RestAssured Reusable Test Automation Framework Design

|  |  |
| --- | --- |
| Version No. | 1.0 |
| Date of Release | 15-Mar-2018 |

Change History

|  |  |  |
| --- | --- | --- |
| Release Date | Created by | Description |
| 15-Mar-2018 | Gaurav Kumar | Framework Design Document |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |
| --- | --- |
| **Canada Office** | **Global Development Centre** |
| 510 – 900 Howe Street  Vancouver, BC, Canada V6Z 2M4  +1 604-736-4600 | 7th Floor, Pinnacle Tower, A-42/6, Block A, Sector 62  Noida, Uttar Pradesh, India 201301  +91 120-6496446 |

**Contents**

[Overview](#_30j0zll)

[Knowledge Required to start with this framework](#_bwv97k15jspn)

[Components of this framework](#_gesh08mgu9yf)

[Structure and Control Flow](#_qmouc8lh7oez)

[Storing and Accessing the Test Data](#_5ovki216qkdp)

[LifeCycle of a Test Case](#_2phdg6ute7cw)

[How to incorporate this framework in your project](#_2ca6exvn2cea)

# 

# 

# Overview

This document describes the design and implementation of reusable API Test Automation Framework created using RestAssured. Using this document one can instantly use this reusable framework to automate the testing of RESTful Web Services.

[GitHub Link to the reusable Framework](https://github.com/gauravk-optimus/RestAssuredReusableTAF.git) //link to be updated after creation of private repository

This document describes the following:

1. Knowledge required to start using this framework
2. Components of Framework
3. Structure and Control Flow
4. Storing and Accessing the test data
5. LifeCycle of a Test Case
6. How to incorporate in any project

# Knowledge Required to start with this framework

Heart of this framework is undoubtedly RestAssured library. One can learn to implement RestAssured from this [link](http://toolsqa.com/api-automation-testing-tutorial/). Apart from knowledge of RestAssured API, to start creating test scripts using this framework one should have good knowledge of TestNG, Maven, & Jackson library. One should also have prior knowledge to create POJO classes for your json requests. Some other libraries are also used which you can learn at the time of implementation.

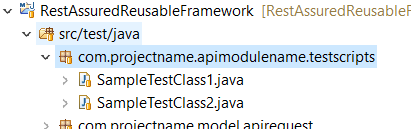
# **Components** of this framework

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Component/Library** | | **Version** |  | **Usage in this framework** |  |
|  | rest-assured | 3.0.6 |  | Headless Client to send customized HTTP Rest API Requests to the Restful server |  |
|  | Java | 1.8.0\_161 |  | JVM platform execution environment |  |
|  | testng | 6.13.1 |  | Test framework |  |
|  | json | 20090211 |  | Used to encode/decode json request and response of the APIs under test |  |
|  | jackson-annotations | 2.9.3 |  | @JsonProperty' annotation for jackson Data Processor is used to map the strings with request fields in pojo model clases |  |
|  | allure-report | 2.4.1 |  | Html reporting for test execution results |  |
|  | allure-testng | 2.0-BETA21 |  | Used to trigger allure report generation while executing the test scripts using testNG |  |
|  | allure-maven | 2.9 |  | Used to trigger allure report generation while running the project using maven |  |
|  | maven-compiler-plugin | 3.7.0 |  | To compile the sources of this framework |  |
|  | log4j | 1.2.17 |  | Logging execution steps, success, failures and exceptions etc. |  |

# Structure and Control Flow

The design is based on loose coupling between test data, test scripts and the API endpoint resources. A brief description of the various components of the TAF is given below. The foundation of this framework is kept using Maven structure along with arrangement of four sections *ApiModules, ‘model’, ‘service’, ‘utils’ & ‘config.properties’* in manageable and scalable manner as follows:

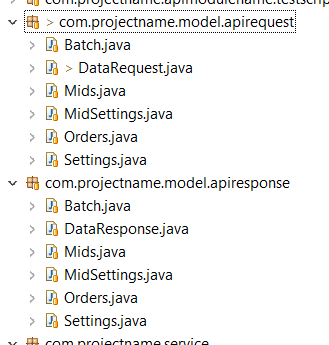
* **apiModule -** ApiModule is a placeholder for actual API modules of the project under test. According the design of this framework, separate packages can be created for various API modules. All the classes containing test scripts of a module are placed in respective packages. Naming convention of test script package is somewhat like ‘*com.projectname.apimodulename.testscripts’.*



Multiple packages can exists with respective module names. In above image we can see two sample tests classes in a package. Test classes are created using TestNG annotations. While implementing this framework in your project rename ‘apimodulename’ to respective module name.

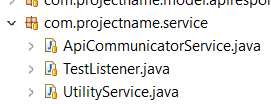
* **model -** This section contains model classes for request and responses. This section comprised mainly of two packages:

*‘com.projectname.model.apirequest’* and *‘com.projectname.model.apiresponse’*



These model classes are POJO (Plain Old Java Object) classes for request and response. Strings variables are mapped to request response fields by ‘@JsonProperty’ using ‘jakson-annotation’ library. While performing assertions, values of desired fields from request and response can be accessed using getters of these model classes.

* **service -** This section contains three classes *‘ApiCommunicatorService’*, *‘Test Listener,* and *‘Utility Service’*.

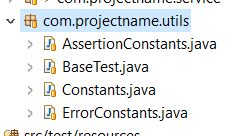


‘ApiCommunicatorService’ class contains various methods to fetch the response for ‘*post’, ‘get’, ‘delete’* and *‘put’* type custom requests. All the test cases are consuming the HTTP methods from this class using singleton pattern.

‘Test Listener’ class contains triggers the actions to be performed on pass, fail and skipped test cases.

‘Utility Service’ class contains various methods to communicate with json request files, properties files, and most important, methods to communicate request response model classes.

* **utils -** This section contains three classes *‘AssertionConstants’, ‘BaseTest’,* ‘*Constants’* and *‘ErrorConstants’*.



‘AssertionConstants’ class contains all the reusable messages which we use in our test cases for assertion failures.

‘BaseTest’ serves three purpose. First, it calls the listeners after test completions. Second, it creates the common base URL of API server for all the test classes. Last, it initialize the property configurator.

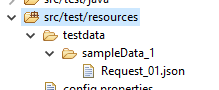
‘Constants’ class contains all the path constants for properties reader and those constants which are used to assert the response, in the form of maps and strings.

‘ErrorConstants’ class contains all the reusable error messages which are actually returned by the API host server in multiple API responses.

* **config.properties -** This file holds the configuration of host urls, ports and api endpoints. Other than this, paths to the test data folders containing custom json requests.

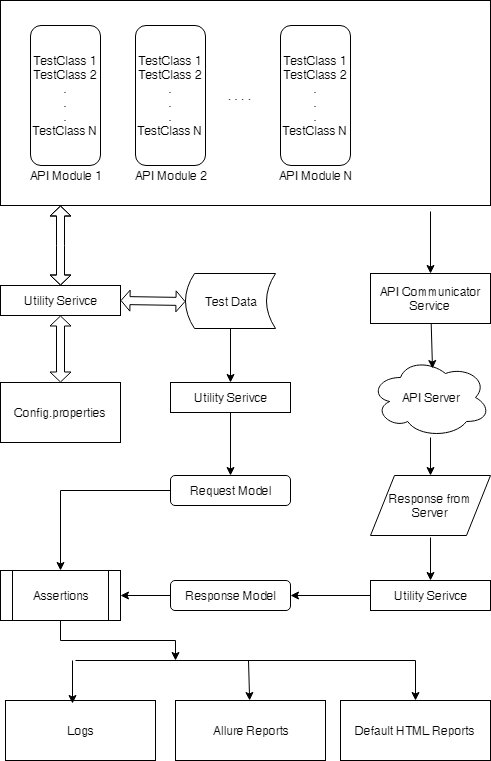
# Storing and Accessing the Test Data

All the Test Cases are consuming request test data from the ‘resources’ folder as JVM can directly access Maven’s resources directory, ‘src\test\resources’. Further subdivision are created based on the number of api modules.



‘testdata’ folder contains parallel sub folder as per the api modules with relevant identifiers. Each folder carries request/test data in json format.

# LifeCycle of a Test Case



# How to incorporate this framework in your project

One can start the test automation of his APIs in few minutes. Just follow these simple steps:

1. Update the property ‘hostUrl’ and ‘port’. (If there is not port, leave the field blank) in ‘config.properties’ at ‘src\test\resources\config.properties’
2. Create a folder for your api module at ‘src\test\resources\testdata\’ and place the json request file inside it.
3. Create POJO classes for your requests and response and place them inside ‘com.projectname.model.apirequest’ & ‘com.projectname.model.apiresponse’ packages. Don’t forget to rename ‘projectname’ to relevant name for all the packages.
4. Create a test class similar to the ‘SampleTestClass1’ class inside the ‘com.projectname.apimodulename.testscripts’. Here, change the ‘apimodulename’ to relevant name. Update the endpoint URL property key in the constructor. Create test cases similar to sample which I have created.
5. Add assertions as mentioned in the sample test cases.
6. Update the testNG file.
7. Run the test cases using Maven or TestNG as per the need of your project. Preferably Maven.
8. See the test execution results at ‘allure-results’ folder in the base directory. That's all!

Note: If you are new to create POJO classes, then for a quick start you can go to ‘http://pojo.sodhanalibrary.com/' URL to create them online. But best way is to create them manually.