

Exploring and Testing APIs with Postman

1. Introduction

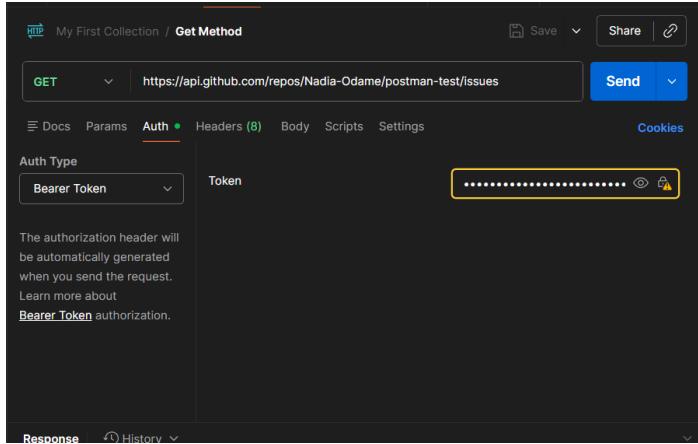
This assignment focussed on exploring and testing a real-world API using Postman with emphasis on authentication, CRUD operations, environment variables and automation. For this task, the **GitHub Rest API** (Issues endpoint) was selected because it supports authenticated access and full CRUD operations making it suitable for practical API testing and documentation.

The work was carried out entirely in Postman using a Personal Access Token (PAT) for authentication. All requests were tested, responses verified and environment variables used to make the workflow dynamic and reusable.

2. API Choice and Base URL

- **API Used:** Github Rest API -Issues
- **Base URL:** <https://api.github.com>
- **Repository Used:** Nadia-Odame/postman-test
- **Primary Resources:** Issues

This API allows authenticated users to create, read, update, label, comment on and delete issue related resources.

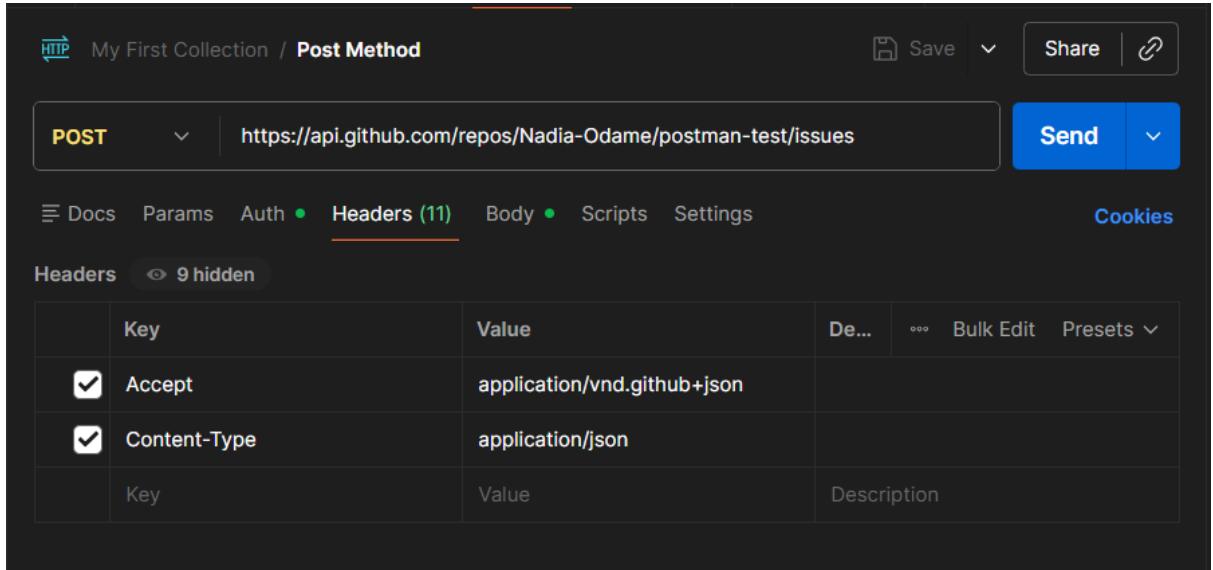


3. Authentication Method

Authentication Type: Personal Access Token (Bearer Token)

Authentication was handled using a GitHub Personal Access Token

stored securely as an environment variable in Postman.



The screenshot shows the Postman interface with a collection named "My First Collection" and a method named "Post Method". The request is a POST to <https://api.github.com/repos/Nadia-Odame/postman-test/issues>. The "Headers" tab is selected, displaying two headers: "Accept" with value "application/vnd.github+json" and "Content-Type" with value "application/json". Other tabs like "Params", "Auth", "Body", "Scripts", and "Settings" are also visible.

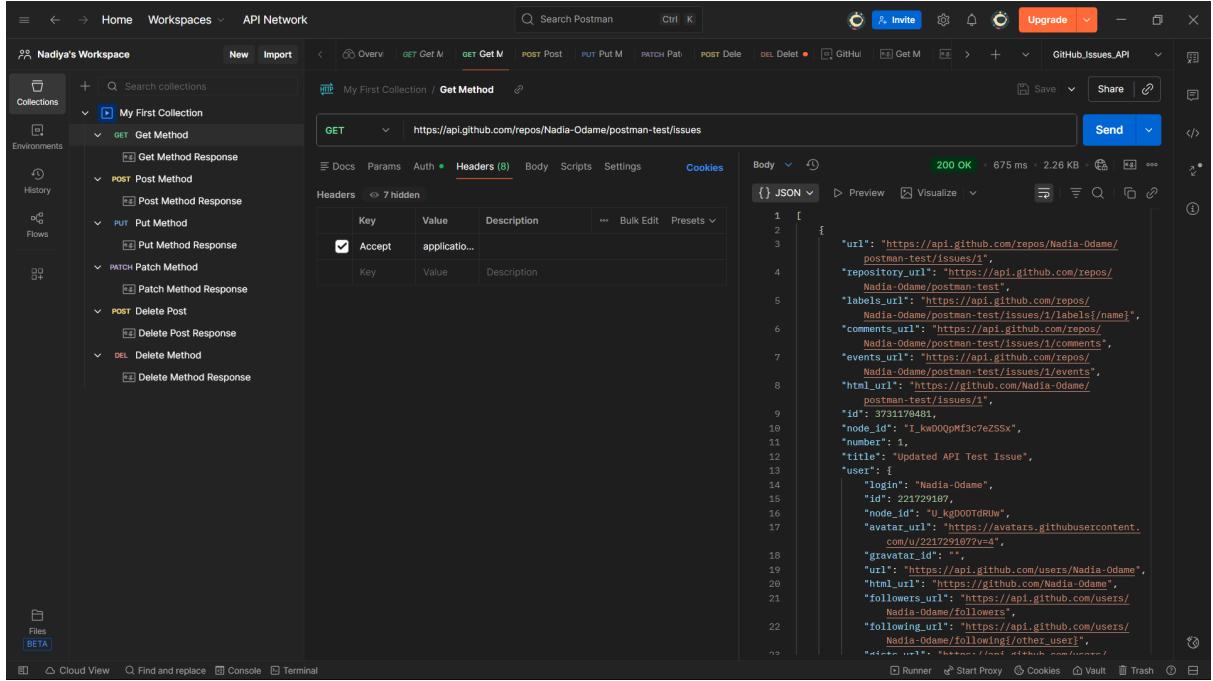
Headers Used in All Requests:

- **Authorization:** Bearer {{auth_token}}
- **Accept:** application/vnd.github+json
- **Content-Type:** application/json

The token was manually generated from GitHub and stored in the Postman environment as auth_token. This token was then reused across all requests causing variable interpolation. (Token automation using pre-request script was not implemented because GitHub PATs are generated externally. However, token reuse via environment variables was successfully demonstrated.)

4. CRUD Operations Demonstration

4.1. GET - Retrieve Issues (READ)



The screenshot shows the Postman application interface. On the left, the sidebar displays 'Nadia's Workspace' with collections like 'My First Collection' and methods including 'GET Get Method'. The main workspace shows a 'Get Method' request for 'https://api.github.com/repos/Nadia-Odame/postman-test/issues'. The 'Headers' tab is selected, showing an 'Accept' header with the value 'application/json'. The 'Body' tab shows a JSON response with a single issue object. The response body is as follows:

```
1 [ { 2 "url": "https://api.github.com/repos/Nadia-Odame/postman-test/Issues/1", 3 "repository_url": "https://api.github.com/repos/Nadia-Odame/postman-test", 4 "labels_url": "https://api.github.com/repos/Nadia-Odame/postman-test/Issues/1/labels{/name}", 5 "comments_url": "https://api.github.com/repos/Nadia-Odame/postman-test/Issues/1/comments", 6 "events_url": "https://api.github.com/repos/Nadia-Odame/postman-test/Issues/1/events", 7 "html_url": "https://github.com/Nadia-Odame/postman-test/Issues/1", 8 "id": 3731178481, 9 "node_id": "1_kw0Qpff3c7eZSSx", 10 "number": 1, 11 "title": "Updated API Test Issue", 12 "user": { 13 "login": "Nadia-Odame", 14 "id": 221729187, 15 "node_id": "U_kgDO0TtR0w", 16 "avatar_url": "https://avatars.githubusercontent.com/u/221729187?v=4", 17 "gravatar_id": "", 18 "url": "https://api.github.com/users/Nadia-Odame", 19 "followers_url": "https://api.github.com/users/Nadia-Odame/followers", 20 "following_url": "https://api.github.com/users/Nadia-Odame/following{/other_user}", 21 "created_at": "2017-01-23T10:10:47Z", 22 "updated_at": "2017-01-23T10:10:47Z" } } ]
```

Endpoint:

GET /repos/{owner}/{repo}/issues

Full URL:

<https://api.github.com/repos/Nadia-Odame/postman-test/issues>

Request Body: None

Response:

- Returned an array of issues in JSON format
- Initially returned an empty array ([]) before any issues were created

This request confirmed successful authentication and access to repository data.

4.2. POST - Create an Issue (CREATE)

Endpoint:

POST /repos/{owner}/{repo}/issues

Full URL:

<https://api.github.com/repos/Nadia-Odame/postman-test/issues>

Request Body:

```
{
  "title": "Test issue from Postman",
  "body": "This issue was created using the GitHub API via Postman."
}
```

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Nadja's Workspace' containing collections like 'My First Collection' with various methods (Get, Post, Put, Patch, Delete) and their corresponding responses. The main area shows a 'Post Method' for the URL `https://api.github.com/repos/Nadia-Odame/postman-test/issues`. The 'Body' tab is selected, showing a JSON payload:

```

1 {
2   "title": "Test issue from Postman",
3   "body": "This issue was created using the GitHub API via
4   Postman."
5 }
```

On the right, the response details are shown: '201 Created' status, 698 ms duration, and 3.67 KB size. Below the response, the detailed JSON response is displayed, which includes the issue's URL, repository URL, labels, comments, events, and user information.

Response Highlights:

- HTTP Status: 201 created
- Issue number automatically generated (#1)
- Issue state set to open
 - `Html_url` provided for viewing the issue on GitHub

This confirmed successful creation of a new resource via the API.

4.3. PUT - Add Labels to an Issue (UPDATE)

The screenshot shows the Postman application interface. On the left, the sidebar displays 'Nadya's Workspace' with various collections and environments. The main workspace shows a collection named 'My First Collection' with a single 'Put Method'. The request details pane shows a 'PUT' method at the URL <https://api.github.com/repos/Nadia-Odame/postman-test/issues/1/labels>. The 'Body' tab is selected, showing the JSON payload:

```
[ "bug", "api-test" ]
```

. The response pane shows a successful 200 OK status with a response body containing two label objects:

```
1 [ { 2   "id": 9833778311, 3     "node_id": "L_kw000pMf3c8AAAACSIntRw", 4     "url": "https://api.github.com/repos/Nadia-Odame/postman-test/labels/bug", 5       "name": "bug", 6       "color": "d73ada", 7       "default": true, 8       "description": "Something isn't working" }, 9   { 10    "id": 9834024234, 11      "node_id": "L_kw000pMf3c8AAAACSidNkg", 12      "url": "https://api.github.com/repos/Nadia-Odame/postman-test/labels/api-test", 13        "name": "api-test", 14        "color": "edded", 15        "default": false, 16        "description": null 17   } 18 } 19 ] 20 }
```

Endpoint:

PUT /repos/{owner}/{repo}/issues/{issue_number}/labels

Full URL:

<https://api.github.com/repos/Nadia-Odame/postman-test/issues/1/labels>

Request Body:

`["bug", "api-test"]`

Response Highlights:

- Returned an array of label objects
- Confirmed that both labels were successfully attached to the issue

This request demonstrated updating an existing resource by associating metadata (labels).

4.4. PATCH - Update Issue Title (UPDATE)

Endpoint:

PATCH /repos/{owner}/{repo}/issues/{issue_number}

Full URL:

<https://api.github.com/repos/Nadia-Odame/postman-test/issues/1>

Request body:

{

`"title": "Updated API Test Issue",`

```

    "state": "open"
}

```

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Nadiya's Workspace' containing collections like 'My First Collection' with various methods (GET, POST, PUT, PATCH) and their responses. The main area shows a 'PATCH' request to the URL <https://api.github.com/repos/Nadia-Odame/postman-test/issues/1>. The 'Body' tab is selected, showing the JSON payload:

```

1 {
2   "title": "Updated API Test Issue",
3   "state": "open"
4 }

```

The response on the right is a 200 OK status with a timestamp of 587 ms and a size of 2.25 KB. The response body is also shown in JSON format:

```

1 {
2   "url": "https://api.github.com/repos/Nadia-Odame/postman-test/issues/1",
3   "repository_url": "https://api.github.com/repos/Nadia-Odame/postman-test",
4   "labels_url": "https://api.github.com/repos/Nadia-Odame/postman-test/issues/1/labels{/name}",
5   "comments_url": "https://api.github.com/repos/Nadia-Odame/postman-test/issues/1/comments",
6   "events_url": "https://api.github.com/repos/Nadia-Odame/postman-test/issues/1/events",
7   "html_url": "https://github.com/Nadia-Odame/postman-test/issues/1",
8   "id": 3731179481,
9   "node_id": "U_kwQ0ppMf3c7ezSSx",
10  "number": 1,
11  "title": "Updated API Test Issue",
12  "user": {
13    "login": "Nadia-Odame",
14    "id": 221729107,
15    "node_id": "U_kg00TrdRUw",
16    "avatar_url": "https://avatars.githubusercontent.com/u/221729107?v=4",
17    "gravatar_id": "",
18    "url": "https://api.github.com/users/Nadia-Odame",
19    "html_url": "https://github.com/Nadia-Odame",
20    "followers_url": "https://api.github.com/users/Nadia-Odame/followers",
21    "following_url": "https://api.github.com/users/Nadia-Odame/following{/other_user}",
22    "gists_url": "https://api.github.com/users/Nadia-Odame/gists{/gist_id}"
}

```

Response Highlights:

- Issue title updated successfully
- Labels remained unchanged
- `updated_at` timestamp reflected the modification

This showed partial updates using the PATCH method.

4.5. DELETE - Delete a Comment (DELETE)

GitHub does not allow deleting issues via API. Instead, deletion was demonstrated using issue comments which still fulfills the DELETE operation requirement.

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Nadja's Workspace' containing collections like 'My First Collection' and various method types (GET, POST, PUT, PATCH, DELETE). The main area shows a POST request to <https://api.github.com/repos/Nadia-Odame/postman-test/issues/1/comments>. The 'Pre-request' script contains the following code:

```

pm.test("Comment created", function () {
    pm.response.to.have.status(201);
});

```

The 'Post-res' tab shows the JSON response body:

```

{
  "id": 3668764423,
  "node_id": "NC_kw00Qpmf3c7eMtUH",
  "user": {
    "login": "Nadia-Odame",
    "id": 221729187,
    "node_id": "U_kg000IdR0w",
    "avatar_url": "https://avatars.githubusercontent.com/u/221729187?v=4",
    "gravatar_id": "",
    "url": "https://api.github.com/users/Nadia-Odame",
    "html_url": "https://github.com/Nadia-Odame",
    "followers_url": "https://api.github.com/users/Nadia-Odame/followers",
    "following_url": "https://api.github.com/users/Nadia-Odame/following{/other_user}",
    "gists_url": "https://api.github.com/users/Nadia-Odame/gists{/gist_id}",
    "starred_url": "https://api.github.com/users/Nadia-Odame/starred{/owner}{/repo}",
    "subscriptions_url": "https://api.github.com/users/Nadia-Odame/subscriptions",
    "organizations_url": "https://api.github.com/users/Nadia-Odame/orgs",
    "repos_url": "https://api.github.com/users/Nadia-Odame/repos",
    "events_url": "https://api.github.com/users/Nadia-Odame/events",
    "received_events_url": "https://api.github.com/users/Nadia-Odame/received_events"
}

```

The status bar at the bottom indicates a successful 201 Created response.

Step 1: Create a Comment

Endpoint:

POST /repos/{owner}/{repo}/issues/{issue_number}/comments

Full URL:

<https://api.github.com/repos/Nadia-Odame/postman-test/issues/1/comments>

Test Script Used:

```

pm.test("Comment created", function () {
    pm.response.to.have.status(201);
});

```

```

var jsonData = pm.response.json();
pm.environment.set("comment_id", jsonData.id);

```

This script verified successful creation and automatically stored the `comment_id`.

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (My First Collection), 'Environments' (GitHub_Issues_API), 'History', 'Flows', and 'Files'. The main area shows a collection named 'My First Collection' with several methods: 'Get Method', 'Post Method', 'Put Method', 'Patch Method', 'Delete Post', and 'Delete Method'. A specific 'DELETE' request is selected, pointing to the URL <https://api.github.com/repos/Nadia-Odame/postman-test/issues/comments/3660764423>. The request body is set to 'none'. The response status is '204 No Content' with a duration of '843 ms' and a size of '1.05 KB'. Below the main interface, there are tabs for 'Runner', 'Start Proxy', 'Cookies', 'Vault', and 'Trash'.

Step 2: Delete the Comment

Endpoint:

`DELETE /repos/{owner}/{repo}/issues/comments/{comment_id}`

Full URL:

https://api.github.com/repos/Nadia-Odame/postman-test/issues/comments/{comment_id}

Response Highlights:

- HTTP Status: 204 No content
- Comment successfully deleted

This demonstrated a complete create and delete workflow using dynamic variables.

5. Postman Environment & Variables

A Postman environment (GitHub_Issues_API) was created with the following variables:

Variable Name	Purpose
base_url	Stores the Github API base URL

owner	Repository owner
repo	Repository name
auth_token	GitHub Personal Access Token
issue_number	Issue Identifier
comment_id	Comment identifier (auto-filled)

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (selected), 'Environments' (selected), 'History', 'Flows', and 'Files [BETA]'. The main area shows 'Nadila's Workspace' with a 'Globals' section and a 'GitHub_Issues_API' collection. The 'GitHub_Issues_API' collection has a table of environment variables:

Variable	Value
base_url	https://api.github.com
owner	Nadia-Odame
repo	postman-test
auth_token	ghp_Ge1ky7y4g9BdVrlZdvP43tN7Rc9Oz23NH4H...
comment_id	3660764423

At the bottom, there are tabs for 'Cloud View', 'Find and replace', 'Console', and 'Terminal'. A small 'Add a device' dialog is open in the bottom right corner.

Using environment variables improved security, reduced duplication and made requests reusable across collections.

6. Automation and Testing

- Basic Test scripts were added to validate response status codes.
- Dynamic data (comment ID) was captured and reused automatically
- Requests were structured to allow execution in sequence using the Collection Runner

7. Conclusion

This assignment successfully demonstrated authenticated API interaction using Postman and the GitHub Rest API. All core CRUD operations were implemented and verified using real endpoints. Environment variables and basic automation improved efficiency and showed practical understanding of API testing workflows.