**Project Report: NVD CVE API Integration and Visualization**

This project involved integrating with the NVD CVE API to retrieve, store, manage, and visualize CVE data. Here’s a breakdown of how we tackled the project requirements.

**Problem Statement Breakdown**

**1. Consuming CVE Information**

* **Objective:** Retrieve CVE data from the NVD CVE API and store it in a MySQL database.
* **Approach:** Use Python to call the API, handle pagination with **startIndex** and **resultsPerPage** parameters, and insert the data into MySQL.

**2. Data Cleansing and De-duplication**

* **Objective:** Ensure the data is clean.
* **Approach:** Implement logic in Python to check for null values before inserting into the database. Use unique constraints on primary keys in MySQL to prevent duplicate entries.

**3. Periodic Synchronization**

* **Objective:** Regularly update the database with new or modified CVE data.
* **Approach:** Schedule the Python script to run at specific intervals (e.g., daily). Use the **lastModifiedDate** field to fetch only updated records for incremental updates.

**4. Developing APIs**

* **Objective:** Create APIs to filter and read CVE details based on specific parameters.
* **Approach:** Use Node.js to develop APIs that connect to the MySQL database. Implement endpoints to:
  + Retrieve CVE details by CVE ID.
  + Filter CVEs by year.
  + Filter CVEs by CVSS score.
  + Get CVEs modified in the last N days.

**5. Visualization**

* **Objective:** Display CVE data in a user-friendly interface.
* **Approach:** Create a web page using HTML, CSS, and JavaScript. Use AJAX to call the Node.js APIs and populate the data in an HTML table. Display a "Total Records" count at the top of the table.

**6. API Documentation**

* **Objective:** Document each API endpoint for clarity and usability.
* **Approach:** Create documentation detailing each API endpoint, including request methods, parameters, and example responses. Ensure the documentation is clear and comprehensive.

**7. Unit Testing**

* **Objective:** Ensure the functionality is robust and error-free.
* **Approach:** Write unit tests for the Python data ingestion script, the Node.js APIs, and the JavaScript functions. Use testing frameworks like unittest for Python and Mocha for Node.js.

**8. Code Quality and Security**

* **Objective:** Maintain high code quality and security standards.
* **Approach:** Follow best practices for coding standards, including proper error handling, code reviews, and using secure coding practices to protect against vulnerabilities.

**Implementation Details**

**Database Setup**

* **MySQL Database:** Chose MySQL for its efficiency with large datasets.
* **Schema Design:** Created a table **cve\_list** with fields for **cve\_id**, **description**, **published\_date**, **last\_modified\_date**, **cvss\_v2\_base\_score**, **cvss\_v3\_base\_score**, etc.

**Data Ingestion**

* **Python Script:** Developed a script to call the NVD CVE API and manage pagination. Inserted the data into the MySQL database, checking for duplicates to maintain data integrity.

**Periodic Synchronization**

* **Scheduled Task:** Set up a cron job (or similar scheduler) to run the Python script at regular intervals to keep the database updated.

**API Development**

* **Node.js APIs:** Built APIs to access and filter CVE data. Endpoints include fetching by CVE ID, filtering by year, CVSS score, and recently modified records.

**Visualization**

* **Frontend Development:** Created an HTML page with a table to display CVE data. Used CSS for styling and JavaScript for AJAX calls to the Node.js APIs. Displayed a total record count.

**API Documentation**

* **Detailed Docs:** Prepared documentation for each API endpoint, specifying the request methods, parameters, and example responses.

**Unit Testing**

* **Comprehensive Tests:** Wrote unit tests for the data ingestion script, APIs, and frontend functionality to ensure reliability and correctness.

**Code Quality and Security**

* **Best Practices:** Followed coding standards and secure coding practices, conducted code reviews, and implemented proper error handling to ensure the application is robust and secure.

This project involved a lot of moving parts, but by breaking it down into manageable steps and following best practices, we were able to create a robust and user-friendly solution for managing and visualizing CVE data.

**Output**

Now lets get into the Output Screenshots

Database and Table Creation:

A screenshot of a computer

Description automatically generated

Now that the table is created lets populate it using python where we remove any input with null values and the table is created in such a way that it only takes in unique ways by adding the primary key and unique prompt to the cveID.

A screenshot of a computer

Description automatically generated

This ensures that any data with any missing values don’t enter the database and duplicate values are also checked

A black background with colorful text

Description automatically generated

Error Handling is done

The populated database:

A screenshot of a computer

Description automatically generated

The data is stored in the database created in MySQL.

API endpoints:

A computer code on a black background

Description automatically generated

API endpoints for page1 and page2 where the default is set to page1

A black screen with text

Description automatically generated

This API endpoint helps fetch CVE data

A black screen with colorful text

Description automatically generated

API endpoint to fetch CVE data for Page 2

Now that the data is in the database we can populate the table as asked in the first ui page

A list of cve list

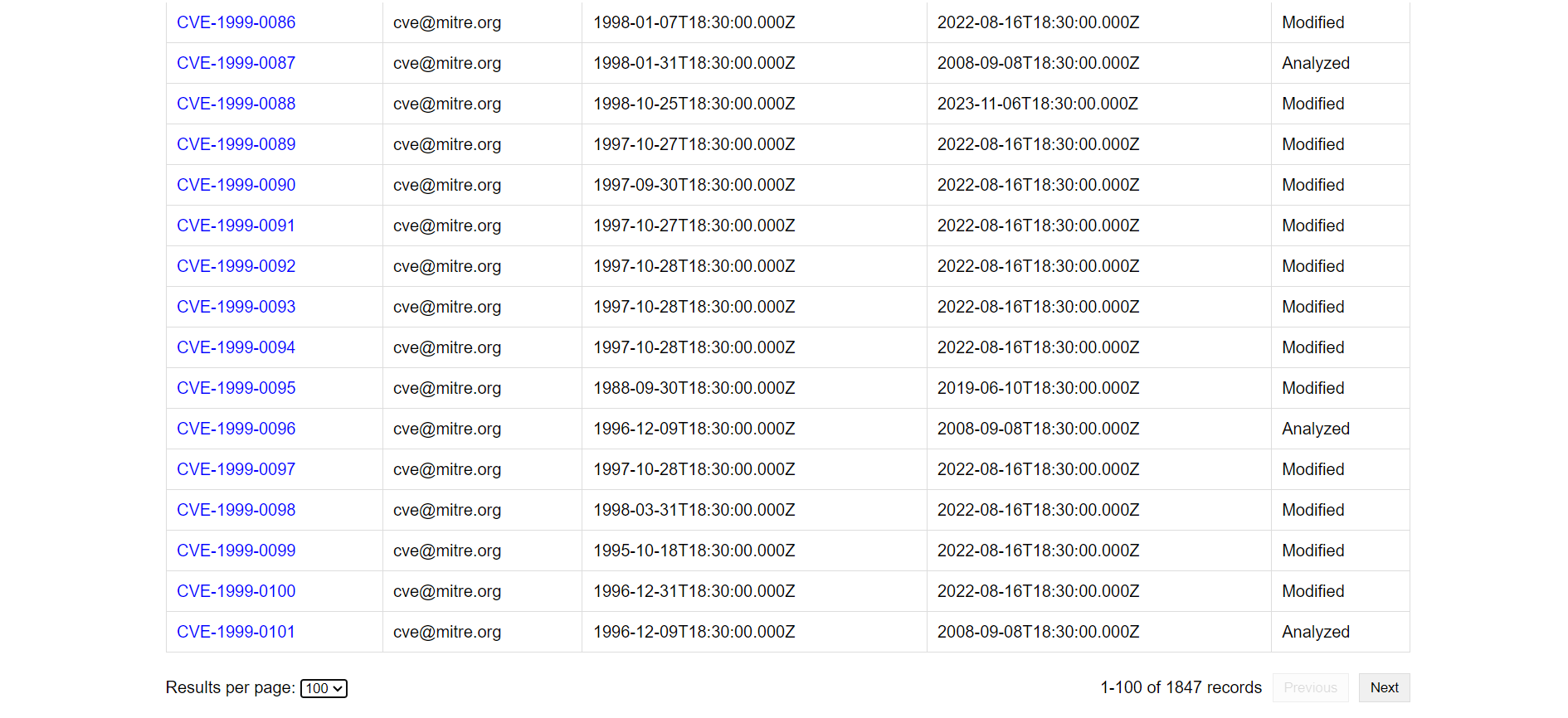
Description automatically generated with medium confidence

This shows the result when the results per page is set to 10.

A white background with black and white text

Description automatically generated

This when it is set to 50.



And this when it is set to 100.

The next output shows when it is in a different page

A close-up of a list

Description automatically generated

We can see that it shows 31-40.

A black screen with white text

Description automatically generated

Now on clicking any cve ID on the first UI Page it shows the details of that particular cveID

A screenshot of a computer

Description automatically generated

Another example:

A screenshot of a computer

Description automatically generated

This gives all the requested details printed.

To address the project requirements, I first focused on understanding the data structure and the specific fields we needed to extract from the NVD CVE API. Using Python for data ingestion, I designed a script to handle API calls, pagination, and data extraction. I chose MySQL as the database due to its reliability and support for handling large datasets efficiently. The schema design included a unique primary key (cve\_id) to prevent duplicates. To ensure data integrity, I implemented exception handling in the script to skip duplicate entries, leveraging MySQL's IntegrityError for this purpose. The backend APIs were developed using Node.js to provide easy access to the data, while the frontend was created using HTML, CSS, and JavaScript for a user-friendly interface. This approach ensured a seamless integration of data retrieval, storage, and visualization.