REST Assured Schema Validation

In **Rest Assured**, schema validation is used to ensure that the JSON or XML response returned from an API follows a specific structure. This is useful for verifying API contracts during testing.

✅ Types of Schema Validation in Rest Assured:

1. **JSON Schema Validation**
2. **XML Schema Validation**

We'll focus on **JSON Schema Validation** as it's more common in REST APIs.

🔧 Maven Dependency for Schema Validation

Add this to your **`pom.xml`**:

```
xml


<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>json-schema-validator</artifactId>
    <version>5.3.1</version> <!-- or latest -->
    <scope>test</scope>
</dependency>
```

🗂️ Folder Structure

Place your JSON schema in the **`src/test/resources/schemas/`** directory.

Example:

```
pgsql


src/
 └── test/
     └── resources/
         └── schemas/
             └── user_schema.json
```

🔪 Example: JSON Schema Validation

✅ Sample JSON Response

```
json


{
  "id": 1,
  "name": "John",
```

```
  "email": "john@example.com",
  "active": true
}
```

✅ JSON Schema (`user_schema.json`)

```
json


{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "required": ["id", "name", "email", "active"],
  "properties": {
    "id": { "type": "integer" },
    "name": { "type": "string" },
    "email": { "type": "string", "format": "email" },
    "active": { "type": "boolean" }
  }
}
```

✅ Rest Assured Test Code

```
java


import io.restassured.RestAssured;
import static io.restassured.RestAssured.*;
import static io.restassured.module.jsv.JsonSchemaValidator.*;

import org.testng.annotations.Test;

public class SchemaValidationTest {

    @Test
    public void validateUserSchema() {
        RestAssured.baseURI = "https://api.example.com";

        given()
        .when()
            .get("/users/1")
        .then()
            .assertThat()
            .body(matchesJsonSchemaInClasspath("schemas/user_schema.json"));
    }
}
```

✅ XML Schema Validation

Use **`matchesXsdInClasspath("schema.xsd")`** in place of JSON schema method. You'll also need the **`xml-path`** dependency.

✅ Summary

Type	Method Used
JSON Schema	**`matchesJsonSchemaInClasspath()`**
XML Schema	**`matchesXsdInClasspath()`**

Would you like a **mini project** or **assignment** with schema validation?

Reusing **JSON Schema files** in **Rest Assured** allows you to modularize and manage your API tests efficiently — especially when multiple APIs return similar response structures.

✅ Why Reuse JSON Schema Files?

- Reduces duplication in large test suites
- Promotes consistency across test validations
- Makes maintenance easier when API structure changes

🛠️ How to Write and Reuse JSON Schema Files in Rest Assured

🔹 Step 1: Define Your JSON Schema File

Create a file like `user_schema.json` inside `src/test/resources/schemas/`.

Example: `user_schema.json`

```
json

{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "required": ["id", "name", "email", "active"],
  "properties": {
    "id": { "type": "integer" },
    "name": { "type": "string" },
    "email": { "type": "string", "format": "email" },
    "active": { "type": "boolean" }
  }
}
```

🔹 Step 2: Reuse Schema in Test Code

You can load and use this schema in multiple test classes.

Example 1: Validate Single User API

```
java

@Test
public void validateSingleUser() {
    given()
    .when()
        .get("https://api.example.com/users/1")
    .then()
        .body(matchesJsonSchemaInClasspath("schemas/user_schema.json"));
}
```

Example 2: Validate User from Different Endpoint

```
java

@Test
public void validateUserFromSearchAPI() {
    given()
    .queryParam("q", "john")
    .when()
        .get("https://api.example.com/search")
    .then()
        .body("results[0]", matchesJsonSchemaInClasspath("schemas/user_schema.json"));
}
```

✅ Note: The `"results[0]"` points to the first user in the list.

🔹 Step 3: Organize Multiple Schemas

For bigger projects, you can create a structure like:

```
pgsql

src/test/resources/schemas/
 ├── user/
 │    ├── user_schema.json
 │    └── address_schema.json
 ├── product/
 │    └── product_schema.json
 └── common/
      └── error_response_schema.json
```

Then reference them like:

```
java

.matchesJsonSchemaInClasspath("schemas/user/user_schema.json")
```

🔹 Step 4: Reusable Helper Method (Optional)

If you prefer, create a utility to load schemas dynamically:

```
java

public class SchemaUtils {
    public static Matcher<?> getSchema(String schemaPath) {
        return matchesJsonSchemaInClasspath("schemas/" + schemaPath);
    }
}
```

Then use it in tests:

```
java

.then().body(SchemaUtils.getSchema("user/user_schema.json"))
```

🧪 Bonus: Parameterized Test Example with Schema

If you're validating similar objects from multiple endpoints:

```
java

@DataProvider(name = "userEndpoints")
public Object[][] userEndpoints() {
    return new Object[][] {
        { "/users/1" },
        { "/admins/2" },
        { "/customers/5" }
    };
}

@Test(dataProvider = "userEndpoints")
public void validateUserSchemaAcrossEndpoints(String endpoint) {
    given()
    .when()
        .get("https://api.example.com" + endpoint)
    .then()
        .body(matchesJsonSchemaInClasspath("schemas/user/user_schema.json"));
}
```

✅ Summary

Goal	How
Define reusable schema	Place **`.json`** files in **`src/test/resources/schemas/`**
Use in multiple tests	Use **`matchesJsonSchemaInClasspath("schemas/...")`**
Group schemas by entity	Use subfolders: **`user/`**, **`product/`**, **`common/`**, etc.
Optional helper method	Create a utility like **`SchemaUtils`**
Parameterized reuse	Use TestNG/JUnit DataProviders or parameterized tests

Would you like me to generate a **mini-project** structure or **assignment** using reusable schema files with Rest Assured?