

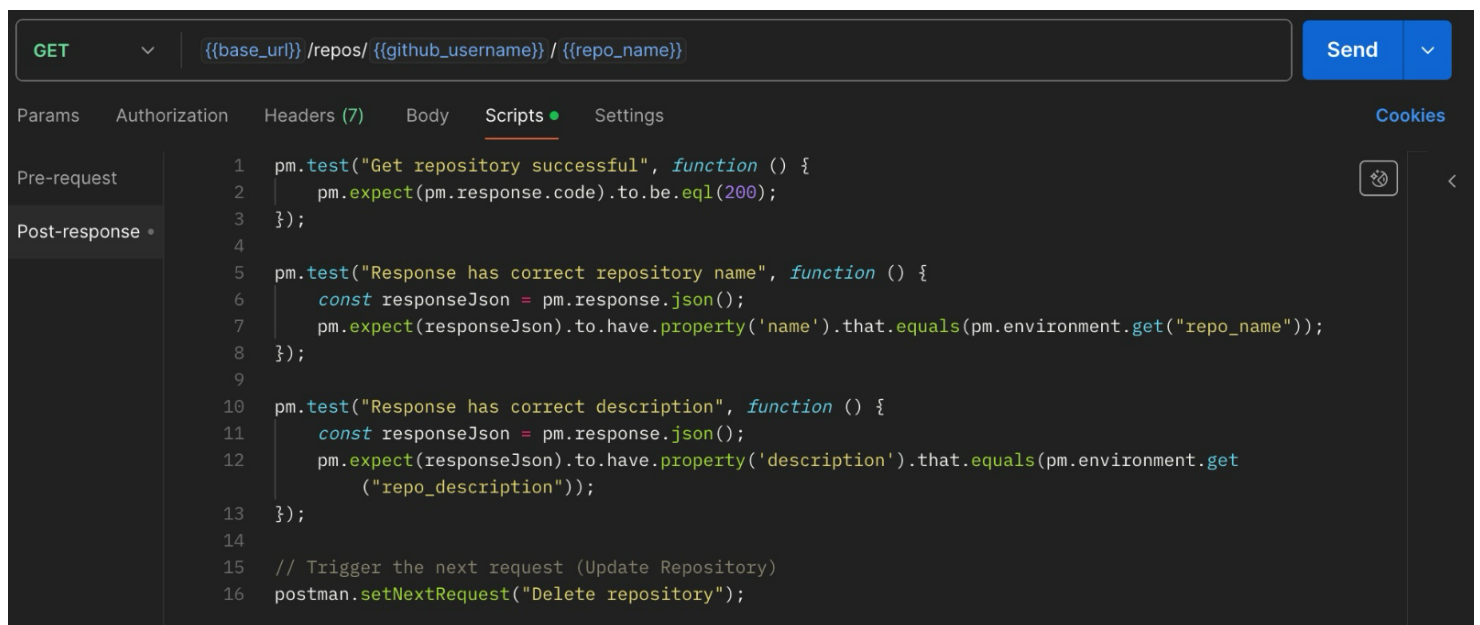
Report on Testing GitHub APIs Using Automation

Step-by-Step Approach to Automating GitHub API Testing

1. Setting Up the Environment

The first step was setting up the development environment. I used **Python** as the primary programming language and relied on libraries like `requests` for making API calls and `unittest` for structuring the test cases.

- I generated an API token from my GitHub account for authentication and securely stored it in environment variables to avoid hardcoding sensitive credentials.
- I also configured Postman as a manual testing tool before automating the process. This helped me understand how each endpoint behaved.



```
GET {{base_url}}/repos/{{github_username}}/{{repo_name}}

Params  Authorization  Headers (7)  Body  Scripts  Settings  Cookies

Pre-request
Post-response *

1  pm.test("Get repository successful", function () {
2    pm.expect(pm.response.code).to.be.eql(200);
3  });
4
5  pm.test("Response has correct repository name", function () {
6    const responseJson = pm.response.json();
7    pm.expect(responseJson).to.have.property('name').that.equals(pm.environment.get("repo_name"));
8  });
9
10 pm.test("Response has correct description", function () {
11   const responseJson = pm.response.json();
12   pm.expect(responseJson).to.have.property('description').that.equals(pm.environment.get
    ("repo_description"));
13 });
14
15 // Trigger the next request (Update Repository)
16 postman.setNextRequest("Delete repository");
```

2. Automating API Operations

GET Request – Fetching Repository Details

The first task was to automate the retrieval of repository information using a GET request.

- I implemented a script that sent a GET request to the endpoint:
`/repos/{username}/{repo_name}`.

- To validate the response, I checked for:
 - Status code: 200 OK
 - Repository properties like name and description.

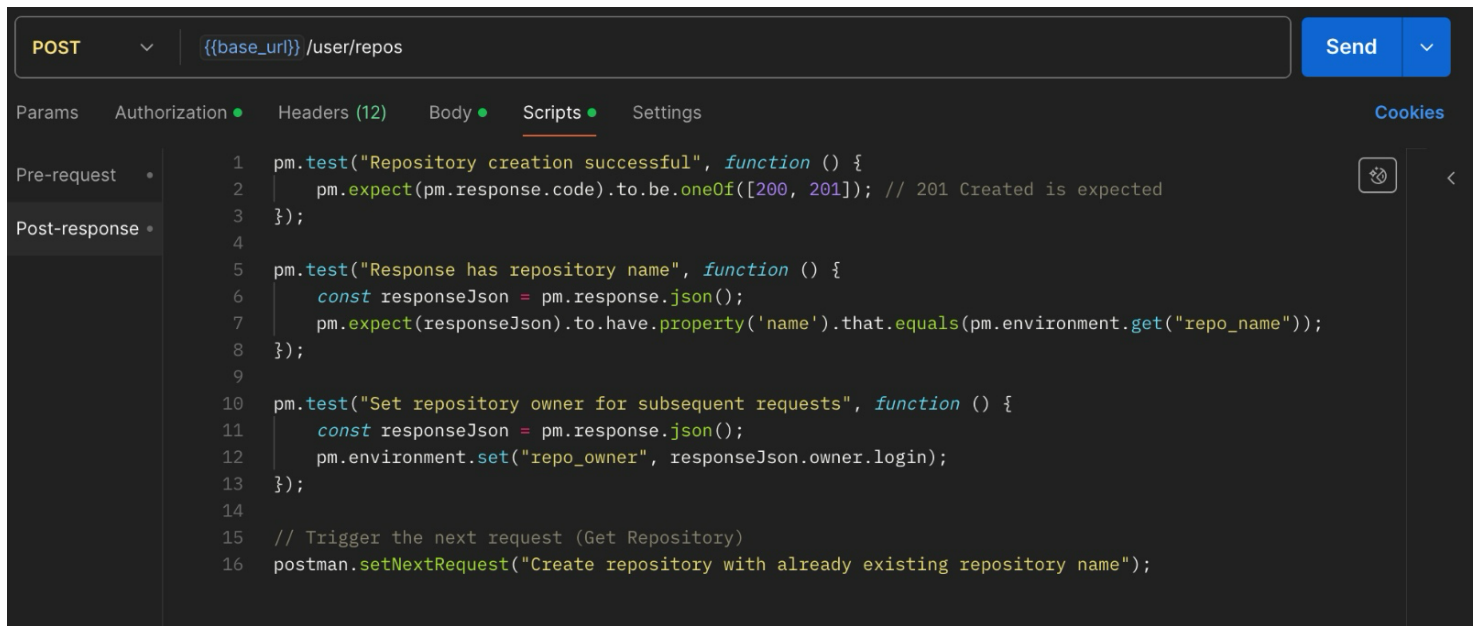
Challenges:

- **404 Error (Not Found):** Encountered when the repository didn't exist.

Solution: Added preconditions to ensure the repository name was correct.

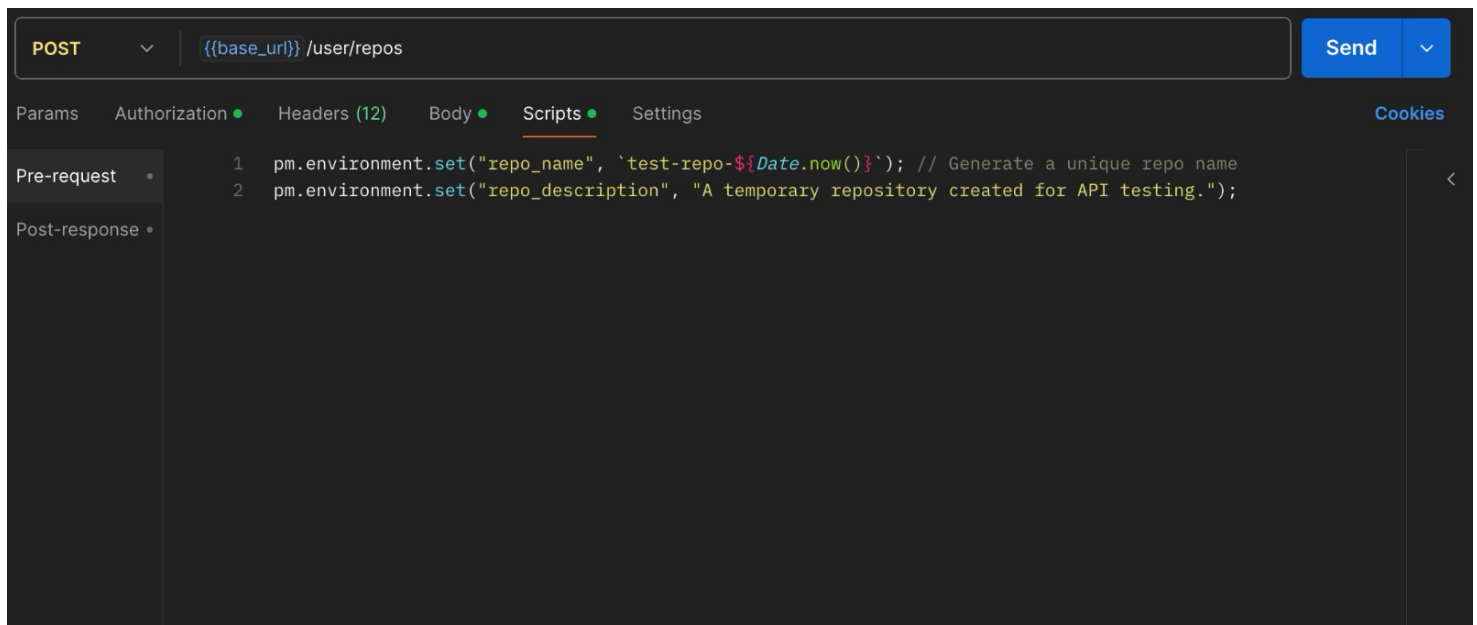
- **403 Error (Rate Limit):** Occurred after making too many requests.

Solution: Introduced delays using `time.sleep()` to comply with GitHub's rate limits.



The image shows the Postman interface for a POST request to the endpoint `{{base_url}}/user/repos`. The **Scripts** tab is active, displaying a JavaScript script for testing. The script includes tests for repository creation success, response JSON structure, and repository name verification, followed by setting the repository owner and triggering the next request.

```
1 pm.test("Repository creation successful", function () {
2   pm.expect(pm.response.code).to.be.oneOf([200, 201]); // 201 Created is expected
3 });
4
5 pm.test("Response has repository name", function () {
6   const responseJson = pm.response.json();
7   pm.expect(responseJson).to.have.property('name').that.equals(pm.environment.get("repo_name"));
8 });
9
10 pm.test("Set repository owner for subsequent requests", function () {
11   const responseJson = pm.response.json();
12   pm.environment.set("repo_owner", responseJson.owner.login);
13 });
14
15 // Trigger the next request (Get Repository)
16 postman.setNextRequest("Create repository with already existing repository name");
```



The image shows the Postman interface for a POST request to the endpoint `{{base_url}}/user/repos`. The **Scripts** tab is active, displaying a JavaScript script for setting environment variables. The script sets `repo_name` to a unique name using a timestamp and sets `repo_description` to a fixed string.

```
1 pm.environment.set("repo_name", `test-repo-${Date.now()}`); // Generate a unique repo name
2 pm.environment.set("repo_description", "A temporary repository created for API testing.");
```

POST Request – Creating a New Repository

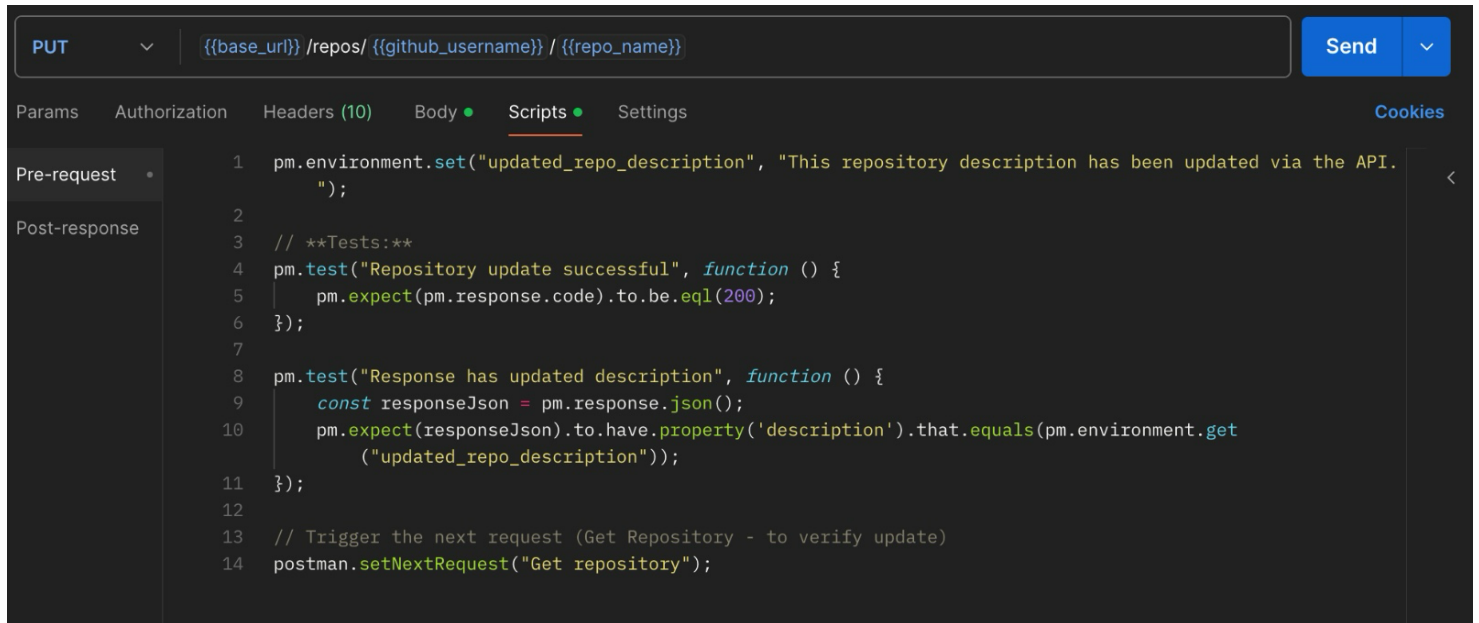
Next, I automated the creation of repositories using a POST request.

- The endpoint used was: `/user/repos`.
- A JSON payload was included in the request body with properties like name and description.
- To ensure dynamic testing, I generated unique repository names using timestamps in the script.
- Validation steps included checking the response for a 201 Created status and verifying the repository's existence with a GET request.

Challenges:

- **422 Error (Unprocessable Entity):** This happened when trying to create a repository with a name that already existed.

Solution: Added a cleanup step to delete existing repositories before re-running the test.



The screenshot shows a Postman interface for a PUT request. The URL is `{{base_url}} /repos/ {{github_username}} / {{repo_name}}`. The 'Scripts' tab is active, showing a sequence of tests and actions. The first test sets an environment variable for the updated description. The second test checks for a 200 status code and verifies the description was updated. Finally, it triggers the next request in the sequence.

```
1 pm.environment.set("updated_repo_description", "This repository description has been updated via the API.");
2
3 // **Tests:**
4 pm.test("Repository update successful", function () {
5   pm.expect(pm.response.code).to.be.eql(200);
6 });
7
8 pm.test("Response has updated description", function () {
9   const responseJson = pm.response.json();
10  pm.expect(responseJson).to.have.property('description').that.equals(pm.environment.get("updated_repo_description"));
11 });
12
13 // Trigger the next request (Get Repository - to verify update)
14 postman.setNextRequest("Get repository");
```

PUT Request – Updating Repository Details

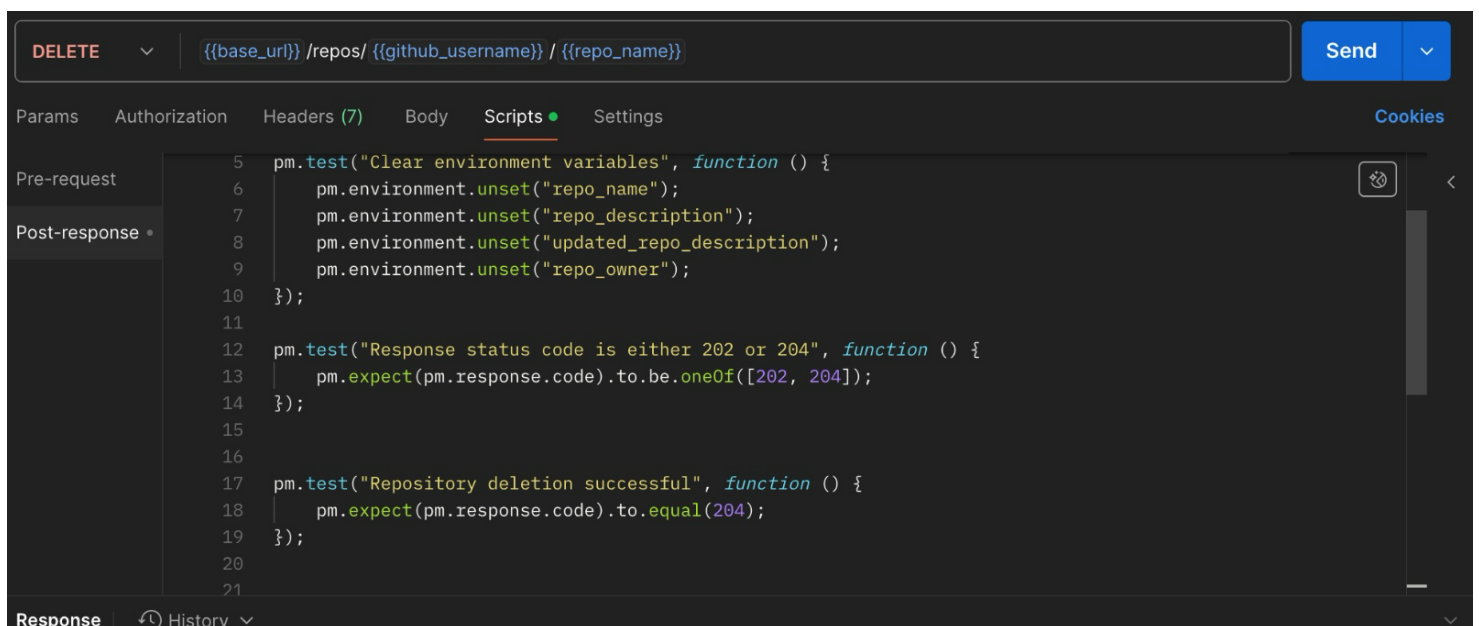
The PUT request was used to modify repository details, such as updating the description.

- The endpoint used was: `/repos/{username}/{repo_name}`.
- The request body included fields like:

"description": "Updated repository description"

}

Response validation checked for a 200 OK status and confirmed changes by fetching the updated details with a GET request.



The screenshot shows a Postman interface for a DELETE request. The URL is `{{base_url}} /repos/ {{github_username}} / {{repo_name}}`. The 'Scripts' tab is active, showing tests to clear environment variables, check for a 202 or 204 status code, and confirm successful deletion.

```
5 pm.test("Clear environment variables", function () {
6   pm.environment.unset("repo_name");
7   pm.environment.unset("repo_description");
8   pm.environment.unset("updated_repo_description");
9   pm.environment.unset("repo_owner");
10 });
11
12 pm.test("Response status code is either 202 or 204", function () {
13   pm.expect(pm.response.code).to.be.oneOf([202, 204]);
14 });
15
16
17 pm.test("Repository deletion successful", function () {
18   pm.expect(pm.response.code).to.equal(204);
19 });
20
21
```

DELETE Request – Removing a Repository

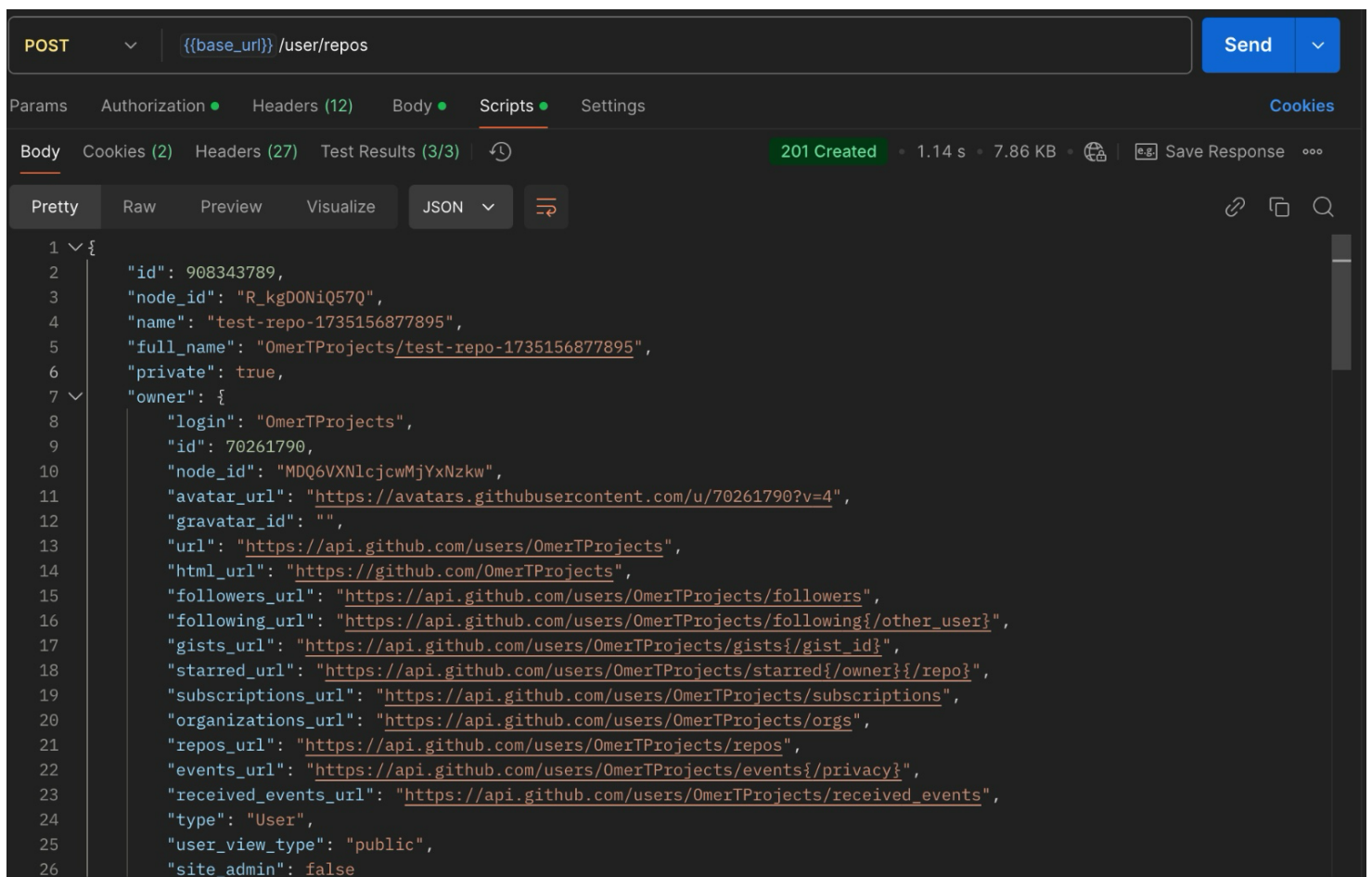
The DELETE request was used to permanently delete a repository.

- Endpoint: `/repos/{username}/{repo_name}`.
- No request body was required.
- Validation involved confirming a 204 No Content status and verifying that the repository was no longer accessible.

Challenges:

- **404 Error:** This occurred when the repository to be deleted didn't exist.

Solution: Ensured the repository was created before testing deletion.



The screenshot shows a REST client interface with a POST request to `{{base_url}}/user/repos`. The response is a 201 Created status, indicating a new repository was successfully created. The response body is a JSON object containing repository details.

```
1 {
2   "id": 908343789,
3   "node_id": "R_kgDONiQ57Q",
4   "name": "test-repo-1735156877895",
5   "full_name": "OmerTProjects/test-repo-1735156877895",
6   "private": true,
7   "owner": {
8     "login": "OmerTProjects",
9     "id": 70261790,
10    "node_id": "MDQ6VXNlcjcwMjYxNzkw",
11    "avatar_url": "https://avatars.githubusercontent.com/u/70261790?v=4",
12    "gravatar_id": "",
13    "url": "https://api.github.com/users/OmerTProjects",
14    "html_url": "https://github.com/OmerTProjects",
15    "followers_url": "https://api.github.com/users/OmerTProjects/followers",
16    "following_url": "https://api.github.com/users/OmerTProjects/following{/other_user}",
17    "gists_url": "https://api.github.com/users/OmerTProjects/gists{/gist_id}",
18    "starred_url": "https://api.github.com/users/OmerTProjects/starred{/owner}/{/repo}",
19    "subscriptions_url": "https://api.github.com/users/OmerTProjects/subscriptions",
20    "organizations_url": "https://api.github.com/users/OmerTProjects/orgs",
21    "repos_url": "https://api.github.com/users/OmerTProjects/repos",
22    "events_url": "https://api.github.com/users/OmerTProjects/events{/privacy}",
23    "received_events_url": "https://api.github.com/users/OmerTProjects/received_events",
24    "type": "User",
25    "user_view_type": "public",
26    "site_admin": false
```

2. Automating API Operations

Scenario 1: Checking if GitHub Website is Up

To test GitHub's website availability, I used the status.github.com/api/status.json endpoint. This endpoint provides real-time information about GitHub's status.

- **Steps:**
- Sent a GET request to the API endpoint: <https://www.githubstatus.com/api/v2/status.json>.

- Checked the status field in the response to ensure the site was operational.
- **Validation:**
- Expected status value: "operational".
- Response code: 200 OK.
- **Challenges:**
- Network issues occasionally lead to timeouts.
- *Solution:* Implemented retry logic in case of failed requests.