**REST ARCHITECTURE, API, WEB API**

**REST ARCHITECTURE**

**What is REST?**

The REST stands for **REpresentational State Transfer**.

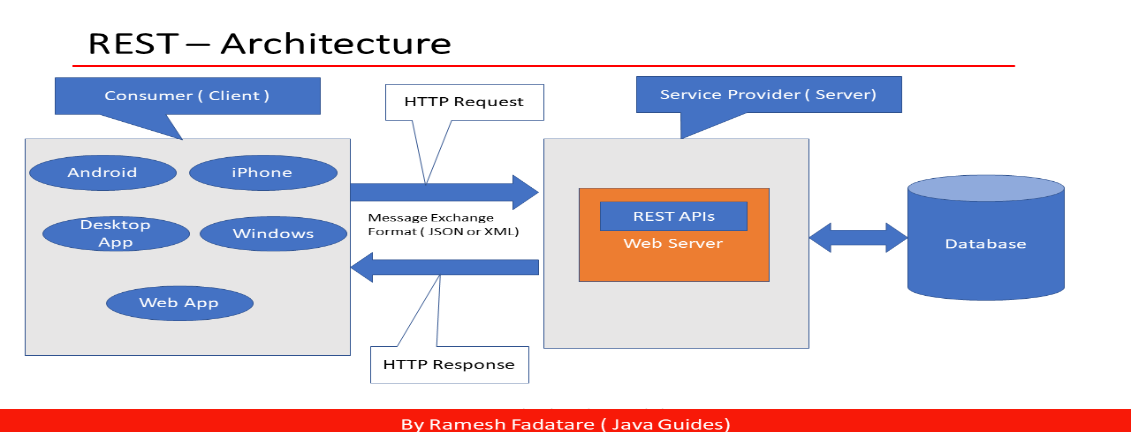
Let's understand the meaning of each word in the REST acronym.

* **State** means data
* **REpresentational** means formats (such as XML, JSON, YAML, HTML, etc)
* **Transfer** means carrying data between consumer and provider using the HTTP protocol

**REpresentational State Transfer**

* A REST API is an intermediary Application Programming Interface that enables two applications to communicate with each other over HTTP, much like how servers communicate to browsers.
* The REST architectural style has quickly become very popular over the world for designing and architecting applications that can communicate.
* The need for REST APIs increased a lot with the drastic increase of mobile devices. It became logical to build REST APIs and let the web and mobile clients consume the API instead of developing separate applications.

# **REST Architecture**

****

**Request and Response:** Request is the input to a web service, and the response is the output from a web service.

**Message Exchange Format:**It is the format of the request and response. There are two popular message exchange formats: XML and JSON.

**Service Provider or Server:** The service provider is one that hosts the web service.

**Service Consumer or Client:**A service consumer is one who is using a web service.

It is the responsibility of the consumer means client application to prepare and send HTTP request message

It is the responsibility of the business component (developed by a service provider) to prepare and send the HTTP response message

# **REST Architectural Constraints**

An API that has the following constraints is known as RESTful API:

**Client-server architecture:** The client is the front-end and the server is the back-end of the service. It is important to note that both the entities are independent of each other.

**Stateless:** No data should be stored on the server during the processing of the request transfer. The state of the session should be saved at the client’s end.

**Cacheable:**The client should have the ability to store responses in a cache. This greatly improves the performance of the API.

**Uniform Interface:** This constraint indicates a generic interface to manage all the interactions between the client and server in a unified way, which simplifies and decouples the architecture.

**Layered System:**The server can have multiple layers for implementation. This layered architecture helps to improve scalability by enabling load balancing.

**Code on Demand:**This constraint is optional. This constraint indicates that the functionality of the client applications can be extended at runtime by allowing a code download from the server and executing the code.

# **REST Key Concepts**

## Resource

The fundamental concept of a REST-based system is the resource. A resource is anything you want to expose to the outside world, through your application.

**Example 1:** Resources for Employee Management System:

- Employee

- Department

- Projects

- Task

- Address

## URI - Uniform Resource Identifier

The resource can be identified by a **Uniform Resource Identifier (URI).** For web-based systems, HTTP is the most commonly used protocol for communicating with external systems. You can identify a unique resource using a URI.

Consider, we are developing a simple blog application and you can define URIs for a blog Post resource:

GET—**http://localhost:8080/api/posts/:** Returns a list of all posts

GET—**http://localhost:8080/api/posts/2**: Returns a post whose ID is 2

POST—**http://localhost:8080/api/posts/**: Creates a new Post resource

PUT—**http://localhost:8080/api/posts/2:** Updates a POST resource whose ID is 2

DELETE—**http://localhost:8080/api/posts/2:** Deletes a POST resource whose ID is 2

**HTTP Methods**

REST APIs enable you to develop any kind of web application having all possible CRUD (create, retrieve, update, delete) operations. REST guidelines suggest using a specific HTTP method on a particular type of call made to the server.

# **HTTP GET Method**

**Use GET requests to retrieve resource representation/information only** – and not to modify it in any way. As GET requests do not change the state of the resource, these are said to be safe methods. Additionally, GET APIs should be idempotent, which means that making multiple identical requests must produce the same result every time until another API (POST or PUT) has changed the state of the resource on the server.

If the address is successfully received (error-free), GET returns a JSON or XML representation in combination with the HTTP status code 200 (OK). In case of an error, the code 404 (NOT FOUND) or 400 (BAD REQUEST) is usually returned.

**Example:** HTTP GET - **http://www.usermanagement/api/users/me** - Get logged in user profile

# **HTTP POST Method**

The HTTP POST request is most commonly used to create new resources. When talking strictly in terms of REST, POST methods are used to create a new resource into the collection of resources.

Upon successful creation, an HTTP code 201 is returned, and the address of the created resource is also transmitted in the ‘Location’ header.

**Examples:**

HTTP POST - **http://www.domain/api/users** - Create User

HTTP POST - **http://www.domain/api/posts**- Create Post

# **HTTP PUT Method**

Use PUT APIs primarily to update existing resource (if the resource does not exist, then API may decide to create a new resource or not). If a new resource has been created by the PUT API, the origin server MUST inform the user agent via the HTTP response code 201 (Created) response and if an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request.

**Examples:**

HTTP PUT - **http://www.domain/api/users/{username} -**Update user

HTTP PUT - **http://www.domain/api/posts/{id}** - Update post by id

# **HTTP DELETE Method**

Use DELETE APIs to delete resources (identified by the Request-URI).

A successful response of DELETE requests SHOULD be HTTP response code 200 (OK) if the response includes an entity describing the status, 202 (Accepted) if the action has been queued, or 204 (No Content) if the action has been performed but the response does not include an entity.

**Examples:**

DELETE **http://www.domain/api/users/{username}** - Delete user

DELETE **http://www.domain/api/posts/{id} -**Delete post

**HTTP Status Code**

Some of the frequently used status codes in this class are as follows:

* **200 OK:**This code indicates that the request is successful and the response content is returned to the client as appropriate.
* **201 Created:** This code indicates that the request is successful and a new resource is created.
* **400 Bad Request:** This code indicates that the server failed to process the request because of the malformed syntax in the request. The client can try again after correcting the request.
* **401 Unauthorized:**This code indicates that authentication is required for the resource. The client can try again with appropriate authentication.
* **403 Forbidden:** This code indicates that the server is refusing to respond to the request even if the request is valid. The reason will be listed in the body content if the request is not a HEAD method.
* **404 Not Found:**This code indicates that the requested resource is not found at the location specified in the request.
* **500 Internal Server Error:**This code indicates a generic error message, and it tells that an unexpected error occurred on the server and that the request cannot be fulfilled.

**API**

In Java, API (Application Programming Interface) refers to a set of classes, interfaces, methods, and constants that provide a way for developers to interact with a particular software component, library, framework, or service. APIs define how software components should interact, allowing developers to use pre-defined functionalities without needing to know the implementation details.

There are various types of APIs in Java, including:

1. **Standard Java APIs**: These are APIs provided by the Java Development Kit (JDK) and Java Standard Edition (Java SE) platform. Examples include:
   * Java Collection Framework (e.g., List, Set, Map)
   * Java I/O API (e.g., FileReader, FileWriter)
   * Java Networking API (e.g., Socket, URLConnection)
   * Java Concurrency API (e.g., Thread, Executor)
   * Java Database Connectivity (JDBC) API
2. **Third-party APIs**: These are APIs developed by third-party vendors or open-source communities to provide additional functionalities beyond the standard Java APIs. Examples include:
   * Apache Commons libraries (e.g., Apache Commons Lang, Apache Commons IO)
   * Google Gson for JSON processing
   * Jackson JSON for JSON processing
   * JUnit for unit testing
   * Apache HttpClient for HTTP client
3. **Framework APIs**: Frameworks in Java often come with their own APIs that define how developers can use and extend the framework's functionalities. Examples include:
   * Spring Framework (e.g., Spring Core, Spring MVC)
   * Hibernate ORM (Object-Relational Mapping)
   * Jersey or Spring MVC for building RESTful web services
   * Apache Struts for web application development
4. **Service APIs**: These are APIs provided by web services, cloud platforms, or external services to interact with their functionalities programmatically. Examples include:
   * Twitter API for accessing Twitter data
   * Google Maps API for integrating maps into applications
   * AWS SDK for accessing Amazon Web Services
   * JavaMail API for sending and receiving emails

APIs play a crucial role in software development by providing a standardized way for developers to interact with software components and services. They abstract away the complexities of implementation details, promote code reusability, and facilitate integration between different software systems. Developers can leverage APIs to build applications more efficiently and effectively by focusing on their specific business logic rather than low-level details

**WEB API**

A Web API (Application Programming Interface) is an interface that allows communication and interaction between different software systems over the internet using standard web protocols and formats. Web APIs enable applications to access and manipulate resources or perform specific operations on remote servers, typically through HTTP (Hypertext Transfer Protocol) requests and responses.

Here are some key characteristics and components of Web APIs:

1. **HTTP Methods**: Web APIs utilize standard HTTP methods such as GET, POST, PUT, DELETE, and others to perform CRUD (Create, Read, Update, Delete) operations on resources. Each HTTP method has a specific meaning:
   * GET: Retrieve a resource or resource representation.
   * POST: Create a new resource.
   * PUT: Update an existing resource or create a new resource if it doesn't exist.
   * DELETE: Delete a resource.
   * And others: PATCH, HEAD, OPTIONS, etc.
2. **Resource URIs**: Resources in a Web API are typically identified by URIs (Uniform Resource Identifiers). Clients access and manipulate these resources by making HTTP requests to the corresponding URIs.
3. **Data Formats**: Web APIs use standard data formats such as JSON (JavaScript Object Notation) or XML (eXtensible Markup Language) to represent resource data in request and response payloads. JSON has become the de facto standard for data exchange in modern Web APIs due to its simplicity and readability.
4. **Authentication and Authorization**: Web APIs often require authentication and authorization mechanisms to control access to resources and ensure security. Common authentication methods include API keys, OAuth tokens, and HTTP Basic/Digest authentication.
5. **Error Handling**: Web APIs should provide meaningful error responses in case of failures or exceptions. Error responses typically include HTTP status codes (e.g., 400 Bad Request, 404 Not Found) and error messages in the response body.
6. **Documentation**: Good Web APIs are well-documented, providing detailed information about available resources, endpoints, parameters, request/response formats, and usage examples. API documentation helps developers understand how to use the API effectively.

Web APIs are commonly used for various purposes, including:

* Integrating with third-party services and platforms
* Exposing functionality of web applications as reusable services
* Building client-server communication in web and mobile applications
* Enabling microservices architecture and service-oriented architecture (SOA)
* Facilitating machine-to-machine (M2M) communication in the Internet of Things (IoT)

Popular examples of Web APIs include the Twitter API, Google Maps API, GitHub API, and many others. These APIs provide access to a wide range of functionalities and services, allowing developers to build innovative applications and solutions by leveraging existing resources and infrastructure.

**REST vs API vs WEB API**

The terms REST, API, and Web API are often used interchangeably, but they refer to different concepts in the context of software development. Here's a breakdown of their meanings and differences:

1. **API (Application Programming Interface)**:
   * An API is a set of rules, protocols, and tools that allows different software applications to communicate with each other. It defines the methods and data formats that developers can use to interact with a particular software component, service, library, or platform.
   * APIs can be implemented in various forms, including libraries, frameworks, protocols, and web services. They provide a standardized way for developers to access and use functionalities without needing to know the internal details or implementations of the software components.
   * APIs can be used for a wide range of purposes, such as integrating with third-party services, accessing system resources, building user interfaces, and automating tasks.
2. **Web API**:
   * A Web API is a type of API that is accessible over the internet using standard web protocols such as HTTP (Hypertext Transfer Protocol). It allows clients to interact with remote servers or services by making HTTP requests and receiving HTTP responses.
   * Web APIs are commonly used for building web services, exposing functionalities of web applications, integrating with cloud platforms and external services, and enabling machine-to-machine communication over the web.
   * Web APIs typically follow the principles of REST (Representational State Transfer) architecture, although they can also use other architectural styles such as SOAP (Simple Object Access Protocol) or GraphQL.
3. **REST (Representational State Transfer)**:
   * REST is an architectural style for designing networked applications, particularly web services. It defines a set of constraints and principles for building scalable, maintainable, and interoperable web services.
   * RESTful web services adhere to these principles, such as using standard HTTP methods (GET, POST, PUT, DELETE) for CRUD operations, representing resources with URIs (Uniform Resource Identifiers), using stateless communication, and supporting various data formats (e.g., JSON, XML) for resource representations.
   * While Web APIs can follow REST principles, not all Web APIs are strictly RESTful. Some APIs may deviate from REST constraints or use different architectural styles while still being accessible over the web.

In summary, an API is a broader concept that encompasses various types of interfaces for software communication, while a Web API specifically refers to APIs accessible over the internet using web protocols. REST, on the other hand, is an architectural style that guides the design of networked applications, including RESTful web APIs.