Rest API

1) WWW is different from internet

2) constraints with 6 categories which are referred as web architechtural style:

2.1) Client - server -> It is separation of concern ie both may be deployed/ implemented independently using any tech but while communication there must be well defined communication protocol, common format of request/reply and error handling mechanism also there is no restrictions defined for the number of clients its solely depends on capacity of server

2.2) Uniform Interface -> Communication between these interfaces (client, server and other network related devices) should follow uniformity ie they have common vocabulary ie http this can bring decoupling of services from architecture without any overhead These Categories are divided in 4 guiding principles:

2.2.1) Identification of resource -> Every resource on internet should be identified by unique

identifier or Unique Resource Identifier (URI). Identifier does not have any relation with format of response. The identifier shouldn’t change

2.2.2) Manipulation of records -> Client will have ability to manipulate representation of resource

eg: application/json ie same resources with same identifier can be represented to different client in different ways. This is also known as content negotiation.

2.2.3) Self descriptive message -> to keep communication stateless need to send all necessary information that can self describe request as well as state of resource.

2.2.4) Hypermedia as engine of application state (HATEOAS) -> Accessing the API should be similar to accessing web page client should be able to determine other part of API from response ie

response of API or state of resource should include links to other related resources

2.3) Layered system -> This means that between client and server there would be many components ie proxies, gateways etc these are used for cacheing ,load balancer, security etc. All these components support common features like:

1) Support standard protocol for communication

2) They only communicate to layer above them and layer below them

3) These layers can be added ,removed and rearranged as per our requirements.

However, this brings latency in the system but this also improves scaling and limits system complexity

2.4) Cache -> It is the ability to store frequently accessed data, It helps to reduce load on server and also reduces server latency, It can exist in any layer between client and server. There are many caching strategy:

1) Browser Cache
2) Proxy Cache
3) Gateway Cache etc.

2.5) Statelessness-> It means application current state or client current state. Server should not store any application state related information. At the time of request client should pall all necessary information

2.6) Code on demand: This tech allows server to send software code to the client to be executed on client machine It is not mandatory on web architecture because client needs to understand code it downloads from server

3) Web Service -> Service : It is a software or piece of code that performs a particular task or response to a specific event as per request and produces specific output eg: system event , human interaction etc

When this piece of code is just on our system it is called service , when it becomes available over the internet it becomes web service

Web service is a service or a piece of code that is available on internet that have standardised communication protocols and information exchange medium.

4) Web API -> API : Application programming interface , Interface is edge or boundary where 2 or more systems can meet and exchange information.

API keeps consistency across applications

API is set of definitions or protocols that allow one application to interact with other

Web API is the API available on the web

Q) Difference between web service and web API

Ans) Web API acts as a interface between 2 application over internet, can use anything hence it is light weight.

Web Service facilitates interaction of 2 machine over internet, Web service can only use SOAP,REST,XML-RPC

All web services are web API but not vice versa

5) REST API -> API or web API that have architectural style of REST

6) Web Architectural style : SOA,OOA,ROA
6.1) Service oriented Architecture (SOA): architecture oriented at the service. The whole system is made up of services and those services perform some operations and they can communicate with each other using interfaces.

Consumer of SOA thinks service is a black box

SOA is stateless hence highly scalable eg, soap over http or soap over JMS
6.2) Object oriented Architecture (OOA): Oriented at object.Involves communication with object instance. And is mainly for

managing lifecycle for objects

Communication is stateful, communication with previously created object instance.
6.3) Resource oriented Architecture (ROA): oriented to resource, resource is anything that can be stored in computer

Resource provider: server that provides resource eg, azure and aws

Resource representation: It is useful information about current state of the resource it

has specific format and language

SOA uses REST and SOAP, ROA uses only REST

Q) compare SOAP & REST

Ans)

SOAP	REST
Representation State Transfer	SIMPLE OBJECT ACCES PROTOCOL - developed by Microsoft
	SOAP is a protocol that defines how to prepare message in XML and it defines goals of request and action
Architectural style	It is a protocol

ROA- Access resources and exposes data, data driven	SOA- exposes application logic as a service,Function driven
Transfer protocol - only HTTP	Transfer protocol- Http,JMS
Communication- Client server communication, one central server hosts entire system	Communication - Distributed communication, Process involve multiple sub processing over different servers.
Data format - plain text, html, json	Data format - only xml
Payload - uses XML	Payload - uses http and Json
Security - support ws security	Security - https
Bandwidth - more	Bandwidth - Less
Caching - Not possible	Caching - possible
State- stateful	State - stateless
Dependency - tightly coupled with server	Dependency - not tightly coupled

SOAP is preferred in financial application because of its security and built in ACID properties

HOW TO DESIGN URI/URL

REST use HTTP and ROA, communication starts when client sends req to server with proper structure.

Request	Response
URI	Response Body
Http Method	http Response code
Header	
Request body	

According to RFC - 3986
Generic URI Sequence -> scheme:[//authority]path[?query][#fragment]

Scheme	letter, numbers and special chars 4 Should be registered by IANA - Internet Assigned Number Authority 5 eg: http, https, mailto, file, data etc 6 Rest uses http and https
Authority	1 Optional 2 Combination of user info (username and password), host & port 3 ipaddress:port we can also use domain name

Path	1 Path of resource. 2 It should be hierarchical and should resemble our file system 3 eg: https://www.jraca.com/user{{userid}}/videos/{{videoid}} 4 forward slash represents that user is parent of video.
Query	1 Optional element 2 Helps to fine-tune our result 3 It adds additional value to URI.
Fragment	1 starts with # sign 2 Takes us to particular section on a web page 3 It is useful for modern web UI Not useful for REST

(:),(?),([]),(@) are reserved for URI design

Guidelines

1. “/” represents hierarchical relationship
2. Trailing “/” shouldn’t be used
3. “-” should be used to improve readability and use underscore as least as possible.
4. use lowercase
5. do not use file extension in URI instead use header to store media type.

Resource Modelling: “/” represents hierarchical relationship of resource to each other. It also represent individual resource.
In REST there are 4 archetype.
Every resource aligns with only one archetype avoid designing hybrid. If resources are closely related then create separate API for that.
Archetype:

Archetype	Definition

Document	1 Singular entity same as object or db entity 2 Representation-> field:value, value can be object, link or anything 3 Document is base archetype and other type are children of this. 4 eg: user
Collection	1 Collection is directory or collection of object managed by server 2 eg users 3 client can add / delete /modify but its effect is decided by server

Store	1 Directory which is managed by client 2 eg: videos directory inside user
Controller	1. Executable functions which take some input and gives output 2. eg: we need to calculate something or send message 3. Actions cannot be mapped to standard methods- get put etc 4. there are no Childs and controller name appear at the last

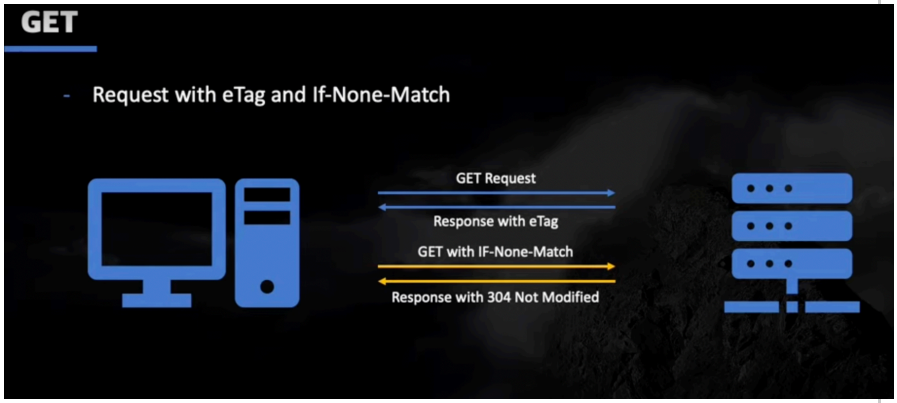
Query: Contains list of parameters. These parameters could be mandatory or optional. Query parameters do not affect caching

mechanism of server.
These can be used to filter/paginate etc

REST:HTTP Methods

HTTP protocol was originally designed as interface for distributed systems
In http we have standard operations on resources. These helps in identifying clients intention behind calling the API.

METHOD	Description

GET	1 Requests representation of a resource 2 Request body is not part of GET request 3 GET request retrieves data but we can attach header properties such as - If-Modified-Since - If-Unmodified-Since - If-Match - If-None-Match - If-Range •  • GET is Safe, Idempotent and Cacheable

<table>
<tr>
<td>POST</td>
<td>
1 Used to submit an entity that can cause change in resource state or it can affect server
2 Post is request to server for accepting entity in the request body as resource or sub resource. Ie to create a new resource
3 eg: adding new user
4 Response is not cacheable but we can make it using cache control or expires header field or redirect.
5 Post is Not Safe, Not Idempotent, Cacheable but not always
</td>
</tr>
<tr>
<td>PUT</td>
<td>
1 It is to replace the resource or create a new one if not found with the request body
2 If resource has been created then server informs with 201 code
3 Post is Not cacheable,Not safe and Idempotent
4 Difference between POST and PUT
<table>
<tr><th>POST</th><th>PUT</th></tr>
<tr><td>-> Not Idempotent</td><td>-> Idempotent</td></tr>
</table>
</td>
</tr>
</table>

DELETE	1 Request to server for deleting resource Identified by URI 2 Response codes -> 200: deleted successfully, 202: Accepted ,but not executed,204: No content, successful deletion but nothing in respond body 3 Not Cacheable, Idempotent,Not safe 4 Making delete idempotent is important 5 Developers should not expose delete last/first by rest because it will stop when nothing is left in the database
Head	1 Requests a Header without a body. It is same as GET 2 If we want to decide that a file is too large to download hence we check its size first in the header (content length) 3 It is Idempotent, Safe and Cacheable

Options	1. Method to request communication option for resource identified by URI 2. Doesn't invoke any action on resource 3. Response of this method doesn't contain body but it can be added to make it more informative. However some server doesn't allow response body 4. Not cacheable but Safe and Idempotent
Patch	1. It is for the partial modification of existing resource 2. Patch is not idempotent and however put is, Put replace entire representation and Patch updates specific part of resource 3. Not Idempotent, Not safe, Not Cacheable

Not all resources have to support all methods. These methods are:

1. Safe -> Request doesn't alter state of server, they shouldn't make any external calls or introduce any side effects. Read only operations are safe. Get and Head

are considered as safe

2. Idempotent -> Methods that when called n times with same request generates same output. Here we should not count errors and resource expiration issues. State of server is not changed other than keeping logs and other statistics eg: put ,delete, safe methods
3. Cacheable -> Response of request can be cacheable, Get and Head are cacheable and some response codes are cacheable

Ideally the request that has both cacheable request and response code are eligible for caching

HTTP Header : Header is the segment in request that provide extra space to client and server, This extra space can be used to put additional information. This is important for client/server/proxies etc. It is collection of key-value pair where key is http property.
Data in header is not the data that client asks for but is data this data helps to make communication better and reduce bandwidth and load.
There are 4 kinds of http header properties:

1. Authentication Properties

Key	Definition
Authorization	This contain credentials for authentication Eg Authorisation : <type> <credential> Type can be basic, bearer, oAuth etc If authentication fails we get 401 code with additional Information about required authentication type or schema

WWW-Autheticate	Comes with response from server. Defines authentication method in order to gain access to a resource. Eg) WWW-Autheticate : <type> realm = <realm>, Where realm is space/ area or group of resource that have same credentials. To get this property we need to configure server.
Proxy-Authorization	Proxy servers are used for logging, caching, security, saving bandwidth and if that also need credentials then it is passed through this key
Proxy-Autheticate	Same as WWW-Autheticate

2 Content Properties: In request or response body specified data is solely related to our business logic. But we also need extra information for ease of communication such as media type of resource.

Key	Definition
Content-Type	Represents Media type of Resource-can be used for both request and response. Eg) Content-Type: <mime-type>;<encoding-type> Mime is multipurpose internet mail extension. It is combination of two values type & subtype eg Application/json, Text/html etc

Content-Length	Represents size of content in bytes Eg) Content-Length:112

Content-Encoding	To make communication faster we should reduce about of data we are exchanging. Data compression is one of the ways to reduce size of payload This field holds information about compress algorithm. When client gets payload it needs to de-compress to get desired type of media. Eg: Content-Encoding: compress, gzip etc

Accept-Encoding	Sometimes client can’t apply the algorithms specified in Content -Encoding In the case we specify Accept-Encoding in request. The server will try to respond in this format. Basically client is negotiating for encoding type Eg Accept-Encoding: gzip

Content-Location	Represents alternate location for returned data. Value of property is URL that return data. Eg: Content-Location: /myfirst-blog, For online transaction we get receipt URL

3 Conditional Properties: Used to make our request conditional.

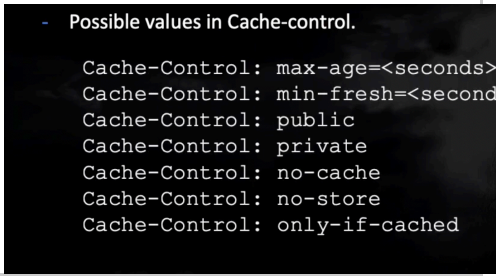
Key	Definitions

Etag	Response header property. Value of this represents version of the resource. ETag:<value> Client can store it. Earlier server used to calculate etag value every time with request now it stores the value once the resource is modified Stage takes precedence over last-modified
If-Match	Pass Etag , server will take action if etag matches with current value Can be used with put
If-None-Match	Pass Etag, server will take action if not matched.

Last-Modified	This is a response property but instead of hashcode like etag it gives time stamp
If-Modified-Since	Success if there is modification after Last-Modified
If-Unmodified-Since	Success if server is unmodified after Last-Modified

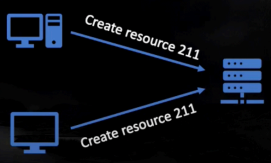
4 Caching Related Properties: Save it now use it later

Key	Definition

Age	Defines time for how long object or data has been in proxy cache. Value of it is in sec Calculated based on current time and time of fetching Client can ask for fresh data if data is sitting in cache for too long
Cache-Control	Defines instruction for caching - both client and server can use this. 

Guidelines

- ✓ Should always use **Content-Type**, **Content-Length** because client and intermediaries use this data a lot.
- ✓ **Last-Modified** or **ETag** must be used to prevent conflict at server side.
- ✓ Stores resource type must use PUT with Conditionals to save simultaneous update.
  - **Store** archetype uses put method for Insert as well as update
  - There is no way to know client's intention behind calling api.



Here, both clients trying to create same resource, Server responds with conflict. If we use PUT with ETag, Or Last-Modified.

- ✓ **Content-Location** must be used to show location of the created resource.
  - If we create resource runtime, we must use Location header