Lexicons for API Tester   
  
  
We all know that [API Testing](https://muuktest.com/blog/what-is-api-testing/) is getting more attention nowadays. Almost every technical job description requires API testing experience.

Despite this trend, not everyone gets the opportunity to work with APIs. Few organizations are focused on front-end automation. But as a Tester, [we should be versatile](https://nordicapis.com/), right? There are a lot of materials available online to get started with. But, of course, finding the right one is always challenging.

How do we start this journey? How can we help you?

We created a API testing glossary that defines the technical jargon used in API Testing. This covers some basic API terms as well.

Keep in mind the basics are a must. They help you to build stronger APIs. So wait? Let’s get started.

A Complete List of Terms to Start in API Testing

**API:** Application Programming Interface, a set of functions that allows an application to interact with external applications, operating systems, microservices, or data.

**API-First:**A development approach that considers application programming interfaces (APIs) before anything else. It ensures that all of the platform/application functionalities are accessible through the APIs.

**API Testing:**Checking whether the API meets expectations in terms of functionality, reliability, performance, and security.

**API Orchestration:**The unification/merging of different APIs into a single front-end. It is now a central component of creating and designing enterprise workflows. API orchestration layer is an abstraction layer that collects data from one or more services and prepares them in favor of client applications.

**API Version:**Similar to document versioning, API versioning tracks changes in the API. Proper versioning supports maintenance and tracking. A new API version is created when major changes to the endpoints/payload are made.

**API Gateway:**A single entry point for all clients. It helps balance the load to the endpoints received from different clients. It also helps in handling the API traffic.

**API Security:**Focused strategies that protect APIs from attacks. Any security breach will lead to data leakage and become a serious threat.

**API Design:**Developing the APIs as per the business process and requirements. This serves as a single source of truth for the API. Good API design helps to minimize the problems. The Design-First approach helps in developing stable APIs that meet expectations.

**API Deprecation:**In some situations, the API is deprecated (no longer used). Various factors contribute to the deprecation of the APIs: project requirement changes, insecure APIs, inefficiency, or replacement, to name a few.

**API Keys:**The unique code used for authentication of the API. One of the authorization methods to access the API. Some APIs use the keys in the params, and some use them in the headers.

**API Request:**API request is the set of details needed to submit to the server to understand and respond properly.

**API Response:**After sending the details as a request, the server interprets and sends the response back to the client.

**API Virtualization:** API Virtualization is the process of mirroring your production APIs that promotes testing efficiency. It helps to test the team with realistic test data.

**API Documentation:**The complete set of technical information and capabilities of the API. It carries all the sufficient information to work with that API.

**BaseURI:**Base Uniform Resource Identifier is the complete path used to locate a specific resource (Base URL, which is the application URL + path)

**Contract:**An agreement details what the consumer can expect from the API. It details how the API works. It is the same as API documentation.

**Content API:**These APIs provide or transfer the contents. Not just a mere communication mode, it also helps to retrieve or collect important content.

**CORS:**  Cross-Origin Resource Sharing (CORS) is an HTTP-header-based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources. Additional HTTP headers are used to instruct the browsers.

**cURL (Client URL):** a cURL is a command-line tool for transferring data from or to a server designed to work without user interaction. cURL helps get information from APIs, download web pages, or submit data to an API.

**Composite APIs:** Composite APIs are the combined APIs that help us access several endpoints through a single API call.

**GraphQL:**GraphQL is an application-level query language and runtime for APIs. It is a more efficient, flexible, and powerful way of working with APIs when compared to REST.

**gRPC (Google Remote Procedure Call):**A modern open-source RPC architecture framework designed by Google can run in any environment. It is robust, scalable, and fast.

**HTTP methods:**An HTTP (HyperText Transfer Protocol) request is an action to be performed on a resource identified by a given Request-URL. Request methods are case-sensitive. The most common ones are: GET, POST, PUT, and DELETE.

**HTTP Headers:**Headers are the additional but essential information sent as part of the request details from the client for the server to understand the client.

**JSON (JavaScript Object Notation):** JSON is an open-standard file format or data interchange format in human-readable style. It is language-independent and represents the data using a key/value pair.

**Karate DSL:**A framework for API automation testing, running on Java and using Apache HTTP client to perform HTTP connections. It can be integrated with the Gatling framework for performance testing.

**Microservices:** A modular software component that does one defined job. It is an architectural style that structures an application as a collection of small autonomous services modeled around a business domain. They are small, independent, and loosely coupled.

**Mock Servers:** In some situations, the mock server behaves like a real server, but it responds with the same set of responses as mocked. It emulates the real server.

**MQTT (Message Queuing Telemetry Transport):** MQTT is the most commonly used messaging protocol for the Internet of Things (IoT). MQTT allows for messaging between devices to the cloud and the cloud to the device.

**OpenAPI:**OpenAPI is a specification that describes, produces, consumes, and visualizes RESTful APIs and web services.

**OAuth:** OAuth is a delegated authorization framework used for REST/APIs. This is one of the efficient methods of authorizing, which restricts the improper usage of the APIs. For example, a user can sign in on one platform and then be authorized to perform actions and view data on another platform with OAuth.

**OWASP (Open Web Application Security Project):**A non-profit organization dedicated to improving web application security. All of its resources are free of charge.

**Pagination of APIs:**The pagination mechanism is used when the API returns a large data set. It is similar to how the Google search response is handled and split into pages. There are different types of Pagination:

* Offset Pagination: is the simplest method; “limit” and “offset” values are used here.
* Keyset Pagination: uses the filter values of the previous page to determine the next set of items.
* Seek Pagination: this is the enhanced version of keyset pagination. It helps filter the particular limit.

**Parameters:** API Parameters are passed along with the endpoint URL, which helps filter resources. Path/Query params are the most frequently used ones.

**Public APIs:** An open or public API saves developers time by allowing them to connect their platform with previously existing tools, reducing the need to create entirely new functions. Most public APIs requires no authorization.

**Private APIs:** Internal APIs are the opposite of open APIs in that they are inaccessible to external consumers and only available to an organization’s internal developers. Internal APIs can enable enterprise-wide initiatives from adopting DevOps and microservice architectures to legacy modernization and digital transformation.

**Payload:** The actual body/content passed as part of the API request. It can be in different forms such as JSON, Text, HTML, XML, etc.

**Postman:** Postman is a platform to build, test, design, modify, and document APIs. It is a simple Graphic User Interface for sending and viewing HTTP requests and responses.

**RAML (RESTful API Modeling Language):** RAML is a powerful YAML-based language used to define API contracts. RAML allows teams to define, build and collaborate on APIs rapidly and easily.

**RapidQL:** RapidQL is a JSON-like query language implemented in JavaScript. This essentially helps describe the various inputs used in your queries.

**REST (REpresentational State Transfer):** REST is a software architectural style that defines a set of methods to build a web Application Programming Interface (API). REST is one of the most popular types of API due to its simplicity and client-friendly nature**.** Requests are sent via several formats: JSON, HTML, XML, plain text. JSON is the most commonly used format.

**Rest Assured:**REST Assured is a Java library used for testing and validating the REST APIs.

**Schema:** It defines the data format, including the data types. This schema validates the API requests.

**SOAP (Simple Object Access Protocol):** SOAP is an XML-based protocol for accessing web services over HTTP. The SOAP message is nothing but an XML document with the envelope, header, and body.

**Status Codes:**The list of numeric codes followed by a message returned from the server to the client. Each response code tells a different story about the API response.

**SSL Certificates:** SSL certificates create a foundation of trust by establishing a secure connection. It is a small data file that cryptographically establishes an encrypted link between a web server and a browser. It protects client-server communication from vulnerable attacks.

**Swagger:**Swagger is an open-source set of rules, specifications, and tools for developing and describing RESTful APIs. Also, there is an open-source tool called Swagger, which is used to design, build, document, and use RESTful web services.

**YAML (Yet Another Markup Language):**YAML is a data serialization format similar to XML and JSON. YAML is a case-sensitive, human-readable data format.

**Webhook APIs:**A webhook is a lightweight API that powers one-way data sharing triggered by events. Webhooks are also called reverse APIs. They help send messages, alerts, notifications from the server to the client.

**Web Scraping:** Web scraping or web data extraction is data scraping used for extracting data from websites. The wanted content is segregated and separated from the list of long content.

**WebSockets:** A WebSocket is a transport protocol defined by a persistent bi-directional communication channel between a client and the server. It was designed to overcome the limitations of HTTP’s basic request/response mechanism.

Ufff, the list is huge, right? Still, they will be helpful to you. Once you understand this technical jargon, you can connect the dots during your API testing journey.

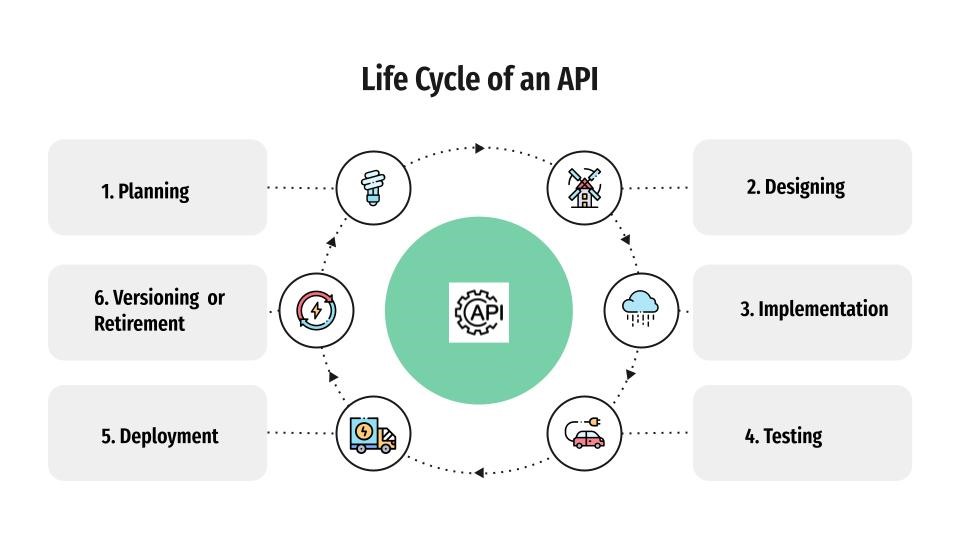
# **Lifecycle of an API**

Due to the evolution of the API first companies like Stripe, Shopify, or Twilio, there’s a lot of attention to APIs in recent times. Thus implementing/adopting those APIs requires a disciplined skill set. It’s not just a single-day process; it’s a commitment towards it.

Everyday “n” number of APIs are created around the globe, and getting to know more about the API lifecycle and relevant processes helps to design better APIs. All the API stakeholders should be aware of the entire lifecycle process. API lifecycle management considers the end-to-end process from planning/creating the APIs to the maintenance/retirement. The API manager is a part of all phases and manages the lifecycle.

API lifecycle phases depend on your [API Architecture](https://www.akana.com/blog/api-architecture), but the basic flow remains the same. So let’s get to know more about it and try to understand where API testing fits in.

Based on the traditional approach, below are the 6 phases of the API. Let’s see each phase in detail:



### **Planning:**

This is the first and basic step for API development. This is similar to the requirement gathering phase in SDLC. Multiple stakeholders discuss here and conclude the few important factors related to the API.

* What are the API capabilities?
* What type of authentication is to be used?
* What schema type to be used
* What is the schema format (JSON or YAML)?
* Which language or tool will be used to develop the API?
* What are the dependencies? What are the timelines?
* Also, to discuss if any certificates are involved in this.

Answers to these questions will help the developer proceed with the design.

### **Designing:**

During the design phase, it’s good to have complete knowledge of the purpose of the API. A draft API is designed, and that helps to verify with stakeholders before developing the actual API. This phase defines the sample endpoints and the sample responses. Mostly it will be the mocked sample responses.

### **Implementation:**

This is the actual development phase of the API. First, developers will add the endpoints and the operation methods to the schema. Starting from the version, URL, path, prams, responses, response codes, headers, authentication type, and error codes are configured in the API builder or some specific IDE based on the language used. Once the complete endpoints are ready, the developer can generate the API documentation or manually give all the endpoints to the Testing team.

### **Testing:**

Unit testing by developers is performed in this phase. Official testing is started when the developer provides the endpoints of the API. Now the API testers can go ahead and start testing the API endpoints by adding the assertions and verifying the behavior. Extensive testing is performed to detect the bugs early. Under API testing, different validations are performed to verify the performance, security, functionality, and load.

### **Deployment:**

The CI/CD pipeline is created here, and the APIs are integrated with them. This ensures seamless automation in the production environment. If there’s any API gateway integration, it is done in this phase. The complete end-to-end flow of the API is taken care of. If we have any dependency on third-party APIs, that is also carefully implemented here.

### **Versioning/Retirement:**

Once the API is deployed into production, it can go through two states based on the needs.

**Versioning:**If the API stays in place and the enhancements are done based on the application changes, the API is versioned to track the changes. Proper versioning is a good practice, and this helps in debugging.

Most of the APIs are published after this, and some can be monetized. Exposing the APIs to the market can bring you revenue.

**Retirement:** In some cases, the API becomes obsolete due to some application migration or retirement. So, the API is retired. Also, in some use cases, the API as such would have been developed for some temporary causes, so once the job is done, the API is retired or deprecated.

## Why is API Testing important?

* [API Testing](https://muuktest.com/blog/what-is-api-testing/) is different from UI Testing. It’s more focused on the core functionalities.
* Currently, most companies are drifting towards the API-First approach, which increases the importance of API testing.
* API Testing can be performed even when the front-end is not yet ready.
* API Testing helps in early bug detection (shift-left defects).
* API Testing helps us to reduce the cost, effort, and time.
* API Testing helps us foresee and reduce the risks of the systems/applications.
* API testing helps in integration testing with a few other applications.
* API test automation requires less coding, and it provides faster and better test coverage.
* For some third-party testing, we might not have access to the application, and in this case, the API testing approach can ensure the quality of the product.
* API Testing is language-independent.
* API Test automation gives the highest ROI.
* Deeper API testing will cover most defects, so UI test load is reduced.
* API Tests require less maintenance.

## How is API Testing different from GUI testing?

[GUI Testing](https://www.inflectra.com/Rapise/Highlights/GUI-Testing.aspx) is related to the front end or the user interface part of the application. A major focus is given to the functionalities, user interfaces, look and feel of the application. In API Testing, we test the core functionalities (Create user, Update user, and Delete user).

In API testing, we cannot feel the application’s front end. Also, the approach is a bit different when compared to front-end application testing. However, all the Testing artifacts are similar, with slight modifications. Here, the requirements will be the API endpoints and their sample responses. This helps us to proceed further with the Test scenarios and Test cases.

## Conclusion:

You might wonder why suddenly there’s so much importance to API Testing. APIs have been in place since the late 1990s. Mostly it began with web services. Salesforce released its first API in 2000. And then, slowly, Amazon, Flickr, Facebook, and Twitter started developing their APIs and publishing them.

The APIs have been used heavily for connecting multiple applications, reusability, third-party integrations, and many more in the last ten years.

Right now, we know the lifecycle of an API and where the API testing fits in. We hope this will give you an aerial view of your APIs, and let’s meet you all soon with another set of interesting content related to API.

# **Types of API Testing**

I hope you enjoyed the previous articles related to APIs and understood [what an API is](https://muuktest.com/blog/what-is-api-testing/), the [life cycle of an API](https://muuktest.com/blog/api-lifecycle/), and where API testing fits in the development process. This section will focus on the detailed validations performed for API testing.

When performing [GUI testing](https://www.guru99.com/gui-testing.html), we are aware of the scenarios we’re testing. However, the test scenarios and test cases for APIs can vary. Also, API Testing ensures the validation of functional and non-functional requirements of the system. Today, we will review the types of API testing and how the validation performed on APIs ensures scalability, reliability, and efficiency.



These are the different types of API tests:

1. [Validation Testing](https://muuktest.com/blog/types-of-api-testing/#Validation-Testing)
2. [Functional Testing](https://muuktest.com/blog/types-of-api-testing/#Functional-Testing)
3. [Integration Testing](https://muuktest.com/blog/types-of-api-testing/#Integration-Testing)
4. [Security Testing](https://muuktest.com/blog/types-of-api-testing/#Security-Testing)
5. [Performance Testing](https://muuktest.com/blog/types-of-api-testing/#Performance-Testing)
6. [Reliability Testing](https://muuktest.com/blog/types-of-api-testing/#Reliability-Testing)
7. [API Documentation Testing](https://muuktest.com/blog/types-of-api-testing/#API-Documentation-Testing)
8. [Regression Testing](https://muuktest.com/blog/types-of-api-testing/#Regression-Testing)

## Validation Testing

Validation testing is performed following the completion of API development. This ensures that the API is fulfilling its purpose. Validation testing starts with the schema validation, checking if the correct product is built and if the developed API behaves as expected and produces desired results.

At this point, the core capabilities of the API are validated. We can proceed to the next level of API tests when the validations pass.

## Functional Testing

These tests ensure that the API is performing the intended job. This splits the behavior of the API into small chunks and verifies if the system is working as expected. Generally, all the happy path testing is covered as part of the [functional tests](https://muuktest.com/blog/qa-functional-testing/). However, the central focus is on the correct behavior. Once we ensure that the APIs are working as expected with the right test data set, we will start with the negative test cases: providing the incorrect parameters, request types, out-of-bound values, etc. Functional tests are categorized as:

#### **Contract Tests:**

A contract is a definition that explains the API functionality. There are two significant actors involved: a provider and a consumer, meaning the contract testing could be provider-driven or consumer-driven. These tests ensure that the contract is appropriately defined.

#### **Component Tests:**

This is the detailed level of validation for each HTTP request. First, every request is tested with both positive and negative test data. Then, response status, code, message, and response time are evaluated. Finally, we add assertions based on the expected behavior.

#### **Scenarios Tests:**

Similar to UI tests, we ensure the API’s behavior in specific scenarios. Scenario testing combines a few requests, creates appropriate chaining, and validates the inter-request communication and the flow. For instance, when the user authentication is successful/unsuccessful, how should the API behave with other requests?

## Integration Testing

Multiple API calls are involved in the end-to-end flow of a single application. Therefore, we need to check if the intra-API communication and data exchange is working as expected during the integration testing. Furthermore, we can ensure that all the related APIs are connected and communicating with each other by performing these tests.

## Security Testing

Security runs are critical to the entire API test process since the vulnerability to malicious attacks will lead to data theft. Similarly, once the API is developed, we should test the user authorization, accessible resources to the intended audience, and data encryption (wherever possible). Make sure to inform the stakeholders before performing the security tests. Also, certain approvals are needed to proceed with these tests. Two significant types of Security tests performed for APIs are:

#### **Penetration Tests:**

Penetration testing or pen testing simulates an attack to monitor the API response. This is an authorized attack on the APIs and a common security exercise.

#### **Fuzz Tests:**

Fuzz testing helps discover security vulnerabilities or bugs in the API by injecting invalid or unexpected inputs to the API. This creates noise in the application and tries to crash the API.

## Performance Testing

Performance tests will evaluate the API performance under a specific set of instructions. We need to run different tests or assertions to assess the APIs’ performance. For instance, if there was a big sale on your site, you might expect a high volume to your APIs, which should be capable of handling the traffic.

Major categories of performance testing are:

#### **Load Testing:**

Loads tests evaluate the capacity of the API and how many calls this API could handle.

#### **Stress Testing:**

After getting a certain level of API calls, the system will reach the stress point. From there, we can see if the API is able to respond normally to high loads.

#### **Spike Testing:**

These tests will induce a sudden increase in API calls. In production, this might happen due to some expected and unexpected situations. Imitating this scenario can ensure that the API is scalable.

#### **Soak Testing:**

Soaks tests are nothing but extended load tests, where the higher requests are sent to the API for a well-defined period of time to evaluate the behavior.

Discuss with the developer if the API could handle the load or stress before doing your performance tests.

## Reliability Testing

How reliable are the APIs? These tests will ensure that the API is consistent and working the way it’s designed. In addition, they exist to verify if the test results are the same even after the integration with multiple systems/devices.

## API Documentation Testing

API documentation is similar to the functional specification in the UI testing. Any change in the requirements should be updated in the API documentation since this is a single source of truth. Any deviation of the API behavior from the documentation should be reported as bugs, and relevant documents should be updated.

## Regression Testing

This is similar to the UI [regression testing](https://muuktest.com/blog/what-is-regression-testing/). API bug fixes or negative testing shouldn’t affect the existing behavior of the API. So, try to have minimal regression cases to ensure the impeccable conduct of the APIs.

## Final thoughts

There are different aspects of APIs to be tested. The API test should be specific to the needs of the client. Still, most of the API tests mentioned above are meant to deliver high-quality APIs through exhaustive testing. Make sure to prepare a well-documented API testing strategy to denote the API testing types to be covered during the testing process.

# **Common Challenges in API Testing and How to Handle Them**

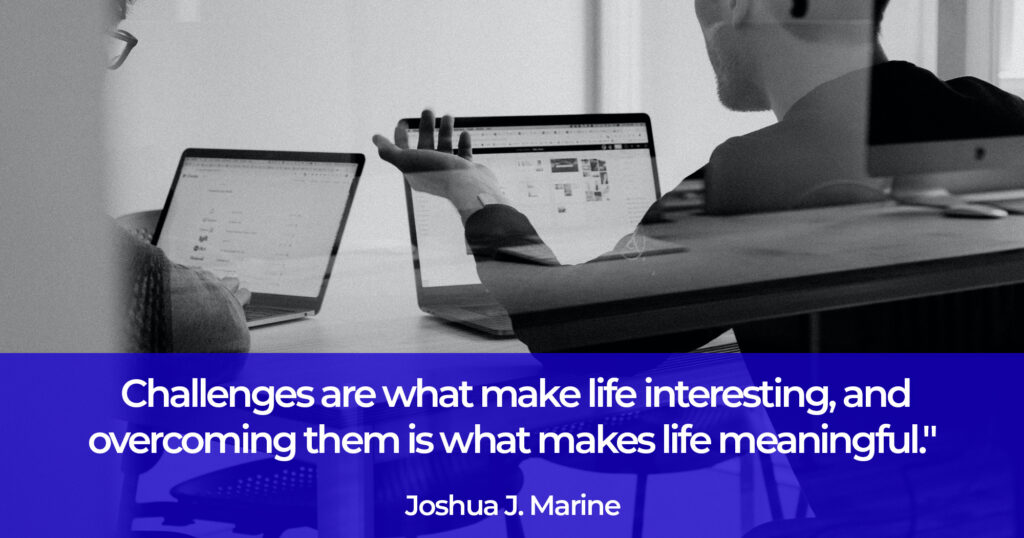
Welcome back to another interesting blog about API testing. You might need to start testing your APIs at some point, and there are some considerations before getting started.



So, the time spent on the initial analysis and planning is worth it for every Tester to have a smooth ride during your API Testing process. Here are a few questions to ask in your analysis to improve your testers’ experience:

1. What is this API for?
2. Is this going to be internal?
3. Who should be ideally using this API?
4. How are we going to test this API?
5. What are the environments available?
6. Do we have a dedicated testing environment for these APIs?
7. What are the behaviors of the individual endpoints?
8. What are the types of testing to be performed?
9. What are the timelines?
10. Does it have any other APIs integrated?

Once we have answers to these questions, we will have a good test strategy in hand. But that’s not all; you should learn about a few common API testing challenges and how to overcome them. Only then are we able to begin the testing process.



Let’s have a look below at a few common challenges to API testing:

### **Initial Project Setup**

Once the API environments and the necessary infrastructure is ready, you should begin your testing process. If the testing is to be automated, choosing the right framework is also part of that process. Then, based on the timelines, the test cases are drafted, and the resourcing is done accordingly.

**Pro tip:** You can better plan the resourcing and estimation in the earlier phases. You can allot the tasks based on the team’s current API testing experience. To plan for automation, first, devise a plan on the feasibility and the percentage of automation.

### **Lack of Proper/Complete API Documentation**

API Documentation is a single source of truth about the API capabilities. So, understanding all the API endpoints, the payload formats/samples, headers, authentication, or any other additional information will be a big challenge. Unfortunately, in most projects, there is no proper documentation.

**Pro tip:** Discuss with the stakeholders ahead of time, and ask them to share or work on the proper API documentation. Try to gain knowledge on the list of available API documentation tools so you can suggest to them some options instead of maintaining the API endpoints in a Google doc.

### **Lack of JSON Knowledge**

As you may know, most of the REST APIs use JSON format for the request body and their response. As a tester, we should have a decent level of knowledge of the JSON format and its advantages. Also, it’s good to have basic XML and HTML formats for handling different APIs. With this knowledge, we can avoid basic errors that occur while tweaking the request payloads. It also helps to parse the response easily.

**Pro tip:** Before using any payload, it’s advisable to go for any [online validator](https://jsonformatter.curiousconcept.com/) and check the format for any validation issues.

### **API Sequencing**

While testing for your projects, you might need to handle many API endpoints performing different GET, POST, PUT, and DELETE options. So, the Tester should have strong knowledge of the API flows and how the system communicates internally. Based on this understanding, one API call is chained with another.

**Pro tip:** Before building your API calls, list your questions and get clarified about the overall flow with the intended stakeholders to avoid confusion. Also, the usage of variables helps for this chaining of requests.

### **Integration Testing**

Other than the original API endpoints to be tested, we might need to communicate with other system APIs. If multiple systems are involved, we must ensure that the data flow is correct between them. This flawless communication is a big challenge since there are multiple dependencies.

**Pro tip:**Always try to get the bigger picture and stay on top of things. During the analysis phase, try to get the information about the other internal or external APIs with which we need to communicate.

### **Frequent Schema Updation**

Based on the business, developers might change the API schema, which might lead to updating the test cases. While there are usually few instances of this, it may be unavoidable.

**Pro tip:**Add a few assertions to validate the schema so that you will be notified if that’s not as expected. Connect with the developers frequently.

### **Test Coverage**

Exhaustive testing of APIs is not feasible most of the time. Based on the [different types](https://muuktest.com/blog/types-of-api-testing), we must prioritize the APIs’ test cases.

**Pro tip:**If there are stringent timelines,

1. Focus on the [functional testing](https://muuktest.com/blog/types-of-api-testing/#Functional-Testing) of the APIs.
2. Navigate to the other aspects based on the agreement with the stakeholders.
3. Try to implement feasible automation scripts.

### **Tool Selection Process**

For API testing, a plethora of tools are available in the market. Multiple choices leave us in difficult situations. Also, budgeting plays a role here.

**Pro tip:**Select tools based on the current team’s expertise, current low/no-code tools, and programming language knowledge. Choose the top two or three tools and do a proper Proof of Concept (PoC), which helps us to select a better tool.

### **Lack of Technical Knowledge About APIs**

If the team is newly built and they have less knowledge about API and [API terminologies](https://muuktest.com/blog/api-testing-glossary/), the process might be a little tough and lengthy. Sometimes this could lead to inefficient testing.

**Pro tip:**Try to hire the appropriate Tester with API knowledge or spend some time learning the API basics. Work closely with the developers and ask questions to help to speed up the learning process.

### **Requires Coding Knowledge**

Most API testing tools/frameworks require decent programming knowledge for API Automation.

**Pro tip:**Adopt no/low code API Automation tools, which help the quick set up for the project.

### **Unstable APIs and Underdeveloped APIs**

Not all APIs will behave the same way. Due to external dependencies, some APIs might not be available in the stipulated time. Therefore, a few APIs might behave inconsistently in the initial development phase.

**Pro tip:**Don’t wait for the entire API development to be completed. Get the sample API response from the developers and set up a mock server to start writing the assertions. Mock servers will be helpful when the APIs are unstable and you require API sequencing as well.

### **Exceptional Handling**

Handling the positive responses is a pretty straightforward scenario. However, the developer might not handle every route properly when it comes to error messages and status codes.

**Pro tip:**Testers should have detailed knowledge about the status code and the universal way of handling those scenarios.

## Conclusion:

We could make anything happen with proper planning and deeper knowledge of the hurdles Testers face. We believe this blog explained how to prepare for the API Testing process and its everyday challenges. If you plan for the worst, the journey could be smoother. We wish you good luck overcoming the challenges during your API Testing process. Do you think we’re missing anything? Let us know what your challenges are.

**Automation using Rest Assured using Junit Jupiter**

[**https://devqa.io/rest-assured-api-requests-examples/**](https://devqa.io/rest-assured-api-requests-examples/)

### **GET Request**

import io.restassured.RestAssured;

import io.restassured.http.ContentType;

import io.restassured.response.Response;

import org.junit.jupiter.api.Assertions;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;

public class RestAssuredRequests {

@BeforeAll

public static void setup() {

RestAssured.baseURI = "https://jsonplaceholder.typicode.com";

}

@Test

public void getRequest() {

Response response = given()

.contentType(ContentType.JSON)

.when()

.get("/posts")

.then()

.extract().response();

Assertions.assertEquals(200, response.statusCode());

Assertions.assertEquals("qui est esse", response.jsonPath().getString("title[1]"));

}

}

### **GET Request With Query Params**

import io.restassured.RestAssured;

import io.restassured.http.ContentType;

import io.restassured.response.Response;

import org.junit.jupiter.api.Assertions;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;

public class RestAssuredRequests {

@BeforeAll

public static void setup() {

RestAssured.baseURI = "https://jsonplaceholder.typicode.com";

}

@Test

public void getRequestWithQueryParam() {

Response response = given()

.contentType(ContentType.JSON)

.param("postId", "2")

.when()

.get("/comments")

.then()

.extract().response();

Assertions.assertEquals(200, response.statusCode());

Assertions.assertEquals("Meghan\_Littel@rene.us", response.jsonPath().getString("email[3]"));

}

}

### **POST Request**

import io.restassured.RestAssured;

import io.restassured.response.Response;

import org.junit.jupiter.api.Assertions;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;

public class RestAssuredRequests {

private static String requestBody = "{\n" +

" \"title\": \"foo\",\n" +

" \"body\": \"bar\",\n" +

" \"userId\": \"1\" \n}";

@BeforeAll

public static void setup() {

RestAssured.baseURI = "https://jsonplaceholder.typicode.com";

}

@Test

public void postRequest() {

Response response = given()

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

.and()

.body(requestBody)

.when()

.post("/posts")

.then()

.extract().response();

Assertions.assertEquals(201, response.statusCode());

Assertions.assertEquals("foo", response.jsonPath().getString("title"));

Assertions.assertEquals("bar", response.jsonPath().getString("body"));

Assertions.assertEquals("1", response.jsonPath().getString("userId"));

Assertions.assertEquals("101", response.jsonPath().getString("id"));

}

}

### **PUT Request** The PUT request updates a resource but requires the full JSON payload.

import io.restassured.RestAssured;

import io.restassured.response.Response;

import org.junit.jupiter.api.Assertions;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;

public class RestAssuredRequests {

private static String requestBody = "{\n" +

" \"title\": \"foo\",\n" +

" \"body\": \"baz\",\n" +

" \"userId\": \"1\",\n" +

" \"id\": \"1\" \n}";

@BeforeAll

public static void setup() {

RestAssured.baseURI = "https://jsonplaceholder.typicode.com";

}

@Test

public void putRequest() {

Response response = given()

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

.and()

.body(requestBody)

.when()

.put("/posts/1")

.then()

.extract().response();

Assertions.assertEquals(200, response.statusCode());

Assertions.assertEquals("foo", response.jsonPath().getString("title"));

Assertions.assertEquals("baz", response.jsonPath().getString("body"));

Assertions.assertEquals("1", response.jsonPath().getString("userId"));

Assertions.assertEquals("1", response.jsonPath().getString("id"));

}

}

### **Patch Request** - The PATCH request updates a resource but requires only the field(s) which is being updated in the payload:

import io.restassured.RestAssured;

import io.restassured.response.Response;

import org.junit.jupiter.api.Assertions;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;

public class RestAssuredRequests {

private static String requestBody = "{\n" +

" \"title\": \"bax\" \n}";

@BeforeAll

public static void setup() {

RestAssured.baseURI = "https://jsonplaceholder.typicode.com";

}

@Test

public void patchRequest() {

Response response = given()

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

.and()

.body(requestBody)

.when()

.patch("/posts/1")

.then()

.extract().response();

Assertions.assertEquals(200, response.statusCode());

Assertions.assertEquals("bax", response.jsonPath().getString("title"));

Assertions.assertEquals("1", response.jsonPath().getString("userId"));

Assertions.assertEquals("1", response.jsonPath().getString("id"));

}

}

### **Delete Request**

import io.restassured.RestAssured;

import io.restassured.response.Response;

import org.junit.jupiter.api.Assertions;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;

public class RestAssuredRequests {

@BeforeAll

public static void setup() {

RestAssured.baseURI = "https://jsonplaceholder.typicode.com";

}

@Test

public void deleteRequest() {

Response response = given()

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

.when()

.delete("/posts/1")

.then()

.extract().response();

Assertions.assertEquals(200, response.statusCode());

}

}

* You can use hamcrest library as well along with Rest Assured.
* Mocking of webservices is also a feature in Rest Assured.
* And you can use TestNG, Allure reporting, Serenity to build a robust framework.

Another [tutorial](https://scalac.io/blog/rest-assured-api-testing-tutorial-java-scala/)  : <https://scalac.io/blog/rest-assured-api-testing-tutorial-java-scala/>

* who will participate in the project, what are their (including mine) skills (can they handle the tool, if not, will there be some time for learning?)
* how big the project is - http clients are great for exploratory testing and tend to be a bit clumsy when it comes to managing many endpoints, a lot of test data, or when you want to reuse some parts (code)
* what intergations we need - how well the tool supports CI/CD pipelines, reporting, ...
* how easy it will be for someone completely new to take over - it's easier to start working with a http client than learn to code and then use a library like Rest-assured

The best setup is when you have enough space to fail. If you're not under a lot of pressure, pick one of the two, start using it and if it doesn't prove efficient on the project, change it. Yes, there'll be some rework, but you have to learn somehow, don't you?