<https://www.selenium-tutorial.com/blog/151996/multiple-ways-of-creating-payload-in-rest-assured-api-tutorial>

**Multiple Ways of Creating Payload in RestAssured**

RestAssured is an open source Java based library to Automate REST and SOAP services. Sending payload with POST and PUT request is a critical part of API testing. I will discuss multiple ways of creating payload in this article. So, without wasting your time, let's quickly move into action to see how things work.

Since this is a beginner level tutorial, I have taken a simple JSON for learning purpose. Once you get hold of concepts, you can try your hands on with complex JSONs as well.

So JSON is a data representation which is in key value pair separated by a colon. Keys are always String in JSON. This is our JSON where user is key, and value is separated by a colon. Things inside { } are called JSON object while things inside [ ] are JSON array. You can read more about JSON [here](https://www.json.org/).

{

"user": "James",

"role": "admin",

"resource permissions": [

"Read",

"Write"

],

"database\_permissions": [

"Insert",

"Update, Delete"

]

}

**Creating Payload as String:**

This is the easiest way of creating payload. You simply need to store your JSON in a String variable and pass it on in body.

String my\_json = "{\r\n" +

" \"user\": \"James\",\r\n" +

" \"role\": \"admin\",\r\n" +

" \"resource\_permissions\": [\r\n" +

" \"Read\",\r\n" +

" \"Write\"\r\n" +

" ],\r\n" +

" \"database\_permissions\": [\r\n" +

" \"Insert\",\r\n" +

" \"Update, Delete\"\r\n" +

" ]\r\n" +

" }";

Response response = *given*().contentType(ContentType.***JSON***).

when().body(my\_json).

post("/endpoint").

then().extract().response();

**Creating Payload as File:**

In order to send payload as file you need to keep your JSON inside a file and then mention this file location using the File class in Java.

So, your file object will look like this:

File file = **new** File(".//JsonFiles/myjson.json");

And your post request will be like below:

Response response = *given*().contentType(ContentType.***JSON***).

when().body(file).

post("/endpoint").

then().extract().response();

**Creating Payload as Byte Array:**

You can also read content of your JSON file as Byte array and pass it on to your body method. In below line of code I am reading data from a file and storing it as byte Array and will pass it with my body method.

**byte**[] jsonData = Files.*readAllBytes*(Paths.*get*(".//JsonFiles/myjson.json"));

This is the way to pass it to your POST or PUT request.

Response response = *given*().contentType(ContentType.***JSON***).

when().body(jsonData).

post("/endpoint").

then().extract().response();

**Creating Payload as JSON Object:**

As you already know that JSON is nothing but a data representation using Object and Array. So we can also build payload using JSONObject and JSONArray classes. However you need to add *org.json*library in your project to use this feature. You can add this dependency in your project to get org.json library.

*<!-- https://mvnrepository.com/artifact/org.json/json -->*

*<dependency>*

*<groupId>org.json</groupId>*

*<artifactId>json</artifactId>*

*<version>20180813</version>*

*</dependency>*

So this is the piece of code with explanation to convert JSON into JSONObject.

//Adding user and role to JSON Object

JSONObject jsonObject = **new** JSONObject();

jsonObject.put("user", "James");

jsonObject.put("role", "admin");

//Resource permission is an Array so first adding its values to an array

JSONArray jsonArray = **new** JSONArray();

jsonArray.put("Read");

jsonArray.put("Write");

//Now adding values of resource permission to the

Object.

jsonObject.put("resource\_permissions", jsonArray);

//Database permissions is an Array so first adding

Its values to an array.

JSONArray jsonArray1 = **new** JSONArray();

jsonArray1.put("Insert");

jsonArray1.put("Update");

jsonArray1.put("Delete");

//Now adding values of database permission to the

Object

jsonObject.put("database\_permissions", jsonArray1);

//This will print a perfectly valid JSON in your

console

System.***out***.println(jsonObject);

And now the only thing you need to do is to send it along with your body method to your endpoint. That’s it.

Response response = *given*().contentType(ContentType.***JSON***).

when().body(jsonObject).

post("/endpoint").

then().extract().response();

**Creating Payload as Hashmap:**

HashMap is a data structure that stores data in key value pair. So it is the most obvious choice for creating payload.

First you need to initialize an empty map and then store JSON attribute keys as key and its value as value in the map. To store, JSON array values you should use Arrays.asList() method.

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

map.put("user", "James");

map.put("role", "admin");

map.put("resource\_permissions", Arrays.*asList*("Read", "Write"));

map.put("database\_permissions", Arrays.*asList*("Insert", "Update", "Delete"));

Now you can pass this map object to your POST or PUT request like below:

Response response = *given*().contentType(ContentType.***JSON***).

when().body(map).

post("/endpoint").

then().extract().response();

*P.S.: If you want a JSON representation from map then you need to pass your map object into object of JSONObject class like this****new****JSONObject(map). If you print this System.****out****.println(****new****JSONObject(map)); in your console you will get a perfectly valid JSON.*

**Creating Payload as POJO Classes:**

This is the widely used process in the industry. In real time, you will be creating POJO classes to convert your JSON into Java object. I would also recommend you create payload in this way if you are going to implement API Automation in your project.

**POJO**stands for Plain Old Java Object. A Pojo class will have fields and getter and setter methods. You can go through this [link](https://en.wikipedia.org/wiki/Plain_old_Java_object) to know more about POJO classes if you are not aware of them.

So this is the POJO class of our JSON:

**public** **class** MyPojo

{

**private** String role;

**private** String[] database\_permissions;

**private** String[] resource\_permissions;

**private** String user;

**public** String getRole ()

{

**return** role;

}

**public** **void** setRole (String role)

{

**this**.role = role;

}

**public** String[] getDatabase\_permissions ()

{

**return** database\_permissions;

}

**public** **void** setDatabase\_permissions (String[] database\_permissions)

{

**this**.database\_permissions = database\_permissions;

}

**public** String[] getResource\_permissions ()

{

**return** resource\_permissions;

}

**public** **void** setResource\_permissions (String[] resource\_permissions)

{

**this**.resource\_permissions = resource\_permissions;

}

**public** String getUser ()

{

**return** user;

}

**public** **void** setUser (String user)

{

**this**.user = user;

}

}

This POJO class has user and role String variables and as I already mentioned several times resource\_permissions and database\_permissions are array so it has array of String for them.

Then it has setter and getter to set the value of user, role, resource\_permissions and database\_permissions.

Now we will go to my RestAssured test class and create a Java Object of our JSON. We will do it step by step.

1. First, we will create object of our POJO Class.

MyPojo pojo = **new** MyPojo();

1. We will set value of user using setUser() method.

pojo.setUser("James");

1. We will set value of role using setRole() method.

pojo.setRole("admin");

1. Since resource\_permissions is an array we will first create an array of String.

String[] resource\_permissions= {"Read", "Write"};

1. Then we will set value of resource\_permissions using its setter method.

pojo.setResource\_permissions(resource\_permissions);

1. Since database\_permissions is an array we will first create an array of String

String[] database\_permissions= {"Insert", "Update", "Delete"};

1. Then we will set value of database\_permissions using its setter method.

pojo.setDatabase\_permissions(database\_permissions);

And our complete code to create payload will look like this:

MyPojo pojo = **new** MyPojo();

pojo.setUser("James");

pojo.setRole("admin");

String[] resource\_permissions= {"Read", "Write"};

pojo.setResource\_permissions(resource\_permissions);

String[] database\_permissions= {"Insert", "Update", "Delete"};

pojo.setDatabase\_permissions(database\_permissions);

So now all you need to do is pass *pojo* object to the RestAssured request. And RestAssured will automatically translate it into a JSON.

This is how our RestAssured request will look like:

Response response = *given*().contentType(ContentType.***JSON***).

when().body(pojo).

post("/endpoint").

then().extract().response();

*P.S: Do not worry about how to create large and complex POJO classes manually. There are many online resources available for this. One such online utility is*[*this*](http://www.jsonschema2pojo.org/)*which not only gives you POJO classes, it also binds them with your packages. If you do not want to use any online resource then you can also use a nice library*[*jsonschema2pojo*](https://github.com/joelittlejohn/jsonschema2pojo)*to convert your JSON into POJO classes programmatically. You can find its maven dependency from*[*here*](https://mvnrepository.com/artifact/org.jsonschema2pojo/jsonschema2pojo-core/1.0.1)*.*

So that’s it in this article. I hope this was a good reading for you. I will be back with another knowledge sharing article soon. Till then Happy Reading!