API Session – Chapter 9

- Variables In Request & Response -

# GoRest APIs

The GoRest Users API allows developers to manage user data through a set of RESTful endpoints. Below are the details of the available endpoints:

## Base URL

**URL:** <https://gorest.co.in/public/v2/users>

## Endpoints

## **Create a New User**

**Method:** POST

**Endpoint:** /public/v2/users

**Description:** This endpoint allows you to create a new user by sending user details in the request body.

**Example**

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

Content-Type: application/json

{

"name": "John Doe",

"email": "john.doe@example.com",

"gender": "male",

"status": "active"

}

## **Get User Details**

**Method:** GET

**Endpoint:** /public/v2/users/6940590

**Description:** Retrieves detailed information about the user with the specified ID (6940590 in this case).

**Example**

GET https://gorest.co.in/public/v2/users/6940590

## **Update User Details**

**Method:** PUT or PATCH

**Endpoint:** /public/v2/users/6940590

**Description:** Updates the details of the user with the specified ID. Use PUT to replace all user details or PATCH to update specific fields.

**Example**

PATCH https://gorest.co.in/public/v2/users/6940590

Content-Type: application/json

{

"status": "inactive"

}

## **Delete User Details**

**Method:** DELETE

**Endpoint:** /public/v2/users/6940590

**Description:** Deletes the user with the specified ID from the system.

**Example**

DELETE https://gorest.co.in/public/v2/users/6940590

# Random Data Generation

In Postman, a prerequisite script is executed before the actual request is sent. This script can be used to set up any data or conditions required for the request. A common use case is generating a random string that can be used dynamically in the request, such as for creating unique usernames, email addresses, or other identifiers.

## Example Scenario

Suppose you're testing an API that requires a unique email address for each request. You can use a prerequisite script to generate a random string and append it to a base email to ensure uniqueness.

## How to Generate a Random String in Postman Prerequisite Script

### **Define the Random String**

1. Use JavaScript functions within the script to generate a random string.
2. You can customize the length and characters used in the string.

### **Store the Random String**

1. Save the random string in a Postman environment variable using pm.environment.set() or pm.globals.set().

### **Use the Random String in the Request**

1. In the request body, headers, or URL, reference the random string by using the environment variable.

## Example

var randomEmail = **Math.random().toString(36).substring(2, 7)@example.com**

**Math.random().toString(36).substring(2, 7)**: Generates two random strings of 5 characters each using base 36 encoding (numbers and letters).

**@example.com**: Appends the domain to the generated email address.

**pm.environment.set("randomEmail",randomEmail);**

Once the Pre-request Script executes, the **randomEmail** environment variable will be available for use in the request body

## Using the Random String in the Request

After generating the random string, you can use the {{**randomEmail**}} variable in your request body

## Benefits

* **Uniqueness:** Ensures that each request has a unique identifier, avoiding conflicts in scenarios where duplicate data is not allowed.
* **Automation:** Facilitates automated testing by eliminating the need to manually change data for each request.
* **Flexibility:** Can be adapted for various uses, such as generating random numbers, UUIDs, or specific formats.

**Note:** Using the above approach, we can generate the random name as well.

# Response Assertion

**pm.test("User email matches stored email", function () {**

**var jsonData = pm.response.json();**

**var userEmail = jsonData.data.email; // Assuming email is under 'data.email' in response**

**var storedEmail = pm.environment.get("randomEmail");**

**pm.expect(userEmail).to.equal(storedEmail);**

**});**

## Accessing Response Data

* pm.response.json() parses the response body into a JavaScript object.
* jsonData.data.email extracts the user's email from the response. Adjust the path according to your API response structure.

## Accessing Stored Email

* pm.environment.get("randomEmail") retrieves the stored email from the environment variable.

## Assertion

* pm.expect(userEmail).to.equal(storedEmail) compares the extracted email with the stored email. If they match, the test passes.