Rest Assured with BDD Framework:

In my current organization we achieve API Automation using Rest Assured with BDD framework.

To achieve API Automation we create maven project, at the time of creating maven project we add group id and artifact id, group id is nothing but the package name and artifact id is nothing but the project name.

Once we create maven project it will generate the source main java, source test java ,source main resource , source test resource packages automatically and it will generate most important file as pom.xml file.

pom.xml (project object model. Extensive markup language) is heart of maven project.

And in this pom.xml file we add different types of dependency as like Rest Assured dependency, json-path dependency, Jackson-databind dependency, Hamcrest dependency, json Schema dependency, Lombok dependency, cucumber-core dependency, cucumber-java dependency, cucumber-testng dependency, testing-dependency, Apache POI dependency , extent-report dependency and extent report cucumber7 adapter dependency in pom.xml file.

For rest assured maven dependency we use 4.5.1 version, for json-path dependency we use 4.51. version, for Jackson-databind dependency we use 2.13.0 version, for Hamcrest maven dependency we use 2.2 version, for json-schema validator we use 4.5.1 version, for Lombok dependency we use 1.18.30 version, for cucumber-core we use 7.12.0 version, for cucumber-java dependency we use 7.12.0 version, for cucumber-testNG we use 7.12.0 version, for maven testNG we use 7.4.0 version, for Apache POI we use 1.9.0 version and for 5.1.1 version dependency, and for cucumber- adapter7 we use 1.10.1 version of dependency,

Once we add different dependency in pom.xml file then we create different types of packages in source main java as like

i) Base API Layer Package
ii) Model Package that is POJO package
iii) Service Layer Package
iv) Reader Packages
v) Constant Package
vi) Utils Layer Package

In source test java we create different types of package as like

i) Feature Package
ii) Step API Definition package
iii) Test Runner Package

In source main resource package we create different types of file as like apiconfig.properties file.

In source test resource package, we create different types of folders as like Request payload folder and JSON Schema folder and extent-config.properties file

And in Project level locations we create Extent Reports folder.

Already **I have already told you in source main java we create BaseAPI Package**, in BaseAPI package we create BaseAPI class, BaseAPI class is parent class of all the classes in framework, here we achieve the hierarchical inheritance concept, inside the BaseAPI class, we have created static method as getRequestSpecification() method and inside this method we set the BaseURI using RestAssured.baseURI method, we store actually BaseURI in properties file in source main resources folder and then we read this file in reader package, by using customize method we read the file in BaseAPI class. And after that we get the RequestSpecification object using given() method, given() method is used to add the precondition for each and every request, and this given() method is present inside the RestAssured() class, and then we generate request all logs using log().all() method and we return the RequestSpecification object so we add the different types of request parameters in Step Api packages.

Then we have created Model Package in source main java, model package is also called as POJO package, in this package we create container classes for each and every resource, just imagine we have 100 resources or end points in application then in this package we create 100 POJO classes, and in this POJO classes we define the Entity private variables with its datatypes and we define the getter() method to get the values from private variables, we define the setter() method to set the new values for variables, and we define toString() method to convert the POJO class object into the String.

And in POJO classes we can generate the getter() and setter() method by using Lombok dependency, Lombok dependency it provides the different annotation to generate the getter() method , setter method , toString() method, default constructor and arguments constructor, hashcode and equals method as like @Getter annotation, @Setter annotation, @ToString annotation, @NoArgsConstructor @AllArgsConstructor and @Data annotation.

@Data annotation it will generate the getter method, setter method, toString() method, HashCodeAndEquals methods.

In my current project we use Lombok dependency to above methods.

Then we have created Reader package in source main java, inside the Reader package we have different 3 different types of classes as like Properties reader class, JSONReader classs and Excel sheet reader class.

Inside the Properties Reader class we read the Properties file using Properties class, in this class we have created static method with 1 string arguments and with string return type. In this method we first we have created object of FileInputStream class by passing the File Path locations once we read the file then we have created object of Properties class to read the properties from properties file, then we have to load the all properties in current class using load() method from Properties class and by passing the FileInputStream instance, then we use getProperty() method by passing the Property key name defined in the actual Properties file and then we return the property values for specific key.

Then inside the Reader package we have defined the JSONReader class to read the values from JSON file, we can read the JSON values from JSON file by using 3 different ways as like by using Jackson-databind dependency, gson dependency and google json simple dependency, but in my current organization we read JSON file using the Jackson-databind dependency,

In JSONReader class we have created static method to read the values from JSON file and convert it into the POJO classes, that is nothing the but the deserialization concept,

To read the values from JSON file first we have created object of FileReader class by passing the JSON File Path location, and then we have created Object of ObjectMapper class.

Then we use readValues() method from ObjectMapper class by passing the FileReader class instance name or object name and this readValues()method it will return the JsonNode and then we use treeToValues() method from ObjectMapper class by passing JsonNode name using get() method and POJO class name and this treeToValues() method it will capture all the values from JSON file and it will set the values to the POJO class Entity variables. And this treeToValues() method it will return the POJO Object, and we can use this POJO class Object to add the request body to the HTTP Requests.

Then inside the Reader package we have created ExcelReader class to read the values from Excel sheet and to read the values from Excel sheet we use Apache POI dependency,

To read the values from excel sheet first we have created object of file class by passing the excel file location then we have 2 different types of File in excel sheet as like .xls file and .xlsx file

if u want read .xlsx file we have to created object of XSSFWorkbook and if u want read the .xls file then we have to created object of HSSFWorkbook. In my current we have used .xlsx file. this XSSFWorkbook object it will load the all the sheets from excel sheet, then we have to focus on specific sheet using getSheetAt() method by passing the Sheet index, and this method it will return the XSSFSheet object, once we get the XSSFSheet object then we can count how many rows present in sheet using getLastRowCount() method, this method it will return total numbers of rows present inside the Excel sheet and if u want to count how many columns present inside the sheet then first we focus on first row using getRo() method then we use getLastCellNum() method to count how many cells in excel sheet.

Once we count how many rows and columns in excel sheet then we use capture the specific sheet values using getRow() by passing rows number , getCell() method by passing the column number and then we use getStringCell() method to capture the string values from Excel sheet. Same if u want to capture the numbers then we use getNumbericCell() method etc.

Then next we have created Constant Package in this package we have created container class to declare the endpoints as per the resource name. in constant package we have created interface and in this interface we have created variables, once we declare any variables in interface we cannot modified it, this variable it will be consider as final. For each and every resources we have different endpoints, inside this package we have defined endpoints for POST Request, GET Request, PUT Request, PATCH Request, DELETE Request.

Then next we have created ServiceLayer package in source main java, in this package we defined the container class to provide the service for each and every endpoint Request. Inside the Service layer package we provide service for POST Request, PUT Request, And PATCH Request.

Inside the service layer package we call method from Reader package to create the Request body for the POST Request, PUT Request, PATCH Request ,already we have read the JSON file then we have convert this method to the JSON of String format because when we hit any HTTP Request we pass the request body in JSON format, so in this service layer package we have created different types of service layer container classes to provide the service for POST, PUT and PATCH Requests. And we call this class method inside the Step API definition packages.

Then Next we have created Utility Layer package in Source main java class ,in this class we have created different types of class as like status code interface, status line interface, Parameters class, Authorization class, and other required reusability classes.

Inside the Status code interface we have define the status code information as like what is used to each and every status code in variables, and then we created status line interface to provide information of status line and its use. And we call this method inside the Step API definition package to add the assertion, then we have created Parameters class, and in this Parameter class we have created different method to add or remove the parameters from the Path Parameter or Query Parameters basically in this class we have created Object of HashMap class, and this HashMap object it accept the key-value pair and same query parameter and path parameter also it accept the key-value pairs values, to add the values we use put()method by passing the key and value and to remove the values we use remove()method by passing key name. inside the Utility layer package we have defined the this types of object so we can reuse same variables multiple times as per the requirements wise.

Then inside the Utility layer package we have created Authorization class to add the authorization for request, basically there are different types of authorization in API, basic authorization, digest authorization, oauth1.0 and OAuth2.0.

In Authorization class we have created different types of method as like basic auth()method, digest auth method, oAuth1 method OAuth2 method and we call these method inside the Step Api definition packages.

And Utility layer packages we have created headers class and in this class have created static method, inside the static method we have created object of HashMap class with String and String generic and we have added all the required headers key-value pair in hashMap object using put() method and then we have called baseAPI class getRequestSpecification method by using class name and then we have used headers ,method by passing the HashMap object name.

And we have created one more method to upload the files in server we use multiPart() method we have created object of File class by passing file location, then we have called the getRequestSpecification method from BaseAPI class then we have used multiPart() method by passing the File object instance name.

Then we have created Feature Package in Source Test Java Package to store all feature file we create multiple files as per the requirement wise but extension of feature file must be dot feature file. and inside the feature file we use plain text language to describe the what is exact requirements with the help of gherkin keyword, in feature file we define first keyword as Feature followed by colon symbol then we define short description of feature , and then next we define the Backgound keyword followed by colon and this background keyword works like it is pre condition for each and every scenario in same feature files and then next we define the Scenario keyword followed by colon symbol and then we mention short description of scenario, then next we define the scenario different steps using Given, When, Then, And ,But and * keyword. Given keyword is used to mention the precondition, When keyword is used to perform actions or events, Then keyword is used to result or outcome or add the assertion, And keyword is used to combine the 2 or more statement and But keyword is used to mention multiple condition and * keyword is global keyword, and if u want to achieve the data driven testing then we use Scenario Outline keyword followed by colon symbol and then we maintain the test data Examples keyword followed by colon symbol and we separate the test data using pipe symbol and as per the requirements wise we can use multiple Examples keyword and in Feature level or Scenario level we use tags with the help of @Symbol followed by tagName.

Then next we have created Step API definition packages in source test java package, in this package we have maintained all generated snippets and inside the step API definition package we call all method required method from source main java classes, and inside the step Api Definition package we add the assertion for status code, status line, response time, response body , response headers using Hamcrest assertion, as per the snippets we are calling methods from the source main java class, to add the path parameters we use pathParams() method from RequestSpecification interface by passing the key-value pair and to the Query Parameters we use queryParam() method from RequestSpecification interface by passing the key-value pair, to add the headers we use header() method from RequestSpecification interface by passing the key-value pair, to upload the file we use multiPart() method RequestSpecification interface by passing file location, to add the request body we use body() method RequestSpecification interface by passing the request body in JSON format, from Service layer package, we use different types of authorization as like basic authorization , we use auth().basicAuth() method from RequestSpecification interface by passing the username and password, and some endpoints it accept the digest authorization then we use auth().digestAuth() method by passing username and password from RequestSpecification interface then some endpoints it accept the oAuth2 then we use auth().preemptive().oAuth2() methods from RequestSpecification interface

by passing the access tokens then we perform different types of actions using RequestSpecification interface as like POST, PUT,PATCH,GET, and DELETE method, post method is used to created a new entity in server, put method is used to update the whole entity in server , patch request used to update the partial entity in server and get is used to retrieve the entity from server and delete is used to delete the entity from the server.

Once we performed any Actions we will get Response object, once we get Response Object then we use then().assertThat() method to validate the different things status code, status line, response time, response body, response headers. assertThat()method it will return the ValidatableResponse interface and by using ValidatableResponse interface we add assertion as like we capture status code using statusCode() method and passing expected status code, and then we capture the status line using statusLine() method by using statusLine() method by passing the expected status line and if u want to check only specific keywords then we use Hamcrest library method as like containString() method from Matchers class, to capture the time we use time()method to validate response time should be below 5000 ms or 5 sec then use Matchers,lessThan() method and to capture the values from response body we use Json_Path library from JsonPath library. In JsonPath library there different methods available to capture the string values we use getString() method , to capture the integer values we use getInt() method, to capture the arrays value we use getList() method, to capture map value or nested json value using getMap() method, once we capture response body values then we add the assertion for that values using the Hamcrest library, in Hamcrest library they different methods to add the assertion for String value as like equalTo() method, containString() method, startWith() method, endsWith() method, is() method, to add the numbers assertion in Hamcrest library we have equalTo() method, greaterThan() method, greaterThenEqualTo() method, lessThan() method , lessThanAndEqualTo() method, to add the assertion for arrays they define the hasItem() method and hasItems() method, to add the assertion for nested json or map they define the hasKey()method, hasValue()method and hasEntry() method we use above methods as per the requirements wise. And also we capture the response headers using header() methods and then we add assertion for response important headers key-value pairs and then we check validate the response json Schema using Json Schema validator library , basically JSON schema validator is used to validate the response body datatype structure and we validate keu-value datatype value is present or not, to validate the JSON schema we use JsonSchemaValidator class to matchesJsonSchema method by passing json schema location and then we generate response logs in console using log().all() method

Then next we have created Test Runner class in source test java, inside this class we have used @CucumerOptions annotation in class level and we have extended the AbstractTestNGCucumberTest class

And inside the @CucumberOptions annotation we have different types of configurations as like feature file location using features keywords and then we mention Step API definition file location using glue keyword, we check mapping between feature file steps vs implemented steps inside the step API definition file using dryRun keyword , them we have mention tags keyword to run the specific feature scenarios, tags keyword we use to run the specific scenario in feature files then we have mentions plugins to generate the extent reports.

Then we have **created source main resource** files inside the this source folder we have created config.properties file

Inside the config.properties file we have mentioned base uri , api username, api password, then we store the generated access token, all frequent changes test data we store in config file.

Then next we have created **source test resource package** and inside this package we have created request payload folder, json schema folder and extent-config.properties file.

Inside the request payload folder we have create different request json payload with dot json extension and inside the json file we have stored the test data in json node format and then we read this json node values inside the reader packages which we already discussed.

Then we have created json schema validator folder and inside this folder we have stored all the json schema as per the resource endpoints wise and by using json schema we validate the response body datatype structure for HTTP request in step API definition package.

And then next we have created extent-config.properties file and inside the extent-config.properties file we have mentioned the configuration details of extent reports, as like reports store location using baseFolder.name we store extent reports in desired location, to start the generating HTML reports we use extent.reporter.spark.start and true keyword and to stop generating reports extent.reporter.spark.out = reports name, and start generating reports we use extent.reporter.pdf.start=true keyword and to stop generating reports we use extent.reporter.pdf.start=report location with extension.

And next we have created reports folder to maintain the all HTML and PDF reports.