CVE Management System API Documentation

Overview

The CVE Management System provides a REST API to interact with and manage CVEs (Common Vulnerabilities and Exposures) data. The API allows for listing, searching, and filtering CVEs, with additional details available on individual CVEs.

Base URL:

- Localhost: http://127.0.0.1:5000
- https://services.nvd.nist.gov/rest/json/cves/2.0

Authentication:

This API does not currently require authentication. All endpoints are publicly accessible.

API Rate Limits:

To ensure optimal performance and fair usage, the API implements rate limits:

- **Maximum Requests**: 100 requests per minute.
- **Exceeded Rate Limit Response**: If the rate limit is exceeded, the API will return a 429 Too Many Requests response. Clients should implement retry logic with exponential backoff to handle this scenario.

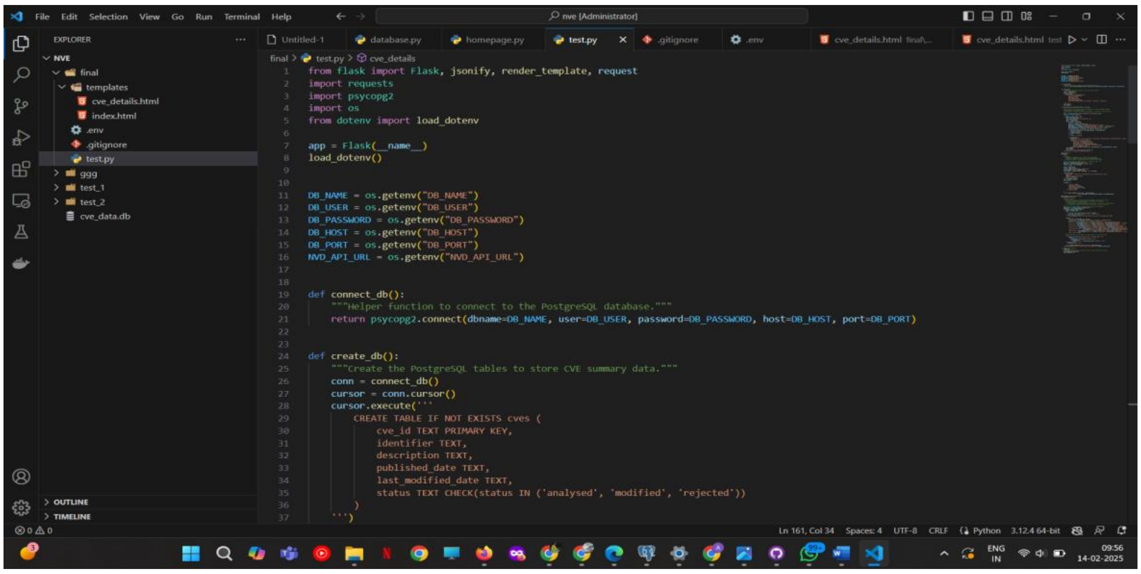
Endpoints

1. **List CVEs**

- **URL**: /cves/list
- **Method**: GET
- **Description**: Retrieves a paginated list of CVEs with options for filtering, sorting, and pagination.
- **Parameters**:
  - page (optional): Page number (default is 1).
  - resultsPerPage (optional): Number of results per page (default is 10).
  - sort (optional): Sort field (cve_id, published_date, modified_date, status).

▪ direction (optional): Sort direction (ASC, DESC).

○ **Response**: Returns a paginated list of CVEs, each including the following fields: CVE ID, description, published date, modified date, cvss_v2_score, status.



```python
from flask import Flask, jsonify, render_template, request
import requests
import psycopg2
import os
from dotenv import load_dotenv

app = Flask(__name__)
load_dotenv()


DB_NAME = os.getenv("DB_NAME")
DB_USER = os.getenv("DB_USER")
DB_PASSWORD = os.getenv("DB_PASSWORD")
DB_HOST = os.getenv("DB_HOST")
DB_PORT = os.getenv("DB_PORT")
NVD_API_URL = os.getenv("NVD_API_URL")


def connect_db():
    """Helper function to connect to the PostgreSQL database."""
    return psycopg2.connect(dbname=DB_NAME, user=DB_USER, password=DB_PASSWORD, host=DB_HOST, port=DB_PORT)


def create_db():
    """Create the PostgreSQL tables to store CVE summary data."""
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS cves (
            cve_id TEXT PRIMARY KEY,
            identifier TEXT,
            description TEXT,
            published_date TEXT,
            last_modified_date TEXT,
            status TEXT CHECK(status IN ('analysed', 'modified', 'rejected'))
        )
    ''')
```

Example:

GET /cves/list?page=1&resultsPerPage=10&sort=published_date&direction=DESC

2.Get CVE by ID

○ **URL**: /cves/<cve_id>

○ **Method**: GET

○ **Description**: Retrieves detailed information about a specific CVE by its ID.

○ **Parameters**:

▪ cve_id: The unique CVE identifier (e.g., CVE-2021-12345).

○ **Response**: Returns detailed information about the specified CVE, including CVE ID, description, published date, modified date, CVSS score, weaknesses, configurations, and reference links.

Example:

GET /cves/CVE-2021-12345

Filter CVEs by Year

○ **URL**: /api/cves/year/<int:year>

○ **Method**: GET

○ **Description**: Retrieves a list of CVEs that were published in a specific year.

○ **Parameters**:

▪ year: The year to filter CVEs by (e.g., 2020).

○ **Response**: A list of CVEs published in the specified year.

Example:

GET /api/cves/year/2020

Filter CVEs by CVSS Score

○ **URL**: /api/cves/score/<float:score>

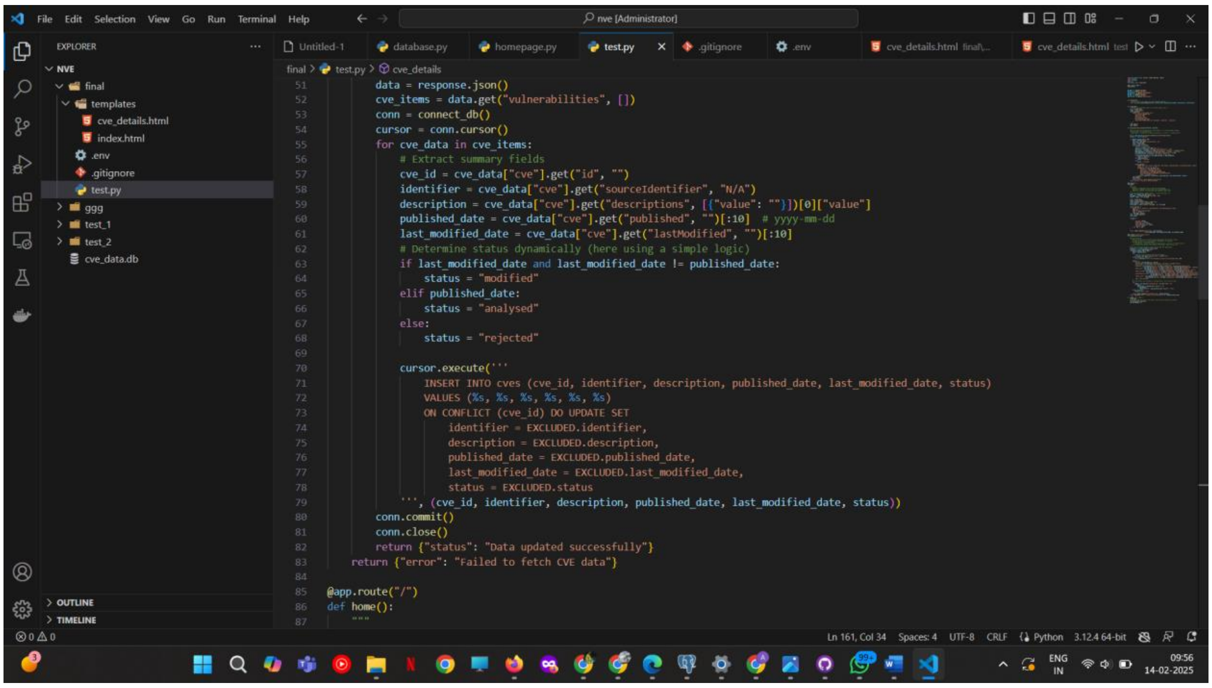
○ **Method**: GET

○ **Description**: Retrieves CVEs with a CVSS score greater than or equal to the specified score.

○ **Parameters**:

▪ score: The minimum CVSS score to filter by (e.g., 7.0).

○ **Response**: A list of CVEs with a CVSS score equal to or higher than the provided value.



```python
        data = response.json()
        cve_items = data.get("vulnerabilities", [])
        conn = connect_db()
        cursor = conn.cursor()
        for cve_data in cve_items:
            # Extract summary fields
            cve_id = cve_data["cve"].get("id", "")
            identifier = cve_data["cve"].get("sourceIdentifier", "N/A")
            description = cve_data["cve"].get("descriptions", [{"value": ""}])[0]["value"]
            published_date = cve_data["cve"].get("published", "")[:10]  # yyyy-mm-dd
            last_modified_date = cve_data["cve"].get("lastModified", "")[:10]
            # Determine status dynamically (here using a simple logic)
            if last_modified_date and last_modified_date != published_date:
                status = "modified"
            elif published_date:
                status = "analysed"
            else:
                status = "rejected"

            cursor.execute('''
                INSERT INTO cves (cve_id, identifier, description, published_date, last_modified_date, status)
                VALUES (%s, %s, %s, %s, %s, %s)
                ON CONFLICT (cve_id) DO UPDATE SET
                    identifier = EXCLUDED.identifier,
                    description = EXCLUDED.description,
                    published_date = EXCLUDED.published_date,
                    last_modified_date = EXCLUDED.last_modified_date,
                    status = EXCLUDED.status
            ''', (cve_id, identifier, description, published_date, last_modified_date, status))
        conn.commit()
        conn.close()
        return {"status": "Data updated successfully"}
    return {"error": "Failed to fetch CVE data"}

@app.route("/")
def home():
    """
```

Example:

GET /api/cves/score/7.0

Filter CVEs by Last Modified Date

○ **URL**: /api/cves/modified/<int:days>

○ **Method**: GET

○ **Description**: Retrieves CVEs that have been modified in the last n days.

○ **Parameters**:

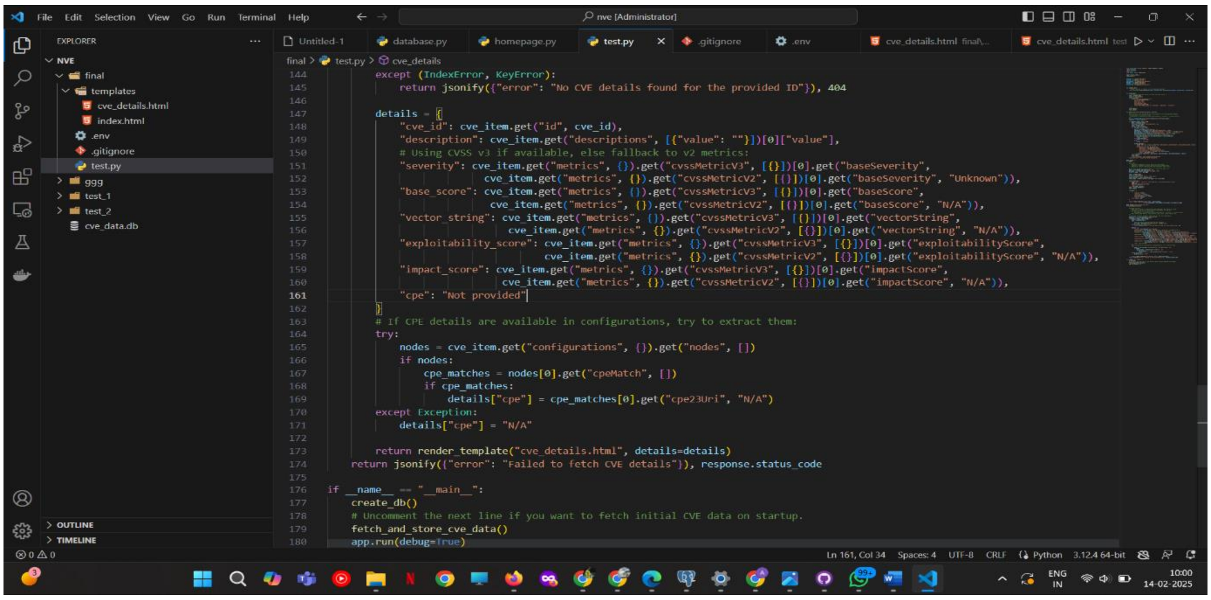
▪ days: The number of days to look back for modified CVEs.

○ **Response**: A list of CVEs modified within the last specified number of days.

Example:

GET /api/cves/modified/30

Database Table Structure: cve_details



```python
        except (IndexError, KeyError):
            return jsonify({"error": "No CVE details found for the provided ID"}), 404

        details = {
            "cve_id": cve_item.get("id", cve_id),
            "description": cve_item.get("descriptions", [{"value": ""}])[0]["value"],
            # Using CVSS v3 if available, else fallback to v2 metrics:
            "severity": cve_item.get("metrics", {}).get("cvssMetricV3", [{}])[0].get("baseSeverity",
                        cve_item.get("metrics", {}).get("cvssMetricV2", [{}])[0].get("baseSeverity", "Unknown")),
            "base_score": cve_item.get("metrics", {}).get("cvssMetricV3", [{}])[0].get("baseScore",
                          cve_item.get("metrics", {}).get("cvssMetricV2", [{}])[0].get("baseScore", "N/A")),
            "vector_string": cve_item.get("metrics", {}).get("cvssMetricV3", [{}])[0].get("vectorString",
                             cve_item.get("metrics", {}).get("cvssMetricV2", [{}])[0].get("vectorString", "N/A")),
            "exploitability_score": cve_item.get("metrics", {}).get("cvssMetricV3", [{}])[0].get("exploitabilityScore",
                                    cve_item.get("metrics", {}).get("cvssMetricV2", [{}])[0].get("exploitabilityScore", "N/A")),
            "impact_score": cve_item.get("metrics", {}).get("cvssMetricV3", [{}])[0].get("impactScore",
                            cve_item.get("metrics", {}).get("cvssMetricV2", [{}])[0].get("impactScore", "N/A")),
            "cpe": "Not provided"
        }
        # If CPE details are available in configurations, try to extract them:
        try:
            nodes = cve_item.get("configurations", {}).get("nodes", [])
            if nodes:
                cpe_matches = nodes[0].get("cpeMatch", [])
                if cpe_matches:
                    details["cpe"] = cpe_matches[0].get("cpe23Uri", "N/A")
        except Exception:
            details["cpe"] = "N/A"

        return render_template("cve_details.html", details=details)
    return jsonify({"error": "Failed to fetch CVE details"}), response.status_code

if __name__ == "__main__":
    create_db()
    # Uncomment the next line if you want to fetch initial CVE data on startup.
    fetch_and_store_cve_data()
    app.run(debug=True)
```

olumn	ata Type	escription
ve_id	ARCHAR(255)	nique identifier for the CVE (e.g., CVE-2021-12345).
escription	EXT	short summary or description of the vulnerability.
verity	ARCHAR(50)	he severity rating of the CVE (e.g., High, Medium, Low).
ase_score	LOAT	VSS base score (e.g., 7.5) indicating the severity of the CVE.
ector_string	ARCHAR(255)	ttack vector details in the CVSS format (e.g., AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H).
xploitability_score	LOAT	xploitability score indicating the likelihood of exploitation.
npact_score	LOAT	npact score showing the potential impact of exploitation.
ublished_date	ATETIME	he date the CVE was published.
odified_date	ATETIME	he date the CVE was last modified.
atus	ARCHAR(50)	atus of the CVE (e.g., new, active).

Sample API Calls

1. Get a list of CVEs

o URL: /cves/list

o Method: GET

o Parameters (optional):

▪ page: Page number (default: 1).

▪ resultsPerPage: Number of results per page (default: 10).

o Response Example:

2. {

3. "cve_id": "CVE-2021-12345",

4. "description": "A vulnerability in the XYZ service...",

5. "severity": "High",

6. "base_score": 7.5,

7. "vector_string": "AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H",

8. "exploitability_score": 3.9,

9. "impact_score": 5.9,

10. "cpe": "Not provided",

11. "published_date": "2021-05-01T00:00:00Z",

12. "modified_date": "2021-07-01T00:00:00Z",

13. "status": "new"

14. }

15. Get a specific CVE by ID

- **URL: /cves/<cve_id>**
- **Method: GET**
- **Response Example:**

16. {

17. "cve_id": "CVE-2021-12345",

18. "description": "A vulnerability in the XYZ service...",

19. "severity": "High",

20. "base_score": 7.5,

21. "vector_string": "AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H",

22. "exploitability_score": 3.9,

23. "impact_score": 5.9,

24. "cpe": "Not provided",

25. "published_date": "2021-05-01T00:00:00Z",

26. "modified_date": "2021-07-01T00:00:00Z",

27. "status": "new"

28. }

29. Filter CVEs by Year

- **URL: /api/cves/year/<int:year>**

- **Method: GET**
- **Response Example:**

30. [

31. {

32. "cve_id": "CVE-2021-12345",

33. "description": "A vulnerability in the XYZ service...",

34. "severity": "High",

35. "base_score": 7.5,

36. "published_date": "2021-05-01T00:00:00Z"

37. }

38.]

39. Filter CVEs by CVSS Score

- **URL: /api/cves/score/<float:score>**
- **Method: GET**
- **Response Example:**

40. [

41. {

42. "cve_id": "CVE-2021-12345",

43. "description": "A vulnerability in the XYZ service...",

44. "severity": "High",

45. "base_score": 7.5,

46. "vector_string": "AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H"

47. }

48.]

49. Filter CVEs by Last Modified Date

- **URL: /api/cves/modified/<int:days>**
- **Method: GET**
- **Response Example:**

50. [

```
    {
      "cve_id": "CVE-2021-12345",
      "description": "A vulnerability in the XYZ service...",
      "severity": "High",
      "modified_date": "2021-07-01T00:00:00Z"
    }
  ]
```

Response Format

All responses are in JSON format. The standard structure includes:

- CVE ID: The unique identifier for the CVE.
- Description: A short summary of the vulnerability.
- Published Date: The date the CVE was first published.
- Modified Date: The date the CVE was last updated.
- CVSS Score: The Common Vulnerability Scoring System (CVSS) severity score.
- Status: The current status of the CVE (e.g., new, active).
- Additional fields may include Weaknesses, Configurations, Reference Links.

Error Handling

Status Code	Meaning	Description
200	OK	Successful request.
400	Bad Request	Invalid parameters were provided in the request.
404	Not Found	equested resource (e.g., CVE) not found.
429	Too Many Requests	ate limit exceeded. Retry after waiting.

500	Internal Server Error	n unexpected error occurred on the server.

Caching : To improve performance, caching may be enabled for specific API responses where data changes infrequently (e.g., CVE listings). Clients should implement caching strategies where necessary to reduce server load and response times.

Technologies Used

- **Flask**: Web framework used for building the API.
- **MySQL**: Database used for storing CVE data.
- **Requests**: Python library used for fetching data from the NVD API.
- **Bootstrap**: Used for front-end design.

Conclusion

The CVE Management System API offers a robust and efficient way to access, manage, and filter data related to Common Vulnerabilities and Exposures (CVEs). With endpoints for listing CVEs, retrieving specific details, and filtering based on critical criteria like year, CVSS score, and modification date, this API allows developers and security professionals to easily integrate vulnerability data into their applications or systems.

Designed with flexibility and scalability in mind, the API supports pagination and sorting, making it capable of handling large datasets efficiently. Additionally, the use of industry-standard metrics such as CVSS ensures that users can assess the severity and exploitability of vulnerabilities at a glance.

The API’s well-structured responses, combined with a simple error-handling mechanism and reasonable rate limits, ensure that users can reliably integrate it into their workflows. Whether it's for vulnerability tracking, security assessments, or integration into larger cybersecurity systems, the CVE Management System API provides a powerful tool for managing vulnerabilities with ease and precision.

This API documentation provides comprehensive details, including rate limits, error codes, and usage examples, to interact with the CVE Management System effectively.

Reference

https://nvd.nist.gov/developers/vulnerabilities