

API Testing

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

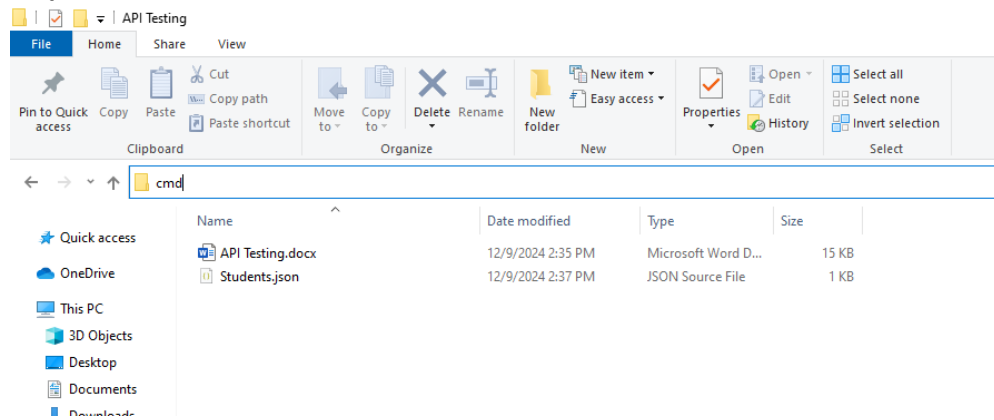
```
{
  "students":[
    {
      "id": 1,
      "name": "John Doe",
      "age": 18,
      "grade": "12th",
      "subjects": [
        "Math",
        "Physics",
        "English"
      ]
    },
    {
      "id": 2,
      "name": "Jane Smith",
      "age": 17,
      "grade": "11th",
      "subjects":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

How to create own API ?

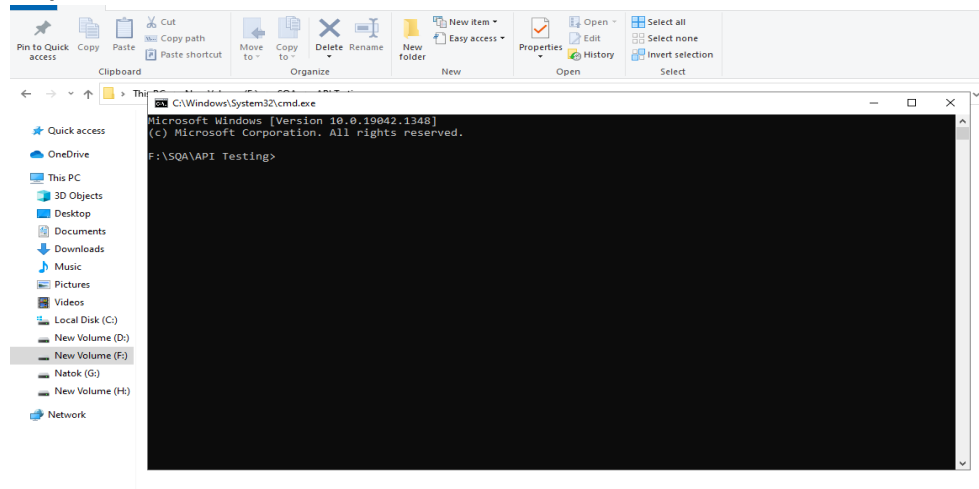
Task no-1

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

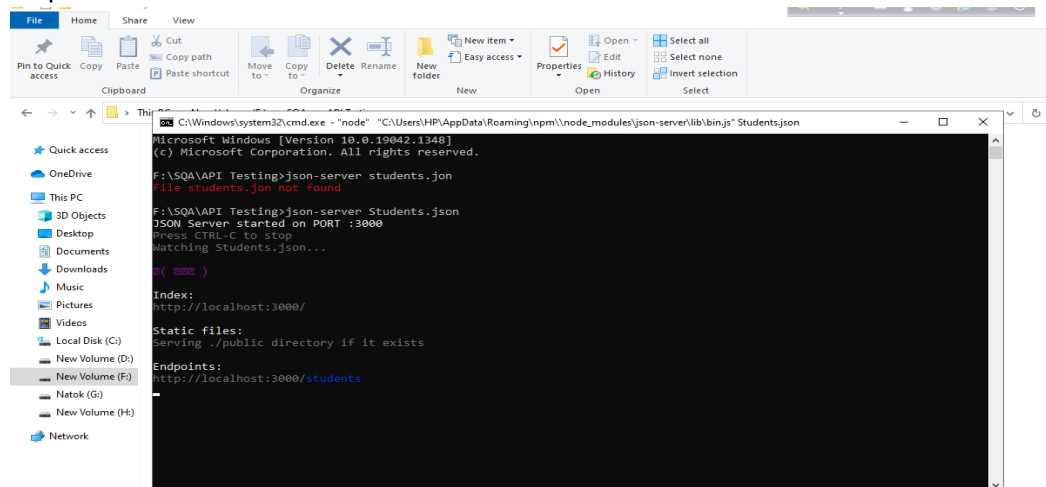
Step-1



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 workspace named 'myfirst'. A new collection 'New Collection1' is selected, and a request named 'Request-1' is created. The request method is 'GET' and the URL is 'http://localhost:3000/students/1'. The response is displayed in the 'Pretty' view, showing a JSON object for a single student.

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

How to get all students data from the above URL?

Task No- 3

The screenshot shows the Postman interface with a workspace named 'myfirst'. A new collection 'New Collection1' is selected, and a request named 'All students' is created. The request method is 'GET' and the URL is 'http://localhost:3000/students'. The response is displayed in the 'Pretty' view, showing a JSON array of two student objects.

```
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,
```

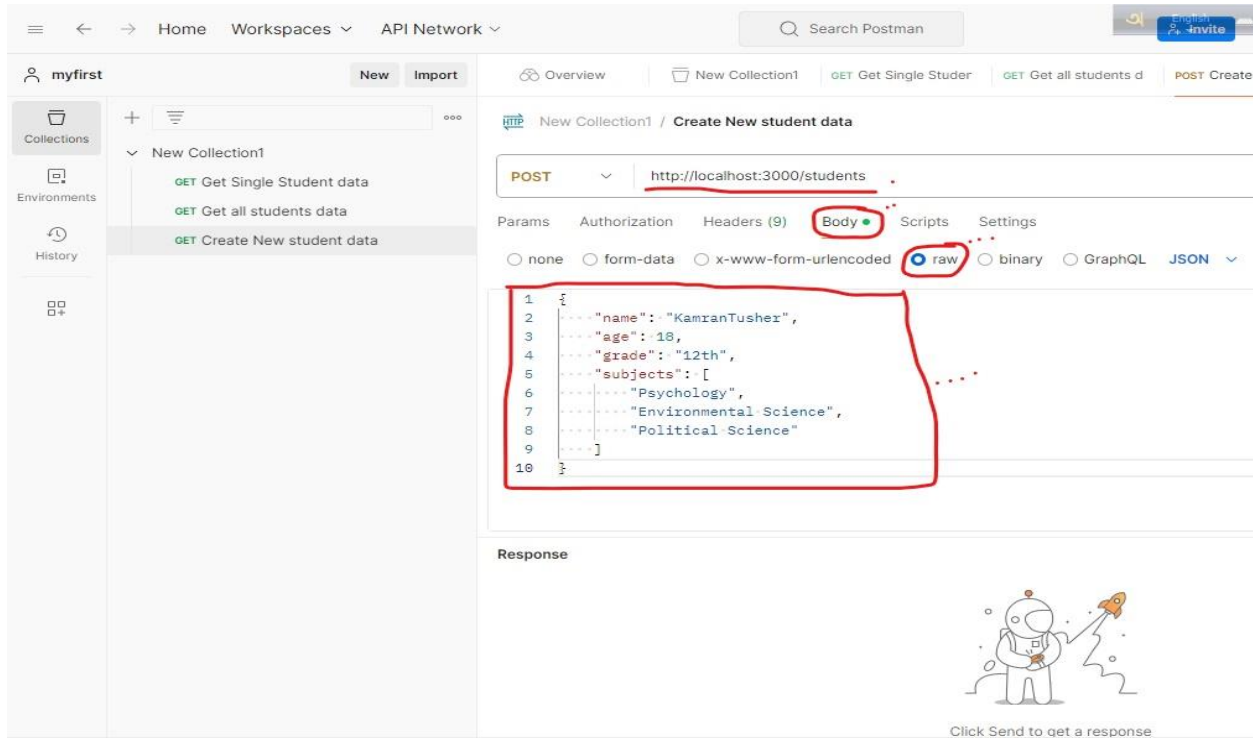
How to create new student data?

Create = Post

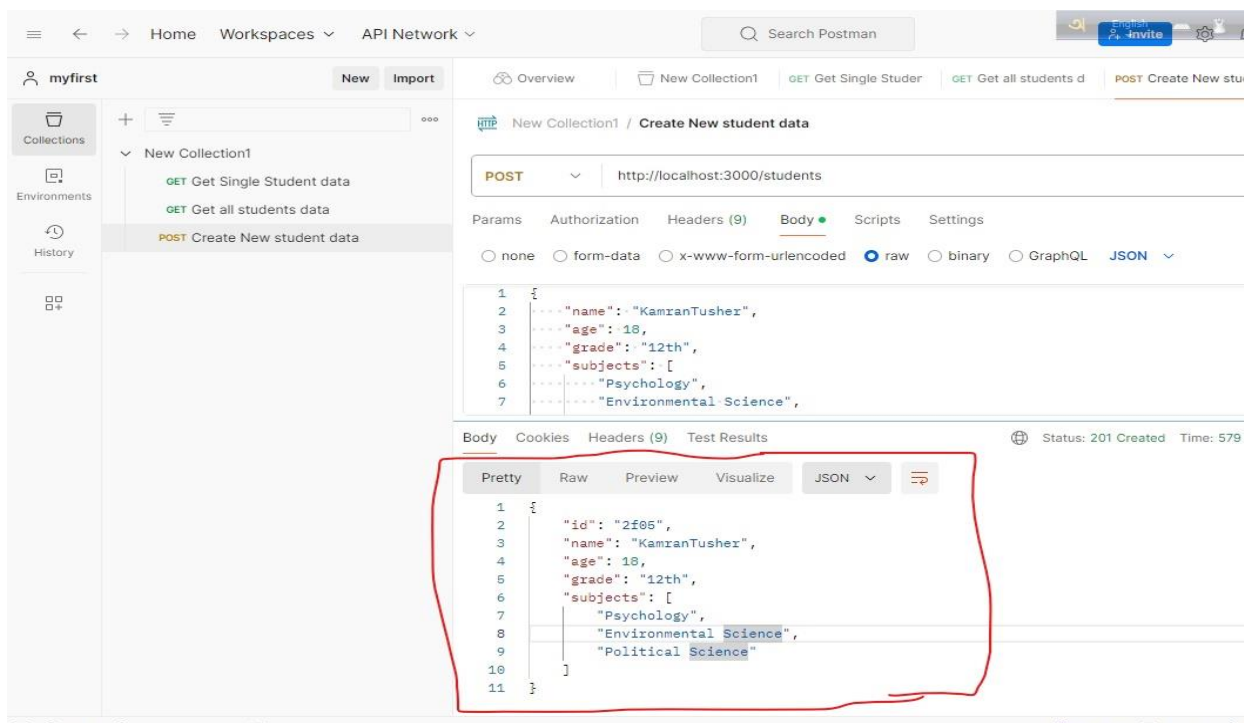
Task No- 4

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

Step-1 (Request Payload)

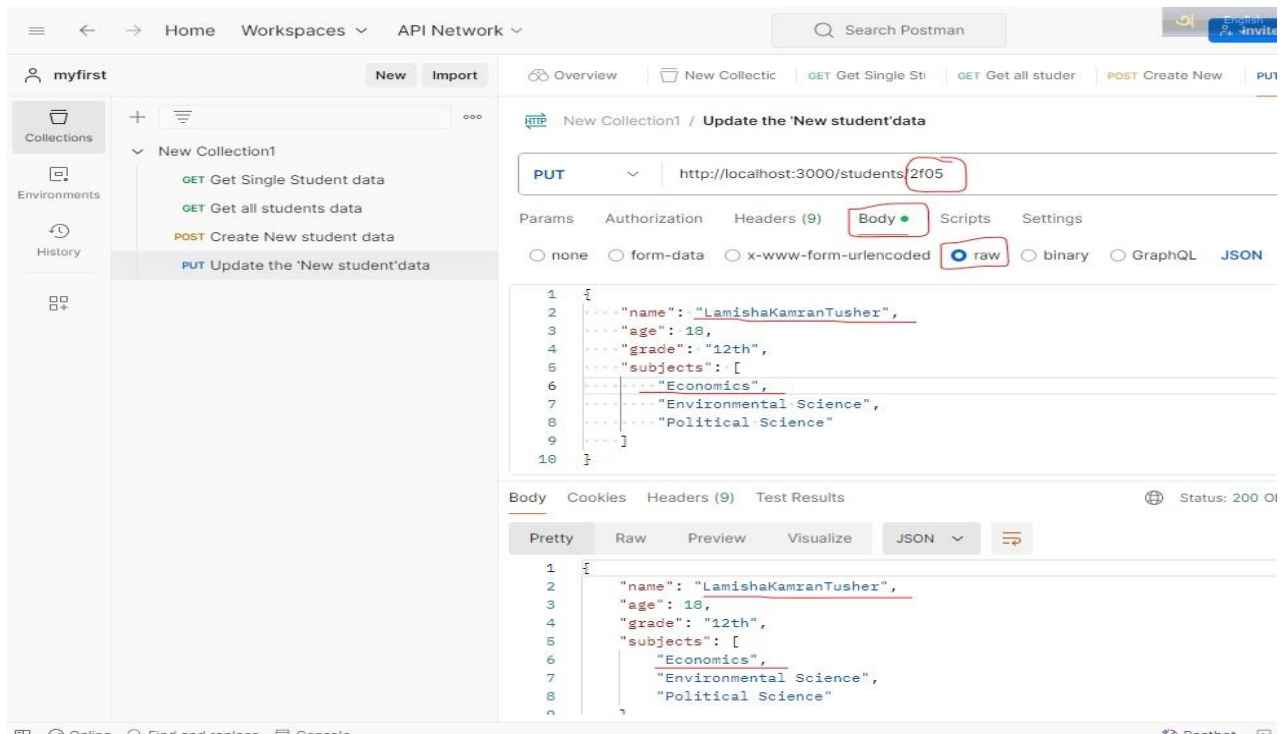


Step-2 (Response Payload)



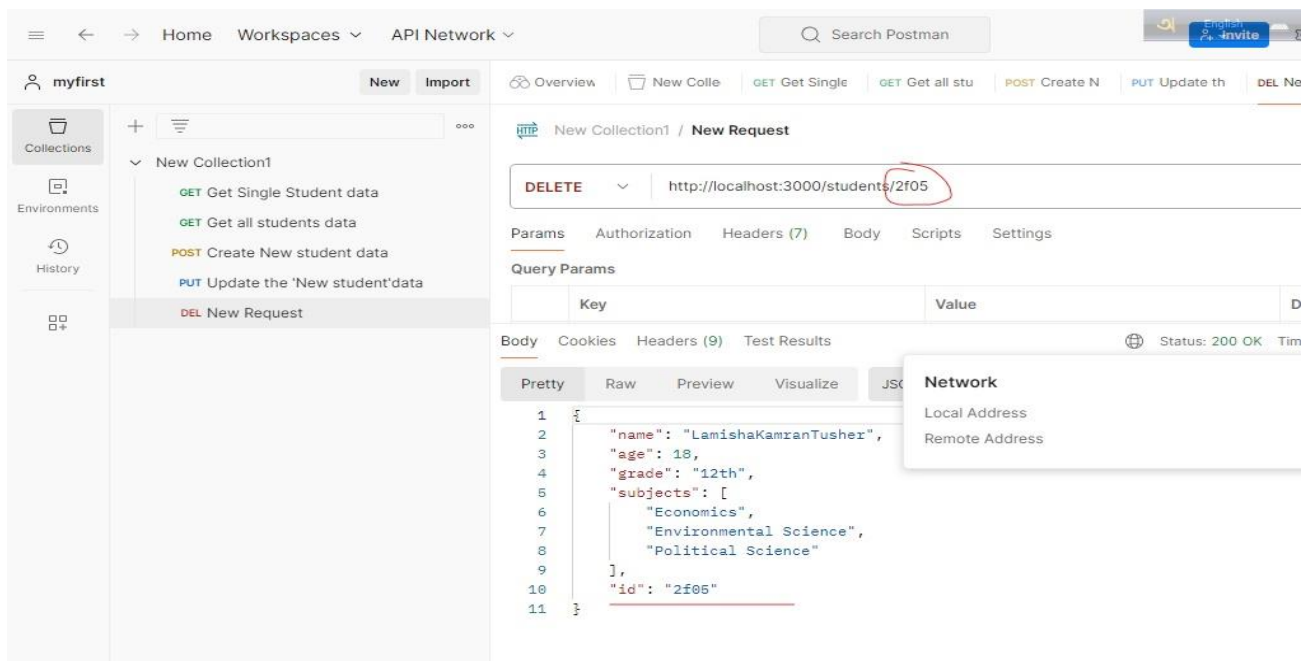
How to update the earlier “New student data” data? Update = Put Task No- 5

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

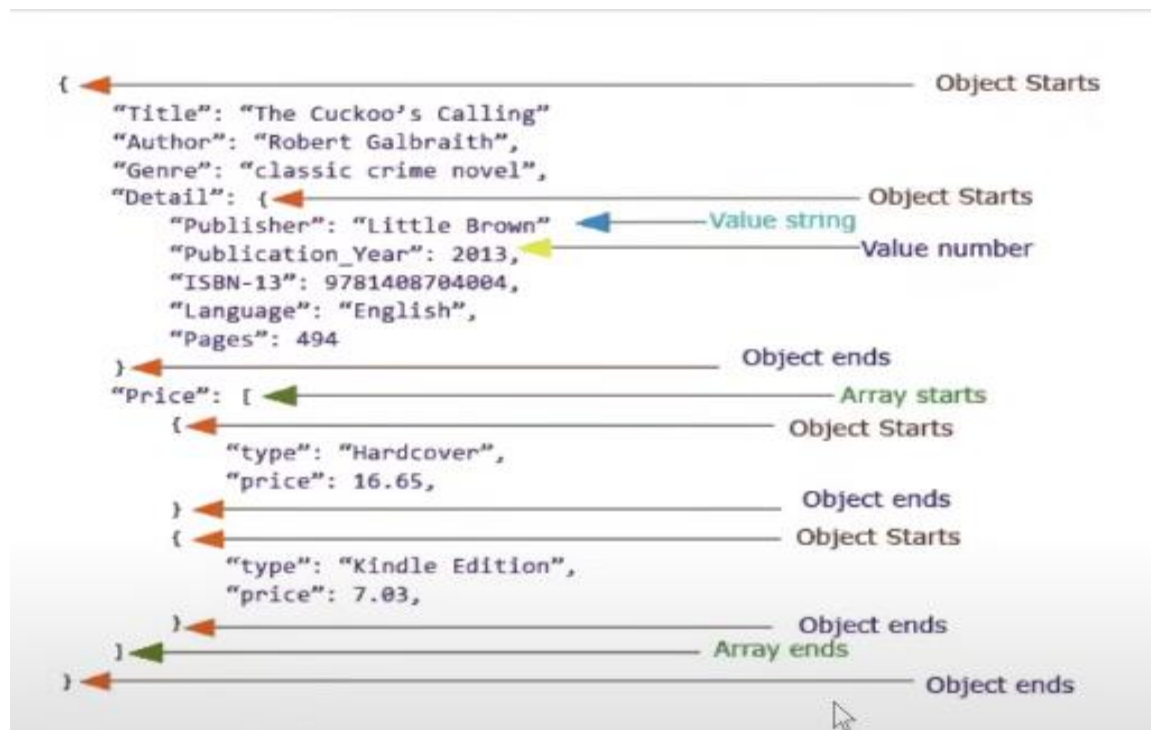
```
{  
  "student": [  
    {  
      "id": "01",  
      "name": "Tom",  
      "lastname": "Price"  
    },  
    {  
      "id": "02",  
      "name": "Nick",  
      "lastname": "Thameson"  
    }  
  ]  
}
```


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 several student objects. The right pane shows the path `x.Student[6].First_NAME` entered, and the extracted value `Anisul` is displayed. The tool also shows the corresponding object details for the selected path.

Path	Value
<code>x.Student[6].First_NAME</code>	<code>Anisul</code>

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" and "Evaluation Results". 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 JSONPath expression to the evaluation 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 response is displayed in the "Body" tab, showing a JSON object with fields `id`, `name`, `age`, `grade`, and `subjects`. The status code `200 OK` is highlighted with a red circle. A tooltip provides additional information about the status code.

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

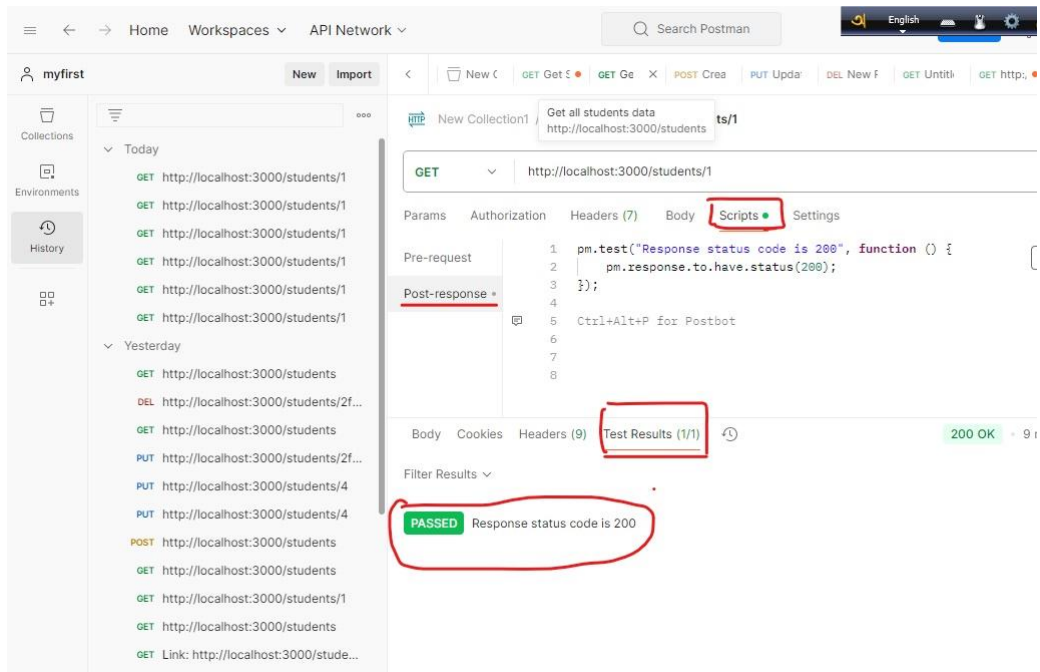
```
1 {
2   "id": "1",
3   "name": "John Doe",
4   "age": 18,
5   "grade": "12th",
6   "subjects": [
7     "Math",
8     "Physics",
9     "English"
10  ]
11 }
```

Standard response for successful HTTP requests. The actual response will depend on the request method used. In a GET request, the response will contain an entity corresponding to the requested resource. In a POST request the response will contain an entity describing or containing the result of the action.

Step-2

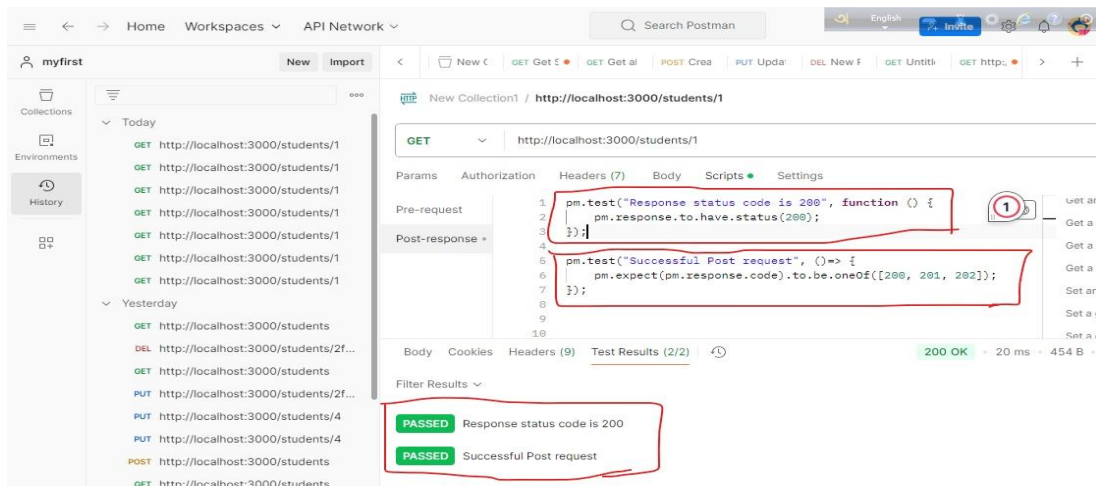
For Single Status code

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



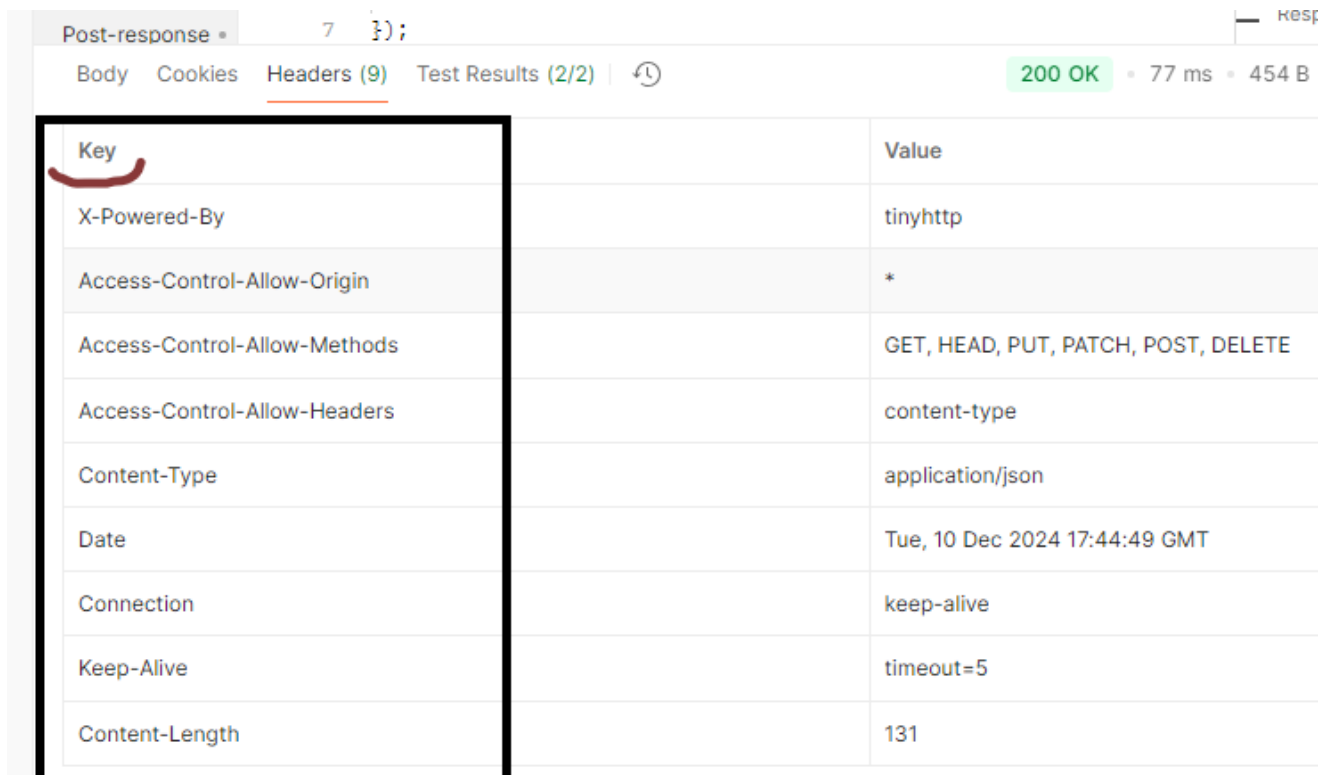
For Multiple Status code

```
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



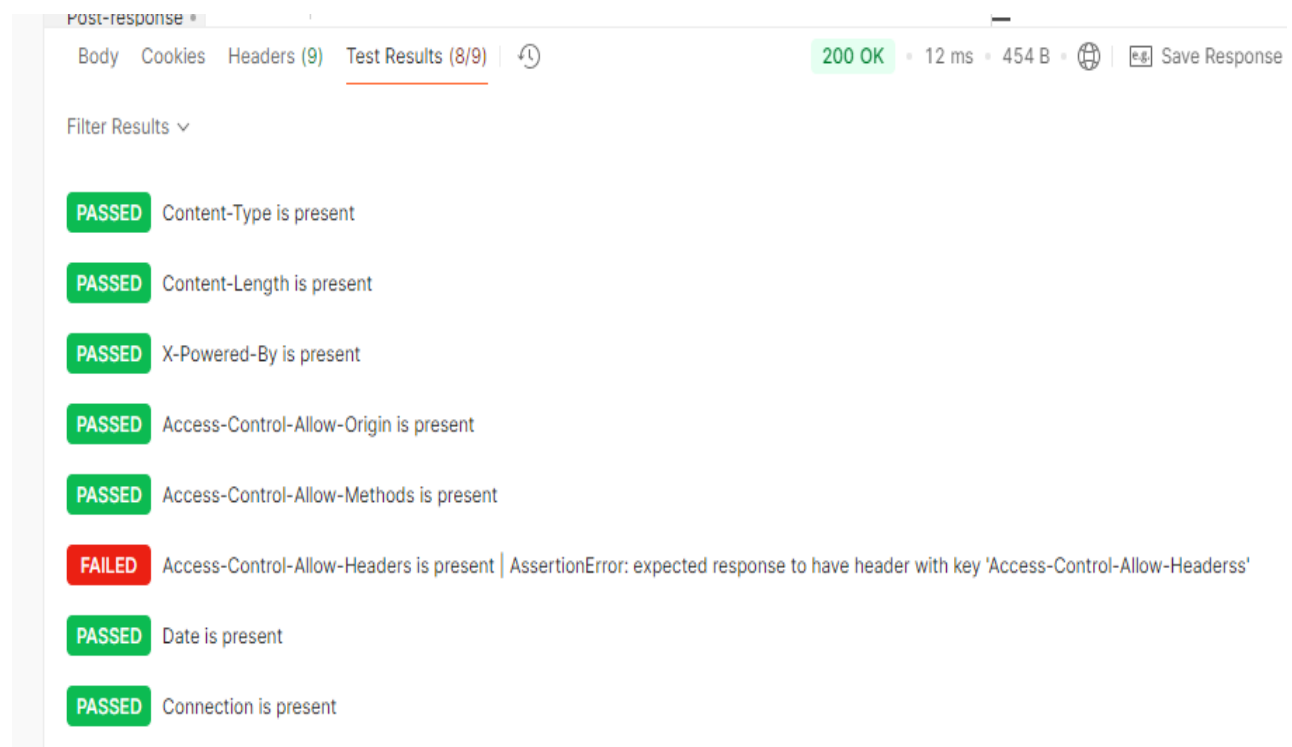
Key	Value
X-Powered-By	tinyhttp
Access-Control-Allow-Origin	*
Access-Control-Allow-Methods	GET, HEAD, PUT, PATCH, POST, DELETE
Access-Control-Allow-Headers	content-type
Content-Type	application/json
Date	Tue, 10 Dec 2024 17:44:49 GMT
Connection	keep-alive
Keep-Alive	timeout=5
Content-Length	131

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

200 OK	77 ms	454 B
Value		
tinyhttp		
*		
GET, HEAD, PUT, PATCH, POST, DELETE		
content-type		
application/json		
Tue, 10 Dec 2024 17:44:49 GMT		
keep-alive		
timeout=5		
131		

Code is given below

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

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

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

The response status is **200 OK** with a response time of **8 ms** and a size of **454 B**.

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 detailed response time breakdown is shown: Prepare (23.16 ms), Socket Initialization (25.26 ms), DNS Lookup (4.48 ms), TCP Handshake (2 ms), Waiting (TTFB) (67.18 ms), Download (7.31 ms), and Process (2.00 ms). The test is marked as FAILED with the message: "Response time is less than 100ms | AssertionError: expected".

The bottom screenshot shows the same GET request with a test script: `pm.test("Response time is less than 200ms", function () { pm.expect(pm.response.responseTime).to.be.below(200); });`. The test results show a status of 200 OK and a response time of 8 ms. A message box states: "200 OK Request successful. The server has responded as required." The test is marked as PASSED with the message: "Response time is less than 200ms".

Response Body:

Task No-

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 });
```

Pre-request Post-response *

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 });
```

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

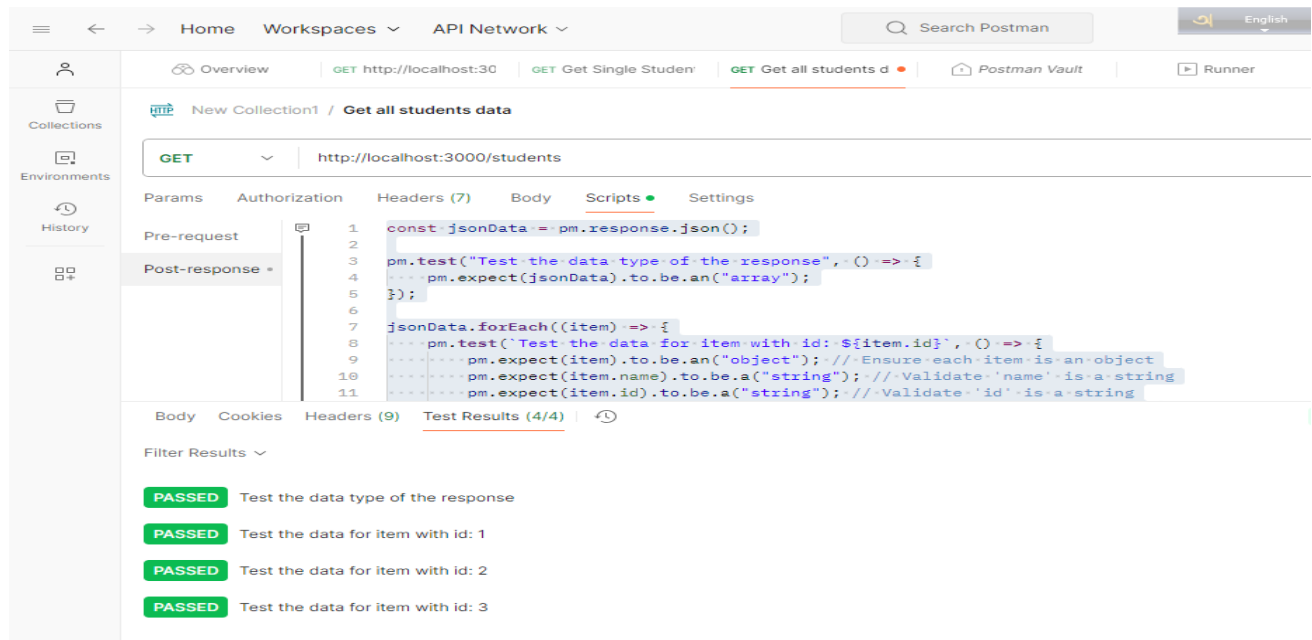
    pm.expect(item.id).to.be.a("string"); // Validate 'id' is a string

    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

  });

});
```



Array properties in the response body :

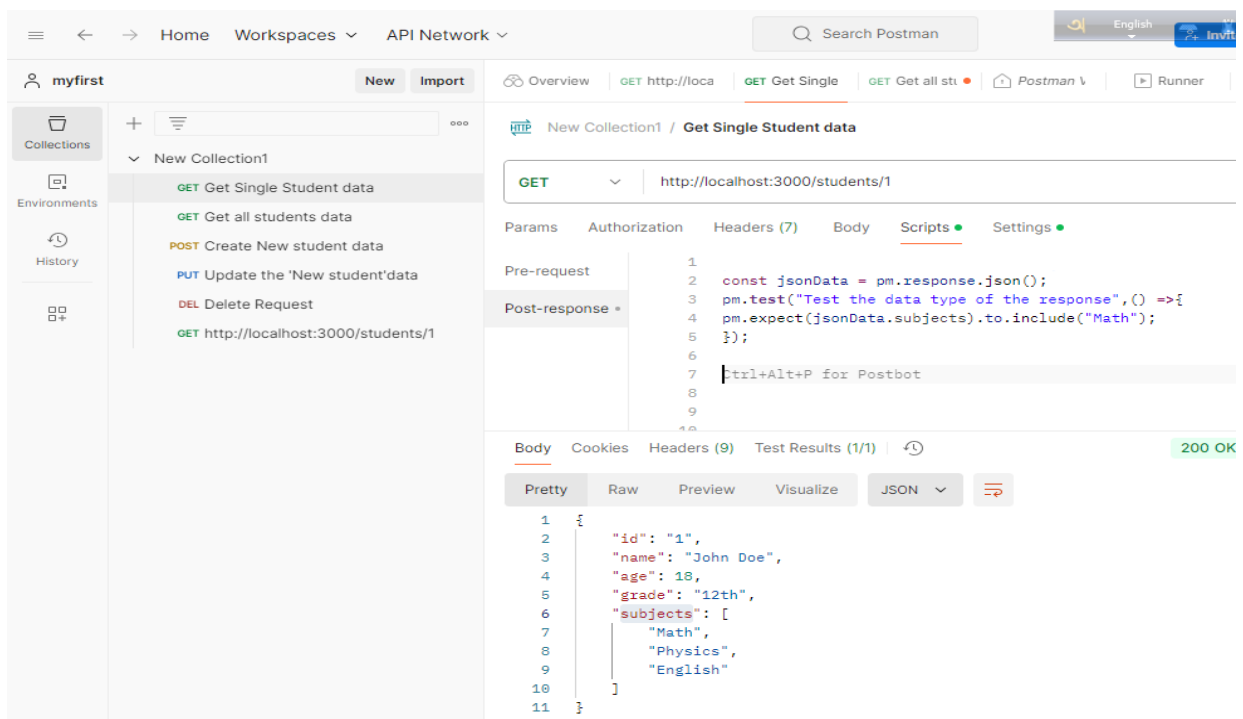
How to validate the Array properties in the response body ?

-(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:

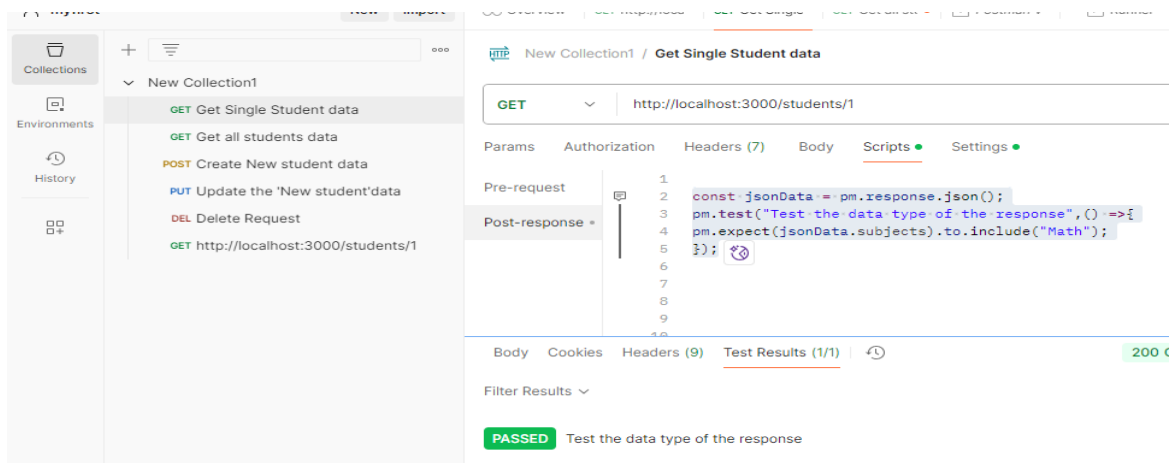


The screenshot shows the Postman interface with a GET request to `http://localhost:3000/students/1`. The response is a JSON object with the following structure:

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

The 'Scripts' tab is active, showing the test 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  
8  
9  
10  
11
```



The screenshot shows the Postman interface with the 'Test Results' tab active. The test results show:

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11
```

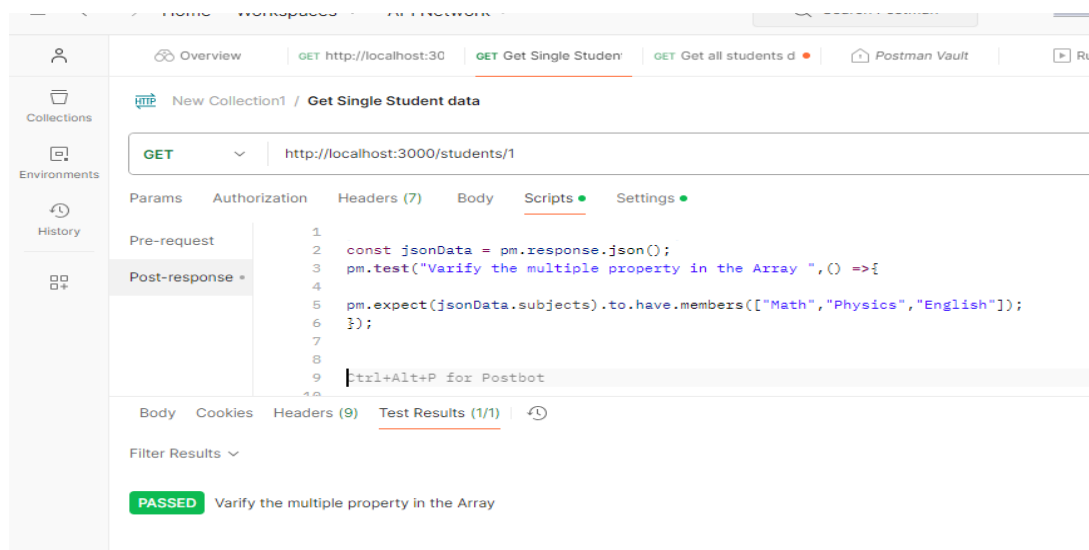
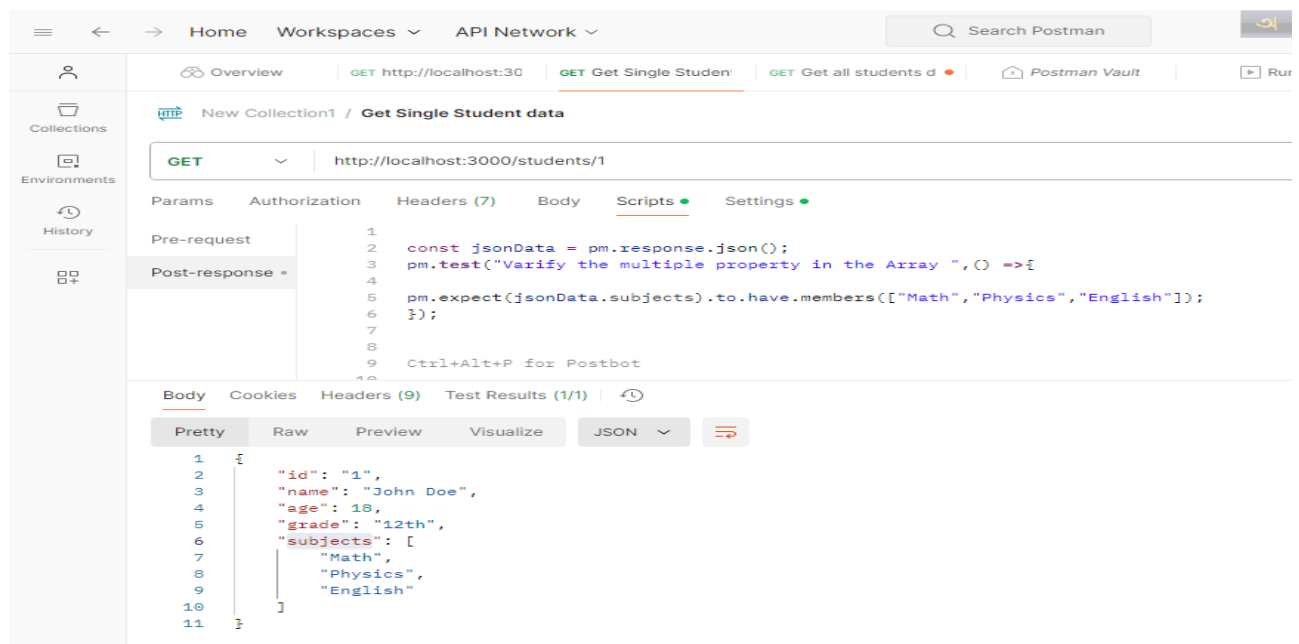
The test results show a green 'PASSED' status for the test 'Test the data type of the response'.

-(Validate a single 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:



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");

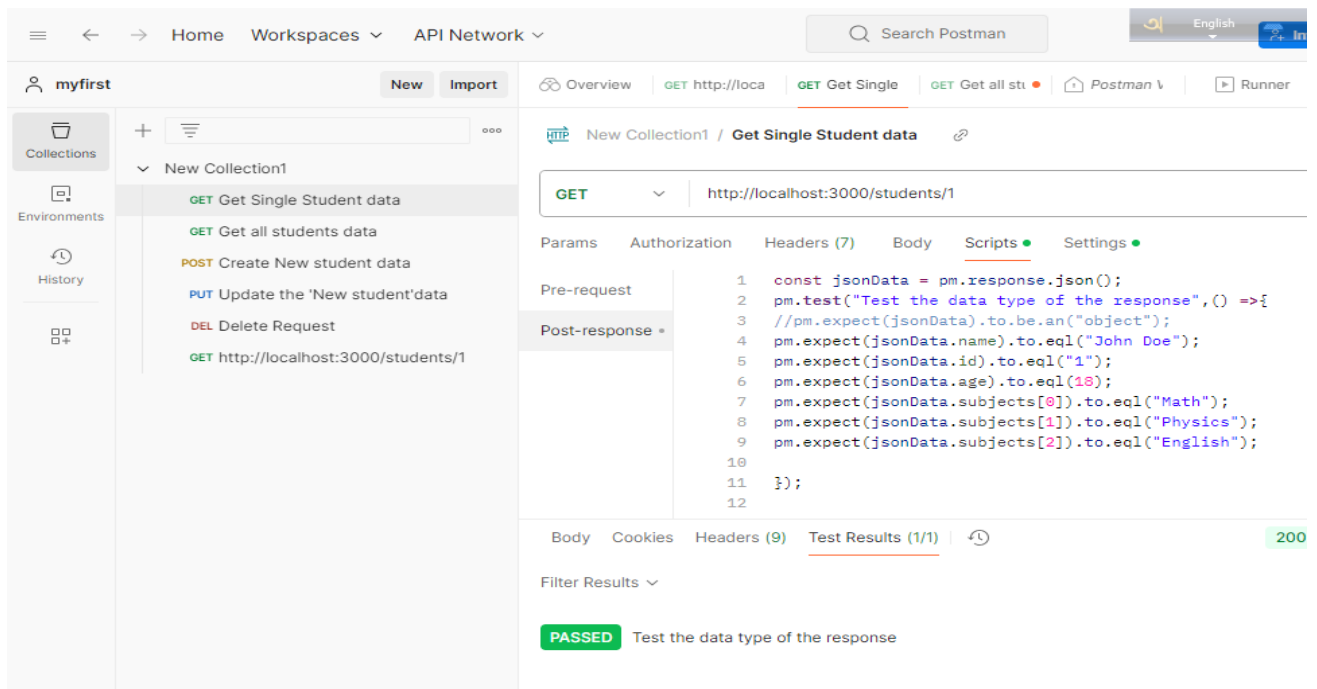
pm.expect(jsonData.subjects[2]).to.eql("English");

});
```

The screenshot displays the Postman interface. On the left, a collection named 'New Collection1' contains several requests, including 'GET Get Single Student data'. The main panel shows a GET request to 'http://localhost:3000/students/1'. The 'Scripts' tab is active, showing the same JavaScript code as provided in the text. The 'Body' tab is also active, showing the JSON response:

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

. The status bar at the bottom right indicates a '200 OK' response.



Validate the Json Schema :

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



```

{
  "$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"
  ]
}

```

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?

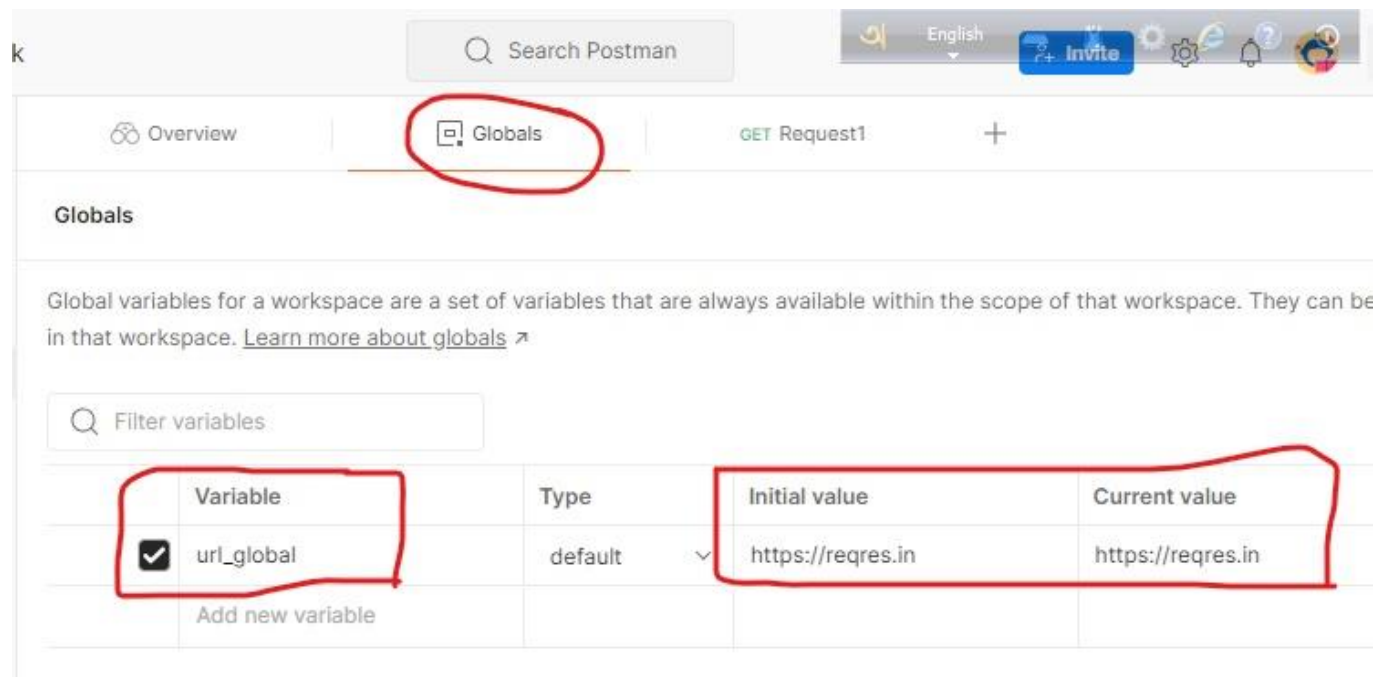
Ans: workspace--→Collection---→Request/Folder.

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](#)

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)

The screenshot shows the Postman interface with a GET request named 'Request1' configured. The URL is `{{url_global}}/api/users?page=2`, where the variable `{{url_global}}` is highlighted with a red circle. The 'Query Params' section shows a table with two entries: 'page' with a value of '2'. The 'Body' tab is selected, showing a JSON response in 'Pretty' format. The response status is '200 OK'.

Key	Value
page	2

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

Step-5 ----(Post Request)

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

The screenshot shows the Postman interface with a POST request named 'Request-2' configured. The URL is `{{url_global}}/api/users`, where the variable `{{url_global}}` is highlighted with a red circle. The 'Query Params' section shows a table with two entries: 'Key' with a value of 'Value'. The 'Body' tab is selected, showing a JSON response in 'Pretty' format. The response status is '201 Cr'.

Key	Value
Key	Value

```
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 'PUT Request-3' selected in the left sidebar. The main panel displays the request configuration for 'Request-3'.

Request Configuration:

- Method:** PUT
- URL:** `{{url_global}}/api/users/2`
- Params:** Authorization, Headers (8), Body, Scripts, Settings
- Query Params:**

Key	Value
Key	Value

Body:

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)

The screenshot shows the Postman interface with the 'DELETE Request-4' selected in the left sidebar. The main panel displays the request configuration for 'Request-4'.

Request Configuration:

- Method:** DELETE
- URL:** `{{url_global}}/api/users/2`
- Params:** Authorization, Headers (7), Body, Scripts, Settings
- Query Params:**

Key	Value	Description
Key	Value	Description

Body:

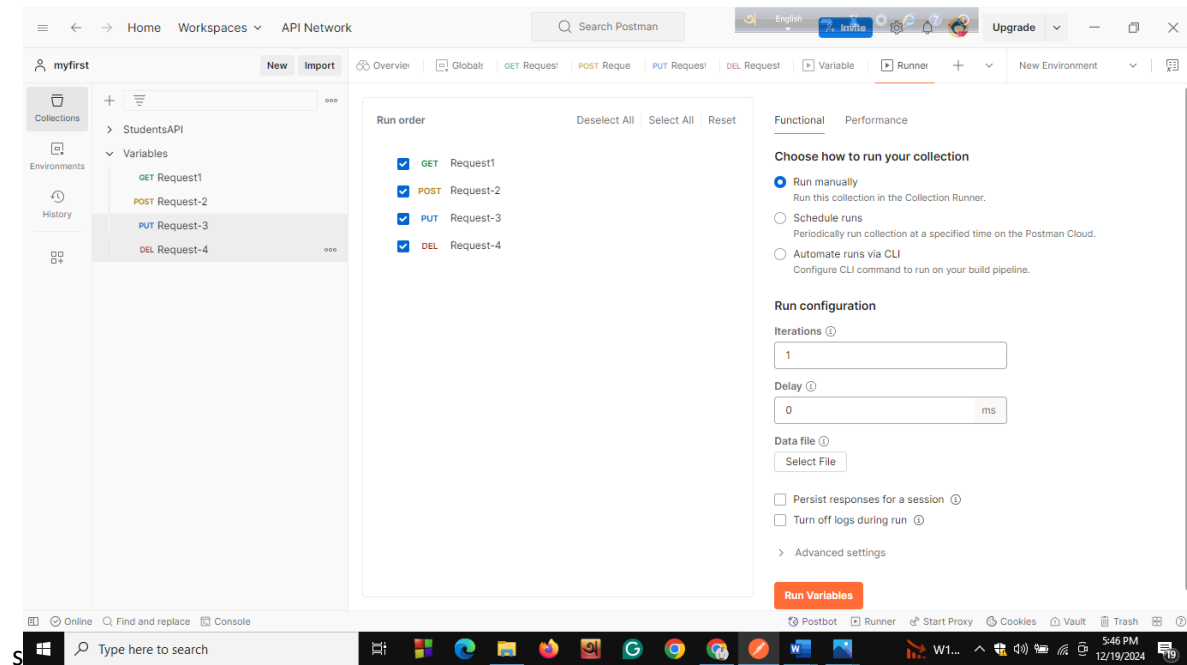
Body | Cookies | Headers (14) | Test Results |

204 No Content • 612 ms • 1 KB • | | | |

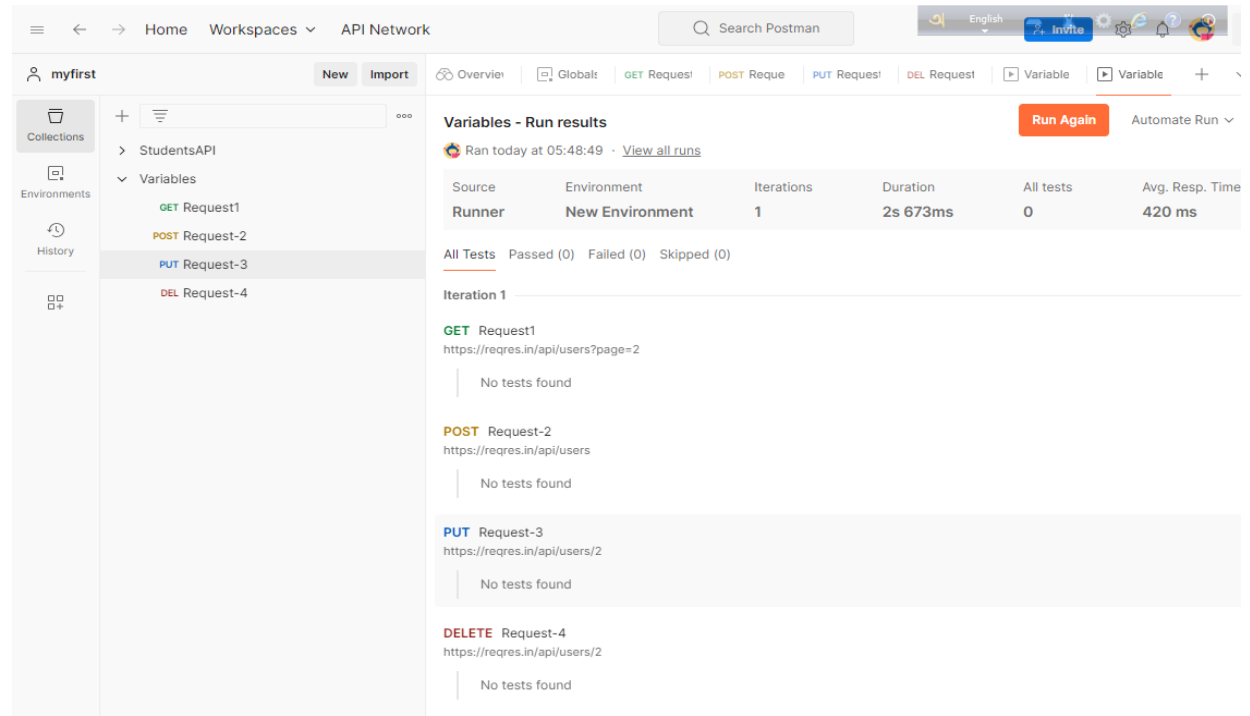
Pretty | Raw | Preview | Visualize | Text |

```
1
```

Step-8



Step-9



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

Step-1

The screenshot shows the Postman interface with the 'Variables' tab selected for the 'StudentsAPI' collection. The left sidebar shows the 'Variables' tab highlighted with a red box labeled '1'. The context menu for the 'Variables' tab is open, with the 'Edit' button highlighted by a red box labeled '2'. The main panel shows the 'Variables' tab with a search bar labeled 'Filter variables' highlighted by a red box labeled '3'. Below the search bar is a table of variables:

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

Step-2

The screenshot shows the Postman interface with a GET request selected. The URL is `GET {{url_collection}}/api/users?page=2`, where the variable `{{url_collection}}` is highlighted by a red box. The 'Query Params' section shows the following parameters:

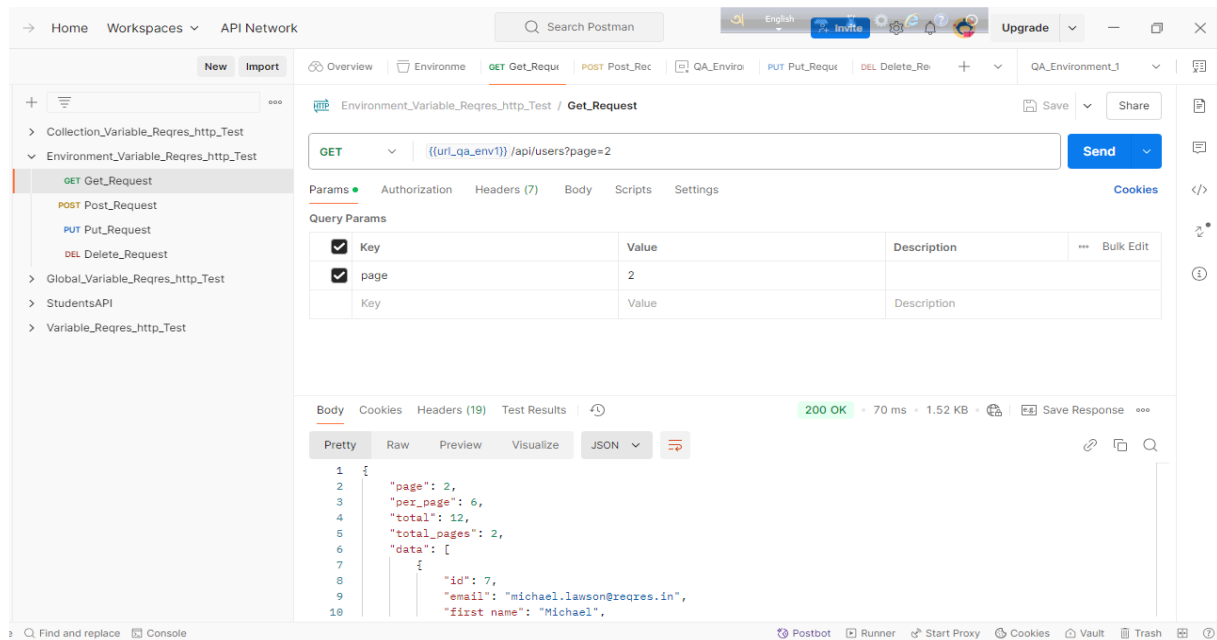
Key	Value
<input checked="" type="checkbox"/> page	2
<input type="checkbox"/> Key	Value

The 'Body' tab is selected, showing the response in 'Pretty' format:

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

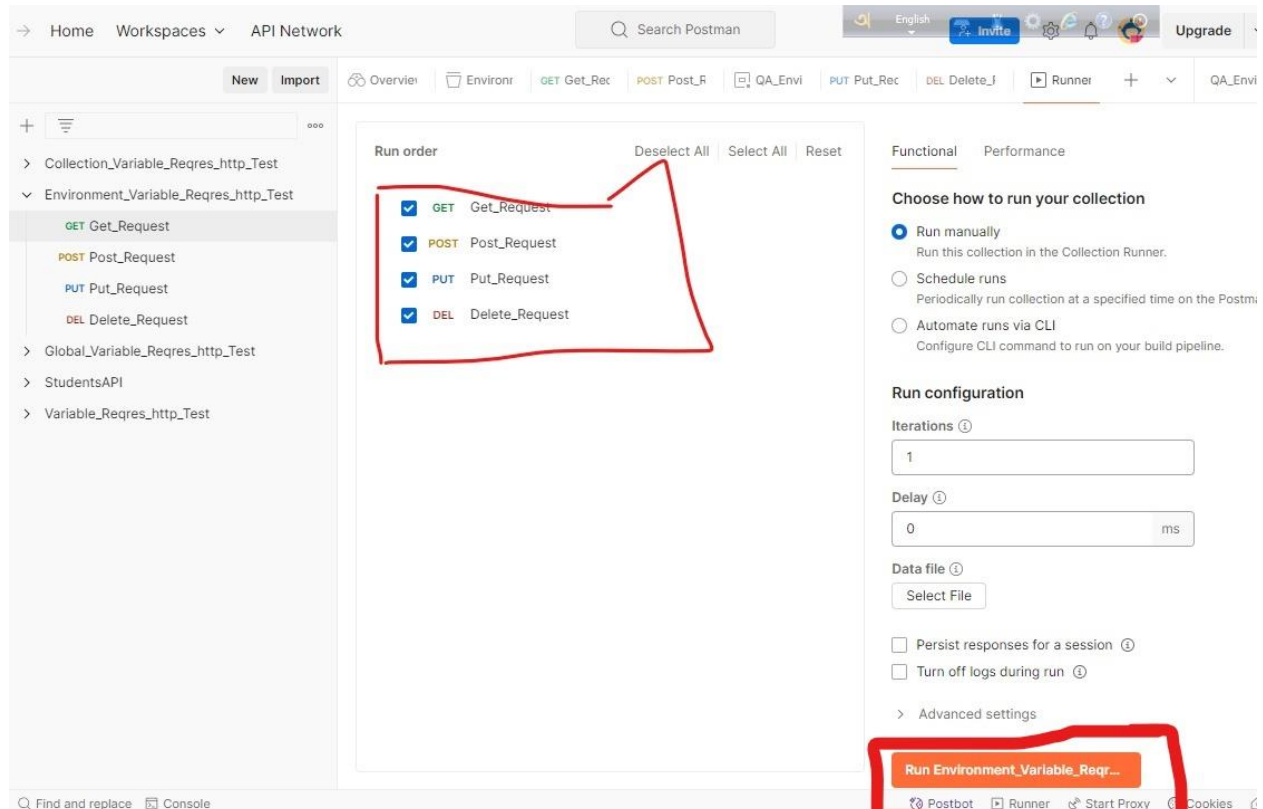
The status bar at the bottom shows '200 OK' and '185'.

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_Reqres...' 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. A table shows the source, environment, iterations, duration, all tests, and average response time. Below the table, the results for each iteration are shown, including the request method, URL, status code, response time, and size. The results for the first iteration are as follows:

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

Iteration 1

Method	URL	Status	Time	Size
GET	https://reqres.in/api/users?page=2	200 OK	97 ms	1.556 KB
POST	https://reqres.in/api/users	201 Created	606 ms	1.114 KB
PUT	https://reqres.in/api/users/2	200 OK	640 ms	1.118 KB
DELETE	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 results of the API requests, including the method, URL, status code, and response time. The results are as follows:

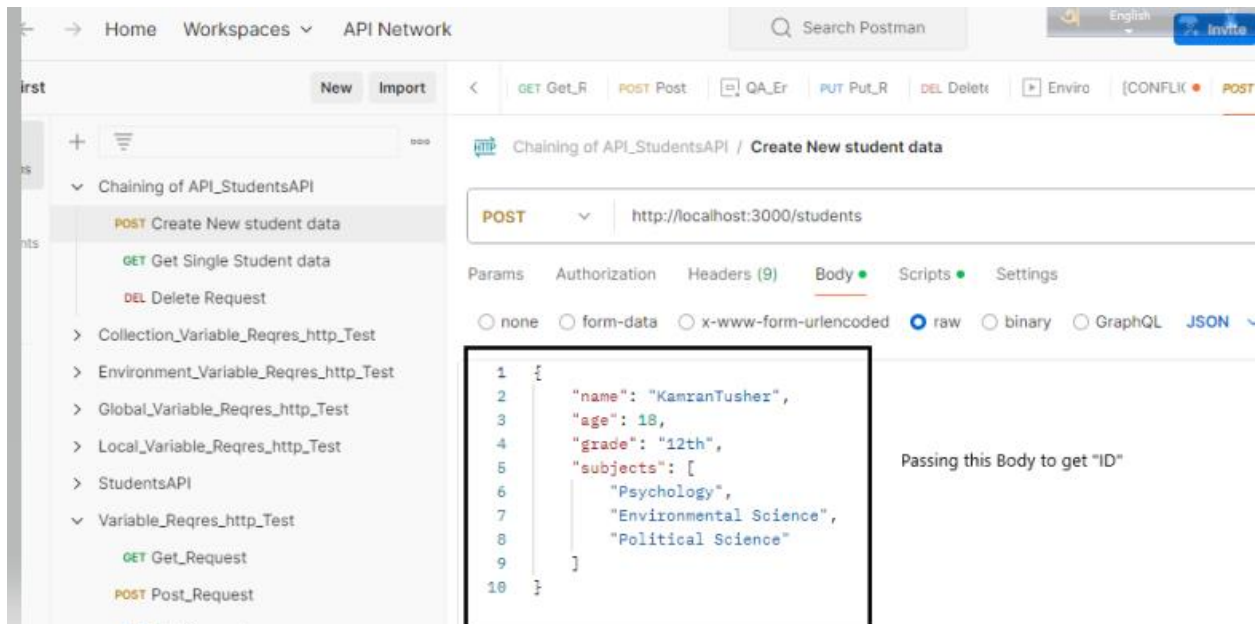
Method	URL	Status	Time
GET	https://reqres.in/api/users?page=2	200	88 ms
POST	https://reqres.in/api/users	201	609 ms
PUT	https://reqres.in/api/users/2	200	623 ms
DELETE	https://reqres.in/api/users/2	204	608 ms
GET	https://reqres.in/api/users?page=2	404	654 ms
GET	https://reqres.in/api/users?page=2	200	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

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.