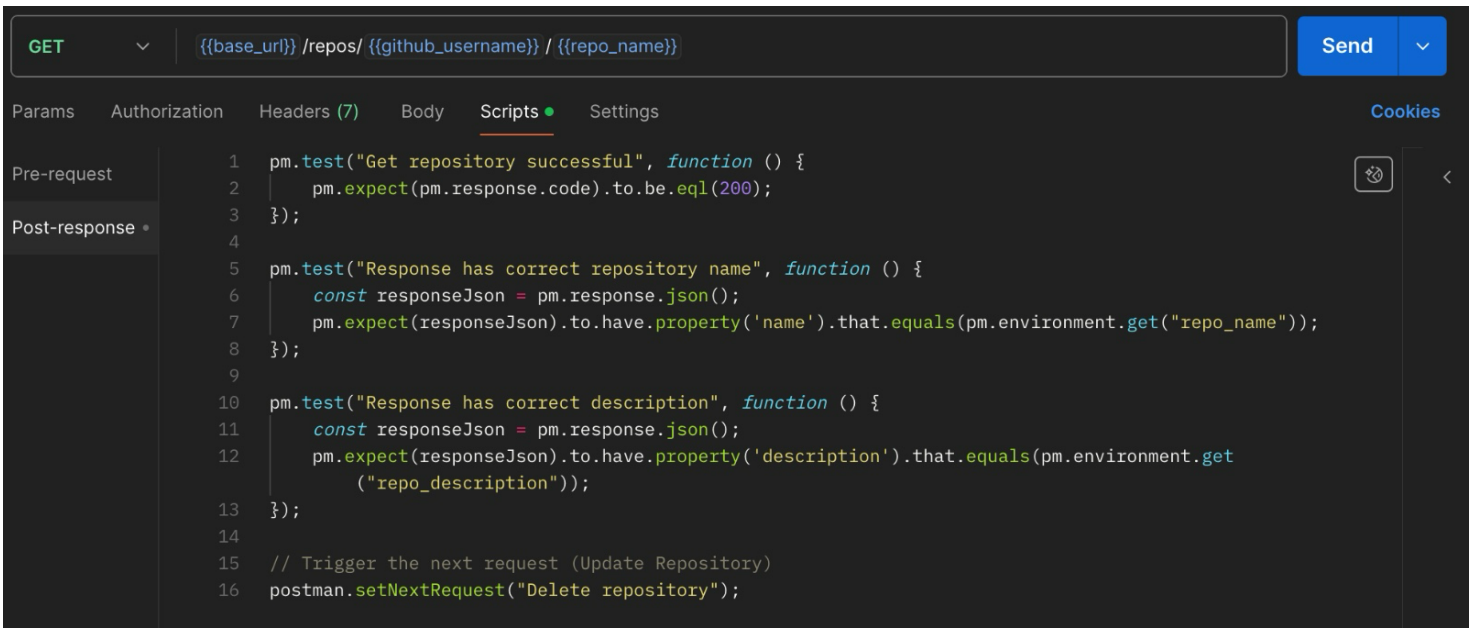
Report on Testing GitHub APIs Using Automation
Step-by-Step Approach to Automating GitHub API Testing

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.



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.

POST {{base_url}} /user/repos Send

Params Authorization Headers (12) Body Scripts Settings Cookies

Pre-request

Post-response

```
pm.test("Repository creation successful", function () {
    pm.expect(pm.response.code).to.be.oneOf([200, 201]); // 201 Created is expected
});

pm.test("Response has repository name", function () {
    const responseJson = pm.response.json();
    pm.expect(responseJson).to.have.property('name').that.equals(pm.environment.get("repo_name"));
});

pm.test("Set repository owner for subsequent requests", function () {
    const responseJson = pm.response.json();
    pm.environment.set("repo_owner", responseJson.owner.login);
});

// Trigger the next request (Get Repository)
postman.setNextRequest("Create repository with already existing repository name");
```

POST {{base_url}} /user/repos Send

Params Authorization Headers (12) Body Scripts Settings Cookies

Pre-request

Post-response

```
pm.environment.set("repo_name", `test-repo-${Date.now()}`); // Generate a unique repo name
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.

PUT {{base_url}} /repos/ {{github_username}} / {{repo_name}} Send

Params Authorization Headers (10) Body Scripts Settings Cookies

Pre-request

Post-response

```
pm.environment.set("updated_repo_description", "This repository description has been updated via the API.
    ");

// **Tests:**
pm.test("Repository update successful", function () {
    pm.expect(pm.response.code).to.be.eql(200);
});

pm.test("Response has updated description", function () {
    const responseJson = pm.response.json();
    pm.expect(responseJson).to.have.property('description').that.equals(pm.environment.get
        ("updated_repo_description"));
});

// Trigger the next request (Get Repository - to verify update)
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.

DELETE {{base_url}} /repos/ {{github_username}} / {{repo_name}} Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Pre-request

Post-response

```
pm.test("Clear environment variables", function () {
    pm.environment.unset("repo_name");
    pm.environment.unset("repo_description");
    pm.environment.unset("updated_repo_description");
    pm.environment.unset("repo_owner");
});

pm.test("Response status code is either 202 or 204", function () {
    pm.expect(pm.response.code).to.be.oneOf([202, 204]);
});


pm.test("Repository deletion successful", function () {
    pm.expect(pm.response.code).to.equal(204);
});
```

Response History

DELETE Request – Removing a Repository

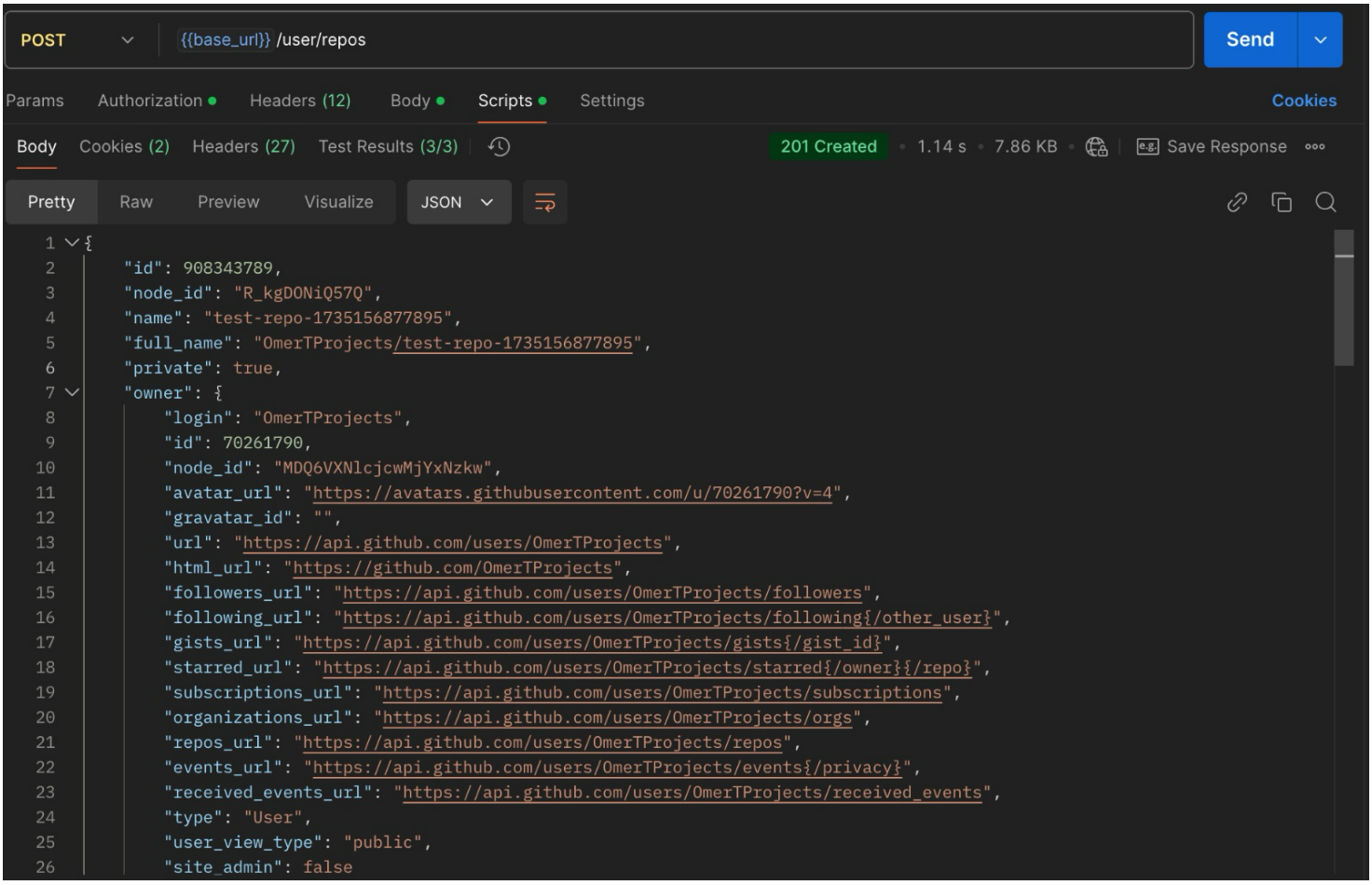
The DELETE request was used to permanently delete a repository.

- 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.



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.