REST Assured Data-Driven Testing

✅ Data-Driven Testing in Rest Assured

Data-driven testing means testing the same API with multiple sets of data. In **Rest Assured**, we often combine this approach with **TestNG** or **JUnit**, along with **data providers** (e.g., TestNG `@DataProvider`) or external data sources like Excel, JSON, or CSV.

🚀 Example: Data-Driven Testing using TestNG + Rest Assured + `@DataProvider`

🔧 Maven Dependencies

Add the following to your `pom.xml`:

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

    <dependency>
        <groupId>org.testng</groupId>
        <artifactId>testng</artifactId>
        <version>7.10.0</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

✅ Test Example: POST Request with Multiple Payloads

🔹 Step 1: Create Test Class

```java
import io.restassured.RestAssured;
import io.restassured.response.Response;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

public class DataDrivenAPITest {

    @DataProvider(name = "userData")
    public Object[][] getData() {
        return new Object[][] {
            {"John", "QA"},
            {"Alice", "Dev"},
            {"Bob", "Manager"}
        };
```

```
    }

    @Test(dataProvider = "userData")
    public void createUserTest(String name, String job) {
        RestAssured.baseURI = "https://reqres.in";

        String requestBody = "{ \"name\": \"" + name + "\", \"job\": \"" + job + "\" }";

        given()
            .header("Content-Type", "application/json")
            .body(requestBody)
        .when()
            .post("/api/users")
        .then()
            .statusCode(201)
            .body("name", equalTo(name))
            .body("job", equalTo(job));
    }
}
```

🗂️ Advanced: Reading Data from Excel (Apache POI)

If you'd like to read test data from Excel:

1. Add Apache POI dependencies to **`pom.xml`**
2. Create a utility to read data from Excel.
3. Use it inside your **`@DataProvider`**.

Let me know if you want a full working example with Excel support. ✅

🔍 Summary

Feature	Implementation
Framework	TestNG + Rest Assured
Data source	**`@DataProvider`**, Excel, JSON, CSV
Use case	Reuse the same test logic with different inputs
Benefits	Scalability, Reusability, Maintainability

Would you like:

- ✅ The same test using Excel data?
- ✅ A version using CSV or JSON files?
- ✅ A GET/PUT/DELETE example with data-driven logic?

Let me know!

Great! Let's explore **how to read test data from Excel and CSV files** for **Data-Driven Testing in Rest Assured** using **Java + TestNG**.

✅ 1. Excel-Based Data-Driven Testing using Apache POI

🔧 Step 1: Add Maven Dependencies

```xml
xml

<dependencies>
    <!-- Rest Assured -->
    <dependency>
        <groupId>io.rest-assured</groupId>
        <artifactId>rest-assured</artifactId>
        <version>5.4.0</version>
        <scope>test</scope>
    </dependency>

    <!-- TestNG -->
    <dependency>
        <groupId>org.testng</groupId>
        <artifactId>testng</artifactId>
        <version>7.10.0</version>
        <scope>test</scope>
    </dependency>

    <!-- Apache POI for Excel -->
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi-ooxml</artifactId>
        <version>5.2.5</version>
    </dependency>
</dependencies>
```

📄 Step 2: Sample Excel File (`TestData.xlsx`)

Name	Job
Alice	Dev
Bob	Tester
Charlie	PM

Save it in your project under **`src/test/resources/TestData.xlsx`**.

🧰 Step 3: Excel Utility to Read Data

```java
java

import org.apache.poi.ss.usermodel.*;
import java.io.File;
import java.io.FileInputStream;
import java.util.Iterator;

public class ExcelUtils {

    public static Object[][] readExcel(String filePath, String sheetName) {
        Object[][] data = null;

        try (FileInputStream fis = new FileInputStream(new File(filePath))) {
```

```
            Workbook workbook = WorkbookFactory.create(fis);
            Sheet sheet = workbook.getSheet(sheetName);

            int rowCount = sheet.getPhysicalNumberOfRows();
            int colCount = sheet.getRow(0).getLastCellNum();

            data = new Object[rowCount - 1][colCount];

            for (int i = 1; i < rowCount; i++) {
                Row row = sheet.getRow(i);
                for (int j = 0; j < colCount; j++) {
                    data[i - 1][j] = row.getCell(j).toString();
                }
            }

        } catch (Exception e) {
            e.printStackTrace();
        }

        return data;
    }
}
```

✅ Step 4: Test Class with DataProvider

```
java


import io.restassured.RestAssured;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

public class ExcelDrivenAPITest {

    @DataProvider(name = "excelData")
    public Object[][] getExcelData() {
        String path = "src/test/resources/TestData.xlsx";
        return ExcelUtils.readExcel(path, "Sheet1");
    }

    @Test(dataProvider = "excelData")
    public void createUserTest(String name, String job) {
        RestAssured.baseURI = "https://reqres.in";

        String requestBody = "{ \"name\": \"" + name + "\", \"job\": \"" + job + "\" }";

        given()
            .header("Content-Type", "application/json")
            .body(requestBody)
        .when()
            .post("/api/users")
        .then()
            .statusCode(201)
            .body("name", equalTo(name))
            .body("job", equalTo(job));
    }
}
```

✅ 2. CSV-Based Data-Driven Testing

🧱 Step 1: Add OpenCSV Dependency

```xml
<dependency>
    <groupId>com.opencsv</groupId>
    <artifactId>opencsv</artifactId>
    <version>5.7.1</version>
</dependency>
```

📄 Step 2: Sample CSV File (`TestData.csv`)

```csv
Name,Job
Alice,Developer
Bob,QA
Charlie,Manager
```

🧰 Step 3: CSV Utility to Read Data

```java
import com.opencsv.CSVReader;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;

public class CSVUtils {

    public static Object[][] readCSV(String filePath) {
        List<Object[]> data = new ArrayList<>();

        try (CSVReader reader = new CSVReader(new FileReader(filePath))) {
            String[] line;
            reader.readNext(); // skip header
            while ((line = reader.readNext()) != null) {
                data.add(new Object[] { line[0], line[1] });
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

        return data.toArray(new Object[0][0]);
    }
}
```

✅ Step 4: Test Class Using CSV Data

```java
import io.restassured.RestAssured;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

public class CSVDrivenAPITest {

    @DataProvider(name = "csvData")
    public Object[][] getCSVData() {
```

```
        String path = "src/test/resources/TestData.csv";
        return CSVUtils.readCSV(path);
    }

    @Test(dataProvider = "csvData")
    public void createUserTest(String name, String job) {
        RestAssured.baseURI = "https://reqres.in";

        String requestBody = "{ \"name\": \"" + name + "\", \"job\": \"" + job + "\" }";

        given()
            .header("Content-Type", "application/json")
            .body(requestBody)
        .when()
            .post("/api/users")
        .then()
            .statusCode(201)
            .body("name", equalTo(name))
            .body("job", equalTo(job));
    }
}
```

📌 Summary

Format	Library Used	File Location	Utility Needed
Excel	Apache POI	**src/test/resources/TestData.xlsx**	**ExcelUtils.java**
CSV	OpenCSV	**src/test/resources/TestData.csv**	**CSVUtils.java**

Would you like me to:

- ✅ Generate a mini Maven project with both Excel & CSV examples?
- ✅ Add JSON file-based data-driven testing?
- ✅ Include negative test cases and validations?

Let me know!

Looping through test cases using **Java logic** is useful when you:

- Want to avoid writing the same test code repeatedly.
- Have test inputs inside a Java collection (like `List`, `Map`, or `Array`).
- Are not using TestNG/JUnit data providers, but still want to test multiple scenarios.

✅ 1. Looping Over Data in a Basic Java Loop (No DataProvider)

🟦 Example: Looping Through Users and Sending POST Requests

```
java

import io.restassured.RestAssured;
import io.restassured.response.Response;

import java.util.*;

import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

public class LoopingTestCases {

    public static void main(String[] args) {
        // List of test data (name, job)
        List<Map<String, String>> users = new ArrayList<>();

        Map<String, String> user1 = new HashMap<>();
        user1.put("name", "Alice");
        user1.put("job", "Dev");

        Map<String, String> user2 = new HashMap<>();
        user2.put("name", "Bob");
        user2.put("job", "QA");

        Map<String, String> user3 = new HashMap<>();
        user3.put("name", "Charlie");
        user3.put("job", "Manager");

        users.add(user1);
        users.add(user2);
        users.add(user3);

        RestAssured.baseURI = "https://reqres.in";

        // Loop over each user and perform POST request
        for (Map<String, String> user : users) {
            String name = user.get("name");
            String job = user.get("job");

            System.out.println("🔄 Testing with: " + name + ", " + job);

            String requestBody = "{ \"name\": \"" + name + "\", \"job\": \"" + job + "\" }";

            Response response = given()
                    .header("Content-Type", "application/json")
                    .body(requestBody)
                    .when()
                    .post("/api/users");

            response.then()
                    .statusCode(201)
                    .body("name", equalTo(name))
                    .body("job", equalTo(job));

            System.out.println("✅ Test passed for " + name + "\n");
        }
```

```
    }
}
```

🔁 2. Using Arrays Instead of Map

```
java

String[][] userData = {
    {"Alice", "Developer"},
    {"Bob", "Tester"},
    {"Charlie", "Manager"}
};

for (String[] user : userData) {
    String name = user[0];
    String job = user[1];

    String requestBody = "{ \"name\": \"" + name + "\", \"job\": \"" + job + "\" }";

    given()
        .header("Content-Type", "application/json")
        .body(requestBody)
    .when()
        .post("https://reqres.in/api/users")
    .then()
        .statusCode(201)
        .body("name", equalTo(name))
        .body("job", equalTo(job));
}
```

🧠 When to Use This Style

Use Case	Recommended?
Quick iteration in **`main()`**	✅ Yes
Replacing **`@DataProvider`**	❌ Not ideal
Internal test utility logic	✅ Yes
CI/CD automation testing	❌ Use TestNG

📌 Summary

- You can **loop through lists, arrays, or maps** for testing different inputs.
- Ideal for **simple one-off test runs**.
- For **reporting, assertions, and scalability**, use TestNG or JUnit.

Would you like me to:

- ✅ Combine this with Excel or CSV input?
- ✅ Wrap it into a proper TestNG suite for CI/CD?
- ✅ Add logging or reporting for looped test cases?

Let me know!