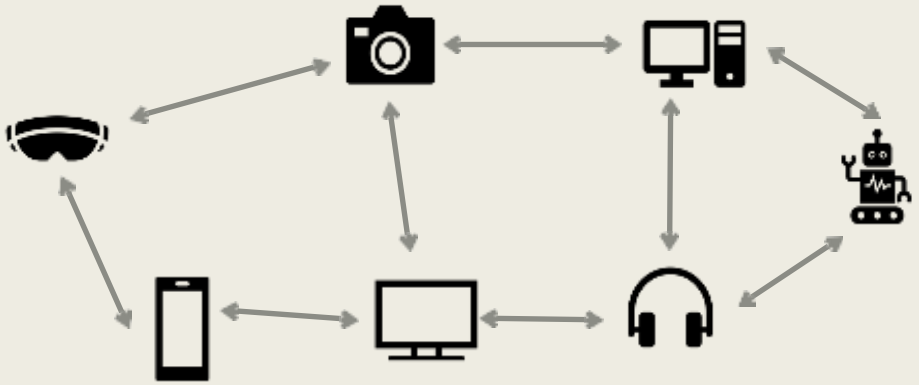
RESTFUL INTERFACES

Contents

APIS

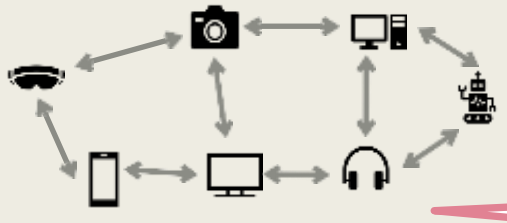
- *Application Programming Interface*
- *Abstract concept / general purpose term*
- *Allows us connect with computer devices all over the world using a variety of devices*

APIs allow people to	
Purchase	Goods & Services
Post	Blog, WeChat, Linkedin
Pick	Items from lists & dropdowns
Pin	Images and sounds to on-line profiles

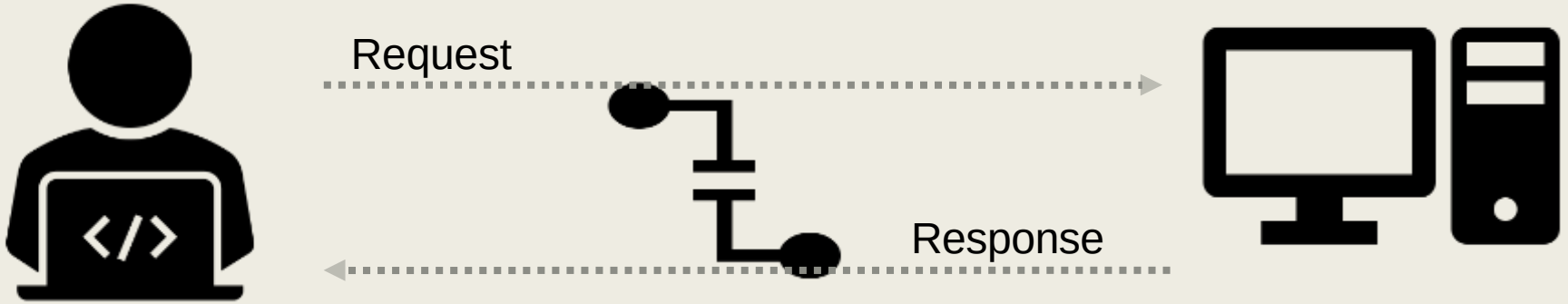


APIS

An API is
The engine that makes all this possible



Previous Slide



The messenger that
1. Takes the request that tells the system what you want to do
2. Returns the response back to you

APIS

An API is a waiter

1. the customer – the client/the device
2. the waiter – the API
3. the chef – the service / the system
4. the request – the order
5. the response – the meal

APIS

Consider accessing a car insurance company's web site

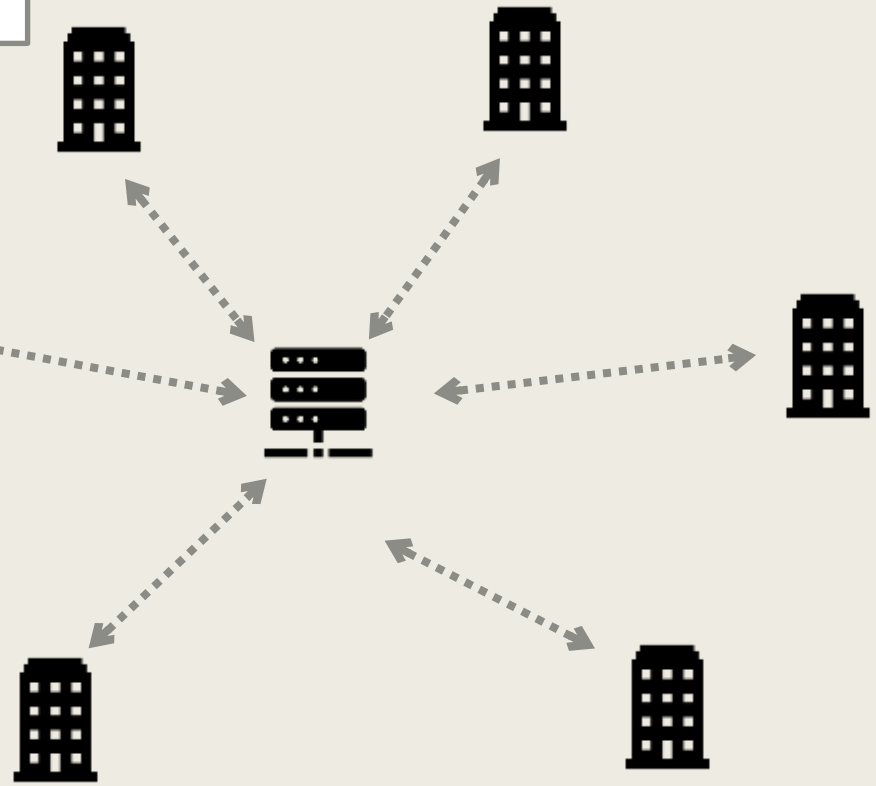
Search on-line, complete a form with variables such as

- Personal details
- Make & model of Car
- From / to
- Value of car

This is done through the website's HTML/JS

APIS

But if car dealer websites expose their data through API…



- Search on-line
- Use a price comparison service rather than going direct to a single company's web site

APIS

Separate Applications

Different types of customers

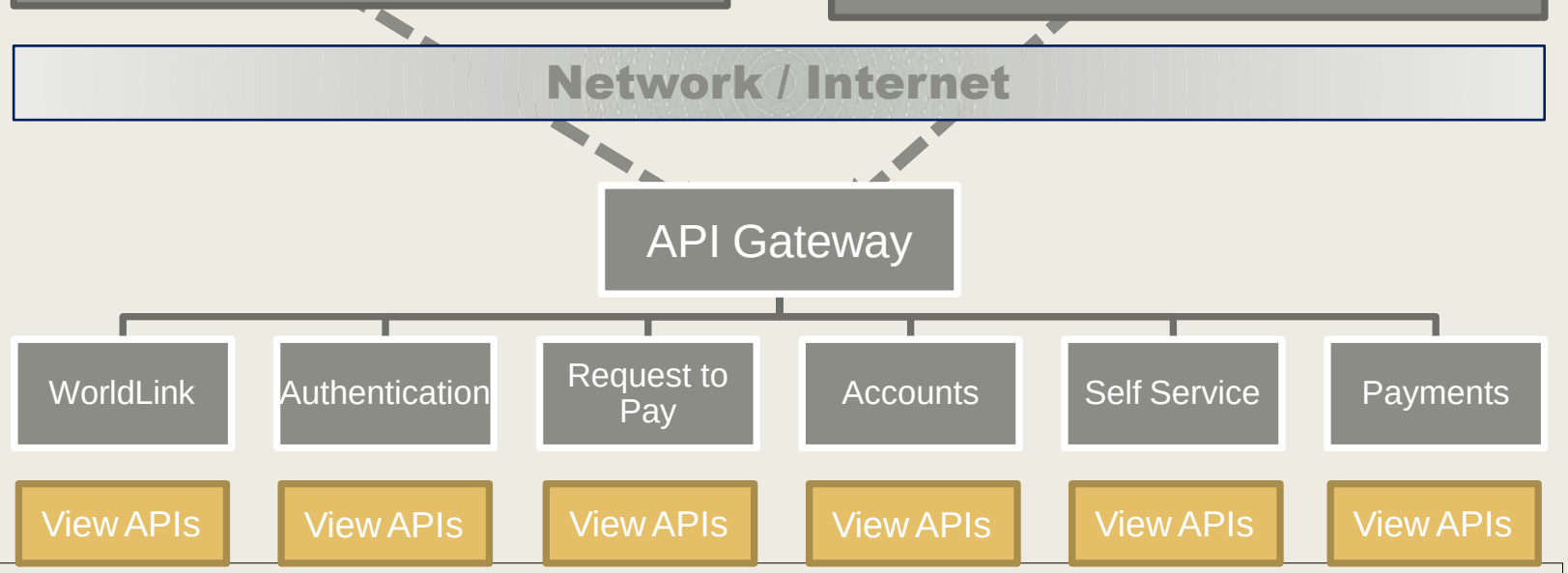
Each decide which API(s) they wish to use

BankOfABC
- Authentication
- Self Service
- Accounts
- Payments

XYZ Holdings
- Authentication
- Self Service
- World Link
- RTP

Payments APIs

Enhanced Payment Status Inquiry
Payment Status Inquiry
Payment Initiation
…



Restful Web Services

The world before REST

1999 - The API landscape was a free-for-all.

- *Few standards, emerging and competing technologies*
- *CORBA, DCOM, SOAP*

A simple SOAP request looked like

```
POST http://soapl.develop.com/cgi-bin/ServerDemo.pl?class=Geometry
HTTP/1.0 SOAPMethodName: http://soapl.develop.com/cgi-bin/ServerDemo.pl?class=Geometry#calculateArea
Host: soapl.develop.com (http://soapl.develop.com/)
Content-Type: text/xml Content-Length: 494
```

SOAP was difficult to build, debug and maintain

as were CORBA and DCOM

REST

2000 – Roy Fielding - Phd Dissertation

Uniform interface.

- *always use HTTP verbs (GET, PUT, POST, DELETE).*
- *always use URIs as resources.*
- *always get an HTTP response with a status and a body.*

Stateless.

- *each request is self-descriptive, and has enough context for the server to process that message.*

Client-server.

- *clear boundaries between roles of the two systems.*
- *server, operationally, has to function as the server that is being called,*
- *Client has to function as the one making the requests.*

Cacheable.

- *unless denoted, a client can cache any representation.*
- *possible thanks to the statelessness—every representation is self-descriptive.*

REST

- **RESTful** Web Services are basically REST Architecture based Web Services.
- In REST Architecture everything is a resource.
- RESTful web services are light weight, highly scalable and maintainable and are very commonly used to create APIs for web-based applications.
- We will look at the basics of RESTful Web Services and contains chapters discussing all the basic components of RESTful Web Services with suitable examples.

Introduction to REST

REST stands for **RE**presentational **S**tate **T**ransfer.

- *REST is web standards based architecture and uses HTTP Protocol.*
- *It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods.*
- *REST was first introduced by Roy Fielding in 2000.*

In REST architecture, a REST Server simply provides access to resources and REST client accesses and modifies the resources.

Here each resource is identified by URIs/ global IDs.

REST uses various representation to represent a resource like text, JSON, XML.

JSON is the most popular one.

HTTP Methods

HTTP methods

Following four HTTP methods are commonly used in REST based architecture.

- ***GET*** *− Provides a read only access to a resource.*
- ***POST*** *− Used to create a new resource.*
- ***DELETE*** *− Used to remove a resource.*
- ***PUT*** *− Used to update a existing resource or create a new resource.*

No definitive list but most are

OPTIONS GET HEAD POST PUT DELETE TRACE CONNECT PROPFIND PROPPATCH MKCOL COPY MOVE
LOCK UNLOCK VERSION-CONTROL REPORT CHECKOUT CHECKIN UNCHECKOUT MKWORKSPACE
UPDATE LABEL MERGE BASELINE-CONTRO LMKACTIVITY

Web Services

A Web service is a collection of

– *open protocols and standards used for exchanging data between applications or systems.*

Applications use web services to exchange data over computer networks

– *written in various programming languages*

– *running on various platforms*

– *interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.*

RESTful web services

– *Web services based on REST Architecture.*

– *Use HTTP methods to implement the concept of REST architecture.*

RESTful web services usually

– *defines a URI (Uniform Resource Identifier) as a service,*

– *provides resource representation such as JSON*

– *defines set of HTTP Methods.*

Restful Web Services

Creating a RESTful web service

A user management with following functionalities

Sr.No.	URI	HTTP Method	POST body	Result
1	/UserService/users	GET	empty	Show list of all the users.
2	/UserService/addUser	POST	JSON String	Add details of new user.
3	/UserService/getUser/:id	GET	empty	Show details of a user.

Resources

REST architecture treats all content as resources.

These resources can be Text Files, Html Pages, Images, Videos or Dynamic Business Data.

REST Server simply provides access to resources and REST client accesses and modifies the resources.

Each resource is identified by URIs/ Global IDs.

Representation of Resources

The focus of a RESTful service is on resources and how to provide access to them

REST does not put a restriction on the format.

XML and JSON are 2 common data formats

XML – eXtensible Markup Language

focuses on data rather than how it looks

JSON – Java Script Object Notation

text-based data format following JavaScript object syntax

XML – eXtensible Markup Language

focuses on data rather than how it looks

```
<Person>
    <Name>John Doe</Name>
    <SOEID>jd12345</SOEID>
</Person>
```

JSON – Java Script Object Notation

text-based data format following JavaScript object syntax

```
{
  "Name": "John Doe",
  "SOEID": "jd12345"
}
```

Good representation of resources

REST does not impose any restriction on the format of a resource representation.

The REST server may be flexible enough to pass the client the resource in the format that the client understands.

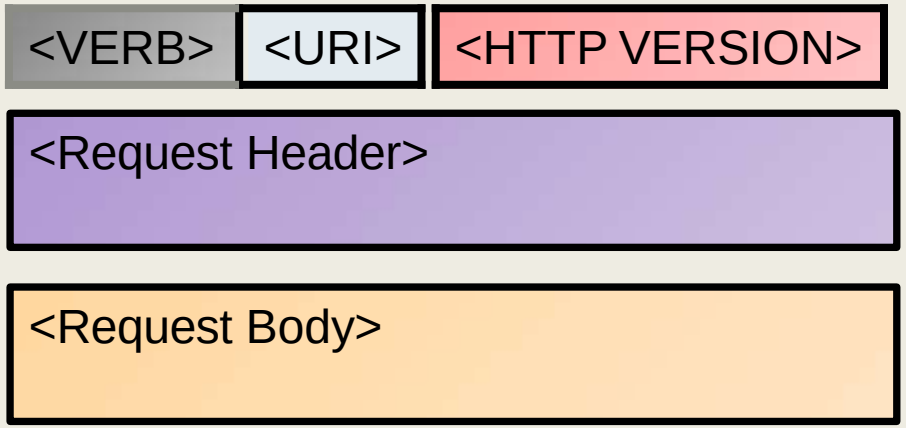
Some important points to be considered while designing a representation format of a resource in RESTful Web Services.

- ***Understandability*** − *Both the Server and the Client should be able to understand and utilize the representation format of the resource.*
- ***Completeness*** − *Format should be able to represent a resource completely. For example, a resource can contain another resource. Format should be able to represent simple as well as complex structures of resources.*
- ***Linkablity*** − *A resource can have a linkage to another resource, a format should be able to handle such situations.*

HTTP Request

An HTTP Request has five major parts −

- ***Verb*** − *Indicates the HTTP methods such as GET, POST, DELETE, PUT, etc.*
- ***URI*** − *Uniform Resource Identifier (URI) to identify the resource on the server.*
- ***HTTP Version*** − *Indicates the HTTP version. For example, HTTP v1.1.*
- ***Request Header*** − *Contains metadata for the HTTP Request message as key-value pairs. For example, client (or browser) type, format supported by the client, format of the message body, cache settings, etc.*
- ***Request Body*** − *Message content or Resource representation.*

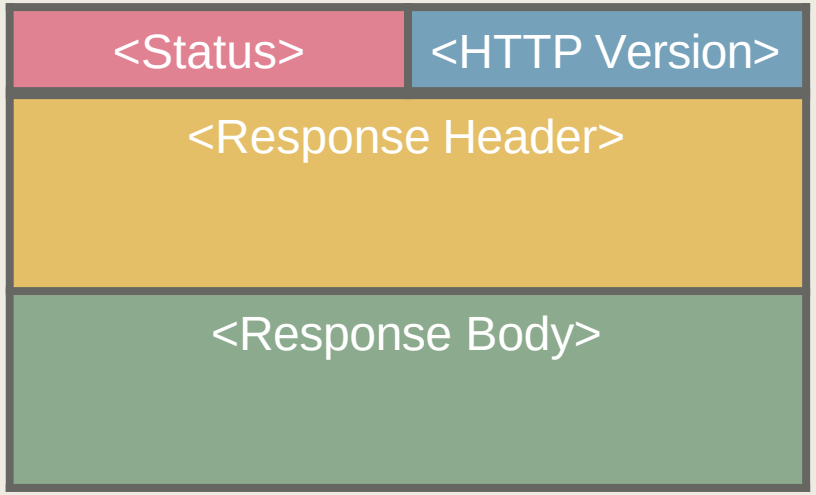


HTTP Request Format

HTTP Response

An HTTP Response has four major parts −

- ***Status/Response Code*** − *Indicates the Server status for the requested resource. For example, 404 means resource not found and 200 means response is ok.*
- ***HTTP Version*** − *Indicates the HTTP version. For example HTTP v1.1.*
- ***Response Header*** − *Contains metadata for the HTTP Response message as keyvalue pairs. For example, content length, content type, response date, server type, etc.*
- ***Response Body*** − *Response message content or Resource representation.*



HTTP Response Format

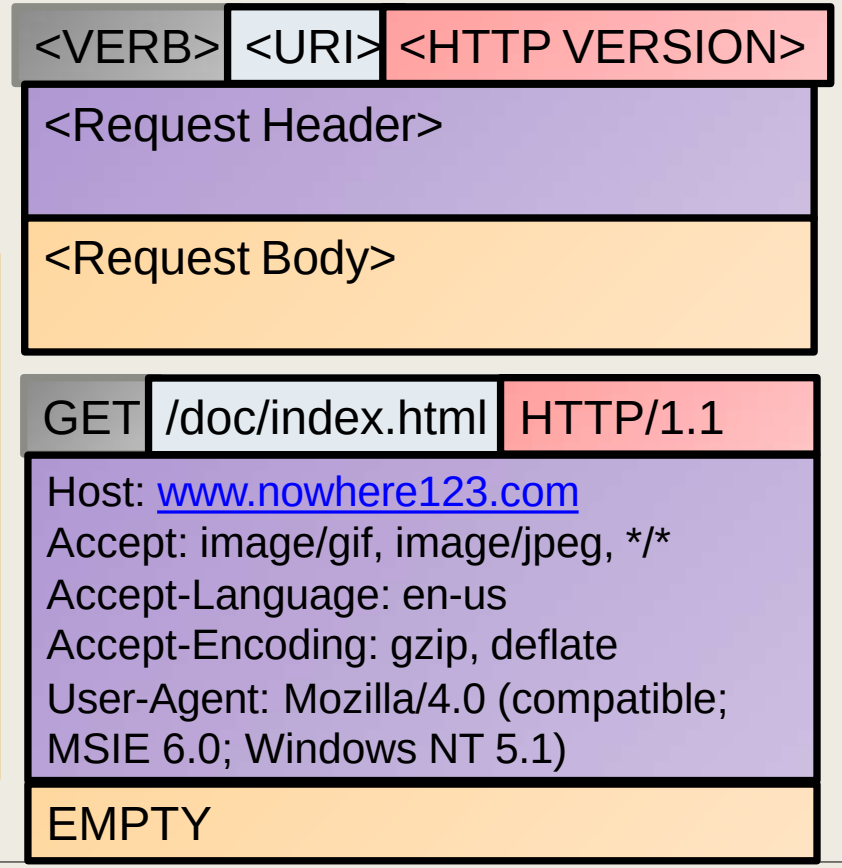
Example GET

- a Browser will translate this address

  http://www.nowhere123.com/doc/index.html

- into this request

```
GET /doc/index.html HTTP/1.1
Host: www.nowhere123.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1)
```



Example POST request

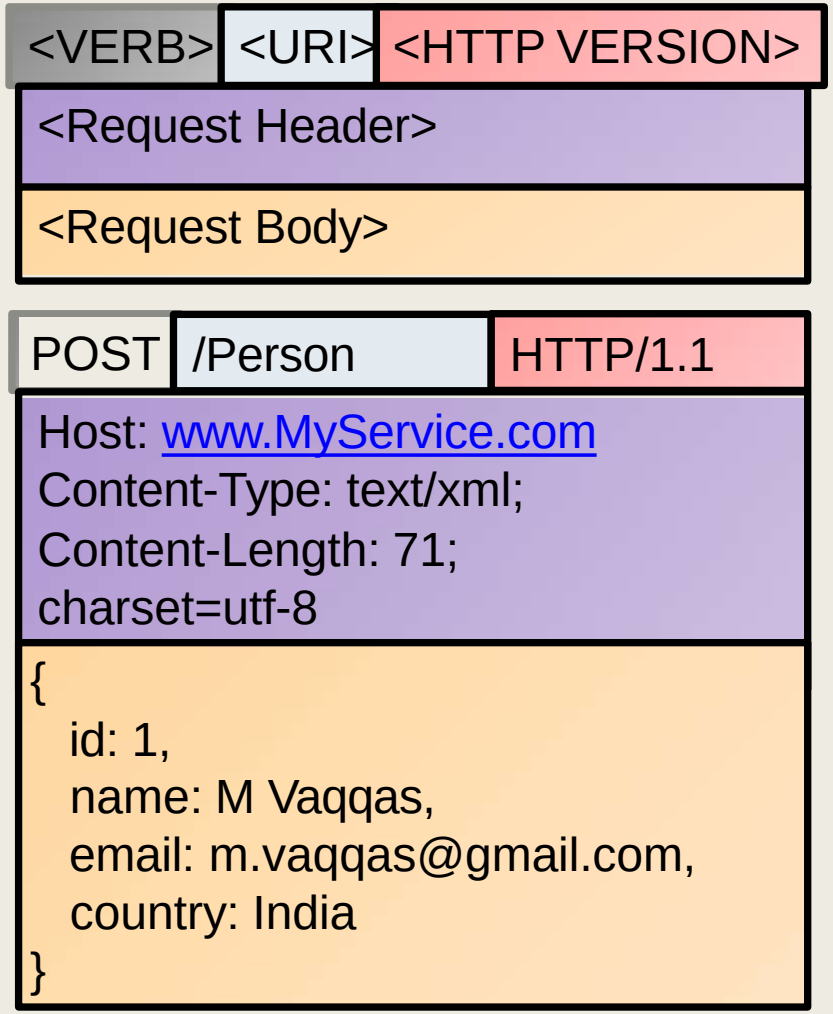
```
POST /person HTTP/1.1
Host: www.myservice.com
Content-Type: application/json; charset=utf-8
Content-Length: 71
{
  id: 1,
  name: M Vaqqas,
  email: m.vaqqas@gmail.com,
  country: India
}
```

- The **POST** command is followed by the URI and the HTTP version.
- May also contains some request headers. Host is the address of the server.
- Content-Type tells about the type of contents in the message body.
- Content-Length is the length of the data in message body.
- Content-Length can be used to verify that the entire message body has been received.

For JSON use Content-Type application/json
There are literally hundreds of content types
http://www.iana.org/assignments/media-types/media-types.xhtml

Example POST request

```
POST /person HTTP/1.1
Host: www.myservice.com
Content-Type: application/json; charset=utf-8
Content-Length: 71
{
  id: 1,
  name: M Vaqqas,
  email: m.vaqqas@gmail.com,
  country: India
}
```



Addressing

Refers to locating a resource or multiple resources lying on the server.

Analogous to locate a postal address of a person.

Each resource in a REST architecture is identified by its URI (Uniform Resource Identifier). A URI is of the following format −

```
<protocol>://<service-name>/<ResourceType>/<ResourceID>
```

Purpose of an URI is to locate a resource(s) on the server hosting the web service.

Another important attribute of a request is VERB which identifies the operation to be performed on the resource.

For example

http://localhost:8080/UserManagement/rest/UserService/users and the **VERB** is **GET**.

Constructing a URI

Use Plural Noun

– *Use plural noun to define resources. For example, we've used users to identify users as a resource.*

Avoid using spaces

– *Use underscore (_) or hyphen (-) when using a long resource name. For example, use authorized_users instead of authorized%20users.*

Use lowercase letters

– *Although URI is case-insensitive, it is a good practice to keep the url in lower case letters only.*

Maintain Backward Compatibility

– *As Web Service is a public service, a URI once made public should always be available. In case, URI gets updated, redirect the older URI to a new URI using the HTTP Status code, 300.*

Use of the HTTP Verb

– *Always use HTTP Verb like GET, PUT and DELETE to do the operations on the resource. It is not good to use operations name in the URI.*

Constructing a URI

Is this good ?

http://localhost:8080/UserManagement/rest/UserService/getUser/1

How about this one ?

http://localhost:8080/UserManagement/rest/UserService/users/1

Web Service Methods - GET

HTTP Method	GET
URI	http://localhost:8080/UserManagement/rest/UserService/users
Operation	Get list of users
Operation Type	Read Only

HTTP Method	GET
URI	http://localhost:8080/UserManagement/rest/UserService/users/1
Operation	Get user of Id 1
Operation Type	Read Only

Web Service Methods - POST

HTTP Method	POST
URI	http://localhost:8080/UserManagement/rest/UserService/users/
Operation	Insert a new user
Operation Type	Non-Idempotent

Web Service Methods - PUT

HTTP Method	PUT
URI	http://localhost:8080/UserManagement/rest/UserService/users/2
Operation	Update User with Id 2
Operation Type	Idempotent

Web Service Methods - DELETE

HTTP Method	DELETE
URI	http://localhost:8080/UserManagement/rest/UserService/users/1
Operation	Delete User with Id 1
Operation Type	Idempotent

Web Service Methods - OPTIONS

HTTP Method	OPTIONS
URI	http://localhost:8080/UserManagement/rest/UserService/users
Operation	List the supported operations in web service
Operation Type	Read Only

Web Service Methods - HEAD

HTTP Method	HEAD
URI	http://localhost:8080/UserManagement/rest/UserService/users
Operation	Returns only HTTP Header, no Body
Operation Type	Read Only

For consideration

GET operations are read only and are safe.

PUT and **DELETE** operations are idempotent means their result will always same no matter how many times these operations are invoked.

PUT and **POST** operation are nearly same with the difference lying only in the result where **PUT** operation is idempotent and **POST** operation can cause different result.

Statelessness

RESTful Web Service should **not** keep a client state on the server.

- *called Statelessness.*
- *the responsibility of the client to pass its context to the server*
- *the server can store this context to process the client's further request*
- *e.g., session maintained by server is identified by session identifier passed by the client.*

RESTful Web Services should adhere to this restriction.

Stateless – Pros and Cons

- *Web services can treat each method request independently.*
- *Web services need not maintain the client's previous interactions. It simplifies the application design.*
- *As HTTP is itself a statelessness protocol, RESTful Web Services work seamlessly with the HTTP protocols.*

- *Web services need to get extra information in each request and then interpret to get the client's state so the client interactions can be properly processed.*

Caching

Caching refers to storing the server response in the client itself.

Client need not make a server request for the same resource again and again.

A server response should have information about how caching is to be done.

Client caches the response for a time-period or never caches the server response.

Caching

Following are the headers which a server response can have in order to configure a client's caching –

No.	Header & Description
1	**Date** Date and Time of the resource when it was created.
2	**Last Modified** Date and Time of the resource when it was last modified.
3	**Cache-Control** Primary header to control caching.
4	**Expires** Expiration date and time of caching.
5	**Age** Duration in seconds from when resource was fetched from the server.

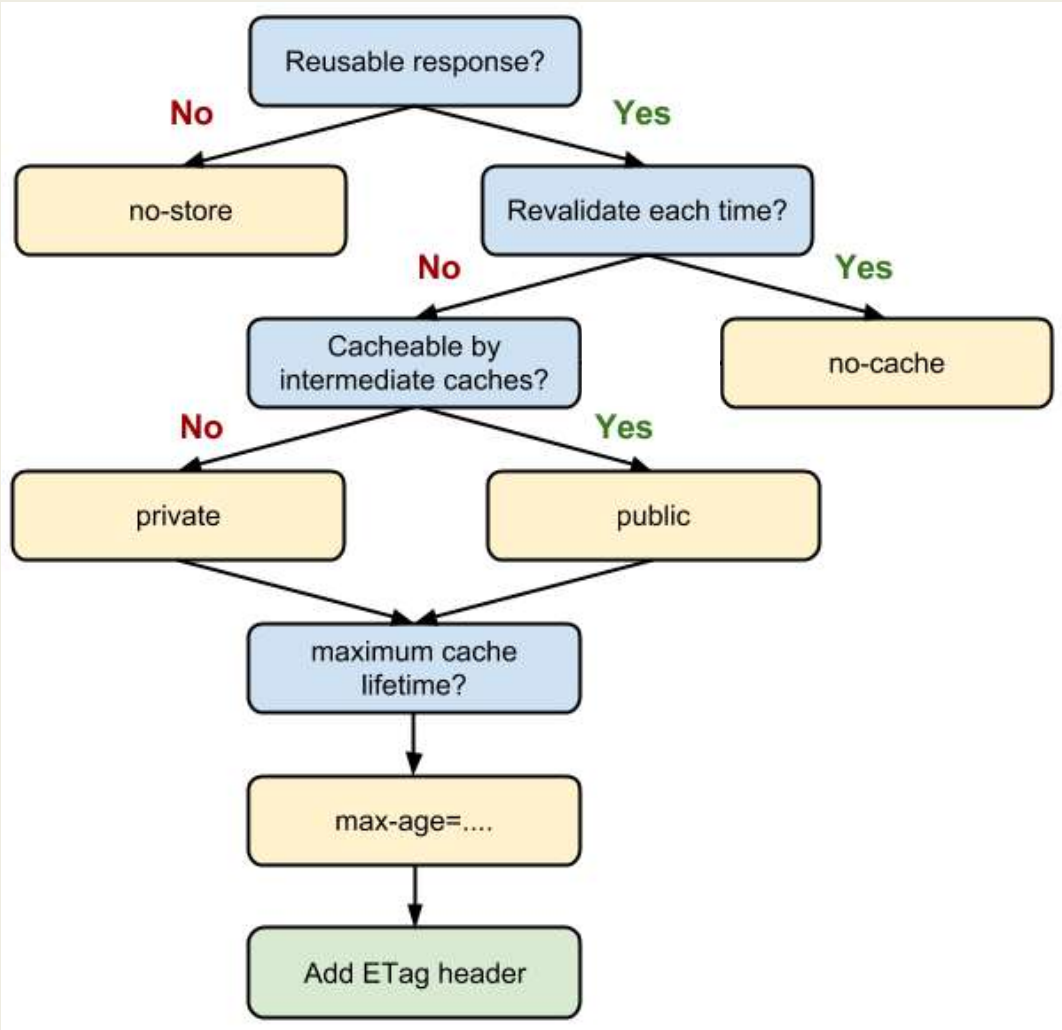
Cache-Control Header

No.	Directive & Description
1	**Public** Indicates that resource is cacheable by any component.
2	**Private** Indicates that resource is cacheable only by the client and the server, no intermediary can cache the resource.
3	**no-cache/no-store** Indicates that a resource is not cacheable.
4	**max-age** Indicates the caching is valid up to max-age in seconds. After this, client has to make another request.
5	**must-revalidate** Indication to server to revalidate resource if max-age has passed.

Contents

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

- *Never keep expiry date too high.*
- *Dynamic content should be cached for a few hours only.*



Security

As RESTful Web Services work with HTTP URL Paths, it is very important to safeguard a RESTful Web Service in the same manner as a website is secured.

Following are the best practices to be adhered to while designing a RESTful Web Service −

Validation

- Validate all inputs on the server. Protect your server against SQL or NoSQL injection attacks.

Session Based Authentication

- Use session based authentication to authenticate a user whenever a request is made to a Web Service method.

No Sensitive Data in the URL

- Never use username, password or session token in a URL, these values should be passed to Web Service via the POST method.

Security

As RESTful Web Services work with HTTP URL Paths, it is very important to safeguard a RESTful Web Service in the same manner as a website is secured.

Following are the best practices to be adhered to while designing a RESTful Web Service −

Restriction on Method Execution

- Allow restricted use of methods like GET, POST and DELETE methods. The GET method should not be able to delete data.

Validate Malformed XML/JSON

- Check for well-formed input passed to a web service method.

Throw generic Error Messages

- A web service method should use HTTP error messages like 403 to show access forbidden, etc.

Response Codes

No.	HTTP Code & Description
1	**200** **OK** − shows success.
2	**201** **CREATED** − when a resource is successfully created using POST or PUT request. Returns link to the newly created resource using the location header.
3	**204** **NO CONTENT** − when response body is empty. For example, a DELETE request.
4	**304** **NOT MODIFIED** − used to reduce network bandwidth usage in case of conditional GET requests. Response body should be empty. Headers should have date, location, etc.
5	**400** **BAD REQUEST** − states that an invalid input is provided. For example, validation error, missing data.

Response Codes

No.	HTTP Code & Description
6	**401** **UNAUTHORIZED** − states that user is using invalid or wrong authentication token.
7	**403** **FORBIDDEN** − states that the user is not having access to the method being used. For example, Delete access without admin rights.
8	**404** **NOT FOUND** − states that the method is not available.
9	**409** **CONFLICT** − states conflict situation while executing the method. For example, adding duplicate entry.
10	**500** **INTERNAL SERVER ERROR** − states that the server has thrown some exception while executing the method.

Tools for RESTful Web Services

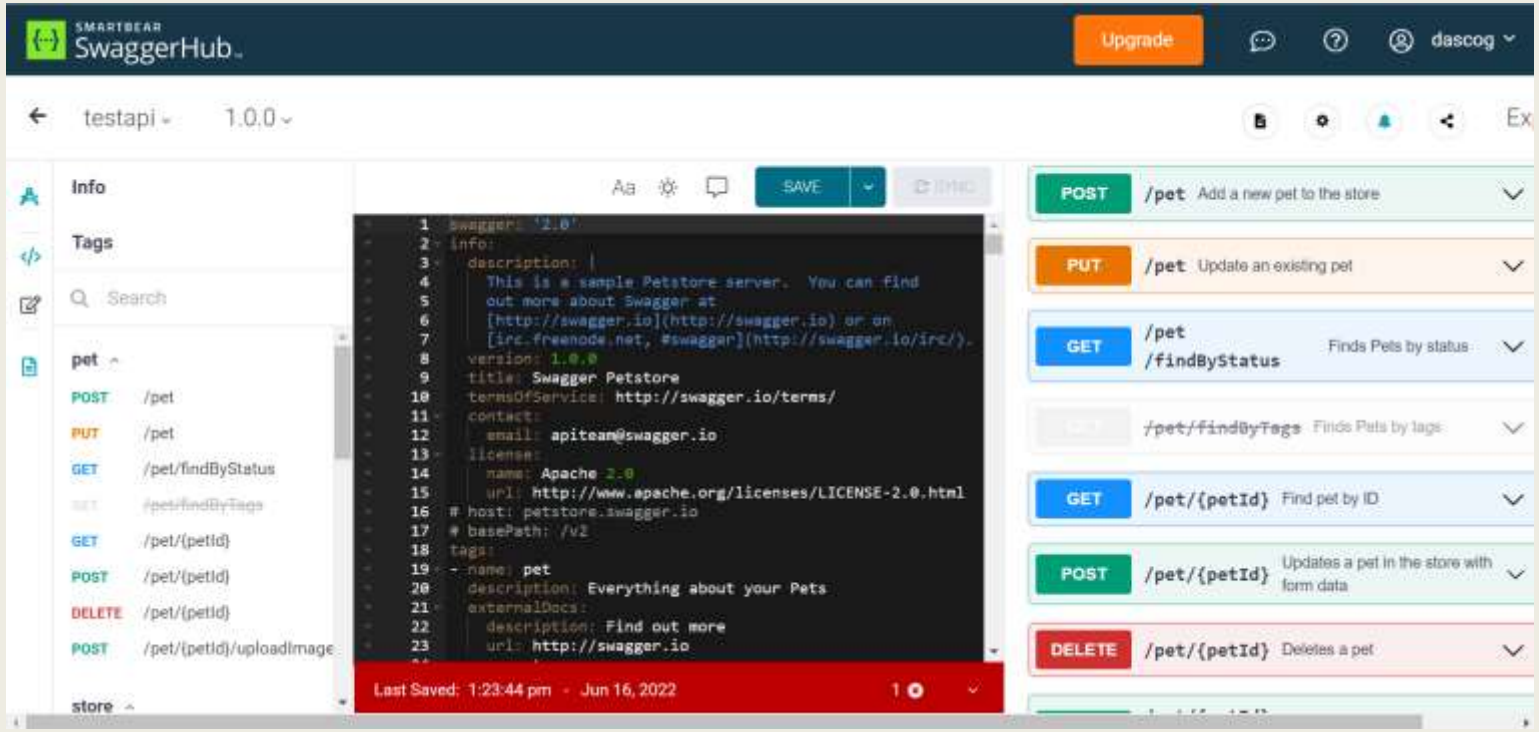
There are lots of tools available to help you define, test and deploy RESTful web services:

- Postman: an API platform for building and using APIs. Supports REST, gRPC, etc
- Insomnia: a competitor to Postman by Kong. Support for multiple protocols, testing etc.
- Swagger and SwaggerHub: Another suite of tools around API development, testing and sharing. We will look at SwaggerHub in more detail.

SwaggerHub

SwaggerHub is an integrated API design and documentation platform

You can use it to design and test you API before you write any backend code



Restful Web Services

Lab: SwaggerHub

In this lab you will sign up for a SwaggerHub account and do a simple getting started tutorial

Questions