**🔷 Module 1: Introduction to API Testing**

**✅ What is an API?**

**API (Application Programming Interface)** is a set of rules that allows software applications to communicate with each other. It acts as a bridge between different systems.

📦 Example:  
A weather app fetches real-time data using an API like:

GET https://api.weather.com/v3/weather/conditions?city=Mumbai

**✅ What is API Testing?**

**API Testing** involves testing APIs directly to ensure:

* They return the correct data.
* Handle edge cases gracefully.
* Follow performance, security, and business rules.

API Testing is done **before** UI testing, often in integration or backend layers.

**✅ Why Do We Need API Testing?**

* Backend logic can be validated early.
* Helps detect bugs before UI is built.
* Saves time and cost in the long run.
* Enables **Continuous Integration** with automated test scripts.

**✅ Benefits of API Testing**

| **Benefit** | **Description** |
| --- | --- |
| 🧪 **Early bug detection** | Test logic before the UI is ready |
| ⚡ **Faster execution** | Much faster than UI tests |
| 🔁 **Reusable test scenarios** | APIs are stable, reusable |
| 📦 **Wide coverage** | One API serves multiple UIs |
| 🔐 **Security validation** | Tokens, encryption, headers, etc. |
| 🔗 **Integration testing** | Confirms data flows across services |

**✅ API Testing Types**

| **Type** | **Description** |
| --- | --- |
| 🧪 **Functional Testing** | Validating API response as per business logic |
| 🔐 **Security Testing** | Auth, access control, tokens, HTTPS |
| 📊 **Load/Performance** | Stress test under high traffic |
| 💥 **Negative Testing** | Handle invalid inputs gracefully |
| 🔁 **Integration Testing** | Validate systems working together |

**✅ API Testing Best Practices**

* Use realistic data and scenarios.
* Validate response codes, body, headers.
* Use environment variables (e.g., Postman).
* Cover positive + negative cases.
* Use chaining (use one API’s output in the next).
* Automate stable APIs with tools like **REST Assured**, **Postman**, **Karate**.
* Include API tests in CI/CD (Jenkins, GitLab).

**✅ Types of Bugs API Testing Detects**

* Missing or incorrect data
* Incorrect HTTP status codes (e.g., 200 vs 500)
* Broken JSON structure
* Unauthorized access (security flaws)
* Performance bottlenecks
* Data leaks or loss between services

**✅ Challenges in API Testing**

| **Challenge** | **Description** |
| --- | --- |
| 🔐 **Authentication complexity** | OAuth2, JWT, API Keys |
| ⚠️ **Changing schema** | Frequent changes in request/response fields |
| 🔁 **Test data dependencies** | APIs depend on database or another API |
| 🧪 **Asynchronous behavior** | Some APIs respond later or via webhooks |
| 🔄 **Environment differences** | Dev, QA, Prod might behave differently |

**✅ API Testing Tools Selection Criteria**

* Supports required protocols (REST, SOAP, GraphQL)
* Can handle authentication types (Bearer, Basic, OAuth)
* Allows automation/integration (Jenkins, CI/CD)
* Easy learning curve
* Support for environment and variables
* Detailed reports/logging

**✅ Popular API Testing Tools**

| **Tool** | **Use Case** |
| --- | --- |
| 🔹 **Postman** | Manual + basic automation |
| 🔹 **REST Assured** | Java-based automation |
| 🔹 **Karate DSL** | BDD style, simple syntax |
| 🔹 **SoapUI** | SOAP + REST |
| 🔹 **Swagger** | API design + testing |
| 🔹 **JMeter** | Load and performance testing |

**🔷 Module 2: Postman Basics (Manual API Testing)**

**✅ Introduction to Postman**

**Postman** is a popular GUI-based API client that helps developers/testers send requests to APIs and validate responses without writing code.

**✅ Why Use Postman?**

* GUI-based – no code needed
* Easy request creation
* Supports GET, POST, PUT, DELETE, PATCH, etc.
* Environment & variable support
* Pre-request scripts and tests
* Automated collection runner

**✅ Download and Install Postman**

1. Visit <https://www.postman.com/downloads>
2. Install on Windows/macOS/Linux
3. Sign in or continue without login

**✅ How to Use Postman**

1. Launch Postman
2. Create a **New Request** (GET, POST, etc.)
3. Enter URL and parameters
4. Send request
5. Validate **Response Code**, **Headers**, **Body**

**✅ Create New Request in Postman**

* Click + New Tab
* Choose method: GET / POST / PUT / DELETE
* Enter URL
* Add headers/body if needed
* Click **Send**

**✅ GET Request Example**

GET https://reqres.in/api/users?page=2

Postman returns:

* Status Code: 200 OK
* JSON Body with user list

**✅ Dealing with Response**

Look at:

* Status Code: 200, 404, 500
* Headers: Content-Type, Auth
* Body: JSON/XML
* Use **Test Tab** for JavaScript-based checks

pm.test("Status is 200", () => {

pm.response.to.have.status(200);

});

**✅ POST Request in Postman**

* Choose POST
* Go to **Body → raw → JSON**

{

"name": "John",

"job": "Engineer"

}

* Add header: Content-Type: application/json
* Send and validate status = 201 Created

**🔷 Module 3: REST Assured (Java-based API Automation)**

**✅ What is REST Assured?**

**REST Assured** is a Java library for automating REST API tests in a BDD-like format.

It integrates well with:

* Maven
* TestNG
* JUnit
* Allure for reporting

**✅ What REST Assured Provides**

* Easy syntax for requests
* JSON/XML parsing
* Path & query parameters
* Assertions on status, headers, body
* Logging

**✅ REST Assured Setup**

Add to pom.xml:

<dependency>

<groupId>io.rest-assured</groupId>

<artifactId>rest-assured</artifactId>

<version>5.4.0</version>

<scope>test</scope>

</dependency>

**✅ First REST Assured Test**

import io.restassured.RestAssured;

import org.testng.annotations.Test;

import static io.restassured.RestAssured.\*;

import static org.hamcrest.Matchers.\*;

public class GetTest {

@Test

public void getUser() {

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

given().

when().

get("/users/2").

then().

statusCode(200).

body("data.id", equalTo(2));

}

}

**✅ POST Request Example**

@Test

public void createUser() {

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

given().

header("Content-Type", "application/json").

body("{ \"name\": \"John\", \"job\": \"Tester\" }").

when().

post("/users").

then().

statusCode(201).

body("name", equalTo("John"));

}

**✅ Validations**

**✅ 1. Setup (BaseTest.java)**

package tests;

import io.restassured.RestAssured;

import org.testng.annotations.BeforeClass;

public class BaseTest {

@BeforeClass

public void setup() {

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

}

}

**✅ 2. Full Validation Test (GetValidationTest.java)**

package tests;

import io.restassured.response.Response;

import org.testng.Assert;

import org.testng.annotations.Test;

import java.util.List;

import static io.restassured.RestAssured.\*;

import static org.hamcrest.Matchers.\*;

public class GetValidationTest extends BaseTest {

@Test

public void validateUserList() {

Response response =

given().

header("Accept", "application/json").

when().

get("/users?page=2").

then().

log().all().

assertThat().

statusCode(200). // ✅ Status Code Validation

statusLine("HTTP/1.1 200 OK"). // ✅ Status Line

header("Content-Type", containsString("json")). // ✅ Header Validation

body("page", equalTo(2)). // ✅ JSON Field Validation

body("data.size()", equalTo(6)). // ✅ Array size

body("data[0].email", containsString("@reqres.in")).

extract().response();

// ✅ Extracting & Validating Specific Value

String firstEmail = response.jsonPath().getString("data[0].email");

Assert.assertTrue(firstEmail.endsWith("@reqres.in"), "Email format invalid");

// ✅ List validation example

List<String> allEmails = response.jsonPath().getList("data.email");

for (String email : allEmails) {

Assert.assertTrue(email.contains("@"), "Invalid email: " + email);

}

}

}

**✅ 3. Full Validation Test (PostValidationTest.java)**

package tests;

import io.restassured.response.Response;

import org.testng.Assert;

import org.testng.annotations.Test;

import static io.restassured.RestAssured.\*;

import static org.hamcrest.Matchers.\*;

public class PostValidationTest extends BaseTest {

@Test

public void validatePostResponse() {

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

Response response =

given().

header("Content-Type", "application/json").

body(requestBody).

when().

post("/users").

then().

log().all().

statusCode(201). // ✅ Status Code for POST

body("name", equalTo("Uday")). // ✅ Body field validation

body("job", equalTo("Tester")).

body("id", notNullValue()). // ✅ ID is generated

body("createdAt", notNullValue()). // ✅ Timestamp exists

extract().response();

// ✅ Extract individual field for assertion

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

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

Assert.assertEquals(name, "Uday");

Assert.assertNotNull(id, "ID should not be null");

}

}