# Rest Assured API Automation Learning

REST Assured is a Java Library for simplifying testing of REST based services built on top of HTTP Builder. It supports POST, GET, PUT, DELETE, HEAD, PATCH and OPTIONS requests and to verify the response of these requests.

**Rest assured is built on top of Appache HTTP client. Sometimes it asked why not to use HTTP client directly?**

If you need to write some kind of general-purpose HTTP Client application, using Appache directly might be better. But if you need to test a webservices, using Rest Assured will better option.

**Advantages of Rest Assured Over HTTP Client:**

1. It requires Less coding.
2. Easy parsing and Validation of response in Json and HTML
3. It follows BDD Keywords like given, When, then which makes code readable
4. Quick assertion
5. Easily integrate with other java libraries like TestNg, junit

**Disadvantages**

1. Does not support testing of SOAP APIs directly.
2. No inbuilt Reports

## Before Begin Rest Assure, we need to know few things.

1. Diff btw API vs Webservices
2. What is URL and URI
3. SOAP Vs REST

### URL VS URI

URL - > It’s the locator of a resource.

URI -> its an Identifier of a resource

### SOAP VS REST:

Soap -> Simple Object Access Protocol

Rest - > Representational State Transfer

Soap only works with XML Formats Whereas Rest work with Xml, Html, Json and Plain Text.

Soap is a protocol Whereas Rest is Architectural pattern. It means Rest doesn’t have any official standards.

### API Vs WebServices

API and Webservices both are used for communication between service providers and service consumers.

The one main difference is Webservice is communicate one application to another application over a network with HTTP Protocol.

All API’s are Webservices and but not all webservices are API’s.

### HTTP Methods

Get – Its used to retrieve the data from api endpoints.

Post – Used to send new data / Create an entity to server.

Put – Replace / Updating existing resource entirely in particular url. If resource is not found on server. It can able to create a new resource.

Patch – Partial update on existing resource.

Delete – Remove existing resource or data at particular url.

### Response Codes:

<https://www.moesif.com/blog/technical/api-design/Which-HTTP-Status-Code-To-Use-For-Every-CRUD-App/>

## Overview of the REST Assured and Available Methods

* Static Import

In order to use REST assured effectively it's recommended to statically import methods from the following classes:

io.restassured.RestAssured.\*

io.restassured.matcher.RestAssuredMatchers.\*

org.hamcrest.Matchers.\*

* Method Chaining

Same return type required

**Json Libraries for Validating responses**

1. Gson -> Google
2. Jackson
3. Json
4. Json Path
5. **Json Simple -> Google**

### Basic Terms and Methods.

**Given:**

Given Keyword lets you to pass the

1. Request Headers
2. Query
3. Path Params
4. Form Parameter
5. Body
6. Cookies
7. Other request properties.

**ReturnType:** RequestSpecification - Interface

**When:**

1. Get /Post / Put /Delete / Patch

**Then:**

Its used to validate the response.

1. Assert and Matcher Condition.

ReturnType: ValidatableResponse -Interface

### Other Methods

* Log
* Header
* Body
* ContentType
* Accept
* StatusCode
* PathParam
* baseUri
* BasePath
* Extract
* asString
* asPrettyString
* Spec
* time
* timeIn
* RequestSpecification (Given)
* ResponseSpecification(expect)
* Matcher

## Required Details to hit the API

1. What is HTTP Method supported?
2. Find URL - > Base URL and Base Path
3. If It’s Post Http Method then Find Payload (Request Body).

Payload information will be available in API documentations.

**(Given)** Build Request

- > **(When)** Hit Request and Get Response

-> **(Then)** Validate Response.

## Code Samples

**get("/lotto").then().body("lotto.lottoId", equalTo(5));**

or perhaps you want to check that the winnerId's are 23 and 54:

**get("/lotto").then().body("lotto.winners.winnerId", hasItems(23, 54));**

Note: equalTo and hasItems are Hamcrest matchers which you should statically import from org.hamcrest.Matchers.

### Json Examples

Example 1

As a first example let's say we want to make the request to "/store" and assert that the titles of the books with a price less than 10 are "Sayings of the Century" and "Moby Dick":

when().

get("/store").

then().

body("store.book.findAll { it.price < 10 }.title", hasItems("Sayings of the Century", "Moby Dick"));

Just as in the XML examples above we use a closure to find all books with a price less than 10 and then return the titles of all the books. We then use the hasItems matcher to assert that the titles are the ones we expect. Using JsonPath we can return the titles instead:

// Get the response body as a String

String response = get("/store").asString();

// And get all books with price < 10 from the response. "from" is statically imported from the JsonPath class

List<String> bookTitles = from(response).getList("store.book.findAll { it.price < 10 }.title");

**Example 2**

Let's consider instead that we want to assert that the sum of the length of all author names are greater than 50. This is a rather complex question to answer and it really shows the strength of closures and Groovy collections. In REST Assured it looks like this:

when().

get("/store");

then().

body("store.book.author.collect { it.length() }.sum()", greaterThan(50));

You could also decide to instead return the entire response if you need to **extract** multiple values from the response:

Response response =

given().

param("param\_name", "param\_value").

when().

get("/title").

then().

contentType(JSON).

body("title", equalTo("My Title")).

**extract**().

response();

String nextTitleLink = response.path("\_links.next.href");

String headerValue = response.header("headerName");

### You can also get headers, cookies, status line and status code:

Response response = get("/lotto");

// Get all headers

Headers allHeaders = response.getHeaders();

// Get a single header value:

String headerName = response.getHeader("headerName");

// Get all cookies as simple name-value pairs

Map<String, String> allCookies = response.getCookies();

// Get a single cookie value:

String cookieValue = response.getCookie("cookieName");

// Get status line

String statusLine = response.getStatusLine();

// Get status code

int statusCode = response.getStatusCode();

### Multi-value headers

To get all values for a header you need to first get the [Headers](http://static.javadoc.io/io.rest-assured/rest-assured/4.3.3/io/restassured/http/Headers.html) object from the [Response](http://static.javadoc.io/io.rest-assured/rest-assured/4.3.3/io/restassured/response/Response.html) object. From the Headers instance you can get all values using the [Headers.getValues( )](http://static.javadoc.io/io.rest-assured/rest-assured/4.3.3/io/restassured/http/Headers.html" \l "getValues(java.lang.String))method which returns a List with all header values.

### Multi-value cookies

To get all values for a cookie you need to first get the [Cookies](http://static.javadoc.io/io.rest-assured/rest-assured/4.3.3/io/restassured/http/Cookies.html) object from the [Response](http://static.javadoc.io/io.rest-assured/rest-assured/4.3.3/io/restassured/response/Response.html) object. From the Cookies instance you can get all values using the [Cookies.getValues()](http://static.javadoc.io/io.rest-assured/rest-assured/4.3.3/io/restassured/http/Cookies.html" \l "getValues(java.lang.String)) method which returns a List with all cookie values.

### Detailed Cookies

If you need to get e.g. the comment, path or expiry date etc from a cookie you need get a [detailed cookie](http://static.javadoc.io/io.rest-assured/rest-assured/4.3.3/io/restassured/http/Cookie.html) from REST Assured. To do this you can use the [Response.getDetailedCookie(java.lang.String)](http://static.javadoc.io/io.rest-assured/rest-assured/4.3.3/io/restassured/response/ResponseOptions.html" \l "getDetailedCookie-java.lang.String-) method. The detailed cookie then contains all attributes from the cookie.

You can also get all detailed response [cookies](http://static.javadoc.io/io.rest-assured/rest-assured/4.3.3/io/restassured/http/Cookies.html) using the [Response.getDetailedCookies()](http://static.javadoc.io/io.rest-assured/rest-assured/4.3.3/io/restassured/response/ResponseOptions.html" \l "getDetailedCookies--) method.

### Measuring Response Time

As of version 2.8.0 REST Assured has support measuring response time. For example:

long timeInMs = get("/lotto").time()

or using a specific time unit:

long timeInSeconds = get("/lotto").timeIn(SECONDS);

where SECONDS is just a standard TimeUnit. You can also validate it using the validation DSL:

when().

get("/lotto").

then().

time(lessThan(2000L)); // Milliseconds

or

when().

get("/lotto").

then().

time(lessThan(2L), SECONDS);

### Request Logging

given().log().all(). .. // Log all request specification details including parameters, headers and body

given().log().params(). .. // Log only the parameters of the request

given().log().body(). .. // Log only the request body

given().log().headers(). .. // Log only the request headers

given().log().cookies(). .. // Log only the request cookies

given().log().method(). .. // Log only the request method

given().log().path(). .. // Log only the request path

### Response Logging

If you want to print the response body regardless of the status code you can do:

get("/x").then().log().body() ..

This will print the response body regardless if an error occurred. If you're only interested in printing the response body if an error occur then you can use:

get("/x").then().log().ifError(). ..

You can also log all details in the response including status line, headers and cookies:

get("/x").then().log().all(). ..

as well as only status line, headers or cookies:

get("/x").then().log().statusLine(). .. // Only log the status line

get("/x").then().log().headers(). .. // Only log the response headers

get("/x").then().log().cookies(). .. // Only log the response cookies

### Log if validation fails

given().log().ifValidationFails(). ..

To log the response do:

.. .then().log().ifValidationFails(). ..

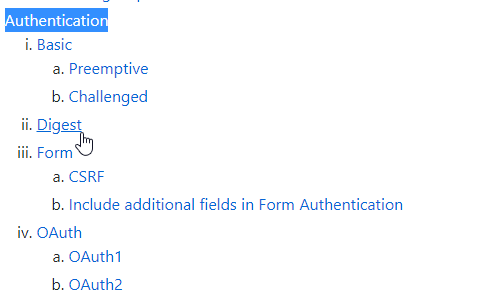
### SSL

given().relaxedHTTPSValidation().when().get("https://some\_server.com"). ..

You can also define this statically for all requests:

RestAssured.useRelaxedHTTPSValidation();

## Authentication



REST assured also supports several authentication schemes, for example OAuth, digest, certificate, form and preemptive basic authentication. You can either set authentication for each request:

**given().auth().basic("username", "password"). ..**

but you can also define authentication for all requests:

**RestAssured.authentication = basic("username", "password");**

or you can use a specification.

### Basic Authentication

There are two types of basic authentication, preemptive and "challenged basic authentication".

### **Preemptive Basic Authentication**

This will send the basic authentication credential even before the server gives an unauthorized response in certain situations, thus reducing the overhead of making an additional connection. This is typically what you want to use in most situations unless you're testing the servers ability to challenge.

Example:

given().auth().preemptive().basic("username", "password").when().get("/secured/hello").then().statusCode(200);

### **Challenged Basic Authentication**

When using "challenged basic authentication" REST Assured will not supply the credentials unless the server has explicitly asked for it. This means that REST Assured will make an additional request to the server in order to be challenged and then follow up with the same request once more but this time setting the basic credentials in the header.

given().auth().basic("username", "password").when().get("/secured/hello").then().statusCode(200);

### Digest Authentication

Currently only "challenged digest authentication" is supported. Example:

given().auth().digest("username", "password").when().get("/secured"). ..

### Form Authentication

Form authentication is very popular on the internet. It's typically associated with a user filling out his credentials (username and password) on a webpage and then pressing a login button of some sort. A very simple HTML page that provide the basis for form authentication may look like this:

<html>

<head>

<title>Login</title>

</head>

<body>

<form action="j\_spring\_security\_check" method="POST">

<table>

<tr><td>User:&nbsp;</td><td><input type='text' name='j\_username'></td></tr>

<tr><td>Password:</td><td><input type='password' name='j\_password'></td></tr>

<tr><td colspan='2'><input name="submit" type="submit"/></td></tr>

</table>

</form>

</body>

</html>

I.e. the server expects the user to fill-out the "j\_username" and "j\_password" input fields and then press "submit" to login. With REST Assured you can test a service protected by form authentication like this:

given().

auth().form("John", "Doe").

when().

get("/formAuth");

then().

statusCode(200);

### CSRF

Today it's common for the server to supply a [CSRF](https://en.wikipedia.org/wiki/Cross-site_request_forgery) token with the response in order to avoid these kinds of attacks. REST Assured has support for automatically parsing and supplying the CSRF token to the server.

### OAuth 1

OAuth 1 requires Scribe in the classpath. To use auth 1 authentication you can do:

given().auth().oauth(..). ..

### OAuth 2

Since version 2.5.0 you can use OAuth 2 authentication without depending on Scribe:

given().auth().oauth2(accessToken). ..

This will put the OAuth2 accessToken in a header. To be more explicit you can also do:

given().auth().preemptive().oauth2(accessToken). ..

There reason why given().auth().oauth2(..) still exists is for backward compatibility (they do the same thing). If you need to provide the OAuth2 token in a query parameter you currently need Scribe in the classpath. Then you can do like this:

given().auth().oauth2(accessToken, OAuthSignature.QUERY\_STRING). ..

## HamCrester Matchers

## How to create a local API server

<https://github.com/typicode/json-server>

npm install -g json-server

json-server –watch db.json