REST Assured Overview

REST Assured is a popular open-source **Java library** used for **testing RESTful APIs**. It provides a simple and readable syntax to send HTTP requests (like GET, POST, PUT, DELETE) and validate responses, making it easier to automate API testing without writing a lot of boilerplate code.

🔍 Why Use REST Assured?

- It's **built on top of Java**, perfect for integration with existing Java projects.
- Supports **BDD-style syntax** (`given()`, `when()`, `then()`), making test cases more readable.
- Helps validate **status codes**, **response bodies**, **headers**, **cookies**, and **JSON/XML responses**.
- Can be integrated with **JUnit**, **TestNG**, and **Maven/Gradle**.

✅ Key Features

Feature	Description
BDD Syntax	Readable and expressive testing style
JSON/XML Support	Easy parsing and validation of API responses
Request Specification	Set headers, params, auth, and body easily
Response Validation	Assert status codes, body content, headers, etc.
Authentication Support	Supports OAuth, Basic, Digest, Form authentication
Integration	Works well with JUnit, TestNG, Maven, Jenkins, etc.

🔧 Basic Syntax Example

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

public class GetAPITest {
    public static void main(String[] args) {
        given()
            .baseUri("https://jsonplaceholder.typicode.com")
        .when()
            .get("/posts/1")
        .then()
            .statusCode(200)
            .body("userId", equalTo(1))
            .body("id", equalTo(1));
    }
}
```

📦 Maven Dependency

Add this to your `pom.xml`:

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

Would you like a hands-on REST Assured API testing lab or mini-project to practice?

🚀 Why Use REST Assured for API Automation?

REST Assured is **widely used in API testing** for Java-based test automation projects. Here's **why it's a go-to tool** for many QA engineers and developers:

✅ 1. Simplifies API Testing in Java

Without REST Assured, testing APIs in Java requires writing **low-level HTTP client code** (like using **`HttpClient`**, **`URLConnection`**, etc.). REST Assured abstracts this complexity using a **concise and fluent API**.

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

✅ 2. BDD Style Readability

REST Assured follows the **Given–When–Then (Gherkin-like)** syntax, making test cases more expressive and readable even for non-developers:

```java
given()
    .baseUri("https://api.example.com")
    .header("Authorization", "Bearer token")
.when()
    .get("/users/1")
.then()
    .statusCode(200)
    .body("name", equalTo("John"));
```

✅ 3. Built-in JSON & XML Parsing

- Easily parse and validate JSON and XML responses.
- No need for external libraries like Jackson or Gson.

```java
.body("data[0].email", equalTo("test@example.com"))
```

✅ 4. Supports Authentication Mechanisms

REST Assured has built-in support for:

- Basic Auth
- Digest Auth
- Form-based Auth
- OAuth 1.0 & 2.0

Example:

```java
auth().basic("user", "password")
```

✅ 5. Seamless Integration with Test Frameworks

- Easily integrates with **JUnit**, **TestNG**, and **Cucumber**.
- Fits well into CI/CD pipelines like Jenkins, GitLab, or GitHub Actions.

✅ 6. Request and Response Specification Reusability

You can define common request settings (base URI, headers, etc.) once and reuse them, reducing duplication:

```java
RequestSpecification reqSpec = given().baseUri("https://api.example.com").header("Content-Type", "application/json");
```

✅ 7. Open Source & Actively Maintained

- REST Assured is open-source and has an active community.
- Well-documented with lots of tutorials and examples.

✅ 8. Supports Complex Validations

Easily perform:

- Status code and header assertions
- JSON path and XPath expressions
- Array size and nested key validations

Summary:

Benefit	Why It Matters
Simple Syntax	Less boilerplate, faster test writing
BDD Style	Improves readability and collaboration
Built-in Auth & Parsing	No need for extra libraries
Java-native	Perfect for Java/TestNG/JUnit projects
Integrates with CI/CD	Great for automated pipelines

Would you like to see a real-time mini-project or framework setup using REST Assured + TestNG or JUnit?

⭐ Features of REST Assured

REST Assured is packed with powerful features to simplify REST API automation, especially in Java-based test automation frameworks.

Here's a breakdown of its key features:

✅ 1. Fluent BDD-Style Syntax

- Uses **`given()`**, **`when()`**, **`then()`** structure (inspired by Gherkin).
- Makes test scripts easy to write and read.

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

✅ 2. Built-in Support for HTTP Methods

- Supports all standard HTTP methods:
  - **`GET`**
  - **`POST`**
  - **`PUT`**
  - **`PATCH`**
  - **`DELETE`**
  - **`OPTIONS`**
  - **`HEAD`**

✅ 3. Request Specification

- Easily configure and reuse:
  - Base URI
  - Headers
  - Query parameters
  - Path parameters
  - Body content
  - Cookies
- Supports file uploads and multi-part requests.

✅ 4. Response Validation

- Validate:

- Status codes
- Response time
- Headers
- JSON/XML body content
- Cookies

```java
.then().statusCode(200).body("id", equalTo(101));
```

✅ 5. JSON & XML Parsing

- Built-in support for:
  - **JSONPath** (similar to XPath for JSON)
  - **XPath** (for XML)

```java
.body("data[0].email", equalTo("test@example.com"))
```

✅ 6. Authentication Support

- Supports multiple authentication schemes:
  - Basic Auth
  - Digest Auth
  - Form-based
  - OAuth 1.0 and OAuth 2.0
  - Preemptive/Challenge-based authentication

```java
auth().basic("username", "password")
```

✅ 7. Logging

- Log requests and responses for debugging:

```java
log().all(); // Logs full request/response
```

✅ 8. Filters and Hooks

- Create custom filters (e.g., for logging, reporting).
- Intercept requests/responses to enhance or validate dynamically.

✅ 9. Session and Cookie Handling

- Easily manage session IDs and cookies during testing.

✅ 10. Schema Validation

- Validate responses against **JSON Schema** or **XML Schema**.

```java
.body(matchesJsonSchemaInClasspath("user-schema.json"))
```

✅ 11. Integration with Java Test Frameworks

- Works seamlessly with:
  - **JUnit**
  - **TestNG**
  - **Cucumber**
- Can be part of BDD test flows using Gherkin + Step Definitions.

✅ 12. Support for Data-Driven Testing

- Combine REST Assured with tools like:
  - Excel/CSV
  - JSON files
  - Apache POI
  - TestNG data providers

✅ 13. Open Source and Well-Maintained

- Free to use
- Supported by a large developer/testing community

📌 Summary Table

Feature	Description
Fluent BDD Syntax	**`given-when-then`** style makes code readable
HTTP Method Support	Full range of REST operations

Feature	Description
JSON/XML Validation	Built-in parsers using JSONPath/XPath
Authentication	Supports Basic, OAuth, Form, Digest, etc.
Schema Validation	Ensures API contract using JSON/XML schemas
Logging	Debug with request/response logs
Reusability	Request/Response Specs and filters
Framework Integration	Easily works with JUnit, TestNG, Cucumber
CI/CD Friendly	Easily plugged into Jenkins/GitHub Actions pipelines

---

Let me know if you'd like a REST Assured mini framework structure or sample project setup using Maven + TestNG.

🔍 REST Assured vs Postman – A Detailed Comparison

REST Assured and Postman are both popular tools used for **API testing**, but they serve different purposes and are used in different contexts. Here's a complete comparison:

🧠 Overview

Tool	Type	Usage Focus
Postman	GUI-based tool	Manual and automation-friendly API testing
REST Assured	Java library (code-based)	Automated API testing in Java frameworks

⚙️ Comparison Table

Feature	Postman	REST Assured
Interface	Graphical UI	Code-based (Java)
Best Use Case	Manual testing, quick checks, team collaboration	Automated testing in CI/CD pipelines
Test Language	Postman scripting (JavaScript)	Java
Automation Support	Via **Newman CLI**	Directly within **Java frameworks** (JUnit, TestNG)
CI/CD Integration	Yes (with Newman in Jenkins, GitLab, etc.)	Yes (TestNG/JUnit integrated with Jenkins, etc.)
Ease of Use	Very beginner-friendly	Requires Java knowledge
Data-Driven Testing	Built-in support with CSV/JSON	Via Java + Excel/CSV + TestNG data providers
Authentication Support	Built-in UI (Basic, OAuth, Bearer, etc.)	Supported via Java methods
Response Validation	Basic JS assertions (Chai syntax)	Advanced validation using Hamcrest/AssertJ
Schema Validation	Yes	Yes
Collaboration	Workspace sharing, versioning, comments	Needs Git or SCM integration
Learning Curve	Low	Moderate (Java required)
Mock Servers	Yes (Postman mock server support)	No
API Documentation	Can auto-generate from collections	No built-in support

✅ When to Use Postman

- You are doing **exploratory** or **manual API testing**
- Your team includes **non-programmers**
- You need to **quickly test endpoints**
- You want to **share APIs**, **mock responses**, or **generate documentation**

✅ When to Use REST Assured

- You are building an **automated test framework**
- You need **integration with Java-based tools** like JUnit, TestNG
- You want to **run tests in CI/CD** pipelines regularly
- You are doing **data-driven** or **contract-based testing**
- You need **fine-grained validation logic**

📌 Summary

Use Case	Best Tool
Manual/Exploratory Testing	**Postman**
API Collaboration & Mocking	**Postman**
Java-based Test Automation	**REST Assured**
CI/CD Automated Testing (Java)	**REST Assured**
Quick Checks Without Code	**Postman**
Complex Validations and Reuse	**REST Assured**

Would you like a side-by-side demo or lab example of the same API tested in both Postman and REST Assured?

Absolutely! In the **banking domain**, REST Assured can be used to automate testing of various RESTful API operations that simulate **real-world banking scenarios** like:

🏦 Real-World REST Assured Use Cases in Banking

✅ 1. Login/Authentication API

Used to verify whether user credentials are valid and to receive an authentication token.

Example:

```java
given()
    .baseUri("https://api.bankexample.com")
    .contentType("application/json")
    .body("{ \"username\": \"user123\", \"password\": \"securePass\" }")
.when()
    .post("/auth/login")
.then()
    .statusCode(200)
    .body("token", notNullValue());
```

✅ 2. Check Account Balance

Verify that the correct balance is returned for a valid account using an auth token.

```java
given()
    .baseUri("https://api.bankexample.com")
    .header("Authorization", "Bearer " + token)
.when()
    .get("/accounts/123456789/balance")
.then()
    .statusCode(200)
    .body("balance", greaterThan(0.0f));
```

✅ 3. Fund Transfer Between Accounts

Validate fund transfer with proper body request and check for updated balances.

```java
String transferBody = "{ \"fromAccount\": \"123456789\", \"toAccount\": \"987654321\",
\"amount\": 500.0 }";

given()
    .baseUri("https://api.bankexample.com")
    .header("Authorization", "Bearer " + token)
    .contentType("application/json")
    .body(transferBody)
.when()
    .post("/transactions/transfer")
.then()
    .statusCode(200)
    .body("transactionId", notNullValue())
    .body("status", equalTo("SUCCESS"));
```

✅ 4. Transaction History API

Test retrieval of transactions over a time range.

```java
given()
    .baseUri("https://api.bankexample.com")
    .header("Authorization", "Bearer " + token)
    .queryParam("accountId", "123456789")
    .queryParam("fromDate", "2025-01-01")
    .queryParam("toDate", "2025-07-01")
.when()
    .get("/transactions/history")
.then()
    .statusCode(200)
    .body("transactions.size()", greaterThan(0));
```

✅ 5. Bill Payment

Simulate payment for utilities, loans, etc.

```java
given()
    .baseUri("https://api.bankexample.com")
    .header("Authorization", "Bearer " + token)
    .contentType("application/json")
    .body("{ \"biller\": \"Electricity\", \"accountId\": \"123456789\", \"amount\": 1200.00 }")
.when()
    .post("/bills/pay")
.then()
    .statusCode(200)
    .body("paymentStatus", equalTo("PAID"));
```

✅ 6. Negative Test – Invalid Login

```java
given()
    .baseUri("https://api.bankexample.com")
    .contentType("application/json")
    .body("{ \"username\": \"wrongUser\", \"password\": \"wrongPass\" }")
.when()
    .post("/auth/login")
.then()
    .statusCode(401)
    .body("error", containsString("Invalid credentials"));
```

📦 You Can Also Automate:

Operation	API Test Scenario
Forgot Password	Validate OTP/email link workflow
Loan Application	Submit and validate loan eligibility
Credit Card Transactions	Validate swipes, EMI conversions
User Profile Update	Validate name, address, contact changes
Statement Download	Validate PDF/Excel download response type

🔄 Integration Possibilities

- Combine with **TestNG or JUnit** for full automation suite.
- Data-driven testing using **Excel/CSV** for multiple accounts/transactions.
- Integrated with **Jenkins** for nightly regression runs.
- Store auth tokens with **RequestSpecification** to reuse across tests.

Would you like a **mini-banking automation project structure** with folders like:

- `tests/`
- `utils/`
- `auth/`
- `transactions/`
- `account/`
  using REST Assured and TestNG or JUnit?