1. What is Rest Assured, and why is it used?

Rest Assured is a Java-based library used to simplify testing and validating RESTful web services. It is widely used for API automation testing due to its ease of use and powerful capabilities. Here are the key reasons why Rest Assured is used:

a. Simplicity and Readability:
Rest Assured provides a simple and intuitive DSL (Domain Specific Language) for writing tests. This makes the code more readable and easier to maintain.

- Example of a basic Rest Assured test

```
given().
   when().
      get("https://api.example.com/resource").
   then().
      statusCode(200).
      body("key", equalTo("value"));
```

b. Integration with Java and Testing Frameworks:
It seamlessly integrates with Java-based testing frameworks like JUnit and TestNG, allowing for smooth test execution and reporting.

- Example of integrating with JUnit:

```
@Test
public void testApi() {
   given().
      when().
         get("https://api.example.com/resource").
      then().
         statusCode(200).
         body("key", equalTo("value"));
}
```

c. **Ease of Use:** Writing tests with Rest Assured is straightforward and requires minimal code.
d. **Comprehensive API Testing:** It supports a wide range of HTTP methods and validation techniques, allowing for thorough API testing.
e. **Integration and Extensibility**: It integrates well with existing Java projects and testing frameworks, making it a versatile choice for API automation.

f. **Community and Documentation:** Rest Assured has a strong community and extensive documentation, providing ample resources for learning and troubleshooting.

2. How do you perform a GET request in Rest Assured?

Performing a GET request in Rest Assured is straightforward and involves using its fluent API to construct and execute the request. Here's a step-by-step guide to performing a GET request in Rest Assured:

Step-by-Step Guide to Perform a GET Request in Rest Assured

1. Add Rest Assured Dependency:

Ensure that Rest Assured is added to your project's dependencies. If you're using Maven, add the following dependency to your pom.xml:

```
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>4.4.0</version> <!-- Use the latest version -->
    <scope>test</scope>
</dependency>
```

2. Import Rest Assured Classes:
Import the necessary classes in your test class.

```
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;
```

3. Perform the GET Request:

Use Rest Assured's given, when, and then methods to construct and execute the GET request.

Basic GET Request Example

Here is a simple example of a GET request using Rest Assured:

```
import io.restassured.RestAssured;
import org.junit.Test;
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;


public class ApiTest {
    @Test
    public void testGetRequest() {
        // Base URI of the API
        RestAssured.baseURI = "https://api.example.com";
        // Perform GET request and validate response
        given().
        when().
            get("/resource").
        then().
            statusCode(200).  // Validate status code
            body("key", equalTo("value"));  // Validate response body
    }
}
```

Explanation of Key Components:

- **Base URI:** Set the base URI of the API using RestAssured.baseURI.
- **Request Logging:** Use log().all() to log all details of the request.
- **Headers:** Add headers to the request using the header method.
- **Request Execution:** Use get("/resource") to perform the GET request.
- **Response Logging:** Use log().all() to log all details of the response.
- **Status Code Validation:** Validate the HTTP status code using statusCode().

- **Header Validation:** Validate response headers using the header method.
- **Body Validation:** Validate the response body using the body method and matchers from the Matchers class.

By following these steps, you can perform and validate GET requests effectively using Rest Assured.

3. What are the ways to validate a response in Rest Assured?

In Rest Assured, there are several ways to validate a response from an API. These include validating the status code, headers, cookies, and the response body. Here are the key methods to perform these validations:

1. Validating Status Code

You can check the HTTP status code of the response to ensure it matches the expected value.

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

2. Validating Headers

You can validate the presence and values of HTTP headers in the response.

```
given().
when().
    get("/resource").
then().
    header("Content-Type", equalTo("application/json")).
    header("Cache-Control", containsString("no-cache"));
```

3. Validating Cookies

You can validate cookies in the response.

```
given().
when().
    get("/resource").
then().
    cookie("session_id", notNullValue());
```

4. **Validating Response Time**
   You can validate the response time to ensure it meets performance requirements.

```
given().
when().
    get("/resource").
then().
    time(lessThan(2000L)); // Validate response time is less than 2000 milliseconds
```

5. **Validating with Custom Assertions**
   You can also perform custom validations using assertions.

```
Response response = given().
    when().
        get("/resource").
    then().
        extract().response();

assertThat(response.statusCode(), is(200));
assertThat(response.path("key"), equalTo("value"));
```

4.How do you add parameters to a request in RestAssured?

Parameters can be added to a request in RestAssured using the queryParam() or formParam() methods, depending on the type of parameter.

5.How do you add headers to a request in RestAssured?

Headers can be added to a request in RestAssured using the header() method.

6.What is the purpose of an authentication mechanism in RestAssured?

Authentication mechanisms are used in RestAssured to ensure that only authorized users can access the API. It provides a way to protect sensitive data and prevent unauthorized access.

7.How do you add authentication to a request in RestAssured?

Authentication can be added to a request in RestAssured using the auth() method, which supports different types of authentication such as basic authentication, OAuth 1.0, OAuth 2.0, and more.

- **Basic Authentication**
- Basic Authentication involves sending a username and password encoded in base64 with the request.

- **Digest Authentication**
- Digest Authentication is a more secure method compared to Basic Authentication and involves a challenge-response mechanism.

- **OAuth 1.0 Authentication**
- OAuth 1.0 requires consumer key, consumer secret, access token, and token secret.

- **OAuth 2.0 Authentication**
- OAuth 2.0 typically requires an access token, which can be sent in the Authorization header as a Bearer token.

- **Form Authentication**
- Form Authentication involves submitting a login form with username and password.

- **Custom Header Authentication**
- Some APIs require custom headers for authentication, such as API keys.

- **Bearer Token Authentication**
- Similar to OAuth 2.0, but specifically sending a Bearer token in the Authorization header.

8.What is a request body?

The request body is the data sent by a client to a server as part of an API request. It can be in different formats such as JSON, XML, or form data.

9.How do you send a request with a request body in RestAssured?

To send a request with a request body in RestAssured, use the body() method to specify the request body content and format.

10.How do you validate a response in RestAssured?

Responses can be validated in RestAssured using the assertThat() method, which supports different types of assertions such as status code, response body, headers, and more.

11.How do you extract values from a response in RestAssured?

Values can be extracted from a response in RestAssured using the extract() method, which supports different types of extraction such as path, jsonPath, xmlPath, and more.

12. How do you handle cookies in RestAssured?

Cookies can be handled in RestAssured using the cookie() method, which supports different types of operations such as adding a cookie, getting a cookie, or deleting a cookie.

13. What is difference between Put & Post?

The primary difference between PUT and POST HTTP methods lies in their intended use and the way they handle resource creation and updates on the server.

PUT Method

- **Idempotent:** A PUT request is idempotent, meaning that multiple identical PUT requests will have the same effect as a single request. This ensures that the state of the resource remains consistent regardless of how many times the request is made.
- **Resource Updates and Creation:** The PUT method is typically used to update an existing resource or create a new resource if it does not exist at the specified URI. The client sends the complete representation of the resource, and the server replaces the existing resource with the provided data.
- **URI Specific:** The PUT request is directed at a specific URI. The client is responsible for determining the URI of the resource.
- **Replacing Resources:** When using PUT, the entire resource is replaced with the data sent by the client. Partial updates are not typically handled by PUT.

POST Method

- **Not Idempotent:** A POST request is not idempotent, meaning that multiple identical POST requests may result in different outcomes. For example, submitting the same order form multiple times could create multiple orders.
- **Resource Creation:** The POST method is primarily used to create new resources. It can also be used to submit data to a server, which may then process the data and return a result.
- **Server-Determined URI:** Unlike PUT, the POST request does not specify the URI of the resource being created. Instead, the server determines the URI of the new resource and typically returns it in the Location header of the response**.**
- **Partial Updates and Actions**: POST can be used for actions that do not fit neatly into GET, PUT, or DELETE requests, such as submitting a complex query or performing an operation on a resource that results in a state change.

14. What is difference between Http/Https?

Feature	HTTP	HTTPS
Security	No encryption	Encrypted (TLS/SSL)
Encryption	None	TLS/SSL encryption
Authentication	None	Digital certificates
Port	80	443
URL Scheme	`http://`	`https://`
Performance	Slightly faster	Slightly slower
Browser Behavior	"Not Secure" warning	Padlock icon, "Secure"
SEO	No ranking boost	Ranking boost

1. Security

HTTP: Data sent using HTTP is not encrypted, meaning it can be intercepted and read by anyone who can access the data transmission. This makes HTTP unsuitable for transmitting sensitive information such as login credentials, payment details, and personal data.

HTTPS: Data sent using HTTPS is encrypted using TLS (Transport Layer Security) or SSL (Secure Sockets Layer), which ensures that the data cannot be easily intercepted and read by unauthorized parties. HTTPS provides data integrity, confidentiality, and authentication.

2. Encryption

HTTP: No encryption. Data is sent in plaintext.

HTTPS: Uses encryption protocols (TLS or SSL) to encrypt the data being transmitted, ensuring it is secure from eavesdroppers.

3. Authentication

HTTP: Does not provide any mechanism to verify the identity of the server, which can lead to man-in-the-middle attacks.

HTTPS: Provides server authentication through digital certificates issued by trusted Certificate Authorities (CAs). This helps ensure that the client is communicating with the intended server and not an imposter.

4. Port Number

HTTP: Uses port 80 by default.

HTTPS: Uses port 443 by default.

5. URL Scheme

HTTP: URLs begin with http://.

HTTPS: URLs begin with https://.

6. Performance

HTTP: Slightly faster than HTTPS because it does not involve the overhead of encryption and decryption processes.

HTTPS: Slightly slower due to the encryption and decryption overhead. However, with modern hardware and optimization techniques like HTTP/2, the performance difference is often negligible.

7. SEO and Browser Behavior

HTTP: Websites using HTTP are often flagged by modern browsers as "Not Secure," which can deter users. Major search engines like Google also give a ranking boost to websites using HTTPS.

HTTPS: Browsers display a padlock icon or a similar indicator to show that the connection is secure. Search engines prioritize HTTPS sites in search results, improving SEO.

15. What is difference between PUT & PATCH?

PUT
Definition: The PUT method replaces the current representation of the target resource with the uploaded content.
Usage: PUT is typically used when you want to update an entire resource. If the resource does not exist, PUT can also create it.
Idempotency: PUT is idempotent, meaning that making the same request multiple times will have the same effect as making it once.
Example: Updating a user's profile with all fields (e.g., name, email, address).

PATCH
Definition: The PATCH method applies partial modifications to a resource.
Usage: PATCH is used when you need to update only certain fields of a resource rather than the entire resource.
Idempotency: PATCH is not necessarily idempotent, although it can be if the partial updates do not change the resource in a way that subsequent identical requests would have different outcomes.
Example: Updating only the email of a user.

16. What is difference between OAuth1.0 and OAuth2.O?

Answer:

Complexity:
OAuth 1.0: More complex due to the requirement of cryptographic signatures.
OAuth 2.0: Simplified with bearer tokens and multiple grant types for different scenarios.

Security:
OAuth 1.0: Uses signatures for security.
OAuth 2.0: Relies on bearer tokens, scopes, and token expiration. Requires HTTPS to protect tokens during transmission.

Token Types:
OAuth 1.0: Uses request tokens and access tokens.
OAuth 2.0: Uses access tokens and refresh tokens.

Flexibility:
OAuth 1.0: Less flexible, primarily focused on web-based applications.
OAuth 2.0: More flexible with multiple grant types supporting various application types (web, mobile, server-to-server).

User Experience:
OAuth 1.0: Requires user interaction for authorization.
OAuth 2.0: Also requires user interaction but offers more streamlined flows for different types of applications.

Use Cases
OAuth 1.0: Mostly used in legacy systems and applications where the additional security provided by signatures is necessary.
OAuth 2.0: Widely adopted in modern applications, including web, mobile, and server-to-server communication, due to its simplicity and flexibility.

17. What are HTTP Methods?

Method	Description
GET	Request to read a Web page
HEAD	Request to read a Web page's header
PUT	Request to store a Web page
POST	Append to a named resource (e.g., a Web page)
DELETE	Remove the Web page
TRACE	Echo the incoming request
CONNECT	Reserved for future use
OPTIONS	Query certain options

18. What is the difference between given(), when(), and then() in RestAssured? Answer:

given(): Used to specify request parameters, headers, cookies, body, and other request details.
when(): Defines the HTTP method to be used (GET, POST, PUT, DELETE, etc.).
then(): Used to validate the response, including status code, headers, and body content

19. How do you set up RestAssured in a project?

Answer: You need to add RestAssured dependencies to your project. If you are using Maven, add the following dependency in your pom.xml file:

```
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>4.3.3</version>
    <scope>test</scope>
</dependency>
```

20. Explain the usage of the log() method in Rest Assured.

The log() method in RestAssured is used to log details of HTTP requests and responses for better visibility and debugging. It helps testers and developers to understand what exactly is being sent to and received from the server. This is particularly useful when debugging test failures or verifying that the requests and responses are as expected.

Usage of the log() Method

The log() method can be used at various points in the request-response cycle to log different parts of the request and response. Here are some common usages:

1. **Log All Request Details**: Logs all details of the request, including headers, parameters, body, and more.

Source Code in Java

```
given()
    .log().all()
.when()
    .get("https://jsonplaceholder.typicode.com/users")
.then()
    .statusCode(200);
```

2. **Log All Response Details:** Logs all details of the response, including status line, headers, and body.

   Source Code in Java:

   ```
   given()
   .when()
       .get("https://jsonplaceholder.typicode.com/users")
   .then()
       .log().all()
       .statusCode(200);
   ```

3. **Log Request Headers:** Logs only the headers of the request.

   ```
   given()
       .log().headers()
   .when()
       .get("https://jsonplaceholder.typicode.com/users")
   .then()
       .statusCode(200);
   ```

4. **Log Request Body**: Logs only the body of the request. Useful when sending complex request bodies.

```
given()
    .body("{ \"name\": \"John Doe\", \"email\": \"john.doe@example.com\" }")
    .log().body()
.when()
    .post("https://jsonplaceholder.typicode.com/users")
.then()
    .statusCode(201);
```

5. **Log Response Body**: Logs only the body of the response.

```
given()
.when()
    .get("https://jsonplaceholder.typicode.com/users")
.then()
    .log().body()
    .statusCode(200);
```

Benefits of Using the log() Method

- **Debugging:** Helps in identifying issues by providing detailed logs of requests and responses.
- **Transparency:** Makes the test scripts more transparent by showing what is being sent to and received from the server.
- **Validation:** Assists in validating that the request payloads and response data are correct.
- **Documentation:** Provides a form of documentation for the API interactions in the test logs**.**

By using the log() method effectively, you can gain deeper insights into the API interactions, making it easier to develop, debug, and maintain your API tests.

21. Assuming you have a list of map in Response body and you have to fetch the third one, what command will you use?

```
@Test(description="validate with jsonpath and json object and pass post body as json file")
public void MethodValidationPUT() throws IOException, ParseException
{
	Response resp=given().when().get("https://reqres.in/api/users");
		System.out.println(resp.path("total").toString());
		assertEquals(resp.getStatusCode(),200);
			Map<String, String> dataEmp = resp.jsonPath().getMap("data[2]");//data is a list of map
	}
```

22. How do you handle timeouts in Rest Assured requests?

Global Timeout Configuration: Set timeouts globally using RestAssuredConfig and HttpClientConfig.

Custom RequestConfig: Use RequestConfig to customize timeouts and apply them globally.

Specific Request Timeout: Configure timeouts for individual requests using config method.

1. Using the Config Module

RestAssured allows you to set the timeouts using the config module, which provides a way to configure the underlying HTTP client.

Example: Setting Connection and Socket Timeouts

```
import static io.restassured.RestAssured.*;
import static io.restassured.config.HttpClientConfig.httpClientConfig;
import static org.apache.http.params.CoreConnectionPNames.*;

import io.restassured.config.RestAssuredConfig;
import org.apache.http.params.CoreConnectionPNames;

public class ApiTest {
    public static void main(String[] args) {
        RestAssuredConfig config = RestAssured.config()
            .httpClient(httpClientConfig()
                .setParam(CONNECTION_TIMEOUT, 5000)  // Connection timeout (in milliseconds)
                .setParam(SO_TIMEOUT, 10000));       // Socket timeout (in milliseconds)

        given()
            .config(config)
        .when()
            .get("https://jsonplaceholder.typicode.com/users")
        .then()
            .statusCode(200);
    }
}
```

2. Using RequestConfig Builder

You can also create a custom RequestConfig and pass it to the HttpClientConfig.

Example: Custom `RequestConfig`

```
import static io.restassured.RestAssured.*;
import static io.restassured.config.HttpClientConfig.httpClientConfig;

import io.restassured.config.RestAssuredConfig;
import org.apache.http.client.config.RequestConfig;

public class ApiTest {
    public static void main(String[] args) {
        RequestConfig requestConfig = RequestConfig.custom()
            .setConnectTimeout(5000)   // Connection timeout (in milliseconds)
            .setSocketTimeout(10000)   // Socket timeout (in milliseconds)
            .build();

        RestAssuredConfig config = RestAssured.config()
            .httpClient(httpClientConfig()
                .setParam("http.connection.timeout", requestConfig.getConnectTimeout())
                .setParam("http.socket.timeout", requestConfig.getSocketTimeout()));

        given()
            .config(config)
        .when()
            .get("https://jsonplaceholder.typicode.com/users")
```

3. Handling Timeouts for Specific Requests

If you want to set timeouts for specific requests rather than globally, you can configure it for individual requests.

Example: Timeout for a Specific Request

```
import static io.restassured.RestAssured.*;
import static io.restassured.config.HttpClientConfig.httpClientConfig;

import io.restassured.config.RestAssuredConfig;

public class ApiTest {
    public static void main(String[] args) {
        RestAssuredConfig config = RestAssured.config()
            .httpClient(httpClientConfig()
                .setParam("http.connection.timeout", 5000)   // Connection timeout (in milliseconds)
                .setParam("http.socket.timeout", 10000));    // Socket timeout (in milliseconds)

        given()
            .config(config)
        .when()
            .get("https://jsonplaceholder.typicode.com/users")
        .then()
            .statusCode(200);
    }
}
```

23. Explain the usage of path parameters in Rest Assured.

Benefits of Using Path Parameters

1. **Readability**: Path parameters make the URL more readable and maintainable.
2. **Reusability**: They allow you to create reusable and flexible endpoint definitions.
3. **Clarity**: They clearly indicate which parts of the URL are variable.

Summary

- **pathParam():** Use this method to specify a single path parameter.
- **pathParams():** Use this method to specify multiple path parameters using a map.
- **Combination with Query Parameters**: Combine path parameters with query parameters as needed.
- RequestSpecification: Use a RequestSpecification for reusable path parameter configurations.
- **Default Path Parameters**: Set default path parameters to simplify common requests.

By using path parameters effectively, you can write cleaner and more maintainable API tests with RestAssured.

24. How do you handle response headers in Rest Assured tests?

Answer: You can use the 'header' method to validate specific headers in the response. For example, checking the 'Content-Type' header:

given() .when() .get("/endpoint") .then() .header("Content-Type", "application/json");

25. What is the purpose of the Matchers class in Rest Assured?

Answer: The Matchers class in Rest Assured provides various static methods for performing different types of assertions on the response.
For example, Matchers.equalTo(value) is used to check if a response value is equal to the expected value.

26. How do you perform a POST request with a JSON payload in Rest Assured?

Answer: To perform a POST request with a JSON payload, you can use the body() method to include the JSON content.
For example:
given() .body("{\"key\": \"value\"}") .when() .post("/endpoint") .then() .statusCode(201);

27. What is the purpose of the 'relaxedHTTPSValidation()' method in Rest Assured?

Answer: The relaxedHTTPSValidation() method is used to disable strict SSL certificate validation, allowing you to make requests to HTTPS endpoints without validating the SSL certificate.

28. What is the purpose of the 'config(JsonConfig.jsonConfig())' method in Rest Assured?

Answer: The config(JsonConfig.jsonConfig()) method is used to configure JSON serialization and deserialization settings. It allows you to customize how JSON data is processed during requests and responses.

29. Explain the purpose of the 'auth().oauth2AuthorizationCodeFlow()' method in Rest Assured.

Answer: The auth().oauth2AuthorizationCodeFlow() method is used for OAuth 2.0 authentication using the authorization code flow. It helps in handling the authentication process with the authorization server.

30. What is the purpose of the auth().none() method in Rest Assured?

Answer: The auth().none() method is used to indicate that no authentication is required for a particular request. It's helpful when dealing with public endpoints that do not require authentication.

31. How can you handle assertion failures gracefully in Rest Assured to continue with the execution of subsequent test steps?

Answer: You can use the softAssertions() method from the AssertJ library to create soft assertions, which allow the test to continue even if there are assertion failures. For example:

```
SoftAssertions softAssert = new SoftAssertions();

softAssert.assertThat(response.getStatusCode()).isEqualTo(200);
softAssert.assertThat(response.getBody().jsonPath().getString("name")).isEqualTo("John Doe");

softAssert.assertAll();
```

32. What is difference between Headers and Header class.

Answer:
Header: Represents a single HTTP header, used for specifying or retrieving individual headers.
Headers: Represents a collection of Header objects, used for specifying or retrieving multiple headers.
Representation:
Header: Represents a single HTTP header.
Headers: Represents a collection of multiple HTTP headers.
Usage:
Header: Used to specify or retrieve individual headers.
Headers: Used to specify or retrieve multiple headers at once.

Constructors:
Header: Constructor takes two arguments: the header name and the header value.
Headers: Constructor takes a varargs of Header objects or a List<Header>.

Methods:
Header: Has methods to get the name and value of the single header.
Headers: Has methods to work with the collection of headers, such as asList(), hasHeaderWithName(), getValue(), and so on.

The Header class represents a single HTTP header. It is used to specify individual headers in a request or to retrieve individual headers from a response.

Example: Creating a Single Header

```
1  import static io.restassured.RestAssured.*;
2  import io.restassured.http.Header;
3
4  public class ApiTest {
5      public static void main(String[] args) {
6          Header header = new Header("Content-Type", "application/json");
7
8          given()
9              .header(header)
10         .when()
11             .get("https://jsonplaceholder.typicode.com/posts")
12         .then()
13             .statusCode(200);
14     }
15 }
16
```

The Headers class represents a collection of Header objects. It is used when you need to specify multiple headers in a request or retrieve multiple headers from a response.

Example: Creating Multiple Headers

```
1  import static io.restassured.RestAssured.*;
2  import io.restassured.http.Header;
3  import io.restassured.http.Headers;
4  import java.util.Arrays;
5
6  public class ApiTest {
7      public static void main(String[] args) {
8          Header header1 = new Header("Content-Type", "application/json");
9          Header header2 = new Header("Authorization", "Bearer your_token");
10
11         Headers headers = new Headers(header1, header2);
12
13         given()
14             .headers(headers)
15         .when()
16             .get("https://jsonplaceholder.typicode.com/posts")
17         .then()
18             .statusCode(200);
19     }
20 }
21
```