API

Q&A

Manas Kumar Jha
Telegram: https://t.me/QAIDEAS
LinkedIn: https://www.linkedin.com/in/manas-jha/

What all challenges are included under API testing?

API testing involves testing the functionality, reliability, performance, and security of an API. Some specific challenges that may be encountered during API testing include:

- Ensuring that the API is functional and returns the correct results for different inputs.
- Verifying that the API is reliable and returns consistent results over time.
- Testing the performance of the API to ensure that it can handle a high volume of requests and has a fast response time.
- Ensuring that the API is secure and cannot be accessed by unauthorized users.
- Testing the API's error handling capabilities to ensure that it returns appropriate error messages and codes when something goes wrong.
- Testing the API's compatibility with different platforms, devices, and operating systems.
- Verifying that the API can be easily integrated into other applications.
- Ensuring that the API documentation is accurate and up-to-date.

What is the difference between API and WebService?

API (Application Programming Interface) is a set of routines, protocols, and tools for building software applications. It provides a way for developers to access and interact with application data and functions.

A WebService is an application that is accessible over a network, such as the Internet, and provides a set of functions and data that can be accessed by other applications. It is a means of communication between two applications, or components of the same application, over a network.

What is the difference between SOAP & Rest API.

SOAP (Simple Object Access Protocol) is a protocol that uses XML as its message format to exchange information between computers over the internet. SOAP is a more complex and rigid protocol, with more complex security features and requires more bandwidth for communication.

REST (Representational State Transfer) is an architectural style that defines a set of constraints and properties based on HTTP. It is a more lightweight and flexible architecture that uses HTTP and JSON as its message format. REST is simpler to use and has better performance, since it uses fewer resources than SOAP. It is also more secure since it has more built-in security features.

Can you write a sample of API(URL) and JSON.

```
API URL:

https://api.example.com/users
```

```
JSON:

{
```

```
 "users": [
  {
   "id": 1,
   "name": "John Doe",
   "email": "johndoe@example.com",
   "age": 25
  },
  {
   "id": 2,
   "name": "Jane Doe",
   "email": "janedoe@example.com",
   "age": 27
  }
 ]
}
```

How do you handle Authentication token.

Authentication tokens are typically used to store a user's identity and other information securely during a session. To handle authentication tokens, a server should first generate a token and store it in a secure location. This token should then be sent to the client with each request that requires authentication. The client should then send the token back with each request, allowing the server to verify the user's identity and process the request accordingly. Additionally, the server should regularly check that the token is still valid, and if not, generate a new one and send it back to the client.

What is difference between OAuth1.0 and OAuth2.O

OAuth1.0

- Uses HMAC-SHA1 signature method

- Requires user to authorize access via signing a request

- Uses plain text tokens

- Uses 2-legged authorization

OAuth2.0

- Uses digital signature or public/private key encryption

- Requires user to authorize access via authenticating with an access token

- Uses JSON Web Tokens (JWT)

- Uses 3-legged authorization

What is baseURI in RestAssured.

BaseURI in RestAssured is the base URL of the web service that is being tested. It is used to set the endpoint of the web service that will be tested. BaseURI is set using the RestAssured.baseURI() method.

Can you explain RequestSpecification request = RestAssured.given();

RequestSpecification request = RestAssured.given(); is an expression used to create a request that can be used in RestAssured to send a request to a specific endpoint. It is used to set up the request and define the parameters of the request such as the headers, cookies, body, and query parameters. It can also be used to set up the authentication for the request.

How to handle different Status Code

To handle different HTTP status codes when testing an API with REST Assured, you can use the statusCode method in the Response object. This method returns the HTTP status code of the HTTP response. You can then use an if statement or a switch statement to execute different blocks of code depending on the status code.

For example, you could use the following code to handle a 200 OK status code:

```
Response response = given()
    .header("Content-Type", "application/json")
    .when()
    .get("https://api.example.com/endpoint");

int statusCode = response.statusCode();

if (statusCode == 200) {
    // code to execute if the status code is 200 OK
}
```

You can also use the assertThat method to verify that the status code of the response is what you expect it to be. For example:

```
Response response = given()
    .header("Content-Type", "application/json")
    .when()
    .get("https://api.example.com/endpoint");
```

```
assertThat(response.statusCode(), equalTo(200));
```

This will cause the test to fail if the status code of the response is not 200 OK.

You can use similar techniques to handle other status codes as well. For example, to handle a 404 NOT FOUND status code, you could use the following code:

```
Response response = given()
   .header("Content-Type", "application/json")
   .when()
   .get("https://api.example.com/endpoint");

int statusCode = response.statusCode();

if (statusCode == 404) {
   // code to execute if the status code is 404 NOT FOUND
}
```

How do you print your response in JSON format

To print the response of an API in JSON format using REST Assured, you can use the prettyPrint method of the Response object. This method returns the body of the response as a formatted JSON string.

Here is an example of how you can use the prettyPrint method to print the response of an API in JSON format:

```
Response response = given()
   .header("Content-Type", "application/json")
   .when()
   .get("https://api.example.com/endpoint");

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

This will print the body of the response in a formatted JSON string. You can also use the asString method to print the body of the response as a raw string, rather than a formatted JSON string.

For example:

```
Response response = given()
   .header("Content-Type", "application/json")
   .when()
   .get("https://api.example.com/endpoint");

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

This will print the raw string representation of the body of the response.

What all are the dependencies for Rest-Assured

To use REST Assured, you will need to add the following dependencies to your project:

JUnit: This is a Java testing framework that is commonly used to write and run test cases for REST Assured.

Rest-assured: This is the core library for REST Assured. It provides the classes and methods for making HTTP requests and validating the responses.

Hamcrest: This is a library of matcher objects that can be used to create assertions in your test cases.

Apache HTTP Client: This is a Java library for making HTTP requests. It is used by REST Assured to send HTTP requests and receive HTTP responses.

To add these dependencies to your project, you will need to include them in your build tool's configuration file. For example, if you are using Maven, you can add the following dependencies to your pom.xml file:

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>4.3.1</version>
    <scope>test</scope>
```

```
  </dependency>
  <dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest-library</artifactId>
    <version>2.2</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.13</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

If you are using a different build tool, you will need to add the dependencies in a different way.

What are status code(2xx ,3xx ,4xx, 5xx) in API.

HTTP status codes are standard response codes that web servers use to communicate the status of a client's request to a server. There are five classes of HTTP status codes:

- **1xx (Informational):** The request was received, continuing process
- **2xx (Successful):** The request was successfully received, understood, and accepted
- **3xx (Redirection):** Further action needs to be taken in order to complete the request
- **4xx (Client Error):** The request contains incorrect syntax or cannot be fulfilled
- **5xx (Server Error):** The server failed to fulfill an apparently valid request

For example, a status code of 200 means that the request was successful, and a status code of 404 means that the requested resource could not be found.

How do you post body in POST and how many ways to post.

There are several ways to include a body in a POST request:

- **application/x-www-form-urlencoded**: This is the default content type for POST requests. The body is sent as a query string, with key-value pairs separated by ampersands (&) and keys and values separated by equal signs (=). This is typically used for simple forms on the web.

- **multipart/form-data:** This content type is used for file uploads. The body consists of a series of "parts", each of which can contain its own headers and a body. This is typically used for file uploads and other binary data.
- **application/json:** This content type is used for sending data in JSON format. The body is a JSON object with keys and values. This is a common format for REST APIs.
- **text/plain:** This content type is used for plain text. The body is just a string of plain text.

There are many other content types that can be used as well, but these are the most common. To specify the content type of the request body, you can set the `Content-Type` header in the request.

To send a POST request with a body, you can use the `fetch()` function in JavaScript or the `requests` library in Python, for example. Here's an example using `fetch()`:

```
fetch('https://example.com/api/endpoint', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    key: 'value'
  })
})
```

And here's an example using the requests library in Python:

```
import requests

url = 'https://example.com/api/endpoint'

headers = {'Content-Type': 'application/json'}

data = {'key': 'value'}

response = requests.post(url, headers=headers, json=data)
```

What is the difference between PUT and POST methods?

The HTTP `PUT` and `POST` methods are used to send data to a server to create or update a resource. The main difference between the two methods is the way in which the data is sent to the server and the intended use of the data.

The `PUT` method is used to send data to the server to update a resource that already exists. When you use the `PUT` method, you are replacing the current resource with a new one.

The `POST` method, on the other hand, is used to send data to the server to create a new resource. When you use the `POST` method, you are creating a new resource that is separate from the existing resources on the server.

Here are some other key differences between the `PUT` and `POST` methods:

- `PUT` is idempotent, which means that multiple identical requests will have the same effect as a single request. `POST` is not idempotent, which means that multiple requests may have different effects.
- `PUT` requests typically include the entire resource in the request body, while `POST` requests may include only a portion of the resource or a representation of the resource to be created.
- `PUT` is generally used to update existing resources, while `POST` is used to create new resources.

What is the difference between PUT and Patch. Have you ever used and where?

The HTTP PUT and PATCH methods are used to update an existing resource on the server. The main difference between the two methods is the way in which the update is applied to the resource.

- The PUT method replaces the entire resource with a new version of the resource. This means that you must send the complete updated resource to the server when using PUT.
- The PATCH method, on the other hand, allows you to apply a set of changes to the resource, rather than replacing the entire resource. This means that you can send just the changed fields to the server when using PATCH, rather than sending the complete updated resource.

I have used both PUT and PATCH in the past when working with REST APIs. The choice between the two methods usually depends on the specific requirements of the API and the use case.

For example, if you are updating a resource and you need to replace the entire resource with a new version, you would use the PUT method. On the other hand, if you only need to make a few changes to the resource and you don't want to send the complete updated resource to the server, you would use the PATCH method.

In REST Assured, you can use the put and patch methods to make PUT and PATCH requests, respectively. For example:

```
// PUT request
Response response = given()
   .header("Content-Type", "application/json")
   .body("{\"field\": \"value\"}")
   .when()
   .put("https://api.example.com/endpoint");

// PATCH request
```

```
Response response = given()
    .header("Content-Type", "application/json")
    .body("{\"field\": \"value\"}")
    .when()
    .patch("https://api.example.com/endpoint");
```

How do you create test cases for an API?

There are several steps you can follow to create test cases for an API:

- Review the API documentation: Familiarize yourself with the API by reading the documentation and understanding the different endpoints, parameters, and response codes.
- Identify the functionality to be tested: Determine the functionality that needs to be tested, such as authentication, CRUD operations, and error handling.
- Determine the test data: Decide on the data that you will use for testing, such as valid and invalid input values and response data. You may need to create test accounts or test data in your database.
- Create test cases: For each piece of functionality you want to test, create one or more test cases that exercise that functionality. A test case should include the input values, the expected output, and any other relevant details.
- Run and verify the test cases: Use a tool or program to send requests to the API with the specified input values, and verify that the response matches the expected output.
- Record and document the results: Record the results of each test case, including any errors or failures, and document your testing process. This will help you to identify and fix issues, and to maintain and update your test cases over time.

How do you decide which API to test first?

There are many factors that can influence the order in which you might choose to test different APIs. Here are a few suggestions:

- Test critical functionality first: If there are certain features of the API that are crucial to the functionality of your application, it might make sense to test those first. This will help you identify any issues with the API as early as possible.
- Test high-risk features: If there are certain features of the API that are more complex or have the potential to cause issues, it might be a good idea to test those features first. This can help you identify and resolve any issues before they become major problems.
- Test high-use features: If you expect certain features of the API to be used heavily by your application, it might be a good idea to test those features first to ensure that they are reliable and performant.
- Consider the dependencies between different features: If one feature of the API relies on another feature, it might make sense to test the dependencies between those features first. This can help you identify any issues that might impact multiple parts of your application.

Ultimately, the order in which you choose to test different APIs will depend on your specific goals and the needs of your application.

What are commonly used HTTP Methods?

There are several HTTP methods that are commonly used in web development. Here is a list of some of the most important HTTP methods:

- `GET`: The `GET` method is used to retrieve a resource from the server.
- `POST`: The `POST` method is used to send data to the server to create a new resource.
- `PUT`: The `PUT` method is used to send data to the server to update an existing resource.
- `DELETE`: The `DELETE` method is used to delete a resource from the server.
- `HEAD`: The `HEAD` method is used to retrieve the header information for a resource.
- `OPTIONS`: The `OPTIONS` method is used to retrieve the HTTP methods that are supported by a server.
- `CONNECT`: The `CONNECT` method is used to establish a tunnel to the server for the purpose of establishing a secure connection.
- `TRACE`: The `TRACE` method is used to perform a message loopback test to the server.

List out few Authentication Techniques used in API's?

There are several authentication techniques that can be used in APIs:

- **Basic authentication:** This is a simple authentication method in which the client sends an HTTP request header containing a user name and password.
- **Token-based authentication:** This is a more secure authentication method in which the client sends a token in the HTTP request header to authenticate itself.
- **OAuth:** This is an open standard for authorization that allows a client to obtain limited access to a user's resources without revealing the user's credentials.
- **JSON Web Token (JWT):** This is an open standard for securely transmitting information between parties as a JSON object. JWTs can be used as tokens for authenticating users.
- **SAML (Security Assertion Markup Language):** This is an open standard for exchanging authentication and authorization data between parties. SAML can be used to authenticate users in an API.
- **Two-factor authentication:** This is a form of authentication that requires the user to provide two different authentication factors.

What are Path Parameters and Query Parameters

Path parameters and query parameters are both used to specify additional information in an HTTP request.

Path parameters are used to identify a specific resource in the path portion of the URL. For example, in the URL `https://api.example.com/users/{userId}`, the `{userId}` path parameter is used to specify the unique identifier for a user resource.

Query parameters are used to specify additional information in the query string portion of the URL. Query parameters are often used to filter or sort data, or to specify the format of the response. For example, in the URL `https://api.example.com/users?sort=asc&limit=10`, the `sort` and `limit` query parameters are used to specify the sort order and maximum number of results to return, respectively.

Can you use GET request instead of PUT to create a resource?

No, GET request only allows read only rights. It enables you to retrieve data from a server but not create a resource. PUT or POST methods should be used to create a resource.

POST should be used when the client sends the page to the server and then the server lets the client know where it put it. PUT should be used when the client specifies the location of the page

How would we define API details in Rest Assured Automation?

In Rest Assured, you can define the details of an API by creating a `RequestSpecification` object. The `RequestSpecification` object allows you to specify the base URI, base path, query parameters, headers, and other details of the API request.

Here is an example of how you might define the details of an API in Rest Assured:

```
RequestSpecification request = given()

  .baseUri("https://api.example.com")

  .basePath("/users")

  .queryParam("sort", "asc")

  .header("Authorization", "Bearer abc123")

  .contentType(ContentType.JSON);
```

This `RequestSpecification` object specifies the base URI of the API as `https://api.example.com`, the base path as `/users`, a query parameter of `sort=asc`, an `Authorization` header with a value of `Bearer abc123`, and a content type of `ContentType.JSON`.

You can then use the `RequestSpecification` object to send an HTTP request to the API and receive a response. For example:

```
Response response = request.when().get();
```

How would you send attachments to API using Rest Assured Test?

To send an attachment to an API using Rest Assured, you can use the `attach` method of the `RequestSpecification` object.

Here is an example of how you might send an attachment to an API using Rest Assured:

```
RequestSpecification request = given()

  .baseUri("https://api.example.com")
```

```
 .basePath("/attachments")

 .header("Content-Type", "multipart/form-data")

 .multiPart("file", new File("/path/to/attachment.jpg"))

 .log().all();



Response response = request.when().post();
```

In this example, the `multiPart` method is used to attach a file located at `/path/to/attachment.jpg` to the request. The `header` method is used to set the `Content-Type` header to `multipart/form-data`, which is required for sending an attachment in a multipart request.

You can also specify other parameters or fields in the request by using the `formParam` or `queryParam` methods.

Different Status Codes and their descriptions

HTTP status codes are used to indicate the status of a request made to a server. Here is a list of some common HTTP status codes and their meanings:

- **200 OK:** This status code indicates that the request was successful and the server was able to fulfill it.
- **201 Created:** This status code indicates that a new resource was successfully created as a result of the request.
- **204 No Content:** This status code indicates that the request was successful, but the server did not send a response body.
- **301 Moved Permanently:** This status code indicates that the requested resource has been permanently moved to a new location.
- **400 Bad Request:** This status code indicates that the request was invalid or could not be understood by the server.
- **401 Unauthorized:** This status code indicates that the request requires authentication.
- **403 Forbidden:** This status code indicates that the server refuses to authorize the request.
- **404 Not Found:** This status code indicates that the requested resource was not found on the server.
- 500 Internal Server Error: This status code indicates that an internal server error occurred while processing the request.

Get the Json Path of “site” from below JSON

```
{
 "dashboard": {
  "purchaseAmount": 910,
  "website": "rahulshettyacademy.com"
 },
 "courses": [
  {
   "title": "Selenium Python",
   "price": 50,
   "copies": 6,
   "details":
   {
    "site" : "http://rahulshettyacademy.com"
   }
  },
  {
   "title": "Cypress",
   "price": 40,
   "copies": 4
  },
  {
   "title": "RPA",
   "price": 45,
   "copies": 10
  }
```

```
 ]

}
```

Answer – courses[0].details.site

To get title = “Selenium Python answer” = courses[0]. Title