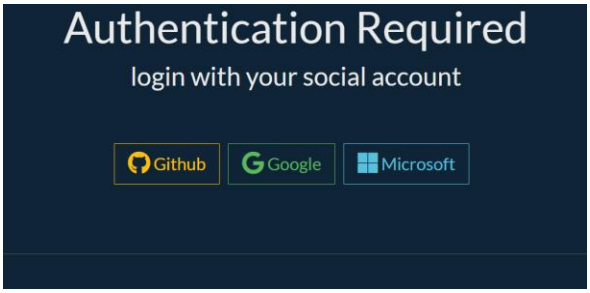
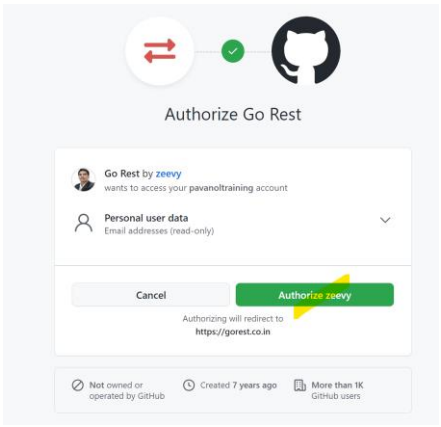
API Chaining

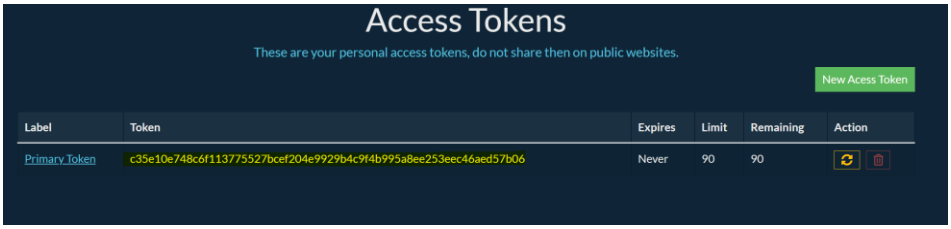
API chaining allows you to execute multiple API requests in sequence, where the output from one request is used as input for subsequent requests.

GoRest API (Users)

https://gorest.co.in/





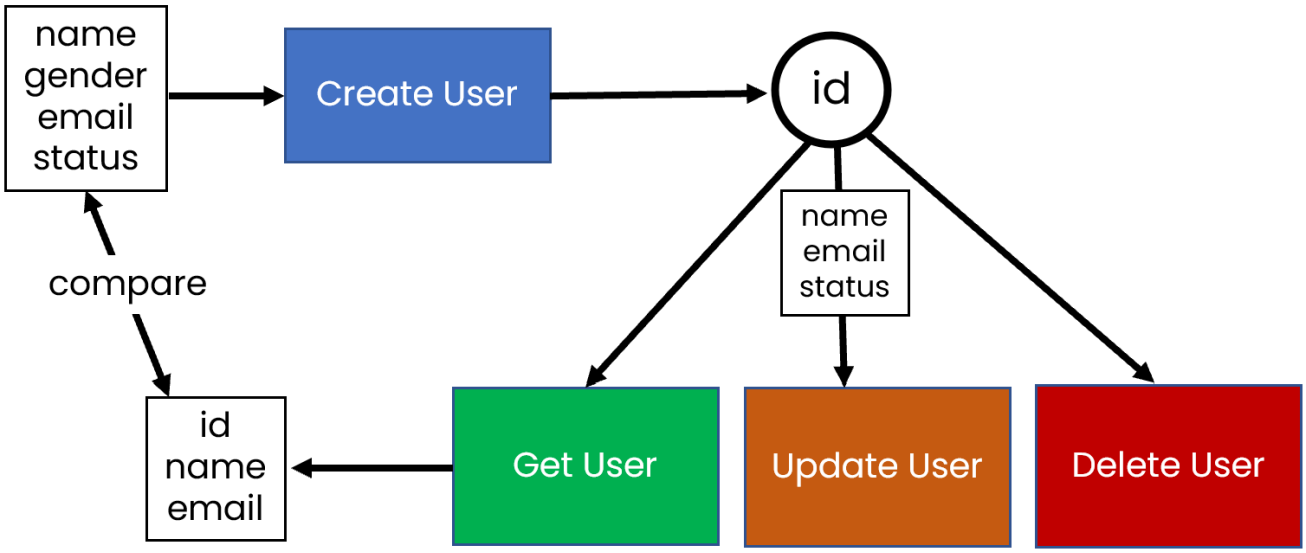


Access Token:

c35e10e[redacted]4c9f4b995a8ee253eec46aed57b06

https://gorest.co.in

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



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");
```

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);
```

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"
}
```

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");
```

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");
```

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

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

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

Objective:

- Use a script in the **Scripts** section to capture the id from the response and store it as an **environment variable**.

Test Script: 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.
   Request URL: http://localhost:3000/students/{{id}}
2. **Objective:**
   - Verify that the response contains the correct student details.
3. **Test Script :**
   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.
   Request URL: http://localhost:3000/students/{{id}}
2. **Objective:**
   - Confirm that the student has been successfully deleted.

Deliverables:

1. A Postman collection with:
   - **POST request** to create a student.
   - **GET request** to retrieve the student details.
   - **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.