**API :** application programing interface

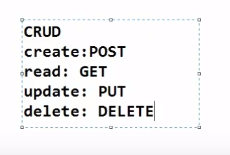
Now REST API is in demand because it offers more benefits over Soap AIPs. Moreover, 90% rest APIs use JSON format instead of XML format. Json is lighter and writing code in JSON is easy as compared to XML.

Two different applications can communicate with each other with the help of AIP.  
It is like a request and response.  
Ex. I can use Citi bank ATM to withdraw money from BOM or BOI account.

There are service providers and consumers   
A Consumer sends request and provider processes and sends back the response. It needs a platform to exchange the information. We need a medium and a format for this.   
Ex. : Medium : http and format : XML / JSON (which both the consumer and the provider understands)

 //////// REST API //////////

Rest API has the below requests :



REST API follows **only**the below-mentioned **requests**-- >

GET : When we fetch data or when we request data from other resources then use GET

POST : When we add new data or create a new entry then use post (ex. create a new user in an application)

PUT : Update the information which was saved before

Delete : to delete

 \*\*\* GET \*\*\*  
GET requests are the most common and widely used methods in APIs and websites. Simply put, the GET method is used to retrieve data from a server at the specified resource. For example, say you have an API with a /users endpoint. Making a GET request to that endpoint should return a list of all available users.  
Since a GET request is only requesting data and not modifying any resources, it's considered a safe and idempotent method.

Testing an API with GET requests  
When you're creating tests for an API, the GET method will likely be the most frequent type of request made by consumers of the service, so it's important to check every known endpoint with a GET request.  
At a basic level, these things should be validated:  
Check that a valid GET request returns a 200 status code.  
Ensure that a GET request to a specific resource returns the correct data. For example, GET /users returns a list of users.  
GET is often the default method in HTTP clients, so creating tests for these resources should be simple with any tool you choose.

In get, we get  status code after sending is : 200 OK. that means data fetched successfully.

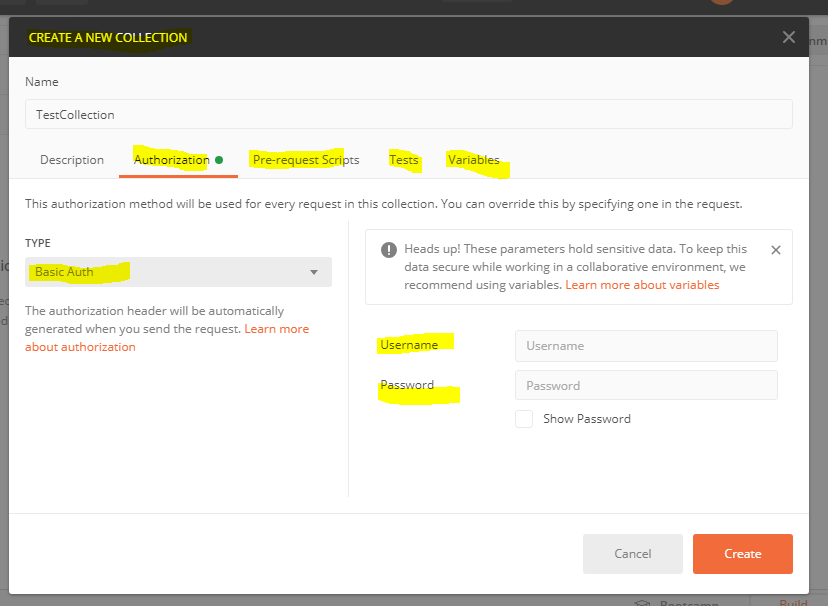
\*\*\* POST \*\*\*

Here we have to send body with the POST URL that will send the data. The status code after sending is : 201 Created. I.e. successfully a new record created.

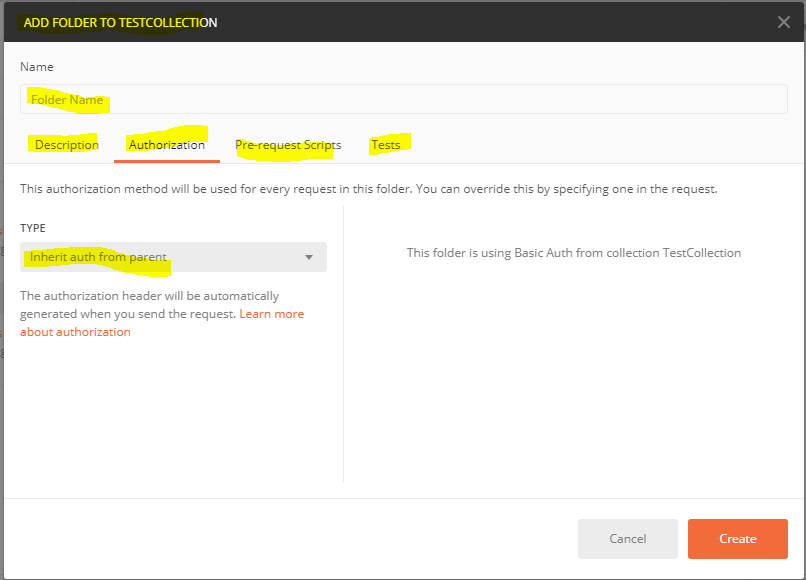
POSTMAN ----->  [https://www.youtube.com/watch?v=ENNsL-XGLus](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DENNsL-XGLus&data=04%7C01%7Cpratik.toke%40fisglobal.com%7C30e8c596a6fd4526efd708d974577483%7Ce3ff91d834c84b15a0b418910a6ac575%7C0%7C0%7C637668741252263742%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=FuZnTqSpbdAbFOHXOfWLrfCT71Oz4yuEjrbFJwbEyew%3D&reserved=0)

**Collections**: It is a group of API requests. Where APIs can be stored and saved in a logical order / arrangement.

We can create folders in a collection. We run the entire collection OR we can also run a specific file in a collection. We can mention Authorization > authentication, pre-request, test, variables etc. at the collection level. When we execute the entire collection these settings will be applied to the entire collection.

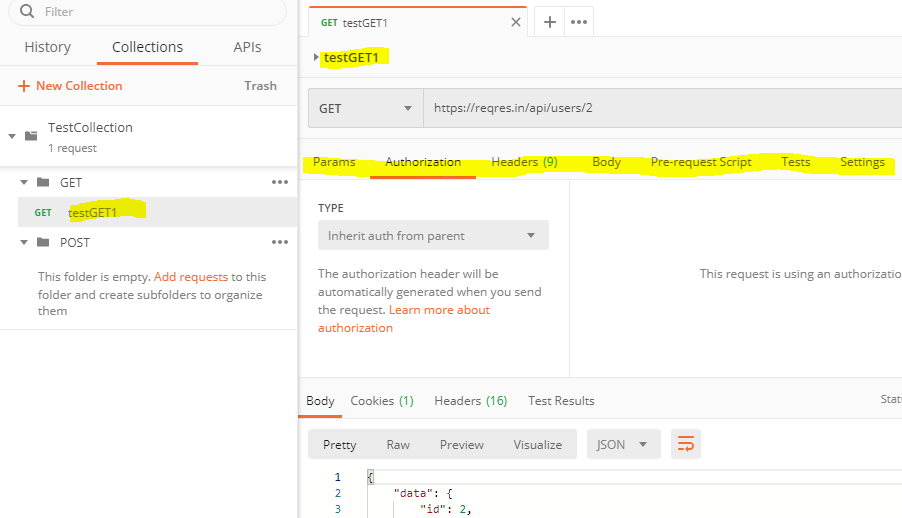


We can create folders inside collection and while creating folders, we can mention Authorization > authentication, pre-request, test.



Inside a folder, we can add individual requests (GET or POST etc.)

INDIVIDUAL requests can have : authorization > authentication type, headers, body, pre-requisite scripts, tests, settings.

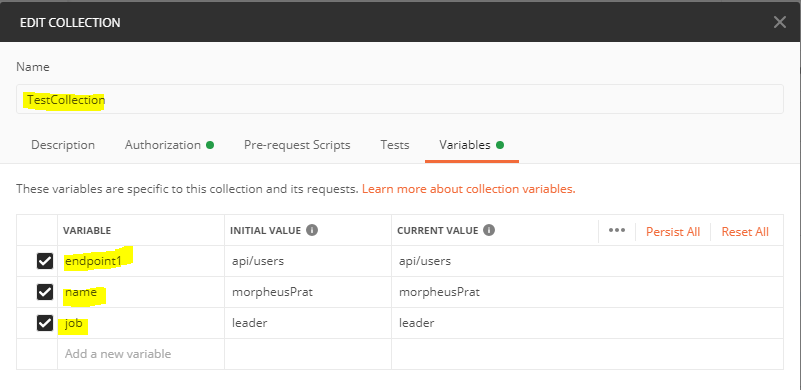


**variables**: use to store variables. It is like the variables we use in a programming language. We can reuse the values at multiple places.

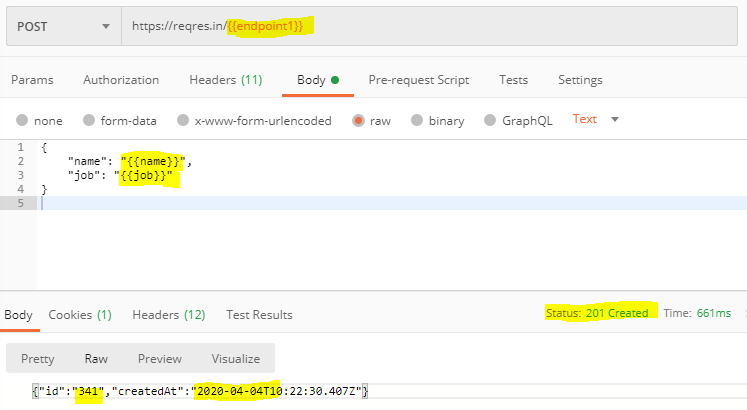
We can create a variable in the collection. Ex. if we have multiple urls that starts with the same domain name then we can store that name in a varible and we can use it in get and post URLs ex. variable pp = [https://abc.com/](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fabc.com%2F&data=04%7C01%7Cpratik.toke%40fisglobal.com%7C30e8c596a6fd4526efd708d974577483%7Ce3ff91d834c84b15a0b418910a6ac575%7C0%7C0%7C637668741252273696%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=lKuA%2B9etf4%2B4OqFX1lsjKD3CJ4%2BgSopWxWSqDvYljQU%3D&reserved=0) and we can use pp instead of using this domain name. we should use brackets while using the variable. ex. {{pp}}user/login?page=2. we can create environment level variables as well. go to gear icon(manage environment) in RHS corner. click on global and create a variable. View > show postman consol > we can see here the value used by the variable at run time.

We can create variables at different levels: 1. collection level variables 2. Environment level.

Here I have created variables at collection level



And I have used it in the POST request

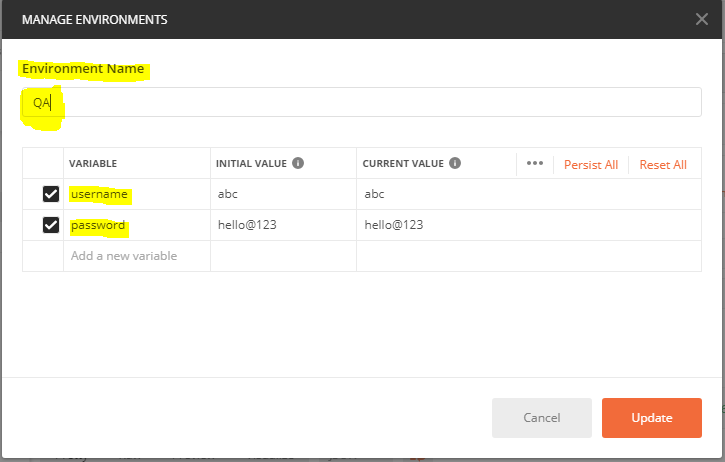


**Environment**:    
It is a set of key-value pair in postmen and we can refer it in our tests.  
In real-time testing we test on different environments. for ex. QA, Dev etc. so for each environment, we have different Uname, Pwd, URL etc. So there are certain values that varies according to the environment. The changing values can be grouped together and can be added to the environment.  
 New > Environment. Add key and values i.e. the values that change according to the environment. Ex. endpoint, Uname, Pwd etc.  
 Now create a normal Post / get request in the environment and in 'Body > Raw' enter user name and pwd in script e.g. : { "email" :"{{email}}"  "password" :"{{password}}"}  
 Post / Get URL : {{URL}}/abs/page=2. Before running a test, select the environment near eye button (RHS corner)

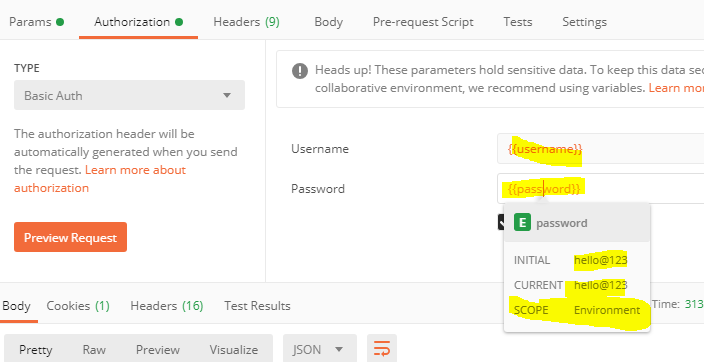
We can have multiple scripts in postman. Collection > edit OR request > edit will open it in edit format. There are two tabs 1. Pre-requiset script : This script will execute before every request in this folder, wereas Tests : These tests will execute after every request in this folder.

We can choose the environment on which we want to run tests.

Setting env. variables :



Using env. level variables.



**Test**:

We can create scripts at different levels : 1. At the collection level (edit collection > pre-requisite scripts and Test)  2. At request level (open request> pre-requisite scripts and Test ) 3.At folder level (edit folder > pre-requisite scripts and Test )

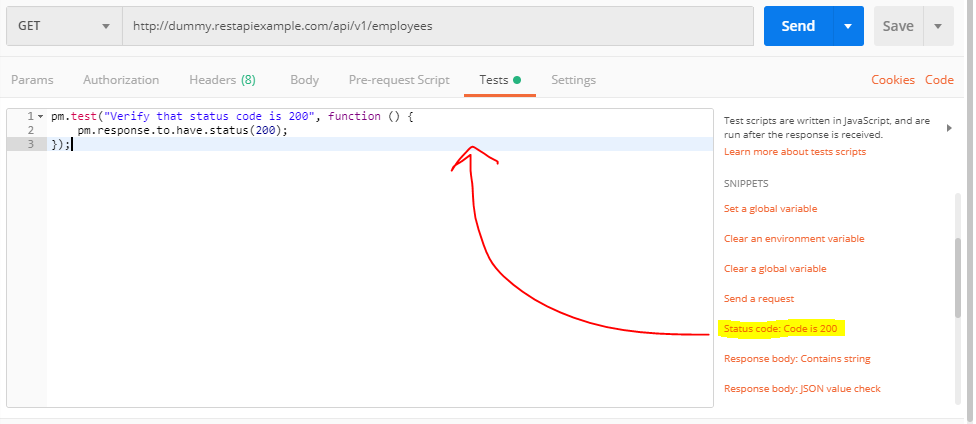
Postman tests are javaScript code that is executed **after**receiving the response from the server

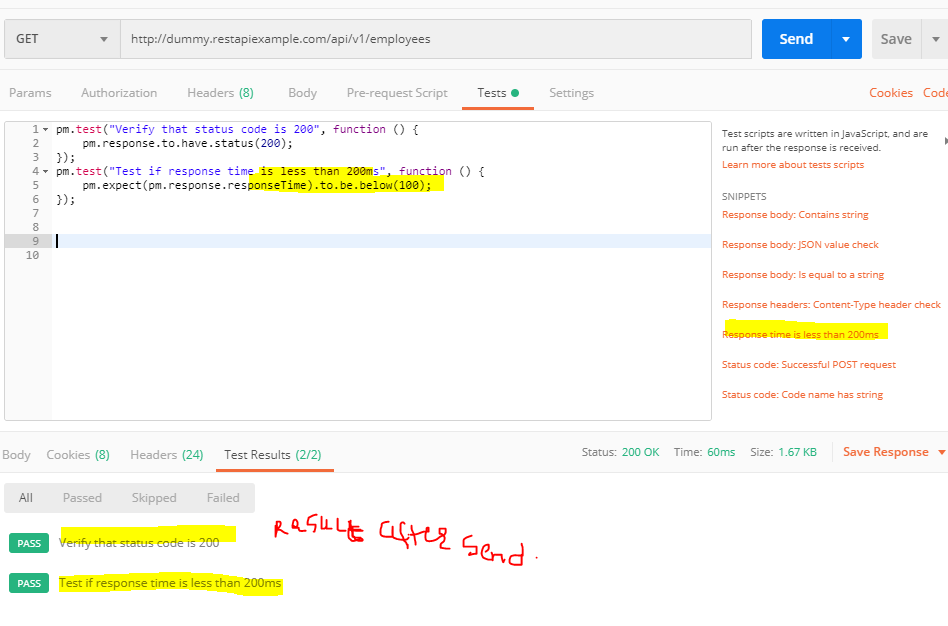
Ready-made code snippets are also available and we can use that as well. We can use predefined functiones to test whether specific string / values are available in the response or not. ex.

pm.test("Body matches string", function ()

{    pm.expect(pm.response.text()).to.include("Expectd String");

});



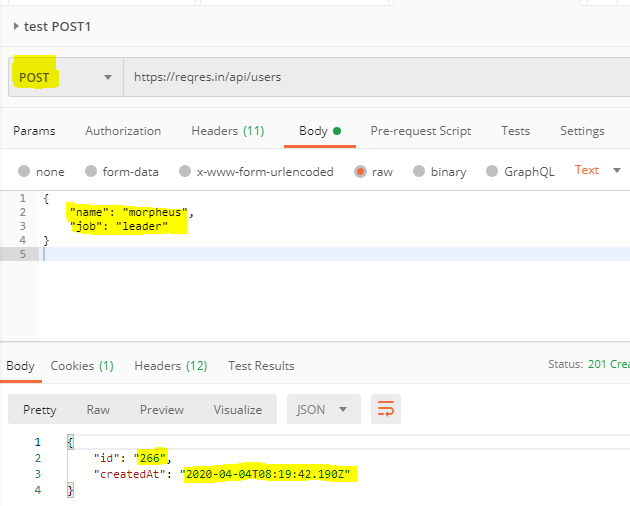


We can see everything on the consol. View > Show postman consol.

**POST**requests :

In POST requests we create some new entries instead of fetching already created entries / records.

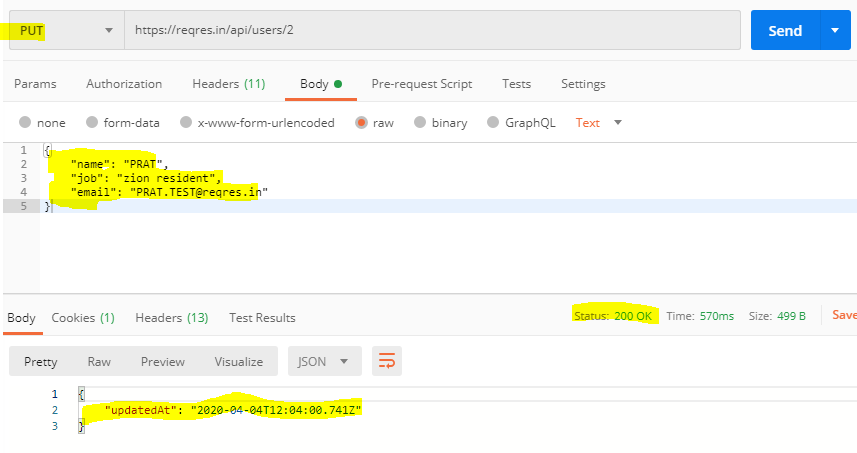
We have to provide **Body (entries to be made)**and then we have to check the response.



In the above example, in a POST request > Body > Raw : we have entered the data we want to enter in the system. Data got added successfully. We got 201 ststus + we got response  "id": "266" : this is the recoreID created in the system. We can now check it in the database.

+ "createdAt": "2020-04-04T08:19:42.190Z" : we got this response as well.

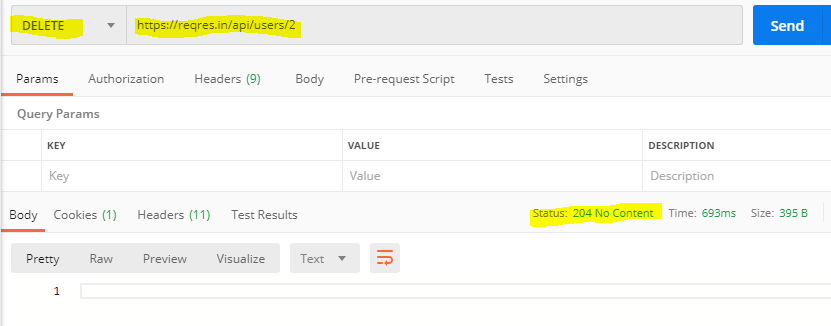
**PUT** request :



we get 200 OK status slong with the UpdatedAt : date time as a confirmation that in this date and time the information got updated.

**Delete** request :

We can delete entries :



we get 204 : No content message as a confirmation that entry got deleted.

STATUSES for REQUESTS  :

1. Get : status : 200OK

2. PUT : status : 200OK

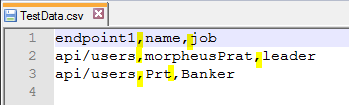
3. POST : status : 201Created

4. DELETE : status : 204No Content

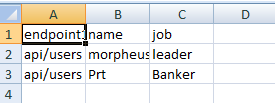
Postman from jenkins :   
[https://www.youtube.com/watch?v=COT6-uVD1-c](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DCOT6-uVD1-c&data=04%7C01%7Cpratik.toke%40fisglobal.com%7C30e8c596a6fd4526efd708d974577483%7Ce3ff91d834c84b15a0b418910a6ac575%7C0%7C0%7C637668741252273696%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=%2FDcmDPB%2FJ7eXGd02lOxqmCHU6l%2FK0QuD4wzma6%2FztOo%3D&reserved=0)

Get / Read sata from CSV or JSON  
  
Uname : [eve.holt@reqres.in](mailto:eve.holt@reqres.in)  
Pwd : pistol  
URL : api/register/prepod  
  
Click on Collections side arrow > Run > Select file. choose file and check the preview. then run it. If issue then open it with notepad ++ and add a comma at the end of each value.

Once we open a file in notepad++, it should look like :



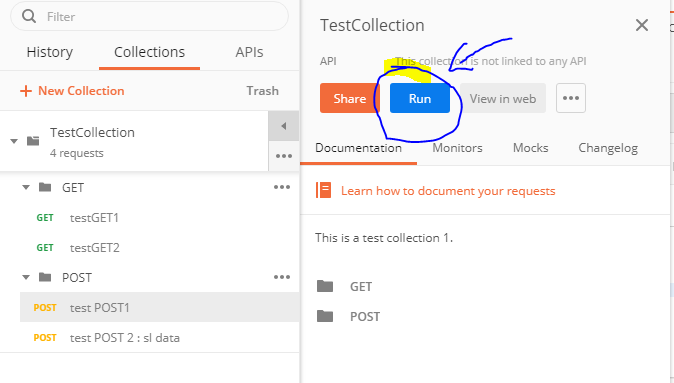
and CSV should look like this :



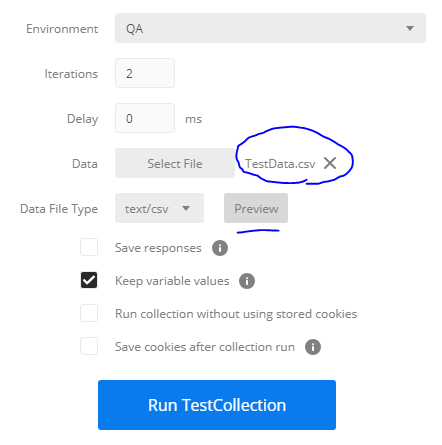
Get a link for your collection to run it :

Click on the collection arraow > share > get public link now generate and copy the link. On command prompt we can run this collection with the help of this URL

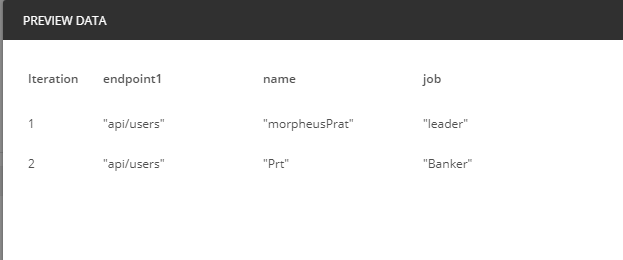
To run it directly with the help of Postman tool > Collections arrow > RUN > Select file > preview and then RUN. ----------- >>



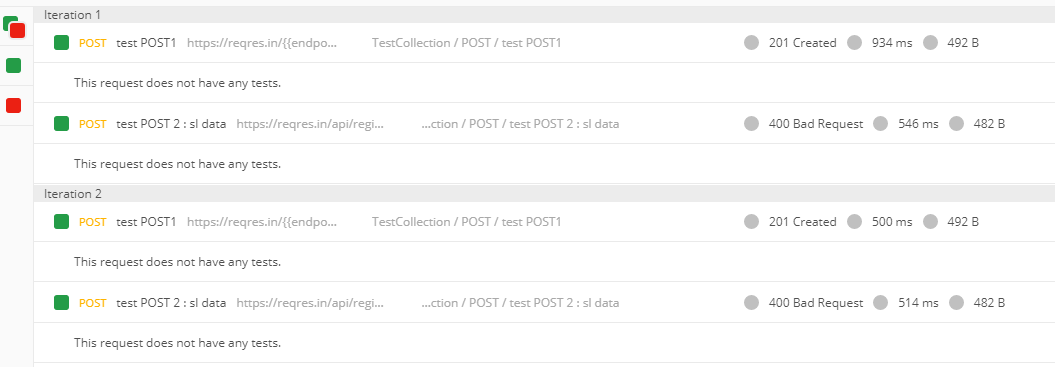
Upload file :



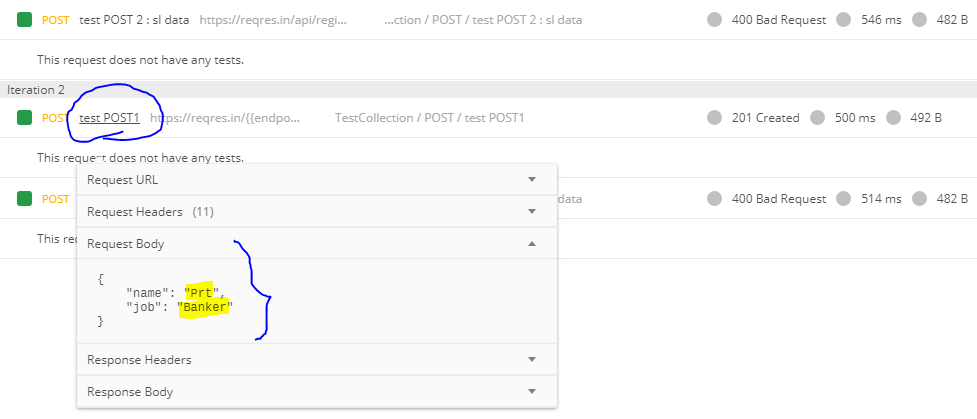
preview :



After running it shows result like this :



We can check our data picked from CSV here : Open the above page : click on the link shown in the below screenshot > Request Body section shows data



POSTMAN BEGINNER TUTORIAL 11 How to DEBUG

CACD : we use Oracle database and we use Oracle -SQL developer to access the database.

1. select \* from Customers;
2. select CustomerName, Address from Customers;
3. select \* from  Customers where  CustomerName = 'pratik' and  Address = 'abc';
4. select \* from Customers where CustomerID = 8 and Country = 'Spain' and City = 'Madrid'; (DONT GIVE COMMA ALWAYS USE AND)
   * The AND operator displays a record if all the conditions separated by AND are TRUE.
   * The OR operator displays a record if any of the conditions separated by OR is TRUE.
5. select \* from Customers where CustomerID = 8 or Country = 'Spain' and City = 'Madrid';
6. select PostalCode, City from Customers order by CustomerID asc;
7. select PostalCode, City from Customers order by PostalCode asc;
8. insert into Customers (customerID, contactName) values (1001,'Prt');
9. select max(CustomerID) from Customers where contactName = 'Prt';
10. select min(CustomerID) from Customers ;
11. select count(CustomerID) from customers;
12. select avg(customerID) from customers;
13. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

* + % - The percent sign represents zero, one, or multiple characters
  + \_ - The underscore represents a single character

1. select city from customers where city like 'B%';
2. select ContactName from customers where ContactName like 'a%';
3. select \* from Customers where city like'b%';
4. select \* from Customers where city not in ('Berlin', 'Bern');
5. select \* from Customers where city in ('Berlin', 'Bern');
6. select \* from products where price between 10 and 20; -- here 10 and 20 both inclusive
7. INNER JOIN :  
   This type of join returns those records which have matching values in both tables. So, if you perform an INNER join operation between the Employee table and the Projects table, all the tuples which have matching values in both the tables will be given as output.
8. select Customers.CustomerName, Customers.Country, Orders.OrderDate, Orders.OrderID from customers **INNER JOIN** orders on Customers.CustomerID = Orders.CustomerID;  ---  customers INNER JOIN orders OR  orders INNER JOIN customers  > gives the same result.
9. Left Join :   
   The LEFT JOIN keyword returns all records from the left table (table1)(i.e. all column values of the column/s mentioned in the query), and the matched records from the right table (table2). The result is NULL from the right side if there is no match.
10. ex. : select Customers.CustomerName, Orders.OrderID, Orders.OrderDate from Customers **left join** orders on Customers.CustomerID = orders.CustomerID; (here all column values of Customers.CustomerName will be fetched and if there is no result from the right side then NULL will be printed in front of those values.)
11. Right Join :   
    The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.
12. Ex. : select Customers.CustomerName, Customers.ContactName, Orders.OrderDate from Customers **right join** orders on Customers.CustomerID = orders.CustomerID;
13. FULL OUTER JOIN / FULL JOIN :   
    The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.  The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.
14. Ex. : SELECT Customers.CustomerName, Orders.OrderID FROM Customers **FULL OUTER JOIN** Orders ON Customers.CustomerID=Orders.CustomerID
15. GroupBy :  The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".  here we can group-by country   
    SELECT CustomerID, Country  
    FROM Customers GROUP BY Country
16. Having :   
    SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country HAVING COUNT(CustomerID) > 5;
17. Distinct :   
    The SELECT DISTINCT statement is used to return only distinct (different) values.  
    Inside a table, a column often contains many duplicate values then   SELECT DISTINCT eliminates duplicate records from the results.    
    ex. :  SELECT DISTINCT Country FROM Customers;
18. SUB Query  :  
    Always sub-query executes first and that result is used to generate the result from the outer query  
    Ex. : select \* from OrderDetails where OrderId in (select OrderId where productID > 70);  
    select \* from Suppliers where SupplierID in (select SupplierID where Country like 'USA');  
    Select \* from OrderDetails where OrderDetailID in (select OrderDetailID from OrderDetails where ProductID > 50 and Quantity > 40);
19. Limit / Top :   
    Limit : used in Mysql : to fetch limited records in DB  
    Top :  used in Microsoft SQL  : to fetch top records in DB

Interview SQL :

1. Second highest sal using MAX :   
   a. select max(Quantity) from OrderDetails where Quantity not in (select max(Quantity) from OrderDetails)   
   b. select max(orderID) from OrderDetails where OrderID < (select max(orderID) from OrderDetails);  
   c. select max(Price) from products where price < (select max(Price) from products);
2. second and third highest salary : / **N**th HIGHEST SAL.  
   a. SELECT Fname, Lname  
   FROM Employee  
   ORDER BY Salary  
   Limit 3  
   OFFSET 10;  
   ( OFFSET 10 : Start from here i.e. it start showing from 11th row till : 13th row i.e. 11, 12, 13 )
3. b.\*\*\* EASY SOLUTION : select distinct (salary) from employee order by salary desc LIMIT 2,1 : 3rd largest salary   > in the above query change DESC to ASC if want smallest sal.  
   c. GENERIC query :   select distinct (salary) from employee order by salary desc LIMIT (n-1,1) : nth largest salary    
   c. Ex. : SELECT distinct(Price) FROM Products order by Price desc limit 2,1;
4. Select TOP 1ST salary :   
   a. SELECT top 1 EmployeeID FROM employee order by salary desc;  
   b. Select max(Salary) from employee;  
   c. SELECT \* from customers limit 1 offset 1;
5. PRINT even / odd/ alternate rows of table :   
   a.  SELECT \* FROM Products where ProductID%1=0;   //prints odd rows   
   b.  SELECT \* FROM Products where ProductID%2=0;   //prints even rows
6. Print all **DUPLICATE**rows available in table :   
   a. select OrderID, CustomerID, EmployeeID, **count(ShipperID)** from Orders group by **ShipperID**having **count(ShipperID**) > 1; ///// here all duplicate ShipperID will be listed  
   b. select OrderID, CustomerID, EmployeeID, ShipperID, **count(\*)** from Orders group by **CustomerID**having **count(\*) > 1**;  ///// here all duplicate CustomerID will be listed
7. Select **1st and last N** rows :   
   a. 1st 5 : Select \* from orders order by orderId desc limit 5;   ///// here 1st top 5 records will be displayed  
   b. Last 5 : Select \* from orders order by orderId asc limit 5;   ///// here Last top 5 records will be displayed   
   c. 1st 5 without any order : Select \* from orders limit 8;   ///// here without sorting 1st 8 records will be displayed
8. Select Nth row from table :   
   a. select \* from Orders order by orderId limit 2 offset 10;    //// start displaying 11th and 12th  
   b. select \* from Orders order by orderId limit 1 offset 5;    //// only 6th Row will  be displayed

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  offset n : start displaying **from**the **n+1 th element**,.

1. INTERSECTION :   
   a.  The SQL INTERSECT operator is used to return the results of 2 or more SELECT statements. However, it only returns the rows selected by all queries or data sets. If a record exists in one query and not in the other, it will be omitted from the INTERSECT results.    
   b. Ex. :  SELECT CustomerID FROM Orders INTERSECT SELECT CustomerID FROM customers;

 \*\*\* start with vector programming. then all selenium write

\*\* kalyani interview Q., apache poi > data read xcel

db joins write \*  
java basic programs : oops \*

collections program, write

java other like static etc.

int. questions

sel framework, reporting tool

sel prog write

\*\*\*\*\*backend automation : read data from API or from DB

\*\*\*\*\* jenkins and git. selenium config

\*\*\*\* basic java programs

\*\*\*\* Jmeter

REMAINING :

1. Nested query \*\*\*\* IMP

2.

Attachments area

[Preview YouTube video POSTMAN BEGINNER TUTORIAL 6 💡VARIABLES | How to create and refer](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DENNsL-XGLus%26authuser%3D1&data=04%7C01%7Cpratik.toke%40fisglobal.com%7C30e8c596a6fd4526efd708d974577483%7Ce3ff91d834c84b15a0b418910a6ac575%7C0%7C0%7C637668741252283654%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=D0pV%2Bya5Udacv5Mg6VStla70Do0YZdRTZrJKxrwkF2Q%3D&reserved=0)

[](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DENNsL-XGLus%26authuser%3D1&data=04%7C01%7Cpratik.toke%40fisglobal.com%7C30e8c596a6fd4526efd708d974577483%7Ce3ff91d834c84b15a0b418910a6ac575%7C0%7C0%7C637668741252283654%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=D0pV%2Bya5Udacv5Mg6VStla70Do0YZdRTZrJKxrwkF2Q%3D&reserved=0)

[](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DENNsL-XGLus%26authuser%3D1&data=04%7C01%7Cpratik.toke%40fisglobal.com%7C30e8c596a6fd4526efd708d974577483%7Ce3ff91d834c84b15a0b418910a6ac575%7C0%7C0%7C637668741252283654%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=D0pV%2Bya5Udacv5Mg6VStla70Do0YZdRTZrJKxrwkF2Q%3D&reserved=0)

[**POSTMAN BEGINNER TUTORIAL 6 💡VARIABLES | How to create and refer**](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DENNsL-XGLus%26authuser%3D1&data=04%7C01%7Cpratik.toke%40fisglobal.com%7C30e8c596a6fd4526efd708d974577483%7Ce3ff91d834c84b15a0b418910a6ac575%7C0%7C0%7C637668741252283654%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=D0pV%2Bya5Udacv5Mg6VStla70Do0YZdRTZrJKxrwkF2Q%3D&reserved=0)

[Preview YouTube video POSTMAN BEGINNER TUTORIAL 13 💡 How to run from JENKINS](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DCOT6-uVD1-c%26authuser%3D1&data=04%7C01%7Cpratik.toke%40fisglobal.com%7C30e8c596a6fd4526efd708d974577483%7Ce3ff91d834c84b15a0b418910a6ac575%7C0%7C0%7C637668741252283654%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=QRuNu1nmrLbD25rEEy1YNJvAMDafYvOdHGcY6MlFUms%3D&reserved=0)

[](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DCOT6-uVD1-c%26authuser%3D1&data=04%7C01%7Cpratik.toke%40fisglobal.com%7C30e8c596a6fd4526efd708d974577483%7Ce3ff91d834c84b15a0b418910a6ac575%7C0%7C0%7C637668741252283654%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=QRuNu1nmrLbD25rEEy1YNJvAMDafYvOdHGcY6MlFUms%3D&reserved=0)

[](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DCOT6-uVD1-c%26authuser%3D1&data=04%7C01%7Cpratik.toke%40fisglobal.com%7C30e8c596a6fd4526efd708d974577483%7Ce3ff91d834c84b15a0b418910a6ac575%7C0%7C0%7C637668741252283654%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=QRuNu1nmrLbD25rEEy1YNJvAMDafYvOdHGcY6MlFUms%3D&reserved=0)

[**POSTMAN BEGINNER TUTORIAL 13 💡 How to run from JENKINS**](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DCOT6-uVD1-c%26authuser%3D1&data=04%7C01%7Cpratik.toke%40fisglobal.com%7C30e8c596a6fd4526efd708d974577483%7Ce3ff91d834c84b15a0b418910a6ac575%7C0%7C0%7C637668741252283654%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=QRuNu1nmrLbD25rEEy1YNJvAMDafYvOdHGcY6MlFUms%3D&reserved=0)



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | Pratik Toke <[pratik.toke07@gmail.com](mailto:pratik.toke07@gmail.com)> | | Feb 23, 2020, 3:32 PM |  |  |
| |  | | --- | | to Pratik, pratik | | | |

What is web services :   
When two machines or application exchange information with each other over a network then it is called as Web-services. This communication happens only on a network.   
Whereas API also used for the same purpose but API can also be used without network and it can only help to build or support any application . For ex. Selenium API are used to build a selenium test framework.

**API vs Web Service**

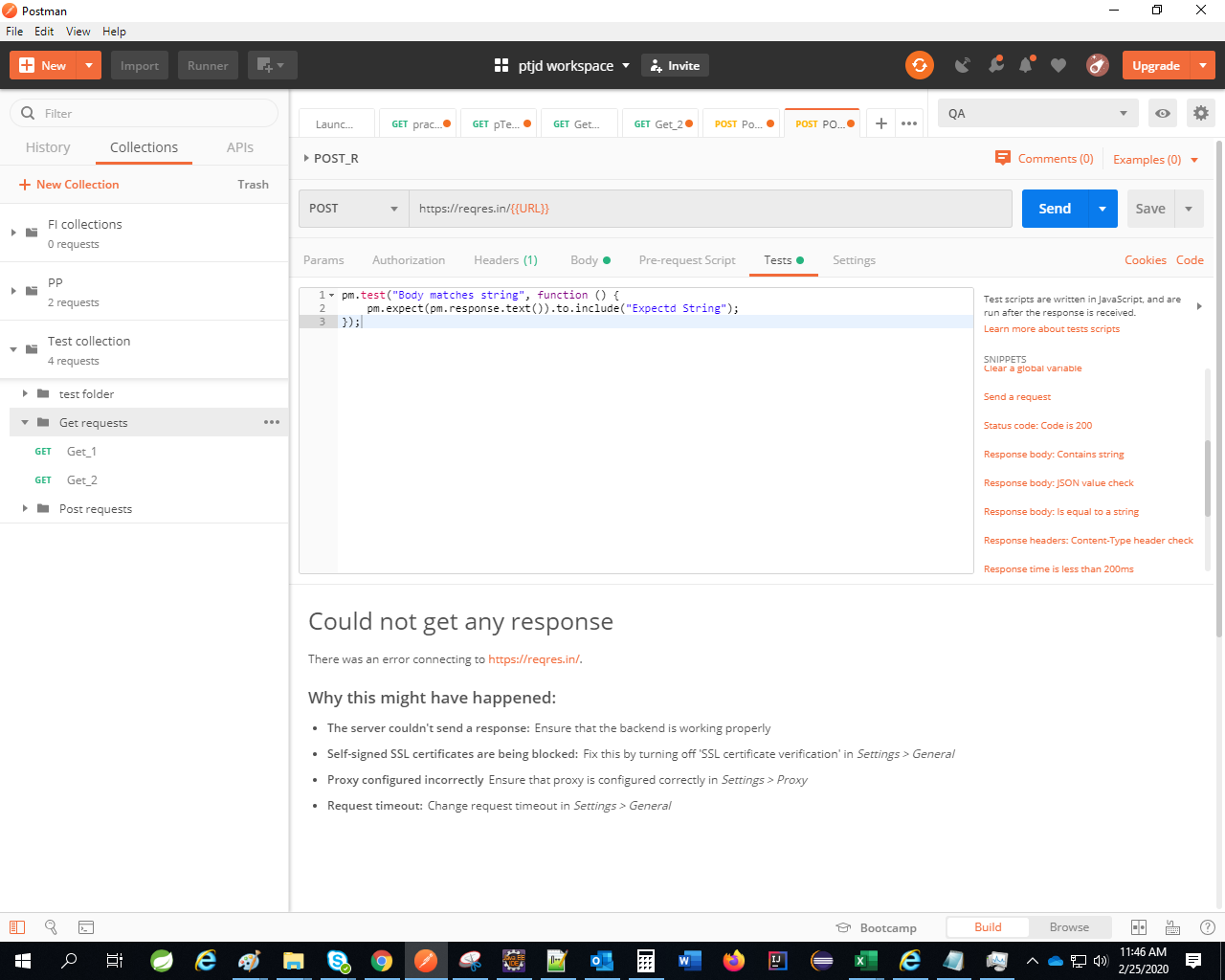
API and Web service serve as a means of communication. The only difference is that a Web service facilitates interaction between two machines over a network. An API acts as an interface between two different applications so that they can communicate with each other. An API is a method by which the third-party vendors can write programs that interface easily with other programs.

REST & SOAP are the two types of web-services 





|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | Pratik Toke <[pratik.toke07@gmail.com](mailto:pratik.toke07@gmail.com)> | | Feb 25, 2020, 11:48 AM |  |  |
| |  | | --- | | to Pratik, pratik | | | |

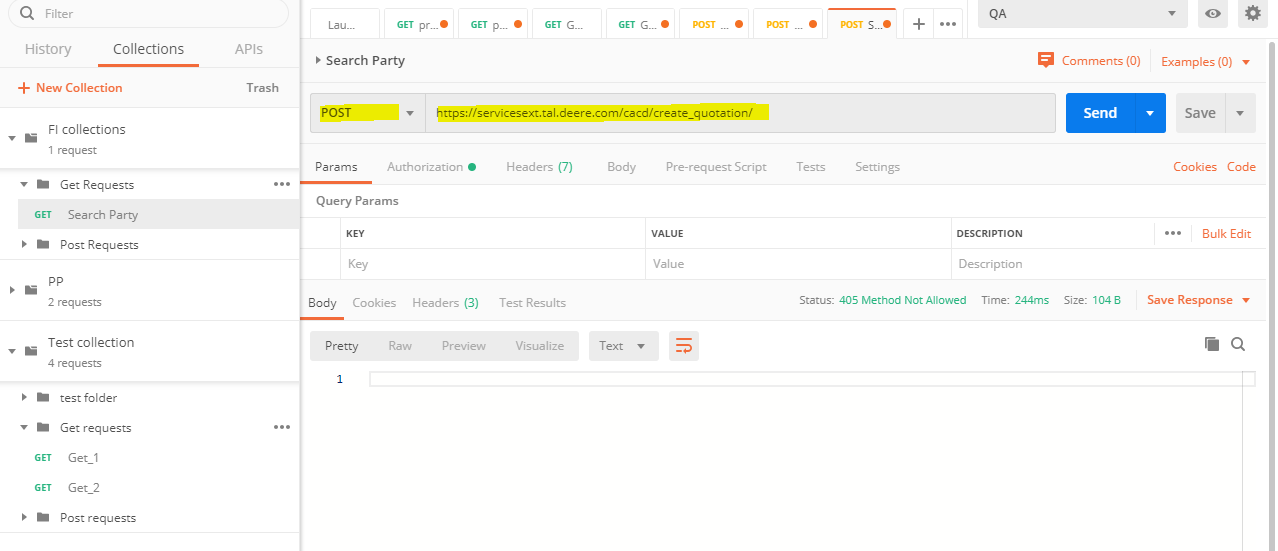




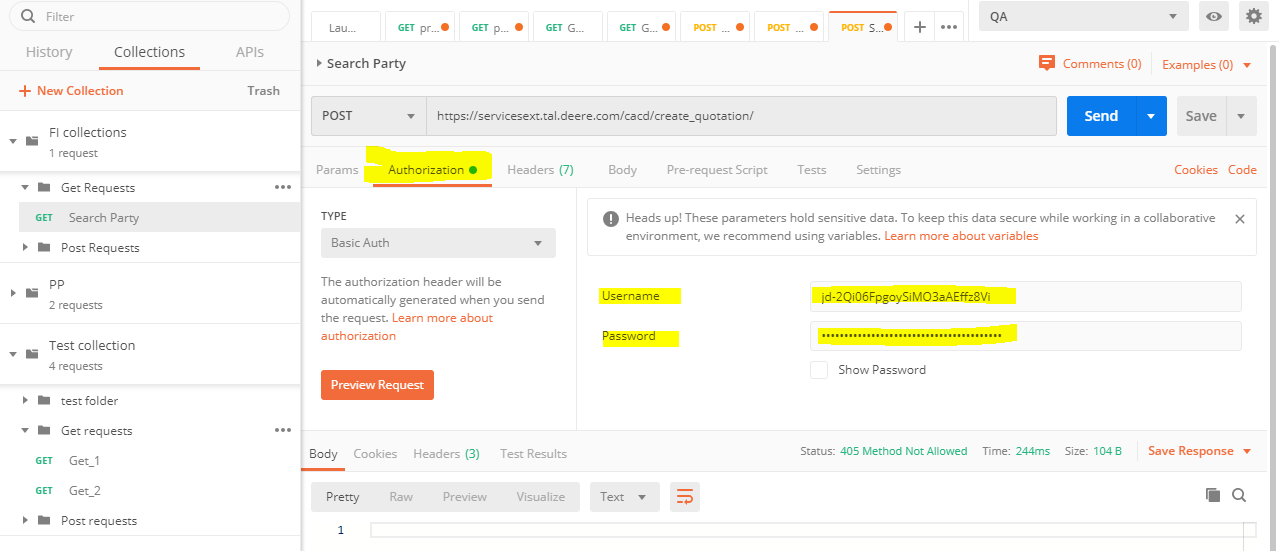


|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | Pratik Toke <[pratik.toke07@gmail.com](mailto:pratik.toke07@gmail.com)> | | Feb 25, 2020, 3:06 PM |  |  |
| |  | | --- | | to Pratik, pratik | | | |

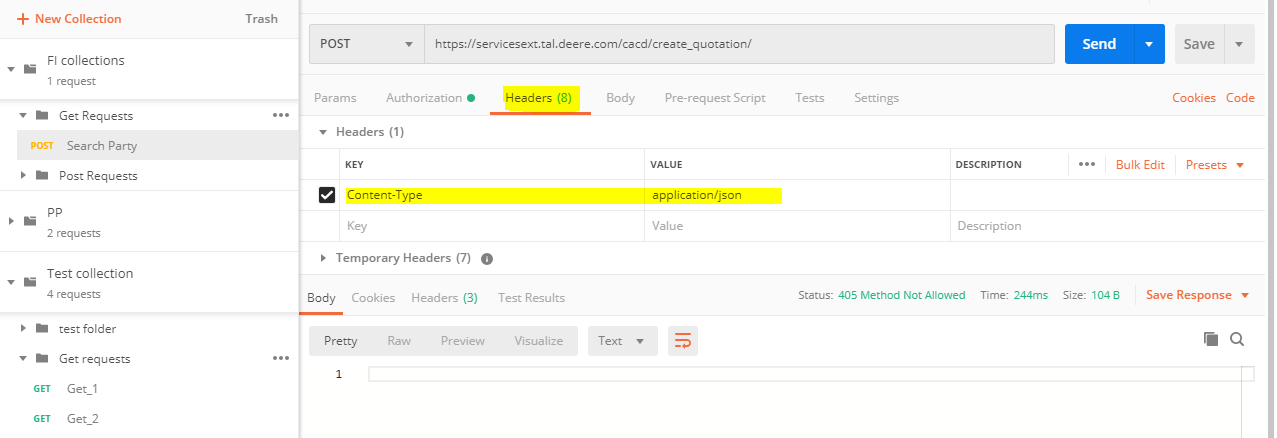
Office execution steps  
1.  take URL from xl file



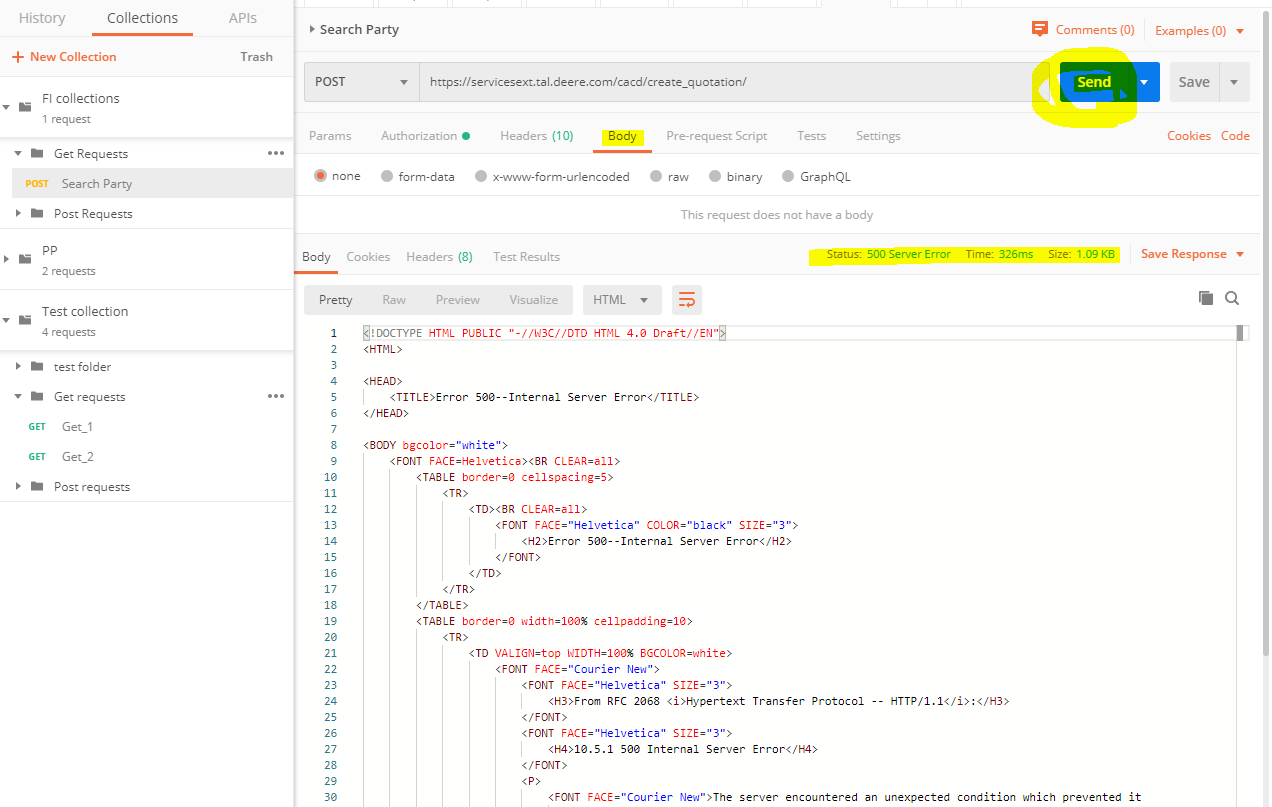
2. Take Uname and Pwd from Xl file



3. Put these values



4. Send and test reply in body







|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | Pratik Toke <[pratik.toke07@gmail.com](mailto:pratik.toke07@gmail.com)> | | Feb 25, 2020, 3:34 PM |  |  |
| |  | | --- | | to Pratik, pratik | | | |

1. Request  : Post : [https://reqres.in/api/users/2](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Freqres.in%2Fapi%2Fusers%2F2&data=04%7C01%7Cpratik.toke%40fisglobal.com%7C30e8c596a6fd4526efd708d974577483%7Ce3ff91d834c84b15a0b418910a6ac575%7C0%7C0%7C637668741252283654%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=%2FA8vjevv9Z%2FgrunOJ4UtMYc%2BZQ%2BG%2FqXn%2BnPFKZ0YyHM%3D&reserved=0) (its trying to fetch data of 2nd user from system )

2. Response : {

"data": {

"id": 2,

"first\_name": "Janet",

"last\_name": "Weaver",

"avatar":"[https://s3.amazonaws.com/uifaces/faces/twitter/josephstein/128.jpg](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fs3.amazonaws.com%2Fuifaces%2Ffaces%2Ftwitter%2Fjosephstein%2F128.jpg&data=04%7C01%7Cpratik.toke%40fisglobal.com%7C30e8c596a6fd4526efd708d974577483%7Ce3ff91d834c84b15a0b418910a6ac575%7C0%7C0%7C637668741252293610%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=8KPkTbn%2BnpDRTmwVJyoJoekcel%2B6%2FPSFsVuBYHzeok4%3D&reserved=0)"

}

}

This is the response. i.e. data of 2nd user is fetched from system. We can check the same values in the database

and we can verify that the response data and database values are same.

3. We can write the below script as well to verify the response automatically.

Create test cases for each assertion

var jsonData = pm.response.json();

pm.test("Verify data ID", function () {

pm.expect([jsonData.data.id](https://eur02.safelinks.protection.outlook.com/?url=http%3A%2F%2Fjsondata.data.id%2F&data=04%7C01%7Cpratik.toke%40fisglobal.com%7C30e8c596a6fd4526efd708d974577483%7Ce3ff91d834c84b15a0b418910a6ac575%7C0%7C0%7C637668741252293610%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=8PPg%2FOJMhssGeMFjchIFx4ggZlIubTxzDT14IWASf8E%3D&reserved=0)).is.to.equal(2);

});

pm.test("Verify first\_name", function () {

pm.expect(jsonData.data.first\_name).is.to.equal("Janet");

});

pm.test("Verify last\_name", function () {

pm.expect(jsonData.data.last\_name).is.to.equal("Weaver");

});

This is how we verify the data came from Postman.