http://makeseleniumeasy.com/2018/08/19/api-testing-tutorial-part-3-understand-terms-urn-url-uri-api/

Resources-> Names or locators

https://www.google.com/maps“. Google will launch Google Maps. Try same with “/news”. Google will launch Google News.

URI>URL

URI-> COMPLETE ADDRESS

URL-> NOT WITH RESOURCE

URN-> WITH RESOURCE

**CRUD**

http://makeseleniumeasy.com/2018/08/20/api-testing-tutorial-part-4-understand-basics-of-http-methods-crud-operations/

GET -> 404 ( Not Found) –> If GET API does not find the requested resource.

200 (OK) –> If GET API finds the requested resource.

400 ( Bad Request) –> If GET request is not formed properly.

Safe Method

**POST**

it returns a status code 201( Created) (Not always) and returns response body.

It may return 200 (OK) and 204 (No Content) status code as well based on how it is created.

*It is also not idempotent* *and invoking two identical POST requests will result in two different resources containing the same information with just different resource ids.*

**PUT**

If PUT request is made to update resource, it should return 200 (OK) and 204 (No Content) status code.

If PUT request is made to create a new resource, it must return a status code 201( Created).

**DELETE**

It returns 200 (OK) and 204 (No Content) status code.

It may return as 202 (Accepted) status code if request is queued.

**PATCH**

An HTTP PATCH method is used to update information of resource partially.

**OPTIONS-**

An HTTP OPTIONS method which is used to get information about allowed operations on given URI.

**HEAD**

header information no resposne bod

Headers represent the metadata of API requests and responses.

Headers” or “HTTP Headers” are key – value pairs which are used by Server and Client to exchange additional information about Request and Response.

We can also say that headers are metadata of Request and Response.

“Accept” and “Content-Type” are HTTP Headers which control input and output formats.

“Accept” is a request header and “Content-Type” is both request and response header.

Accept-> Here is my request and I would like (to Accept) this response format

Content-Type-> Here is my request (or response) and this (Content-Type) is the format of the content

I am sending in my request (or response)

**PUT REQUEST**-> to update any value in json. **Complete payload should be provided.** It will replace the existing payload or create a new data

**401 Unauthorized:**

If the request already included Authorization credentials, then the 401 response indicates that authorization has been refused for those credentials.

**403 Forbidden:**

The server understood the request, but is refusing to fulfill it.

Authrization

OAuth 2.0

cookie

bearer token

**PUT AND PATCH DIFFEENCE**

It is different from PUT as PUT updates/replace complete information of resource while PATCH updates some information of resource

**Variables defined in an Environment are Local variables.**

It means you can use variables of an environment in another environment. So we have another type of variables called Global variables.

Status code to remember

http://makeseleniumeasy.com/2018/09/01/api-testing-tutorial-part-7-http-status-code-must-to-be-remembered/

*Now we will see some status codes which we must know.*

1. **200 (OK)** => All is GOOD. It is the most positive status code which everyone expects. This status code is thrown when requested operation on server by client is successfully processed.
2. **201 (Created)** ==> This status code is thrown when an HTTP method to hit to create a new resource on server and resource is created successfully. This status code makes more sense for HTTP methods which are meant to create new resources.
3. **202(Accepted)** ==> If a request is queued for processing or takes longer time to process, this status code is thrown to client. The request might or might not be eventually acted upon, or even maybe disallowed when processing occurs.
4. **204 (No Content)** ==> When a request is processed successfully but returns no state representation of resource to be included in the response message body, throws 204 status code. Its success without response body.
5. **400 (Bad Request)** ==> It is a client side error made when user submits inappropriate request like malformed request syntax, invalid request message parameters, or deceptive request routing etc. Client should correct request before hitting again.
6. **401 (Unauthorised)** ==> When a client submits a request with no or wrong authorisation on a resource which is protected by authorisation , this status code is thrown. The response must include a WWW-Authenticate header field containing a challenge applicable to the requested resource.
7. **403 (Forbidden)** ==> When a user tries to access a resource on which user has no permission, this error will be thrown. It is not like 401. Request and proper authorization is provided in request body but user has no access on request resource.
8. **404( Not Found)** ==> If the request resource is not available at given URI or rest api can’t map the client’s URI to a resource, this status code is thrown.
9. **405( Method Not Allowed)** ==> When a client calls an HTTP method on a resource which is not applicable to on it, this status code is thrown. For example: If client hits a POST method on a GET resource, it will throw 405. Response includes  “Allow” header, which lists the HTTP methods that the resource supports.
10. **406(Not Acceptable)** ==> If API can not format data as per client’s provided media types in Accept request header, this status code is thrown.  For example, a client request for data formatted as application/xml will receive a 406 response if the API is only willing to format data as application/json.
11. **415 (Media Type)** ==> If API is not able to process the client’s supplied media type, as indicated by the Content-Type request header, this status code is thrown. It indicates that part of the request was in an unsupported format. For example, a client request including data formatted as application/xml will receive a 415 response if the API is only willing to process data formatted as application/json.
12. **500 (Internal Server Error)** ==> It is server side error. When server encounters an exception while processing a request, this status code is thrown. Please note here it is not client side error.
13. **501 (Not Implemented)** ==> When the server does not recognize the request method and is not capable of supporting it for any resource, this status code is thrown. The server either does not recognize the request method, or it lacks the ability to fulfill the request.

### **Six principles of REST:**

**Client–server** **architecture**

1. **Statelessness**
2. **Cacheability**
3. **Uniform interface**
4. **Layered system**
5. **Code on demand**

SOAP uses JAX-WS java API while REST uses JAX-RS

Scripts in postman

*Postman provides****two events where we can write JavaScript codes****to execute in the flow:-*

* ***Pre-request Script*** :- It is executed before a request is sent to server or end points.
* ***Test scripts:-*** *It is executed once response is received from server.*
* *As we know now that Pre-request script will be executed before request is hit to server and tests script will be executed once response is received.*

Prerequest>request>response>tests scripts

Introduction To Postman Sandbox

*The Postman Sandbox is a JavaScript execution environment based on NodeJS that is available to you while writing pre-request and test scripts for requests both in Postman and Newman.*

*Postman Sandbox has many built in JavaScript libraries which we can use while writing scripts. It makes Postman a very powerful tool to write automated scripts for API.*

*Debugging and logging*

*View in Console in postman*

Set, Get And Unset Global & Environment Variables In Postman Scripts

*We can set the variables values manually also.*

*But Sometimes we may need to update or add variables or values at runtime.*

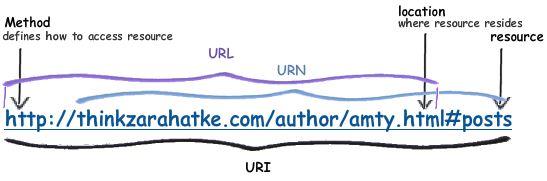
*We can add, update, get or remove global and environment variables at runtime using scripts in Postman. We use a global function named “pm.\*” of Postman Sandbox environment. We can do it in both Pre-request Script and Tests.*

*that “pm” is a global object in Postman which has a method called “test” which has below syntax:*

*For example:- You are creating a user and that user should be used for other APIs. We need to extract user details from response and add to environment. This needs to be done at runtime only.*

*Prerequest script we set environment and In tests we read it*

*URI, URN, URL*

**

***Auth1.0, Auth2.0***

**More OAuth Flows to allow better support for non-browser based applications.** This is a main criticism against OAuth from client applications that were not browser based. For example, in OAuth 1.0, desktop applications or mobile phone applications had to direct the user to open their browser to the desired service, authenticate with the service, and copy the token from the service back to the application. The main criticism here is against the user experience. With OAuth 2.0, there are now new ways for an application to get authorization for a user.

**OAuth 2.0 no longer requires client applications to have cryptography.** This hearkens back to the old Twitter Auth API, which didn't require the application to HMAC hash tokens and request strings. With OAuth 2.0, the application can make a request using only the issued token over HTTPS.

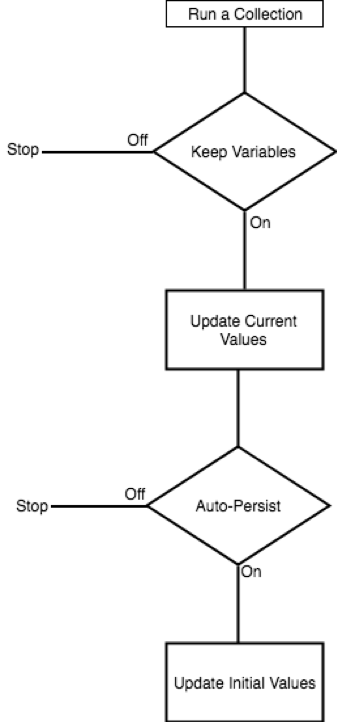
**OAuth 2.0 signatures are much less complicated.** No more special parsing, sorting, or encoding.

**OAuth 2.0 Access tokens are "short-lived".** Typically, OAuth 1.0 Access tokens could be stored for a year or more (Twitter never let them expire). OAuth 2.0 has the notion of refresh tokens. While I'm not entirely sure what these are, my guess is that your access tokens can be short lived (i.e. session based) while your refresh tokens can be "life time". You'd use a refresh token to acquire a new access token rather than have the user re-authorize your application.

**Finally, OAuth 2.0 is meant to have a clean separation of roles between the server responsible for handling OAuth requests and the server handling user authorization.** More information about that is detailed in the aforementioned article.

 OAuth 2.0 has only one security token, and no signature is required.

**Collections Runner Environment**



**Extracting And Asserting Request & Response Headers In Postmam**

*To get all headers of Request as a list :-*

*pm.request.headers : HeaderList*

*To get all headers of Response as a list :-*

*p*m.**response.headers : HeaderList**

To get a specific header value by name:-

*pm.response.headers.get(HeaderName)*