] - Opening excel file
2019-01-03 20:38:49,705 INFO  [Main_Class] - Getting Excel Data into a String array
2019-01-03 20:38:49,705 INFO  [Main_Class] - Selecting sheet(domain) that has run mode as YES
2019-01-03 20:38:49,705 INFO  [Main_Class] - Reading Excel Started
2019-01-03 20:38:49,705 INFO  [Main_Class] - Getting Excel Data into a String array
2019-01-03 20:38:49,721 INFO  [Main_Class] - Excel Reading Completed
2019-01-03 20:38:49,721 INFO  [Main_Class] - getting Base URL path from config file
2019-01-03 20:38:49,721 INFO  [Main_Class] - Selecting data that has run mode as YES
2019-01-03 20:38:49,721 INFO  [Main_Class] - Selecting data that has values other NA
2019-01-03 20:38:49,721 INFO  [Main_Class] - Starting the extent report
2019-01-03 20:38:49,721 INFO  [Main_Class] - Extecuting testcase:ACT0004
2019-01-03 20:38:49,721 INFO  [Main_Class] - Test Case Description: Validate Bom Template ID
2019-01-03 20:38:49,939 INFO  [TestSteps] - Created Json file
2019-01-03 20:38:49,939 INFO  [TestSteps] - POST Request Started for create Solution
2019-01-03 20:38:49,939 INFO  [URL] - URL : https://52.8.106.17/bimpaas/bmsprocess/createSolution/
2019-01-03 20:38:49,939 INFO  [TestSteps] - URL: https://52.8.106.17/bimpaas/bmsprocess/createSolution/
2019-01-03 20:38:49,939 INFO  [RESTCalls] - Inside POSTRequest call
2019-01-03 20:38:53,438 INFO  [TestSteps] - POST Body:{"result":"success","httpStatus":"CREATED","response":"81815-122774-
1546528132589","messages":[{"key":"generic.success.message","message":"Got Records Successfully.","type":0,"code":"201 - Created"}]}
2019-01-03 20:38:53,438 INFO  [TestSteps] - Result:pass
2019-01-03 20:38:53,719 INFO  [TestSteps] - solutionId:81815-122774-1546528132589
```

7 Frameworks for Contract-Testing

In this framework we have tested each API independently.

7.1 Maven Framework

Maven framework was used as it lets you get your package dependencies easily, helps reusing the code also provides scalability (lower level of additional info/code to add a new step to the build process).

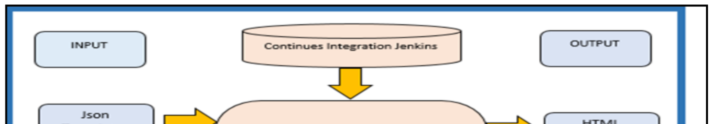


Fig: BIMPaaS Automation Structure (Contract Testing)

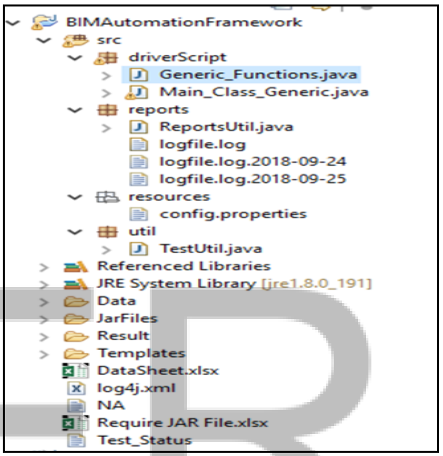


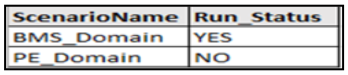
Fig: Folder Structure for Contract testing

7.2 Framework Details

- **driverScript package**: This package consists of different class that consists multiple functions for multiple different operations and logic.
  - **main Class**: Main class helps to read the excel file as well as the properties file and call the method in other packages according to the test cases
  - **generic_fucntions Class**: This class consist of different functions to perform different operations such as Get, Post, Put, Delete with the help of Rest-Assured. Also this class consists of functions that helps to validate the output of the services. This is a function to create the Json file with the help of data provided to it and the template file.
- **Reports package**: This package consists of single class that helps us to create customized HTML Repots with the help of other class.
- **Utils package**: This package consist of single class that is used to set the current date and time of the system to the HTML Report
- **Resources package**: This package consists of config.properties file that is used to set different paths for templates, repots, log file, name of datasheet etc

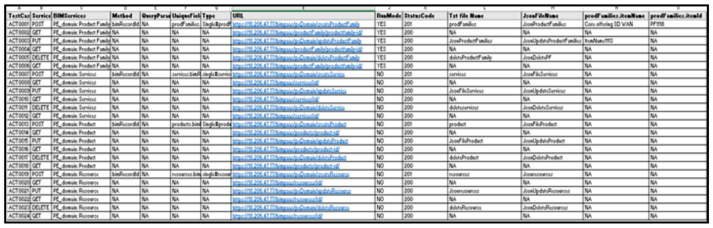
7.3 Input data

Main Input Data-Sheet: -



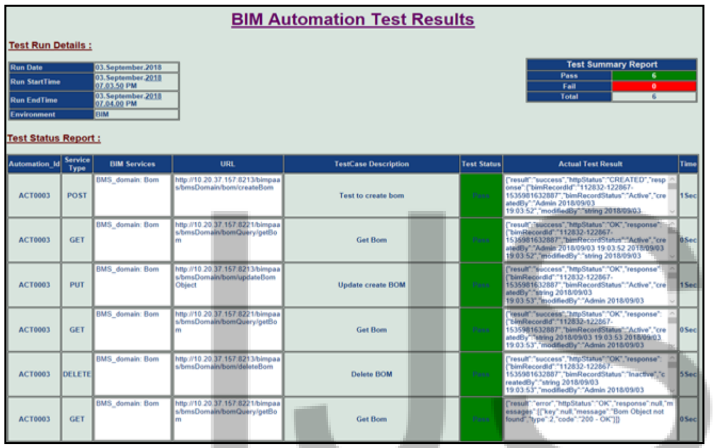
ScenarioName	Run_Status
BMS_Domain	YES
PE_Domain	NO

Input Data-Sheet: -



TestCas	Service	BIMServices	Method	QueryParai	UniqueFiel	Type	URL	RunMode	StatusCode	Txt file Name	JsonFileName	prodFamilies.itemName	prodFamilies.itemId
ACT0001	POST	PE_domain: Product Family	bimRecordId	NA	prodFamilies.	Single#prodF	http://10.206.47.77/bimpaas/peDomain/createProductFamily	YES	201	prodFamilies	JsonProductFamilies	Core offering SD WAN	PF1118
ACT0002	GET	PE_domain: Product Family	NA	NA	NA	NA	http://10.206.47.77/bimpaas/productFamily/productfamily-id/	YES	200	NA	NA	NA	NA
ACT0003	PUT	PE_domain: Product Family	NA	NA	NA	NA	http://10.206.47.77/bimpaas/peDomain/updateProductFamily	YES	200	JsonProductFamilies	JsonUpdateProductFamilies	ItemName1113	NA
ACT0004	GET	PE_domain: Product Family	NA	NA	NA	NA	http://10.206.47.77/bimpaas/productFamily/productfamily-id/	YES	200	NA	NA	NA	NA
ACT0005	DELETE	PE_domain: Product Family	NA	NA	NA	NA	http://10.206.47.77/bimpaas/peDomain/deleteProductFamily	YES	200	deleteProductFamily	JsonDeletePF	NA	NA
ACT0006	GET	PE_domain: Product Family	NA	NA	NA	NA	http://10.206.47.77/bimpaas/productFamily/productfamily-id/	YES	200	NA	NA	NA	NA
ACT0007	POST	PE_domain: Services	bimRecordId	NA	services.bimR	single#servic	http://10.206.47.77/bimpaas/peDomain/createService	NO	201	services	JsonFileServices	NA	NA
ACT0008	GET	PE_domain: Services	NA	NA	NA	NA	http://10.206.47.77/bimpaas/services/id/	NO	200	NA	NA	NA	NA
ACT0009	PUT	PE_domain: Services	NA	NA	NA	NA	http://10.206.47.77/bimpaas/peDomain/updateService	NO	200	JsonFileServices	JsonUpdateServices	NA	NA
ACT0010	GET	PE_domain: Services	NA	NA	NA	NA	http://10.206.47.77/bimpaas/services/id/	NO	200	NA	NA	NA	NA
ACT0011	DELETE	PE_domain: Services	NA	NA	NA	NA	http://10.206.47.77/bimpaas/peDomain/deleteService	NO	200	deleteservices	JsonDeleteServices	NA	NA
ACT0012	GET	PE_domain: Services	NA	NA	NA	NA	http://10.206.47.77/bimpaas/services/id/	NO	200	NA	NA	NA	NA
ACT0013	POST	PE_domain: Product	bimRecordId	NA	products.bim	Single#produ	http://10.206.47.77/bimpaas/peDomain/createProduct	NO	201	product	JsonFileProduct	NA	NA
ACT0014	GET	PE_domain: Product	NA	NA	NA	NA	http://10.206.47.77/bimpaas/products/product-id/	NO	200	NA	NA	NA	NA
ACT0015	PUT	PE_domain: Product	NA	NA	NA	NA	http://10.206.47.77/bimpaas/peDomain/updateProduct	NO	200	JsonFileProduct	JsonUpdateProduct	NA	NA
ACT0016	GET	PE_domain: Product	NA	NA	NA	NA	http://10.206.47.77/bimpaas/products/product-id/	NO	200	NA	NA	NA	NA
ACT0017	DELETE	PE_domain: Product	NA	NA	NA	NA	http://10.206.47.77/bimpaas/peDomain/deleteProduct	NO	200	deleteProduct	JsonDeleteProduct	NA	NA
ACT0018	GET	PE_domain: Product	NA	NA	NA	NA	http://10.206.47.77/bimpaas/products/product-id/	NO	200	NA	NA	NA	NA
ACT0019	POST	PE_domain: Resource	bimRecordId	NA	resources.bim	single#resour	http://10.206.47.77/bimpaas/peDomain/createResource	NO	201	resources	Jsonresources	NA	NA
ACT0020	GET	PE_domain: Resource	NA	NA	NA	NA	http://10.206.47.77/bimpaas/resources/id/	NO	200	NA	NA	NA	NA
ACT0021	PUT	PE_domain: Resource	NA	NA	NA	NA	http://10.206.47.77/bimpaas/peDomain/updateResource	NO	200	Jsonresources	JsonUpdateResources	NA	NA
ACT0022	GET	PE_domain: Resource	NA	NA	NA	NA	http://10.206.47.77/bimpaas/resources/id/	NO	200	NA	NA	NA	NA
ACT0023	DELETE	PE_domain: Resource	NA	NA	NA	NA	http://10.206.47.77/bimpaas/peDomain/deleteResource	NO	200	deleteResource	JsonDeleteResources	NA	NA
ACT0024	GET	PE_domain: Resource	NA	NA	NA	NA	http://10.206.47.77/bimpaas/resources/id/	NO	200	NA	NA	NA	NA

7.4 Custom HTML Reports



BIM Automation Test Results

Test Run Details :

Run Date	03.September.2018
Run StartTime	03.September.2018 07.03.50 PM
Run EndTime	03.September.2018 07.04.00 PM
Environment	BIM

Test Summary Report	
Pass	6
Fail	0
Total	6

Test Status Report :

Automation_Id	Service Type	BIM Services	URL	TestCase Description	Test Status	Actual Test Result	Time
ACT0003	POST	BMS_domain: Bom	http://10.20.37.157:8213/bimpaas/bmsDomain/bom/createBom	Test to create bom	Pass	{"result":"success","httpStatus":"CREATED","response":{"bimRecordId":"112832-122867-1535981632887","bimRecordStatus":"Active","createdBy":"Admin 2018/09/03 19:03:52","modifiedBy":"string 2018/09/03	1Sec
ACT0003	GET	BMS_domain: Bom	http://10.20.37.157:8221/bimpaas/bmsDomain/bomQuery/getBom	Get Bom	Pass	{"result":"success","httpStatus":"OK","response":{"bimRecordId":"112832-122867-1535981632887","bimRecordStatus":"Active","createdBy":"Admin 2018/09/03 19:03:52 2018/09/03 19:03:52","modifiedBy":"string 2018/09/03	0Sec
ACT0003	PUT	BMS_domain: Bom	http://10.20.37.157:8213/bimpaas/bmsDomain/bom/updateBom Object	Update create BOM	Pass	{"result":"success","httpStatus":"OK","response":{"bimRecordId":"112832-122867-1535981632887","bimRecordStatus":"Active","createdBy":"string 2018/09/03 19:03:53","modifiedBy":"Admin 2018/09/03	1Sec
ACT0003	GET	BMS_domain: Bom	http://10.20.37.157:8221/bimpaas/bmsDomain/bomQuery/getBom	Get Bom	Pass	{"result":"success","httpStatus":"OK","response":{"bimRecordId":"112832-122867-1535981632887","bimRecordStatus":"Active","createdBy":"string 2018/09/03 19:03:53 2018/09/03 19:03:53","modifiedBy":"Admin 2018/09/03	0Sec
ACT0003	DELETE	BMS_domain: Bom	http://10.20.37.157:8213/bimpaas/bmsDomain/bom/deleteBom	Delete BOM	Pass	{"result":"success","httpStatus":"OK","response":{"bimRecordId":"112832-122867-1535981632887","bimRecordStatus":"Inactive","createdBy":"string 2018/09/03 19:03:53","modifiedBy":"Admin 2018/09/03	5Sec
ACT0003	GET	BMS_domain: Bom	http://10.20.37.157:8221/bimpaas/bmsDomain/bomQuery/getBom	Get Bom	Pass	{"result":"error","httpStatus":"OK","response":null,"messages":[{"key":null,"message":"Bom Object not found","type":2,"code":"200 - OK"}]}	0Sec

8 Acknowledgments

The author wishes to thank below dignitaries:
1. Dr. Satish Pai – COO (Communication, Media & Entertainment) – Americas, TechMahindra
2. Mr. Subhadeep Chakraborty – Head of Pre Sales, Global Testing at Tech Mahindra.
3. Mr. Dnyanesh Belitkar – Head of Delivery, Global Testing at Tech Mahindra

9 References

1. An Introduction to Testing API's Using REST Assured

(https://www.testdevlab.com/blog/an-introduction-to-testing-apis-using-rest-assured)

2. Testing REST APIs with REST-assured

(https://praveer09.github.io/technology/2016/02/06/testing-rest-apis-with-rest-assured)

2. Testing REST Endpoints Using REST Assured

(https://semaphoreci.com/community/tutorials/testing-rest-endpoints-using-rest-assured)