**What is REST Stands for?**

REST stands for REpresentational State Transfer.

**What is REST?**

REST is web standards-based architecture and uses HTTP Protocol for data communication

**Name some of the commonly used HTTP methods used in REST based architecture?**

* **GET** − Provides a read only access to a resource.
* **PUT** − Used to create a new resource.
* **DELETE** − Used to remove a resource.
* **POST** − Used to update an existing resource or create a new resource.
* **OPTIONS** − Used to get the supported operations on a resource.

**Idempotent REST APIs**

**In context of REST APIs, when making multiple identical requests has the same effect as making a single request – then that REST API is called idempotent.**

**When you design REST APIs, you must realize that API consumers can make mistakes. They can write client code in such a way that there can be duplicate requests as well. These duplicate requests may be unintentional as well as intentional some time (e.g. due to timeout or network issues). You have to design fault-tolerant APIs in such a way that duplicate requests do not leave the system unstable.**

**An idempotent HTTP method is a HTTP method that can be called many times without different outcomes. It would not matter if the method is called only once, or ten times over. The result should be the same. It essentially means that the result of a successful performed request is independent of the number of times it is executed. For example, in arithmetic, adding zero to a number is idempotent operation.**

**If you follow REST principles in designing API, you will have automatically have idempotent REST APIs for GET, PUT, DELETE, HEAD, OPTIONS and TRACE http methods. Only POST APIs will not be idempotent.**

**POST is NOT idempotent.**

**GET, PUT, DELETE, HEAD, OPTIONS and TRACE are idempotent.**

**Let’s analyze how above HTTP methods end up being idempotent – any why POST is not.**

**HTTP POST**

**Generally – not necessarily – POST APIs are used to create a new resource on server. So, when you invoke the same POST request N times, you will have N new resources on server. So, POST is not idempotent.**

**HTTP GET, HEAD, OPTIONS and TRACE**

**GET, HEAD, OPTIONS and TRACE methods NEVER change the resource state on server. They are purely for retrieving the resource representation or meta data at that point of time. So, invoking multiple requests will not have any write operation on server, so GET, HEAD, OPTIONS and TRACE are idempotent.**

**HTTP PUT**

**Generally – not necessarily – PUT APIs are used to update the resource state. If you invoke a PUT API N times, very first request will update the resource; then rest N-1 requests will just overwrite the same resource state again and again – effectively not changing anything. Hence, PUT is idempotent.**

**HTTP DELETE**

**When you invoke N similar DELETE requests, first request will delete the resource and response will be 200 (OK) or 204 (No Content). Other N-1 requests will return 404 (Not Found). Clearly, the response is different from first request, but there is no change of state for any resource on server side because original resource is already deleted. So, DELETE is idempotent.**

**Please keep in mind if some systems may have DELETE APIs like this:**

**DELETE /item/last**

**In above case, calling operation N times will delete N resources – hence DELETE is not idempotent in this case. In this case, a good suggestion might be to change above API to POST – because POST is not idempotent.**

**POST /item/last**

**Now, this is more closer to HTTP spec – hence more REST compliant.**

**Define One-to-one Mapping Between Http Methods and Crud Operations?**

PUT maps to Create, GET maps to Retrieve, POST maps to Update, DELETE maps to Delete. .

**What Are Idempotent Methods?**

A method is said to be idempotent if repeated calls to it does not cause duplicates.

**In Order to Answer Http Put Requests, How Will You Annotate Your Resource Methods?**

The @PUT annotation is used to answer HTTP PUT requests.

**Why Put Methods Is Idempotent?**

PUT is used to create a resource and repeating the operation should never create another copy of the resource.

**List Resource Method Designator Annotations?**

@GET, @PUT, @POST, @DELETE

**How Can You Define A Regular Expression for A Variable Say Username?**

@Path("users/{username: [a-zA-Z][a-zA-Z\_0-9]\*}")

**What Happens If Regular Expression of a Uri Embedded Variable Is Not Matched?**

A 404 (Not Found) response will occur.

**How Can You Declare More Than One Media Type in The Same @produces Declaration?**

@Produces({"application/plain", "application/xml", "application/json"})

**How Will You Specify the Mime Media Types of Representations A Restful Resource Can Use?**

@Consumes annotation is to be used.

**Give an Example of Declaring More Than One Media Type In @consumes Declaration?**

@POST   
@Consumes({"application/plain”, “application/xml"})   
public void postMessage(String message) {   
...   
}

**Can Restful Web Services Support Multiple Types of Response (mime) Formats?**

RESTful Web Services, by confirming to HTTP, does support multiple types of response (MIME) formats e.g. XML, JSON, PLAIN etc.