**Adidas technical challenge**

**Framework and project explanation –**

* BDD and Data Driven framework is used as the part of the project which concludes that the project is of Hybris framework.
* Reason for using BDD is that is allows automation of API’s and functional validation in easily readable and understandable format.
* Cucumber tool is used to for the BDD approach.
* Gherkin is a simple, lightweight and structured language which uses regular spoken language to describe requirements and scenarios.
* Data driven framework is used to store set of data in external file, here a property file is used to store the **API** url and **APP\_ID** and conveniently used as key-value pairs and be used in the different files where ever it is necessary.
* Mainly used for maintenance and re-usability of code.

**Project explanation –**

* **Features -**

Features files contain all the test case which has been identified and written in Gherkin language. It follows the syntax of the gherkin and is mapped to step definitions. There are 4 features files which has been identified which are as mentioned below.

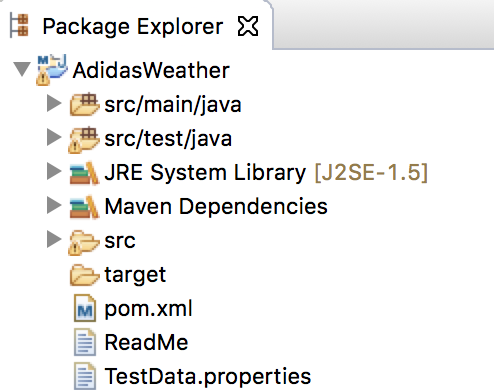
* + **GetTemperatureByCity –** Use to retrieve temperature based on City name.
  + **GetTemperatureByCityID -** Use to retrieve temperature based on City ID.
  + **GetTemperatureByZipCode -** Use to retrieve temperature based on Zip code.
  + **GetTemperatureByLatLong -** Use to retrieve temperature based on latitude and longitude.
* **Options –**

Option package mainly contain cucumbertests java file wherein a cucumber test in run from. It has following things –

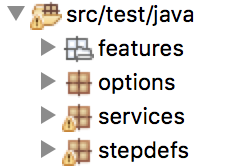
* Pretty plugin declaration for generating test reports.
* Glue declaration which helps cucumber locate step definition file.
* Features path declaration.
* **Services –**
* It mainly contains Utils java file.
* A java program to read the data from the properties file is written in it.
* **Stepdefs –** 
  + This is where the automation code is written.
  + The steps in the gherkin file directly map to the step definitions.
  + Each step in the gherkin maps to a step definition that in turn contains a block of code that gets executed when the scenario is ran.
* **Pom.xml –** 
  + All the dependencies are added in here.
  + Jars are added to the project as when the maven update is run.
* **ReadMe file –** 
  + It has technical specification used in the project.
  + Set up needed for the project to run.
  + Steps to execute the project.
* **TestData.properties**
  + A property file wherein it contains data and can be called in the project as and when required.

**Screen references of the project –**

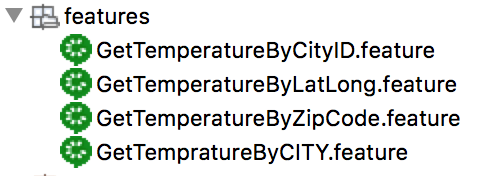
1. **Adidas weather project**

****

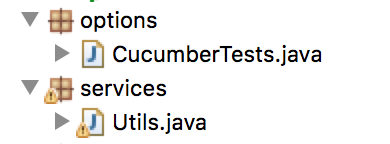
1. **Contents of the project**



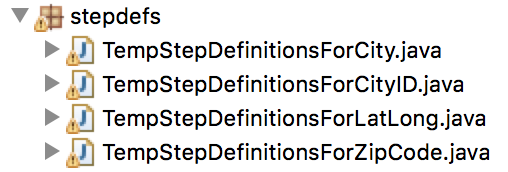
1. **Feature files**



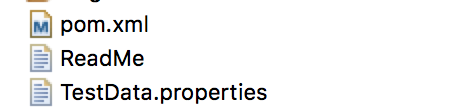
1. **Options and Services**



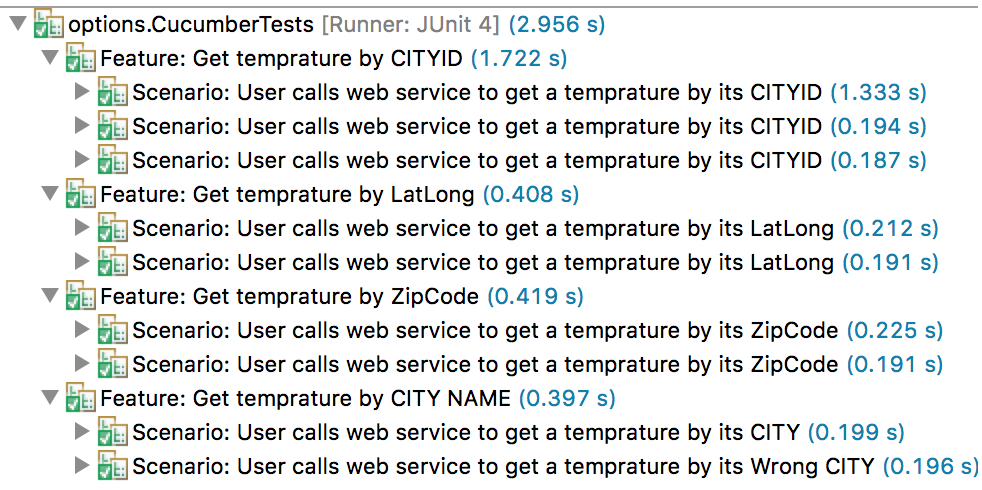
1. **Step definitions**

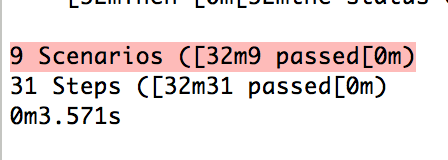


1. **Other files**



1. **Test results**





**Multiple choice question** -

|  |  |
| --- | --- |
| How do you evaluate if your testing suite is good ? | |
|  | Checking the test reports are all green |
|  | Checking the function, statement, branch, condition coverage result |
|  | Checking the profiler report |
|  | Checking if removing code from the SUT the test result is still green |

Answer –

My first priority would be **“Checking the function, statement, branch, condition coverage result**”. Basically, test coverage is foremost import thing for any given testing.

**Test case coverage and test result** –

1. Get temperature by City
   1. Positive case by adding valid city name and validating the response for the country name. Expected response code is set to 200
   2. Negative case by adding invalid city name. Expected response code is set to 404 for city name not found.
2. Get temperature by City ID
   1. Positive case by adding valid city ID and validating the response for the country name. Expected response code is set to 200
   2. Negative case by adding invalid city ID. Expected response code is set to 404 for city ID not found.
   3. Negative case by adding invalid city ID and expected result would be set to 400 which results in bad request.
3. Get temperature by Zip code
   1. Positive case by adding valid Zip code and validating the response for the country name. Expected response code is set to 200
   2. Negative case by adding invalid Zip code. Expected response code is set to 404 for city name not found.
4. Get temperature by Lat-Lon
   1. Positive case by adding valid lat-long and validating the response for the country name. Expected response code is set to 200
   2. Negative case by adding invalid lat-long. Expected response code is set to 400 for which results in bad request.