

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 :

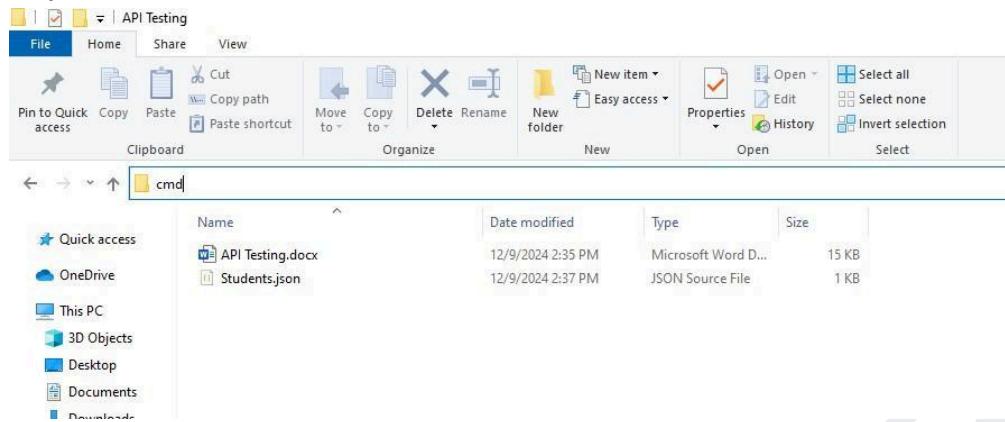
```
{  
  "students": [  
    {  
      "id": 1,  
      "name": "John Doe",  
      "age": 18,  
      "grade": "12th",  
      "subjects": [  
        "Math",  
        "Physics"  
        ,  
        "English"  
      ]  
    },  
    {  
      "id": 2,  
      "name": "Jane Smith",  
      "age": 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?

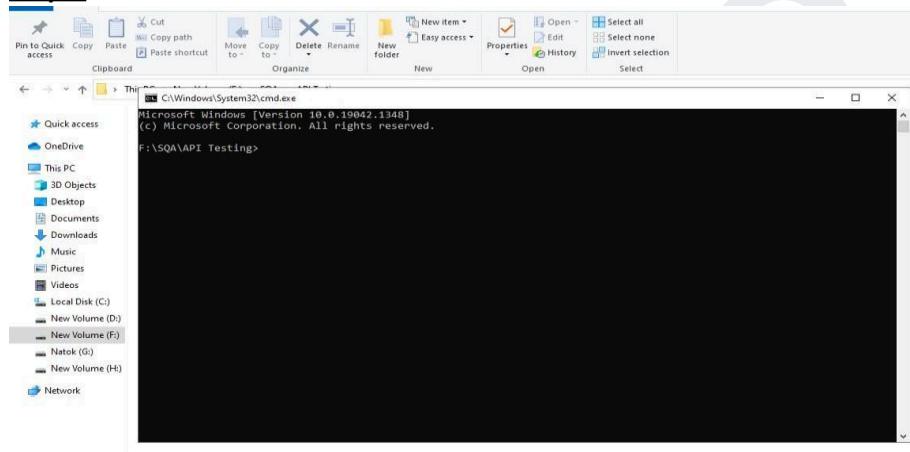
Task no-1

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

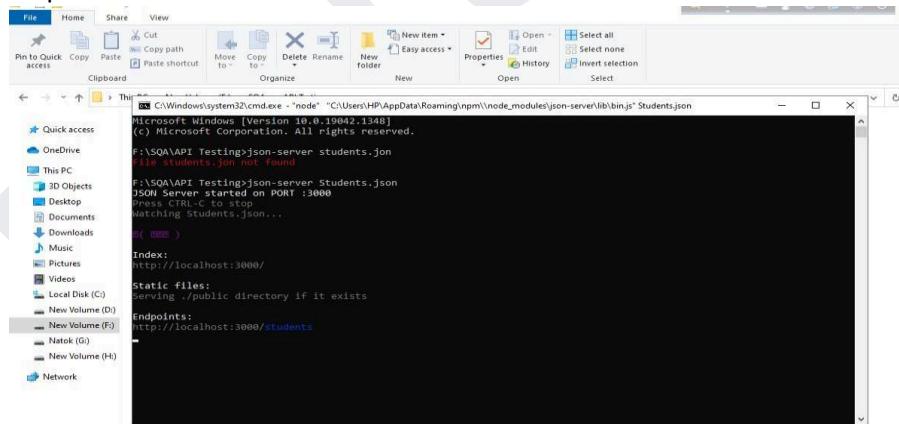
Step-1



Step-2



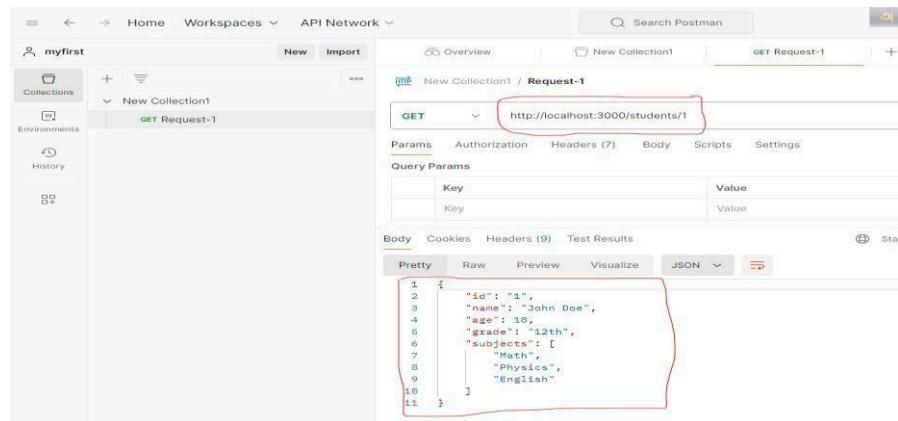
Step-3



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

How to get single student data from the above URL?

Task No- 2



The screenshot shows the Postman interface with a collection named "myfirst". A new collection "New Collection1" is expanded, showing a "GET Request-1" with the URL `http://localhost:3000/students/1`. The "Body" tab is selected, displaying the JSON response:

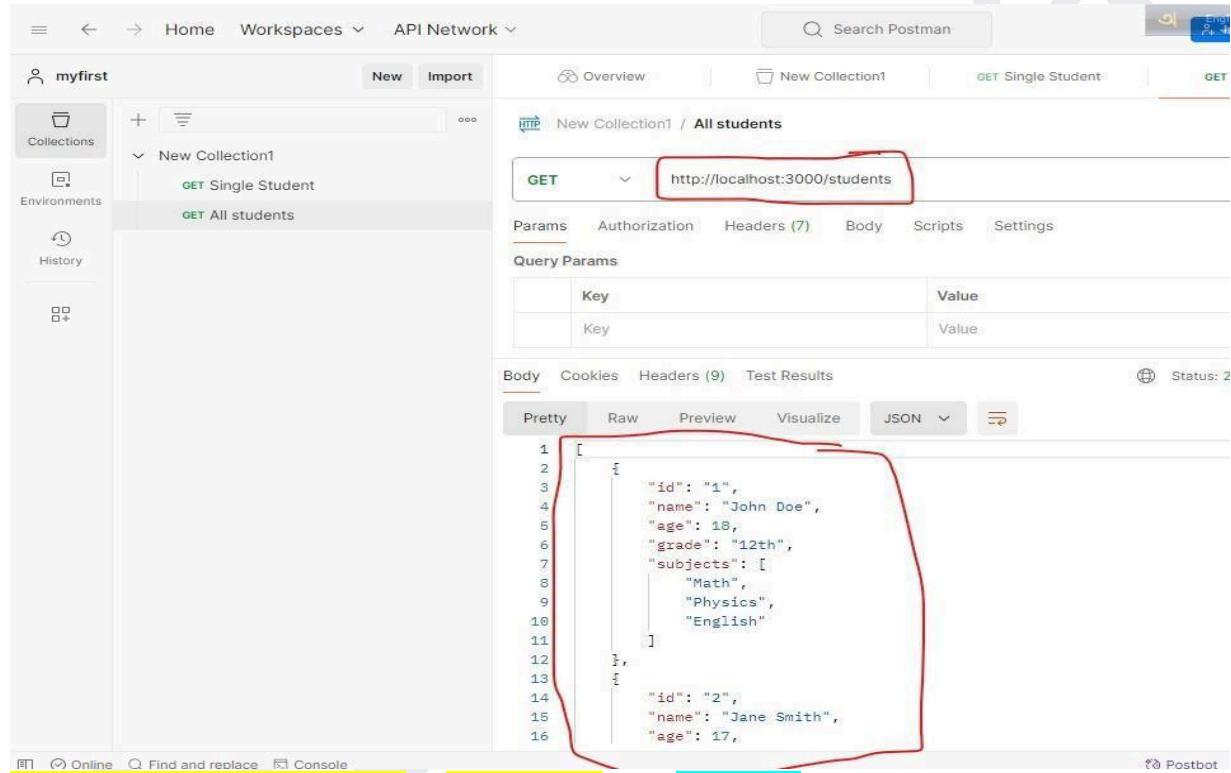
```

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

```

How to get all students data from the above URL?

Task No- 3



The screenshot shows the Postman interface with a collection named "myfirst". A new collection "New Collection1" is expanded, showing a "GET All students" request with the URL `http://localhost:3000/students`. The "Body" tab is selected, displaying the JSON response:

```

1  [
2    {
3      "id": "1",
4      "name": "John Doe",
5      "age": 18,
6      "grade": "12th",
7      "subjects": [
8        "Math",
9        "Physics",
10       "English"
11     ]
12   },
13   {
14     "id": "2",
15     "name": "Jane Smith",
16     "age": 17,
17   }
18 ]

```

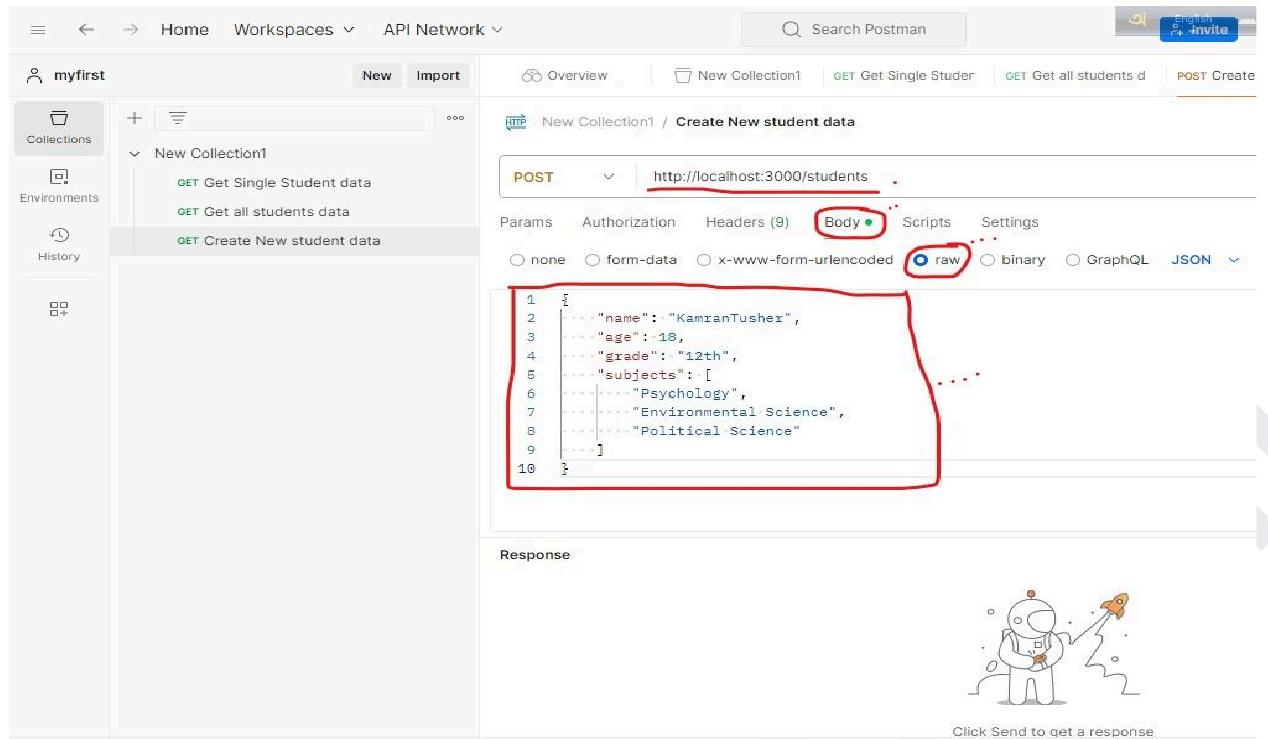
How to create new student data?

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)



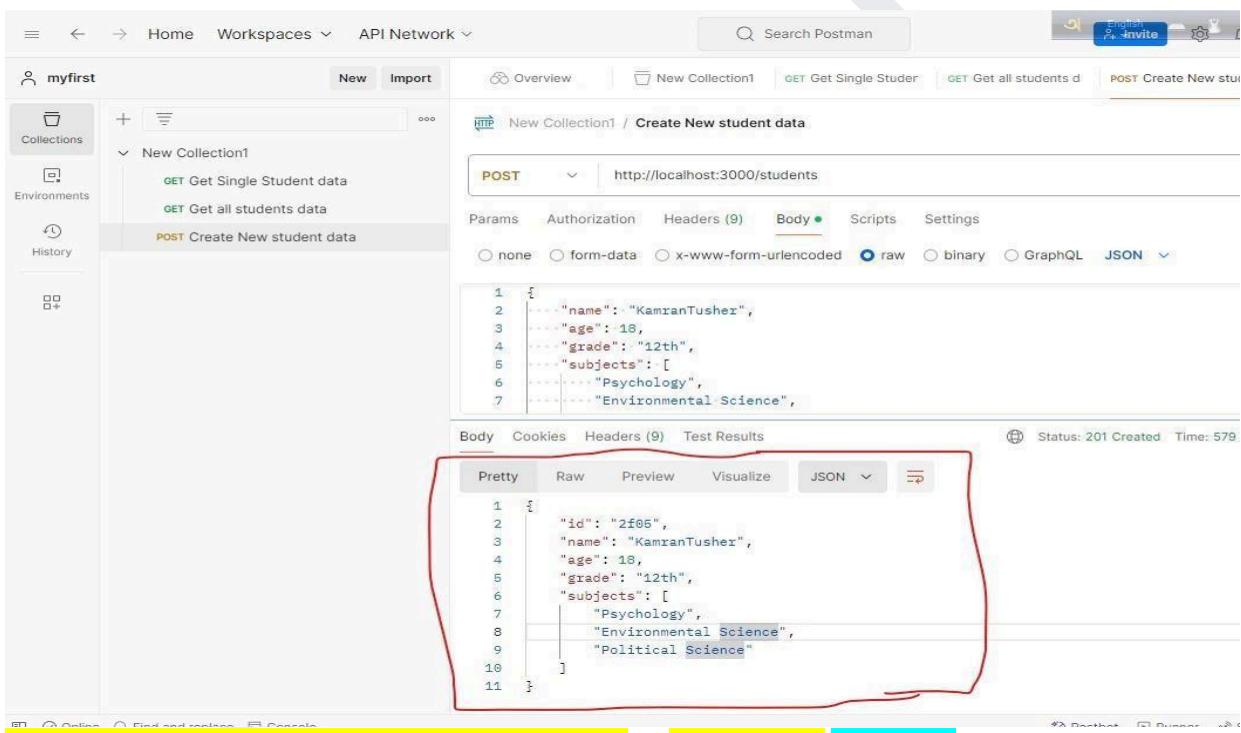
The screenshot shows the Postman interface with a collection named "myfirst". A POST request is selected with the URL `http://localhost:3000/students`. The "Body" tab is active, and the "raw" option is selected. A red box highlights the JSON payload in the body editor:

```

1  {
2   "name": "KamranTusher",
3   "age": 18,
4   "grade": "12th",
5   "subjects": [
6     "Psychology",
7     "Environmental Science",
8     "Political Science"
9   ]
10 }

```

Step-2 (Response Payload)



The screenshot shows the same Postman interface after the request was sent. The status bar indicates "Status: 201 Created Time: 579". The response body is displayed in the "Pretty" tab of the results panel:

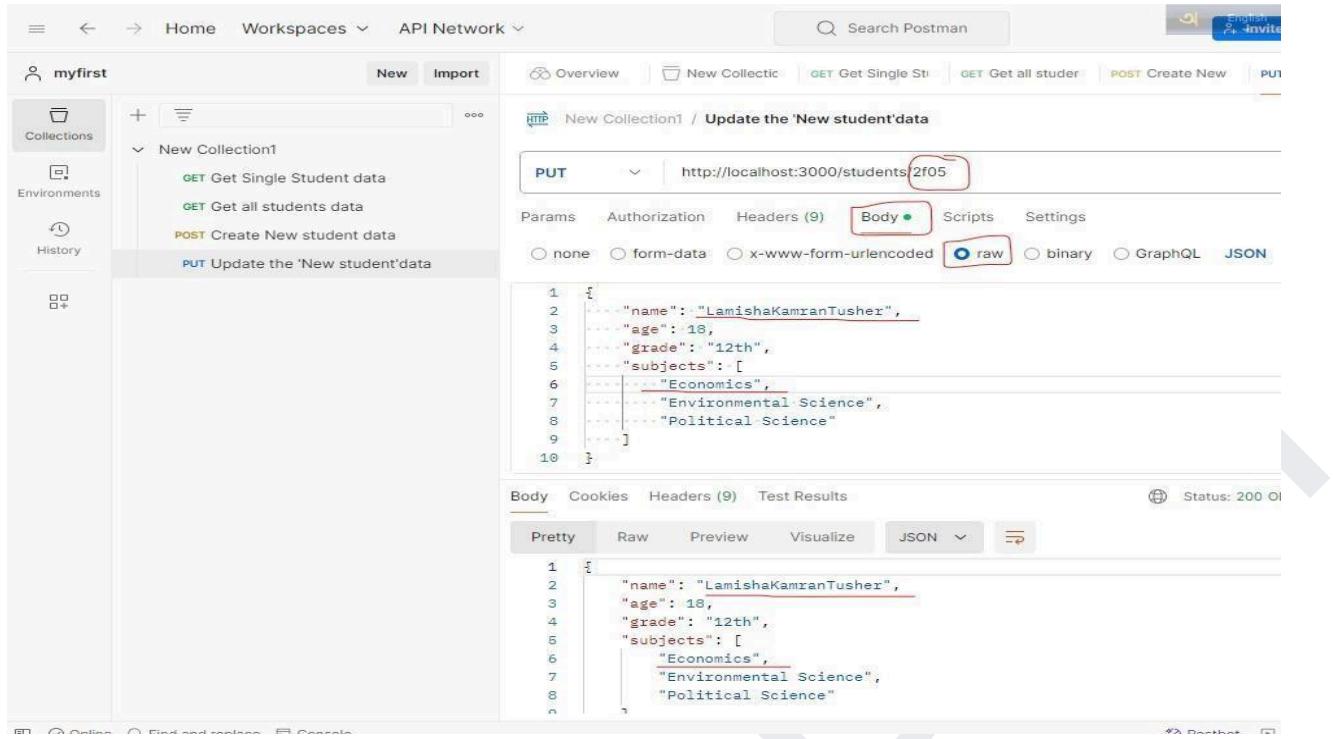
```

1  {
2   "id": "2f05",
3   "name": "KamranTusher",
4   "age": 18,
5   "grade": "12th",
6   "subjects": [
7     "Psychology",
8     "Environmental Science",
9     "Political Science"
10 ]
11 }

```

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>)



The screenshot shows the Postman interface with a collection named "myfirst". A PUT request is selected to update student data. The URL is set to `http://localhost:3000/students/2f05`. The "Body" tab is active, showing a JSON payload:

```

1  {
2    "name": "LamishaKamranTusher",
3    "age": 18,
4    "grade": "12th",
5    "subjects": [
6      "Economics",
7      "Environmental Science",
8      "Political Science"
9    ]
10 }

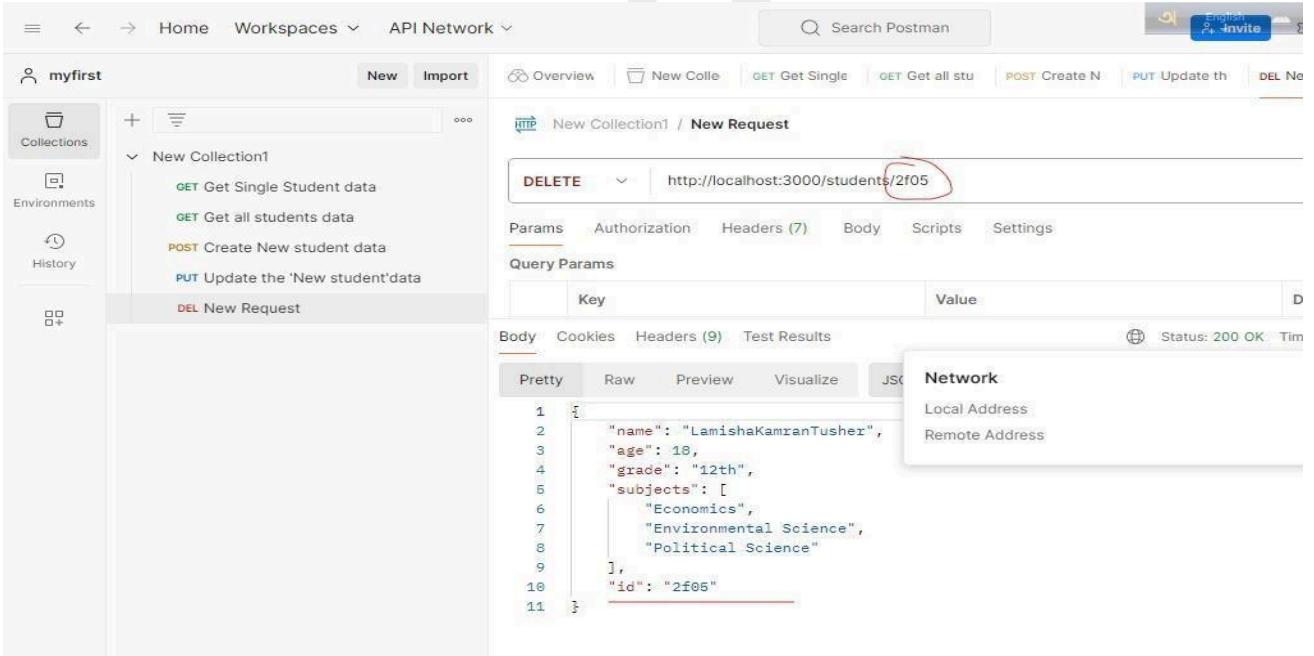
```

The response status is 200 OK, and the body of the response is identical to the request body.

How to delete any data from the request ?

Task No- 6

Ans : I want to delete ID = “ 2f05 ”



The screenshot shows the Postman interface with a collection named "myfirst". A DELETE request is selected to delete student data. The URL is set to `http://localhost:3000/students/2f05`. The "Body" tab is active, showing a JSON payload:

```

1  {
2    "name": "LamishaKamranTusher",
3    "age": 18,
4    "grade": "12th",
5    "subjects": [
6      "Economics",
7      "Environmental Science",
8      "Political Science"
9    ],
10   "id": "2f05"
11 }

```

A tooltip labeled "Network" is visible, showing "Local Address" and "Remote Address". The response status is 200 OK.

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

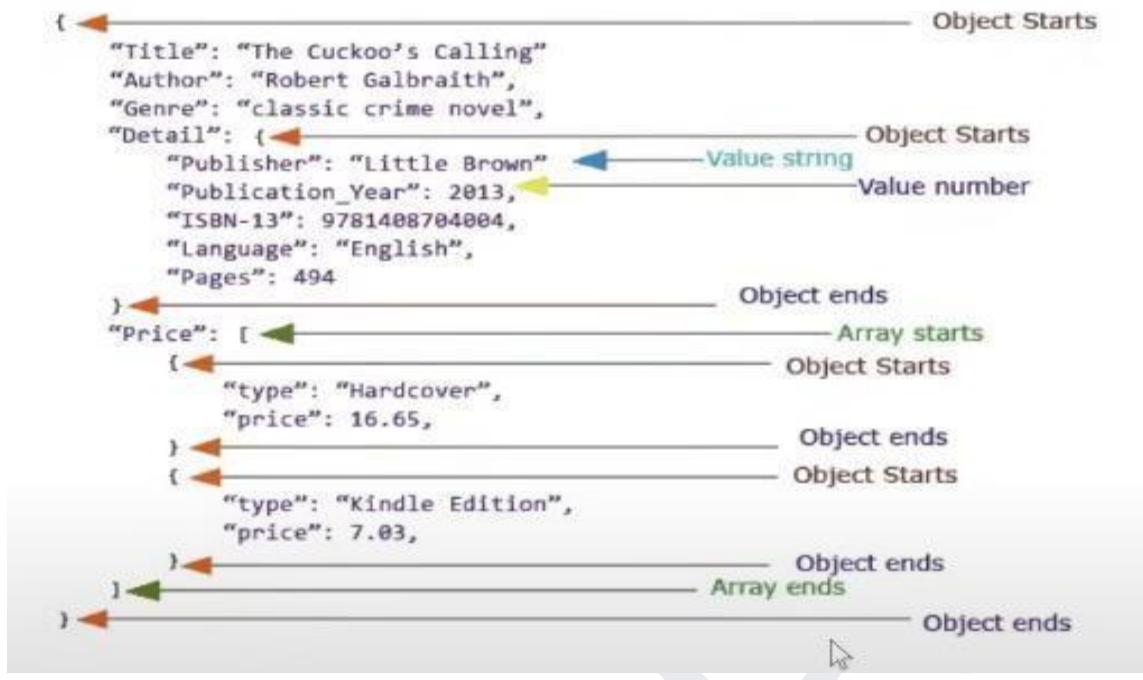
```
{
  "student": [
    {
      "id": "01",
      "name": "Tom",
      "lastname": "Price"
    },
    {
      "id": "02",
      "name": "Nick",
      "lastname": "Thameson"
    }
  ]
}
```

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.

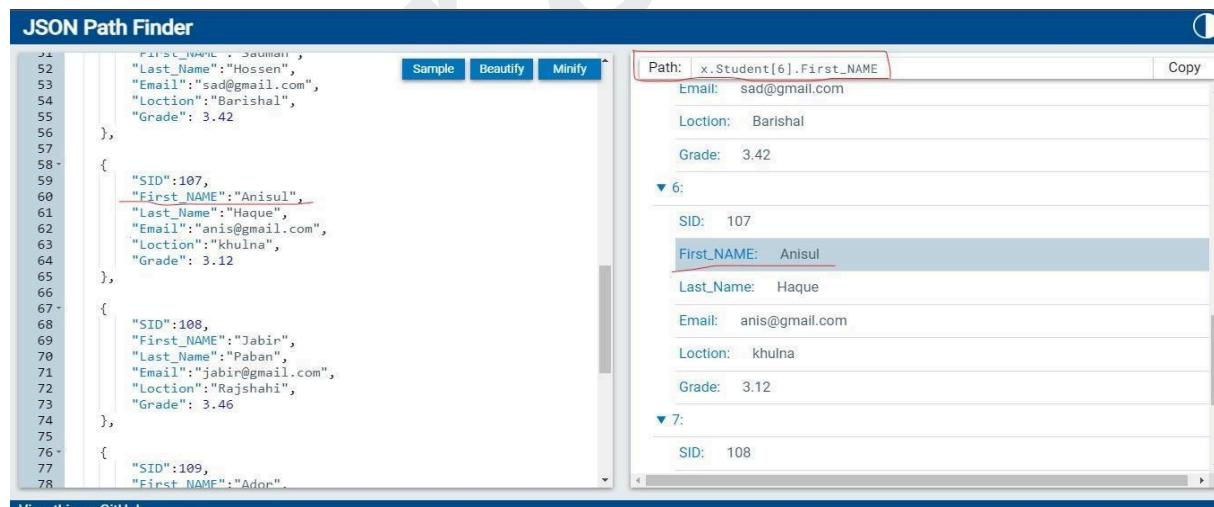
Extract Procedure— JSON Path : **Student[1].SName** **Lamisha Rahman**

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](#)

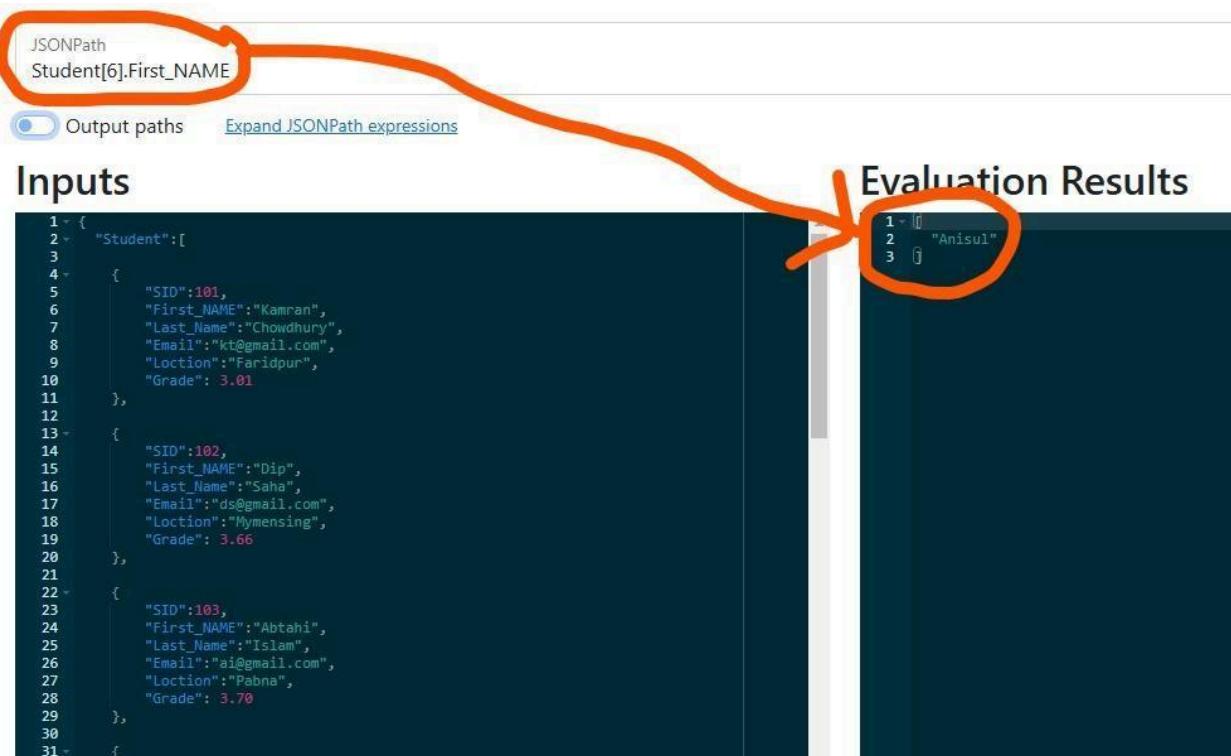
JSON Path Finder



The screenshot shows the JSON Path Finder interface. On the left, a large JSON document is displayed with several fields highlighted in red, including "SName" and "First_NAME". On the right, a results pane shows the extracted data for the second student (index 1). The results pane includes a search bar with the path "x.Student[1].First_NAME", and below it, the extracted values for "Email", "Loction", "Grade", "SID", "First_NAME", "Last_Name", and "Email".

JSON path Validation

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



The screenshot shows a JSONPath validation interface. On the left, under "Inputs", there is a JSON document with 31 lines of student data. On the right, under "Evaluation Results", the output of the JSONPath expression "Student[6].First_NAME" is shown, which evaluates to "Anisul". Both sections are circled in orange.

```

1+ {
2+   "Student": [
3+
4+     {
5+       "SID":101,
6+       "First_NAME":"Kamran",
7+       "Last_Name":"Chowdhury",
8+       "Email":"kt@gmail.com",
9+       "Loction":"Faridpur",
10+      "Grade": 3.01
11+    },
12+
13+    {
14+      "SID":102,
15+      "First_NAME":"Dip",
16+      "Last_Name":"Saha",
17+      "Email":"ds@gmail.com",
18+      "Loction":"Mymensing",
19+      "Grade": 3.66
20+    },
21+
22+    {
23+      "SID":103,
24+      "First_NAME":"Abtahi",
25+      "Last_Name":"Islam",
26+      "Email":"ai@gmail.com",
27+      "Loction":"Pabna",
28+      "Grade": 3.70
29+    },
30+
31+
  
```

Evaluation Results

```

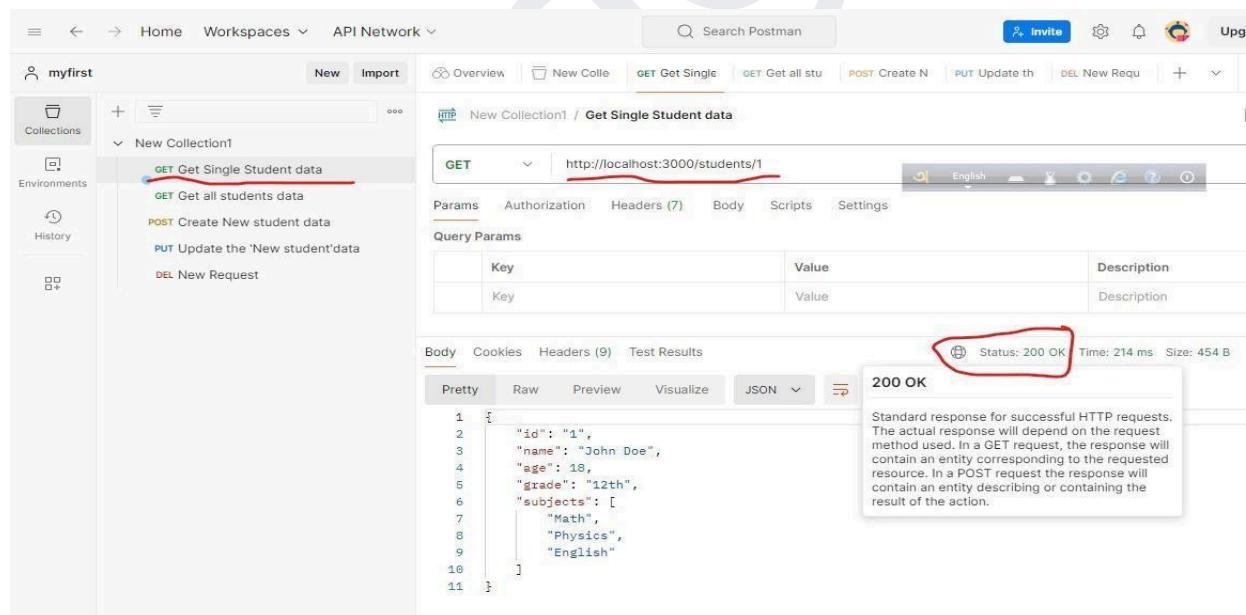
1+ []
2+ "Anisul"
3+ []
  
```

Response Validation

Task No--8

1. Status Code:

Step-1



The screenshot shows a Postman collection named "myfirst" with a single collection named "New Collection1". It contains a GET request for "Get Single Student data" with the URL "http://localhost:3000/students/1". The response status is 200 OK, and the response body is a JSON object representing a student record.

New Collection1 / Get Single Student data

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

Body

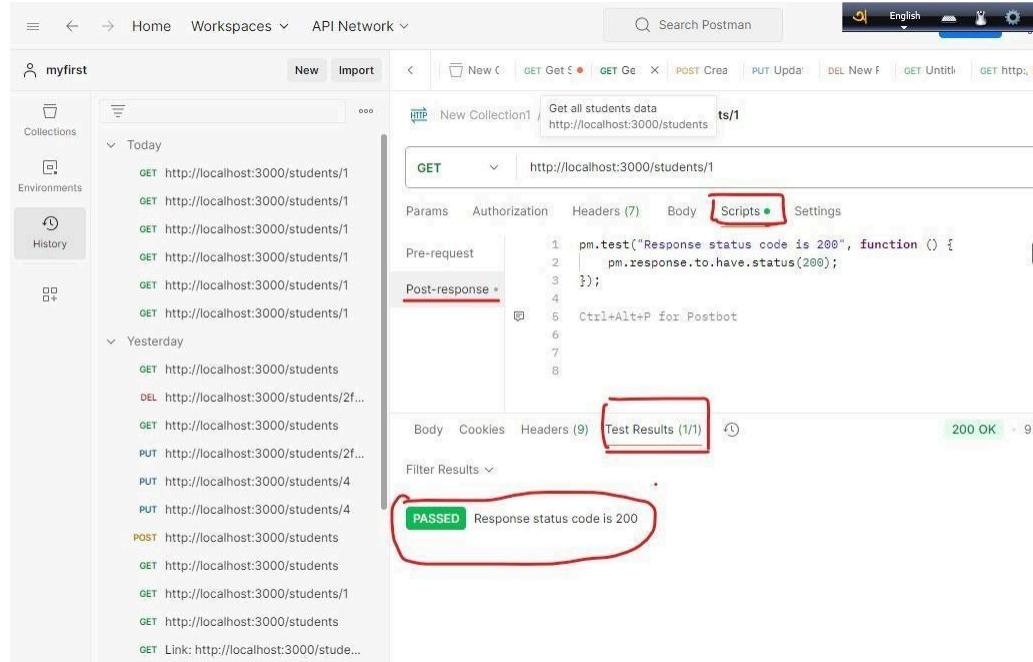
```

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);
});
```



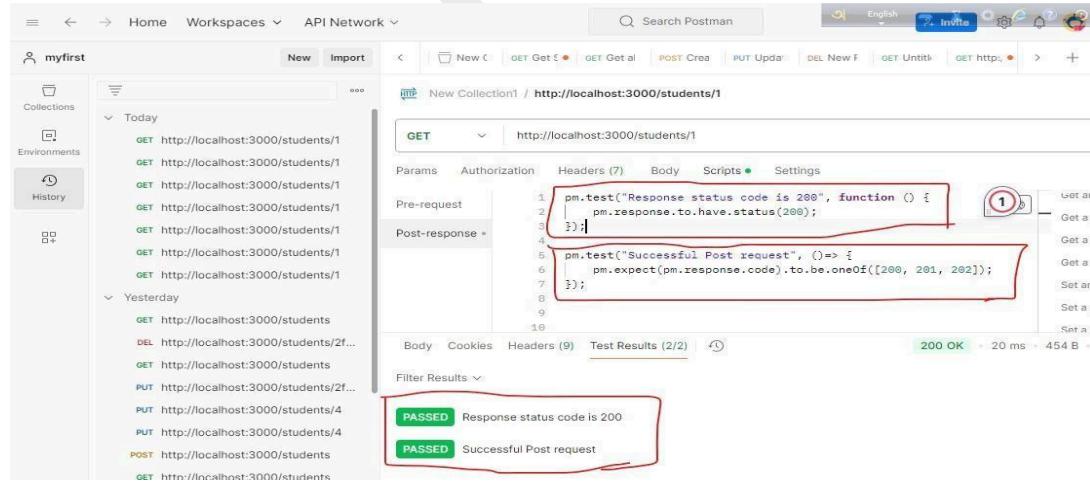
The screenshot shows the Postman interface with a collection named "myfirst". A new collection titled "New Collection1" is selected, with the URL "http://localhost:3000/students/1". The "Test" tab is open, showing a "Post-response" script:

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

The "Test Results" section shows one result: "PASSED Response status code is 200".

For Multiple Status code

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



The screenshot shows the Postman interface with a collection named "myfirst". A new collection titled "New Collection1 / http://localhost:3000/students/1" is selected, with the URL "http://localhost:3000/students/1". The "Test" tab is open, showing a "Post-response" script:

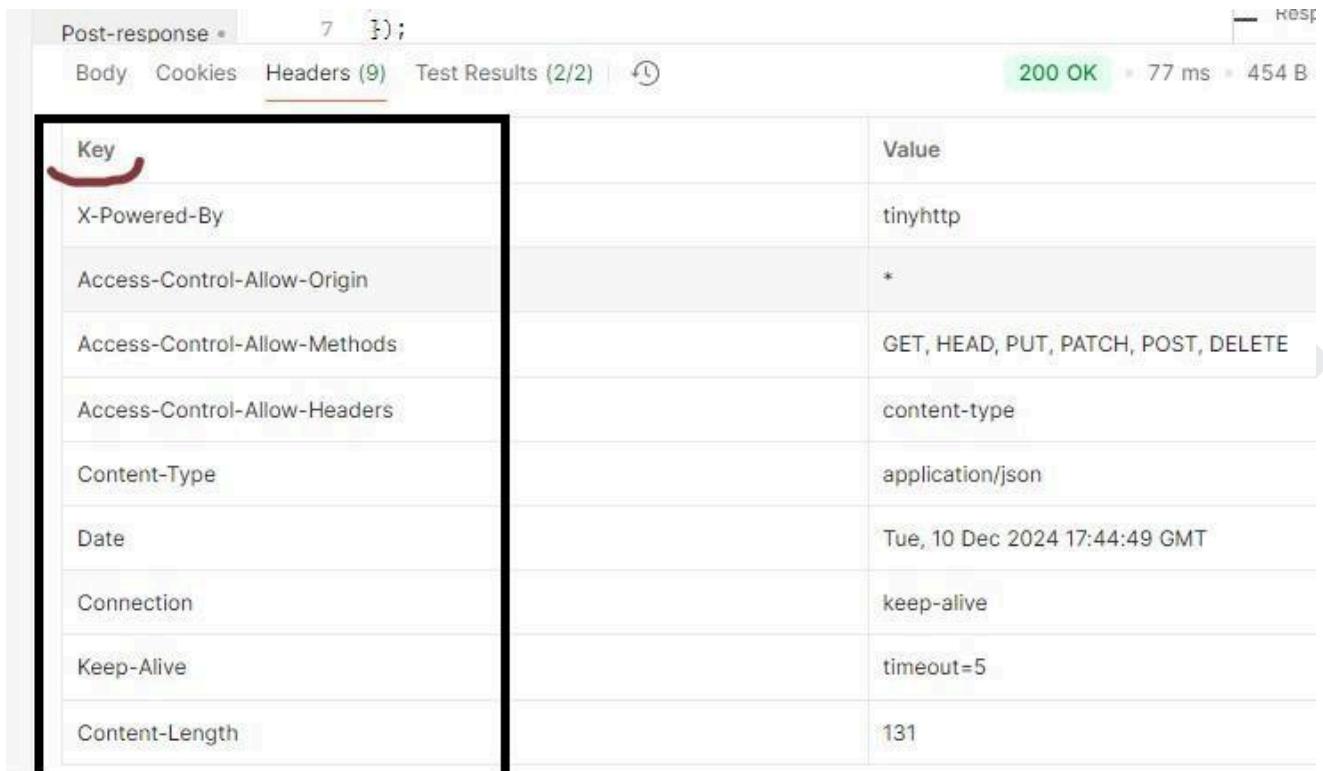
```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});
pm.test("Successful Post request", ()=> {
  pm.expect(pm.response.code).to.be.oneOf([200, 201, 202]);
});
```

The "Test Results" section shows two results: "PASSED Response status code is 200" and "PASSED Successful Post request".

1. Headers Validation

Task No—9

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



The screenshot shows the Postman interface with the "Headers" tab selected. A red arrow points to the "Key" column of the table.

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");
});

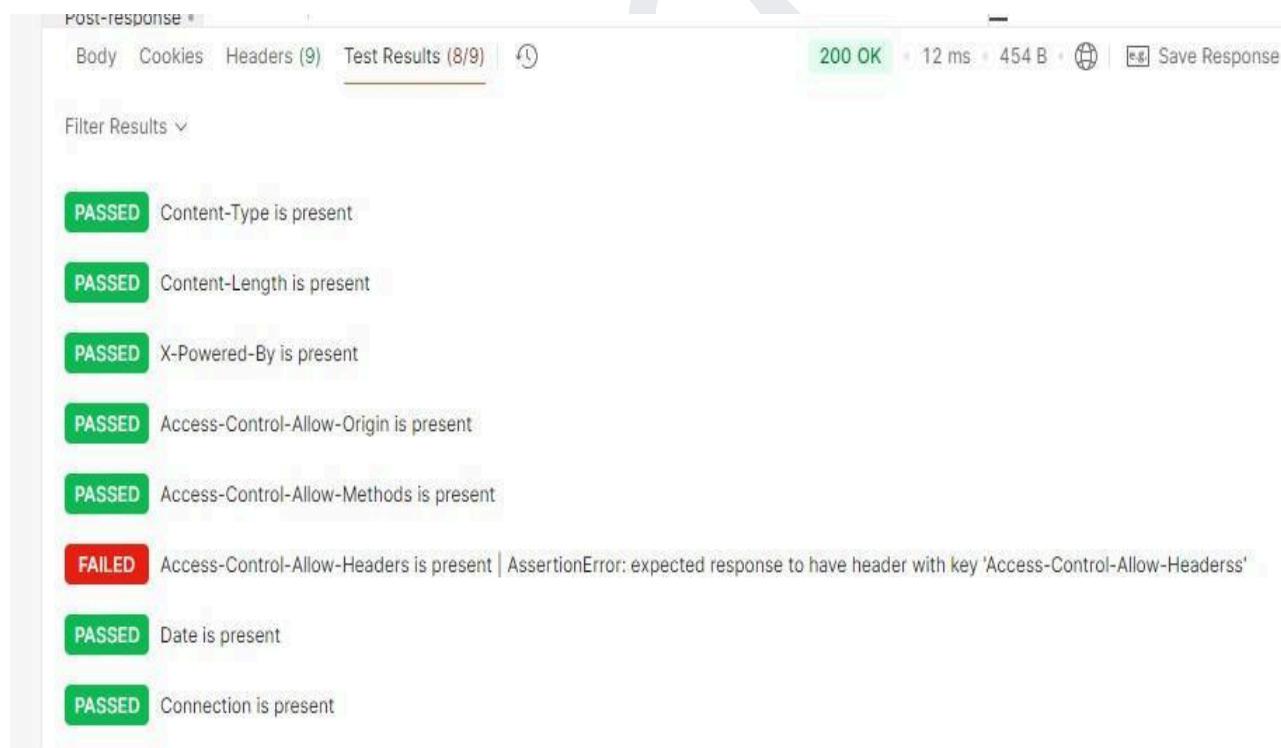
pm.test("Access-Control-Allow-Headers is present", function () {
  pm.response.to.have.header("Access-Control-Allow-Headersss");
});

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:



The screenshot shows the Postman interface displaying test results. The top navigation bar includes 'Post-response', 'Body', 'Cookies', 'Headers (9)', 'Test Results (8/9)' (which is underlined), and a help icon. To the right, it shows '200 OK', '12 ms', '454 B', a network icon, and 'Save Response'.

The 'Test Results' section has a 'Filter Results' dropdown. Below it, eight assertions are listed:

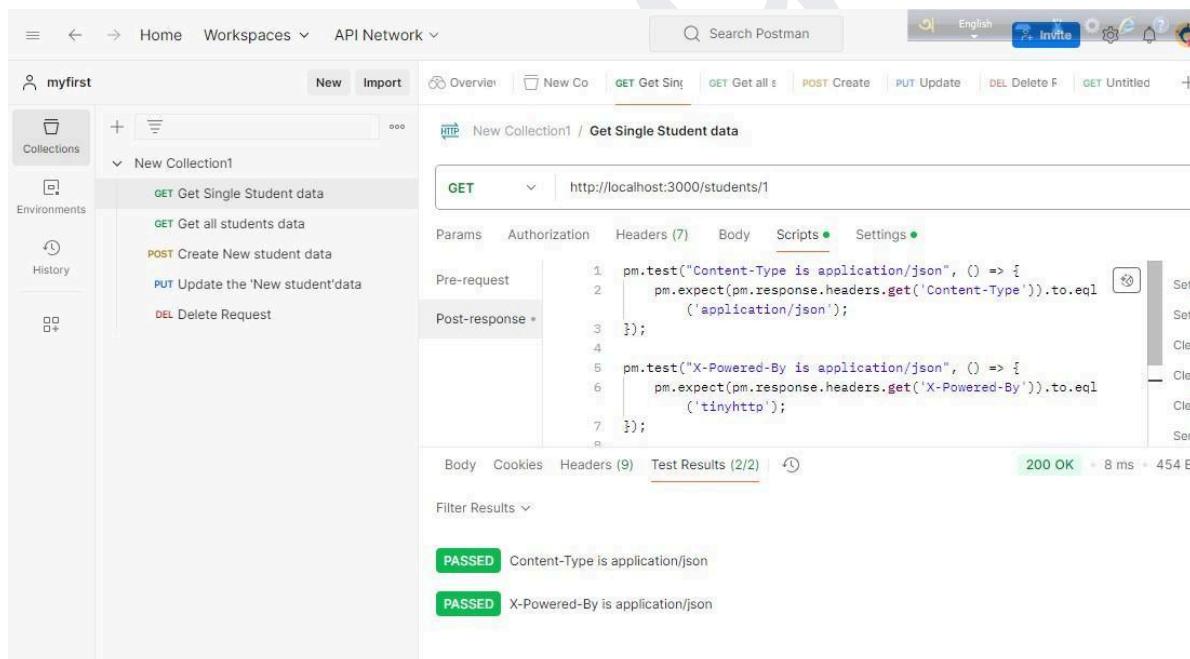
- PASSED Content-Type is present
- PASSED Content-Length is present
- PASSED X-Powered-By is present
- PASSED Access-Control-Allow-Origin is present
- PASSED Access-Control-Allow-Methods is present
- FAILED Access-Control-Allow-Headers is present | Assertion Error: expected response to have header with key 'Access-Control-Allow-Headersss'
- PASSED Date is present
- PASSED Connection is present

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 the following details:

- Collections:** myfirst
- New Collection1** is currently selected.
- Request:** GET Get Single Student data
- URL:** http://localhost:3000/students/1
- Pre-request Script:**

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

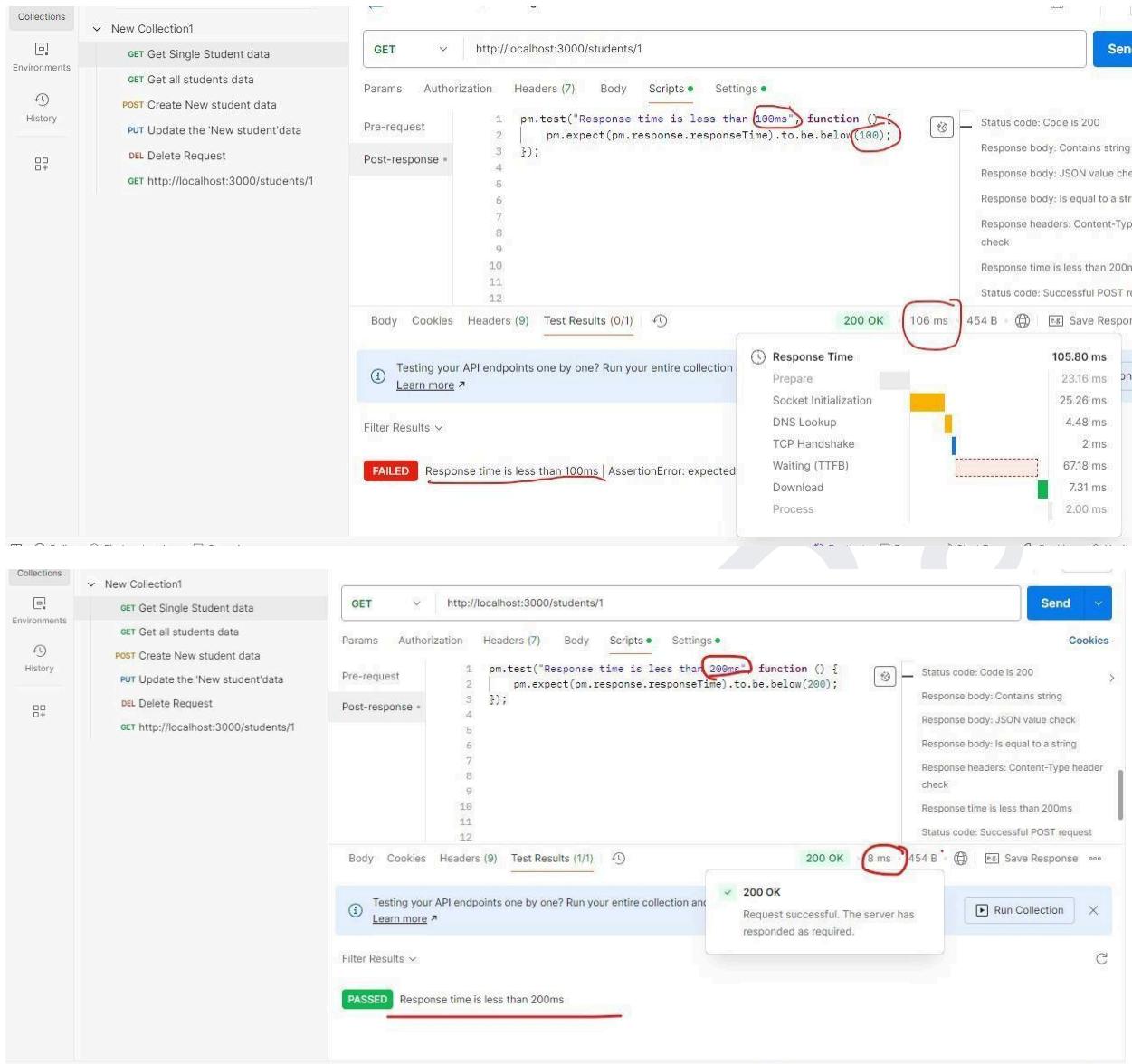
Ans:

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 screenshot shows two separate API requests in Postman:

- Request 1 (Top):** GET http://localhost:3000/students/1. Response time: 106 ms. Status code: 200 OK. Assertion failed: "Response time is less than 100ms".
- Request 2 (Bottom):** GET http://localhost:3000/students/1. Response time: 8 ms. Status code: 200 OK. Assertion 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 Runner New

Collections Environments History

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

HTTP New Collection1 / Get Single Student data

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 );
```

Pre-request Post-response

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

200 OK 81

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 }
```

myfirst

New Import

Overview GET http://loca GET Get Single GET Get all stu Postman Runner

Collections Environments History

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

HTTP New Collection1 / Get Single Student data

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 );
```

Pre-request Post-response

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 shows the Postman application interface. On the left, there's a sidebar with icons for Overview, Collections, Environments, and History. The main area displays a collection titled "New Collection1 / Get all students data". A GET request is defined with the URL "http://localhost:3000/students". The "Scripts" tab is selected, showing a pre-request script and a post-response script. The post-response script contains the following 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
    });
});
```

Below the script, there are tabs for Body, Cookies, Headers (9), and Test Results (4/4). The test results show four passed assertions:

- 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:

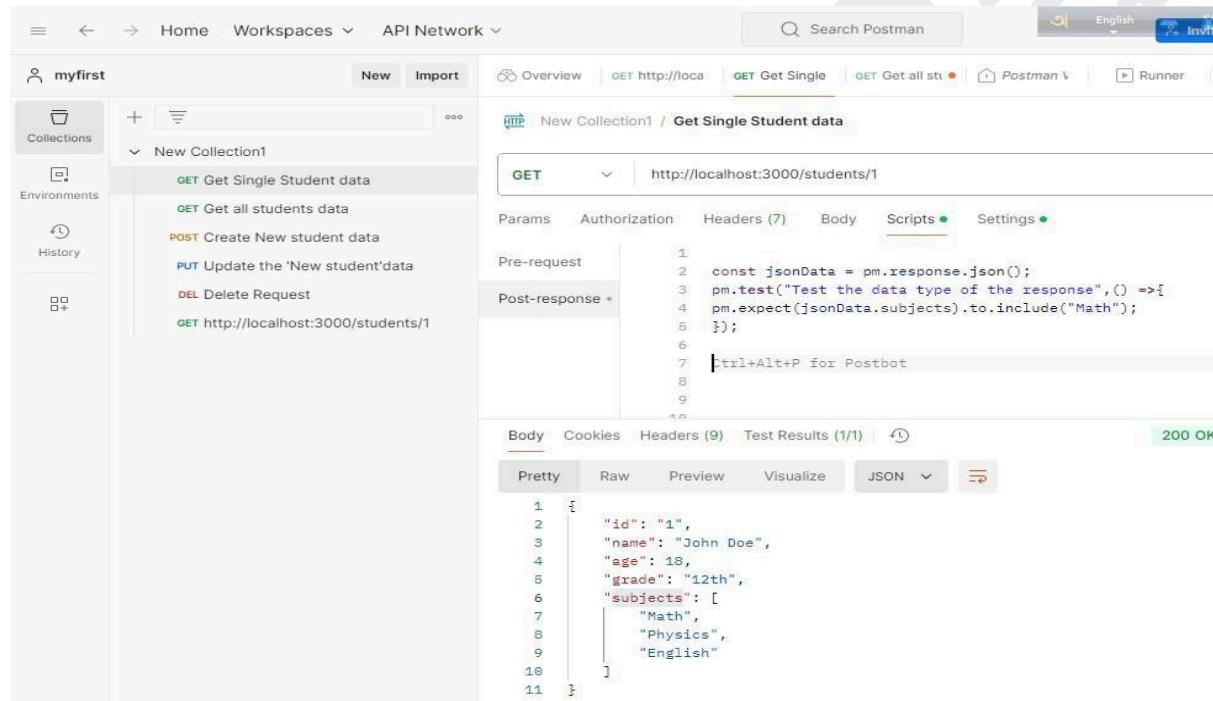
```
const jsonData = pm.response.json();

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

pm.expect(jsonData.subjects).to.include("Math");

});
```

Input:

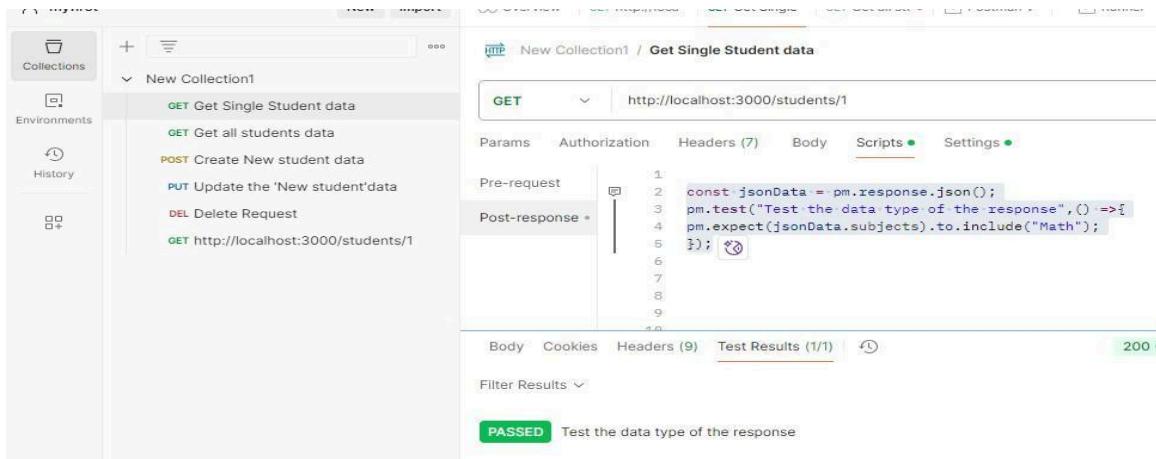


The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (selected), 'Environments', and 'History'. The main area shows a collection named 'myfirst' with one item: 'New Collection1 / Get Single Student data'. This item has a 'GET' request with the URL 'http://localhost:3000/students/1'. In the 'Tests' tab of the request configuration, there is a test script:

```
1 const jsonData = pm.response.json();
2 pm.test("Test the data type of the response",() =>{
3 pm.expect(jsonData.subjects).to.include("Math");
4 });
5 
```

The 'Body' tab of the response panel shows the JSON response in a pretty-printed 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 }
```



```

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

PASSSED Test the data type of the response

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

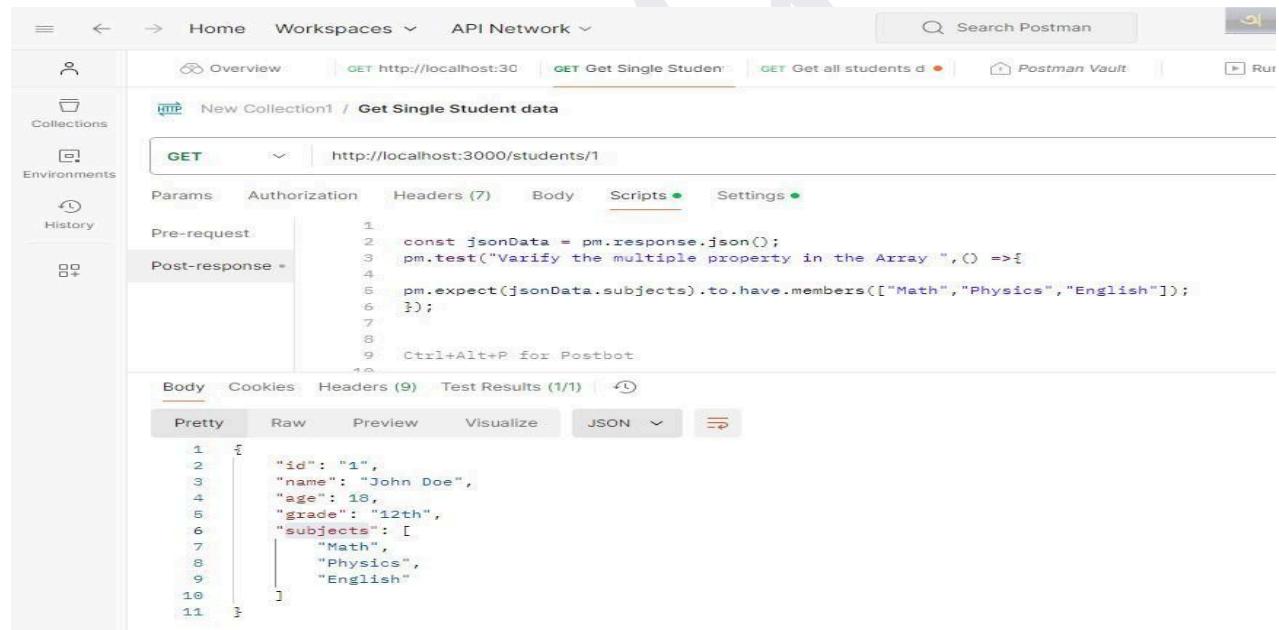
CODE:

```

const jsonData = pm.response.json();

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

Input:



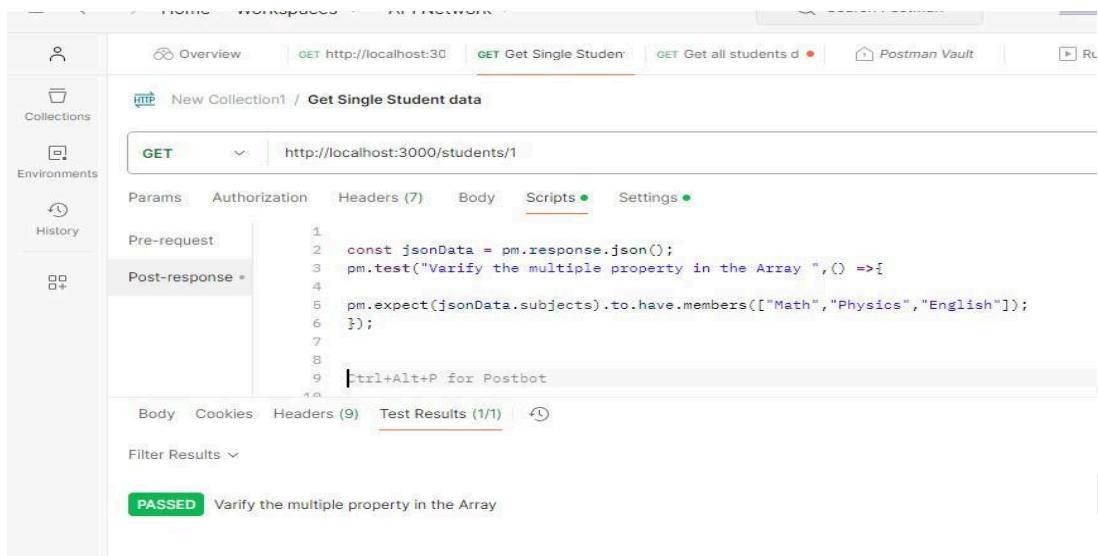
```

1 const jsonData = pm.response.json();
2 pm.test("Varify the multiple property in the Array ",() =>{
3   pm.expect(jsonData.subjects).to.have.members(["Math","Physics","English"]);
4 });
5
6
7
8
9 Ctrl+Alt+P for Postbot
10
  
```

Pretty Raw Preview Visualize JSON

```

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



The screenshot shows the Postman application interface. A collection named 'New Collection1 / Get Single Student data' is selected. A GET request is defined with the URL `http://localhost:3000/students/1`. In the 'Scripts' tab of the request settings, a pre-request script is present:

```

1 const jsonData = pm.response.json();
2 pm.test("Verify the multiple property in the Array ",() =>{
3   pm.expect(jsonData.subjects).to.have.members(["Math","Physics","English"]);
4 });
5
6
7
8
9
  
```

Below the request, the 'Test Results' section shows one test case passed:

PASSED Verify the multiple property in the Array

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 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");

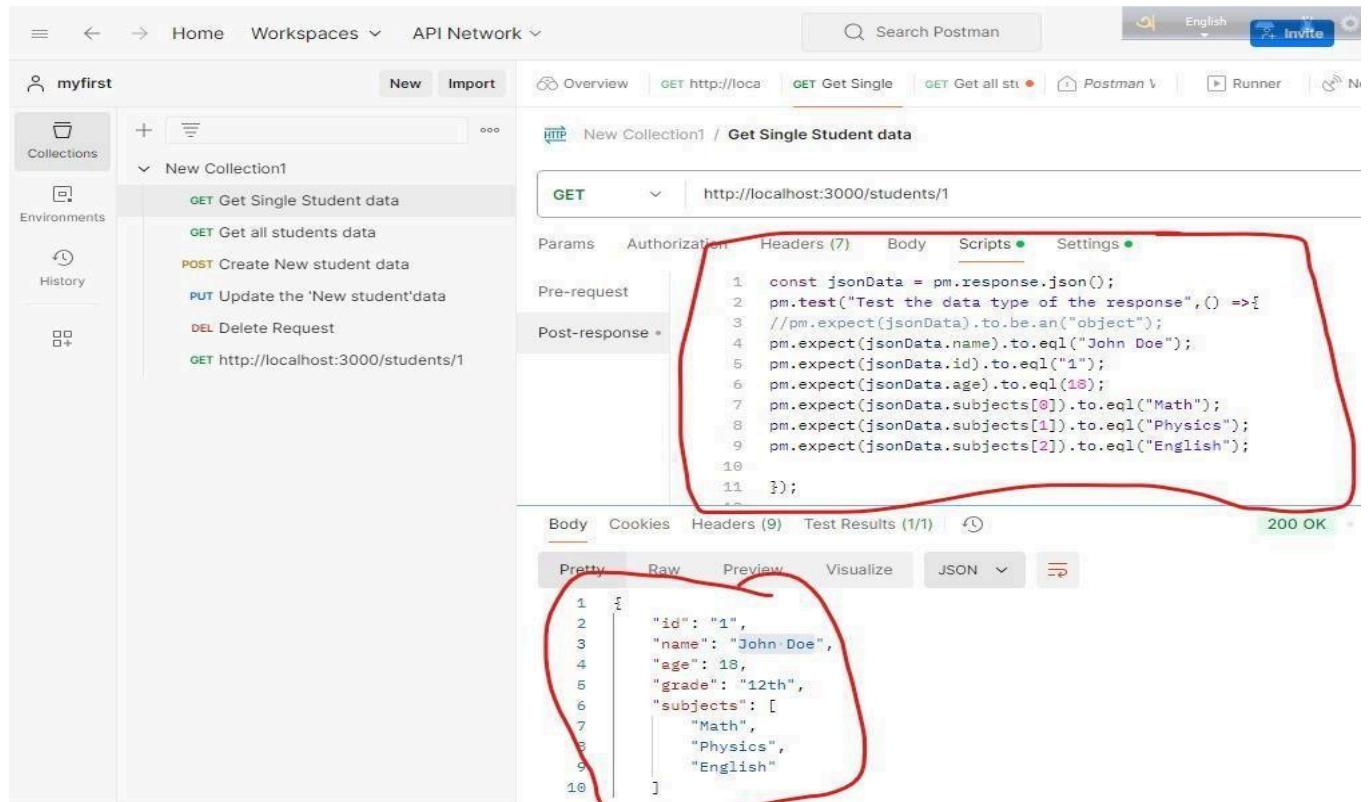
pm.expect(jsonData.age).to.eql(18);

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 collection named 'myfirst'. A POST request to 'http://localhost:3000/students/1' is selected. The 'Post-response' tab contains a JavaScript test script:

```

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 });

```

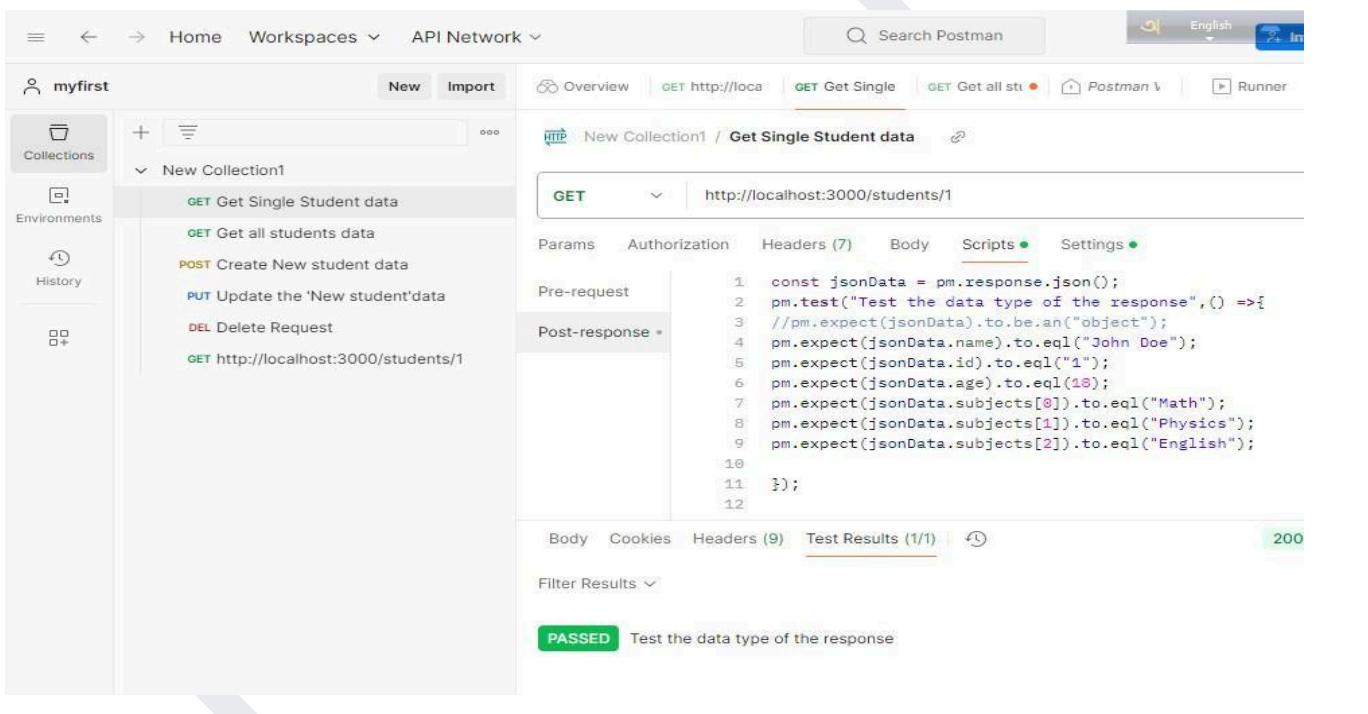
The response body is displayed in 'Pretty' format:

```

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

```

The status bar shows '200 OK'.



The screenshot shows the same Postman interface after the test has run. The status bar at the bottom now displays 'PASSED'.

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?

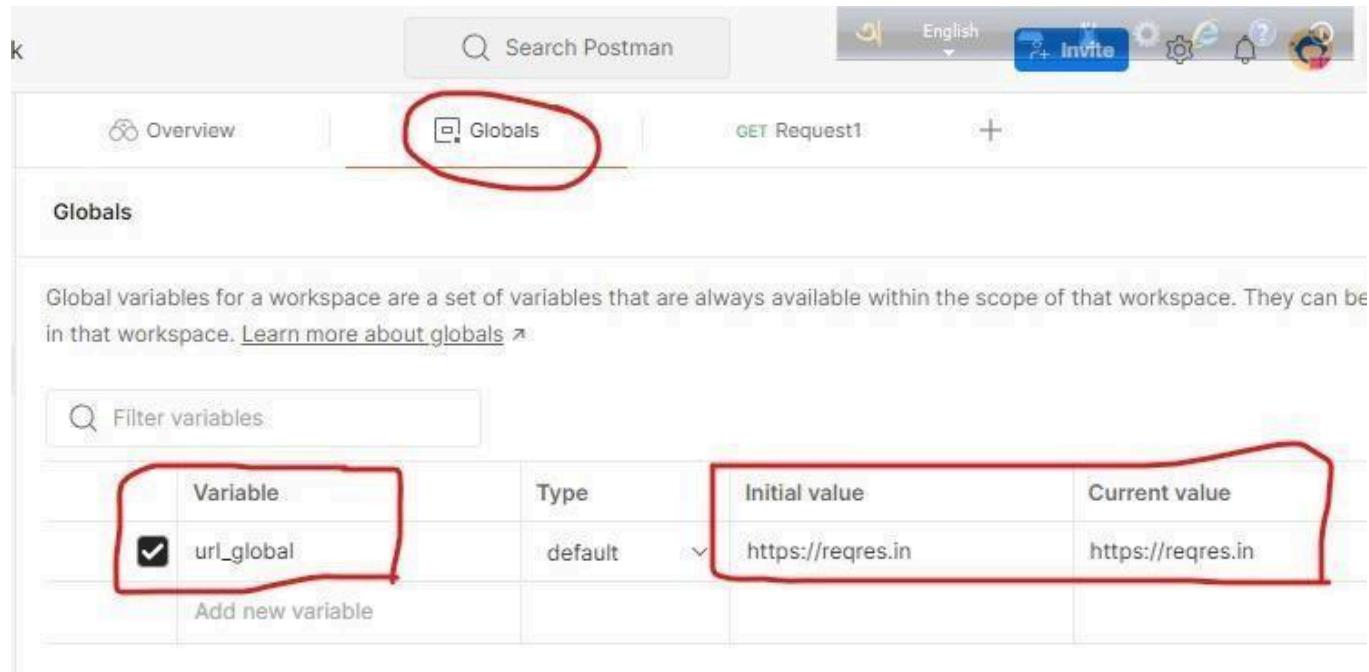
Ans: workspace--□Collection □Request/Folder.

A global variable is accessible throughout the every workspace.

Step-1

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

Step-2

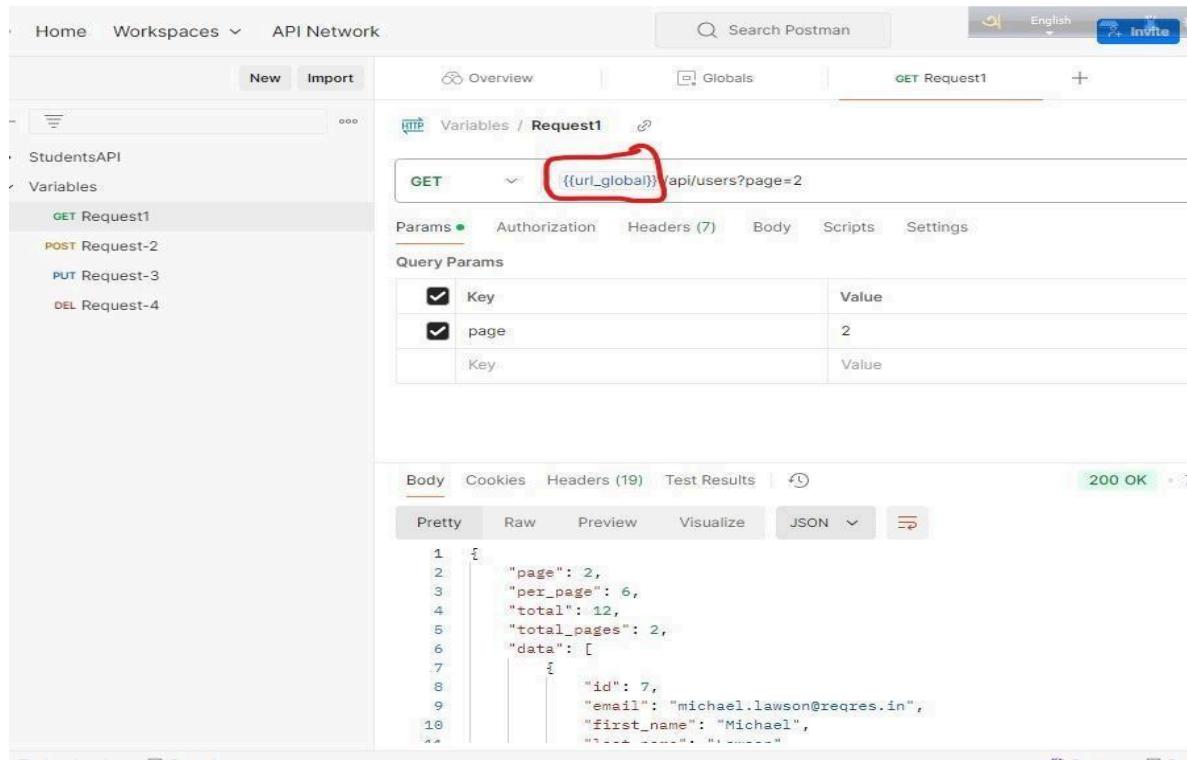


The screenshot shows the Postman interface with the 'Globals' tab selected. A red box highlights the 'Globals' tab in the navigation bar. Another red box highlights the 'url_global' variable in the list below, which has its 'Initial value' and 'Current value' both set to 'https://reqres.in'.

Variable	Type	Initial value	Current value
url_global	default	https://reqres.in	https://reqres.in

Step-3 (Get Request)

Parag



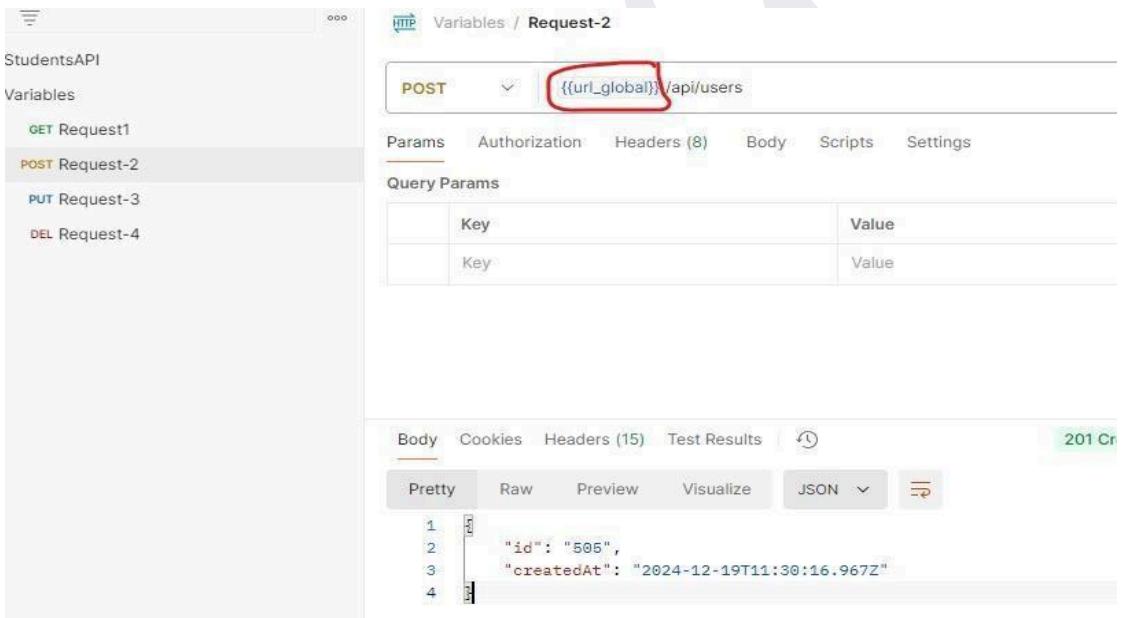
The screenshot shows the Postman interface with a collection named "StudentsAPI". A GET request named "Request1" is selected. The URL is set to `{{url_global}}/api/users?page=2`, with the `url_global` part highlighted by a red box. The "Query Params" section contains a key "page" with the value "2". The response status is "200 OK" and the response body is a JSON object:

```

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  ]
  
```

Step-5 ----(Post Request)

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

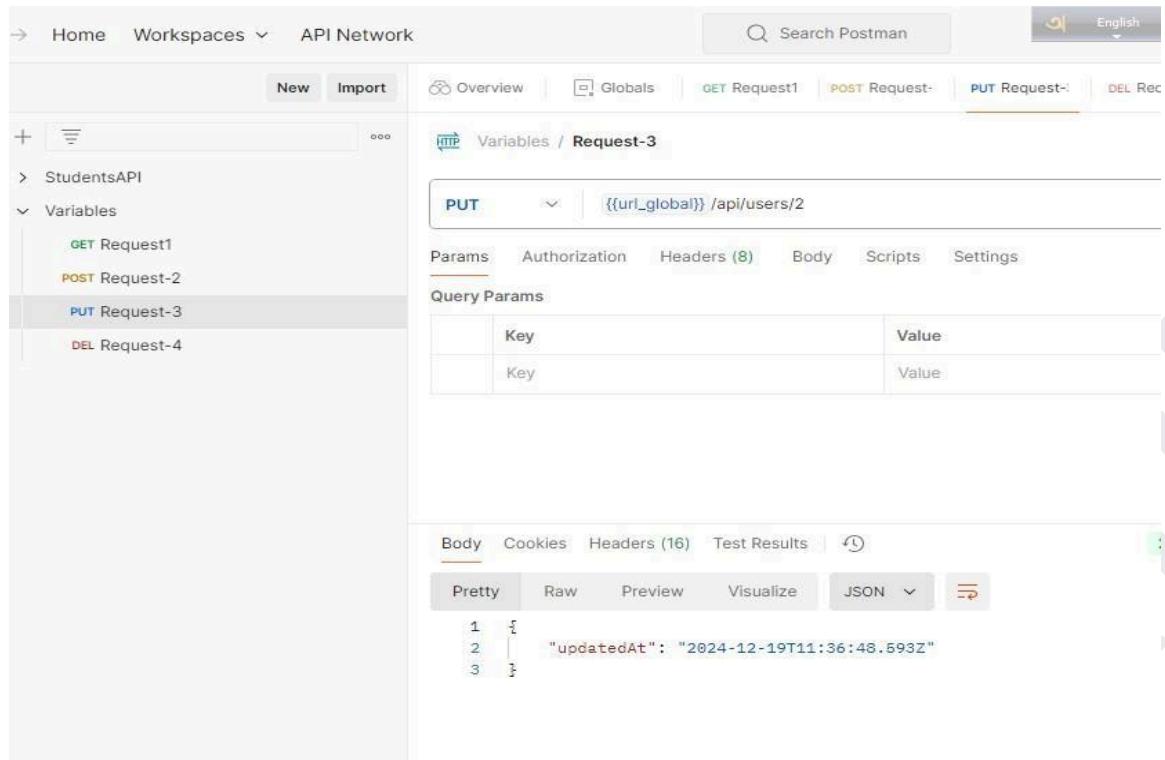


The screenshot shows the Postman interface with the same "StudentsAPI" collection. A POST request named "Request-2" is selected. The URL is set to `{{url_global}}/api/users`, with the `url_global` part highlighted by a red box. The "Query Params" section is empty. The response status is "201 Cr" and the response body is a JSON object:

```

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

Step-6 (Put Request)



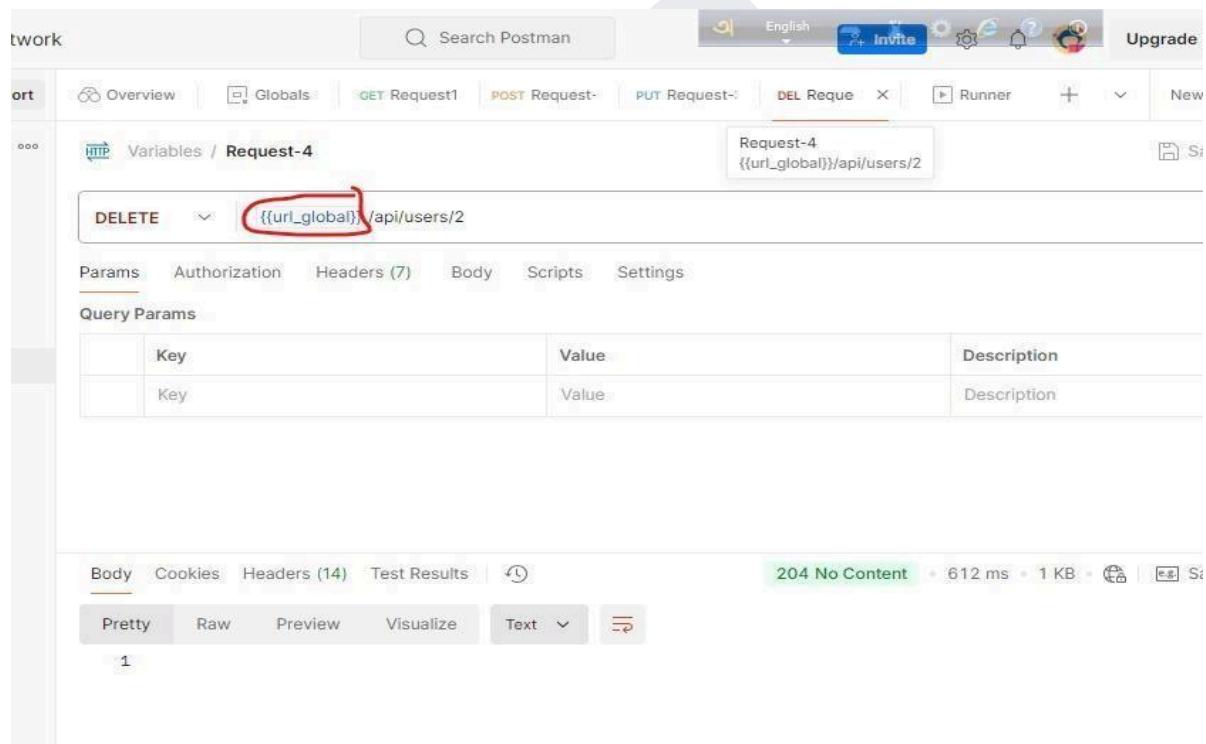
The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Shows a workspace named "StudentsAPI" containing four requests: "GET Request1", "POST Request-2", "PUT Request-3" (selected), and "DEL Request-4".
- Request Preview:** The "PUT" request is selected. The URL is {{url_global}} /api/users/2.
- Request Details:** Headers tab is selected, showing "Headers (8)".
- Body:** The "Pretty" tab is selected, showing the JSON response:


```

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

Step-7 (Delete Request)

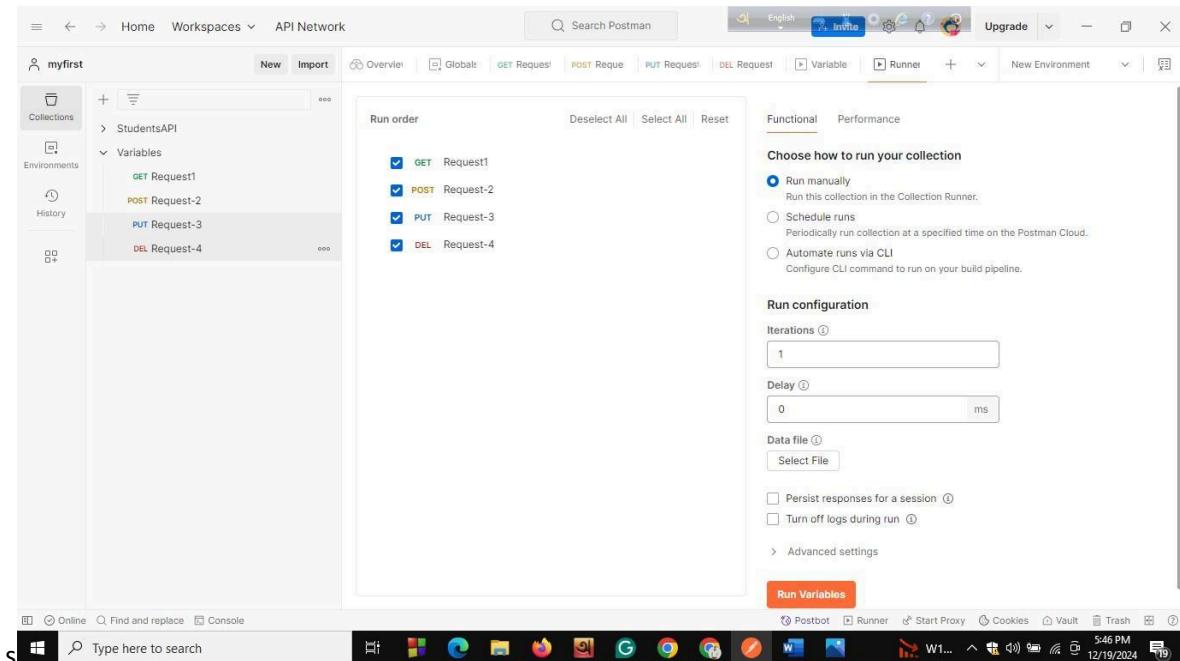


The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Shows a workspace named "twork" containing four requests: "GET Request1", "POST Request-", "PUT Request-", and "DEL Request-4" (selected).
- Request Preview:** The "DELETE" request is selected. The URL is {{url_global}} /api/users/2.
- Request Details:** Headers tab is selected, showing "Headers (7)".
- Body:** The "Text" tab is selected, showing the response:


```
1
```

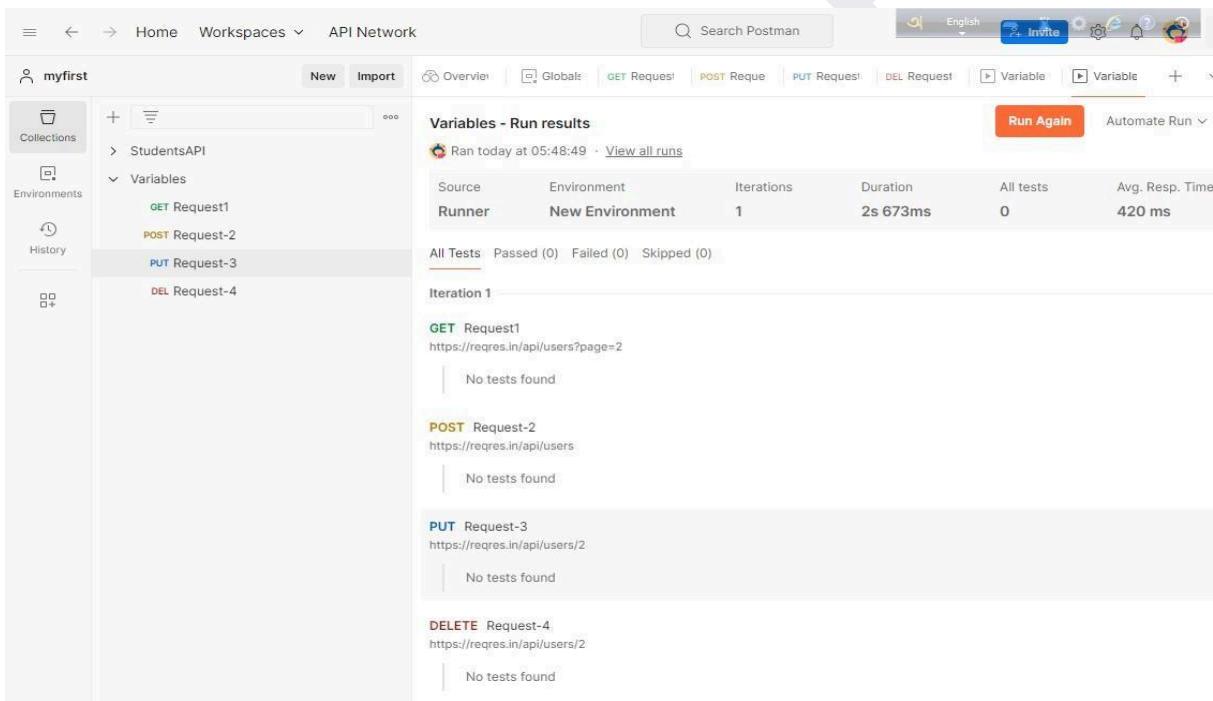
Step-8



The screenshot shows the Postman interface with the following details:

- Collections:** myfirst
- Environments:** None
- Variables:** None
- Requests:**
 - GET Request1
 - POST Request-2
 - PUT Request-3
 - DEL Request-4
- Run order:** All requests are selected.
- Functional Performance:** Functional tab is selected.
- Choose how to run your collection:**
 - Run manually: Run this collection in the Collection Runner.
 - Schedule runs: Periodically run collection at a specified time on the Postman Cloud.
 - Automate runs via CLI: Configure CLI command to run on your build pipeline.
- Run configuration:**
 - Iterations: 1
 - Delay: 0 ms
 - Data file: Select File
 - Persist responses for a session
 - Turn off logs during run
 - Advanced settings
- Run Variables:** None

Step-9

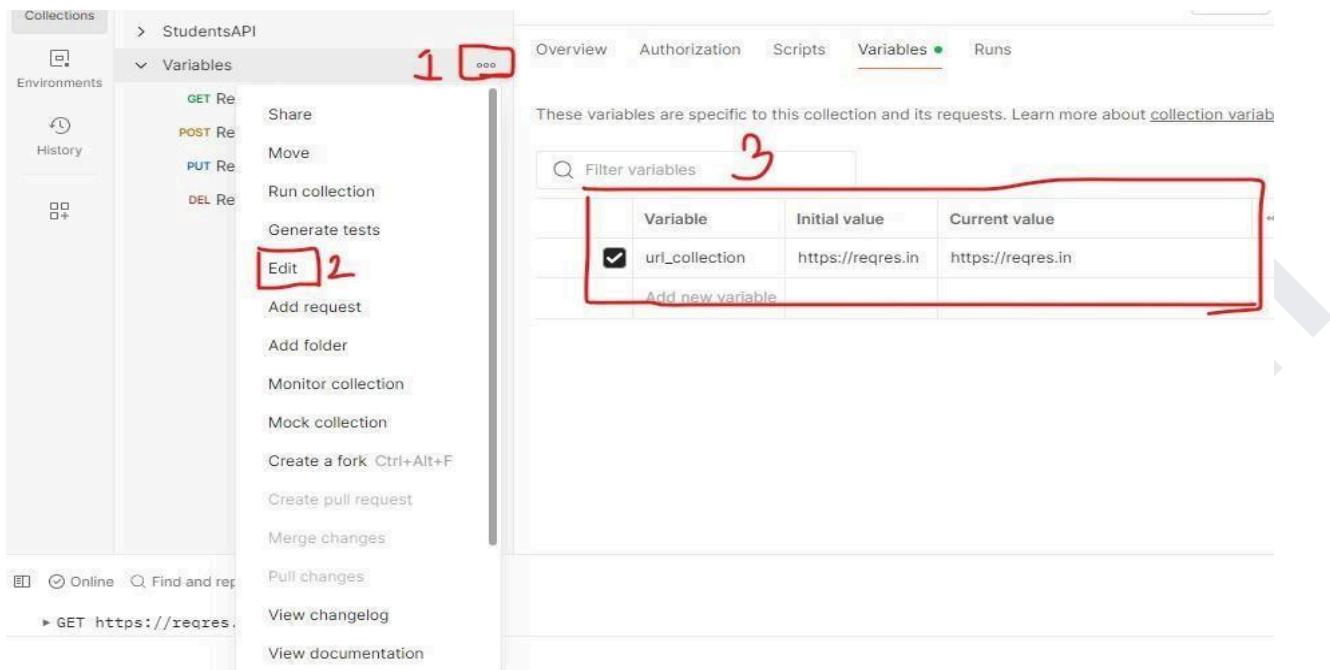


The screenshot shows the Postman interface after a run, displaying the following results:

- Variables - Run results:**
 - Ran today at 05:48:49 · [View all runs](#)
 - Runner:** New Environment
 - Iterations:** 1
 - Duration:** 2s 673ms
 - All tests:** 0
 - Avg. Resp. Time:** 420 ms
- Iterations 1:**
 - GET Request1**
https://reqres.in/api/users?page=2
No tests found
 - POST Request-2**
https://reqres.in/api/users
No tests found
 - PUT Request-3**
https://reqres.in/api/users/2
No tests found
 - DELETE Request-4**
https://reqres.in/api/users/2
No tests found

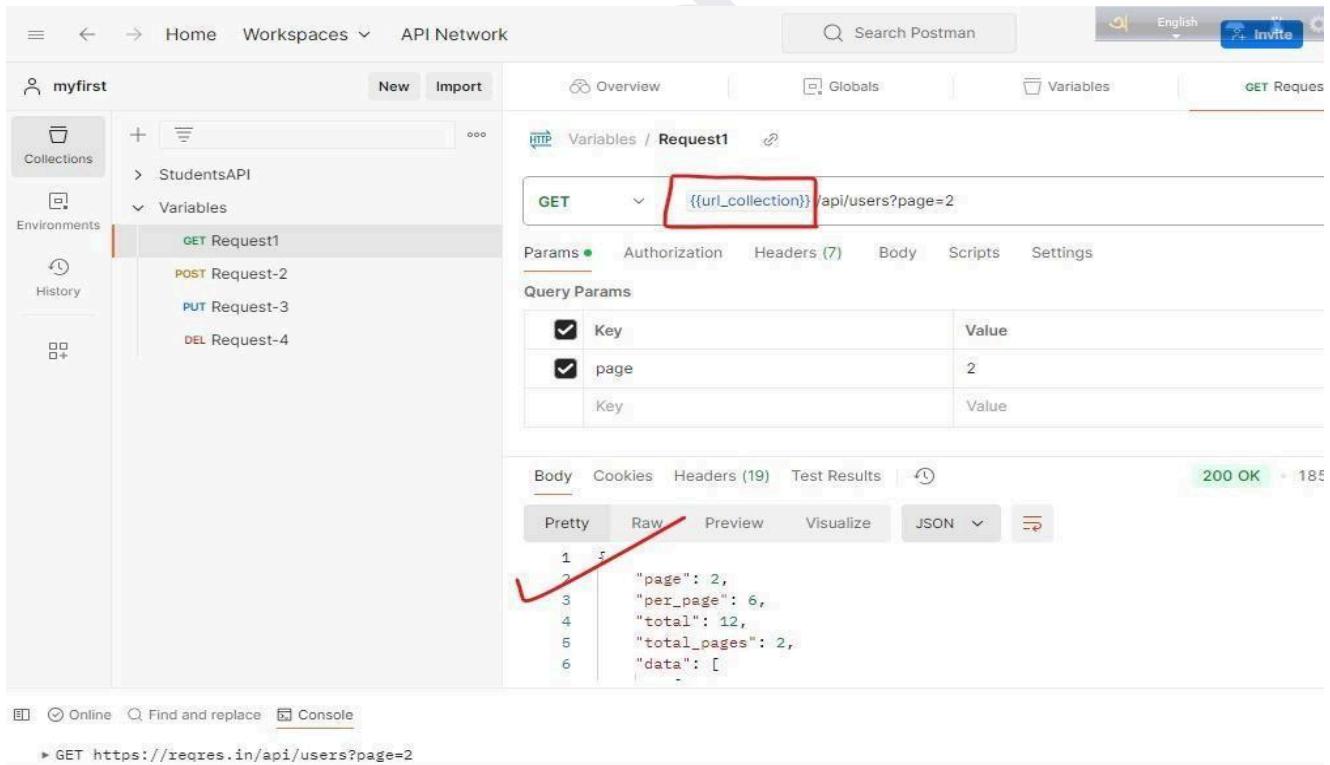
Collection variable : Collection variable is accessible within the Collection among multiple requests.

Step-1



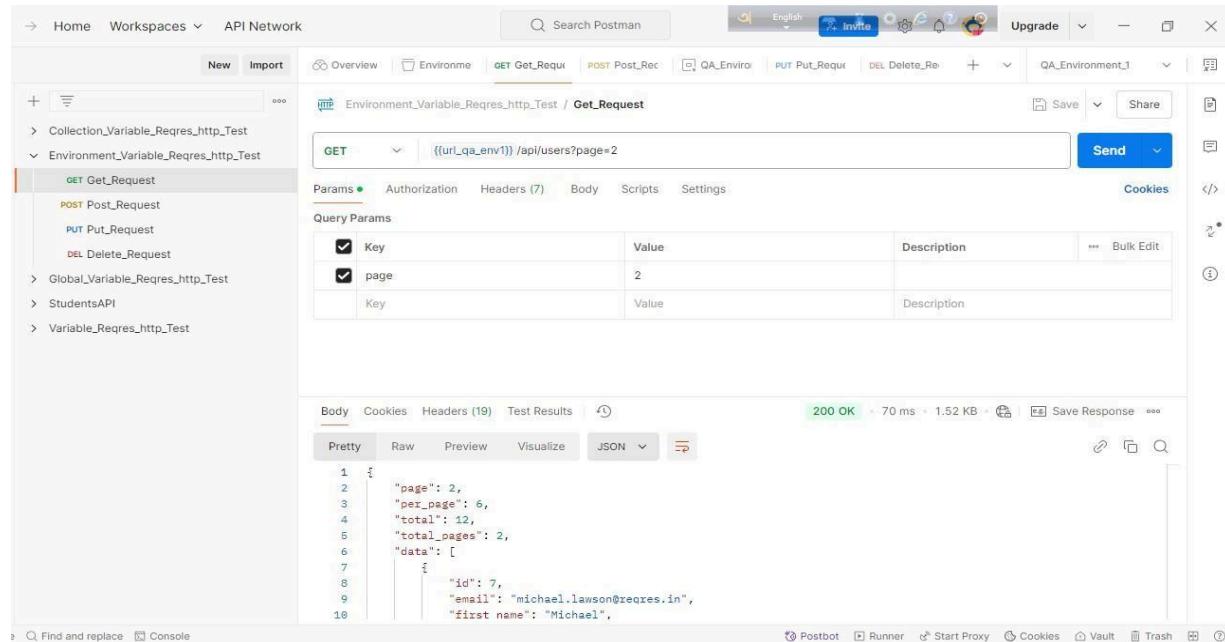
The screenshot shows the Postman interface for a collection named "StudentsAPI". On the left sidebar, under "Variables", there is a red box labeled "1" over the "Variables" section. In the main content area, under the "Variables" tab, there is a table showing a single variable "url_collection" with an initial value of "https://reqres.in" and a current value of "https://reqres.in". A red box labeled "2" is over the "Edit" button in the sidebar, and another red box labeled "3" is over the table header. Below the table, there is a note: "These variables are specific to this collection and its requests. Learn more about [collection variables](#)".

Step-2



The screenshot shows the Postman interface for a workspace named "myfirst". On the left sidebar, under "Variables", there is a red box labeled "1" over the "Variables" section. In the main content area, a "GET Request1" is selected. The URL field contains "{{url_collection}}/api/users?page=2", with a red box labeled "2" over it. The "Raw" tab in the response preview is selected, showing a JSON response with page 2 data. A red checkmark is drawn next to the "Raw" tab. The response status is "200 OK" with 185 bytes.

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



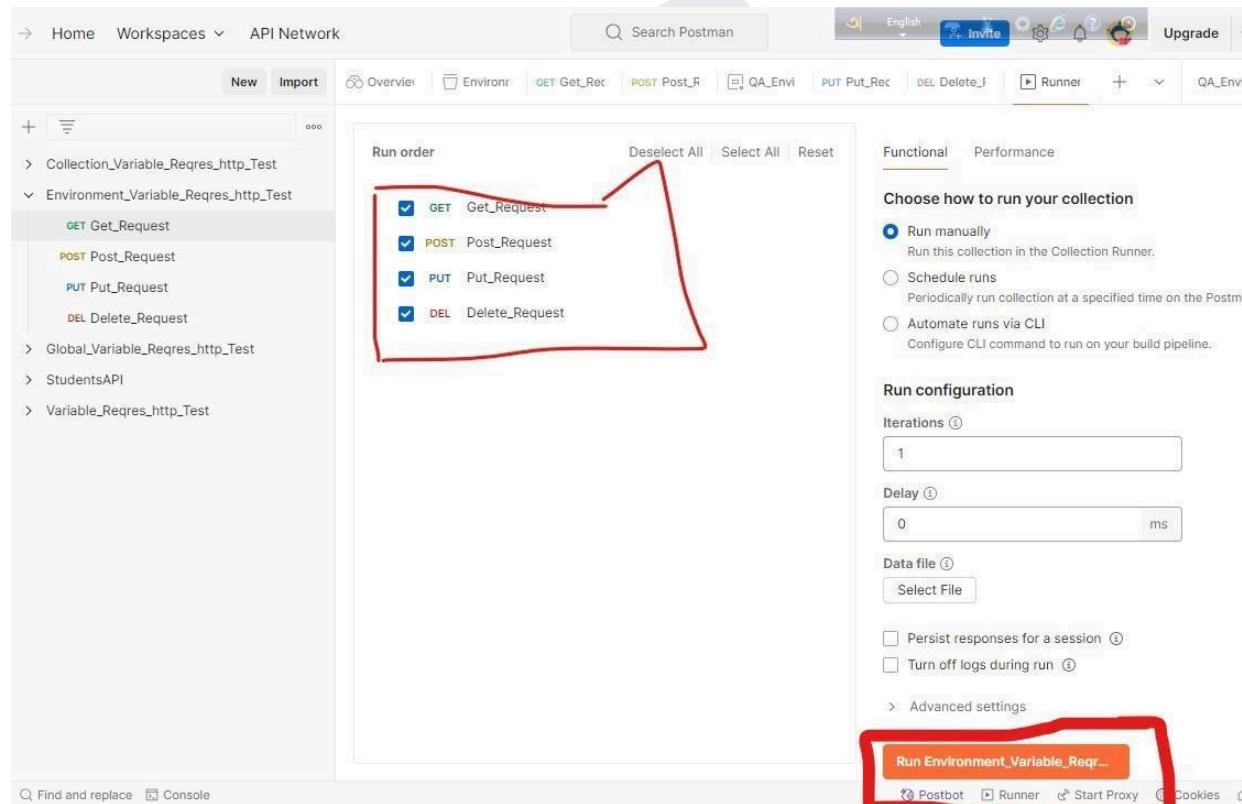
The screenshot shows the Postman interface with a collection named "Environment_Variable_Reqres_http_Test". A specific GET request is selected, with the URL being `GET {{url_qa_env1}} /api/users?page=2`. The "Params" tab shows a "page" parameter set to 2. The "Body" tab displays the JSON response:

```

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     }
12   ]
13 }
```

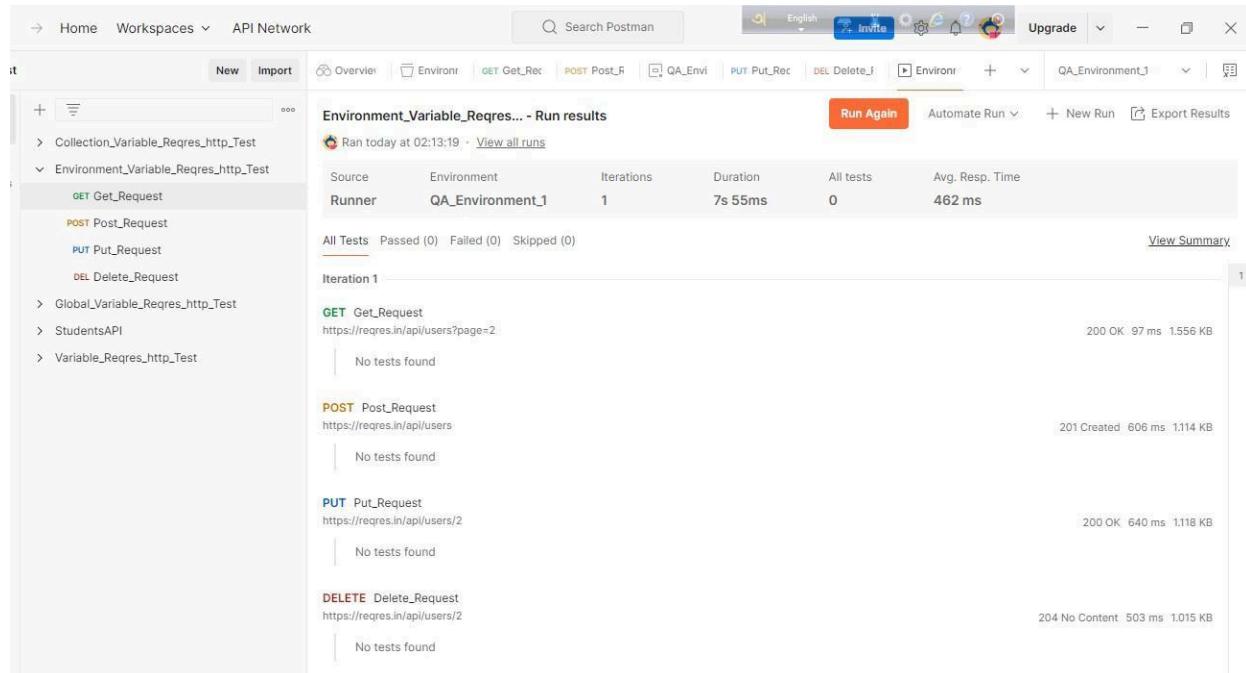
Run the whole collection of Environment Variable:

Step-1



The screenshot shows the Postman interface with the same collection selected. In the "Run order" section, four requests are checked: GET Get_Request, POST Post_Request, PUT Put_Request, and DEL Delete_Request. On the right, the "Choose how to run your collection" section is visible, with "Run manually" selected. At the bottom, the "Run configuration" section includes fields for Iterations (1), Delay (0 ms), and Data file (Select File). The "Run Environment_Variable_Reqr..." button is highlighted with a red box.

Step-2



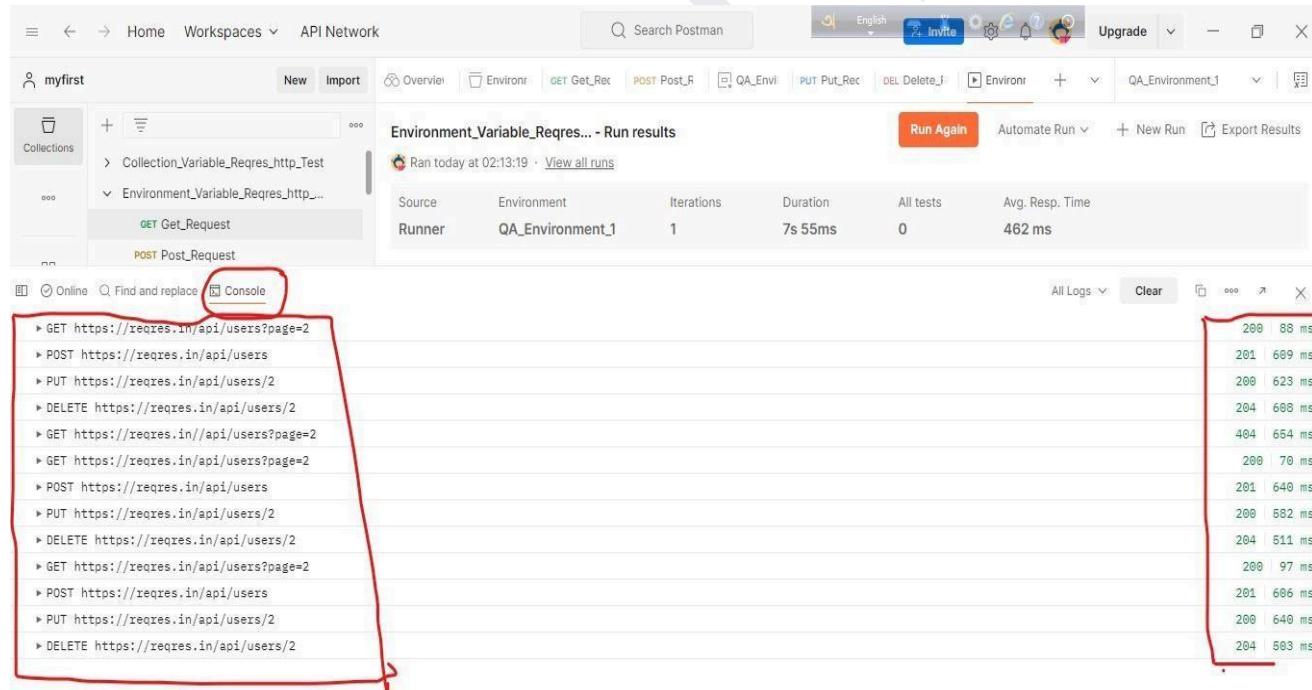
Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	QA_Environment_1	1	7s 55ms	0	462 ms

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

Iteration 1

- GET Get_Request**
https://reqres.in/api/users?page=2
No tests found
- POST Post_Request**
https://reqres.in/api/users
No tests found
- PUT Put_Request**
https://reqres.in/api/users/2
No tests found
- DELETE Delete_Request**
https://reqres.in/api/users/2
No tests found

Console result



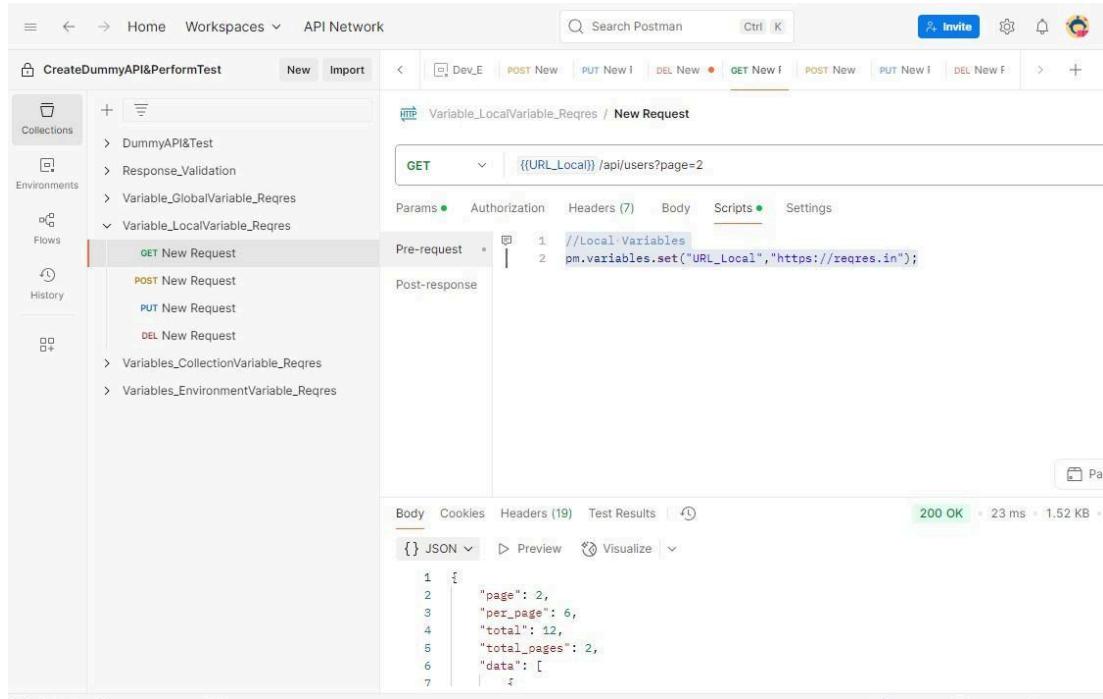
Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	QA_Environment_1	1	7s 55ms	0	462 ms

Find and replace [Console](#)

```

▶ GET https://reqres.in/api/users?page=2
▶ POST https://reqres.in/api/users
▶ PUT https://reqres.in/api/users/2
▶ DELETE https://reqres.in/api/users/2
▶ GET https://reqres.in/api/users?page=2
▶ GET https://reqres.in/api/users?page=2
▶ POST https://reqres.in/api/users
▶ PUT https://reqres.in/api/users/2
▶ DELETE https://reqres.in/api/users/2
▶ GET https://reqres.in/api/users?page=2
▶ POST https://reqres.in/api/users
▶ PUT https://reqres.in/api/users/2
▶ DELETE https://reqres.in/api/users/2
  
```

Local Variables: Accessible only within the request.



The screenshot shows the Postman interface with a collection named 'CreateDummyAPI&PerformTest'. A new request titled 'Variable_LocalVariable_Reqres / New Request' is selected. In the 'Scripts' tab of the request settings, a 'Pre-request' script is defined with the following code:

```

1 //Local Variables
2 pm.variables.set("URL_Local","https://reqres.in");

```

The 'Body' tab of the request shows a GET request to `https://reqres.in/api/users?page=2`. The response body is displayed as JSON:

```

1 {
2   "page": 2,
3   "per_page": 6,
4   "total": 12,
5   "total_pages": 2,
6   "data": [
7     {

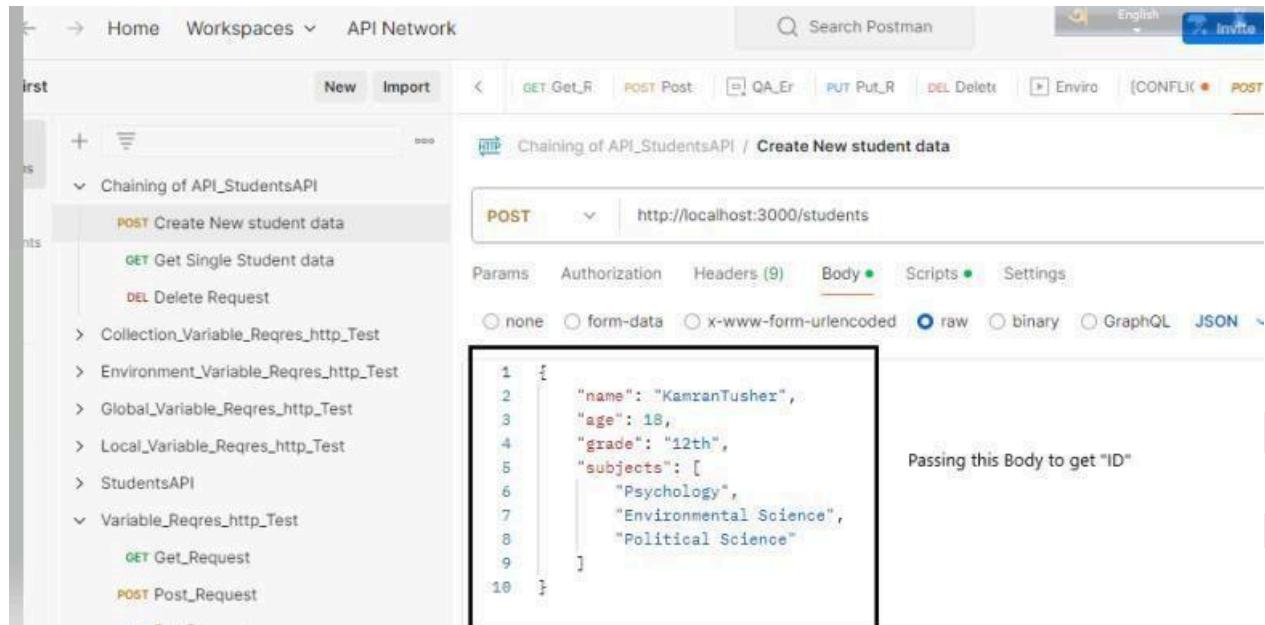
```

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



The screenshot shows the Postman application interface. On the left, there's a sidebar with a tree view of API collections. The current collection is 'Chaining of API_StudentsAPI', which contains a 'POST Create New student data' request. The main workspace shows a POST request to 'http://localhost:3000/students'. The 'Body' tab is selected, showing a JSON payload:

```

1  {
2   "name": "KamranTusher",
3   "age": 18,
4   "grade": "12th",
5   "subjects": [
6     "Psychology",
7     "Environmental Science",
8     "Political Science"
9   ]
10 }

```

A tooltip 'Passing this Body to get "ID"' is visible next to the JSON code.

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: cf842adb5472f7db0196c587b55a74dfcccd50e82e0f361ab59326843207adddf

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:

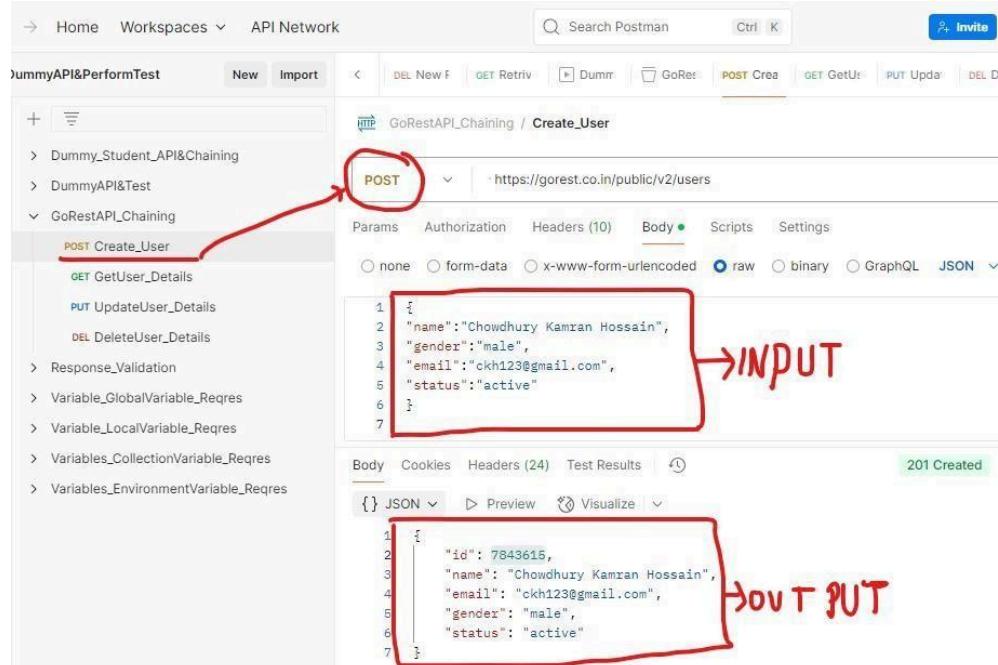
Step-1: Create a Collection called “GoRestAPI_Chaining

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.



POST

INPUT

OUTPUT

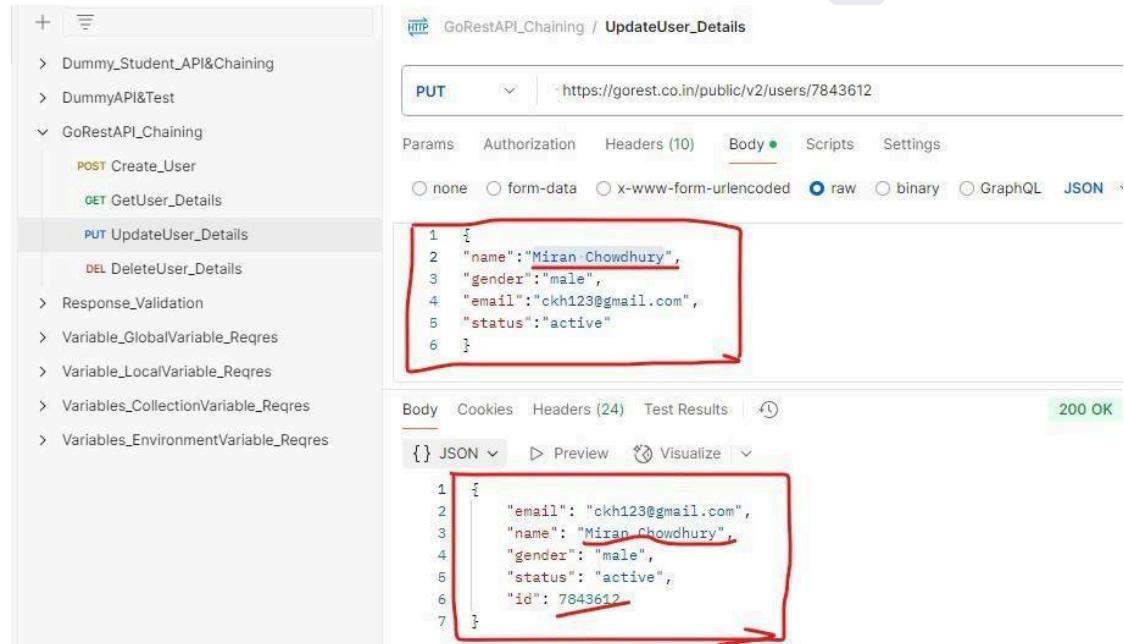
```

1 {
2   "name": "Chowdhury Kamran Hossain",
3   "gender": "male",
4   "email": "ckh123@gmail.com",
5   "status": "active"
6 }
7
  
```

```

1 {
2   "id": 7843615,
3   "name": "Chowdhury Kamran Hossain",
4   "email": "ckh123@gmail.com",
5   "gender": "male",
6   "status": "active"
7 }
  
```

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



PUT

PUT

```

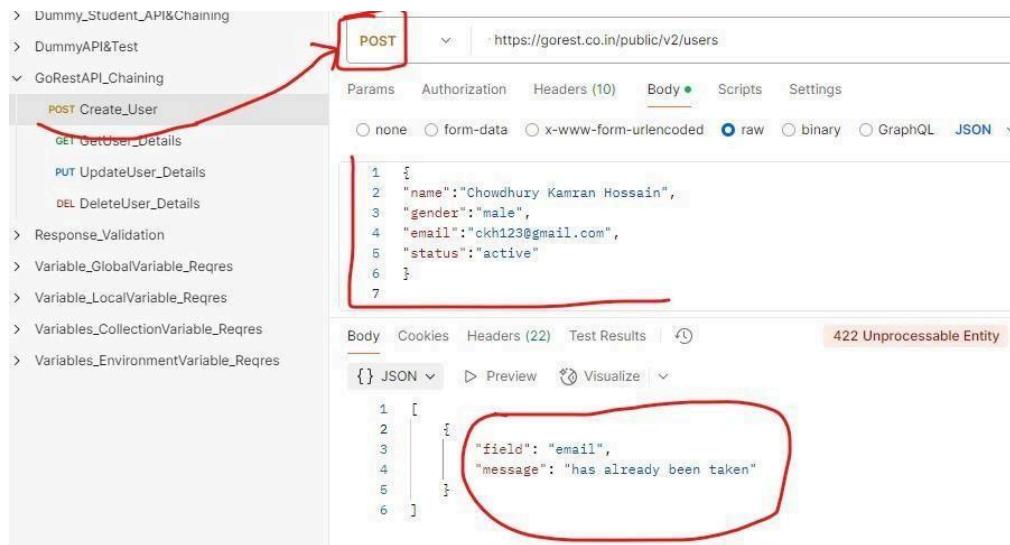
1 {
2   "name": "Miran Chowdhury",
3   "gender": "male",
4   "email": "ckh123@gmail.com",
5   "status": "active"
6 }
  
```

```

1 {
2   "email": "ckh123@gmail.com",
3   "name": "Miran Chowdhury",
4   "gender": "male",
5   "status": "active",
6   "id": 7843612
7 }
  
```

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

Image:



The screenshot shows a POST request to <https://gorest.co.in/public/v2/users>. The request body is a JSON object:

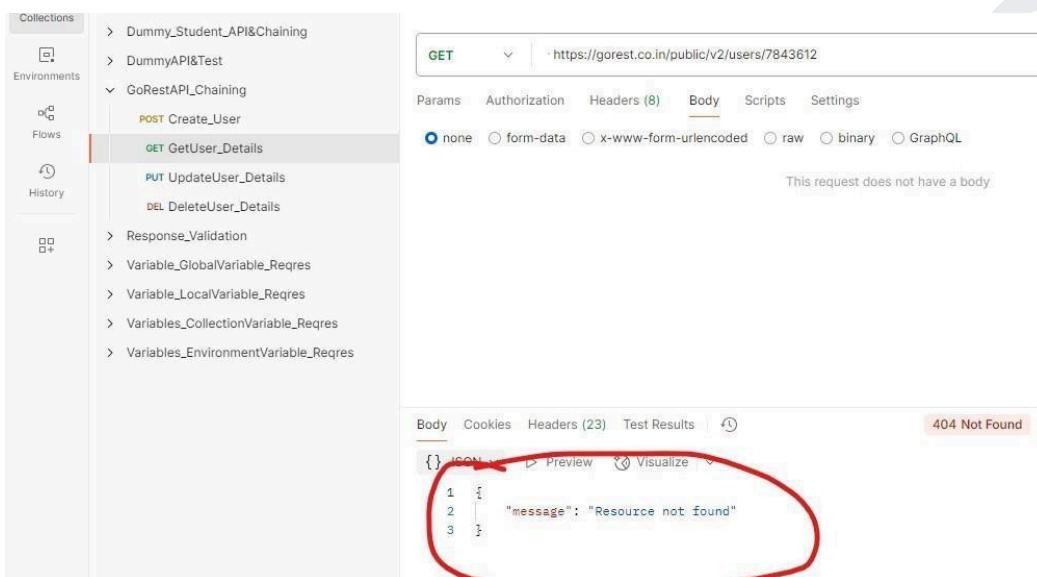
```

1 {
2   "name": "Chowdhury Kamran Hossain",
3   "gender": "male",
4   "email": "ckh123@gmail.com",
5   "status": "active"
6 }
7
  
```

The response status is 422 Unprocessable Entity. The response body is:

```

1 [
2   {
3     "field": "email",
4     "message": "has already been taken"
5   }
6 ]
  
```



The screenshot shows a GET request to <https://gorest.co.in/public/v2/users/7843612>. The response status is 404 Not Found. The response body is:

```

1 {
2   "message": "Resource not found"
3 }
  
```

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

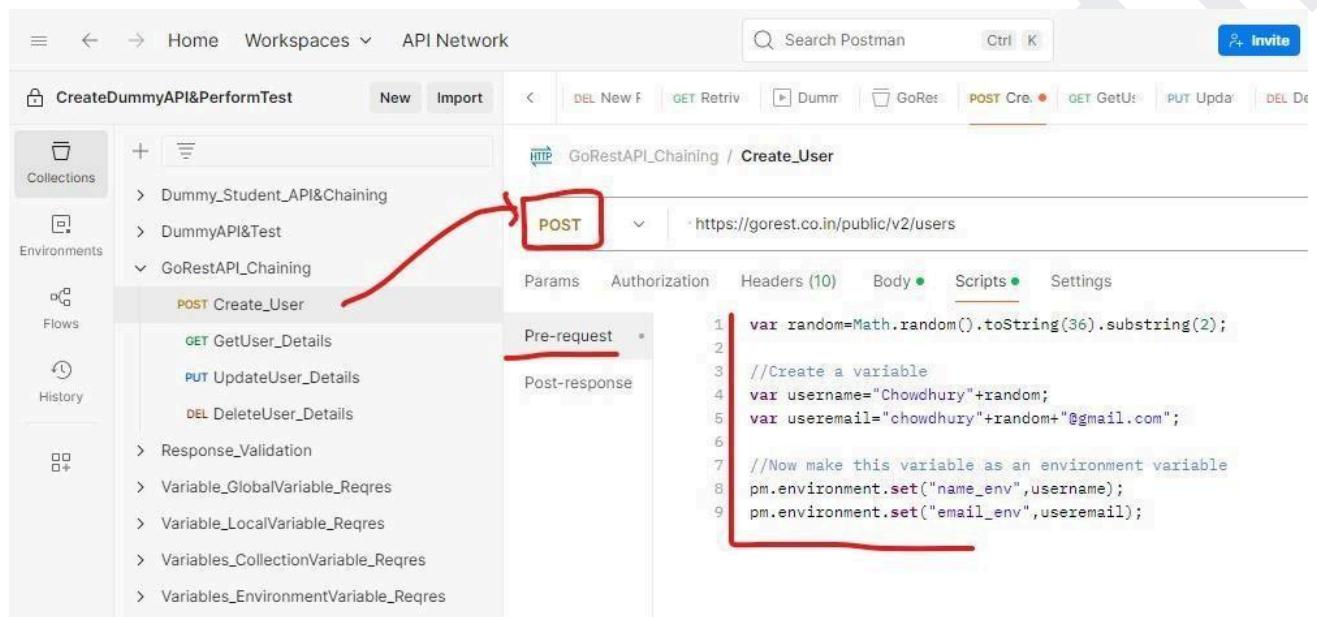
code:

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

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

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

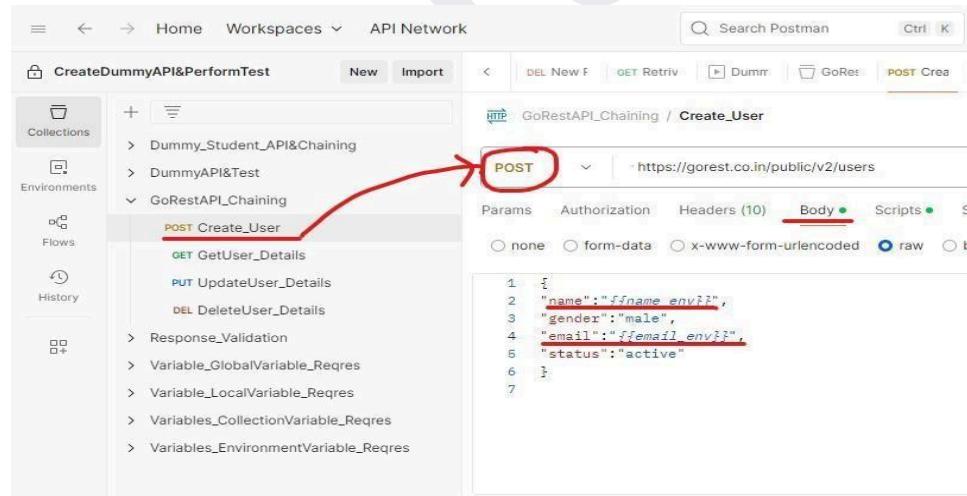
Input:



The screenshot shows the Postman interface with a collection named "CreateDummyAPI&PerformTest". In the left sidebar, under "GoRestAPI_Chaining", there is a "POST Create_User" request. The "Pre-request" tab is selected, and its content is highlighted with a red box. The code in the "Pre-request" tab is:

```
1 var random=Math.random().toString(36).substring(2);
2 //Create a variable
3 var username="Chowdhury"+random;
4 var useremail="chowdhury"+random+"@gmail.com";
5
6 //Now make this variable as an environment variable
7 pm.environment.set("name_env",username);
8 pm.environment.set("email_env",useremail);
```

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



The screenshot shows the Postman interface with the same collection and request setup as the previous screenshot. The "Body" tab is highlighted with a red box, and its content is:

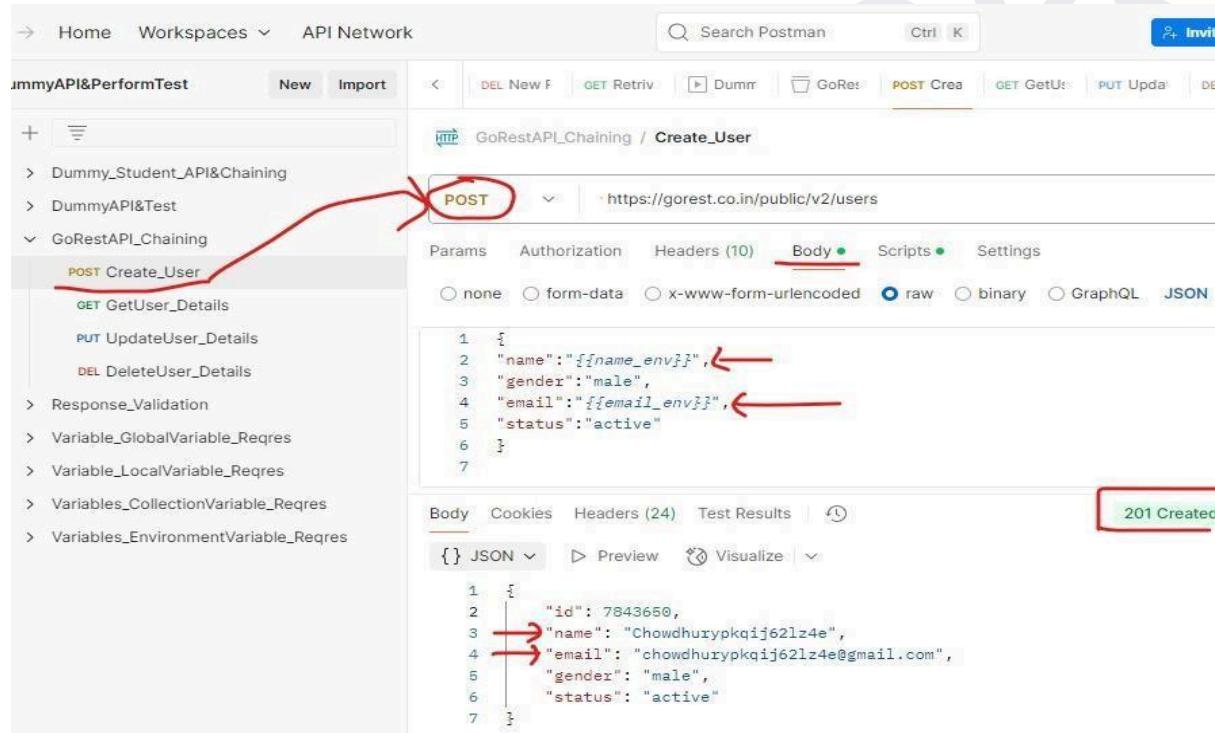
```
1 {
2   "name": "$name_env",
3   "gender": "male",
4   "email": "$email_env",
5   "status": "active"
6 }
```

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:



The screenshot shows the Postman interface with the following details:

- Collection:** JimmyAPI&PerformTest
- Request Type:** POST
- URL:** https://gorest.co.in/public/v2/users
- Body (raw JSON):**

```

1  {
2    "name": "{{name_env}}",
3    "gender": "male",
4    "email": "{{email_env}}",
5    "status": "active"
6  }

```
- Status:** 201 Created
- Response Body (JSON):**

```

1  {
2    "id": 7843650,
3    "name": "Chowdhurypkqij62lz4e",
4    "email": "chowdhurypkqij62lz4e@gmail.com",
5    "gender": "male",
6    "status": "active"
7  }

```

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 jsonData= 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

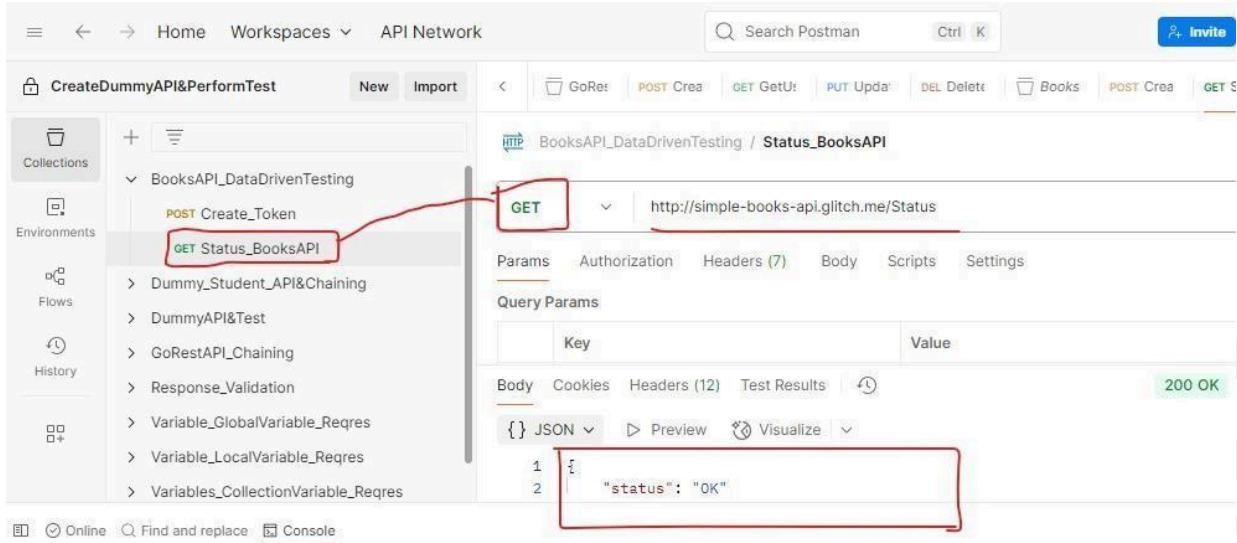
this Url :

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

End Point: /Status

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

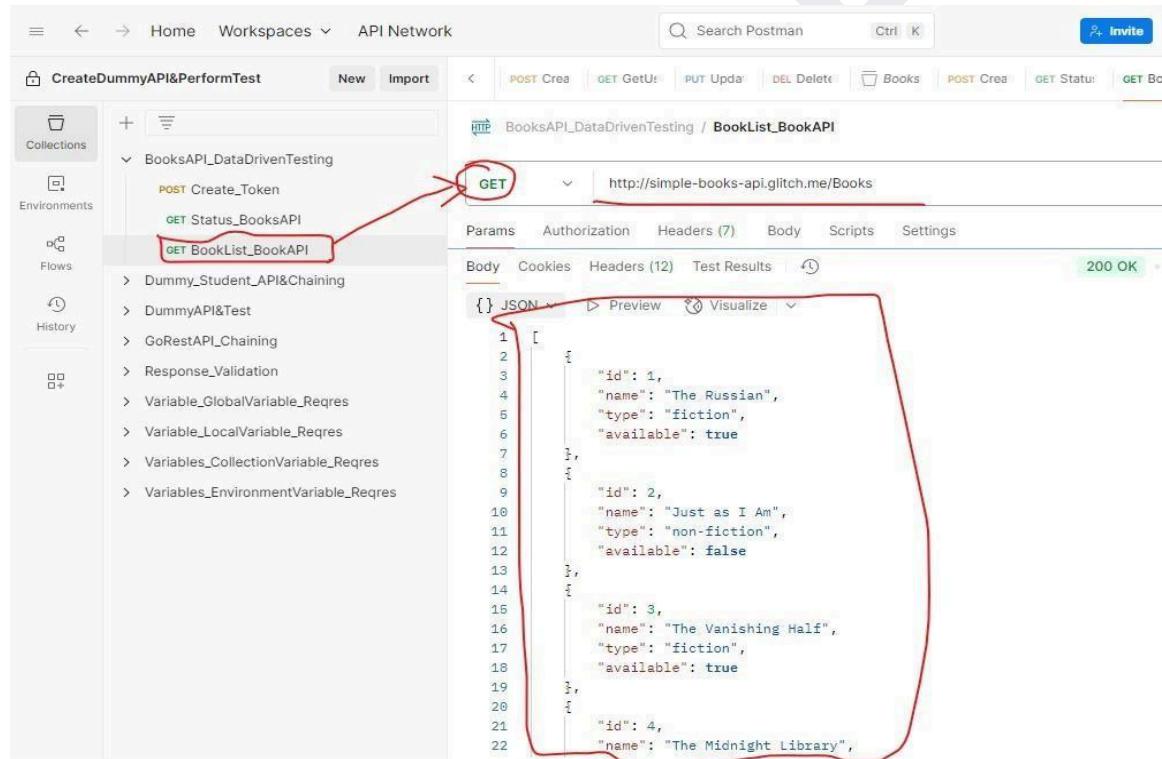
Image:



The screenshot shows the Postman interface with a collection named "CreateDummyAPI&PerformTest". A red box highlights the "GET Status_BooksAPI" request under the "BooksAPI_DataDrivenTesting" collection. The request URL is `http://simple-books-api.glitch.me>Status`. The response status is 200 OK, and the response body is a JSON object: `{"status": "OK"}`.

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

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



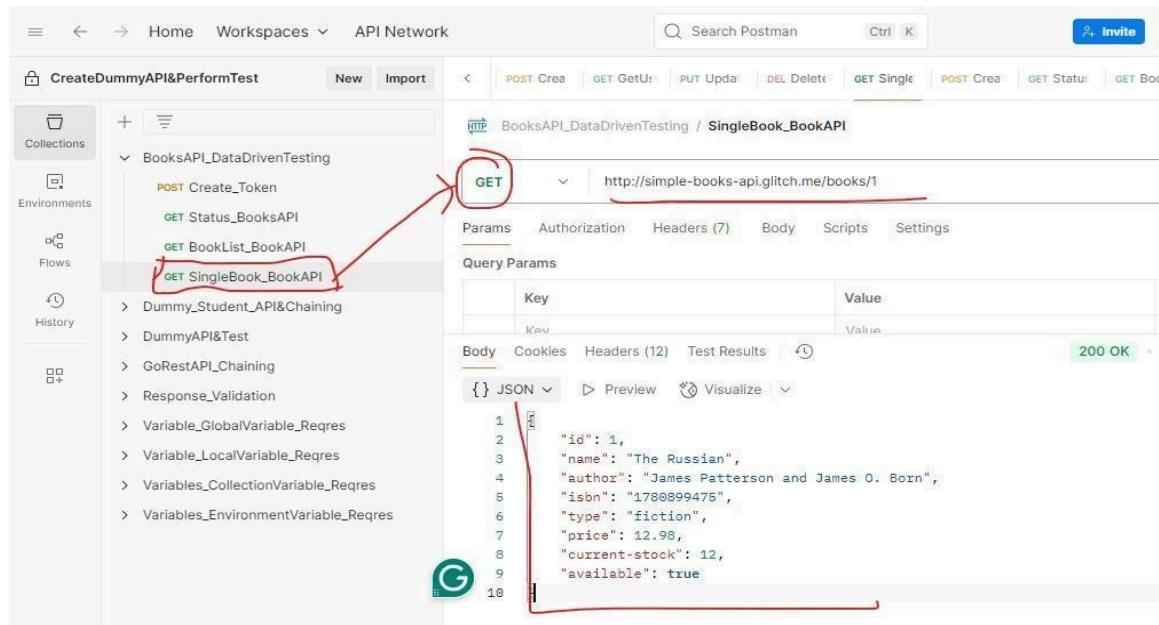
The screenshot shows the Postman interface with the same collection "CreateDummyAPI&PerformTest". A red box highlights the "GET BookList_BookAPI" request under the "BooksAPI_DataDrivenTesting" collection. The request URL is `http://simple-books-api.glitch.me/Books`. The response status is 200 OK, and the response body is a JSON array of four book objects:

```

[{"id": 1, "name": "The Russian", "type": "fiction", "available": true}, {"id": 2, "name": "Just as I Am", "type": "non-fiction", "available": false}, {"id": 3, "name": "The Vanishing Half", "type": "fiction", "available": true}, {"id": 4, "name": "The Midnight Library", "type": "fiction", "available": true}
  
```

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

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



The screenshot shows the Postman interface. On the left, there's a sidebar with collections, environments, flows, and history. A collection named 'BooksAPI_DataDrivenTesting' is expanded, showing several API endpoints: 'POST Create_Token', 'GET Status_BooksAPI', 'GET BookList_BookAPI', 'GET SingleBook_BookAPI' (which is highlighted with a red circle), and others like 'Dummy_Student_API&Chaining', 'DummyAPI&Test', etc. The main panel shows a 'SingleBook_BookAPI' endpoint with a 'GET' method selected. The URL is set to 'http://simple-books-api.glitch.me/books/1'. The 'Body' tab is selected, showing a JSON response with the following content:

```

1   {
2     "id": 1,
3     "name": "The Russian",
4     "author": "James Patterson and James O. Born",
5     "isbn": "1780899478",
6     "type": "fiction",
7     "price": 12.98,
8     "current-stock": 12,
9     "available": true
10
  
```

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/>

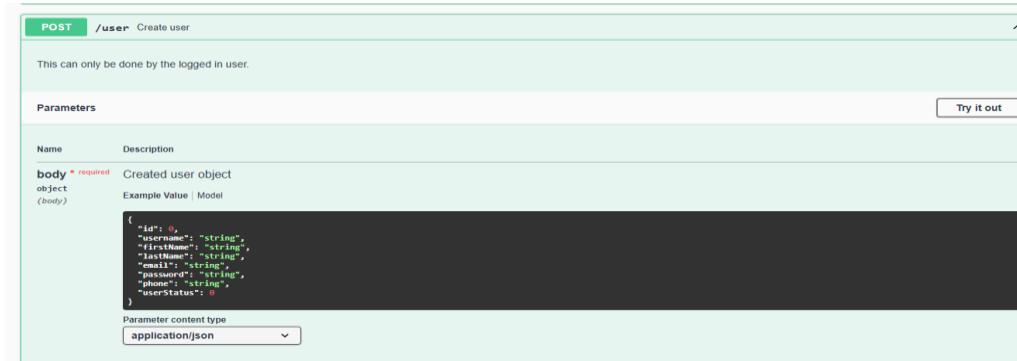
User model provides the responses in the Json format.Different API request in User model



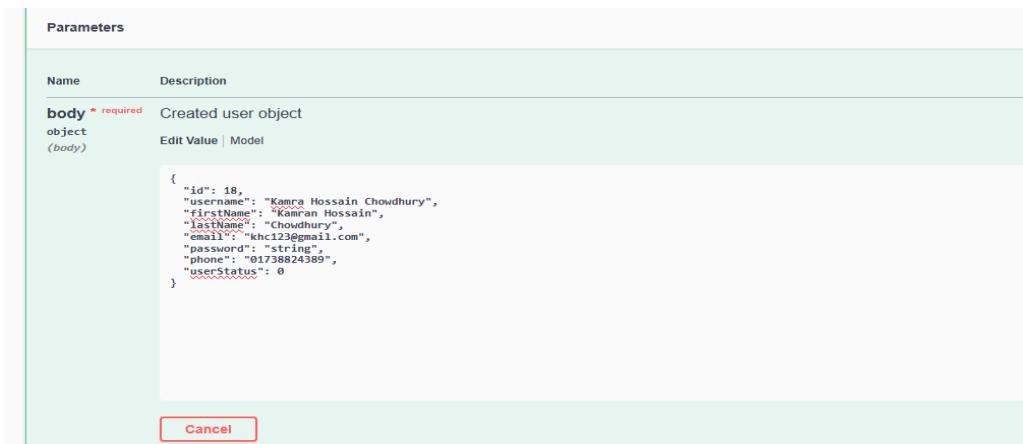
The screenshot shows the Petstore Swagger UI for the 'user' model. It lists the following API endpoints:

- 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 (This endpoint is highlighted with a red border)
- 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

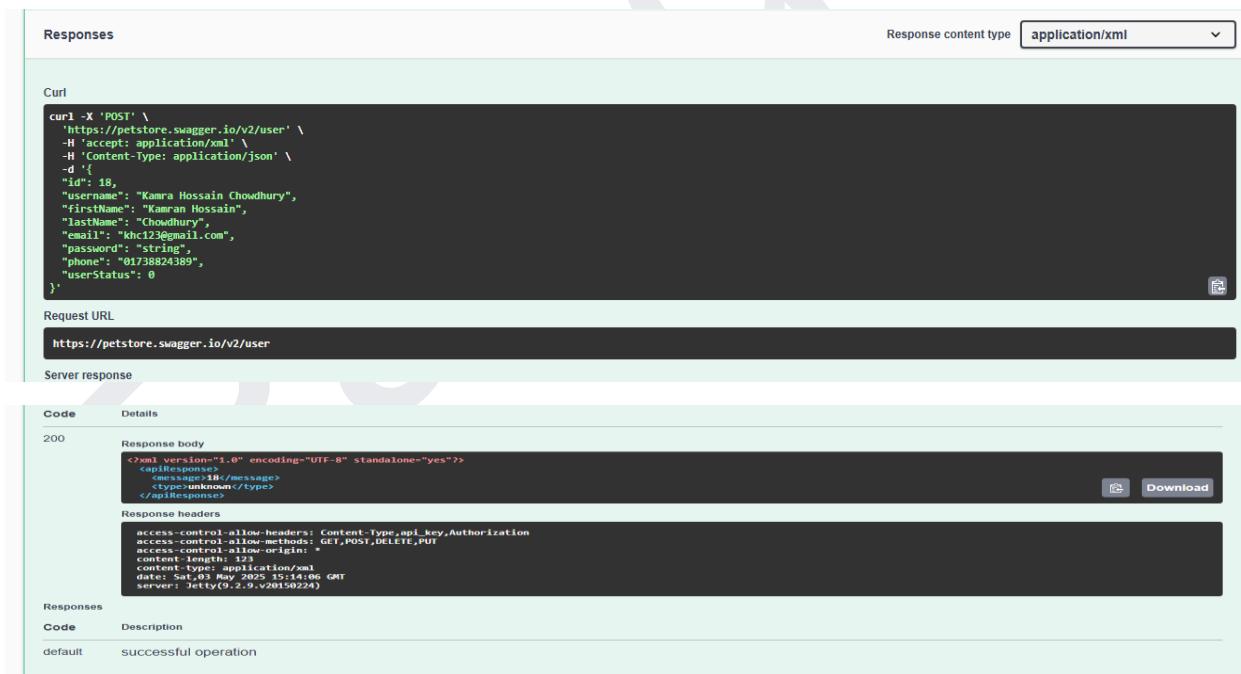
Now click on **Post** request, which is in the given screenshot. Then this page will appear



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



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



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