

Introduction

For this assignment, I chose to work with the GitHub REST API because it provides real-world authentication, supports full CRUD (Create, Read, Update, Delete) operations, and is widely used in software development. The API base URL is <https://api.github.com>.

Objectives:

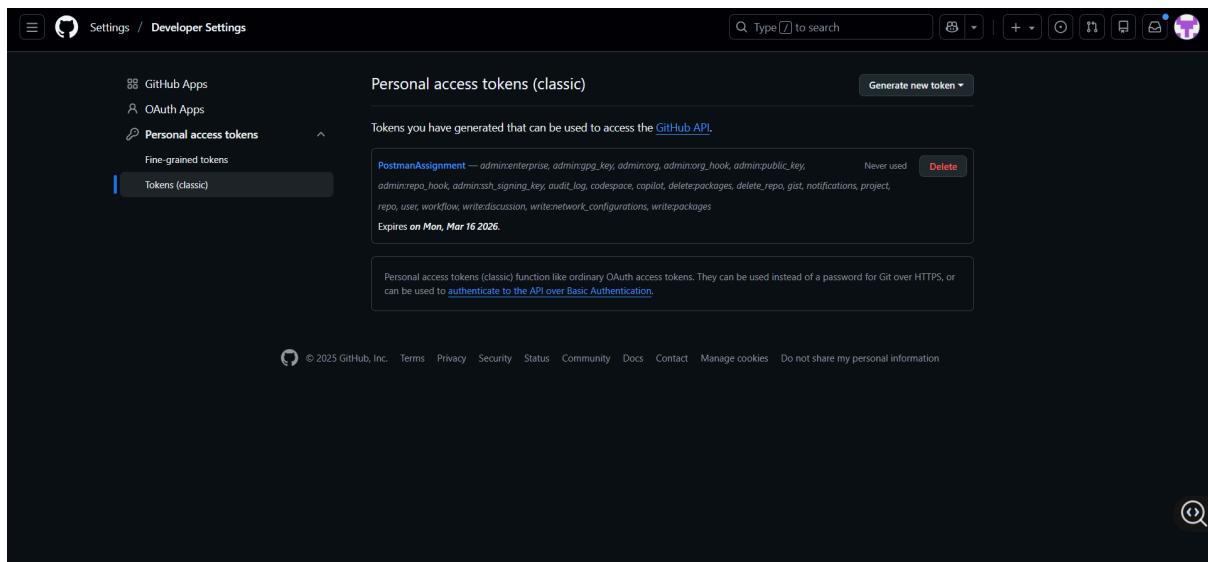
1. Understand API authentication using tokens
2. Implement CRUD operations (Create, Read, Update, Delete)
3. Use Postman environments and variables for dynamic testing
4. Automate testing with Postman Collection Runner
5. Document the complete workflow

Authentication Setup

I used Personal Access Token authentication with the Bearer Token method.

Steps Taken:

1. Generated a token from GitHub → Settings → Developer Settings
2. Selected repo and user scopes for necessary permissions
3. Stored the token securely in Postman environment variables as auth_token
4. Configured all requests to use: Authorization: Bearer {{auth_token}}



Screenshot 1: GitHub token generation page

API Endpoints Documentation

Endpoint	Method	Purpose	Headers Required	Body Format
/user	GET	Get authenticated user info	Authorization: Bearer {token}	None
/user/repos	POST	Create new repository	Authorization: Bearer {token}	JSON: name, description, private
/repos/{owner}/{repo}	PATCH	Update repository	Authorization: Bearer {token}	JSON: updated fields
/repos/{owner}/{repo}	DELETE	Delete repository	Authorization: Bearer {token}	None

CRUD Operations Implementation

READ (GET)

Request: GET {{base_url}}/user

Response: 200 OK with user profile information

Verified: Authentication working correctly

The screenshot shows two separate instances of the Postman application interface, both displaying a GET request to `https://api.github.com/user`.

Top Instance (Successful Request):

- Request URL:** `https://api.github.com/user`
- Headers:** (7 items) - Authorization, Content-Type, User-Agent, Host, Connection, Accept, Cache-Control.
- Script (Post-response):**

```
1 // Test 1: Check if response status is 200
2 pm.test("Status code is 200", function () {
3     pm.response.to.have.status(200);
4 });
5
6 // Test 2: Check if response has your login/username
7 pm.test("Response has login field", function () {
8     var jsonData = pm.response.json();
9     pm.expect(jsonData.login).to.be.a('string');
10 });
11
12 // Test 3: Check if response has your ID
13 pm.test("Response has ID", function () {
14     var jsonData = pm.response.json();
15     pm.expect(jsonData.id).to.be.a('number');
16 });
```
- Test Results:** 200 OK, 550 ms, 2.24 KB, Save Response
- Body (JSON Preview):**

```
{
  "login": "AR-JUNA",
  "id": 218765731,
  "node_id": "U_kgD00oZow",
  "avatar_url": "https://avatars.githubusercontent.com/u/218765731?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/AR-JUNA",
  "html_url": "https://github.com/AR-JUNA",
  "followers_url": "https://api.github.com/users/AR-JUNA/followers",
```

Bottom Instance (Failing Request):

- Request URL:** `https://api.github.com/user`
- Headers:** (7 items) - Authorization, Content-Type, User-Agent, Host, Connection, Accept, Cache-Control.
- Script (Post-response):**

```
1 // Test 1: Check if response status is 200
2 pm.test("Status code is 200", function () {
3     pm.response.to.have.status(200);
4 });
5
6 // Test 2: Check if response has your login/username
7 pm.test("Response has login field", function () {
8     var jsonData = pm.response.json();
9     pm.expect(jsonData.login).to.be.a('string');
10 });
11
12 // Test 3: Check if response has your ID
13 pm.test("Response has ID", function () {
14     var jsonData = pm.response.json();
15     pm.expect(jsonData.id).to.be.a('number');
16 });
```
- Test Results:** 200 OK, 401 ms, 2.23 KB, Save Response
- Body (JSON Preview):**

```
PASSED Status code is 200
PASSED Response has login field
PASSED Response has ID
```

Screenshot 3: GET request showing user data

CREATE (POST)

Request: POST {{base_url}}/user/repos

Body:

json

```
{  
  "name": "test-repo-postman",  
  "description": "Repository created via Postman",  
  "private": false  
}
```

The screenshot shows the Postman interface with the following details:

- Collection:** Arjuna's Workspace
- Environment:** Github API Assignment
- Method:** POST
- URL:** https://api.github.com/user/repos
- Body (JSON):**

```
{  
  "name": "gyannnn-test-final-001",  
  "description": "Final test repository",  
  "private": false  
}
```
- Response Status:** 201 Created
- Response Body (JSON):**

```
{  
  "id": 1117501176,  
  "node_id": "U_kgDQ0Qp0u2-A",  
  "name": "gyannnn-test-final-001",  
  "full_name": "AR-JUNA/gyannnn-test-final-001",  
  "private": false,  
  "owner": {  
    "login": "AR-JUNA",  
    "id": 210765731,  
    "node_id": "U_kgDQ0Qp0zow",  
    "avatar_url": "https://avatars.githubusercontent.com/u/210765731?v=4",  
    "gravatar_id": "",  
    "url": "https://api.github.com/users/AR-JUNA",  
    "html_url": "https://github.com/AR-JUNA",  
    "followers_url": "https://api.github.com/users/AR-JUNA/followers",  
    "following_url": "https://api.github.com/users/AR-JUNA/following{/other_user}",  
    "gists_url": "https://api.github.com/users/AR-JUNA/gists{/gist_id}",  
    "starred_url": "https://api.github.com/users/AR-JUNA/starred{/owner}/{/repo}",  
    "subscriptions_url": "https://api.github.com/users/AR-JUNA/subscriptions",  
    "repos_url": "https://api.github.com/users/AR-JUNA/repos"  
  }  
}
```

The screenshot shows the Postman interface with the following details:

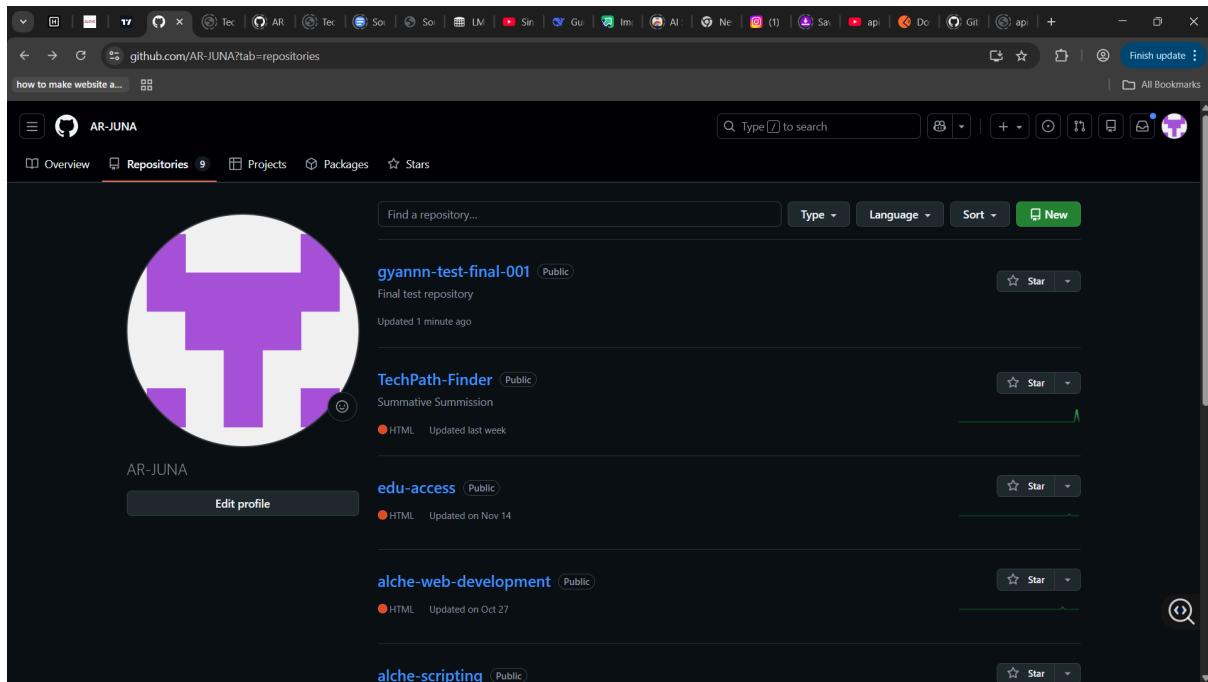
- Collection:** Arjuna's Workspace
- Environment:** Github API Assignment
- Method:** POST
- URL:** https://api.github.com/user/repos
- Pre-request Script:**

```
// Simple test - check status only
```
- Post-response Script:**

```
pm.test("Repository created", function () {  
  pm.response.to.have.status(201);  
  console.log("Repository created successfully!");  
});
```
- Response Status:** 201 Created
- Test Result:** PASSED

Response: 201 Created with repository details

Screenshot 4: POST request and successful response



Screenshot 5: After the POST request, the repository was created on GitHub

UPDATE (PATCH)

Request: PATCH {{base_url}}/repos/{username}/test-repo-postman

Body:

json

```
{  
  "description": "Updated description via PATCH request"  
}
```

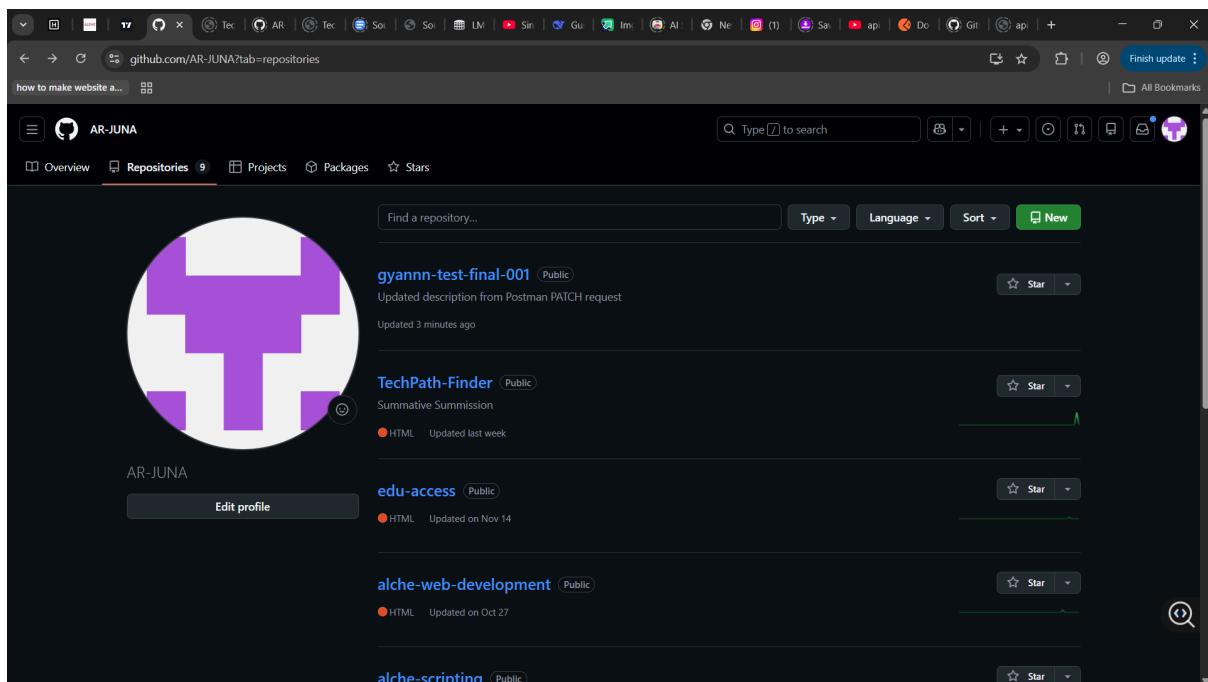
The screenshot shows two instances of the Postman application interface. Both instances display a collection named 'Arjuna's Workspace' containing a 'Github API Assignment' folder with a 'PATCH Update Repository' item. The URL for both requests is <https://api.github.com/repos/AR-JUNA/gyann-test-final-001>. The 'Body' tab is selected, showing the JSON payload:

```
{  
  "description": "Updated description from Postman PATCH request",  
  "private": false  
}
```

Both requests result in a **200 OK** response with a status message of **PASSED** and the message **Update successful**. The response body contains the updated repository data, including the new description and private status.

Response: 200 OK with updated repository data

Screenshot 6: PATCH request and updated response



Screenshot 7: After the PATCH request, the README for the repository got updated

DELETE (DELETE)

Request: DELETE {{base_url}}/repos/{username}/test-repo-postman

Response: 204 No Content (successful deletion)

The screenshot shows the Postman application interface. On the left, the sidebar displays 'Arjuna's Workspace' with sections for Collections, Environments, History, APIs, and Flows. Under 'Collections', there is a 'Github API Assignment' folder containing four items: 'Get My User Info' (GET), 'Create New Repository' (POST), 'Update Repository' (PATCH), and 'Delete Repository' (GET). The 'Delete Repository' item is selected.

The main workspace shows a 'Github API Assignment / Delete Repository' collection. A 'DELETE' request is selected, with the URL <https://api.github.com/repos/AR-JUNA/gyann-test-final-001> displayed in the header bar. The 'Send' button is highlighted in blue.

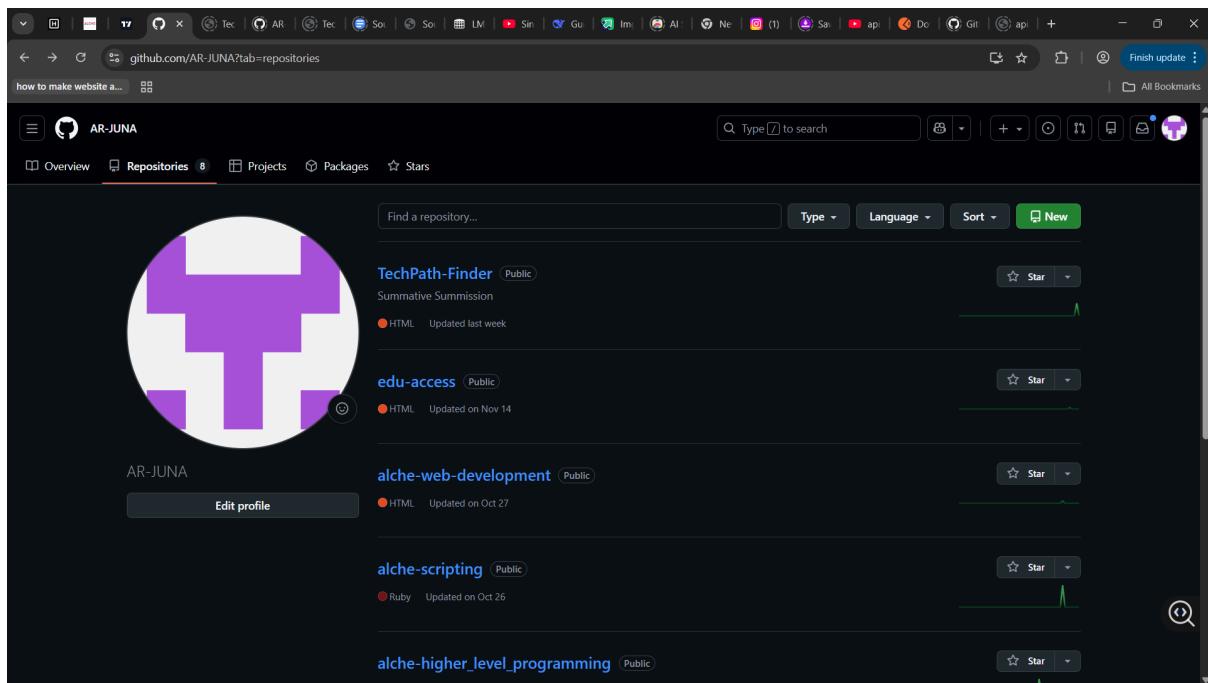
The 'Pre-request' script contains the following code:

```
pm.test("Delete successful", function () {
    pm.response.to.have.status(204);
});
```

The 'Test Results' section shows a single test case labeled '1' with the status 'PASSED' and the message 'Delete successful'.

At the bottom, the status bar indicates '204 No Content' with a duration of '691 ms' and a size of '1.41 KB'. Other options like 'Save Response' and 'Runner' are also visible.

Screenshot 8: DELETE request with 204 response



Screenshot 9: Verified: Repository removed from GitHub account

Postman Environment & Variables

I created a Postman environment called GitHub API Environment with:

Variables Used:

base_url: <https://api.github.com>

auth_token: [My GitHub personal access token]

repo_name: test-repo-postman

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Arjuna's Workspace' containing 'Collections', 'Environments', 'APIs', and 'Flows'. The 'Environments' section is expanded, showing a 'GitHub API Environment' entry. The main panel displays the 'GitHub API Environment' variables table:

Variable	Value
base_url	https://api.github.com
auth_token	ghp_2HO4a4C7RB03mVaJERi0uy5qz2YyB2x75XF
repo_name	gyannn-unique-test-001
Add variable	

Screenshot 10: Environment variables panel showing all variables

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Arjuna's Workspace' containing sections for Collections, Environments, History, APIs, and Flows. The main area is titled 'GitHub API Assignment - Run results' and shows a single run from today at 04:15:25 PM. It displays a summary table with 7 tests, 0 errors, and an average response time of 781 ms. Below the table, the 'Iteration 1' section lists four requests: 'Get My User Info' (status 200, PASS), 'Create New Repository' (status 201, PASS), 'Update Repository' (status 200, PASS), and 'Delete Repository' (status 204, PASS). Each request has its URL, status code, duration, and file size listed. At the bottom of the interface, there are various navigation and utility buttons.

Screenshot 11: After running all the requests, all the results passed.

Automation & Testing

Test Scripts

Added validation tests to each request:

javascript

```
// Example test for GET request
pm.test("Status code is 200", function() {
    pm.response.to.have.status(200);
});

pm.test("Response has user login", function() {
    var jsonData = pm.response.json();
    pm.expect(jsonData.login).to.be.a('string');
});
```

Challenges & Solutions

Challenge 1: 422 Validation Error

Problem: Repository name already existed

Solution: Used unique naming with a timestamp, deleted conflicting repos

Challenge 2: Test Placement

Problem: Tests in Pre-request Script instead of Tests tab

Solution: Moved validation scripts to the correct location

Challenge 3: Token Permissions

Problem: Insufficient scopes initially

Solution: Regenerated token with repo and user scopes