SECURIN ASSESSMENT

**PROBLEM STATEMENT**

Keep track of the latest Common Vulnerabilities and Exposures (CVEs) listed on the National Vulnerability Database (NVD) using their API available at <https://nvd.nist.gov/developers/vulnerabilities> And offer a user-friendly interface to easily read and visualize the data related to these CVEs.

**OVERVIEW**

* The Project utilises Express.js as Web Server.
* Local MongoDB Container as Database with mongoose for connectivity.
* EJS for rendering HTML template to visualise the user interface.
* JEST for implementing Unit tests.

**FUNCTIONALITIES IMPLEMENTED**

1. The CVE data retrieval from the NVD API batchwise using **startIndex** and **resultsPerPage** as query parameters and stored in Mongo DB
2. Periodically update CVE data from CVE HISTORY API to my local Mongo DB every 30 minutes.
3. Remove the **REJECTED** vulnerabilities from the Database
4. Present the CVE data in a tabular format.
5. Limit the number of rows displayed in table.(10, 50 ,100).
6. Pagination on the server side as well on client side is implemented to present the split the records into different pages.
7. Search Filtration of record using Multiple Parameters

- Year

- CVE id

- Last Modified

- Base Score

1. Each of the row is linked to detailed description of the particular CVE
2. Total Records in the database and range of records is displayed
3. Sorting of dates in server side is implemented
4. Unit tests for the API endpoints are implemented using JEST

**API ENDPOINTS**

ENDPOINT **:** http://<domain>:<port>/

**/ & /Cve/lists**

* Lists all the CVE Data from the database

QUERY PARAMETERS

* ?page=

-returns the contents of page requested

* ?limit=

-limits the number of records

-options:10 , 50 ,100

* ?year=

-returns all the CVEs in that particular year

* ?lastmodified=

-returns a sorted records of last modified data in descending order and limits is set to number of days given i.e N days .

* ?lt=

-returns Base score value of CVE which is lesser than input given

* ?gt=

-returns Base score value of CVE which is greater than input given

* ?id=

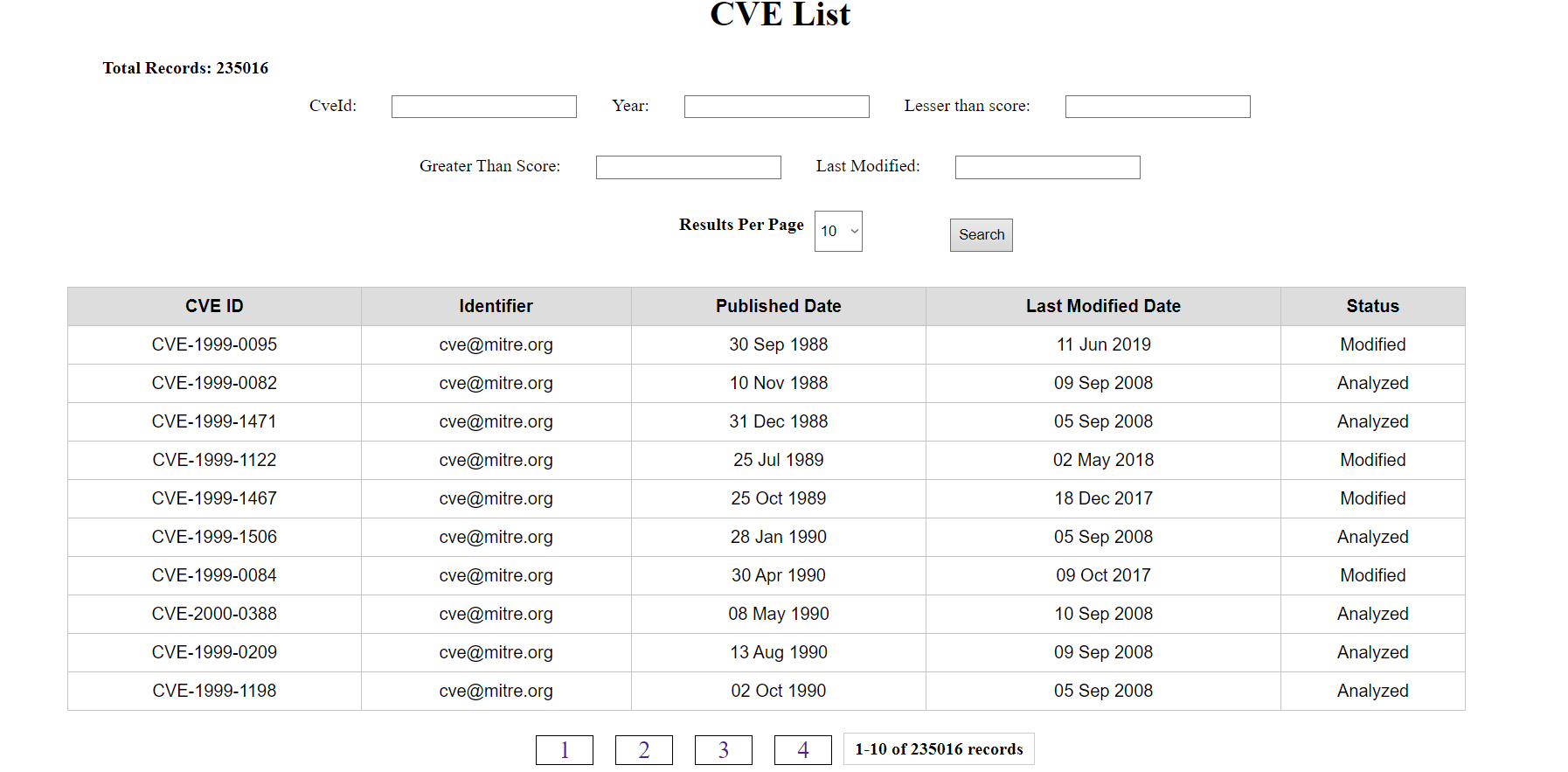
-returns the records of CVE with ids with matching with input given partially or fully.

**/cves/<CVE-ID>**

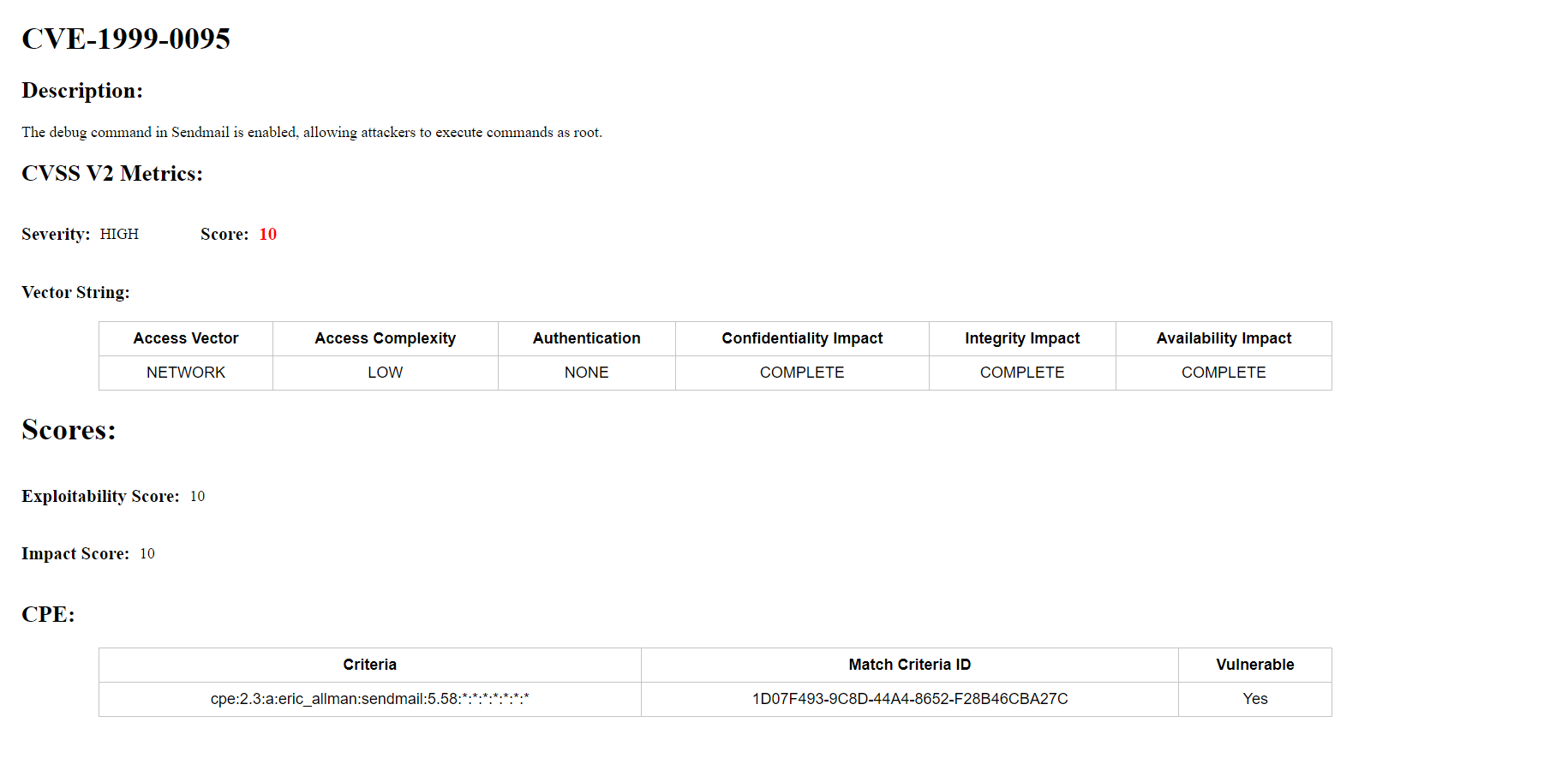
* Return the detailed description of the particular CVE matching with the following CVE-ID.

**OUTPUT**

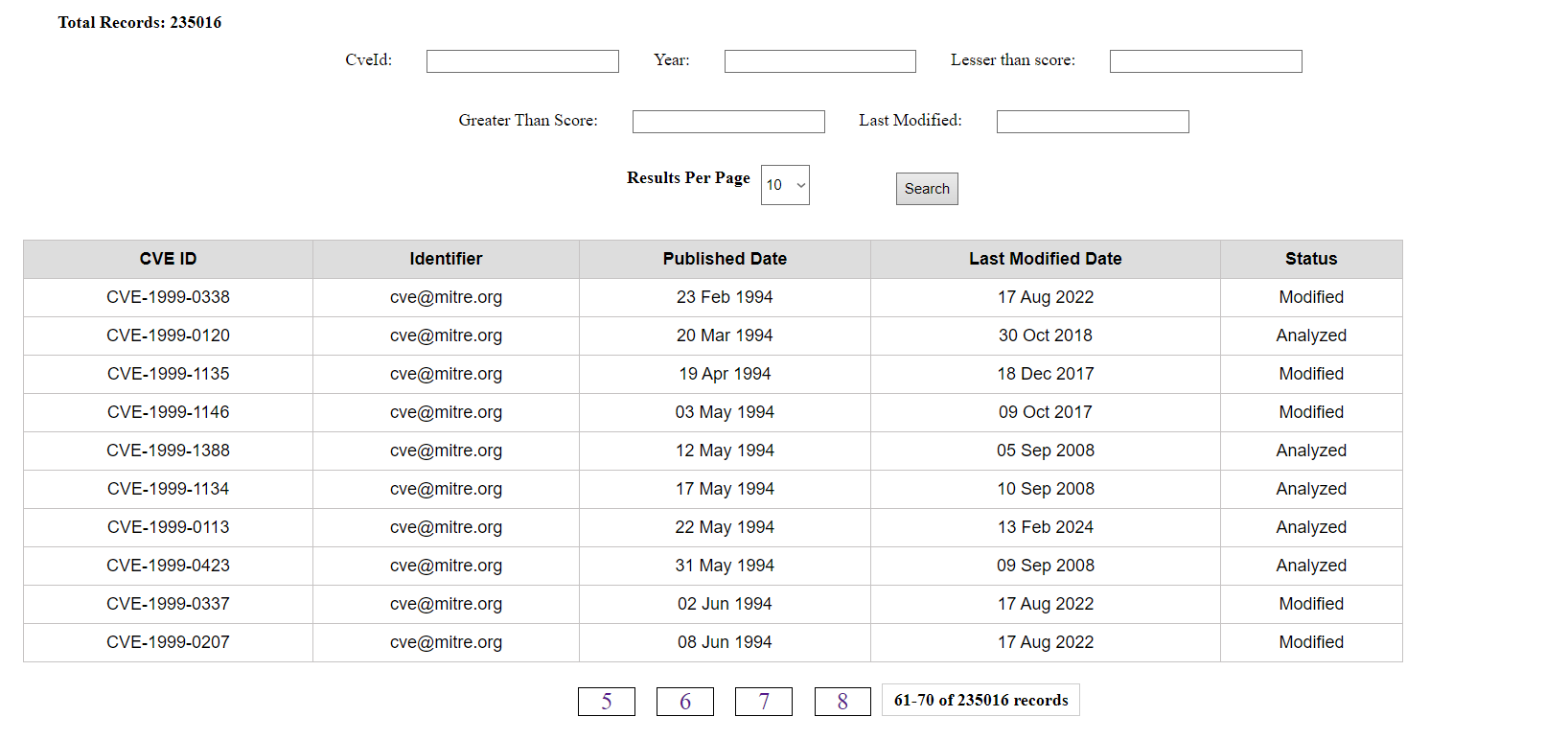
CVE LIST PAGE

****

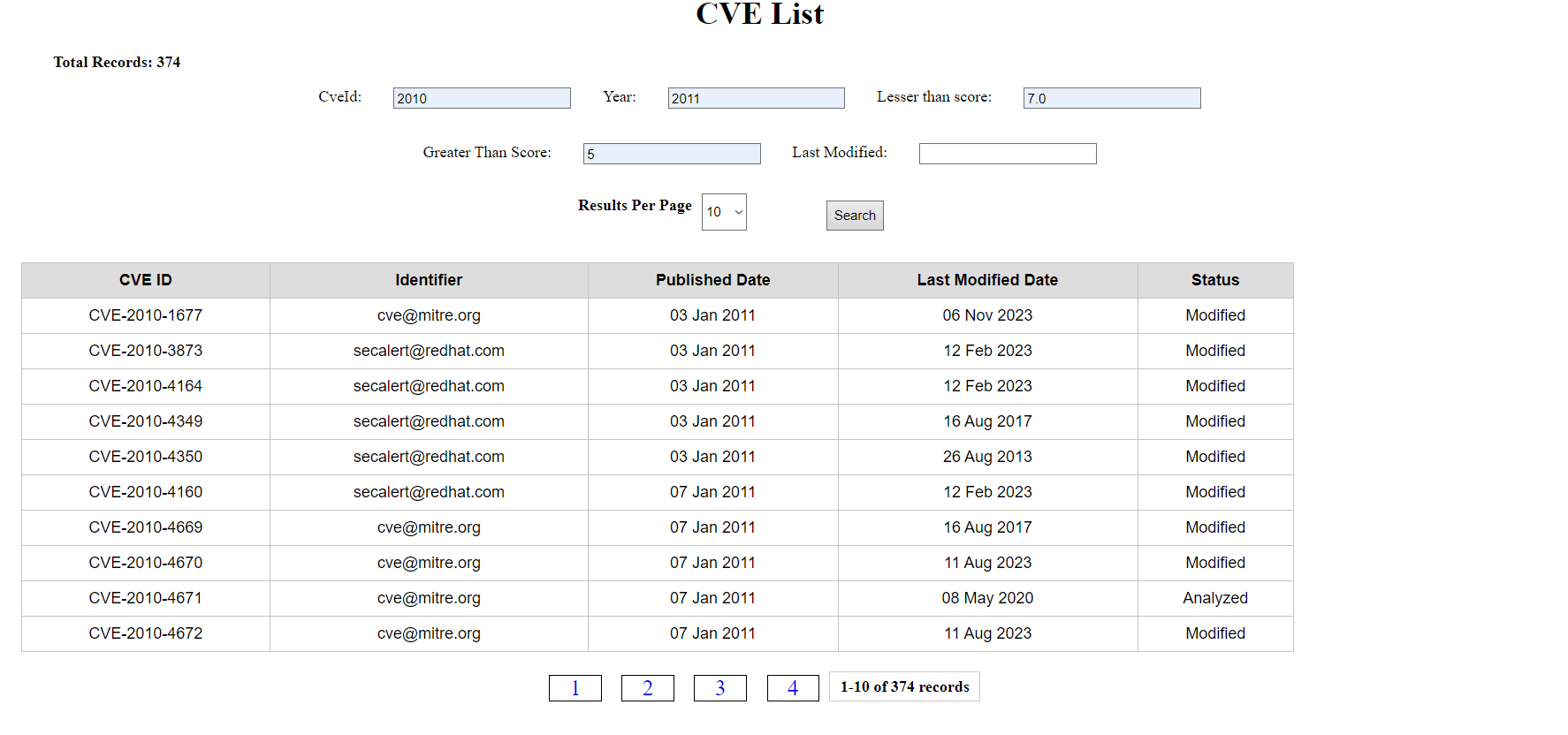
CVE DETAILS PAGE



PAGINATION EXAMPLE



CVE TABLES WITH FILTERS



**WORKFLOW**

Admin launches the NODE.JS server, connecting it to an empty MongoDB container initially, specified in the .env file's MONGO\_URL. The server operates on the designated port specified in the .env file as PORT. To populate the local database with data from the NVD API, the admin starts the helper.js module of the application and the data is fetched in batches, with batch size determined by configurations, taking approximately 30 minutes to 1 hour depending on data volume.

For public access, two endpoints are available. Initially, users are directed to the CVE lists page via http://<domain>:<PORT>/cves/lists. The URL query parameters defaults

- page : 1,

- limit : 10,

- year : “ ” (i.e. empty string)

- lastModified: -1

- lt(lesser than score):10.0

- gt(greater than score):0.0

- id : “ ” (i.e. empty string)

Interaction with search parameter modifies these parameters, reflecting corresponding changes in displayed table.

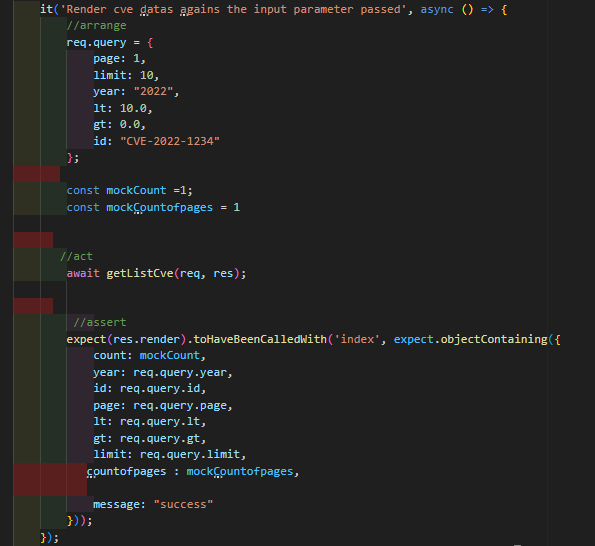
Additionally, the server automatically checks for NVD updates from the <https://services.nvd.nist.gov/rest/json/cvehistory/2.0> API every 30 minutes, updating the local database accordingly.

**UNIT TESTS**

1) **Unit test case for /Cve/lists Endpoint**

This unit test validates the functionality of rendering CVE data for the input parameters passed in the CVE application. This test sets the request query parameters to include a page number, limit, year (set to "2022" in this case), lt (lesser than), gt (greater than), and the CVE ID ("CVE-2022-1234" in this case).

With the expected count of CVE entries being 1 and the count of pages being 1, the test verifies that the application renders the data appropriately. Upon executing the getListCve function asynchronously, the test verifies that the application renders the 'index' view with the expected parameters, including the count of CVEs, year, CVE ID, page number, limit, lt, gt, count of pages, and a success message.



2) **Unit test case for /cves/<CVE-ID>**

This unit test case verifies the functionality of fetching details for a specific CVE. It operates by providing the CVE ID as input through URL parameters. The expected outcome is that the returned CVE data count is 1, with only one page as result. These anticipated values serve as the basis for assertions. The test proceeds by sending a GET request to the /cves/<CVE-ID> endpoint. It then confirms that the application successfully renders the CVE data, ensuring that the displayed CVE ID corresponds accurately to the input provided endpoint.



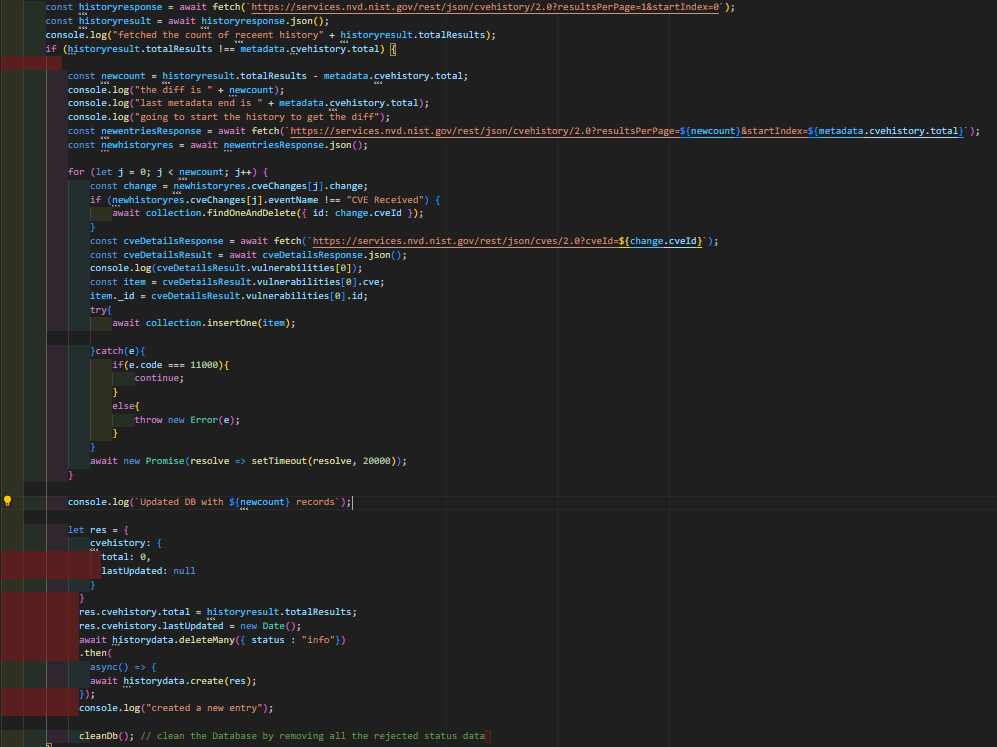
**Periodic Update**

The Function uses two API provided by NVD.

1. <https://services.nvd.nist.gov/rest/json/cvehistory/2.0>
2. <https://services.nvd.nist.gov/rest/json/cves/2.0>

How the update function works is explained below:

1. First the total number of results returned by the history API by setting the results displayed to 1 and start index as 0 are fetched.
2. The fetched count is compared with the local count of the last metadata check from the history API.
3. If they are equal , both local database and the API are in sync and the check is complete.
4. If they are not equal , the difference between the local count and the actual count is calculated.
5. Then the GET request is sent to history API to fetch the new or modified records by setting the parameters of resultsPerpage to difference value and startIndex is set to last local count.
6. The new records are then iterated and corresponding cve data is extracted from the CVE API using modified CVE ID fetched from the history API and last local copy of CVE data is deleted if present and new CVE data is inserted.
7. Once the record iterations are completed , DB and the API are in sync and wait for another 30 minutes.



**FILTERING AND SORTING LOGIC:**

The server receives inputs for filtering as query parameters, which are subsequently passed into find function with input as query filter to extract the relevant records. All the records are sorted in the increasing order of the published date. All computational tasks are executed on the server side.

