When performing **API testing**, the goal is to ensure that the API works correctly, securely, and efficiently. Here's what you typically validate during API testing:

✅ 1. Functional Validation

- **Status Code**: Validate correct HTTP response codes (e.g., `200 OK`, `201 Created`, `400 Bad Request`, `404 Not Found`, `500 Internal Server Error`).
- **Response Body**: Verify the response structure and content matches expected schema or data.
- **Request Methods**: Check the API supports correct methods (GET, POST, PUT, DELETE, etc.).
- **Business Logic**: Ensure API operations perform the expected actions (e.g., creating a user actually adds it to the database).

✅ 2. Data Validation

- **Input Validation**: Verify API handles valid and invalid inputs appropriately (e.g., missing fields, incorrect data types).
- **Data Accuracy**: Check that the data returned matches what's expected in the DB or from prior requests.
- **CRUD Operations**: Ensure all Create, Read, Update, Delete operations work as intended.

✅ 3. Security Testing

- **Authentication**: Confirm that endpoints require proper authentication (e.g., API keys, OAuth tokens).
- **Authorization**: Validate role-based access control (e.g., regular user shouldn't access admin data).
- **Data Privacy**: Ensure sensitive data is masked or encrypted (e.g., no plain text passwords).

✅ 4. Performance Testing

- **Response Time**: API should respond within an acceptable time (typically < 1-2 seconds).
- **Load Testing**: Check how the API handles concurrent users or high volumes.
- **Rate Limiting**: Validate how the API behaves when request limits are exceeded.

✅ 5. Error Handling

- **Error Codes**: Ensure proper HTTP error codes are returned for invalid requests.
- **Error Messages**: Check error messages are informative, consistent, and do not leak sensitive info.

✅ 6. Headers & Cookies

- **Content-Type**: e.g., `application/json` or `application/xml`.
- **CORS headers**: Especially for browser-based clients.
- **Authorization/Session cookies**: Validate they are secure and used appropriately.

✅ 7. Schema Validation

- Use tools like **JSON Schema** or **Swagger/OpenAPI** to validate:
  - Field types
  - Required vs optional fields
  - Enum values, min/max length, etc.

✅ 8. Integration Testing

- **Workflow Tests**: Chain multiple API calls to validate real user scenarios (e.g., login → create item → fetch item).
- **3rd-party APIs**: If your API interacts with external services, validate those integrations as well.

🛠️ Tools Commonly Used:

- **Postman** – Manual and automated API testing
- **RestAssured / Karate / HTTPClient** – Java-based automation
- **Python Requests / Pytest** – Python automation
- **Swagger / OpenAPI** – Contract validation
- **WireMock / Mountebank** – Mocking APIs for isolation

Would you like sample validation code in Postman or Java/Python?

You said:

How do you validate multifactor authentications

ChatGPT said:

Validating **Multi-Factor Authentication (MFA)** in API testing involves confirming that the system correctly enforces and processes multiple authentication steps (typically **something you know** + **something you have**). Here's how you approach it:

✅ MFA Validation Steps

1. Initiate Login (Step 1: Username/Password)

- **Send**: `POST /login` with username and password.
- **Validate**:
  - API returns a 200 or 202 indicating successful first-factor verification.
  - API indicates that MFA is required.
  - Response includes a session/token or a challenge ID (depending on the implementation).

2. MFA Challenge (Step 2: OTP, TOTP, SMS, App Code)

- **Send**: `POST /mfa/verify` with the MFA code (from SMS, Authenticator app, etc.) and session/challenge ID.
- **Validate**:
  - Status code `200 OK` or `204 No Content` on success.
  - Token or session ID is issued for access to protected endpoints.
  - Proper error codes (`401`, `403`) on incorrect or expired codes.

🔐 Common MFA Methods You Might Test

MFA Type	How to Validate
Email OTP	Check email is triggered, OTP is accepted or expired appropriately
SMS OTP	Mock SMS service (e.g., Twilio sandbox or intercept) and validate OTP
Authenticator App (TOTP)	Use a shared secret to generate a valid code (using tools like `pyotp` for automation)
Push Notification	Simulate approval/rejection if possible via mocks/stubs or manual trigger
Security Questions	Validate answers and error paths
Biometric/Hardware Token (e.g., YubiKey)	Often tested manually or using mock services

✅ Security Validations

- Ensure MFA code is valid **only once** or expires after a short time.
- **Rate limit** MFA attempts (e.g., lockout after 5 wrong attempts).
- **Secure transmission** – MFA codes should never be sent via query params (only body).
- Tokens issued after MFA should have **different scopes** or be more privileged.

- Check that **MFA is required again** after session expiration or risky login attempts.

🔄 Automation Tips

- **Mock MFA code generation** (e.g., intercept OTP via logs or use TOTP generators).
- Use **test environments** where MFA logic is controllable or has test hooks.
- In **Karate or Python**, you can automate TOTP like this:

```
python
CopyEdit
# Python + pyotp example
import pyotp
totp = pyotp.TOTP("BASE32SECRETKEY")
code = totp.now()
print("Current MFA code:", code)
```

🧪 Tools

- **Postman Pre-request Scripts**: for generating TOTP dynamically
- **PyTest/Requests + pyotp**: for full API test automation
- **WireMock/Mountebank**: for mocking SMS/Email services
- **Appium**: if MFA includes mobile interactions (e.g., push notifications)

Let me know your exact MFA setup (TOTP, SMS, email, etc.), and I can tailor the validation or test script accordingly.

You said:

example http code + api testing

ChatGPT said:

Here's a concise and practical example that shows **API testing with HTTP status code validation** using **Postman** (manual) and **Java + RestAssured** (automated):

✅ Scenario: Create a User API

Endpoint: `POST /api/users`
Payload:

```
json
CopyEdit
{
  "name": "Anand",
  "job": "Tester"
}
```

Expected Response:

- **Status Code**: `201 Created`
- **Response Body**:

```
json
CopyEdit
{
  "name": "Anand",
  "job": "Tester",
  "id": "123",
  "createdAt": "2025-06-24T12:34:56.789Z"
}
```

🔷 1. Postman Test Example

✅ Setup

- Method: `POST`
- URL: `https://reqres.in/api/users` (public dummy API)
- Body (raw → JSON):

```
json
CopyEdit
{
  "name": "Anand",
  "job": "Tester"
```

```
}
```

🔍 Tests Tab (add this JS code):

```
javascript
CopyEdit
pm.test("Status code is 201", function () {
    pm.response.to.have.status(201);
});

pm.test("Response has id and createdAt", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData).to.have.property("id");
    pm.expect(jsonData).to.have.property("createdAt");
});
```

🔷 2. Java + RestAssured Example

🧪 Maven Dependencies (add to `pom.xml`)

```
xml
CopyEdit
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>5.3.0</version>
    <scope>test</scope>
</dependency>
```

✅ Java Test Code

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

public class CreateUserTest {

    @Test
```

```
    public void createUserTest() {
        String requestBody = "{ \"name\": \"Anand\",
\"job\": \"Tester\" }";

        Response response = RestAssured.given()
            .baseUri("https://reqres.in")
            .basePath("/api/users")
            .header("Content-Type", "application/json")
            .body(requestBody)
            .post();

        // Status Code Validation
        Assert.assertEquals(response.getStatusCode(), 201,
"Expected status code 201");

        // Body Field Validation
        String id = response.jsonPath().getString("id");
        String createdAt =
response.jsonPath().getString("createdAt");

        Assert.assertNotNull(id, "User ID should not be
null");
        Assert.assertNotNull(createdAt, "createdAt should
not be null");
    }
}
```

🔍 Status Codes to Validate in API Testing

Code	Meaning	When to Expect
200	OK	Successful GET/PUT/DELETE request
201	Created	Successful POST request
204	No Content	DELETE success with no response body
400	Bad Request	Invalid input or missing parameters
401	Unauthorized	Missing/invalid auth token

403	Forbidden	No permission for resource
404	Not Found	Endpoint or resource doesn't exist
500	Internal Server Error	Server error or exception

Would you like the same in **Python** or **Karate** as well?

You said:

API example with get ,post,put,patch and delete method in framework

ChatGPT said:

Here's a complete **API Testing Framework Example** in **Java using RestAssured + TestNG** covering all major HTTP methods:

- `GET` – Retrieve user
- `POST` – Create user
- `PUT` – Update user (full update)
- `PATCH` – Update user (partial update)
- `DELETE` – Delete user

You can use https://reqres.in as a dummy API for testing.

✅ Project Structure

```
bash
CopyEdit
api-framework/
├── pom.xml
└── src/
    └── test/
        └── java/
```

```
└── tests/
    ├── BaseTest.java
    └── UserApiTest.java
```

🔧 `pom.xml` Dependencies

xml
CopyEdit

```xml
<dependencies>
    <dependency>
        <groupId>io.rest-assured</groupId>
        <artifactId>rest-assured</artifactId>
        <version>5.3.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.testng</groupId>
        <artifactId>testng</artifactId>
        <version>7.8.0</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

🔷 `BaseTest.java`

java
CopyEdit

```java
package tests;

import io.restassured.RestAssured;
import org.testng.annotations.BeforeClass;

public class BaseTest {
    @BeforeClass
    public void setup() {
        RestAssured.baseURI = "https://reqres.in";
    }
}
```

🔷 UserApiTest.java

java
CopyEdit

```
package tests;

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

import static org.testng.Assert.*;
import static io.restassured.RestAssured.*;

public class UserApiTest extends BaseTest {

    String userId;

    @Test(priority = 1)
    public void testPost_CreateUser() {
        String requestBody = "{ \"name\": \"Anand\",
\"job\": \"Tester\" }";

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

        assertEquals(response.getStatusCode(), 201);
        userId = response.jsonPath().getString("id");
        System.out.println("Created User ID: " + userId);
    }

    @Test(priority = 2)
    public void testGet_User() {
        Response response = get("/api/users/2");
        assertEquals(response.getStatusCode(), 200);

assertEquals(response.jsonPath().getString("data.first_name
"), "Janet");
    }
```

```
    @Test(priority = 3)
    public void testPut_UpdateUser() {
        String updateBody = "{ \"name\": \"Anand Updated\",
\"job\": \"SDET\" }";

        Response response = given()
                .header("Content-Type", "application/json")
                .body(updateBody)
                .put("/api/users/" + userId);

        assertEquals(response.getStatusCode(), 200);

assertTrue(response.getBody().asString().contains("Anand
Updated"));
    }

    @Test(priority = 4)
    public void testPatch_UpdateUser() {
        String patchBody = "{ \"job\": \"Senior SDET\" }";

        Response response = given()
                .header("Content-Type", "application/json")
                .body(patchBody)
                .patch("/api/users/" + userId);

        assertEquals(response.getStatusCode(), 200);

assertTrue(response.getBody().asString().contains("Senior
SDET"));
    }

    @Test(priority = 5)
    public void testDelete_User() {
        Response response = delete("/api/users/" + userId);
        assertEquals(response.getStatusCode(), 204); // No
Content
    }
}
```

✅ To Run the Tests:

```
bash
CopyEdit
mvn test
```

📌 Summary of Endpoints Tested

Method	Endpoint	Purpose
POST	`/api/users`	Create new user
GET	`/api/users/2`	Get user info
PUT	`/api/users/{id}`	Full update
PATCH	`/api/users/{id}`	Partial update
DELETE	`/api/users/{id}`	Delete user

Would you like this in **Python**, **Karate**, or **Cypress API plugin** version too?

You said:

what is token here
2/2
ChatGPT said:

In **API testing**, a **token** usually refers to an **authentication token**—a secure, unique string issued by a server to a client to **identify and authorize** the user for subsequent API requests.

✅ What is a Token?

A **token** is:

- A **digital key** that grants access to protected resources.
- Issued after successful login (e.g., username & password).
- Sent in API requests to prove the user's identity.

🔐 Types of Tokens in API Testing

Token Type	Description
Bearer Token	Common in OAuth 2.0. Sent via `Authorization: Bearer <token>` header.
JWT (JSON Web Token)	Compact, URL-safe token containing encoded user data and signature.
API Key	A simple token string tied to the client or app, used for basic authentication.
Session Token	A temporary token representing a user session, usually stored in cookies or headers.

📦 Example: How Tokens Work

1. 🔑 Login (get the token)

Request:

```
http
CopyEdit
POST /api/login
Content-Type: application/json

{
  "username": "anand",
  "password": "mypassword"
}
```

Response:

```
json
CopyEdit
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5..."
}
```

2. 🔐 Use token in authorized requests

Request:

```
http
CopyEdit
GET /api/user/profile
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5...
```

Response:

```
json
CopyEdit
{
  "id": 101,
  "name": "Anand"
}
```

✅ Why You Validate Tokens in API Testing

- Ensure the token is issued correctly (after login).
- Validate expiry, revocation, refresh logic.
- Confirm protected endpoints **require the token** and return `401 Unauthorized` if missing/invalid.
- Check token content (in case of JWT: decode and assert payload fields like userId, role, expiry).

🧪 Token Handling in Automated API Tests (Java + RestAssured)

```
java
CopyEdit
```

```
String token = given()
    .header("Content-Type", "application/json")
    .body("{\"username\":\"anand\", \"password\":
\"pass\"}")
    .post("/api/login")
    .jsonPath()
    .getString("token");

given()
    .header("Authorization", "Bearer " + token)
    .get("/api/protected-endpoint")
    .then()
    .statusCode(200);
```

Let me know if you want to **automatically generate tokens** in Postman or any specific token strategy like **JWT validation** or **refresh token handling**.

You said:

How do you parse Json

ChatGPT said:

Parsing JSON means converting a JSON string into a usable object in your programming language (e.g., Java, Python, JavaScript). Here's how it's done in different contexts—**API testing** included.

🔷 1. Java (RestAssured + JsonPath)

✅ Example: Parse JSON from API Response

```
java
CopyEdit
import io.restassured.path.json.JsonPath;
import io.restassured.response.Response;

Response response = given().get("https://reqres.in/api/
users/2");
JsonPath json = response.jsonPath();
```

```
String firstName = json.getString("data.first_name");
System.out.println("First name: " + firstName);
```

🔷 2. JavaScript (Postman or Node.js)

✅ In Postman

```
javascript
CopyEdit
let jsonData = pm.response.json();
console.log(jsonData.data.first_name);
```

✅ In Node.js

```
javascript
CopyEdit
const response = '{"name":"Anand","role":"QA"}';
const obj = JSON.parse(response);
console.log(obj.name); // Anand
```

🔷 3. Python

✅ Using `json` module

```
python
CopyEdit
import json

json_str = '{"name": "Anand", "role": "QA"}'
data = json.loads(json_str)

print(data['name'])  # Anand
```

✅ From API response (with `requests`)

```
python
CopyEdit
import requests
```

```
response = requests.get('https://reqres.in/api/users/2')
json_data = response.json()

print(json_data['data']['first_name'])  # Janet
```

🔷 4. Using JSON Schema Validators

For validating structure, use libraries like:

- Java: `everit-org/json-schema`
- Python: `jsonschema`
- Postman: `tv4` or built-in schema validation

✅ Common Use-Cases in API Testing

Task	What to Parse
Validate field	`json.get("data.id")`
Loop through array	`json.getList("items")`
Check error message	`json.get("error.message")`
Extract token/session ID	`json.get("token")`

Let me know your language/framework (e.g., Karate, Python, Postman, Java) and I can give you tailored parsing code.

You said:

what are the other validation we can do in api testing apart from validating status code , headers and json body

ChatGPT said:

Great question! In **API testing**, apart from the common validations like **status code**, **headers**, and **JSON body**, there are **many other important validations** that ensure the API behaves correctly under various scenarios.

Here's a complete checklist of **advanced and often overlooked validations**:

✅ Other Important Validations in API Testing

🔐 1. Authentication & Authorization

- Is a token required? Does the API reject unauthorized requests?
- Validate role-based access control (e.g., admin vs user).
- Check if expired/invalid tokens are rejected (`401`, `403`).
- Is sensitive data hidden or masked properly?

⏱️ 2. Response Time / Performance

- Validate the API responds within expected time (e.g., < 1 sec).
- Use tools like Postman monitors or JMeter for load/performance testing.
- Simulate concurrent users (threaded or looped API calls).

🔄 3. Data Consistency / DB Validation

- Compare API response with actual DB records (SQL query/Mongo query).
- Validate create/update/delete operations reflected in the DB.
- Check that no stale or unexpected data is returned.

🧹 4. Data Integrity & Schema Validation

- Validate against a JSON Schema (e.g., OpenAPI, Swagger).
- Required fields, correct data types, min/max length, enum values.

- Validate no unexpected/extra fields are returned (overexposure).

🔁 5. Idempotency Check (for PUT, DELETE)

- Call the same request multiple times — should have no side effects.
- e.g., deleting a user twice should give 204 first, 404 next.

🧪 6. Boundary Value & Negative Testing

- Send too short/long strings, nulls, special characters.
- Send invalid IDs (e.g., `/users/-1` or `/users/abc`).
- Missing or extra parameters.

🔁 7. State Transition Validation

- Create a resource → Fetch it → Update it → Delete it.
- Ensure state changes as expected across each operation.

📉 8. Rate Limiting / Throttling

- Validate API behavior when hitting limits (429 Too Many Requests).
- Check headers like `Retry-After`.

🌐 9. CORS and OPTIONS Preflight (for web clients)

- Verify headers like `Access-Control-Allow-Origin`.
- Ensure `OPTIONS` requests are handled if needed.

📦 10. File Upload/Download Validation

- Check content type, size limits, binary integrity.

- For download, validate `Content-Disposition`, file extension.

📜 11. Logging and Auditing

- Validate whether the API logs correct actions (especially sensitive ones).
- Audit trails — were logs created after API call?

🧯 12. Error Handling & Fault Tolerance

- Trigger backend failure or bad data to ensure graceful failure.
- Check meaningful error messages (without leaking internals).
- Validate custom error structures (e.g., code + message + traceId).

🧰 Tools That Help

Tool	Usage
Postman	Manual testing, scripts, monitors
RestAssured	Java automation
Karate	BDD style API automation
JMeter/Gatling	Load and performance testing
Swagger / JSON Schema Validator	Schema validation
WireMock / Mountebank	Mocking and fault testing

Would you like a **sample API test scenario** demonstrating these validations with a real API?

You said:

SSL ceritifaction handling in API testing

ChatGPT said:

Handling **SSL certificates** in API testing is crucial, especially when testing APIs over **HTTPS**. SSL ensures **encrypted communication**, **trust**, and **authentication** between the client and server.

Here's a full breakdown of **how SSL certificate handling works in API testing** — including how to **test**, **validate**, or **bypass** SSL in tools like **Postman**, **RestAssured**, and **Python**.

✅ What to Test in SSL Certificate Handling

Check	Purpose
🔐 **Valid Certificate**	Ensure HTTPS is configured correctly and certificate is valid (not expired or self-signed).
📅 **Expiration Date**	Validate SSL certificate hasn't expired.
🧾 **Hostname Match**	Domain name in cert must match the API URL.
🧾 **Trusted CA**	Certificate should be issued by a trusted Certificate Authority.
🚫 **Rejection of Invalid SSL**	The client (your test) should fail or warn if cert is invalid unless explicitly allowed.

🛠️ Tool-Specific Handling

🔷 1. Postman

✅ Validate SSL by default

- Postman **validates SSL certificates** by default.
- If using a **self-signed certificate**, Postman will throw an error.

🛑 To Ignore SSL:

- Go to `Settings → General → SSL certificate verification → OFF`

🔷 2. Java + RestAssured

✅ **Default: SSL errors will throw exceptions.**

🛠️ **Bypass SSL Certificate Validation (e.g., for self-signed certs):**

```
java
CopyEdit
RestAssured.useRelaxedHTTPSValidation();
```

✅ **For Custom Trust Store (if needed):**

```
java
CopyEdit
System.setProperty("javax.net.ssl.trustStore", "path/to/
truststore.jks");
System.setProperty("javax.net.ssl.trustStorePassword",
"password");
```

🔷 3. Python (requests)

✅ **Default behavior:**

- Will **validate SSL**.
- Will throw an exception if certificate is invalid or expired.

🛠️ **To bypass SSL (not recommended in production):**

```
python
CopyEdit
import requests
response = requests.get("https://self-signed.badssl.com/",
verify=False)
print(response.status_code)
```

⚠️ `verify=False` bypasses validation — **use only in dev/test** environments.

🔷 4. CURL (CLI testing)

```
bash
CopyEdit
curl https://api.example.com
```

🛠️ **Bypass SSL check:**

```
bash
CopyEdit
curl -k https://api.example.com   # -k means --insecure
```

🧪 SSL Certificate Automation Test Example

✅ Python: Check expiry date

```
python
CopyEdit
import ssl, socket
import datetime

hostname = 'api.example.com'
context = ssl.create_default_context()

with socket.create_connection((hostname, 443)) as sock:
    with context.wrap_socket(sock,
server_hostname=hostname) as ssock:
        cert = ssock.getpeercert()
        exp_date =
datetime.datetime.strptime(cert['notAfter'], "%b %d %H:%M:
%S %Y %Z")
        print("SSL expires on:", exp_date)
```

⚠️ Best Practices

Do ✅	Don't ❌
Use HTTPS in all API environments	Use HTTP in prod

Validate SSL cert expiry & hostname	Bypass SSL in production
Use a trusted CA for signing certs	Use self-signed certs unless necessary
Rotate certs before expiry	Let certs expire silently

Would you like a working **SSL expiration checker** integrated in Java or Postman automation?

You said:

i have listed the api testing activities , add me more points and give more info how we test API and what we test API Test: Schema validation (Contract testing) Status codes Verify the error messages Verify the Json body response verify the fields with values Edge case scenarios: Null Empty maximum length of the fields Rate limit/retries DOS - Denial of service authorization/ authentication validate the parametes/ query parameters Headers Cookies Mandatory fields - null/empty

ChatGPT said:

Great list! Here's a **comprehensive enhancement** of your API testing checklist, along with **what we test** and **how we test it** — suitable for interviews or documentation.

✅ Enhanced API Testing Activities

🔷 1. Contract Testing / Schema Validation

- Use OpenAPI/Swagger or JSON Schema to validate:
  - Required fields
  - Field types
  - Enum values
  - Nested objects & arrays
- ✅ Tools: Postman schema validation, RestAssured + JSON schema, Karate, Dredd

🔷 2. Status Code Validation

- Check for correct status codes:
  - `200 OK`, `201 Created`, `204 No Content`
  - `400 Bad Request`, `401 Unauthorized`, `403 Forbidden`, `404 Not Found`, `429 Too Many Requests`, `500+` for server errors
- Validate status codes for both success and failure scenarios

🔷 3. Error Message Validation

- Check error structure (code, message, trace ID)
- Ensure messages are:
  - User-friendly
  - Consistent
  - Do not expose internal system details

🔷 4. JSON Body / Field Validation

- Verify presence and correctness of:
  - Fields and sub-fields
  - Data types
  - Default values
  - Format (email, date, currency)
  - Optional vs mandatory fields

🔷 5. Edge Case / Negative Testing

- Send:

    - Empty payload
    - Null values
    - Overly long strings
    - Special characters
    - Invalid types (e.g., string instead of int)
- Validate graceful error responses

🔷 6. Rate Limiting & Retry Logic

- Trigger rate limits (burst requests)
- Validate response like `429 Too Many Requests`
- Check for `Retry-After` headers
- Simulate retry mechanism (manual or automated with backoff)

🔷 7. Security Testing (DoS, Auth, etc.)

a. DoS / Load Resilience

- Flood API with large payloads or multiple requests
- Observe stability, memory consumption, throttling

b. Authorization & Authentication

- Validate access control:
    - Access token required
    - Token expiration
    - User roles (admin/user)
- Test with:
    - Missing/invalid/expired tokens
    - Role-based access

🔷 8. Parameter & Query Validation

- Test valid and invalid:
  - Path parameters (`/user/123`)
  - Query params (`/users?page=2`)
- Validate encoding (`%20` for space)
- Omit mandatory parameters to test validation

🔷 9. Header Validation

- Check required headers:
  - `Content-Type`, `Accept`, `Authorization`, `X-Request-ID`
- Test with missing or incorrect headers
- Validate custom headers behavior

🔷 10. Cookie Validation

- APIs that use cookies (especially session-based)
- Check `Set-Cookie`, secure flags, expiration, domain restrictions

🔷 11. Mandatory Field Testing

- Send:
  - Missing fields
  - Null values
  - Empty strings
- Validate:
  - Proper 400 Bad Request response

    - Clear error messages indicating missing field

🔷 12. State Transition / CRUD Workflow

- Create → Read → Update → Delete flow
- Validate each operation affects resource as expected
- Use assertions to check data integrity

🔷 13. Idempotency Testing

- Repeat `PUT` or `DELETE` requests
- Ensure no side effects or duplicates
- Validate proper response on second/third call (e.g., 404 after delete)

🔷 14. Versioning Check

- Ensure correct behavior when calling:
    - `/v1/users`
    - `/v2/users`
- Verify backward compatibility and deprecation warnings

🔷 15. Caching Behavior

- Check for `ETag`, `Last-Modified`, `Cache-Control` headers
- Validate response when cached version is still valid (304 Not Modified)

🔷 16. Localization & Time Zone Handling

- Test APIs with `Accept-Language` header (e.g., `en-US`, `fr-FR`)
- Ensure dates/times respect user time zones

🔷 17. Pagination & Sorting

- Check page size, limits, and links (`next`, `prev`)
- Validate sorting logic (asc, desc) with valid/invalid fields

🛠️ How We Test APIs (Manual + Automation)

Activity	Manual (Postman)	Automation (Java/Python/ etc.)
Functional Testing	Collections, scripts, runner	RestAssured, Karate, PyTest
Schema Validation	Add schema in Postman, test with runner	JSON schema validators
Security/Auth Testing	Use tokens, cookies manually	Automated token generation
Load Testing	Postman Monitor (limited)	JMeter, Locust
Contract Testing	Swagger + Dredd	Pact, Postman + schema
Negative Testing	Use Pre-request script/random inputs	Fuzzing libraries

Let me know if you want this checklist in a **markdown**, **Excel**, or **Postman template collection**.

You said:

how we check data integrity

ChatGPT said:

✅ How We Check Data Integrity in API Testing

Data integrity testing ensures that data sent via APIs is **accurate**, **consistent**, and **unaltered** across systems — from source (request/input) to destination (DB/response/output).

Here's a practical guide to checking **data integrity** during API testing:

🔍 What Is Data Integrity?

It means that the **data received** or **stored** via an API is:

- Correct
- Complete
- Consistent
- Uncorrupted during transmission

✅ Key Checks for Data Integrity in API Testing

🔷 1. Request vs Response Consistency

- Validate that the data sent in `POST` or `PUT` request is **exactly reflected** in the response.
- **Example:**

```
json
CopyEdit
Request:
{ "name": "Anand", "role": "QA" }

Expected Response:
{ "name": "Anand", "role": "QA", "id": "123", "createdAt":
"..." }
```

🔷 2. API vs Database Validation (Source of Truth)

- After API call, query the backend **database** or **external system** to confirm:
  - The data stored matches the request
  - No additional/unexpected changes occurred
- 🔧 Tools: JDBC (Java), SQLAlchemy (Python), Postman DB verification plugins

Example in SQL:

```
sql
CopyEdit
SELECT * FROM users WHERE id = '123';
-- Verify name = 'Anand', role = 'QA'
```

🔷 3. Update Operation Validation

- For `PUT` or `PATCH`, check that only intended fields are updated.
- Confirm no old data is left behind or corrupted.

🔷 4. Data Format & Type Validation

- Validate formats: date (ISO 8601), currency, phone numbers, emails, etc.
- Validate types: string, number, boolean, object, array

🔷 5. Field Length and Boundaries

- Ensure field values do not exceed max allowed lengths.
- Truncation or overflow must not silently happen.
- Example: `username` max = 20 chars → test with 21 chars and expect rejection.

🔷 6. Encoding and Special Characters

- Check data with:
  - Unicode/emoji
  - Special characters (`<`, `&`, `"`, etc.)
  - Accents (é, ü)
- Ensure encoding (UTF-8, etc.) doesn't corrupt data.

🔷 7. Concurrency Testing

- Make simultaneous API updates and verify:
  - No partial overwrites
  - No race conditions
  - Data remains in a valid, consistent state

🔷 8. Data Duplication Prevention

- Ensure `POST` doesn't allow duplicate records unless intended.
- Use uniqueness constraints (e.g., email, user ID).

🔷 9. Integrity Across Systems

- Validate data passed between microservices or external APIs is not modified unintentionally.
- Use logs or message queues (Kafka/SQS) to verify transformation.

🧪 How We Test It

Method	Description
API + DB Validation	Send request → Validate DB update via SQL
Field-by-field Assertions	Assert each expected field in response
Schema Validation	Ensure structure/type conformity
Data Replay Testing	Resend same data and validate consistent outcome
Audit Log Verification	Check audit trail or logs for integrity
Hashing/Checksums (advanced)	Compare MD5/SHA values for payload integrity

🧰 Tools Used

Tool	Use Case
Postman + DB plugin	Manual API + DB integrity
RestAssured + JDBC	Java automation with DB checks
Karate	End-to-end BDD test
DBUnit / Flyway	Data state setup & verification
JMeter	Data verification under load
Faker / RandomLib	For generating test data (to validate integrity under variations)

✅ Example: Java (RestAssured + JDBC)

```
java
CopyEdit
@Test
public void testUserCreationIntegrity() throws SQLException
{
    String name = "Anand";
    String job = "QA";

    Response response = given().body("{ \"name\": \"" +
name + "\", \"job\": \"" + job + "\" }")
        .post("/api/users");

    String id = response.jsonPath().getString("id");

    Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/
db", "user", "pass");
    Statement stmt = con.createStatement();
```

```
    ResultSet rs = stmt.executeQuery("SELECT * FROM users
WHERE id = '" + id + "'");

    rs.next();
    Assert.assertEquals(rs.getString("name"), name);
    Assert.assertEquals(rs.getString("job"), job);
}
```

How to Check Atomicity in API Testing (ACID Principle)

Atomicity is the **"A" in ACID** (Atomicity, Consistency, Isolation, Durability) — a fundamental principle in transactional systems, especially databases and APIs that modify data.

💡 What is Atomicity?

Atomicity means all operations in a transaction must succeed, or none should happen.
If an API operation involves multiple steps, **it must not leave the system in a partial or inconsistent state** if any step fails.

🔍 When is Atomicity Relevant in API Testing?

- **Multi-step operations**: Create user + create profile + assign role
- **Transactional APIs**: Money transfer, order placement, batch uploads
- **Microservices-based APIs**: Where different services must update in sync

✅ What to Test for Atomicity

Scenario	Test Goal
Partial Failure	Simulate failure in step 2 → Ensure rollback
Exception Handling	System should not update partially on errors
Network Timeout / Retry	Ensure no partial data on timeout
Duplicate Submission (Race)	Verify idempotency, atomicity under concurrency

DB Rollback	DB changes should rollback if full API fails

🧪 How to Test Atomicity

🔷 1. Manual API Transaction Simulation

✅ Example:

You call an API that internally:

- Adds an order → deducts stock → sends confirmation

💥 Test Case:

- Simulate failure after step 1 (e.g., network failure or invalid stock).
- ✅ Validate:
  - Order **not created**
  - Stock **not deducted**
  - No confirmation sent

🔷 2. Automated API + DB Rollback Testing

Use a test like:

- Make a POST request to create something that depends on multiple inserts/updates.
- Break one part (e.g., invalid field in nested object).
- Query the DB afterward.

```sql
CopyEdit
-- Make sure no partial inserts happened
```

```
SELECT * FROM orders WHERE order_id = '123';   -- Should
return 0 rows
SELECT * FROM inventory_log WHERE order_id = '123'; --
Should return 0 rows
```

🔷 3. Failure Injection / Fault Testing

- Use tools like **WireMock**, **Mountebank**, or **ChaosMonkey** to simulate downstream failures.
- Check if API transaction is **aborted cleanly**.

🔷 4. Log and Audit Checks

- Look at backend logs or audit tables.
- Confirm that either **all steps logged** or **none**.

🔷 5. Idempotency + Atomicity Together

- Send same API twice (intentionally)
- Confirm that second attempt:
    - Doesn't duplicate data
    - Doesn't corrupt partial state

🔧 Tools for Atomicity Testing

Tool	Usage
Postman	Manual multi-step testing
RestAssured	API + DB validation
WireMock	Simulate failures in service chains
DB Query	Validate rollback or missing partial data

JMeter/ Locust	Concurrent users with partial failure check

✅ Example: Java Pseudo-Test for Atomicity

java
CopyEdit

```
@Test
public void testAtomicityOnOrderFailure() {
    String payload = "{ \"orderId\": \"123\", \"items\":
[ ... ], \"payment\": null }";

    Response response = given()
        .contentType("application/json")
        .body(payload)
        .post("/api/orders");

    assertEquals(response.getStatusCode(), 400); // Payment
is missing

    // Verify nothing is saved
    ResultSet rs = stmt.executeQuery("SELECT * FROM orders
WHERE order_id = '123'");
    assertFalse(rs.next()); // Assert no partial insert
}
```

✅ Summary: What to Ensure for Atomicity

Principle	Ensure
All or None	No partial data
Rollback	If failure occurs at any step
Validation	DB, logs, or downstream systems

Simulate Errors	Inject failures intentionally

Would you like a **sample atomic transaction flow** test using RestAssured or Python requests + DB validation?

2/2

You said:

what are the tools used to validate api security /vulnerbility

ChatGPT said:

Validating **API security and vulnerabilities** is critical to ensure your APIs are safe from attacks like unauthorized access, injection, data leaks, and more. Below is a list of **top tools used for API security testing**, grouped by purpose, along with what they help you validate:

✅ Top Tools for API Security & Vulnerability Testing

🔒 1. OWASP ZAP (Zed Attack Proxy)

- **Purpose**: Dynamic vulnerability scanning (DAST)
- **Use Case**: Detects XSS, SQL injection, broken auth, insecure headers
- **How to Use**:
  - Intercept API traffic via proxy
  - Scan API endpoint
- **Strength**: Free, scriptable, OWASP-backed
- 🔧 Can be automated in CI/CD

🧪 2. Postman + Security Tests

- **Purpose**: Functional and basic security validation
- **What to Test**:
  - Missing auth headers
  - Broken access control
  - Token expiration
- **Add-ons**:
  - JWT verification
  - JSON schema validation
  - Manual fuzz input

🧰 3. Burp Suite (Community/Pro)

- **Purpose**: Web/API vulnerability testing
- **Features**:
  - Intercept & replay requests
  - Scanner for SQLi, XSS, etc.
  - Active/passive scanning
- **Use for**:
  - Auth bypass, CORS misconfig, header poisoning
- **Pro Version** adds automation & deep scan

⚙️ 4. Nikto

- **Purpose**: Web server security scanning
- **Checks**: Insecure server configurations, default files, outdated servers
- Works well for REST APIs hosted on HTTP servers

🧬 5. Nmap + NSE Scripts

- **Purpose**: Network scanning + vulnerability detection
- **Use for**:
  - Discover open ports where APIs are exposed
  - Use NSE scripts to test known API vulns

🧰 6. Kali Linux Tools (e.g., sqlmap, dirb, wfuzz)

- **sqlmap**: Auto-detect SQL injection on API endpoints
- **wfuzz**: Fuzzing API inputs
- **dirb/gobuster**: Discover hidden or undocumented API endpoints

🛡️ 7. API-Specific Tools

Tool	Description
CrAPI (OWASP)	A deliberately vulnerable API (for practice)
Tinfoil Security	API vulnerability scanner with CI integration
42Crunch	Advanced OpenAPI security linting + runtime scanning
APIsec	Automated security test generation from OpenAPI spec
Probely	Cloud-based security scanner (DAST + API tests)

🔍 8. Static & Contract Tools

- **Spectral**: Lint OpenAPI/Swagger files for security violations
- **Swagger/OpenAPI validators**: Ensure correct use of auth headers, roles, scopes

- **SonarQube (for API source code)**: Scan backend code for insecure API implementations

✅ Security Areas You Can Validate with These Tools

Area	Example Attacks	Tool Examples
Authentication	Missing tokens, brute force	Postman, ZAP, Burp
Authorization	Broken role access, IDOR	Burp, Postman, custom scripts
Input Validation	SQLi, XSS, NoSQL injection	sqlmap, ZAP, Burp
Rate Limiting & DoS	Request flooding, replay attacks	Locust, JMeter, Burp
Information Exposure	Stack traces, sensitive fields	ZAP, Burp, Nikto
TLS/SSL	Expired, weak ciphers	ssllabs.com, Nmap
CORS Misconfiguration	Unrestricted domains	ZAP, Burp, CORS test plugins
Security Headers	Missing `X-Content-Type-Options` etc.	ZAP, curl -I, Burp

✅ Bonus: CI/CD Integration

- **OWASP ZAP**: Integrate using Docker + scripts in Jenkins/GitLab
- **Postman**: Add security test cases to Postman collections in CI
- **APIsec / 42Crunch**: Integrate OpenAPI spec security testing in PR pipelines

Option 1: Automate API Security Testing with OWASP ZAP (Headless in CI)

🔧 Prerequisites

- Install **ZAP Docker image** (ideal for CI pipelines)
- You have a **Postman collection** or can trigger APIs via a known base URL

🧰 Example: Using ZAP Docker + Command Line

🔷 Step 1: Run ZAP as a daemon in Docker

```
bash
CopyEdit
docker run -u zap -p 8090:8090 -d \
    -v $(pwd):/zap/wrk/:rw \
    owasp/zap2docker-stable zap.sh -daemon \
    -port 8090 -host 0.0.0.0 -config api.disablekey=true
```

🔷 Step 2: Spider your API (start crawling it)

```
bash
CopyEdit
curl "http://localhost:8090/JSON/spider/action/scan/?
url=https://your-api-base-url.com"
```

🔷 Step 3: Active scan

```
bash
CopyEdit
curl "http://localhost:8090/JSON/ascan/action/scan/?
url=https://your-api-base-url.com"
```

🔷 Step 4: Download the security report

```
bash
CopyEdit
curl "http://localhost:8090/OTHER/core/other/htmlreport/"
-o zap_report.html
```

✅ **Result**: A full vulnerability report with:

- XSS
- SQLi
- Missing headers
- CORS issues
- Insecure cookies
- Broken auth

🚀 ZAP in CI Example (GitLab/YAML)

```
yaml
CopyEdit
zap_scan:
  image: owasp/zap2docker-stable
  script:
    - zap.sh -daemon -host 0.0.0.0 -port 8090 -config
api.disablekey=true
    - curl "http://localhost:8090/JSON/spider/action/scan/?
url=https://your-api.com"
    - curl "http://localhost:8090/JSON/ascan/action/scan/?
url=https://your-api.com"
    - curl "http://localhost:8090/OTHER/core/other/
htmlreport/" -o zap_report.html
  artifacts:
    paths:
      - zap_report.html
```

✅ Option 2: Add API Security Checks in Postman Collection

🔷 Step 1: Add Negative Auth Test (in Pre-Request Script or separate request)

```
javascript
CopyEdit
pm.sendRequest({
  url: pm.environment.get("api_base") + "/secure/endpoint",
  method: "GET",
  header: {
    // Intentionally leave out or corrupt token
    "Authorization": "Bearer invalid_token"
  }
}, function (err, res) {
  pm.test("Unauthorized access blocked", function () {
    pm.expect(res.code).to.eql(401);
  });
});
```

🔷 Step 2: Add Schema + Security Header Checks

```
javascript
CopyEdit
pm.test("Has secure headers", function () {
  pm.expect(pm.response.headers.get("Strict-Transport-
Security")).to.include("max-age");
  pm.expect(pm.response.headers.get("Content-Security-
Policy")).to.not.be.undefined;
  pm.expect(pm.response.headers.get("X-Content-Type-
Options")).to.eql("nosniff");
});
```

🔷 Step 3: Run in CI using Newman

```
bash
CopyEdit
newman run my_collection.json -e environment.json -r
html,cli,junit
```

✅ Add to your **GitHub Actions / GitLab CI / Jenkins pipeline** to trigger on every commit.

🧪 Bonus: Use Test Data to Simulate Attacks

Add test cases that:

- Submit `"<script>alert(1)</script>"` in input fields (XSS)
- Inject SQL-like payloads (`' OR 1=1--`)
- Send broken or expired JWT tokens
- Test file upload APIs with `.php`, `.exe`, `.bat`