RESTful Web Services

RESTful Web Services

- REST stands for Representational State Transfer
- More than a decade after its introduction, REST has become one of the most important technologies for Web applications.
- Every major development language now includes frameworks for building RESTful Web services.

RESTful Web Services

- It is primarily used to build Web services that are
  - lightweight
  - maintainable
  - scalable
- A service based on REST is called a RESTful service.
- REST is not dependent on any protocol, but almost every RESTful service uses HTTP as its underlying protocol.

Features of RESTful Web Services

- Every system uses resources.
- These resources can be pictures, video files, Web pages, business information, or anything that can be represented in a computer-based system.
- The purpose of a service is to provide a window to its clients so that they can access these resources.

Features of RESTful Web Services

- RESTful services should have following properties and features:
  - Representations
  - Messages
  - URIs
  - Uniform interface
  - Stateless
  - Links between resources
  - Caching

Representations

- The focus of a RESTful service is on resources and how to provide access to these resources.
- A resource can easily be thought of as an object as in OOP. A resource can consist of other resources.

Representations

- While designing a system, the first step is to identify the resources and determine how they are related to each other. This is similar to the first step of designing a database: Identify entities and relations.
- Second step: find a way to represent these resources. You can use any format for representing the resources, as REST does not put a restriction on the format of a representation.

Representations

```
{
    "ID": "1",
    "Name": ”Jessica Miller",
    "Email": ”j.miller@gmail.com",
    "Country": ”USA"
}
```

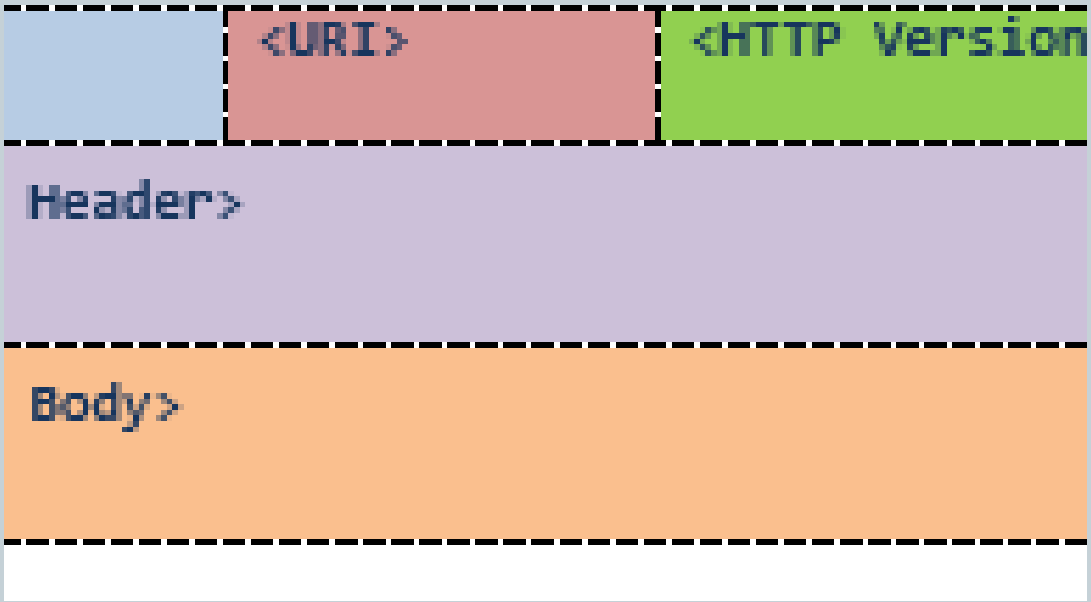
```
<Person>
<ID>1</ID>
<Name>Jessica Miller</Name>
<Email>j.miller@gmail.com</Email>
<Country>USA</Country>
</Person>
```

Representations

- Whichever format you use, a good representation should have some obvious qualities:
- Both client and server should be able to comprehend this format of representation.
- A representation should be able to completely represent a resource.
- The representation should be capable of linking resources to each other.

Messages

The client and service talk to each other via messages. Clients send a request to the server, and the server replies with a response.



HTTP Request Format

HTTP Request

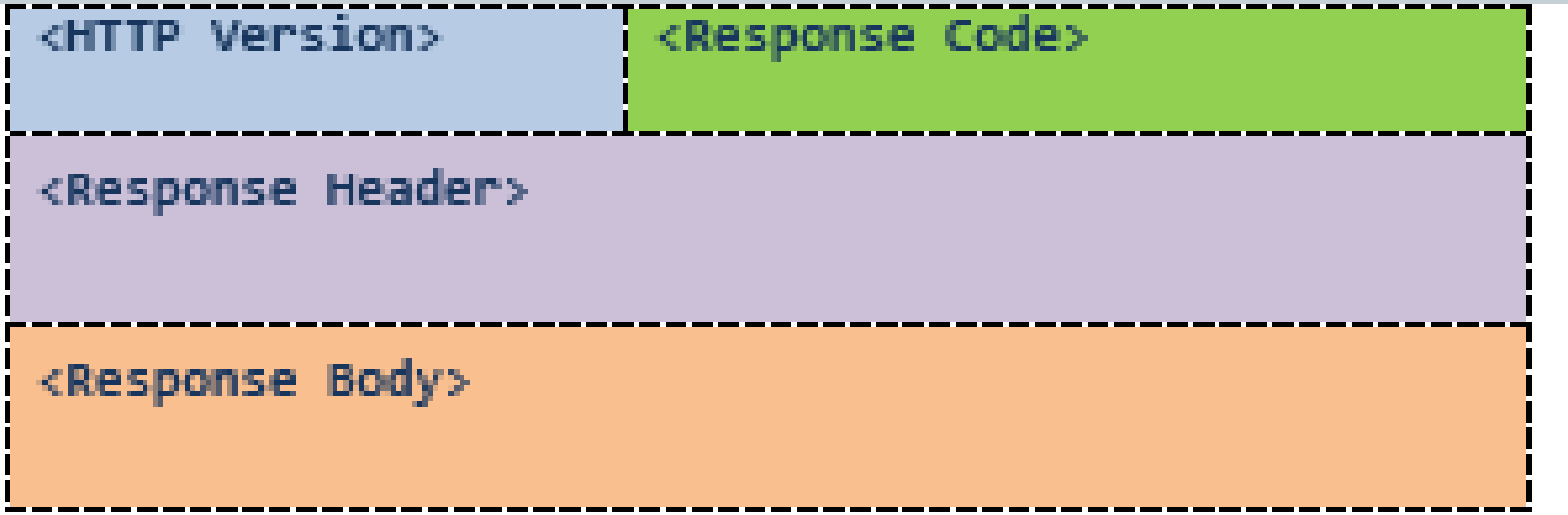
- <VERB> is one of the HTTP methods like GET, PUT, POST, DELETE, OPTIONS, etc
- <URI> is the URI of the resource on which the operation is going to be performed
- <HTTP Version> is the version of HTTP, generally "HTTP v1.1" .
- <Request Header> contains the metadata as a collection of key-value pairs of headers and their values.
- <Request Body> is the actual message content. In a RESTful service, that's where the representations of resources sit in a message.

HTTP Request

```
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>Jennifer Miller</Name>
  <Email>j.miller@gmail.com</Email>
  <Country>USA</Country>
</Person>
```

A sample POST request

HTTP Response



HTTP Response

```
HTTP/1.1 200 OK
Date: Sat, 23 Aug 2014 18:31:04 GMT
Server: Apache/2
Last-Modified: Wed, 01 Sep 2004 13:24:52 GMT
Accept-Ranges: bytes
Content-Length: 32859
Cache-Control: max-age=21600, must-revalidate
Expires: Sun, 24 Aug 2014 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>
...
```

An actual response to a GET request

Addressing Resources

- REST requires each resource to have at least one URI. A RESTful service uses a directory hierarchy like human readable URIs to address its resources.
- Suppose we have a database of persons and we wish to expose it to the outer world through a service. A resource person can be addressed like this:

  http://MyService/Persons/1

- This URL has following format:

Protocol://ServiceName/ResourceType/ResourceID

Addresssing Resources

Some important recommendations for well-structured URIs:

- Use plural nouns for naming your resources.
- Avoid using spaces as they create confusion
- A URI is case insensitive.
- You can have your own conventions, but stay consistent throughout the service.
- Avoid verbs for your resource names. Verbs are more suitable for the names of operations. For example, a RESTful service should not have the URIs http://MyService/FetcthPerson/1 or http://MyService/DeletePerson?id=1.

Query Parameters

- The basic purpose of query parameters is to provide parameters to an operation that needs the data items. For example, if you want the format of the presentation to be decided by the client. You can achieve that through a parameter like this:

http://MyService/Persons/1?format=xml&encoding=UTF8

or

http://MyService/Persons/1?format=json&encoding=UTF8

Methods

Method	Operation performed on server	Quality
`GET`	Read a resource.	Safe
`PUT`	Insert a new resource or update if the resource already exists.	Idempotent
`POST`	Insert a new resource. Also can be used to update an existing resource.	N/A
`DELETE`	Delete a resource .	Idempotent
`OPTIONS`	List the allowed operations on a resource.	Safe
`HEAD`	Return only the response headers and no response body.	Safe

Methods

- GET is probably the most popular method on the Web. It is used to fetch a resource.
- HEAD returns only the response headers with an empty body. This method can be used in a scenario when you do not need the entire representation of the resource. For example, HEAD can be used to quickly check whether a resource exists on the server or not.

Methods

- The method OPTIONS is used to get a list of allowed operations on the resource. Consider following request:

```
OPTIONS http://MyService/Persons/1 HTTP/1.1
HOST: MyService
```

- The service after authorizing and authenticating the request can return something like:

```
200 OK
Allow: HEAD, GET, PUT
```

- The second line contains the list of operations that are allowed for this client.

Methods

- You should use these methods only for the purpose for which they are intended. For instance, never use GET to create or delete a resource on the server.

- GET http://MyService/DeletePersons/1 HTTP/1.1
- HOST: MyService

- DELETE http://MyService/Persons/1 HTTP/1.1
- HOST: MyService

Difference between PUT and POST

Request	Operation
`PUT http://MyService/Persons/`	Won't work. `PUT` requires a complete URI
`PUT http://MyService/Persons/1`	Insert a new person with `PersonID=1` if it does not already exist, or else update the existing resource
`POST http://MyService/Persons/`	Insert a new person every time this request is made and generate a new `PersonID`.
`POST http://MyService/Persons/1`	Update the existing person where `PersonID=1`

Statelessness

- A RESTful service is stateless and does not maintain the application state for any client. A request cannot be dependent on a past request and a service treats each request independently.

Links between resources

- A resource representation can contain links to other resources like an HTML page contains links to other pages. The representations returned by the service should drive the process flow as in case of a website.

Links between resources

- Let's consider the case in which a client requests one resource that contains multiple other resources. Instead of dumping all these resources, you can list the resources and provide links to them. Links help keep the representations small in size.

Links between resources

For an example, if multiple Persons can be part of a Club, then a Club can be represented in MyService as follows:

```
<Club>
    <Name>Authors Club</Name>
    <Persons>
        <Person>
            <Name>M. Vaqqas</Name>
            <URI>http://MyService/Persons/1</URI>
        </Person>
        <Person>
            <Name>S. Allamaraju</Name>
            <URI>http://MyService/Persons/12</URI>
        </Person>
    </Persons>
</Club>
```

Caching

- Caching is the concept of storing the generated results and using the stored results instead of generating them repeatedly if the same request arrives in the near future. This can be done on the client, the server, or on any other component between them, such as a proxy server.

Caching

Caching can be controlled using HTTP headers:

Header	Application
`Date`	Date and time when this representation was generated.
`Last Modified`	Date and time when the server last modified this representation.
`Cache-Control`	The HTTP 1.1 header used to control caching.
`Expires`	Expiration date and time for this representation. To support HTTP 1.0 clients.
`Age`	Duration passed in seconds since this was fetched from the server. Can be inserted by an intermediary component.

Resources

- http://en.wikipedia.org/wiki/Representational_state_transfer
- http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069
- http://rest.elkstein.org/2008/02/what-is-rest.html
- http://www.ibm.com/developerworks/library/ws-restful/

SOAP vs. REST

- https://www.youtube.com/watch?v=v3OMEAU_4HI