Introduction to API Testing

API Testing is a software testing practice that focuses on verifying Application Programming Interfaces (APIs) directly. Unlike UI testing, API testing checks the business logic, data responses, performance, and security at the service layer.

Key Points:

- Ensures that APIs return correct responses for valid and invalid requests
- Validates integration between different software systems
- Covers functionality, reliability, performance, and security
- Faster and more reliable than UI testing, enabling early defect detection
- API Testing is essential in modern **automation frameworks**, especially for microservices and cloud-based applications.

Why Rest Assured for Automation ?

Java-based & Easy Integration

Built on top of Java, integrates seamlessly with frameworks like TestNG, JUnit, Maven, Gradle, and CI/CD tools (Jenkins).

Simple & Readable Syntax

Provides a BDD-style (Given-When-Then) approach, making API tests more readable and maintainable.

Supports All HTTP Methods

GET, POST, PUT, DELETE, PATCH, and HEAD requests are easy to implement.

Rich Validation & Assertions

Inbuilt support for validating response codes, headers, cookies, and JSON/XML body with minimal code.

JSON & XML Support

Handles both JSON and XML payloads efficiently without needing extra libraries.

Open-Source & Community Support

Actively maintained, with large community and documentation available.

Perfect for Automation Frameworks

Can be easily integrated into hybrid, data-driven, or BDD frameworks for enterprise-level automation.

What is Rest Assured?

- Rest Assured is an open-source Java library used for testing and validating RESTful APIs.
- Built on top of Java + HTTP Client + JSON parsers, it makes API automation simple and powerful.
- It provides a domain-specific language (DSL) for writing readable and maintainable API tests.

What is a RESTful API ?

API (Application Programming Interface)

An API is like a bridge that allows two software systems to talk to each other.
Example: Your mobile app talks to a backend server via APIs.

- REST (Representational State Transfer) is an architectural style for designing APIs.
- It uses HTTP methods (GET, POST, PUT, PATCH, DELETE).
- Data is usually exchanged in JSON or XML.
- It is stateless → each request is independent.
- An API that follows REST principles is called a RESTful API.

Principles of RESTful API:

- Stateless – Server does not store client state.
- Client-Server – Clear separation between frontend (client) and backend (server).
- Uniform Interface – Standard way of accessing resources.
- Cacheable – Responses can be cached for better performance.
- Layered System – APIs can be designed in layers (security, load balancer, etc).

What is a Domain-Specific Language (DSL)?

- A Domain-Specific Language (DSL) is a mini-language designed for a specific purpose (domain).
- Unlike general-purpose programming languages (Java, Python), a DSL focuses on making tasks in its domain simpler, more readable, and expressive.

- In Rest Assured's case:
- The domain is API testing.
- Instead of writing long boilerplate Java code for HTTP calls, Rest Assured provides a DSL (specialized syntax) that makes the test look almost like plain English.

Example without DSL (Java + HTTPClient style):

```
HttpClient client = HttpClient.newHttpClient();
HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create("https://reqres.in/api/users?page=2"))
        .build();
HttpResponse<String> response = client.send(request,
HttpResponse.BodyHandlers.ofString());

System.out.println(response.body());
```

Example with DSL in Rest Assured:

```
given()
    .baseUri("https://reqres.in")
.when()
    .get("/api/users?page=2")
.then()
    .statusCode(200);
```

See the difference?

- The Rest Assured DSL makes it look like a sentence:
- “Given a base URI, when I send a GET request, then the response should have status 200.”

So in simple words:

- DSL = A way of writing code that feels like natural language, specialized for a particular job.
- In Rest Assured, DSL = Writing API tests in a human-readable and maintainable way.

CRUD Operations in API Testing Using Rest Assured with Java

What is CRUD ?

- CRUD stands for Create, Read, Update, Delete.
- These are the four basic operations performed on data in APIs or databases.

Key Features:

- Supports GET, POST, PUT, PATCH, DELETE requests (CRUD operations).
- Built-in support for JSON and XML.
- Easy integration with TestNG, JUnit, Cucumber.
- Supports authentication (OAuth, OAuth2, Basic, Bearer tokens).
- Provides BDD-style syntax (given(), when(), then()).
- Rich assertion capabilities with Hamcrest matchers.

Maven Dependency

```
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>5.5.0</version>
 </dependency>
```

Assertions:

```
<dependency>
   <groupId>org.hamcrest</groupId>
   <artifactId>hamcrest</artifactId>
   <version>2.2</version>
</dependency>
```

Create (POST)

- Adds new data (e.g., add a user)
- HTTP Method: POST

- Example (Rest Assured):

```
given()
.contentType("application/json")
.body("{\"name\":\"Anil\",\"job\":\"QA\"}")
.when()
.post("/users")
.then()
.statusCode(201);
```

Read (GET)

- Fetch existing data (e.g., user details)
- HTTP Method: GET

- Example (Rest Assured):

```
given()
.when()
.get("/users/2")
.then()
.statusCode(200);
```

Update (PUT/PATCH)

- Modify existing data
- PUT → Update full record
- PATCH → Update partial record

- Example (PUT):

```
given()
.contentType("application/json")
.body("{\"name\":\"Anil\",\"job\":\"Lead QA\"}")
.when()
.put("/users/2")
.then()
.statusCode(200);
```

Delete (DELETE)

- Remove data (e.g., delete user)
- HTTP Method: DELETE

- Example (Rest Assured):

```
given()
.when()
.delete("/users/2")
.then()
.statusCode(204);
```

CRUD Summary

- C → Create → POST
- R → Read → GET
- U → Update → PUT/PATCH
- D → Delete → DELETE

- CRUD ensures full coverage of API functionality.

Rest Assured Examples – API Testing in Java

- Subtitle: Query Params, Path Params, Headers, Logging, Pagination
- Query Parameters:

```
Response response = RestAssured.given()
        .queryParam("page",2)
        .when()
        .get("https://jsonplaceholder.typicode.com/users")
        .then()
        .statusCode(200)
        .extract().response();
```

Explanation:

Used to filter/search resources.

Example: ?page=2 fetches the 2nd page of users

Path Parameters

```
Response response = RestAssured.given()
        .pathParam("userId",2)
        .when()
        .get("/users/{userId}")
       .then().statusCode(200).extract().response();
```

Explanation:

Insert values directly into endpoint path.

Example: /users/2 fetches user with ID 2

Adding Headers

```
Response response = RestAssured.given()
    .header("Content-Type","application/json")
    .header("Authorization","Bearer token123")
    .when()
   .get("https://jsonplaceholder.typicode.com")
    .then().extract().response();
```

Explanation:

Headers provide metadata (Content-Type, Authorization).

Required for auth & content negotiation.

Logging Requests & Responses

```
Response response = RestAssured.given()
    .log().all()
    .when()
  .get("https://jsonplaceholder.typicode.com")
    .then().extract().response();
```

Explanation:

- log().all() prints full request & response.
- Helpful for debugging tests.

Extract Example

```
Response response = RestAssured.given()
    .get("https://reqres.in/api/users/2")
    .then().extract().response();
System.out.println(response.jsonPath().getInt("data.id"));
```

Explanation:

- extract() gets Response object.
- JSONPath helps retrieve specific fields (like id).

Limit Example

```
Response response = RestAssured.given()
        .queryParam("limit",5)
        .when()
.get("https://dummy.restapiexample.com/api/v1/employees")
        .then().extract().response();
```

- Explanation:
- Query parameter limit=5 returns only 5 employees.
- Useful for large dataset optimization.

Pagination Example

```
Response response = RestAssured.given()
    .queryParam("page",2)
    .queryParam("per_page",5)
    .when()
    .get("https://gorest.co.in/public/v2/users")
    .then().extract().response();
```

Explanation:

- page=2 → second page of results.
- per_page=5 → only 5 users per page.