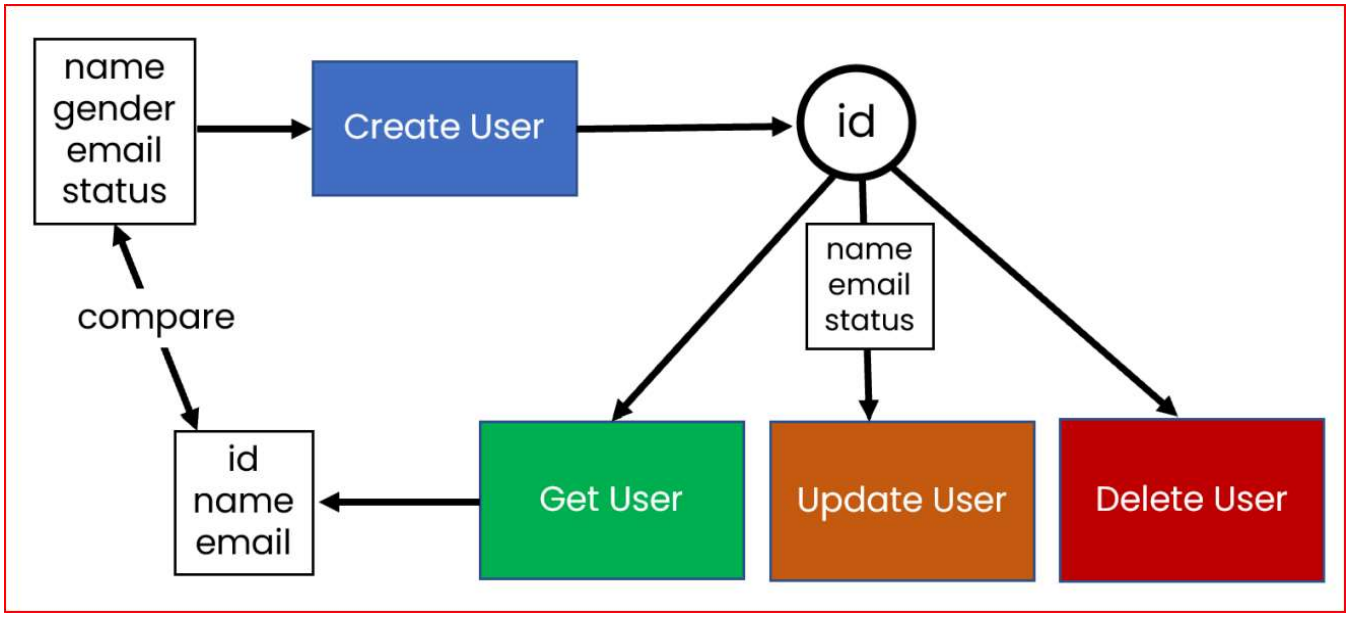
5. API Chaining

API Chaining

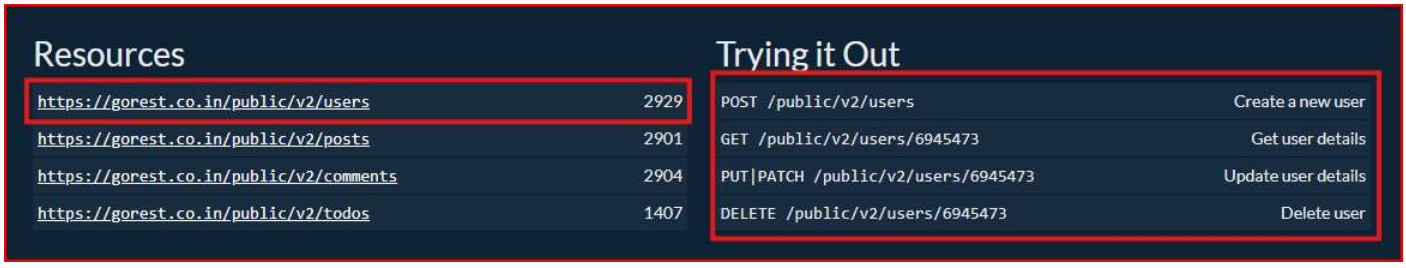
- ➔ API chaining involves executing multiple API requests in **sequence**, where the response or part of the response from one request is used as input for the next or subsequent requests.



- ➔ If there will be dependencies between multiple API requests in that case we will use API Chaining.
- ➔ We will store Response in variables and we can use those variables as part of the request.

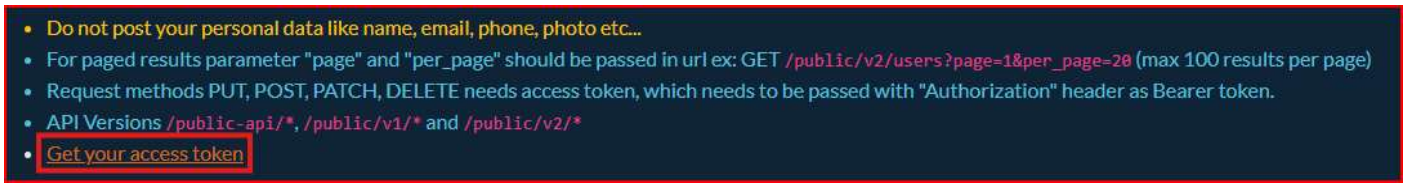
GoRest API

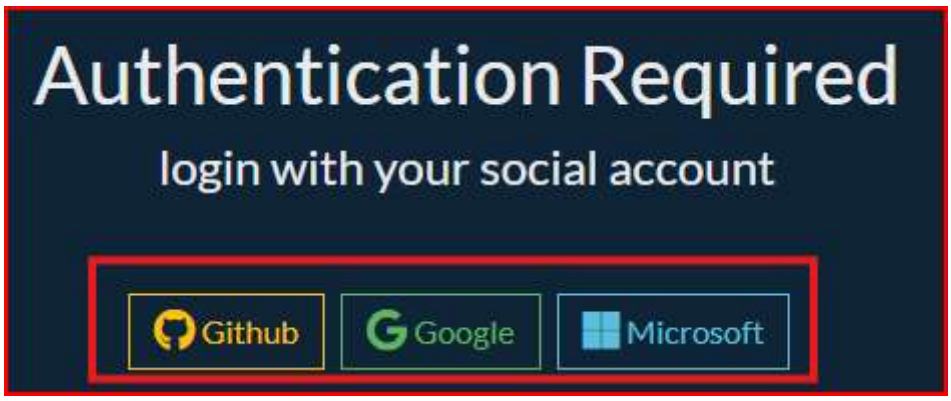
- ➔ GoRest API is a popular public API used for API chaining demonstrations and requires **token-based authentication**.
- ➔ GoRest API Supports GraphQL, RestAPI, etc many more things.
- ➔ Navigate to https://gorest.co.in/ to see API Documentation.
- ➔ We Will use **users Resource** for our demo purpose. It has POST, GET, PUT, DELETE requests.



POST	/public/v2/users	Create a new user
GET	/public/v2/users/23	Get user details
PUT	/public/v2/users/23	Update user details
DELETE	/public/v2/users/23	Delete user

- ➔ Click on **Get your Access Token** to create an Access Token for working with **users resource**.

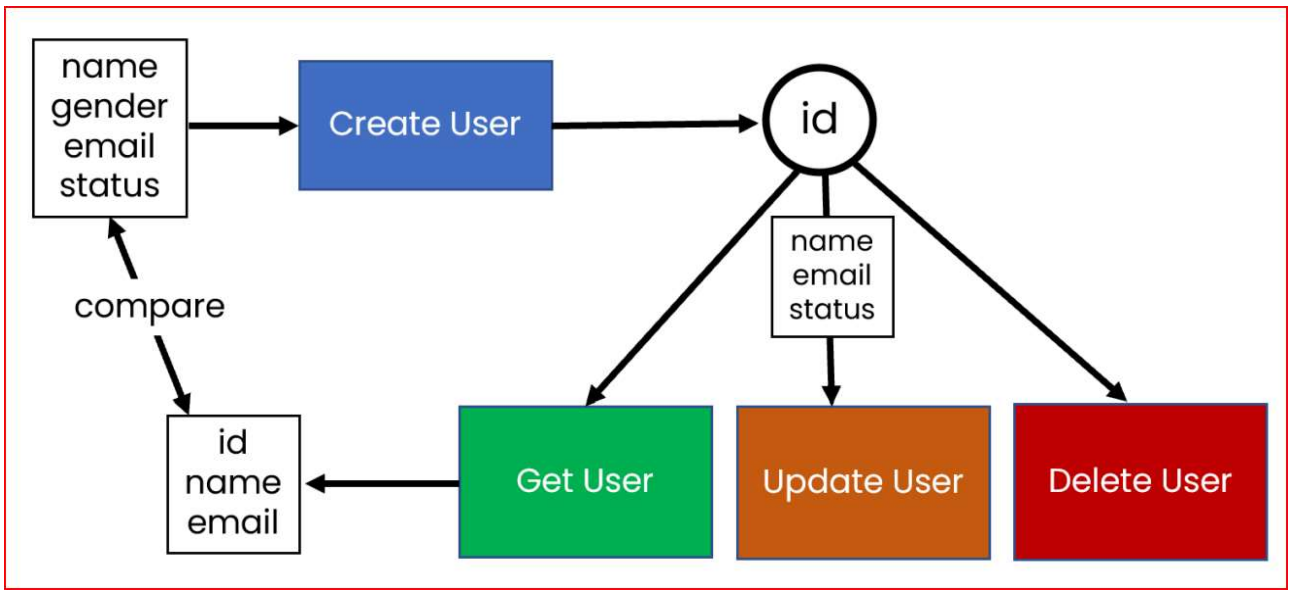




➔ Once we signup navigate to https://gorest.co.in/my-account/access-tokens to see active tokens.

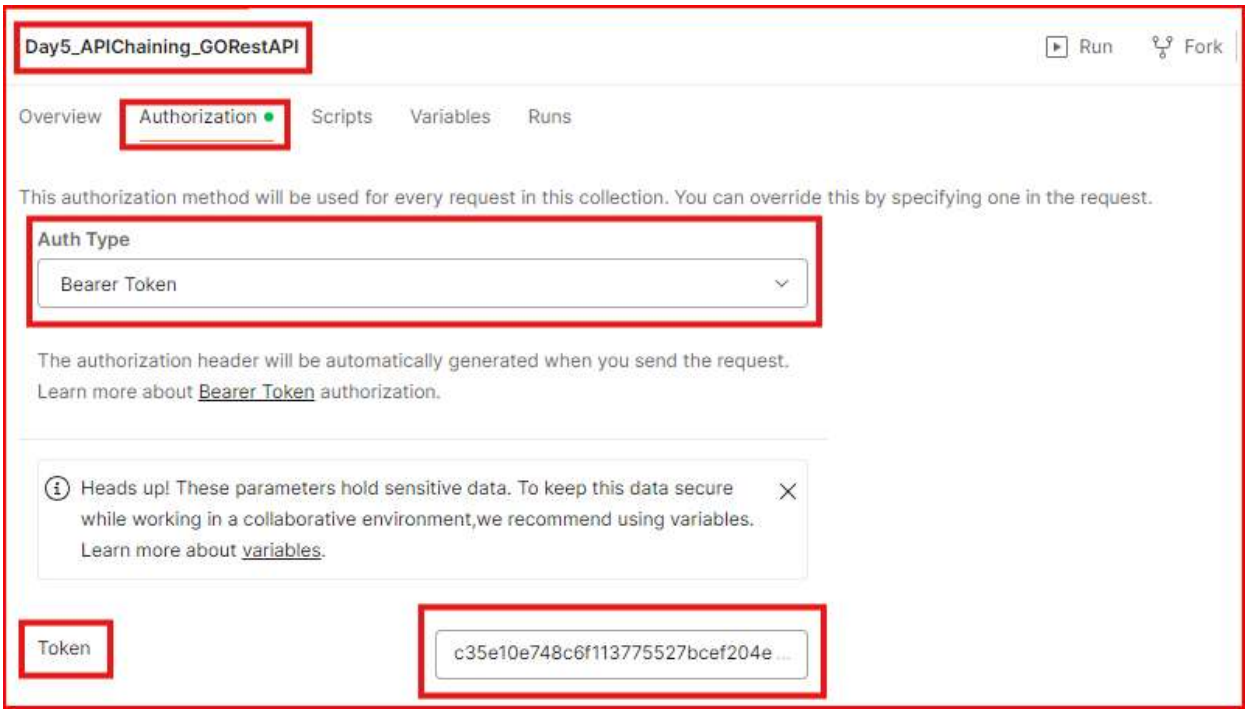
Workout

➔ Create **Day5_APIChaining_GORestAPI** collection.

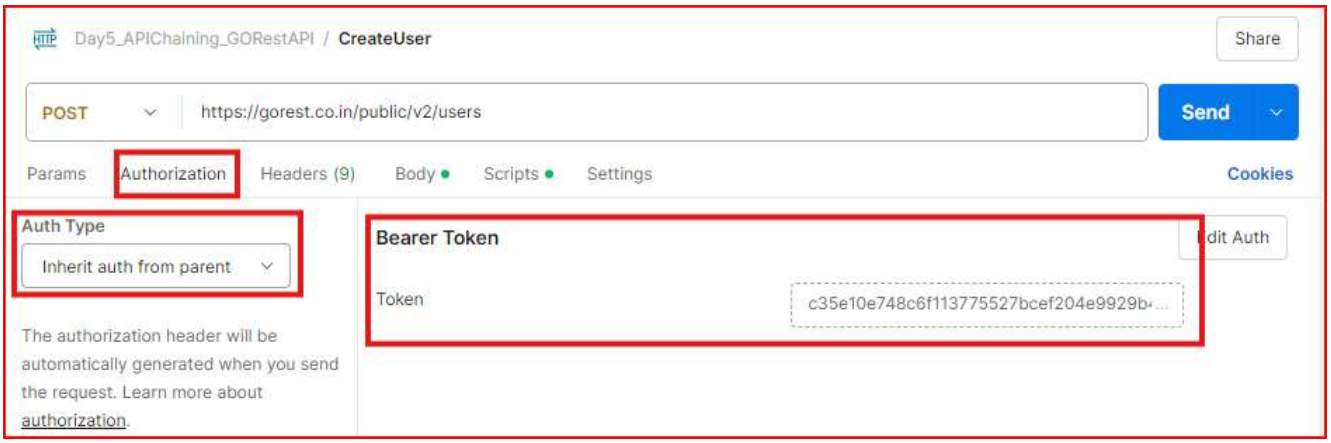


➔ Body contains Request payload, variables to hold data are in pre-request script and validations are in post-response script.

➔ At the collection level, select Bearer Token under Authorization from Auth Type, add the created token.



➔ Choose **Inherit auth from Parent** at the request level under Authorization to apply the token to all requests in the collection.



Step 1: Create a User

➔ **HTTP Method:** POST

➔ **Request URL:** https://gorest.co.in/public/v2/users

Request Body:

```
{
  "name": "{{username}}",
  "gender": "male",
  "email": "{{useremail}}",
  "status": "inactive"
}
```

Pre-request Script:

Set variables for the user's **name** and **email** before sending the request.

```
pm.collectionVariables.set("username", "abcxyz");
pm.collectionVariables.set("useremail", "abcxyz@gmail.com");
```

➔ Send the Request now and observe response in the body.

Post-response Script:

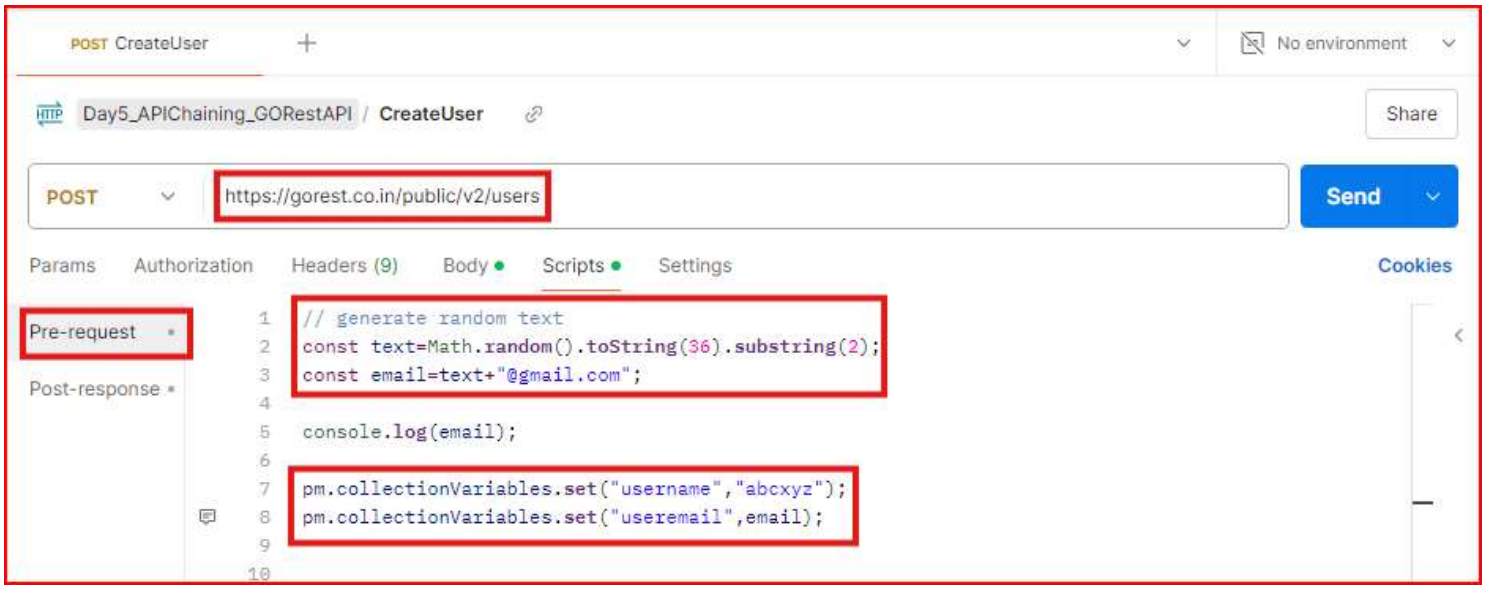
After creating the user, capture the **user ID** from the response and store it as a variable for later use.

```
const jsonData = pm.response.json();
pm.collectionVariables.set("userid", jsonData.id);
```

➔ After adding post-response script send response now and observe message as below

➔ Generate random text for email instead of duplicate email in the prequest script of request and send the request

- **const text = Math.random().toString(36).substring(2);**
- **const email = text+"@gmail.com";**
- **console.log(email);**



➔ Observe userid is also captured in Variables

Step 2: Get User Details

➔ **HTTP Method:** GET

➔ **Request URL:** https://gorest.co.in/public/v2/users/**{{userid}}**

Post-response Script:

Validate the response by checking if the user details (ID, email, and name) match the variables.

```
pm.test("Validate JSON fields", () => {
 const jsonData = pm.response.json();
  pm.expect(jsonData.id).to.eql(pm.collectionVariables.get("userid"));
  pm.expect(jsonData.email).to.eql(pm.collectionVariables.get("useremail"));
  pm.expect(jsonData.name).to.eql(pm.collectionVariables.get("username"));
});
```

Step 3: Update User Details

➔ **HTTP Method:** PUT

➔ **Request URL:** https://gorest.co.in/public/v2/users/**{{userid}}**

Request Body:

```
{
  "name": "{{username}}",
  "email": "{{useremail}}",
  "status": "active"
}
```

➔ In the request body of update request change status from **Inactive to active.**

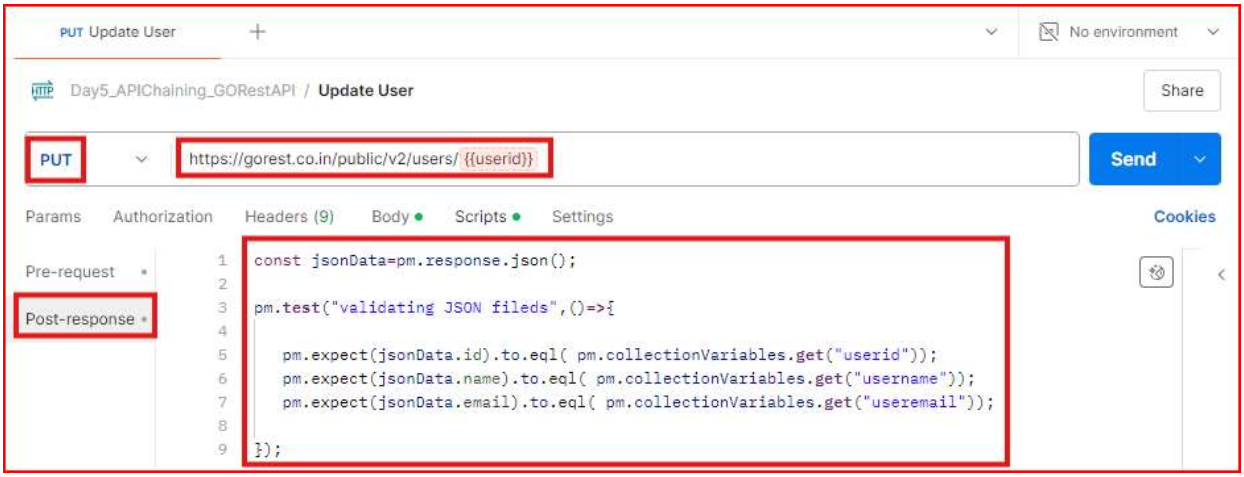
Pre-request Script:

Update the variables for the username and email before sending the request.

```
pm.collectionVariables.set("username", "abc123");
pm.collectionVariables.set("useremail", "abcxyz123@gmail.com");
```

➔ In the pre-request script dynamically generate useremail instead of using duplicate email.

➔ Add post-response script also as below as like **Get Users**.



Note

➔ Go to **Test Results** to see status of validations.

Step 4: Delete User

➔ **HTTP Method:** DELETE

➔ **Request URL:** https://gorest.co.in/public/v2/users/**{{userid}}**

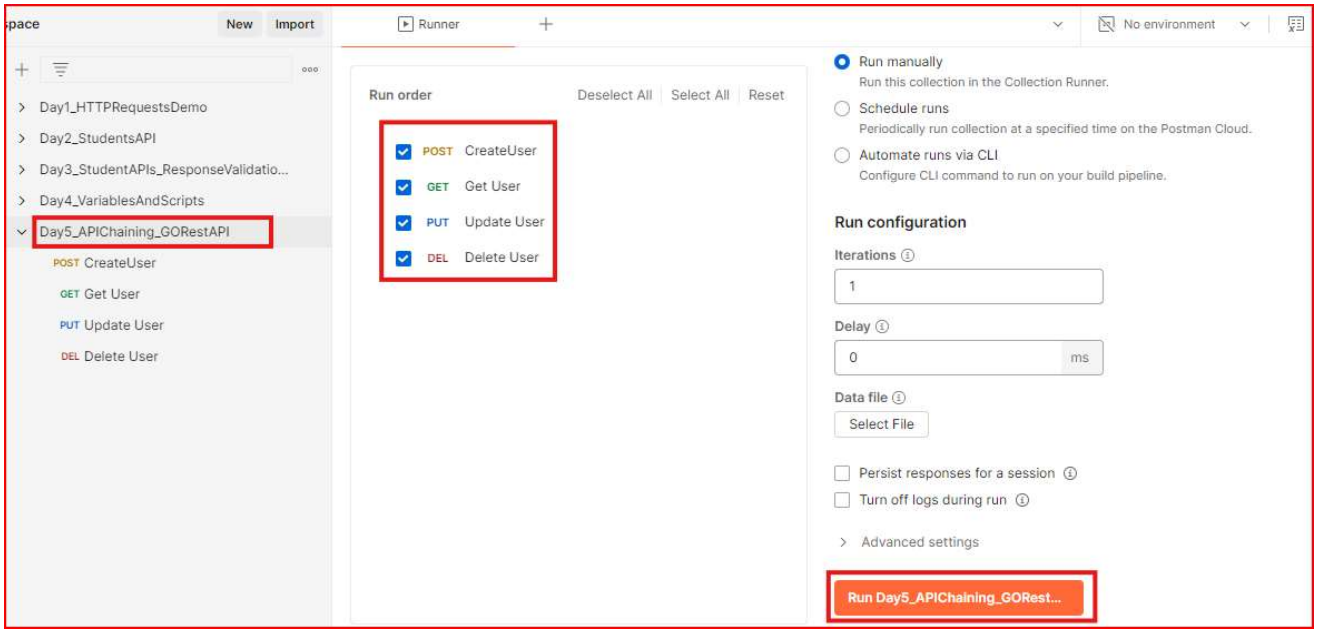
Post-response Script:

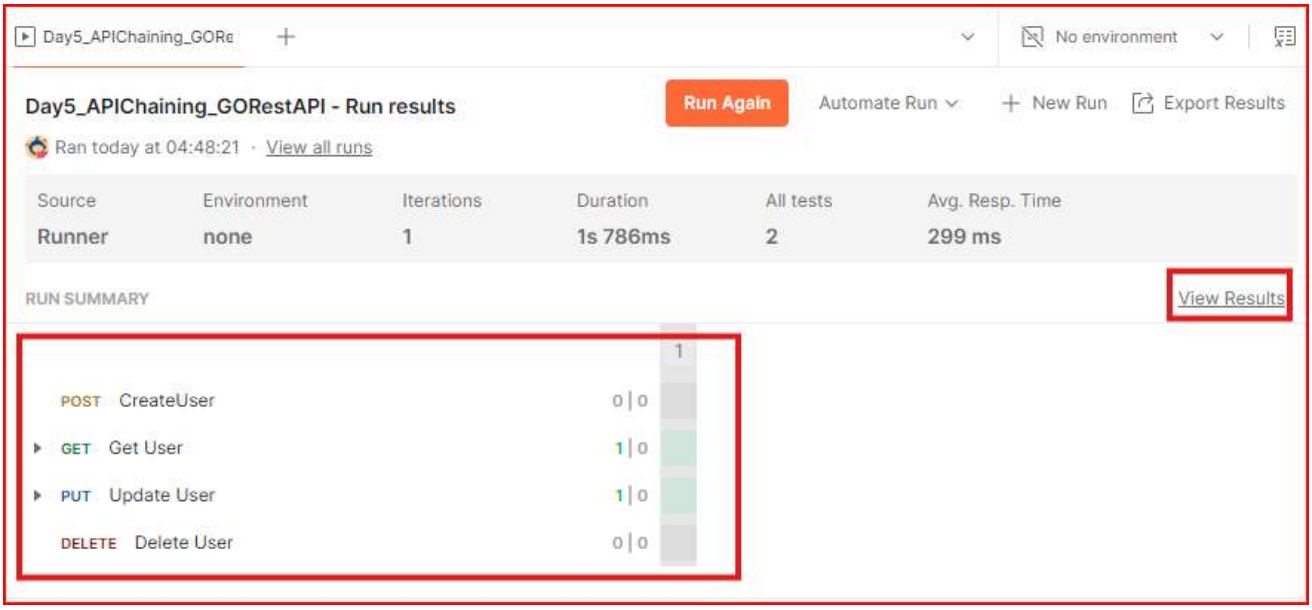
Remove all variables associated with the user after deletion.

```
pm.collectionVariables.unset("userid");
pm.collectionVariables.unset("useremail");
pm.collectionVariables.unset("username");
```

Run all requests

➔ Run all requests from the collection, observe the results, and check the console for additional details.





Key Concepts

1. **Collection Variables:** Store dynamic data like username, useremail, and userid that can be shared across requests.
2. **Pre-request Script:** Code that runs before sending the request, used to set or update variables.
3. **Post-response Script:** Code that runs after the response, used to validate or extract data from the response.
4. **Chaining:** Using the output of one request (e.g., userid) as input for the next request.

Assignment: Students API Chaining in Postman

➔ This assignment focuses on API chaining using Postman. Follow the steps to create, retrieve, and delete a student record.

Instructions

Step 1: Create a Student

1. **Send a POST request** to create a new student.
2. **Request URL:** http://localhost:3000/students

Request Body:

```
{
    "name": "Scott",
    "location": "France",
    "phone": "123456",
    "courses": [
        "C",
        "C++"
    ]
}
```

3. **Objective**
    a. Use a script in the Scripts section to capture the id from the response and store it as an environment variable.
4. **Test Script**
    a. Add the following script in the Post-response tab:

```
// Parse the response and capture the id
const jsonData = pm.response.json();
pm.environment.set("id", jsonData.id);
console.log("Captured Student ID:", jsonData.id);
```

Step 2: Display the Created Student

1. **Send a GET request** to retrieve the student details using the id captured in Step 1.
2. **Request URL:** http://localhost:3000/students/{{id}}
3. **Objective**
    a. Verify that the response contains the correct student details.
4. **Test Script**
    a. Add the following script in the **Post-response** to validate the response

```
pm.test("Validate student details", () => {
    var jsonData = pm.response.json();
    pm.expect(jsonData.id).to.eql(pm.environment.get("id"));
    pm.expect(jsonData.name).to.eql("Scott");
    pm.expect(jsonData.location).to.eql("France");
});
```

Step 3: Delete the Student

1. **Send a DELETE request** to delete the student using the captured id.
2. **Request URL:** http://localhost:3000/students/{{id}}
3. **Objective**
    a. Confirm that the student has been successfully deleted.

Deliverables

1. A Postman collection with
    a. **POST request** to create a student.
    b. **GET request** to retrieve the student details.
    c. **DELETE request** to remove the student.
2. Use **environment variables** to pass the id across requests.
3. Include scripts for validation and environment variable handling.