Rest Assured - Complete Guide

Table of Contents

---

1. Introduction to Rest Assured

Rest Assured is a Java library that provides a domain-specific language (DSL) for writing powerful and maintainable tests for RESTful APIs. It simplifies API testing by making the test code readable and expressive.

Key Features:

- Easy-to-use DSL for API testing
- Support for all HTTP methods (GET, POST, PUT, DELETE, PATCH, etc.)
- Built-in JSON and XML path support
- Integration with testing frameworks (JUnit, TestNG)
- Support for various authentication mechanisms
- Request and response specifications for reusability

Maven Dependency:

xml

```
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>5.3.0</version>
```

```xml
    <scope>test</scope>
</dependency>
```

Gradle Dependency:

gradle

```gradle
testImplementation 'io.rest-assured:rest-assured:5.3.0'
```

---

2. Setup and Configuration

Basic Import Statements

java

```java
import static io.restassured.RestAssured.*;
import static io.restassured.matcher.RestAssuredMatchers.*;
import static org.hamcrest.Matchers.*;
import io.restassured.RestAssured;
import io.restassured.response.Response;
import io.restassured.specification.RequestSpecification;
```

Setting Base URI

java

```java
// Method 1: Set globally
RestAssured.baseURI = "https://api.example.com";
RestAssured.basePath = "/v1";
RestAssured.port = 8080;

// Method 2: Use in request
given()
    .baseUri("https://api.example.com")
    .basePath("/v1")
    .when()
    .get("/users");
```

---

3. Core Concepts

The Given-When-Then Pattern

Rest Assured follows the BDD (Behavior Driven Development) style:

- **given()**: Setup conditions (headers, parameters, body, authentication)
- **when()**: Perform the action (HTTP request)
- **then()**: Validate the response (status code, body, headers)

Basic Structure:

java

```
given()
    .header("Content-Type", "application/json")
    .body("{\"name\":\"John\"}")
.when()
    .post("/users")
.then()
    .statusCode(201)
    .body("name", equalTo("John"));
```

---

4. Making HTTP Requests

GET Request

java

```
// Simple GET
given()
    .when()
    .get("/users")
    .then()
    .statusCode(200);

// GET with path parameters
given()
    .pathParam("id", 123)
    .when()
    .get("/users/{id}")
    .then()
```

```java
    .statusCode(200);

// GET with query parameters
given()
    .queryParam("page", 1)
    .queryParam("size", 10)
    .when()
    .get("/users")
    .then()
    .statusCode(200);
```

POST Request

```java
// POST with JSON body
given()
    .header("Content-Type", "application/json")
    .body("{ \"name\": \"John\", \"age\": 30 }")
    .when()
    .post("/users")
    .then()
    .statusCode(201);

// POST with object (serialization)
User user = new User("John", 30);
given()
    .contentType(ContentType.JSON)
    .body(user)
    .when()
    .post("/users")
    .then()
    .statusCode(201);
```

PUT Request

java

```
given()
    .contentType(ContentType.JSON)
    .body("{ \"name\": \"John Updated\", \"age\": 31 }")
    .when()
    .put("/users/123")
    .then()
    .statusCode(200);
```

PATCH Request

java

```
given()
    .contentType(ContentType.JSON)
    .body("{ \"age\": 32 }")
    .when()
    .patch("/users/123")
    .then()
    .statusCode(200);
```

DELETE Request

java

```
given()
    .pathParam("id", 123)
    .when()
    .delete("/users/{id}")
    .then()
    .statusCode(204);
```

---

5. Response Validation

Status Code Validation

java

```
.then()
    .statusCode(200)
    .statusCode(is(200))
    .statusCode(anyOf(is(200), is(201)));
```

Response Body Validation

java

```
// JSON path validation
.then()
    .body("name", equalTo("John"))
    .body("age", greaterThan(18))
    .body("email", containsString("@"))
    .body("users.size()", equalTo(10))
    .body("users[0].name", equalTo("John"))
    .body("users.name", hasItems("John", "Jane"));

// Multiple assertions
.then()
    .body("name", equalTo("John"),
          "age", equalTo(30),
          "email", notNullValue());
```

Header Validation

java

```
.then()
    .header("Content-Type", "application/json")
    .header("Server", containsString("nginx"))
    .headers("Content-Type", "application/json",
             "Content-Length", "1024");
```

Response Time Validation

java

```
.then()
    .time(lessThan(2000L)); // milliseconds
```

Extract Response

java

```
// Extract entire response
Response response = given()
    .when()
    .get("/users")
    .then()
    .extract().response();

// Extract specific values
String name = given()
    .when()
    .get("/users/123")
    .then()
    .extract().path("name");

int statusCode = response.getStatusCode();
String body = response.getBody().asString();
String header = response.getHeader("Content-Type");
```

---

6. Authentication

Basic Authentication

java

```
given()
    .auth()
    .basic("username", "password")
    .when()
    .get("/secure/users")
    .then()
    .statusCode(200);

// Preemptive basic auth (sends credentials without waiting for 401)
given()
```

```
    .auth()
    .preemptive()
    .basic("username", "password")
    .when()
    .get("/secure/users");
```

Bearer Token Authentication

java

```
given()
    .auth()
    .oauth2("your_access_token")
    .when()
    .get("/secure/users")
    .then()
    .statusCode(200);

// Or using header
given()
    .header("Authorization", "Bearer your_access_token")
    .when()
    .get("/secure/users");
```

API Key Authentication

java

```
// As query parameter
given()
    .queryParam("api_key", "your_api_key")
    .when()
    .get("/users");

// As header
given()
    .header("X-API-Key", "your_api_key")
```

```java
    .when()
    .get("/users");
```

OAuth 2.0

java

```java
given()
    .auth()
    .oauth2("access_token")
    .when()
    .get("/secure/resource");
```

---

7. Request Specifications

Request specifications allow you to reuse common request configurations.

Creating Request Specification

java

```java
RequestSpecification requestSpec = new RequestSpecBuilder()
    .setBaseUri("https://api.example.com")
    .setBasePath("/v1")
    .setContentType(ContentType.JSON)
    .addHeader("Authorization", "Bearer token")
    .build();

// Use the specification
given()
    .spec(requestSpec)
    .when()
    .get("/users")
    .then()
    .statusCode(200);
```

Static Request Specification

java

```java
RestAssured.requestSpecification = new RequestSpecBuilder()
```

```
    .setBaseUri("https://api.example.com")
    .setContentType(ContentType.JSON)
    .build();

// Now all requests use this spec by default
given()
    .when()
    .get("/users")
    .then()
    .statusCode(200);
```

---

8. Response Specifications

Response specifications allow you to reuse common response validations.

Creating Response Specification

java

```
ResponseSpecification responseSpec = new ResponseSpecBuilder()
    .expectStatusCode(200)
    .expectContentType(ContentType.JSON)
    .expectResponseTime(lessThan(3000L))
    .build();

// Use the specification
given()
    .when()
    .get("/users")
    .then()
    .spec(responseSpec)
    .body("name", equalTo("John"));
```

Static Response Specification

java

```
RestAssured.responseSpecification = new ResponseSpecBuilder()
```

```
    .expectStatusCode(200)
    .expectContentType(ContentType.JSON)
    .build();

// Now all responses are validated against this spec
given()
    .when()
    .get("/users");
```

---

9. Serialization and Deserialization

Object to JSON (Serialization)

```java
public class User {
    private String name;
    private int age;
    // constructors, getters, setters
}

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

given()
    .contentType(ContentType.JSON)
    .body(user) // Automatically serialized to JSON
    .when()
    .post("/users")
    .then()
    .statusCode(201);
```

JSON to Object (Deserialization)

```java
User user = given()
    .when()
    .get("/users/123")
    .then()
    .extract()
    .as(User.class); // Automatically deserialized from JSON

System.out.println(user.getName());
```

List Deserialization

```java
List<User> users = given()
    .when()
    .get("/users")
    .then()
    .extract()
    .jsonPath()
    .getList(".", User.class);
```

Using Jackson/Gson

```java
// Rest Assured uses Jackson by default
// Add dependency for Gson if needed
given()
    .config(RestAssuredConfig.config()
        .objectMapperConfig(new ObjectMapperConfig(ObjectMapperType.GSON)))
    .body(user)
    .when()
    .post("/users");
```

---

10. Advanced Features

JSON Path

```java
// Extract using JSON path
String firstName = response.jsonPath().getString("users[0].firstName");
```

```java
List<String> names = response.jsonPath().getList("users.name");
int userCount = response.jsonPath().getInt("users.size()");

// Complex JSON path queries
List<String> activeUsers = response.jsonPath()
    .getList("users.findAll { it.status == 'active' }.name");
```

XML Path

java

```java
// For XML responses
String name = response.xmlPath().getString("user.name");
List<String> names = response.xmlPath().getList("users.user.name");
```

File Upload

java

```java
given()
    .multiPart("file", new File("/path/to/file.txt"))
    .when()
    .post("/upload")
    .then()
    .statusCode(200);

// Multiple files
given()
    .multiPart("file1", new File("/path/to/file1.txt"))
    .multiPart("file2", new File("/path/to/file2.txt"))
    .when()
    .post("/upload");
```

File Download

java

```java
byte[] fileData = given()
    .when()
    .get("/download/file.pdf")
    .then()
    .extract()
    .asByteArray();

// Save to file
File downloadedFile = given()
    .when()
    .get("/download/file.pdf")
    .then()
    .extract()
    .asFile();
```

Cookies

java

```
// Set cookie
given()
    .cookie("session_id", "abc123")
    .when()
    .get("/users");

// Multiple cookies
given()
    .cookies("cookie1", "value1", "cookie2", "value2")
    .when()
    .get("/users");

// Extract cookie
String sessionId = given()
    .when()
    .get("/login")
    .then()
    .extract()
    .cookie("session_id");
```

Request/Response Logging

java

```
// Log everything
given()
    .log().all()
    .when()
    .get("/users")
    .then()
    .log().all();

// Log only on failure
given()
    .when()
    .get("/users")
    .then()
    .log().ifError();

// Log specific parts
given()
    .log().headers()
    .log().body()
    .when()
    .get("/users")
    .then()
    .log().status()
    .log().body();
```

Filters

java

```
// Custom filter for logging
given()
```

```
    .filter(new RequestLoggingFilter())
    .filter(new ResponseLoggingFilter())
    .when()
    .get("/users");

// Custom filter implementation
public class CustomFilter implements Filter {
    @Override
    public Response filter(FilterableRequestSpecification req,
                          FilterableResponseSpecification resp,
                          FilterContext ctx) {
        // Modify request/response
        return ctx.next(req, resp);
    }
}
```

Proxy Configuration

```java
given()
    .proxy("proxy.example.com", 8080)
    .when()
    .get("/users");

// With authentication
given()
    .proxy(auth("username", "password")
        .host("proxy.example.com")
        .port(8080))
    .when()
    .get("/users");
```

SSL/TLS Configuration

java

```
// Ignore SSL certificate validation (use only for testing)
given()
    .relaxedHTTPSValidation()
    .when()
    .get("https://secure-api.example.com/users");


// Custom SSL configuration
given()
    .config(RestAssured.config()
        .sslConfig(SSLConfig.sslConfig()
            .trustStore("/path/to/truststore.jks", "password")))
    .when()
    .get("https://secure-api.example.com/users");
```

URL Encoding

```
java
// Automatic URL encoding
given()
    .queryParam("name", "John Doe") // Automatically encoded to John%20Doe
    .when()
    .get("/users");


// Disable URL encoding
given()
    .urlEncodingEnabled(false)
    .queryParam("name", "John%20Doe")
    .when()
    .get("/users");
```

---

11. Best Practices

1. Use Request and Response Specifications

Reuse common configurations across multiple tests to reduce code duplication.

2. Extract and Organize Test Data

```java
public class TestData {
    public static final String BASE_URI = "https://api.example.com";
    public static final String API_KEY = "your_api_key";
}
```

3. Use POJOs for Request/Response

Create Java objects that represent your API data structures for better type safety and maintainability.

4. Implement Proper Logging

Use conditional logging to debug issues without cluttering test output:

```java
.then()
    .log().ifValidationFails()
```

5. Handle Dynamic Data

```java
// Use matchers for dynamic values
.then()
    .body("id", notNullValue())
    .body("timestamp", matchesPattern("\\d{4}-\\d{2}-\\d{2}"));
```

6. Organize Tests with TestNG/JUnit

```java
@Test
public void testGetUser() {
    given()
        .spec(commonRequestSpec)
        .when()
        .get("/users/123")
        .then()
        .spec(commonResponseSpec)
        .body("name", equalTo("John"));
}
```

7. Use Soft Assertions

For validating multiple conditions without failing at the first assertion:

java

```
// Not built into Rest Assured, use AssertJ or similar
```

8. Parameterize Tests

java

```
@Test(dataProvider = "userIds")
public void testMultipleUsers(int userId) {
    given()
        .pathParam("id", userId)
        .when()
        .get("/users/{id}")
        .then()
        .statusCode(200);
}
```

---

12. Common Examples

Example 1: Complete User CRUD Operations

java

```
public class UserAPITest {

    private static RequestSpecification requestSpec;
    private static ResponseSpecification responseSpec;

    @BeforeClass
    public static void setup() {
        requestSpec = new RequestSpecBuilder()
            .setBaseUri("https://api.example.com")
            .setBasePath("/v1")
            .setContentType(ContentType.JSON)
            .build();
        responseSpec = new ResponseSpecBuilder()
            .expectContentType(ContentType.JSON)
            .build();
    }

    @Test
    public void testCreateUser() {
        User user = new User("John", "john@example.com");

        int userId = given()
            .spec(requestSpec)
            .body(user)
            .when()
            .post("/users")
            .then()
            .spec(responseSpec)
            .statusCode(201)
            .body("name", equalTo("John"))
            .body("email", equalTo("john@example.com"))
            .extract()
            .path("id");

        System.out.println("Created user with ID: " + userId);
    }

    @Test
    public void testGetUser() {
        given()
            .spec(requestSpec)
            .pathParam("id", 123)
            .when()
            .get("/users/{id}")
            .then()
            .spec(responseSpec)
            .statusCode(200)
            .body("id", equalTo(123))
            .body("name", notNullValue());
    }

    @Test
    public void testUpdateUser() {
        User updatedUser = new User("John Updated", "john.updated@example.com");

        given()
            .spec(requestSpec)
            .pathParam("id", 123)
            .body(updatedUser)
            .when()
            .put("/users/{id}")
            .then()
            .spec(responseSpec)
            .statusCode(200)
            .body("name", equalTo("John Updated"));
    }

    @Test
    public void testDeleteUser() {
        given()
            .spec(requestSpec)
            .pathParam("id", 123)
            .when()
```

```
            .delete("/users/{id}")
            .then()
            .statusCode(204);
    }
}
```

Example 2: Testing Pagination

java

```
@Test
public void testPagination() {
    given()
        .spec(requestSpec)
        .queryParam("page", 1)
        .queryParam("limit", 10)
        .when()
        .get("/users")
        .then()
        .statusCode(200)
        .body("data.size()", equalTo(10))
        .body("pagination.page", equalTo(1))
        .body("pagination.limit", equalTo(10))
        .body("pagination.total", greaterThan(10));
}
```

Example 3: Testing Error Responses

java

```
@Test
public void testUserNotFound() {
    given()
        .spec(requestSpec)
        .pathParam("id", 99999)
        .when()
        .get("/users/{id}")
```

```java
        .then()
        .statusCode(404)
        .body("error", equalTo("User not found"))
        .body("errorCode", equalTo("USER_NOT_FOUND"));
}

@Test
public void testInvalidRequest() {
    given()
        .spec(requestSpec)
        .body("{\"name\": \"\"}") // Invalid: empty name
        .when()
        .post("/users")
        .then()
        .statusCode(400)
        .body("errors.name", hasItem("Name cannot be empty"));
}
```

Example 4: Testing with Authentication

java

```java
@Test
public void testWithBearerToken() {
    String token = getAuthToken(); // Method to obtain token

    given()
        .spec(requestSpec)
        .auth().oauth2(token)
        .when()
        .get("/secure/profile")
        .then()
        .statusCode(200)
        .body("username", notNullValue());
}

private String getAuthToken() {
    return given()
        .baseUri("https://auth.example.com")
        .contentType(ContentType.JSON)
        .body("{\"username\":\"user\",\"password\":\"pass\"}")
        .when()
        .post("/login")
        .then()
        .statusCode(200)
        .extract()
        .path("token");
}
```

Example 5: Data-Driven Testing

java

```
@DataProvider(name = "userCredentials")
public Object[][] getUserCredentials() {
    return new Object[][] {
        {"user1@example.com", "password1", 200},
        {"user2@example.com", "password2", 200},
        {"invalid@example.com", "wrong", 401}
    };
}

@Test(dataProvider = "userCredentials")
public void testLogin(String email, String password, int expectedStatus) {
    given()
        .spec(requestSpec)
        .body(String.format("{\"email\":\"%s\",\"password\":\"%s\"}",
            email, password))
        .when()

        .post("/login")

        .then()

        .statusCode(expectedStatus);

}
```

Summary

Rest Assured is a powerful library for API testing in Java. Key takeaways include understanding the given-when-then pattern, utilizing request and response specifications for reusability, properly handling authentication mechanisms, leveraging serialization and deserialization for working with objects, and implementing comprehensive validation of responses including status codes, headers, body content, and response times. By following best practices and organizing your tests effectively, you can create maintainable and robust API test suites.