Question

1. Format the rest services support is xml, json, test
2. rest can also implement web services
3. performance is very good, it is light (url) and it uses http protocol
4. security is bad, everything is in url
5. url split into base
   1. url: domain (ex: [www.google.com](http://www.google.com))
   2. resource : action example: search, delete, update
   3. parameter
6. web services cannot implement rest services
7. web services support xml
8. performances is very bad because it is heavy with soap protocol (Simple Object access protocol)
9. Security is very good. Soap envelopes cannot be broken
10. REST → Representational State Transfer
11. HTTP methods for testers
    1. Get: it is used to extract information for the url
       1. ex: going to keka and retrieving the information about praveen like
          1. phone number
          2. DOB
          3. DOJ
          4. photo
          5. Designation
          6. Status
          7. Email
          8. Group
    2. POST: is used to send details to the server
       1. ex: going to keka and filling details of balisha regarding training like
          1. score
          2. functional SDET
          3. good performance
          4. Award
    3. PUT/ Patch: if we want to update some of the information, we will use PUT/ PATCH
    4. Delete: when we want to delete my information
12. Open the REST services in SOAP UI, we will use POST to create a google map
13. When we want to add another service we can
    1. right click, add new services from url
14. Add delete and add a script to delete from the response
15. Complete URL for GET\*\*:[http://216.10.245.166/maps/api/place/add/xml?key=qaclick123\*\*](http://216.10.245.166/maps/api/place/add/xml?key=qaclick123**)
    1. **Base URL** :<http://216.10.245.166>
    2. **Resource** : /maps/api/place/**add**/xml
    3. **Parameters**: key
16. POST is a super class, it can be used to post and delete
17. POST can be used as a HTTP request only if the url contains delete
18. Complete URL for Delete: \*\*[http://216.10.245.166/maps/api/place/delete/json?key=qaclick123\*\*](http://216.10.245.166/maps/api/place/delete/json?key=qaclick123**)
    1. **Base URL** :<http://216.10.245.166>
    2. **Resource** : /maps/api/place/**delete**/json
    3. **Parameters**: key
19. The First step in automation is to create a test suite
20. When doing a property transfer in REST and the file is stored in JSON, we will use JSON pathway
21. the object is place\_id
22. http code and the link to find it is: [HTTP CODE MEANINGS](https://restfulapi.net/http-status-codes/)
    1. 200 means ok
    2. 100 means continue
    3. 201 means created
    4. 202 means accepted
    5. 204 means no content
    6. 205 means reset contents
    7. 300 means multiple choices
    8. 302 means found
    9. 400 means bad request
    10. 401 means unauthorized
    11. 402 means payment required
    12. 403 means forbidden
    13. 404 means Not found
    14. 406 means not accepted
23. JSON: JAVA Script Object Notation
24. When we want to read an object, we use $.object name
25. We use the $. only when we are fetching from the json response file
26. property expansion syntax for REST **response**: ${Rest Service#Response#$.object name}
    1. ${add#Response#$.place\_id}
27. Syntax is the same for postman, and other json services file
28. Property expansion syntax for REST **request** when we want the token from the property transfer file
    1. ${Property Transfer#place\_id}
29. When we call a data from outside the test case, we will use the #TestSuite name
30. We can store the values in properties using property transfer
31. When we are using XML, we will use the XmlHolder
32. When we are using JSON, we will use JsonSluper().parseText
33. The code to input JSON response stored at any level of the services

// We need to input the library to store values

import groovy.json.JsonSlurper

//we need to store the response

def response = context.testCase.getTestStepByName("add").getPropertyValue("Response")

//Stripping the response into a json script

def holder = new JsonSlurper().parseText(response)

// getting only the place id

def placeid = holder.place\_id

//storing the id in testsuite

context.testCase.testSuite.setPropertyValue("id", placeid)

1. REST Service type of xml are pure xml and will find the response in //value used
2. Why no namespace in REST while in XML is because REST makes it lighter
3. XSD Xml terms and definition
4. To fetch data from json
   1. JsonPath count
   2. JsonPath Existence Match
   3. JsonPath Match
5. JsonPath Count we will get the object name in the jsonpath count and use select from current
6. JsonPath Existence Match checks the existence. Only for two object (counts / users)
   1. we can also do user[200].email
7. JsonPath Match will match the data user[233].\_id
   1. user is the parent object
   2. [233] is the particular number
   3. \_id is the child object
   4. to get the \_id from only the email
      1. user[?(@email==’liesl@gmail.com’)]\*\*.\_id\*\*
      2. the ?(@email == ‘liesl@gmail.com’) is a regex
      3. the \_id is the data it will retrive
8. to get the count from all users

// We need to input the library to read json files

import groovy.json.JsonSlurper

//we need to store the response

def response = context.testCase.getTestStepByName("Get All Users").getPropertyValue("Response")

//Stripping the response into a json script

def holder = new JsonSlurper().parseText(response)

// getting only the count

// def count = holder.count

// fetching the id value for a particular email

/\*for(def i=0; i< count; i++){

if(holder.users[i].email == 'sachi@gmail.com'){

def id = holder.users[i].\_id

log.info id

def idle = id.toString()

context.testCase.testSuite.setPropertyValue("id", idle)

}

}

\*/

// getting the id from user of 200 and the \_id

// this will delete the user at 200

def id = holder.users[200].\_id

// setting the custome property to test suite

context.testCase.testSuite.setPropertyValue("id", id)

1. for this particular delete, we use /user/${TestSuite#id}