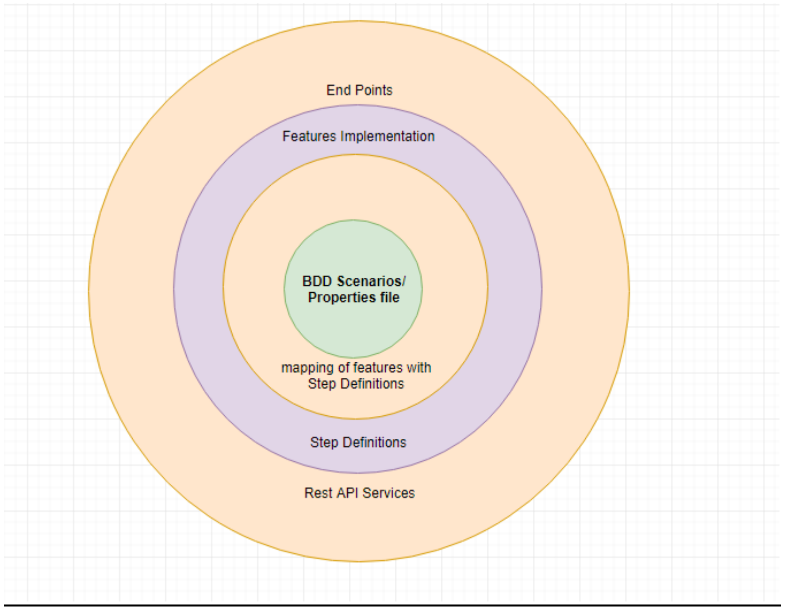
REST API ARCHITECTURE

Designed By
Rajaneesh Balakrishnan
Test Automation Engineer
Sydney,Australia

Table of Contents

Architecture

This is a behavioural Driven Development Architecture and hence the core component of the architecture is the scenarios of the features. All the other components are layered on top of it which can be removed or modified easily based on the requirements. The Architecture can be depicted as below diagram.


End Points
Features Implementation
BDD Scenarios/ Properties file
mapping of features with Step Definitions
Step Definitions
Rest API Services

Tools /Technologies Used

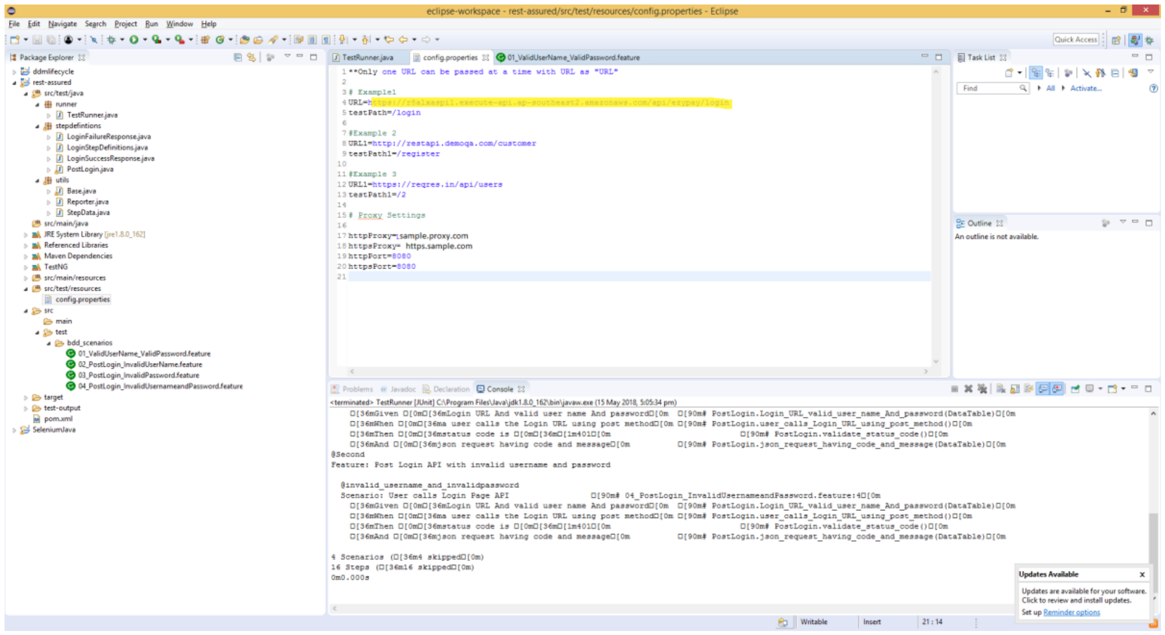
1. Eclipse Oxygen
2. Maven
3. Rest Assured API
4. Post Man
5. Cucumber
6. Java
7. Junit
8. Jenkins

Dependencies – POM.XML

```
<dependencies>
        <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>3.8.1</version>
                <scope>test</scope>
        </dependency>

        <!-- https://mvnrepository.com/artifact/io.rest-assured/rest-assured -->
        <dependency>
                <groupId>io.rest-assured</groupId>
                <artifactId>rest-assured</artifactId>
                <version>3.0.2</version>
                <scope>test</scope>
        </dependency>

        <!-- https://mvnrepository.com/artifact/io.rest-assured/json-path -->
        <dependency>
                <groupId>io.rest-assured</groupId>
                <artifactId>json-path</artifactId>
                <version>3.0.2</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient -->
        <dependency>
                <groupId>org.apache.httpcomponents</groupId>
                <artifactId>httpclient</artifactId>
                <version>4.5.3</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.testng/testng -->
        <dependency>
                <groupId>org.testng</groupId>
                <artifactId>testng</artifactId>
                <version>6.11</version>
                <scope>test</scope>
        </dependency>

        <!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->
        <dependency>
                <groupId>com.google.code.gson</groupId>
                <artifactId>gson</artifactId>
                <version>2.8.1</version>
        </dependency>


</dependencies>
```

How to Use the framework?

1. Setting up the Endpoint
   The below highlighted End pint should be configured as the URL to be tested.



eclipse-workspace - rest-assured/src/test/resources/config.properties - Eclipse

```
**Only one URL can be passed at a time with URL as "URL"

# Example1
URL=https://r8alxaspi1.execute-api.ap-southeast2.amazonaws.com/api/ezypay/login
testPath=/login

#Example 2
URL1=http://restapi.demoqa.com/customer
testPath1=/register

#Example 3
URL1=https://reqres.in/api/users
testPath1=/2

# Proxy Settings

httpProxy=;sample.proxy.com
httpsProxy= https.sample.com
httpPort=8080
httpsPort=8080
```

```
<terminated> TestRunner [JUnit] C:\Program Files\Java\jdk1.8.0_162\bin\javaw.exe (15 May 2018, 5:05:34 pm)
    □[36mGiven □[0m□[36mLogin URL And valid user name And password□[0m  □[90m# PostLogin.Login_URL_valid_user_name_And_password(DataTable)□[0m
    □[36mWhen □[0m□[36ma user calls the Login URL using post method□[0m □[90m# PostLogin.user_calls_Login_URL_using_post_method()□[0m
    □[36mThen □[0m□[36mstatus code is □[0m□[36m□[1m401□[0m                        □[90m# PostLogin.validate_status_code()□[0m
    □[36mAnd □[0m□[36mjson request having code and message□[0m         □[90m# PostLogin.json_request_having_code_and_message(DataTable)□[0m
@Second
Feature: Post Login API with invalid username and password

  @invalid_username_and_invalidpassword
  Scenario: User calls Login Page API                 □[90m# 04_PostLogin_InvalidUsernameandPassword.feature:4□[0m
    □[36mGiven □[0m□[36mLogin URL And valid user name And password□[0m  □[90m# PostLogin.Login_URL_valid_user_name_And_password(DataTable)□[0m
    □[36mWhen □[0m□[36ma user calls the Login URL using post method□[0m □[90m# PostLogin.user_calls_Login_URL_using_post_method()□[0m
    □[36mThen □[0m□[36mstatus code is □[0m□[36m□[1m401□[0m                        □[90m# PostLogin.validate_status_code()□[0m
    □[36mAnd □[0m□[36mjson request having code and message□[0m         □[90m# PostLogin.json_request_having_code_and_message(DataTable)□[0m

4 Scenarios (□[36m4 skipped□[0m)
16 Steps (□[36m16 skipped□[0m)
0m0.000s
```

Updates Available

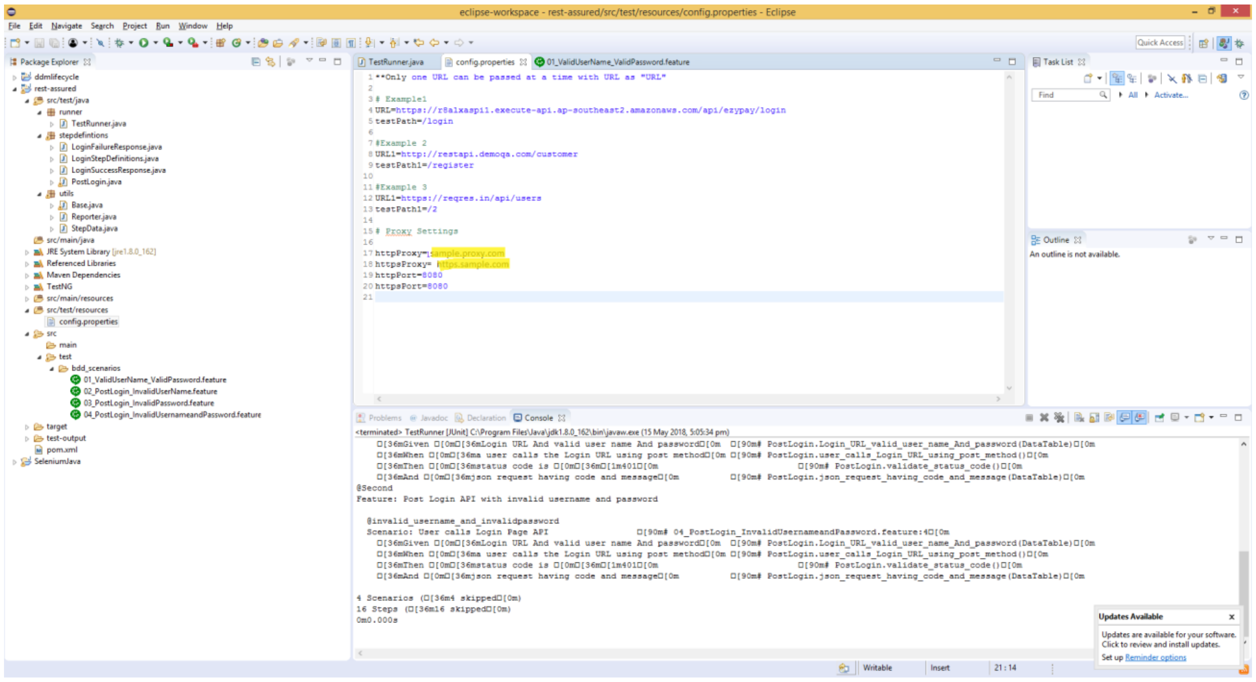
Updates are available for your software. Click to review and install updates.

Set up Reminder options

I have also provided more examples to test the HTTP endpoints.

2. Configuring the HTTP/HTTPS Proxy

Testing API's involves authorisation and in most of the environment the entire setup will be behind the firewalls and proxies. To overcome this issue and as an assumption of prerequisites, the framework expects the http and https proxy parameters such as Hostname and port number as below.



eclipse-workspace - rest-assured/src/test/resources/config.properties - Eclipse

```
**Only one URL can be passed at a time with URL as "URL"

# Example1
URL=https://r8alxaspi1.execute-api.ap-southeast2.amazonaws.com/api/ezypay/login
testPath=/login

#Example 2
URL1=http://restapi.demoqa.com/customer
testPath1=/register

#Example 3
URL1=https://reqres.in/api/users
testPath1=/2

# Proxy Settings

httpProxy=;sample.proxy.com
httpsProxy= https.sample.com
httpPort=8080
httpsPort=8080
```

```
<terminated> TestRunner [JUnit] C:\Program Files\Java\jdk1.8.0_162\bin\javaw.exe (15 May 2018, 5:05:34 pm)
    □[36mGiven □[0m□[36mLogin URL And valid user name And password□[0m  □[90m# PostLogin.Login_URL_valid_user_name_And_password(DataTable)□[0m
    □[36mWhen □[0m□[36ma user calls the Login URL using post method□[0m □[90m# PostLogin.user_calls_Login_URL_using_post_method()□[0m
    □[36mThen □[0m□[36mstatus code is □[0m□[36m□[1m401□[0m                        □[90m# PostLogin.validate_status_code()□[0m
    □[36mAnd □[0m□[36mjson request having code and message□[0m         □[90m# PostLogin.json_request_having_code_and_message(DataTable)□[0m
@Second
Feature: Post Login API with invalid username and password

  @invalid_username_and_invalidpassword
  Scenario: User calls Login Page API                 □[90m# 04_PostLogin_InvalidUsernameandPassword.feature:4□[0m
    □[36mGiven □[0m□[36mLogin URL And valid user name And password□[0m  □[90m# PostLogin.Login_URL_valid_user_name_And_password(DataTable)□[0m
    □[36mWhen □[0m□[36ma user calls the Login URL using post method□[0m □[90m# PostLogin.user_calls_Login_URL_using_post_method()□[0m
    □[36mThen □[0m□[36mstatus code is □[0m□[36m□[1m401□[0m                        □[90m# PostLogin.validate_status_code()□[0m
    □[36mAnd □[0m□[36mjson request having code and message□[0m         □[90m# PostLogin.json_request_having_code_and_message(DataTable)□[0m

4 Scenarios (□[36m4 skipped□[0m)
16 Steps (□[36m16 skipped□[0m)
0m0.000s
```

Updates Available

Updates are available for your software. Click to review and install updates.

Set up Reminder options

3. Sample Test Case

```
@First
Feature: Post Login API with valid user name and password
  @valid_username_and_password
  Scenario: User calls web service to connect to Login Page
    Given Login URL And valid user name And password
    |UserName|Password|
    |test-user|abc123|
    When a user calls the Login URL using post method
    Then status code is 200
    And json request having code and message
    |UserName|Password|
    |test-user|abc123|
```

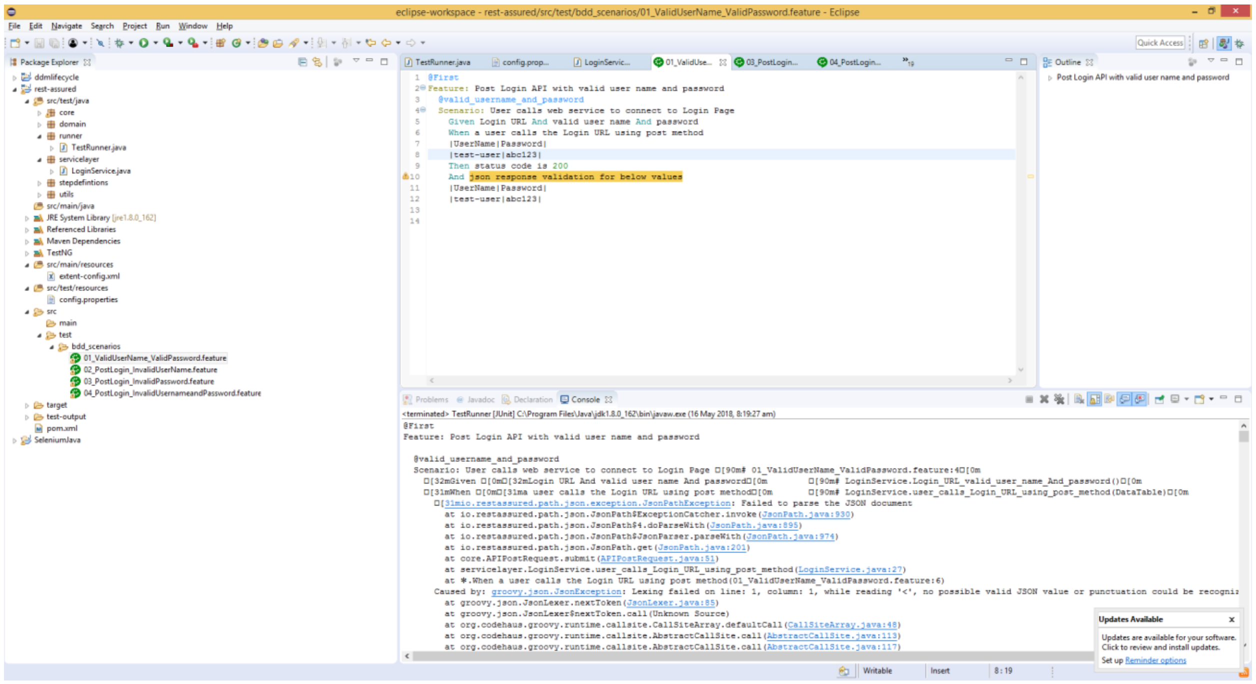
Expected Result:

The following response should be displayed:

```
{
"username": "sample-username",
"password": "sample-passw0rd"
}
```

4. Updating the feature file for test data

The test data can be modified from the feature file using the corresponding feature file as shown below. The value of the test data can be modified from the test cases after opening the feature file.



```
@First
Feature: Post Login API with valid user name and password
  @valid_username_and_password
  Scenario: User calls web service to connect to Login Page
    Given Login URL And valid user name And password
    When a user calls the Login URL using post method
    |UserName|Password|
    |test-user|abc123|
    Then status code is 200
    And json response validation for below values
    |UserName|Password|
    |test-user|abc123|
```

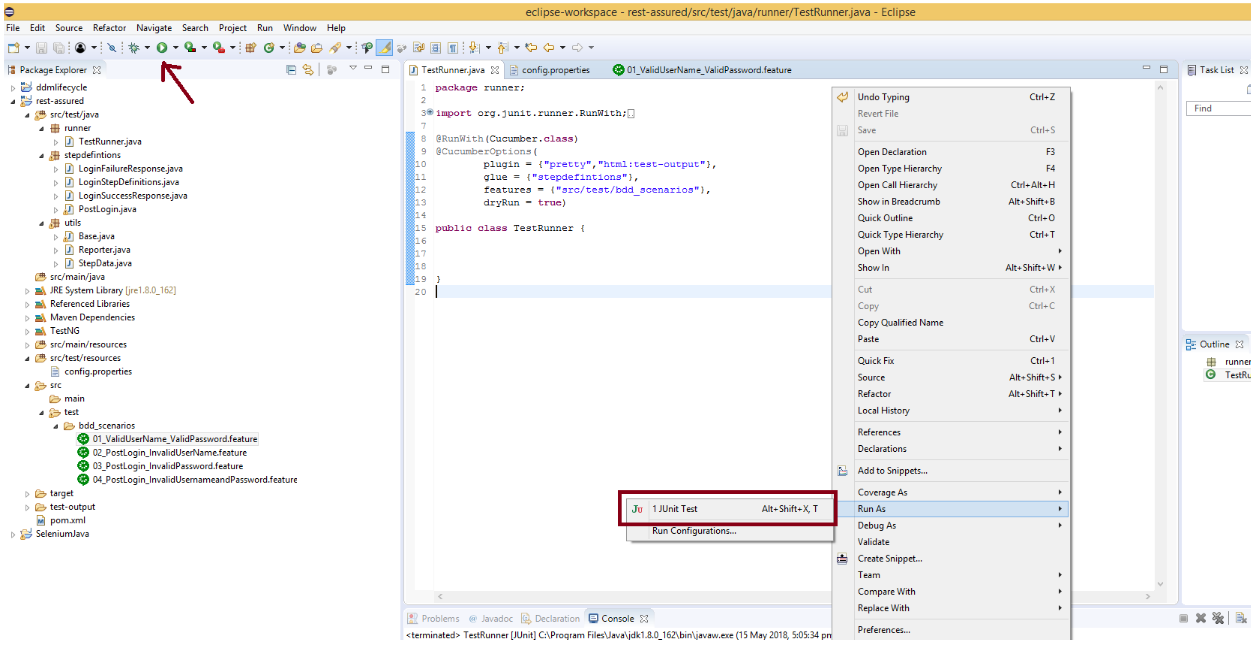
```
<terminated> TestRunner [JUnit] C:\Program Files\Java\jdk1.8.0_162\bin\javaw.exe (16 May 2018, 8:19:27 am)
@First
Feature: Post Login API with valid user name and password

  @valid_username_and_password
  Scenario: User calls web service to connect to Login Page □[90m# 01_ValidUserName_ValidPassword.feature:4□[0m
    □[32mGiven □[0m□[32mLogin URL And valid user name And password□[0m        □[90m# LoginService.Login_URL_valid_user_name_And_password()□[0m
    □[31mWhen □[0m□[31ma user calls the Login URL using post method□[0m       □[90m# LoginService.user_calls_Login_URL_using_post_method(DataTable)□[0m
      □[31mio.restassured.path.json.exception.JsonPathException: Failed to parse the JSON document
        at io.restassured.path.json.JsonPath$ExceptionCatcher.invoke(JsonPath.java:930)
        at io.restassured.path.json.JsonPath$4.doParseWith(JsonPath.java:895)
        at io.restassured.path.json.JsonPath$JsonParser.parseWith(JsonPath.java:974)
        at io.restassured.path.json.JsonPath.get(JsonPath.java:201)
        at core.APIPostRequest.submit(APIPostRequest.java:51)
        at servicelayer.LoginService.user_calls_Login_URL_using_post_method(LoginService.java:27)
        at ✽.When a user calls the Login URL using post method(01_ValidUserName_ValidPassword.feature:6)
      Caused by: groovy.json.JsonException: Lexing failed on line: 1, column: 1, while reading '<', no possible valid JSON value or punctuation could be recogniz
        at groovy.json.JsonLexer.nextToken(JsonLexer.java:85)
        at groovy.json.JsonLexer$nextToken.call(Unknown Source)
        at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCall(CallSiteArray.java:48)
        at org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.java:113)
        at org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.java:117)
```

Updates Available
Updates are available for your software. Click to review and install updates.
Set up Reminder options

5. Running the test cases

The Test run can be initiated from the TestRunner.java class as below option.

1. Click on the Run button from tool bar
2. Right Click Run As → Junit Test
3. Run from Command Prompt by executing the mvn test command

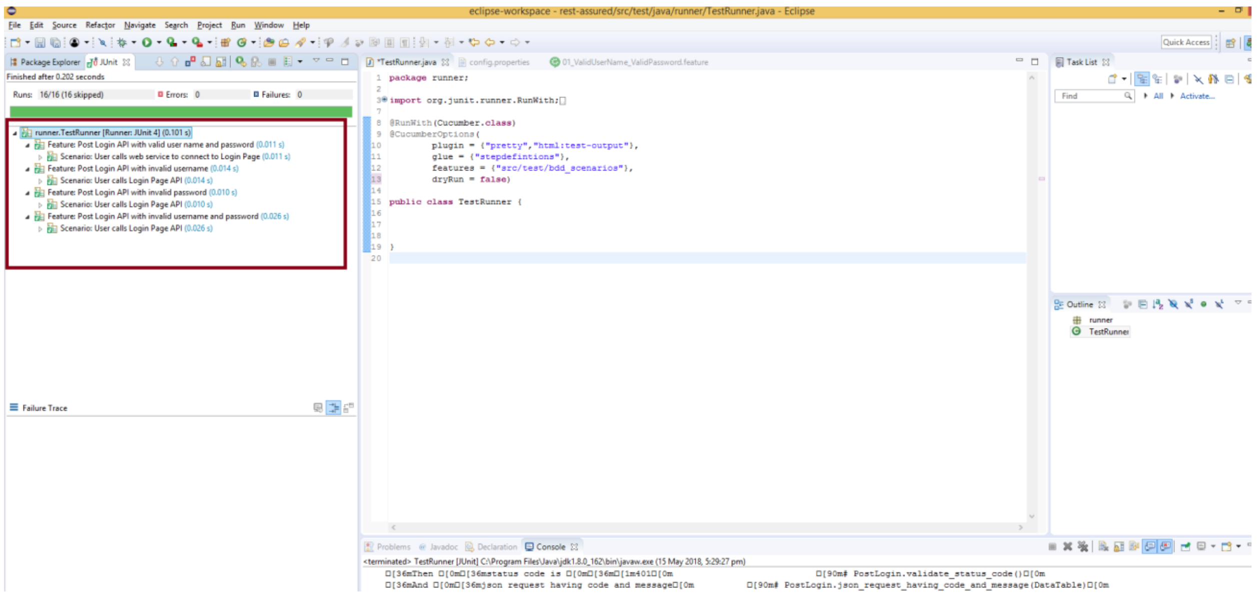


```java
package runner;

import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
        plugin = {"pretty","html:test-output"},
        glue = {"stepdefintions"},
        features = {"src/test/bdd_scenarios"},
        dryRun = true)

public class TestRunner {



}
```

```
<terminated> TestRunner [JUnit] C:\Program Files\Java\jdk1.8.0_162\bin\javaw.exe (15 May 2018, 5:05:34 pm
```

6. Viewing the output

Once test run is completed, the result can be viewed from the left hand side menu in eclipse. Addition option


eclipse-workspace - rest-assured/src/test/java/runner/TestRunner.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
*TestRunner.java
config.properties
01_ValidUserName_ValidPassword.feature
Task List
Outline
runner
TestRunner
Failure Trace
Problems
Javadoc
Declaration
Console

```
Finished after 0.202 seconds
Runs: 16/16 (16 skipped)   Errors: 0   Failures: 0

runner.TestRunner [Runner: JUnit 4] (0.101 s)
  Feature: Post Login API with valid user name and password (0.011 s)
    Scenario: User calls web service to connect to Login Page (0.011 s)
  Feature: Post Login API with invalid username (0.014 s)
    Scenario: User calls Login Page API (0.014 s)
  Feature: Post Login API with invalid password (0.010 s)
    Scenario: User calls Login Page API (0.010 s)
  Feature: Post Login API with invalid username and password (0.026 s)
    Scenario: User calls Login Page API (0.026 s)
```

```java
package runner;

import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
        plugin = {"pretty","html:test-output"},
        glue = {"stepdefintions"},
        features = {"src/test/bdd_scenarios"},
        dryRun = false)

public class TestRunner {



}
```

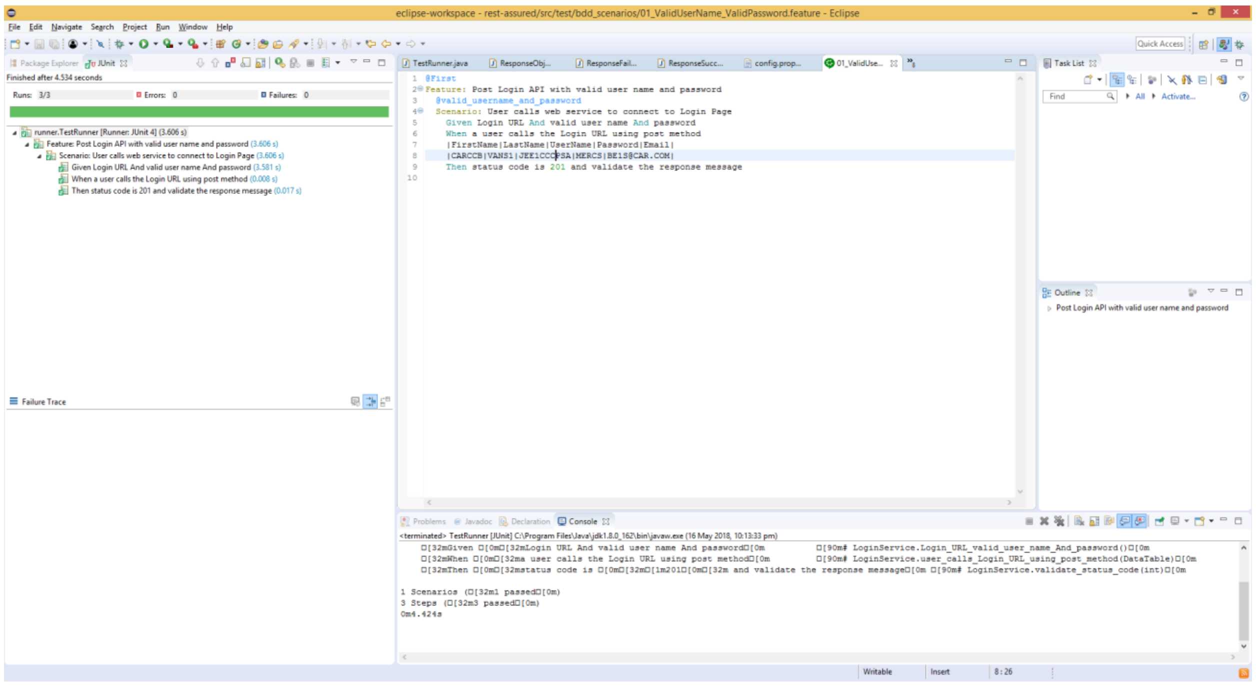
```
<terminated> TestRunner [JUnit] C:\Program Files\Java\jdk1.8.0_162\bin\javaw.exe (15 May 2018, 5:29:27 pm)
    □[36mThen □[0m□[36mstatus code is □[0m□[36m□[1m401□[0m                    □[90m# PostLogin.validate_status_code()□[0m
    □[36mAnd □[0m□[36mjson request having code and message□[0m         □[90m# PostLogin.json_request_having_code_and_message(DataTable)□[0m
```

Sample Testing on Open source http API's

URL:http://restapi.demoqa.com/customer/register

Gherkin Scenario:

```
@First
Feature: Post Login API with valid user name and password
  @valid_username_and_password
  Scenario: User calls web service to connect to Login Page
    Given Login URL And valid user name And password
    When a user calls the Login URL using post method
            |FirstName|LastName|UserName|Password|Email|
    |CARCCB|VANS1|JEE1CCCPSA|MERCS|BE1S@CAR.COM|
    Then status code is 201 and validate the response message
```


eclipse-workspace - rest-assured/src/test/bdd_scenarios/01_ValidUserName_ValidPassword.feature - Eclipse

```
Finished after 4.534 seconds
Runs: 3/3    Errors: 0    Failures: 0

runner.TestRunner [Runner: JUnit 4] (3.606 s)
  Feature: Post Login API with valid user name and password (3.606 s)
    Scenario: User calls web service to connect to Login Page (3.606 s)
      Given Login URL And valid user name And password (3.581 s)
      When a user calls the Login URL using post method (0.008 s)
      Then status code is 201 and validate the response message (0.017 s)
```

```
@First
Feature: Post Login API with valid user name and password
  @valid_username_and_password
  Scenario: User calls web service to connect to Login Page
    Given Login URL And valid user name And password
    When a user calls the Login URL using post method
    |FirstName|LastName|UserName|Password|Email|
    |CARCCB|VANS1|JEE1CCCPSA|MERCS|BE1S@CAR.COM|
    Then status code is 201 and validate the response message

```

```
<terminated> TestRunner [JUnit] C:\Program Files\Java\jdk1.8.0_162\bin\javaw.exe (16 May 2018, 10:13:33 pm)
    □[32mGiven □[0m□[32mLogin URL And valid user name And password□[0m          □[90m# LoginService.Login_URL_valid_user_name_And_password()□[0m
    □[32mWhen □[0m□[32ma user calls the Login URL using post method□[0m          □[90m# LoginService.user_calls_Login_URL_using_post_method(DataTable)□[0m
    □[32mThen □[0m□[32mstatus code is □[0m□[32m□[1m201□[0m□[32m and validate the response message□[0m □[90m# LoginService.validate_status_code(int)□[0m

1 Scenarios (□[32m1 passed□[0m)
3 Steps (□[32m3 passed□[0m)
0m4.424s
```

FAQs

Q: How would you scale out your answer in the first task to cover an entire system? Would there be tweaks?

A: This framework provides an excellent coverage of test cases and have highly flexible reusable components. There are four important components.

1. Core Layer – All reusable business logics and repeated operations can be placed here and can be used for any number of new functional services.
2. Domain Layer – All important request and response operations on the Business Logics performed in this layer.
3. Service Layer - This is specific to the endpoints and the test scenarios and can be also called as step definitions. The implementation of test cases are dealt in this component. The methods implemented are highly dynamic in nature and can accept different parameters.
4. Configuration Layer – Any number of Endpoints can be configured if the number of API's are rapidly increasing

The main change would be to add as much as scenarios

Q. How much do you test? What scenarios do you test?
A: This mainly depends of the number of functionality ready to be tested. The number of scenarios that will functionally cover end to end for all the endpoints on the given microservices will be tested using the behavioural driven approach.

- System test cases of all micro services components
- End to End Testing of all End points

- Multiple feature files to validate the Endpoints and the features

Q: What factors would you consider to ensure you get the widest end-to-end coverage, AND yet keep it easy to maintain?

A: There are various factors involved to achieve the end to end coverage of the testing. The key factors include

- All the requirements should have a corresponding gherkin scenario in the feature file
- Gherkin scenarios should be easy to understand and step by step in nature
- Input data should be separated from the step definitions and should be parameterized
- Methods should be dynamic in nature and can handle different set of data (eg: Data Table)
- Handling the input components (data table) also should be dynamic. User should be allowed to change the column header name and the data
- All configuration values should be separated from the step definitions and the wrapper classes
- All the common functionalities should be part of core layer that can be reused