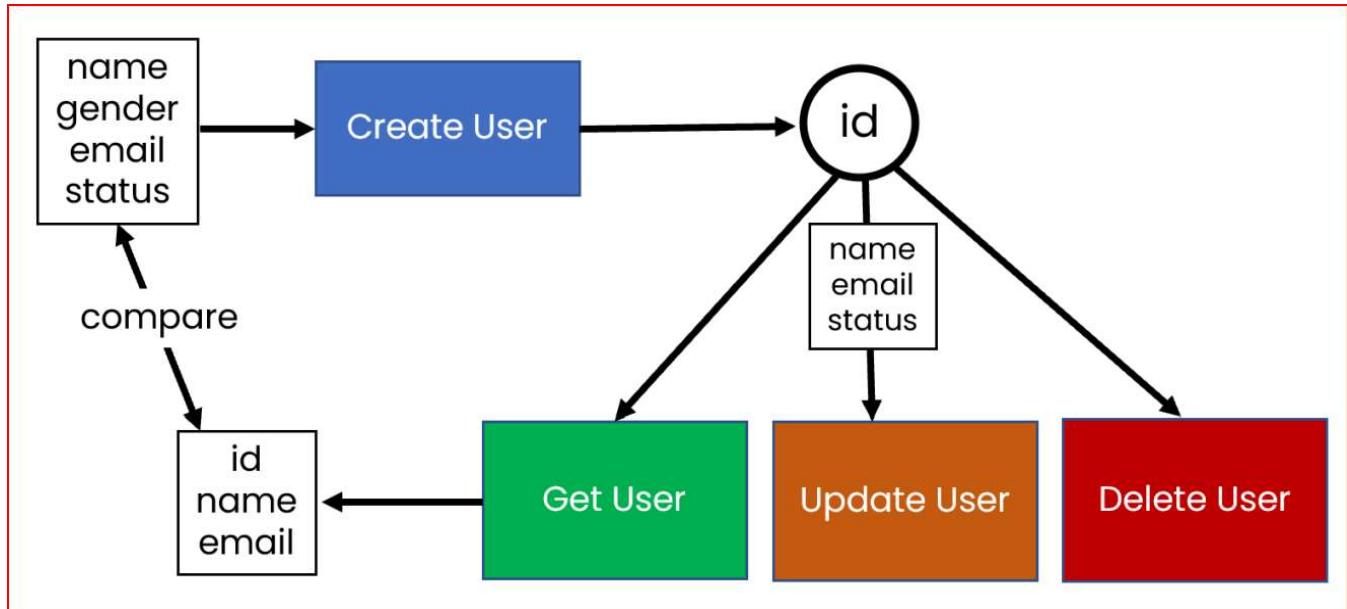


5. API Chaining

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.

Resources		Trying it Out	
https://gorest.co.in/public/v2/users	2929	POST /public/v2/users	Create a new user
https://gorest.co.in/public/v2/posts	2901	GET /public/v2/users/6945473	Get user details
https://gorest.co.in/public/v2/comments	2904	PUT PATCH /public/v2/users/6945473	Update user details
https://gorest.co.in/public/v2/todos	1407	DELETE /public/v2/users/6945473	Delete user

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**.

- Do not post your personal data like name, email, phone, photo etc...
- For paged results parameter "page" and "per_page" should be passed in url ex: GET /public/v2/users?page=1&per_page=20 (max 100 results per page)
- Request methods PUT, POST, PATCH, DELETE needs access token, which needs to be passed with "Authorization" header as Bearer token.
- API Versions /public-api/*, /public/v1/* and /public/v2/*
- [Get your access token](#)

Authentication Required

login with your social account



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

Access Tokens

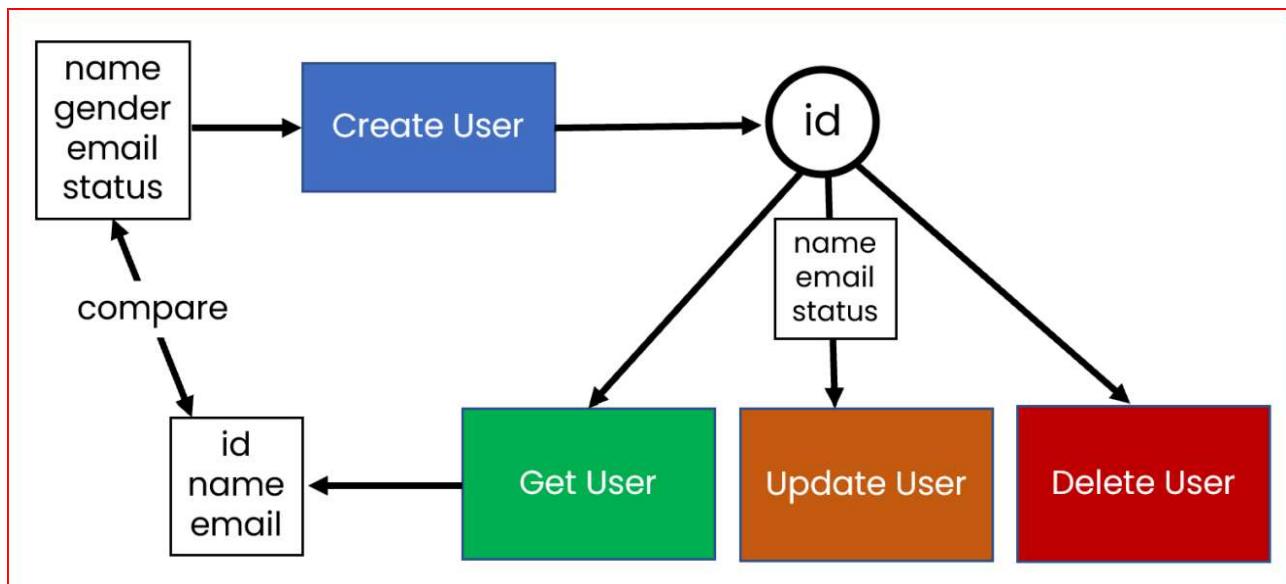
These are your personal access tokens, do not share them on public websites.

[New Access Token](#)

Label	Token	Expires	Limit	Remaining	Action
Primary Token	5c50315930957 [REDACTED] 12fcd79755949a9d3b0a	Never	90	90	

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.

Day5_APIChaining_GORestAPI

Run Fork

Overview Authorization Scripts Variables Runs

This authorization method will be used for every request in this collection. You can override this by specifying one in the request.

Auth Type

Bearer Token

The authorization header will be automatically generated when you send the request.
Learn more about [Bearer Token](#) authorization.

Info Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables.
Learn more about [variables](#).

Token

c35e10e748c6f113775527bcef204e...

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

HTTP Day5_APIChaining_GORestAPI / CreateUser

POST https://gorest.co.in/public/v2/users Send

Params Authorization Headers (9) Body Scripts Settings Cookies

Auth Type

Inherit auth from parent

Bearer Token

Token

c35e10e748c6f113775527bcef204e...

The authorization header will be automatically generated when you send the request. Learn more about [authorization](#).

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 CreateUser

HTTP Day5_APIChaining_GORestAPI / CreateUser

POST https://gorest.co.in/public/v2/users Send

Params Authorization Headers (9) Body Scripts Settings Cookies

Pre-request

```

1 pm.collectionVariables.set("username", "abcxyz");
2 pm.collectionVariables.set("useremail", "abcxyz@yahoo.com");
3
4
5
6
7
8
9
10

```

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

Body Cookies Headers (27) Test Results

201 Created 1.02 s 1.47 KB Save Response

{ } JSON Preview Visualize

```

1 {
2   "id": 7630965,
3   "name": "abcxyz",
4   "email": "nyp6eny37h@gmail.com",
5   "gender": "male",
6   "status": "inactive"
7 }

```

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

Body Cookies Headers (25) Test Results

422 Unprocessable Entity 225 ms 1.33 KB Save Response

{ } JSON Preview Visualize

```

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

```

→ Generate random text for email instead of duplicate email in the prerequest script of request and send the request

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

POST CreateUser

HTTP Day5_APIChaining_GORestAPI / CreateUser

POST https://gorest.co.in/public/v2/users

Send

Params Authorization Headers (9) Body Scripts Settings Cookies

Pre-request * Post-response *

```

1 // generate random text
2 const text=Math.random().toString(36).substring(2);
3 const email=text+"@gmail.com";
4
5 console.log(email);
6
7 pm.collectionVariables.set("username","abcxyz");
8 pm.collectionVariables.set("useremail",email);
9
10

```

→ Observe userid is also captured in Variables

Collection	
username	abcxyz
useremail	ivo8ksy2kn@gmail.com
userid	7630982

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.

The screenshot shows the Postman interface for a PUT request to `https://gorest.co.in/public/v2/users/{userid}`. The 'Pre-request' script is highlighted with a red box and contains the following code:

```
//generate random text
const text=Math.random().toString(36).substring(2);
const email=text+"@gmail.com";
console.log(email);
pm.collectionVariables.set("username","abcxyz123");
pm.collectionVariables.set("useremail",email);
```

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

The screenshot shows the Postman interface for a PUT request to `https://gorest.co.in/public/v2/users/{userid}`. The 'Post-response' script is highlighted with a red box and contains the following code:

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

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.

The screenshot shows the Postman Collection Runner interface. On the left, a sidebar lists collections: Day1_HTTPRequestsDemo, Day2_StudentsAPI, Day3_StudentAPIs_ResponseValidatio..., Day4_VariablesAndScripts, and Day5_APIChaining_GORestAPI. The Day5_APIChaining_GORestAPI collection is expanded, showing four requests: POST CreateUser, GET Get User, PUT Update User, and DELETE Delete User. The 'Run order' section on the right has checkboxes for all four requests, which are highlighted with a red box. The 'Run configuration' section includes fields for Iterations (set to 1), Delay (0 ms), and Data file (Select File). Below these are options for Persist responses for a session and Turn off logs during run. A large red box highlights the 'Run Day5_APIChaining_GORest...' button at the bottom.

The screenshot shows the Postman Run results interface for the 'Day5_APIChaining_GORestAPI' collection. At the top, it says 'Ran today at 04:48:21 · View all runs'. It features a 'Run Again' button and links for Automate Run, New Run, and Export Results. The main area is titled 'Day5_APIChaining_GORestAPI - Run results' and contains a 'RUN SUMMARY' table:

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	none	1	1s 786ms	2	299 ms

Below the table is a 'View Results' button. The results table lists the four requests with their status and count:

Request	Status	Count
POST CreateUser	1	0 0
▶ GET Get User	1	1 0
▶ PUT Update User	1	1 0
DELETE Delete User	0	0 0

Key Concepts

- Collection Variables:** Store dynamic data like username, useremail, and userid that can be shared across requests.
- Pre-request Script:** Code that runs before sending the request, used to set or update variables.
- Post-response Script:** Code that runs after the response, used to validate or extract data from the response.
- 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.