# Rest Assured - Assertions and Methods with Complex JSON

## Overview

This document shows how to use Rest Assured to validate a complex JSON response using assertions and helper methods such as `then()`, `extract()`, `body()`, and JsonPath.

## Sample Complex JSON Response

{  
 "status": "success",  
 "user": {  
 "id": 501,  
 "name": "Alice Smith",  
 "email": "alice.smith@example.com",  
 "isActive": true,  
 "address": {  
 "street": "123 Elm Street",  
 "city": "Los Angeles",  
 "zipcode": "90001"  
 },  
 "orders": [  
 {  
 "orderId": 1001,  
 "orderDate": "2024-01-10",  
 "items": [  
 {"productId": "A100", "productName": "Laptop", "price": 999.99},  
 {"productId": "A101", "productName": "Mouse", "price": 25.49}  
 ]  
 },  
 {  
 "orderId": 1002,  
 "orderDate": "2024-03-15",  
 "items": [  
 {"productId": "B200", "productName": "Keyboard", "price": 45.00}  
 ]  
 }  
 ]  
 }  
}

## Using Rest Assured with Assertions

import io.restassured.RestAssured;  
import static io.restassured.RestAssured.\*;  
import static org.hamcrest.Matchers.\*;  
import org.testng.annotations.Test;  
  
public class RestAssuredAssertionsTest {  
  
 @Test  
 public void validateComplexJson() {  
 RestAssured.baseURI = "https://api.example.com";  
  
 given()  
 .when()  
 .get("/user/details")  
 .then()  
 .statusCode(200)  
 .body("status", equalTo("success"))  
 .body("user.name", equalTo("Alice Smith"))  
 .body("user.isActive", is(true))  
 .body("user.address.city", equalTo("Los Angeles"))  
 .body("user.orders.size()", equalTo(2))  
 .body("user.orders[0].items[0].productName", equalTo("Laptop"))  
 .body("user.orders[1].items[0].price", equalTo(45.00f));  
 }  
}

## Extract Data and Assert Using JsonPath

Response response = given().when().get("/user/details");  
JsonPath jsonPath = response.jsonPath();  
  
String name = jsonPath.getString("user.name");  
assert name.equals("Alice Smith");  
  
float price = jsonPath.getFloat("user.orders[0].items[0].price");  
assert price == 999.99f;

## Summary of Useful Methods

|  |  |
| --- | --- |
| Method | Purpose |
| given() | Set up request specification |
| when() | Send HTTP request |
| then() | Start response validation chain |
| body() | Assert values in JSON response |
| statusCode() | Assert HTTP status code |
| extract() | Extract values for reuse |
| jsonPath() | Parse and extract using path expressions |

1. **Request Methods**:

* **get(String path)**: Sends a GET request to the specified path.
* **post(String path)**: Sends a POST request to the specified path.
* **put(String path)**: Sends a PUT request to the specified path.
* **delete(String path)**: Sends a DELETE request to the specified path.
* **patch(String path)**: Sends a PATCH request to the specified path.
* **options(String path)**: Sends an OPTIONS request to the specified path.

1. **Response Validation**:

* **then()**: Allows you to specify expectations for the response after making a request.
* **statusCode(int statusCode)**: Validates that the response has the expected HTTP status code.
* **contentType(String contentType)**: Validates that the response has the expected content type.
* **body(String path, Matcher<?> matcher)**: Validates the response body using a specified matcher (e.g., checking JSON values).

1. **Request Specification**:

* **given()**: Used to set up the request specification, including headers, parameters, and body.
* **header(String name, String value)**: Adds a header to the request.
* **queryParam(String name, String value)**: Adds a query parameter to the request.
* **body(Object body)**: Sets the body of the request for POST and PUT requests.

1. **Response Specification**:

* **expect()**: Used to set up expectations for the response, similar to then() but allows for more complex assertions.
* **cookie(String name, String value)**: Validates that a specific cookie is present in the response.

**Example Usage**

Here’s a simple example of how to use REST Assured to test an API endpoint:

import static io.restassured.RestAssured.\*;

import static org.hamcrest.Matchers.\*;

public class ApiTest {

@Test

public void testGetRequest() {

given()

.header("Authorization", "Bearer token")

.when()

.get("http://my-api.com/resource/123")

.then()

.statusCode(200)

.contentType("application/json")

.body("id", equalTo(123));

}

Serialization and Deserialization in Rest Assured Framework

# What is Serialization?

Serialization is the process of converting a Java object into JSON so that it can be sent in the body of an API request (mostly in POST or PUT).

Example: Creating a new user by sending JSON to an API.

# POJO Class Example:

public class User {

private String name;

private int age;

public User(String name, int age) {

this.name = name;

this.age = age;

}

public String getName() { return name; }

public void setName(String name) { this.name = name; }

public int getAge() { return age; }

public void setAge(int age) { this.age = age; }

}

# POST Request using Serialization:

User user = new User("Kumar", 30);

given()

.contentType("application/json")

.body(user)

.when()

.post("https://api.example.com/users")

.then()

.statusCode(201);

# What is Deserialization?

Deserialization is the process of converting JSON response into a Java object so that we can easily read and validate values.

# GET Request using Deserialization:

Response response = given()

.when()

.get("https://api.example.com/users/1");

User userFromResponse = response.as(User.class);

System.out.println(userFromResponse.getName());

System.out.println(userFromResponse.getAge());

# What is a POJO?

POJO = Plain Old Java Object

A simple Java class with private variables and public getters/setters.

Used for mapping data between Java and JSON (both serialization and deserialization).

# Summary Table:

Concept | Purpose | Involves JSON? | Direction | Real Use Case

----------------|----------------------------------------|----------------|-----------------|-----------------------------

Serialization | Convert Java Object → JSON | Yes | Java → JSON | Sending a POST request

Deserialization | Convert JSON → Java Object | Yes | JSON → Java | Reading a GET/POST response

POJO | Acts as data structure for mapping | Yes | Used for both | Structure for both flows

# Real-Life Scenario:

User user = new User("Adwaitha", 5);

Response postResponse = given()

.contentType("application/json")

.body(user)

.when()

.post("https://api.example.com/users")

.then()

.statusCode(201)

.extract().response();

User userCreated = postResponse.as(User.class);

System.out.println("Name from Response: " + userCreated.getName());

System.out.println("Age from Response: " + userCreated.getAge());

# Why Use These Concepts in Testing:

Use Case | Concept | Benefit

----------------------------------|-----------------|-------------------------------

POST/PUT requests with objects | Serialization | Avoid writing JSON manually

GET/POST response validation | Deserialization | Easily extract and assert

API chaining | Both | Clean and reusable code

# Rest Assured - Complex JSON and Data Extraction (JsonPath & POJO)

## Complex JSON Response (sample.json)

{  
 "status": "success",  
 "user": {  
 "id": 501,  
 "name": "Alice Smith",  
 "email": "alice.smith@example.com",  
 "isActive": true,  
 "address": {  
 "street": "123 Elm Street",  
 "city": "Los Angeles",  
 "zipcode": "90001"  
 },  
 "orders": [  
 {  
 "orderId": 1001,  
 "orderDate": "2024-01-10",  
 "items": [  
 {"productId": "A100", "productName": "Laptop", "price": 999.99},  
 {"productId": "A101", "productName": "Mouse", "price": 25.49}  
 ]  
 },  
 {  
 "orderId": 1002,  
 "orderDate": "2024-03-15",  
 "items": [  
 {"productId": "B200", "productName": "Keyboard", "price": 45.00}  
 ]  
 }}

## Extracting Data using JsonPath

JsonPath jsonPath = new JsonPath(new File("src/test/resources/sample.json"));  
  
String name = jsonPath.getString("user.name");  
String city = jsonPath.getString("user.address.city");  
List<Float> itemPrices = jsonPath.getList("user.orders.items.price");  
  
String firstOrderProduct = jsonPath.getString("user.orders[0].items[0].productName");  
int secondOrderId = jsonPath.getInt("user.orders[1].orderId");  
  
System.out.println("Name: " + name);  
System.out.println("City: " + city);  
System.out.println("Prices: " + itemPrices);  
System.out.println("First Product: " + firstOrderProduct);  
System.out.println("Second Order ID: " + secondOrderId);

## Extracting Data using POJO Classes

Define the POJO classes as shown below:

class User {  
 public int id;  
 public String name;  
 public String email;  
 public boolean isActive;  
 public Address address;  
 public List<Order> orders;  
}  
  
class Address {  
 public String street;  
 public String city;  
 public String zipcode;  
}  
  
class Order {  
 public int orderId;  
 public String orderDate;  
 public List<Item> items;  
}  
  
class Item {  
 public String productId;  
 public String productName;  
 public float price;  
}

## Using Jackson ObjectMapper to Deserialize JSON

ObjectMapper mapper = new ObjectMapper();  
User user = mapper.readValue(new File("src/test/resources/sample.json"), UserWrapper.class).user;  
  
System.out.println("User Name: " + user.name);  
System.out.println("User City: " + user.address.city);  
System.out.println("First Item Price: " + user.orders.get(0).items.get(0).price);

# Rest Assured - Sample JSON and JsonPath Usage

## Sample JSON Response (sample.json)

{  
 "status": "success",  
 "data": {  
 "id": 101,  
 "name": "John Doe",  
 "email": "john.doe@example.com",  
 "roles": ["admin", "editor"],  
 "address": {  
 "city": "New York",  
 "zipcode": "10001"  
 },  
 "projects": [  
 {  
 "projectId": 1,  
 "projectName": "Alpha"  
 },  
 {  
 "projectId": 2,  
 "projectName": "Beta"  
 }  
 ]  
 }  
}

## Maven Dependency

<dependency>  
 <groupId>io.rest-assured</groupId>  
 <artifactId>rest-assured</artifactId>  
 <version>5.4.0</version>  
 <scope>test</scope>  
</dependency>

## Java Code Example Using JsonPath

import io.restassured.path.json.JsonPath;  
import org.testng.annotations.Test;  
  
import java.io.File;  
import java.util.List;  
  
public class JsonPathExample {  
  
 @Test  
 public void readJsonData() {  
 File file = new File("src/test/resources/sample.json");  
 JsonPath jsonPath = new JsonPath(file);  
  
 String status = jsonPath.getString("status");  
 int id = jsonPath.getInt("data.id");  
 String email = jsonPath.getString("data.email");  
 List<String> roles = jsonPath.getList("data.roles");  
 String city = jsonPath.getString("data.address.city");  
 String firstProjectName = jsonPath.getString("data.projects[0].projectName");  
 int secondProjectId = jsonPath.getInt("data.projects[1].projectId");  
  
 System.out.println("Status: " + status);  
 System.out.println("ID: " + id);  
 System.out.println("Email: " + email);  
 System.out.println("Roles: " + roles);  
 System.out.println("City: " + city);  
 System.out.println("First Project Name: " + firstProjectName);  
 System.out.println("Second Project ID: " + secondProjectId);  
 }  
}

## JsonPath Expressions Summary

|  |  |
| --- | --- |
| Expression | Meaning |
| Status | Root level string |
| data.id | Nested key |
| data.roles[0] | First item in roles array |
| data.projects[1].projectId | Second object’s projectId |
| data.address.city | City inside nested object |

## 1. What is Rest Assured?

Rest Assured is a Java-based library used to test RESTful APIs. It provides a domain-specific language (DSL) for writing powerful, maintainable tests for REST services.

Example:

given().get("/api/users").then().statusCode(200);

## 2. How do you perform a GET request using Rest Assured?

Example:

given()

.when()

.get("https://api.example.com/users")

.then()

.statusCode(200);

## 3. How do you send a POST request with a JSON body?

Example:

String json = "{\"name\":\"John\",\"age\":30}";

given()

.header("Content-Type", "application/json")

.body(json)

.when()

.post("https://api.example.com/users")

.then()

.statusCode(201);

## 4. How do you validate JSON response fields?

given()

.get("/api/users/1")

.then()

.body("name", equalTo("John"));

## 5. How to pass path parameters?

given()

.pathParam("id", 1)

.get("/api/users/{id}")

.then()

.statusCode(200);

## 6. Sample Rest Assured Question 6

// Sample code or explanation for Question 6

given()

.get("/api/sample")

.then().statusCode(200);