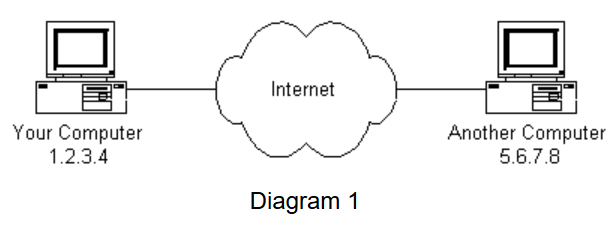
<https://web.stanford.edu/class/msande91si/www-spr04/readings/week1/InternetWhitepaper.htm>

How internet Works?

the Internet is a global network of computers each computer connected to the Internet **must** have a unique address. Internet addresses are in the form **nnn.nnn.nnn.nnn** where nnn must be a number from 0 - 255. This address is known as an IP address. (IP stands for Internet Protocol.



If you connect to the Internet through an Internet Service Provider (ISP), you are usually assigned a temporary IP address for the duration of your dial-in session. If you connect to the Internet from a local area network (LAN) your computer might have a permanent IP address or it might obtain a temporary one from a DHCP (Dynamic Host Configuration Protocol) server. In any case, if you are connected to the Internet, your computer has a unique IP address.

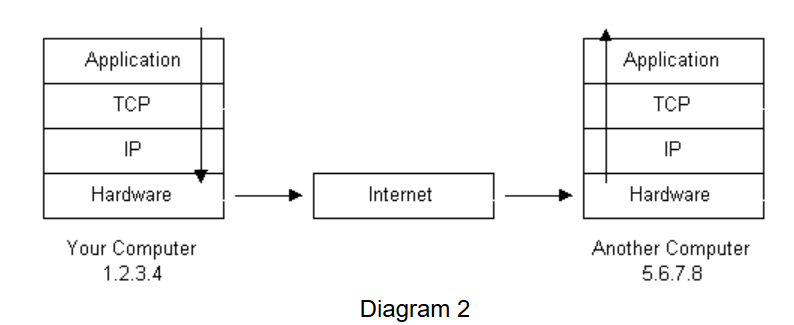
How does it 'talk' to other computers connected to the Internet?

An example should serve here: Let's say your IP address is 1.2.3.4 and you want to send a message to the computer 5.6.7.8. The message you want to send is "Hello computer 5.6.7.8!". Obviously, the message must be transmitted over whatever kind of wire connects your computer to the Internet. Let's say you've dialed into your ISP from home and the message must be transmitted over the phone line. Therefore the message must be translated from alphabetic text into electronic signals, transmitted over the Internet, then translated back into alphabetic text.

How is this accomplished?

Through the use of a **protocol stack**. Every computer needs one to communicate on the Internet and it is usually built into the computer's operating system (i.e. Windows, Unix, etc.). The protocol stack used on the Internet is refered to as the TCP/IP protocol stack because of the two major communication protocols used.

|  |  |
| --- | --- |
| **Protocol Layer** | **Comments** |
| Application Protocols Layer | Protocols specific to applications such as WWW, e-mail, FTP, etc. |
| Transmission Control Protocol Layer | TCP directs packets to a specific application on a computer using a port number. |
| Internet Protocol Layer | IP directs packets to a specific computer using an IP address. |
| Hardware Layer | Converts binary packet data to network signals and back. (E.g. ethernet network card, modem for phone lines, etc.) |



1. The message would start at the top of the protocol stack on your computer and work it's way downward.
2. If the message to be sent is long, each stack layer that the message passes through may break the message up into smaller chunks of data. This is because data sent over the Internet (and most computer networks) are sent in manageable chunks. On the Internet, these chunks of data are known as **packets**.
3. The packets would go through the Application Layer and continue to the TCP layer. Each packet is assigned a **port number**. Ports will be explained later, but suffice to say that many programs may be using the TCP/IP stack and sending messages. We need to know which program on the destination computer needs to receive the message because it will be listening on a specific port.
4. After going through the TCP layer, the packets proceed to the IP layer. This is where each packet receives it's destination address, 5.6.7.8.
5. Now that our message packets have a port number and an IP address, they are ready to be sent over the Internet. The hardware layer takes care of turning our packets containing the alphabetic text of our message into electronic signals and transmitting them over the phone line.
6. On the other end of the phone line your ISP has a direct connection to the Internet. The ISPs **router** examines the destination address in each packet and determines where to send it. Often, the packet's next stop is another router. More on routers and Internet infrastructure later.
7. Eventually, the packets reach computer 5.6.7.8. Here, the packets start at the bottom of the destination computer's TCP/IP stack and work upwards.
8. As the packets go upwards through the stack, all routing data that the sending computer's stack added (such as IP address and port number) is stripped from the packets.
9. When the data reaches the top of the stack, the packets have been re-assembled into their original form, "Hello computer 5.6.7.8!"

H:\>ping www.google.com

Pinging www.google.com [64.233.177.99] with 32 bytes of data:

Reply from 64.233.177.99: bytes=32 time=217ms TTL=40

Reply from 64.233.177.99: bytes=32 time=217ms TTL=40

Traceroute will print out a list of all the routers, computers, and any other Internet entities that your packets must travel through to get to their destination.

H:\>tracert www.google.com

Tracing route to www.google.com [64.233.177.99]

over a maximum of 30 hops:

1 1 ms \* 5 ms 135.52.136.3

2 4 ms \* 3 ms 135.75.231.122

3 1164 ms 219 ms 219 ms 135.37.15.10

4 452 ms 217 ms 217 ms gdgaaxx005-gi2-2-2.gnoc.att.com [135.37.15.9]

5 216 ms 216 ms 216 ms alprgaedcrain01.pmtr.sest.att.com [135.28.160.40]

6 452 ms 217 ms 217 ms 135.28.172.126

7 218 ms 218 ms 218 ms gaalxbm05a-intdef\_1-inetpmtrvirt.pst.cso.att.com [144.160.226.53]

8 218 ms 218 ms 219 ms gaalxbm05a-intdef\_1-inetpmtrvirt.pst.cso.att.com [144.160.226.53]

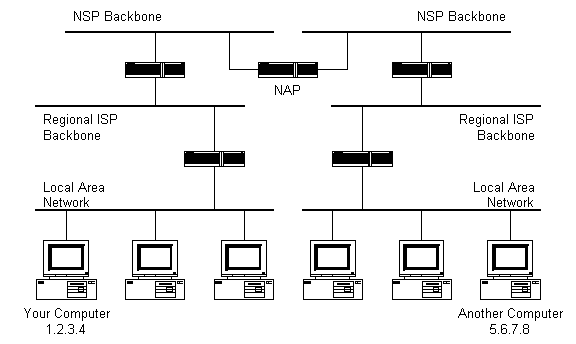
9 225 ms 221 ms 219 ms gaalxbm05a-intdef\_1-inetpmtrvirt.pst.cso.att.com [144.160.226.53]

10 219 ms 218 ms 218 ms gaalxbm05a-intdef\_1-inetpmtrvirt.pst.cso.att.com [144.160.226.53]

NSP(Network Service Provider);

he Internet backbone is made up of many large networks which interconnect with each other. These large networks are known as **Network Service Providers** or **NSP**s.

Some of the large NSPs are UUNet, CerfNet, IBM, BBN Planet, SprintNet, PSINet, as well as others. These networks **peer** with each other to exchange packet traffic. Each NSP is required to connect to three **Network Access Points** or **NAP**s. NAPs are connected to **Metropolitan Area Exchanges** or **MAE**s



how do packets find their way across the Internet?

Does every computer connected to the Internet know where the other computers are?

Do packets simply get 'broadcast' to every computer on the Internet?

The answer to both the preceeding questions is 'no'. No computer knows where any of the other computers are, and packets do not get sent to every computer.

The information used to get packets to their destinations are contained in routing tables kept by each router connected to the Internet.

A **router** is usually connected between networks to route packets between them. Each router knows about it's sub-networks and which IP addresses they use.

When a packet arrives at a router, the router examines the IP address put there by the IP protocol layer on the originating computer. The router checks it's routing table. If the network containing the IP address is found, the packet is sent to that network. If the network containing the IP address is not found, then the router sends the packet on a default route, usually up the backbone hierarchy to the next router. Hopefully the next router will know where to send the packet. If it does not, again the packet is routed upwards until it reaches a NSP backbone. The routers connected to the NSP backbones hold the largest routing tables and here the packet will be routed to the correct backbone, where it will begin its journey 'downward' through smaller and smaller networks until it finds it's destination.

**Domain Names and Address Resolution**

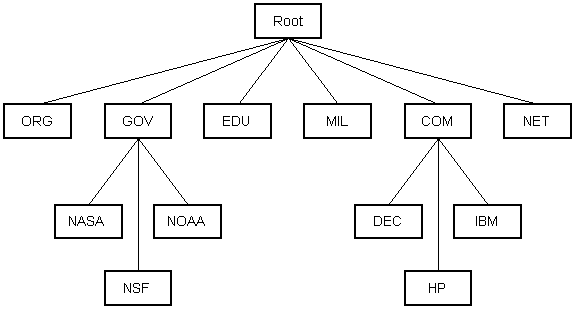
what if you don't know the IP address of the computer you want to connect to?

What if the you need to access a web server referred to as *www.anothercomputer.com*?

How does your web browser know where on the Internet this computer lives?

The answer to all these questions is the **Domain Name Service** or **DNS**. The DNS is a distributed database which keeps track of computer's names and their corresponding IP addresses on the Internet.

No DNS server contains the entire database; they only contain a subset of it. If a DNS server does not contain the domain name requested by another computer, the DNS server re-directs the requesting computer to another DNS server.



These include the TCP and IP protocols, routing protocols, medium access control protocols, application level protocols, etc. The following sections describe some of the more important and commonly used protocols on the Internet. Higher level protocols are discussed first, followed by lower level protocols.

**Application Protocols: HTTP and the World Wide Web**

One of the most commonly used services on the Internet is the World Wide Web (WWW). The application protocol that makes the web work is **Hypertext Transfer Protocol** or **HTTP**.

HTTP is the protocol that web browsers and web servers use to communicate with each other over the Internet.

A **web server** is a computer that stores **web server** software and a website's component files (e.g. HTML documents, images, CSS stylesheets, and JavaScript files). It is connected to the Internet and supports physical data interchange with other devices connected to the **web**.

HTTP is a connectionless text based protocol. Clients (web browsers) send requests to web servers for web elements such as web pages and images. After the request is serviced by a server, the connection between client and server across the Internet is disconnected. A new connection must be made for each request. Most protocols are connection oriented.

When you type a URL into a web browser, this is what happens:

1. If the URL contains a domain name, the browser first connects to a domain name server and retrieves the corresponding IP address for the web server.
2. The web browser connects to the web server and sends an HTTP request (via the protocol stack) for the desired web page.
3. The web server receives the request and checks for the desired page. If the page exists, the web server sends it. If the server cannot find the requested page, it will send an HTTP 404 error message. (404 means 'Page Not Found' as anyone who has surfed the web probably knows.)
4. The web browser receives the page back and the connection is closed.
5. The browser then parses through the page and looks for other page elements it needs to complete the web page. These usually include images, applets, etc.
6. For each element needed, the browser makes additional connections and HTTP requests to the server for each element.
7. When the browser has finished loading all images, applets, etc. the page will be completely loaded in the browser window.

## Application Protocols: SMTP and Electronic Mail

E-mail uses an application level protocol called **Simple Mail Transfer Protocol** or **SMTP**. SMTP is also a text based protocol, but unlike HTTP, SMTP is connection oriented.

When you open your mail client to read your e-mail, this is what typically happens:

1. The mail client (Netscape Mail, Lotus Notes, Microsoft Outlook, etc.) opens a connection to it's default mail server. The mail server's IP address or domain name is typically setup when the mail client is installed.
2. The mail server will always transmit the first message to identify itself.
3. The client will send an SMTP HELO command to which the server will respond with a 250 OK message.
4. Depending on whether the client is checking mail, sending mail, etc. the appropriate SMTP commands will be sent to the server, which will respond accordingly.
5. This request/response transaction will continue until the client sends an SMTP QUIT command. The server will then say goodbye and the connection will be closed.

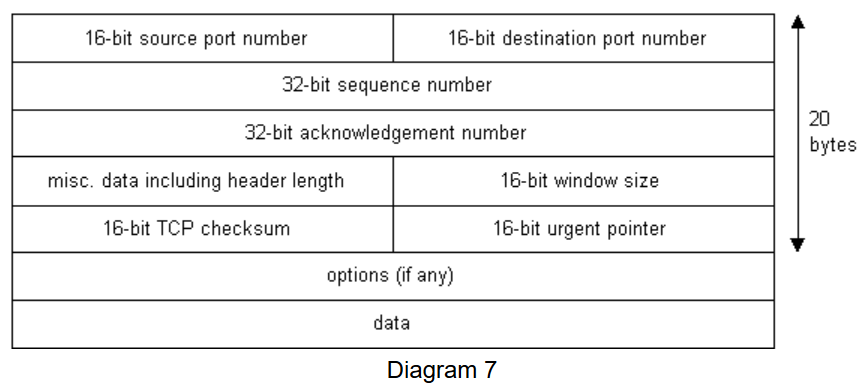
## Transmission Control Protocol

**TCP is responsible for routing application protocols to the correct application on the destination computer**.

TCP works like this:

* When the TCP layer receives the application layer protocol data from above, it segments it into manageable 'chunks' and then adds a TCP header with specific TCP information to each 'chunk'. The information contained in the TCP header includes the port number of the application the data needs to be sent to.
* When the TCP layer receives a packet from the IP layer below it, the TCP layer strips the TCP header data from the packet, does some data reconstruction if necessary, and then sends the data to the correct application using the port number taken from the TCP header.

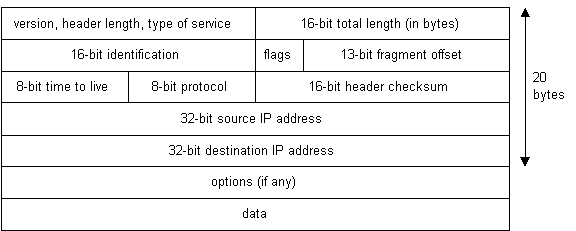
**TCP is a connection-oriented, reliable, byte stream service.** Connection-oriented means that two applications using TCP must first establish a connection before exchanging data. TCP is reliable because for each packet received, an acknowledgement is sent to the sender to confirm the delivery. TCP also includes a checksum in it's header for error-checking the received data. The TCP header looks like this:



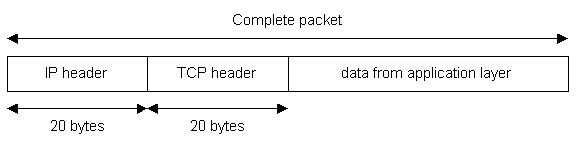
Notice that there is no place for an IP address in the TCP header. This is because TCP doesn't know anything about IP addresses. TCP's job is to get application level data from application to application reliably.

## Internet Protocol

**IP is an unreliable, connectionless protocol**. IP doesn't care whether a packet gets to it's destination or not. Nor does IP know about connections and port numbers. **IP's job is too send and route packets to other computers**. IP packets are independent entities and may arrive out of order or not at all. It is TCP's job to make sure packets arrive and are in the correct order.



Below is what a packet looks like after passing through the application layer, TCP layer, and IP layer. The application layer data is segmented in the TCP layer, the TCP header is added, the packet continues to the IP layer, the IP header is added, and then the packet is transmitted across the Internet.



**UDP** (User Datagram Protocol) is an alternative communications protocol to Transmission Control Protocol (TCP) used primarily for establishing low-latency and loss-tolerating connections between applications on the internet.

Where UDP enables process-to-process communication, TCP supports host-to-host communication. TCP sends individual packets and is considered a reliable transport medium; UDP sends messages, called [datagrams](https://searchnetworking.techtarget.com/definition/datagram), and is considered a best-effort mode of communications.

Difference between TCP and UDP

**TCP** is a connection-oriented protocol and **UDP** is a connection-less protocol. **TCP** establishes a connection **between** a sender and receiver before data can be sent. **UDP** does not establish a connection before sending data.

|  |  |
| --- | --- |
| TCP is a connection-oriented protocol. Connection-orientation means that the communicating devices should establish a connection before transmitting data and should close the connection after transmitting the data. | UDP is the Datagram oriented protocol. This is because there is no overhead for opening a connection, maintaining a connection, and terminating a connection. UDP is efficient for broadcast and multicast type of network transmission. |
| TCP is reliable as it guarantees delivery of data to the destination router. | The delivery of data to the destination cannot be guaranteed in UDP. |
| TCP provides extensive error checking mechanisms. It is because it provides flow control and acknowledgment of data. | UDP has only the basic error checking mechanism using checksums. |
| Sequencing of data is a feature of Transmission Control Protocol (TCP). this means that packets arrive in-order at the receiver. | There is no sequencing of data in UDP. If ordering is required, it has to be managed by the application layer. |
| TCP is comparatively slower than UDP. | UDP is faster, simpler and more efficient than TCP. |
| Retransmission of lost packets is possible in TCP, but not in UDP. | There is no retransmission of lost packets in User Datagram Protocol (UDP). |
| TCP has a (20-80) bytes variable length header. | UDP has a 8 bytes fixed length header. |
| TCP is heavy-weight. | UDP is lightweight. |
|  |  |
| TCP doesn’t supports Broadcasting. | UDP supports Broadcasting. |
| UDP is used by DNS, DHCP, TFTP, SNMP, RIP, and VoIP. |  |

Web Services:

A **web service** is any piece of software that makes itself available over the **internet** and uses a standardized XML messaging system. XML is used to encode all communications to a **web service**. For example, a client invokes a **web service** by sending an XML message, then waits for a corresponding XML response.

Exposes the web services.

Return of web services will be in XML/JSON.

Type Of Webservices

REST Web Services: REpresentational State Transfer

SOAP Web Services:

HTTP Exchange:

Client 🡪 HTTP request ->Webservice -> server🡪 http response -> Client

HTTP Protocol: (Message Format)

SOAP – Simple object Access Protocol.

REST – Change message in format of XML , JSON and text.

HTTP request:

Services definition:

SOAP:

Definition of services is WSDL, it will be available to for client to know data type, return type, argument and methods available.

REST : NO Services Definition.

SOAP Web Services Specifications: is list of rules must followed by all SOAP.

REST: REpresentational State Transfer: No rules

HTTP -> Hyper Text Transfer Protocol.

Hypertest – is text which contain Hyperlinks

Address:

Resources based URL is good practices

[www.sachin.org/test/1232](http://www.sachin.org/test/1232)

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

### HTTP and RESTful Web Services

HTTP provides the base layer for building web services. Therefore, it is important to understand HTTP. Here are a few key abstractions

### Resource

A resource is a key abstraction that HTTP centers round. A resource is anything you want to expose to the outside world, through your application. instances of resources are:

Get user, get list of user and other application resources.

When you develop RESTful services, you need to focus your thinking on the resources in the application. The way we identify a resource to expose, is to assign a **URI** - **Uniform Resource Identifier** - to it. For example:

Get user -- /user/get , get list of user -- /user/range

#### Resource Representation

REST does not worry about how you represent your resource. It could be XML, HTML, JSON or something entirely different! The only important thing is you clearly define your resource, and perform whatever actions that are supported on it by making use of features already provided by HTTP. Examples are:

* Create a user: POST /users
* Delete a user: DELETE /users/1
* Get all users: GET /users
* Get a single user: GET /users/1

how a REST service is generally implemented:

* **Data Exchange Format**: No restriction is imposed over here. JSON is a highly popular format, although other such as XML can be used as well
* **Transport**: Always HTTP. REST is completely built on top of HTTP.
* **Service Definition** : There is not standard to specify this, and REST is flexible. This could be a drawback in some scenarios, as it might be necessary for the consuming application to understand the request and response formats. There are widely used ones however, such as WADL (Web Application Definition Language) and Swagger.

### The Components Of HTTP

HTTP defines the following for a request:

* **Method**
* **Headers**
* **Body**

For the response, HTTP defines the

* **Headers**
* **Body**

#### HTTP Request Methods

The method used in a HTTP request indicates what action you want to perform with that request. Important examples are:

* GET: Retrieve details of a resource
* POST : Create a new resource
* PUT: Update an existing resource
* DELETE: Delete a resource.

# REST API Best Practices - With Design Examples from Java and Spring Web Services

1. Customer first approach:
2. Contract first approach:
3. YARAS

**YARAS** stands for **Yet Another RESTful API Standard**. YARAS provides standards, guidelines and conventions to be followed while developing RESTful web services. It defines tips for things such as:

* How you should name your services?
* How you should structure your request and response?
* How you should implement filtering, sorting, paging and other actions?
* How you should approach versioning?
* How you need to approach API documentation?

Typical features include:

* Request and response structures
* Error handling
* Filtering
* Searching
* Versioning
* Support for mock responses
* HATEOAS

#### Use Proper HTTP Request Methods

#### Use Appropriate HTTP Response Status

### Focus On Representation (Request and response representation)

### Use Plurals (Always use plurals when you name resources.)

### Have Great Documentation(Swagger now its open API doc standards )

Support Versioning(Use Different URIs/ Use A Request Parameter/ Use A Header headers="X-API-VERSION=1" / Media Type versioning)

### Think About Error Handling

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# REST Web Services 01 - Introduction

Characters of web service

* Web Service communication – backed by http
* HTTP exchanges – data format
* HTTP protocol/ Methods - GET etc
* Service Definition- SOAP(WSDL), nothing for REST

SOAP has specification and rules.

REST web services have no rules.

JSON - JavaScript Object Notation

# REST and HTTP

Protocol is machinima or language for communities the data/ hypertext (hyperlink).

REST API

1. Resource based URI:
2. HTTP methods.
3. HTTP status code.
4. Metadata – request header , response header.
5. Content Negotiation (we can send both JSON and XML )

MetaData: header contain metadata of request/response

http status code: 200 – success, 500 – Server internal error, 404 – resources not available

content Type – text/xml, application/json

# Resource URIs

Design URI (Uniform resources Identifier) use nouns (like profile, messages etc)

# Collection URIs

Types:

* Instance Resources URI – is having unique value to identify instances of resources. (/profile/{id})
* Collection Resources URI – is used get all the resources or filtering result or custom filter. (/messages, /profiles)

# HTTP Methods

HTTP Methods: mint to do some set of action.

GET: to get the resources.

POST: to create/save the resources;  
PUT: to update the resource.

DELETE: to delete the resource.

# Method Idempotence

Readonly(Safety) : GET

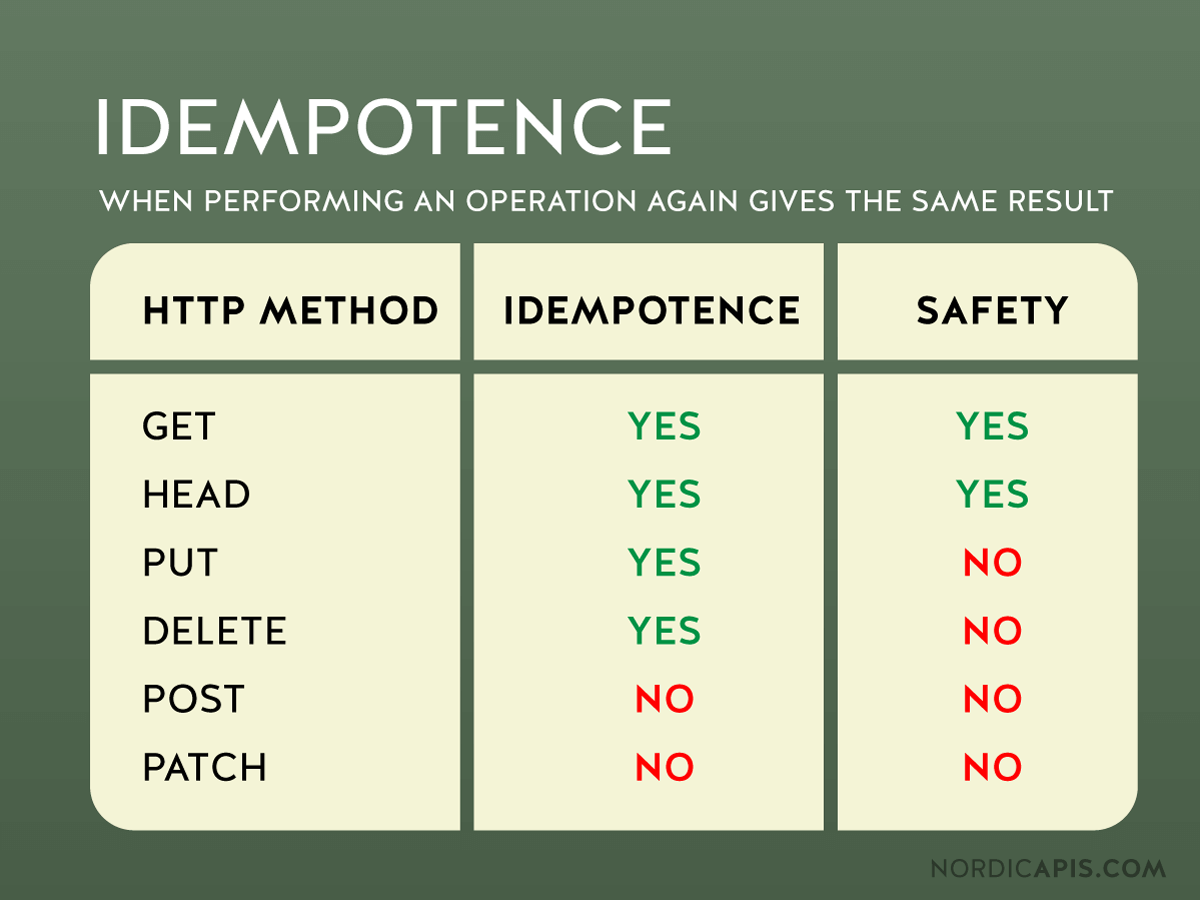
Writeonly: PUT , POST, DELETE

GET is safe to make duplicate request, because no change in sever side.

Safe Repeatable(idempotence): GET, DELETE, PUT

Not safe Repeatable(not idempotence): POST

Idempotence is the property of certain operations in mathematics and computer science whereby they can be applied multiple times without changing the result beyond the initial application.



Difference between PUT vs POST?

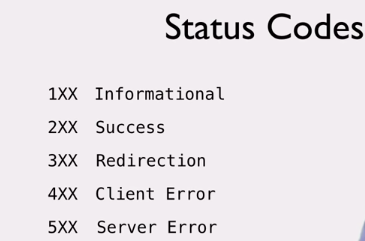
PUT is idempotence

POST is not idempotence

# REST Response

Representing of the resources any other type JSON/XML. So REST.

* Content-type
* HTTP Headers
* Status code





# HATEOAS

(Hypermedia As The Engine Of Application State)

Is way to provide the links to resources in API response, so need to deal with API document

REST – no service Definition.

Hypermedia in API response drive the application state. In other word HATEOAS.





# The Richardson Maturity Model

To check how RESTful is API?

LEVEL 0: swamp of POX (plain old xml) only one endpoint/resource URI.

LEVEL 1: Individual URIs for each of resource,

LEVEL 2: Use standard HTTP Methods, status code and idempotence.

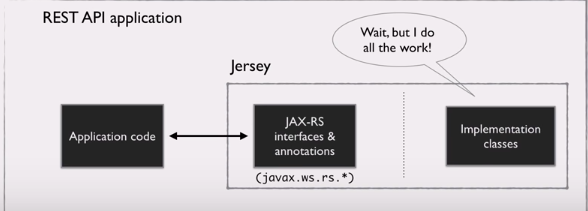
LEVEL 3: Implements HATEOAS, Response have links that client can use.

# What Is JAX RS?

There are libraries to implement the REST API. Like Rest light, Jersey etc.

All the libraries follow common interface/API to implement API. This common API is called JAX RS.





Why Jersey?

Its reference implementation.

MAVEN is management tool.

# Understanding the Application Structure

Servlet and filters are only way to handle request.

URL : <http://localhost:8080/message/webapi/myresources>

[http://localhost:8080/<application\_name>/webapi/<resources](http://localhost:8080/%3capplication_name%3e/webapi/%3cresources)>

web.xml is where servlet is configured and servlet mapping is configured with urlPattern

<servlet-mapping>

<url-patter>/webapi/\*</url-pattern>

</servlet-mapping>

<http://localhost:8080/message/webapi> -> Jersey will handle upto this URI

/<resource> -> must user should configure to handle resources.

<init-param> is used to scan the package specified.

# Creating a Resource

@GET – is used to map the request if requestmapping is not given at the method level

@Produces(MediaType.PLAN\_TEXT) – is used say that is type of response.

@Produces(MediaType.APPLICATION\_XML) – is used to return XML format response it uses JAXB API to convert to xml and in POJO class add @XmlRootElement

@PathParam(“id”) long id – is used to access the pathparam (/message/1)

@Consumes(MediaType.APPLICATION\_JSON)

@POST

@PUT

@DELETE

@QueryParam(“name”) String name – is used to access query parameter like /profile?name=Sachin

@MatrixPatam(“name”) String name – is used to access matrix value like /profile;name=sachin

@HeaderParam(“customHeaderValue”) String headerValue – is used to access header value.

@CookieParam(“sessionId”) String int id – is used to access the value from cookies.

@FormParam(“submitForm”) Map<String,String> formValue – is used when form submit is used.

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. XML is often used in data exchange between applications.

Spring Boot starters are a set of convenient dependency descriptors which greatly simplify Maven configuration. The spring-boot-starter-parent has some common configurations for a Spring Boot application. The spring-boot-starter-web is a starter for building web applications with Spring MVC including RESTFul applictions. It uses Tomcat as the default embedded container. The spring-boot-starter-data-jpa is a starter for using Spring Data JPA with Hibernate. The spring-boot-starter-test is a starter for testing Spring Boot applications with libraries including JUnit, Hamcrest and Mockito.

In the application.yml file we write various configuration settings of a Spring Boot application. The port sets for server port and the context-path context path (application name). After these settings, we access the application at localhost:8086/rest/. With the banner-mode property we turn off the Spring banner.

# Using Context and BeanParam annotations