RESTful Web Services - Concept, Architecture and Communication Model

1. Introduction to REST

REST (Representational State Transfer) is an architectural style for designing web services that use HTTP for communication. It is lightweight, scalable, and widely used for building APIs for web and mobile apps.

A RESTful Web Service exposes resources (like users, products, orders) that can be accessed and manipulated using a uniform set of HTTP operations (GET, POST, PUT, DELETE, etc.).

Unlike SOAP, REST does not use XML-based envelopes or heavy protocols; it's lightweight, scalable, and simple.

Also, it is not a protocol like SOAP but a design principle that defines how web services should behave. REST emphasizes resources, representations (usually in JSON or XML), and stateless interactions.

Today, most modern APIs like Google, Facebook, and GitHub are RESTful. And REST was introduced by Dr. Roy Fielding

2. What is a Resource?

A resource is any piece of information that can be accessed via the web - for example, a student record or a course list.

Each resource has a unique identifier called a URI (Uniform Resource Identifier). URIs should be descriptive, use nouns, and follow naming conventions such as lowercase and plural words.

Examples:

https://example.com/students → Collection of students

https://example.com/students/101 → Specific student with ID 101

https://example.com/students/101/courses → Courses taken by student 101

3. Core Principles of REST Architecture

a. **Client-Server:** The client and server operate independently. The client handles the user interface, while the server handles resource storage and processing.

b. **Stateless:** Every client request must contain all the information the server needs to process it. The server does not store client session data between requests, making the system scalable.

c. **Cacheable:** Responses can be marked as cacheable, allowing clients or intermediaries to reuse data and improve performance.

d. **Uniform Interface:** REST uses a standard way to access and manipulate resources, usually through HTTP verbs and consistent URLs.

e. **Layered System:** REST architecture may include multiple layers (security, caching, proxy), but each layer remains independent.

f. **Code on Demand (optional):** The server can extend client functionality by sending code (like JavaScript) for execution.

4. HTTP Methods in REST

RESTful services use standard HTTP methods to perform operations on resources:

GET → Retrieve information from the server
POST → Create a new resource
PUT → Update or replace an existing resource
DELETE → Remove a resource
PATCH → Modify part of an existing resource

Example:

Request
GET /students/101 HTTP/1.1
Host: universityapi.edu

Response
HTTP/1.1 200 OK
{"id": 101, "name": "John Doe", "department": "ICT"}

Request

GET /students HTTP/1.1

Host: universityapi.edu

Response

HTTP/1.1 200 OK

```
[
  {"id": 101, "name": "John"},
  {"id": 102, "name": "Sara"}
]
```

5. Data Formats and Communications

REST supports multiple data formats, including XML, JSON, and plain text. However, JSON has become the most popular because it is lightweight, human-readable, and directly supported by JavaScript and most modern languages.
Typical REST communication occurs via HTTP requests and responses. Each request includes a URL, HTTP method, and optional body.
Responses usually return a status code and data representation (JSON/XML).

6. HTTP Status Codes

Code	Meaning
200	OK / Success
201	Created
204	No Content
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
500	Internal Server Error

7. URI Design Best Practices

- Use nouns not verbs (*/students* **not** */getStudents*)

- Use lowercase and plural nouns

- Avoid file extensions (.json, .xml)

8. REST vs SOAP Conceptual Comparison

Aspect	SOAP	REST
Architecture	Protocol Based	Architectural Style
Data Format	XML Only	XML, JSON or Other
Transport	Any (HTTP, SMTP)	Usually HTTP
Performance	Slower, more secure	Faster, light weight
Coupling	Tight Coupled (WSDL)	Loosely coupled
Security	WS-Security standards	HTTPS, OAuth
Use Cases	Enterprises & financial	Web & mobile APIs

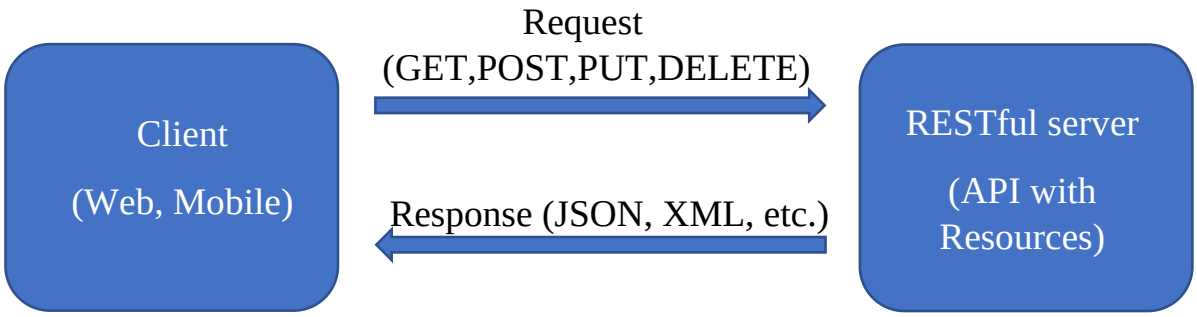
9. Advantages and Limitations

Advantages:

- Simple and easy to understand
- Lightweight and faster performance
- Platform independent
- Scalable due to stateless architecture
- Works seamlessly with browsers and mobile applications

Limitations:

- Lacks strict standards like WSDL
- Statelessness may require clients to handle more data
- REST does not have built-in security standards like WS-Security in SOAP



10. Real word examples

Service	REST Endpoint Example
Google Maps API	`GET https://maps.googleapis.com/maps/api/geocode/json?address=Colombo`
GitHub API	`GET https://api.github.com/users/ahamedaadhil/repos`
Twitter API	`POST https://api.twitter.com/2/tweets`
OpenWeatherMap API	`GET https://api.openweathermap.org/data/2.5/weather?q=London`