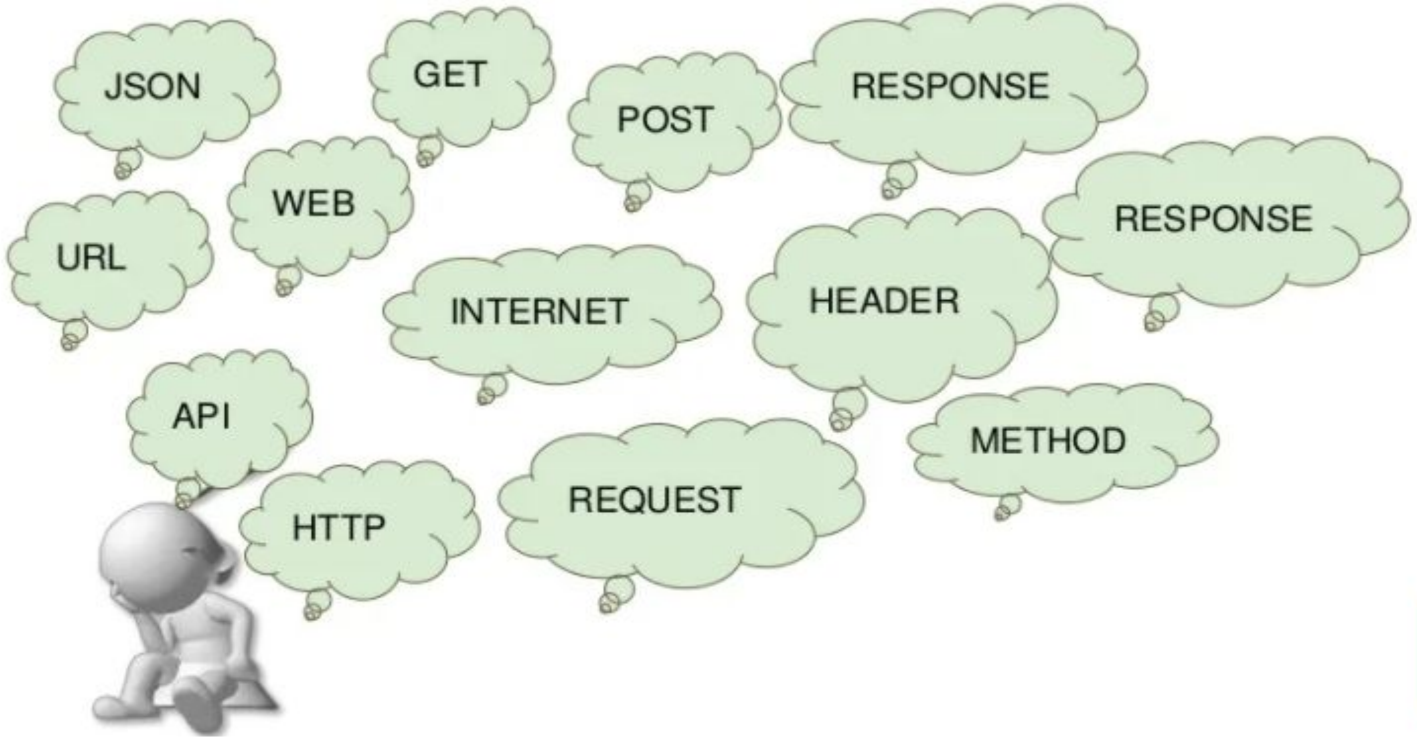
An Introduction To REST API

Created : Partha Goswami

What is it ?


JSON
GET
POST
RESPONSE
WEB
URL
RESPONSE
INTERNET
HEADER
API
METHOD
REQUEST
HTTP

What is it ?

REST means

REpresentational

State

Transfer

REpresentational ? State ? Transfer ?

It represent the state of database at a time. But how???

REST is an architectural style which is based on web-standards and the **HTTP** protocol.

In a REST based architecture everything is a **Resource**.

A resource is accessed via a common interface based on the HTTP standard methods.

You typically have a REST server which provides access to the resources and a REST client which accesses and modifies the REST resources.

REpresentational ? State ? Transfer ?

Every resource should support the HTTP common operations.

Resources are identified by global IDs (which are typically **URIs** or **URLs**).

REST allows that resources have different representations, e.g., text, XML, JSON etc.

Stateless in nature. **Excellent** for **distributed system.**

Stateless components can be freely redeployed if something fails, and they can **scale** to accommodate load changes.

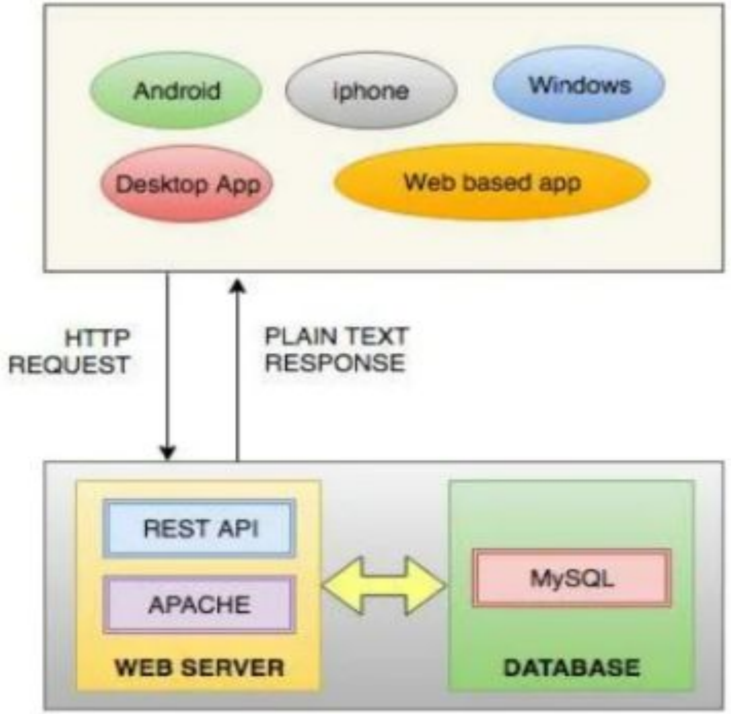
This is because any request can be directed to any instance of a component.

HTTP Methods

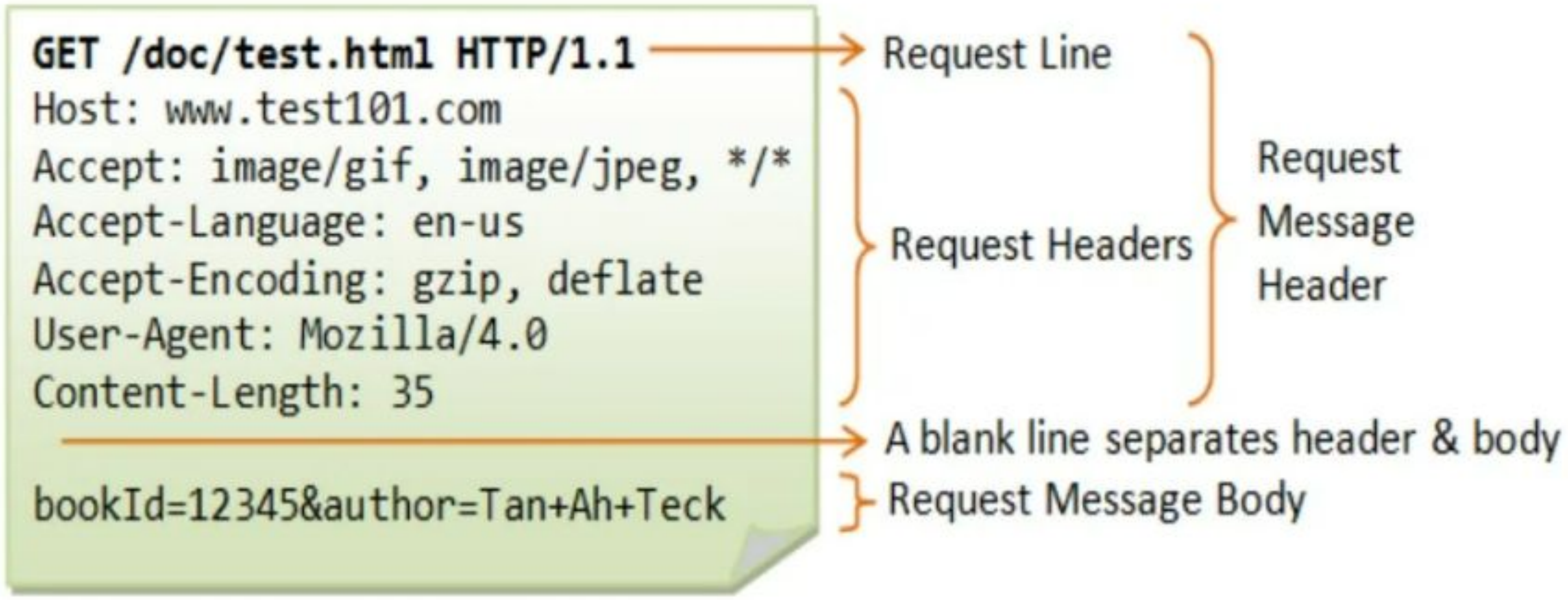
The *PUT*, *GET*, *POST* and *DELETE* methods are typically used in REST based architectures. The following table gives an explanation of these operations:

HTTP Method	CRUD Operation	Description
POST	INSERT	Addes to an existing resource
PUT	UPDATE	Overrides existing resource
GET	SELECT	Fetches a resource. The resource is never changed via a GET request
DELETE	DELETE	Deletes a resource

Architecture:



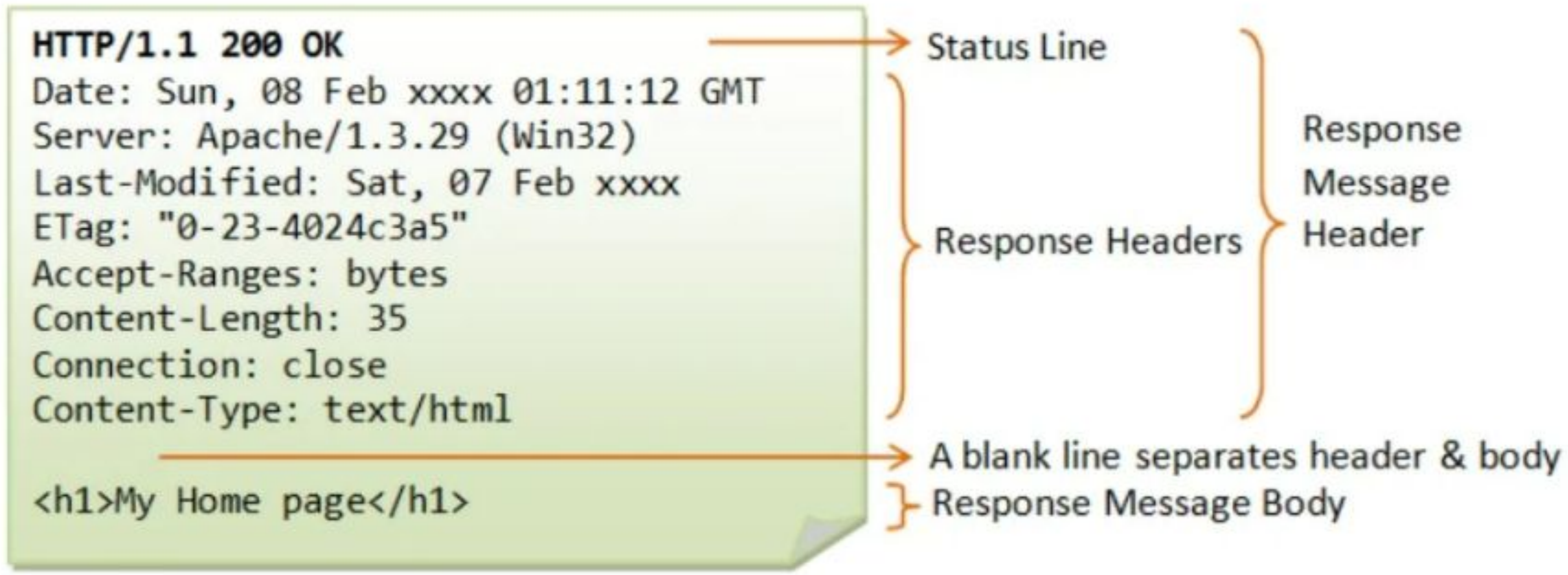
HTTP Request Example:



```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

HTTP Response Example:



```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

HTTP REST Request:

GET https://www.myhost.com/api/v1/user/1/cities

Read, All the cities for user whose id is 1

```
GET /user/1/cities http/1.1
host: https://www.myhost.com/api/v1
Content-Type: application/json
Accept-Language: us-en
state_id: 2
```

HTTP REST API Request

HTTP REST Response:

```
HTTP/1.1 200 OK (285ms)
Date: Fri, 21 Apr 2017 10:27:20 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.1e-fips PHP/7.0.16
X-Powered-By: PHP/7.0.16
Content-Length: 109
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json; charset=UTF-8
- - - - - - - - - - - - - - - - - - - - - - - - - - -
{"status":"success","message":"City
List","data":[{"city_name":"Visakhapatnam"},{"city_name":"Vijayawada"}]}
```

HTTP REST API Response

HTTP Response Status Code:

1xx	Informational Codes
2xx	Successful Codes
3xx	Redirection Codes
4xx	Client Error Code
5xx	Server Error Codes

Use Nouns but no verbs in path/URI:

Purpose	Method	Incorrect	Correct
Retrieves a list of users	GET	/getAllCars	/users
Create a new user	POST	/createUser	/users
Delete a user	DELETE	/deleteUser	/users/10
Get balance of user	GET	/getUserBalance	/users/11/balance

Use plural nouns:

Do not mix up singular and plural nouns. Keep it simple and use only plural nouns for all resources.

/cars instead of /car

/users instead of /user

/products instead of /product

/settings instead of /setting

GET method should not alter the state:

Use **PUT, POST** and **DELETE** methods instead of the **GET** method to alter the state.

Do not use **GET** method or Query parameters for state changes:

GET /users/711?activate or

GET /users/711/activate

Use sub-resources for relation:

If a resource is related to another resource use subresources

```
GET /cars/711/drivers/  (Returns a list of drivers for car 711)

GET /cars/711/drivers/4  (Returns driver #4 for car 711)
```

Use HTTP headers for serialization formats:

Both, client and server need to know which format is used for the communication. The format has to be specified in the HTTP-Header.

Content-Type defines the request format.

Accept defines a list of acceptable response formats.

Versioning is important:

Make the API Version mandatory and do not release an unversioned API.

Use a simple ordinal number and avoid dot notation such as 2.5

We are using the url for the API versioning starting with the letter “v”

/blog/api/v1

Handle Errors with HTTP Status code:

200	OK (Everything is working)	403	Forbidden (The server understood the request, but is refusing it or the access is not allowed)
201	OK (New resource has been created)	404	Not found (There is no resource behind the URI)
204	OK (Resource successfully deleted)	405	Method not allowed
400	Bad Request (The request was invalid or cannot be served. The exact error should be explained in the error payload. E.g. „The JSON is not valid“)	408	Request timeout
401	Unauthorized (The request requires an user authentication)	500	Internal server error

Filtering:

Use a unique query parameter for all fields or a query language for filtering.

GET /cars?color=red (Returns a list of red cars)

GET /users?name=tom (Returns a list of users whose name matches tom)

Sorting:

Allow ascending and descending sorting over multiple fields.

```
GET  /cars?sort=-manufacturer,+model
```

This returns a list of cars sorted by descending manufacturers and ascending models)

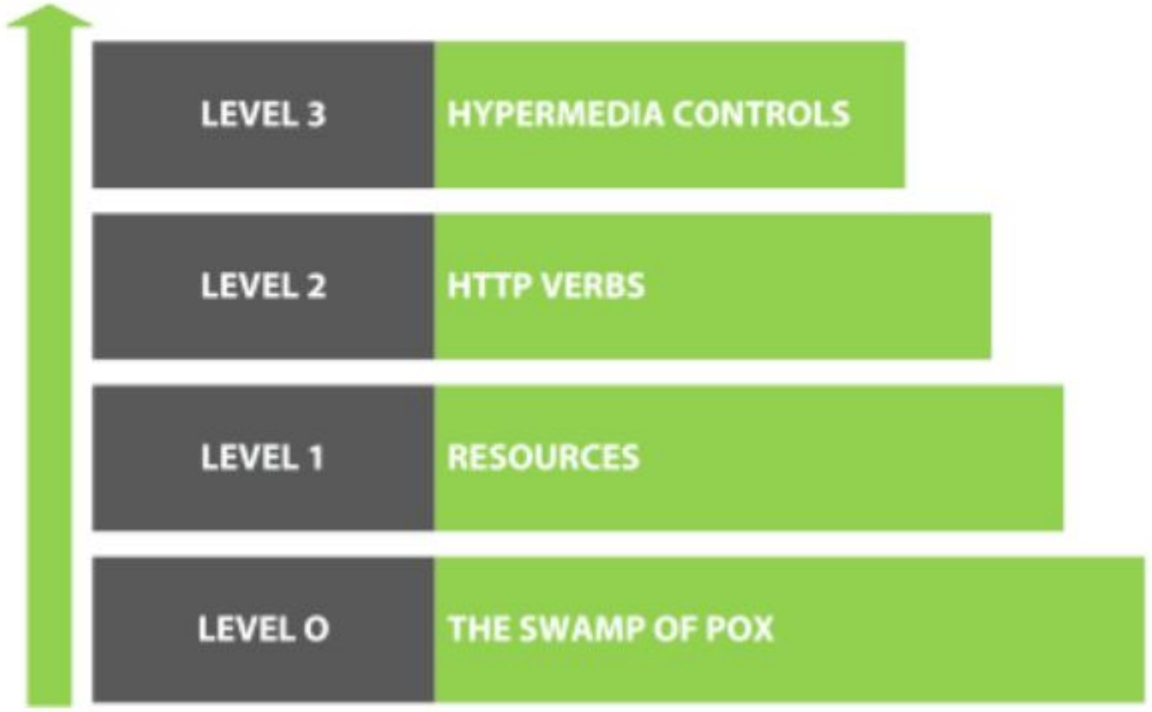
Paging:

Use limit and offset. It is flexible for the user and common in leading databases.

The default should be limit=20 and offset=0

```
GET /cars?offset=10&limit=5
```

Richardson Maturity Model



Level 0 : The Swamp of POX

Level 0 is also known as POX (Plain Old XML). At level 0, HTTP is used only as a transport protocol that is used as a remote interaction. It does not take the advantages of HTTP like different HTTP methods, and HTTP cache. To get and post the data, we send a request to the same URI, and only the POST method may be used. These APIs use only one URI and one HTTP method called POST. In short, it exposes SOAP web services in the REST style.

Level 1 : Resources

When an API can distinguish between different resources, it might be at level 1. It uses multiple URIs. Where every URI is the entry point to a specific resource. It exposes resources with proper URI. Level 1 tackles complexity by breaking down huge service endpoints into multiple different endpoints. It also uses only one HTTP method POST for retrieving and creating data.

Level 2 : HTTP Verbs

Level 2 indicates that an API must use the protocol properties to deal with scalability and failures. At level 2, correct HTTP verbs are used with each request. It suggests that in order to be truly RESTful, HTTP verbs must be used in API. For each of those requests, the correct HTTP response code is provided.

Level 3 : Hypermedia Controls

Level 3 is the highest level. It is the combination of level 2 and HATEOAS. It also provides support for HATEOAS. It is helpful in self-documentation.

For example, if we send a GET request for customers, we will get a response for customers in JSON format with self-documenting Hypermedia.

HATEOAS

HATEOAS (Hypermedia as the Engine of Application State) is a constraint of the REST application architecture. HATEOAS keeps the REST style architecture unique from most other network application architectures.

The term “hypermedia” refers to any content that contains links to other forms of media such as images, movies, and text.

REST architectural style lets us use the hypermedia links in the API response contents. It allows the client to dynamically navigate to the appropriate resources by traversing the hypermedia links.

Navigating hypermedia links is conceptually the same as browsing through web pages by clicking the relevant hyperlinks to achieve a final goal.

Thanks for your Time & Patience!

Any Question?