EVOLUTION OF APIs

# XML-RPC

* XML-RPC 🡪 Extensive Markup Language – Remote Procedure Calls
* 98% deprecated/outdated
* 2% usage

# SOAP API

* SOAP 🡪 Simple Object Access Protocol
* 90% deprecated/outdated
* 10% usage

# REST API

* REST 🡪 Representational State Transfer
* 100% usage

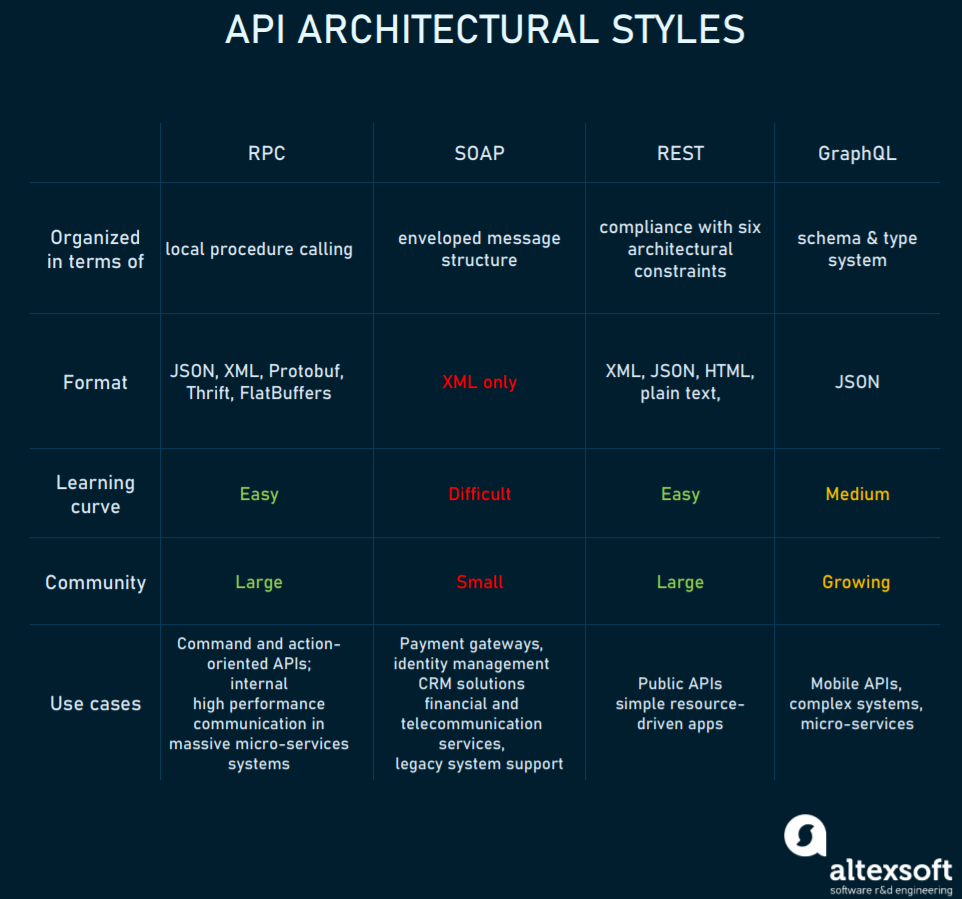
SOAP API vs REST API

# SOAP API

* SOAP 🡪 Simple Object Access Protocol
* SOAP is a protocol
* HTTP Methods that can be used for developing SOAP APIs is only POST method
* The request data/payload from client to server has to be in XML format/language only
* The response data from server to client has to be in XML format/language only
* SOAP often is termed as a rigid way of developing APIs i.e., SOAP APIs development is not a flexible option

# REST API

* REST 🡪 Representational State Transfer
* REST or RESTful is not a protocol
* REST is an architectural style
* HTTP methods that can be used for developing REST APIs include GET, POST, PUT, PATCH, DELETE etc
* The request data/payload from client to server can be in any format [XML, JSON, TEXT, HTML, JSON SCRIPT, YAML etc]
* The response data from server to client can be in any format [XML, JSON, TEXT, HTML, JSON SCRIPT, YAML etc]
* REST APIs are often termed as flexible way of developing APIs
* REST APIs are initially developed at Google Inc



API SYNTAX

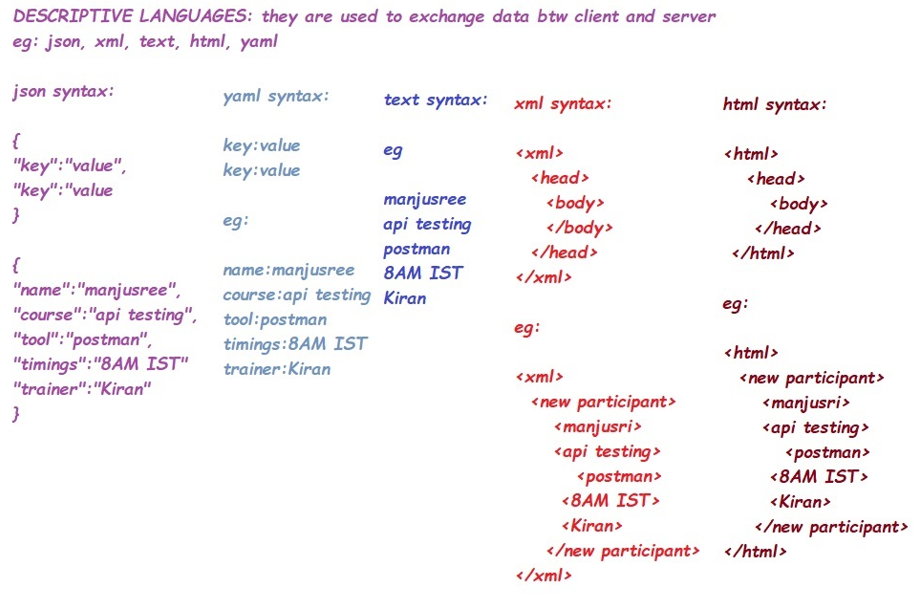
# SYNTAX

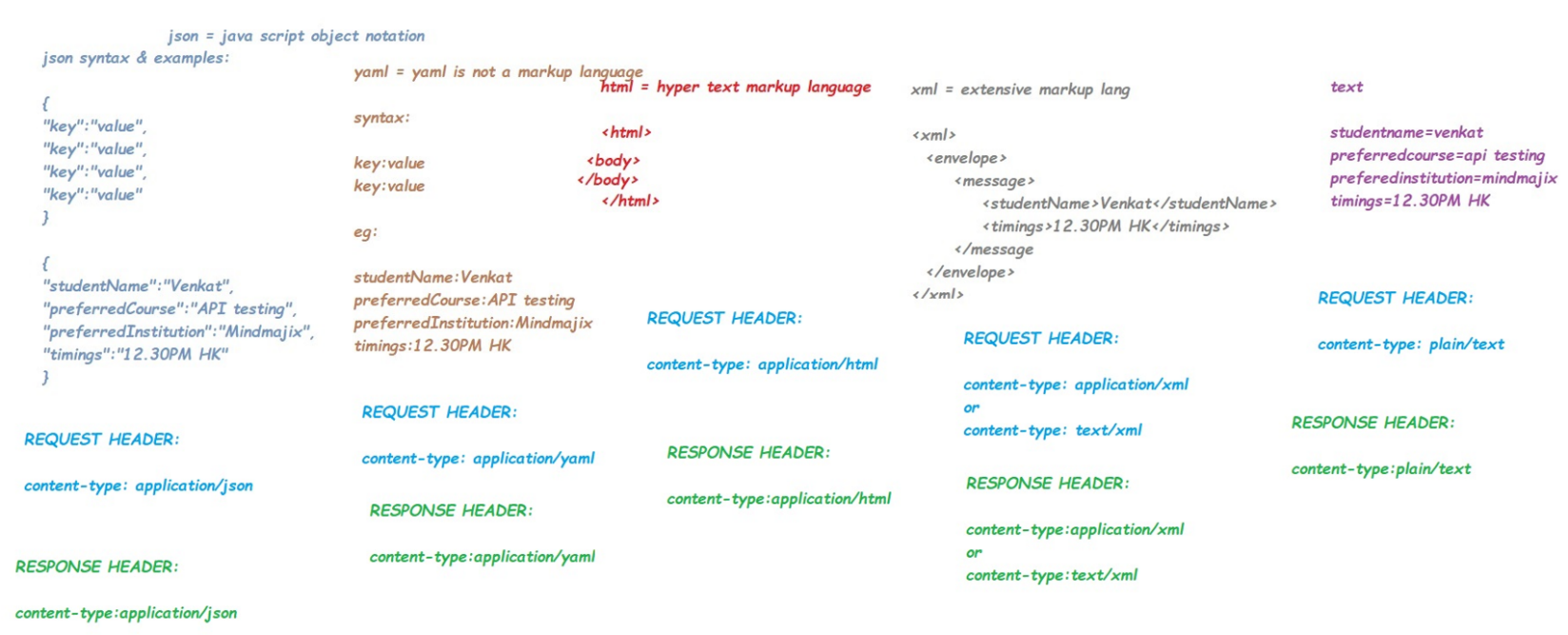
* API looks like an URL
* The API syntax comprises of the below fields:
  + HTTP METHOD [mandatory]
  + PROTOCOL [mandatory]
  + BASE URL/DOMAIN NAME/SERVER NAME/IP ADDRESS [mandatory]
  + END POINT/RESOURCE PATH [optional]
  + QUERY PARAMETER(S) [optional]
  + REQUEST BODY/PAYLOAD [not needed for GET method ; mandatory for POST, PUT, PATCH, optional for DELETE]
  + AUTHENTICATION [optional]
  + REQUEST HEADERS [mandatory]
  + REQUEST COOKIES [optional]
* The syntax of an API is

***httpMethod-protocol-baseURL-endPoint-QueryParam-RequestBody-RequestHeaders-Authentication-Cookies***

* HTTP METHOD 🡪 This can be GET, POST, PUT, PATCH, DELETE etc
* PROTOCOL 🡪 This can be HTTP or HTTPS
* BASE URL 🡪 This is the domain name or the IP address of the server/host
* END POINT 🡪 This is the resource we wanted to access on the server/host
* QUERY PARAMETERS 🡪 This is the params/strings we wanted to query/search on the host & end point location
* REQUEST BODY/PAYLOAD 🡪 This is the request data supplied from the client to the server, to create new data OR update existing data on the server
* AUTHENTICATION 🡪 This is the authorization process needed to execute the API at the server
  + APIs are of 2 types:
    - Public 🡪 These are un-secured APIs by not having any authorization process i.e., any person can have access to these APIs
    - Private 🡪 These are protected/secured APIs by leveraging the concept of authorization by the developers i.e., only restricted people have access to these APIs
* REQUEST HEADERS 🡪 This is the additional meta-data sent by Client to Server to reveal more details about the API request
* REQUEST COOKIES 🡪 This are the cookies sent by Client to Server to reveal the preferences/settings of the client, so that Server can acknowledge the request and process the request quickly

DESCRIPTIVE LANGUAGE-EXAMPLES SYNTAX





API EXAMPLES – SAMPLE DOCUMENTATION

# EXAMPLE#1

GET <https://www.mindmajix.com> // this API will fetch existing content of mindmajix website

* METHOD = GET
* PROTOCOL = HTTPS
* BASE URL = [www.mindmajix.com](http://www.mindmajix.com)
* The response code for success scenario will be 2XX
* The response code for failure scenarios will 4xx OR 5xx

Let us write few simple test cases for this API:

TC#1: Use GET Method and send the API using postman tool 🡪 ER: 200 OK

TC#2: Use POST Method and send the API using postman tool 🡪 ER: 405 METHOD NOT ALLOWED

TC#3 Use PUT Method and send the API using postman tool 🡪 ER: 405 METHOD NOT ALLOWED

TC#4: Use DELETE Method and send the API using postman tool🡪 ER: 405 METHOD NOT ALLOWED

TC#5: Use HTTP protocol and send the API using postman tool 🡪 ER: 400 BAD SYNTAX

TC#6: Use HTTPS protocol and send the API using postman tool 🡪 ER: 200 OK

# EXAMPLE#2

GET [https://www.mindmajix.com/all-courses //](https://www.mindmajix.com/all-courses%20//) this API will fetch all courses on mindmajix website

* METHOD = GET
* PROTOCOL = HTTPS
* BASE URL = [www.mindmajix.com](http://www.mindmajix.com)
* END POINT = /all-courses
* The response code for success scenario will be 2XX
* The response code for failure scenarios will 4xx OR 5xx

# EXAMPLE#3

GET [https://www.mindmajix.com/all-courses/api-testing-postman //](https://www.mindmajix.com/all-courses/api-testing-postman%20//) this API will fetch api-testing-postman course content details on mindmajix website

* METHOD = GET
* PROTOCOL = HTTPS
* BASE URL = [www.mindmajix.com](http://www.mindmajix.com)
* END POINT = /all-courses/api-testing-postman
* The response code for success scenario will be 2XX
* The response code for failure scenarios will 4xx OR 5xx

# EXAMPLE#4

GET [https://www.mindmajix.com/all-courses?courseName=apiTestingPostman //](https://www.mindmajix.com/all-courses?courseName=apiTestingPostman%20%20%20//) this API will fetch api-testing-postman course content details on mindmajix website, by using query parameters

* METHOD = GET
* PROTOCOL = HTTPS
* BASE URL = [www.mindmajix.com](http://www.mindmajix.com)
* END POINT = /all-courses
* QUERY PARAMETER = courseName = apiTestingPostman

OR

GET <https://www.mindmajix.com/all-courses?courseName=apiTestingPostman&trainerName=Kiran>

* METHOD = GET
* PROTOCOL = HTTPS
* BASE URL = [www.mindmajix.com](http://www.mindmajix.com)
* END POINT = /all-courses
* QUERY PARAMETER#1= courseName = apiTestingPostman
* QUERY PARAMETER#2= trainerName = Kiran
* The response code for success scenario will be 2XX
* The response code for failure scenarios will 4xx OR 5xx

# EXAMPLE#5

POST [https://www.mindmajix.com/add-course //](https://www.mindmajix.com/add-course%20//) this API will create a new data on the database/server

{

“courseName”:”JMeter-Performance Testing”,

“courseDuration”:”20 hours”

}

* METHOD = POST
* PROTOCOL = HTTPS
* BASE URL = [www.mindmajix.com](http://www.mindmajix.com)
* END POINT = /add-course
* REQUEST BODY/PAYLOAD =

{

“courseName”:”JMeter-Performance Testing”,

“courseDuration”:”20 hours”

}

* The response code for success scenario will be 2XX
* The response code for failure scenarios will 4xx OR 5xx

# EXAMPLE#6

PUT [https://www.mindmajix.com/update-course?courseId=123 //](https://www.mindmajix.com/update-course?courseId=123%20%20%20%20%20%20%20%20%20//) this API will update existing details on the database

{

“courseName”:”JMeter-Performance Testing”,

“courseDuration”:”24 hours”,

“trainerName”:”Kiran”,

“courseContent”:”PDF-Download”

}

OR

PATCH [https://www.mindmajix.com/update-course?courseId=123 //](https://www.mindmajix.com/update-course?courseId=123%20%20%20%20%20%20%20%20%20%20%20%20//) this API will update existing details on the database

{

“courseName”:”JMeter-Performance Testing”,

“courseDuration”:”24 hours”,

“trainerName”:”Kiran”,

“courseContent”:”PDF-Download”

}

* METHOD = POST
* PROTOCOL = HTTPS
* BASE URL = [www.mindmajix.com](http://www.mindmajix.com)
* END POINT = /update-course
* REQUEST BODY/PAYLOAD =

{

“courseName”:”JMeter-Performance Testing”,

“courseDuration”:”24 hours”,

“trainerName”:”Kiran”,

“courseContent”:”PDF-Download”

}

* The response code for success scenario will be 2XX
* The response code for failure scenarios will 4xx OR 5xx

# EXAMPLE#7

DELETE <https://www.mindmajix.com/delete-course?courseId=123> // this API will delete the data from the database where courseId matches 123

OR

DELETE [https://www.mindmajix.com/delete-course?courseName=JMeter-Performance Testing](https://www.mindmajix.com/delete-course?courseName=JMeter-Performance%20Testing)

OR

DELETE <https://www.mindmajix.com/delete-course>

{

“courseId”=”123”

}

OR

DELETE <https://www.mindmajix.com/delete-course>

{

“courseName”=”JMeter-Performance Testing”

}

* METHOD = DELETE
* PROTOCOL = HTTPS
* BASE URL = [www.mindmajix.com](http://www.mindmajix.com)
* END POINT = /delete-course
* REQUEST BODY/PAYLOAD =

{

“coursed”=”123”

}

* The response code for success scenario will be 2XX
* The response code for failure scenarios will 4xx OR 5xx

# EXAMPLE#8

**PROJECT NAME**: CITI BANK CREDIT CARD API PROJECT

**API#1**

BELOW ARE THE APIs TO GET CREDIT CARD DETAILS

API Description : to get consumer name of credit card user

HTTP Method: GET

Protocol: HTTPS

Request Body: NA

API URL: [https://citibank.dev.creditcard.com\allUsers\{last4digitsofcreditcardnumber}](https://citibank.dev.creditcard.com\allUsers\%7blast4digitsofcreditcardnumber%7d)

Authentication: Enabled/Yes

Auth Type : Basic Authentication

UN: Kiran

PWD: citiKiran

**Fr success scenario,**

**resp code**: 200 OK

**resp body**:

{

credit card number: <16 digit number>,

firstname: <first name>

lastname: <last name>

}

**Response headers**:

* Connection=closed
* Transfer\_encoding=gzip
* Content-Type=application/json
* Server = Apache Tomcat
* X-Powered-By=ASP.net
* Cache: Private ; No Cache
* Cookies: No cookies will be sent by server

**For failure scenarios,**

resp code: 401, un-authorized

resp code: 400, bad request

resp code: 500, internal server error

# EXAMPLE#9

**API#2**

User-story: As a user, I can create a new credit card number to an eligible bank customer

**API#3**

User-story: As a user, I can modify the customer account details of an existing bank customer

**API#4**

User-story: As a user, I can delete the customer account details of an existing bank customer

**API#5**

User-story: As a user, I can validate whether or not the given bank customer is an active user or an inactive user

HTTP Method: GET

Protocol: https

Host Name: citibank.dev.creditcard.com

End Point: /v1/kyc

Query Parameter: creditcardnumber={last4digitsofcreditcardnumber}

**API URL:**

GET https://citibank.dev.creditcard.com/v1/kyc?creditcardnumber={last4digitsofcreditcardnumber}

**For success scenarios,**

**RESPONSE CODE: 200 OK**

**RESPONSE BODY:**

{

“customer status”:”Active or Inactive”,

“last transaction date”:”mm-dd.yyyy hh-mm-ss”

}

**For failure scenarios,**

RESPONSE CODES:

400-BAD REQUEST

401- UN-AUTHORIZED

500-INTERNAL SERVER ERROR