Exploring REST and RESTful Web Services: A review

Chinmaya Kannur[1], Mamatha T[2]

[1]*Under Graduate Student, Dept. of Computer Science and Engineering, RV College of Engineering, Bengaluru, Karnataka*
[2]*Assistant Professor, Dept. of Computer Science and Engineering, RV College of Engineering, Bengaluru, Karnataka*

---***---

Abstract - *In order to make powerful, reliable and user-friendly and web applications and web services, developing consolidated web applications is very important. In this paper we will showcase the goals of REST, the idea of REST and RESTful web service design principles and features of RESTful services with examples. In recent times, REST gained more popularity and is being widely used for web services development. Learning RESTful web Service can make the web development easier in many ways.*

Key Words: **Rest, Restful Web Services, Representation, Caching, Stateless.**

1.INTRODUCTION

REST (REpresentational State Transfer), is derived from Roy Fielding's Ph.D. thesis [1]. In his thesis, REST architectural style was introduced and developed as a hypothetical model of the web architecture to guide the renovation and interpretation of the URIs and HTTP. This style characterizes the base of WWW (World Wide Web). Few technologies are used to help in putting together this base include mark-up languages such as HTML (Hyper Text Mark-up Language) and XML (Extensible Mark-up Language), Hypertext Transfer Protocol (HTTP), Uniform Resource Identifier (URI) and web-favourable formats such as JSON. REST is an architectural approach for structured applications. The target of REST to create a reliable web program and to take it further. For instance, when a user clicks a link on a web page, the program would form another web page, return to the user and progress further. For example, a client requests the details of the bus running from source A to destination B, the web page in the return would include the list of buses run on that particular route and the links to the corresponding timetable. When the client would get the result, he/she can choose a link to find which bus to board. REST is not a protocol, it is a derived architecture for characterizing a stateless, caching client-server distributed-media platform. A REST architecture can be achieved using several communication protocols, but HTTP is the most preferred protocol.

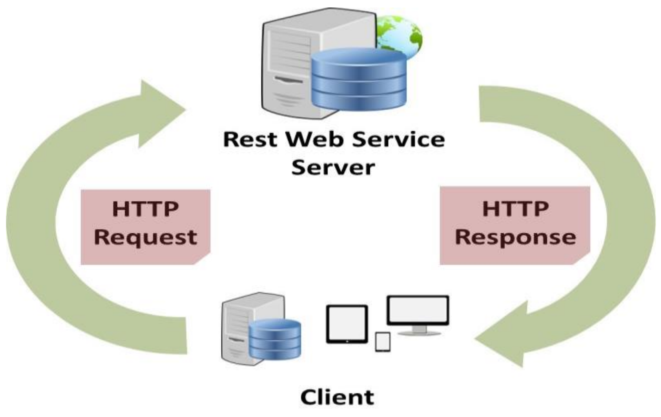


Fig - 1: REST architecture

1.1 Goals of REST

Main goals of REST are as follows:
(1) Interactions among components should be scalable
(2) Abstract interfaces
(3) Deployment of components independently
(4) Intermediate components for reducing latency in interaction and security enforcement
The components in the REST system should adhere to the following constrain:
(1) Resource identification
(2) Resource manipulation through representations
(3) Self-explanatory messages.

1.2 Concept of REST

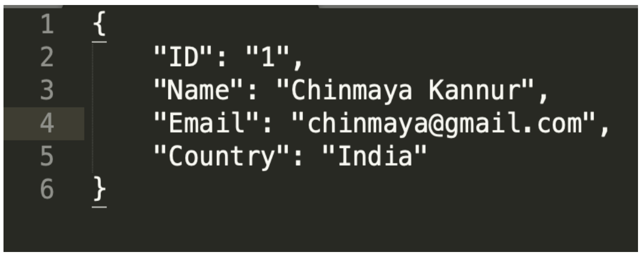
In REST system, all the resources have an URI tagged to themselves. Using the standard HTTP methods, a user can access the resources. Developers have to examine every method's syntax to decide the suitable methods for each resource. The GET method must be safe for clients to call many times because returns a depiction of a resource and returning a different depiction every time is not acceptable. The PUT method puts a new state in the resource replacing the older state, while DELETE method is used to delete resources. The POST method is used for creating or extending resources.

2. Features of a RESTful services

Resources are integral part of each and every system in the world. Resources can be anything which can be represented in a computer-based system such as, business information, web pages, video files, pictures, etc. The aim of a service is to give its clients a window, so that resources can be accessed by the clients. For high quality performance of clients' RESTful services are expected to have following features:

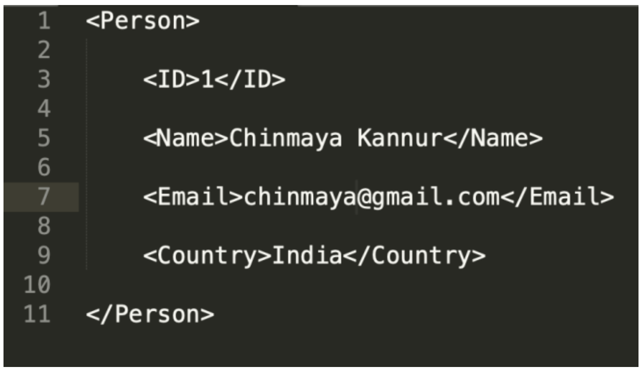
- Representation
- Message
- URI
- Uniform interface
- Stateless
- Caching

Representations: The main point in a RESTful service is about resources and how to give access to those resources. The first thing to do while designing a system, is to identify the resources and identify the relationship among themselves. This step is identical to the first step in database designing: identifying entities and relations. When the resources are identified, the next thing to do is finding a way to represent the resources in the system. Any format can be used to represent the resources, as REST does not stick to a specific format for representations. Generally, XML or JSON are preferred. For example, a "Person" resource is depicted as:



```
{
    "ID": "1",
    "Name": "Chinmaya Kannur",
    "Email": "chinmaya@gmail.com",
    "Country": "India"
}
```

Fig - 2: JSON representation of a resource



```
<Person>

    <ID>1</ID>

    <Name>Chinmaya Kannur</Name>

    <Email>chinmaya@gmail.com</Email>

    <Country>India</Country>

</Person>
```

Fig - 3: XML representation of a resource

More than one format can be used depending on the category of client or some request criteria. For a representation to be considered as good, it should have few mandatory qualities:

• Client and server must be capable of understanding the format of representation.

• A resource must be represented completely by the representation. If there is need to represent a resource partially, then the resource has to be broken down into child resources. This would make it easy for transfer of resources and it will take less time in resource creation. Hence making a service faster.

• Representation must be able to link resources among each other. This can be ensured by mentioning the URI of the linked resource in the representation.

Messages: The client and server communicate with each other through messages which contains some metadata. Client sends a request to the server, and the server interprets and replies back with suitable response.

HTTP Request

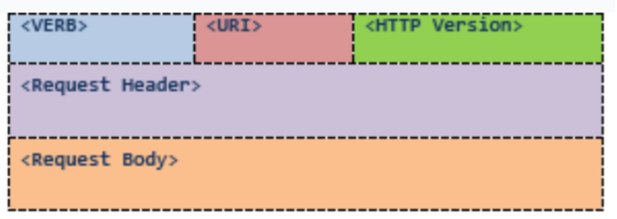
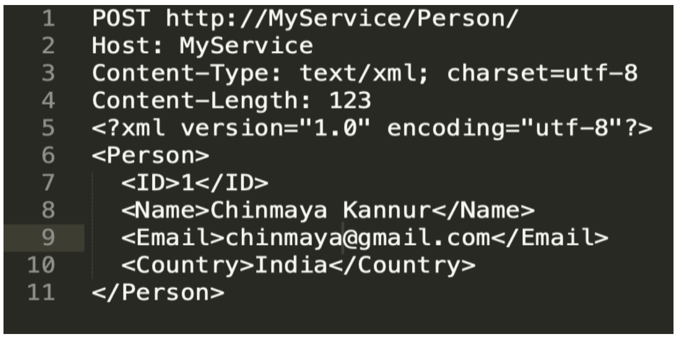


Fig - 4: HTTP request format

- VERB: indicates the HTTP method used.
- URI: is the resource URI on which operation is going to be done.
- HTTP Version: HTTP v1.1.
- Request Header: has the metadata in the form of a set of key/value pairs of headers and its corresponding values. These headers contain details about the message and its sender such as, authorization used, formats supported by the client, type of format of message body, cache configurations for response, type of client and much more details.
- Request Body: is the absolute message content. This is the place which shows the resource representations.

Fig - 4 is a sample request message using POST method, which will place a new resource called "Person".



```
POST http://MyService/Person/
Host: MyService
Content-Type: text/xml; charset=utf-8
Content-Length: 123
<?xml version="1.0" encoding="utf-8"?>
<Person>
  <ID>1</ID>
  <Name>Chinmaya Kannur</Name>
  <Email>chinmaya@gmail.com</Email>
  <Country>India</Country>
</Person>
```

Fig - 5: Sample request using POST method

HTTP Response

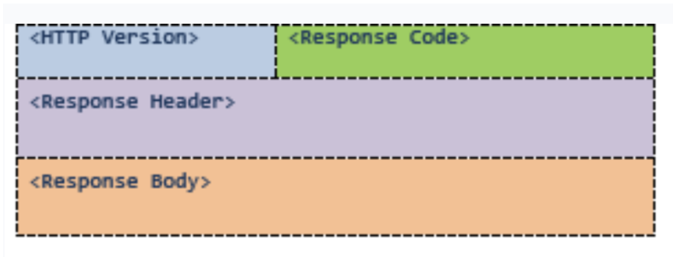
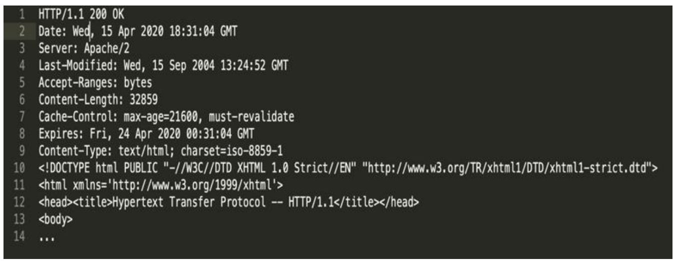


Fig - 6: HTTP response format

- Response code: server returns a 3-digit response code, which indicates request status.
- Response Header: has the metadata about the response.
- Response Body: has the representation if request was successful.

Fig - 5 is a sample of response received for the request mentioned in fig - 3:



```
HTTP/1.1 200 OK
Date: Wed, 15 Apr 2020 18:31:04 GMT
Server: Apache/2
Last-Modified: Wed, 15 Sep 2004 13:24:52 GMT
Accept-Ranges: bytes
Content-Length: 32859
Cache-Control: max-age=21600, must-revalidate
Expires: Fri, 24 Apr 2020 00:31:04 GMT
Content-Type: text/html; charset=iso-8859-1
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml'>
<head><title>Hypertext Transfer Protocol -- HTTP/1.1</title></head>
<body>
...
```

Fig - 7: An actual response to a GET request

Addressing Resources: Each resource should have at least a single URI tagged to it in REST service. The function of an URI is to determine a resource or a set of resources. A RESTful service maintains a directory with hierarchy of URIs. The actual action is decided by the HTTP verb. The URI should not reveal anything about the operation or action because this will allow the client to call same URI with various HTTP verbs to perform various actions.

For an instance there is a database of persons and to show it to any end user, person resource can be addressed like this:

http://myservice/persons/1

Fixed format for URL:
Protocol://ServiceName/ResourceType/ResourceID

These are few suggestions for well-maintained URIs:

- For naming resources make use of plural nouns.
- Do not use spaces, they may create confusion. Use an underscore or hyphen instead.
- A URI is case insensitive.
- Stick to a single naming convention.
- A URI never changes.

Query Parameters in URI

The following URI has a query parameter:

http://myservice/persons?id=1

However, this approach has a few disadvantages.

- Will reduce readability and increases complexity.
- Search engine indexers and crawlers do not catch URIs with query parameters. This will hide the web service from search engines.

The basic aim of query parameters is to give parameters to an operation that requires the data items. For example, if the format of the presentation has to be decided by the client, then it can be done like this:

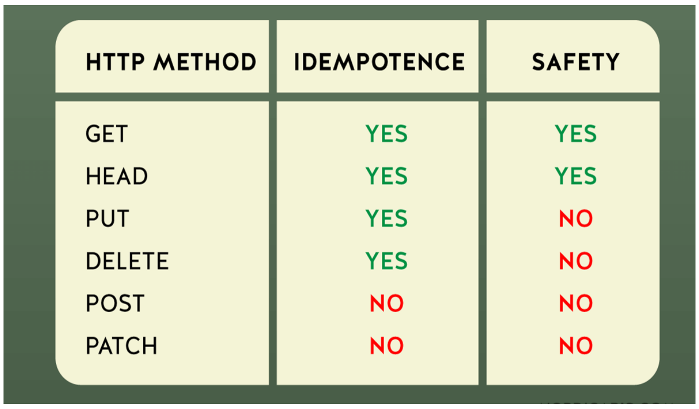
http://myservice/persons/1?format=xml&encoding=UTF8

Putting the query parameters in the URI as h hierarchy itself will not make any sense because, they do not possess any relation among themselves:

http://myservice/persons/1/xml/UTF8

Query parameters should only be used when there is a need to provide parameter values to a process.

Uniform Interface: Uniform interface is necessary for RESTful services. For this purpose, HTTP 1.1 gives a set of methods, called as verbs. Few of important verbs:



HTTP METHOD	IDEMPOTENCE	SAFETY
GET	YES	YES
HEAD	YES	YES
PUT	YES	NO
DELETE	YES	NO
POST	NO	NO
PATCH	NO	NO

Fig - 8: Verbs

An operation is called as safe if it does not affect the original value of the resource. An operation is called as idempotent when an operation gives the same result no matter how many times you perform it. Similarly, a safe HTTP method does not create any changes to the resources on a server. An Idempotent HTTP method does not depend on number of times it is performed.

Statelessness: A RESTful service is a stateless service. A request is not dependent on a past request and a service assumes each request as an independent request. By design HTTP is a stateless protocol, and extra effort has to be put to design a stateful service with help of HTTP. A clear understanding has to be there about a stateful and stateless design to avoid misinterpretation.

A sample of stateless design URI:

Request1: GET http://myservice/pets/1 HTTP/1.1

Request2: GET http://myservice/pets/2 HTTP/1.1

A stateful design URI sample:

Request1: GET http://myservice/pets/1 HTTP/1.1

Request2: GET http://myservice/nextpet HTTP/1.1

To execute the Request2, the server should know the PetID that the client last requested i.e. the server has to know the current state — or else Request2 will not be executed. Therefore, designing a stateless service is more preferred, so that it can never refer to previous state. Stateless services are easier to maintain, scalable, easy to host. Even these types of services give good response time to requests.

Caching: Caching is a technique of storing the produced results and using those stored results in the near future instead of producing the results repeatedly. This process can be done either on the client side, server side, or on any intermediate component between client and server, such as a proxy server. Caching is a good way to improve the working of REST service, but if not handled properly, it can serve stale results to the client.
Caching can be controlled using these HTTP headers:

- Date – date and time of representation generation
- Last Modified - date and time when the server last modified this representation
- Cache-control - HTTP 1.1 header is used for control caching
- Expires - expiry date and time of the representation.
- Age - duration passed in seconds since the information was fetched from the server

Values of the above headers can be used along with the directives in a Cache-Control header to check if the cached results are still valid or not. The most common directives for Cache-Control header are:

- Public
- Private
- no-cache/no-store
- max-age

3. CONCLUSIONS

This paper introduced the basic knowledge about RESTful web service, introduced the goals of REST, the idea of REST, RESTful web service design principles and features of RESTful services with examples. With the need of REST and the RESTful web service in the market, it is a must for a Java developer to know REST. Because of the REST, RESTful web service will be more important in Java Enterprise Applications development in coming years.

REFERENCES

[1] R. T. Fielding, "Architectural Styles and the Design of Network- based Software Architectures," California, Irvine: University of California, 2000

[2] J. Sandoval, Restful Java Web Services. Packt Publishing, 2009.

[3] L. Richardson and S. Ruby, RESTful Web Services .O'Reilly Media, 2007.

[4] S.Vinoski, RESTful Web Services Development Checklist.

[5] https://www.drdobbs.com/web-development/restful-web-services-a-tutorial

[6] Xuan Shi. "Sharing Service Semantics using SOAP-Based and REST Web Services.". IT PRO. Vol 8, Issue 2, March/April 2006. pp18-24 in press

[7] Cesare Pautasso, Olaf Zimmermann and Frank Leymann. "RESTful Web Services VS. "Big" Web Services: Making the Right Architectural Decision". in press

[8] Leonard Richardson and Sam Ruby. "RESTful Web Services". Dongnan Industry Press, 2007, pp243-251.