Postman API Documentation Discovery & Collection.

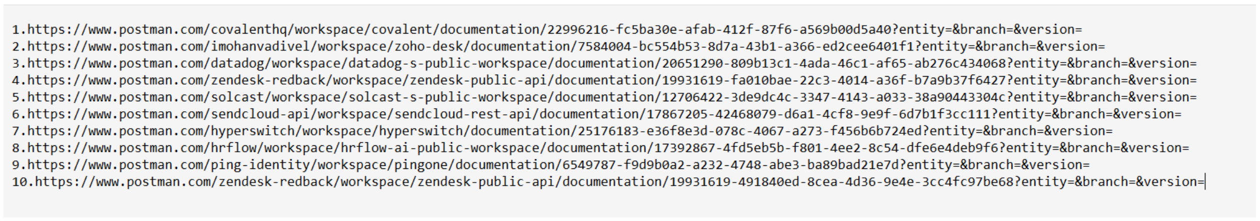
1. Introduction

This documentation provides an in-depth overview of the approach and methods utilized for discovering and extracting API documentation from the Postman API platform. It encompasses the techniques, scripts, challenges faced, and potential solutions.

2. Objectives

- Discover API documentation .
- Extract API endpoints and relevant metadata.
- Organize the extracted data efficiently for future analysis.

3. API Documentation URL Structure



```
1.https://www.postman.com/covalenthq/workspace/covalent/documentation/22996216-fc5ba30e-afab-412f-87f6-a569b00d5a40?entity=&branch=&version=
2.https://www.postman.com/imohanvadivel/workspace/zoho-desk/documentation/7584004-bc554b53-8d7a-43b1-a366-ed2cee6401f1?entity=&branch=&version=
3.https://www.postman.com/datadog/workspace/datadog-s-public-workspace/documentation/20651290-809b13c1-4ada-46c1-af65-ab276c434068?entity=&branch=&version=
4.https://www.postman.com/zendesk-redback/workspace/zendesk-public-api/documentation/19931619-fa010bae-22c3-4014-a36f-b7a9b37f6427?entity=&branch=&version=
5.https://www.postman.com/solcast/workspace/solcast-s-public-workspace/documentation/12706422-3de9dc4c-3347-4143-a033-38a90443304c?entity=&branch=&version=
6.https://www.postman.com/sendcloud-api/workspace/sendcloud-rest-api/documentation/17867205-42468079-d6a1-4cf8-9e9f-6d7b1f3cc111?entity=&branch=&version=
7.https://www.postman.com/hyperswitch/workspace/hyperswitch/documentation/25176183-e36f8e3d-078c-4067-a273-f456b6b724ed?entity=&branch=&version=
8.https://www.postman.com/hrflow/workspace/hrflow-ai-public-workspace/documentation/17392867-4fd5eb5b-f801-4ee2-8c54-dfe6e4deb9f6?entity=&branch=&version=
9.https://www.postman.com/ping-identity/workspace/pingone/documentation/6549787-f9d9b0a2-a232-4748-abe3-ba89bad21e7d?entity=&branch=&version=
10.https://www.postman.com/zendesk-redback/workspace/zendesk-public-api/documentation/19931619-491840ed-8cea-4d36-9e4e-3cc4fc97be68?entity=&branch=&version=
```

All API documentation URLs in Postman follow this structure:

https://www.postman.com/{workspace_name}/workspace/{team_name}/documentation/{unique_id}?entity=&branch=&version=

Where:
- `{workspace_name}`: Represents the workspace name.
- `{team_name}`: Describes the team.
- `{unique_id}`: A unique identifier for the documentation.

4. Dynamic Content Loading

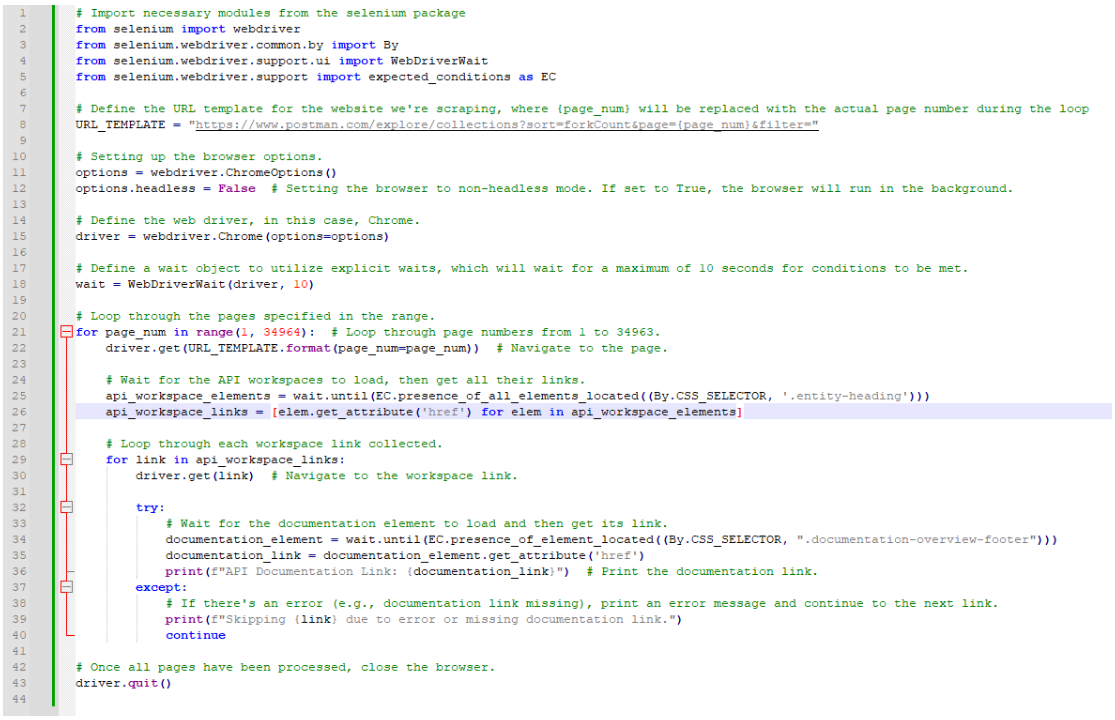
The Postman platform uses JavaScript to fetch and display content after the initial page load. Traditional scraping methods like using the `requests` library might not fetch this dynamically loaded content.

5. Browser Automation with Selenium

To bypass the dynamic loading challenge, Selenium, a browser automation tool, was employed. This approach, while slower, allows for the execution of JavaScript and fetches dynamically loaded content.

Browser Automation with Selenium: This method uses a real web browser to navigate the site, execute JavaScript, and fetch dynamically loaded content. It's often the most direct way to scrape content from dynamic sites. So we can Extract the API Documentation Url from this .

Python code

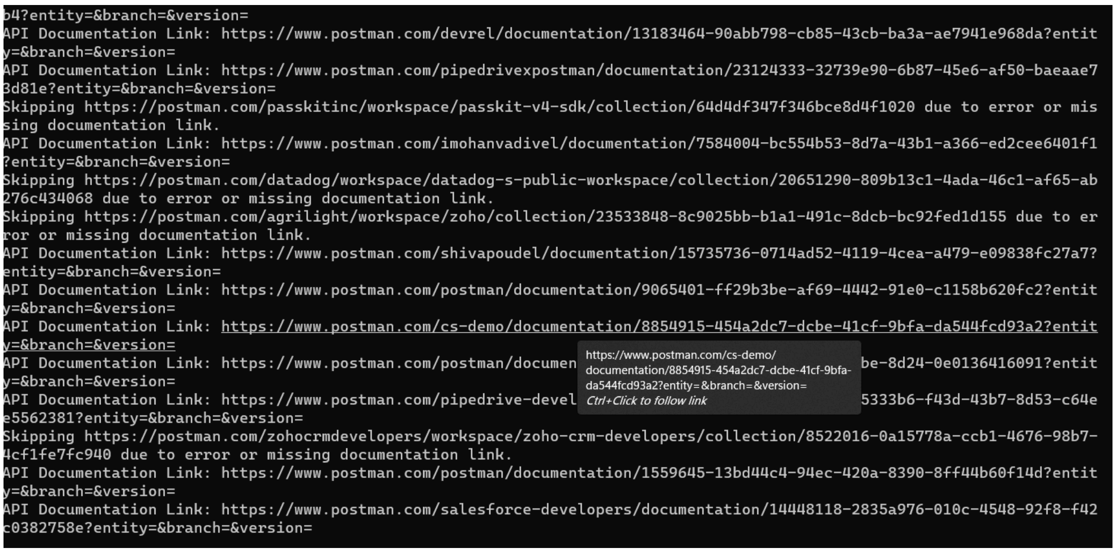


```python
# Import necessary modules from the selenium package
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

# Define the URL template for the website we're scraping, where {page_num} will be replaced with the actual page number during the loop
URL_TEMPLATE = "https://www.postman.com/explore/collections?sort=forkCount&page={page_num}&filter="

# Setting up the browser options.
options = webdriver.ChromeOptions()
options.headless = False  # Setting the browser to non-headless mode. If set to True, the browser will run in the background.

# Define the web driver, in this case, Chrome.
driver = webdriver.Chrome(options=options)

# Define a wait object to utilize explicit waits, which will wait for a maximum of 10 seconds for conditions to be met.
wait = WebDriverWait(driver, 10)

# Loop through the pages specified in the range.
for page_num in range(1, 34964):  # Loop through page numbers from 1 to 34963.
    driver.get(URL_TEMPLATE.format(page_num=page_num))  # Navigate to the page.

    # Wait for the API workspaces to load, then get all their links.
    api_workspace_elements = wait.until(EC.presence_of_all_elements_located((By.CSS_SELECTOR, '.entity-heading')))
    api_workspace_links = [elem.get_attribute('href') for elem in api_workspace_elements]

    # Loop through each workspace link collected.
    for link in api_workspace_links:
        driver.get(link)  # Navigate to the workspace link.

        try:
            # Wait for the documentation element to load and then get its link.
            documentation_element = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, ".documentation-overview-footer")))
            documentation_link = documentation_element.get_attribute('href')
            print(f"API Documentation Link: {documentation_link}")  # Print the documentation link.
        except:
            # If there's an error (e.g., documentation link missing), print an error message and continue to the next link.
            print(f"Skipping {link} due to error or missing documentation link.")
            continue

# Once all pages have been processed, close the browser.
driver.quit()
```

This code automates and checks for all the api DOCS in the Postman API website .

Drawbacks :
* It takes a lot of time to extract all the urls , it has a total of 34000 pages so it takes a lot of time .
* Skips Some API Documentations which has Proper Documentations .

OUTPUT:



```
b4?entity=&branch=&version=
API Documentation Link: https://www.postman.com/devrel/documentation/13183464-90abb798-cb85-43cb-ba3a-ae7941e968da?entit
y=&branch=&version=
API Documentation Link: https://www.postman.com/pipedrivexpostman/documentation/23124333-32739e90-6b87-45e6-af50-baeaae7
3d81e?entity=&branch=&version=
Skipping https://postman.com/passkitinc/workspace/passkit-v4-sdk/collection/64d4df347f346bce8d4f1020 due to error or mis
sing documentation link.
API Documentation Link: https://www.postman.com/imohanvadivel/documentation/7584004-bc554b53-8d7a-43b1-a366-ed2cee6401f1
?entity=&branch=&version=
Skipping https://postman.com/datadog/workspace/datadog-s-public-workspace/collection/20651290-809b13c1-4ada-46c1-af65-ab
276c434068 due to error or missing documentation link.
Skipping https://postman.com/agrilight/workspace/zoho/collection/23533848-8c9025bb-b1a1-491c-8dcb-bc92fed1d155 due to er
ror or missing documentation link.
API Documentation Link: https://www.postman.com/shivapoudel/documentation/15735736-0714ad52-4119-4cea-a479-e09838fc27a7?
entity=&branch=&version=
API Documentation Link: https://www.postman.com/postman/documentation/9065401-ff29b3be-af69-4442-91e0-c1158b620fc2?entit
y=&branch=&version=
API Documentation Link: https://www.postman.com/cs-demo/documentation/8854915-454a2dc7-dcbe-41cf-9bfa-da544fcd93a2?entit
y=&branch=&version=
API Documentation Link: https://www.postman.com/postman/documen                                  be-8d24-0e0136416091?entit
y=&branch=&version=
API Documentation Link: https://www.postman.com/pipedrive-devel                                  5333b6-f43d-43b7-8d53-c64e
e5562381?entity=&branch=&version=
Skipping https://postman.com/zohocrmdevelopers/workspace/zoho-crm-developers/collection/8522016-0a15778a-ccb1-4676-98b7-
4cf1fe7fc940 due to error or missing documentation link.
API Documentation Link: https://www.postman.com/postman/documentation/1559645-13bd44c4-94ec-420a-8390-8ff44b60f14d?entit
y=&branch=&version=
API Documentation Link: https://www.postman.com/salesforce-developers/documentation/14448118-2835a976-010c-4548-92f8-f42
c0382758e?entity=&branch=&version=
```

https://www.postman.com/cs-demo/documentation/8854915-454a2dc7-dcbe-41cf-9bfa-da544fcd93a2?entity=&branch=&version=
Ctrl+Click to follow link

6. Data Extraction and Organization

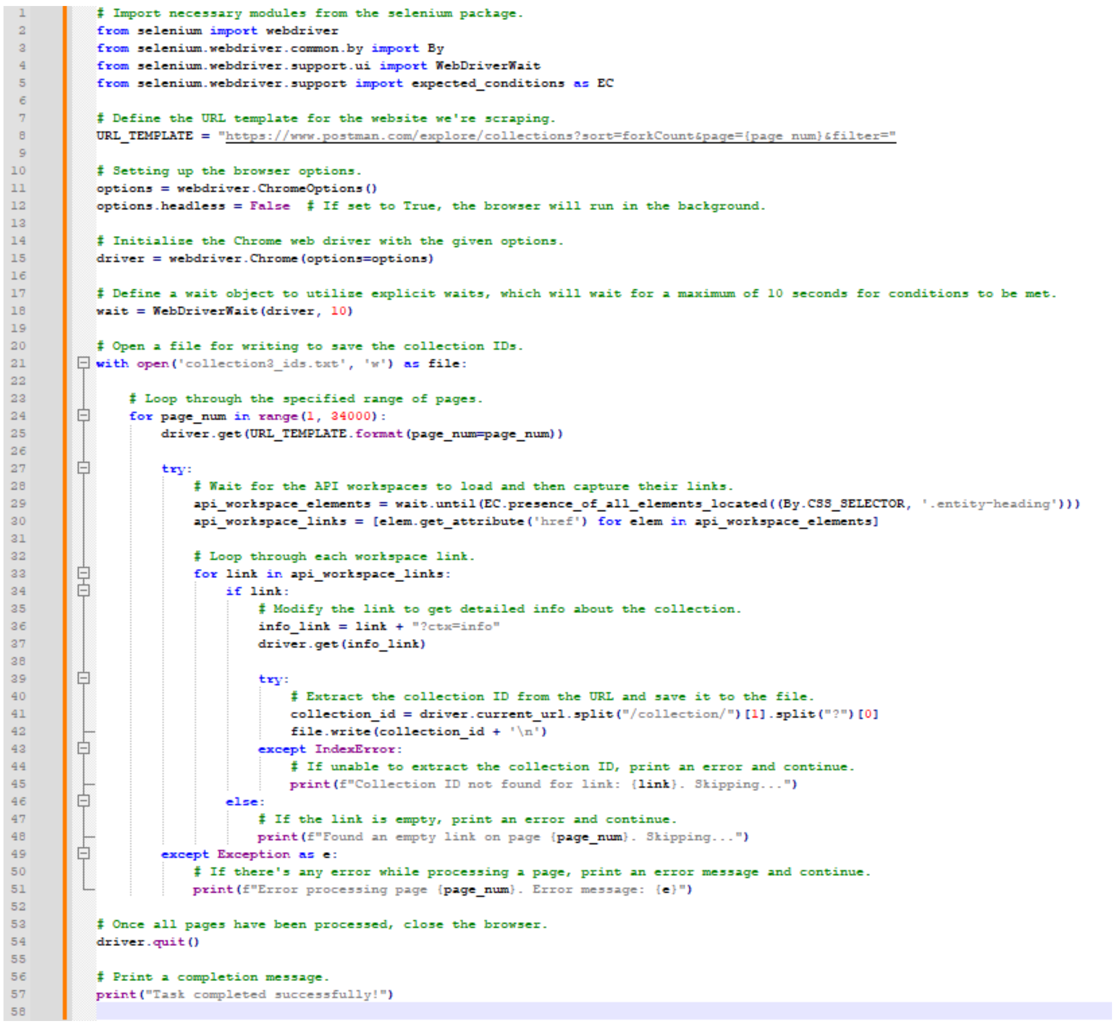
Post the discovery phase; data needs to be extracted, processed, and organized.

6.1. Extracting Collection IDs

The collection IDs were extracted using a Selenium script, with each ID representing a unique API collection in Postman.

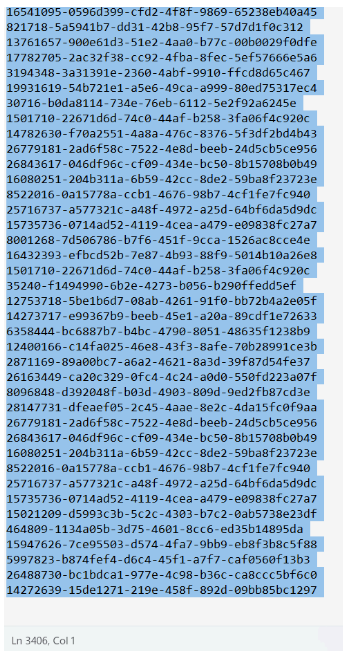
Collected Collection ID of APIS from POSTMAN API Platform using Selenium

Script :



```python
# Import necessary modules from the selenium package.
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

# Define the URL template for the website we're scraping.
URL_TEMPLATE = "https://www.postman.com/explore/collections?sort=forkCount&page={page_num}&filter="

# Setting up the browser options.
options = webdriver.ChromeOptions()
options.headless = False  # If set to True, the browser will run in the background.

# Initialize the Chrome web driver with the given options.
driver = webdriver.Chrome(options=options)

# Define a wait object to utilize explicit waits, which will wait for a maximum of 10 seconds for conditions to be met.
wait = WebDriverWait(driver, 10)

# Open a file for writing to save the collection IDs.
with open('collection3_ids.txt', 'w') as file:

    # Loop through the specified range of pages.
    for page_num in range(1, 34000):
        driver.get(URL_TEMPLATE.format(page_num=page_num))

        try:
            # Wait for the API workspaces to load and then capture their links.
            api_workspace_elements = wait.until(EC.presence_of_all_elements_located((By.CSS_SELECTOR, '.entity-heading')))
            api_workspace_links = [elem.get_attribute('href') for elem in api_workspace_elements]

            # Loop through each workspace link.
            for link in api_workspace_links:
                if link:
                    # Modify the link to get detailed info about the collection.
                    info_link = link + "?ctx=info"
                    driver.get(info_link)

                    try:
                        # Extract the collection ID from the URL and save it to the file.
                        collection_id = driver.current_url.split("/collection/")[1].split("?")[0]
                        file.write(collection_id + '\n')
                    except IndexError:
                        # If unable to extract the collection ID, print an error and continue.
                        print(f"Collection ID not found for link: {link}. Skipping...")
                else:
                    # If the link is empty, print an error and continue.
                    print(f"Found an empty link on page {page_num}. Skipping...")
        except Exception as e:
            # If there's any error while processing a page, print an error message and continue.
            print(f"Error processing page {page_num}. Error message: {e}")

# Once all pages have been processed, close the browser.
driver.quit()

# Print a completion message.
print("Task completed successfully!")
```

OUTPUT: #Change Text File Location Accordingly

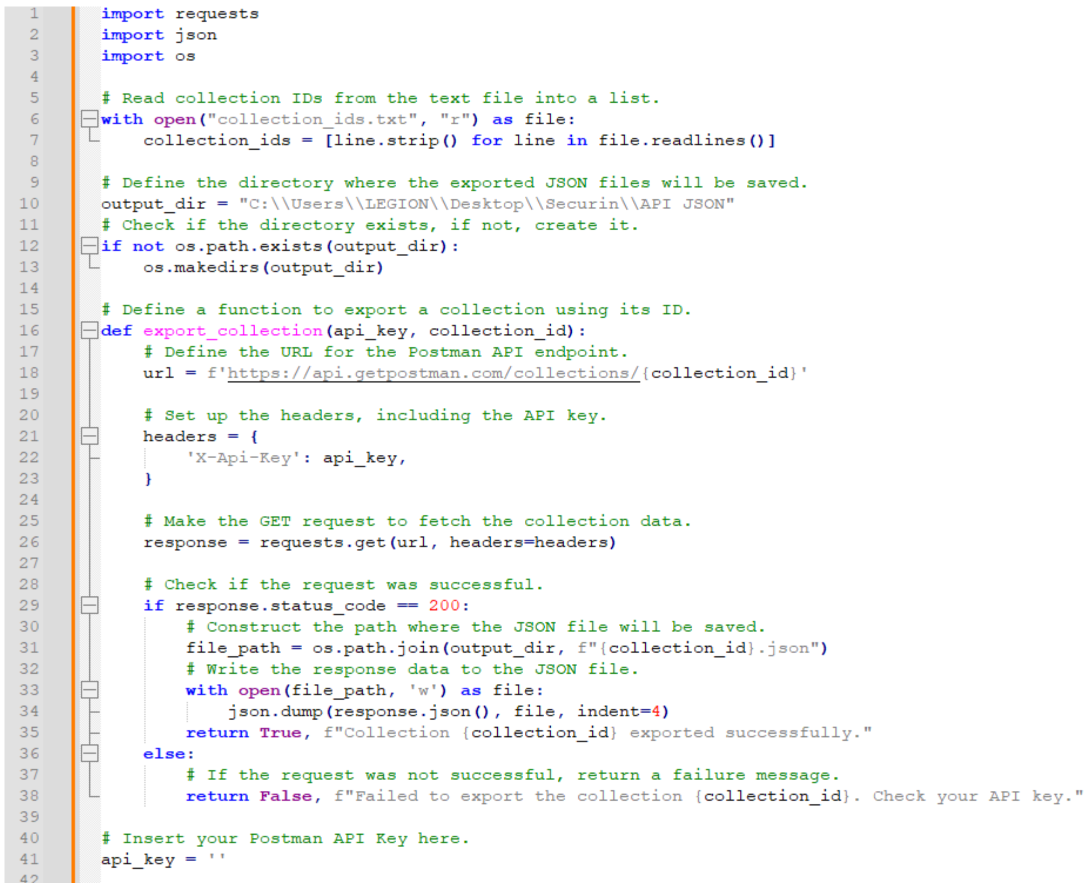


```
16541095-0596d399-cfd2-4f8f-9869-65238eb40a45
821718-5a5941b7-dd31-42b8-95f7-57d7d1f0c312
13761657-900e61d3-51e2-4aa0-b77c-00b0029f0dfe
17782705-2ac32f38-cc92-4fba-8fec-5ef57666e5a6
3194348-3a31391e-2360-4abf-9910-ffcd8d65c467
19931619-54b721e1-a5e6-49ca-a999-80ed75317ec4
30716-b0da8114-734e-76eb-6112-5e2f92a6245e
1501710-22671d6d-74c0-44af-b258-3fa06f4c920c
14782630-f70a2551-4a8a-476c-8376-5f3df2bd4b43
26779181-2ad6f58c-7522-4e8d-beeb-24d5cb5ce956
26843617-046df96c-cf09-434e-bc50-8b15708b0b49
16080251-204b311a-6b59-42cc-8de2-59ba8f23723e
8522016-0a15778a-ccb1-4676-98b7-4cf1fe7fc940
25716737-a577321c-a48f-4972-a25d-64bf6da5d9dc
15735736-0714ad52-4119-4cea-a479-e09838fc27a7
8001268-7d506786-b7f6-451f-9cca-1526ac8cce4e
16432393-efbcd52b-7e87-4b93-88f9-5014b10a26e8
1501710-22671d6d-74c0-44af-b258-3fa06f4c920c
35240-f1494990-6b2e-4273-b056-b290ffedd5ef
12753718-5be1b6d7-08ab-4261-91f0-bb72b4a2e05f
14273717-e99367b9-beeb-45e1-a20a-89cdf1e72633
6358444-bc6887b7-b4bc-4790-8051-48635f1238b9
12400166-c14fa025-46e8-43f3-8afe-70b28991ce3b
2871169-89a00bc7-a6a2-4621-8a3d-39f87d54fe37
26163449-ca20c329-0fc4-4c24-a0d0-550fd223a07f
8096848-d392048f-b03d-4903-809d-9ed2fb87cd3e
28147731-dfeaef05-2c45-4aae-8e2c-4da15fc0f9aa
26779181-2ad6f58c-7522-4e8d-beeb-24d5cb5ce956
26843617-046df96c-cf09-434e-bc50-8b15708b0b49
16080251-204b311a-6b59-42cc-8de2-59ba8f23723e
8522016-0a15778a-ccb1-4676-98b7-4cf1fe7fc940
25716737-a577321c-a48f-4972-a25d-64bf6da5d9dc
15735736-0714ad52-4119-4cea-a479-e09838fc27a7
15021209-d5993c3b-5c2c-4303-b7c2-0ab5738e23df
464809-1134a05b-3d75-4601-8cc6-ed35b14895da
15947626-7ce95503-d574-4fa7-9bb9-eb8f3b8c5f88
5997823-b874fef4-d6c4-45f1-a7f7-caf0560f13b3
26488730-bc1bdca1-977e-4c98-b36c-ca8ccc5bf6c0
14272639-15de1271-219e-458f-892d-09bb85bc1297

Ln 3406, Col 1
```

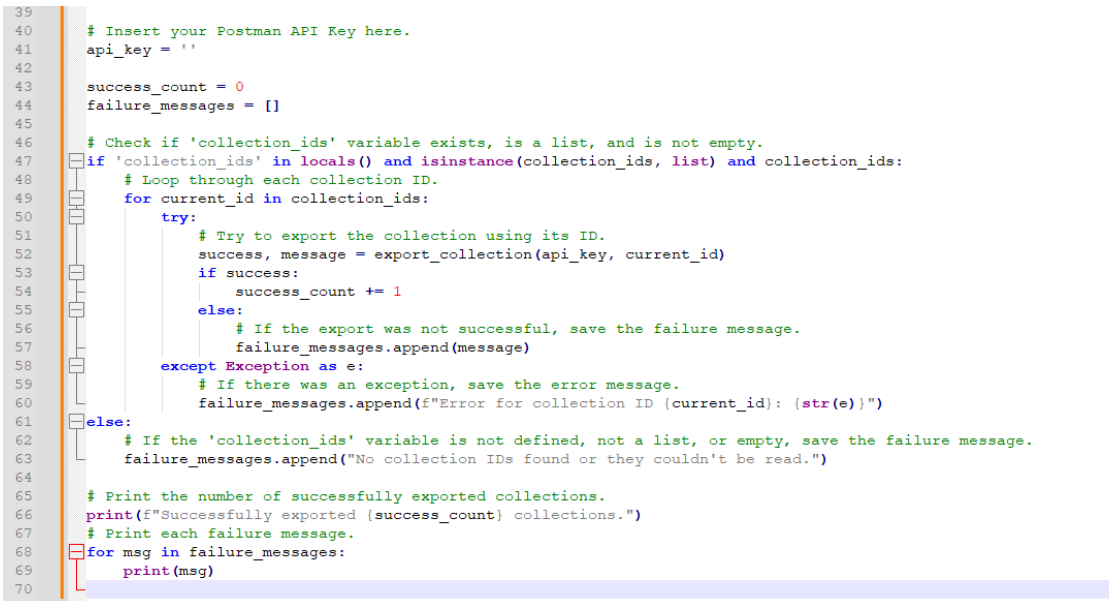
6.2. Exporting API Collections

With the collection IDs, Postman's API was used to export the collections in JSON format. Due to the 1000 API calls/month limit, an alternative solution was required.

Script

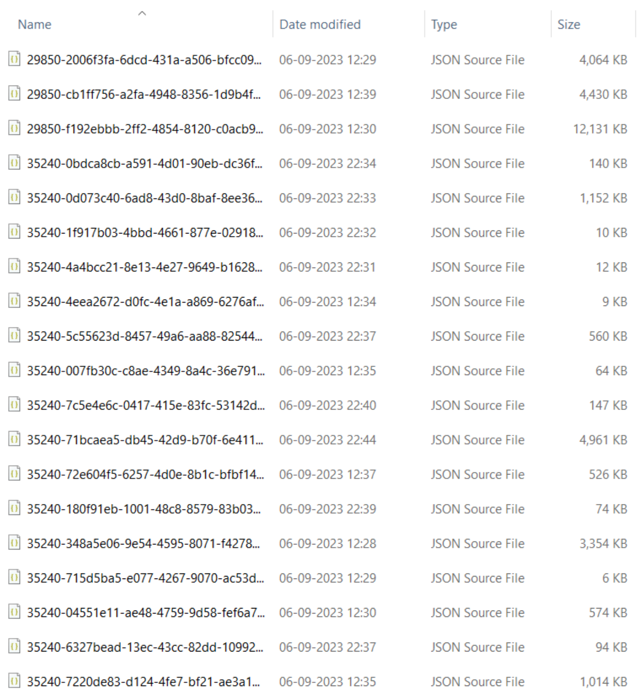


```python
import requests
import json
import os

# Read collection IDs from the text file into a list.
with open("collection_ids.txt", "r") as file:
    collection_ids = [line.strip() for line in file.readlines()]

# Define the directory where the exported JSON files will be saved.
output_dir = "C:\\Users\\LEGION\\Desktop\\Securin\\API JSON"
# Check if the directory exists, if not, create it.
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Define a function to export a collection using its ID.
def export_collection(api_key, collection_id):
    # Define the URL for the Postman API endpoint.
    url = f'https://api.getpostman.com/collections/{collection_id}'

    # Set up the headers, including the API key.
    headers = {
        'X-Api-Key': api_key,
    }

    # Make the GET request to fetch the collection data.
    response = requests.get(url, headers=headers)

    # Check if the request was successful.
    if response.status_code == 200:
        # Construct the path where the JSON file will be saved.
        file_path = os.path.join(output_dir, f"{collection_id}.json")
        # Write the response data to the JSON file.
        with open(file_path, 'w') as file:
            json.dump(response.json(), file, indent=4)
        return True, f"Collection {collection_id} exported successfully."
    else:
        # If the request was not successful, return a failure message.
        return False, f"Failed to export the collection {collection_id}. Check your API key."

# Insert your Postman API Key here.
api_key = ''
```



```python

# Insert your Postman API Key here.
api_key = ''

success_count = 0
failure_messages = []

# Check if 'collection_ids' variable exists, is a list, and is not empty.
if 'collection_ids' in locals() and isinstance(collection_ids, list) and collection_ids:
    # Loop through each collection ID.
    for current_id in collection_ids:
        try:
            # Try to export the collection using its ID.
            success, message = export_collection(api_key, current_id)
            if success:
                success_count += 1
            else:
                # If the export was not successful, save the failure message.
                failure_messages.append(message)
        except Exception as e:
            # If there was an exception, save the error message.
            failure_messages.append(f"Error for collection ID {current_id}: {str(e)}")
else:
    # If the 'collection_ids' variable is not defined, not a list, or empty, save the failure message.
    failure_messages.append("No collection IDs found or they couldn't be read.")

# Print the number of successfully exported collections.
print(f"Successfully exported {success_count} collections.")
# Print each failure message.
for msg in failure_messages:
    print(msg)
```

OUTPUT : This Output is an example for the above Script for reference .(Should change directories accordingly)

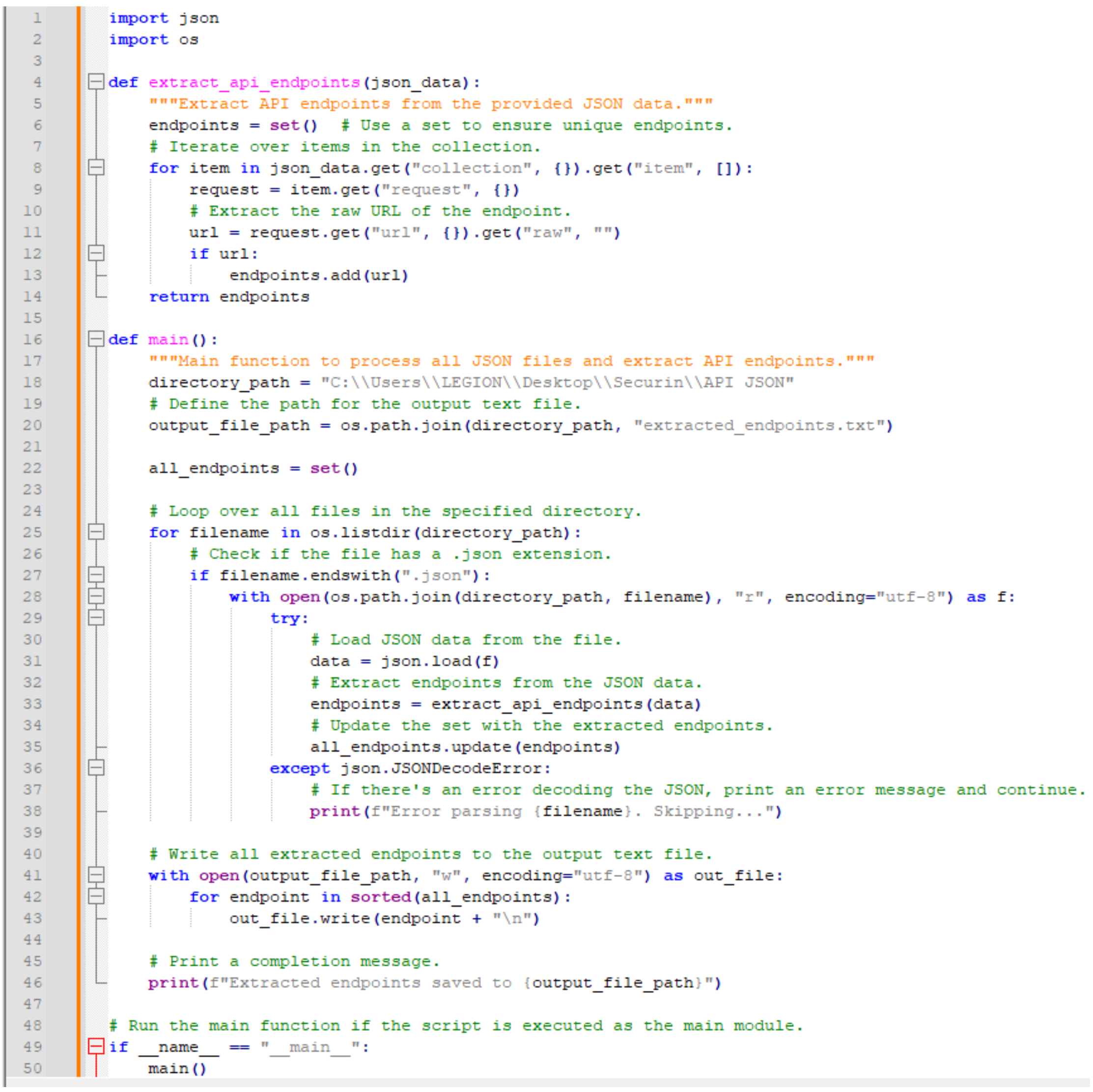


Name	Date modified	Type	Size
29850-2006f3fa-6dcd-431a-a506-bfcc09...	06-09-2023 12:29	JSON Source File	4,064 KB
29850-cb1ff756-a2fa-4948-8356-1d9b4f...	06-09-2023 12:39	JSON Source File	4,430 KB
29850-f192ebbb-2ff2-4854-8120-c0acb9...	06-09-2023 12:30	JSON Source File	12,131 KB
35240-0bdca8cb-a591-4d01-90eb-dc36f...	06-09-2023 22:34	JSON Source File	140 KB
35240-0d073c40-6ad8-43d0-8baf-8ee36...	06-09-2023 22:33	JSON Source File	1,152 KB
35240-1f917b03-4bbd-4661-877e-02918...	06-09-2023 22:32	JSON Source File	10 KB
35240-4a4bcc21-8e13-4e27-9649-b1628...	06-09-2023 22:31	JSON Source File	12 KB
35240-4eea2672-d0fc-4e1a-a869-6276af...	06-09-2023 12:34	JSON Source File	9 KB
35240-5c55623d-8457-49a6-aa88-82544...	06-09-2023 22:37	JSON Source File	560 KB
35240-007fb30c-c8ae-4349-8a4c-36e791...	06-09-2023 12:35	JSON Source File	64 KB
35240-7c5e4e6c-0417-415e-83fc-53142d...	06-09-2023 22:40	JSON Source File	147 KB
35240-71bcaea5-db45-42d9-b70f-6e411...	06-09-2023 22:44	JSON Source File	4,961 KB
35240-72e604f5-6257-4d0e-8b1c-bfbf14...	06-09-2023 12:37	JSON Source File	526 KB
35240-180f91eb-1001-48c8-8579-83b03...	06-09-2023 22:39	JSON Source File	74 KB
35240-348a5e06-9e54-4595-8071-f4278...	06-09-2023 12:28	JSON Source File	3,354 KB
35240-715d5ba5-e077-4267-9070-ac53d...	06-09-2023 12:29	JSON Source File	6 KB
35240-04551e11-ae48-4759-9d58-fef6a7...	06-09-2023 12:30	JSON Source File	574 KB
35240-6327bead-13ec-43cc-82dd-10992...	06-09-2023 22:37	JSON Source File	94 KB
35240-7220de83-d124-4fe7-bf21-ae3a1...	06-09-2023 12:35	JSON Source File	1,014 KB

6.3. Extracting API Endpoints

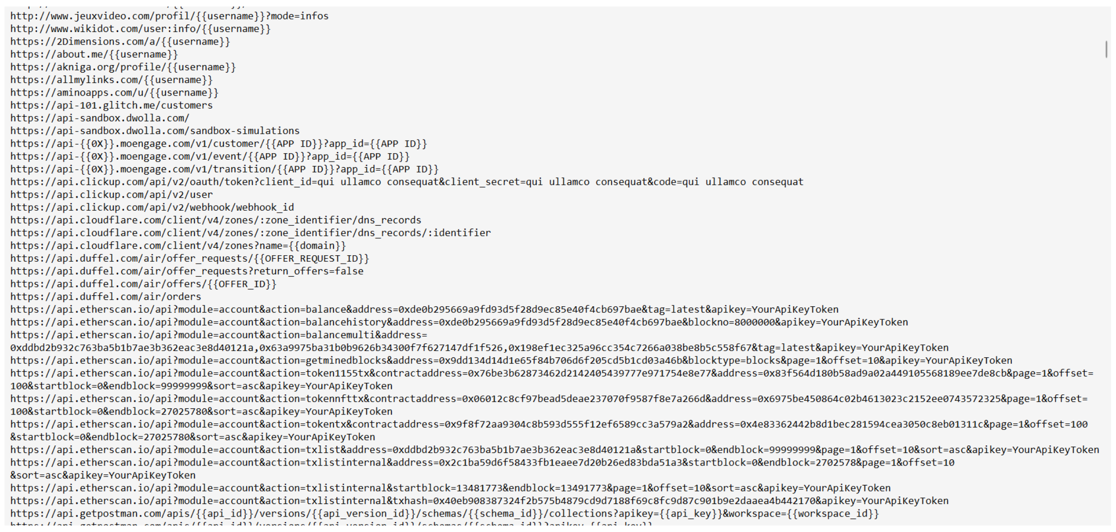
A script was developed to extract relevant API endpoints from the exported JSON data.

CODE:



```python
import json
import os

def extract_api_endpoints(json_data):
    """Extract API endpoints from the provided JSON data."""
    endpoints = set()  # Use a set to ensure unique endpoints.
    # Iterate over items in the collection.
    for item in json_data.get("collection", {}).get("item", []):
        request = item.get("request", {})
        # Extract the raw URL of the endpoint.
        url = request.get("url", {}).get("raw", "")
        if url:
            endpoints.add(url)
    return endpoints

def main():
    """Main function to process all JSON files and extract API endpoints."""
    directory_path = "C:\\Users\\LEGION\\Desktop\\Securin\\API JSON"
    # Define the path for the output text file.
    output_file_path = os.path.join(directory_path, "extracted_endpoints.txt")

    all_endpoints = set()

    # Loop over all files in the specified directory.
    for filename in os.listdir(directory_path):
        # Check if the file has a .json extension.
        if filename.endswith(".json"):
            with open(os.path.join(directory_path, filename), "r", encoding="utf-8") as f:
                try:
                    # Load JSON data from the file.
                    data = json.load(f)
                    # Extract endpoints from the JSON data.
                    endpoints = extract_api_endpoints(data)
                    # Update the set with the extracted endpoints.
                    all_endpoints.update(endpoints)
                except json.JSONDecodeError:
                    # If there's an error decoding the JSON, print an error message and continue.
                    print(f"Error parsing {filename}. Skipping...")

    # Write all extracted endpoints to the output text file.
    with open(output_file_path, "w", encoding="utf-8") as out_file:
        for endpoint in sorted(all_endpoints):
            out_file.write(endpoint + "\n")

    # Print a completion message.
    print(f"Extracted endpoints saved to {output_file_path}")

# Run the main function if the script is executed as the main module.
if __name__ == "__main__":
    main()
```

OUTPUT : The path of the Script should be changed Accordingly.

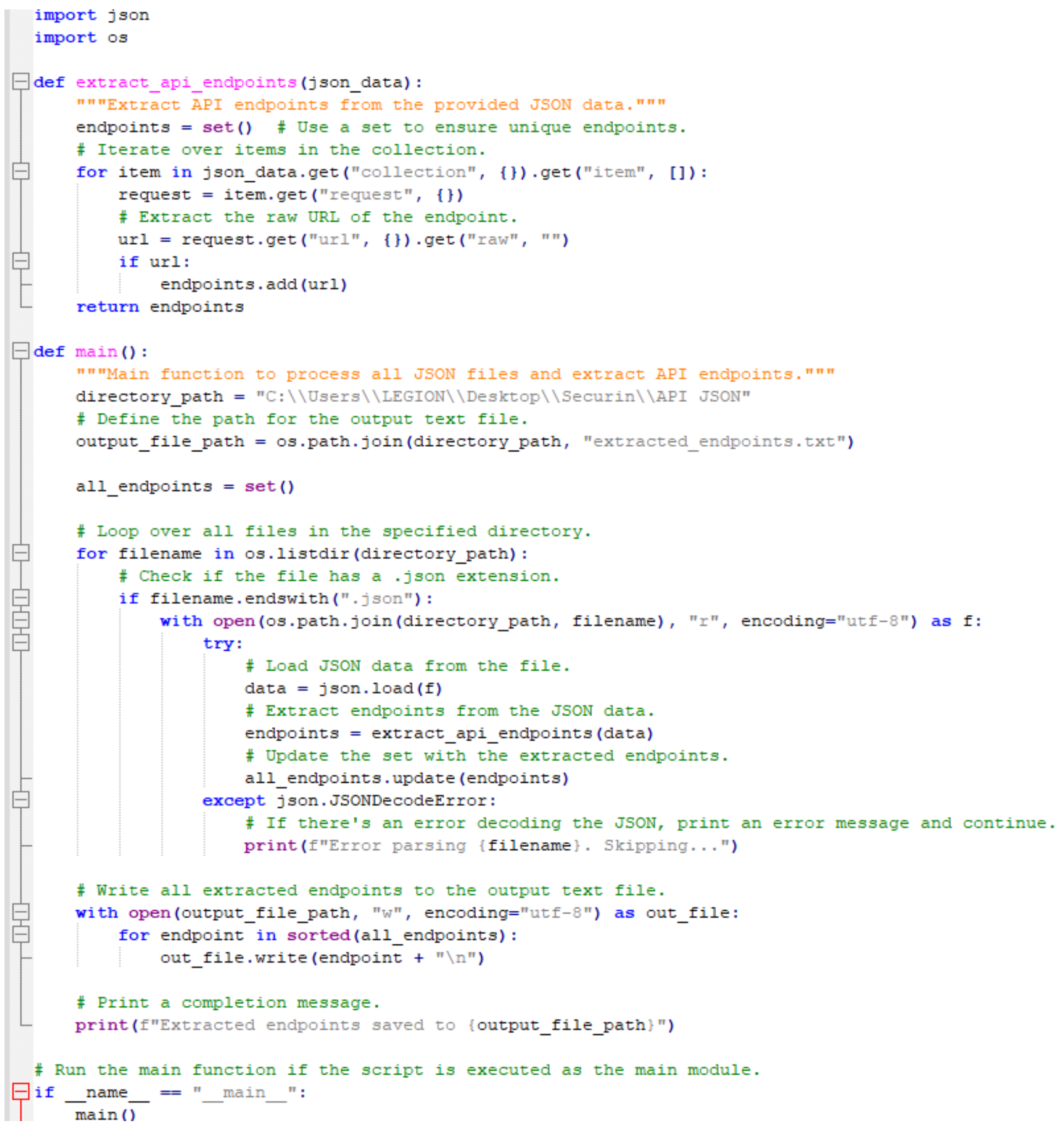


```
http://www.jeuxvideo.com/profil/{{username}}?mode=infos
http://www.wikidot.com/user:info/{{username}}
https://2Dimensions.com/a/{{username}}
https://about.me/{{username}}
https://akniga.org/profile/{{username}}
https://allmylinks.com/{{username}}
https://aminoapps.com/u/{{username}}
https://api-101.glitch.me/customers
https://api-sandbox.dwolla.com/
https://api-sandbox.dwolla.com/sandbox-simulations
https://api-{{0X}}.moengage.com/v1/customer/{{APP ID}}?app_id={{APP ID}}
https://api-{{0X}}.moengage.com/v1/event/{{APP ID}}?app_id={{APP ID}}
https://api-{{0X}}.moengage.com/v1/transition/{{APP ID}}?app_id={{APP ID}}
https://api.clickup.com/api/v2/oauth/token?client_id=qui ullamco consequat&client_secret=qui ullamco consequat&code=qui ullamco consequat
https://api.clickup.com/api/v2/user
https://api.clickup.com/api/v2/webhook/webhook_id
https://api.cloudflare.com/client/v4/zones/:zone_identifier/dns_records
https://api.cloudflare.com/client/v4/zones/:zone_identifier/dns_records/:identifier
https://api.cloudflare.com/client/v4/zones?name={{domain}}
https://api.duffel.com/air/offer_requests/{{OFFER_REQUEST_ID}}
https://api.duffel.com/air/offer_requests?return_offers=false
https://api.duffel.com/air/offers/{{OFFER_ID}}
https://api.duffel.com/air/orders
https://api.etherscan.io/api?module=account&action=balance&address=0xde0b295669a9fd93d5f28d9ec85e40f4cb697bae&tag=latest&apikey=YourApiKeyToken
https://api.etherscan.io/api?module=account&action=balancehistory&address=0xde0b295669a9fd93d5f28d9ec85e40f4cb697bae&blockno=8000000&apikey=YourApiKeyToken
https://api.etherscan.io/api?module=account&action=balancemulti&address=
0xddbd2b932c763ba5b1b7ae3b362eac3e8d40121a,0x63a9975ba31b0b9626b34300f7f627147df1f526,0x198ef1ec325a96cc354c7266a038be8b5c558f67&tag=latest&apikey=YourApiKeyToken
https://api.etherscan.io/api?module=account&action=getminedblocks&address=0x9dd134d14d1e65f84b706d6f205cd5b1cd03a46b&blocktype=blocks&page=1&offset=10&apikey=YourApiKeyToken
https://api.etherscan.io/api?module=account&action=token1155tx&contractaddress=0x76be3b62873462d2142405439777e971754e8e77&address=0x83f564d180b58ad9a02a449105568189ee7de8cb&page=1&offset=
100&startblock=0&endblock=99999999&sort=asc&apikey=YourApiKeyToken
https://api.etherscan.io/api?module=account&action=tokennfttx&contractaddress=0x06012c8cf97bead5deae237070f9587f8e7a266d&address=0x6975be450864c02b4613023c2152ee0743572325&page=1&offset=
100&startblock=0&endblock=27025780&sort=asc&apikey=YourApiKeyToken
https://api.etherscan.io/api?module=account&action=tokentx&contractaddress=0x9f8f72aa9304c8b593d555f12ef6589cc3a579a2&address=0x4e83362442b8d1bec281594cea3050c8eb01311c&page=1&offset=100
&startblock=0&endblock=27025780&sort=asc&apikey=YourApiKeyToken
https://api.etherscan.io/api?module=account&action=txlist&address=0xddbd2b932c763ba5b1b7ae3b362eac3e8d40121a&startblock=0&endblock=99999999&page=1&offset=10&sort=asc&apikey=YourApiKeyToken
https://api.etherscan.io/api?module=account&action=txlistinternal&address=0x2c1ba59d6f58433fb1eaee7d20b26ed83bda51a3&startblock=0&endblock=2702578&page=1&offset=10
&sort=asc&apikey=YourApiKeyToken
https://api.etherscan.io/api?module=account&action=txlistinternal&startblock=13481773&endblock=13491773&page=1&offset=10&sort=asc&apikey=YourApiKeyToken
https://api.etherscan.io/api?module=account&action=txlistinternal&txhash=0x40eb908387324f2b575b4879cd9d7188f69c8fc9d87c901b9e2daaea4b442170&apikey=YourApiKeyToken
https://api.getpostman.com/apis/{{api_id}}/versions/{{api_version_id}}/schemas/{{schema_id}}/collections?apikey={{api_key}}&workspace={{workspace_id}}
```

6.4. Organizing Data

A relational database was proposed to manage the extracted data efficiently. This design consists of two tables: `Data` and `Meta`. Before that we collected all the json format data in a textfile . Then later we exported the data into a CSV File for better understanding of the data . Here below attaching the description of the Textfile data.

Code :



```python
import json
import os

def extract_api_endpoints(json_data):
    """Extract API endpoints from the provided JSON data."""
    endpoints = set()  # Use a set to ensure unique endpoints.
    # Iterate over items in the collection.
    for item in json_data.get("collection", {}).get("item", []):
        request = item.get("request", {})
        # Extract the raw URL of the endpoint.
        url = request.get("url", {}).get("raw", "")
        if url:
            endpoints.add(url)
    return endpoints

def main():
    """Main function to process all JSON files and extract API endpoints."""
    directory_path = "C:\\Users\\LEGION\\Desktop\\Securin\\API JSON"
    # Define the path for the output text file.
    output_file_path = os.path.join(directory_path, "extracted_endpoints.txt")

    all_endpoints = set()

    # Loop over all files in the specified directory.
    for filename in os.listdir(directory_path):
        # Check if the file has a .json extension.
        if filename.endswith(".json"):
            with open(os.path.join(directory_path, filename), "r", encoding="utf-8") as f:
                try:
                    # Load JSON data from the file.
                    data = json.load(f)
                    # Extract endpoints from the JSON data.
                    endpoints = extract_api_endpoints(data)
                    # Update the set with the extracted endpoints.
                    all_endpoints.update(endpoints)
                except json.JSONDecodeError:
                    # If there's an error decoding the JSON, print an error message and continue.
                    print(f"Error parsing {filename}. Skipping...")

    # Write all extracted endpoints to the output text file.
    with open(output_file_path, "w", encoding="utf-8") as out_file:
        for endpoint in sorted(all_endpoints):
            out_file.write(endpoint + "\n")

    # Print a completion message.
    print(f"Extracted endpoints saved to {output_file_path}")

# Run the main function if the script is executed as the main module.
if __name__ == "__main__":
    main()
```

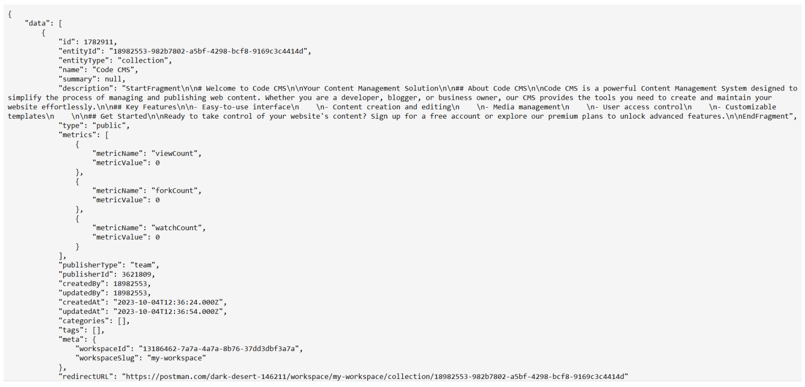
When you run the provided script, it fetches a list of data entities from the Postman API. Each entity corresponds to a "collection" in Postman, which is essentially a group of saved requests. The output provides a detailed breakdown of each collection.

Here's a description of the data structure for each collection entity:

id: A unique identifier for the collection.
entityId: A secondary identifier, potentially for different referencing.
entityType: Type of the entity. In this context, it's "collection".
name: Name given to the collection.
summary: A brief summary or description of the collection (might be null).

description: A detailed account of the collection. May include rich-text indicators like "StartFragment" and "EndFragment".
type: Denotes the accessibility level. "public" indicates it's open for public access.
metrics: Metrics associated with the collection:
viewCount: Times the collection has been viewed.
forkCount: Number of forks or copies made from the collection.
watchCount: Number of subscribers or watchers.
publisherType: Indicates if a user or a team published the collection.
publisherId: Unique ID for the publisher.
createdBy: ID of the user who initiated the collection.
updatedBy: ID of the user who last updated the collection.
createdAt: Timestamp marking the creation of the collection.
updatedAt: Timestamp of the most recent update.
categories: Categories tagged with the collection (can be empty).
tags: Tags linked with the collection (can be empty).
meta: Additional metadata for the collection. This includes:
workspaceId: ID for the workspace housing the collection.
workspaceSlug: A user-friendly descriptor or slug for the workspace.
redirectURL: Direct URL to the collection on the Postman platform.
By understanding this structure, you can efficiently navigate and utilize the data fetched from the Postman API.

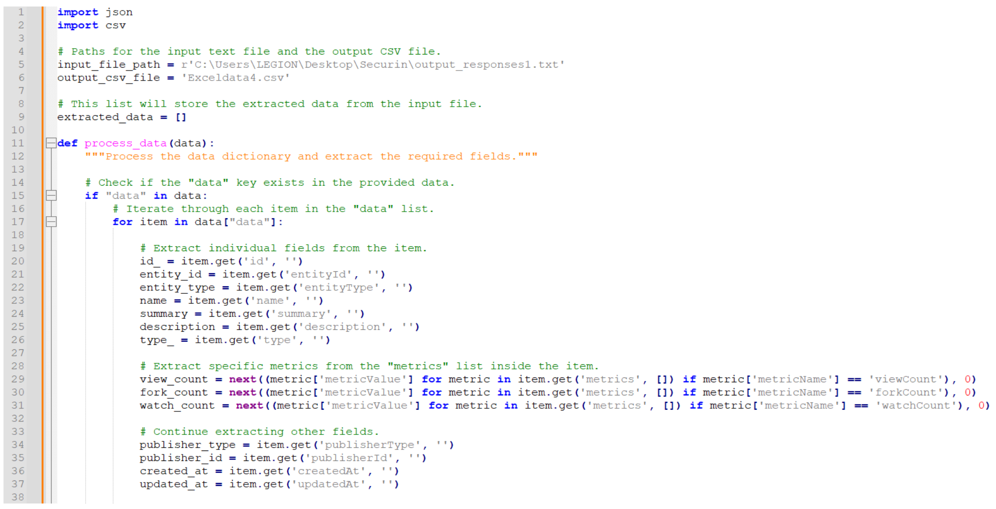
OUTPUT :



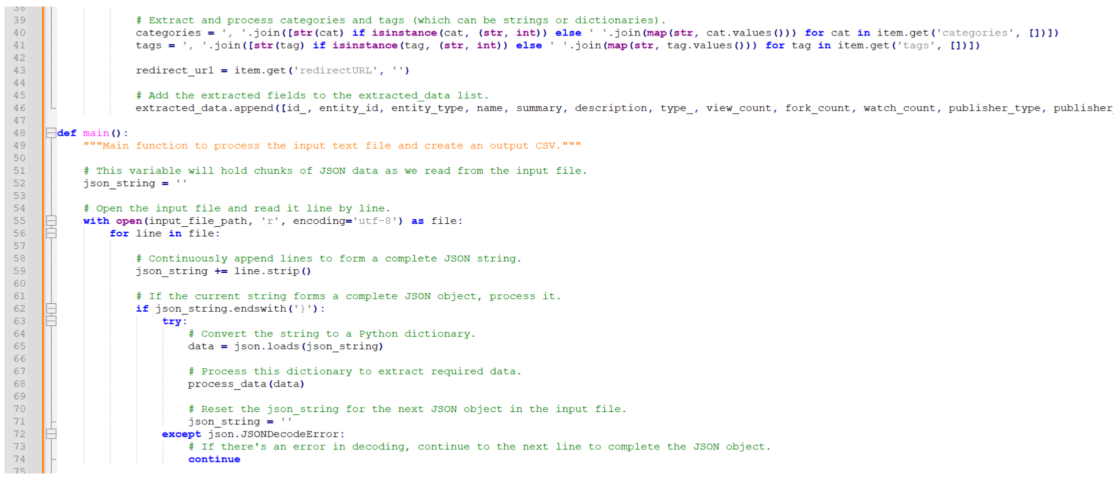
```
{
    "data": [
        {
            "id": 1782911,
            "entityId": "18982553-982b7802-a5bf-4298-bcf8-9169c3c4414d",
            "entityType": "collection",
            "name": "Code CMS",
            "summary": null,
            "description": "StartFragment\n\n# Welcome to Code CMS\n\nYour Content Management Solution\n\n## About Code CMS\n\nCode CMS is a powerful Content Management System designed to
simplify the process of managing and publishing web content. Whether you are a developer, blogger, or business owner, our CMS provides the tools you need to create and maintain your
website effortlessly.\n\n## Key Features\n\n- Easy-to-use interface\n    \n- Content creation and editing\n    \n- Media management\n    \n- User access control\n    \n- Customizable
templates\n    \n\n## Get Started\n\nReady to take control of your website's content? Sign up for a free account or explore our premium plans to unlock advanced features.\n\nEndFragment",
            "type": "public",
            "metrics": [
                {
                    "metricName": "viewCount",
                    "metricValue": 0
                },
                {
                    "metricName": "forkCount",
                    "metricValue": 0
                },
                {
                    "metricName": "watchCount",
                    "metricValue": 0
                }
            ],
            "publisherType": "team",
            "publisherId": 3621809,
            "createdBy": 18982553,
            "updatedBy": 18982553,
            "createdAt": "2023-10-04T12:36:24.000Z",
            "updatedAt": "2023-10-04T12:36:54.000Z",
            "categories": [],
            "tags": [],
            "meta": {
                "workspaceId": "13186462-7a7a-4a7a-8b76-37dd3dbf3a7a",
                "workspaceSlug": "my-workspace"
            },
            "redirectURL": "https://postman.com/dark-desert-146211/workspace/my-workspace/collection/18982553-982b7802-a5bf-4298-bcf8-9169c3c4414d"
```

Then After getting the Textfile , Later Extracted specific Details from the textfile and added it in the CSV File For better understanding .

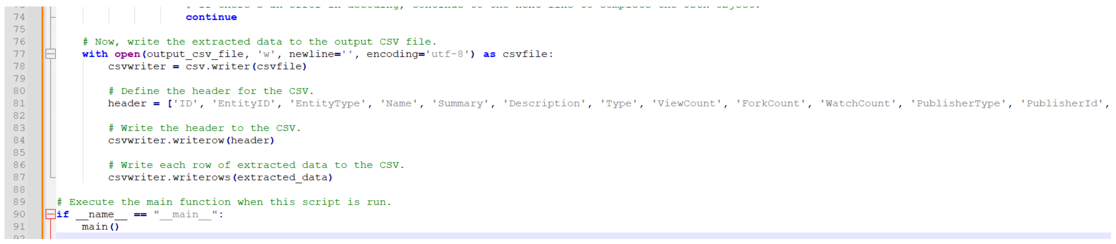
Code :



```python
import json
import csv

# Paths for the input text file and the output CSV file.
input_file_path = r'C:\Users\LEGION\Desktop\Securin\output_responses1.txt'
output_csv_file = 'Exceldata4.csv'

# This list will store the extracted data from the input file.
extracted_data = []

def process_data(data):
    """Process the data dictionary and extract the required fields."""

    # Check if the "data" key exists in the provided data.
    if "data" in data:
        # Iterate through each item in the "data" list.
        for item in data["data"]:

            # Extract individual fields from the item.
            id_ = item.get('id', '')
            entity_id = item.get('entityId', '')
            entity_type = item.get('entityType', '')
            name = item.get('name', '')
            summary = item.get('summary', '')
            description = item.get('description', '')
            type_ = item.get('type', '')

            # Extract specific metrics from the "metrics" list inside the item.
            view_count = next((metric['metricValue'] for metric in item.get('metrics', []) if metric['metricName'] == 'viewCount'), 0)
            fork_count = next((metric['metricValue'] for metric in item.get('metrics', []) if metric['metricName'] == 'forkCount'), 0)
            watch_count = next((metric['metricValue'] for metric in item.get('metrics', []) if metric['metricName'] == 'watchCount'), 0)

            # Continue extracting other fields.
            publisher_type = item.get('publisherType', '')
            publisher_id = item.get('publisherId', '')
            created_at = item.get('createdAt', '')
            updated_at = item.get('updatedAt', '')

```

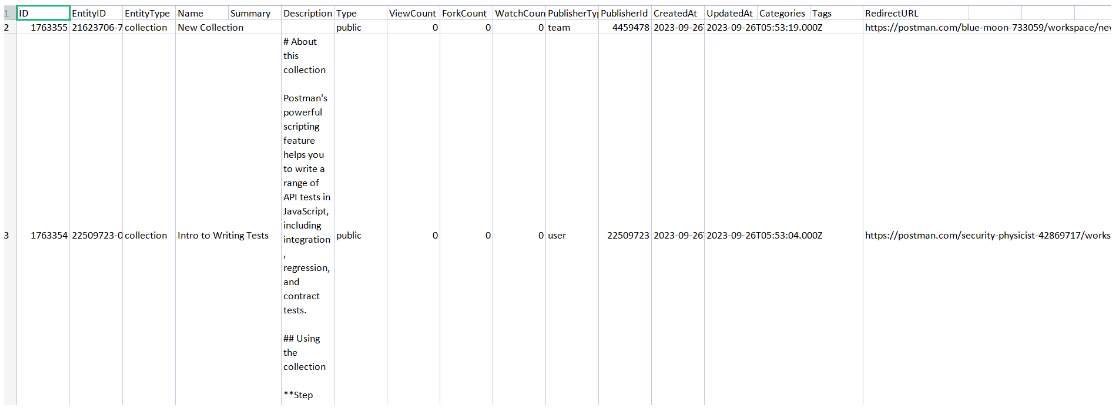


```python
            # Extract and process categories and tags (which can be strings or dictionaries).
            categories = ', '.join([str(cat) if isinstance(cat, (str, int)) else ' '.join(map(str, cat.values())) for cat in item.get('categories', [])])
            tags = ', '.join([str(tag) if isinstance(tag, (str, int)) else ' '.join(map(str, tag.values())) for tag in item.get('tags', [])])

            redirect_url = item.get('redirectURL', '')

            # Add the extracted fields to the extracted_data list.
            extracted_data.append([id_, entity_id, entity_type, name, summary, description, type_, view_count, fork_count, watch_count, publisher_type, publisher

def main():
    """Main function to process the input text file and create an output CSV."""

    # This variable will hold chunks of JSON data as we read from the input file.
    json_string = ''

    # Open the input file and read it line by line.
    with open(input_file_path, 'r', encoding='utf-8') as file:
        for line in file:

            # Continuously append lines to form a complete JSON string.
            json_string += line.strip()

            # If the current string forms a complete JSON object, process it.
            if json_string.endswith('}'):
                try:
                    # Convert the string to a Python dictionary.
                    data = json.loads(json_string)

                    # Process this dictionary to extract required data.
                    process_data(data)

                    # Reset the json_string for the next JSON object in the input file.
                    json_string = ''
                except json.JSONDecodeError:
                    # If there's an error in decoding, continue to the next line to complete the JSON object.
                    continue
```



```python
                    continue

    # Now, write the extracted data to the output CSV file.
    with open(output_csv_file, 'w', newline='', encoding='utf-8') as csvfile:
        csvwriter = csv.writer(csvfile)

        # Define the header for the CSV.
        header = ['ID', 'EntityID', 'EntityType', 'Name', 'Summary', 'Description', 'Type', 'ViewCount', 'ForkCount', 'WatchCount', 'PublisherType', 'PublisherId',

        # Write the header to the CSV.
        csvwriter.writerow(header)

        # Write each row of extracted data to the CSV.
        csvwriter.writerows(extracted_data)

# Execute the main function when this script is run.
if __name__ == "__main__":
    main()
```

OUTPUT :

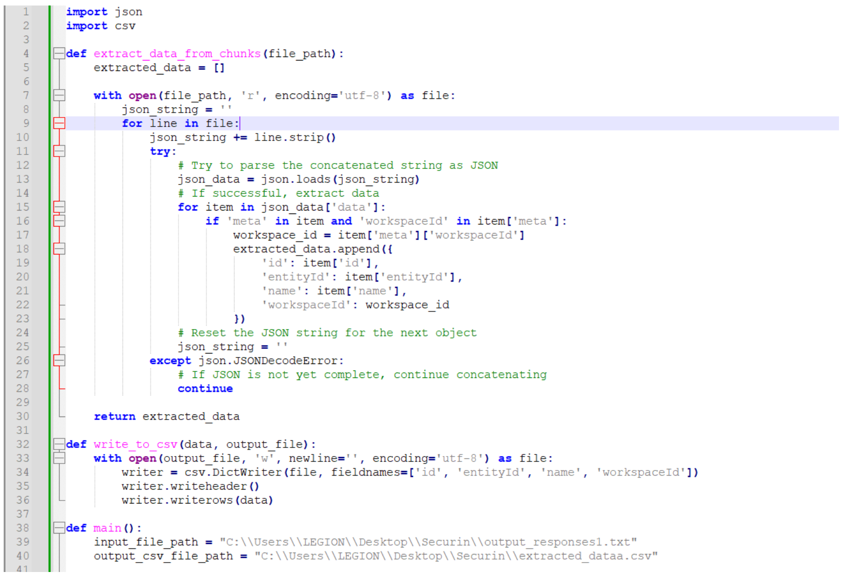


ID	EntityID	EntityType	Name	Summary	Description	Type	ViewCount	ForkCount	WatchCoun	PublisherTy	PublisherId	CreatedAt	UpdatedAt	Categories	Tags	RedirectURL
1763355	21623706-7	collection	New Collection			public	0	0	0	team	4459478	2023-09-26	2023-09-26T05:53:19.000Z			https://postman.com/blue-moon-733059/workspace/ne
1763354	22509723-0	collection	Intro to Writing Tests		# About this collection Postman's powerful scripting feature helps you to write a range of API tests in JavaScript, including integration , regression, and contract tests. ## Using the collection **Step	public	0	0	0	user	22509723	2023-09-26	2023-09-26T05:53:04.000Z			https://postman.com/security-physicist-42869717/works

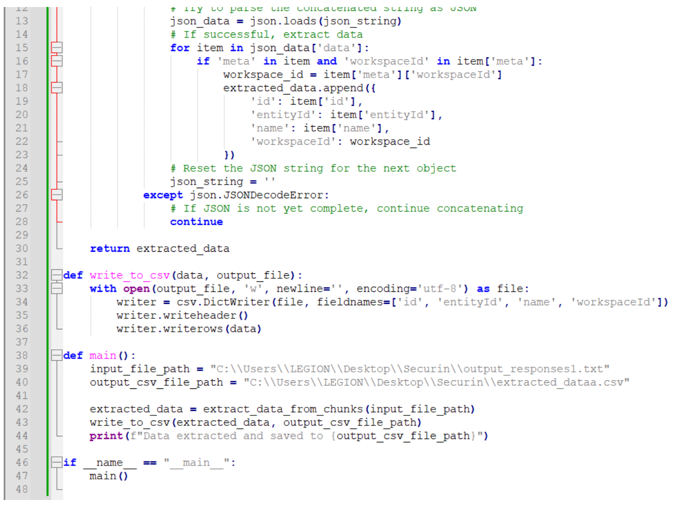
Notes:
Ensure that the input file path points to the correct location of your JSON text file.
Ensure that the directory for the output CSV file exists; otherwise, there might be errors during file writing.

Following the exploration phase, the project progresses to collect workspace IDs in Postman. Workspaces in Postman are collaborative spaces where APIs can be developed and tested. By gathering the workspace IDs, the project aims to organize and reference the APIs more effectively. The significance of collecting workspace IDs lies in their role in generating environment IDs. Environment IDs in Postman are used to manage different sets of variables, such as base URLs, which are essential for the APIs to function correctly in different contexts. These variables are vital as they can change based on the environment the API is operating in, such as development, testing, or production. Collecting and managing these environment IDs ensures that the APIs can be configured and used appropriately in different scenarios.

CODE :



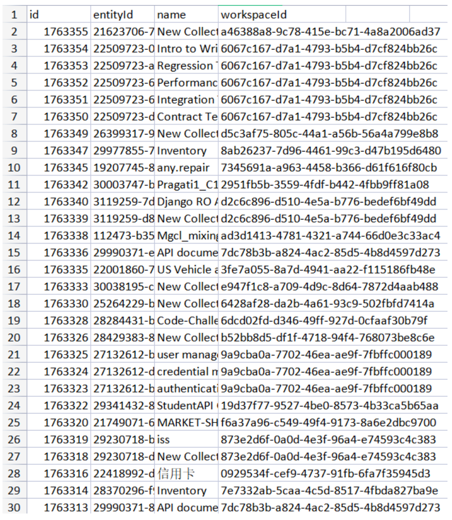
```python
import json
import csv

def extract_data_from_chunks(file_path):
    extracted_data = []

    with open(file_path, 'r', encoding='utf-8') as file:
        json_string = ''
        for line in file:
            json_string += line.strip()
            try:
                # Try to parse the concatenated string as JSON
                json_data = json.loads(json_string)
                # If successful, extract data
                for item in json_data['data']:
                    if 'meta' in item and 'workspaceId' in item['meta']:
                        workspace_id = item['meta']['workspaceId']
                        extracted_data.append({
                            'id': item['id'],
                            'entityId': item['entityId'],
                            'name': item['name'],
                            'workspaceId': workspace_id
                        })
                # Reset the JSON string for the next object
                json_string = ''
            except json.JSONDecodeError:
                # If JSON is not yet complete, continue concatenating
                continue

    return extracted_data

def write_to_csv(data, output_file):
    with open(output_file, 'w', newline='', encoding='utf-8') as file:
        writer = csv.DictWriter(file, fieldnames=['id', 'entityId', 'name', 'workspaceId'])
        writer.writeheader()
        writer.writerows(data)

def main():
    input_file_path = "C:\\Users\\LEGION\\Desktop\\Securin\\output_responses1.txt"
    output_csv_file_path = "C:\\Users\\LEGION\\Desktop\\Securin\\extracted_dataa.csv"
```



```python
                json_data = json.loads(json_string)
                # If successful, extract data
                for item in json_data['data']:
                    if 'meta' in item and 'workspaceId' in item['meta']:
                        workspace_id = item['meta']['workspaceId']
                        extracted_data.append({
                            'id': item['id'],
                            'entityId': item['entityId'],
                            'name': item['name'],
                            'workspaceId': workspace_id
                        })
                # Reset the JSON string for the next object
                json_string = ''
            except json.JSONDecodeError:
                # If JSON is not yet complete, continue concatenating
                continue

    return extracted_data

def write_to_csv(data, output_file):
    with open(output_file, 'w', newline='', encoding='utf-8') as file:
        writer = csv.DictWriter(file, fieldnames=['id', 'entityId', 'name', 'workspaceId'])
        writer.writeheader()
        writer.writerows(data)

def main():
    input_file_path = "C:\\Users\\LEGION\\Desktop\\Securin\\output_responses1.txt"
    output_csv_file_path = "C:\\Users\\LEGION\\Desktop\\Securin\\extracted_dataa.csv"
```

```python

    extracted_data = extract_data_from_chunks(input_file_path)
    write_to_csv(extracted_data, output_csv_file_path)
    print(f"Data extracted and saved to {output_csv_file_path}")

if __name__ == "__main__":
    main()
```

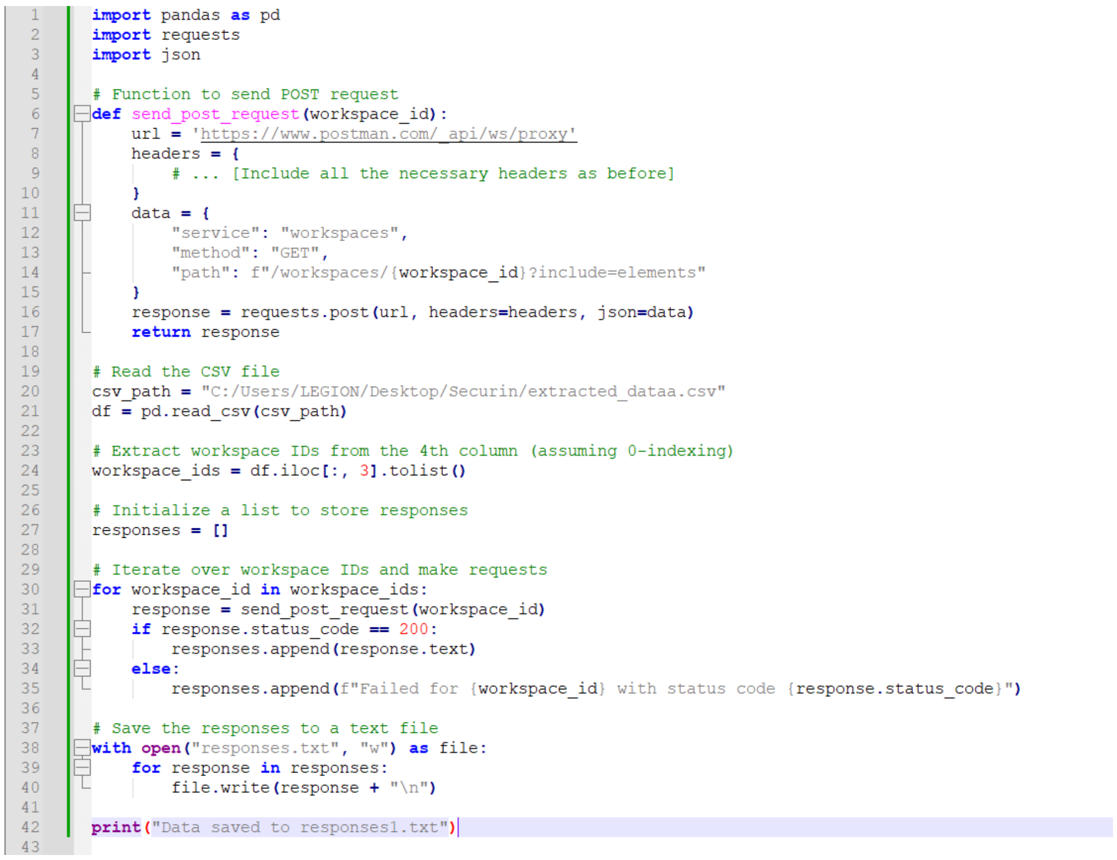
OUTPUT :



1	id	entityId	name	workspaceId
2	1763355	21623706-7	New Collect	a46388a8-9c78-415e-bc71-4a8a2006ad37
3	1763354	22509723-0	Intro to Wri	6067c167-d7a1-4793-b5b4-d7cf824bb26c
4	1763353	22509723-a	Regression	6067c167-d7a1-4793-b5b4-d7cf824bb26c
5	1763352	22509723-6	Performanc	6067c167-d7a1-4793-b5b4-d7cf824bb26c
6	1763351	22509723-6	Integration	6067c167-d7a1-4793-b5b4-d7cf824bb26c
7	1763350	22509723-d	Contract Te	6067c167-d7a1-4793-b5b4-d7cf824bb26c
8	1763349	26399317-9	New Collect	d5c3af75-805c-44a1-a56b-56a4a799e8b8
9	1763347	29977855-7	Inventory	8ab26237-7d96-4461-99c3-d47b195d6480
10	1763345	19207745-8	any.repair	7345691a-a963-4458-b366-d61f616f80cb
11	1763342	30003747-b	Pragati1_C1	2951fb5b-3559-4fdf-b442-4fbb9ff81a08
12	1763340	3119259-7d	Django RO /	d2c6c896-d510-4e5a-b776-bedef6bf49dd
13	1763339	3119259-d8	New Collect	d2c6c896-d510-4e5a-b776-bedef6bf49dd
14	1763338	112473-b35	Mgcl_mixin	ad3d1413-4781-4321-a744-66d0e3c33ac4
15	1763336	29990371-e	API docume	7dc78b3b-a824-4ac2-85d5-4b8d4597d273
16	1763335	22001860-7	US Vehicle a	3fe7a055-8a7d-4941-aa22-f115186fb48e
17	1763333	30038195-c	New Collect	e947f1c8-a709-4d9c-8d64-7872d4aab488
18	1763330	25264229-b	New Collect	6428af28-da2b-4a61-93c9-502fbfd7414a
19	1763328	28284431-b	Code-Challe	6dcd02fd-d346-49ff-927d-0cfaaf30b79f
20	1763326	28429383-8	New Collect	b52bb8d5-df1f-4718-94f4-768073be8c6e
21	1763325	27132612-b	user manag	9a9cba0a-7702-46ea-ae9f-7fbffc000189
22	1763324	27132612-d	credential m	9a9cba0a-7702-46ea-ae9f-7fbffc000189
23	1763323	27132612-b	authenticati	9a9cba0a-7702-46ea-ae9f-7fbffc000189
24	1763322	29341432-8	StudentAPI (	19d37f77-9527-4be0-8573-4b33ca5b65aa
25	1763320	21749071-6	MARKET-SH	f6a37a96-c549-49f4-9173-8a6e2dbc9700
26	1763319	29230718-b	iss	873e2d6f-0a0d-4e3f-96a4-e74593c4c383
27	1763318	29230718-d	New Collect	873e2d6f-0a0d-4e3f-96a4-e74593c4c383
28	1763316	22418992-d	信用卡	0929534f-cef9-4737-91fb-6fa7f35945d3
29	1763314	28370296-f	Inventory	7e7332ab-5caa-4c5d-8517-4fbda827ba9e
30	1763313	29990371-8	API docume	7dc78b3b-a824-4ac2-85d5-4b8d4597d273

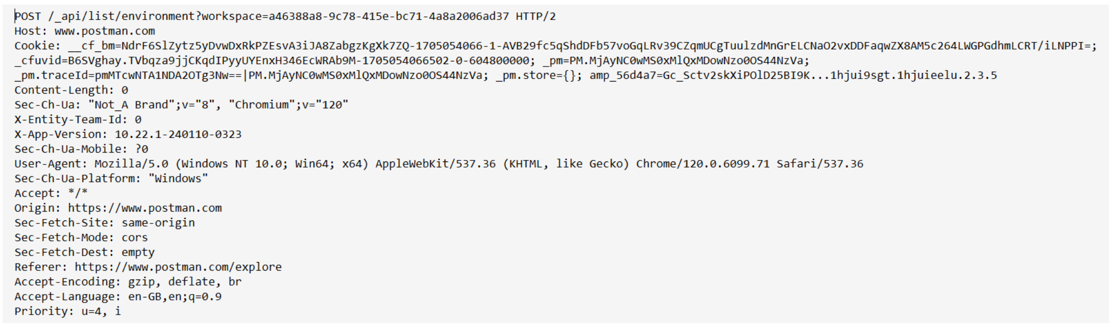
This showcases an HTTP POST request to Postman's API endpoint for fetching environment details from a specific workspace, an action likely part of an automated process developed in Python to collect workspace and environment IDs. This request, indicated by the /api/list/environment endpoint, is formulated to interact with Postman's API, including various headers such as User-Agent and cookies that are essential for session management and server communication. While the request's structure aligns with the expected parameters of the Postman API, including the workspace ID, the actual environment details and variables, such as base URLs, would be part of the API's response, which is not visible in the screenshot. This automated retrieval is a strategic component in managing multiple API environments efficiently, facilitating a seamless workflow for developers working across different stages of the API lifecycle.

CODE :

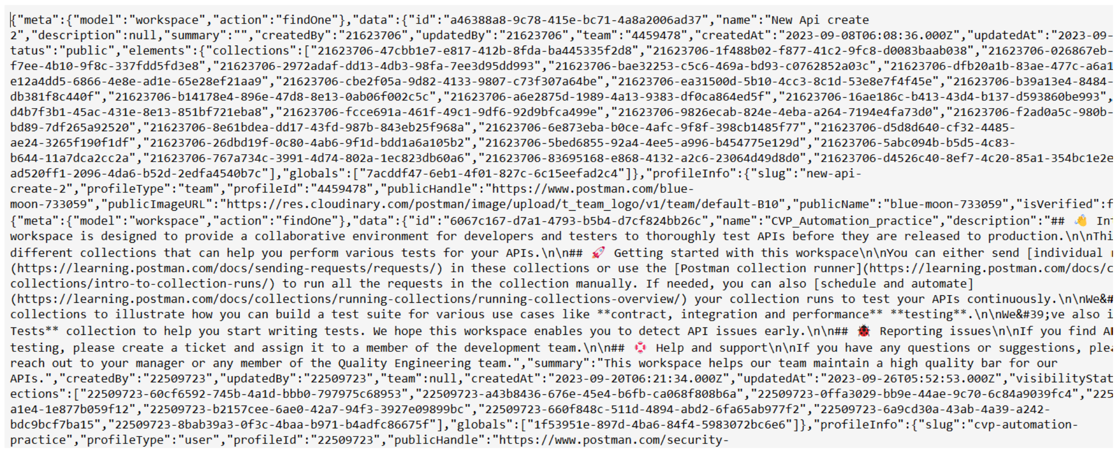


```python
import pandas as pd
import requests
import json

# Function to send POST request
def send_post_request(workspace_id):
    url = 'https://www.postman.com/_api/ws/proxy'
    headers = {
        # ... [Include all the necessary headers as before]
    }
    data = {
        "service": "workspaces",
        "method": "GET",
        "path": f"/workspaces/{workspace_id}?include=elements"
    }
    response = requests.post(url, headers=headers, json=data)
    return response

# Read the CSV file
csv_path = "C:/Users/LEGION/Desktop/Securin/extracted_dataa.csv"
df = pd.read_csv(csv_path)

# Extract workspace IDs from the 4th column (assuming 0-indexing)
workspace_ids = df.iloc[:, 3].tolist()

# Initialize a list to store responses
responses = []

# Iterate over workspace IDs and make requests
for workspace_id in workspace_ids:
    response = send_post_request(workspace_id)
    if response.status_code == 200:
        responses.append(response.text)
    else:
        responses.append(f"Failed for {workspace_id} with status code {response.status_code}")

# Save the responses to a text file
with open("responses.txt", "w") as file:
    for response in responses:
        file.write(response + "\n")

print("Data saved to responses1.txt")
```

OUTPUT :



```
POST /_api/list/environment?workspace=a46388a8-9c78-415e-bc71-4a8a2006ad37 HTTP/2
Host: www.postman.com
Cookie: __cf_bm=NdrF6SlZytz5yDvwDxRkPZEsvA3iJA8ZabgzKgXk7ZQ-1705054066-1-AVB29fc5qShdDFb57voGqLRv39CZqmUCgTuulzdMnGrELCNaO2vxDDFaqwZX8AM5c264LWGPGdhmLCRT/iLNPPI=;
_cfuvid=B6SVghay.TVbqza9jjCKqdIPyyUYEnxH346EcWRAb9M-1705054066502-0-604800000; _pm=PM.MjAyNC0wMS0xMlQxMDowNzo0OS44NzVa;
_pm.traceId=pmMTcwNTA1NDA2OTg3Nw==|PM.MjAyNC0wMS0xMlQxMDowNzo0OS44NzVa; _pm.store={}; amp_56d4a7=Gc_Sctv2skXiPOlD25BI9K...1hjui9sgt.1hjuieelu.2.3.5
Content-Length: 0
Sec-Ch-Ua: "Not_A Brand";v="8", "Chromium";v="120"
X-Entity-Team-Id: 0
X-App-Version: 10.22.1-240110-0323
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.6099.71 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Accept: */*
Origin: https://www.postman.com
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://www.postman.com/explore
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en;q=0.9
Priority: u=4, i
```



```
{"meta":{"model":"workspace","action":"findOne"},"data":{"id":"a46388a8-9c78-415e-bc71-4a8a2006ad37","name":"New Api create
2","description":null,"summary":"","createdBy":"21623706","updatedBy":"21623706","team":"4459478","createdAt":"2023-09-08T06:08:36.000Z","updatedAt":"2023-09-
tatus":"public","elements":{"collections":["21623706-47cbb1e7-e817-412b-8fda-ba445335f2d8","21623706-1f488b02-f877-41c2-9fc8-d0083baab038","21623706-026867eb-
f7ee-4b10-9f8c-337fdd5fd3e8","21623706-2972adaf-dd13-4db3-98fa-7ee3d95dd993","21623706-bae32253-c5c6-469a-bd93-c0762852a03c","21623706-dfb20a1b-83ae-477c-a6a1
e12a4dd5-6866-4e8e-ad1e-65e28ef21aa9","21623706-cbe2f05a-9d82-4133-9807-c73f307a64be","21623706-ea31500d-5b10-4cc3-8c1d-53e8e7f4f45e","21623706-b39a13e4-8484-
db381f8c440f","21623706-b14178e4-896e-47d8-8e13-0ab06f002c5c","21623706-a6e2875d-1989-4a13-9383-df0ca864ed5f","21623706-16ae186c-b413-43d4-b137-d593860be993",
d4b7f3b1-45ac-431e-8e13-851bf721eba8","21623706-fcce691a-461f-49c1-9df6-92d9bfca499e","21623706-9826ecab-824e-4eba-a264-7194e4fa73d0","21623706-f2ad0a5c-980b-
bd89-7df265a92520","21623706-8e61bdea-dd17-43fd-987b-843eb25f968a","21623706-6e873eba-b0ce-4afc-9f8f-398cb1485f77","21623706-d5d8d640-cf32-4485-
ae24-3265f190f1df","21623706-26dbd19f-0c80-4ab6-9f1d-bdd1a6a105b2","21623706-5bed6855-92a4-4ee5-a996-b454775e129d","21623706-5abc094b-b5d5-4c83-
b644-11a7dca2cc2a","21623706-767a734c-3991-4d74-802a-1ec823db60a6","21623706-83695168-e868-4132-a2c6-23064d49d8d0","21623706-d4526c40-8ef7-4c20-85a1-354bc1e2e
ad520ff1-2096-4da6-b52d-2edfa4540b7c"],"globals":["7acddf47-6eb1-4f01-827c-6c15eefad2c4"]},"profileInfo":{"slug":"new-api-
create-2","profileType":"team","profileId":"4459478","publicHandle":"https://www.postman.com/blue-
moon-733059","publicImageURL":"https://res.cloudinary.com/postman/image/upload/t_team_logo/v1/team/default-B10","publicName":"blue-moon-733059","isVerified":f
{"meta":{"model":"workspace","action":"findOne"},"data":{"id":"6067c167-d7a1-4793-b5b4-d7cf824bb26c","name":"CVP_Automation_practice","description":"## 👋 In
workspace is designed to provide a collaborative environment for developers and testers to thoroughly test APIs before they are released to production.\n\nThi
different collections that can help you perform various tests for your APIs.\n\n## 🚀 Getting started with this workspace\n\nYou can either send [individual
(https://learning.postman.com/docs/sending-requests/requests/) in these collections or use the [Postman collection runner](https://learning.postman.com/docs/c
collections/intro-to-collection-runs/) to run all the requests in the collection manually. If needed, you can also [schedule and automate]
(https://learning.postman.com/docs/collections/running-collections/running-collections-overview/) your collection runs to test your APIs continuously.\n\nWe&#
collections to illustrate how you can build a test suite for various use cases like **contract, integration and performance** **testing**.\n\nWe&#39;ve also i
Tests** collection to help you start writing tests. We hope this workspace enables you to detect API issues early.\n\n## 🐞 Reporting issues\n\nIf you find AI
testing, please create a ticket and assign it to a member of the development team.\n\n## 🛟 Help and support\n\nIf you have any questions or suggestions, ple
reach out to your manager or any member of the Quality Engineering team.","summary":"This workspace helps our team maintain a high quality bar for our
APIs.","createdBy":"22509723","updatedBy":"22509723","team":null,"createdAt":"2023-09-20T06:21:34.000Z","updatedAt":"2023-09-26T05:52:53.000Z","visibilityStat
ections":["22509723-60cf6592-745b-4a1d-bbb0-797975c68953","22509723-a43b8436-676e-45e4-b6fb-ca068f808b6a","22509723-0ffa3029-bb9e-44ae-9c70-6c84a9039fc4","225
a1e4-1e877b059f12","22509723-b2157cee-6ae0-42a7-94f3-3927e09899bc","22509723-660f848c-511d-4894-abd2-6fa65ab977f2","22509723-6a9cd30a-43ab-4a39-a242-
bdc9bcf7ba15","22509723-8bab39a3-0f3c-4baa-b971-b4adfc86675f"],"globals":["1f53951e-897d-4ba6-84f4-5983072bc6e6"]},"profileInfo":{"slug":"cvp-automation-
practice","profileType":"user","profileId":"22509723","publicHandle":"https://www.postman.com/security-
```

CHALLENGES FACED AND CONCLUSION :

In the pursuit of efficient API management and documentation retrieval from the Postman platform, the project entailed a detailed process to discover, extract, and organize API documentation effectively. The initial phase involved identifying the structure of Postman's API documentation URLs, which follow a consistent pattern, allowing for the systematic location of documentation resources. However, challenges arose due to Postman's use of dynamic content loading, which required a more sophisticated approach than simple HTTP requests.

Selenium, a tool for browser automation, was implemented to overcome the hurdle of dynamically loaded content. While effective, this method introduced drawbacks, including extended processing times due to the sheer volume of pages (upwards of 34,000) and the occasional omission of APIs with comprehensive documentation. The extraction process yielded collection IDs unique to each API, which were then exported in JSON format using Postman's API. Constraints such as API call limits necessitated the creation of alternative solutions to circumvent such limitations.

Organizing the voluminous data into a manageable format was the next step. Initially, data was compiled into a text file, and subsequently, specific details were extracted and placed into a CSV file for clearer comprehension. The approach was not without its challenges; the data often contained irrelevant or redundant information, such as duplicate APIs, numerous forks, and unfinished API entries. Additionally, contact URLs often lacked valuable information, leading to a collection of incomplete data points.

To address these issues, a multi-pronged solution was devised: cleaning the data to remove duplicates and incomplete entries, refining the extraction scripts to better identify and skip over redundant or non-valuable data, and implementing a relational database to enhance data management and accessibility. This database design incorporated tables for raw data and metadata, streamlining the organization and retrieval of information.

The final stage involved scripting to automate the collection of environment IDs from workspaces, a crucial step for setting up the correct environments for API interaction. The Python script, as evidenced by the screenshot of the HTTP POST request, was crafted to interact with Postman's API, adhering to the required parameters for successful data retrieval.

In conclusion, the project tackled the intricate task of API documentation discovery and collection from Postman. By overcoming challenges such as the handling of dynamic content and the filtration of valuable data from a plethora of sources, the project developed robust methodologies for extracting, organizing, and utilizing API documentation efficiently. These efforts culminated in a streamlined workflow that catered to the diverse needs of developers working across various stages of the API development lifecycle.