**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*API Testing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**What is API?**  **VVVVIMMMPPPPPPPPPPP**

- API means application Programming interface.,

- API is used to communicate between 2 or more software components.

- API is usd to exchange the data between 2 ore more componenets

- Everydays we use application as like whats up,gmail,google, facebook, phonepay or google pay etc we are using API.

**What is API Testing?** **VVVVIMMMPPPPPPPPPPP**

- API testing is type of software testing.

- Inside the API Testing we focus business logic layer.

- API is lack of GUI/UI means API does not have UI/GUI, we have validate the busines logic of the functionality with the

of the API Testing.

- API testing is performed on Message layer or Business logic layer

**What is Web Services? VVVVIMMMPPPPPPPPPPP**

- These are services available on internet/network/web thar provide the way to communicate between 2 more or software components.

- Web Services work over on HTTP/HTTPs Protocol.

- We Need internet connection compulsory to perform its operations.

- We servicees does not have GUI/UI same like API.

- if we call API over HTTP/HTTPs protocol then it become the Web services.

**What is Diff between API and Web Services VVVVIMMMPPPPPPPPPPP**

Web Services API

i) It always need network for its operation API does not need network for its operations

ii) All Web services are API All API are not web services

iii) Web Services communicate with HTTP/HTTPs API communicate with CRUD calls.

protocol

iv) we can not test the API Without internet If we get any calls over the network then it become the Webv Services

**What are the different Request present inside the API/Web Services VVVVVIMMMPPPPPP**

i) POST Request it is used to create new Entity in server

ii) PUT Request it is used to update whole entity in server

iii) PATCH Request it is used to partial Entity in server

iv) GET it is used Retrieve the Entity information from server

v) DELETE Request it is used to delete the entity from the server

vi) HEAD request Retrieve the Entity information from server without Response body

vii) Options Request it is used to give information related to the Request,

which types of request we can hit on endpoints

**what are the different tools**

1)Postman

2) Karate

3) Soap UI

4) HTTP master

5) TOSCA

6) HP UFT/QTP

7) Rest Assured

8) Rest Pro

9) Swagger tool

10) pyRest Tool

**What are the different tools available in market to create the API documentations? VVVVVVVVVVVVVIMPPPPPPPPPPPP**

- there are differen tools present in market to create the API documentations

1) Swagger / OpenAPI

2) Postman tool

3) Apiary tool

4) RAML (Restful API Modeling Language) tools

1) Swagger / OpenAPI

- Swagger/ OpenAPI is widely used tool avaiable in market to create the RestFul API documentation.

- it allows developer to define the API Specification in standard format.

- In my current project we use swagger tool to create the API documentations

2) Postman tool

- Postman tool famous for API Testing but by using postman tool we create the API documentations also.

- in my current project we use Postman tool to Test the API manuallly

3) Apiary tool

- Apiary tool powerful tool for designing the API, creating documentation and mocking the APIs.

- in my current project we use Apiary tool for creating the mock servers.

4) RAML (Restful API Modeling Language) tools

- RAML tool is based on YAML language, and it use for creating the API documentations.

- it enables the developer to define the API Structure and request/response parameter and security in human readable format.

**Who will create API Documentations?**  **VVVVVVVVVVVVVIMPPPPPPPPPPPP**

Development team is responsible to create the API documentations

**What are the different tools available to test the API? VVVVVVVVVVVVVIMPPPPPPPPPPPP**

1) Postman tool to test the API Manually we use Postman tool

2) Rest Assured to Test the API automation we use Rest Assured.

3) Soap UI

4) Karate

5) HTTP master

6) TOSCA

7) Rest Pro

8) HP UFT/QTP

9) PyRest tool

etc

**What are the different types of HTTP Requests? VVVVVVVVVVVVVIMPPPPPPPPPPPP**

**there are 7 types of Request in API**

1) GET Request

2) POST Request

3) PUT Request

4) PATCH Request

5) DELETE Request

6) HEAD request

7) OPTIONS request

**there are different status code series present in API VVVVVVVVVVVVVIMPPPPPPPPPPPP**

1) 1\*\* series

- it provide the information about the API

2) 2\*\* series

- it return the successful response series

3) 3\*\* series VVVVVVVVVVVVVIMPPPPPPPPPPPP

it used navigate from one API to Another API.

or

it is also known as Navigation API series

4) 4\*\* series VVVVVVVVVVVVVIMPPPPPPPPPPPP

it is known as client side error code series

5) 5\*\* series VVVVVVVVVVVVVIMPPPPPPPPPPPP

it is known as server side error code series

Note:

Start JSON Fake server we use below command

json-server watch db.json

1) GET Request VVVVVVVVVVVVVIMPPPPPPPPPPPP

- GET Request used for retrieve the entity from server.

Verification/ verify

- Verify/ validate the status code 200

- verify/validate the status line 200 OK

- Validate the response time

- validate the response size

- Validate response body/ Response Payload

API Documentation /API contract Documentation / API Swagger Documentation

URI/endpoints : http://localhost:3000/employees/1

- it will return employee id 1 user informations

2) Delete Request: VVVVVVVVVVVVVIMPPPPPPPPPPPP

- it is used to delete the entity from server

Verification

- validate status code in real time server 204

fake server: status code 200

- validate status line in real time server 204 No Content

fake server: status line 200 OK

- validate response time

- validate response size

API Documentation /API contract Documentation / API Swagger Documentation

HTTP Request Type : Delete Request

URL/Endpoints http://localhost:3000/employees/2

3) POST Request: **VVVVVVVVVVVVVIMPPPPPPPPPPPP**

- it used to create a new entity in server

- Add request body/ Request Payload to the HTTP POST Request

- We pass Request body/Payload in JSON format.

- if we hit POST Request 100 times then it will create 100 new entity in server

how to add request body in postman tool/

- click on body icon

- click on raw icon

- select JSON options

- Enter JSON format

API contract Documentations / API Swagger Documentation

URI/Ends points http://localhost:3000/employees

Request Body/Payload

{

"firstName" : "Pooja" ,

"lastName" : "Mane",

"address" : "Delhi",

"mobileNumber" : 90909090

}

Verification/

- Validate status code -> 201

- validate status line > 201 Created

- Response time

- Response size

- Response payload / body

(What are the different famous HTTP Request in API?

1) POST Request

2) GET Request

3) PUT Request

4) PATCH Request

5) DELETE Request

6) HEAD Request

7) OPTIONS Request

-

1) POST Request; **VVVIMMMPPPPPPPP**

- it is used to create a new entity in server

- When we select JSON format automatically postman tool it will add content-type key and application /json values in headers.

- at the time of hitting the POST Request we have to send request body in JSON(Java Script Object Notation) format.

- And it will generate new id numbers

- After Hitting POST Request,

- server will return 201 as status code.

- server will return 201 Created as status line

- POST Request is called as secured request.

-server will return the Response body in POST Request.

- We can not bookmark the POST Request.

-

2) GET Request **VVVIMMMPPPPPPPP**

- it used to retrieve the entity from the server

- we never pass the request body at the time of hitting the GET Request.

- After hitting GET Request,

- server will return status code as 200

- server will return status line as 200 OK

- GET Request is not secured Request.

- GET Request can be bookmarked

-server will return the Response body in GET Request.

- by using GET Request we can retrieve all user information as well as single user information

e.g

http://localhost:3000/employees it will retrieve all employee information

http://localhost:3000/employees/1 it willl retrieve employee 1 information

-

3) DELETE Request VVVIMMMPPPPPPPP

- it used to delete the entity from server

- we never pass the request body at the time of hitting the DELETE Request.

- After hitting DELETE Request,

- server will return the status code as 204

- server will return the status line as 204 No Content

- server will not return the Response body Delete Request.

- DELETE Request can not be bookmarked.

- DELET request is secured request.

-

4) PUT Request **VVVIMMMPPPPPPPP**

- it used to whole entity in server

- When we select JSON format automatically postman tool it will add content-type key and application /json values in headers.

- at the time of hitting the PUT Request we have to send request body in JSON(Java Script Object Notation) format.

- Server will return status code as 200

- server will return status line as 200 OK

- PUT Request can not be bookmarked

- PUT Request is secured.

API Documentation/ API Contract Documentation

URI: http://localhost:3000/employees/1

Request Body/Request Payload

{

"firstName" : "Aarti",

"lastName" : "shah",

"address" : "Delhi",

"mobileNumber" : 808080

}

Verification:

- validate status code 200

- validate status line 200 OK

- validate response time

- validate response size

- validate updated response body/ payload

-

5) PATCH Request

- PATCH request is used to update partial Entity in server

- When we select JSON format automatically postman tool it will add content-type key and application /json values in headers.

- at the time of hitting the PATCH Request we have to send request body in JSON(Java Script Object Notation) format.

- Server will return status code as 200

- server will return status line as 200 OK

- PATCH Request can not be bookmarked

- PATCH Request is secured.

API Documentation/ API Contract Documentation

URI: http://localhost:3000/employees/1

Request Body /Request Payload

**How to access values from response body in postman tool? VVVVVIMMMPPP**

step 1: capture all response body and store in variables

or

parse JSON Response body in variables

var xyz JSON.parse(responseBody);

step 2: focus on key name and capture its values

var ab xyz.id;

console.log(ab);

Req: **VVVVVIMMMPPP**

print firstName values in console

step : capture all response body and store in variable

or

parse JSON response body in variables

var respbody JSON.parse(responseBody);

step 2: capture firstName value by focusing on key name

var b respbody.firstName;

console.log(b);

step 3: capture lastName value by focusing on key name

var c respbody.lastName;

console.log(c);

var respbody JSON.parse(responseBody);

var a respbody.firstName;

console.log(a);

var b respbody.lastName;

console.log(b);

Capture ID number/values from response body and store in global Environment variables

step 1: capture all response body and store in variables

or

parse JSON response body in variables

var respbody JSON.parse(responseBody);

step 2: capture Id number values

var b respbody.id;

step 3: set Id values in global environment variables

pm.globals.set("empid",b) VVVVVIMMMPPP

pm -postman

globals object name in postman tool

set() method

empid global environment variable name

b b is value for id variable in response body

Scenario:

1) create new entity > POST Request

i) capture id value and store in global variables

2) Retrieve the created Entity > GET Request

3) update new created whole entity -> PUT Request

4) Retrieve the update entity -> GET Request

5) Delete updated entity -> Delete Request

API contract documentation / API Documentation

Request body:

{

"firstName" : "Aboli",

"lastName" : "Shinde',

"city" : "Pune",

"gender" : true

"dob" : "ddMMyyyy",

"education" : "BE Comp",

"age" : 23

}

Scenario:

1) create new entity > POST Request

i) capture id value and store in global variables

2) Retrieve the created Entity > GET Request

3) update new created whole entity -> PUT Request

4) Retrieve the update entity -> GET Request

5) Delete updated entity -> Delete Request

**diff between GET and POST VVVVVIMMMPPP**

**diff between GET and PUT VVVVVIMMMPPP**

diff between GET and PATCH VVVVVIMMMPPP

diff between GET and DELETE VVVVVIMMMPPP

diff between POST and PUT VVVVVIMMMPPP

diff between POST and PATCH VVVVVIMMMPPP

diff between POST and DELETE VVVVVIMMMPPP

diff between PUT and PATCH VVVVVIMMMPPP

diff between POST and DELETE VVVVVIMMMPPP

diff between PATCH and DELETE VVVVVIMMMPPP

diff between GET and HEAD VVVVVIMMMPP

**What are the different variables in postman tool? VVVVVVVVVVVVVIMMMMPPPP**

- there are 3 different types of variables in postman tool

1) Collection variables

2) Environment variables

3) globals variables

1) Collection variables

- if declare any variables within the collection1 then access scope of this variables within the collection1 only.

- if declare any variables within the collection2 then access scope of this variables within the collection2 only.

how to set the collections variables in test tag?

syntax:

pm.collectionVariables.set("variableName",variableValue);

how to get values from the collections variables in test tag?

syntax:

pm.collectionVariables.get("variableName");

How to access Collections variables in Request URL bar?

- we access variables in Request URL bar mentioning variable name in 2 times curly bracket symbol

syntax:

{{variableName}}

2) Environment variables

- if we declare any variable within the SIT Environment then access scope of these variable within the SIT Environment only.

- if we declare any variable within the UAT Environment then access scope of these variable within the UAT Environment only.

how to set the variables in Environment?

syntax;

pm.environment.set("variableName",variableValue);

how to get Environment variables in test tag?

syntax:

pm.environment.get("variableName");

How to access Environment variables in Request URL bar?

- we access variables in Request URL bar mentioning variable name in 2 times curly bracket symbol

syntax:

{{variableName}}

3) global variables

- if we declare any variable as globally then access scope of this variable anywhere in workspace or project name.

how to set the global variables?

syntax;

pm.globals.set("variableName",variableValue);

how to get global variables in test tag?

syntax:

pm.globals.get("variableName");

How to access global variables in Request URL bar?

- we access variables in Request URL bar mentioning variable name in 2 times curly bracket symbol

syntax:

{{variableName}}

Scenario 1: **VVVVVVVVVVVVVIMMMMPPPP**

- if we have same variable name present in Collection level ,Environment level and in global level.

- it give priority access to the Environment level variables.

Scenario 2: **VVVVVVVVVVVVVIMMMMPPPP**

- if we have same variable name present in Collection level ,Environment level,

- it give priority access to the Environment level variables.

Scenario 3: **VVVVVVVVVVVVVIMMMMPPPP**

- if we have same variable name present in Collection level and global level,

- it give priority access to the Collection level variables.

Scenario 4: **VVVVVVVVVVVVVIMMMMPPPP**

- if we have same variable name present in Environment level and global level,

- - it give priority access to the Environment level variables.

API documentation/API Contract Documentation/Swagger Documentation

SIT Enviroment

POST Request:

BaseURI : http://localhost:4000

Resource Name :/employees

Content-Type application/json

Request Body

{

"firstName" : "{String}",

"lastName" : "{String}",

"address" : "{String}",

"mobileNumber" : {long}

}

POST Response

Status code 201

Status Line 201 Created

Content-Type application/json

Response Body

{

"id" : {int},

"firstName" : "{String}",

"lastName" : "{String}",

"address" : "{String}",

"mobileNumber" : {long}

}

GET Request; it will get All Employee information

BaseURI : http://localhost:4000

Resource Name :/employees

GET Response

Status code 200

Status Line 200 OK

Content-Type application/json

Response Body

[

{

"id" : {int},

"firstName" : "{String}",

"lastName" : "{String}",

"address" : "{String}",

"mobileNumber" : {long}

}

]

GET Request; it will get single Employee information

BaseURI : http://localhost:4000

Resource Name :/employees

Path Parameter {id}

GET Response

Status code 200

Status Line 200 OK

Content-Type application/json

Response Body

{

"id" : {int},

"firstName" : "{String}",

"lastName" : "{String}",

"address" : "{String}",

"mobileNumber" : {long}

}

PUT Request:

BaseURI : http://localhost:4000

Resource Name :/employees

Path Parameter /{id}

Content-Type application/json

Request Body

{

"firstName" : "{String}",

"lastName" : "{String}",

"address" : "{String}",

"mobileNumber" : {long}

}

PUT Response

Status code 200

Status Line 200 OK

Content-Type application/json

Response Body

{

"id" : {int},

"firstName" : "{String}",

"lastName" : "{String}",

"address" : "{String}",

"mobileNumber" : {long}

}

PATCH Request:

BaseURI : http://localhost:4000

Resource Name :/employees

Path Parameter /{id}

Content-Type application/json

Request Body

{

"firstName" : "{String}",

"lastName" : "{String}",

"address" : "{String}",

"mobileNumber" : {long}

}

PATCH Response

Status code 200

Status Line 200 OK

Content-Type application/json

Response Body

{

"id" : {int},

"firstName" : "{String}",

"lastName" : "{String}",

"address" : "{String}",

"mobileNumber" : {long}

}

DELETE Request:

BaseURI : http://localhost:4000

Resource Name :/employees

Path Parameter /{id}

DELETE Response

Status code 204

Status Line 204 No Content

SIT Env http://localhost:4000

Global Env /employees

global id id

UAT Env http://localhost:5000

Req:

Validate employee end to end collection in SIT as well as UAT Env

POST

capture id number post request and set the globally.

GET -> retrieve the created entity

PUT

GET

PATCH

GET

DELETE

**how to start server in different port**

json-server watch db.json port5000

json-server watch db.json port9191

Customer Request:

{

"customer" :

[

{

"id" : 1,

"fname" : "Rohini",

"lname" : "takale",

"productName" : "Iphone",

"productPrice" : 909900,

"billingAddress" : "Pune",

"state" : "MH",

"country" : "INDIA"

}

]

}

POST

capture id number post request and set the globally.

GET -> retrieve the created entity

PUT

GET

PATCH

GET

DELETE

Ass:

SIT Env

http://localhost:4000

UAT Env

http://localhost:5000

Global var

CustomerPath customers

employeePath employees

CustomerId custid

EmployeeId empid

Req:

Validate employee and customers end to end collection in SIT as well as UAT Env

POST

capture id number post request and set the globally.

GET -> retrieve the created entity

PUT

GET

PATCH

GET

DELETE

**What is Data driven Testing in postman tool? VVVVVVVVVVVVVIMMPPPPPPPP**

or

**How to achieve data driven testing in postman tool? VVVVVVVVVVVVVIMMPPPPPPPP**

- Data driven Testing means we store the test data or inputs in files then we capture stored test data and we use these test data to

HTTP Request.

**- there** **are 2 ways we can achieve the data driven testing in postman tool**

**1) by mentioning test data in JSON File**

**2) by mentioning test data in CSV Files**

- We maintain the test data or inputs in JSON or CSV files and at the time of running the collections we upload the JSON/CSV file

- automaticallly postman tool will pick up the test data from file and it will map to the request body as per key name.

API Contract Documentation / API Documentation

Request

Base URI : http://localhost:3000

Base Resource /employees

Header Content-Type application/json

Request Body/Payload

{

"firstName": "{String}",

"lastName": "{String}",

"address": "{String}",

"mobileNumber": long,

}

Map test data JSON file to Request body

{

"firstName": "{{firstName}}",

"lastName" : "{{lastName}}",

"address" : "{{address}}",

"mobileNumber" : 1234512345

}

Ass:

Create new Workspace:

Create SIT Environment

BaseURI http://localhost:4000

Create UAT Environment

BaseURI http://localhost:5000

Global Environment

EMPPATH employees

CUSTPATH customers

EMPID

CUSTID

maintain the test data in JSON test data file for 6 iteration

Run the collections

1) POST

2) GET

3) PUT

4) GET

5) PATCH

6) GET

7) DELETE

**What is Data driven Testing in postman tool? VVVVVVVVVVVVVIMMPPPPPPPP**

**or**

**How to achieve data driven testing in postman tool? VVVVVVVVVVVVVIMMPPPPPPPP**

- Data driven Testing means we store the test data or inputs in files then we capture stored test data and we use these test data to HTTP Request.

**- there are 2 ways we can achieve the data driven testing in postman tool**

**1) by mentioning test data in JSON File**

**2) by mentioning test data in CSV Files**

- We maintain the test data or inputs in JSON or CSV files and at the time of running the collections we upload the JSON/CSV file

and automaticallly postman tool will pick up the test data from file and it will map to the stored request body as per key name in Request body.

2) by mentioning test data in CSV Files

API Documentations:

POST Request

BaseURI: http://localhost:3000

Resource /employees

Request Header Content-Type application/json

Request body:

{

"firstName" : "{{testdatafirstNameKeyName}}",

"lastName" : "{{testdataLastNameKeyName}}",

"address" : "{{testdataaddressKeyName}}",

"mobileNumber" : "{{testdatamobileNumberKeyName}}",

}

SIT : http://localhost:4000

syntax:

json-server watch empandcustdb.json port4000

UAT: http://localhost:5000

syntax:

json-server watch empandcustdb.json port5000

API Contract Documentation:

"customers":

{

"id": "1",

"fName": "Anjali",

"lName": "More",

"productName": "Iphone",

"price": 909090,

"quantity": 2

}

**What are the different Parameters in Postman tools? VVVVVIMMMMPPPPPPP**

**or**

**What are the different Parameters we use in RestFul web Services? VVVVVIMMMMPPPPPPP**

there are different parameters in postman tool or RestFul Web services

1) Query Parameters

2) Path Parameters

3) Request Body Parameters

4)Headers Parameters

5) Cookies Parameters

1) Query Parameters **VVVVVIMMMMPPPPPPP**

- Query Parameter are key-value pair appendend in the URL after quetions mark (?).

- if we mention multiple query parameters then it will separated by ampersands (&).

- Query Parameter is used to filtering test data, sorting test data according to conditions.

- Especially We use Query Parameters in GET Request.

- Once we add the Query Parameters it will visible in URL.

it find all employees

http://localhost:3000/employees

find all employee whose firstname is shital in API

http://localhost:3000/employees?firstName'shital'

How to find all employees whose first name is Shital in backend database?

e.g.

select \* from employees where firstName'Shital';

how to find all employees whose address is pune in API?

http://localhost:3000/employees?address'Pune'

how to find all employees whose address is pune in backend database?

select \* from employees where addressPune;

how to find all employees whose lastname is Patil and address is pune in API?

http://localhost:3000/employees?lastNamePatil&addressPune

how to find all employee whose lastname is Patil and address is Mumbai in backend database?

select \* from employees where lastName'Patil' and address'Mumbai';

find all employee salary in ascending order in API?

http://localhost:3000/employees?salarysort

find all employee salary in ascending order

select \* from employees order by salary asc;

find all employee salary in descending order in API?

http://localhost:3000/employees?salary-sort

find all employee salary in descending order

select \* from employees order by salary desc;

2) Path Parameter **VVVVVIMMMMPPPPPPP**

- it used to identify and retrieve the specific resource in server or endpoint in server.

- Once we add path parameter automatically it wil appended in URL.

- we define path parameter in double curly bracket to access the environment variables

-we define path parameter in request level by using : (colon) then parameter name and values

- Path parameter it replace the values when we hit the request.

can we add multiple path parameter in 1 request?

yes

-

find employee whose id is 10 in API?

http://localhost:3000/employee/:id

Path parameter

id 10

find employee whose id is 10 in endback database?

select \* from employees where id10;

-

http://localhost:3000/:Path

Path Parameter

Path employees

find all employees?

select \* from employee;

http://localhost:3000/:EmpPath/{{id}}

**Diff between Query Parameter and Path Parameter? VVVVVVIMMMPPPPPPPPPPPPPP**

3) Request Body Parameters

- Request Body Parameters we use in POST, PUT and PATCH Request.

- Request Body Parameters are used to send test data to the server.

- we can write Request Body Parameters in different format as like JSON, XML , form data format,Multipart format,

X WWW form url encoded format, GraphQL format.

- Request body parameters are not visible in URL,

- Request Body Parameters are used to provide the privacy for sensitive test data.

- Request Body Parameters can handle the complex data structures.

**how many ways we can send payload to the server? VVVVVVIMMMPPPPPPPPPPPPPP**

**or**

**How many ways we can send request body to the server? VVVVVVIMMMPPPPPPPPPPPPPP**

**or**

**How many ways we can send request payload to the server? VVVVVVIMMMPPPPPPPPPPPPPP**

there are 5 ways we can send payload to the server

1) by using form-data format

2) by using x-www-form-urlencoded format

3) by using raw format

4) by using binary format

5) by using GraphQL format

- by current project we send test data to the server by using JSON raw format.

**how to upload files in RestFul API? VVVVVVIMMMPPPPPPPPPPPPPP**

**or**

**How to upload file in Postman tools? VVVVVVIMMMPPPPPPPPPPPPPP**

we upload file in postman tool by using ""form-data format"" or multipart/form-data format

and there we select option as file and we upload the file for desktop location

**Which postman version u are using? VVVVVVIMMMPPPPPPPPPPPPPP**

PostmanRuntime/7.37.0

**How to create test cases in postman tool? VVVVIMMMPPPP**

we write the test cases by using java script language and chai assertion

**Which languages u are using in postman tool to write the test cases? VVVVIMMMPPPP**

we use java script language

**which assertion we use to validate the response payloads in postman? VVVVIMMMPPPP**

we use chai assertion in postman tool

1) Test cases for POST Request status code

- POST Request status code 201

syntax:

pm.test("POST Request status code", function()

{

pm.response.to.have.status(201);

});

GET Request Status Code 200

pm.test("GET Request status code",function()

{

pm.response.to.have.status(200);

});

TC for DELETE Request 204

syntax:

pm.test("Delete Status code",function()

{

pm.response.to.have.status(204);

});

TC for PUT/PATCH Request 200

syntax:

pm.test("PUT Status code",function(){

pm.response.to.have.status(200);

});

2) TC for POST Request Status line

-> status line is Created

syntax:

pm.test("POST Status line",function(){

pm.response.to.have.status("Created");

});

GET status Line Test cases OK

syntax:

pm.test("GET Request status line", function(){

pm.response.to.have.status("OK");

});

DELETE Request status line > No Content

syntax:

pm.test("Delete Request status line", function(){

pm.response.to.have.status("No Content");

});

PUT/PATCH Request status line > OK

syntax:

pm.test("PUT Request status line TC", function(){

pm.response.to.have.status("OK");

});

pm.test("Test cases for POST Request Status Code",function(){

pm.response.to.have.status(201);

});

pm.test("Test Cases for POST Request status Line",function(){

pm.response.to.have.status("Created");

});

pm.test("Test cases for GET Request Status code",function(){

pm.response.to.have.status(200);

});

pm.test("Test cases for GET Request Status Line",function(){

pm.response.to.have.status("OK");

});

pm.test("test case for DELETE Request status code",function(){

pm.response.to.have.status(204);

});

pm.test("Test cases for DELETE Request status Line", function(){

pm.response.to.have.status("No Content");

});

pm.test("Test cases for PATCH Request status code",function(){

pm.response.to.have.status(200);

});

pm.test("Test cases for PATCH Request status Line",function(){

pm.response.to.have.status("OK");

});

Test cases for Response time

syntax:

pm.test("Test cases for POST Request Response time", function()

{

pm.expect(pm.response.responseTime).to.be.below(2000);

});

e.g

2000ms 2 sec

3000ms 3 sec

4000ms 4 sec

pm.test("Test Cases for GET Request Response time",function(){

pm.expect(pm.response.responseTime).to.be.below(2000);

});

pm.response.responseTime

it will captue actual response time

pm.test("Test cases for PATCH Request Response time ",function(){

pm.expect(pm.response.responseTime).to.be.below(2000);

});

//capture actual status code

var a pm.response.code;

console.log("Status code is ",a);

//capture actual status line

var b pm.response.status;

console.log("status line is ",b);

//capture actual response time

var c pm.response.responseTime;

console.log("Response time is ",c);

verification/Assertion

i) status code

ii) status line

iii) response time

POST 201 Created

GET 200 OK

PUT 200 OK

GET 200 OK

PATCH 200 OK

GET 200 OK

DELETE 200 OK

GET 404 Not Found

pm.test("Test cases for POST Request status code",function(){

pm.response.to.have.status(201);

});

pm.test("Test cases for POST Request status Line",function(){

pm.response.to.have.status("Created");

});

pm.test("Test cases for POST Request Response time",function(){

pm.expect(pm.response.responseTime).to.be.below(2000);

});

var respbody JSON.parse(responseBody);

var abc respbody.id;

pm.globals.set("EMPID",abc);

https://bookstore.toolsqa.com/swagger/#/

**How to capture values from response body in postman tool? VVVVIMMMMPPPP**

Response Body:

{

"id" : 1,

"name" : "Harshini",

"username" : "harshini@gmail.com",

"dob" :

{

"date" : 10,

"month" : "dec",

"year" : 2001

}

}

step 1: parse JSON response body into the variable

var respbody JSON.parse(responseBody);

step 2: capture username values from response body.

var abc respbody.username;

step 3: create test cases for username

pm.test("Test cases for username", function(){

pm.expect(abc).to.be.eql("harshini@gmail.com");

});

step 3: create test cases for username

pm.test("Test cases for username",function(){

pm.expect(pm.response.json().username).to.be.eql("harshini@gmail.com");

});

var respbody1 JSON.parse(responseBody);

var a respbody1.username;

console.log(a);

var respbody2 pm.response.json();

var b respbody2.username;

console.log(b);

pm.test("Test cases for response body username1", function()

{

pm.expect(a).to.be.eql("harshini@gmail.com");

});

pm.test("Test cases for response body username2", function()

{

pm.expect(b).to.be.eql("harshini@gmail.com");

});

var respbody1 JSON.parse(responseBody);

var month1 respbody1.dob.month;

console.log(month1);

var respbody2 pm.response.json();

var month2 respbody2.dob.month;

console.log(month2);

pm.test("Test cases for response body month ", function(){

pm.expect(month1).to.be.eql("dec");

});

pm.test("Test cases for response body month2 ",function()

{

pm.expect(month2).to.be.eql("dec");

});

pm.test("Test cases for response body month ", function(){

pm.expect(JSON.parse(responseBody).dob.month).to.be.eql("dec");

});

pm.test("Test cases for response body month2 ",function()

{

pm.expect(pm.response.json().dob.month).to.be.eql("dec");

});

var respbody1 JSON.parse(responseBody);

var a respbody1.dob;

console.log(a);

pm.test("test cases for response body year ",function(){

pm.expect(pm.response.json().dob.year).to.be.eql(2001);

});

pm.test("test cases for response body year 2",function()

{

pm.expect(JSON.parse(responseBody).dob.year).to.be.eql(2001);

});

//capture Pune values from response body and print in console

var a JSON.parse(responseBody).address.currentAddress.city;

console.log(a);

var b pm.response.json().address.currentAddress.city;

console.log(b);

//create the test cases for response body City values

pm.test("test cases for response body city values ", function(){

pm.expect(JSON.parse(responseBody).address.currentAddress.city).to.be.eql("Pune");

});

pm.test("test cases for response body city values ", function(){

pm.expect(pm.response.json().address.currentAddress.city).to.be.eql("Pune");

});

pm.test("Test cases for responseBody gender values ",function(){

pm.expect(pm.response.json().gender).to.be.eql("female");

});

JSON format

1:1 one key have 1 value json

1: many one key have multiple values json array

1: {many: many} one key have multiple key-value pair [nested json ]

var respbody JSON.parse(responseBody);

var a respbody.topics[2];

console.log(a);

var b pm.response.json().topics[2];

console.log(b);

pm.test("test cases for respbody body topic values" ,function(){

pm.expect(pm.response.json().topics[2]).to.be.eql("API automation");

});

pm.test("Test cases for response body topics values ",function()

{

pm.expect(pm.response.json().topics).to.be.include("db automation");

});

var a pm.response.json().address[1].State;

console.log(a);

pm.test("Test cases for response body State values ",function(){

pm.expect(pm.response.json().address[1].State).to.be.eql("MH2");

});

pm.test("Test cases for response body State values ",function(){

var abc pm.response.json().address;

for(var i0;i<abc.length;i++)

{

var a abc[i].State;

if(a"MH1")

{

pm.expect(a).to.be.include("MH1");

}

}

});

pm.test("Test cases for responseBody tools name",function(){

pm.expect(pm.response.json().Topics.Automationtesting.UIAutomation.toolsName[1]).to.be.eql("pom");

});

pm.test("Test cases for responseBody tools name",function(){

pm.expect(pm.response.json().Topics.Automationtesting.UIAutomation.toolsName).to.be.include("pom");

});

pm.test("Test cases for responseBody tools name for API" ,function(){

pm.expect(pm.response.json().Topics.Automationtesting.APIAutomation.toolsName).to.be.include("Hamcrest");

});

pm.test("Test cases for responseBody tools name for db" ,function(){

pm.expect(pm.response.json().Topics.Automationtesting.databaseAutomation).to.be.include("JDBC");

});

https://jsonpath.com/

https://jsonpathfinder.com/

**What is Headers in API? VVVVVIMMMMPPPPPPPP**

-

- Headers are used to provide additional information about the API /emdpoints

- Headers are used to control the behavior of the server to handle the request.

there are 2 types of Headers

1) Request Header

2) Response Header

1) Request Header

- Request Header is used to send more information from api client to server.

- in Request Header we mention different things as like

i) Authorization type

- we are adding Authorization types for API once we add Authorization types then automatically it will add inside request headers

- we have different types of Authorization,

a) basic auth

b) digest Auth

c) bearer token

d) OAuth 1.0

e) OAuth 2.0

- by using Authorization we are verifing the user is valid or not to access the API/endpoints.

ii) Content-Type

- Content-Type is used to specify the format of request body such as in JSON format,XML format, multiPart format, x-WWW-urlencoded format etc.

- Content-Type it is used to mention request body format types.

iii) Accept

- inform server about the type of content that client can understand,

- Accept it is used to response content negotiaton.

- in Accept header we mention response body format types.

iv) User-Agent

- It provides the more information related to the software or devices.

v) Cache-Control

-direction of caching mechanisms along with request and response chaining.

vi) Cookies

- previously stored cookies send to the server for session management.

or

- Send stored information/cookies to the server.

vii) Custom Header

- it add additional information about application functionality . which may contains we have metadata specific information .

viii) Host

- we deploye application on server

or

- it provide the server locations

2) Response Header

i) Content-Type

- it provide format of response body return from the server.

ii) Cache- Control

- direction of controlling caching mechanism on the client side.

iii) Access-Control-Allow-Origin

define the which origins are allowed to access the response data via cross origin resource sharing.

iv) set-cookies

- used to set the cookies on the client side for maintaining the session information.

v) date:

specifies the date and time which we are sending the request

vii) Content-Length

- it gives information about the response body size in bytes

viii) Etags

- it provide information about the specific software version /application version typical used for caching and conditional requests.

ix) x-RateLimit-limit

- x-RateLimit-limit means we can send maximum request within the times

x) X-RateLimit-Remaining

- remaining request we can send within the specific times

xi) X-RateLimit-Used

- it gives information about how many times we hit the requests/

xii) Server

server name

var a pm.response.headers.get("Content-Type");

console.log(a);

pm.test("Test cases for Content-Type response header",function()

{

pm.expect(pm.response.headers.get("Content-Type")).to.be.include("application/json");

});

var a pm.response.headers.get("Content-Type");

console.log(a);

pm.test("Test cases for Content-Type response header",function()

{

pm.expect(pm.response.headers.get("Content-Type")).to.be.include("application/json");

});

pm.test("Test cases for X Rate Limit ",function(){

pm.expect(pm.response.headers.get("X-RateLimit-Limit")).to.be.eql("60");

});

pm.test("Test cases for Server names",function(){

pm.expect(pm.response.headers.get("Server")).to.be.include("GitHub.com");

});

pm.test("Test cases for Content-Type",function(){

pm.expect(pm.response.headers.get("Content-Type")).to.be.include("application/json; charsetutf-8");

});

pm.test("Test cases for server",function(){

pm.expect(pm.response.headers.get("Server")).to.be.eql("cloudflare");

});

401 Unauthorized

403 Forbidden

404 Page Not Found

405 method not allowed

415 Unsupported media type (format)

**What are the different types of tools available to create API documentation VVVVVIMMMPPPPPPPP**

there are different tools

1) Swagger tool

2) Postman tool

3) Apiary tool

4) RAML tool.

1) Swagger tool

- Swagger/openAPI tool used to create API documentation and

- it allowed to developer to define the API specification in standard format.

- in my current project we use swagger tool to create API documentation

2) Postman tool

- In Postman tool also we create API documentation

- but in my current project we use Postman tool to test the APIs or Web Services manually

3) Apiary tool

- by using Apiary tool also we can create API documentations

- but in my current project we use Apiary tool to creating Mocking API.

4) RAML tool.

- RAML means RestFul API Modelling Language tool.

- RAML tool used for creating the API documentations.

- RAML tool is used on YAML language.

- it enables to developer to define the API structure of request or response parameters in human readable format.

**What are the different types of Authorization in API**  **VVVVVIMMMPPPPPPPP**

- there are different types of Authorization in API

1) No Auth

2) Basic Auth

3) Digest Auth

4) Bearer token

5) OAuth 1.0

6) OAuth 2.0

7) JWT Auth

8) API Key

**What is API Documentation? VVVVVIMMMPPPPPPPP**

there different important things present in API documentation

1) Endpoints

2) Request Parameter

3) Request and Response format

4) Authentication and Authorization

5) Response code/ status Code and Error Handling code

6) Usages Examples or code Snippets

7) Interactive documentations

8) Versioning and changelogs

1) Endpoints

- in API documentation we have different endpoints available to interact with API.

- Each endpoints are used for different purpose as like retrieve entity,create a entity, update the entity and delete the entity.

- To perform this operation we use different types of request as like GET Request, POST Request, PUT/PATCH Request or DELETE request.

- in API documentation we mention endpoints along with request type.

2) Parameter

- in API documentation they clearly mention parameter as like path parameter,query parameter, request body paramter

header parameter, cookies parameter, form multipart parameter along with key-value pair and its datatype information

- API documentation also provide more information regarding the parameters as like which parameter compulsory or optional.

3) Request and Response format

- in API documentation, they clearly mention we have to send request body in JSON/XML format to the server

and after getting the response, server should return response body in JSON/XML format.

- Which type of request body format is accepted by endpoints and What will be response body format will return from server.

Can we send request body in JSON or XML format for same end points?

- Yes

can we send request body in combination of JSON and XML format?

No

4) Authentication and Authorization

- in API documentation, it should provide the instruction regarding how to add Authorization or how to use them

- also it provide Authorization types of information

5) Response code/ status Code and Error Handling code

- in API documentation must cover various status code and error code message that API Can return in Response.

- This helps to handle different request scenario as like successful request, error code, rate limiting and authentication fail (401/403)

6) Usages Examples or code Snippets

- in API documentation, it includes the Practial usage examples and code snippets with how to hit the API effectively.

- in API documentation, they should provide dummy examples to how to use APIs.

7) Interactive documentations

- Interactive documentations means same tools allows to make the request directly without other tools.

8) Versioning and changelogs

- in API Documentation, it clearly mention the API version details and any recent changes introduced in each and every version.

- Change log it helps to track the updates .

**What is HTTP status code**  VVVVIMMMPPPPPPPPP

1\*\* Information about the API

100 Continue

2\*\* Successful Response Series VVVVIMMMPPPPPPPPP

status code Status Line

200 OK\*\* - Successful we hit the request and Successful we get response from server.

201 Created\*\* - Successful we have created new entity in server

204 No Content\*\* - Successful we have deleted entity in server.

205 Reset Content - Successful we have removed all written content

3\*\* Navigation Series / Navigate from one API to Another API/

301 Moved Permanently

302 Found

4\*\* Client Side Error VVVVIMMMPPPPPPPPP

status code Status Line

400 Bad Request - client added the wrong url or request parameters.

401 UnAuthorized\*\* - user is invalid [e.g if we enter invalid credentails then we will get status code as 401]

402 Payment Required\*\* - First we have to do the payment then we can access this functionality

403 Forbidden\*\* - user dont have permission to acces the API.[user enter valid credentails but dont have

permission to access the API].

404 Not Found\*\* - request page is not available.

405 Method not Allowed\*\* - As per the API documetation if they given request type is GET and and we are trying to hit other request type then we will get 405 status code.

415 Unsupported Media type\*\* - if we dont add the content-type header then we will get Unsupported media type

422 Unprocessable Content\*\* - if we have already username exist in database and we are trying to use same username to create new entity then we will get 422 status code

can we create facebook multiple new account by using same username?

5\*\* Server Side Error **VVVVIMMMPPPPPPPPP**

status code Status Line

500 Internal Server Error\*\* -

501 Not implemented\*\* - in backend server still functionality is not implemented

502 Bad Gateway\*\*

503 Service Unavaileble\*\* - server is stopped due to some misinterpresentation or issue.

504 Gateway timeout\*\* - if we enter valid OTP but after the timout then we will get 504 statuc dode

**What we have to check/validate at the time of performing API testing? VVVVIMMMPPPPPPPPP**

i) Accuracy of data

ii) Schema Validation : we have to validate response key and which types of values is accepted by the key means datatype

iii) Data type validation, order validation and completeness of order.

iv) Authorization validations

v) Response Payload

vi) Error code in case API returns

vii) Resonse headers

viii) Response cookies

ix) Non Functional testing like performance testing and security testing

**Book API End to End Flow with Authorization**

1) create user

Request Type: POST

BaseURI:https://bookstore.toolsqa.com/

Resource Name:/Account/v1/User

Content-Type: application/json

Request body

{

"userName" : {String},

"password" : {String}

}

2) Generate Token

Request Type: POST

BaseURI:https: //bookstore.toolsqa.com/

Resource Name: /Account/v1/GenerateToken

Content-Type: application/json

Request body

{

"userName" : {String},

"password" : {String}

}

3) Authorized

Request Type: POST

BaseURI:https: //bookstore.toolsqa.com/

Resource Name: /Account/v1/Authorized

Content-Type: application/json

Request body

{

"userName" : {String},

"password" : {String}

}

4) get User

Request Type : GET

BaseURI:https: //bookstore.toolsqa.com/

Resource Name: /Account/v1/User

Path Parameter {userId}

Bearer Token : access token

5) get All books

Request Type : GET

BaseURI:https: //bookstore.toolsqa.com/

Resource Name: /BookStore/v1/Books

6) Add Books

Request Type : POST

BaseURI:https: //bookstore.toolsqa.com/

Resource Name: /BookStore/v1/Books

Bearer Token : access token

Request Body:

{

"userId": "string",

"collectionOfIsbns": [

{

"isbn": "string",

"title": "Automation Test",

"subTitle": "UI Automation",

"author": "Softtech",

"publish\_date": "2024-03-30T02:34:30.529Z",

"publisher": "Softtech",

"pages": 700,

"description": "UI and API",

}

]

}

**What are the different challenges face during the API Testing? VVVVVVIMMMPPPPPPPPP**

- common challenges we face at the time of performing API In testing

1) incomplete API documentation

- When an API documentation should be up to date.

- if development team have not maintain the API documentation then testing team maye face struggle to test the APIs.

- Development team should maintain the well structured API documentations.

2) Data Handling

- Testing team have to test the same API endpoints in different formats as like JSON/XML/Java script/html etc format with large set of data.

- Test data integrity can be challenging task.

3) Authentication and Authorization

- API require Authentication and Authorization mechanism can be applied on API and

we have to handle these Authentication and Authorization using different Authorization types as like

basic Authorization, digest Authorization,access token, bearer token, custom key-value and if we access any third party application then we have to use OAuth Authorization.

- if enter wrong credentails then API should return negative status code or 4\*\* series status code,

- and if we enter the expire access token API should be return negative status code or 4\*\* series status code.

- if any user is Authenticated but if we denied the access of some functionality for users then API should be return the negative response code.

- if we check Authentication and Authorization for different user roles.

4) Error Handling

- API should handle the various error scenarions as like request type error, data format errors, and latency/ timeour error etc.

- Request Type Error means if development team they use request type for each and every request and if testing team use the other request type for

API endpoints then API endpoints should return the 405 (Method Not Allowed) as status code.

- Data Format Errors means if development team use request body format as JSON then API should be accept the same format, if we use other format type

then API should written return the 415 error code. (Unsupported Media type).

- Latency Error means API should return the response within the time,

5) Performance of the API

- Ensuring each and every API should expected load and response within the time frame.

- if user sending 100 request within the time frame we have verify the response time for APIs.

- we use Jmeter or load runner tool to check API performance.

6) Integration of API

- API integration with other software ensure the seamless integrations.

- in API integration we have to capture the data from one application and we have to transfer these data to the other applications.

7) Versioning and backward/forward compatibility

- API undergo changes and version of over on the times.

- testing team test the the old versions with of functionality after new API updated.

- and it can be challenging task beacuse sometime some API face issue in compatibility with latest version of of the software.

- API should supports for new Version of the software or old version of the software.

- If we add new functionality then new functionality should be compatibility with old functionality

8) Handling Rate limits

- by using X- Rate limits development team they are developing the limits for API request and if client sending more request then API should return errors message.

- we test the rate limits by using response headers

**What is Latency in API? VVVVVVIMMMPPPPPPP**

- Latency in API means amount of time it takes from user to send request to the server and receive response from server.

- Latency ususally we measured in milliseconds.

- Latency means amout of time it takes for single request to complete the cycle.

- higher in Latency means slower of the API response.

- Latency API means response time.

write the test cases for latency?

pm.test("Test cases for response time",function(){

pm.expect(pm.response.responseTime).to.be.below(2000);

});

What are the different testing types we performed in API? VVVVIMMMMPPPP

-we performed different testing types on API

1) Functional testing : by using Functional testing we validate the application behavior as per the requirment documents

or

by using Functional testing we validate the functionality of the application.

2) Security testing : by Security testing we test to ensure that data send in secured mode.

or

: by Security testing we validate application access functionality as per the user roles wise.

3) Performance testing : by using Performance testing we validate the speed and accuracy of API response.

4) Integration testing : by using Integration testing we validate API integration flow from one to another system.

5) compatibility testing : by using compatibility testing we validate application with different versions.

**What are the different types of API VVVVIMMMMPPPP**

1) public API

2) private API

3) shared API

4) composite API

5) Internal API

API protocol **VVVVIMMMMPPPP**

1) REST API

2) SOAP API

3) GraphQL API

4) WebSocket API

**diff between SOAP API and REST API VVVVIMMMMPPPP**

XML JSON/XML format

**what is GraphQL? VVVVIMMMMPPPP**

**diff between HTTP and HTTPs VVVVIMMMMPPPP**

**what are the different HTTP Protocol in API?**

**- GET/POST/PUT/PATCH/DELETE/HEAD/OPTION VVVVIMMMMPPPP**

Basic Authorization we share credentails in base 64 format

Digest Authorization > we share credentails in Hashed format/

**What is Payload and its types? VVVVIMMMMPPPP**

**how to upload binary file in API? VVVVIMMMMPPPP**

**or**

**How to upload the files?**

multipart/form-data; boundary<calculated when request is sent>

**what is mock API? VVVVIMMMMPPPP**

**What are the different API types? VVVVIMMMMPPPPPP**

- there are different types of APIs

1) Private API

2) Partner API

3) Open / Public API

4) Composite API

5) Internal API

1) Private API

- Development team use these API within the team.

- These API is created in house by development team and it is highly personalized API according to team needs.

- These API can not be used by external developer or company.

- these API specially build for own project or company use.

2) Partner API

- We use Partner API to collaborate on Projects.

- Partner developer they can modify the API as per the requirements wise.

- These API allow or maintains the Organization API flows .

3) Open / Public API

- Open API or public API is used to any organization.

- it is free to available in market , any company they can use these API.

- some Open API have free of cost and some have with charges.

- if u want to use Open / Public API then we have register application information in their portal.

4) Composite API

- Composite API it allows to application to make the request in different platform (OS and Language)

- Composite API is used widely by organization to capture the test data from different platforms.

5) Internal API

- Internal API is strictly used in within the organization or company.

- if u want to connect to internal software component or functionality then we use Internal API

**What are the different types of API protocol? VVVVVVIMMMMPPPP**

- API protocol it is used to exchange the data between 2 or more software components

there are different types of API protocol

1) REST API

2) SOAP API

3) GraphQL API

4) Webhook API

1) REST API

- Representation State Transfer (REST) it is used on web applications.

- it is used to exchange the data between 2 or more software components

- it uses HTTP or HTTPs request to send or retrieve the data from server.

- REST is stateless means they dont store any session information/ test data between the requests.

- REST it support for different format as lile JSON/XML/java script/html etc.

-it is more preferred then other API protocol.

- REST is architectual style, so easily we can implements SOAP protocol inside the REST.

- in REST service it inherit the securities from other Protocol.

2) SOAP API

- Simple Object Access Protocol used in complex transactions that require higher level of security.

- SOAP uses only XML format.

-SOAP is protocol, we can not implements or inherit securities from other protocol.

- SOAP define the their own securities

- SOAP is slower than the REST.

- SOAP is preferred is less compare with REST but in transactions level API we preferred SOAP protocol.

3) GraphQL API

- GraphQL API is mostly used in mobile based applications.

- in GraphQL we use single endpoints for the multiple purpose.

- GraphQL has ability to query multiple resources in single requests

- GraphQL it allows to more precise control over the test data that is retrived format.

4) Webhook API

- Webhook API is used in real time notifications when specific events is occurs.

- Webhook API is mostly used in social media applications.

**diff between SOAP and REST? VVVVVVIMMMMPPPPVVVVVVIMMMMPPPP**

customer API

Request type : POST

Base URI localhost:9191

Resource name : /api/v2/customers

Content-Type application/xml

Request body:

<Customer>

<caddress>string</caddress>

<gender>string</gender>

<name>string</name>

<state>string</state>

</Customer>

GET Specific Customer

Request type : GET

Base URI localhost:9191

Resource name : /api/v2/customers

Path Parameter /{id}

GET all Customer

Request type : GET

Base URI localhost:9191

Resource name : /api/v2/customers

GET all Customer using customer state

Request type : GET

Base URI localhost:9191

Resource name : /api/v2/customers

query Parameter :

state {value}

Request type : PUT

Base URI localhost:9191

Resource name : /api/v2/customers

Path Parameter /{id}

Content-Type application/xml

Request body:

<Customer>

<caddress>string</caddress>

<gender>string</gender>

<name>string</name>

<state>string</state>

</Customer>

Request type : DELETE

Base URI localhost:9191

Resource name : /api/v2/customers

Path Parameter /{id}

convert xml response body into the json response body

var respbody xml2Json(responseBody);

console.log(respbody);

//capture the values from converted response body and store in environment variable

pm.environment.set("CustId", respbody.Customer.id);

var respbody xml2Json(responseBody);

console.log(respbody);

pm.environment.set("CustId", respbody.Customer.id);

pm.test("TC for status code",function(){

pm.response.to.be.status(201);

});

pm.test("TC for status code",function(){

pm.response.to.be.status("Created");

});

|  |  |
| --- | --- |
| SOAP | REST |
| SOAP means simple Object Access Protocol | REST means Representational State Transfer |
| SOAP is protocol | REST is architectural style |
| SOAP support for XML (Extensive Markup) format | REST support for JSON (Java Script Object Notation) , XML , java scripts, html etc.. |
| SOAP is slower than REST | REST is faster |
| SOAP define the own securities | REST is inherit securities from other protocol |
| SOAP is less preferred | REST Is more preferred |
| SOAP can not be REST | REST can be SOAP means in REST we can borrow all securities from SOAP. |
| SOAP is more weight means we have write code from start to end | REST is light weight means we can write less code or can inherit the code from other technology |

|  |  |
| --- | --- |
| **Query Parameter** | **Path Parameter** |
| Query Parameter is filter or sort data as per the condition. | Path Parameter are used to identify the resource in server |
| Query Parameter we define using key-value pair | Path Parameter we define using double curly bracket or colon symbol then we define value in request level or environment level. |
| Query parameter is option in URL | Path Parameter are end points of the URL and it is not optional. |
| <http://localhost:3000/employees?firstName=Sujit> | [http://localhost:3000/employee/{{id}}](http://localhost:3000/employee/%7b%7bid%7d%7d) |

**Diff between Authorization and Authentication? VVVVVVIMMPPP**

|  |  |
| --- | --- |
| **Authentication** | **Authorization** |
| Authentication we have to check user is valid or not | Authorization we check user have access or not |
| If user enter invalid credentials, then we will get status code as 401 | If user don’t have permission, then we will get status code as 403. |
| **If user enter valid credentials** and if have access to the functionality then we will get status code as 200 | **If user enter valid credentials** and if we don’t have access to the functionality then we will get status code as 403. |
| Authentication comes before the Authorization | Authorization comes after the Authentication |

**Diff between 401 and 403 status code VVVVVVIMMPPP**

|  |  |
| --- | --- |
| **401 Status code** | **403 Status** |
| 401 means unauthorized | 403 means forbidden |
| If enter invalid credentials then we will get status code as 401 | If we enter valid credentials and if we don’t have permission to access the functionality then we will get 403 as status code. |

**Diff between OAuth 1.0 and OAuth 2.0 VVVVVVIMMPPP**

|  |  |
| --- | --- |
| **OAuth 1.0** | **OAuth 2.0** |
| OAuth 1.0 is complex protocol and require cryptographic signature to develop the application with OAuth 1.0 | OAuth 2.0 is simpler and more user friendly to compare with OAuth 2.0 |
| In OAuth 1.0 have infinite time for access token | In OAuth 2.0 have limited time for access token |
| In OAuth 1.0 has less security | In OAuth 2.0 has more security |
| In OAuth 1.0,uses single endpoints for all the request | OAuth 2.0 user different endpoints for all the request. |
| In OAuth 1.0 has limited access scope | In OAuth 2.0 has more access scope control |
| In OAuth 1.0, don’t have grant type permission | In OAuth 2.0, user resource owner (user/end user) password credentials for grant type. |

|  |  |
| --- | --- |
| **GET Request** | **POST Request** |
| It is used to retrieve the all or single entity from the server | It is used to create new entity in server |
| We never pass the request body in JSON/XML format GET Request | We need to pass request body in JSON/XML format in POST Request |
| GET Request status code is 200 | POST Request Status code is 201 |
| GET Request status line is 200 OK | POST Request status line is 201 Created |
| GET is not secured | POST Request is secured |
| GET Request is bookmarked | POST Request is cannot bookmark |
| We don’t need Content-Type header in GET Request | We need Content-Type header in POST Request |
| If we hit GET Request 100 times then it will retrieve same entity from server | If we hit POST Request 100 times then it will create 100 new entity with different id numbers. |

|  |  |
| --- | --- |
| **GET Request** | **PUT Request** |
| It is used to retrieve the all or single entity from the server | It is used to update whole entity in server |
| We never pass the request body in JSON/XML format GET Request | We need to pass request body in JSON/XML format in PUT Request |
| GET is not secured | PUT Request is secured |
| GET Request is bookmarked | PUT Request is cannot bookmark |
| We don’t need Content-Type header in GET Request | We need Content-Type header in PUT Request |
| If we hit GET Request 100 times then it will retrieve same entity from server | If we hit PUT Request 100 times then it will update same entity 100 times in server |

|  |  |
| --- | --- |
| **GET Request** | **PATCH Request** |
| It is used to retrieve the all or single entity from the server | It is used to update partial entity in server |
| We never pass the request body in JSON/XML format GET Request | We need to pass request body in JSON/XML format in PATCH Request |
| GET is not secured | PATCH Request is secured |
| GET Request is bookmarked | PATCH Request is cannot bookmark |
| We don’t need Content-Type header in GET Request | We need Content-Type header in PATCH Request |
| If we hit GET Request 100 times then it will retrieve same entity from server | If we hit PATCH Request 100 times then it will update same entity 100 times in server |

|  |  |
| --- | --- |
| **GET Request** | **DELETE Request** |
| It is used to retrieve the all or single entity from the server | It is used to delete entity from server |
| GET Request status code is 200 | DELETE Request Status code is 204 |
| GET Request status line is 200 OK | DELETE Request status line is 204 No Content |
| GET is not secured | DELETE Request is secured |
| GET Request is bookmarked | DELETE Request is cannot bookmark |
| If we hit GET Request 100 times then it will retrieve same entity from server | If we hit DELETE Request 100 times then first time it will delete the entity from server and next time it will return 404 error code that is Not Found |

|  |  |
| --- | --- |
| **POST Request** | **PUT Request** |
| It is used to create new entity in server | It is used to update whole entity from server |
| Status code for POST Request is 201 | Status code for PUT Request 200 |
| Status line for POST Request is 201 Created | Status line for PUT Request is 200 OK |
| We hit POST Request 100 times then it will create new 100 entity in server | If we hit PUT Request 100 times then it will update same entity 100 times in server |

|  |  |
| --- | --- |
| **POST Request** | **PATCH Request** |
| It is used to create new entity in server | It is used to update partial entity from server |
| Status code for POST Request is 201 | Status code for PATCH Request 200 |
| Status line for POST Request is 201 Created | Status line for PATCH Request is 200 OK |
| We hit POST Request 100 times then it will create new 100 entity in server | If we hit PATCH Request 100 times then it will update same entity 100 times in server |

|  |  |
| --- | --- |
| **POST Request** | **DELETE Request** |
| It is used to create new entity in server | It is used to delete entity in server |
| Status code for POST Request is 201 | Status code for DELETE Request 204 |
| Status line for POST Request is 201 Created | Status line for DELETE Request is 204 No Content |
| We hit POST Request 100 times then it will create new 100 entity in server | If we hit DELETE Request 100 times then first time it will delete the entity from server and next time it will return 404 error code that is Not Found |
| We send request body in JSON/XML format | We never send request body in JSON/XML Format |
| We need Content-Type header in POST Request | We don’t need Content-Type header in DELETE Request |

|  |  |
| --- | --- |
| **PUT Request** | **PATCH Request** |
| It is used to update whole entity in server | It is used to update partial entity in server |

|  |  |
| --- | --- |
| **PUT Request** | **DELETE Request** |
| It is used to update whole entity in server | It is used to delete entity in server |
| Status code for PUT Request is 200 | Status code for DELETE Request 204 |
| Status line for PUT Request is 200 OK | Status line for DELETE Request is 204 No Content |
| We hit PUT Request 100 times then it will update same entity 100 times in server | If we hit DELETE Request 100 times then first time it will delete the entity from server and next time it will return 404 error code that is Not Found |
| We send request body in JSON/XML format | We never send request body in JSON/XML Format |
| We need Content-Type header in PUT Request | We don’t need Content-Type header in DELETE Request |

|  |  |
| --- | --- |
| **PATCH Request** | **DELETE Request** |
| It is used to update partial entity in server | It is used to delete entity in server |
| Status code for PATCH Request is 200 | Status code for DELETE Request 204 |
| Status line for PATCH Request is 200 OK | Status line for DELETE Request is 204 No Content |
| We hit PATCH Request 100 times then it will update same entity 100 times in server | If we hit DELETE Request 100 times then first time it will delete the entity from server and next time it will return 404 error code that is Not Found |
| We send request body in JSON/XML format | We never send request body in JSON/XML Format |
| We need Content-Type header in PATCH Request | We don’t need Content-Type header in DELETE Request |

**What are the different types of authorization in API?**

There are different types of authorization in API

1. **Basic Authorization**
2. **Digest Authorization**
3. **Bearer token Authorization**
4. **OAuth Authorization**
5. **JWT Authorization**
6. **API key Authorization**
7. **Basic Authorization**

* It simple authentication schema built on HTTPs protocol where we share username and password in Base64 encoded format.
* After adding the username and password automatically postman tool will add in headers.

1. **Digest Authorization**

* - it is similar to basic Authorization but credentials are hashed before sharing to the server.
* After adding the username and password automatically postman tool will add in headers.

1. **Bearer token Authorization**

* **It** is token based authentication where token is passed in request header to grant the access of the API.

1. **OAuth Authorization**

* it is an protocol is allowed the secure standard authorization, and it includes the different grant type like authorization code, client credentials, resource owner password credentials etc.

1. **JWT Authorization**

* JWT mean JSON Web token.
* It is also known as bearer token for authentication.
* It is URL safe, means representing of data transfer between 2 different applications securely.

1. **API key Authorization**

* It is unique identifier password as part of the request headers to authenticate the API

**ghp\_y5R4mNfdBuNK9KKcSuw6bVnsML3ZOc1AQJWR**