### Explain REST?

**Ans.** REST stands for Representational State Transfer. REST is an architectural style of developing web services which take advantage of the ubiquity of HTTP protocol and leverages HTTP method to define actions. It revolves around resource where every component is a resource which can be accessed by a common interface using HTTP standard methods.

In REST architecture, a REST Server provides access to resources and REST client accesses and presents those resources. Here each resource is identified by URIs or global IDs. REST uses different ways to represent a resource like text, JSON, and XML.XML and JSON are the most popular representations of resources these days.

REST is an **architectural style** for designing web services that allows systems to communicate over the internet using standard HTTP methods. It provides a set of constraints and principles for building scalable, efficient, and stateless APIs.

What Is The Most Popular Way To Represent A Resource In REST?

**Ans.** REST uses different representations to define a resource like text, JSON, and XML.

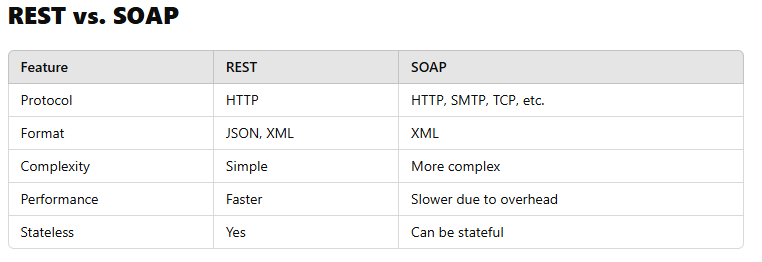
JSON is the most popular representations of resources.

### Explain What Is A “Resource” In REST?

**Ans.** REST architecture treats every content as a resource. These resources can be either text files, HTML pages, images, videos or dynamic business data.

REST Server provides access to resources and REST client accesses and modifies these resources. Here each resource is identified by URIs/ global IDs.

Any piece of information that can be identified, stored, and manipulated through a web API. Here each resource is identified by URIs/ global IDs.



### Which Protocol Is Used By RESTful Web Services?

**Ans.** RESTful web services make use of HTTP protocol as a medium of communication between client and server.

### What Is Messaging In RESTful Web Services?

**Ans.** RESTful web services make use of HTTP protocol as a medium of communication between client and server. The client sends a message in the form of an HTTP Request.

In response, the server transmits the HTTP Response. This technique is called Messaging. These messages contain message data and metadata i.e. information about the message itself.

### State The Core Components Of An HTTP Request?

**Ans.** Each HTTP request includes five key elements.

**1.** The Verb which indicates HTTP methods such as GET, PUT, POST, DELETE.  
**2.** URI stands for Uniform Resource Identifier (URI).It is the identifier for the resource on the server.  
**3.** HTTP Version which indicates HTTP version, for example-HTTP v1.1.  
**4.** Request Header carries metadata (as key-value pairs) for the HTTP Request message. Metadata could be a client (or browser) type, the format that client supports, message body format, and cache settings.  
**5.** Request Body indicates the message content or resource representation.

### ****What is Rest Assured?****

In order to test REST APIs, I found **REST Assured library** so useful. It is developed by JayWay Company and it is a really powerful catalyzer for automated testing of **REST-services**. REST-assured provides a lot of nice features, such as **DSL-like syntax**, **XPath-Validation**, **Specification Reuse**, **easy file uploads** and with those features we will handle automated API testing much easier.

### How to declare the API details in Rest Assured Test?

Using Given(), When(), Then()

### Name The Most Commonly Used HTTP Methods Supported By REST?

**Ans.** There are a few HTTP methods in REST which are more popular.

1. GET -It requests a resource at the request-URL. It should not contain a request body as it will get discarded. Maybe it can be cached locally or on the server.  
   **2.** POST – It submits information to the service for processing; it should typically return the modified or new resource.  
   **3.** PUT – At the request URL it updates the resource.  
   **4.** DELETE – It removes the resource at the request-URL.  
   **5.** OPTIONS -It indicates the supported techniques.  
   **6.** HEAD – It returns meta information about the request URL.

### Is There Any Difference Between PUT And POST Operations? Explain It.

**Ans.** PUT and POST operation are almost same. The only difference between the two is in the terms of the result generated by them.

PUT operation is idempotent (No matter how many times you send the same **PUT** request, the result will always be the same. It **updates or creates** a resource at a specific location.) while POST operation can give a different result.

### What Is URI? Explain Its Purpose In REST Based Web Services. What Is Its Format?

**Ans.** URI stands for Uniform Resource Identifier. URI is the identifier for the resource in REST architecture.

The purpose of a URI is to locate a resource(s) on the server hosting the web service. A URI is of the following format-

**<protocol>://<service-name>/<ResourceType>/<ResourceID>**

### How to compare the response values with Rest Assured Assertion?

### Example :

### given().

### parameters("firstName", "John", "lastName", "Doe").

### when().

### post("/greetXML").

### then().

### body("greeting.firstName", equalTo("John")).

### body("greeting.lastName", equalTo("Doe"));

**How to Insert cookies in Testing the API using Rest Assured?**

given().cookie("username", "John").when().get("/cookie").then().body(equalTo("username"));

**How to Insert headers in Testing the API using Rest Assured?**

given().header("MyHeader", "Something").

**How to Validate Response Headers with Rest Assured?**

get("/x").then().assertThat().header("headerName", "headerValue").

**How to handle Basic Authentication with Rest Assured?**

given().auth().preemptive().basic("username", "password").when().get("/secured/hello").then().statusCode(200);

**What Do You Understand By Payload In RESTFul Web Service?**

**Ans.** Request body of every HTTP message includes request data called as Payload. This part of the message is of interest to the recipient.

We can say that we send the payload in POST method but not in <GET> and <DELTE> methods.

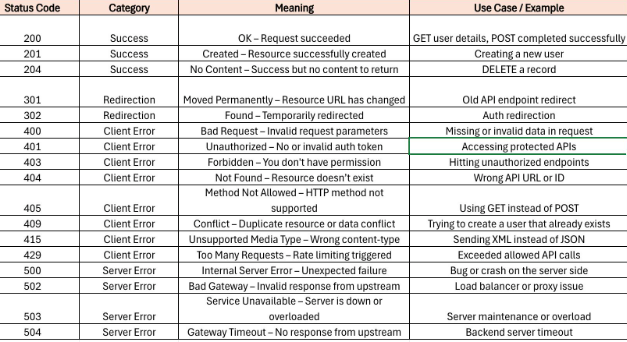
**What is the difference between resource and endpoint?**

The term **endpoint** is focused on the **URL** that is used to make a request.  
The term **resource** is focused on the **data set** that is returned by a request.

Think of a **library**:

* **Resource:** "Books" (a collection of all books).
* **Endpoints:**
  + GET /books (Get all books).
  + GET /books/10 (Get book with ID 10).
  + POST /books (Add a new book).

**HTTP Status Codes**



𝗛𝗼𝘄 𝘁𝗼 𝗢𝗽𝘁𝗶𝗺𝗶𝘇𝗲 𝗔𝗣𝗜 𝗟𝗼𝗮𝗱 𝗧𝗶𝗺𝗲?

Cache Responses – Reduce redundant processing  
Optimize Queries – Fetch only what’s needed  
Use Load Balancing – Distribute traffic smartly  
Reduce Payload – Send only essential data  
Go Asynchronous – Speed up heavy tasks

 **How has automation helped detect regression issues in a critical system?**

Automation catches unexpected changes. For example, an automated test for an e-commerce checkout system once detected a price calculation bug after a new update, preventing revenue loss.

 **What is the best testing strategy for a microservices-based application?**

Focus on API testing, contract validation, and mocking services.

Example: Testing a payment system microservice by simulating requests from the order processing service and verifying responses.

 **How do you handle flaky tests in a CI/CD pipeline?**

Use retries, stabilize test environments, and improve waiting mechanisms.

Example: A login test failed randomly due to slow API response, so adding a dynamic wait solved the issue.

 **What approach do you take when testing features that involve multiple services?**

Use integration testing with real services and mocks when necessary.

Example: Testing a bank transfer feature requires verifying transactions between account and notification services.

 **Which framework design pattern do you prefer for automation testing, and why?**

Page Object Model (POM) for UI automation and Data-driven testing for wide coverage.

Example: In a travel booking site, POM helps manage UI elements like flight search fields efficiently.

 **What are the challenges of testing asynchronous operations or eventual consistency?**

Delayed data updates and race conditions. Use polling mechanisms and timeouts.

Example: In a chat app, verifying message delivery after a delay ensures the backend syncs correctly.

 **How do you manage test data across different environments?**

Use seed scripts, data factories, and dynamic test data generation.

Example: A retail site uses predefined test accounts to ensure consistent user behavior in testing.

 **How do you ensure alignment with the test pyramid in your testing approach?**

Prioritize unit tests, followed by API tests, and fewer UI tests.

Example: A social media app heavily relies on unit tests for user actions like posting and API tests for data retrieval.

 **How do you validate logs and system metrics during automated testing?**

Use log monitoring tools like ELK Stack or CloudWatch to verify transactions.

Example: In an online payment system, checking logs for failed transactions helps detect processing errors.

𝗥𝗲𝘀𝗽𝗼𝗻𝘀𝗲 𝗧𝗶𝗺𝗲 𝗧𝗲𝘀𝘁𝗶𝗻𝗴 - Ensure performance SLAs (e.g., <200ms)  
-Use JMeter/Postman/Apidog: Measure response time under various loads.  
  
𝗗𝗮𝘁𝗮 𝗩𝗮𝗹𝗶𝗱𝗮𝘁𝗶𝗼𝗻 𝗶𝗻 𝗥𝗲𝘀𝗽𝗼𝗻𝘀𝗲 - Verify correct data types & values:  
-Use JSON Schema Validation/Postman: Validate required fields & data consistency.  
  
𝗘𝗿𝗿𝗼𝗿 𝗛𝗮𝗻𝗱𝗹𝗶𝗻𝗴 & 𝗠𝗲𝘀𝘀𝗮𝗴𝗲𝘀 - Check meaningful error responses:  
-Send invalid data: Ensure proper messages for missing parameters, incorrect payloads.  
  
𝗔𝘂𝘁𝗵𝗲𝗻𝘁𝗶𝗰𝗮𝘁𝗶𝗼𝗻 & 𝗔𝘂𝘁𝗵𝗼𝗿𝗶𝘇𝗮𝘁𝗶𝗼𝗻 - Secure API access:  
Test valid/invalid tokens & role-based access: Ensure 401 Unauthorized and 403 Forbidden work correctly.  
  
𝗕𝗼𝘂𝗻𝗱𝗮𝗿𝘆 𝗧𝗲𝘀𝘁𝗶𝗻𝗴 - Check API behavior with edge values:  
-Pass extreme inputs (empty strings, min/max values) to verify handling.  
  
𝗥𝗮𝘁𝗲 𝗟𝗶𝗺𝗶𝘁𝗶𝗻𝗴 & 𝗧𝗵𝗿𝗼𝘁𝘁𝗹𝗶𝗻𝗴 - Test API rate enforcement:  
-Send multiple requests rapidly: Expect 429 Too Many Requests when limits are exceeded.  
  
𝗔𝗣𝗜 𝗟𝗼𝗮𝗱 𝗧𝗲𝘀𝘁𝗶𝗻𝗴 - Ensure stability under high load:  
Use JMeter, Gatling, k6: Simulate concurrent users & analyze performance.