

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 :

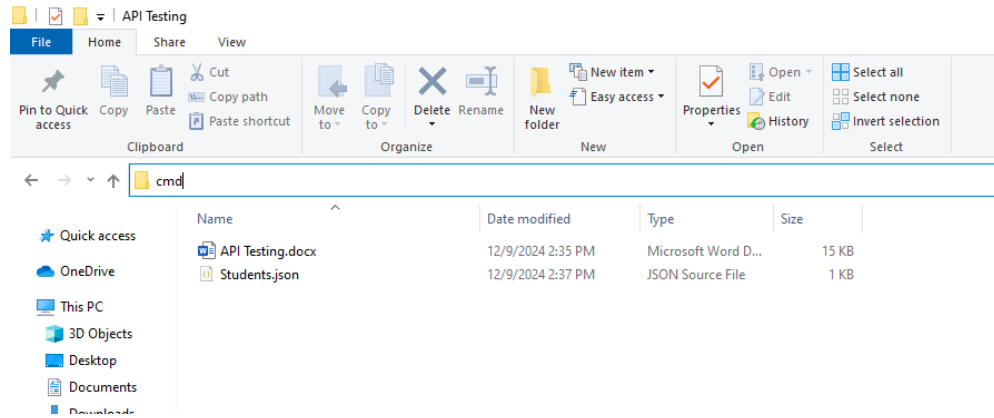
```
{
  "students":[
    {
      "id": 1,
      "name": "John Doe",
      "age": 18,
      "grade": "12th",
      "subjects": [
        "Math",
        "Physics",
        "English"
      ]
    },
    {
      "id": 2,
      "name": "Jane Smith",
      "age": 17,
      "grade": "11th",
      "subjects":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

How to create own API ?

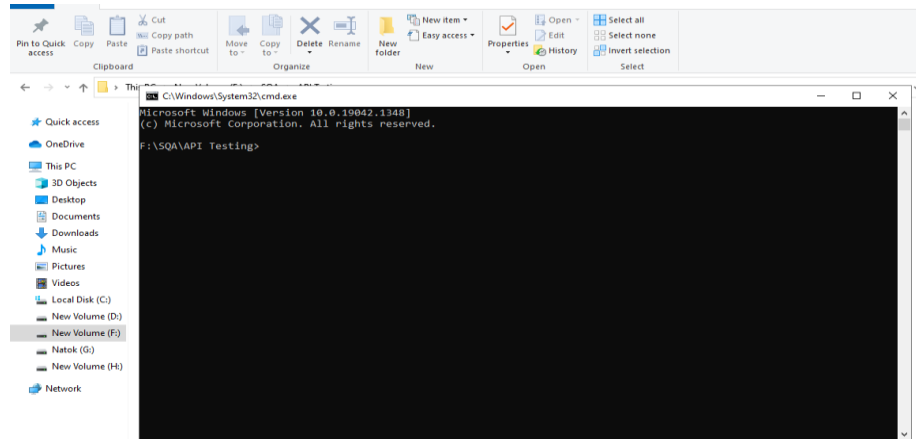
Task no-1

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

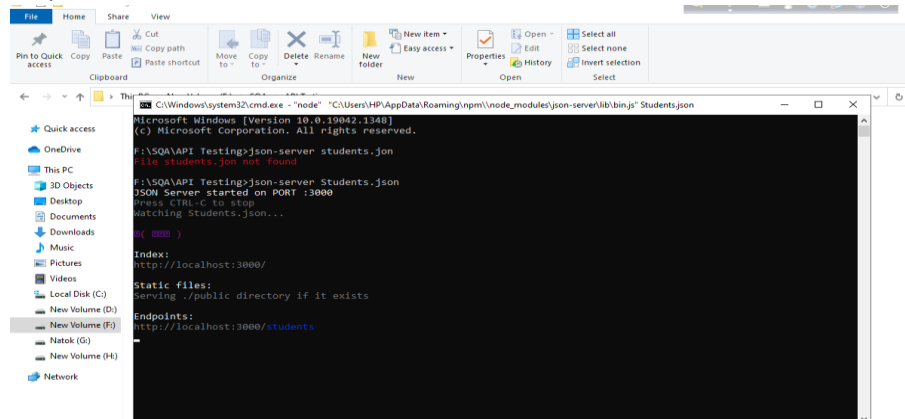
Step-1



Step-2



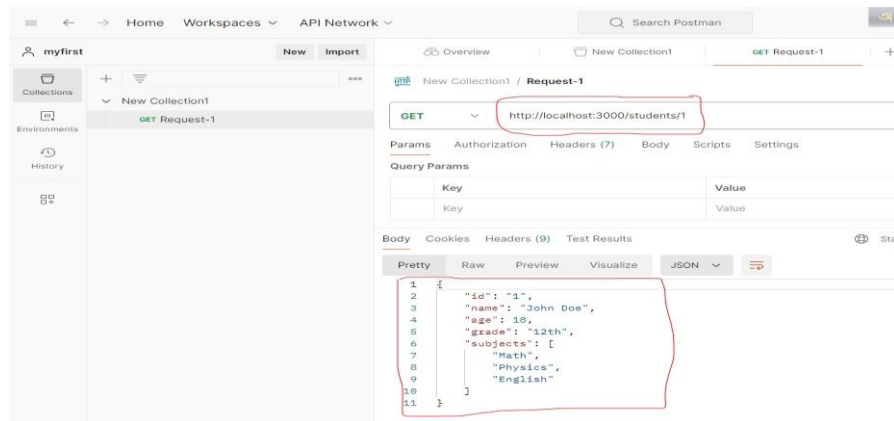
Step-3



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

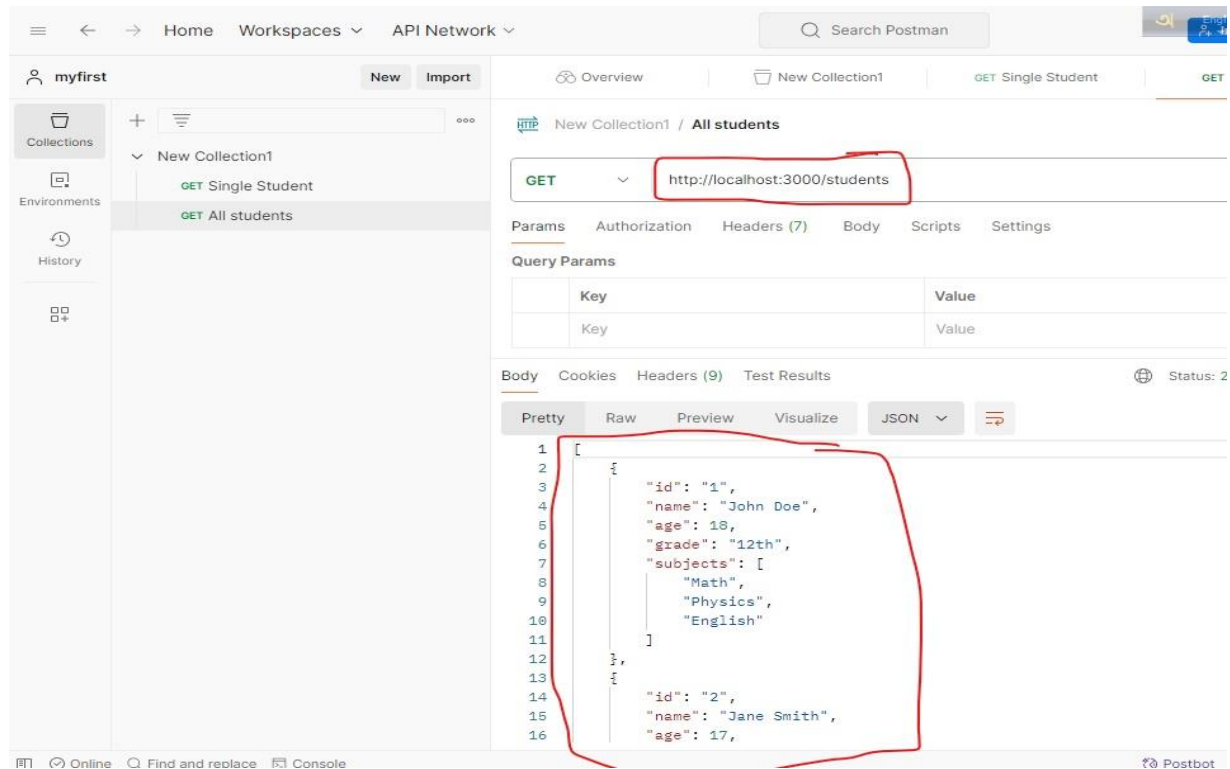
How to get single student data from the above URL?

Task No- 2



How to get all students data from the above URL?

Task No- 3



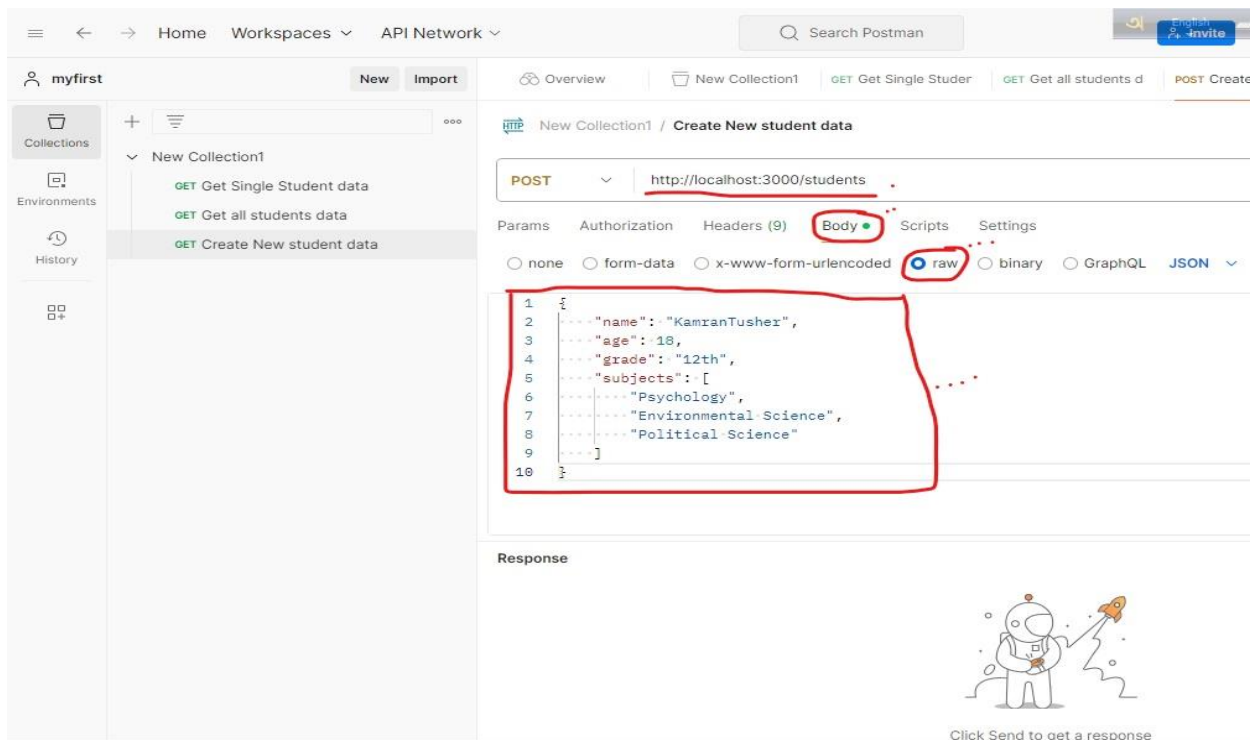
How to create new student data?

Create = Post

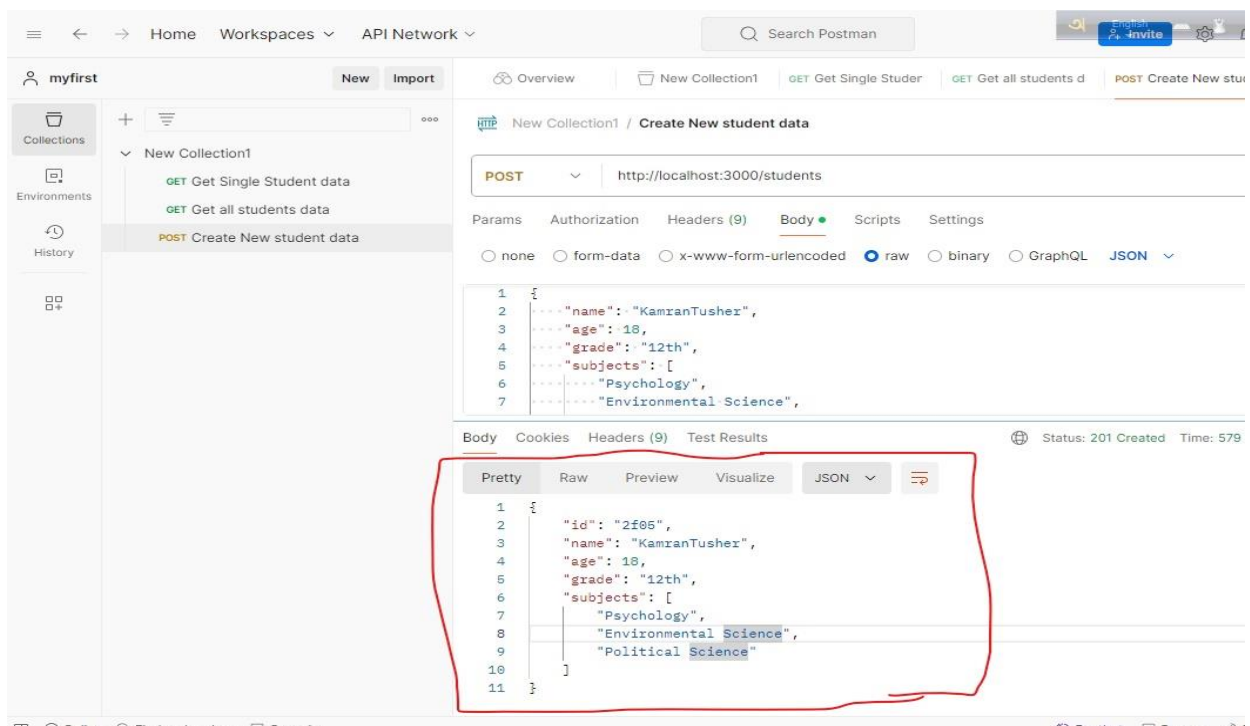
Task No- 4

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

Step-1 (Request Payload)

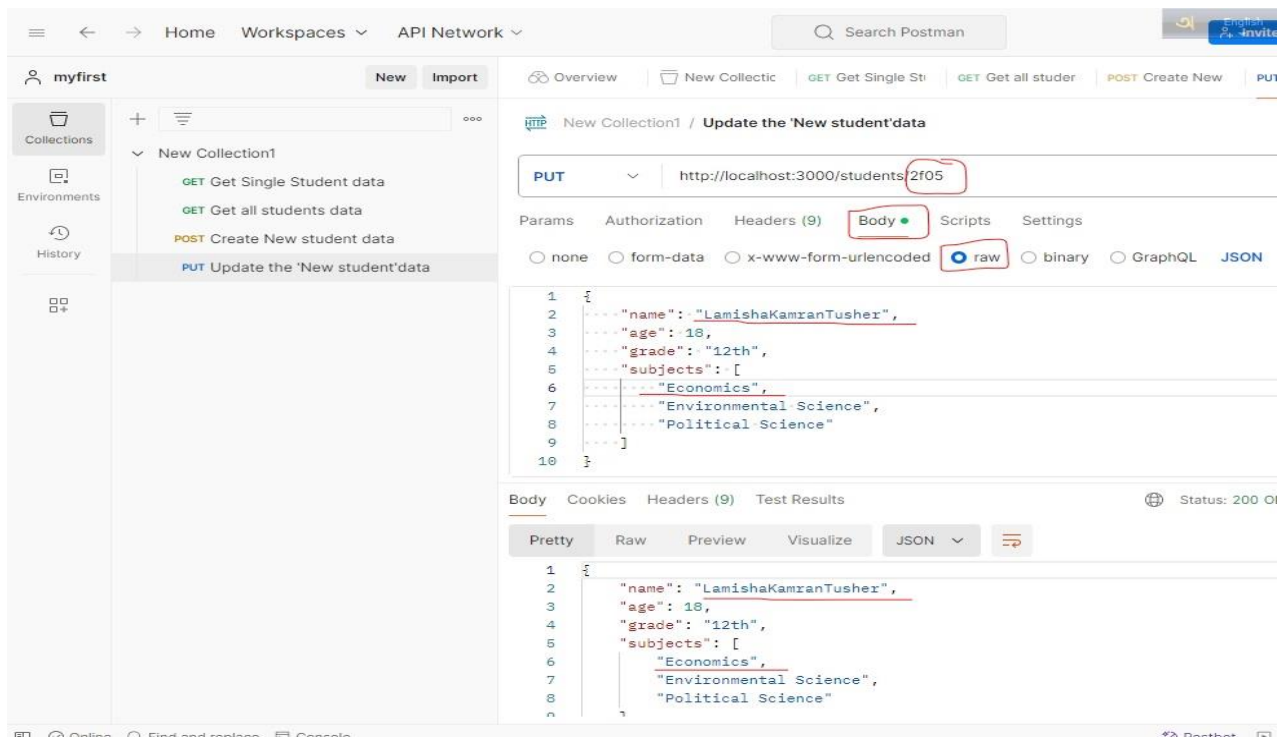


Step-2 (Response Payload)



How to update the earlier "New student data" data? Update = Put **Task No- 5**

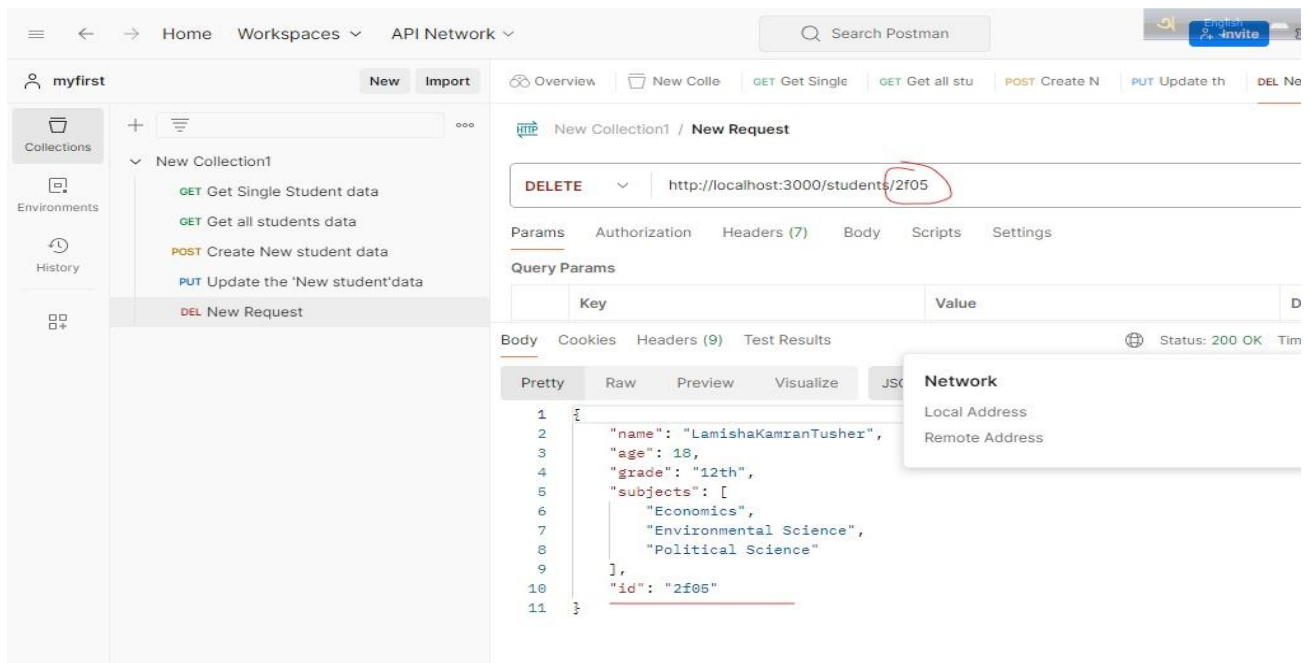
Ans: Make sure to set the "Id Number" beside the link_ (<http://localhost:3000/students/2f05>)



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

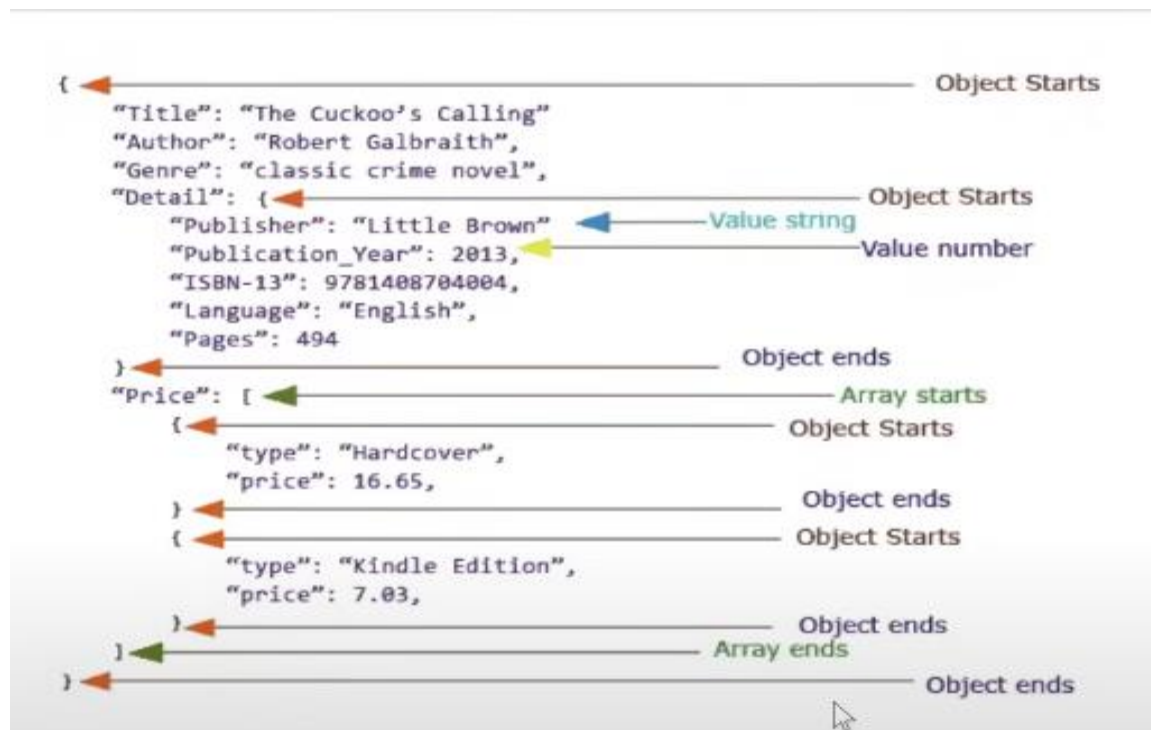
```
{  
  "student": [  
    {  
      "id": "01",  
      "name": "Tom",  
      "lastname": "Price"  
    },  
    {  
      "id": "02",  
      "name": "Nick",  
      "lastname": "Thameson"  
    }  
  ]  
}
```

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 displays the JSON Path Finder tool. On the left, a JSON array is shown with line numbers 51 to 78. The array contains four student objects. The second object (index 1) is highlighted in blue. The right pane shows the path 'x.Student[6].First_NAME' entered in the 'Path' field. Below the path, the extracted data is displayed: 'Email: sad@gmail.com', 'Location: Barishal', 'Grade: 3.42', and 'First_NAME: Anisul'. The 'First_NAME' field is highlighted in blue.

Path	Value
x.Student[6].First_NAME	Anisul

JSON path Validation

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

The screenshot shows a JSONPath validation tool interface. On the left, the JSONPath expression `Student[6].First_NAME` is entered and highlighted with an orange circle. Below it, there are toggle switches for "Output paths" and a link for "Expand JSONPath expressions". The main area is split into two panels: "Inputs" on the left and "Evaluation Results" on the right. The "Inputs" panel displays a JSON array of three student objects. The "Evaluation Results" panel shows the result of the path validation, which is `"Anisul"`, also highlighted with an orange circle. An orange arrow points from the path expression to the result.

```
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 }
```

1
2 "Anisul"
3

Response Validation

Task No--8

1. Status Code:

Step-1

The screenshot shows the Postman interface. A GET request is configured with the URL `http://localhost:3000/students/1`. The request is executed, and the response is shown in the "Test Results" tab. The status code is `200 OK`, which is highlighted with a red circle. The response body is displayed in JSON format, showing a student object with fields like `id`, `name`, `age`, `grade`, and `subjects`.

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

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (9) Test Results

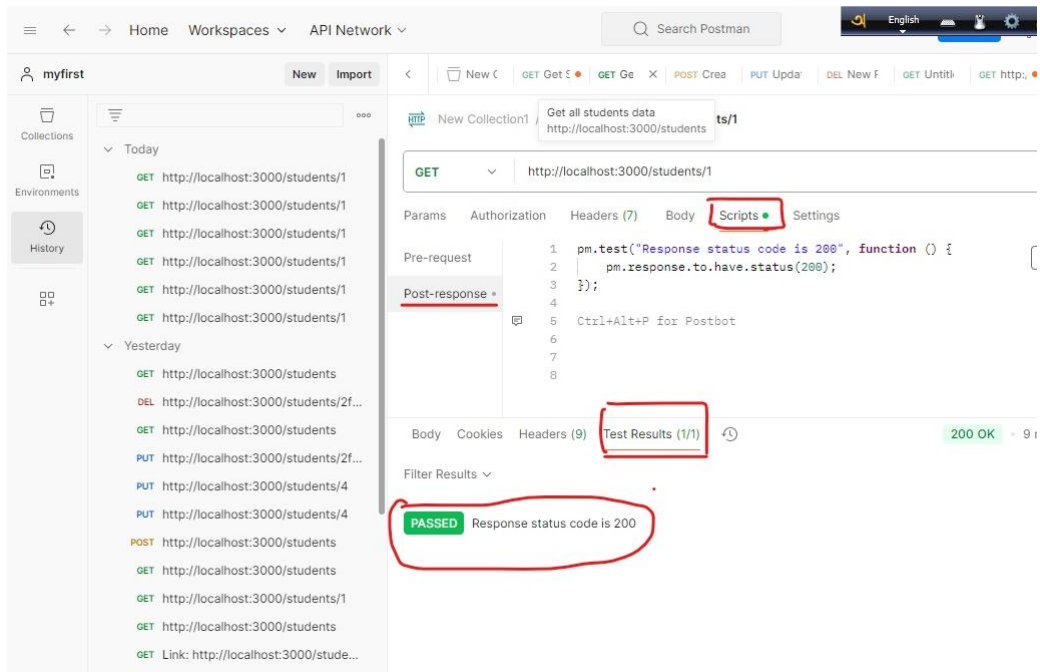
200 OK Status: 200 OK Time: 214 ms Size: 454 B

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

Step-2

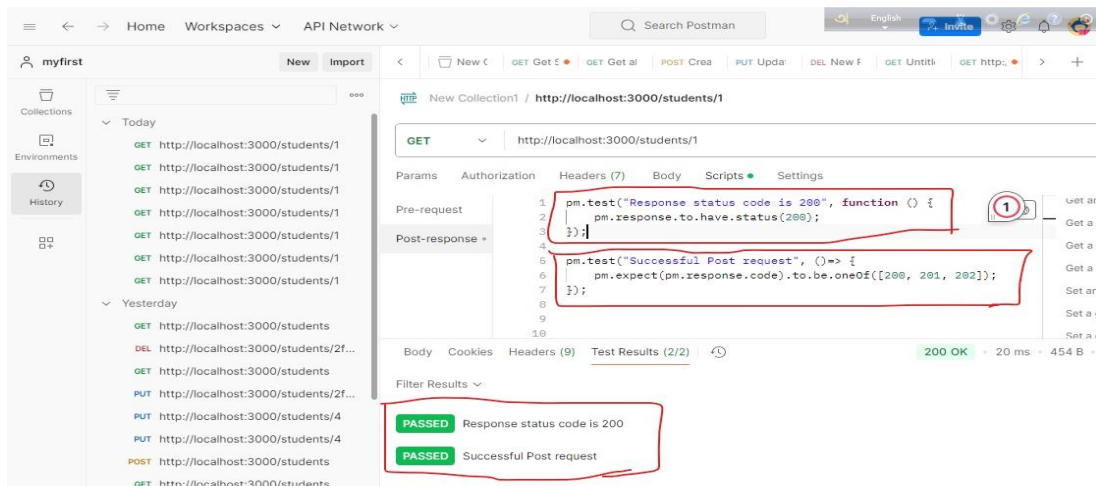
For Single Status code

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



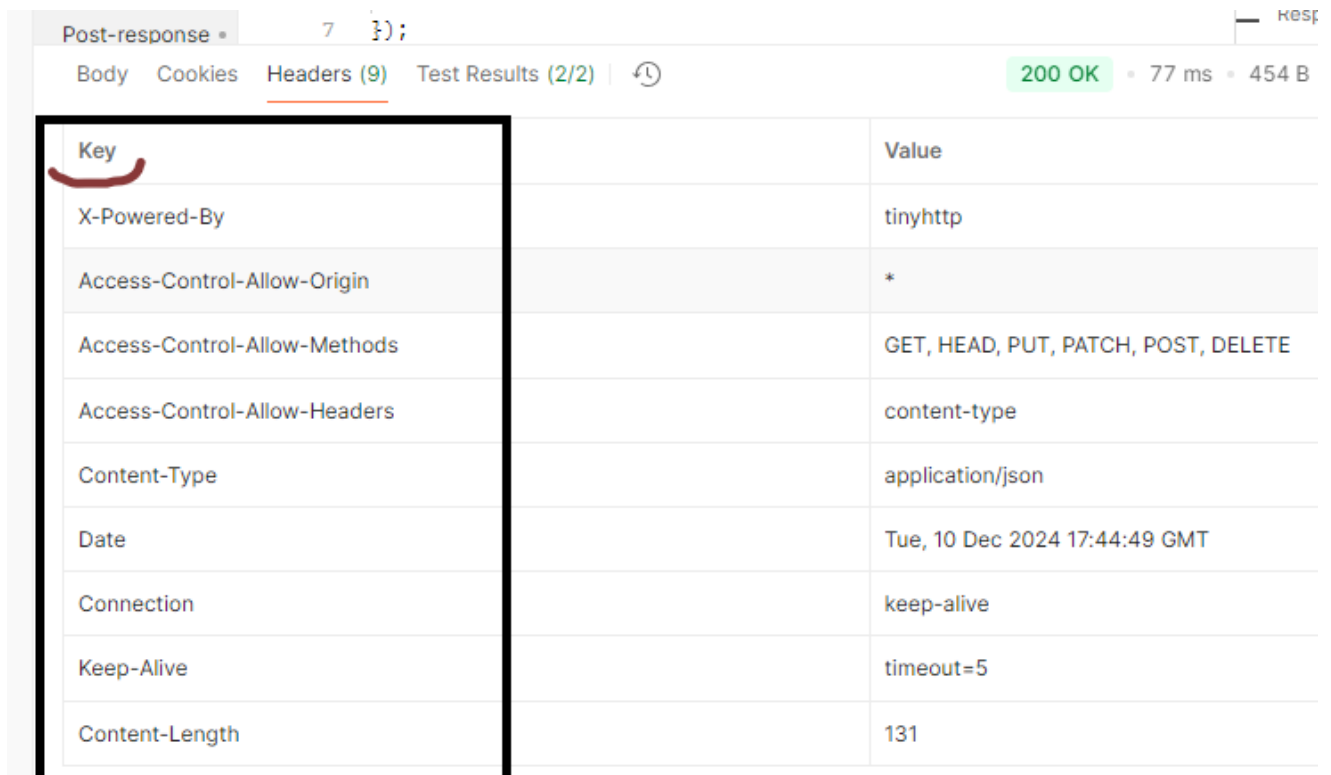
For Multiple Status code

```
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



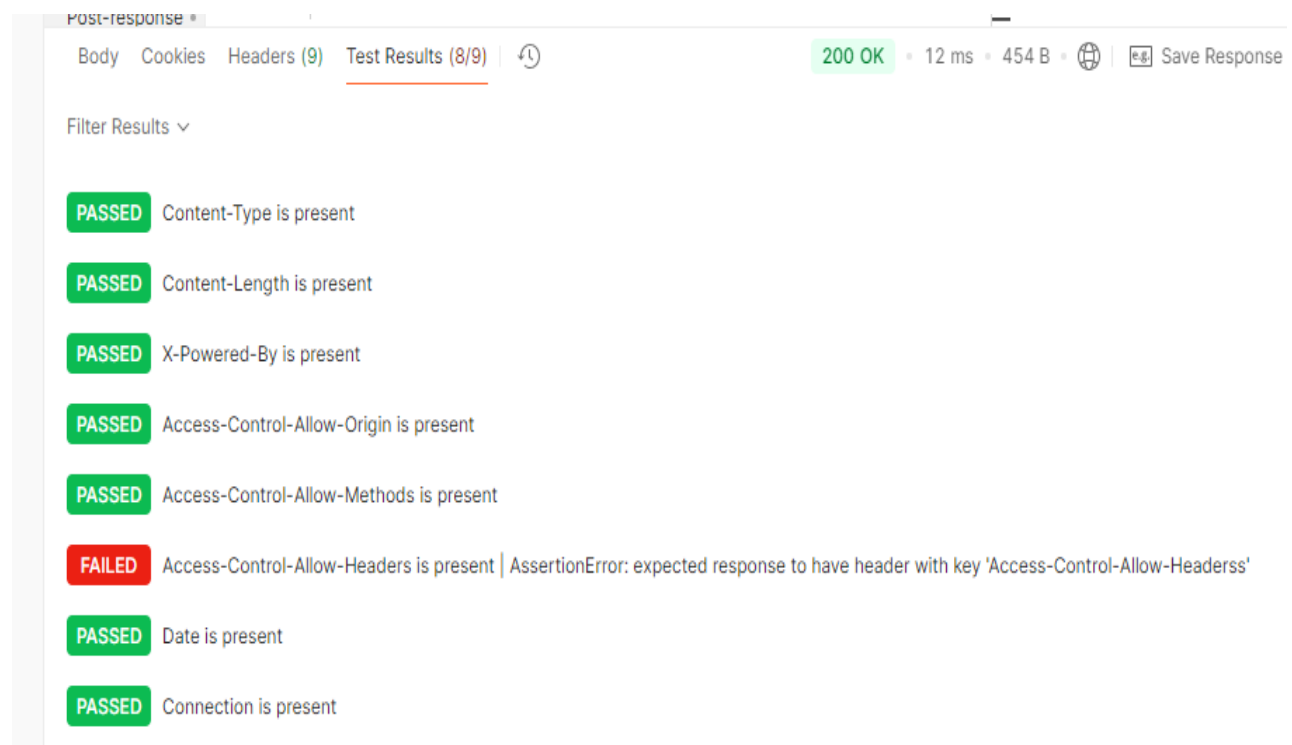
Key	Value
X-Powered-By	tinyhttp
Access-Control-Allow-Origin	*
Access-Control-Allow-Methods	GET, HEAD, PUT, PATCH, POST, DELETE
Access-Control-Allow-Headers	content-type
Content-Type	application/json
Date	Tue, 10 Dec 2024 17:44:49 GMT
Connection	keep-alive
Keep-Alive	timeout=5
Content-Length	131

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 response is `200 OK` with a status of `8 ms` and a size of `454 B`. The response headers are visible, including `Content-Type: application/json` and `X-Powered-By: application/json`. The test results show that both assertions passed.

```
1 pm.test("Content-Type is application/json", () => {  
2     pm.expect(pm.response.headers.get('Content-Type')).to.eql(  
3         'application/json'  
4     );  
5  
6     pm.test("X-Powered-By is application/json", () => {  
7         pm.expect(pm.response.headers.get('X-Powered-By')).to.eql(  
8             'application/json'  
9         );  
10    });  
11 }
```

Test Results (2/2):

- 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 failure message: "FAILED Response time is less than 100ms | AssertionError: expected". A detailed response time breakdown is shown, totaling 105.80 ms.

The bottom screenshot shows the same GET request but 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 success message: "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

Collections + New Collection1

- GET Get Single Student data
- GET Get all students data
- POST Create New student data
- PUT Update the 'New student' data
- DEL Delete Request
- GET http://localhost:3000/students/1

Environments History

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

Params Authorization Headers (7) Body Scripts Settings

```
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

Collections + New Collection1

- GET Get Single Student data
- GET Get all students data
- POST Create New student data
- PUT Update the 'New student' data
- DEL Delete Request
- GET http://localhost:3000/students/1

Environments History

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

Params Authorization Headers (7) Body Scripts Settings

```
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

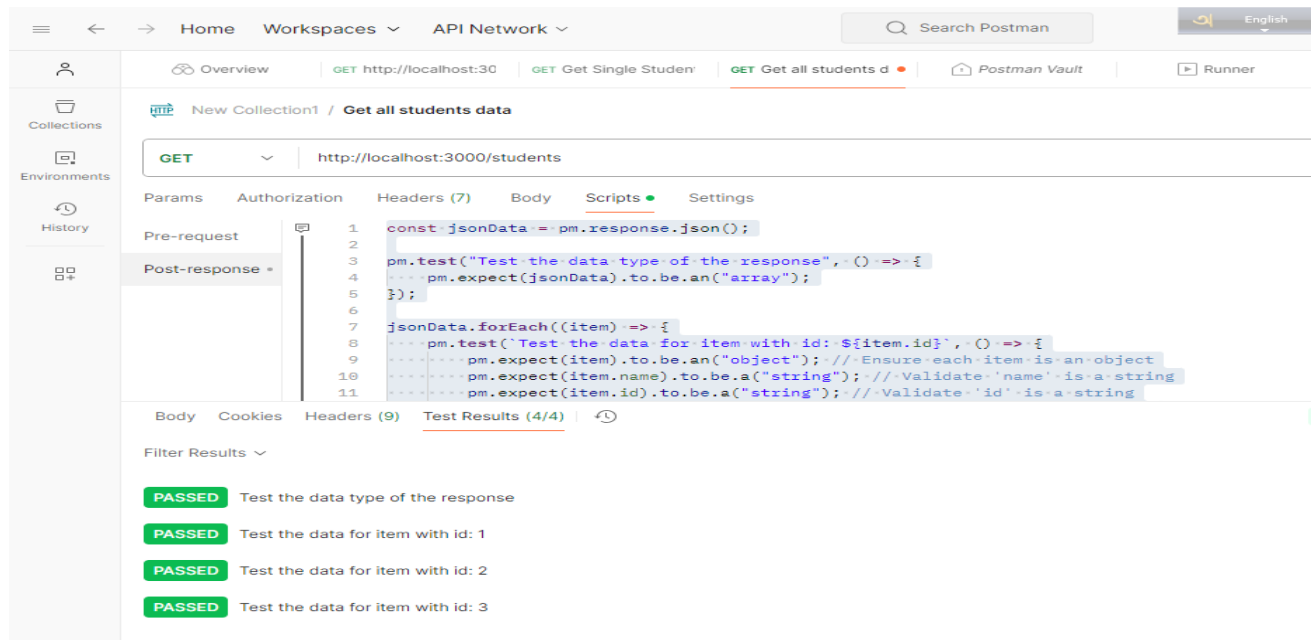
    pm.expect(item.id).to.be.a("string"); // Validate 'id' is a string

    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

  });

});
```



The screenshot displays the Postman application interface. The top navigation bar includes 'Home', 'Workspaces', and 'API Network'. The main workspace shows a collection named 'New Collection1' with a request 'Get all students data'. The request is a GET method to 'http://localhost:3000/students'. The 'Scripts' tab is active, showing the following JavaScript code:

```
1 const jsonData = pm.response.json();
2
3 pm.test("Test the data type of the response", () => {
4   pm.expect(jsonData).to.be.an("array");
5 });
6
7 jsonData.forEach((item) => {
8   pm.test(`Test the data for item with id: ${item.id}`, () => {
9     pm.expect(item).to.be.an("object"); // Ensure each item is an object
10    pm.expect(item.name).to.be.a("string"); // Validate 'name' is a string
11    pm.expect(item.id).to.be.a("string"); // Validate 'id' is a string
```

The bottom section shows the 'Test Results (4/4)' tab, indicating that all tests passed:

- PASSED** Test the data type of the response
- PASSED** Test the data for item with id: 1
- PASSED** Test the data for item with id: 2
- PASSED** Test the data for item with id: 3

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:

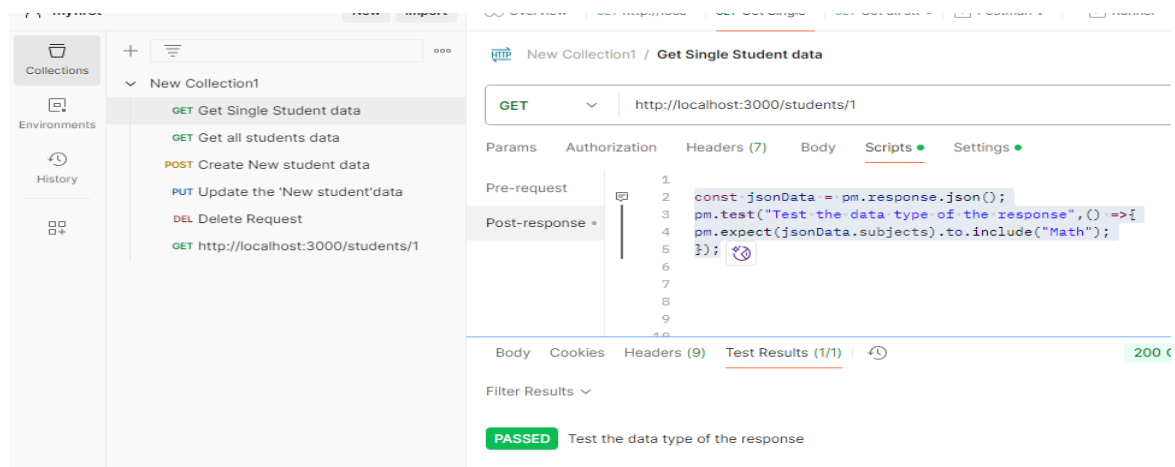
The screenshot shows the Postman interface. On the left, a collection named 'myfirst' contains a 'New Collection1' with several requests. The selected request is 'GET Get Single Student data' with the URL 'http://localhost:3000/students/1'. The main panel shows the 'Scripts' tab with the following code:

```
1  
2 const jsonData = pm.response.json();  
3 pm.test("Test the data type of the response", () =>{  
4   pm.expect(jsonData.subjects).to.include("Math");  
5 });  
6  
7 |ctrl+Alt+P for Postbot  
8  
9  
10  
11
```

The 'Body' tab shows the 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 status bar at the bottom right indicates '200 OK'.

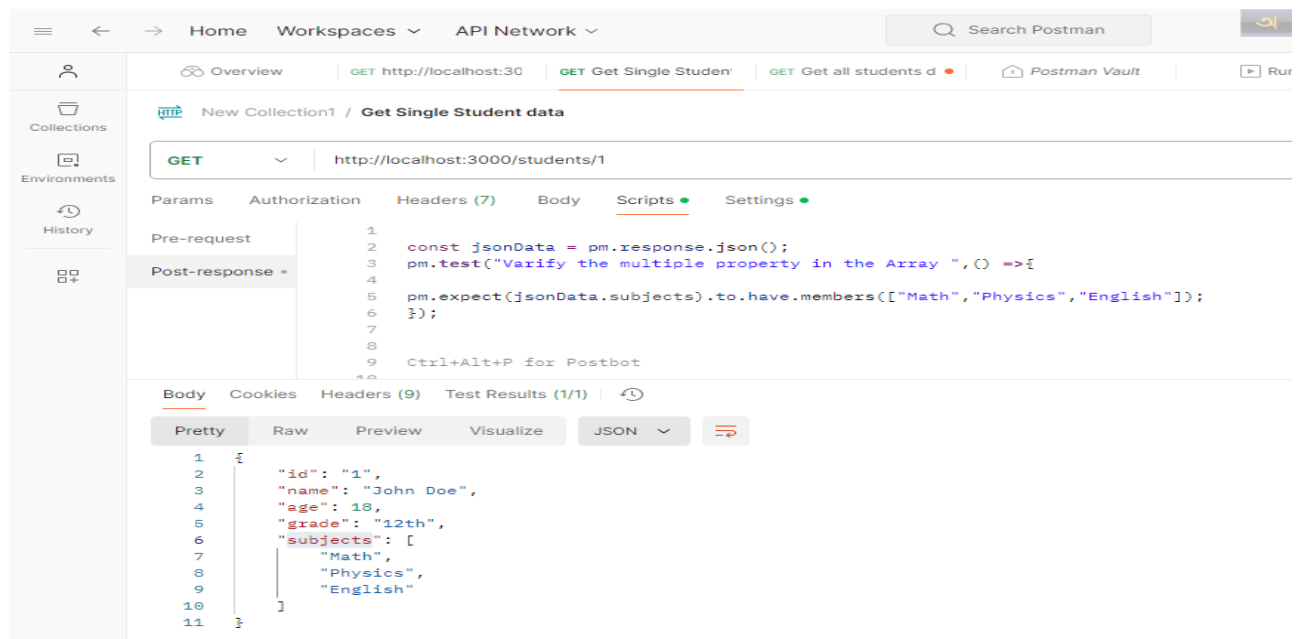


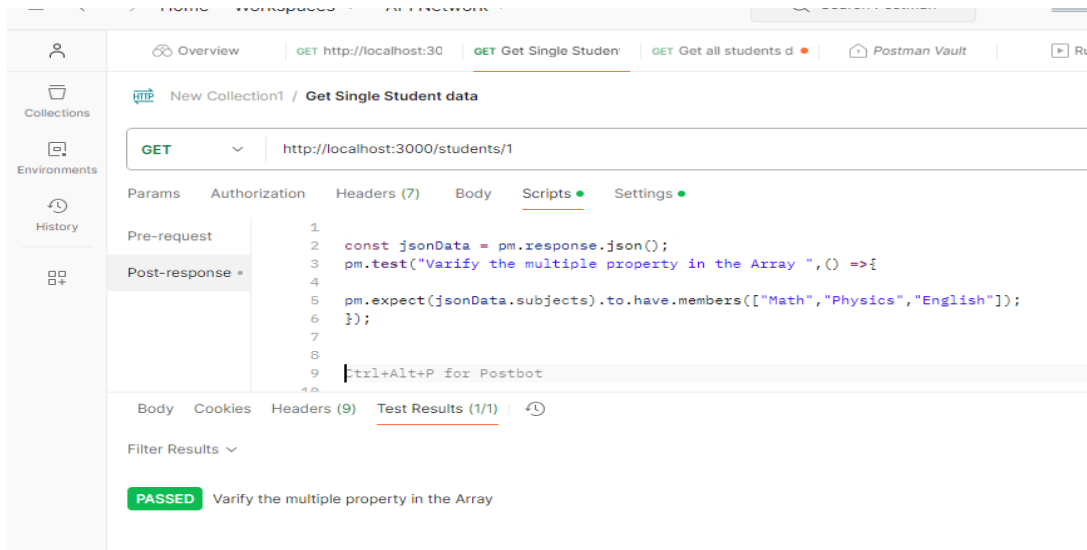
-(Validate a Multiple property in the Array from the response body)

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");  
});
```

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

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

Params | Authorization | Headers (7) | Body | Scripts | Settings

```
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 });
```

200 OK

Body | Cookies | Headers (9) | Test Results (1/1)

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  ]
```

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

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

Params | Authorization | Headers (7) | Body | Scripts | Settings

```
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 });
12
```

Body | Cookies | Headers (9) | Test Results (1/1)

Filter Results

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?

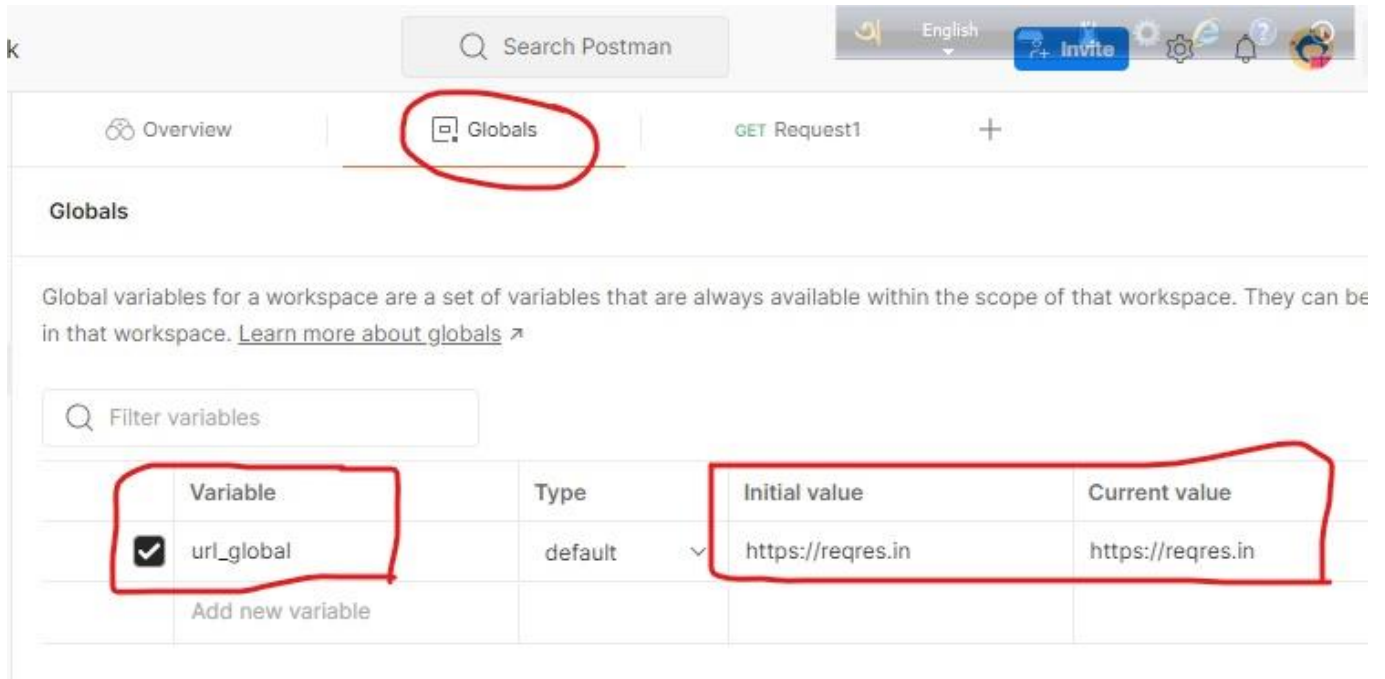
Ans: workspace--→Collection---→Request/Folder.

A global variable is accessible throughout the every workspace.

Step-1

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

Step-2

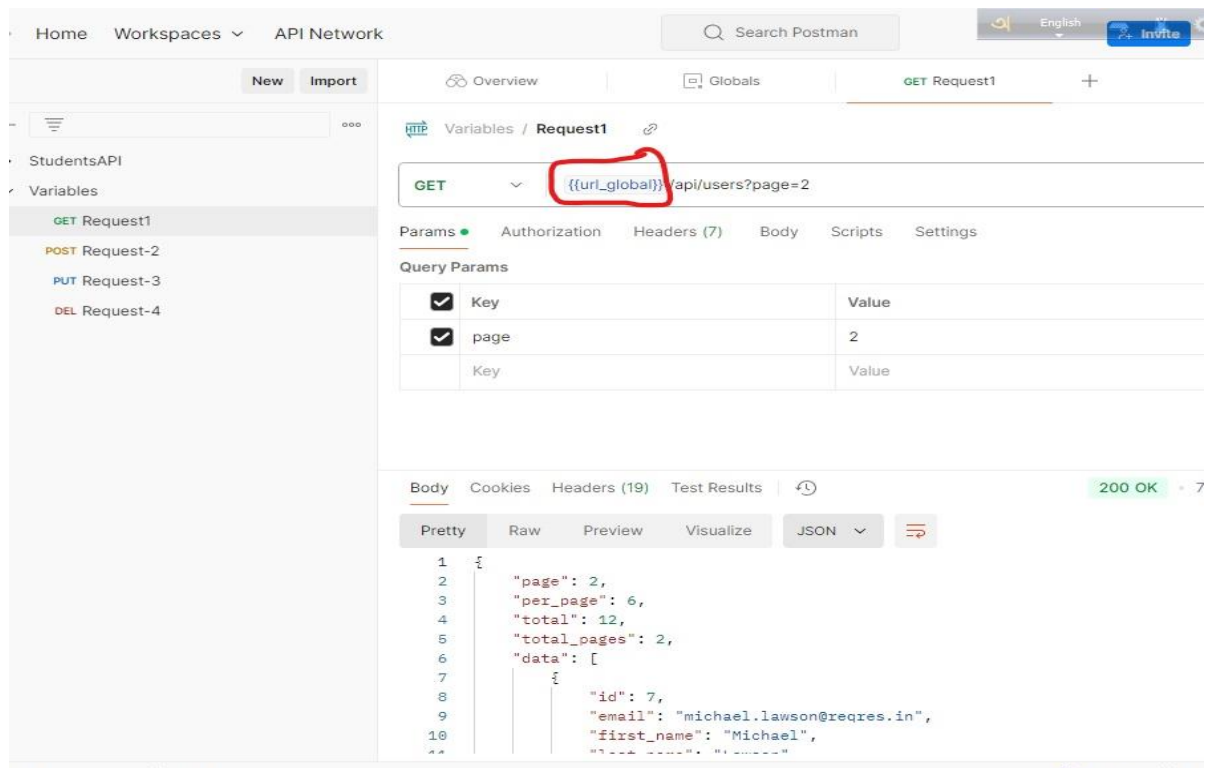


Global variables for a workspace are a set of variables that are always available within the scope of that workspace. They can be used in that workspace. [Learn more about globals](#)

Filter variables

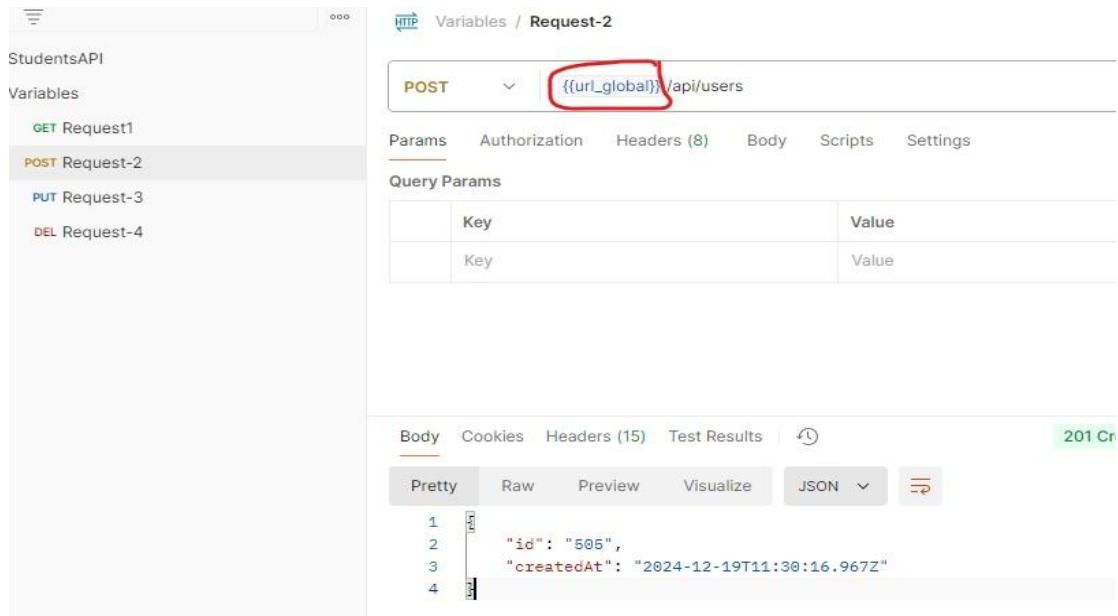
Variable	Type	Initial value	Current value
<input checked="" type="checkbox"/> url_global	default	https://reqres.in	https://reqres.in
Add new variable			

Step-3 ----(Get Request)



Step-5 ----(Post Request)

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



Step-6 (Put Request)

→ Home Workspaces ▾ API Network

Search Postman

English

New Import

Overview Globals GET Request1 POST Request- PUT Request- DEL Rec

+ ...

> StudentsAPI

▾ Variables

- GET Request1
- POST Request-2
- PUT Request-3
- DEL Request-4

Variables / Request-3

PUT ▾ {{url_global}}/api/users/2

Params Authorization Headers (8) Body Scripts Settings

Query Params

	Key	Value
	Key	Value

Body Cookies Headers (16) Test Results ↺

Pretty Raw Preview Visualize JSON ▾

```
1 {  
2   "updatedAt": "2024-12-19T11:36:48.593Z"  
3 }
```

Step-7 ----(Delete Request)

work

Search Postman

English Invite

Upgrade

ort Overview Globals GET Request1 POST Request- PUT Request- DEL Reque x Runner + ▾ New

... Variables / Request-4

Request-4
{{url_global}}/api/users/2

DELETE ▾ {{url_global}}/api/users/2

Params Authorization Headers (7) Body Scripts Settings

Query Params

	Key	Value	Description
	Key	Value	Description

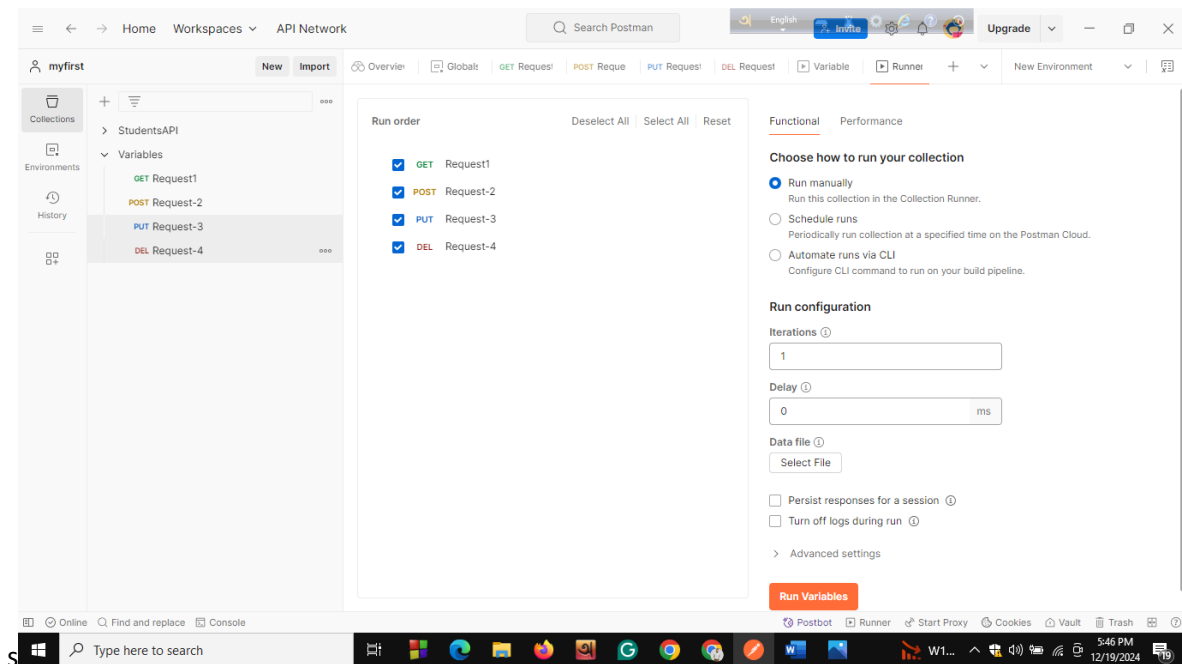
Body Cookies Headers (14) Test Results ↺

204 No Content • 612 ms • 1 KB • 🌐 📄 Si

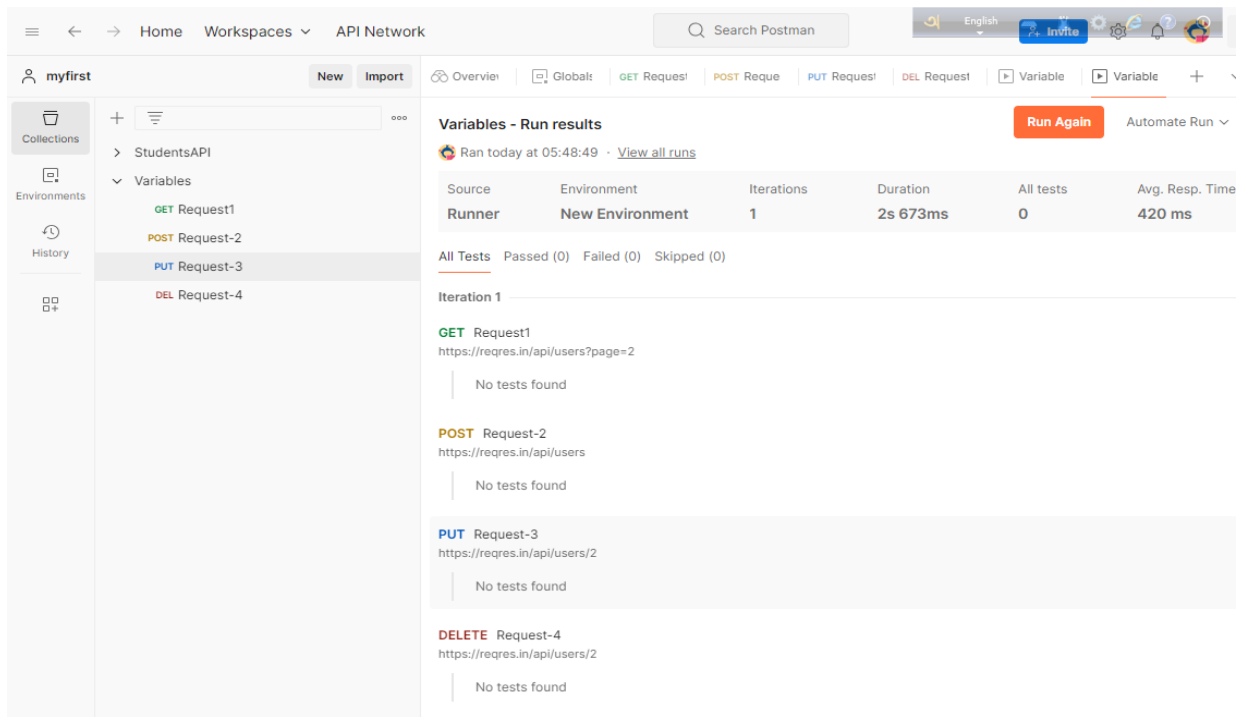
Pretty Raw Preview Visualize Text ▾

```
1
```

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](#)

Filter variables

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

Step-2

GET `{{url_collection}}/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

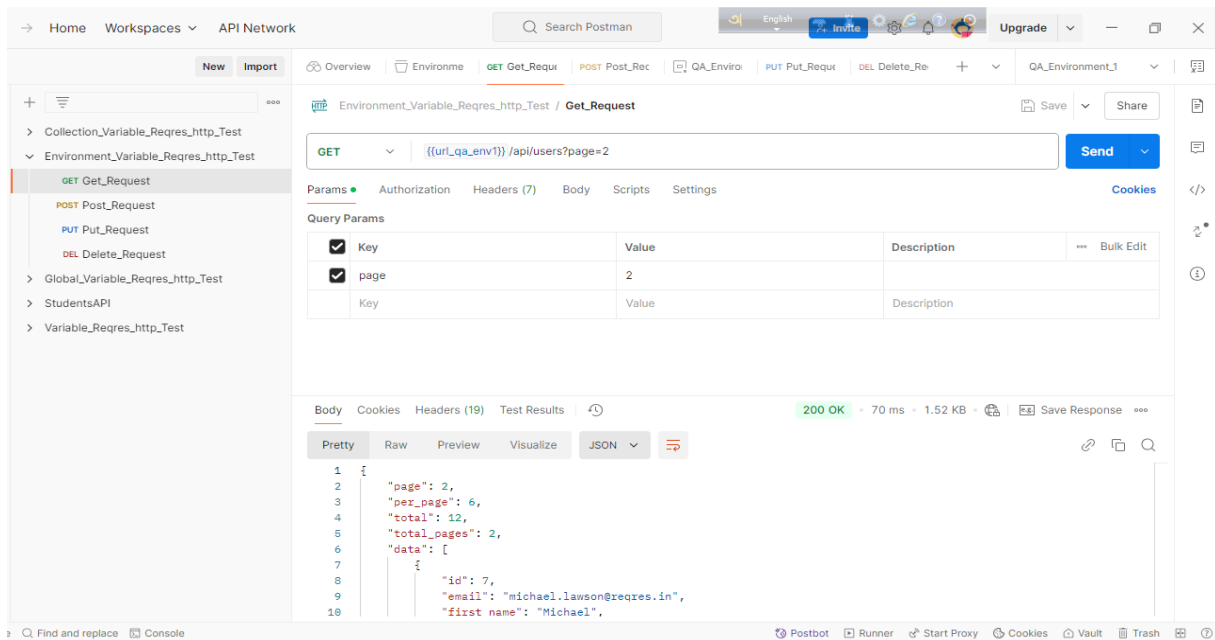
200 OK • 185

Pretty Raw Preview Visualize JSON

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

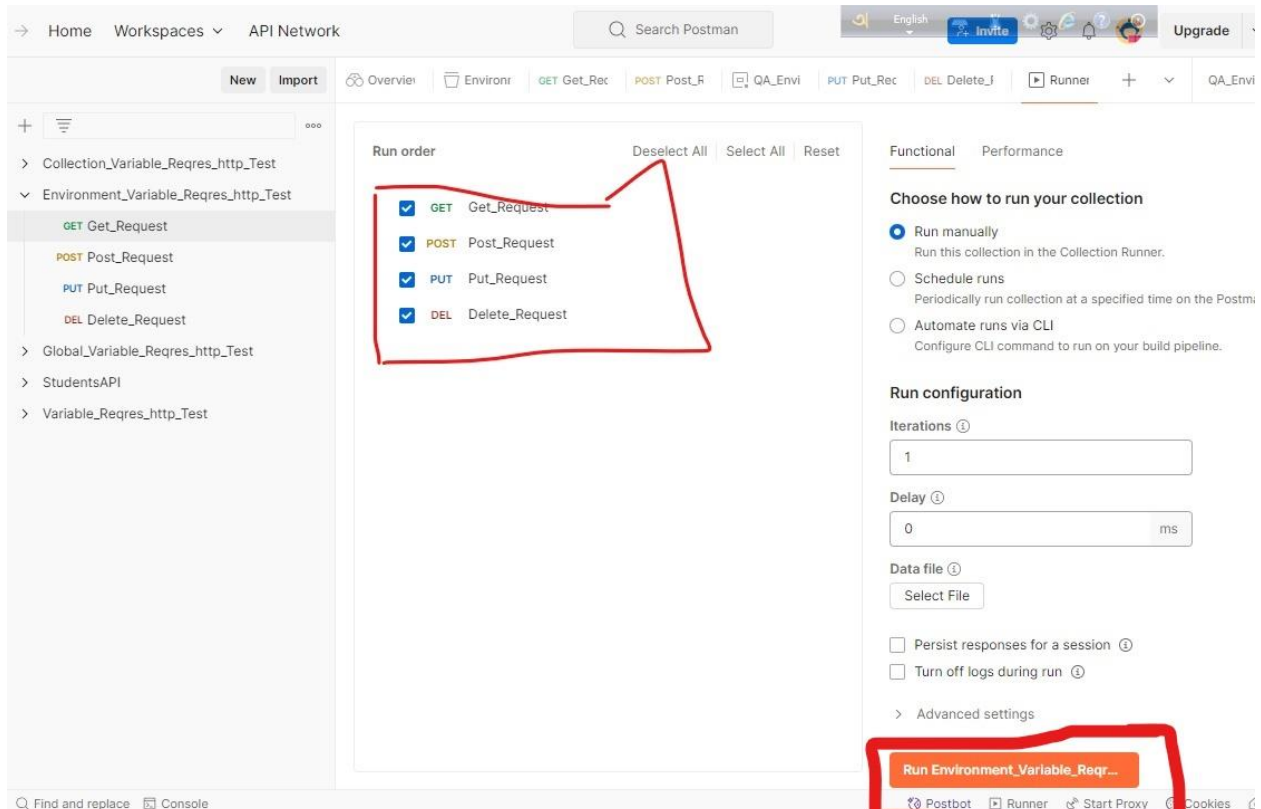
GET `https://reqres.in/api/users?page=2`

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

The screenshot shows the Postman interface with the 'Run results' tab selected for the 'Environment_Variable_Regres...' collection. The interface includes a sidebar with a tree view of collections and environments, a top navigation bar with 'Home', 'Workspaces', and 'API Network', and a search bar. The main panel displays the 'Run results' for the 'QA_Environment_1' environment. The results table shows the following data:

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

Below the table, the 'All Tests' section shows 'Passed (0)', 'Failed (0)', and 'Skipped (0)'. The 'Iteration 1' section lists the following requests and their results:

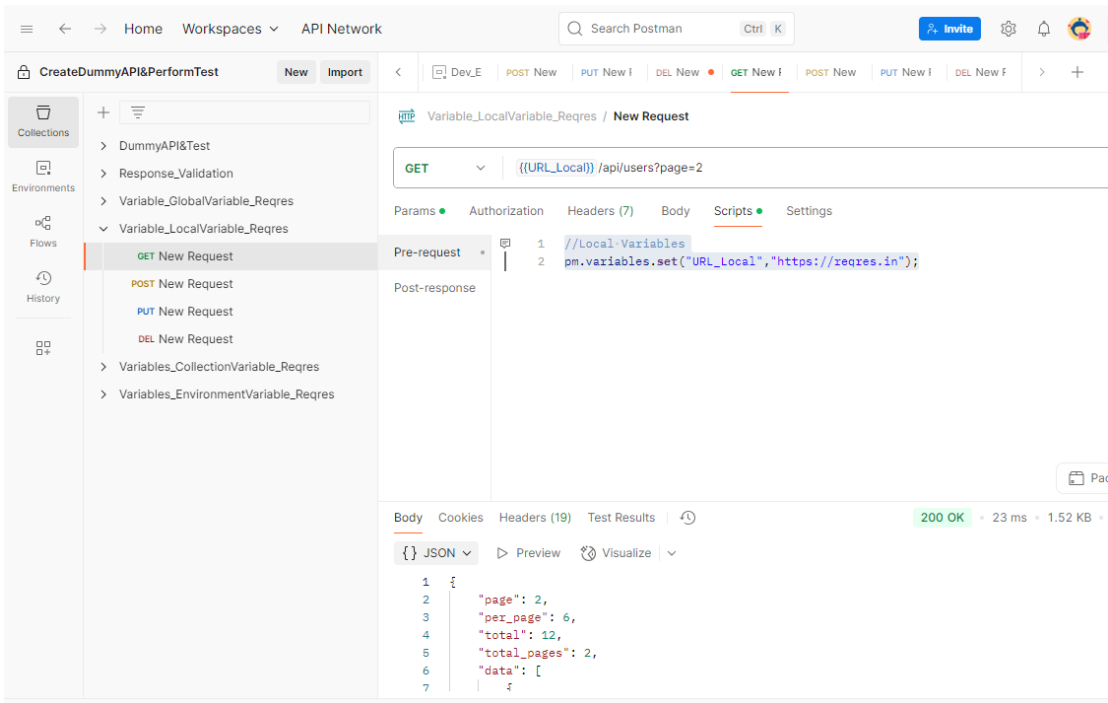
- GET Get_Request**: `https://reqres.in/api/users?page=2`, 200 OK, 97 ms, 1.556 KB
- POST Post_Request**: `https://reqres.in/api/users`, 201 Created, 606 ms, 1.114 KB
- PUT Put_Request**: `https://reqres.in/api/users/2`, 200 OK, 640 ms, 1.118 KB
- DELETE Delete_Request**: `https://reqres.in/api/users/2`, 204 No Content, 503 ms, 1.015 KB

Console result

The screenshot shows the Postman interface with the 'Console' tab selected. The console displays the following log entries:

Log Entry	Time
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	204 583 ms

Local Variables: Accessible only within the request.

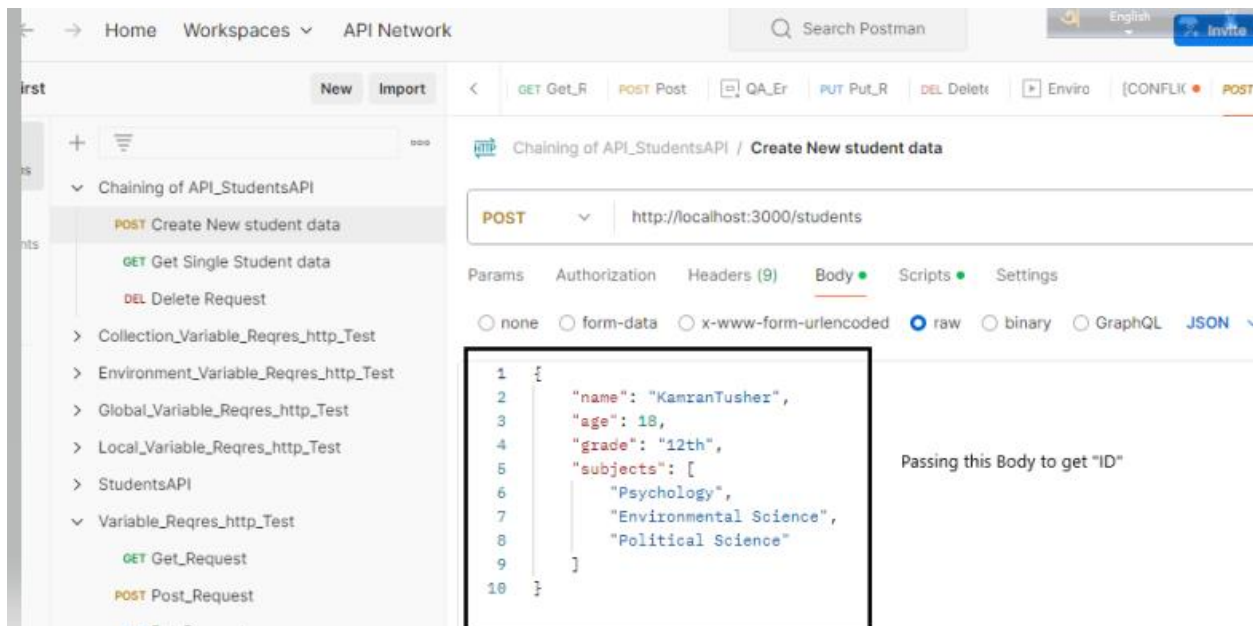


Chaining of API : The response of one API becomes the request of another API is called Chaining of API.

Task:

When I pass a body it will create an 'ID' in response to request from the body..Then this ' ID ' will 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: cf842adb5472f7db0196c587b55a74dfccd50e82e0f361ab59326843207adddf

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:

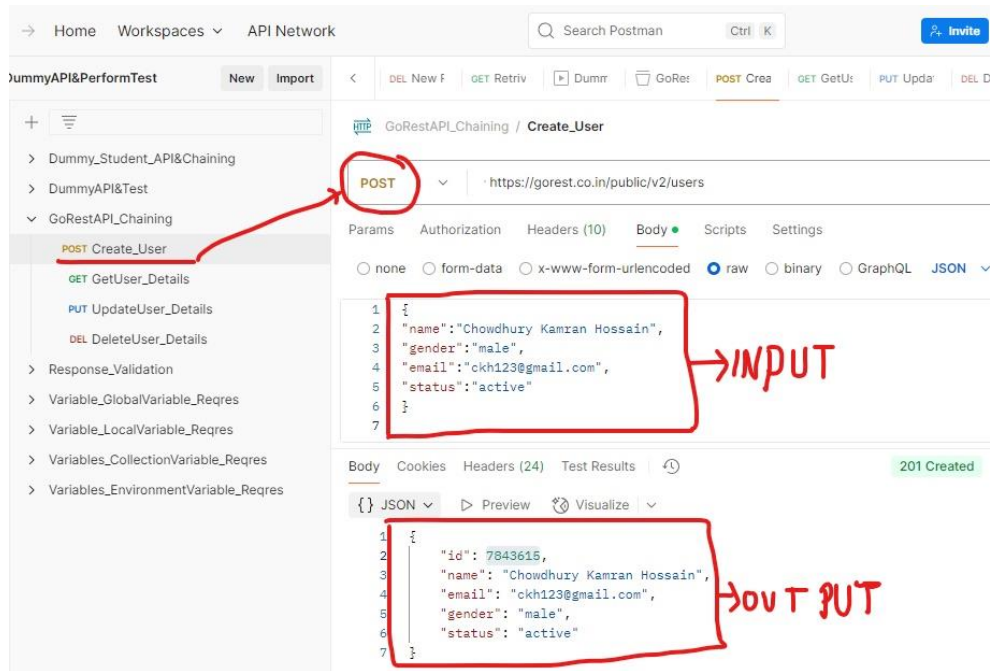
Step-1: Create a Collection called "GoRestAPI_Chaining"

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

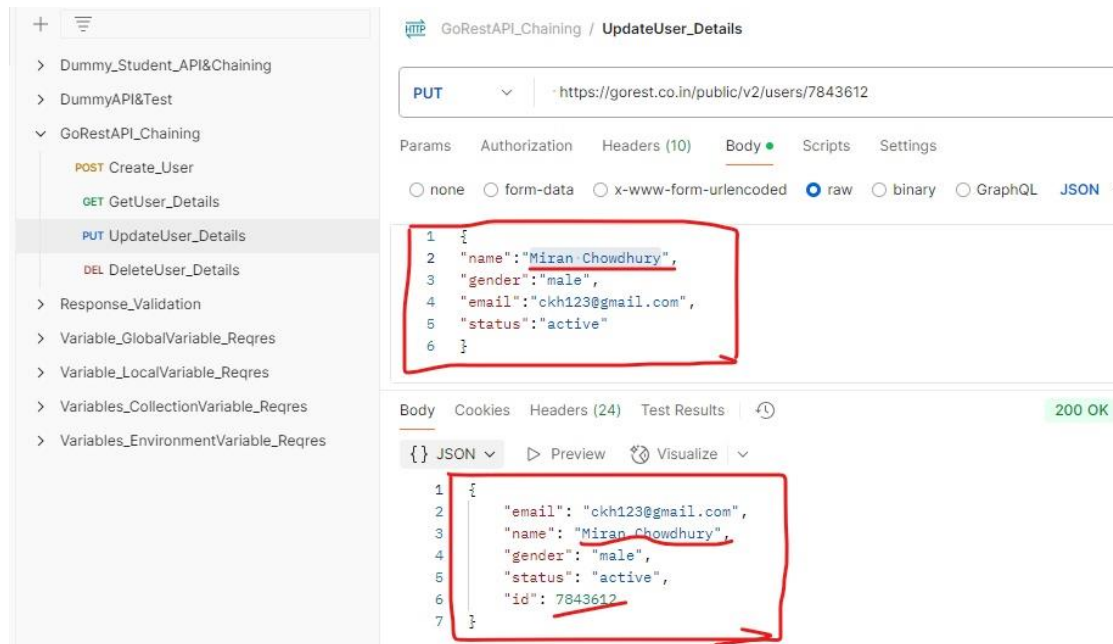
Step-3: Set the Token in the collection Level for authorization

Step-4 : Then Execute all the request

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

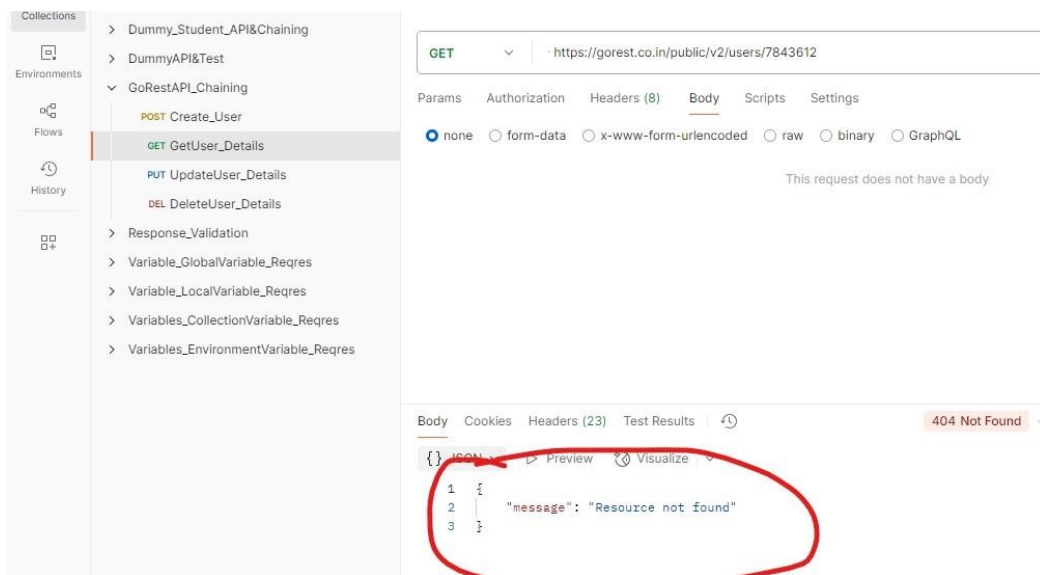
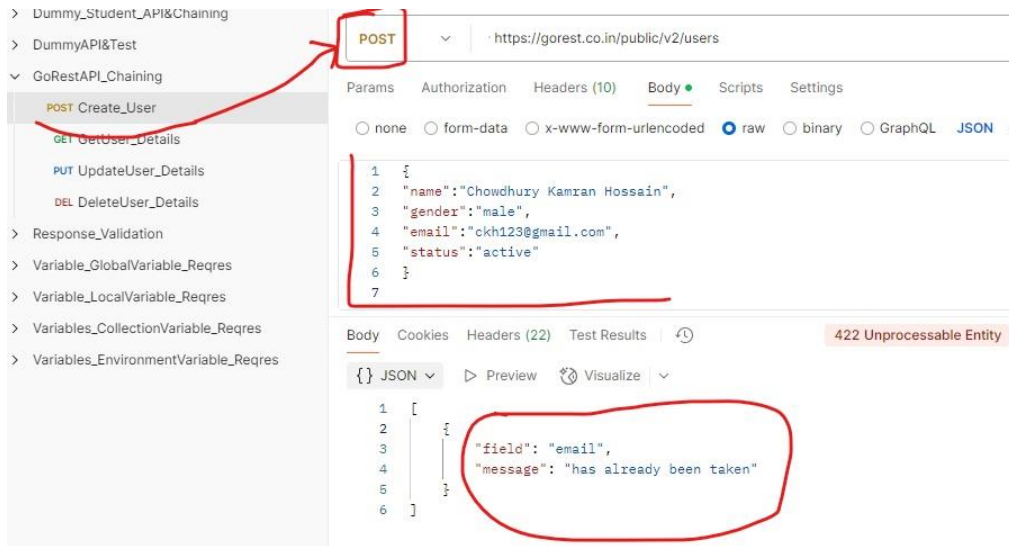


After getting the result I have updated some informations against the ID which is generated.



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 every time, 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 data 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

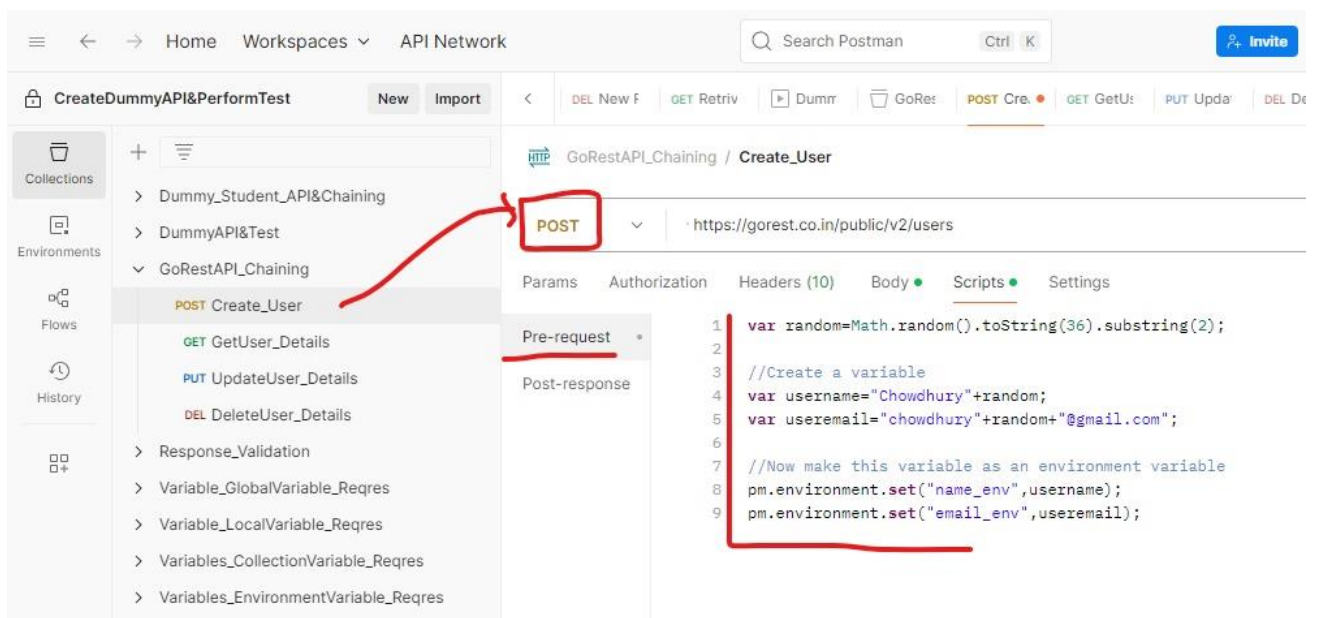
code:

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

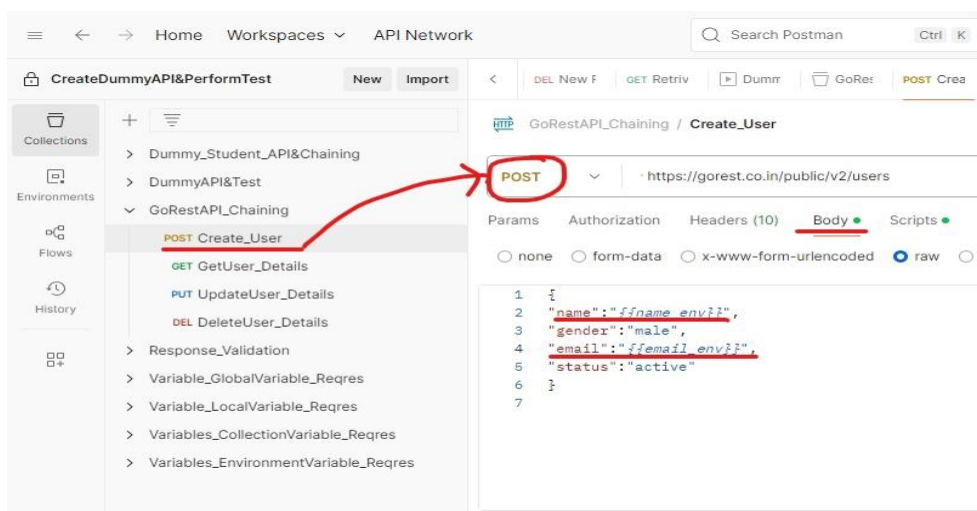
```
var username="Chowdhury"+random;
```

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

Input:



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

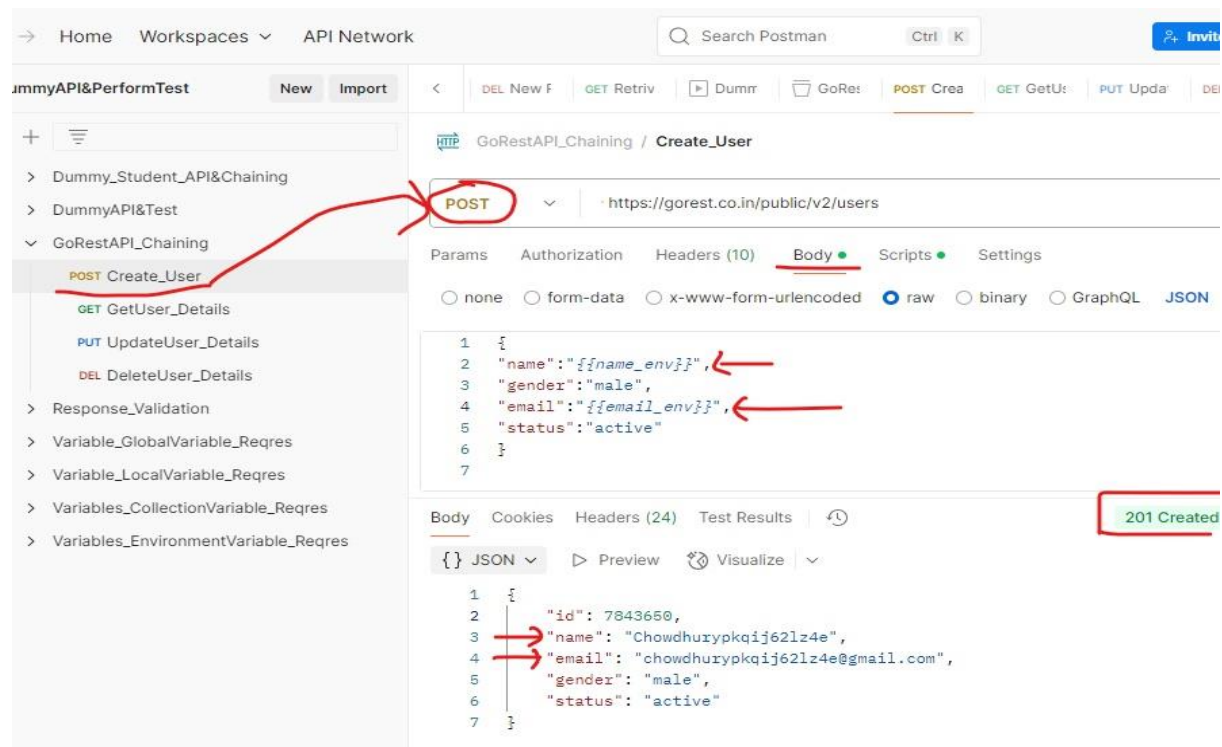


Same script is written in the body of the post request here is that code

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:

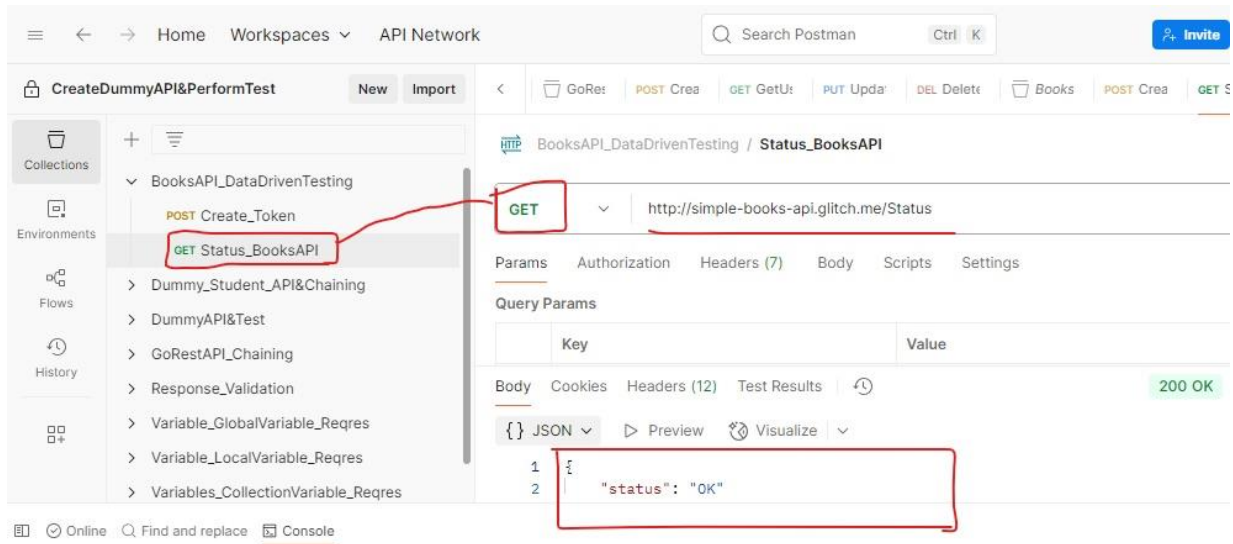
Step-2: Perform Get request using this

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

And End Point: /Status

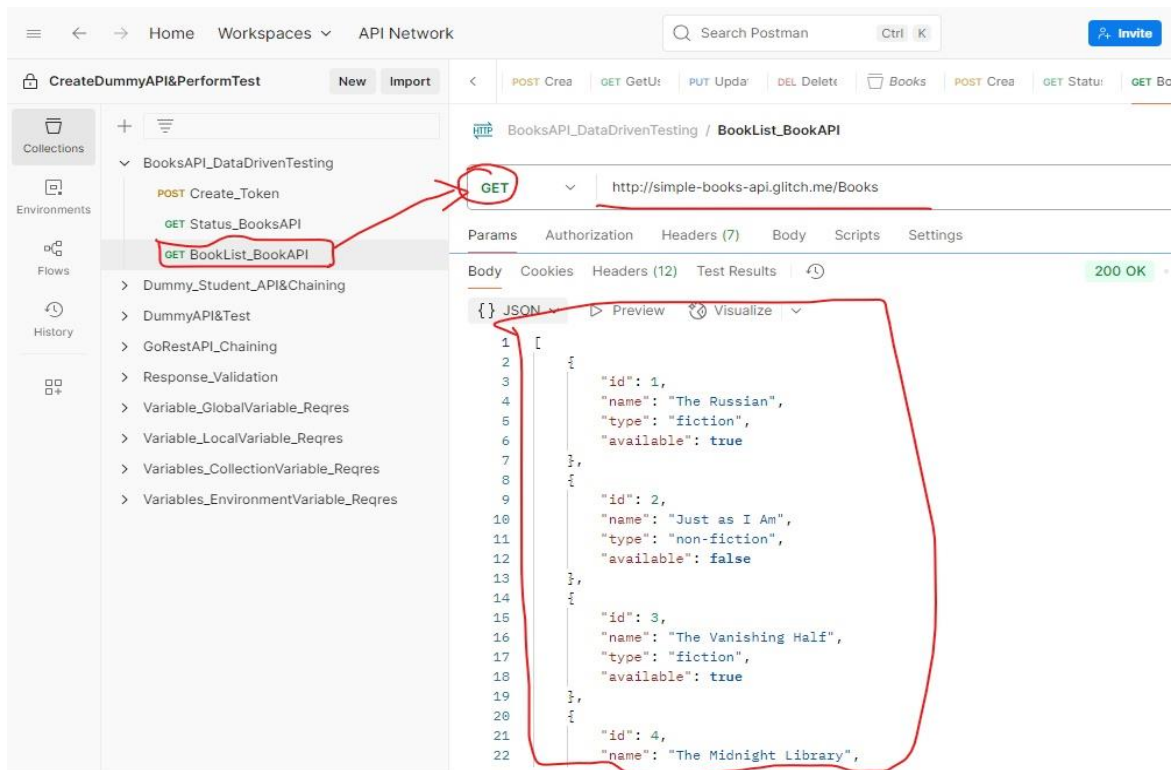
Get Request : <http://simple-books-api.glitch.me/Status>

Image:



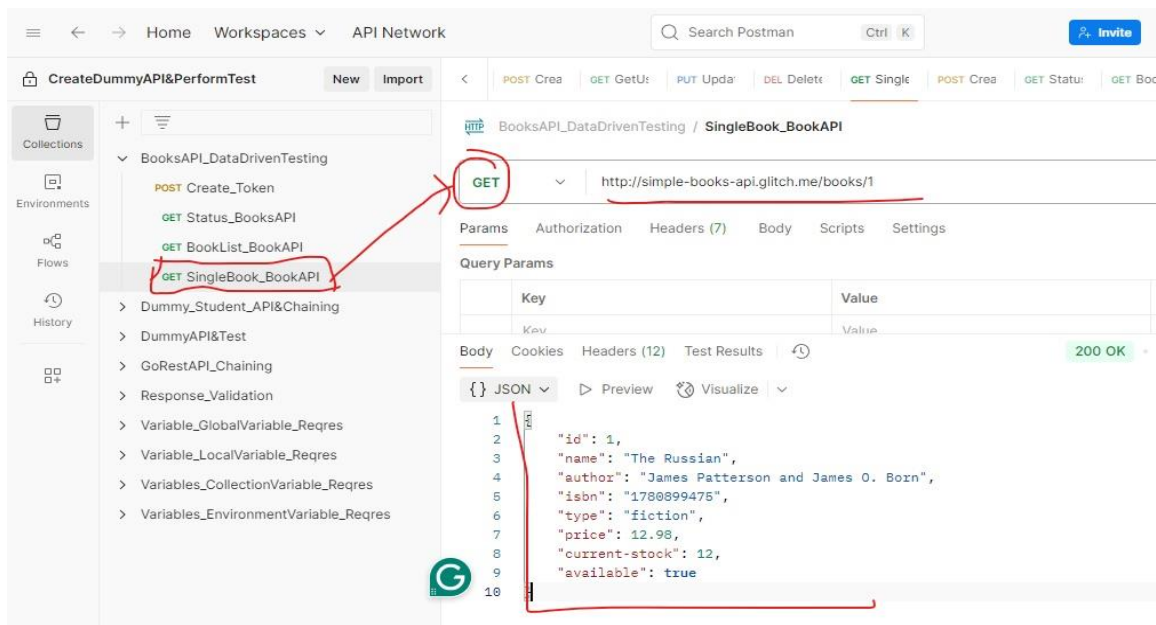
Step -3 : Perform get request using the given url for getting BookList from the BookAPI.

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 Json format.Different API request in User model

user Operations about user		Find out more about our store ^
POST	/user/createWithList Creates list of users with given input array	▼
GET	/user/{username} Get user by user name	▼
PUT	/user/{username} Updated user	▼
DELETE	/user/{username} Delete user	▼
GET	/user/login Logs user into the system	▼
GET	/user/logout Logs out current logged in user session	▼
POST	/user/createWithArray Creates list of users with given input array	▼
POST	/user Create user	▼

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

The screenshot shows the Swagger UI configuration for a POST request to the /user endpoint. The 'body' parameter is required and is of type 'object'. The 'body' tab is selected, showing a JSON schema for a 'Created user object'. The schema includes fields: id (integer), username (string), firstName (string), lastName (string), email (string), password (string), phone (string), and userStatus (integer). The 'Parameter content type' is set to 'application/json'. A 'Try it out' button is visible in the top right corner.

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

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

The screenshot shows the 'Try it out' modal for the POST /user endpoint. The 'body' parameter is required and is of type 'object'. The 'body' tab is selected, showing a JSON schema for a 'Created user object'. The schema includes fields: id (integer), username (string), firstName (string), lastName (string), email (string), password (string), phone (string), and userStatus (integer). The 'Parameter content type' is set to 'application/json'. A 'Cancel' button is visible at the bottom.

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

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

The screenshot shows the Swagger UI 'Responses' section for the POST /user endpoint. The 'Response content type' is set to 'application/xml'. The 'Curl' section shows the curl command for the request. The 'Request URL' is https://petstore.swagger.io/v2/user. The 'Server response' section shows the response body, which is an XML document with a status of 200 and a message of '18'. The response headers include access-control-allow-headers, access-control-allow-methods, access-control-allow-origin, content-length, content-type, date, and server.

```
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
  }'
```

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

Response headers:

```
access-control-allow-headers: Content-Type,api_key,Authorization
access-control-allow-methods: GET,POST,DELETE,PUT
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)
```

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