

API Testing

Get Request: Sending a request to get some data from the server is called get request.

Post Request: Sending a request to store some data into the database/server is called post request.

Put Request: Sending a request to update the existing data in the server.

Note: To create your own API these two software must be installed.

Creating our own API's

step1) NodeJS

npm - node package manager

node --version

npm --version

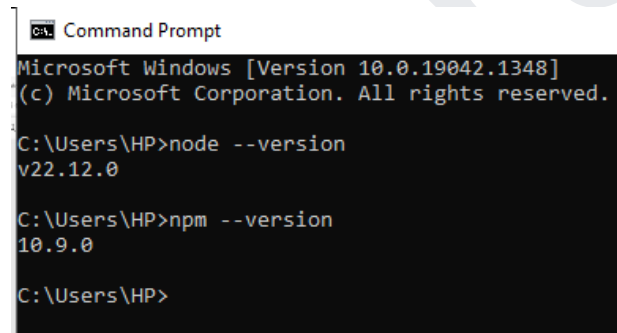
step2) json-server

run the below command in the cmd/terminal

npm install -g json-server

Install: 1. Nodejs software with npm

Check the software is available or not from the cmd : Nodejs: **node --version** and npm: **npm --version**



```
cmd Command Prompt
Microsoft Windows [Version 10.0.19042.1348]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>node --version
v22.12.0

C:\Users\HP>npm --version
10.9.0

C:\Users\HP>
```

Install : 2. Json-server.

Create an Own/Dummy REST API “ Students.Json “ file with the following data :

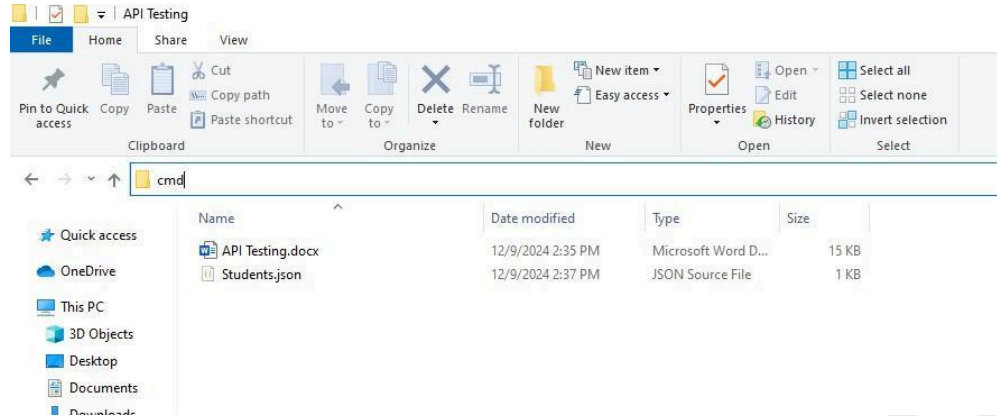
```
{
  "students":[
    {
      "id": 1,
      "name": "John Doe",
      "age": 18,
      "grade": "12th",
      "subjects": [
        "Math",
        "Physics"
      ],
      "English"
    },
    {
      "id": 2,
      "name": "Jane Smith",
      "age": 17,
      "grade": "11th",
      "subjects": [
        "Biology",
        "Chemistry",
        "History"
      ]
    },
    {
      "id": 3,
      "name": "David Johnson",
      "age": 16,
      "grade": "10th",
      "subjects": [
        "Computer Science",
        "Spanish",
        "Art"
      ]
    }
  ]
}
```

How to create own API ?

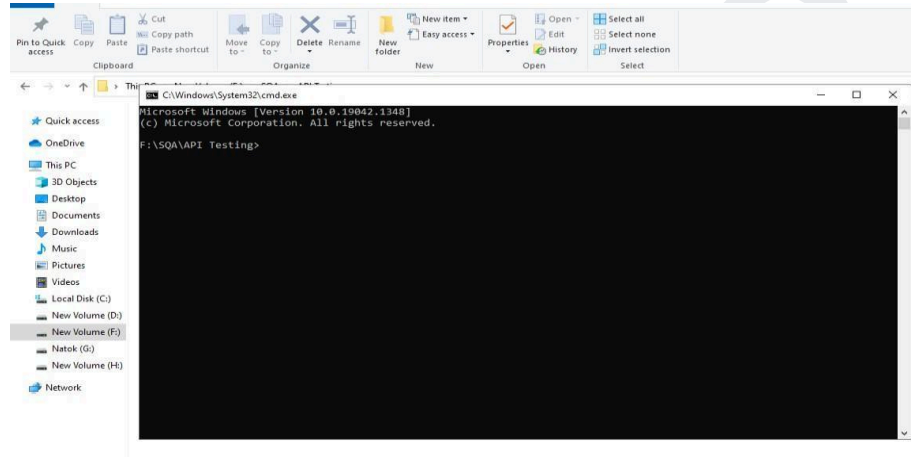
Task no-1

Ans: Goto the location where the “ Students.json ” File is stored.

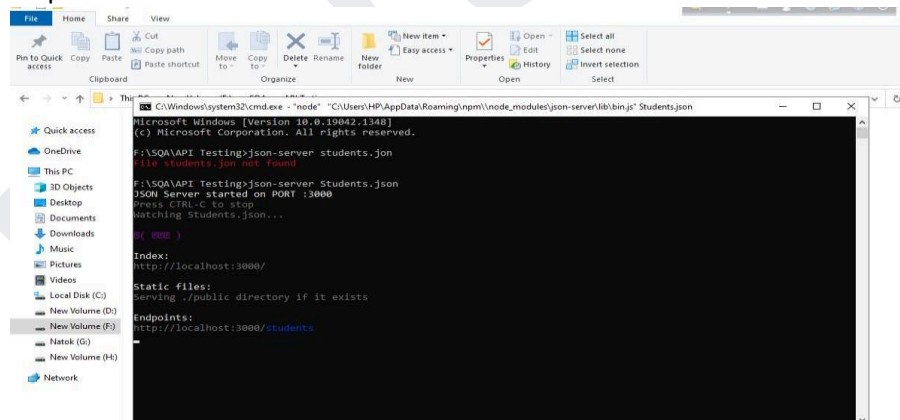
Step-1



Step-2



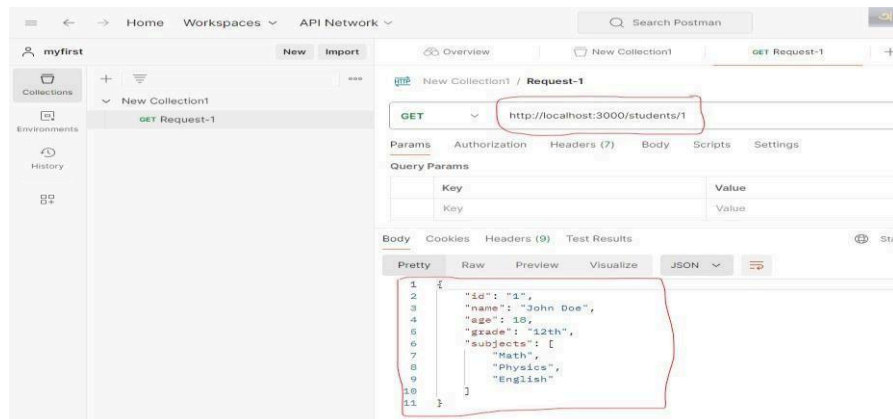
Step-3



Link: <http://localhost:3000/students>

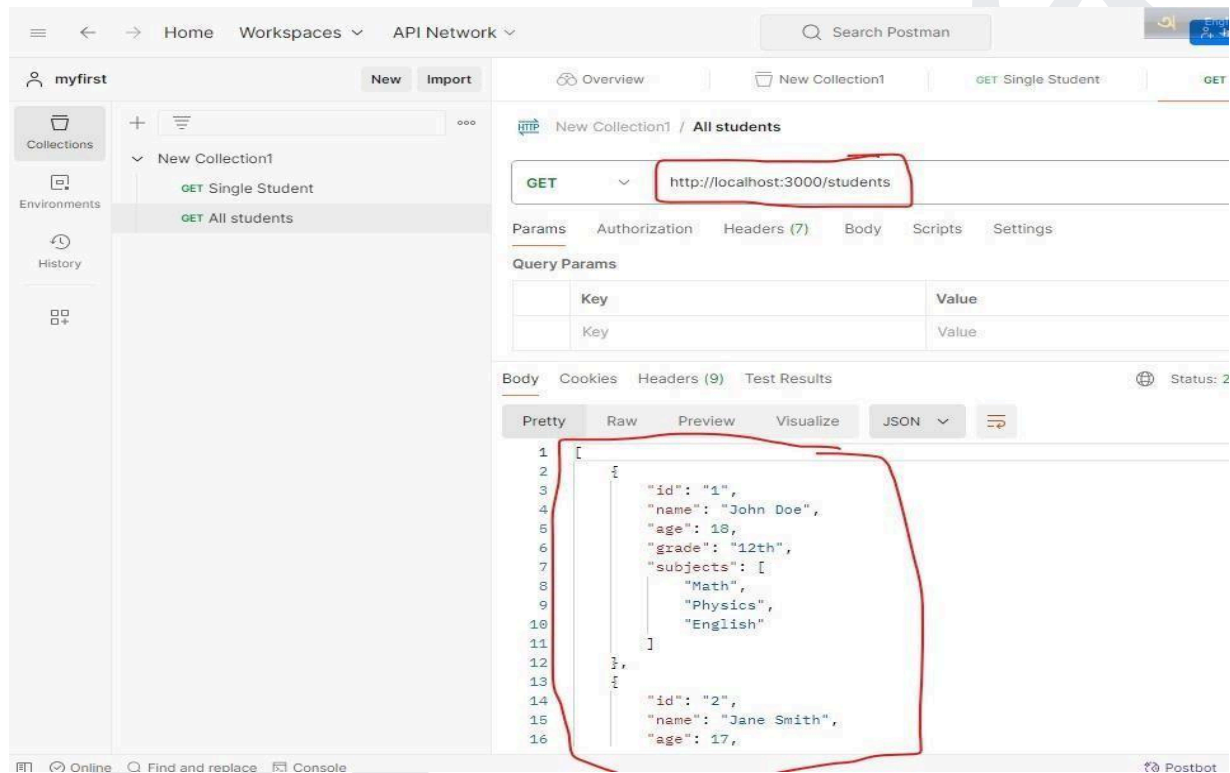
How to get single student data from the above URL?

Task No- 2



How to get all students data from the above URL?

Task No- 3



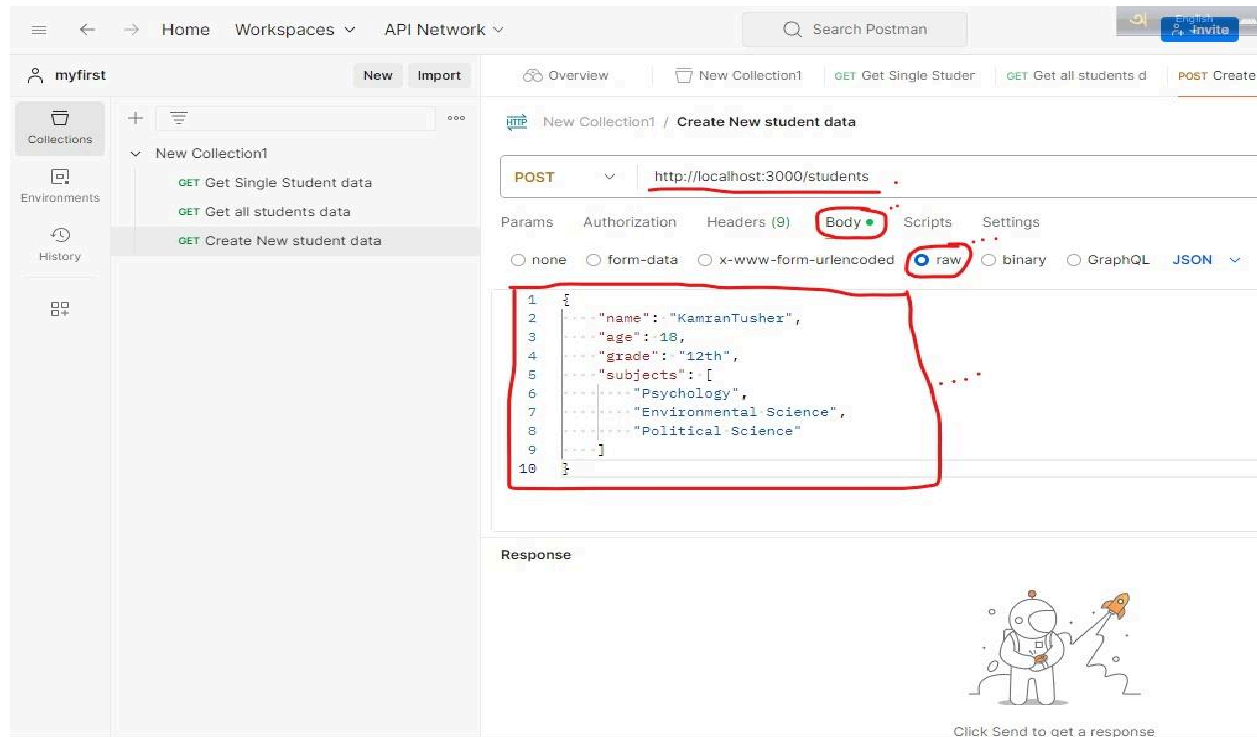
How to create new student data?

Create = Post

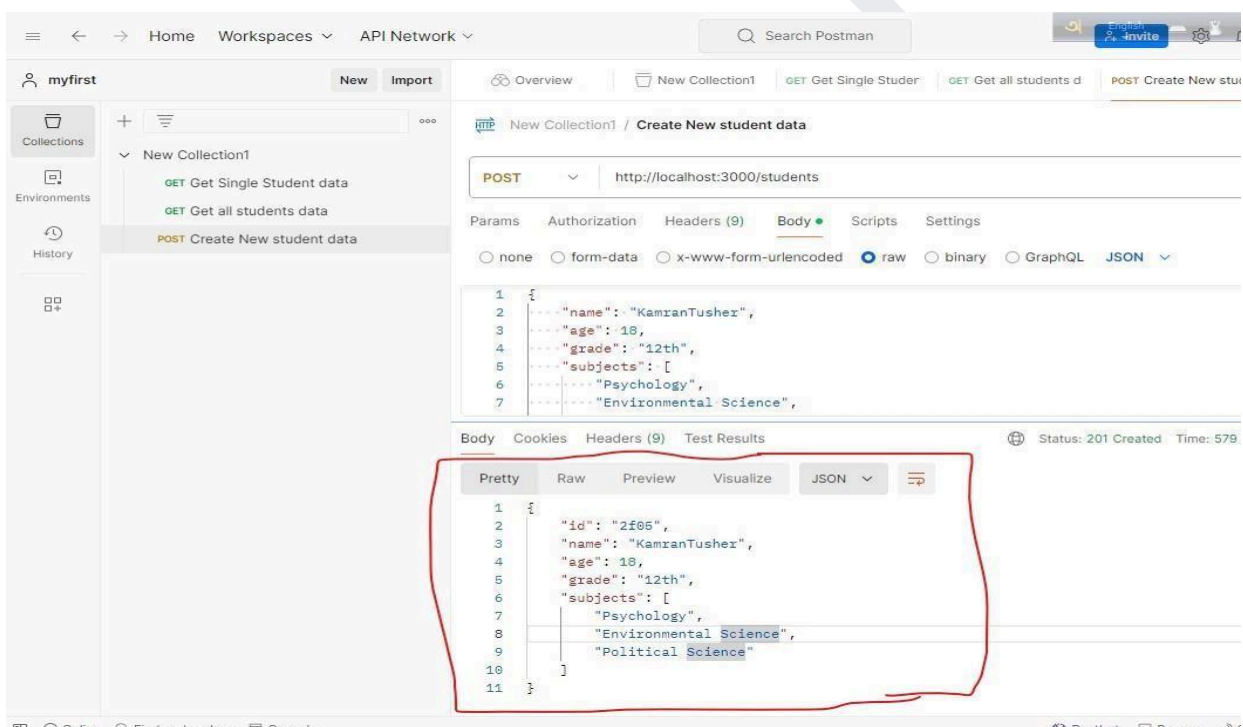
Task No- 4

Ans: Add a new collection " Post " then give the url. Then insert a new request in the "Request Payload "

Step-1 (Request Payload)



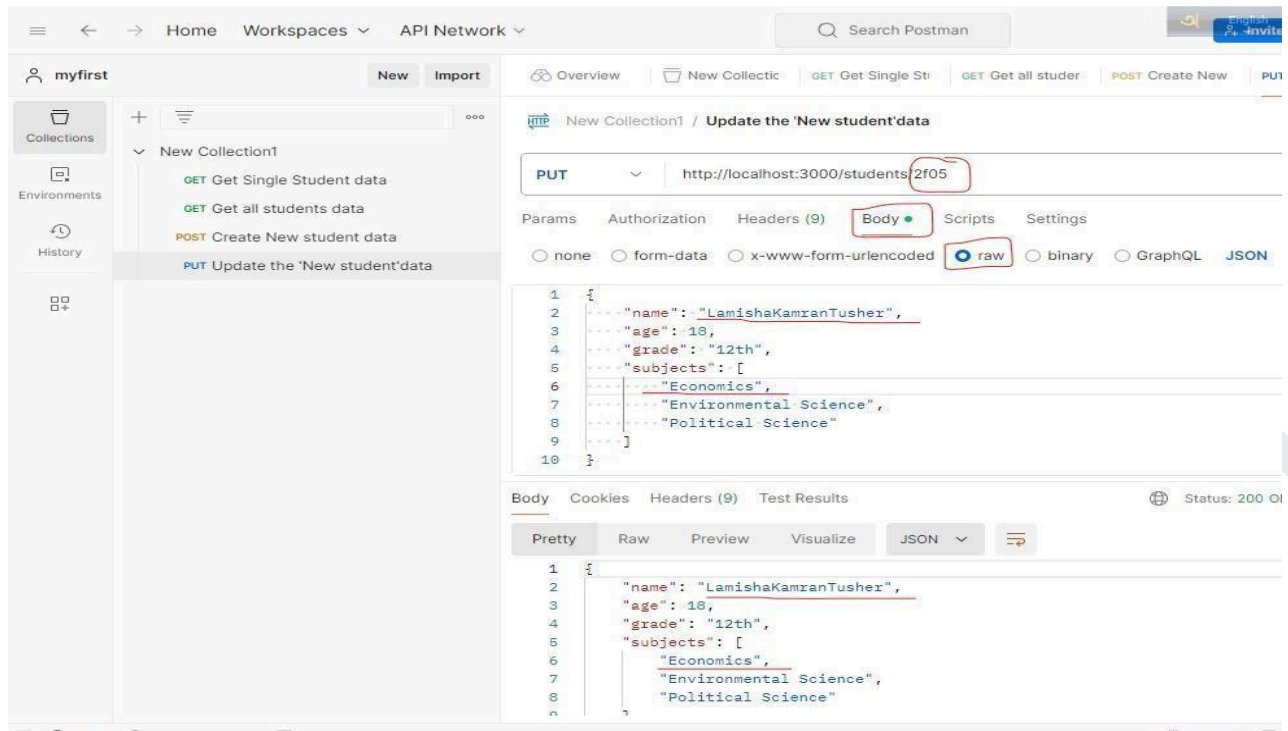
Step-2 (Response Payload)



How to update the earlier "New student data" data?

Update = Put **Task No- 5**

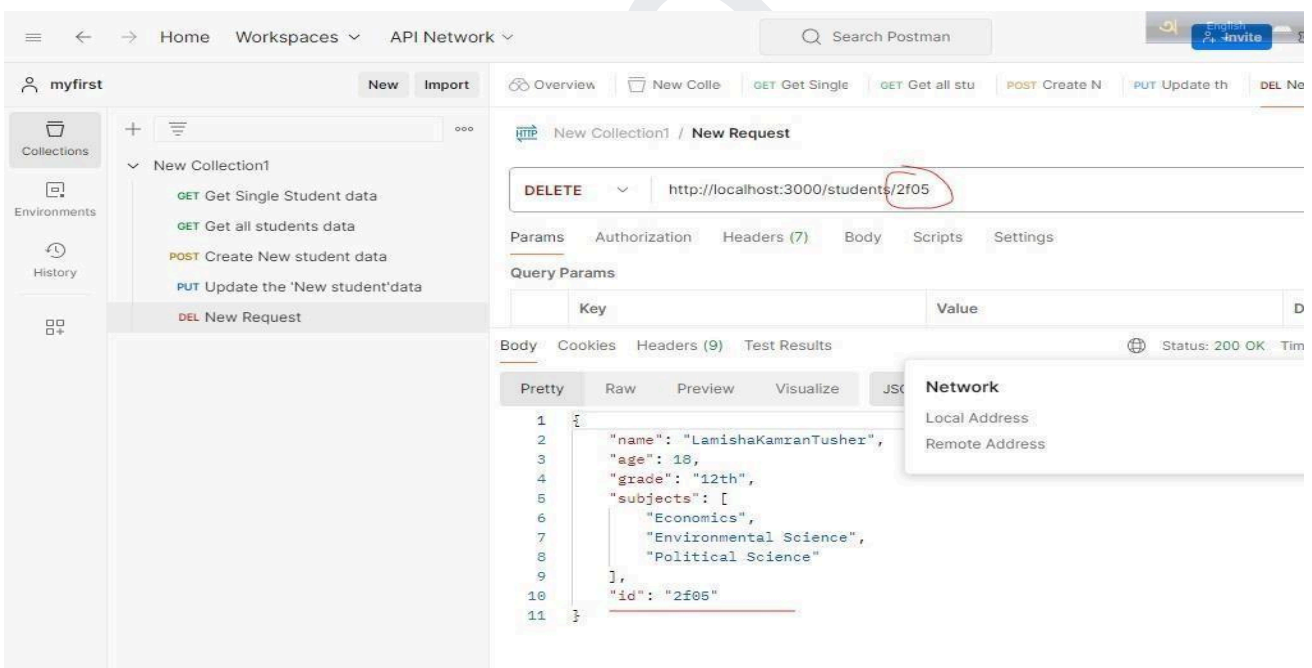
Ans.: Make sure to set the "Id Number" beside the link_ (<http://localhost:3000/students/2f05>)



How to delete any data from the request ?

Task No- 6

Ans : I want to delete ID = " 2f05 "



Validation :

1. Response Body

2. Headers
3. Cookies
4. Status Code
5. Time

JSON ---Javascript object Notation

What is JSON?

- JSON – **Java Script Object Notation**
- JSON is a syntax for storing and exchanging data.
- Basically It was designed for human-readable data interchange.
- JSON is text, written with Java Script Object Notation.
- It has been extended from the JavaScript scripting language
- The filename extension is **.json**
- JSON internet Media type is **application/json**

JSON Data Types

- Number
- String
- Boolean
- Null
- Object
- Array

Note: In JSON format, we have to represent the data in the form of **KEY: Value Pair**

Data Types

- **String**

- Strings in JSON must be written in double quotes.
- Example:

```
{ "name": "John" }
```

- **Numbers**

- Numbers in JSON must be an integer or a floating point.
- Example:

```
{ "age": 30 }
```

- **Object**

- Values in JSON can be objects.
- Example:

```
{
  "employee": { "name": "John", "age": 30, "city": "New York" }
}
```

String

Example

```
{
  "name": "John",
}
```

Note: KEY is always included in “ ” double quotation

```
"Key": Value
{ "name": "John" }
```

Here { "name" : " John " }-----name is included in double quotation But John included in double quotation here because John is string.

When we input multiple inputs in one variable then we use [] this is called **JSON Array**.

Example:

```
{
  "name": "John",
  "age": 30,
  "phone": [12345,6789]
}
```


Data Types

- **Array**
 - Values in JSON can be arrays.
 - Example:

```
{  
  "employees": [ "John", "Anna", "Peter" ]  
}
```
- **Boolean**
 - Values in JSON can be true/false.
 - Example:

```
{ "sale": true }
```
- **Null**
 - Values in JSON can be null.

```
{ "middlename": null }
```

Example :

```
{  
  "Firstname": "John",  
  "Lastname": Null,  
  "age": 30,  
  "phone": [12345,6789],  
  "Status": true  
}
```

JSON - Syntax

- Data should be in name/value pairs
- Data should be separated by commas
- Curly braces should hold objects
- Square brackets hold arrays

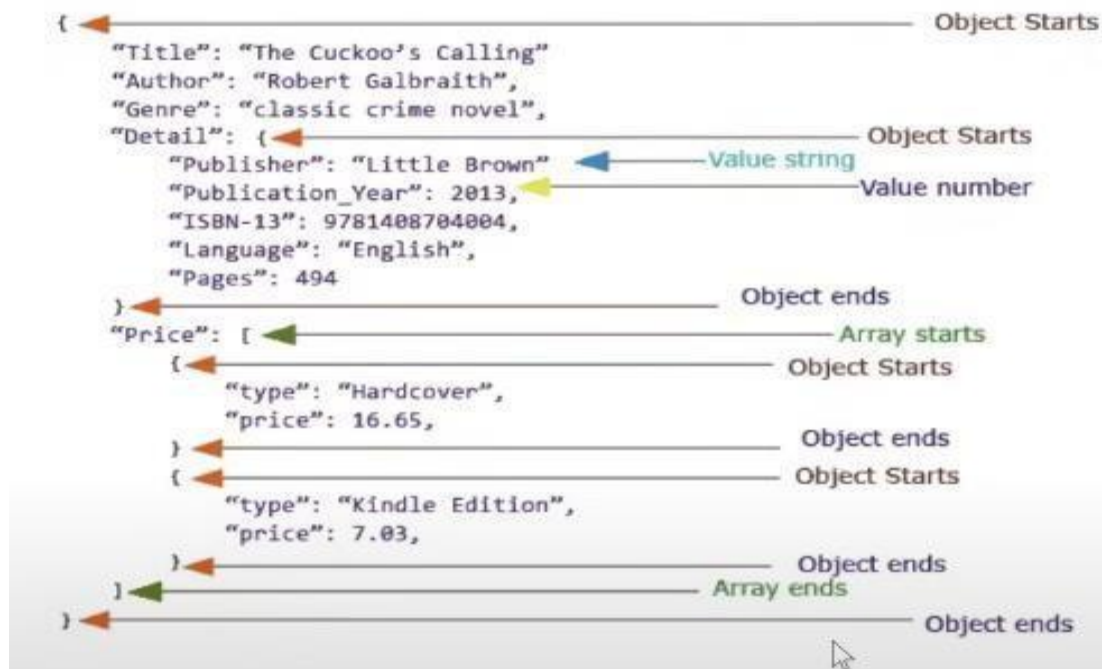
```
{  
  "student": [  
    {  
      "id": "01",  
      "name": "Tom",  
      "lastname": "Price"  
    },  
    {  
      "id": "02",  
      "name": "Nick",  
      "lastname": "Thameson"  
    }  
  ]  
}
```

JSON Object object—Which Contains Multiple KEY : Value Pairs

Student Data (Student Contains SID, SName, Grade) HERE 4 Student/Object.

```
{  
  
  "Student": [  
  
    {  
      "SID": 101,  
      "SName": "Kamran Tusher",  
      "Grade": "A",  
    },  
  
    {  
      "SID": 102,  
      "SName": "Lamisha Rahman",  
      "Grade": "A+",  
    },  
  
    {  
      "SID": 103,  
      "SName": "Zenith Chowdhury",  
      "Grade": "A",  
    },  
  
    {  
      "SID": 104,  
      "SName": "Liyana Lio",  
      "Grade": "A",  
    },  
  ],  
}
```

Explanation



JSON vs XML

JSON	XML
JSON is simple to read and write.	XML is less simple as compared to JSON.
It also supports array .	It doesn't support array.
JSON files are more human-readable than XML.	XML files are less human readable .
It supports only text and number data type	It supports many data types such as text , number , images , charts , graphs , etc.

Validate JSON Path :

Task No- 7

```
{
  "Student": [
    {
      "SID": 101,
      "SName": "Kamran Tusher",
      "Grade": "A"
    },
    {
      "SID": 102,
      "SName": "Lamisha Rahman",
      "Grade": "A"
    },
    {
      "SID": 103,
      "SName": "Zenith Chowdhury",
      "Grade": "A"
    },
    {
      "SID": 104,
      "SName": "Liyana Lio",
      "Grade": "A"
    }
  ]
}
```

How to extract any data from the above JSON file by using JSON path ?

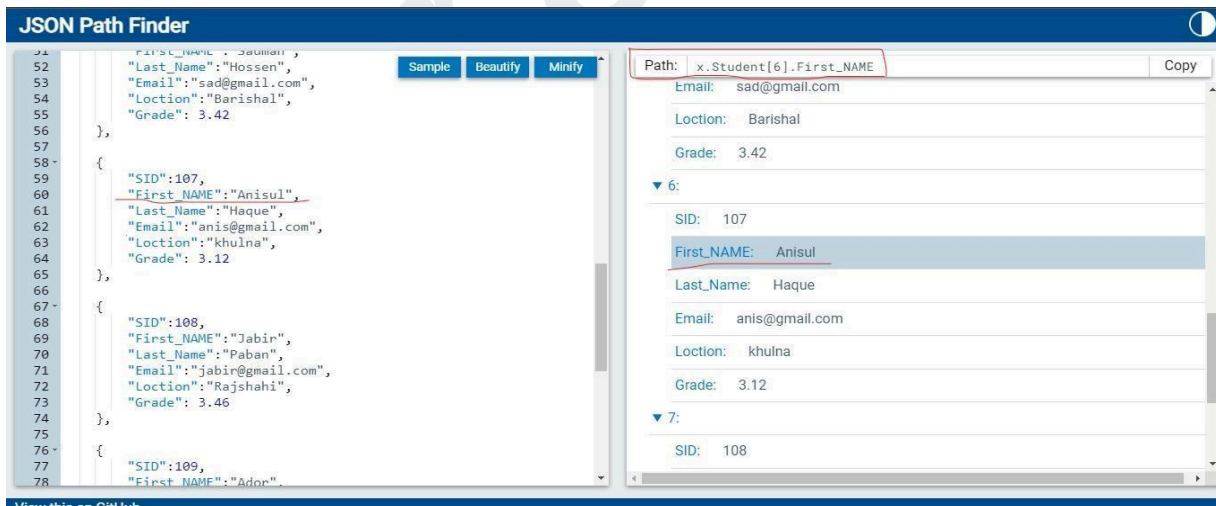
Ans: I want to extract the red-marked data from the file.

Extract Procedure— JSON Path : **Student[1].SName** **Lamisha Rahman**

Note: For complex JSON file we need to use tools to extract the data

Tools: 1)JSON Pathfinder. 2) JSON .com

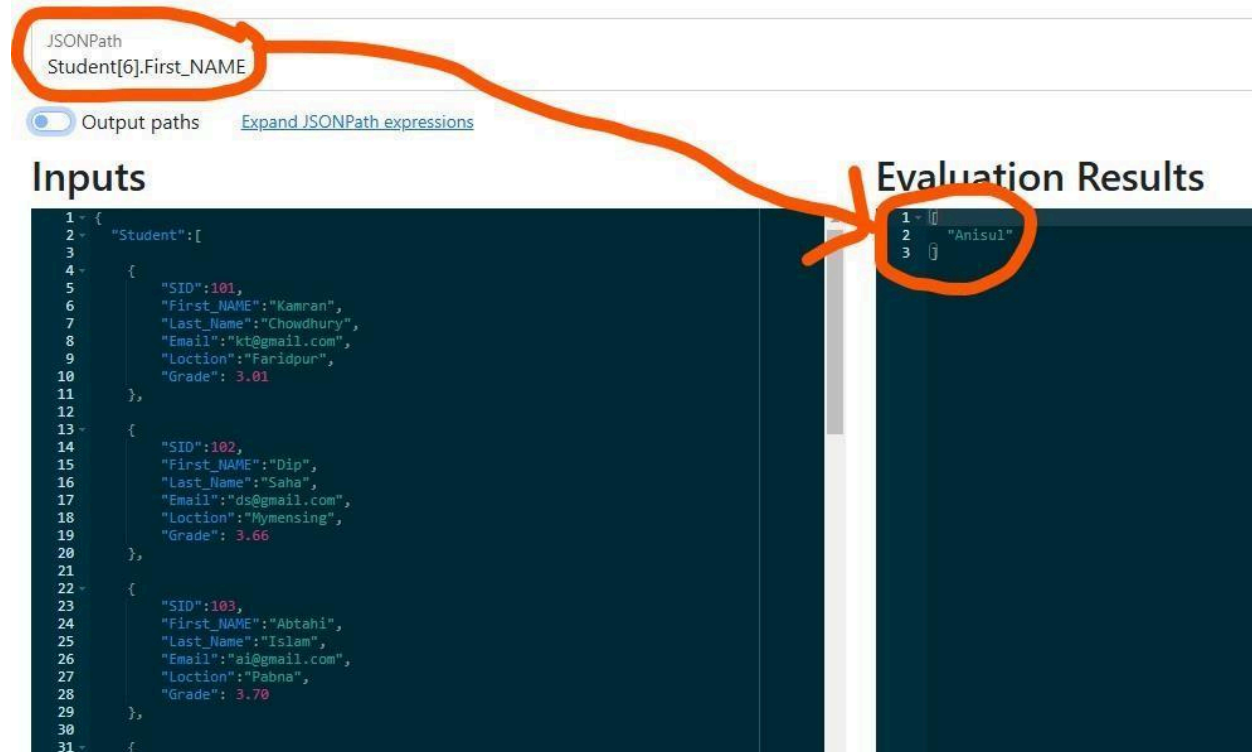
Example: I want get the 2nd object's first name data JSON path Site Link : [Link](#)



The screenshot shows the JSON Path Finder tool interface. On the left, a JSON array of student objects is displayed. The second object (index 1) is highlighted. On the right, the 'Path' field is set to `x.Student[1].First_NAME`. The tool has successfully extracted the value 'Anisul' for the 'First_NAME' field of the second student.

JSON path Validation

(For path validation Site link: [Link](#))



The screenshot shows the JSONPath validation tool interface. The input path is `Student[6].First_NAME`. The tool is set to "Output paths" and "Expand JSONPath expressions". The "Inputs" section shows a JSON array of three student objects. The "Evaluation Results" section shows the result of the path validation, which is `"Anisul"`.

```

1 {
2   "Student": [
3     {
4       "SID": 101,
5       "First_NAME": "Kamran",
6       "Last_Name": "Chowdhury",
7       "Email": "kt@gmail.com",
8       "Location": "Faridpur",
9       "Grade": 3.01
10    },
11    {
12      "SID": 102,
13      "First_NAME": "Dip",
14      "Last_Name": "Saha",
15      "Email": "ds@gmail.com",
16      "Location": "Mymensing",
17      "Grade": 3.66
18    },
19    {
20      "SID": 103,
21      "First_NAME": "Abtahi",
22      "Last_Name": "Islam",
23      "Email": "ai@gmail.com",
24      "Location": "Pabna",
25      "Grade": 3.70
26    }
27  ]
28 }
  
```

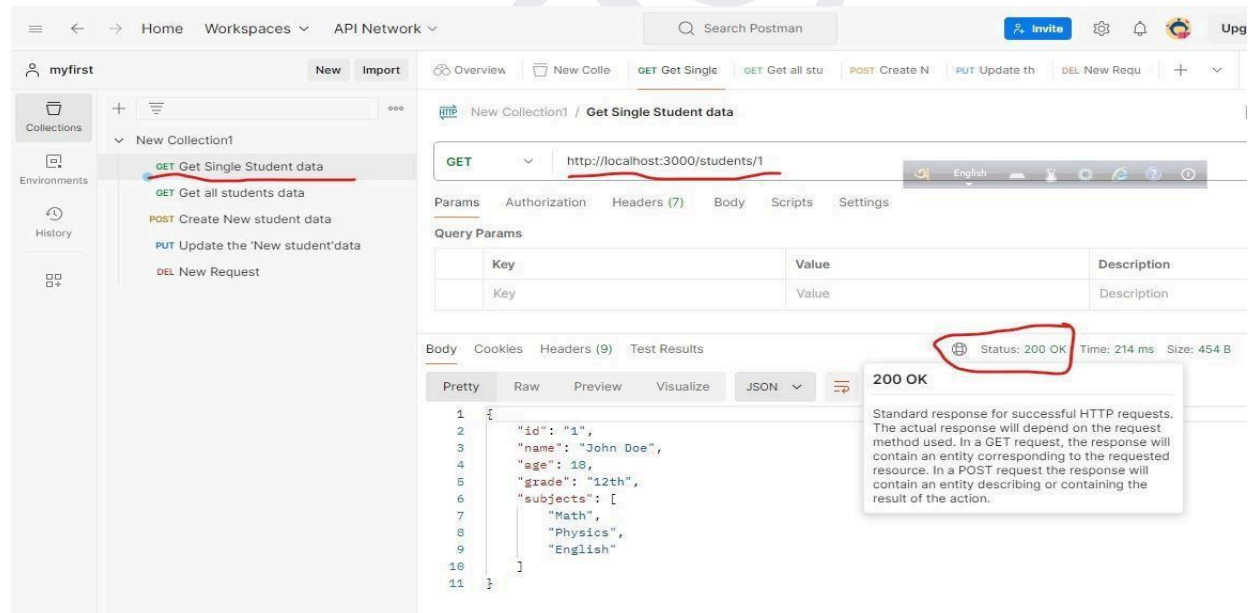
The evaluation results show the path `Student[6].First_NAME` evaluates to `"Anisul"`.

Response Validation

Task No--8

1. Status Code:

Step-1



The screenshot shows the Postman interface. A GET request is sent to `http://localhost:3000/students/1`. The response status is `200 OK`. The response body is a JSON object representing a student.

```

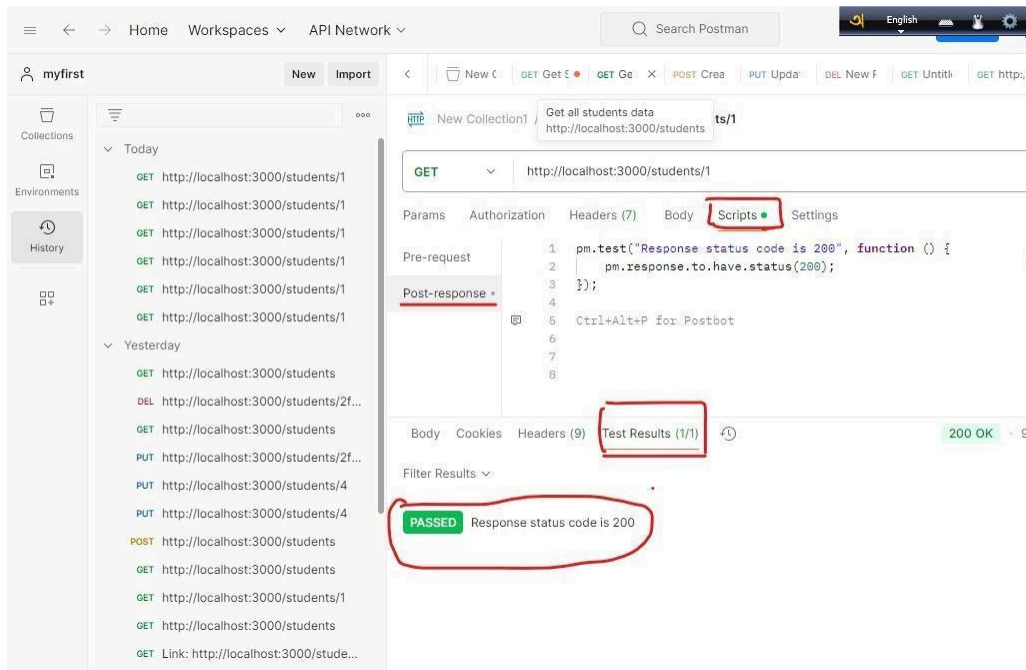
1 {
2   "id": "1",
3   "name": "John Doe",
4   "age": 18,
5   "grade": "12th",
6   "subjects": [
7     "Math",
8     "Physics",
9     "English"
10  ]
11 }
  
```

The status bar shows `Status: 200 OK`, `Time: 214 ms`, and `Size: 454 B`. A tooltip explains that `200 OK` is the standard response for successful HTTP requests.

Step-2

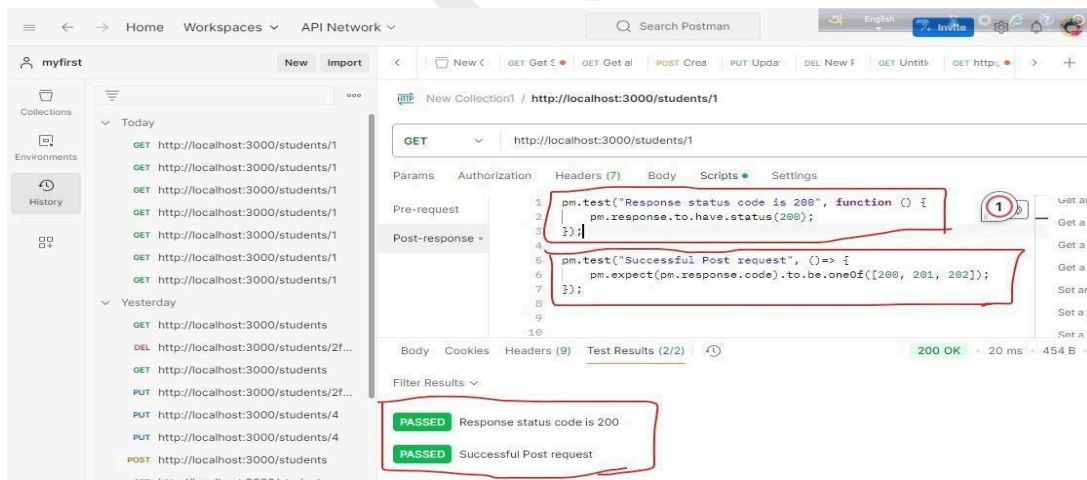
For Single Status code

```
pm.test("Response status code is 200", function () {  
  
    pm.response.to.have.status(200);  
  
});
```



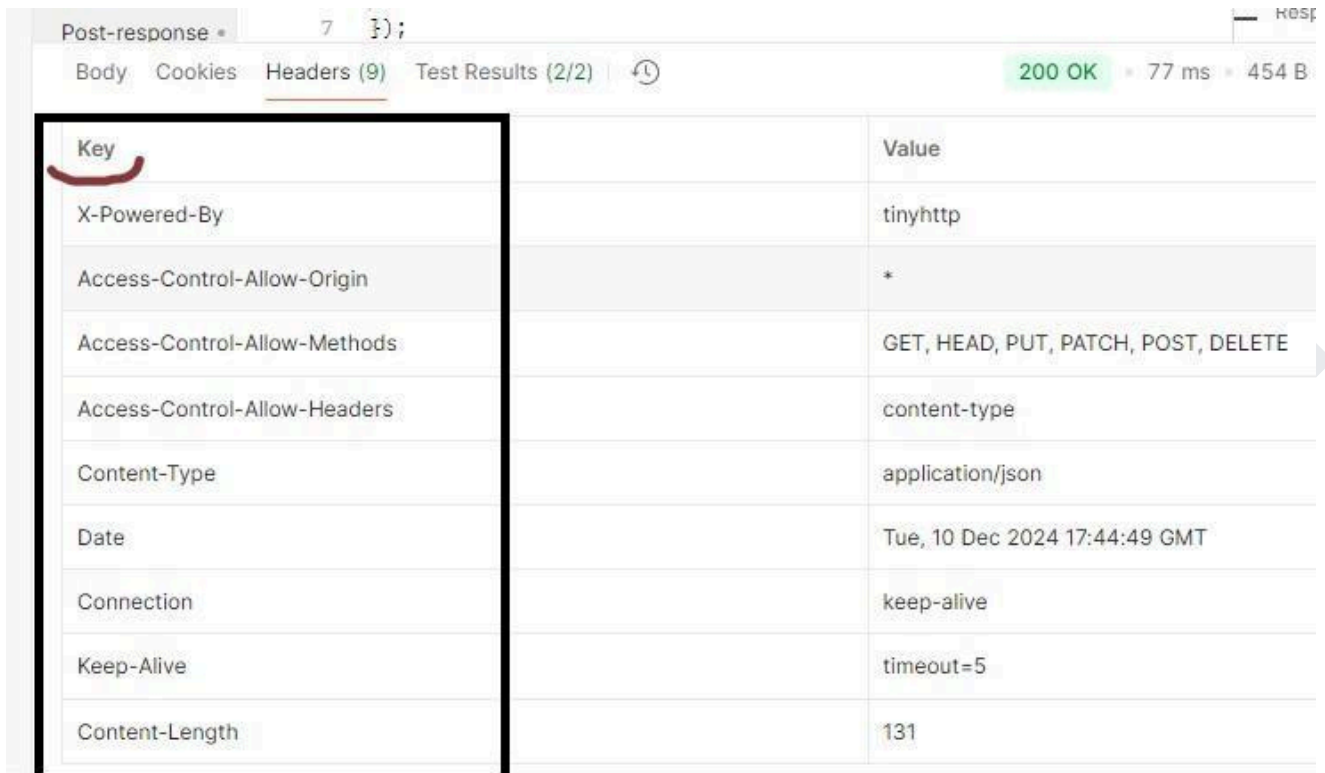
For Multiple Status code

```
pm.test("Successful Post request", () => {  
  
    pm.expect(pm.response.code).to.be.oneOf([200, 201, 202]);  
  
});
```



1. Headers Validation Task No—9

We will Validate “Key” of the header “ given below



Key	Value
X-Powered-By	tinyhttp
Access-Control-Allow-Origin	*
Access-Control-Allow-Methods	GET, HEAD, PUT, PATCH, POST, DELETE
Access-Control-Allow-Headers	content-type
Content-Type	application/json
Date	Tue, 10 Dec 2024 17:44:49 GMT
Connection	keep-alive
Keep-Alive	timeout=5
Content-Length	131

How to validate All the “KEY” response header is present :

```
pm.test("Content-Type is present", function () {
    pm.response.to.have.header("Content-Type");
});

pm.test("Content-Length is present", function () {
    pm.response.to.have.header("Content-Length");
});

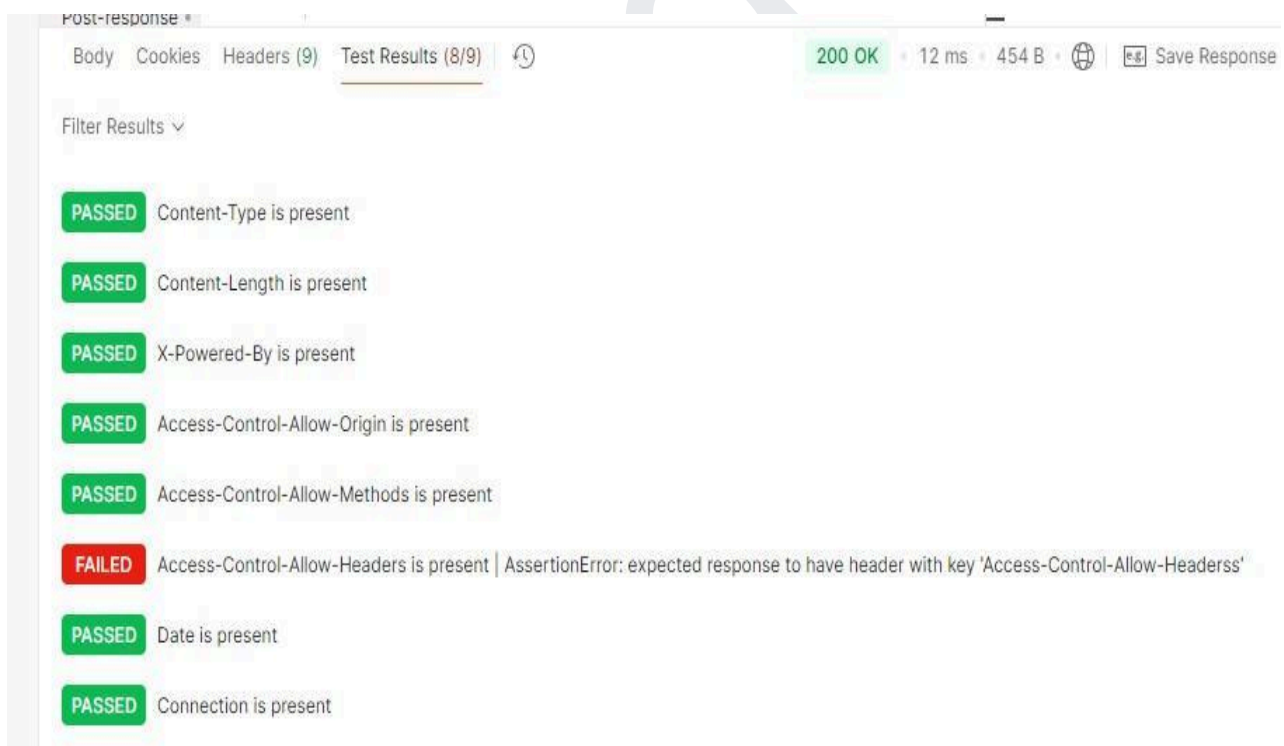
pm.test("X-Powered-By is present", function () {
    pm.response.to.have.header("X-Powered-By");
});

pm.test("Access-Control-Allow-Origin is present", function () {
    pm.response.to.have.header("Access-Control-Allow-Origin");
});
```



```
pm.test("Access-Control-Allow-Methods is present", function () {  
    pm.response.to.have.header("Access-Control-Allow-Methods");  
});  
  
pm.test("Access-Control-Allow-Headers is present", function () {  
    pm.response.to.have.header("Access-Control-Allow-Headerss");  
});  
  
pm.test("Date is present", function () {  
    pm.response.to.have.header("Date");  
});  
  
pm.test("Connection is present", function () {  
    pm.response.to.have.header("Connection");  
});  
  
pm.test("Keep-Alive is present", function () {  
    pm.response.to.have.header("Keep-Alive");  
});
```

Here Is the result Images:



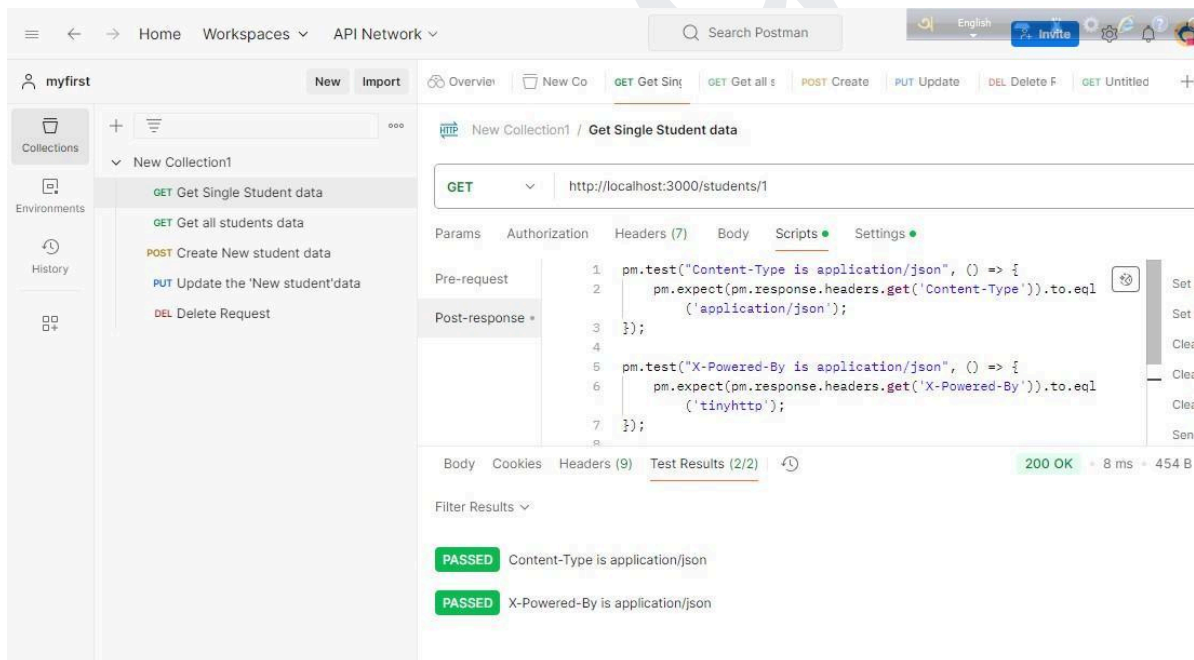
How to Validate the “value” of the header ?

Task No-10

Value	
tinyhttp	
*	
GET, HEAD, PUT, PATCH, POST, DELETE	
content-type	
application/json	
Tue, 10 Dec 2024 17:44:49 GMT	
keep-alive	
timeout=5	
131	

Code is given below

```
pm.test("Content-Type is application/json", () => {
  pm.expect(pm.response.headers.get('Content-Type')).to.eql('application/json');
});
```



The screenshot shows the Postman interface with a GET request to `http://localhost:3000/students/1`. The 'Test Results' tab is active, displaying two passed tests:

- PASSED** Content-Type is application/json
- PASSED** X-Powered-By is application/json

All the " Value " validated together using an Array

```
const headers = [  
  { key: "X-Powered-By", value: "tinyhttp" },  
  { key: "Access-Control-Allow-Origin", value: "*" },  
  { key: "Access-Control-Allow-Methods", value: "GET, HEAD, PUT, PATCH, POST, DELETE" },  
  { key: "Access-Control-Allow-Headers", value: "content-type" },  
  { key: "Content-Type", value: "application/json" },  
  //{ key: "Date", value: "Tue, 10 Dec 2024 20:04:59 GMT" },  
  { key: "Connection", value: "keep-alive" },  
  { key: "Keep-Alive", value: "timeout=5" },  
  { key: "Content-Length", value: "131" }  
];  
  
// Loop through headers and validate each using a basic for loop  
for (let i = 0; i < headers.length; i++) {  
  const header = headers[i];  
  pm.test(`${header.key} is ${header.value}`, () => {  
    pm.expect(pm.response.headers.get(header.key)).to.eql(header.value);  
  });  
}
```

2. Cookies Validation :

We need to verify the **cookie name** and the **value**

How to check the cookies are present in the response?

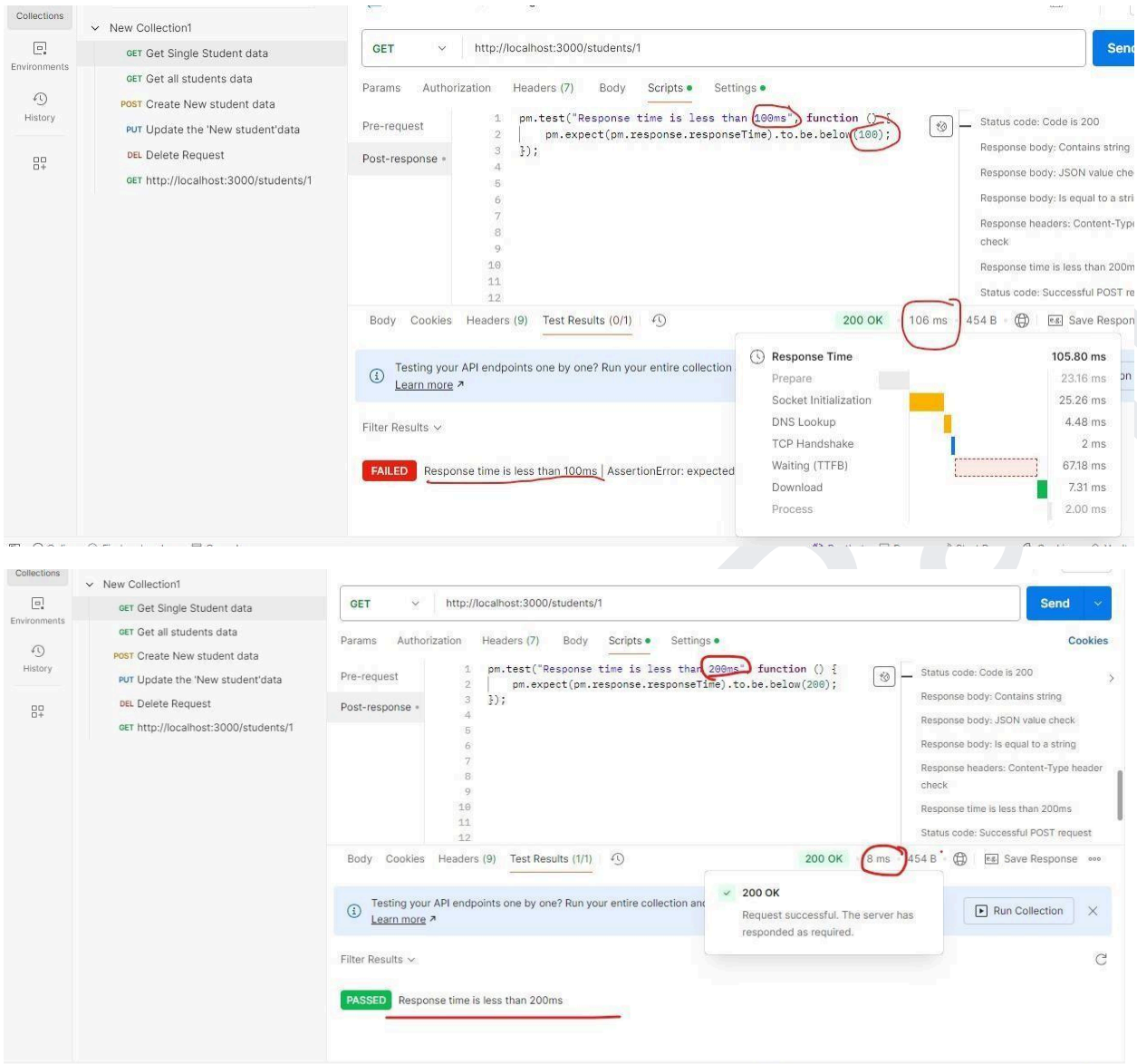
Ans:

3. Response Time validation :

How to check the response time ?

```
Ans: pm.test("Response time is less than 200ms", function () {  
  pm.expect(pm.response.responseTime).to.be.below(200);  
});
```

Actually, the response time keep changing during the execution.



The top screenshot shows a GET request to `http://localhost:3000/students/1` with a test script: `pm.test("Response time is less than 100ms", function () { pm.expect(pm.response.responseTime).to.be.below(100); });`. The test results show a status of 200 OK and a response time of 106 ms. A red box highlights the test failure: **FAILED** Response time is less than 100ms | AssertionError: expected 106ms to be below 100ms.

The bottom screenshot shows the same GET request with a test script: `pm.test("Response time is less than 200ms", function () { pm.expect(pm.response.responseTime).to.be.below(200); });`. The test results show a status of 200 OK and a response time of 8 ms. A green box highlights the test success: **PASSED** Response time is less than 200ms.

Response Body:

Task No-

Different types of validation are done in Response Body validation:

1. Validate the Data type of the Data in the fields
2. Validate the Array Properties/Array Content in the fields
3. Validate the Data of the fields are matched/Correct or Not
4. Validate the Json schema

1. Validate the Data type of the Data in the fields :

Validate the “Type” of the value: Validate all the “data type” of these data from every assertion in this Response body.

How to validate the ‘Data type ‘ of the value from these assertions in the response body?

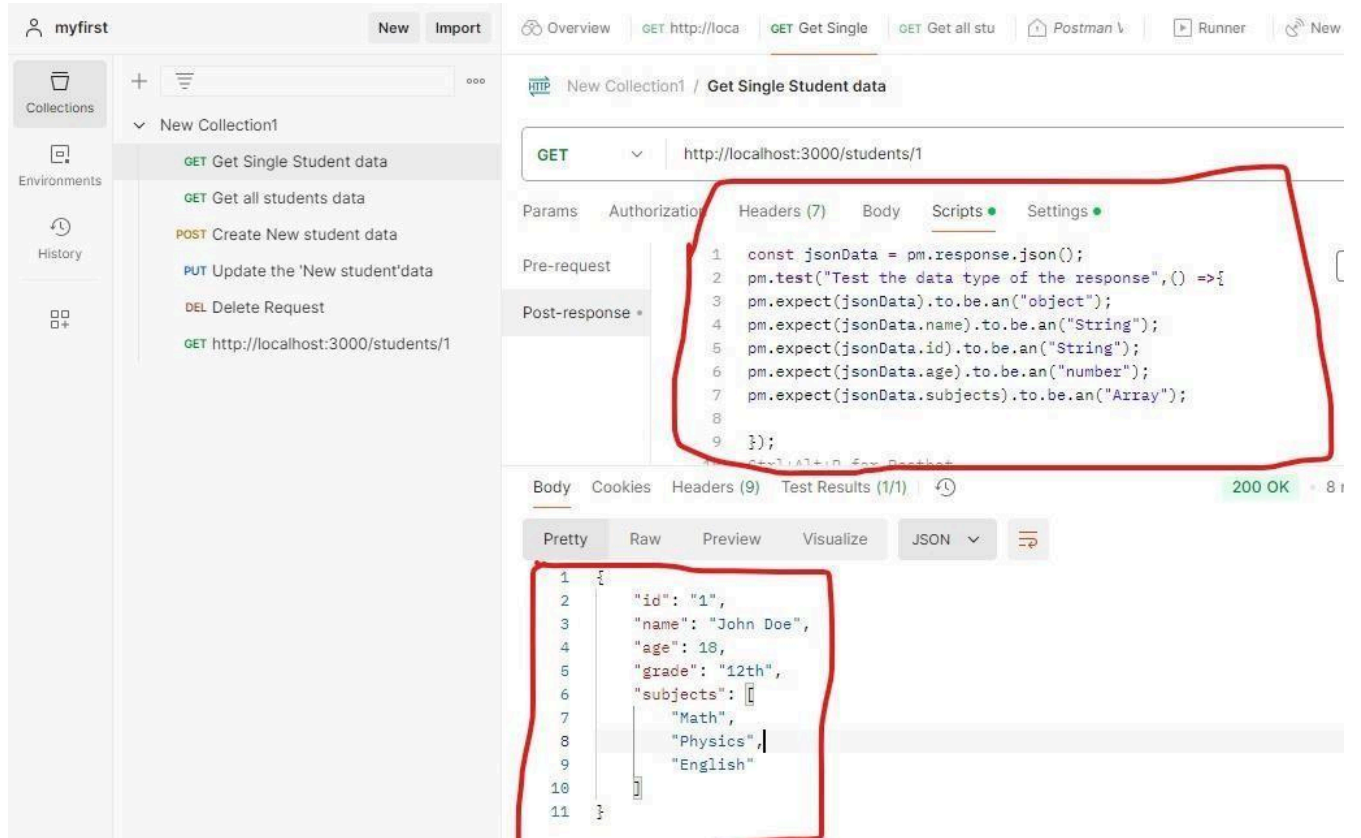
Here is the response body: (For single object in the response body)

```
{
  "id": "1",           //verify the data type of id
  "name": "John Doe", //verify the data type of name
  "age": 18,          //verify the data type of id
  "grade": "12th",    //verify the data type of id
  "subjects": [       //verify the data type of subject
    "Math",
    "Physics",
    "English"
  ]
}
```

CODE:

```
const jsonData = pm.response.json();
pm.test("Test the data type of the response", () =>{
  pm.expect(jsonData).to.be.an("object");
  pm.expect(jsonData.name).to.be.an("String");
  pm.expect(jsonData.id).to.be.an("String");
  pm.expect(jsonData.age).to.be.an("number");
  pm.expect(jsonData.subjects).to.be.an("Array");

});
```



myfirst New Import Overview GET http://loca GET Get Single GET Get all stu Postman v Runner New

New Collection1 / Get Single Student data

GET http://localhost:3000/students/1

Params Authorization Headers (7) Body Scripts Settings

Pre-request

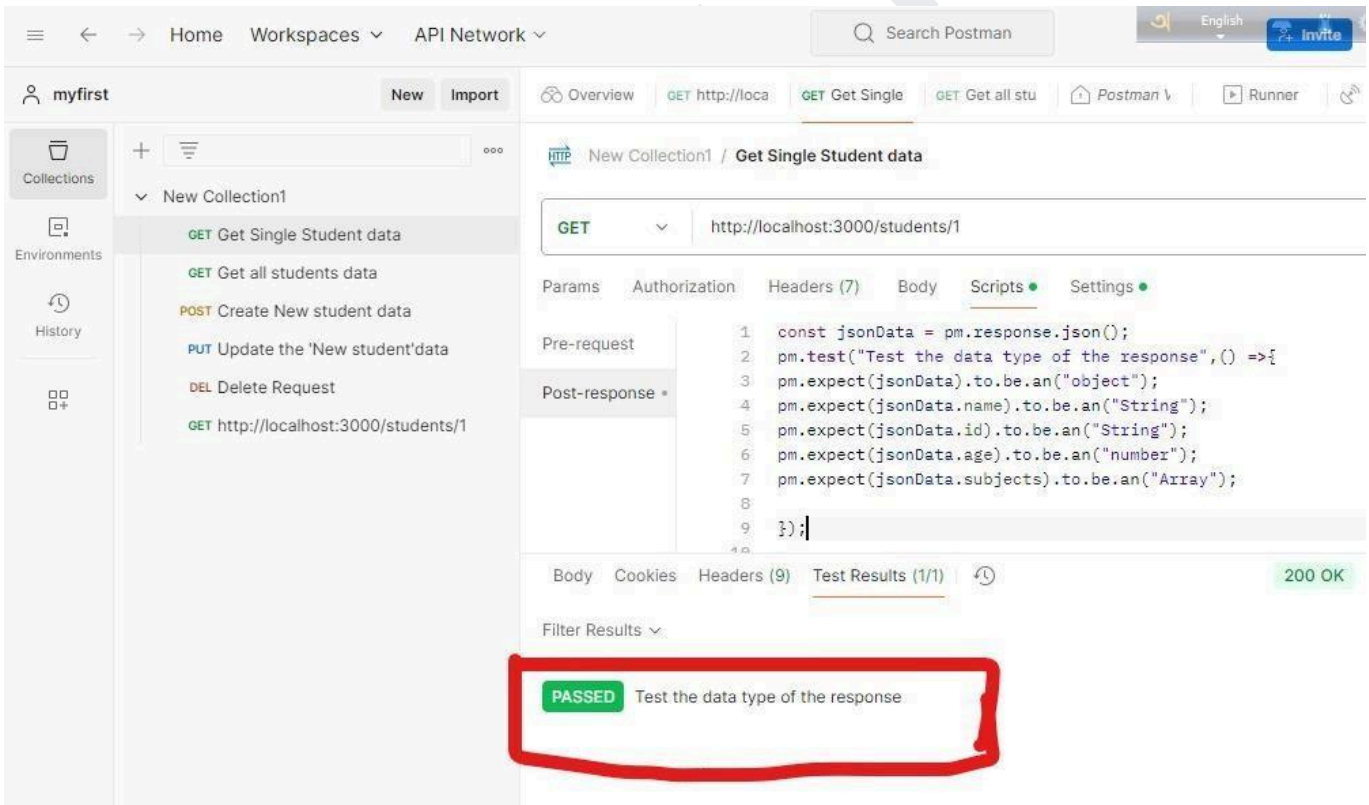
Post-response =

```
1 const jsonData = pm.response.json();
2 pm.test("Test the data type of the response", () =>{
3   pm.expect(jsonData).to.be.an("object");
4   pm.expect(jsonData.name).to.be.an("String");
5   pm.expect(jsonData.id).to.be.an("String");
6   pm.expect(jsonData.age).to.be.an("number");
7   pm.expect(jsonData.subjects).to.be.an("Array");
8
9 });
```

Body Cookies Headers (9) Test Results (1/1) 200 OK

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "1",
3   "name": "John Doe",
4   "age": 18,
5   "grade": "12th",
6   "subjects": [
7     "Math",
8     "Physics",
9     "English"
10  ]
11 }
```



Home Workspaces API Network Search Postman English Invite

myfirst New Import Overview GET http://loca GET Get Single GET Get all stu Postman v Runner New

New Collection1 / Get Single Student data

GET http://localhost:3000/students/1

Params Authorization Headers (7) Body Scripts Settings

Pre-request

Post-response =

```
1 const jsonData = pm.response.json();
2 pm.test("Test the data type of the response", () =>{
3   pm.expect(jsonData).to.be.an("object");
4   pm.expect(jsonData.name).to.be.an("String");
5   pm.expect(jsonData.id).to.be.an("String");
6   pm.expect(jsonData.age).to.be.an("number");
7   pm.expect(jsonData.subjects).to.be.an("Array");
8
9 });
```

Body Cookies Headers (9) Test Results (1/1) 200 OK

Filter Results

PASSED Test the data type of the response

(For **multiple object** in the response body);

CODE:

```
const jsonData = pm.response.json();

pm.test("Test the data type of the response", () => {

    pm.expect(jsonData).to.be.an("array");

});

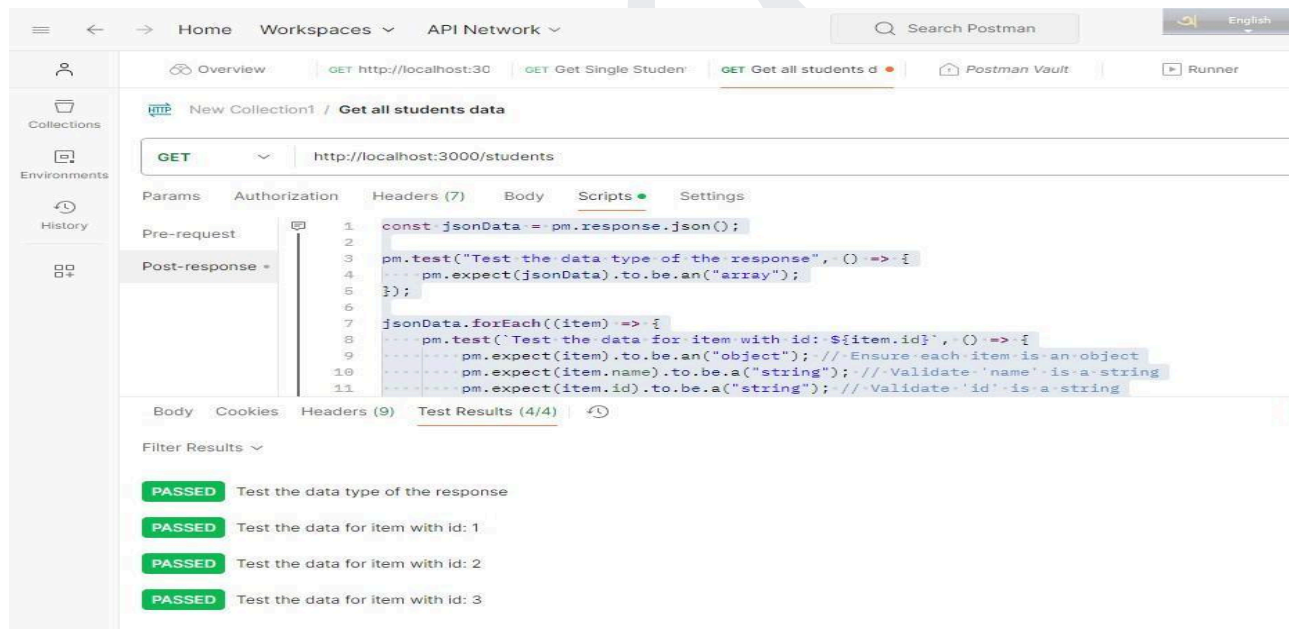
jsonData.forEach((item) => {

    pm.test(`Test the data for item with id: ${item.id}`, () => {

        pm.expect(item).to.be.an("object"); // Ensure each item is an object
        pm.expect(item.name).to.be.a("string"); // Validate 'name' is a string
        pm.expect(item.id).to.be.a("string"); // Validate 'id' is a string
        pm.expect(item.age).to.be.a("number"); // Validate 'age' is a number
        pm.expect(item.subjects).to.be.an("array"); // Validate 'subjects' is an array

    });

});
```



2. Validate the Array Properties/Array Content in the fields:

Array properties in the response body :

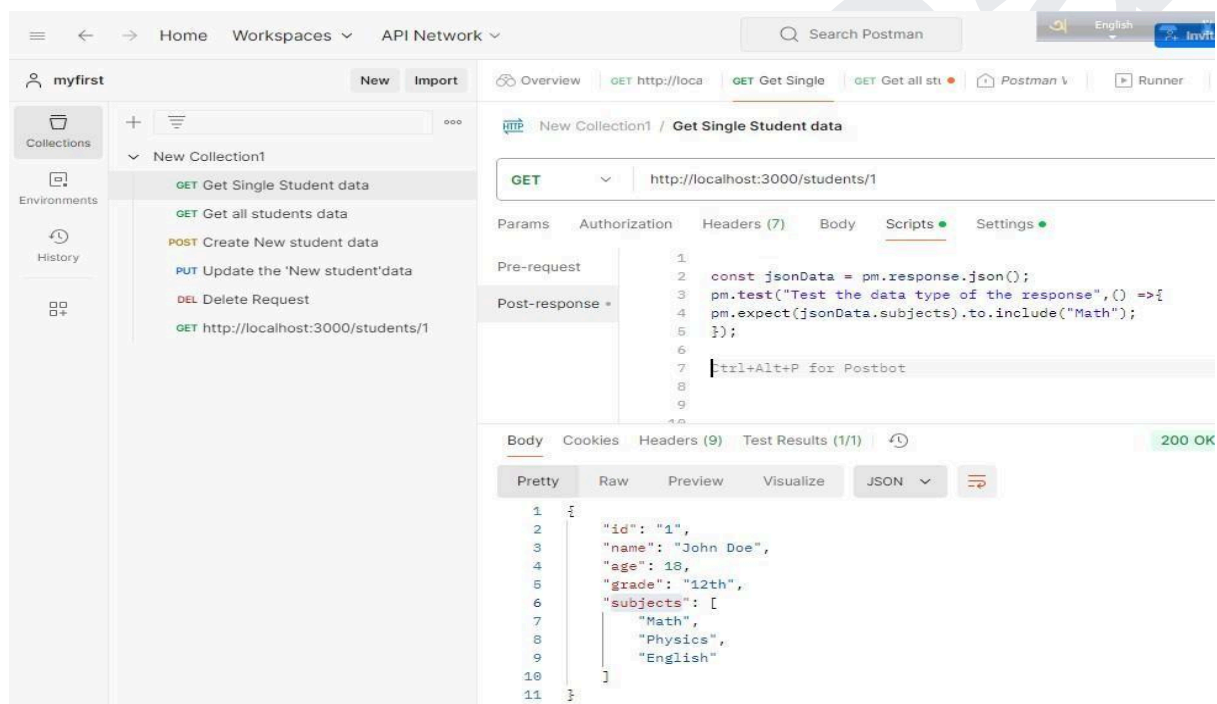
How to validate the Array properties in the response body ?

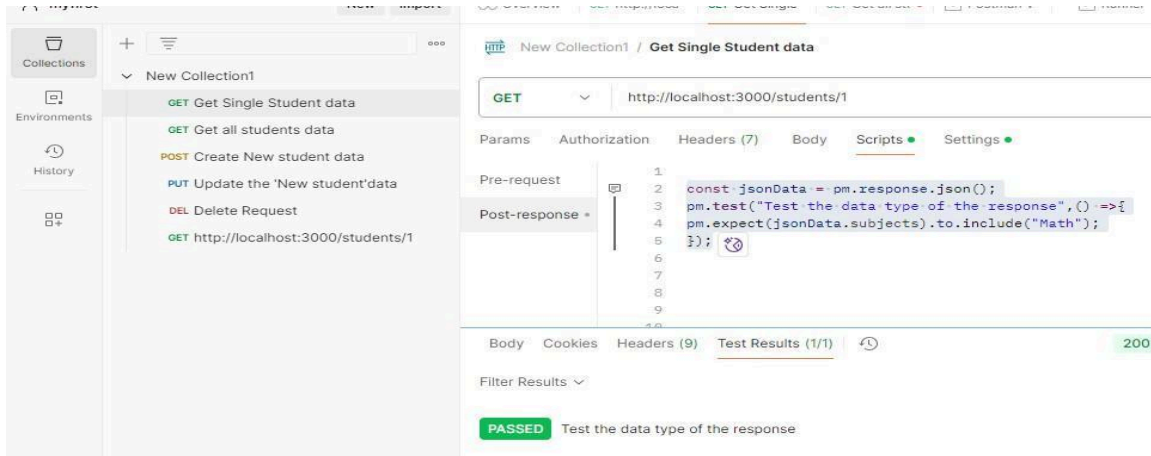
-(Validate a **single property** in the Array from the response body)

CODE:

```
const jsonData = pm.response.json();  
pm.test("Test the data type of the response", () =>{  
  pm.expect(jsonData.subjects).to.include("Math");  
});
```

Input:





-(Validate a Multiple property in the Array from the response body)

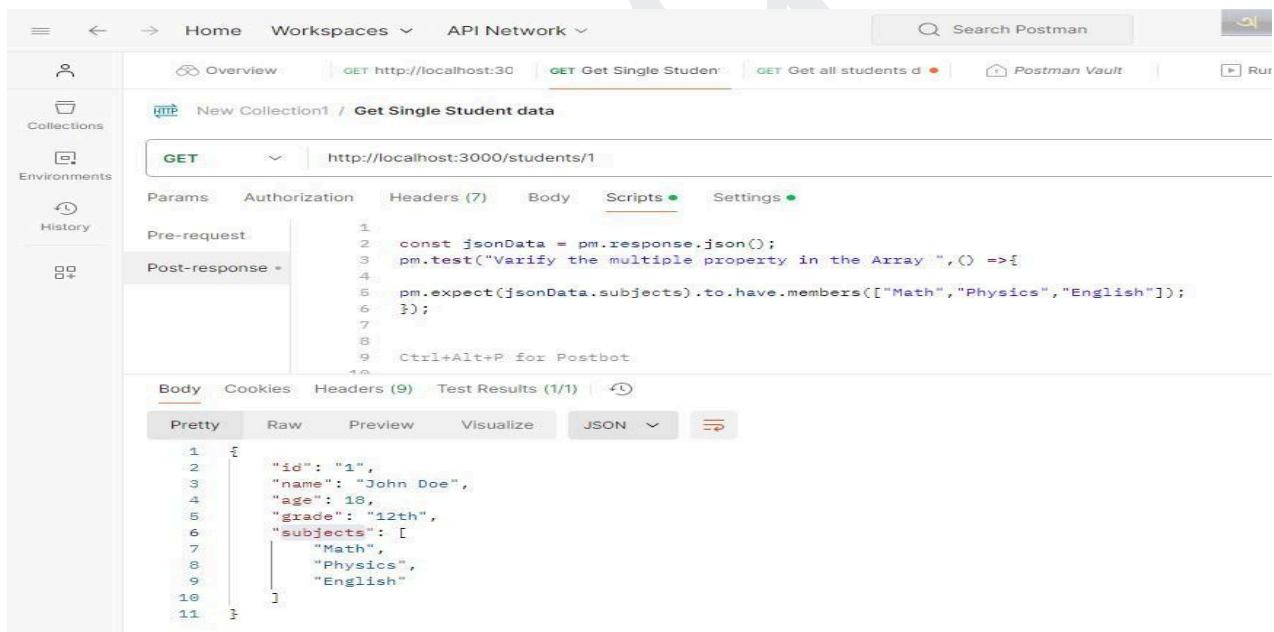
CODE:

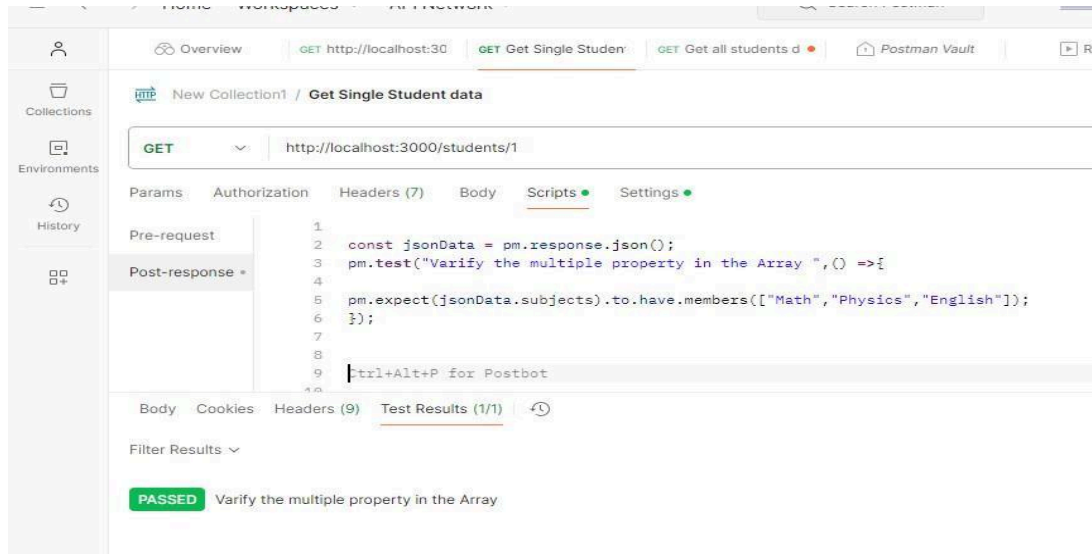
```

const jsonData = pm.response.json();
pm.test("Verify the multiple property in the Array ",() =>{
  pm.expect(jsonData.subjects).to.have.members(["Math","Physics","English"]);
});

```

Input:





3. Validate the Data of the fields are matched/Correct or Not :

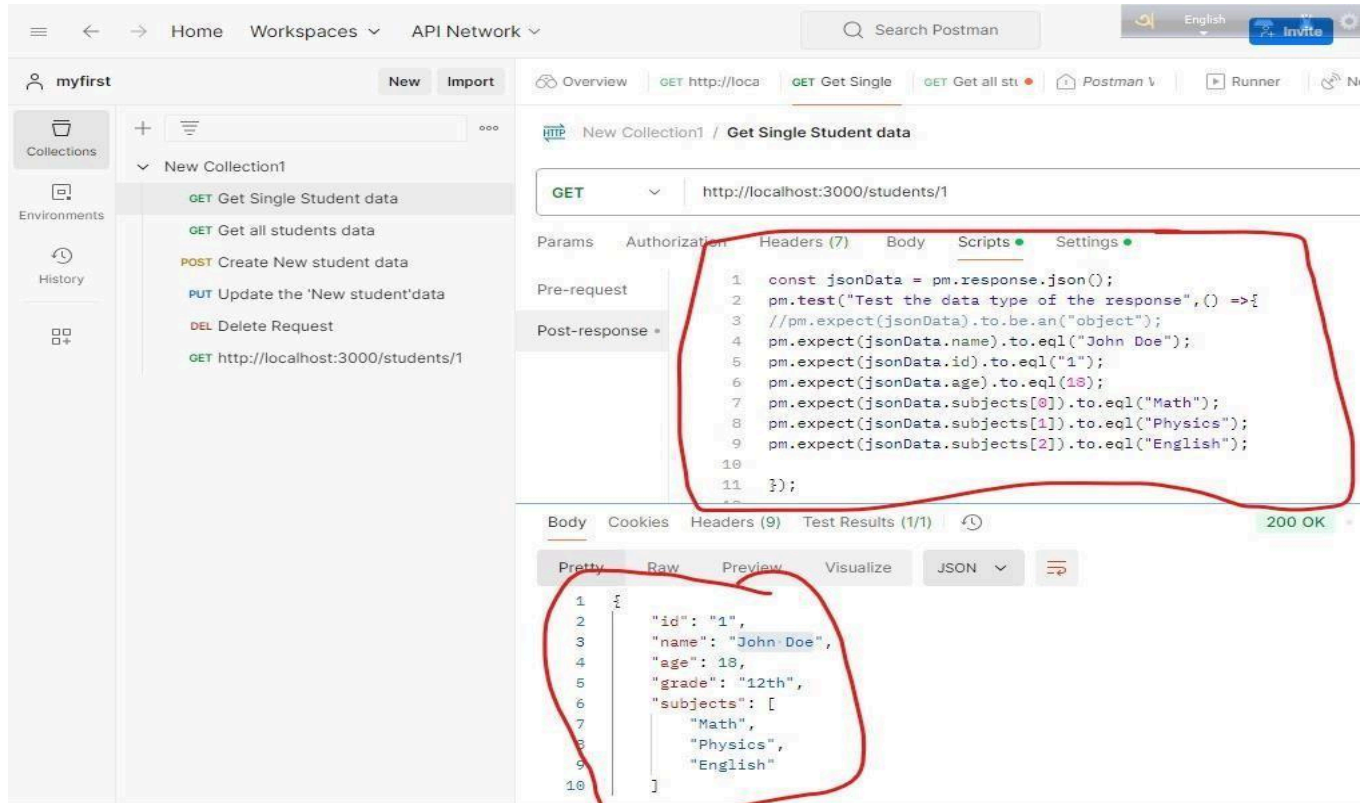
Validate the “value “ Of the response body

How to validate the value of of every field/assertion is match or not from the response body?

Ans:

CODE:

```
const jsonData = pm.response.json();
pm.test("Test the data type of the response", () =>{
//pm.expect(jsonData).to.be.an("object");
pm.expect(jsonData.name).to.eql("John Doe");
pm.expect(jsonData.id).to.eql("1");
pm.expect(jsonData.age).to.eql(18);
pm.expect(jsonData.subjects[0]).to.eql("Math");
pm.expect(jsonData.subjects[1]).to.eql("Physics");
pm.expect(jsonData.subjects[2]).to.eql("English");
});
```

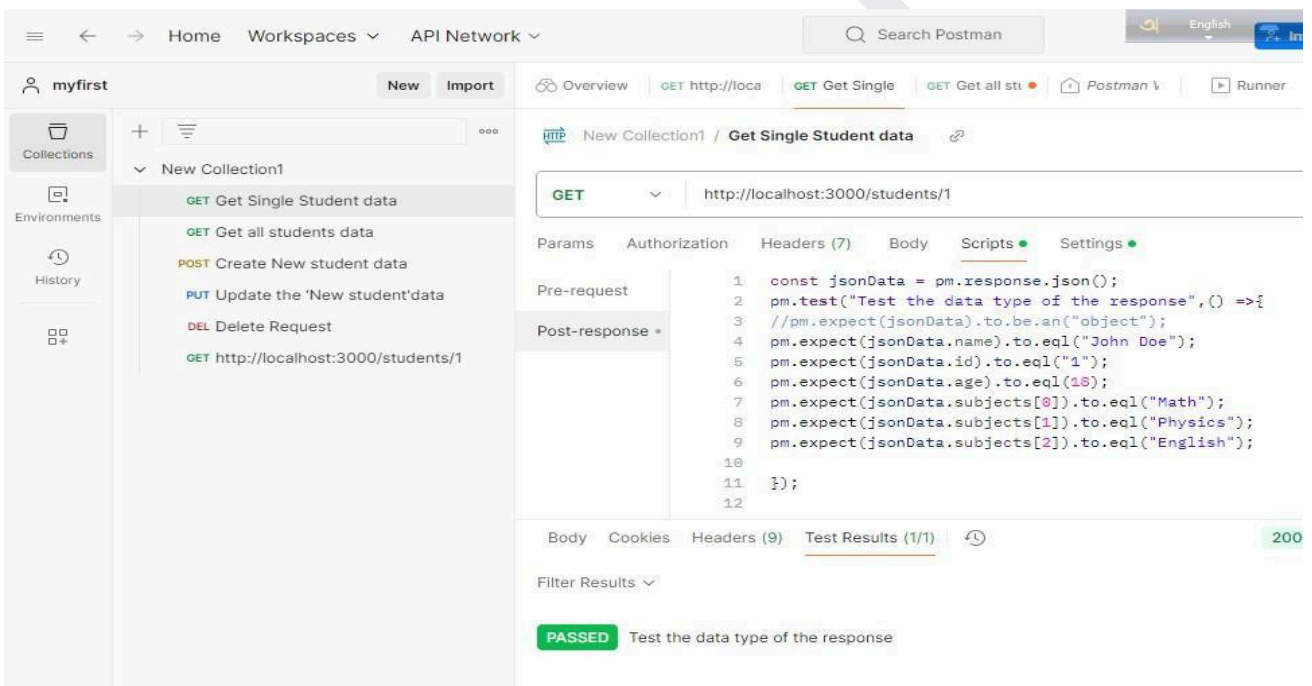


The screenshot shows the Postman interface with a new collection named 'New Collection1' containing a GET request 'Get Single Student data' to 'http://localhost:3000/students/1'. The 'Scripts' tab is active, showing a JavaScript script to validate the response structure. The script is as follows:

```
1 const jsonData = pm.response.json();
2 pm.test("Test the data type of the response", () =>{
3 //pm.expect(jsonData).to.be.an("object");
4 pm.expect(jsonData.name).to.eql("John Doe");
5 pm.expect(jsonData.id).to.eql("1");
6 pm.expect(jsonData.age).to.eql(18);
7 pm.expect(jsonData.subjects[0]).to.eql("Math");
8 pm.expect(jsonData.subjects[1]).to.eql("Physics");
9 pm.expect(jsonData.subjects[2]).to.eql("English");
10
11 });
```

The 'Body' tab is also active, showing the JSON response in 'Pretty' format:

```
1 {
2   "id": "1",
3   "name": "John Doe",
4   "age": 18,
5   "grade": "12th",
6   "subjects": [
7     "Math",
8     "Physics",
9     "English"
10  ]
11 }
```



The screenshot shows the same Postman interface, but now the 'Test Results' tab is active. It displays a single test result: 'PASSED Test the data type of the response'.

4. Validate the Json schema :

Validate the Json Schema :

Convert Json to JSON schema link : [Link](#)

Sample JSON Document

```
1 {  
2   "id": "1",  
3   "name": "John Doe",  
4   "age": 18,  
5   "grade": "12th",  
6   "subjects": [  
7     "Math",  
8     "Physics",  
9     "English"  
10  ]  
11 }
```

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "object",  
  "properties": {  
    "id": {  
      "type": "string"  
    },  
    "name": {  
      "type": "string"  
    },  
    "age": {  
      "type": "integer"  
    },  
    "grade": {  
      "type": "string"  
    },  
    "subjects": {  
      "type": "array",  
      "items": [  
        {  
          "type": "string"  
        },  
        {  
          "type": "string"  
        },  
        {  
          "type": "string"  
        }  
      ]  
    }  
  },  
  "required": [  
    "id",  
    "name",  
    "age",  
    "grade",  
    "subjects"  
  ]  
}
```

Schema validation code: //Validate Json Schema of the response body.

```
pm.test("Status code is 200", function () {  
    pm.expect(tv4.validate(jsonData, schema)).to.be.true;  
});
```

PostMan variables

What is a variable?

Ans: Variable is something which contains some data.

Why is variable need in postman?

Ans: Variable Is used to avoid the duplicate value

Where is use variable in Postman?

Ans : Variable is used in multiple level like Collection, and Environment. Request level.

Scope?

Ans : where we can create the variables

Scope to set up the variables:

1. Global variable ---Set the variable in global level
2. Collection variable Set the variable in collection level
3. Request variable Set the variable in request level
4. Environment variable Set the variable in environment level
5. Data variable Set the variable in data level.

1. Global variable:

How do we create a global variable?

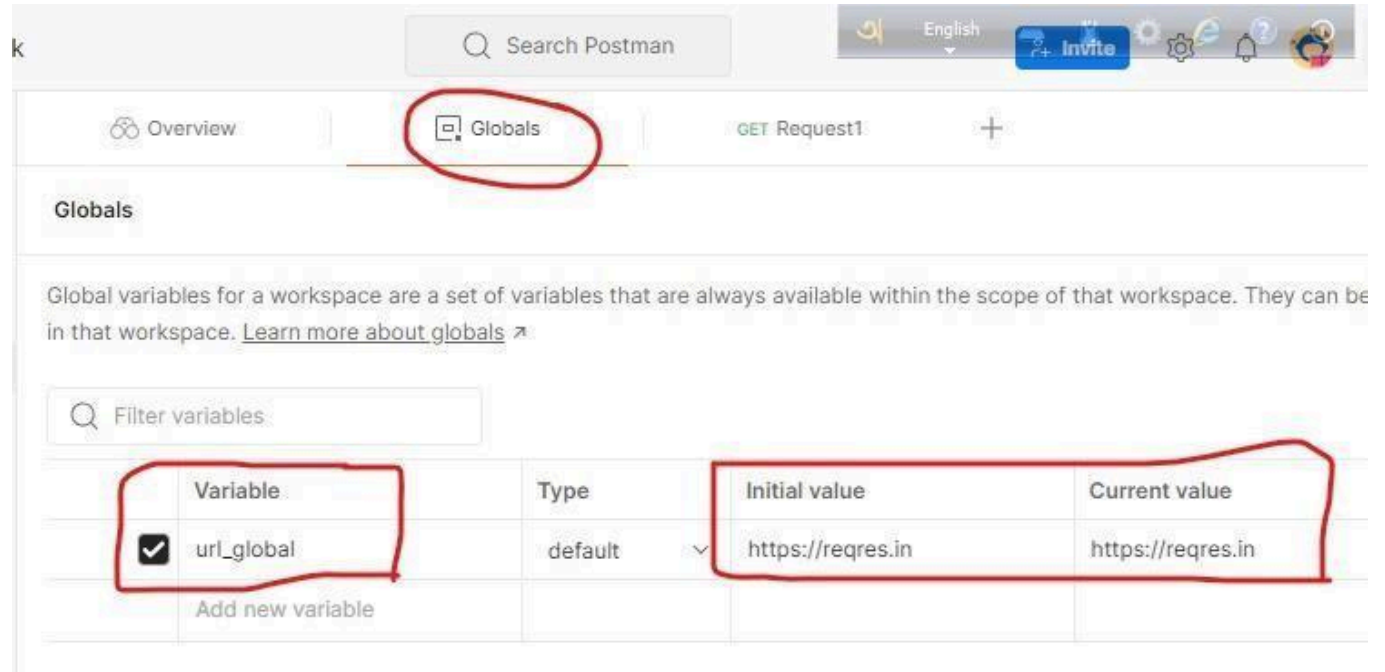
Ans: workspace--□Collection □Request/Folder.

A global variable is accessible throughout the every workspace.

Step-1

Create a **Global variable** on the collection and save.

Step-2

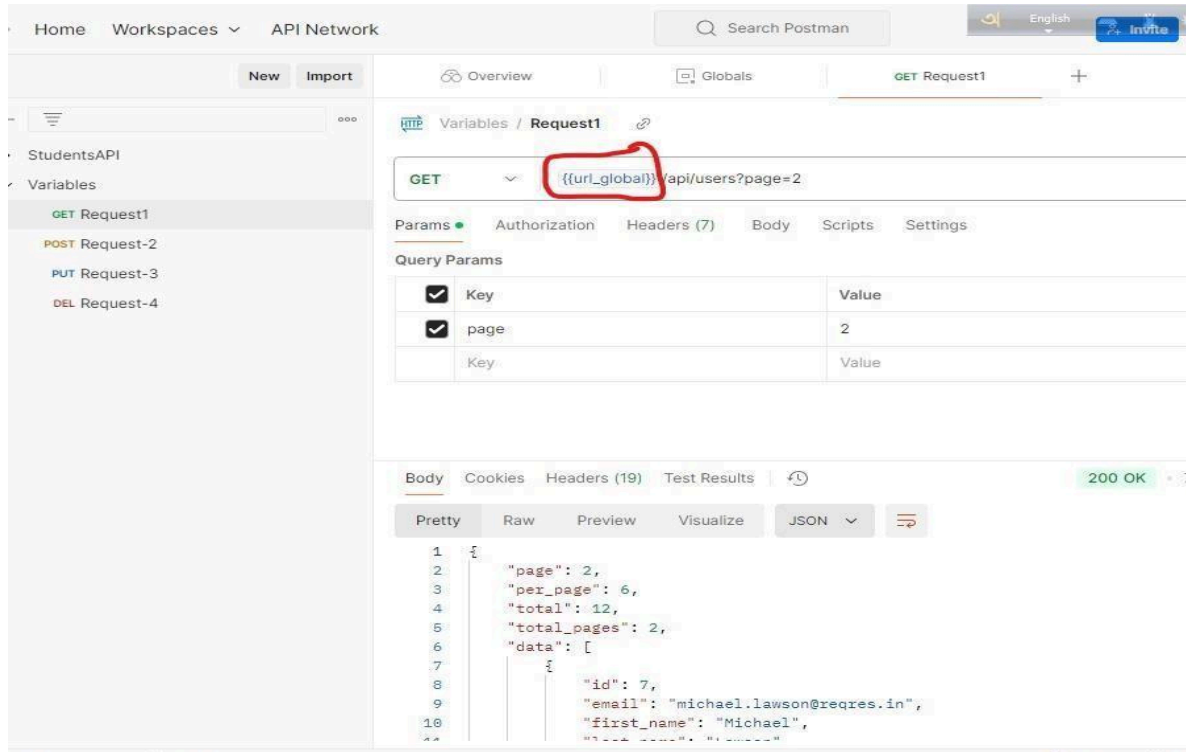


Global variables for a workspace are a set of variables that are always available within the scope of that workspace. They can be used in that workspace. [Learn more about globals](#)

Filter variables

Variable	Type	Initial value	Current value
<input checked="" type="checkbox"/> url_global	default	https://reqres.in	https://reqres.in
Add new variable			

Step-3 (Get Request)



Home Workspaces API Network Search Postman English Invite

New Import Overview Globals GET Request1 +

Variables / Request1

GET `{{url_global}}/api/users?page=2`

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value
page	2
Key	Value

Body Cookies Headers (19) Test Results 200 OK

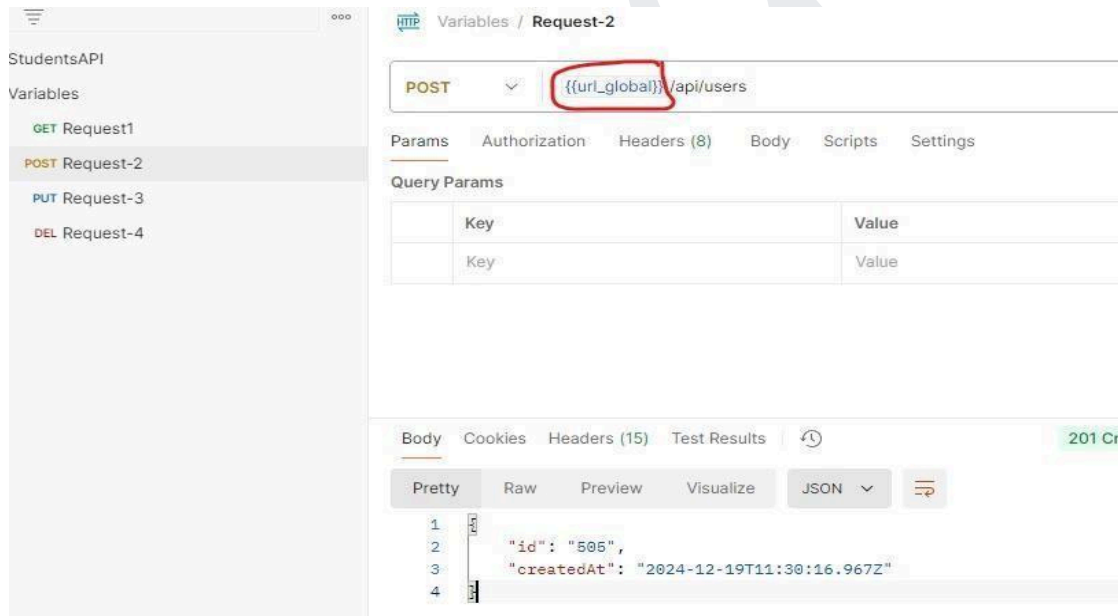
Pretty Raw Preview Visualize JSON

```

1 {
2   "page": 2,
3   "per_page": 6,
4   "total": 12,
5   "total_pages": 2,
6   "data": [
7     {
8       "id": 7,
9       "email": "michael.lawson@reqres.in",
10      "first_name": "Michael",
11      "last_name": "Lawson"
12    }
13  ]
14 }
```

Step-5 ----(Post Request)

Put this “ `{{url_global}}` ” variable in every collection accordingly to change the value



Variables / Request-2

POST `{{url_global}}/api/users`

Params Authorization Headers (8) Body Scripts Settings

Query Params

Key	Value
Key	Value

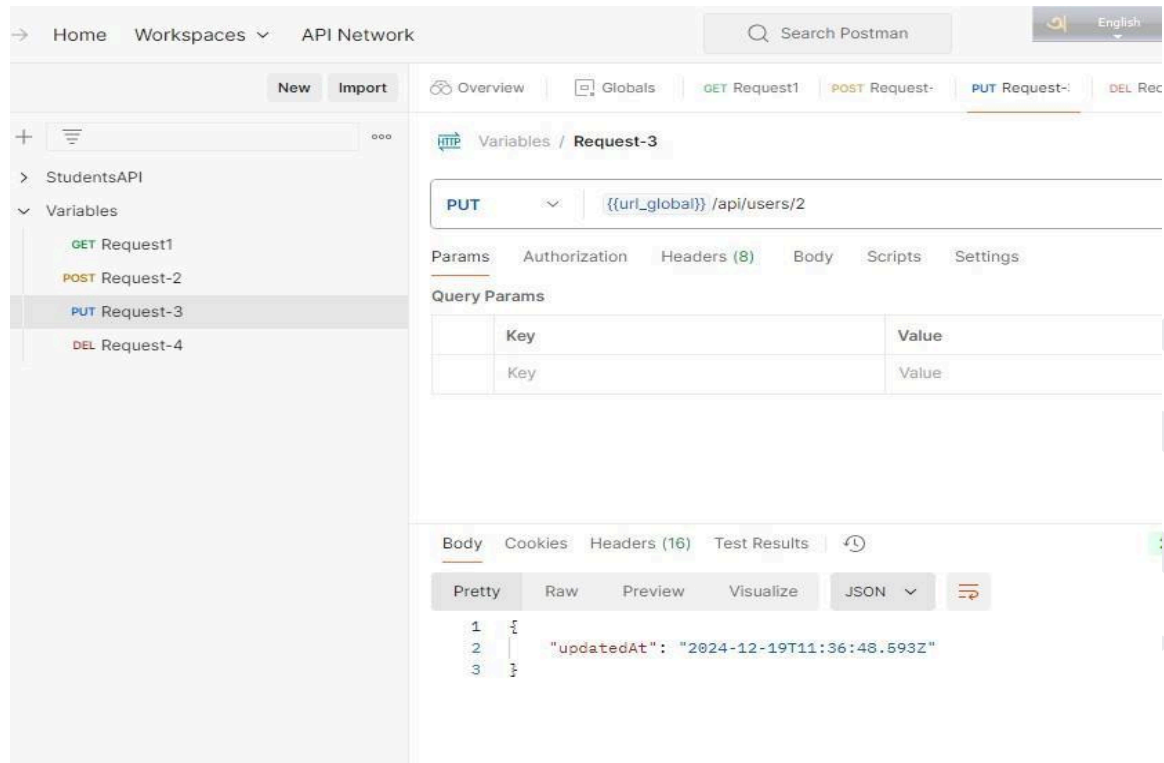
Body Cookies Headers (15) Test Results 201 Cr

Pretty Raw Preview Visualize JSON

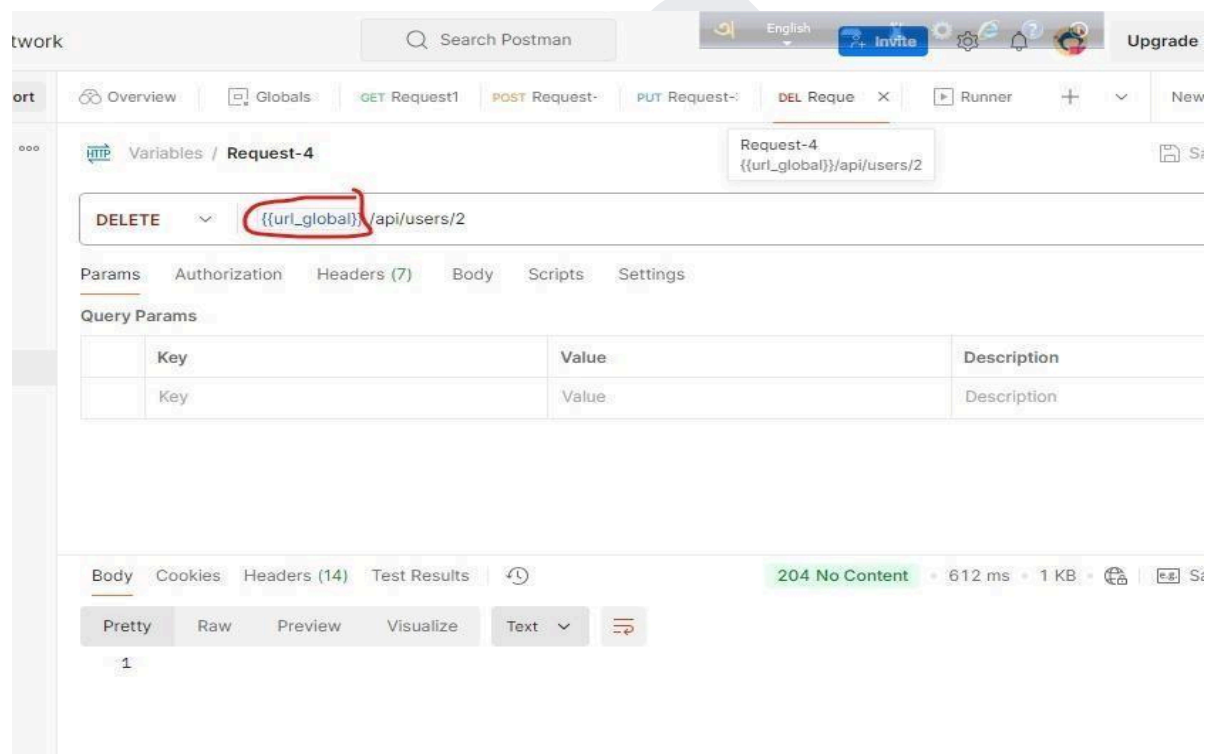
```

1 {
2   "id": "505",
3   "createdAt": "2024-12-19T11:30:16.967Z"
4 }
```

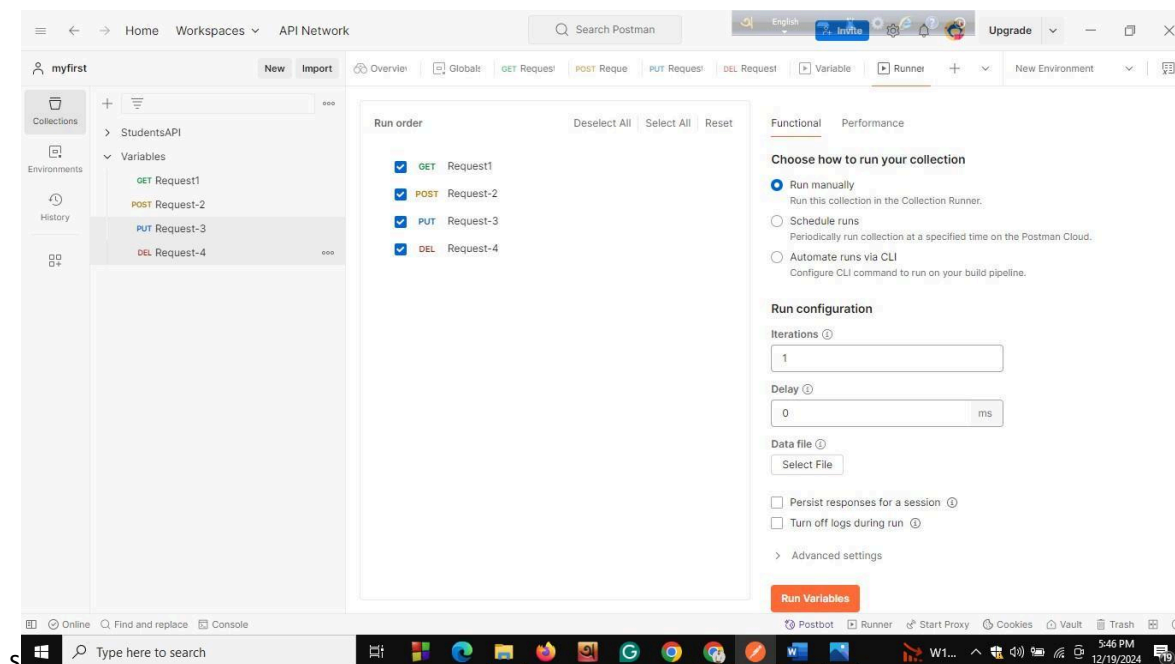
Step-6 (Put Request)



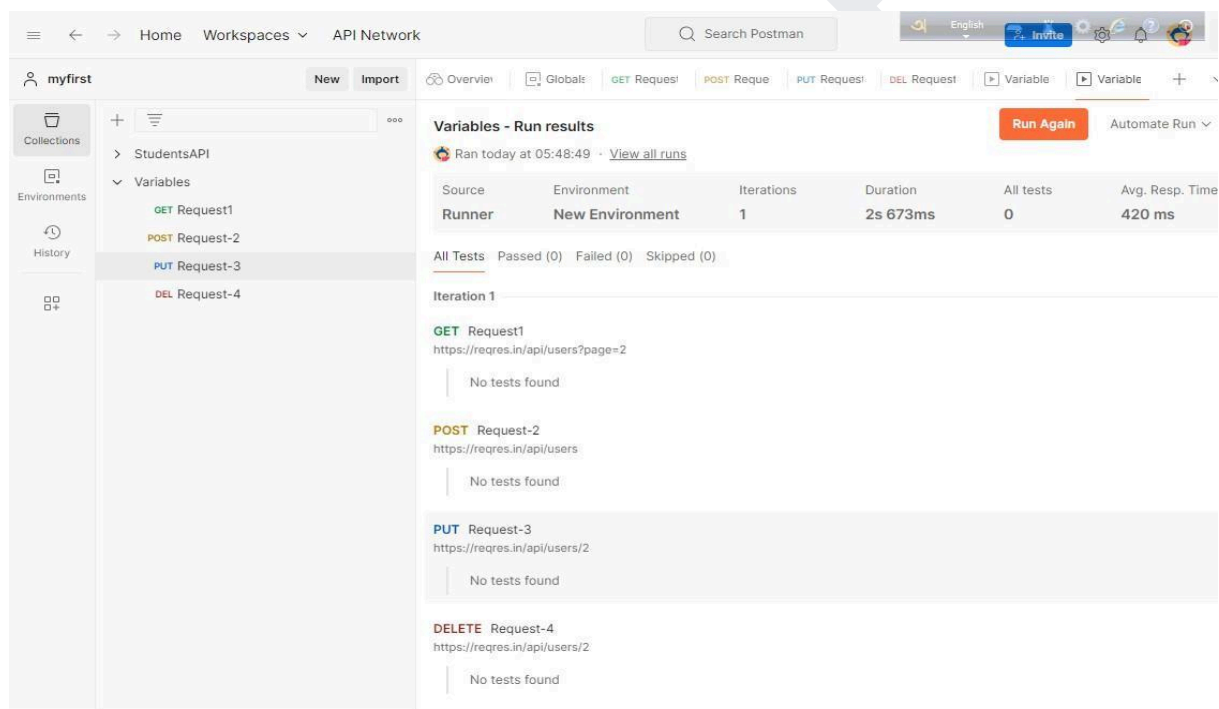
Step-7 (Delete Request)



Step-8

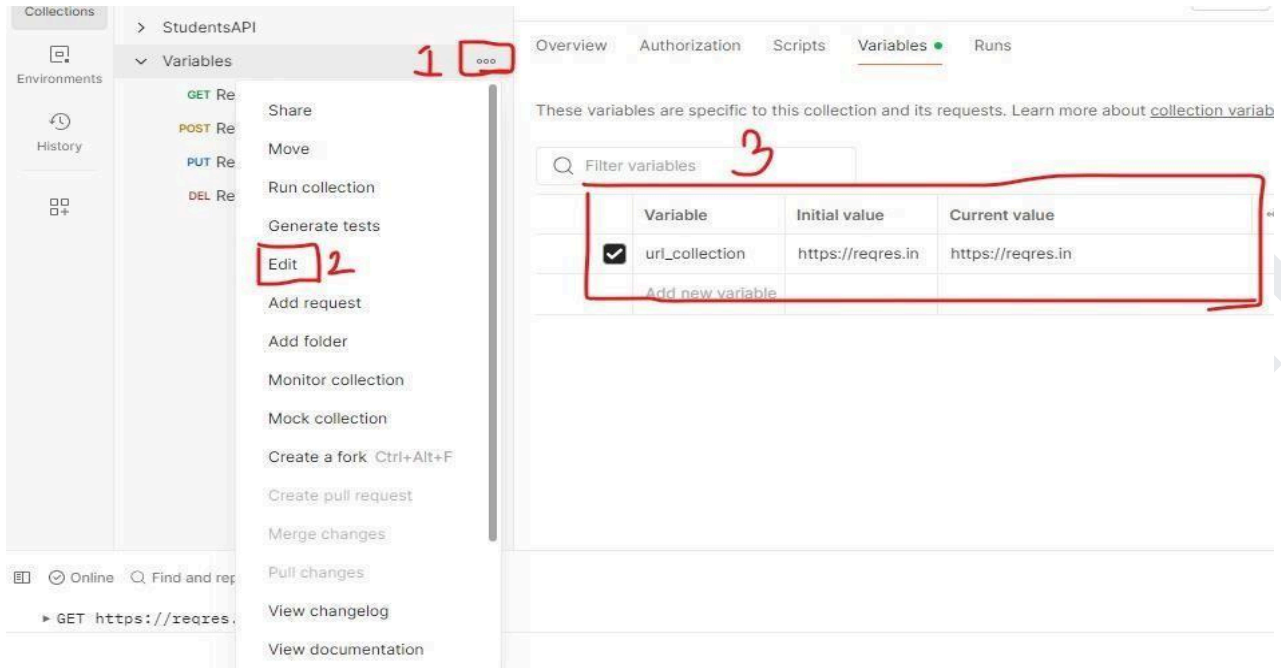


Step-9



Collection variable : **Collection variable** is accessible within the Collection among multiple requests.

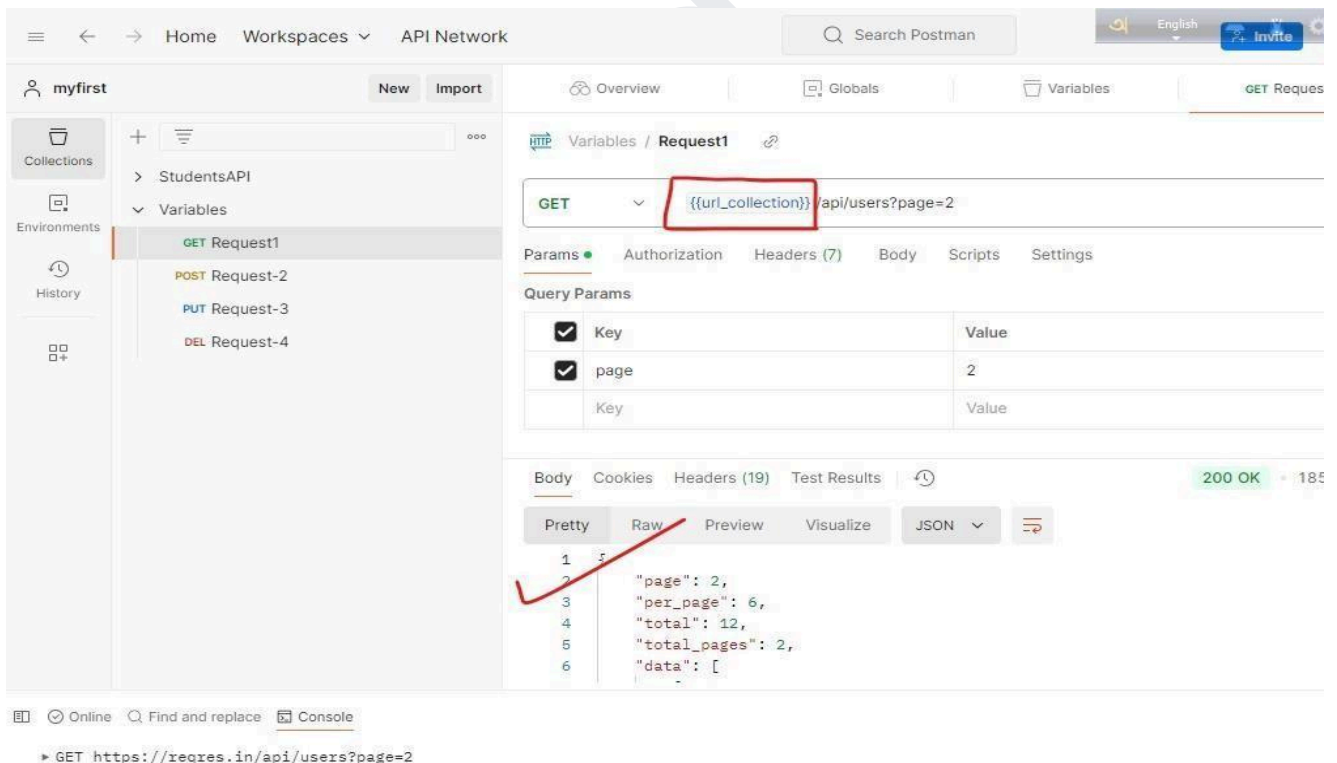
Step-1



These variables are specific to this collection and its requests. Learn more about [collection variables](#)

Variable	Initial value	Current value
<input checked="" type="checkbox"/> url_collection	https://reqres.in	https://reqres.in
Add new variable		

Step-2



GET `https://reqres.in/api/users?page=2`

Params • Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value
<input checked="" type="checkbox"/> page	2
<input type="checkbox"/> Key	Value

Body Cookies Headers (19) Test Results

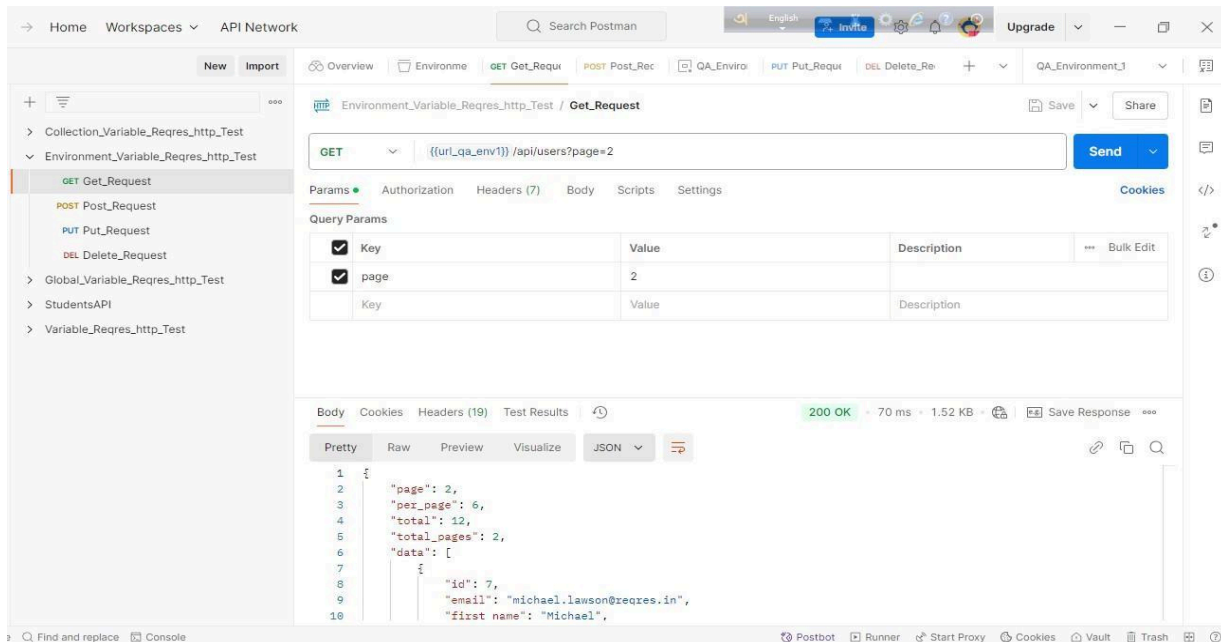
200 OK • 185

```

1 {
2   "page": 2,
3   "per_page": 6,
4   "total": 12,
5   "total_pages": 2,
6   "data": [

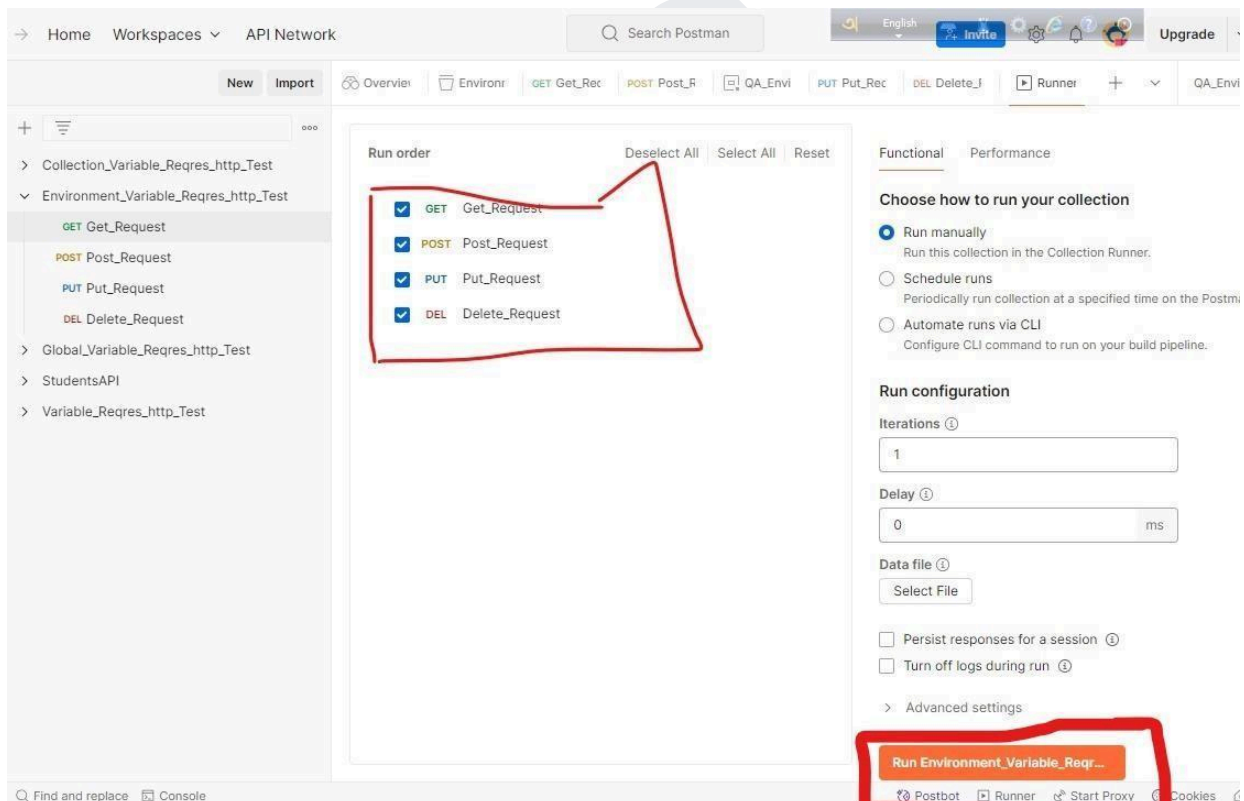
```

Environment Variable: Accessible in all collections but we use it for a specific environment

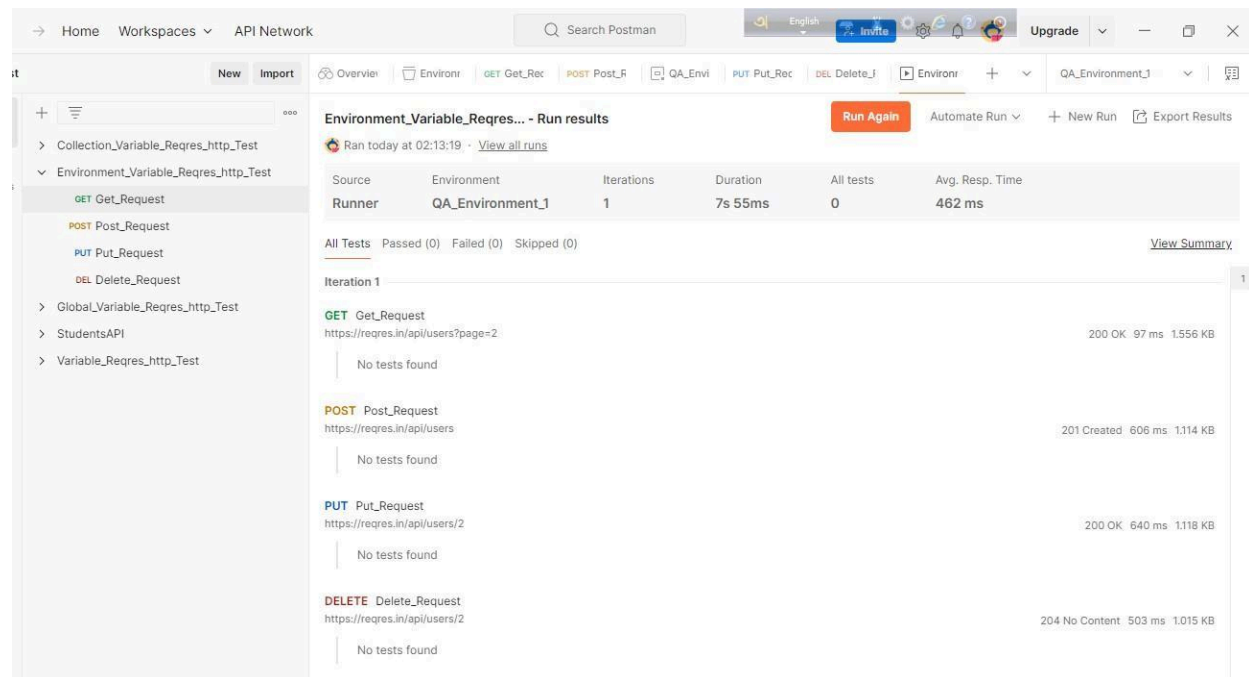


Run the whole collection of Environment Variable:

Step-1



Step-2



Environment Variable Regres... - Run results

Ran today at 02:13:19 · [View all runs](#)

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	QA_Environment_1	1	7s 55ms	0	462 ms

All Tests Passed (0) Failed (0) Skipped (0) [View Summary](#)

Iteration 1

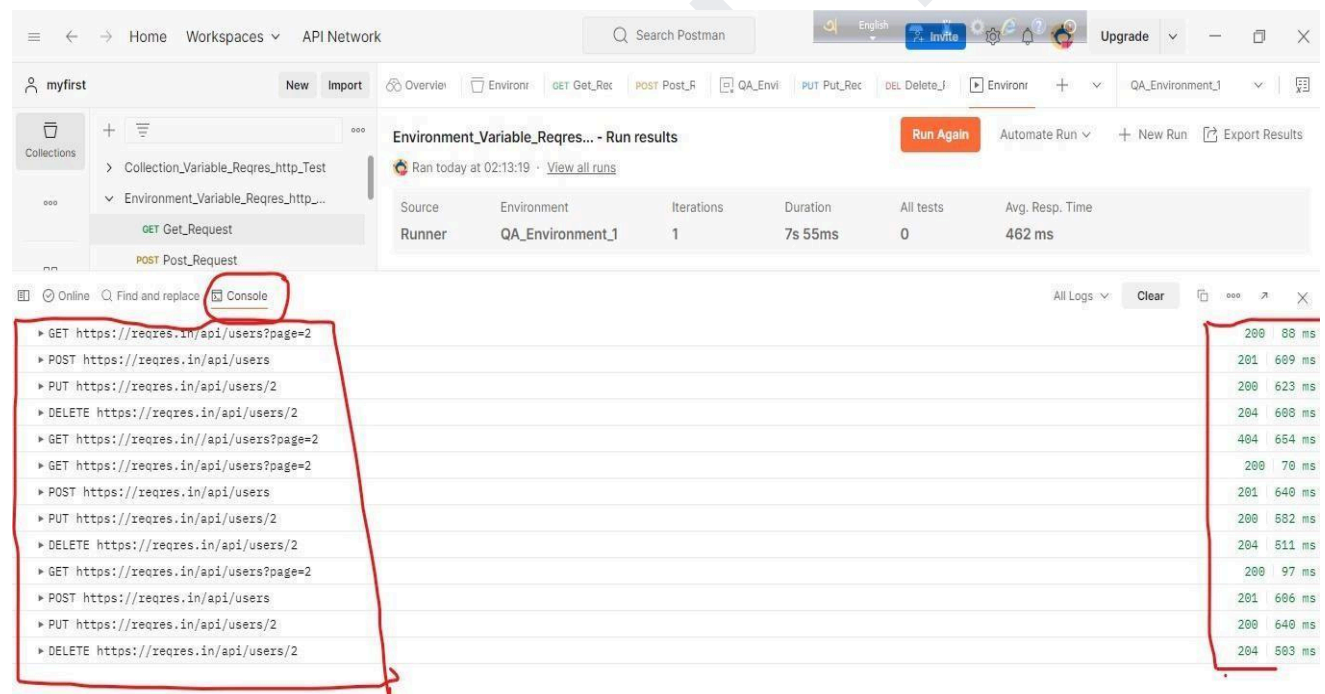
GET Get_Request
https://reqres.in/api/users?page=2
200 OK 97 ms 1.556 KB
No tests found

POST Post_Request
https://reqres.in/api/users
201 Created 606 ms 1.114 KB
No tests found

PUT Put_Request
https://reqres.in/api/users/2
200 OK 640 ms 1.118 KB
No tests found

DELETE Delete_Request
https://reqres.in/api/users/2
204 No Content 503 ms 1.015 KB
No tests found

Console result



Environment Variable Regres... - Run results

Ran today at 02:13:19 · [View all runs](#)

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	QA_Environment_1	1	7s 55ms	0	462 ms

All Tests Passed (0) Failed (0) Skipped (0) [View Summary](#)

Iteration 1

GET Get_Request
https://reqres.in/api/users?page=2
200 OK 97 ms 1.556 KB
No tests found

POST Post_Request
https://reqres.in/api/users
201 Created 606 ms 1.114 KB
No tests found

PUT Put_Request
https://reqres.in/api/users/2
200 OK 640 ms 1.118 KB
No tests found

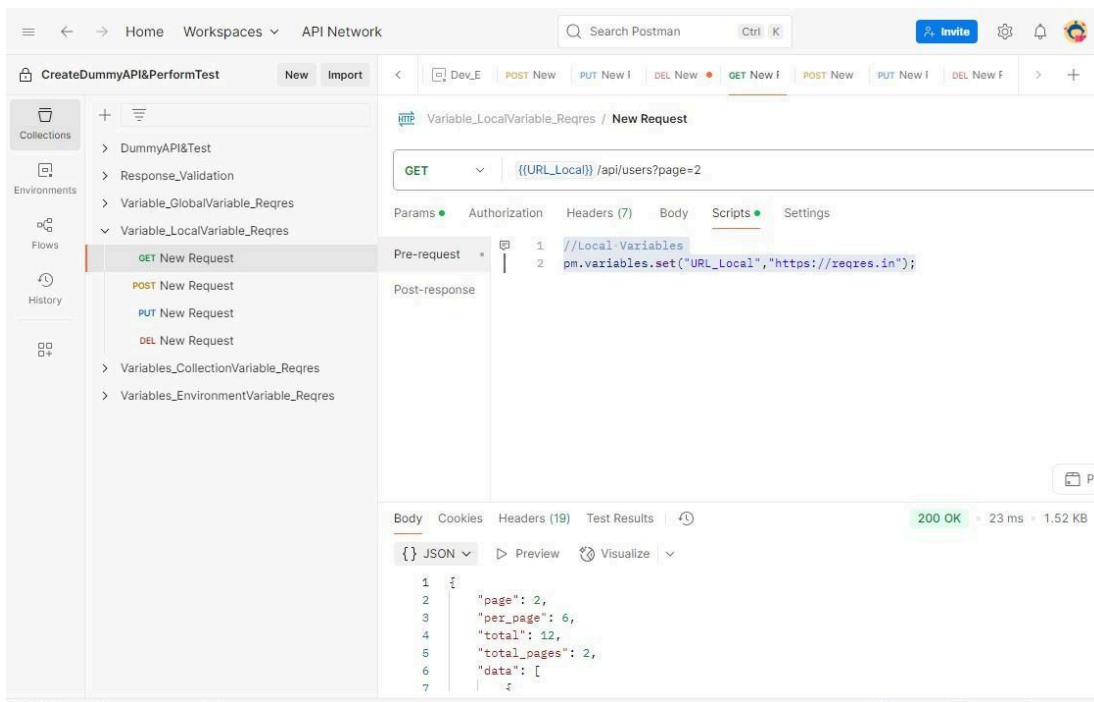
DELETE Delete_Request
https://reqres.in/api/users/2
204 No Content 503 ms 1.015 KB
No tests found

Console

```

▶ GET https://reqres.in/api/users?page=2 200 88 ms
▶ POST https://reqres.in/api/users 201 609 ms
▶ PUT https://reqres.in/api/users/2 200 623 ms
▶ DELETE https://reqres.in/api/users/2 204 608 ms
▶ GET https://reqres.in/api/users?page=2 404 654 ms
▶ GET https://reqres.in/api/users?page=2 200 70 ms
▶ POST https://reqres.in/api/users 201 640 ms
▶ PUT https://reqres.in/api/users/2 200 582 ms
▶ DELETE https://reqres.in/api/users/2 204 511 ms
▶ GET https://reqres.in/api/users?page=2 200 97 ms
▶ POST https://reqres.in/api/users 201 606 ms
▶ PUT https://reqres.in/api/users/2 200 640 ms
▶ DELETE https://reqres.in/api/users/2 204 503 ms
  
```

Local Variables: Accessible only within the request.

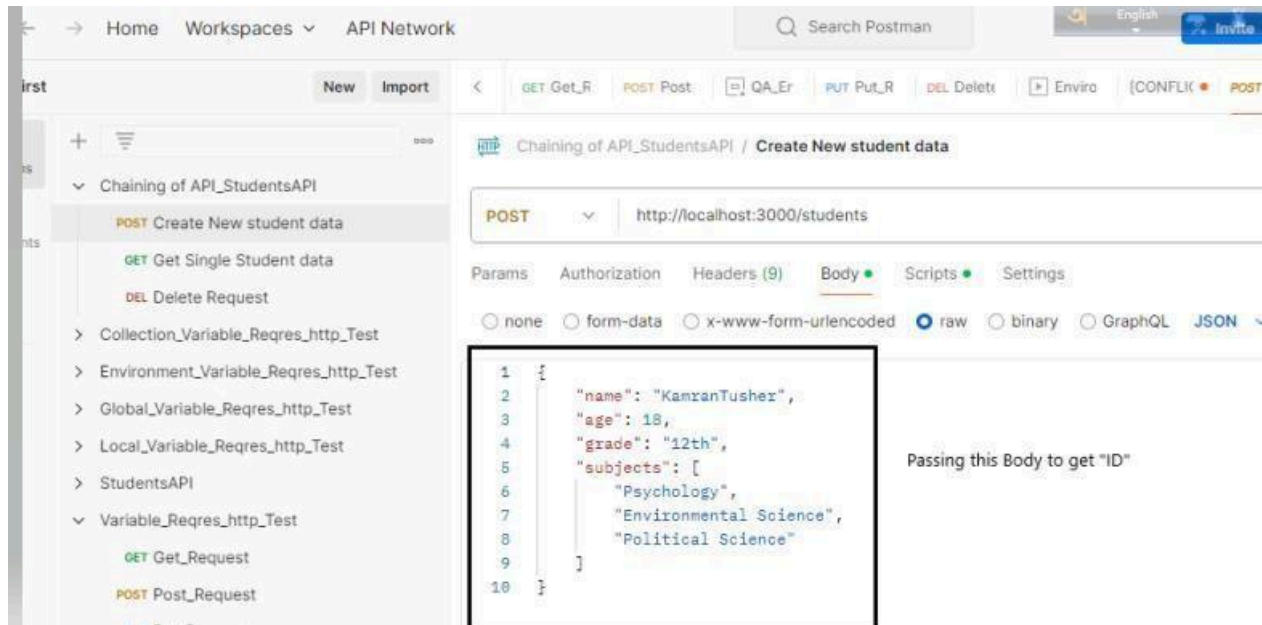


Chaining of API : The response of one API becomes the request of another API is called Chaining of API.

Task:

When I pass a body it will create an 'ID' in response to request from the body. Then this 'ID' will be stored in a variable.

Step-1) Pass this body



Step-2

Write a test script in the “ post-response “ body.

Script

```
var jsonData= JSON.parse(ResponseBody);
```

```
pm.environment("id".jsonData.id);
```

Here this particular script will set an environment and give a id for the particular body which is given on the top.

Another Sample API from internet:

Step -1 : Take an API from the website : (https://gorest.co.in/#google_vignette).Here we will find some sample API .

Step-2 : To access this API We need to generate a token and pass it as part of the Authorization.

Note: Most of the time , whatever API is accessing through internet, those API would have some authorization

How to get Access token?

Sign up for Github/Google and click on anyone then get the access.

Sample API:

URL: <https://gorest.co.in/>

End Point

POST	/public/v2/users	Create a new user
GET	/public/v2/users/7386739	Get user details
PUT PATCH	/public/v2/users/7386739	Update user details
DELETE	/public/v2/users/7386739	Delete user

Created Token: cf842adb5472f7db0196c587b55a74dfccd50e82e0f361ab59326843207addf

Response Body

```
{  
  "name": "Chowdhury Kamran Hossain",  
  "gender": "male",  
  "email": "ckh123@gmail.com",  
  "status": "active"  
}
```

Note: This Response Body is required for Post and Put requests only.

Note: Then Use the token in the Authorization section in the Collection level to cover all the request

GorestApi_Chaining example :

Process:

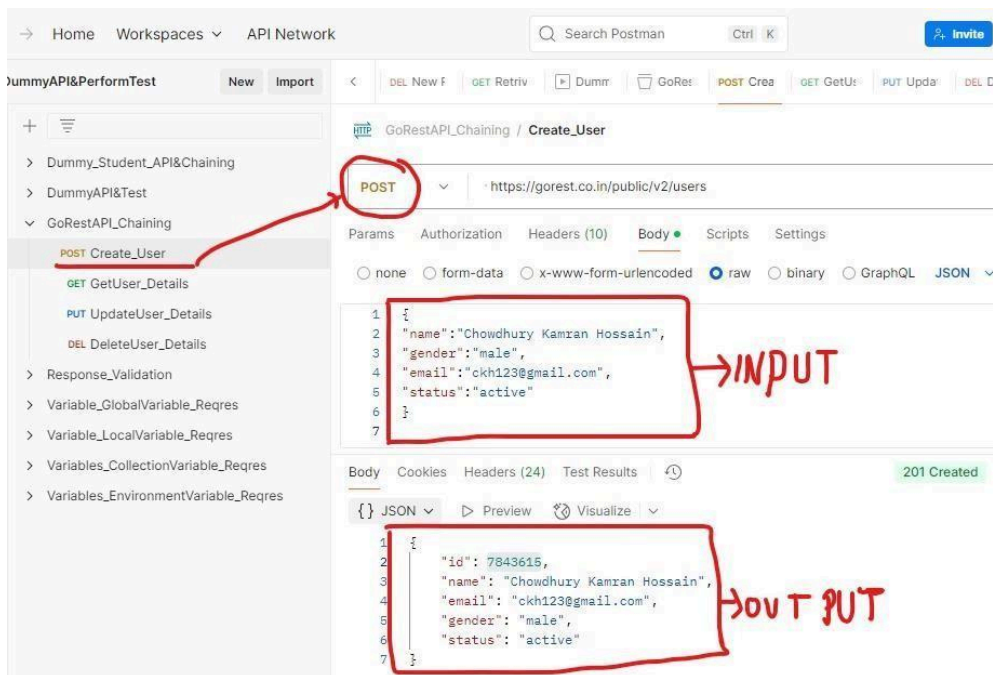
Step-1: Create a Collection called "GoRestAPI_Chaining"

Step-2: Create Post , Get, Put and Delete Request along with the given Url

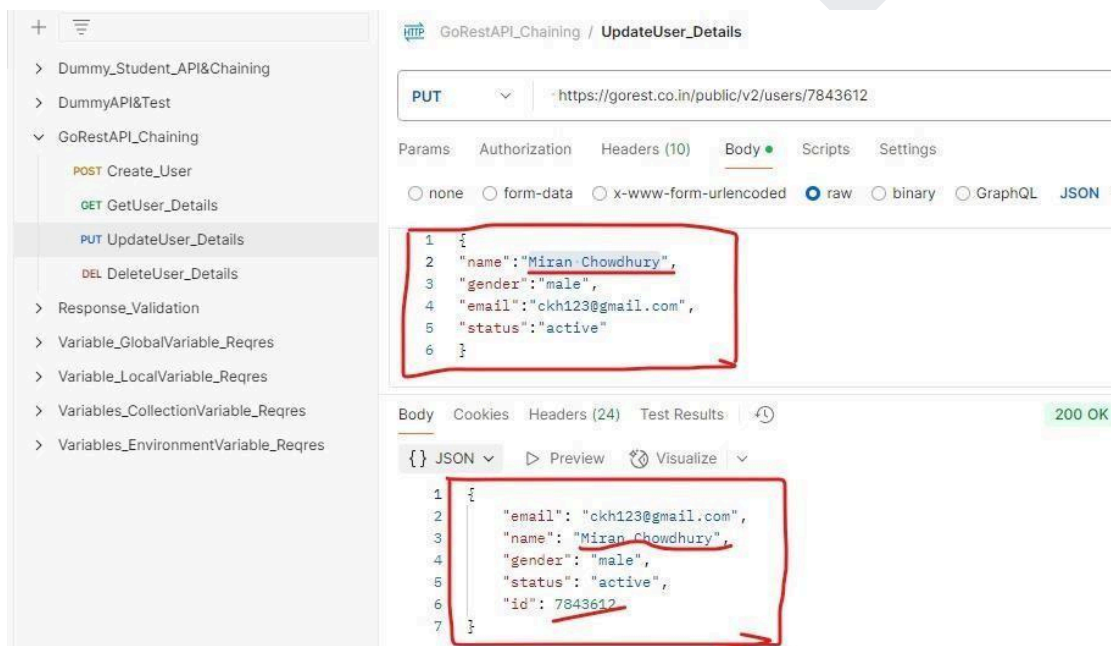
Step-3: Set the Token in the collection Level for authorization

Step-4 : Then Execute all the request

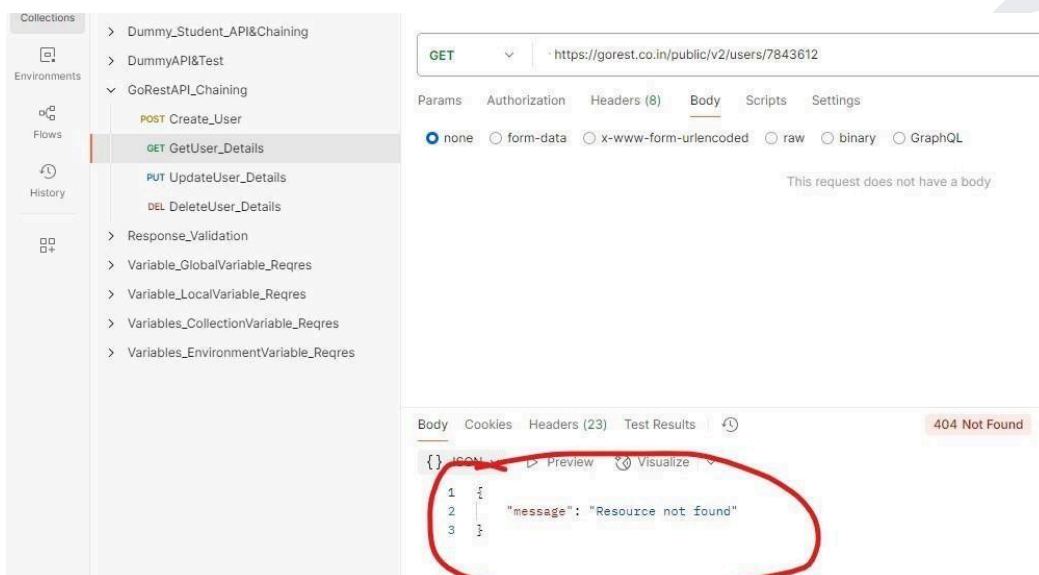
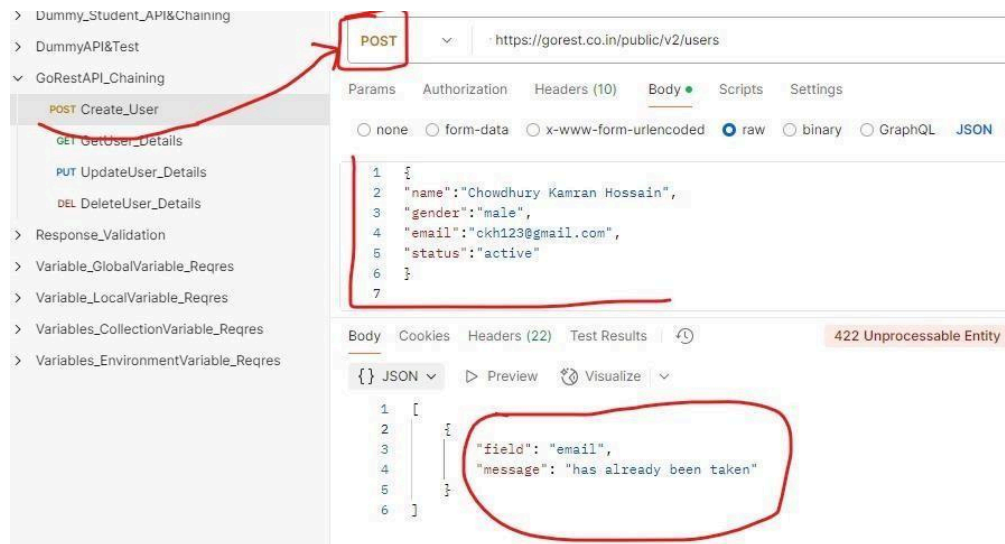
Step-1: First of all execute a “ Post Request “ create a record.



After getting the result I have updated some informations against the ID which is generated.



The request will not be executed further with the same records. Below you can see the result Image:



That is why we need to change the information of the record simultaneously. But it is difficult to change the data manually in everytime, that is why we need to make it as automated.

How to change the data in the records automatically every time?

Ans: Before sending the 'Post Request' we need to **write some script** to change the date in the record automatically.

We will write the Script in the 'Pre-request' section.

Here will change the Name and Email automatically

Here is the process ;

Firstly , Write the script in the pre-request section with this

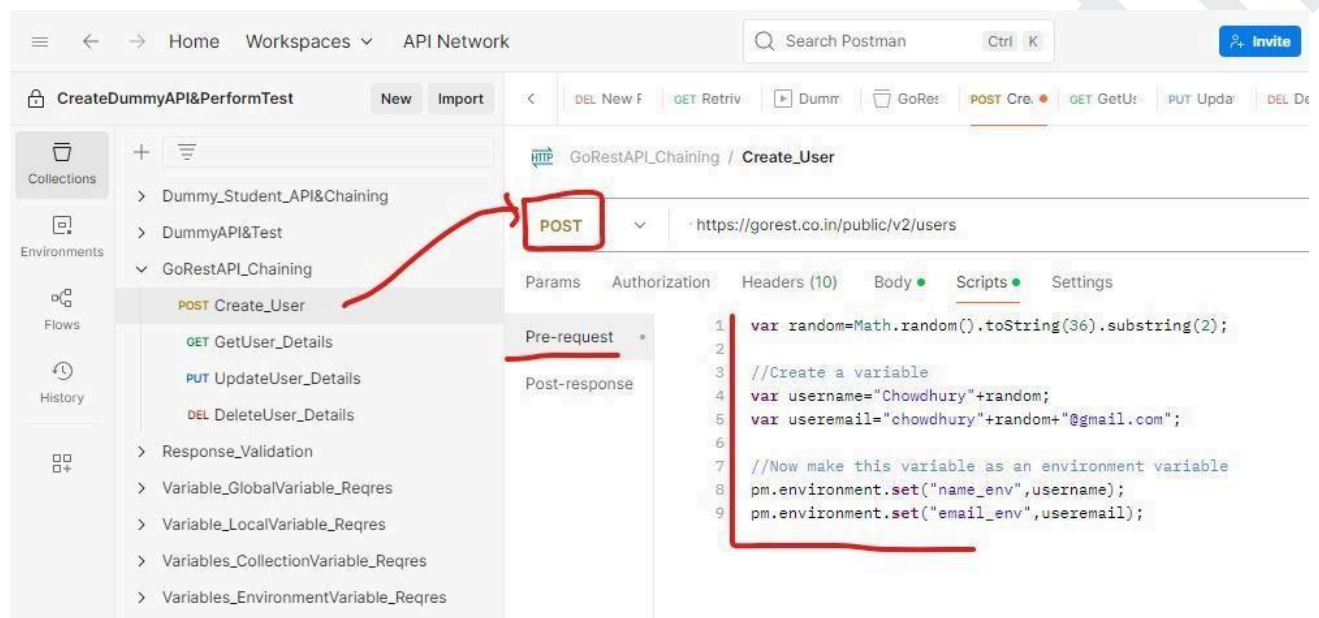
code:

```
var random=Math.random().toString(36).substring(2);
```

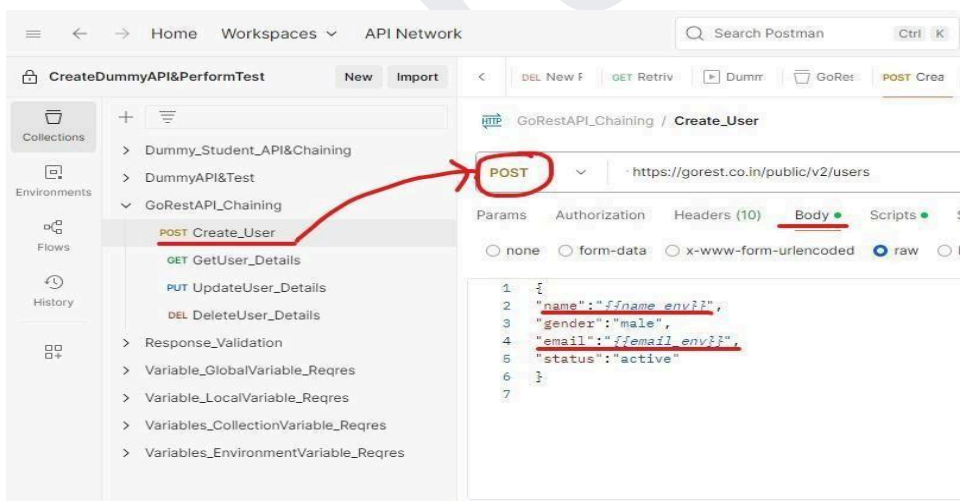
```
var username="Chowdhury"+random;
```

```
var useremail="chowdhury"+random+"@gmail.com";
```

Input:



Secondly, same script should write in the body of the post request

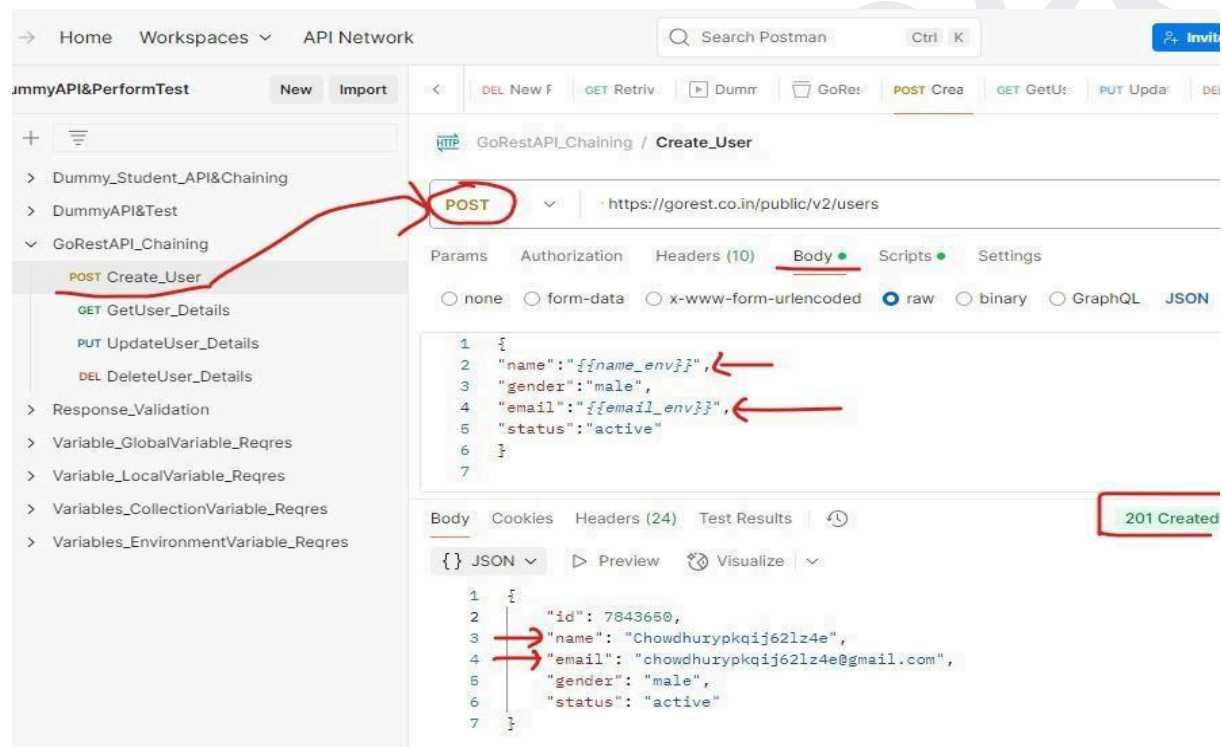


Same script is written in the body of the post request here is that code

Code:

```
{  
  "name": "{{name_env}}",  
  "gender": "male",  
  "email": "{{email_env}}",  
  "status": "active"  
}
```

Output:



After execute the post request then an ID will generate . This ID will need for other request .so, we need to extract this ID from the response .

So write this script in the Post- Request section.

Code:

```
var json.Data= JSON.parse(responseBody );  
pm.environment.set("userid_env",jsonData.id);
```

Parameterisation | Data Driven Testing:

How can we use data parameter/Data variable?

Process: We can specify the variables and value in the external files like CSV file.CSV or Json file.

BooksApi: Two things are going to describe I) Books (Do not need token)II) Order . For order request, (we need to use a token for authentication.)

URL: <http://simple-books-api.glitch.me/>

BooksAPI: Endpoint

Types of request:

Status: Check the books are available or not .

GET /Status

List of Books:

GET /books

Get a single Book:

GET /books/ :bookid

Task: Now we are going to perform Data Driven testing on this Particular API.

Step-1: First, we need to execute a Post request to generate a Token for Authentication through the given **link** and **body**.

Link/Url: <http://simple-books-api.glitch.me/api-clients/>

Body:

Generated Token:

Step-2 : Perform Get request using

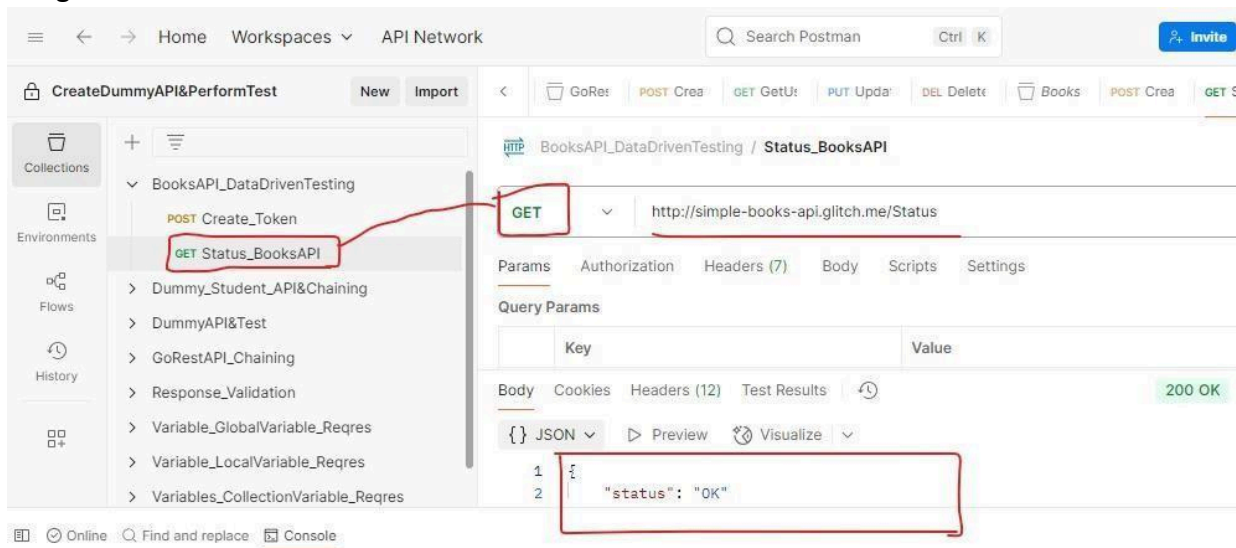
this Url :

<http://simple-books-api.glitch.me> And

End Point: /Status

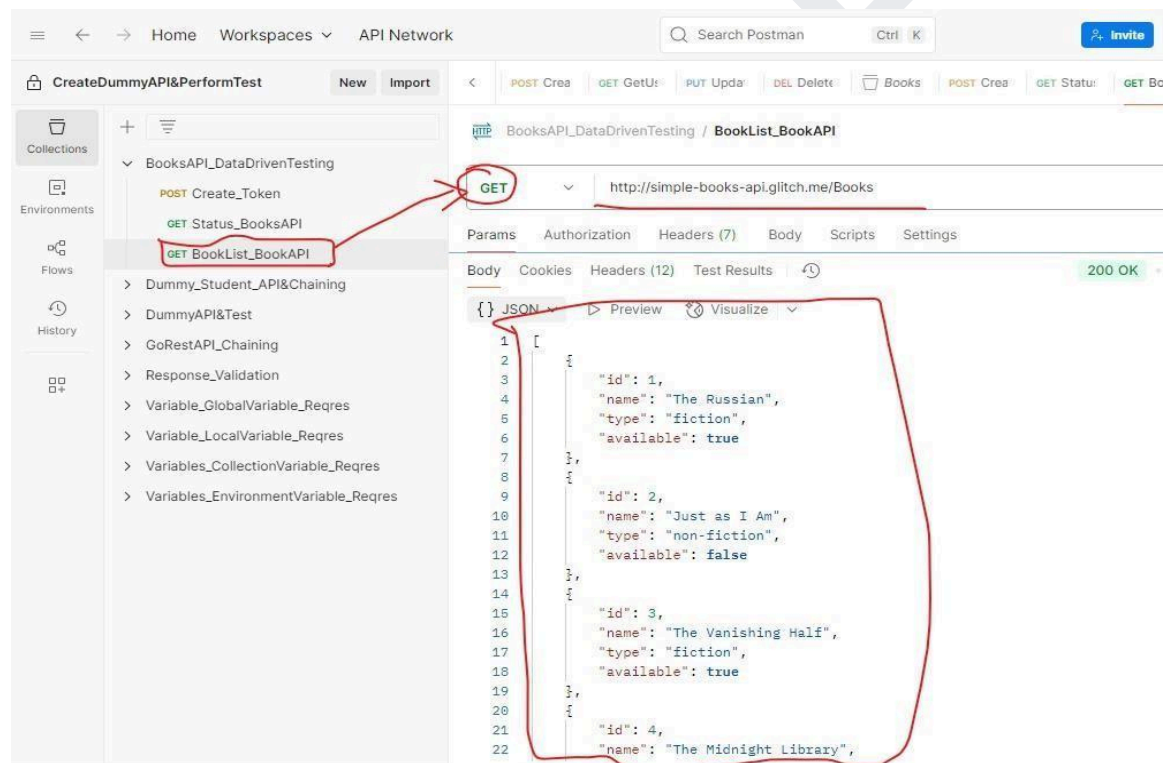
Get Request : <http://simple-books-api.glitch.me/Status>

Image:



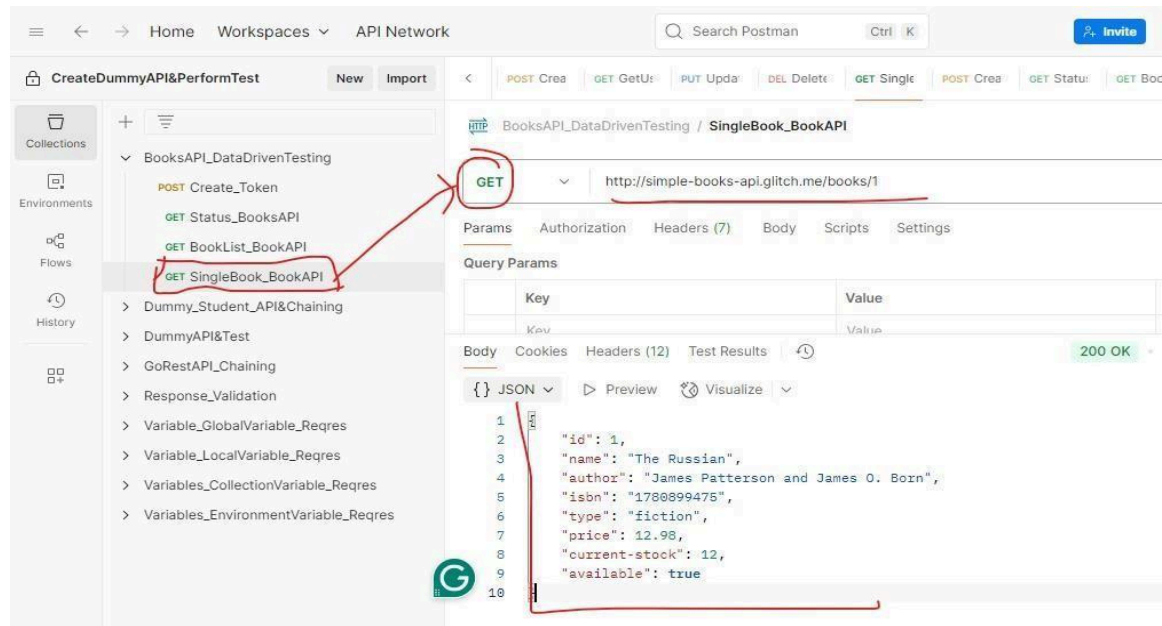
Step -3 : Perform get request using the given url for getting BookList from the BookAPI.

URL: <http://simple-books-api.glitch.me/books>



Step -4: Perform get request for getting single book/specific book from the BookAPI.

Url: <http://simple-books-api.glitch.me/books/1>



Swagger:

Petstore: It's a free API .

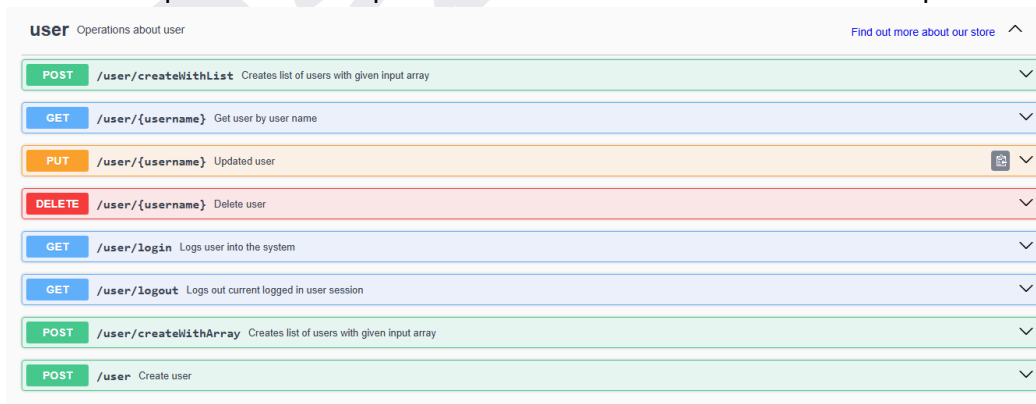
Petstore support two types of response json and xml

Here will see how to validate JSON and xml response. for this, we need to access a API called Petstore.

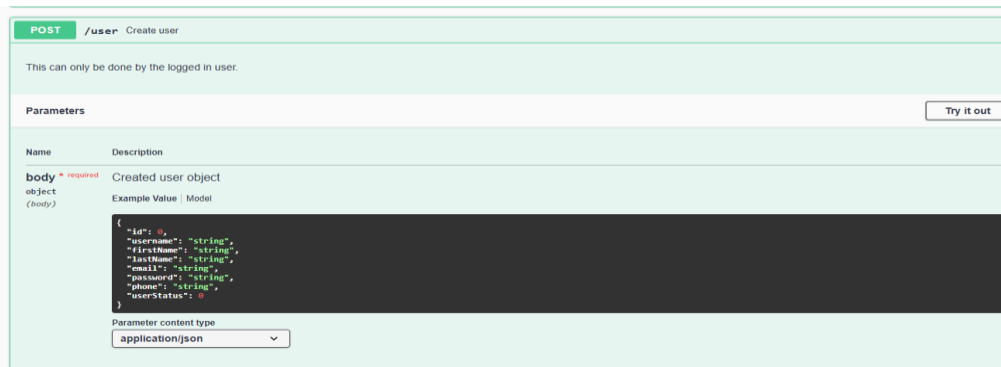
How to access the Petstore documentations?

Url of the Petsore documentation: <https://petstore.swagger.io/>

User model provides the responses in the Jsn format.Different API request in User model



Now click on **Post** request, which is in the given screenshot. Then this page will appear



POST /user Create user

This can only be done by the logged in user.

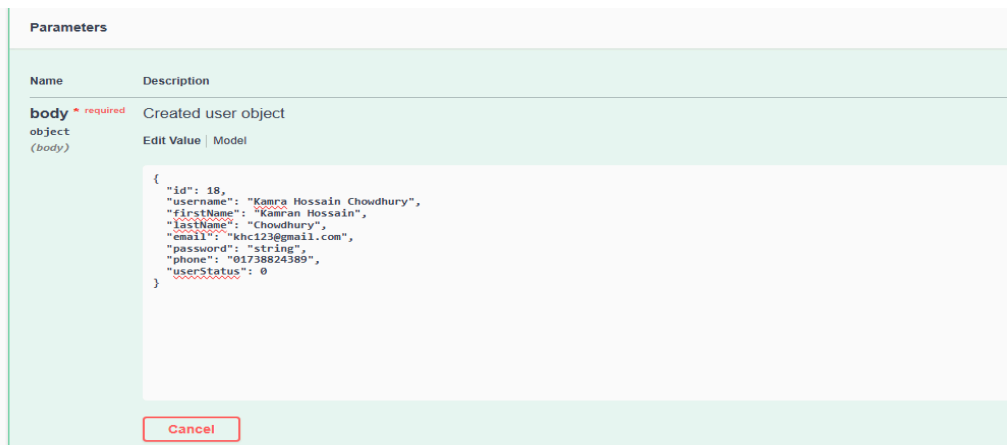
Parameters Try it out

Name	Description
body * required	Created user object
object (body)	Example Value Model

```
{
  "id": 0,
  "username": "string",
  "firstName": "string",
  "lastName": "string",
  "email": "string",
  "password": "string",
  "phone": "string",
  "userStatus": 0
}
```

Parameter content type:

Now click on the **Try it out** to insert the data in the body.



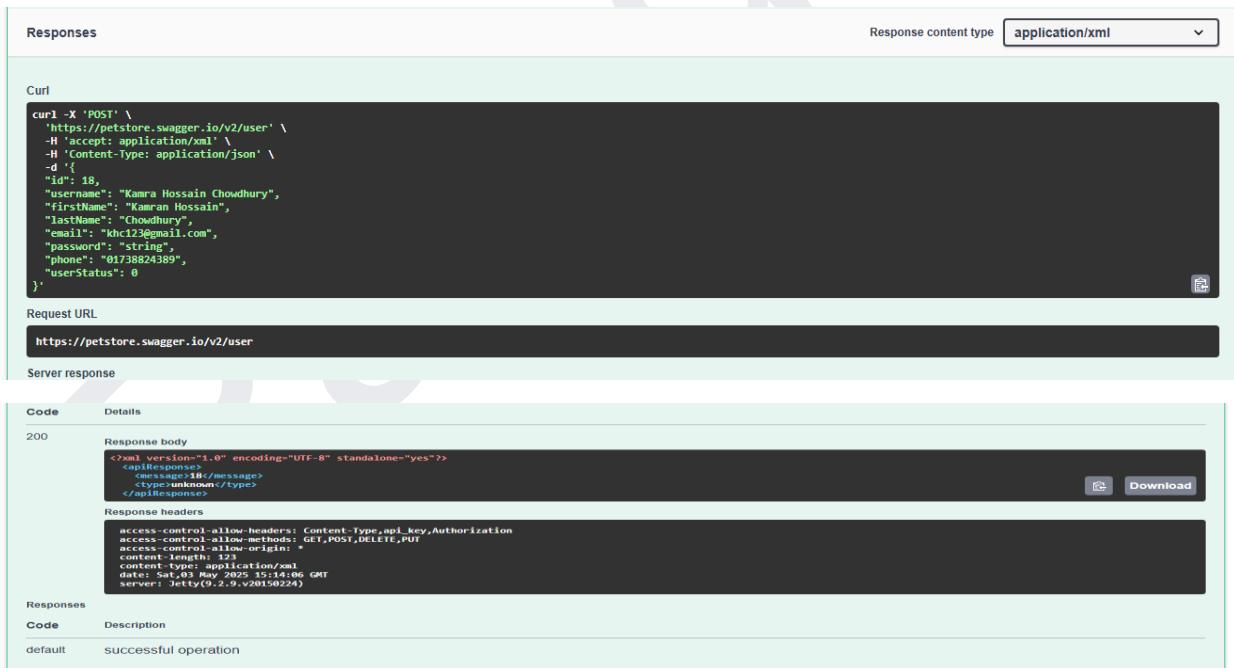
Parameters

Name	Description
body * required	Created user object
object (body)	Edit Value Model

```
{
  "id": 18,
  "username": "Kamra Hossain Chowdhury",
  "firstName": "Kamran Hossain",
  "lastName": "Chowdhury",
  "email": "khc123@gmail.com",
  "password": "string",
  "phone": "01738824389",
  "userStatus": 0
}
```

Cancel

After inserting the data the click on execute and get the output which is in below



Responses Response content type: application/xml

Curl

```
curl -X 'POST' \
  'https://petstore.swagger.io/v2/user' \
  -H 'accept: application/xml' \
  -H 'Content-type: application/json' \
  -d '{
    "id": 18,
    "username": "Kamra Hossain Chowdhury",
    "firstName": "Kamran Hossain",
    "lastName": "Chowdhury",
    "email": "khc123@gmail.com",
    "password": "string",
    "phone": "01738824389",
    "userStatus": 0
  }'
```

Request URL

<https://petstore.swagger.io/v2/user>

Server response

Code Details

200

Response body

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<apiResponse>
  <message>18/message</message>
  <type>unknown</type>
</apiResponse>
```

Response headers

```
access-control-allow-headers: Content-type,api_key,Authorization
access-control-allow-methods: GET,POST,DELETE,PUI
access-control-allow-origin: *
content-length: 123
content-type: application/xml
date: Sat, 03 May 2025 15:14:06 GMT
server: Jetty(9.2.9.v20150224)
```

Responses

Code	Description
default	successful operation

Now I will import this API in the PostMan softwarez:

First copy the curl:

```
curl -X 'POST' \  
  'https://petstore.swagger.io/v2/user' \  
  -H 'accept: application/xml' \  
  -H 'Content-Type: application/json' \  
  -d '{  
    "id": 18,  
    "username": "Kamra Hossain Chowdhury",  
    "firstName": "Kamran Hossain",  
    "lastName": "Chowdhury",  
    "email": "khc123@gmail.com",  
    "password": "string",  
    "phone": "01738824389",  
    "userStatus": 0  
  }'
```

Then Create a collection in the postman and import the curl