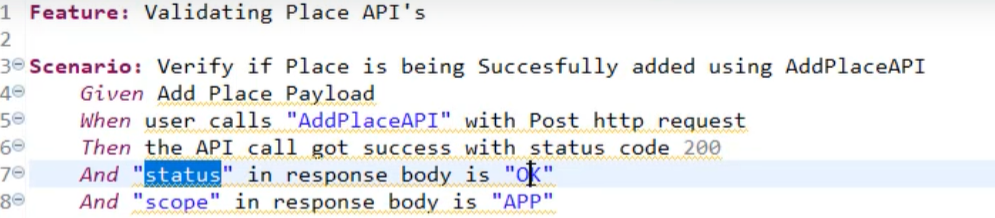


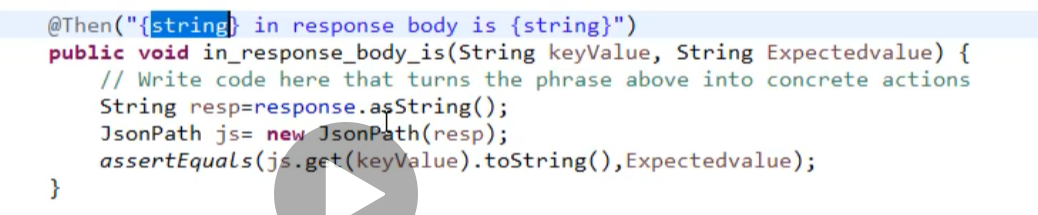
1. Create a Maven project in Eclipse and also install the plugin cucumber eclipse to work with gherkin language.
2. Add the following dependencies
3. RestAssured
4. Cucumber java
5. Cucumber Junit
6. ~~TestNg~~ (when you run it as junit, please do not add this else when running via maven you will not see the test case count)
7. Jackson databind
8. Jackson annotation
9. Jackson core
10. Gson
11. Create packages
12. features 🡪 to have all the feature files
13. Step Definition 🡪 To have all the scripts related to the feature file
14. Test Runner 🡪 to run the test cases
15. Resources 🡪 to have class like test data (setting data for serialization) and utils class (to have repeated codes like request and response specification) and global properties file
16. Pojo 🡪 under src/main/java package to define the setter and getters for request payload

In the below feature file, we are adding AddPlaceAPI, status, OK and other in double quotes. The reason is we can make use of the same step definition file with different key word on it. Let us say DeleteplaceAPI, instead of status we can pass other key from response like message, scope etc., To get the step definition template, we can simply run the test runner class without creating test class for step definition and in console, it will suggest implementing the steps wrote in feature file.

A screenshot of a computer screen

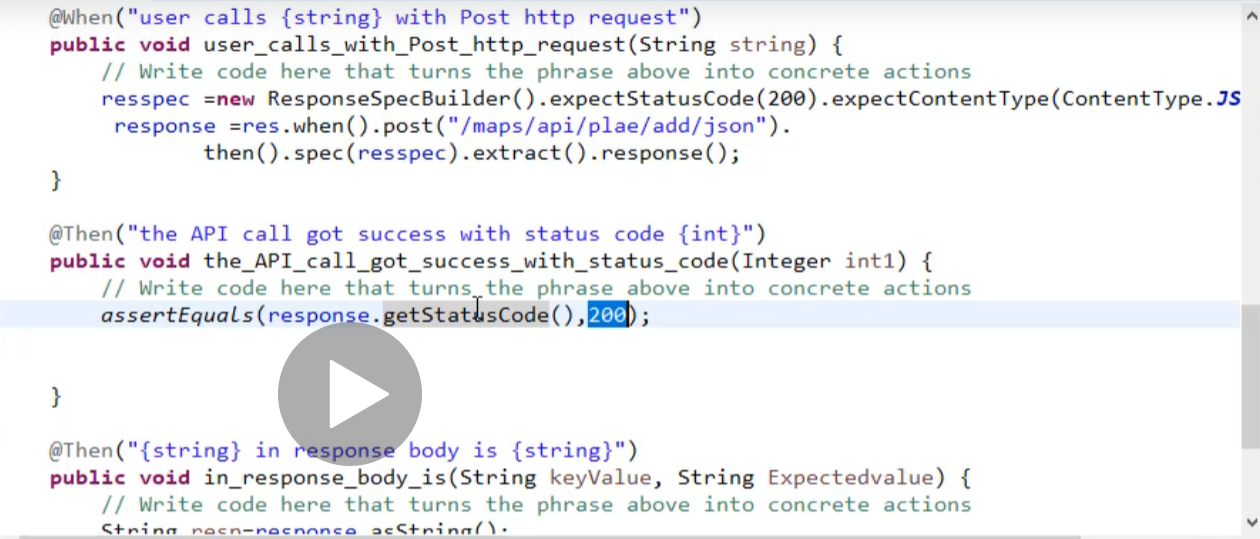
Description automatically generated











1. Create the packages as mentioned above like feature, step definition, test runner, utils (RequestSpec, ReadProperty), POJO (POJO classes, test data builder) et.,
2. Update the feature file for one scenario in gherkin language
3. Create the test runner file with Junit like run with and cucumber options which takes features, step definition as arguments.
4. Simply run the class and the console will give you the steps which need implementation.
5. Copy pastes the same in the step definition file.
6. Create the POJO class with all attributes of the request body
7. Create a test builder class with a method to build the request body which returns of the object of the instantiated POJO class.
8. Create request specification class under utils with a method to return the request specification builder.
9. Create Read property class with a method to read the data from property file and return as the string object.
10. In step definition, create objects for all the class that we want to work with as shown below

A screenshot of a computer program

Description automatically generated

1. To make your payload dynamic let us say phone number, name in payload to be dynamic, then update the feature file scenario into scenario outline with examples key word.

A screenshot of a computer program

Description automatically generated

1. Make the step definition to take arguments and update the method name as well as the test builder method

A screen shot of a computer code

Description automatically generated

1. A screenshot of a computer code

   Description automatically generated
2. Parameterize the test case using scenario outline with examples key word as discussed above but that will create a problem where the log file will overwrite the old values for each run.
3. So, to maintain the logs for all runs, we need to modify the request specification class as below

A screenshot of a computer program

Description automatically generated

**Enum:**

Why do we need Enum?

Right now, all the resources are hard coded in our step definition file. We can send it from another class using the getters method by creating an object in the step definition file. But let us assume what if there are 100 resources in real time. Then it will be hard to create a 100 getters method. To overcome this, we can make use of Enum class.

**What is Enum and how to declare it?**

A special class nothing but collection of constants and methods

To declare a Enum class, use the word Enum instead of class i.e., create a class first and change the class to enum as mentioned below

Public Enum EnumName{

}

Inside Enum, we should declare methods as

When you have one method

AddPlaceAPI(“/maps/api/place/add/json”);

If you have multiple methods then

AddPlaceAPI(“/maps/api/place/add/json”),

getPlaceAPI(“/maps/api/place/add/json”),

DeletePlaceAPI(“/maps/api/place/add/json”);

**Note**: If it is collection of methods then separate them by comma and put semi-colon at the last method

Also use a constructor which should take argument if method takes else no. In our case, since the methods takes parameters, so does the constructor.

EnumName(String resource){

}

Then to retrieve the value of the string from the methods, create a private variable and use the following methods and change the constructor as

Private String **resource**;

Enumname(String resource){

this.**resource** = resource

}

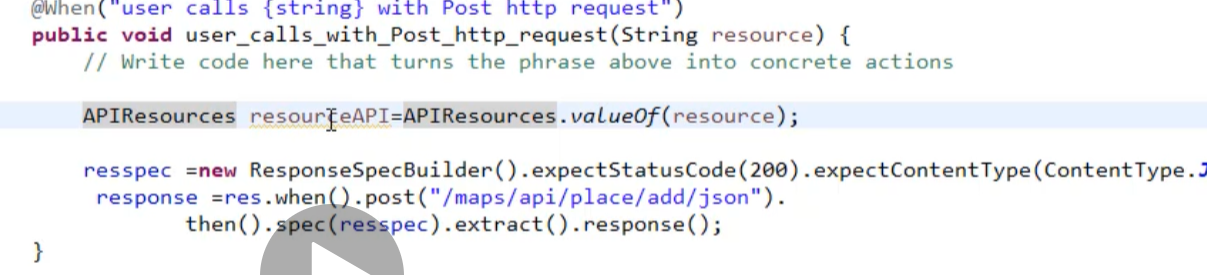
Public String getResource(){ // getters method for the private variable

Return **resource**

}



Redefine your steps definition as below where value of is a method of enum class.





From the screen shot where APIResources is the enum class name and resourceAPI is an object of the enum class.

Expand the test case by adding verify the place\_id matches the added place with get method as well as add a new scenario to delete the place that was added in add method. Now if you observe carefully delete scenario, all we need to write script is only for given because rest of the lines are already built-in.



**Note**: When you want to pass variable from one scenario to another, then make the globally declared variable as static.

**Hooks are used to set pre-conditions and post-condition**

Assume you have two test cases one is to add place and other is delete place but due to req, you are supposed to run only delete test case. In this situation, unless to add a place and have its place id, we can’t delete it. During this scenario we can run our test cases using hooks as below, where we will create a separate class in step Definition and with @before annotation, mention the scenario name which has the tags

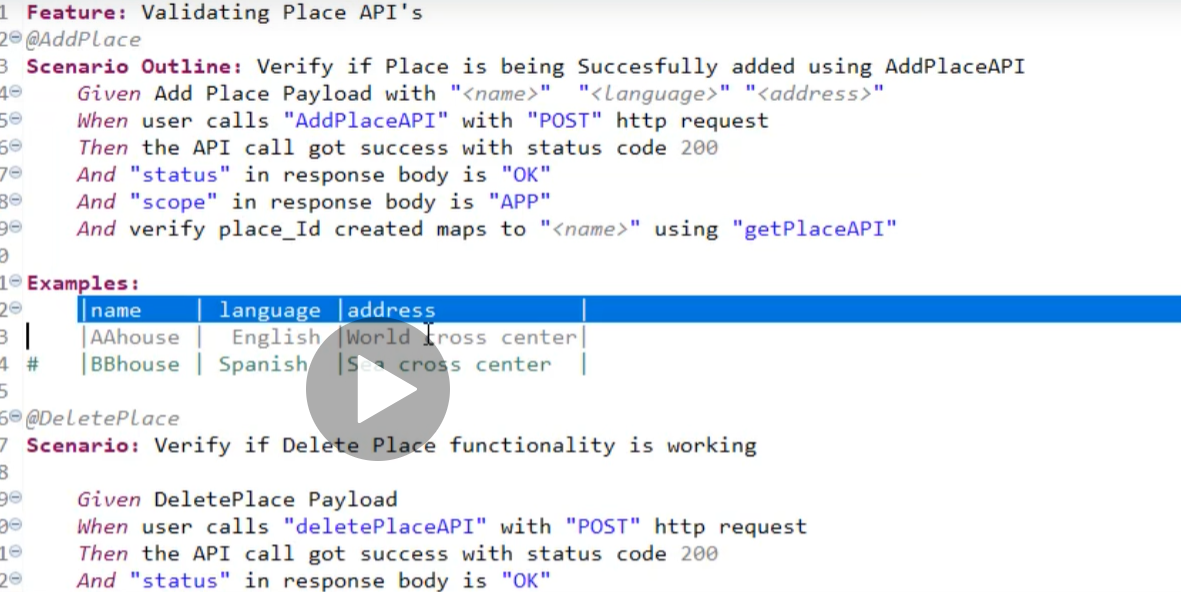


Test runner with tags… In the below screen shot tags are commented out for some specific scenario but in real time the cucumber options should be written as

@CucumberOPtions(features=”src/test/java/features”,glue={“stepDefinition”},tags={“@AddPlace”})

Note: @Add place is tag written on top of scenarios in feature file





***To run the framework in maven***

Open the command prompt and change the directory to the project location using

1. Cd “path of project(can get using right click on project 🡪 properties)”
2. mvn test
3. To run a particular tag use the following command in maven

mvn test –Dcucumber.options=”-- tags @AddPlace”

To get reports in json format, add one more option in test runner like

Plugin=”json:target/JsonReport/cucumber-reports.json”



**Reports:**

To create report, we need to have a new jar in pom and few plugins. To get the code, google maven cucumber report and go with the one which reads “DamianScze……” in git hub

Modify the test runner cucumber options as below i.e., add a new key named plugins

@CucumberOPtions(features=”src/test/java/feature”,plugin=”json:target/jsonReports/cucumber-reports.json”,glue={“stepDefinition”}, tags={“@AddPlace”})

Then move to command prompt and change the directory to project home and simply say ***mvn test verify*** *and press enter*

After this is success, reports will be placed under target folder

***Jenkins:***

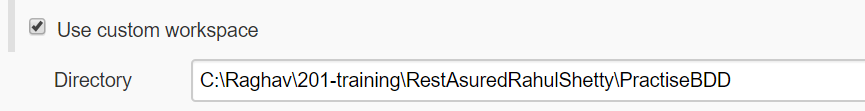
After you download and launch Jenkins using command prompt by navigating to the path where we have Jenkins with “java –jar Jenkins.war httpPort=9090” (without quotes) port number can be of user’s wish and in my case it is 8080

Open the browser and say localhost:8080

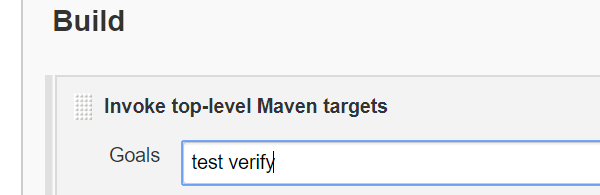
Login into Jenkins (admin, Rumble@12)

Create a new project maven / freestyle and give description

Under general tab, select “use custom workspace”and give the path of the root directory of the framework / project i.e., **General 🡪 Advanced 🡪 Use Custom Workspace 🡪 value of root directory of project**



Then go to build section and select “Invoke top-level maven targets” and give the command that you used to run the project in command prompt say “test verify” (no need to mention mvn since we are building maven section)



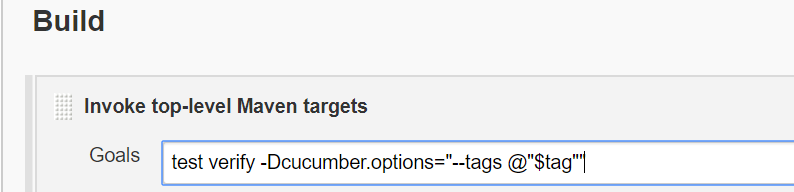
Save the project and click Build now.

***To parameterize the jekins jobs***

General 🡪 This project is parameterized 🡪 choice parameter 🡪 Give a name and in the choices box add all the tags names that you have added for your scenarios in feature file.

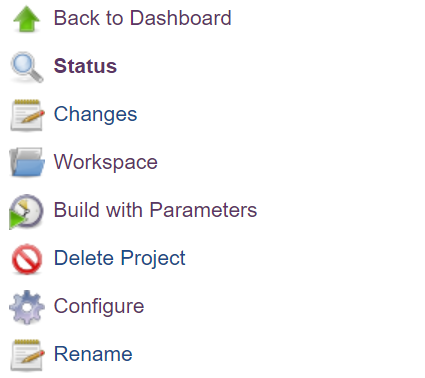
say you have given the maven command previously as “test verify –Dcucumber.options=”—tags @**Addplace**”, then every time it will keep running the add place related tag test cases. So to overcome and to parameterize the Jenkins, check “This is a parameterized project” under general section and give all the tag name under description box say you have add place, delete place and regression tags

Then the same command can be re-written as “test verify –Dcucumber.options=”—tags @”$nameoftag””

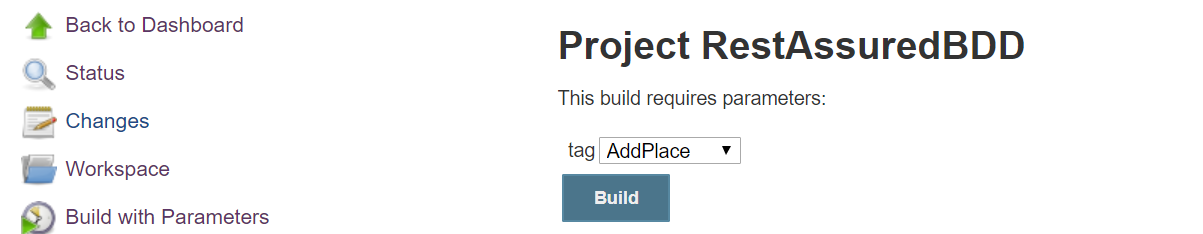


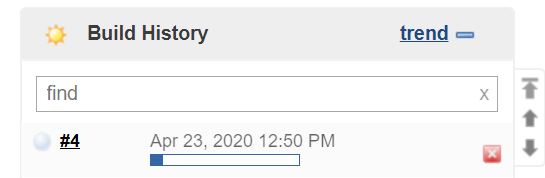
Where name of tag is the name given for parameterize description

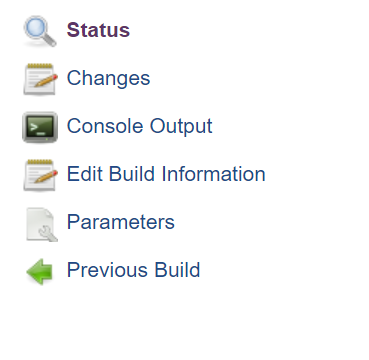
To confirm whether the jenkin jobs are parameterized, in the dashboard you can see “Build with parameter” instead of “Build”. Upon clicking it will ask the user to select the options available in the dropdown (which are nothing but the parameters name which you have provided when you selected this is a parameterized project”)

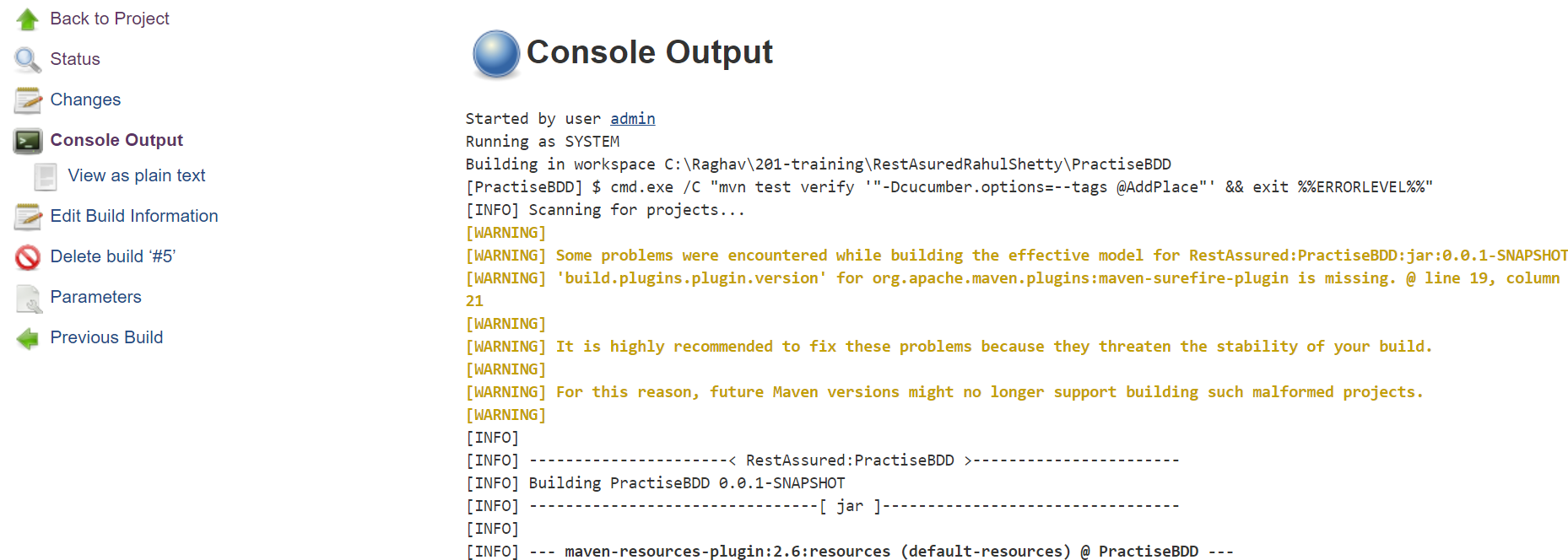


Click Build With Parameters



To run the project select the tag name and click build now. To see the console, refresh 🡪 click the build progress bar and then click console 





To see the reports. Click Back to project 🡪 Workspace 🡪 Target 🡪 cucumber-html-reports (name as in you configure in test runner file under plugin tag) 🡪 overviewfeatures.html