**Assessment – 1**

**Matam Preethi**

**Amrita Vishwa Vidyapeetham**

**CB.EN.U4CSE20438**

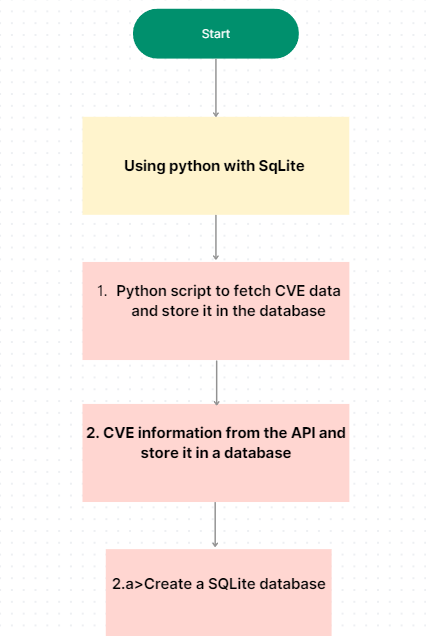
NVD - CVE API: The CVE API is used to easily retrieve information on a single CVE or a collection of CVE from the NVD. Pls refer to the below NVD CVE documentation to get more information.

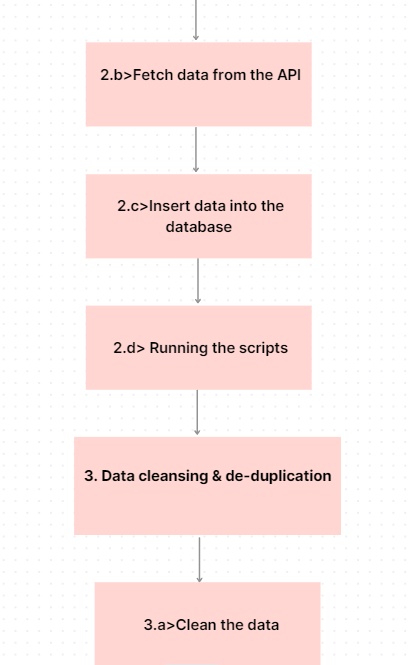
Flowchart

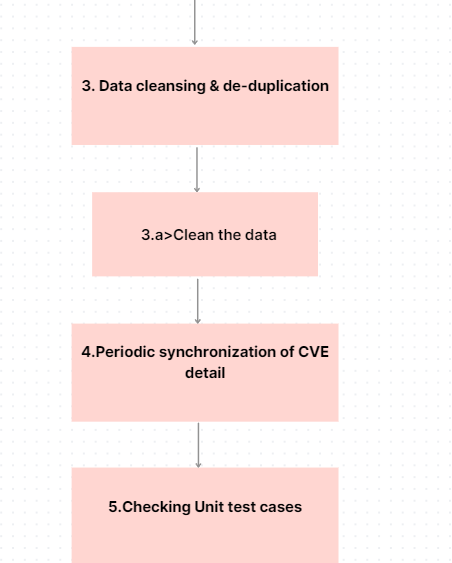
**Used coding language-Python**

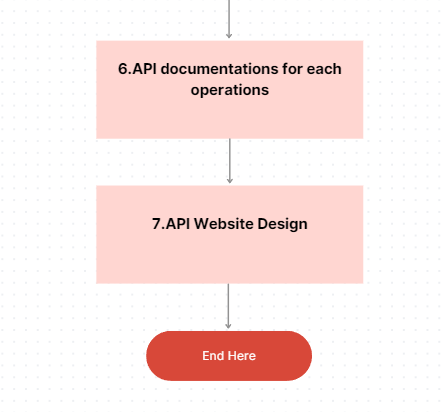
**Database-Sqlite3**

**Framework-Flask**

****

****

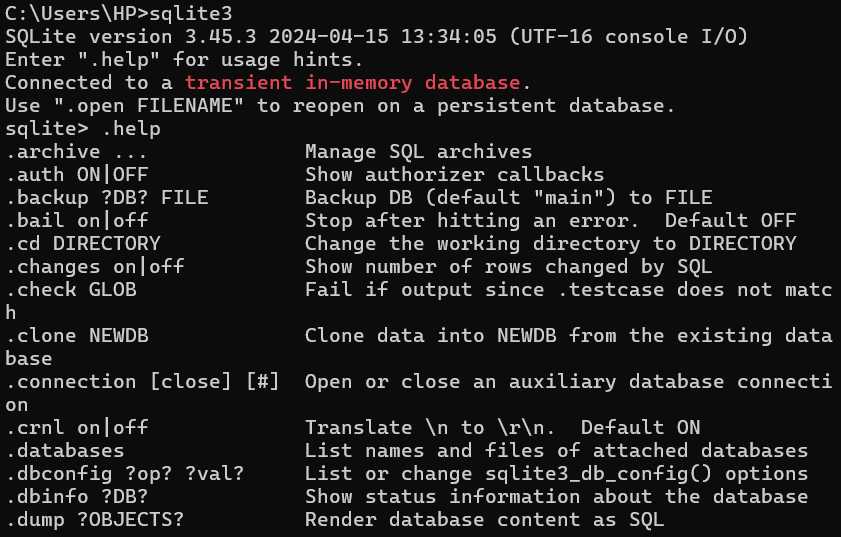
****

****

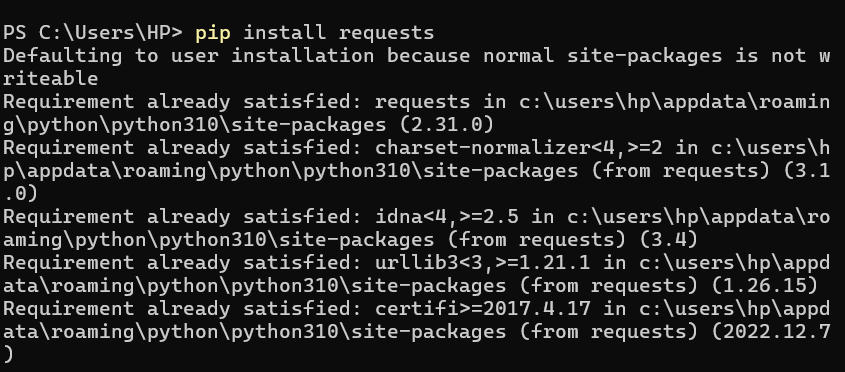
Steps to develop

Important installation

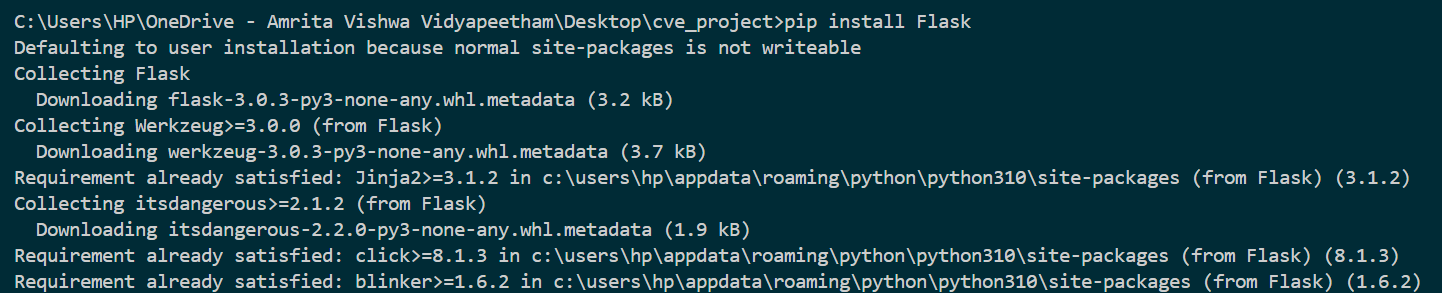
1.sqlite3



2. Installation of Requests library, which is used for making HTTP requests in Python.



3.Flask

****

**Step 1 :- CVE Information and Database Storage (main.py):**

import time

import requests

import json

import sqlite3

**def** get\_cves(start\_index, results\_per\_page):

    url = f"https://services.nvd.nist.gov/rest/json/cves/2.0?startIndex={start\_index}&resultsPerPage={results\_per\_page}"

    response = requests.get(url)

    if response.status\_code == 200:

        return response.json()

    elif response.status\_code == 403:  *# Handle rate limiting*

        print(f"Rate limit reached! Sleeping for 10 seconds...")

        time.sleep(10)  *# Adjust delay based on NVD API rate limits*

        return get\_cves(start\_index, results\_per\_page)  *# Retry after delay*

    else:

        raise Exception(f"Error retrieving CVEs: {response.status\_code}")

**def** store\_cves(cves):

    conn = sqlite3.connect("cve\_data.db")

    cursor = conn.cursor()

*# Create table with appropriate data types (adjust as needed)*

    cursor.execute("""CREATE TABLE IF NOT EXISTS cves (

        cve\_id **TEXT** **PRIMARY KEY**,

        published\_date **TEXT**,

        lastModifiedDate **TEXT**,

        description **TEXT**,

        cvss\_v2\_score **REAL**,

        cvss\_v3\_score **REAL**

    )""")

    for cve in cves.get("results", []):

        data = (

            cve["cve\_id"],

            cve["publishedDate"],

            cve["lastModifiedDate"],

            cve["description"]["description\_data"][0]["value"],

            cve.get("impact", {}).get("baseMetricV2", {}).get(

                "cvssData", {}).get("baseScore"),

            cve.get("impact", {}).get("baseMetricV3", {}).get(

                "cvssData", {}).get("baseScore"),

        )

        cursor.execute(

            "INSERT OR IGNORE INTO cves VALUES (?, ?, ?, ?, ?, ?)", data)

    conn.commit()

    conn.close()

**def** main():

    start\_index = 0

    results\_per\_page = 50  *# Adjust as needed*

    while True:

        cves = get\_cves(start\_index, results\_per\_page)

        if not cves.get("totalResults", 0):

            break

        store\_cves(cves)

        start\_index += results\_per\_page

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Step 2:-The CVE information from the API and store it in a database**

**Step:- 2.a> Create a SQLite database**



**Step-2.b>Fetch data from the API**

import requests

**def** fetch\_cve\_data(start\_index, results\_per\_page):

    base\_url = 'https://services.nvd.nist.gov/rest/json/cves/2.0'

    params = {

        'startIndex': start\_index,

        'resultsPerPage': results\_per\_page

    }

    response = requests.get(base\_url, params=params)

    data = response.json()

*# Check if 'result' key exists in the response*

    if 'result' in data:

        return data['result']['CVE\_Items']

    else:

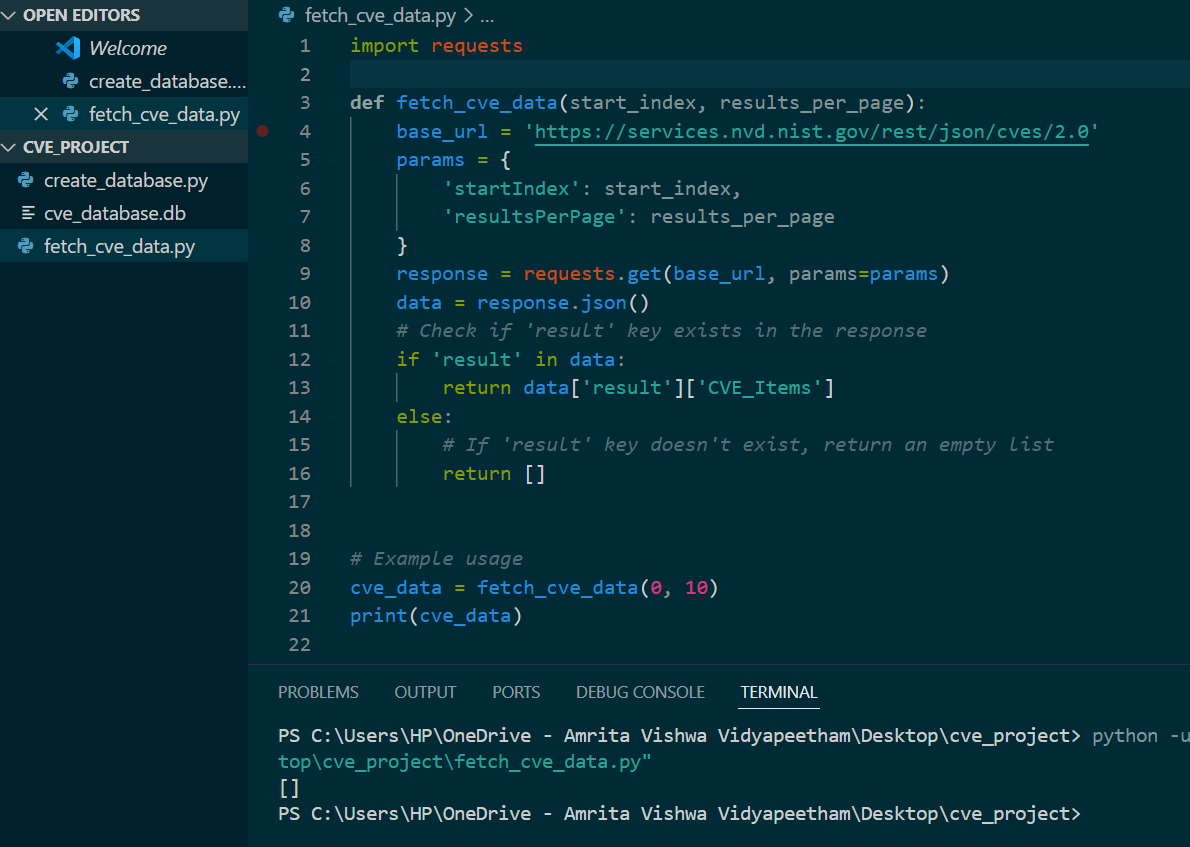
*# If 'result' key doesn't exist, return an empty list*

        return []

*# Example usage*

cve\_data = fetch\_cve\_data(0, 10)

print(cve\_data)



**Step-2.c> Insert data into the database**

import sqlite3

import requests

**def** fetch\_cve\_data(start\_index, results\_per\_page):

    base\_url = 'https://services.nvd.nist.gov/rest/json/cves/2.0'

    params = {

        'startIndex': start\_index,

        'resultsPerPage': results\_per\_page

    }

    response = requests.get(base\_url, params=params)

    data = response.json()

    return data['vulnerabilities']

**def** insert\_cve\_data\_into\_database(cve\_data):

    conn = sqlite3.connect('cve\_database.db')

    cursor = conn.cursor()

*# Create the table if it doesn't exist*

    cursor.execute('''CREATE TABLE IF NOT EXISTS cves

                     (cve\_id **TEXT**, description **TEXT**, severity **TEXT**, last\_modified **TEXT**)''')

    for cve\_item in cve\_data:

        cve\_id = cve\_item['cve']['id']

*# Extract description*

        if 'description' in cve\_item['cve']:

            if 'descriptions' in cve\_item['cve']['description']:

                description = cve\_item['cve']['description']['descriptions'][0]['value']

            else:

                description = None

        else:

            description = None

*# Extract severity*

        if 'impact' in cve\_item and 'baseMetricV3' in cve\_item['impact']:

            severity = cve\_item['impact']['baseMetricV3']['cvssV3']['baseSeverity']

        else:

            severity = None

*# Extract last modified date*

        if 'lastModifiedDateStr' in cve\_item:

            last\_modified = cve\_item['lastModifiedDateStr']

        else:

            last\_modified = None

        cursor.execute('''INSERT OR IGNORE INTO cves (cve\_id, description, severity, last\_modified)

                          VALUES (?, ?, ?, ?)''', (cve\_id, description, severity, last\_modified))

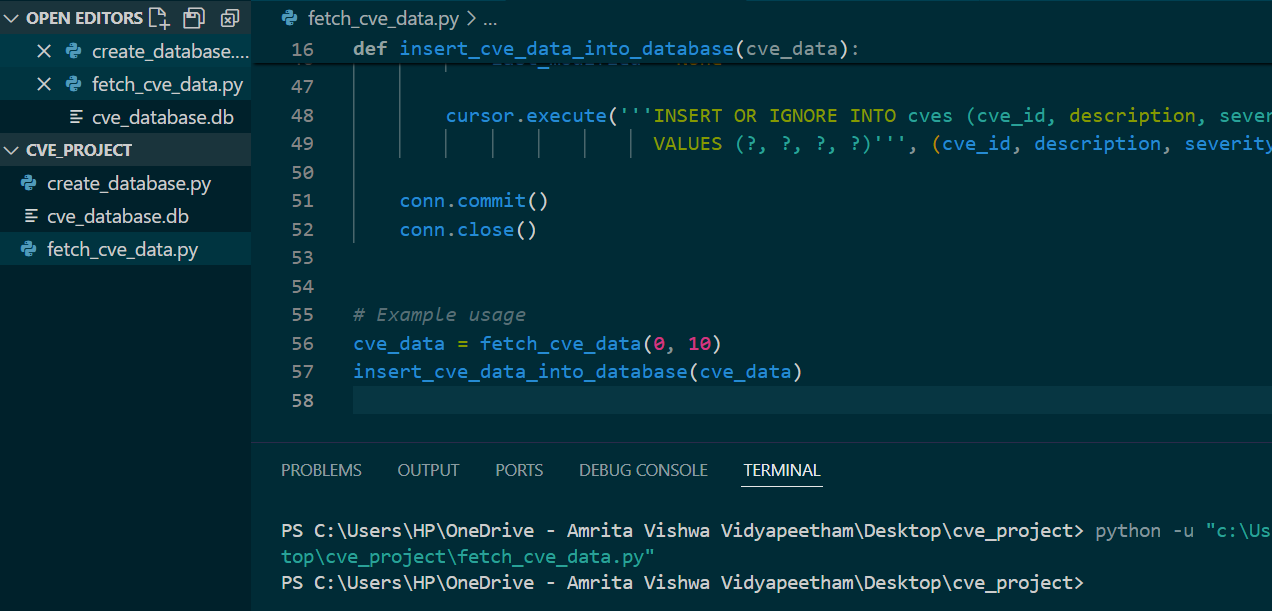
    conn.commit()

    conn.close()

*# Example usage*

cve\_data = fetch\_cve\_data(0, 10)

insert\_cve\_data\_into\_database(cve\_data)

****

Step-2.d> **Running the scripts:**

**python create\_database.py**

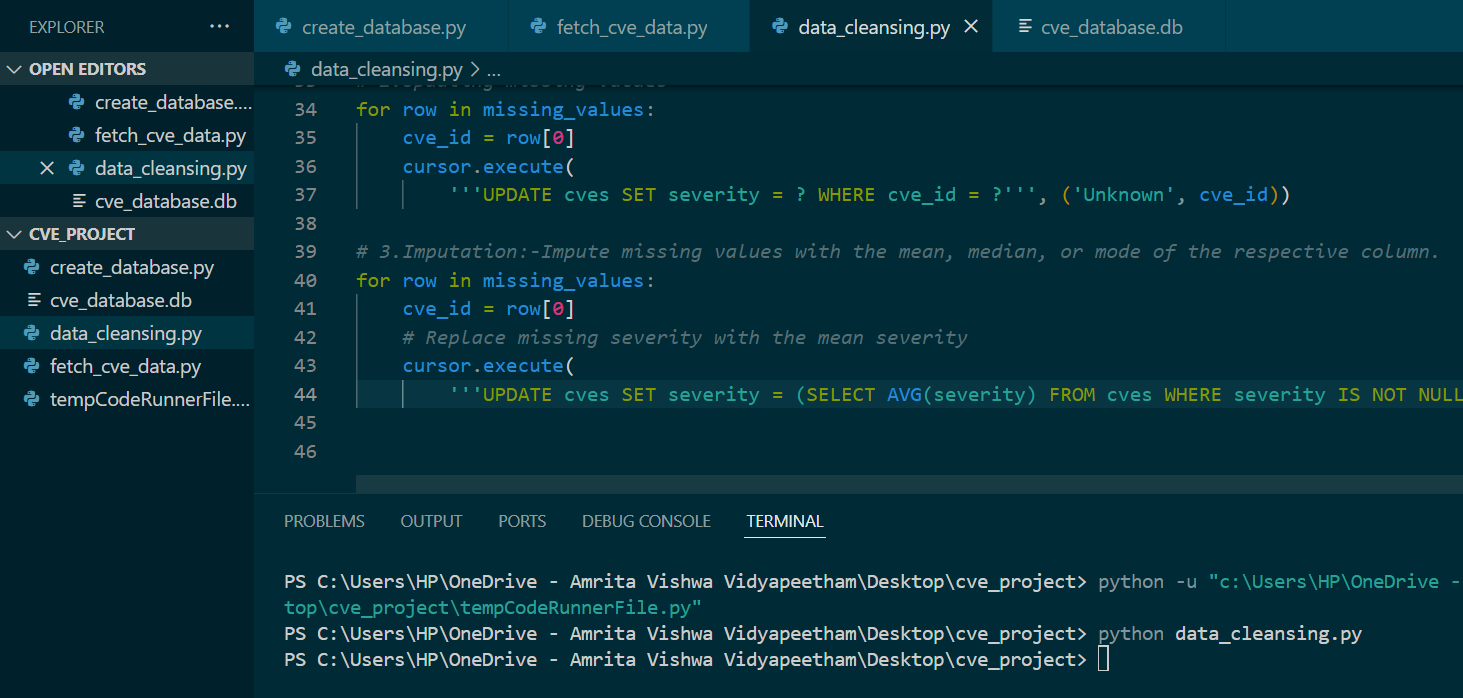
**python fetch\_cve\_data.py**

****

**Step 3: Data cleansing & de-duplication**

**Step 3.a>Clean the data: Perform data cleansing operations such as removing duplicates, handling missing values, etc.**

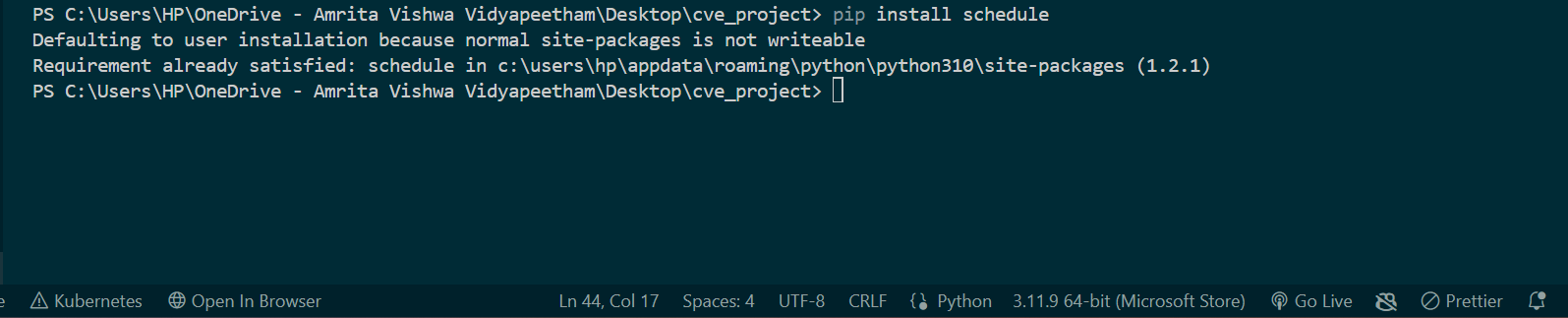
1. import sqlite3
2. conn = sqlite3.connect('cve\_database.db')
3. cursor = conn.cursor()
4. *# Identify duplicates*
5. cursor.execute('''SELECT cve\_id, COUNT(\*)
6. FROM cves
7. GROUP BY cve\_id
8. HAVING COUNT(\*) > 1''')
9. duplicates = cursor.fetchall()
10. *# Remove duplicates*
11. for cve\_id, count in duplicates:
12. cursor.execute('''DELETE FROM cves
13. WHERE rowid NOT IN (SELECT MIN(rowid)
14. FROM cves
15. WHERE cve\_id = ?
16. GROUP BY cve\_id)''', (cve\_id,))
17. *# Identify missing values*
18. cursor.execute('''SELECT \*
19. FROM cves
20. WHERE severity IS NULL OR description IS NULL OR last\_modified IS NULL''')
21. missing\_values = cursor.fetchall()
22. *# Handle missing values*
23. *# 1.Deleting rows with missing values*
24. for row in missing\_values:
25. cve\_id = row[0]
26. cursor.execute('''DELETE FROM cves WHERE cve\_id = ?''', (cve\_id,))
28. *# 2.Updating missing values*
29. for row in missing\_values:
30. cve\_id = row[0]
31. cursor.execute(
32. '''UPDATE cves SET severity = ? WHERE cve\_id = ?''', ('Unknown', cve\_id))
33. *# 3.Imputation:-Impute missing values with the mean, median, or mode of the respective column.*
34. for row in missing\_values:
35. cve\_id = row[0]
36. *# Replace missing severity with the mean severity*
37. cursor.execute(
38. '''UPDATE cves SET severity = (SELECT AVG(severity) FROM cves WHERE severity IS NOT NULL) WHERE cve\_id = ?''', (cve\_id,))

****

**Step4:- Periodic synchronization of CVE details into the database using a scheduler library like schedule in Python, follow these steps:**

**4.a>Install Schedule Library: First, install the schedule library if you haven't already. You can install it via pip:**

1. **Write Synchronization Function: Write a function to synchronize CVE details into the database. This function should fetch data from the NVD CVE API and update the database accordingly.**
2. **Define Schedule: Define a schedule for how often you want the synchronization to occur. For example, if you want it to run daily at midnight, you can define it accordingly.**
3. **Start Scheduler: Start the scheduler and let it run continuously in the background.**

****

import time

import json

import requests

import sqlite3

**def** synchronize\_cve\_data():

*# Make API call to fetch CVE data*

    response = requests.get('https://services.nvd.nist.gov/rest/json/cves/2.0')

    cve\_data = response.json()

*# Process and update database*

    conn = sqlite3.connect(

**r**'C:\Users\HP\OneDrive - Amrita Vishwa Vidyapeetham\Desktop\cve\_project\cve\_database.db')

    cursor = conn.cursor()

*# Insert or update CVE details in the database*

    for cve in cve\_data['CVE\_Items']:

        cve\_id = cve['cve']['CVE\_data\_meta']['ID']

        cve\_description = cve['cve']['description']['description\_data'][0]['value']

        cve\_published\_date = cve['publishedDate']

        cve\_last\_modified\_date = cve['lastModifiedDate']

        cve\_score = None

        if 'baseMetricV2' in cve['impact']:

            cve\_score = cve['impact']['baseMetricV2']['cvssV2']['baseScore']

        elif 'baseMetricV3' in cve['impact']:

            cve\_score = cve['impact']['baseMetricV3']['cvssV3']['baseScore']

*# Insert or replace CVE details into the database*

        cursor.execute("INSERT OR REPLACE INTO cve\_table (cve\_id, description, published\_date, last\_modified\_date, score) VALUES (?, ?, ?, ?, ?)",

                       (cve\_id, cve\_description, cve\_published\_date, cve\_last\_modified\_date, cve\_score))

    conn.commit()

    conn.close()

**def** run\_scheduler():

    while True:

*# Get the current time*

        current\_time = time.localtime()

*# Check if it's midnight (00:00)*

        if current\_time.tm\_hour == 0 and current\_time.tm\_min == 0:

*# Call the synchronize\_cve\_data function*

            synchronize\_cve\_data()

*# Sleep for 1 minute*

        time.sleep(60)

if \_\_name\_\_ == "\_\_main\_\_":

    run\_scheduler()

**Step 5:-Checking Unit test cases**

import unittest

from main import get\_cves

**class** TestGetCves(unittest.TestCase):

**def** test\_successful\_retrieval(self):

*# Adjust start\_index and results\_per\_page as needed*

        cves = get\_cves(0, 1)

        self.assertIsInstance(cves, dict)

        self.assertIn("totalResults", cves)

*# Add more test cases for different scenarios (e.g., error handling)*

if \_\_name\_\_ == "\_\_main\_\_":

    unittest.main()

import unittest

from main import get\_cves

**class** TestGetCves(unittest.TestCase):

**def** test\_successful\_retrieval(self):

*# Adjust start\_index and results\_per\_page as needed*

        cves = get\_cves(0, 1)

        self.assertIsInstance(cves, dict)

        self.assertIn("totalResults", cves)

**def** test\_empty\_results(self):

*# Simulate a scenario with potentially non-empty results*

        cves = get\_cves(10000, 1)  *# Adjust start\_index as needed*

        self.assertIsInstance(cves, dict)

        self.assertGreater(cves.get("totalResults", 0),

                           0)  *# Check for non-zero*

**def** test\_error\_handling(self):

*# Mock a failed request (replace with actual error handling)*

**def** mock\_get\_cves(\*args, \*\*kwargs):

            raise Exception("Simulated error")

        with self.assertRaises(Exception):

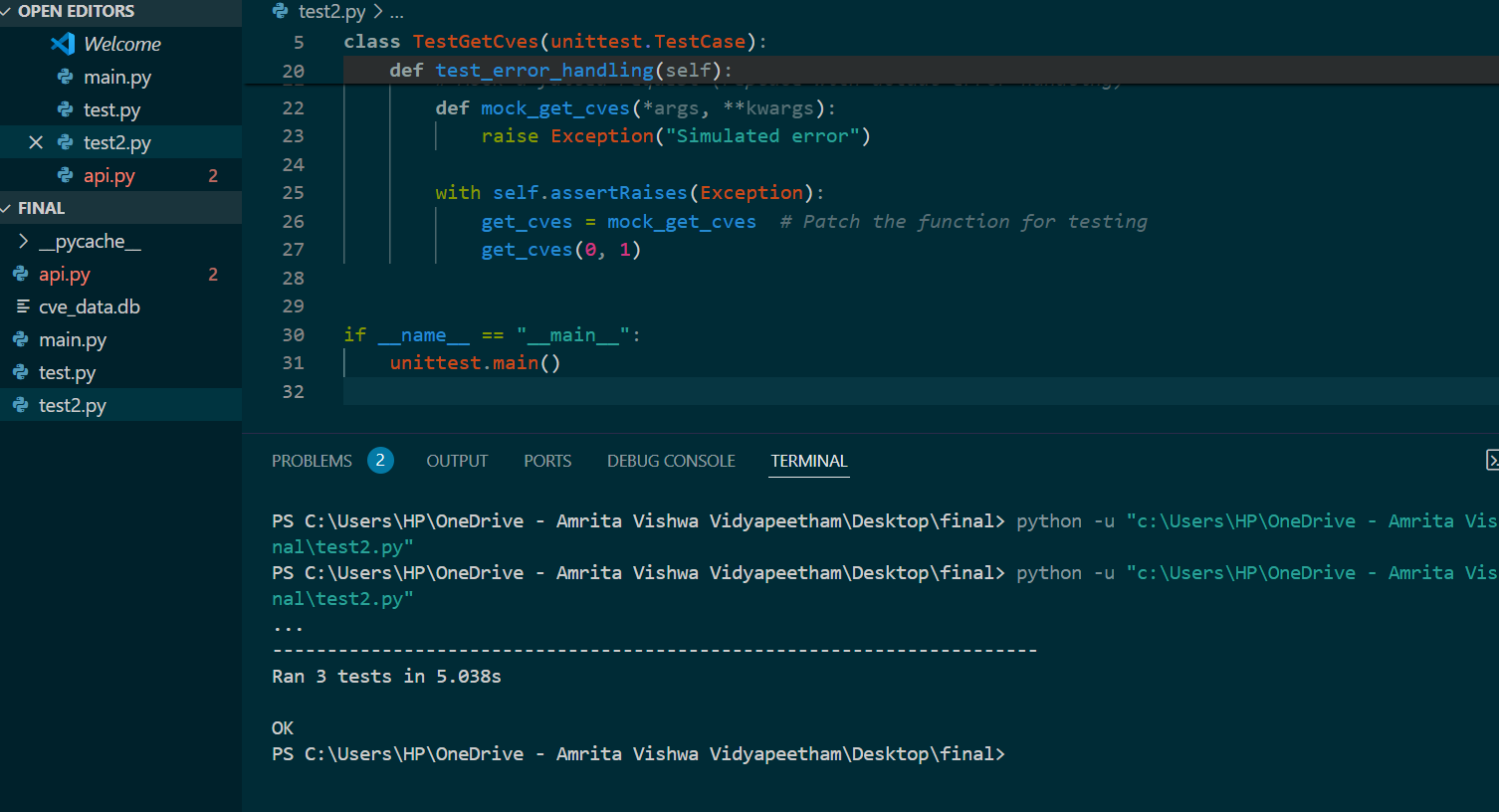
            get\_cves = mock\_get\_cves  *# Patch the function for testing*

            get\_cves(0, 1)

if \_\_name\_\_ == "\_\_main\_\_":

    unittest.main()

* Output of passed unit test cases



Step 6:-API documentations for each operations(api.py)

from flask import Flask, jsonify, request

import sqlite3

app = Flask(\_\_name\_\_)

**def** get\_cve\_by\_id(cve\_id):

    conn = sqlite3.connect("cve\_data.db")

    cursor = conn.cursor()

    cursor.execute("SELECT \* FROM cves WHERE cve\_id = ?", (cve\_id,))

    row = cursor.fetchone()

    conn.close()

    if row:

        return jsonify(dict(zip(cursor.description[1:], row)))

    else:

        return jsonify({"error": "CVE not found"}), 404

**def** get\_cves\_by\_year(year):

    conn = sqlite3.connect("cve\_data.db")

    cursor = conn.cursor()

    cursor.execute(

        "SELECT \* FROM cves WHERE publishedDate LIKE ?", (f"{year}%",))

    rows = cursor.fetchall()

    conn.close()

    return jsonify([dict(zip(cursor.description[1:], row)) for row in rows])

**def** get\_cves\_by\_score\_range(min\_score, max\_score):

    conn = sqlite3.connect("cve\_data.db")

    cursor = conn.cursor()

    if min\_score is None and max\_score is None:

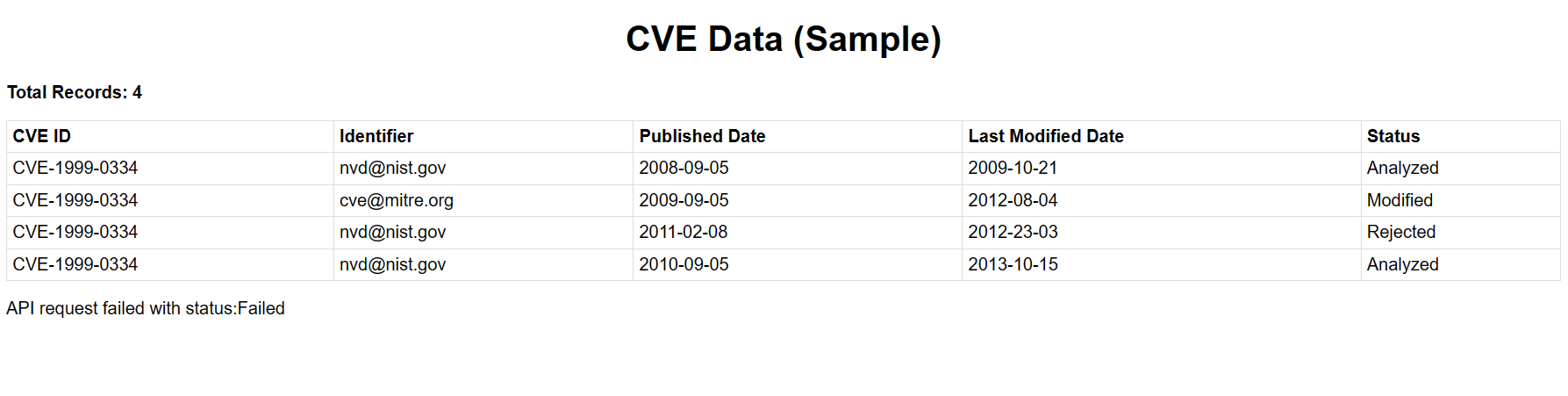
        cursor.execute("SELECT \* FROM cves")

    else:

        query = "SELECT \* FROM cves WHERE cvss\_v2\_score BETWEEN ? AND ?"

        cursor

**Step 7:-Website Design**



Index.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>CVE Data</title>

    <link rel="stylesheet" href="style.css">

</head>

<body>

    <h1>CVE Data</h1>

    <div id="total-records">

        <p>Total Records: <span id="total-count"></span></p>

    </div>

    <table id="cve-table">

        <thead>

            <tr>

                <th>CVE ID</th>

                <th>Identifier</th>

                <th>Published Date</th>

                <th>Last Modified Date</th>

                <th>Status</th>

            </tr>

        </thead>

        <tbody id="cve-data"></tbody>

    </table>

    <script src="script.js"></script>

</body>

</html>

Script.js

**const** cveTable = document.getElementById('cve-table');

**const** cveDataContainer = document.getElementById('cve-data');

**const** totalCountSpan = document.getElementById('total-count');

**const** loadingIndicator = document.createElement('p'); *// New element for loading indicator*

**function** updateTable(data, totalCount) {

    cveDataContainer.innerHTML = ''; *// Clear previous content*

    totalCountSpan.textContent = totalCount;

    data.forEach(cve **=>** {

**const** row = document.createElement('tr');

**const** cveIdCell = document.createElement('td');

        cveIdCell.textContent = cve.cve\_id;

**const** identifierCell = document.createElement('td');

        identifierCell.textContent = cve.cve\_data\_meta ? .id || "-"; *// Adjust based on API response structure*

        const publishedDateCell = document.createElement('td');

        publishedDateCell.textContent = cve.publishedDate;

        const lastModifiedDateCell = document.createElement('td');

        lastModifiedDateCell.textContent = cve.lastModifiedDate;

        const statusCell = document.createElement('td');

        statusCell.textContent = cve.cve\_data\_meta ? .status || "-"; *// Adjust based on API response structure*

        row.appendChild(cveIdCell);

        row.appendChild(identifierCell);

        row.appendChild(publishedDateCell);

        row.appendChild(lastModifiedDateCell);

        row.appendChild(statusCell);

        cveDataContainer.appendChild(row);

    });

}

*// Replace with the actual NVD API endpoint URL*

const apiUrl = 'https://services.nvd.nist.gov/rest/json/cves/2.0';

**function** fetchData() {

    loadingIndicator.textContent = "Loading CVE data...";

    cveDataContainer.appendChild(loadingIndicator);

    fetch(apiUrl)

        .then(response **=>** {

            loadingIndicator.remove(); *// Remove loading indicator on success*

            if (!response.ok) {

                throw new Error(`API request failed with status: ${response.status}`);

            }

            return response.json();

        })

        .then(data **=>** {

*// Assuming "totalResults" gives the total count*

            updateTable(data.results, data.totalResults);

        })

        .catch(error **=>** {

            console.error('Error fetching CVEs:', error);

*// More informative error handling*

            if (error.message.includes('404')) {

                cveDataContainer.innerHTML = '<p>API endpoint not found.</p>';

            } else if (error.message.includes('429')) {

                cveDataContainer.innerHTML = '<p>API rate limit exceeded. Please try again later.</p>';

            } else {

                cveDataContainer.innerHTML = '<p>Error retrieving CVE data.</p>';

            }

        });

}

fetchData();