### REST (Representational State Transfer)

**REST** stands for Representational State Transfer.

**HATEOAS** stands for Hypermedia as the engine of Application state.

**Rest** is an **architectural style based on the Http protocol** **to help computer system communicate over the internet**.

It is always **easier to build loosely coupled microservices** using Restful API.

REST uses Http protocol for data communication. Each resources are identified by URI

**In REST architecture, a REST server simply provides access to Resources** and **REST client accesses and presents the resources**.

**Http Standard methods** like GET, POST, PUT and DELETE **are** **used to access resources in RESTful web service architecture.**

**Messages are the collection of information about the data**.

RESTful webservices uses HTTP protocol as a medium of communication between client and server to transfer information. A client sends a message in form of a Http Request and server responds in form of HTTP Response. This technique is termed as Messaging. These messages contain message data and meta data.

**Core Components of Request**

**verb** - includes methods like GET,POST,PUT, etc.

**URI** - Uniform Resource Identifier for identifying the resources on the server.

**HTTP Version** - Specifying the Http version.

**Request Header** - **Contains meta data for the Http Request message as key-value pairs**.

**Request body** - that **contains the representation of the resources** in use.

**Core Components of Response**

**Request Code**: This contains various codes which determine the status of the server response.

**Response body:** that contains the representation of the resources in use.

**Http Request** contains **Http Method, Http URL, Http headers, Http Body**.

**Http Response** contains **Http Status code, Http Headers, Http Body**.

**Http Status code** basically are **the representation of** **the status of the task** **that has been performed on the server**, with the mode of some code.

Every code has their meaning.

Some of the Http Status codes with meaning are as follows:

Code 200 : success(OK)

code 201 : successfully created

code 204 : no content in the response body.

code 404 : This indicates that there is no method available.

**REST specifies a set of architectural constraints. Any Services which satisfies these constraints called REST Services.**

1. Client - Server 2. Stateless 3. Caching 4. Uniform Interface 5. Layered

**1. Client - Server**

- **Uniform interface separates the client and server**.

- **This means that client application and server application must be able to evolve separately without any dependency on each other. A client should know only resource URI’s and that’s all.**

- **Servers and clients may be replaced and developed independently** as long as **the interface alters**.

**2. Stateless**

**- As per REST architecture, RESTful web service should not keep the client state on server. This restriction are called Stateless.**

**- It is the responsibility of the client to pass its context to server and then server can store this context to process client further requests.**

**For example, session maintained by server is identified by session identifier passed by the client.**

**Advantages**

* Web services can treat each method request independently.
* Web services need not maintain client’s previous interactions. It simplifies application design.

**Disadvantages**

* Web services need to get extra information in each request and then interpret to get the client’s state in case client interactions are to be taken care of.

**- Make all client-server interaction stateless.**

**- The server will not store anything about the Http Request the client made. It will treat every request as new. No Session, No history.**

**-** If the client application needs to be a stateful application for the end-user, where user logs in once and do other authorized operations after that, then **each request from the client should contain all the information necessary to service the request – including authentication and authorization details.**

- **URI uniquely identifies the resources** and **the body contains the state(or state change) of that resource**.

- **Then after server does it processing**, **the appropriate state** or the pieces of state that matter, **are communicated back to the client via headers, status, and response body.**

**3. Cacheable**

**Caching refers to storing the server response in client itself so that a client need not to make server request for same resource again and again.**

**A server response should have information about how a caching is to be done so that a client caches response for a period of time or never caches server response.**

**Caching of data and responses is of utmost important** wherever they are applicable/possible.

Caching **brings performance improvement for the client-side** and **better scope for scalability for a server because reduced load**.

In REST, caching shall be applied to resources when applicable, and then these resources MUST declare themselves cacheable. Caching can be implemented on the server or client-side.

**4. Uniform Interface**

- Uniform interface constraints **defines the interaction between client and server**.

- It simplifies and decouples the architecture, which enables **each part to evolve independently**.

**Four guiding principles of Uniform interfaces are**

i. **Identification of Resources**

**Individual resources are identified** **in requests using URI's** as resource identifier.

ii. **Manipulation of resources through representations**

The resource representations across the system should follow specific guidelines such as naming conventions, link formats, or data format (XML or/and JSON).

All resources should be accessible through a common approach such as HTTP GET and similarly modified using a consistent approach.

iii. **Self-Descriptive message**

**Each message includes the information** to describe how to process the message.

iv. **HATEOAS(Hypermedia as the engine of Application state)**

- Client delivers state via body contents, query string parameters, request headers and the requested URI ( the resource name).

- **Server delivers state via body contents, response code and response headers**.

- This is technically referred as **Hypermedia**.

- HATEOAS also means that **links are contained in the returned body** ( or headers) **to supply the URI for retrieval of the object itself or related objects**.

**5. Layer System**

- REST **allows you to use a layered system architecture** where you deploy the APIs on server A, and store data on server B and authenticate requests in Server C, for example. A client **cannot ordinarily tell whether it is connected directly to the end server**, or to an intermediary along the way.

- Intermediatory server **may improve system scalability by enabling load balancing** and **by providing the shared caches**.

**OPTIONS method**

* It should list down the supported operations in a web service and should be read only.

**HEAD method**

* It should return only Http Header
* It has no body.
* It should be read only.

**Safe method**

**Method Requests do not change the state of the resource**, these are said to be safe methods.

**GET and HEAD methods should be used only for retrieval of resource representations** – **and they do not update/delete the resource on the server**.

Both methods are said to be considered “safe“.

This allows user agents to represent other methods, such as POST, PUT and DELETE, in a special way, so that the user is made aware of the fact that a possibly unsafe action is being requested – and they can update/delete the resource on server and so should be used carefully.

**Idempotent operations**

**Idempotent operations** means their result will always be same no matter how many times these operations are invoked.

operation that **will produce the same results if identical requests executed** **multiple times**.

In HTTP specification, **the methods GET, HEAD, PUT and DELETE are declared idempotent methods.**

**Use Appropriate HTTP Request methods**

**Http Get Request methods**

- Get requests are used to **read or retrieve specific resource or a collection of resources.**

- 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**.

- For any given Http GET API, **if the resource is found on the server**, then it must return Http response code 200(OK) - **along with response body**, **which is usually either XML or JSON content**.

- **In case resource is not found on server** then it must return HTTP response code 404(Not Found).

Similarly, if it is determined that **GET request itself is not correctly formed** **then server will return Http response code 400(Bad Request)**.

- **It should never modify any resource on the server**.

|  |
| --- |
| Get requests are used to read or retrieve specific resource or a collection of resources. |
| If the resource is found on the server, it will return Http Response Code 200(OK) along with response body, which is usually either XML or JSON content.  In case resource is not found on the server, it will return Http Response code 404(Not Found).  If the Get request itself is not correctly formed, it will return Http response code 400(Bad Request). |
| Get is safe operation. It does not change the state of the resource. It should never modify any resource on the server. |
| Get is idempotent. It produces the same result if executes multiple times. |

**Http Post Request methods**

- Post APIs **are used to create new resources**.

- Ideally, **if the resource has been created on the origin server**, **the response should be Http response code 201(Created)** and contains an entity which describes the status of the request and refers to the new resource an a location header.

- On successful creation, return Http Status 201(Created)

**If the resource already exists, return 409(conflicts)** or 404(not found).

- Post is **neither safe nor idempotent** and invoking two identical post request will produce the different results if executed multiple times.

|  |
| --- |
| Post requests are used to create a new resource. |
| If the resource has been created, it will return Http Response code 201(Created)  If the resource is already existing, it will return **409(Conflicts)** |
| Post is neither safe nor idempotent. It will produce the different results if executed multiple times. |

**Http Put Request methods**

- PUT APIs primarily **used to update specific resources by an identifier** or collection of resources.

- **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 code** should be sent to indicate successful completion of the request.

- Put is not **safe operation in that it modifies or creates state on the server**.

-But **it is idempotent**. If you update a resource using PUT and then make the same call again, the resource is still there and still has the same state as it did with first call.

|  |
| --- |
| Put requests are used to update **a specific resource by an identifier**. If the resource does not exist, then it creates new resource. |
| If a new resource has been created, it returns http response code 201(Created).  If an existing resource has been modified, it return http response code either 200(OK) or 204(No Content) |
| Put is not safe operation in that it modifies or creates state on the server. |
| PUT is idempotent. Repeatedly calling PUT API on the **resource will not change the outcome** |

**Http Patch request methods**

- Http Patch requests **are used to make partial update on a resource**.

- If you see PUT requests **also modify a resource** entity. **PATCH method is the correct choice for partially updating an existing resource and PUT should only be used if you are replacing a resource in its entirely.**

- Patch is **neither safe nor idempotent.**

|  |
| --- |
| Patch requests are used to make partial update on a specific resource. |
| Patch method is correct choice for partially updating an existing resource and Put method should be used only if you are replacing resource entirely |
| Patch method is neither safe nor idempotent |

**Http Delete Request methods**

- **Delete APIs are used to delete a specific resource by an identifier**. Resources identified by a 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**.

- **DELETE operations are idempotent. If you DELETE a resource, it’s removed from the collection of resources**. **Repeatedly calling DELETE API on that resource will not change the outcome** – **however, calling DELETE on a resource a second time will return a 404 (NOT FOUND) since it was already removed**. Some may argue that it makes the DELETE method non-idempotent. It’s a matter of discussion and personal opinion.

|  |
| --- |
| * Delete requests are used to delete a specific resource by an identifier. |
| * After successful deletion, it returns http response code 200(OK) if the response includes entity or else * It returns http response code 204(No Content) if the response does not include entity. * 202(Accepted) if the action has been queued. |
| * Delete operations are not safe. It modifies state on the server. |
| * Delete operations are idempotent. Repeatedly calling Delete API on the resource will not change the outcome, however, calling delete on resource a second time, it returns a 404(Not Found) |

**REST Resource Naming Guide**

The **URL is sentence, where resources are nouns and http methods are verbs**.

1. **Use nouns for resource identification**

GET – /users - Return a list of users.

GET - /users/10 – return a specific user

POST- /users – create a new user

POST- /users/10 – update a specific user

DELETE -/users/10 – delete a specific user

2. **User proper headers**

- Content-type defines the request format.

- Accept defines a list of acceptable response formats

3. **Get method and query parameters should not alter the state.**

4. **Versioning**

- /blog/api/v1/users

- /blog/api/v2/users

5. **Provides the links for navigating through your API(HATEOAS)**

Hypermedia As The Engine Of Application State is a principle that **hypertext link should be used to create** **better navigation through the API**.

6. **Use plurals in URL segments to keep your API URIs consistent across all HTTP methods**, using the collection metaphor.

**Recommended**: /customers/33245/orders/8769/lineitems/1

**Not**: /customer/33245/order/8769/lineitem/1

7. **Design for your clients, not for your data.**

8. **Use identifiers in your URLs instead of in the query-string.** Using URL query-string parameters is fantastic for filtering, but not for resource names.

**Good**: /users/12345

**Poor**: /api?type=user&id=12345

Always keep static contents like images, CSS, JavaScript cacheable with expiration date of 2 to 3 days.

Never keep expiry date too high.

Dynamic contents should be cached for few hours only.

**A HttpRequest has five major parts**

* Verb - indicates Http methods such as GET, POST, PUT,PATCH,DELETE. – identifies the operation to be performed on the resource.
* URI – Uniform Resource Identifier – identify the resource on the server.
* Http version – indicates Http Version
* Request Header – contains meta data for the Http Request message as key value pairs.

For example, client or browser type, format supported by client, format of message body, cache settings etc.,

* RequestBody – Message content or resource representation.

**A HttpResponse has four major parts**

* Status/Response code – indicates the server status for the requested resource.
* Http version – indicates Http Version
* Response Header – contains meta data for the Http Response message as key value pairs.

For example, content length, content type, response date, server type etc.,

* Response Body – Response message content or resource representation.

**Date Header** provides the date and time of the resource when it was created.

**Last Modified Header** provides the date and time of the resource when it was last modified.

**Cache-Control header** is primary header to control caching.

**Expires header** set expiration date and time of caching.

**Public directive** indicates that resource is cacheable by any component.

**Private directive** indicates that resource is catchable by only client and server, no intermediatory can cache the resource.

**No-cache/no-store directive** indicates that resource is not cacheable.

**Max-age** directive indicates that caching is valid up to max-age in seconds. After this, client has to make another request.

**Must-revalidate** directive provides indication to server to revalidate the resource if max-age has passed.

**URI (Uniform Resource Identifier)**

* URI stands for Uniform Resource Identifier. Each resource in REST architecture is identified by its URI.
* Purpose of the URI is to locate a resource on the server hosting the web service.
* URI format - <protocol>://<service-name>/<Resource Type>/<Resourced>

**Difference between REST and SOAP.**

There is no real comparison between REST and SOAP.

* REST is set of architectural principles defining how a restful service should look like.
* SOAP is message exchange format . SOAP defines the structure of the message to exchanged.

* REST implementation is JSON/XML over Http.
* SOAP implementation is XML over SOAP over Http.
* REST has better performance and scalability. REST reads can be cached.
* SOAP based reads cannot be cached.
* REST permits many different data formats.
* SOAP only permits XML data format only.