

Report& User Manual for: Data-Source Analyst Challenge

Dedicated to: Vladimir

Company: Hiretechfast®

Candidate: George Tadros

Email: George150485@gmail.com

Contents

1	Abstraction	3
2	Description	3
2.1	Objective	3
2.2	Task Overview	3
2.3	Description of the GitHub API	3
2.4	Downloading the Command Line Interface of Github	4
2.5	Authenticating the Github CLI	5
3	Step 1: Prepare & Test a List of Reports	8
3.1	Create an account on Postman web or download Postman on Desktop	8
3.2	Searching a specific repository	9
3.3	Getting Commits from a specific user repo	10
3.4	contents of a repo API	11
3.5	Result of the Research of Github APIs	12
3.5.1	1. Identify the Endpoints Necessary to Cover Client's Needs	12
3.5.2	2. Research GitHub API Documentation for Requests Logic, Pagination, Rate Limits, and Error Handling	13
4	Set Up a GitHub Repository	15
5	Creating a new Postman Collection	17
6	Create a token for Authentication	18
7	Using GoLab	19
8	GitHub API CRUD Operations in Python3 Colab	19



1 ABSTRACTION

This report addresses the evaluation of the experience and the skills for the position **Developer (Data-Source API Analyst)**, it is assumed that the task will evaluate, not only technical experience and skills, but also language and documentation skills, moreover, the mentality to address tasks and challenges,

2 DESCRIPTION

2.1 OBJECTIVE

The goal of this assignment is to assess your understanding of APIs, your ability to work with data extraction requirements, and your troubleshooting approach. Complete each section to demonstrate your knowledge

2.2 TASK OVERVIEW

- Research **GitHub API**
- Test it via Postman & Colab
- Prepare documentation and output results based on your work (check the **Step 5: Get Result** to be shown in section 5 of this report)

2.3 DESCRIPTION OF THE GITHUB API

The **GitHub REST API** is a set of HTTP-based endpoints that allow developers to interact programmatically with GitHub's features and data. Using this API, you can automate various tasks, access repository data, and manage your GitHub projects from scripts or applications.

Key Features:

- **Repository Management:** Create, delete, or update repositories and their contents.
- **Issues and Pull Requests:** Access and manage issues, pull requests, comments, and reviews.
- **User Data:** Retrieve or update information about users, organizations, and teams.
- **Authentication:** Use personal access tokens, OAuth tokens, or GitHub Apps for secure API access.
- **Webhooks:** Set up webhooks to trigger actions based on repository events (e.g., push, pull request).



Example Endpoints:

- **Get User Info:**

```
GET https://api.github.com/users/{username}
```

- **List Repository Issues:**

```
GET https://api.github.com/repos/{owner}/{repo}/issues
```

Usage Scenarios:

- Automating repetitive tasks (e.g., creating issues or pull requests).
- Integrating with CI/CD pipelines.
- Fetching data for analytics or custom dashboards.
- Managing project workflows through scripts.

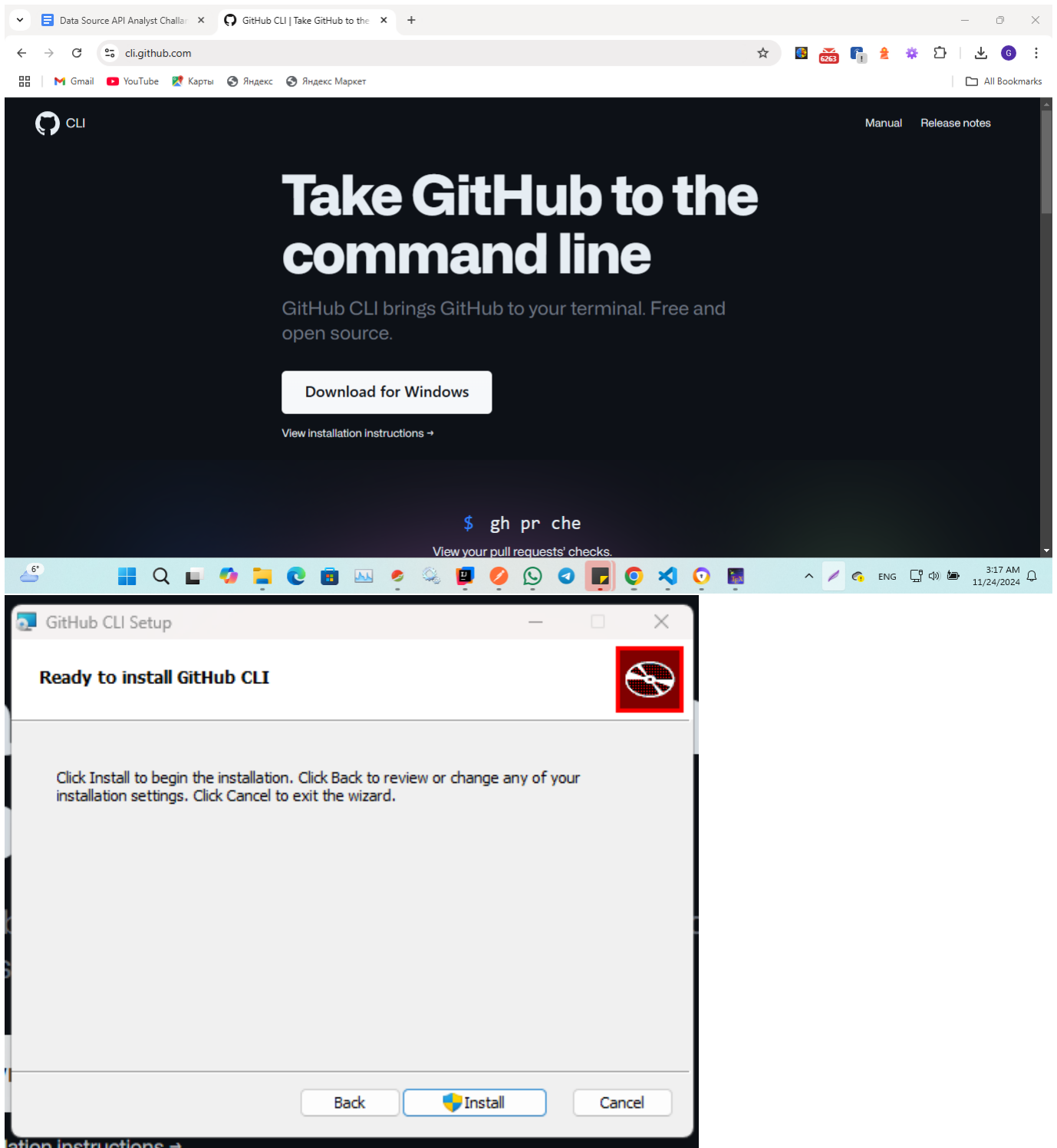
Tools for Testing:

- **Postman:** Manually test API requests and inspect responses.
- **Curl:** Command-line tool for sending HTTP requests.
- **Colab/Jupyter:** Integrate API calls into Python notebooks for data processing.

2.4 DOWNLOADING THE COMMAND LINE INTERFACE OF GITHUB

Using the following link the Github CLI can be installed on MAC, Linux and Windows(In my case) <https://cli.github.com/>





2.5 AUTHENTICATING THE GITHUB CLI

The github cli requires authentication, in-order to be used or accessed, this simply can be done by running the following command `gh auth login` on windows, note that it is recommended to run CMD as an Administrator as shown in the picture and follow the steps as shown



Administrator: Command Prompt - gh auth login

Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>gh auth login

? Where do you use GitHub? [GitHub.com](https://github.com)

? What is your preferred protocol for Git operations on this host? [HTTPS](https://github.com)

? Authenticate Git with your GitHub credentials? [Yes](#)

? How would you like to authenticate GitHub CLI? [Login with a web browser](#)

! First copy your one-time code: 5DF9-65D1

Press Enter to open <https://github.com/login/device> in your browser...

github.com/login?return_to=https%3A%2F%2Fgithub.com%2Flogin%2Fdevice

Booking.com AliExpress Addons Store Amazon.de eBay Facebook YouTube Gmail YouTube Карты Яндекс Яндекс Маркет All Bookmarks

Sign in to GitHub

Username or email address

Password [Forgot password?](#)

[Sign in](#)

[Sign in with a passkey](#)

[New to GitHub? Create an account](#)

[Terms](#) [Privacy](#) [Docs](#) [Contact GitHub Support](#) [Manage cookies](#) [Do not share my personal information](#)



github.com/login/device?skip_account_picker=true

Booking.com AliExpress Addons Store Amazon.de eBay Facebook YouTube Gmail YouTube Карты Яндекс Яндекс Маркет All Bookmarks

Device Activation

Signed in as Gtadros150485

Enter the code displayed on your device

Continue

GitHub staff will never ask you to enter your code on this page.

```
Administrator: Command Prompt - gh auth login
Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>gh auth login
? Where do you use GitHub? GitHub.com
? What is your preferred protocol for Git operations on GitHub.com? https
? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? Login
! First copy your one-time code: 5Dp...
Press Enter to open https://github.com/login/device in
```

© 2024 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information

github.com/login/device/confirmation

Booking.com AliExpress Addons Store Amazon.de eBay Facebook YouTube Gmail YouTube Карты Яндекс Яндекс Маркет All Bookmarks

GitHub CLI by GitHub wants to access your Gtadros150485 account

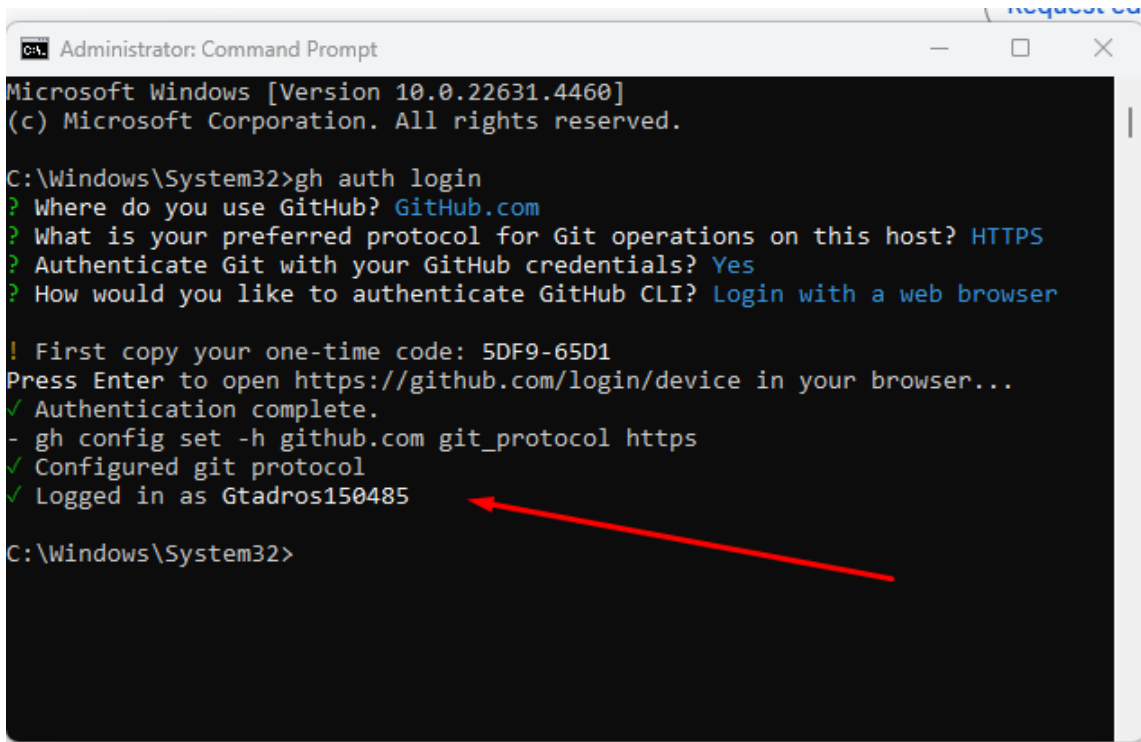
- Gists Read and write access
- Organizations and teams Read-only access
- Repositories Public and private
- Workflow Update GitHub Action Workflow files.

Cancel Authorize github

Requested from Frankfurt am Main 57.129.24.78 on November 24th, 2024 at 03:40 (+05)

Owned & operated by GitHub Created 6 years ago More than 1K GitHub users





```

Administrator: Command Prompt
Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>gh auth login
? Where do you use GitHub? GitHub.com
? What is your preferred protocol for Git operations on this host? HTTPS
? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? Login with a web browser

! First copy your one-time code: 5DF9-65D1
Press Enter to open https://github.com/login/device in your browser...
✓ Authentication complete.
- gh config set -h github.com git_protocol https
✓ Configured git protocol
✓ Logged in as Gtadros150485

C:\Windows\System32>

```

Once the

user is authorized, everything is settled and ready for usage.

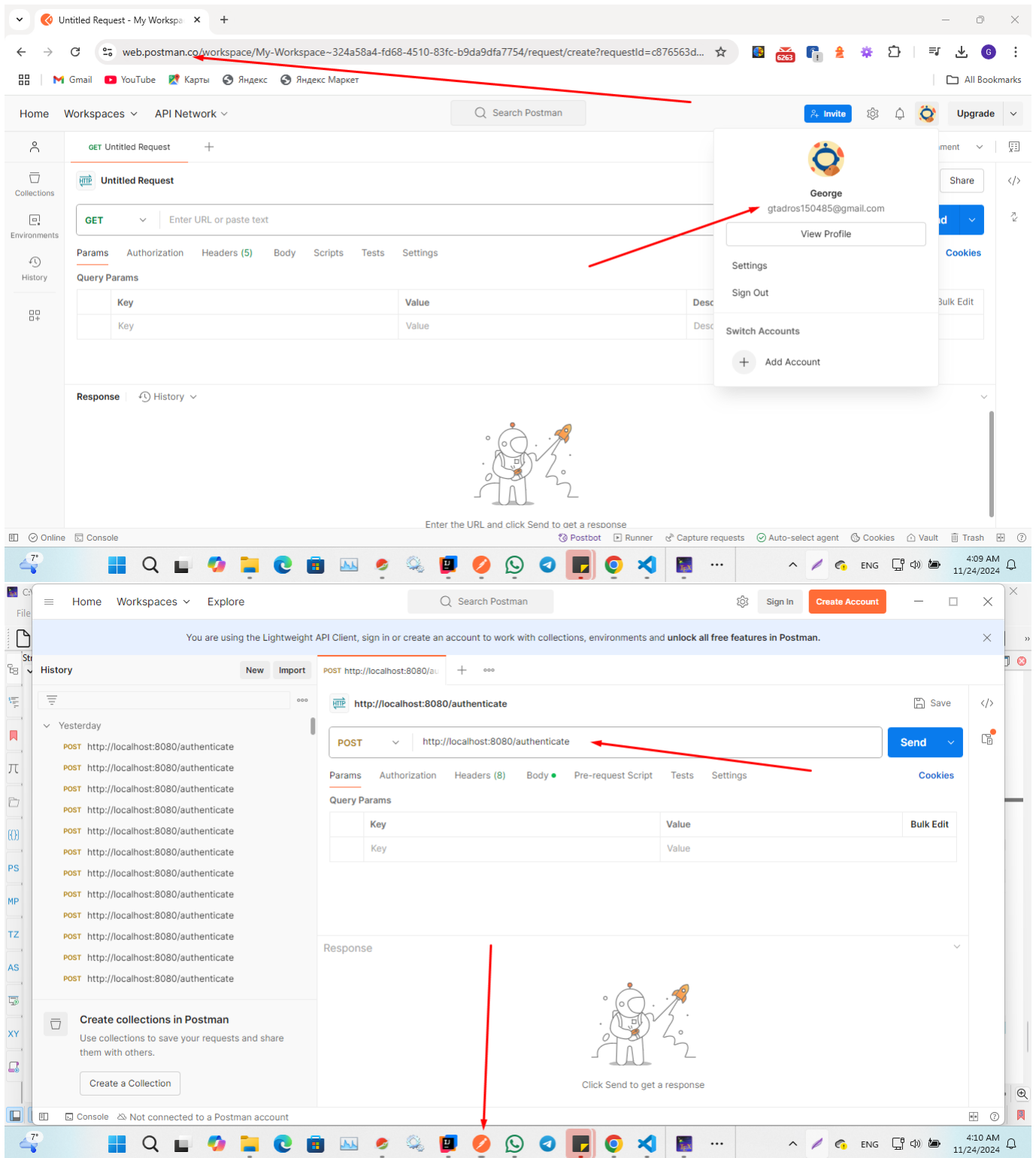
GitHub CLI automatically stores your Git credentials for you when you choose HTTPS as your preferred protocol for Git operations and answer "yes" to the prompt asking if you would like to authenticate to Git with your GitHub credentials. This can be useful as it allows you to use Git commands like git push and git pull without needing to set up a separate credential manager or use SSH.

3 STEP 1: PREPARE & TEST A LIST OF REPORTS

3.1 CREATE AN ACCOUNT ON POSTMAN WEB OR DOWNLOAD POSTMAN ON DESKTOP

Postman is an API handling tool that allow users to interacts with endpoints either to send requests or to receive requests, as shown in the following figure





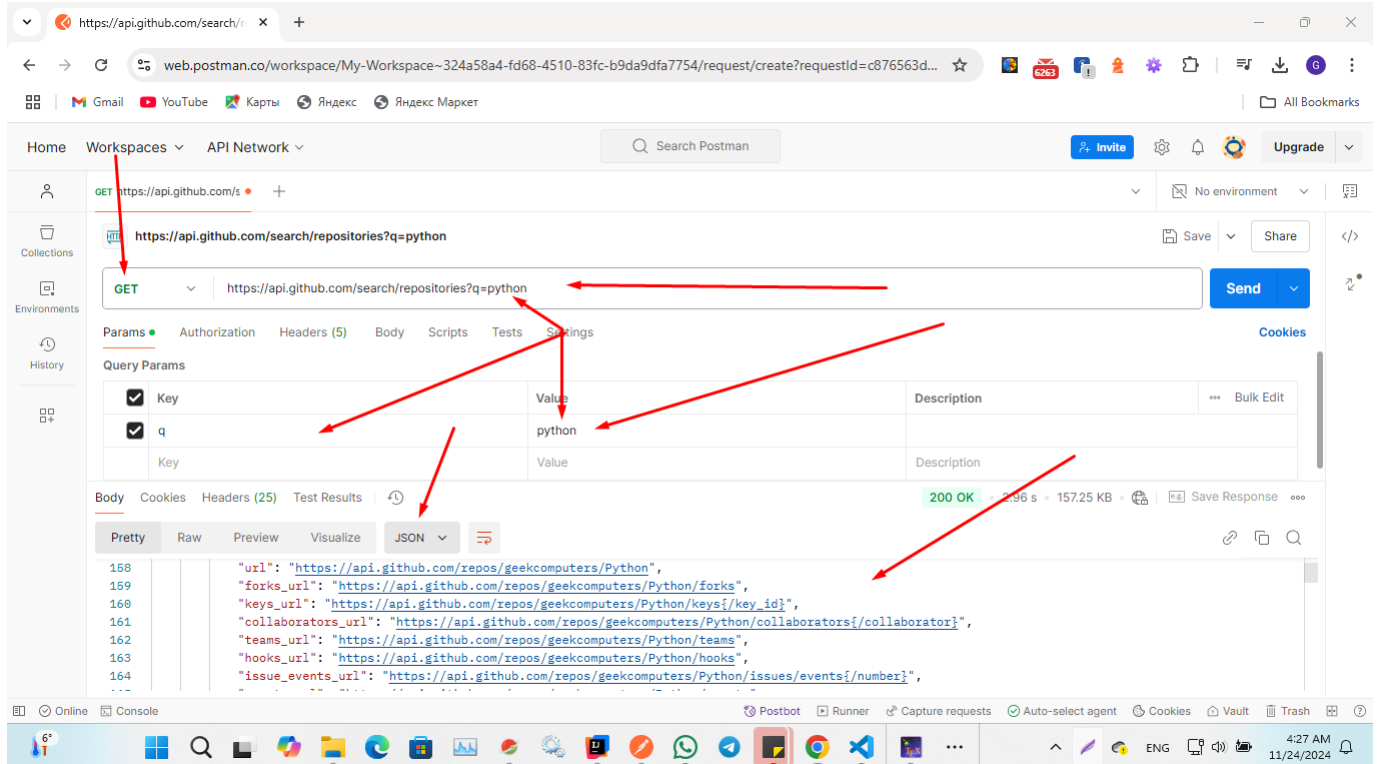
3.2 SEARCHING A SPECIFIC REPOSITORY

One of the Github APIs for searching a specific repository that is of-course **Public repository**, well, first a repository is a container of files for a specific shared project on the web, a public repository or repo for short is the accessibility for this container, so a public repo is set by user to be shared for public usage to everyone, however, a private repo is usually accessed only by specific set of users chosen as per the author of the repo, inshort, an API is an Application User



Interface(end-point) which acts as a middle point between system or as a communication way between different systems usually they send messages to each other using JSON file format. In-order to search a repo we can use the following API(end-point) GET /search/repositories in action we can use it like this GET <https://api.github.com/search/repositories?q=python>

here I am using the API to search for all repos containing the word python...



So as you can see in the picture, we put the API in the search bar, we define either if it a get or a post request, thirdly, we check the parameters and define them in the example it is (q=pyhton), then the result can be shown button-down of the page as a JSON file.

Please note that we are having a response of 200 in green which means that the request is a successful request with a success message we will learn about these codes later on in this report.

3.3 GETTING COMMITS FROM A SPECIFIC USER REPO

Using this API

GET /repos/owner/repo/commits we can get all commits coming from a specific owner/user in a specific repository for example

GET <https://api.github.com/repos/Gtadros150485/ems/commits>



GET https://api.github.com/repos/Gtadros150485/ems/commits

200 OK • 817 ms • 105.7 KB

```

split( , ,1=0;1<n.length-1;1++)11( object :=typeOf(i=i[n[1]]);return;i=i[n.length-1];return i;const y={accountID:void 0,trustKey:void 0,agentID:void 0,
licenseKey:void 0,applicationID:void 0,xpid:void 0},A={};function w(e){if(!e)throw new Error("All loader-config objects require an agent identifier!");if
(!A[e])throw new Error("LoaderConfig for ".concat(e, " was never set"));return A[e]}function E(e,t){if(!e)throw new Error("All loader-config objects
require an agent identifier!");A[e]=(0,i.D)(t,y);const i=(0,n.ek)(e);x&&(x.loader_config=A[e]);const _=(0,n.mF)().o;var x=x(385),R=i(6818);const S=
{buildEnv:R.Re,distMethod:R.gF,version:R.q4,originTime:x.sK},T={customTransaction:void 0,disabled:!!1,isolatedBacklog:!!1,loaderType:void 0,maxBytes:3e4,
oneError:void 0,origin:""+x._A.location,ptid:void 0,releaseIds:{},appMetadata:{},session:void 0,denyList:void 0,harvestCount:0,timeKeeper:void 0},N={};
function O(e){if(!e)throw new Error("All runtime objects require an agent identifier!");if(!N[e])throw new Error("Runtime for ".concat(e, " was never
set"));return N[e]}function I(e,t){if(!e)throw new Error("All runtime objects require an agent identifier!");N[e]=...0(i.D)(t,T)...S;const i=(0,n.ek)

```

3.4 CONTENTS OF A REPO API

you can access the contents of any repo using the following API GET /repos/owner/repo/contents/path
 you can also choose the specific response format you want for your endpoint result

GET https://api.github.com/repos/Gtadros150485/ems/contents/

200 OK • 689 ms • 105.7 KB

```

1 <!DOCTYPE html>
2 <html lang="en-US" nonce="jI/PV11zvXbHDvS5HYCW/V1qzevhkVnI6aeb55FhdyIyCUCK">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>Postman</title>
7

```



3.5 RESULT OF THE RESEARCH OF GITHUB APIS

To research the **GitHub API** and identify the necessary endpoints for your client's needs (search repositories, commits, and contents), we'll follow these steps. We'll also look into the **API documentation** to understand the request logic, pagination, rate limits, and error handling.

3.5.1 1. IDENTIFY THE ENDPOINTS NECESSARY TO COVER CLIENT'S NEEDS

Your client has requested data related to **public repositories**, **commits**, and **contents**. Let's break down the necessary GitHub API endpoints to fulfill these needs:

SEARCH REPOSITORIES (PUBLIC)

To search public repositories, the GitHub API offers the `search/repositories` endpoint. This allows you to search for repositories based on a query string.

Endpoint:

`GET /search/repositories`

Required Parameters:

- `q`: The search query (e.g., keywords like "machine learning").
- `page`: The page number for paginated results (optional).
- `per_page`: The number of results per page (default is 30, maximum is 100).

Example Request:

`GET https://api.github.com/search/repositories?q=machine-learning&per_page=10`

GET COMMITS

To retrieve commits from a specific repository, use the `repos/{owner}/{repo}/commits` endpoint. This is useful for tracking the changes made to a repository.

Endpoint:

`GET /repos/{owner}/{repo}/commits`

Required Parameters:

- `owner`: The owner of the repository.
- `repo`: The repository name.
- `sha` (optional): The branch or commit SHA to limit the scope of commits (e.g., "main").
- `page`: The page number (optional, for paginated results).



- `per_page`: Number of commits per page (optional).

Example Request:

```
GET https://api.github.com/repos/{owner}/{repo}/commits?per_page=5
```

GET REPOSITORY CONTENTS

To get the contents of a repository (files, directories, etc.), use the `repos/{owner}/{repo}/contents/{path}` endpoint. This retrieves the contents at the specified path in the repository.

Endpoint:

```
GET /repos/{owner}/{repo}/contents/{path}
```

Required Parameters:

- `owner`: The owner of the repository.
- `repo`: The repository name.
- `path`: The path within the repository (optional).

Example Request:

```
GET https://api.github.com/repos/{owner}/{repo}/contents/{path}
```

3.5.2 2. RESEARCH GITHUB API DOCUMENTATION FOR REQUESTS LOGIC, PAGINATION, RATE LIMITS, AND ERROR HANDLING

REQUEST LOGIC

The GitHub API follows **RESTful** principles. The basic HTTP methods used for requests are:

- **GET**: Retrieve data from the server.
- **POST**: Create new resources.
- **PATCH**: Update existing resources.
- **DELETE**: Remove resources.

GitHub API endpoints require parameters either in the query string or as part of the URL. Data returned from the GitHub API is typically in **JSON** format.



PAGINATION

GitHub API supports **pagination** to handle large datasets efficiently. When querying resources that might return a large number of results (like repositories, commits, or contents), you need to use pagination parameters:

- **page:** The page number (defaults to 1).
- **per_page:** The number of results per page (defaults to 30, maximum is 100).

GitHub API will include pagination information in the response headers, such as:

- **X-Total-Count:** The total number of results.
- **Link:** Contains URLs to navigate to the next or previous pages of results.

Example of pagination:

```
GET https://api.github.com/search/repositories?q=machine-learning&page=2&per_page=100
```

To handle paginated results, your system can check the **Link** header in the response to determine if there are additional pages.

RATE LIMITS

GitHub enforces **rate limits** to prevent abuse of their API. The rate limit varies depending on whether the request is authenticated or not:

- **Unauthenticated requests:** 60 requests per hour.
- **Authenticated requests:** 5,000 requests per hour (with a Personal Access Token or OAuth).

To check your current rate limit status, you can query the rate limit endpoint:

```
GET https://api.github.com/rate_limit
```

This will return information like:

```
{ "resources": { "core": { "limit": 5000, "remaining": 4999, "reset": 1611289923 } } }
```

X-RateLimit-Remaining header can be used to monitor how many requests are left.



ERROR HANDLING

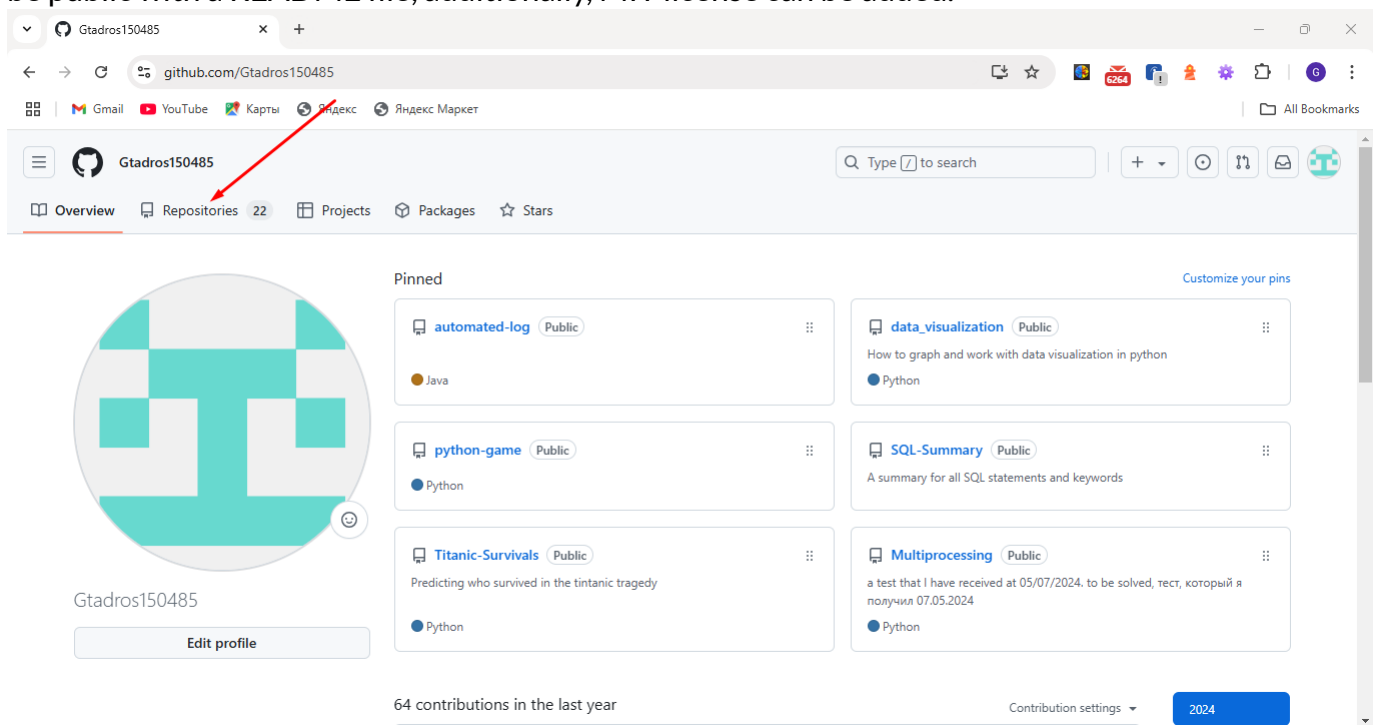
GitHub API provides meaningful error responses in case something goes wrong. Common error codes include:

- 401 Unauthorized: Authentication required or failed (missing/incorrect API token).
- 403 Forbidden: Rate limit exceeded or access is denied.
- 404 Not Found: The requested resource could not be found (e.g., repository or commit not found).
- 422 Unprocessable Entity: The request is well-formed but contains invalid data.
- 429 Too Many Requests: The rate limit for the API has been exceeded.

Each error response includes a JSON body describing the issue, often with a message and documentation link.

4 SET UP A GITHUB REPOSITORY

The easiest way to create a repository is to go the Github website, login or create an account if the user doesn't have an account yet then follow the following steps. note that the repo has to be public with a README file, additionally, MIT license can be added.



The image shows two screenshots of a web browser. The top screenshot displays the GitHub profile of user Gtadros150485, who has 22 repositories. A red arrow points to the 'New' button in the repository list. The bottom screenshot shows the 'Create a new repository' page, where the repository name is 'Data-Source-API-Analyst-' and the description is 'A repo for testing APIs specifically Github APIs'. The 'Public' option is selected for the repository's visibility.

Top Screenshot: GitHub Profile

Browser tabs: Your Repositories | +

Address bar: github.com/Gtadros150485?tab=repositories

Search: Type / to search

Navigation: Overview | **Repositories 22** | Projects | Packages | Stars

Profile: Gtadros150485 (Avatar: Teal and white geometric pattern) | Edit profile

Repositories List:

- ems** (Public) | Employment management system | Java | Updated 11 hours ago | Star
- task-management** (Private) | a java spring boot project for task management for developers maybe | MIT License | Updated last week | Star
- spring-boot-in-practice** (Private) | Summary for Book Somnath Musib - Spring Boot in Practice-Manning Publications (2022) | Java | Updated last week | Star

Bottom Screenshot: Create a new repository

Browser tabs: New repository | +

Address bar: github.com/new

Search: Type / to search

Navigation: New repository

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner * | Repository name *

Gtadros150485 / Data-Source-API-Analyst-

✔ Data-Source-API-Analyst-Test is available.

Great repository names are short and memorable. Need inspiration? How about **potential-fortnight** ?

Description (optional)

A repo for testing APIs specifically Github APIs

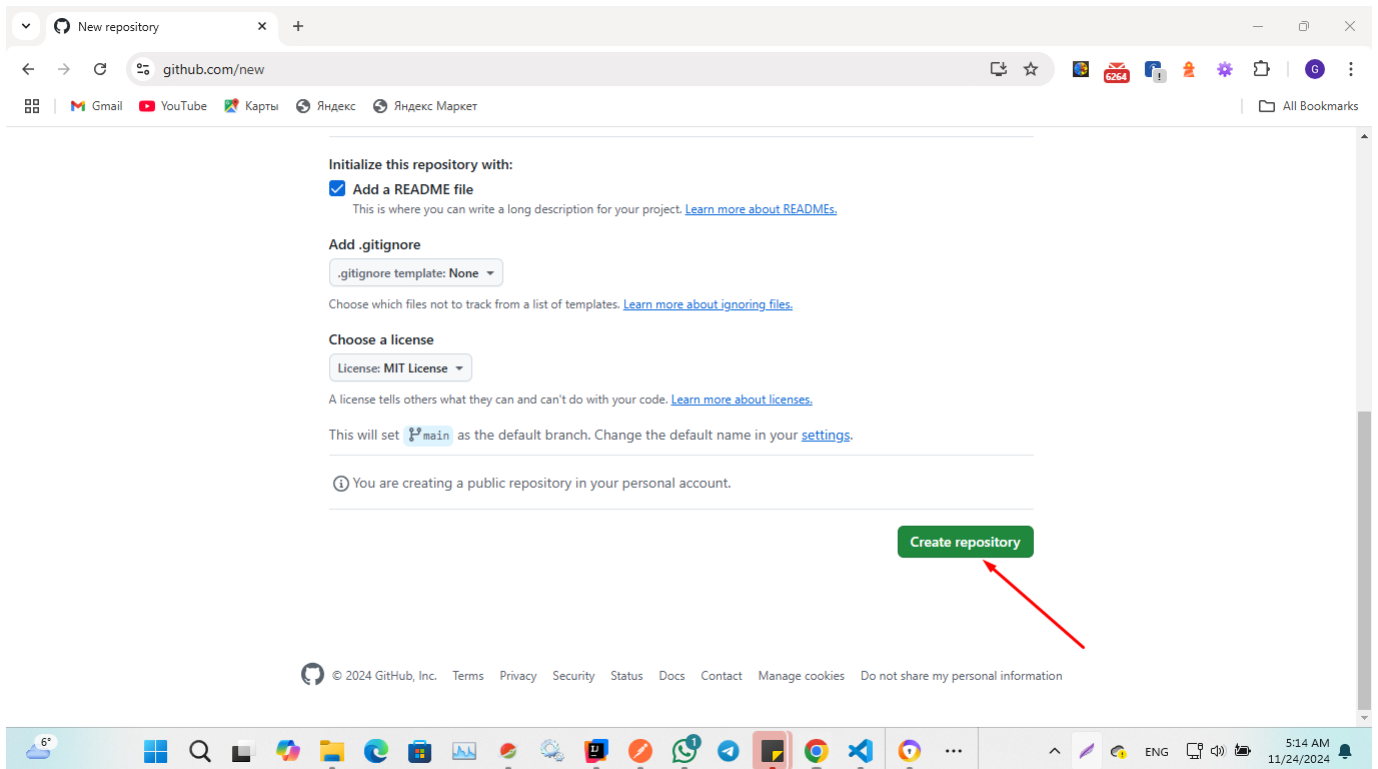
☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

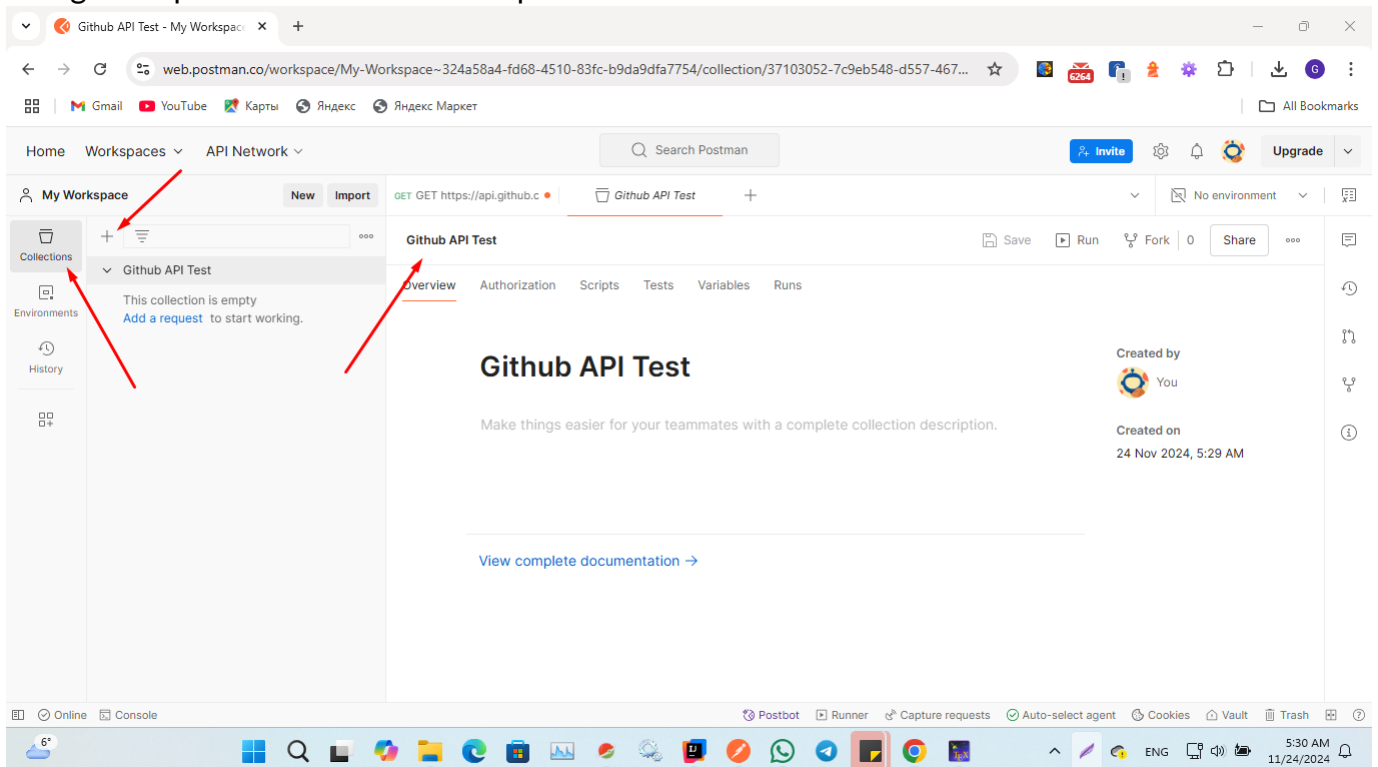
☒ Add a README file





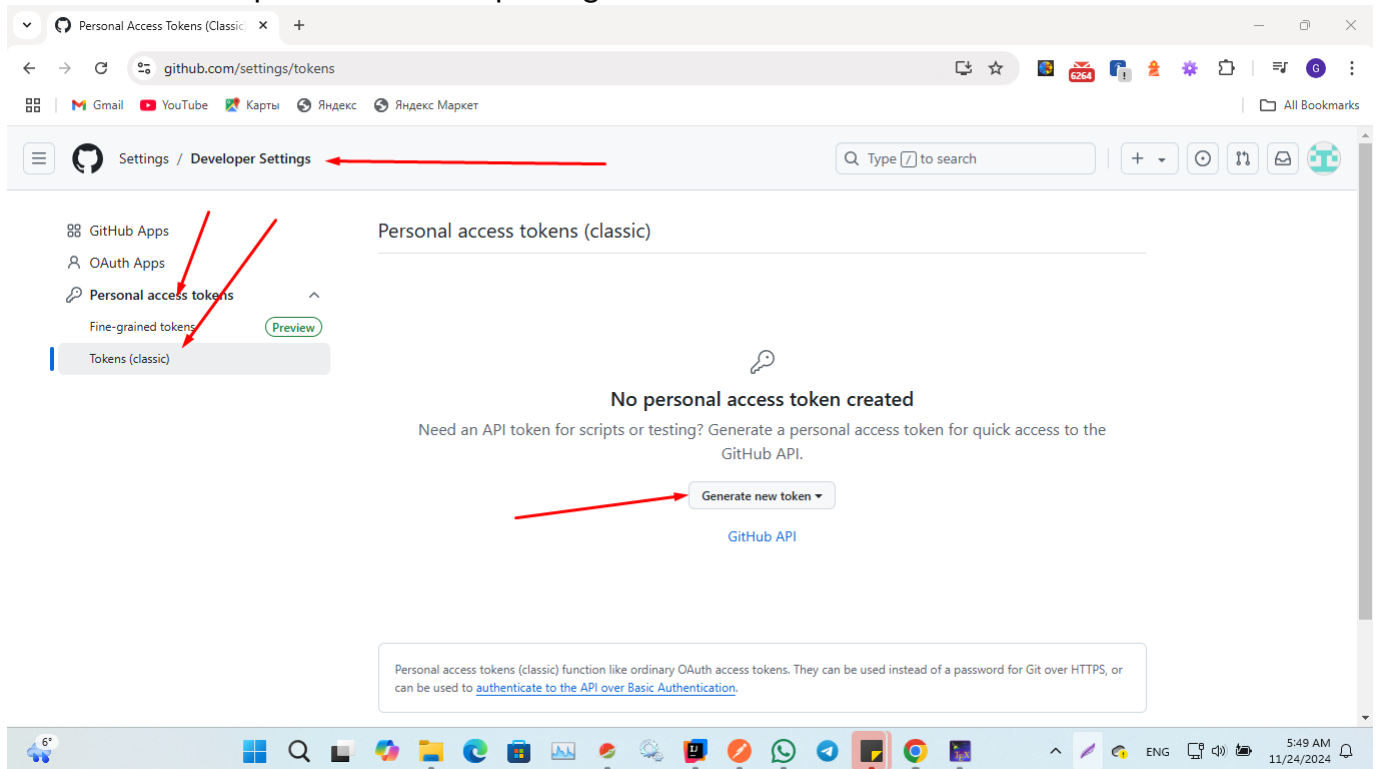
5 CREATING A NEW POSTMAN COLLECTION

Navigate to postman then follow the picture for illustration:

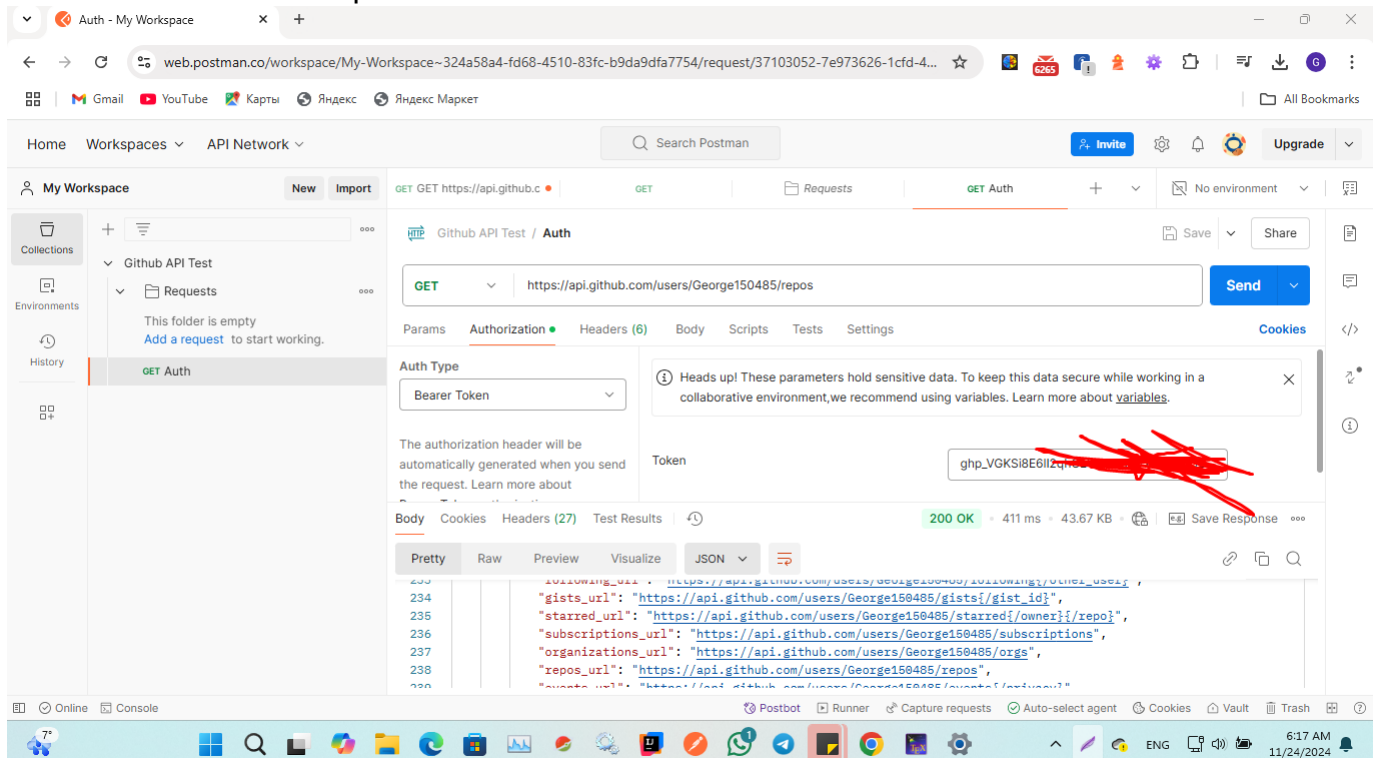


6 CREATE A TOKEN FOR AUTHENTICATION

Navigate to your Github account find the token Under the **Developer Settings** then generate token with the required access and privileges.



chose Bearer token then put the token as shown then click on send



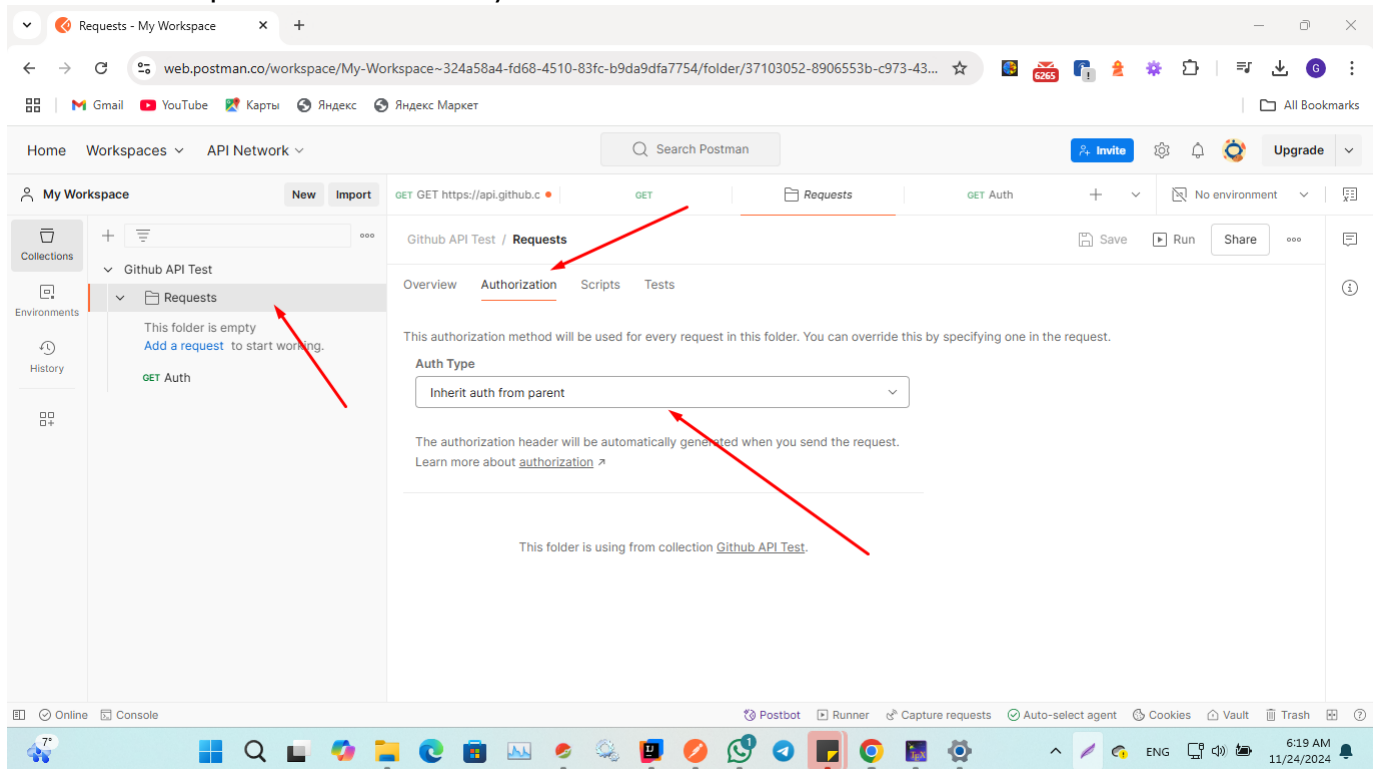
using the shell `curl -request GET`
`-url "https://api.github.com/users/username/repos"`
`-header "Accept: application/vnd.github+json"`



-header "Authorization: Bearer YOUR-TOKEN"

or you can use https://api.github.com/users/username/repos right away on Postman

Note for all requests make sure they inherit the main token as shown



7 GITHUB API CRUD OPERATIONS IN PYTHON3 COLAB

The following Python code demonstrates how to perform CRUD operations (Create, Read, Update, and Delete) on GitHub repositories using the GitHub REST API.

```
import requests

# Replace with your GitHub token
GITHUB_TOKEN = 'YOUR_PERSONAL_ACCESS_TOKEN'
GITHUB_USER = 'YOUR_GITHUB_USERNAME'

# GitHub API base URL
base_url = "https://api.github.com"

# Headers to include the authorization token
headers = {
    'Authorization': f'token {GITHUB_TOKEN}',
    'Accept': 'application/vnd.github.v3+json'
}

# Create a repository (POST)
def create_repo(repo_name, description):
```



```

url = f'{{base_url}}/user/repos'
data = {
    "name": repo_name,
    "description": description,
    "private": False # Set to True if you want a private repo
}

response = requests.post(url, json=data, headers=headers)

if response.status_code == 201:
    print(f"Repository '{{repo_name}}' created successfully.")
    return response.json() # Return repository data
else:
    print(f"Failed to create repository: {response.status_code}")
    print(response.json())
    return None

# Get repository details (GET)
def get_repo_details(repo_name):
    url = f'{{base_url}}/repos/{{GITHUB_USER}}/{{repo_name}}'
    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        print(f"Repository details:\n{response.json()}")
    else:
        print(f"Failed to retrieve repository details: {response.status_code}")
        print(response.json())

# Update repository (PATCH)
def update_repo(repo_name, new_description):
    url = f'{{base_url}}/repos/{{GITHUB_USER}}/{{repo_name}}'
    data = {
        "description": new_description
    }

    response = requests.patch(url, json=data, headers=headers)

    if response.status_code == 200:
        print(f"Repository '{{repo_name}}' updated successfully.")
        return response.json()
    else:
        print(f"Failed to update repository: {response.status_code}")
        print(response.json())

# Delete repository (DELETE)
def delete_repo(repo_name):
    url = f'{{base_url}}/repos/{{GITHUB_USER}}/{{repo_name}}'

```



```
response = requests.delete(url, headers=headers)

if response.status_code == 204:
    print(f"Repository '{repo_name}' deleted successfully.")
else:
    print(f"Failed to delete repository: {response.status_code}")
    print(response.json())

# Now let's run the CRUD operations on the GitHub API:
# 1. Create a repository
repo_name = "test-repo-api"
description = "This is a test repository created via GitHub API."
repo = create_repo(repo_name, description)

# Check if the repository was created successfully
if repo:
    # 2. Get repository details
    get_repo_details(repo_name)

# 3. Update the repository description
new_description = "Updated description for the test repository."
update_repo(repo_name, new_description)

# 4. Delete the repository
delete_repo(repo_name)
```

