REST
Section-A

Acknowledgement: Java Brains

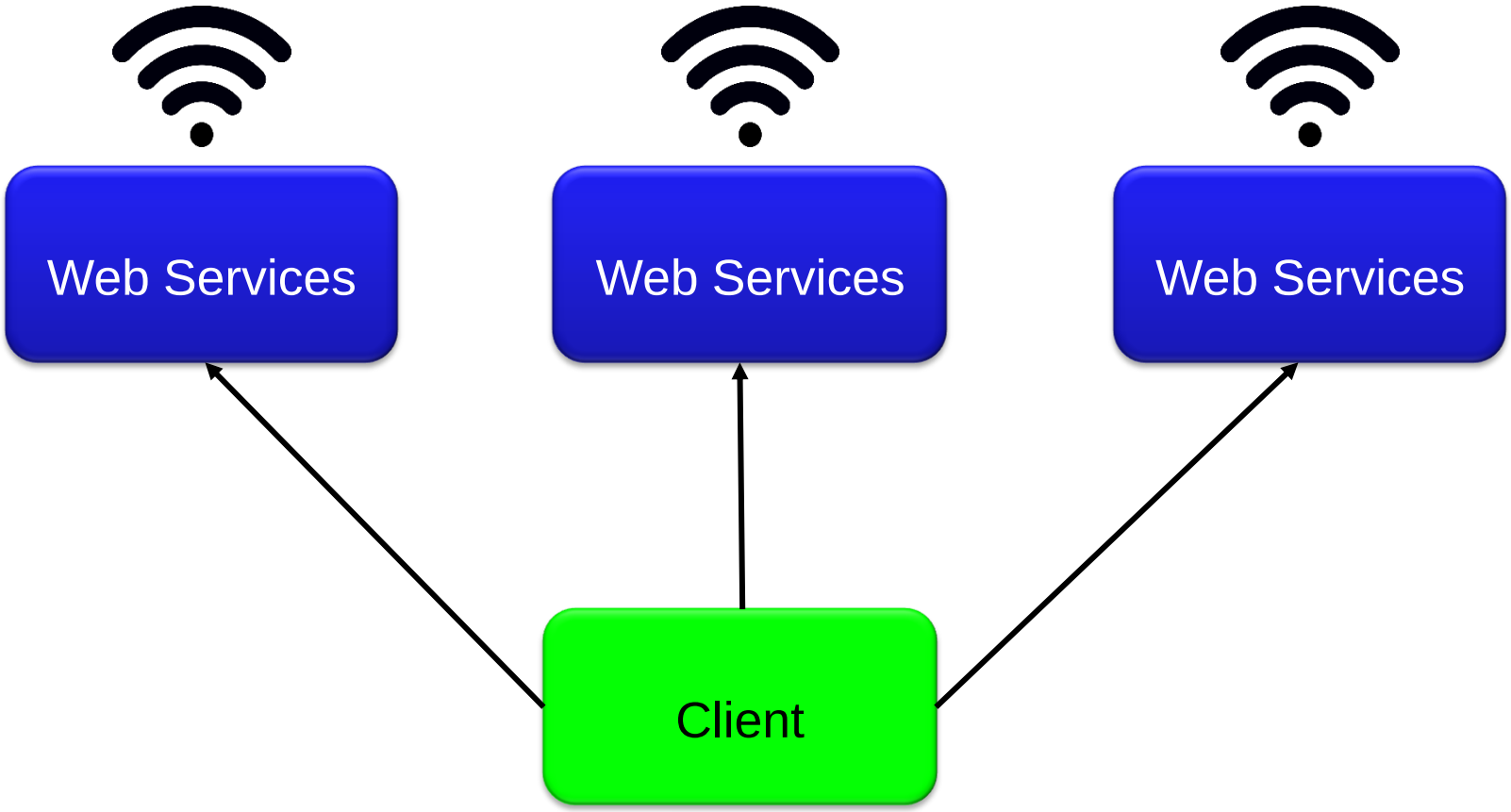
Two Parts

- **Section A** REST API concepts
- **Section B** RESTful Web services implementation using JAX-RS

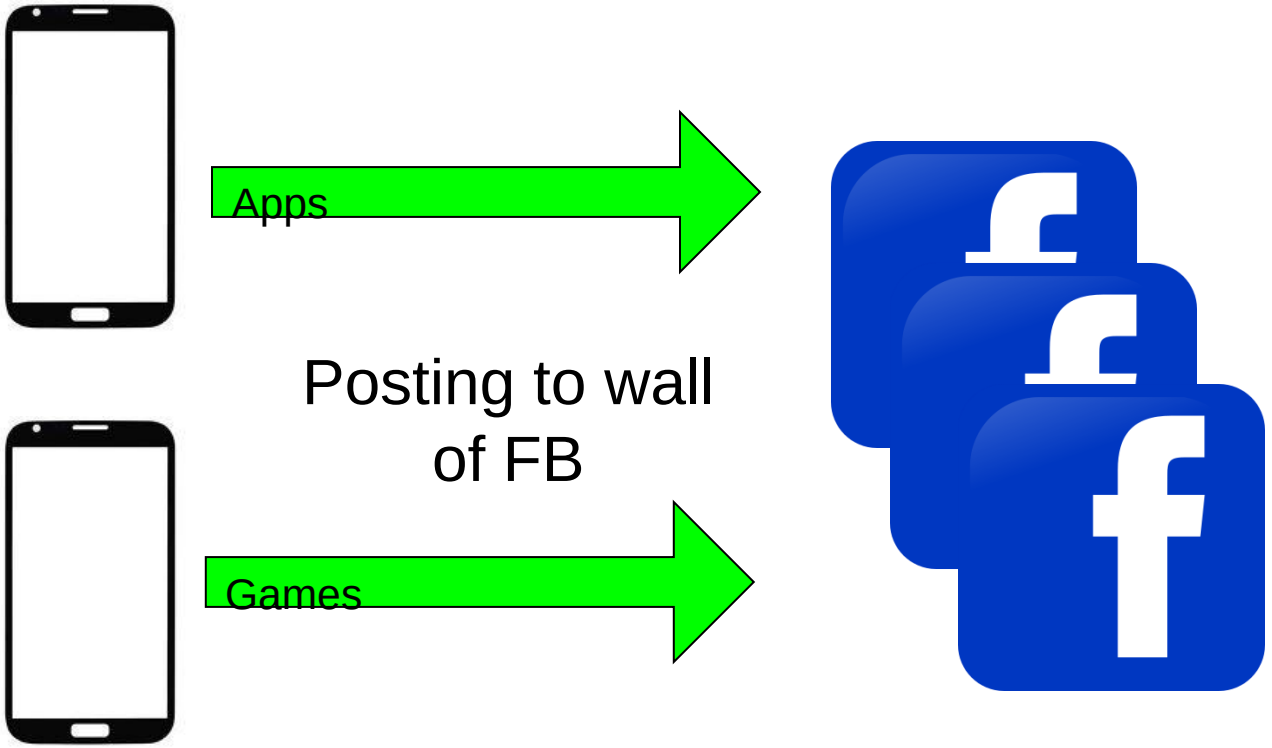
Web Services

- are Services that are exposed on Internet for the Programmatic accesses.
- They are online API's that you can call from your code

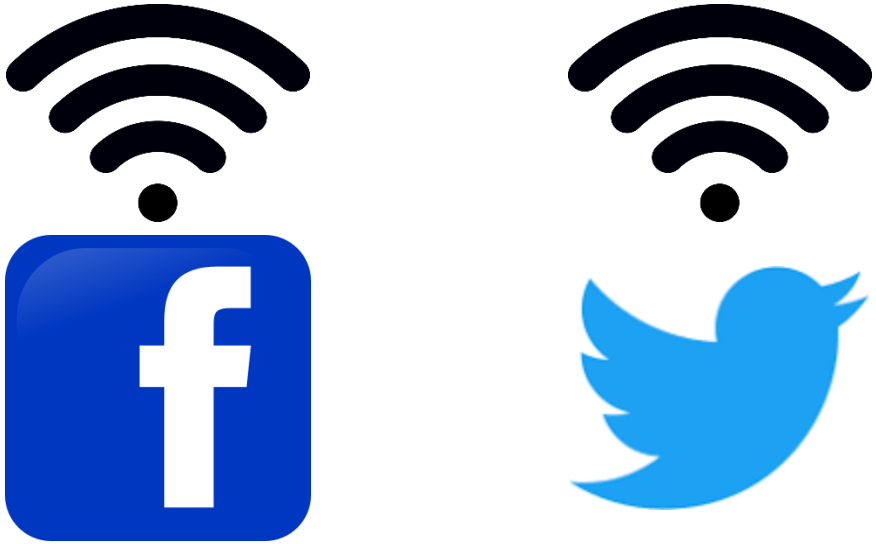
Web Services



Web Services



Web Services



Publishes their API's so other can call them from the code

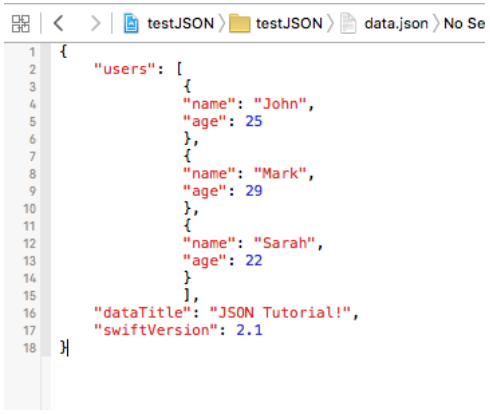
Web Services

http://twitter.com

html pages
with css and
styling

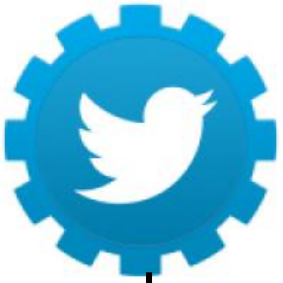
http://api.twitter.com

JSON/XML/Text



Web Services

http://twitter.com

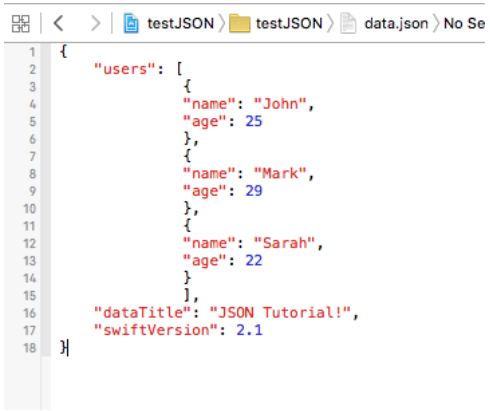


Developers can parse the data and use

html pages with CSS and styling

http://ap[illegible].twitter.com

JSON/XML/Text



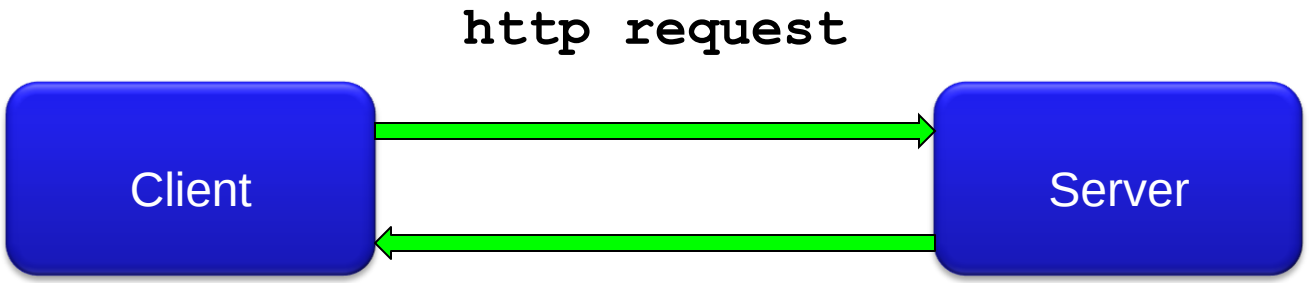
Web Services

REST Web Services: light weight

SOAP Web Services: heavy

Characteristics

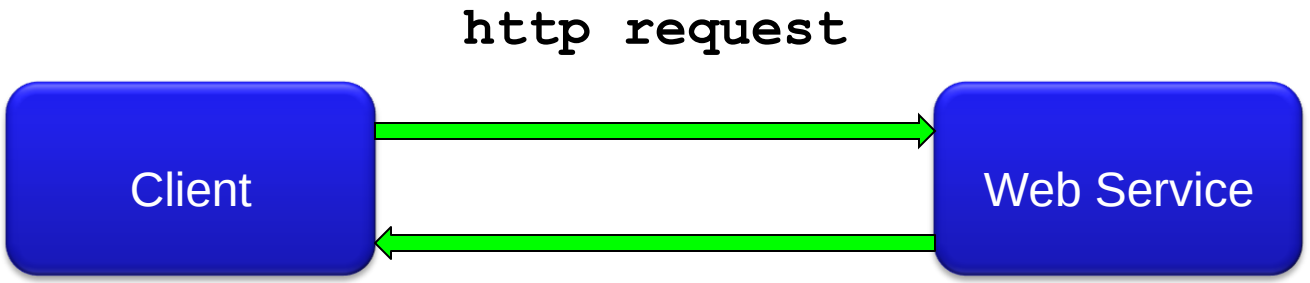
- Exchange of data over web



In the form of
pages

Characteristics

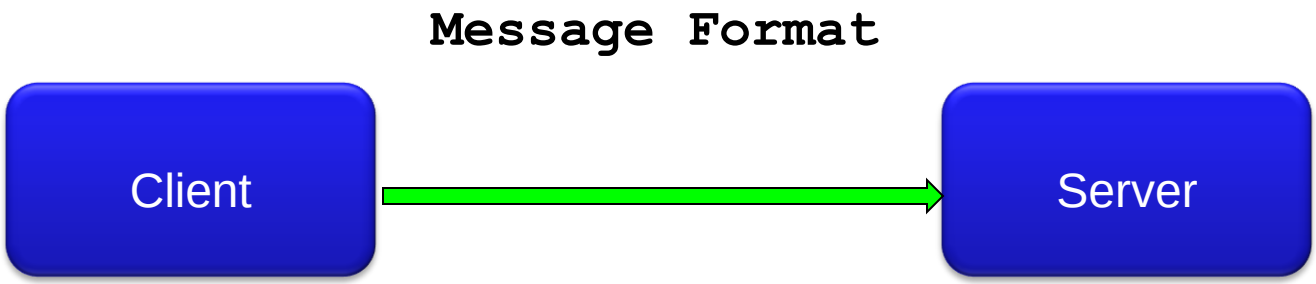
- Exchange of data over web



Could be in
the form of
data

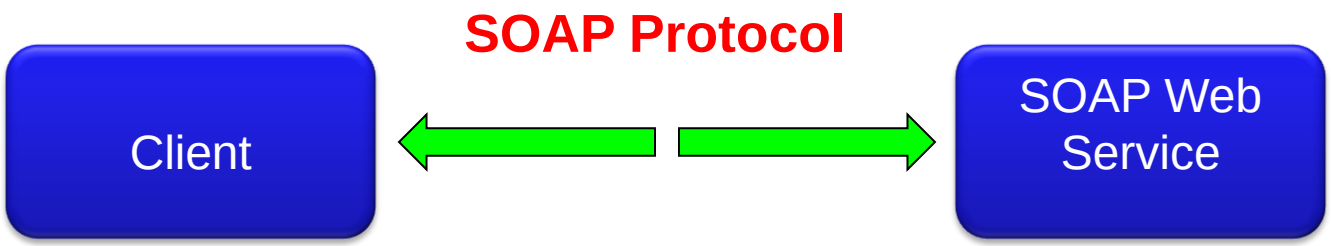
Protocol

- Exchange of data over web



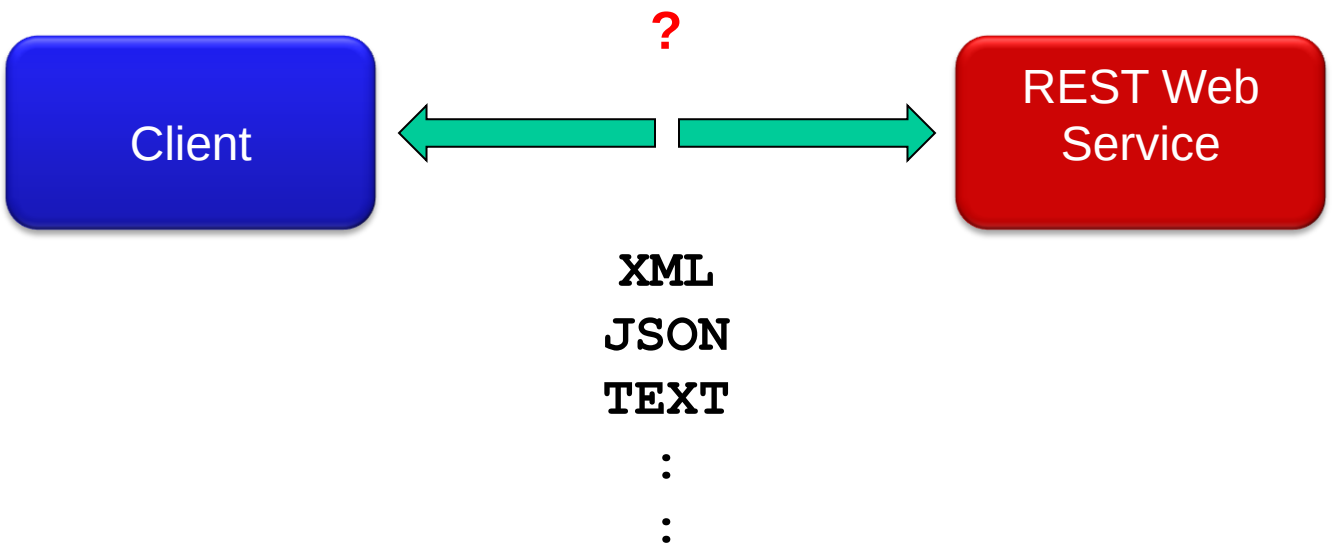
Protocol

- This exchange is standardized

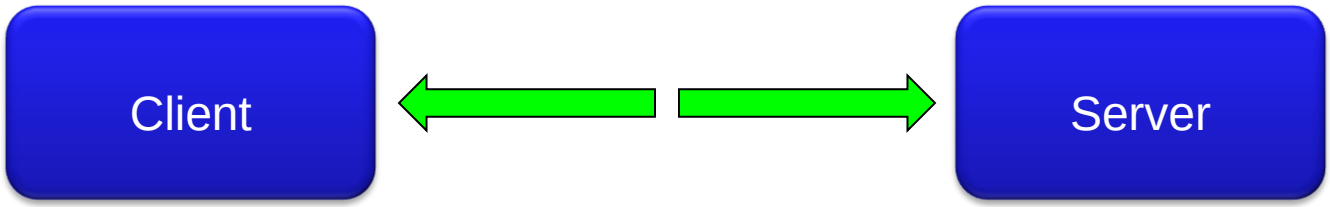


Protocol

- No protocol



HTTP Exchange



HTTP request

Several method? Which Methods?

- No rule as such

Service Definition

- In SOAP **WSDL**
- In REST **NO**
- Best REST web services wouldn't need any documentation
- SOAP uses standard specifications (rules)

REST

Roy Fielding

One of the author of HTTP Specification

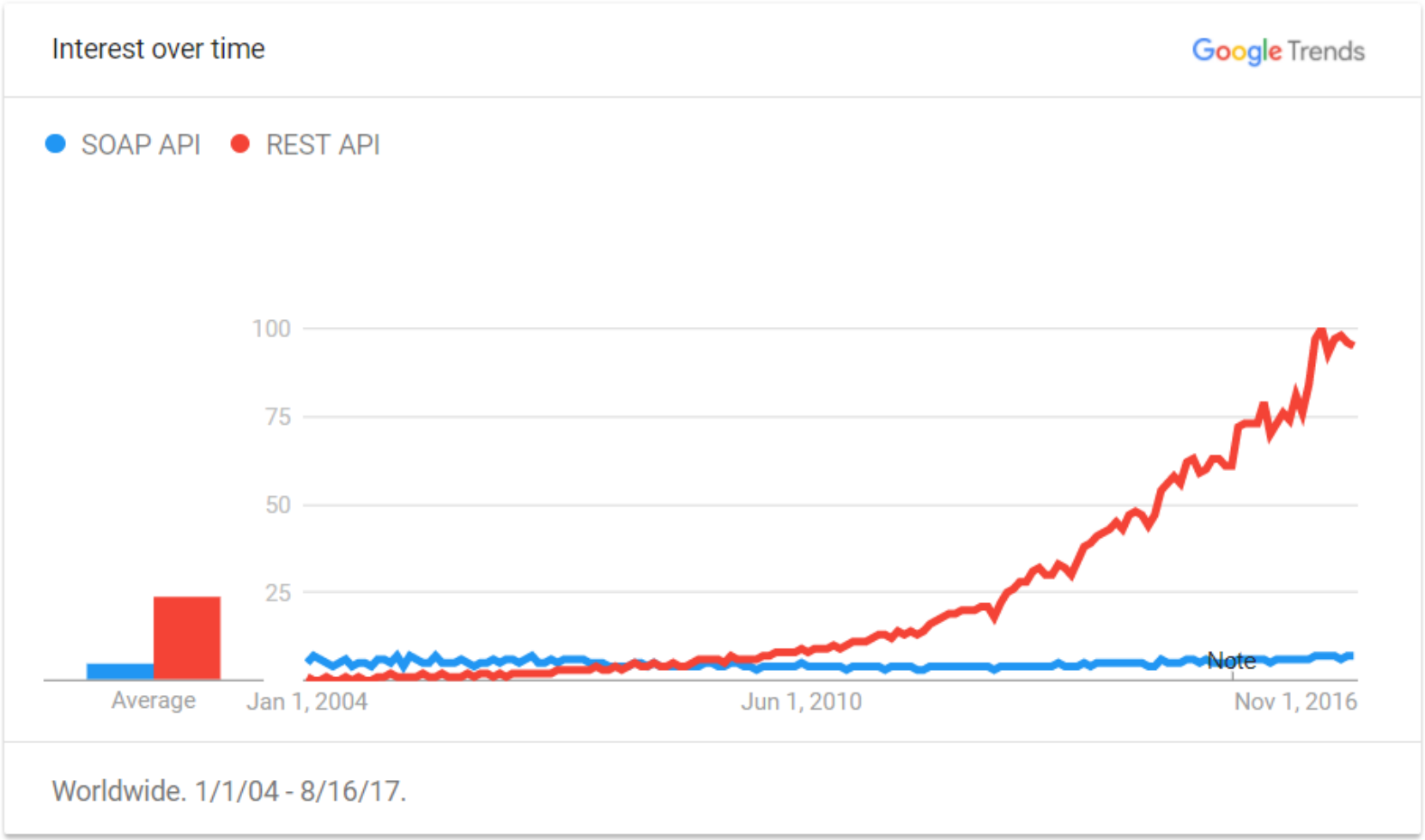
REST

REpresentational State Transfer

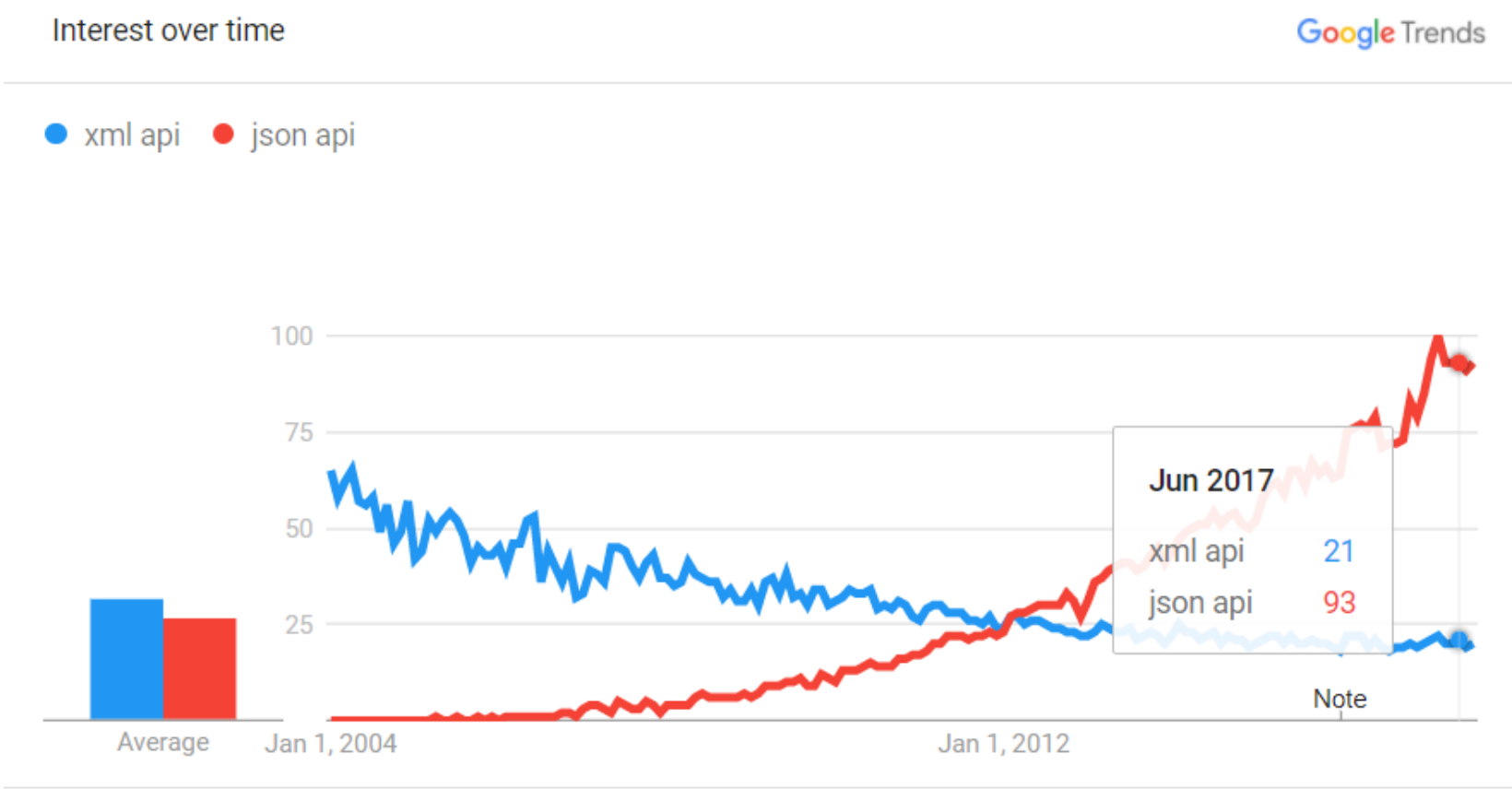
Architectural Style, guideline

REST + Web Services = RESTful Web Services

SOAP vs REST


Interest over time
Google Trends
SOAP API
REST API
100
75
50
25
Note
Average
Jan 1, 2004
Jun 1, 2010
Nov 1, 2016
Worldwide. 1/1/04 - 8/16/17.

XML Vs JSON



https://trends.google.com/trends/explore?date=2004-01-01%202017-08-16&q=SOAP%20API,REST%20API

REST architecture constraints

1. **Uniform Interface** – Resources are manipulated via **CRUD** (create, read, update, delete) operations. CRUD operations are managed via POST, GET, PUT, and DELETE request methods.
2. **Stateless** – In REST the state is contained within the request itself, or as part of the URI, query-string parameters, body or in the headers. After processing the request, the state may be communicated back via the headers, status or response body.

REST architecture constraints

3. **Cacheable** – Responses from the web service to its clients are explicitly labeled as cacheable or non-cacheable. This way, the service, the consumer, or one of the intermediary middleware components can cache the response for reuse in later requests.

REST architecture constraints

4. **Client Server** –The client/server requirement ensures that a distributed environment exists. It requires the client that sends requests and a server component that receives the requests and return a response to the client. Error responses may be transmitted as well, which requires the client to be responsible for taking any corrective action.

REST architecture constraints

5. **Layered System** – A client should not be able to tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers may add security policies, or improve scalability.
6. **Code On Demand** – This is an optional constraint. It allows a client to have logic locally via the ability to download and execute code from a remote server.

Not RESTful Not fully RESTful

Completely RESTful

Goal: Make easy for the consumer

- primarily designed to work well with HTTP/1.1
- The resources and methods are known as *nouns* and verbs of REST APIs

HTTP

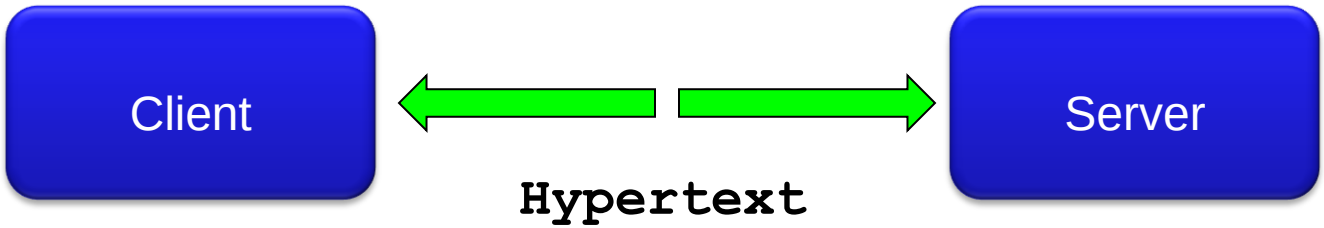
REST is highly inspired by HTTP specifications

Hyper Text Transfer Protocol

Way to exchange information

HTTP

- What you exchange



Hypertext

- Structured form of text
- Property that it contains: logical link to other text called “hyperlinks”
- We write hypertext using HTML
- Let us see how REST is inspired by HTTP

Concepts

- Resource Locations
- Since REST API responses are not to be read by the human it contains only data
- Ex. Weather application returns only:

```
{
        Weather
        Temperature
        Wind speed
}
```

Addresses

- Earlier we have Action based address

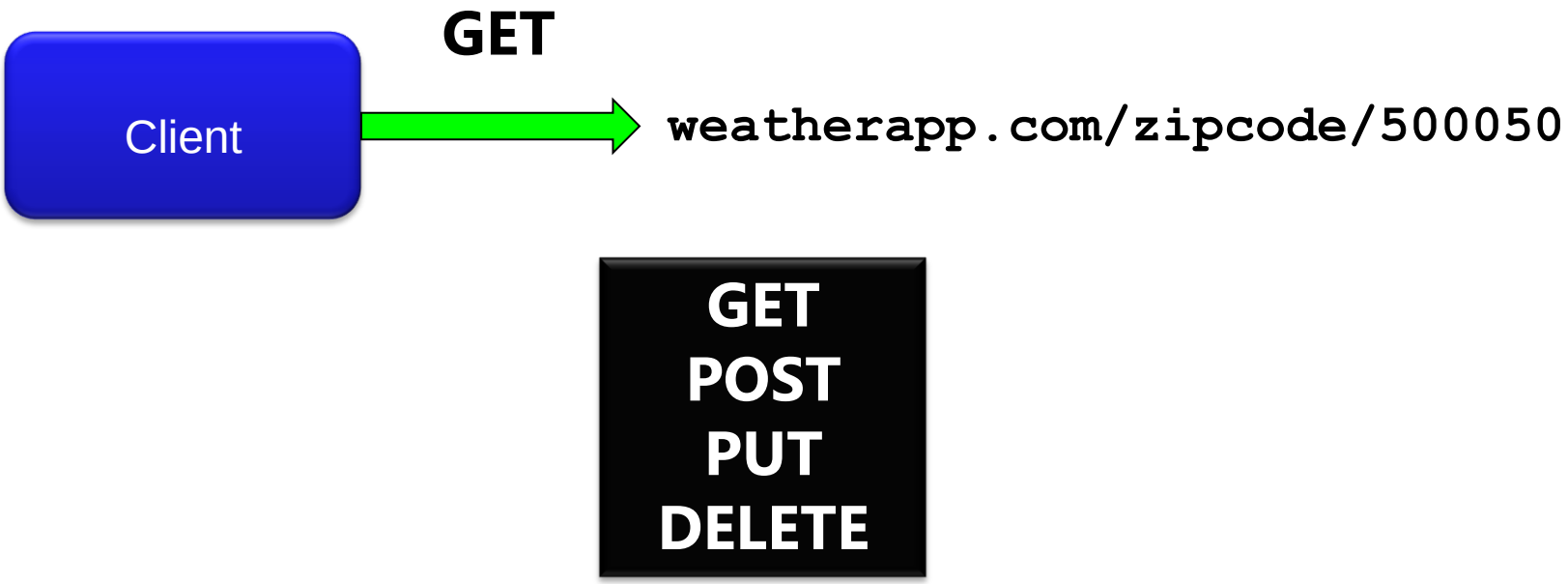
- `weatherapp.com/weatherLookpup.do?zipcode=500050`
- `http://safar.tropmet.res.in/weather-forecast-current.php?for=&city_id=3`

- Resource based (not asking to do anything)

- `weatherapp.com/zipcode/500050`
- `weatherapp.com/countries/India`

HTTP methods

HTTP methods

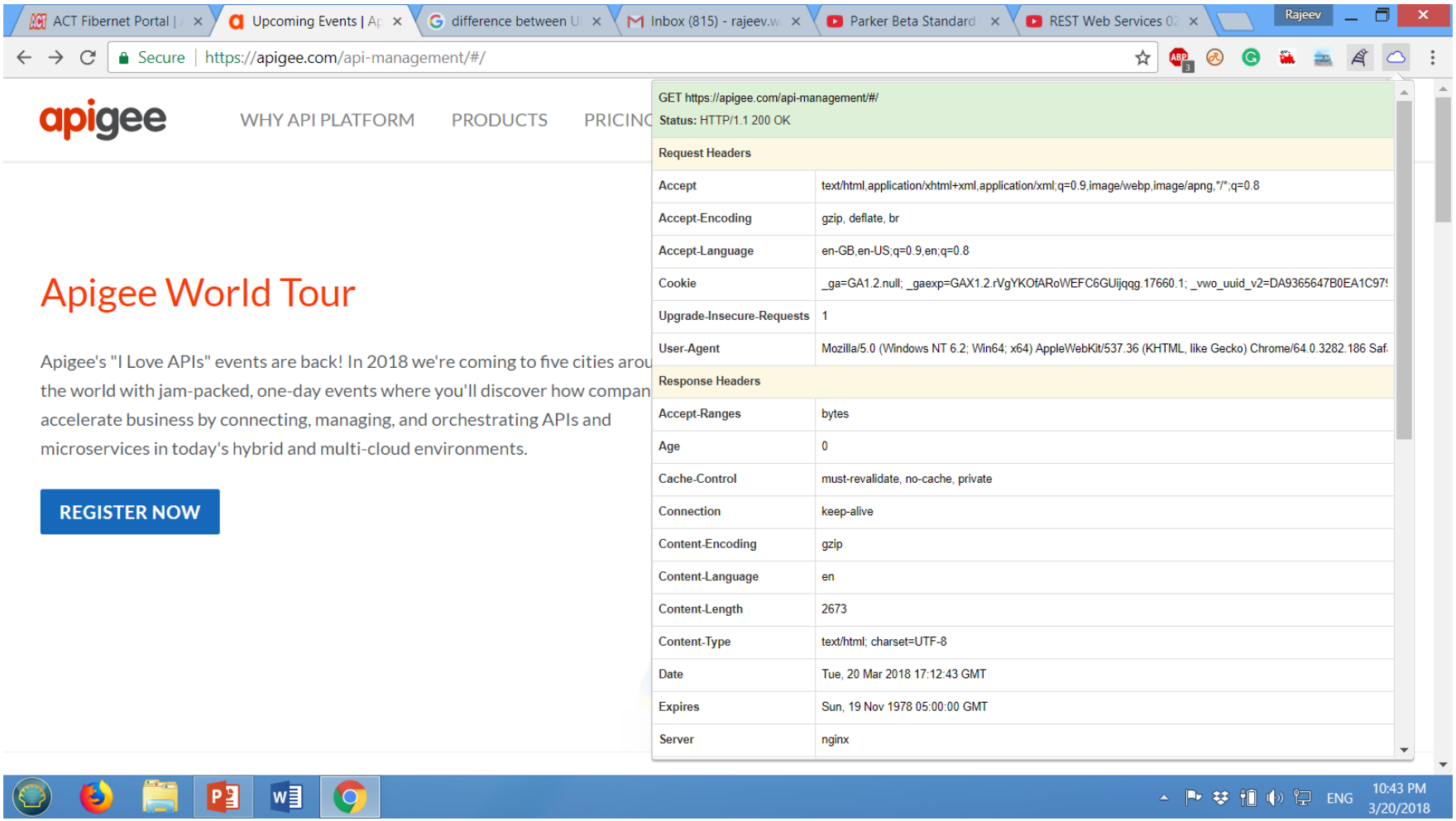


Are used based on the action they want to perform

Metadata

- In addition to the response, user gets the additional information from the server in the form of response
- HTTP status code at the first line in the response

Metadata


ACT Fibernet Portal |
Upcoming Events | Ap
difference between U
Inbox (815) - rajeev.w
Parker Beta Standard
REST Web Services 02
Rajeev
Secure
https://apigee.com/api-management/#/
apigee
WHY API PLATFORM
PRODUCTS
PRICINC
Apigee World Tour
Apigee's "I Love APIs" events are back! In 2018 we're coming to five cities arou
the world with jam-packed, one-day events where you'll discover how compan
accelerate business by connecting, managing, and orchestrating APIs and
microservices in today's hybrid and multi-cloud environments.
REGISTER NOW
GET https://apigee.com/api-management/#/
Status: HTTP/1.1 200 OK
Request Headers
Accept
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding
gzip, deflate, br
Accept-Language
en-GB,en-US;q=0.9,en;q=0.8
Cookie
_ga=GA1.2.null; _gaexp=GAX1.2.rVgYKOfARoWEFC6GUijqqg.17660.1; _vwo_uuid_v2=DA9365647B0EA1C97
Upgrade-Insecure-Requests
1
User-Agent
Mozilla/5.0 (Windows NT 6.2; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.186 Saf
Response Headers
Accept-Ranges
bytes
Age
0
Cache-Control
must-revalidate, no-cache, private
Connection
keep-alive
Content-Encoding
gzip
Content-Language
en
Content-Length
2673
Content-Type
text/html; charset=UTF-8
Date
Tue, 20 Mar 2018 17:12:43 GMT
Expires
Sun, 19 Nov 1978 05:00:00 GMT
Server
nginx
ENG
10:43 PM
3/20/2018

HTTP status

Success 200

Server error 500

Not found 404

Why status code?

Since the client is NOT a user it is a code

Content type

- Header information **Content Type field** shows the type of the value returned

text/xml

Application/json

- Same API can return multiple content types (xml/json/text…) based on the choice of the client
- This powerful feature is known as **Content Negotiation**

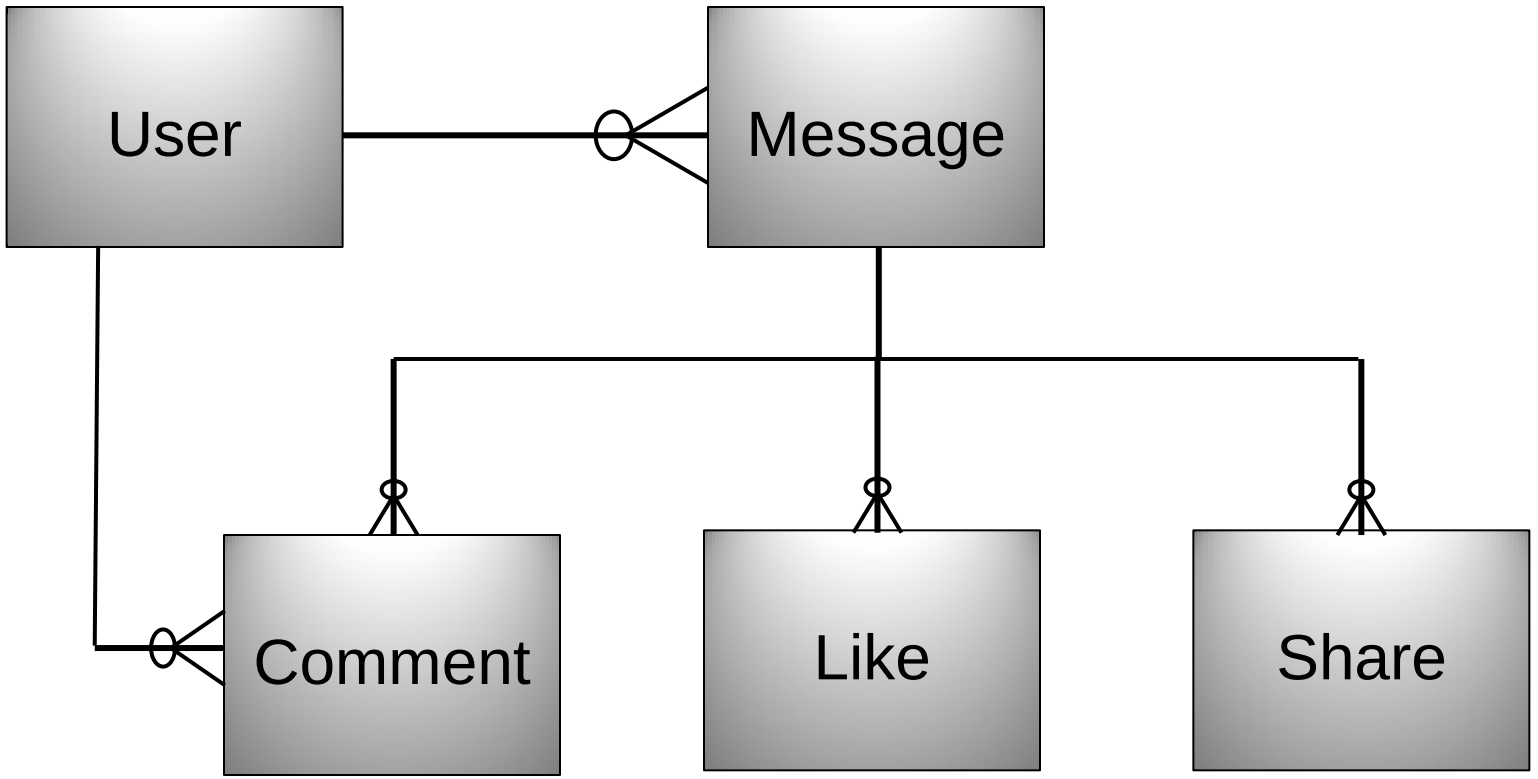
Messenger Social media Application

Design REST API

Messenger Social media Application

- Post messages
- Comment on messages
- Like and share messages
- User profile

ER Diagram



Tables and their interrelationship

For Web Application

- In case it is a web application you need a page to view a message with a query:

Get a message with ID 6

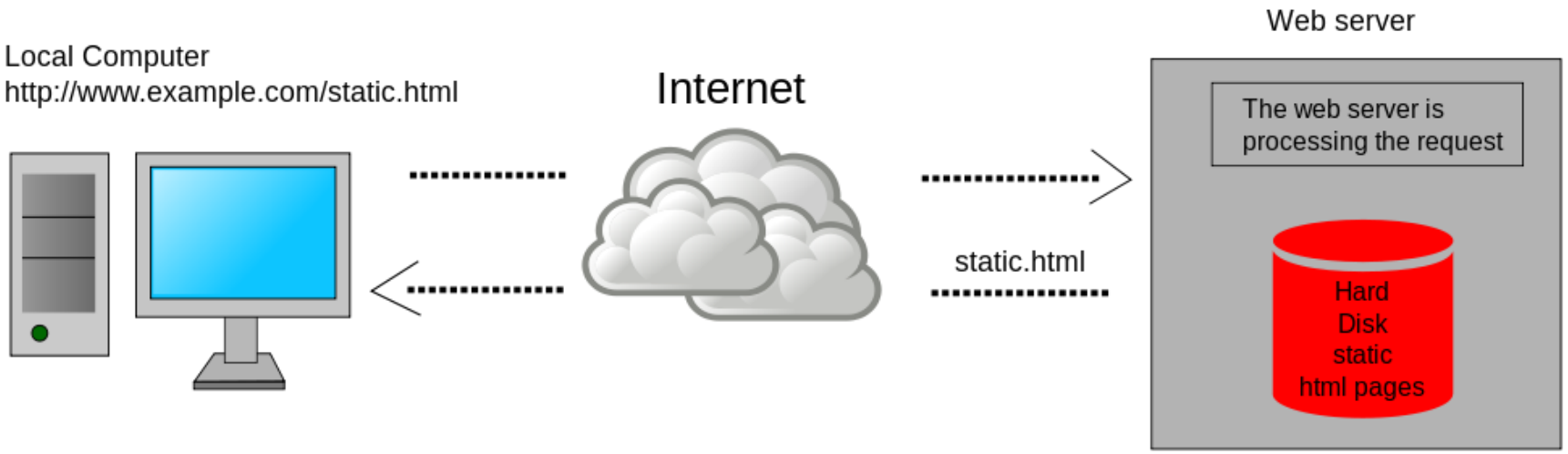
- You can use any framework to get this information using valid URI
- If user can remember URL to the home page a link can retrieve the information (user needn't have to remember it)

For REST API

- In case it is a REST application, consumer (developer) need to aware of URI:
- Need to have common URI convention for different entities!
- So what is the best practice? (nothing wrong/right)

Static Pages

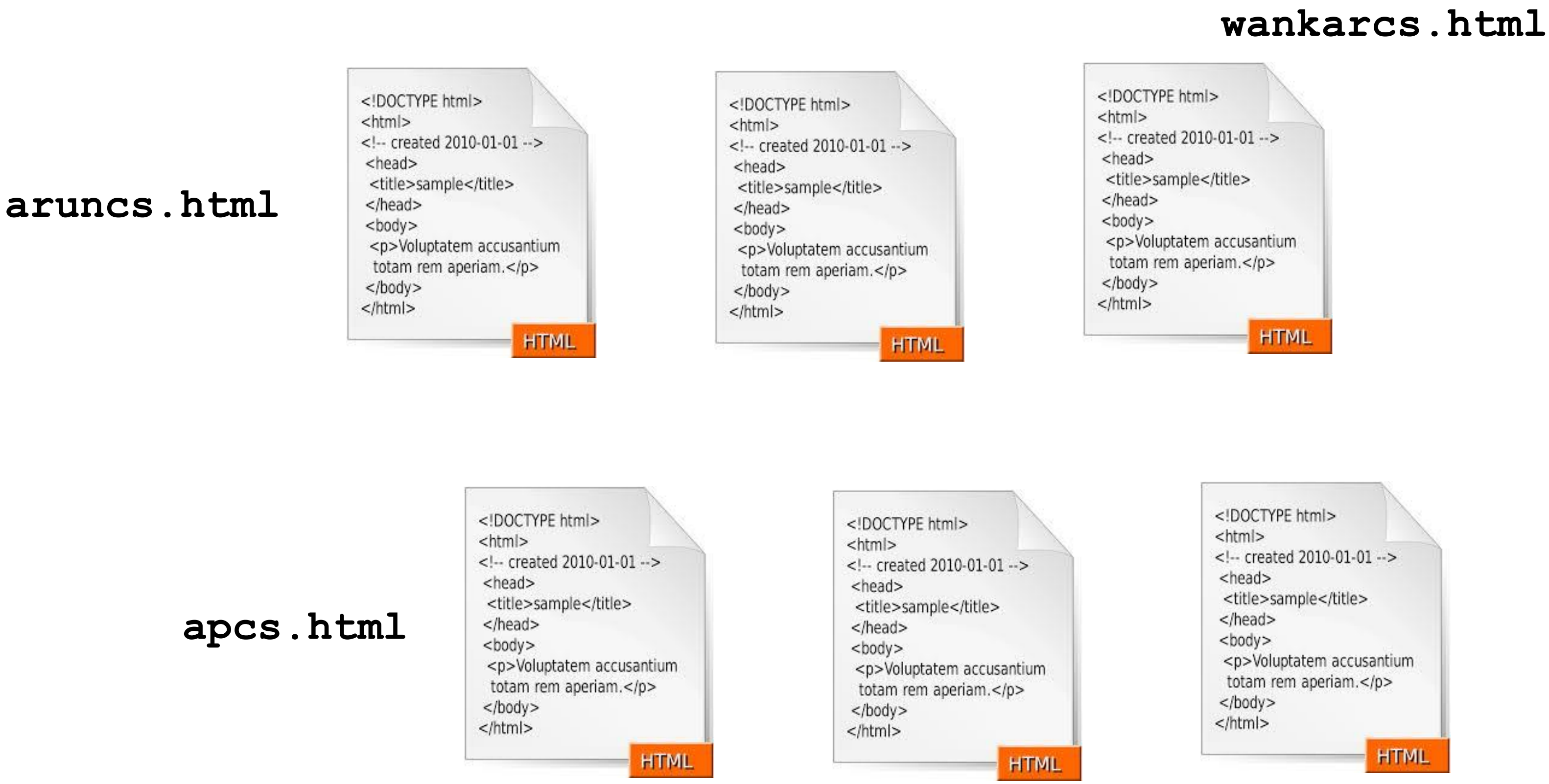
- In early days we used **static page** to access the contents
- Every page has a unique URI



Static Pages

- That’s exactly the concept used in REST
- Every entity has unique URL that is standard
- Assume that we have faculty website of SCIS
- Every faculty has their own profile

Static Pages



Static Page Profiles

- Best way to organize them is to have profile directory and store all the static html pages
- URI will be:

/profiles/wankarcs.html

RESTful URI is /profiles/wankarcs

More generic /profiles/{profileName}

- Create unique URI for resources

Resource based URI

- To our earlier message example URI will be

  /messages/{messageID}

  /messages/6

  /messages/12

  - **URI contains nouns not verbs**
  - **Resource names are all plural**

Resource based URI

- There are no verbs, or language/framework specific syntax in URI
- So if you change technology: fine it works in the same way.
- In case of the earlier example nouns are (resources)
  - users
  - likes
  - comments
  - shares

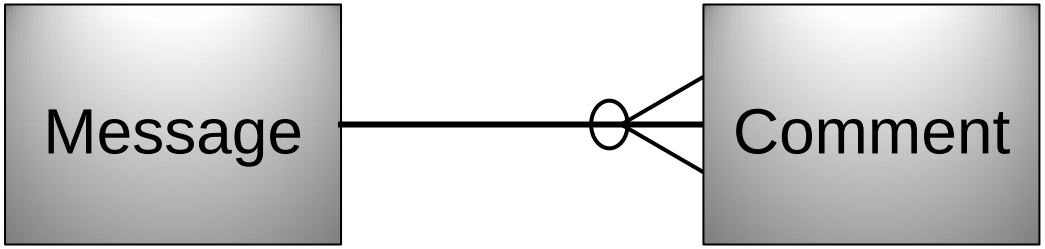
Resource based URI

- What is the resource URI for the comments for user ID 6?

  /comments/6

- What is resource relations?

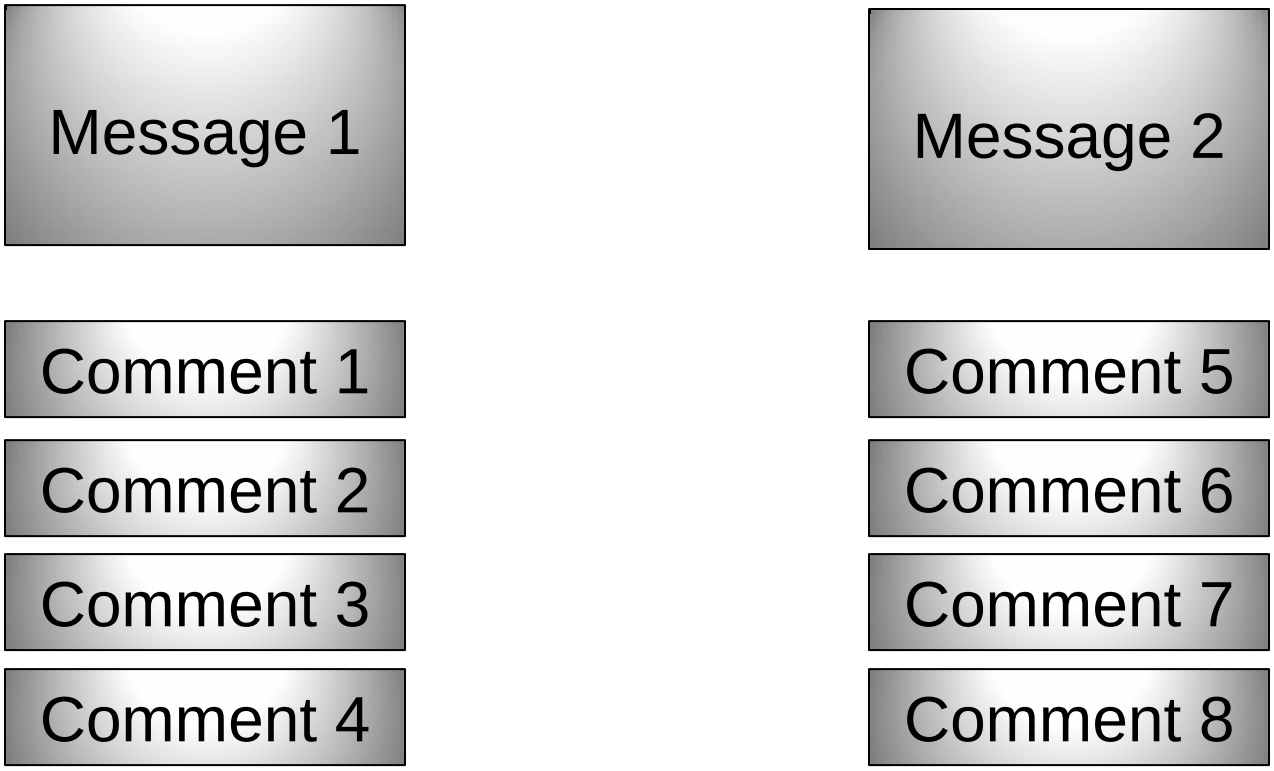
Resource Relations



/comments/{commentID}

- Design is good but it treats Message and Comment as two independent resources
- does not show the relation between the resources
- Can't have all messages in one folder!

Resource Relations



messages/1/comments/6
messages/{messageID}/comments/{commentID}

Resource Relations

/messages/{messageID}/comments/{commentID}

/messages/{messageID}/likes/{likeID}

/messages/{messageID}/shares/{shareID}

RESTful URI types

- RESTful URI belongs to **two** type Instances resource URI and Collection resource URIs

/messages/{messageID}/comments/{commentID}

/messages/{messageID}/likes/{likeID}

/messages/{messageID}/shares/{shareID}

Above are Instance Message URIs they have unique ID to identify the resource

RESTful URI types

- What if I want all messages?

/messages

/messages/2/comments

Collection Message URIs

RESTful URI types

- What if I want all comments irrespective of messages?

/messages/{messageID}/comments

/messages/2/comments

Above will not work as we need to provide messageID

/comments (works but not good)

Message filtering

- What if I want few messages?

/messages?offset=10&limit=15

starting from 10 next 15

/messages?year=2017

Operations and data

- Action based URI Vs RESTful URI

Get the information

/getMessages.do?id=10

RESTful will be /messages/10

Post the information

/postMessages.do?id=10

RESTful will be /messages/10

Operations and data

- Action based URI Vs RESTful URI

Delete the information

/deleteMessages.do?id=10

/messages/10

HTTP methods

HTTP methods

- Commonly used

GET

PUT

POST

DELETE

- Uncommon

HEAD

OPTIONS

HTTP method

- In the REST calls we have

/getMessages.do?id=10 

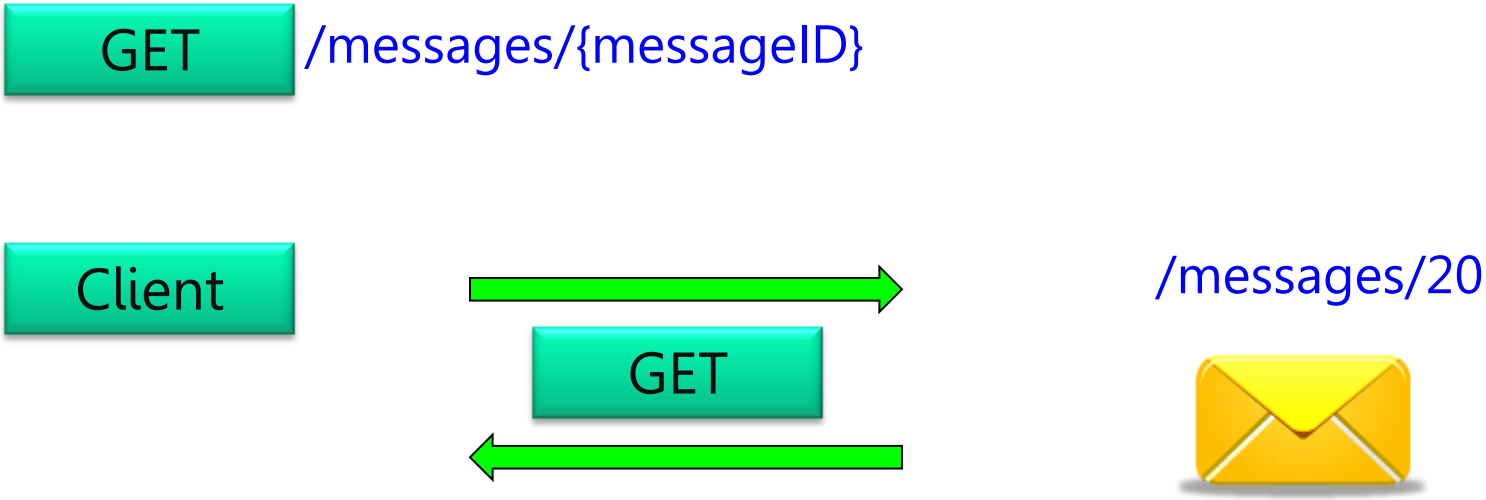
/deleteMessages.do?id=10 

GET /messages/10 

DELETE /messages/10 

Messenger

- In messenger application **getting** a message



Messenger

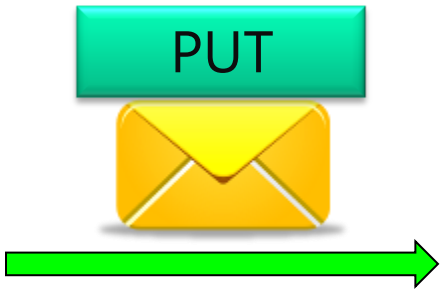
- **Updating** a message

PUT /messages/{messageID}

Content need to be send in the body of the PUT request

Client

/messages/20



messenger

- **Deleting** a message

DELETE /messages/{messageID}

Message body is not required

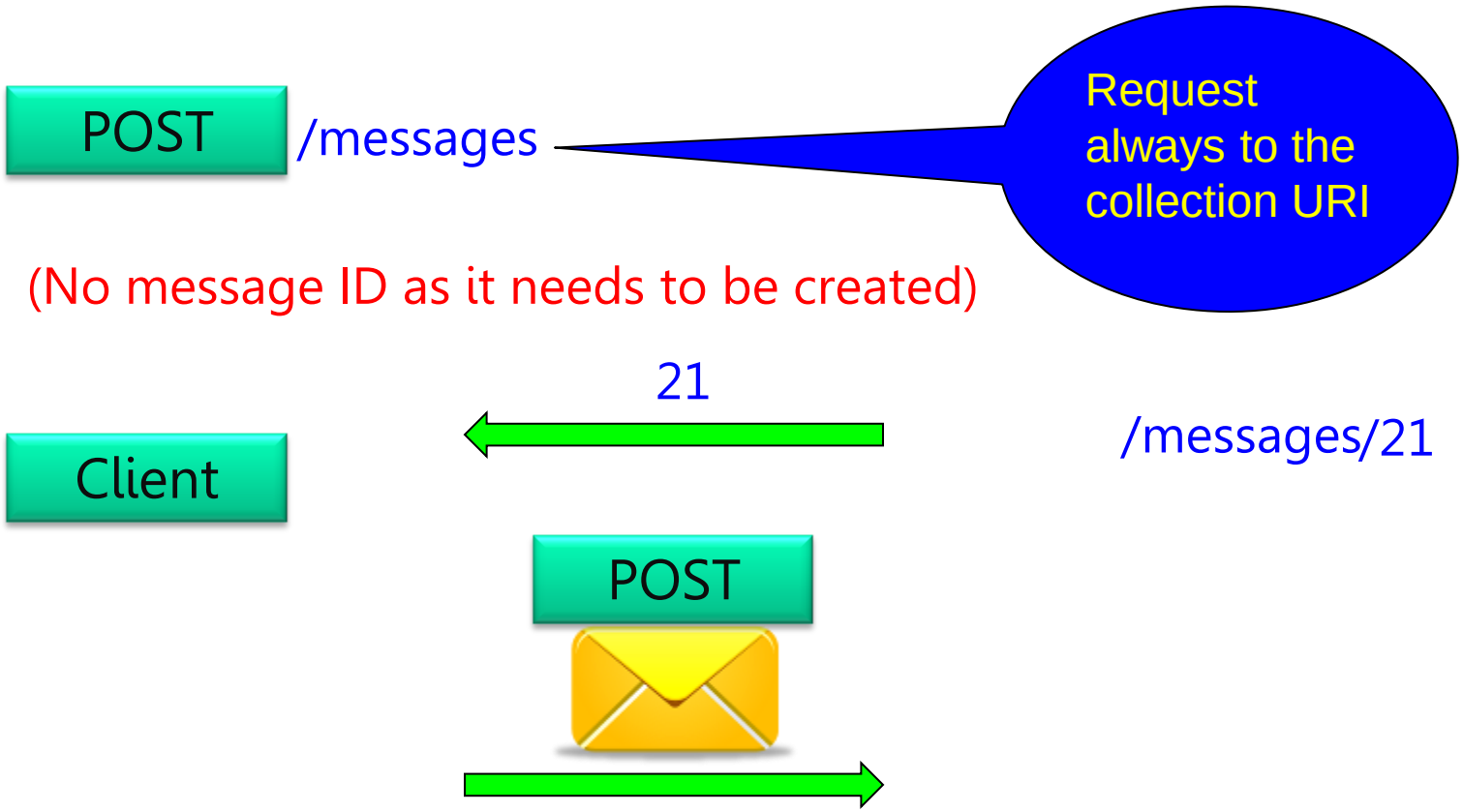
Client

DELETE

/messages/20

messenger

- **Creating** a message requires **collection URI**



Collection URI & messages

GET /messages

gets all messages

DELETE /messages/6/comments

delete all comments of message 6

GET /messages/6/comments

get all comments of message 6

Collection URI & messages

POST /messages/6/comments

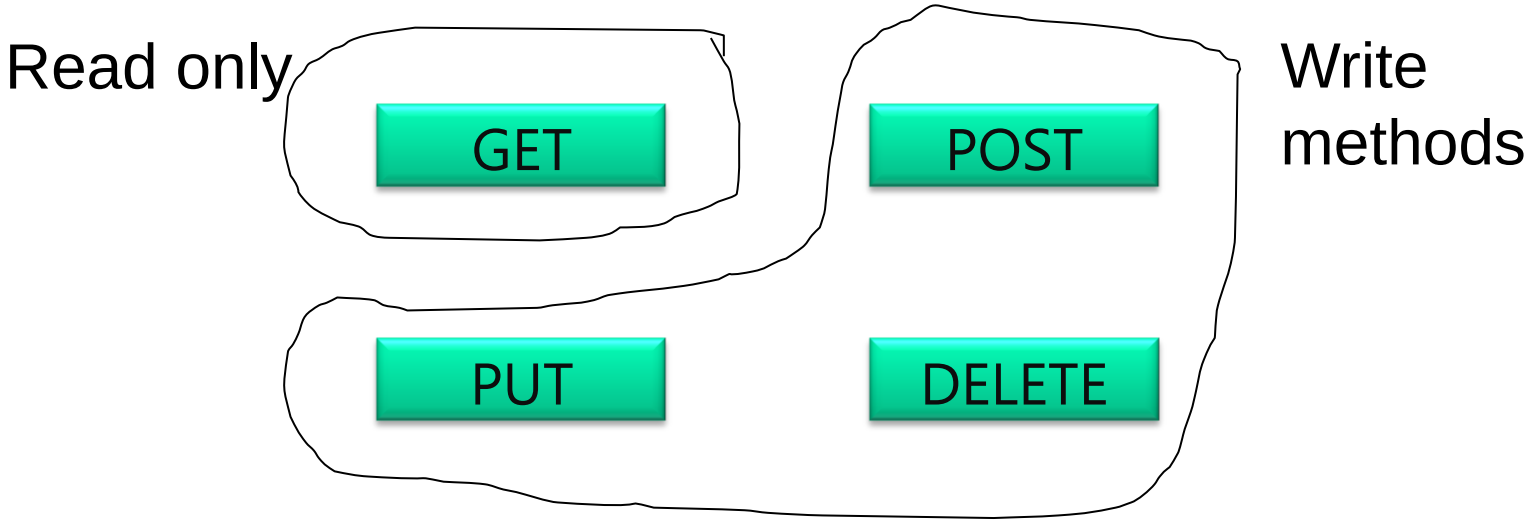
creates new comment for message 6

PUT /messages/6/comments

replace all comments for message 6 with new list

Method Idempotence

- **What is the difference between PUT and POST?**
- Common HTTP methods are



Method Idempotence

- Safe to make multiple GET request to a server.
- What about PUT, POST and DETELE?

Method Idempotence

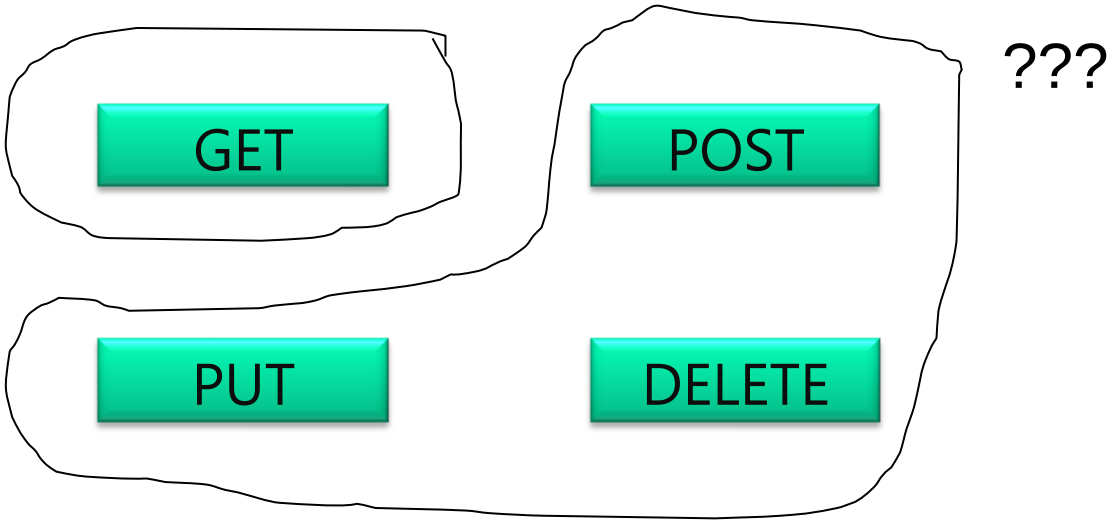
- p = 200;
- p = 200;
- p = 200;
- p = 200;

Repeatable operation Vs

non-repeatable operation

Method Idempotence

repeatable



Method Idempotence

- **Deleting** a message

DELETE /messages/{messageID}

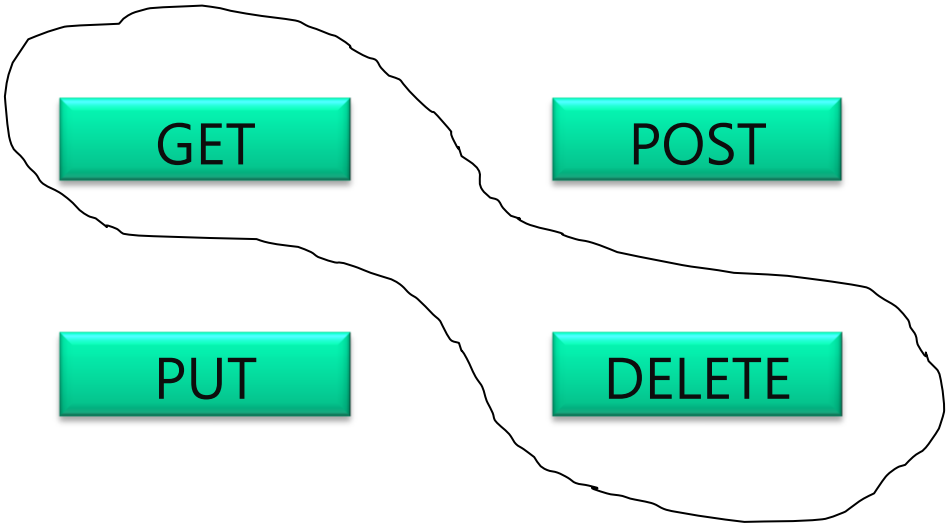
Client

DELETE

/messages/20

Method Idempotence

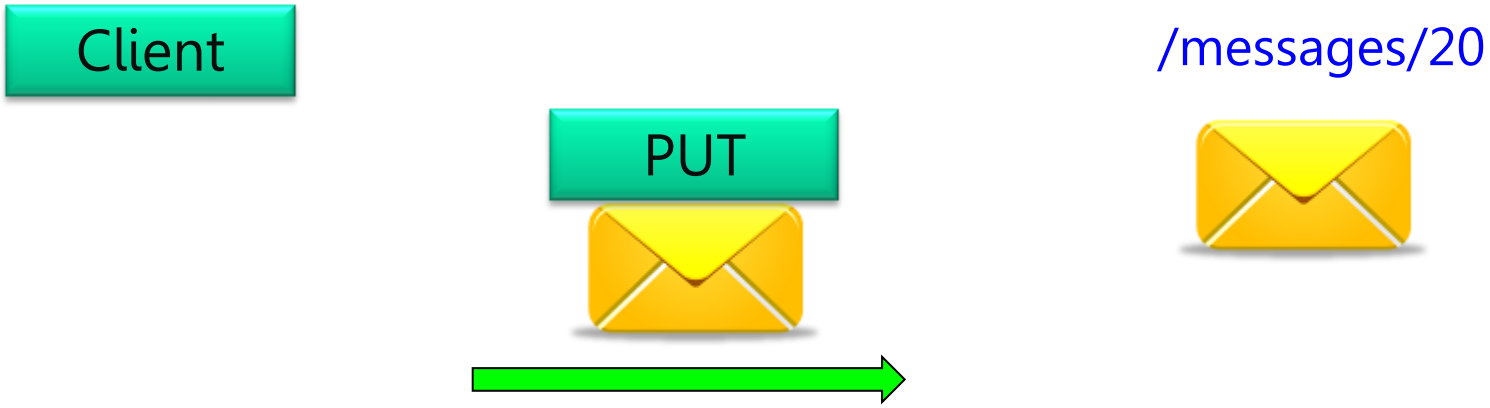
repeatable



???

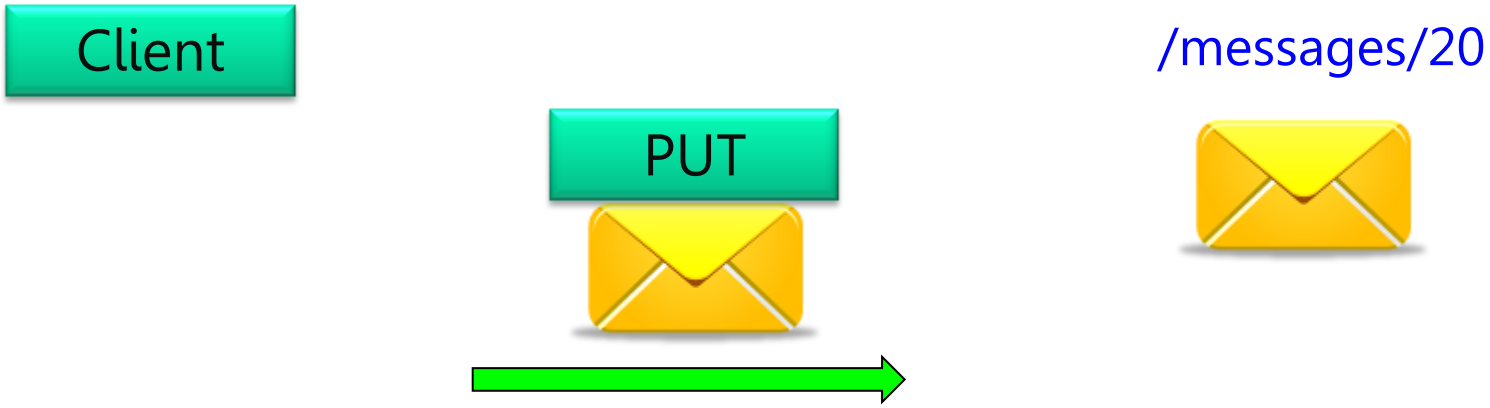
Method Idempotence

- **PUT**



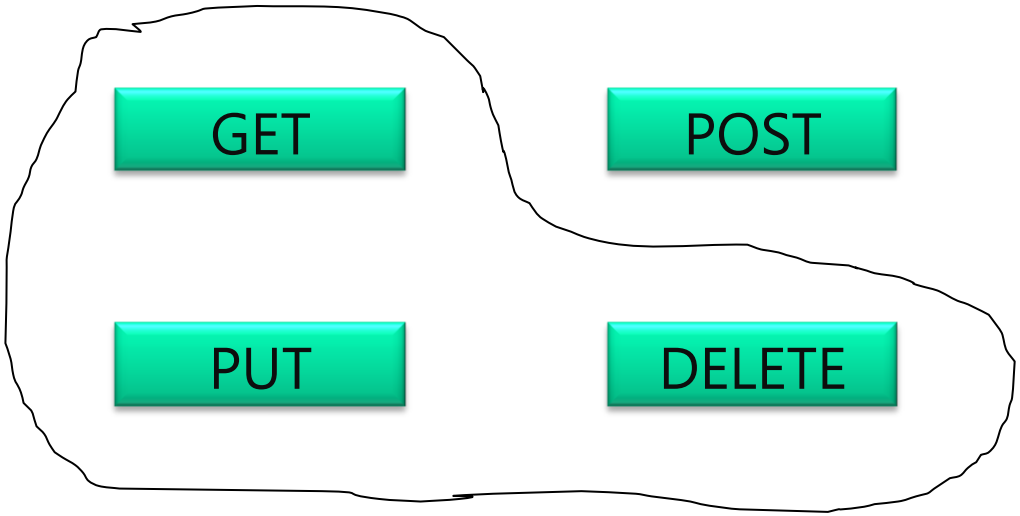
Method Idempotence

- **PUT (once more)**



Method Idempotence

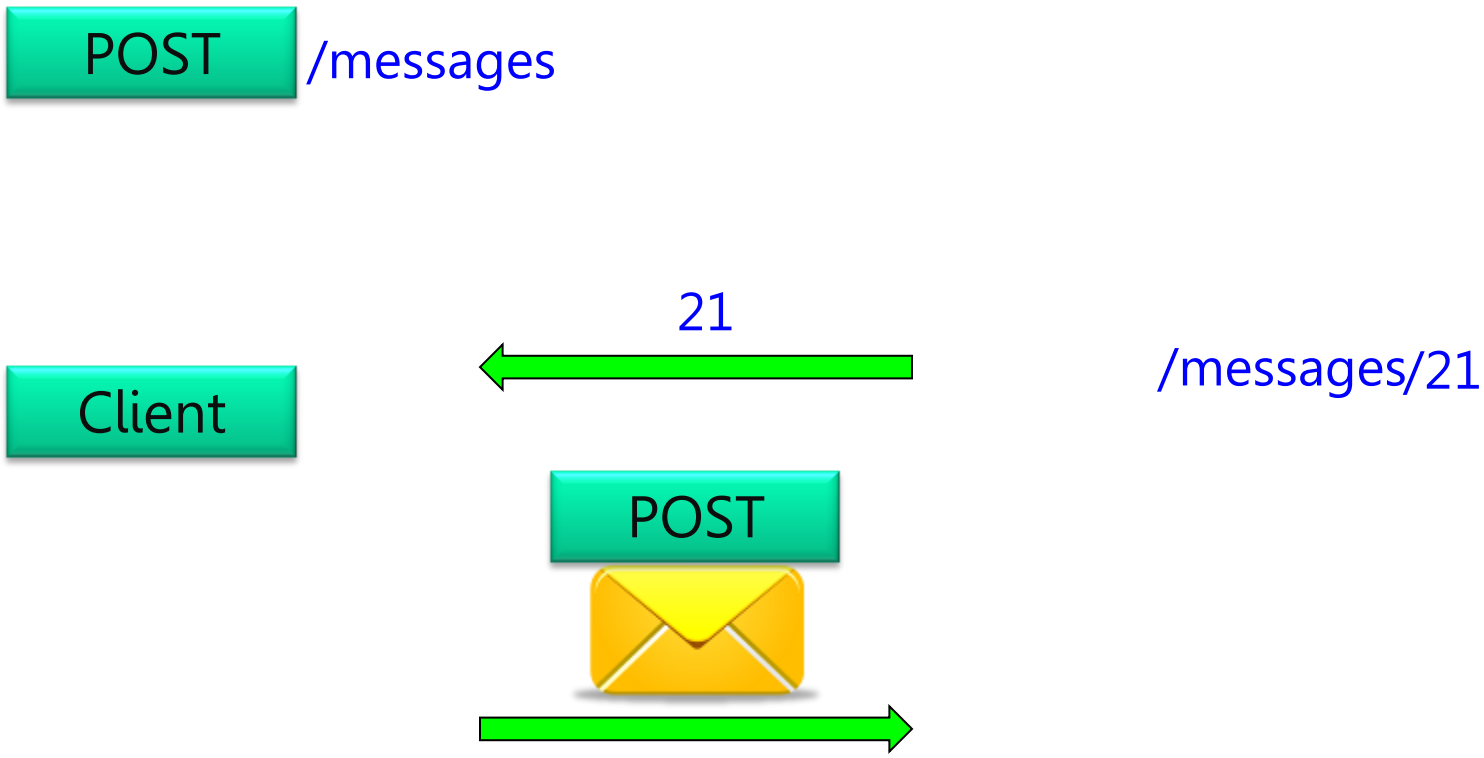
repeatable



???

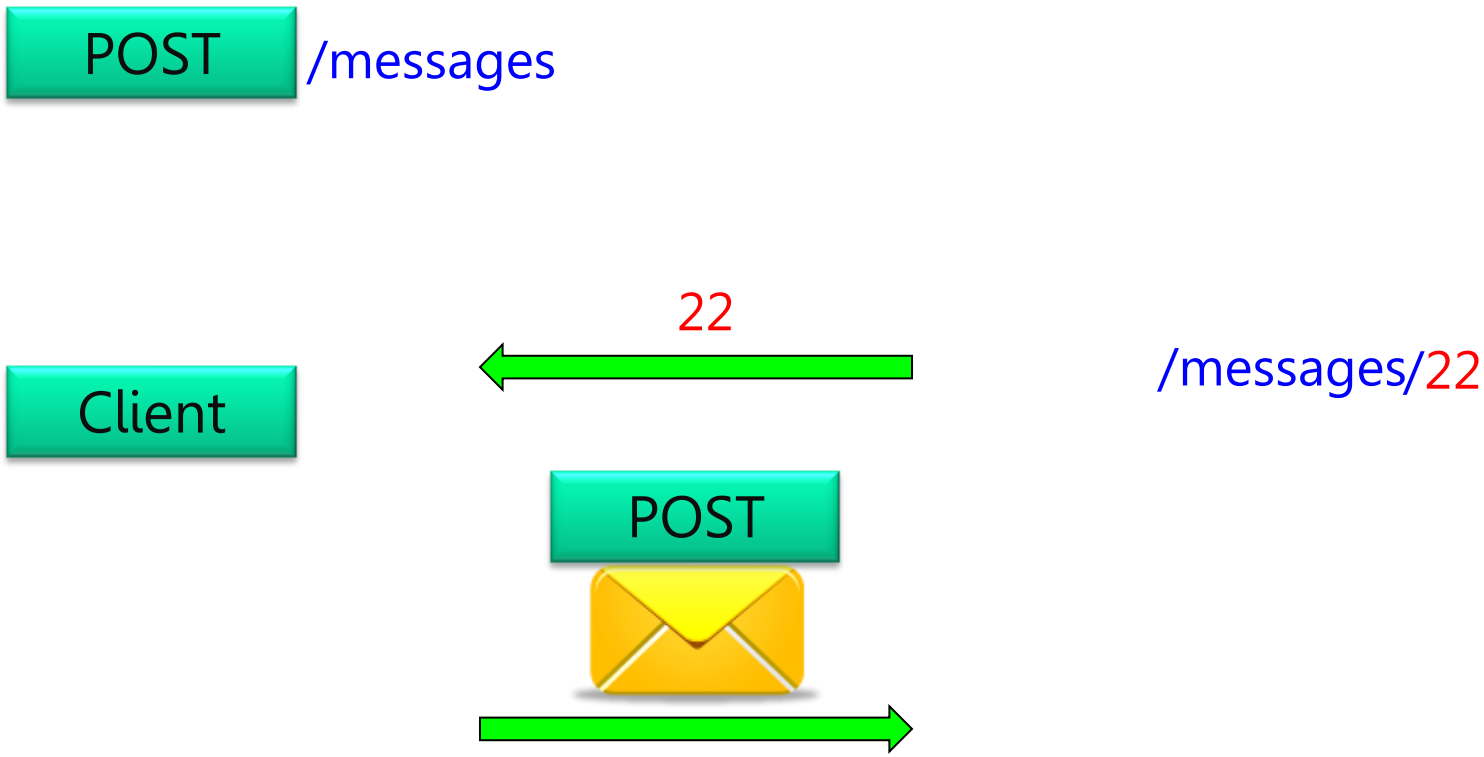
Method Idempotence

- **POST**



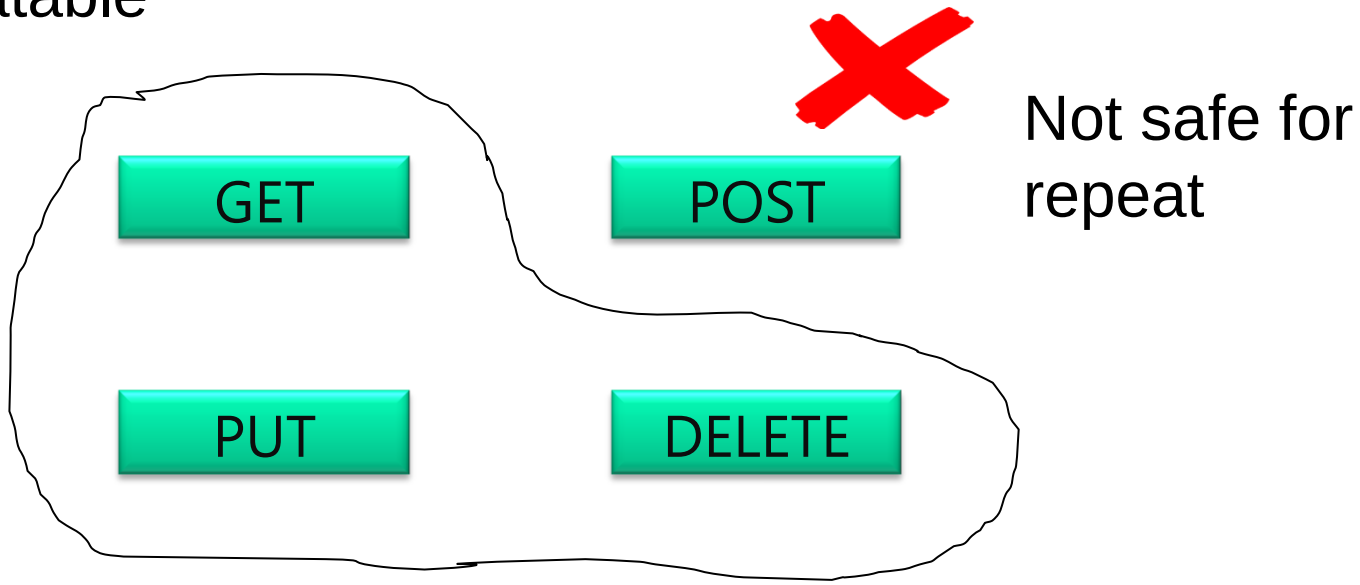
Method Idempotence

- **POST (once again)**



Method Idempotence

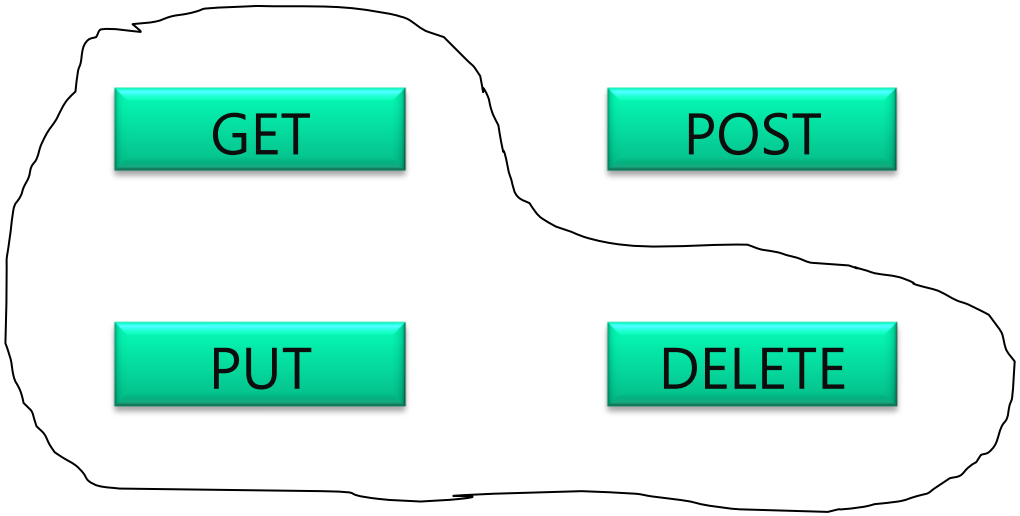
repeatable



Not safe for repeat

Method Idempotence

Idempotent



Non Idempotent

Wiki Definition

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

Wiki Definition

HTTP Method	Idempotent	Safe
OPTIONS	YES	YES
GET	YES	YES
HEAD	YES	YES
PUT	YES	NO
POST	NO	NO
DELETE	YES	NO
PATCH	NO	NO

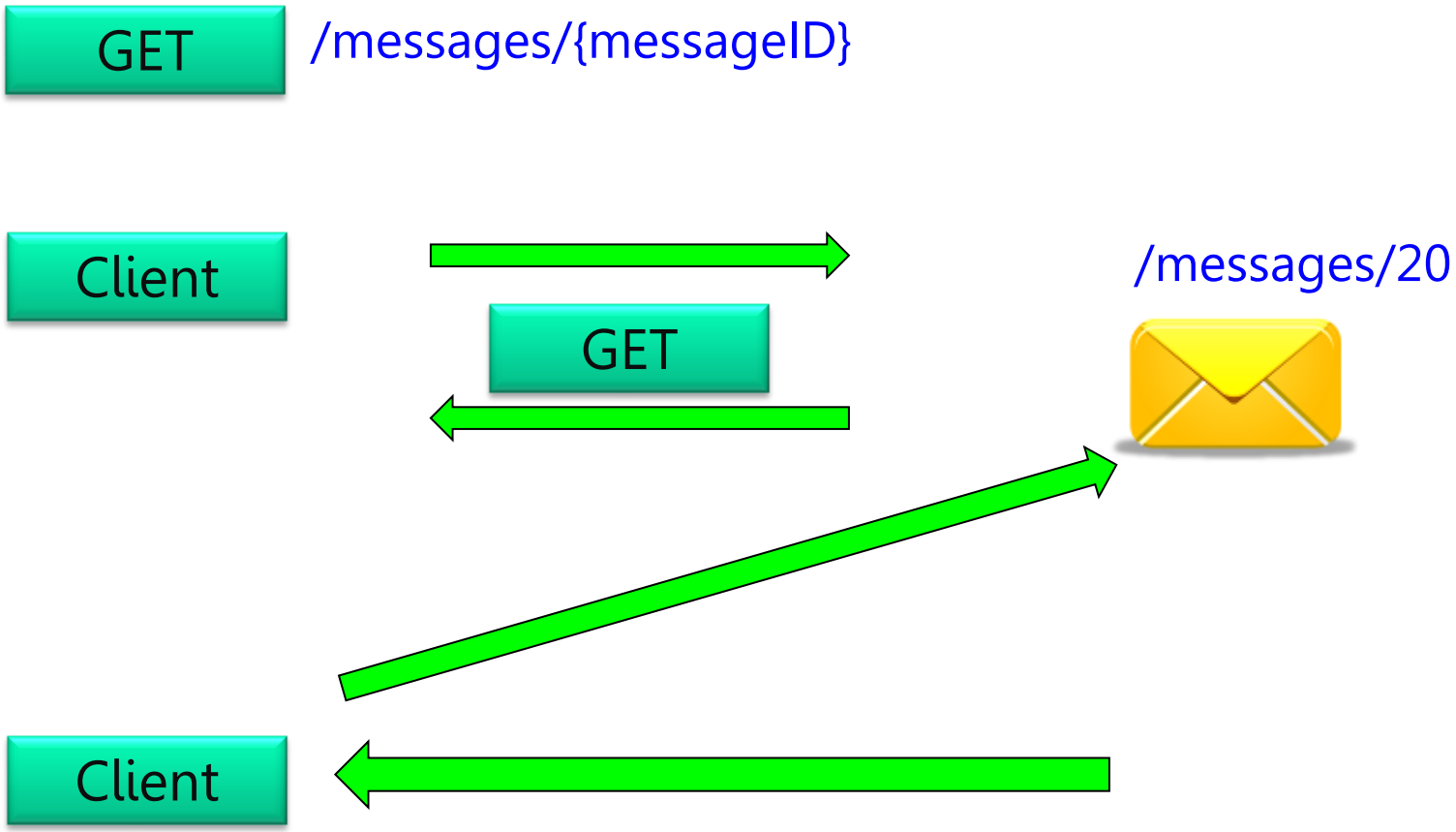
Other HTTP methods

- The HEAD method asks for a response identical to that of a GET request, but without the response body
- The PATCH method is used to apply partial modifications to a resource.
- The CONNECT method establishes a tunnel to the server identified by the target resource.

POST vs PUT

- **Resource creation** request should be always be **POST**
- **Update request** should use **PUT** method

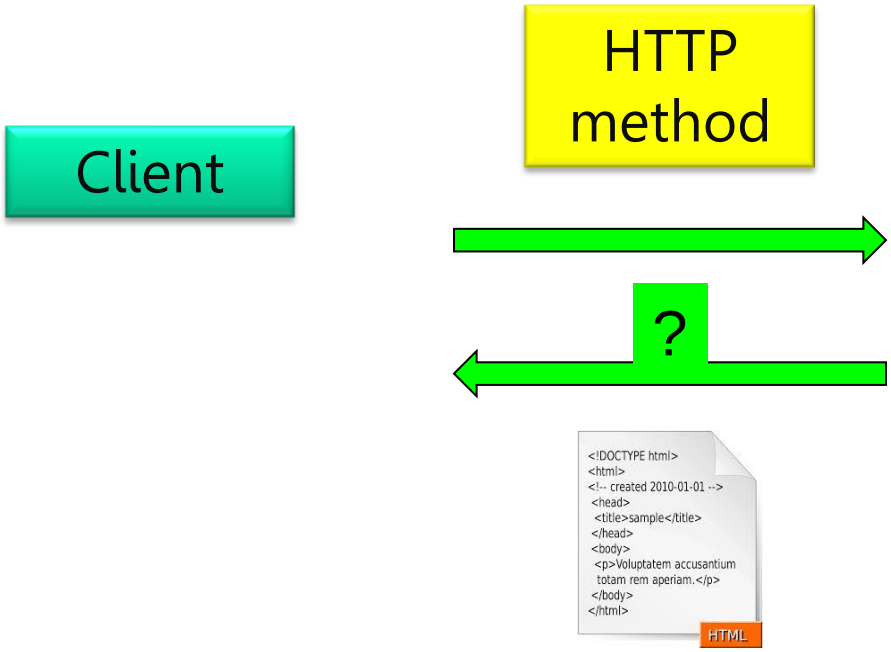
CACHing GET responses



Refresh Brower effect

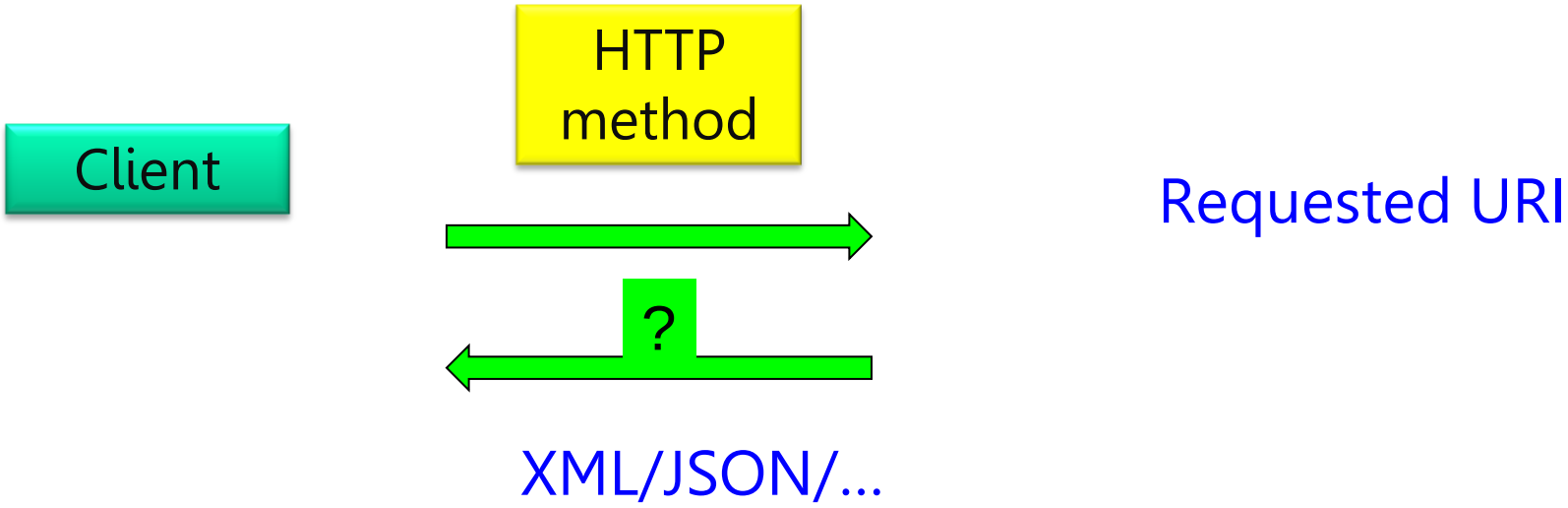
- F5 results in performing last HTTP request
- If **Idempotent** no issues
- If **non Idempotent ??** Like submitting a form
- Brower should ask you before making changes (safeguard)

REST responses



For a web application the response is html page

REST responses



JSON is gaining popularity as the client is a browser and mostly runs JavaScript code, it is handy if response is in JSON

Message Entity class & JSON response

```
public class MessageEntity {
        private long id;
        private string message;
        private string name;
        private date datecreation;
}
```

```
{
        "id":10,
        "message": "Cloud Computing",
        "name": "Rajeev",
        "datecreation": "2018-03-18T20:07:17.123"
}
```

Message Entity & XML response

```
public class MessageEntity {
        private long id;
        private string message;
        private string name;
        private date datecreation;
}
```

```
<messageEntity>
   <id>10</id>
   <message>Cloud Computing</message>
   <name>Rajeev</name>
   <datecreation>2018-03-18T20:07:17.123</datecreation>
</messageEntity>
```

Responses to the same resource

Message id 10

```
{
        "id":10,
        "message": "Cloud Computing",
        "name": "Rajeev",
        "datecreation": "2018-03-18T20:07:17.123"
}
```

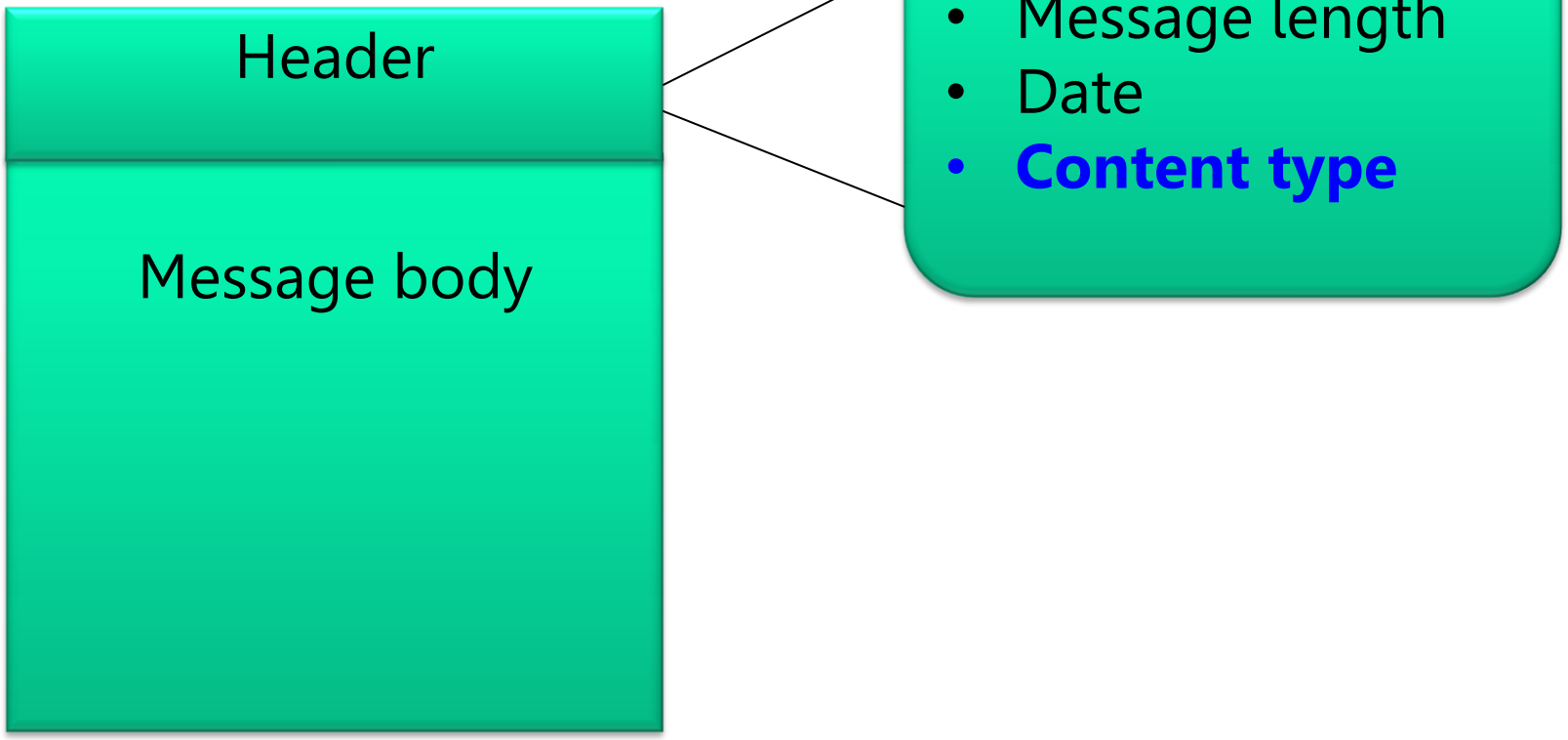
```
<messageEntity>
   <id>10</id>
   <message>Cloud Computing</message>
   <name>Rajeev</name>
   <datecreation>2018-03-18T20:07:17.123</datecreation>
</messageEntity>
```

Transferring the Representational State

Responses to the same resource

- How does client know what format the response is in?
- As a client one can ask for the particular format but there is no guarantee
- Using HTTP header
- Header/body
- Header contains the metadata
- Content type response (JSON/XML/…)

Parses based on content type



Status Codes

- When the client is not human then error code needs to be returned
- Done using status code
- **HTTP specification requires that the first line of the response as “status code”**
- **200 OK**
- **404 Not Found**

Status Codes

- Starts from 100 and go till 599
- **Not all are valid codes**
- There are **five classes** of codes.
- **1XX** **informational** (ex. Ack. responses)
- **2XX** **Success**
  - 200 success OK
  - 201 successful resource creation, use with POST
  - 204 No content, use with DELETE

Status Codes

- **3XX Redirection**
- Server asks client to do further action to complete the request
  - 302 found
  - 307 temporary redirect (don't ask me, ask other URL)
  - 304 not modified (error code)

Status Codes

- **4XX Client error**
- Error in the request
  - 400 bad request
  - 401 unauthorized
  - 403 Forbidden
  - 404 not found
  - 415 unsupported media type

Status Codes

- **5XX Server error**
- Error in the request
  - 500 internal server error (with details)
  - Ex. Run time exception

Common HTTP code and Operations

Common HTTP status codes

MuleSoft

Code	Definition	Returned by
200	OK – The request succeeded.	GET, DELETE, PATCH, PUT
201	Created – A new resource or object in a collection.	POST
304	Not modified – Nothing was modified by the request.	PATCH, PUT
400	Bad request – The request could not be performed by the server due to bad syntax or other reason in request.	All
401	Unauthorized – Authorization credentials are required or user does not have access to the resource/method they are requesting.	All
404	Resource not found – The URI is not recognized by the server.	All
500	Server error – Generic something went wrong on the server side.	All

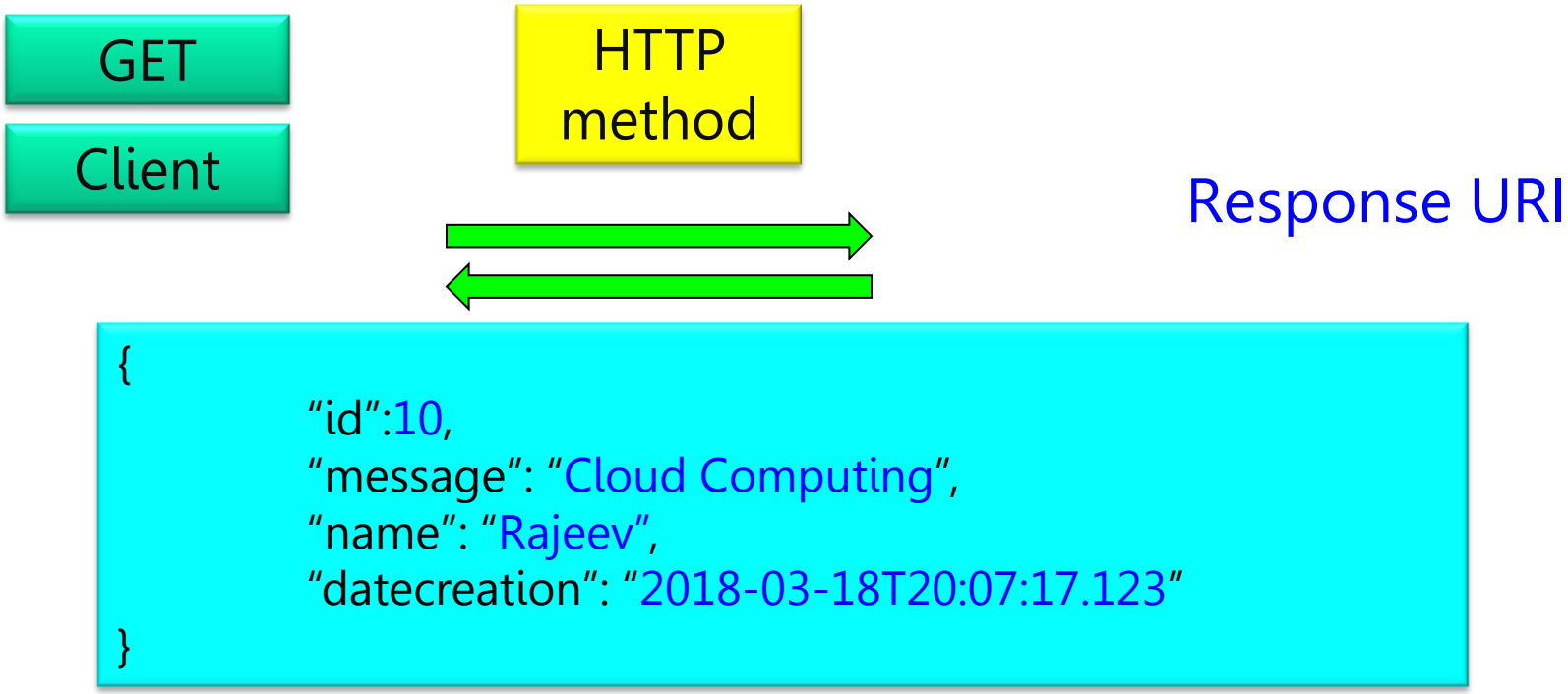
H A T E O A S

- **H**ypermedia
- **A**s
- **T**he
- **E**ngine
- **O**f
- **A**pplication
- **S**tate

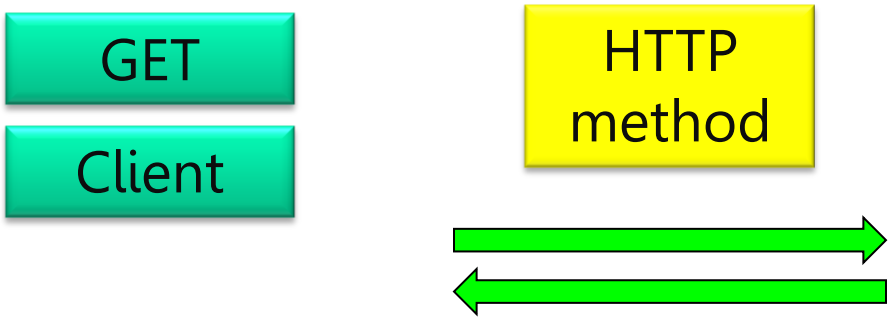
HTTP links

- There is not service definition for REST
- Best REST don’t need documentation
- Ex. To visit a website do you need documentation?
- Click link and we navigate
- Advantage of using HTTP
- Hyperlink allow you to navigate
- **What if we use the same concept here?**

HATEOAS


GET
Client
HTTP method
Response URI
{
"id":10,
"message": "Cloud Computing",
"name": "Rajeev",
"datecreation": "2018-03-18T20:07:17.123"
}

HATEOAS




```
{
        "id":10,
        "message": "Cloud Computing",
        "name": "Rajeev",
        "datecreation": "2018-03-18T20:07:17.123"
        "commentsUri": "api/messages/6/comments",
        "linksUri": "api/messages/6/likes",
        "sharesUri": "api/messages/6/shares",
        "authorProfileUrl": "api/profiles/Rajeev"
}
```

HATEOAS

- Additional information sent by the server to the client for further use (assistance) same as the hypertext.
- Clients don’t have to hardcode these URI
- Driver or the engine of the application state.

H A T E O A S

- **H**ypermedia
- **A**s
- **T**he
- **E**ngine
- **O**f
- **A**pplication
- **S**tate

Message Entity class

```
public class MessageEntity {
        private long id;
        private string message;
        private string name;
        private date datecreation;
}
```

Best Practices

```
{
   "id":10,
   "message": "Cloud Computing",
   "name": "Rajeev",
   "datecreation": "2018-03-18T20:07:17.123",
},

{
   "id":11,
   "message": "Hi there",
   "name": "Argha",
   "datecreation": "2018-03-18T20:07:17.123",
},
{
   "id":12,
   "message": "I am on leave",
   "name": "Kiran",
   "datecreation": "2018-03-18T20:07:17.123",
}
```

Best Practices


```
{
   "id":10,
   "message": "Cloud Computing",
   "name": "Rajeev",
   "datecreation": "2018-03-18T20:07:17.123",
},
```

```
{
   "id":11,
   "message": "Hi there",
   "name": "Argha",
   "datecreation": "2018-03-18T20:07:17.123",
},
{
   "id":12,
   "message": "I am on leave",
   "name": "Kiran",
   "datecreation": "2018-03-18T20:07:17.123",
},
```

Best Practices

```
{
   "id":10,
   "message": "Cloud Computing",
   "name": "Rajeev",
   "datecreation": "2018-03-18T20:07:17.123",
},

{
   "id":11,
   "message": "Hi there",
   "name": "Argha",
   "datecreation": "2018-0
},
{
   "id":11,
   "message": "I am on leave",
   "name": "Kiran",
   "datecreation": "2018-03-18T20:07:17.123",
}
```

```
{
   "id":      + "10"
      "/message/"
                    oud Computing",
                    v",
   "datecreation": "2018-03-18T20:07:17.123",
}
```

Message resource ID

Best Practices


```
{
  "id":10,
  "message": "Cloud Computing",
  "name": "Rajeev",
  "datecreation": "2018-03-18T20:07:17.123",
  "href":"/messages/10",
  "comments-href": "/messages/6/comments",
  "links-href": "/messages/10/likes",
  "shares-href": "/messages/10/shares",
  "authorProfile-href": "/profiles/Rajeev",
  "comment-post-href":"/message/10/comments"
}
```


- If we add **href**, it becomes handy for the client

Best Practices


```
{
  "id":10,
  "message": "Cloud Computing",
  "name": "Rajeev",
  "datecreation": "2018-03-18T20:07:17.123"
  "href":"/messages/10",
  "comments-href": "/messages/6/comments",
  "links-href": "/messages/10/likes",
  "shares-href": "/messages/10/shares",
  "authorProfile-href": "/profiles/Rajeev",
  "comment-post-href":"/message/10/comments",
}
```


- **BUT!!!** It becomes messy, client needs to remember many things

"rel" attribute

```
{
   "id":10,
   "message": "Cloud Computing",
   "name": "Rajeev",
   "datecreation": "2018-03-18T20:07:17.123",
   "links": [
                {
                "href": "/messages/1",
                "rel"   : "self"
                }
               ]
}
```

It shows the relationship between the current document and the link document

“rel” attribute

```
{
  “id”:10,
  “message”: “Cloud Computing”,
  “name”: “Rajeev”,
  “datecreation”: “2018-03-18T20:07:17.123”
  “links”: [
            {
              “href”: ”/messages/1”,
              “rel”   : “self”
            },
           {
              “href”: ”/messages/1/comments”,
              “rel”   : “comments”
            },
            :
            :
          ]
}
```

The Richardson Maturity Model

- When we design API for the consumption, can we say it is fully RESTful?
- Leonard Richardson suggested a model known as "**The Richardson Maturity Model**"

The Richardson Maturity Model

- Every API belongs to one of these levels

- **Level 3**
- **Level 2**
- **Level 1**
- **Level 0**

The Richardson Maturity Model

- **Level 0**

- One URI (endpoint where service exposed)
- Message body contains the all information details, in SOAP
- Swamp of POX (Plain Old XML)
- No use of HTTP construct in the SOAP

The Richardson Maturity Model

- **Level 1 (Starting level)**

- Individual URI for each resource
- Ex. Message/Comment/Profile have individual URI

The Richardson Maturity Model

- **Level 2 (Starting level)**

- Use right HTTP method with status code

The Richardson Maturity Model

- **Level 3 (Starting level)**

- Use HATEOAS (**H**ypermedia **A**s **T**he **E**ngine
- **O**f **A**pplication **S**tate)

- Response have the links that the client can use