**POSTMAN**

**Validation – Scripts**

pm.test("",function()

{

**pm.response.to.have.status(200); //TO VALIDATE THE STATUS CODE**

**pm.response.to.have.status("OK"); //TO VALIDATE THE STATUS MSG**

**pm.expect(pm.response.responseTime).to.be.below(1000); / .above(1000) //TO VALIDATE THE RESPONSE TIME**

**pm.response.to.have.header("Connection"); //TO VALIDATE IF HEADER HAS A SPECIFIC KEY**

**pm.response.to.be.header("Keep-Alive","timeout=60"); //TO VALIDATE IF THE HEADER KEY POSSESSESS THE SPECIFIED VALUE**

}  );

pm.test("TitleName", function () {

**var jsonData = pm.response.json();**

**pm.expect(jsonData[index].keyName).to.eql("value");**

});

pm.test("titlename, () => {

**pm.expect(pm.response.headers.get('Location')).to.eql(“abcd”);**

});

How to set Variables?

**var body = JSON.parse(responseBody);**

**pm.globals.set("variable\_key", body[0].projectId);**

**pm.environment.set("variable\_key", body[0].projectId);**

**pm.collectionVariables.set("variable\_key", body[0].projectId);**

**REST ASSURED**

|  |  |
| --- | --- |
| **Status Code** | **Description** |
| 100 Continue | An interim response. Indicates to the client that the initial part of the request has been received and has not yet been  rejected by the server. The client SHOULD continue by sending the remainder of the request or, if the request has already been completed, ignore this response. The server MUST send a final response after the request has been completed. |
| 101 Switching Protocol | Sent in response to an Upgrade request header from the client, and indicates the protocol the server is switching to. |
| 102 Processing (WebDAV) | Indicates that the server has received and is processing the request, but no response is available yet. |
| 103 Early Hints | Primarily intended to be used with the Link header. It suggests the user agent start preloading the resources while the server  prepares a final response. |
|  |  |
|  |  |
| 200 OK | Indicates that the request has succeeded. |
| 201 Created | Indicates that the request has succeeded and a new resource has been created as a result. |
| 202 Accepted | Indicates that the request has been received but not completed yet. It is typically used in log running requests  and batch processing. |
| 203 Non-Authoritative Information | Indicates that the returned metainformation in the entity-header is not the definitive set as available from the origin server,  but is gathered from a local or a third-party copy. The set presented MAY be a subset or superset of the original version. |
| 204 No Content | The server has fulfilled the request but does not need to return a response body.  The server may return the updated meta information. |
|  |  |
| 205 Reset Content | Indicates the client to reset the document which sent this request. |
| 206 Partial Content | It is used when the Range header is sent from the client to request only part of a resource. |
| 207 Multi-Status (WebDAV) | An indicator to a client that multiple operations happened, and that the status for each operation can be found in the body of  the response. |
| 208 Already Reported (WebDAV) | Allows a client to tell the server that the same resource (with the same binding) was mentioned earlier. It never appears as a  true HTTP response code in the status line, and only appears in bodies. |
| 226 IM Used | The server has fulfilled a GET request for the resource, and the response is a representation of the result of  one or more instance-manipulations applied to the current instance. |
| 300 Multiple Choices | The request has more than one possible response. The user-agent or user should choose one of them. |
| 301 Moved Permanently | The URL of the requested resource has been changed permanently. The new URL is given by the Location header field in the response. This response is cacheable unless indicated otherwise. |
| 302 Found | The URL of the requested resource has been changed temporarily. The new URL is given by the Location field in the response.  This response is only cacheable if indicated by a Cache-Control or Expires header field. |
| 303 See Other | The response can be found under a different URI and SHOULD be retrieved using a GET method on that resource. |
| 304 Not Modified | Indicates the client that the response has not been modified, so the client can continue to use the same cached version of the response. |
| 305 Use Proxy (Deprecated) | Indicates that a requested response must be accessed by a proxy. |
| 306 (Unused) | It is a reserved status code and is not used anymore. |
| 307 Temporary Redirect | Indicates the client to get the requested resource at another URI with same method that was used in the prior request.  It is similar to 302 Found with one exception that the same HTTP method will be used that was used in the prior request. |
| 308 Permanent Redirect (experimental) | Indicates that the resource is now permanently located at another URI, specified by the Location header.  It is similar to 301 Moved Permanently with one exception that the same HTTP method  will be used that was used in the prior request. |
| 400 Bad Request | The request could not be understood by the server due to incorrect syntax.  The client SHOULD NOT repeat the request without modifications. |
| 401 Unauthorized | Indicates that the request requires user authentication information.  The client MAY repeat the request with a suitable Authorization header field |
| 402 Payment Required (Experimental) | Reserved for future use. It is aimed for using in the digital payment systems. |
| 403 Forbidden | Unauthorized request. The client does not have access rights to the content.  Unlike 401, the client’s identity is known to the server. |
| 404 Not Found | The server can not find the requested resource. |
| 405 Method Not Allowed | The request HTTP method is known by the server but has been disabled and cannot be used for that resource. |
| 406 Not Acceptable | The server doesn’t find any content that conforms to the criteria given by the user agent in the Accept header  sent in the request. |
| 407 Proxy Authentication Required | Indicates that the client must first authenticate itself with the proxy. |
| 408 Request Timeout | Indicates that the server did not receive a complete request from the client within the server’s allotted timeout period. |
| 409 Conflict | The request could not be completed due to a conflict with the current state of the resource. |
| 410 Gone | The requested resource is no longer available at the server. |
| 411 Length Required | The server refuses to accept the request without a defined Content- Length. The client MAY repeat the request if it adds a valid Content-Length header field. |
| 412 Precondition Failed | The client has indicated preconditions in its headers which the server does not meet. |
| 413 Request Entity Too Large | Request entity is larger than limits defined by server. |
| 414 Request-URI Too Long | The URI requested by the client is longer than the server can interpret. |
| 415 Unsupported Media Type | The media-type in Content-type of the request is not supported by the server. |
| 416 Requested Range Not Satisfiable | The range specified by the Range header field in the request can’t be fulfilled. |
| 417 Expectation Failed | The expectation indicated by the Expect request header field can’t be met by the server. |
| 418 I’m a teapot (RFC 2324) | It was defined as April’s lool joke and is not expected to be implemented by actual HTTP servers. (RFC 2324) |
| 420 Enhance Your Calm (Twitter) | Returned by the Twitter Search and Trends API when the client is being rate limited. |
| 422 Unprocessable Entity (WebDAV) | The server understands the content type and syntax of the request entity, but still server is unable to process the  request for some reason. |
| 423 Locked (WebDAV) | The resource that is being accessed is locked. |
| 424 Failed Dependency (WebDAV) | The request failed due to failure of a previous request. |
| 425 Too Early (WebDAV) | Indicates that the server is unwilling to risk processing a request that might be replayed. |
| 426 Upgrade Required | The server refuses to perform the request. The server will process the request after the client upgrades to a different protocol. |
| 428 Precondition Required | The origin server requires the request to be conditional. |
| 429 Too Many Requests | The user has sent too many requests in a given amount of time (“rate limiting”). |
| 431 Request Header Fields Too Large | The server is unwilling to process the request because its header fields are too large. |
| 444 No Response (Nginx) | The Nginx server returns no information to the client and closes the connection. |
| 449 Retry With (Microsoft) | The request should be retried after performing the appropriate action. |
| 450 Blocked by Windows Parental Controls (Microsoft) | Windows Parental Controls are turned on and are blocking access to the given webpage. |
| 451 Unavailable For Legal Reasons | The user-agent requested a resource that cannot legally be provided. |
| 499 Client Closed Request (Nginx) | The connection is closed by the client while HTTP server is processing its request, making the server unable to send the  HTTP header back. |
| 500 Internal Server Error | The server encountered an unexpected condition that prevented it from fulfilling the request. |
| 501 Not Implemented | The HTTP method is not supported by the server and cannot be handled. |
| 502 Bad Gateway | The server got an invalid response while working as a gateway to get the response needed to handle the request. |
| 503 Service Unavailable | The server is not ready to handle the request. |
| 504 Gateway Timeout | The server is acting as a gateway and cannot get a response in time for a request. |
| 505 HTTP Version Not Supported (Experimental) | The HTTP version used in the request is not supported by the server. |
| 506 Variant Also Negotiates (Experimental) | Indicates that the server has an internal configuration error: the chosen variant resource is configured to engage in transparent content negotiation itself, and is therefore not a proper endpoint in the negotiation process. |
| 507 Insufficient Storage (WebDAV) | The method could not be performed on the resource because the server is unable to store the representation needed to  successfully complete the request. |
| 508 Loop Detected (WebDAV) | The server detected an infinite loop while processing the request. |
| 510 Not Extended | Further extensions to the request are required for the server to fulfill it. |
| 511 Network Authentication Required | Indicates that the client needs to authenticate to gain network access. |

**AUTHENTICATION**

given().**auth().basic(“username”,”password”).**when().get(“<http://localhost:8080>”).log().all(); |Credentials are encoded – can be hacked but **not visible** in console

given().**auth().digest(“username”,”password”)**.when().get(“<http://localhost:8080>”).log().all(); |Credentials are encoded and encrypted using MD5 protocol | **Not visible** in console

given().**auth().preemptive().basic(“username”,”password”)**.when().get(“<http://localhost:8080>”).log().all(); |Credentials are encrypted using Base64 of Rest Assured : Less secure – can be decrypted | **Visible** in console

given().**auth().oauth2(“token”).**when().get(“<http://localhost:8080>”).log().all(); | For token generation an API is needed : use formParameter to pass client id,client secret and grantType and auth url for token generation | **Not visible** in console

**REQUEST PAYLOAD**

**1.String jbody = "";**

given().contentType(ContentType.JSON).body**(jbody**).when().post("");

**2. File Path ---🡪 save json body in a file as file.json**

File filePath= new File("./path")

given().spec().body(**filePath**).when().post();

**3. file.json**

FileInputStream fis = new FileInputStream("./");

given().spec().body(**fis**).when().post();

**4. Map<String,Object> map = new HashMap<>();**

map.put("","");

map.put("","");

given().spec().body(**map**).when().post();

**5. JSONObject jobj = new JSONObject();**

jobj.put("","");

jobj.put("","");

jobj.put("","");

given().spec().body(**jobj**).when().post();

**6. Pojo pobj = new Pojo("","","");**

given().spec().body(**pobj**).when().post();

**RESPONSE – EXTRACT DATA**

1. resp.getStatusCode()
2. resp.getStatusLine()
3. resp.getTime()
4. resp.getTimeIn(TimeUnit.SECONDS)
5. resp.getContentType()
6. resp.getBody().asString() 🡪 retunrs the entire response body as string
7. resp.jsonPath().getString(“key”) 🡪 returns string value of key
8. resp.jsonPath().getInt(“key”) 🡪 returns int value of key | get(“content[0].projectName”)
9. resp.path(“key”) 🡪RA automatically detects xml or jsonpath and returns value
10. resp.xmlPath.getString(“key”)
11. JsonPath.read(resp.asString(),”content[\*].[?(@.key1==’value1’)].key2”) 🡪 returns value of key2 when key1 has value1
12. List<String> l=JsonPath.read(resp.asString(), “content[\*].key”) 🡪returns list of values of key if it has multiple values
13. resp.getHeader(“key”) -🡪returns value of the key
14. List<Header> list=resp.getHeaders().asList(); 🡪 returns list of headers

for(Header h: list)

{

Sysout(h);

}

**RESPONSE – DATA VALIDATION**

1. resp.then().assertThat().statusCode(200)
2. resp.then().assertThat().statusLine(“”)
3. resp.then().assertThat().contentType(ContentType.JSON)
4. resp.then().assertThat().header(“key”,”expData”) || value of key is compared with expData
5. resp.then().assertThat().time(Matchers.lessThan(2000L));
6. resp.then().assertThat().body(“content[0].arrayName”,Matchers.hasItem(“cooking”))
7. resp.then().assertThat().body(“content[0].projectName”, Matchers.equalTo(“expName”)
8. resp.then().assertThat().body(“[0].name”,Matchers.equalTo(“abc”)).and().body(“[0].org”,Matchers.equalTo(“xyz”))

Note : we can assert even without using assertThat()

**PARAMETERS**

1. **formParam** (parameter is hidden)

given().formParam(“teamSize”,10).when().**get**(“URI”);

Console : Request URI: <http://49.249.28.218:8091/project> | **parameter filter query is not visible**

given().formParam(“client\_id”,”abc”).formParam(“client\_secret”,”abc”).formParam(“grant\_type”,”xyz”).formParam(“grant\_type”,”mno”).when().**post**(“authTokenURL”);

Console : **Not visible**

1. **queryParam** (for filtering query | visible in uri

given().queryParam(“teamSize”,10).when().get(“uri”);

Console : Request URI: <http://49.249.28.218:8091/project?teamSize=10>

Returns response body with teamSize=10 | For filtering

1. **param |** get – acts as queryParam , post- acts as formParam

**given().param(“teamSize”,10).when().get(“uri”)**

console : Request URI: [**http://49.249.28.218:8091/project?teamSize=5**](http://49.249.28.218:8091/project?teamSize=5)

**given().param(“teamSize”, 10).when().post(“uri”)**

console: Request URI: **http://49.249.28.218:8091/project**

1. **pathParam()** | use a variable instead of displaying data directly

given().**pathParam(“Variable”,”value”).**when().post(“http://baseURL/{Variable})

console : Request URI: [**http://49.249.28.218:8091/project/value**](http://49.249.28.218:8091/project/value)

**FILTER INTERFACE**

FILTER INTERFACE – Base interface to modify request and response of a http communication

**void filter(RequestSpecification requestSpec, ResponseSpecification responseSpec, FilterContext filterContext);**

REQUEST INTERFACE (Extends Filter Interface) – to modify request

**void filter(RequestSpecification requestSpec, FilterContext filterContext);**

RESPONSE INTERFACE (Extends Filter Interface) – to modify response

**void filter(RequestSpecification requestSpec, Response response, FilterContext filterContext);**

FilterContext Interface - It provides access to the next filter in the chain and the current request and response objects.

**DATA SIMULATOR**

**HOW TO PASS PATH IN RUN TIME?**

\* Passing the path in Run time

\* Stmt : System.out.println(args[0]);

\* RC - Run As - Run Configuration - Add Argument

\* -paste the path - Apply - Run

\* ===>path should be displayed in Console

How to convert Data Simulator to .jar file?

============================================

1. Ensure JRE is V 1.8 or above

2. Rest Assured and apache poi versions should be latest (apache poi - 5.2.5 | Rest Assured - 5.2.0)

3. Get the path from run time using arg[0]

4. RC project - export - Expand java and select Runnable JAR File

-First dropdown select the class under use

-Second - select the location to store the .jar file

-Library handling - Select Package required libraries into generated JAR

-Finish

How to run the .jar file?

=====================

open command prompt from the location where the jar file is present

>java -jar (dragAndDrop the jar file) (dragAndDrop the excel file with data) |Enter

**json server By Noushad**

========================

cmd

>npm install -g json-server

added \* packages in \*s

>json-server dragndrop json file |Enter

Serve launched and generates a URL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Open Postman - use the URL from command prompt to send Request

Data from JSON will reflect in the body

**WAYS to perform Serialization and Deserialization**

1. Gson from google
2. Jackson-Databind from faster-xml

JACKSON SERIALIZATION (POJO – JSON)

PojoClass pobj = new PojoClass(“”,””,…);

ObjectMapper omap = new ObjectMapper();

Omap.writeValue(new File(“./Filename.json”), pobj);

JACKSON DESERIALIZATION (POJO – JSON)

ObjectMapper omap = new ObjectMapper();

PojoClass pobj=(Poomap.readValue();

**JSON SCHEMA VALIDATOR**

**https://www.liquid-technologies.com/online-json-to-schema-converters**

Copy/paste the json body in the above link and copy the schema

Eclipse-RC project – new File – paste the schema and save with .json extension

@test pvm(){

File f=new File("./jsonSchema.json");

RestAssured.*given*(). when().get("https://reqres.in/api/users/2").

then().body(**JsonSchemaValidator.*matchesJsonSchema*(f)**). log().all();

}

**Additional logging feature in Rest Assured**

\*\*\*\* Enable logging of request and response details\*\*\*\*

RestAssured.*config* = RestAssured.*config*().logConfig(LogConfig.*logConfig*().enableLoggingOfRequestAndResponseIfValidationFails());

|  |  |  |  |
| --- | --- | --- | --- |
| node.fieldNames(); |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |