**API TESTING**

**WHAT IS API**

- API ==> Application Programming Interface

- API is an interface or communication protocol between different applications or parts of same application.

- API is the engine of application under the hood.

- Normally we have a physical cable to connect our computer to internet at home (in past times). API is a kind of cable that connects application to application.

- Human being has different type of communication ways to speak with each other such as speaking (in different languages), mimics, body language, face gestures. In classroom we talk in English because that is the only way to provide a common communication channel in classroom. API has the same issue. There are some types of API.

- API types:

- .jar files

- DDL (Dynamic Link Library)

- Operating Systems

- Web Services

- There are two types of web service. It might be based on REST or SOAP.

- For example; Selenium WebDriver is an .jar type API. It enables us to talk with browsers. It is a communication channel. JDBC library is also .jar type API which enables us to communicate with databases.

**WEB SERVICES**

- Web Service is a program that works over the Internet and does something useful. This program/application can be built on java/python/ruby/etc.

- Web service is just another form of API.

- To be able to communicate with that application, we need some kind of API. There are two main types of web services ==> REST and SOAP

- Difference is huge. If REST is more like architecture, means that it defines some conventions for building web services where SOAP is a protocol itself (based on HTTP).

- REST is not secure, so data can be easily hacked. So, that is why some companies such as banks use SOAP since it is more secured.

- SOAP based web services use JAX-WS java API as security system.

- RESTful based web services use JAX-RS java API as security system.

- HTTP protocol used everywhere, nowadays.

- HTTP vs HTTPS ==> HTTPS is more secured version of the HTTP. It is using SSL (Secure Sockets Layer) certificate to encrypt information.

- Web Service might be a program which retrieves some information from another application’s web services.

- For example; www.kayak.com ==> it shows you the flight tickets/price/etc. How can it do that? Can it access to all database of the flight companies? No, I don't think so... It uses APIs to interact with the all flight companies, not only kayak.com but also the other web applications which are similar to kayak.com, do the same thing.

- In database there is not any process about data, it is only a storage. Everything about data is being performed in web service (business) layer.

- Web application consist of 3 main components;

- UI is first layer (Presentational Layer)

- Webservice is second layer (Business Layer)

- Database storage is third level (Data Layer)

--> Request --> --> Request -->

Browser ---------------------------------> Web Service (API) ---------------------------------> Database

<-- Response <-- <-- Response <--

- HTTP ==> Hypertext Transfer Protocol

- HTTP is a set of rules for transferring files such as text, video, image, etc.

- Like all languages, http has also some rules.

- HTTP Format ==> scheme://host:port/path/query?key=value& key=value&...

https://google.com/search?q=amazon

- scheme: HTTP or HTTPS

- host: google.com

- port: search

- name="q" ==> whatever you enter into this input box, that value will be used as query parameter.

- URL(Uniform Resource Locator) ==> https://www.google.com

- URN(Uniform Resource Name) ==> www.google.com/index.html

- URI(Uniform Resource Identifier) ==> https://www.google.com/index.html

- When we are doing something with API, it means that we are skipping UI and directly get the data/info from Web Services.

- Big companies use database in a very simple way. Because SQL is very slow languages. It doesn't have OOP concept. So, instead of doing manipulation in database, companies would rather to do in Web Services. Because they use Java, Python, C++ in web service layer and those are much faster than SQL.

- JDBC is an .jar type API. It is enables Java to interact with the Oracle Database.

- Applications such as Uber and Waze pay some money to google and get the permission to access google maps API to use google maps in their own application. This communication is provided by API.

- Or Google uses other newspapers to publish news on Google news. Google simply connects to these news sources (CNN, Fox, New York Times) by using API.

- Or we can login to eBay by using our account information or through Facebook/Google account. Because eBay uses the API to interact with the Facebook/Google.

- Web Service doesn't talk only with the UI and DB but also talks with other companies' APIs.

- %70 of the web services runs based on Rest API. Rest is much better because, information that will be sent in Rest API is mostly in json format. Json mostly works like map interface in Java. It uses key and value configuration.

{

"name": "John",

"age": "45", ==> json format

"DOB": "01/01/1975",

}

- JSON: **J**ava**S**cript **O**bject **N**otation

\* REST (**RE**presentational **S**tate **T**ransfer):

- It is an architecture style. It tells how to build web service.

- Stateless ==> ?

- Requires to use HTTP

- Uses mostly JSON but can also use XML, HTML, CSV, YAML, plain text, etc.

- Documentation is easy to understand

- More efficient and faster

- It requires less bandwidth to transfer message than SOAP.

- Less secure

- REST based web services use JAX-RS java API as security system.

- JAX-RS API enables developers to rapidly build Web applications in Java according to the REST architectural pattern. JAX-RS uses [annotations](https://en.wikipedia.org/wiki/Java_annotation), such as @Path, @Get, @Put, @Delete, @PathParam, @QueryParam to simplify the development and deployment of web service clients and endpoints.

\* SOAP (Simple Object Access Protocol):

- It is an XML based message protocol

- State or Stateless ==> ?

- Language, platform and transport independent, can use SMTP(Simple Mailing Transfer Protocol), FTP(File Transfer Protocol) or etc. (REST requires to use HTTP)

- Uses only XML

- Documentation is complex and hard to understand

- Slower than REST

- More secure

- SOAP based web services use JAX-WS Java API as security system.

- SOAP is like XML based message protocol with strictly defined message structure, authorization mechanism and required documentation. It looks like HTML because both use XML format.

- Learning SOAP is much harder than REST because SOAP requires specific message structure.

- SOAP more secure, more stable but heavier(slower). It requires more bandwidth to transfer message than REST.

In SQL:

- Select ==> get data from data base

- Insert - add new record

- Update - update existing record

- Delete - delete existing record

In API:

- Get - get information from server

- Put - update some record

- Post - add new record into application

- Delete - delete some records

- Whenever we make request to the server, we need to tell what we want to do: delete, update, insert or get.

- Where you can find information about what you can do with web service?

- Usually every application has API documentation. For example: swagger - tool that can generate API documentation. API documentation has list of end points type of request.

We are performing GET request to the api.zippopotam.us server.

api.zippopotam.us : base url

/country/postal\_code : end point

http://api.zippopotam.us/us/22043

This is what we got as a response from the server. This response is in a form of json file:

{

"post code": "22043",

"country": "United States",

"country abbreviation": "US",

"places": [

{

"place name": "Falls Church",

"longitude": "-77.1895",

"state": "Virginia",

"state abbreviation": "VA",

"latitude": "38.8994"

}

]

}

- Test Case example of BDD scenario for API Testing:

Scenario: Verify that place name for postal code 22043 is Falls Church

Given accept type is “application/json”

And pathParam is “/22043”

When user performs "GET" request to "http://api.zippopotam.us/us"

Then user verifies that status code is "200"

And user verifies that "place name" is "Falls Church"

- Test Case example of BDD scenario for API Testing:

Scenario: Verify that department\_id is 60

Given accept type is “application/json”

And pathParam is “/107”

When user performs "GET" request to "http://xxxx.com:1000/ords/hr/employees"

Then user verifies that status code is "200"

And user verifies that department\_id is "60

- To do same thing like we did with Selenium + Cucumber, we are going to use Rest Assured Library.

- Postman, SOAPUI, CURL, Rest Assured (for Java), Native Java HTTP Client ==> Tools that you can use for API automation

- Postman and SAOPUI are mostly used for manual API testing however they are able to perform automation too.

- HTTP status code describes how went your request.

- API testing scenarios:

Valid request:

- body

- query params

- header

- authorization

**\* Most Common HTTP Status Codes:**

- 200 OK

- 201 Created

- 300 Multiple Choices

- 301 Moved Permanently

- 302 Found

- 304 Not Modified

- 307 Temporary Redirect

- 400 Bad Request

- 401 Unauthorized

- 403 Forbidden

- 404 Not Found

- 410 Gone

- 500 Internal Server Error

- 501 Not Implemented

- 503 Service Unavailable

- 550 Permission denied

**- You can learn STATUS CODES from ==> httpstatuses.com**

HTTP Status Code - 200 OK: The request has succeeded. The information returned with the response is dependent on the method used in the request.

HTTP Status Code - 201 Created: The request has been fulfilled and has resulted in one or more new resources being created.

HTTP Status Code - 300 Multiple Choices: The requested resource has different choices and cannot be resolved into one. For example, there may be several index.html pages depending on which language is wanted (such as Dutch).

HTTP Status Code - 301 Moved Permanently: The requested resource has been assigned a new permanent URI and any future references to this resource should use one of the returned URIs.

HTTP Status Code - 302 Found: The requested resource resides temporarily under a different URI. Since the redirection might be altered on occasion, the client SHOULD continue to use the Request-URI for future requests.

HTTP Status Code - 304 Not Modified: the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server SHOULD respond with this status code.

HTTP Status Code - 307 Temporary Redirect: The requested resource resides temporarily under a different URI. Since the redirection MAY be altered on occasion, the client SHOULD continue to use the Request-URI for future requests.

HTTP Status Code - 400 Bad Request: The request could not be understood by the server due to malformed syntax. The client SHOULD NOT repeat the request without modifications.

HTTP Status Code - 401 Unauthorized: The request requires user authentication. The response MUST include a WWW-Authenticate header field containing a challenge applicable to the requested resource.

HTTP Status Code - 403 Forbidden: The server understood the request, but is refusing to fulfill it. Authorization will not help and the request SHOULD NOT be repeated.

HTTP Status Code - 404 Not Found: The server has not found anything matching the Request-URI. No indication is given of whether the condition is temporary or permanent.

HTTP Status Code - 410 Gone: The requested resource is no longer available at the server and no forwarding address is known.

HTTP Status Code - 500 Internal Server Error: server encountered an unexpected condition which prevented it from fulfilling the request.

HTTP Status Code - 501 Not Implemented: The server does not support the functionality required to fulfill the request.

HTTP Status Code - 503 Service Unavailable: Your web server is unable to handle your HTTP request at the time. There is a myriad of reasons why this can occur but the most common are:

- server crash

- server maintenance

- server overload

- server maliciously being attacked

- a website has used up its allotted bandwidth

- server may be forbidden to return the requested document

HTTP Status Code - 550 Permission Denied: The server is stating the account you have currently logged in as does not have permission to perform the action you are attempting. You may be trying to upload to the wrong directory or trying to delete a file.

\*\*\* Karate API is very easy and basic API tool based on Cucumber. You don't have to know coding at all, tool implements codes internally. You have to just send URL, end points and some basic scenarios. You cannot build complex framework with Karate API tool.

- when you send this request from Postman it retrieves employees table from HR data base:

http://ec2-54-146-141-21.compute-1.amazonaws.com:1000/ords/hr/employees

- Actually it retrieves the data from ORDS server which is the Rest API web service that Oracle provides for practice purpose. It is the same as "select \* from employees".

- To develop API automation framework, we can use Rest Assured library. We use Selenium for UI test automation. Rest Assured we are going to use for API automation (only for Rest web services not for SOAP).

- Restful web services use json to transfer data. Json is very light to work with it.

- Web service that is based on Rest architecture is called Restful web service.

- Rest architecture based on HTTP protocol.

**WHAT IS REST?**

- REST is acronym for **RE**presentational **S**tate **T**ransfer. It is architectural style for distributed hypermedia systems and was first presented by Roy Fielding in 2000 in his famous dissertation (Links to an external site.). In simple words, it's an architecture for web services development.

- Like any other architectural style, REST also have its own guiding constraints (Links to an external site) which must be satisfied if an interface needs to be referred as RESTful. These principles are listed below.

- Guiding Principles of REST

- Client–server – By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.

- Stateless – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.

- Cacheable – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.

- Uniform interface – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. **REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and hypermedia as the engine of application state.**

- Layered system – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot “see” beyond the immediate layer with which they are interacting.

- Code on demand (optional) – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

- REST and HTTP are not same!!!

- A lot of people prefer to compare HTTP with REST. REST and HTTP are not same.

- Though, because REST also intends to make the web (Internet) more streamline and standard, he advocates using REST principles more strictly. And that’s from where people try to start comparing REST with web (HTTP). Roy fielding, in his dissertation, nowhere mentioned any implementation directive – including any protocol preference and HTTP. Till the time, you are honoring the 6 guiding principles of REST, you can call your interface RESTful.

- In simplest words, in the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs). The resources are acted upon by using a set of simple, well-defined operations. The clients and servers exchange representations of resources by using a standardized interface and protocol – typically HTTP.

**WHAT IS SOAP?**

- SOAP stands for **S**imple **O**bject **A**ccess **P**rotocol

- SOAP is an application communication protocol

- SOAP is a format for sending and receiving messages

- SOAP is platform independent

- SOAP is based on XML (

What is XML?

XML stands for e**X**tensible **M**arkup **L**anguage

XML is a markup language much like HTML

XML was designed to store and transport data

XML was designed to be self-descriptive

XML is a W3C Recommendation

)

- SOAP is a W3C recommendation (W3C stands for The World Wide Web Consortium (W3C) is an international community that develops open standards to ensure the long-term growth of the Web.)

- SOAP is a standard communication protocol system that permits processes using different operating systems like Linux and Windows to communicate via HTTP (Links to an external site.) and its XML (Links to an external site.). SOAP based APIs are designed to create, recover, update and delete records like accounts, passwords, leads, and custom objects.

- These offers over twenty different kinds of calls that make it easy for the API developers to maintain their accounts, perform accurate searches and much more. These can then be used with all those languages that support web services.

- SOAP APIs take the advantages of making web-based protocols such as HTTP and its XML that are already operating the all operating systems that are why its developers can easily manipulate web services and get responses without caring about language and platforms at all.

**Differences:**

- REST API has no official standard at all because it is an architectural style. SOAP API, on the other hand, has an official standard because it is a protocol.

- REST APIs uses multiple standards like HTTP, JSON, URL, and XML while SOAP APIs is largely based on HTTP and XML.

- As REST API deploys multiple standards, so it takes fewer resources and bandwidth as compared to SOAP that uses XML for the creation of Payload and results in the large sized file.

- The ways both APIs exposes the business logics are also different. REST API takes advantage of URL exposure like @path("/WeatherService") while SOAP API use of services interfaces like @WebService.

- SOAP API defines too many standards, and its implementer implements the things in a standard way only. In the case of miscommunication from service, the result will be the error. REST API, on the other hand, don't make emphasis on too many standards and results in corrupt API in the end.

- REST API uses Web Application Description Language, and SOAP API used Web Services Description language for describing the functionalities being offered by web services.

- REST APIs are more convenient with JavaScript and can be implemented easily as well. SOAP APIs are also convenient with JavaScript but don't support for greater implementation.

**API TESTING**

- What we are going to do exactly about API Testing? ==> We will perform Web Service Testing rather than API. API is a larger definition and Web Service is more specific definition. If any API is HTTP based, it is called Web Service. However, everybody calls it as API Testing in the market.

- Normally we need an HTTP client which is mostly browser. And browser request some data/information from web service. Our job on API testing is to test whether web services send a valid/true response to the client. In this case we are going to use Postman or Rest Assured Library to simulate as we are sending requests from an HTTP client.

**Approach of API Testing:**

Following points helps the user to do API Testing approach:

1. Understanding the functionality of the API program and clearly define the scope of the program
   * 1. Go to Swagger and read documentation IOT figure out the structure of request and response
     2. Go to UI and try to understand the functionality (Inspect/Network/XHR tabs/Right Click to any query under XHR and Copy and Copy as cURL)
     3. Paste this cURL into Postman and start to play send requests with different types of parameters and authentication
2. Apply testing techniques such as equivalence classes, boundary value analysis, and error guessing and write test cases for the API (try some positive and negative testing)
3. Input Parameters for the API need to be planned and defined appropriately
4. Execute the test cases and compare expected and actual results.

**Set-up of API Test environment**

* API Testing is different than other software testing types as GUI is not available, and yet you are required to setup initial environment that invokes API with a required set of parameters and then finally examines the test result.
* Hence, setting up a testing environment for API testing seems a little complex.
* Database and server should be configured as per the application requirements.
* Once the installation is done, the API Function should be called to check whether that API is working.

**How to do API Testing**

API testing should cover at least following testing methods apart from usual SDLC process

* **Discovery testing:** The test group should manually execute the set of calls documented in the API like verifying that a specific resource exposed by the API can be listed, created and deleted as appropriate
* **Usability testing:**This testing verifies whether the API is functional and user-friendly. And does API integrates well with another platform as well
* **Security testing:**This testing includes what type of authentication is required and whether sensitive data is encrypted over HTTP or both
* **Automated testing:**API testing should culminate in the creation of a set of scripts or a tool that can be used to execute the API regularly
* **Documentation:**The test team has to make sure that the documentation is adequate and provides enough information to interact with the API. Documentation should be a part of the final deliverable

**Best Practices of API Testing:**

* Test cases should be grouped by test category
* On top of each test, you should include the declarations of the APIs being called.
* Parameters selection should be explicitly mentioned in the test case itself
* Prioritize API function calls so that it will be easy for testers to test
* Each test case should be as self-contained and independent from dependencies as possible
* Avoid "test chaining" in your development
* Special care must be taken while handling one-time call functions like - Delete, CloseWindow, etc...
* Call sequencing should be performed and well planned
* To ensure complete test coverage, create test cases for all possible input combinations of the API.

**Types of Bugs that API testing detects**

* Fails to handle error conditions gracefully
* Unused flags
* Missing or duplicate functionality
* Reliability Issues. Difficulty in connecting and getting a response from API.
* Security Issues
* Multi-threading issues
* Performance Issues. API response time is very high.
* Improper errors/warning to a caller
* Incorrect handling of valid argument values
* Response Data is not structured correctly (JSON or XML)

**Challenges of API Testing**

Challenges of API testing includes:

* Main challenges in Web API testing is **Parameter Combination, Parameter Selection, and Call Sequencing**
* There is no GUI available **to test the application which makes** difficult to give input values
* Validating and Verifying the output in a different system is little difficult for testers
* Parameters selection and categorization is required to be known to the testers
* Exception handling function **needs to be tested**
* Coding knowledge is necessary for testers

**Conclusion:**

API consists of a set of classes/functions/procedures which represent the business logic layer. If API is not tested properly, it may cause problems not only the API application but also in the calling application. It is an indispensable test in software engineering.

**RESTFUL API PROCEDURES**

- How do you test restful APIs?

- Firstly, I learn API's available endpoints and how they work,

- Then I send GET, POST, PUT, DELETE type of requests to those endpoints,

- Then I verify API response if it is matching with expected behavior.

- I do positive and negative tests.

- When I verify the API response, I check: status codes, headers and body.

- Where do you keep API reference documentation in your Project?

- We use SWAGGER to keep API endpoints and descriptions. I normally go there and learn about API methods and how they work.

- Why do you think API testing is important?

- API level in the application is developed before UI level, so API testing will enable to start QA process/Testing process earlier in the project

- If APIs are working without bugs, UI functionality will also have less bugs

- What is the role of API testing in your current project?

In our project, all major functionalities of the application are available in UI side, and API side.

So, I do automation testing of the application in both UI and API layers.

API layer also enables external applications to communicate with our application. At the same time, our application sends requests to other company applications for certain functionalities and data exchange.

For UI automation I use -> Java+Selenium WebDriver+Cucumber

For RESTful API automation I use -> Java with Rest-Assured Library.

- How does Rest API testing work?

Each Rest API will have its endpoints, and we can send GET, POST, PUT, DELETE type of requests to those endpoints. Developers store those endpoints in swagger.

- I do positive and negative API test cases automation.

For positive scenarios:

- valid headers

- valid credentials

- valid json body/payload

- valid parameters and values

- valid request type

For negative scenarios:

- invalid path or query parameter values

- invalid headers

- invalid json body while posting.

-> structure might be invalid

-> unacceptable data (unique error etc, invalid keys )

**Restful API General Syntaxes**

- kind of controller that takes your request when you are trying to find employee

@GetMapping(/employees/{id})

Public Employee get EmployeeByID(@PathParam Sting id){

Return repo.getEmployee(id);

}

\*\*\* Path parameter stands for fetching specific resource such as, some city info, user, etc...It points on a specific resource.

\*\*\* Query parameter stands for filtering purpose. It filters results or describe new resources. Out of all users, we need user with last name Bond or user that have masters degree, etc.

Exp-1: POST ==> /users?name=James&age=60&job-title=SDET

Exp-2: GET ===> /employee?name=Jamal --> get all employees with name of Jamal.

- To do test automation of Restful web services

==> /users/100/ ==> 100 is a path parameter

==> /users/255/ ==> 255 is a path parameter

==> /users/255?name=James ==> name is a query parameter (key=value) , key it's a query parameter

public void test(){

given()

.parameters("first\_name", "John")

.when()

.get("/xxxx") ===> General concept of the BDD approach for REST Assured

.then() ===> This kind of pattern is called "Building Pattern"

.body("xxxxx", is("yyyy"))

}

\* org.harmcrest.Matches.\* ==> It provides more way to assert some specific conditions. It works well with Java and JUnit as well...

.assertThat().body("region\_name", is("Europe"))

.assertThat().body("region\_name", is(not("Europe"))

.assertThat().body("region\_id", is(1))

.assertThat().body("region\_id", is(not(1)))

.assertThat().body("region\_id", equalTo(1))

.time(lessThan(1000L), TimeUnit.MILLISECONDS)

- .prettyPrint() ==> only prints body

- .log().body()==> prints only BODY in pretty format

- .log().headers()==> prints only HEADERS in pretty format

- .log().status()==> prints only STATUS CODE in pretty format

- .log().all() ==> prints HEADER + BODY + STATUS CODE

- .extract() method ==>

**API REQUEST**

- What are the types of Requests?

GET -> Read data

POST -> Create Data

PUT -> To Update data

DELETE -> Delete

- Parameters in REST API:

2 Types of Parameters:

-> path/url parameter

-> query parameters

path/url parameter:

value of the parameter is part of URL

/api/spartans/{id}

/api/spartans/5

query parameters:

values are passed as K+V - key and value format after API endpoint.

https://google.com/search?q=automation+jobs+near+me

- What are the parts of GET request?

1. Base URL + API method ----> API Endpoint

2. Parameters:

a. -> PATH/URL params -> values is part of URL

DOC: 54.164.195.86:1000/ords/hr/regions/{id}

ex: 54.164.195.86:1000/ords/hr/regions/1

b. -> QUERY params -> Key+Value

Ex:

www.etsy.com/api/products?prodid=34566&seller=Roman

https://www.google.com/search?q=selenium

3. Headers:

- Accept-Type=Json | Accept-Type=xml

- Authorization tokens, Credentials, Username passwords etc

- Parts of RESPONSE to GET REQUEST?

- status code

- response body in JSON or XML, or any other format

- headers (Content-type=application/json)

- PARTS of POST REQUEST?

-> Base URL + API method <---- API Endpoint

-> Request JSON BODY

-> Headers:

- Accept-Type=Application/Json | Accept-Type=Application/xml

- Content-Type=Application/json

- Authorization tokens, Credentials, Username passwords etc

Accept-Type=Application/Json --> I want response to be in JSON format

Content-Type=Application/json --> I am sending you data in JSON format

- Parts of RESPONSE to POST REQUEST?

-> status code

-> response body in JSON or XML, or any other format

-> headers (Content-type=application/json)

- baseURI ==> It is a static variable in RestAssured class. Once you assign the base URL address to it, you will use .get("/restOfTheEndPoint")

- given, when, then, and ==> same logic as BDD. We can also type our codes without using them.

- get("/employees") ==> gets the data from "baseURI/employees"

- pathParam("employee\_id","101") ==> it means that add "101" after URI in get() method"

Exp:

Response response=given().

queryParam("base", "USD").

get(baseURI+"latest");

- queryParam("q", "shoes") ==> https://www.google.com/search?q=

Exp:

private String baseURI = "https://api.openrates.io/";

Response response=given().

queryParam("start\_at", "2000-01-01").

queryParam("end\_at", "2000-12-31").

queryParam("base", "USD").

queryParam("symbols", "EUR","GBP","JPY").

get(baseURI+"history");

response.prettyPrint();

- baseUri() and basePath() methods

Exp:

private String baseURI = "https://api.openrates.io/";

Response response=given().

baseUri(baseURI).

basePath("latest").

queryParam("base", "USD").

get();

- accept(ContentType.JSON) or accept("application/json") ==> we want response as JSON format

- contentType(ContentType.JSON) ==> we want to post new content as JSON format

- body()/getBody() ==> resturns body as Response type. Use “.asString” if you want to print

- headers()/getHeaders() ==> returns all headers as Response type. Use “.toString” if you want to print

- header("headerName")/getHeader("headerName") ==> returns specified header name as string

- statuscode()/getStatuscode() ==> returns status code as int

- prettyPrint() ==> prints the response body and returns body as String. So you can assign like below:

String responseBody = response.printPretty(); ==> assigns and prints at the same time

- prettyPeek() ==> prints the response body, header and status code in pretty format

- System.out.println(response.getBody().asString()); ==> prints non-pretty format of the body

- If we don't put “.asString()” ==> it will give hash code

\*\*\* If you use prettyPrint(), you cannot add any chain after that. But prettyPeek() allows to add chain after itself.

- Exp:

private String baseURI = "https://api.openrates.io/";

Response response = given().get(baseURI + "latest").prettyPeek();

System.out.println(response.getHeaders().toString());

System.out.println(response.getHeader("Date"));

System.out.println(response.getBody().asString());

response.prettyPrint();

**HAMCREST MATCHERS**

\*\*\* assertThat() is special assertion method which comes from Hamcrest. It is very useful if don't need to store any data in String, List, Map or List of Map. It directly asserts the condition but we cannot use the content of the response.

\*\*\* If we need to use content and store into some type of data structure, we need to use JsonPath and assert then in a regular way by using assertEqual, assertTrue, etc.

\*\*\* We cannot use JsonPath with asserThat(). That is the most important information about Hamcrest. We can assert something using hamcrest without using JsonPath or vice versa.

- Exp:

given()

.accept("application/json")

.when()

.get("/employees")

.then()

.assertThat().statusCode(200)

.and()

.assertThat().contentType("application/json")

.log().ifError()

.log().all(true);

\*\*\* We cannot use hamcrest matchers methods if we don't use ".then()" after ".get()" method.

- After assertThat() we can use:

- assertThat().statusCode(200)

- assertThat().contentType("application/json")

- assertThat().time(lessThan(2000L), TimeUnit.MILLISECONDS)

- assertThat().body("employee\_id", is(100))

- assertThat().body("last\_name", is("King")

- assertThat().body("employee\_id", is(100), "department\_id", is(not(0)), "last\_name", is("King"))

- assertThat().body("items.salary", everyItem(greaterThan(0)))

- assertThat().body("items.location", everyItem(is("xxxxx")))

- assertThat().body("itmes[0]",hasEntry("first\_name", "Jack"))

- assertThat().body("itmes[0]",hasEntry("last\_name", "Taylor"))

Hamcrest comes with a library of useful matchers. Here are some of the most important ones.

Core Matchers

anything - always matches, useful if you don’t care what the object under test is

describedAs - decorator to adding custom failure description

is - decorator to improve readability - see “Sugar”, below

Logical Matchers

allOf - matches if all matchers match, short circuits (like Java &&)

anyOf - matches if any matchers match, short circuits (like Java ||)

not - matches if the wrapped matcher doesn’t match and vice versa

Object Matchers

equalTo - test object equality using Object.equals

hasToString - test Object.toString

instanceOf, isCompatibleType - test type

notNullValue, nullValue - test for null

sameInstance - test object identity

Beans Matchers

hasProperty - test JavaBeans properties

Collections Matchers

array - test an array’s elements against an array of matchers

hasEntry, hasKey, hasValue - test a map contains an entry, key or value

hasItem, hasItems - test a collection contains elements

hasItemInArray - test an array contains an element

Number Matchers

closeTo - test floating point values are close to a given value

greaterThan, greaterThanOrEqualTo, lessThan, lessThanOrEqualTo - test ordering

Text Matchers

equalToIgnoringCase - test string equality ignoring case

equalToIgnoringWhiteSpace - test string equality ignoring differences in runs of whitespace

containsString, endsWith, startsWith - test string matching

**EXTRACTING DATA FROM RESPONSE**

\*\*\* How to navigate through Json and read values from the keys.

- There are multiple different ways of navigating through json body/payload.

Popular ones:

- using path() method

- using JsonPath

- using deserialization to a collection

- using deserialization to a java custom object (POJO)

**NOTE:** JsonPath returns an immutable data. We have to reassign into same data type in order to manipulate the data if want to sort it or something like that.

1. PATH() METHOD

Response response = given()

.accept(ContentType.JSON)

.pathParam("student\_id", 2414)

.when()

.get("/student/{student\_id}").prettyPeek();

String studentName = response.body().path("students[0].firstName");

int studentBatch = response.body().path("students[0].batch");

2. JSONPATH

- JsonPath is an alternative to using XPath for easily getting values from a Object document. It follows the Groovy GPath syntax when getting an object from the document. You can regard it as an alternative to XPath for JSON.

- It allows us to extract all needed data from Json body.

- Exp:

{"store": {

"book": [

{ "category": "reference",

"author": "Nigel Rees",

"title": "Sayings of the Century",

"price": 8.95

},

{ "category": "fiction",

"author": "Evelyn Waugh",

"title": "Sword of Honour",

"price": 12.99

}

],

"bicycle": {

"color": "red",

"price": 19.95

}

}

}

JsonPath json = given()

.accept("application/json")

.when()

.get("/stores")

.thenReturn()

.jsonPath();

- price ==> json.getInt(“store.book[0].price”); ==> it will return an integer

- color ==> json.getString(“store.bicycle.color”); ==> it will return a String

- book ==> json.getList(“store.book”) ==> it will return a List of Map

- bicycle ==> json.getMap(“store.bicycle”) ==> it will return a Map

- store ==> json.getMap(“store”) ==> it will return a Map

3. USING JAVA DATA STRUCTURE/COLLECTIONS

Response Json Data:

{

"year", 2000,

"make", "fiat",

"model", "f500",

"mileage", 50234

}

Map<String, Object> dataMap = response.body().as(Map.class); or

Map<String, Object> dataMap = response.jsonPath().getMap(“”);

dataMap.get("year") => 2000

**VERIFYING RESPONSE DATA**

1) Not recommended:

- treat the response json as a String and do contains assertions on it.

AssertTrue(response.body().asString().contains("Java"));

2) PATH() method.

Extract values from JSON using path() method, use Junit assertions for verification.

String city = response.path("employee.address.city");

assertEquals("New York", city);

3) JSONPATH object:

Convert Response data into JsonPath object and use jsonpath getter methods to extract values.

Do assertions using JUnit

JsonPath json = response.jsonPath();

assertEquals("Coffee", json.getString("drink"));

4) HAMCREST MATCHERS WITH PATH USING CHAINING:

We can do assertions in single statement by chaining methods in Rest-Assured. To find values in the json body, we use the same path syntax:

.then()

.assertThat()

.body("teachers.firstName",contains("Esen"),

"teachers.lastName",contains("Niiazov"),

"teachers.emailAddress",contains("eniiazov@gmail.com"));

5) Java Collections/Data Structures to manipulate Json Data and then verification with this data.

Response response = given()

.accept(ContentType.JSON)

.when()

.get("http://54.164.195.86:8000/api/spartans/33");

Map<String, Object> spartanMap = response.body().as(Map.class);

System.out.println("spartanMap = " + spartanMap.toString());

System.out.println(spartanMap.get("id"));

Now we can easily read the map for values and do assertions, or we can compare with other maps.

**How to use first index and last index from response body?**

First index is basically [0] as you know. However, the important point is about last index.

You can use **index[-1]** to get value from the end of data structure…

String salary\_1 = json.getString("items.salary[0]")**; 🡺 first index­­­­­**

String salary\_2 = json.getString("items.salary[-1]")**; 🡺 last index**

**How to use “find{it.key==value}” and “findAll{it.key==value}” while extracting data?**

int holidays = response.jsonPath().

.getInt("response.countries.find{it.country\_name==\"Germany\"}.total\_holidays")**;**List<String> countries = response.jsonPath()

.get("response.countries.findAll{it.supported\_languages==4}.country\_name")**;**

**SERIALIZATION AND DE-SERIALIZATION.**

I use GSON library for this process.

POJO -> Plain Old Java Object.

I create POJO classes to match json data from API.

By using POJO classes, I can de-serialize Json response into Java pojo class object. Then easily do assertions, or any other data manipulation.

SERIALIZATION:

Java Object >>> Json format

DE-SERIALIZATION:

Json format >>> Java object

We can convert json response body to custom class object or a java data structure.

{

"schoolname":"Cybertek",

"city":"Mclean", 🡺 Json format

"state":"VA"

}

public class School {

String schoolName;

String city; 🡺 POJO object in any POJO Class

String state;

}

**How do you convert from POJO to JSON using Gson: (Serialization)**

1. Create a POJO class

2. Create an object from this class

3. Do the conversion using toJson() method from Gson Class

Job sdet\_pojo = new Job("SDET"**,** "SDET"**, 90\_000, 125\_000**)**;**Gson gson = new Gson()**;**String sdet\_gson = gson.toJson(sdet\_pojo)**;** //==> from POJO to JSON: Serialization

**How to convert from JSON format to POJO: (Deserialization)**

1. Create Response class object. If you want to create only one pojo you can use

Spartan spartan=response.as(Spartan.class); or

Spartan spartan=json.getObject (“items[0]”, Spartan.class);

1. If you want to make a collection of Spartans

List<Spartan> spartan= json.getList(“items”, Spartan.class);

**How to generate request body for Post action:**

1. Spartan spartan=new Spartan(“Male”, “Jack”, 5711234567);
2. Spartan spartan=new Spartan();

spartan.setGender=”Male”;

spartan.setName=”Jack”;

spartan.setPhone=5711234567;

1. Map<String, Object> spartan = new HashMap<>();

spartan.put(“gender”,”Male”);

spartan.put(“name”,”Jack”);

spartan.put(“phone”, 5711234567);

1. File spartan = new File(filePath);

And then pass the spartan object into Post command in request for any kind of object above.

Response response = *given*().  
 contentType(ContentType.*JSON*).  
 body(spartan).  
 when().post("/api/spartans")**;**

**POST ACTIONS**

POST method is used to create data by using API request.

Unlike GET requests, for POST we need to provide JSON body.

Given accept type and Content type is JSON

And request json body is:

{"gender":"Male",

"name":"Maximus",

"phone":8877445596}

When user sends POST request to '/spartans/'

Then status code 201

And content type should be application/json

And json payload/response should contain:

"A Spartan is Born!" message

and same data what is posted

**DATA DRIVEN TESTING WITH JUNIT-5**

Junit-5 provides some tags in order to achieve DDT for API testing frameworks. There are three type of tag for that purpose…

- @ParameterizedTest --> Junit-5 annotations for data driven tests can come from multiple sources  
- Data can come from csv files, methods, enums or @valueSource

- The source is indicated next to the @ParameterizedTest tag

- @ValueSource --> data is given with the test method

@ParameterizedTest  
@ValueSource(ints = {**1,2,3,4**})  
public void validateRegionNameTest1(int id) {

@ParameterizedTest  
@CsvSource({  
 "1, Europe"**,** "2, Americas"**,** "3, Asia"**,** "4, Middle East and Africa"})  
public void validateRegionNameTest2(int id**,** String name) {

@ParameterizedTest  
@CsvFileSource(resources = "/regions.csv")  
public void validateRegionNameTest3(int id**,** String name) {

**AUTHENTICATION AND AUTHORIZATION IN REST API**

AUTHENTICATION -> verifies who you are

-> checks for valid credentials

AUTHORIZATION -> verifies if you have access/permission to certain API resource/action.

401 -> unauthorized 🡪 invalid credentials

403 -> forbidden 🡪 valid credentials but not enough access level

- Basic authorization:

- username, password in the request header

- Other popular type of authorization is OAuth 1.0 or OAuth 2.0

- API key

- API token

- How does API Authentication work in your application?

I send a get request to special API endpoint and provide valid credentials, then it will return Authorization/Access token.

I use that token in my request header and access other API endpoints.

I send request with that access token in the header

**API KEY VS AUTHENTICATION**

- Main rule 🡺 API keys are for projects; authentication is for users.

* API keys identify the calling project (the application or site) making the call to an API.
* Authentication tokens identify a user (the person) that is using the app or site.

- API key schemes are designed to serve two main purposes:

* Project identification: Identify the project or the application making the call to the API.
* Project authorization: To help check whether the application making the call has access to call it. It also checks whether the API in this project is enabled.

- Authentication schemes are designed to serve two main purposes:

* User authentication: verify that the person making the call is the person he or she is claiming to be.
* User authorization: check whether the user making the call has permission to make this kind of request.

- You should note that the API keys are not as secure as the tokens used for authentication purposes. However, they do assist in identifying the project or the application that is behind the call.

**4 Most Used Authentication Methods**

**1. Basic Authentication**

HTTP Basic Authentication is rarely recommended due to its inherent security vulnerabilities.

This is the most straightforward method and the easiest. With this method, the sender places a username:password into the request header. The username and password are encoded with Base64, which is an encoding technique that converts the username and password into a set of 64 characters to ensure safe transmission.

This method does not require cookies, session IDs, login pages, and other such specialty solutions, and because it uses the HTTP header itself, there’s no need to handshakes or other complex response systems.

*given*()  
 .auth().basic("admin"**,**"admin")  
.when()  
 .get("/basic\_auth").prettyPeek();

**2. Bearer Authentication**

Bearer authentication (also called token authentication) is an HTTP authentication scheme that involves security tokens called bearer tokens.

The name “Bearer authentication” can be understood as “give access to the bearer of this token.” The bearer token allowing access to a certain resource or URL and most likely is a cryptic string, usually generated by the server in response to a login request.

The client must send this token in the Authorization header when making requests to protected resources.

**Similarly, to Basic authentication, Bearer authentication should only be used over HTTPS (SSL).**

public String getToken(){  
 Response response =  
 *given*()  
 .queryParam("email"**,** ConfigurationReader.*getProperty*("team.leader.email"))  
 .queryParam("password"**,** ConfigurationReader.*getProperty*("team.leader.password"))  
 .when()  
 .get("/sign").prettyPeek()**;** return response.jsonPath().getString("accessToken")**;**}  
  
@Test  
@DisplayName("Get list of all rooms")  
public void test1() {  
 Response response =  
 *given*()  
 .header("Authorization"**,** getToken())  
 .get("/api/rooms").prettyPeek()**;**

**3. API KEY**

API Keys were created as somewhat of a fix to the early authentication issues of HTTP Basic Authentication and other such systems. In this method, a unique generated value is assigned to each first time user, signifying that the user is known. When the user attempts to re-enter the system, their unique key (sometimes generated from their hardware combination and IP data, and other times randomly generated by the server which knows them) is used to prove that they’re the same user as before.

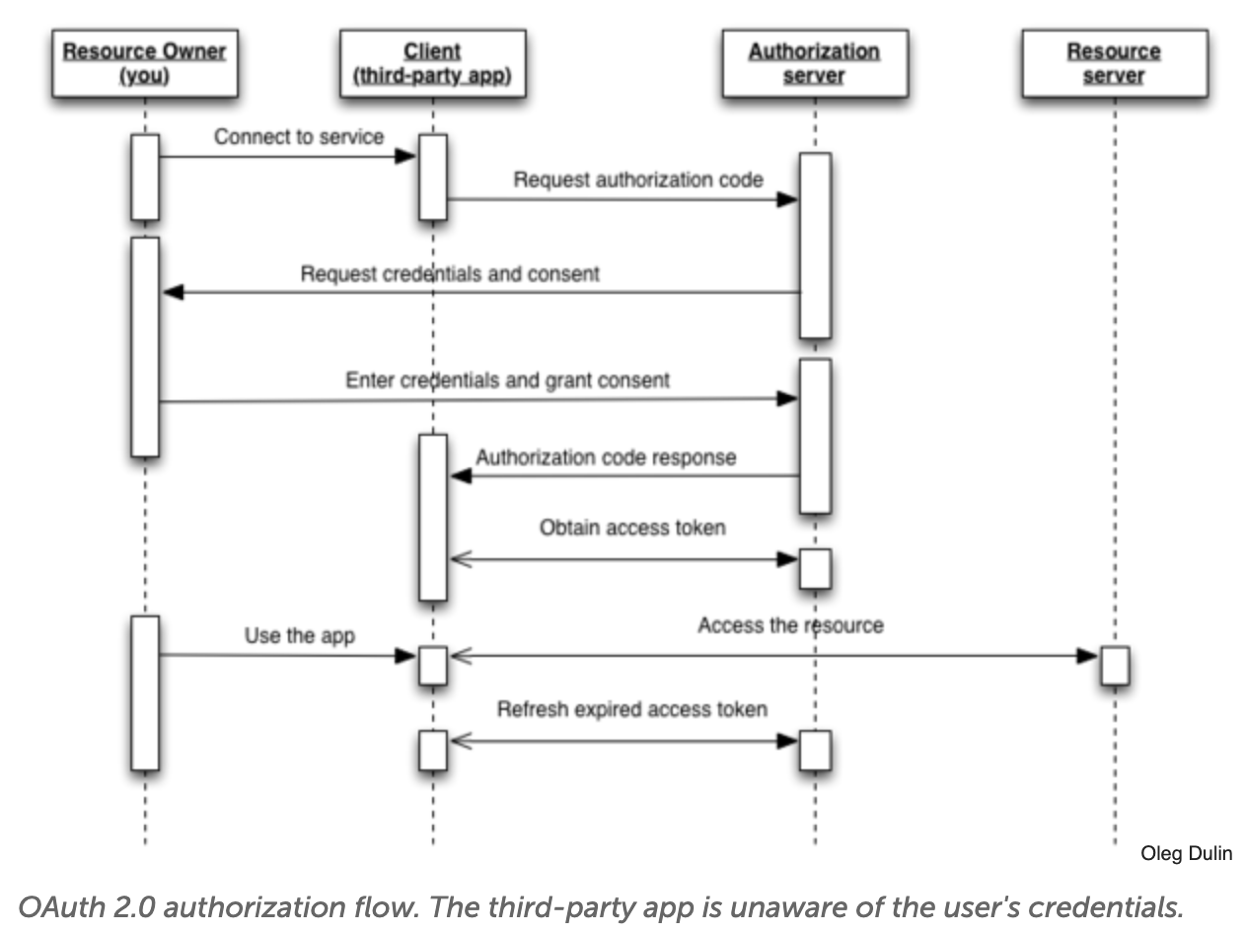
We can send api\_key either as query parameter or in request header.

*given*()  
 .accept(ContentType.*JSON*)  
 .queryParam("apikey"**,** ConfigurationReader.*getProperty*("omdb.apikey"))  
.when()  
 .get()**;**

**4. OAUTH 2.0 (Bearer Token Scheme)**

OAuth 2.0 is the best choice for identifying personal user accounts and granting proper permissions. In this method, the user logs into a system. That system will then request authentication, usually in the form of a token. The user will then forward this request to an authentication server, which will either reject or allow this authentication. From here, the token is provided to the user, and then to the requester. Such a token can then be checked at any time independently of the user by the requester for validation and can be used over time with strictly limited scope and age of validity.

* The most popular authentication protocol/type is OAuth 2.0
* It is based on Bearer token
* It is used by Facebook, Google, Twitter and many more companies.
* We are able to log in some web application with Google or Facebook account through this OAuth 2.0 authentication protocol.
* OAuth 2.0 provides access via tokens. **You give username and password then server generates a token and then with every request you have to carry token**
* OAuth 1.0 is much more secured but it is not easy to implement by developers.
* OAuth 2.0 allows authentication via 3rd party services, without sharing your password. It means that, once you login with your google account into any web site, they will never learn your google password
* Also. There are 1 factor, 2 factors and multi factor authentication. For example; Okta is a 2 factors service. You have to provide credentials, plus confirm push notification on your phone.
* Authentication 🡺 Confirmation of who you are
* Authorization 🡺 Confirmation of what’s your role, what kind of permission you have (admin, user, SDET, Manager, etc.)



First, we need to send username and password to get token.

* public static String getToken() {  
   Response response = *given*().  
   queryParam("email"**,** Environment.*LEADER\_USERNAME*).  
   queryParam("password"**,** Environment.*LEADER\_PASSWORD*).  
   when().  
   get("/sign").prettyPeek()**;** return response.jsonPath().getString("accessToken")**;**}

Then, we can use the token what we extract from fisrt response and send it as OAuth 2.0 as below.

String token = APIUtilities.getToken();

Response response = *given*().  
 accept(contentType).  
 auth().oauth2(token).  
 when().  
 get(path).prettyPeek()**;**

**Preemptive Auth:**

Actually, it will make 2 calls:

1st: with no credentials, then will get 401, in order to ensure that only requested server will get our credentials. If any leaking occurs on the way it will not send credentials on second call. If our request only taken by requested server then 2nd call will be sent with credentials. It is very old fashion authentication way.

*given*()  
 .auth().preemptive().basic("admin"**,** "admin")  
.when().get("/basic\_auth").prettyPeek()  
 .then().assertThat().statusCode(**200**)**;**

**How to Override SSL Certificates (Bad SSL Connection)**

Some APIs require a Certificate to work.

There are 2 Options: HTTP vs HTTPS

HTTPS -> secure HTTP

Anytime you visit a website which is https, secured and encrypted connection will be set up between you and the server.

It means when you type card numbers, SSN, data will not be stolen on the way.

Sometimes APIs also require Certificate.

2 Options:

1. given().**relaxedHTTPSValidation()** is to trust the API without providing Certificate KEY.

2. Provide certificate key 🡺 given().keyStore(KeyStoreFile Path)

Response response=*given*().relaxedHTTPSValidation().when().get().prettyPeek()**;**

//==> This method overrides all the SSL certificates.

// ==> gives SSLHandshakeException without relaxedHTTPSValidation()

**API SCHEMA**

Database Schema: description of database

- table names

- columns names

- relation between tables

Json Schema:

- description of json data

Json Schema example, describing the json data above.

It is a template for the json response, request body.

Exp:

Properties:

"country\_id": required, String, 2 characters, null by default

"country\_name": required, String, up to 100 characters, null be default

"population" : required, Integer, 0 by default