**Ahamed Rizwan .J.S**

**IIIrd yr-AI&DS**

**Dr.N.G.P Institute of Technology**

**Problem Statement :**

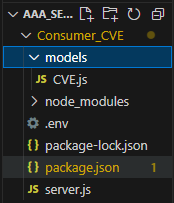
1.Consume the CVE information from the CVE API for all the CVE's and store it in a Database of your choice. - (API BaseURL - <https://services.nvd.nist.gov/rest/json/cves/2.0>)

To perform this, I created a separate API which Fetches the data from the URL and push it in a MONGODB database in a structured format.

**Steps and Logic:**

* Create a folder CONSUMER\_API and install all the necessary packages needed to perform this.
* ie) axios dotenv mongoose node-cron nodemon
* Create a folder named models inside the CONSUMER\_API and create a CVE.js file in which we define the entire schema of the API data.
* Create a .env file and define the MONGODB URL there just to secure the MONGODB username and password.
* At Last we create a server.js file inside CONSUMER\_API folder to perform our required task.

**SAMPLE STRUCTURE :**



**CODE for defining Schema :**

**/Consumer\_CVE/**models**/**CV

Entire API data Schema is defined in this file.

const mongoose = require("mongoose");

const cvssDataSchema = new mongoose.Schema({

version: {

type: String,

},

vectorString: String,

accessVector: {

type: String,

},

accessComplexity: {

type: String,

const nodeSchema = new mongoose.Schema({

  operator: String,

  negate: Boolean,

  cpeMatch: [cpeMatchSchema],

});

const configurationSchema = new mongoose.Schema({

  nodes: [nodeSchema],

});

const referenceSchema = new mongoose.Schema({

  url: String,

  source: String,

});

const CVEschema = new mongoose.Schema(

  {

    id: { type: String, required: true },

    sourceIdentifier: String,

    published: Date,

    lastModified: Date,

    vulnStatus: String,

    descriptions: [descriptionSchema],

    metrics: {

      cvssMetricV2: [cvssMetricV2Schema],

    },

    weaknesses: [weaknessSchema],

    configurations: [configurationSchema],

    references: [referenceSchema],

  },

  { collection: "cves" }

);

module.exports = mongoose.model("CVE", CVEschema);

**Inside .env file** :

To secure the mongodb username and password (while pushing into github)

To configure this .env file we have imported dotenv package, so that we can use those keys.

**Server.js file – Code to fetch data from API and push it to mongodb**

require("dotenv").config();

const axios = require("axios");

const mongoose = require("mongoose");

const CVE = require("./models/CVE");

const cron = require("node-cron");

const fetchCVEs = async () => {

  try {

    const response = await axios.get(

      "https://services.nvd.nist.gov/rest/json/cves/2.0"

    );

    return response.data && response.data.vulnerabilities

      ? response.data.vulnerabilities

      : [];

  } catch (error) {

    console.error("Error fetching CVE data:", error);

    return [];

  }

};

const cleanseData = (data) => {

  return data.map((item) => {

    if (item.cve && item.cve.descriptions) {

      item.cve.descriptions = item.cve.descriptions.filter(

        (desc) => desc.lang === "en"

      );

    }

    return item;

  });

};

const storeCVEs = async (cveData) => {

  try {

    await mongoose.connect(process.env.DB\_URI);

    for (let data of cveData) {

      await CVE.updateOne({ id: data.id }, data, { upsert: true });

    }

    console.log(`Updated records in database.`);

  } catch (error) {

    console.error("Error storing CVEs to MongoDB:", error);

  } finally {

    mongoose.disconnect();

  }

};

async function processCVEs() {

  try {

    const data = await fetchCVEs();

    const cleanedData = cleanseData(data);

    const transformedData = cleanedData

      .map((item) => {

        if (!item.cve) {

          console.error("CVE object is undefined.");

          return null;

        }

        return {

          id: item.cve.id,

          sourceIdentifier: item.cve.sourceIdentifier,

          published: new Date(item.cve.published),

          lastModified: new Date(item.cve.lastModified),

          vulnStatus: item.cve.vulnStatus,

          descriptions: item.cve.descriptions,

          metrics: item.cve.metrics,

          weaknesses: item.cve.weaknesses,

          configurations: item.cve.configurations,

          references: item.cve.references,

        };

      })

      .filter((item) => item !== null);

    await storeCVEs(transformedData);

  } catch (error) {

    console.error("Error in processing CVEs:", error);

  }

}

cron.schedule("0 0 \* \* \*", () => {

  console.log("Running scheduled CVE update...");

  processCVEs();

});

processCVEs();

**Functions :**

* **fetchCVEs :** Function to fetch all the data from API and return the required data which needs to be pushed into the database.
* **CleansData :** To ignore unwanted data which was not being used**.**
* **StoreCVE :** Stores unique data and override if any duplicate.
* **ProcessCVEs:** It stores Common Vulnerabilities(CVE) data from an external source into a MongoDB database, while also scheduling periodic updates using node-cron.

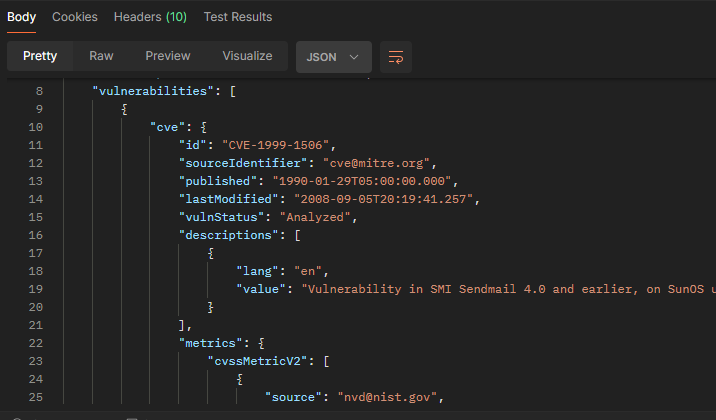
**Problem Statement :**

**2.** Hint for accessing all the CVE's from API - Through a series of smaller

“chunked” responses controlled by an offset startIndex and a page limit resultsPerPage users may page through all the CVE in the NVD.

* Using query parameters and fetching the data .This approach has been used in my code as per the given hint.
* <https://services.nvd.nist.gov/rest/json/cves/2.0?startIndex=5&resultsPerPage=1>

**Output (Postman) :**



**Problem Statement :**

**3.** Apply data cleansing & de-duplication, ensure data quality wherever

Applicable

**Data cleansing :**

In description key we have both English and Spanish we never use this Spanish description anywhere so we don’t take it.

const cleanseData = (data) => {

  return data.map((item) => {

    if (item.cve && item.cve.descriptions) {

      item.cve.descriptions = item.cve.descriptions.filter(

        (desc) => desc.lang === "en"

      );

    }

    return item;

  });

};

**De-duplication :**

If same id is there it will just override. (upsert =True) This will take care of it.

const storeCVEs = async (cveData) => {

  try {

    await mongoose.connect(process.env.DB\_URI);

    for (let data of cveData) {

      await CVE.updateOne({ id: data.id }, data, { upsert: true });

    }

    console.log(`Updated records in database.`);

  } catch (error) {

    console.error("Error storing CVEs to MongoDB:", error);

  } finally {

    mongoose.disconnect();

  }

};

**Problem Statement :**

**4**. CVE details should be synchronized into Database periodically in batch

mode in a specific time period

**“0 0 \* \* \*” This is an cron expression specifying the schedule. It run the tasks daily at midnight.**

cron.schedule("0 0 \* \* \*", () => {

  console.log("Running scheduled CVE update...");

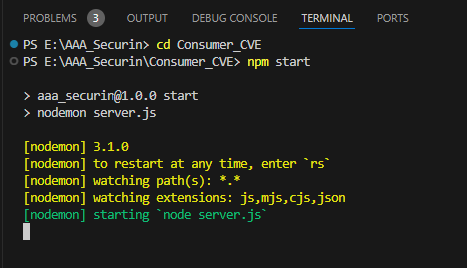
  processCVEs();

});

processCVEs();

**OUTPUT :**

**Terminal**



**MONGODB COMPASS**



**Problem Statement :**

**5**. Develop API’s to read & filter the CVE details by below parameters -

* CVE ID
* CVE ID’s belongs to a specific year
* CVE Score (Field to ref - metrics.cvssMetricV2.cvssData.baseScore
* last Modified in N days.

To perform this, I created a separate API which has all the methods to be used by the front–end.

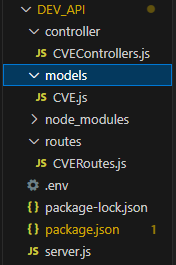
**Steps and Logic:**

* Create a folder DEV\_API and install all the necessary packages needed to perform this.

ie) cors dotenv mongoose express nodemon

* Create a copy of model folder present in CONSUMER\_API and reuse it ,as it has defined schema which is needed here to perform the required task.
* Create a .env file and define the MONGODB URL and Port No. there just to security while pushing into github.
* Create a route folder and create a file CVE\_routes.js to define the routes and its responses.
* AtLast, we create our main file **server.js** to connect front-end and back-end

**SAMPLE STRUCTURE :**



**Back-end methods are being defined in this CVEController.js file :**

The Below back-end methods are used in the front-end when any onclick event occurs.

**Function To get the total number of records in the database.**

const CVE = require("../models/CVE");

exports.getTotalCount = async (req, res) => {

  try {

    const count = await CVE.countDocuments();

    res.json({ total: count });

  } catch (error) {

    res.status(500).json({ message: error.message });

  }

}

**Function To retrieve CVE data based on a specific Start Index and Count .**

This function will retrieve the first 10 documents from the database based on startIndex.

exports.getDataByIndexAndCount = async (req, res) => {

  try {

    const startIndex = parseInt(req.query.startIndex) || 0;

    const count = parseInt(req.query.count) || 10;

    const data = await CVE.find({}, {

      id: 1,

      sourceIdentifier: 1,

      published: 1,

      lastModified: 1,

      vulnStatus: 1

    }).skip(startIndex).limit(count);

    res.json(data);

  } catch (error) {

    res.status(500).json({ message: error.message });

  }

}

**Function To retrieve CVE data based on a specific CVE id**

exports.getDataById = async (req, res) => {

    try {

      const keyId = req.params.id;

      const data = await CVE.findOne({ id: keyId });

      if (!data) {

        return res.status(404).json({ message: "Data not found" });

      }

      res.json(data);

    } catch (error) {

      res.status(500).json({ message: error.message });

    }

  }

**Routes file to specify the request and response for the events**

The above defined back-end methods and its responses are integrated with the respective request.

const express = require("express");

const router = express.Router();

const { getTotalCount, getDataByIndexAndCount,getDataById } = require("../controller/CVEControllers");

router.get("/total-records", getTotalCount);

router.get("/list", getDataByIndexAndCount);

router.get("/:id", getDataById);

module.exports = router;

**Server.js File to communicate btw the Front-end and back-end**

require("dotenv").config();

const express = require("express");

const mongoose = require("mongoose");

const cors = require("cors");

const cveRoutes = require("./routes/CVERoutes");

const app = express();

app.use(cors());

app.use(express.json());

const PORT = process.env.PORT;

mongoose

  .connect(process.env.DB\_URI)

  .then(() => console.log("MongoDB connected"))

  .catch((err) => console.log(err));

app.use("/cves/",cveRoutes);

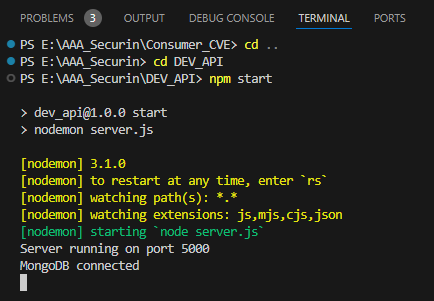
app.listen(PORT, () => {

  console.log(`Server running on port ${PORT}`);

});

**OUTPUT :**

**Terminal**



**REQ AND RESPONSE IN POSTMAN**

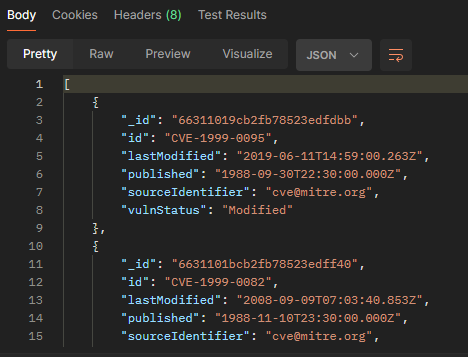
Seeing our back-end method responses



It Fetches data such as \_id, id, lastmodified, published, sourceIdentifier, vulnStatus which is need in the front-end.



**OUTPUT**

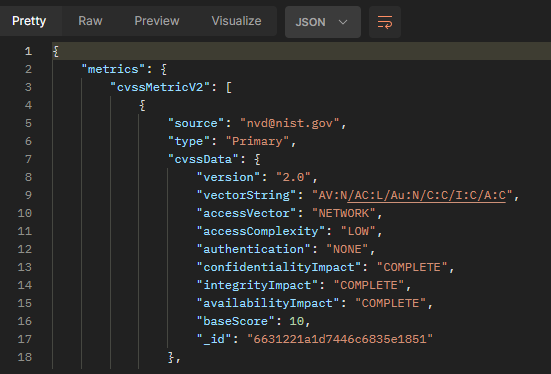




It Fetches data based on particular id.



**OUTPUT :**



**Problem Statement :**

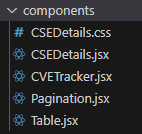
**6.** Read the API and visualise it in UI using HTML, CSS and Javascript.

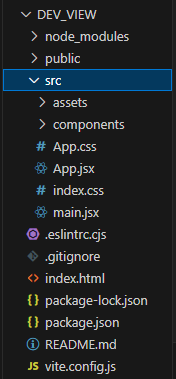
To perform this, I created a separate API Folder which contains all the Front-end files and its flows.

**Steps and Logic:**

* Create a folder **DEV\_VIEW** and install all the necessary packages needed to perform this.
* ie) vite react react-dom react-router-dom
* All required files and folders where already been set up using the command . **npm create vite@latest**.
* All files has been already configured with the above commands.
* To design the UI, create a folder named Components inside the src folder and create all the required components needed in the frontend.

**SAMPLE STRUCTURE :**





**CSEDetails.css** : contains the styling of all html elements **Pagination.jsx :** Methods to handle previous/next pages

**Table.jsx** : It contains the table Structure (first-page) **CSEDetails.jsx** : it contains the structure of second page

**CSETracker.jsx :** It combines all the components being included

**CSETracker.jsx :**

import React, { useState, useEffect } from 'react';

import axios from 'axios';

import Table from './Table';

import Pagination from './Pagination';

function CVETractor() {

  const [data, setData] = useState([]);

  const [totalCount, setTotalCount] = useState(0);

  const [startIndex, setStartIndex] = useState(0);

  const [count, setCount] = useState(10);

  const url = 'http://localhost:5000/cves';

  useEffect(() => {

    fetchData();

    fetchTotalCount();

  }, [startIndex, count]);

  const fetchData = async () => {

    try {

      const response = await axios.get(`${url}/list?startIndex=${startIndex}&count=${count}`);

      setData(response.data);

    } catch (error) {

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

    }

  };

  const fetchTotalCount = async () => {

    try {

      const totalCountResponse = await axios.get(`${url}/total-records`);

      setTotalCount(totalCountResponse.data.total);

    } catch (error) {

      console.error('Error fetching total count:', error);

    }

  };

  const handlePagination = (newStartIndex) => {

    setStartIndex(newStartIndex);

  };

  const handleCountChange = (event) => {

    setCount(parseInt(event.target.value));

    setStartIndex(0);

  };

  return (

    <div>

      <h1>CVE Tracker</h1>

      <div>

        Total Count: {totalCount}

        <select value={count} onChange={handleCountChange}>

          <option value={10}>10</option>

          <option value={50}>50</option>

          <option value={100}>100</option>

        </select>

      </div>

      <Table data={data} />

      <Pagination totalCount={totalCount} startIndex={startIndex} count={count} onPageChange={handlePagination} />

    </div>

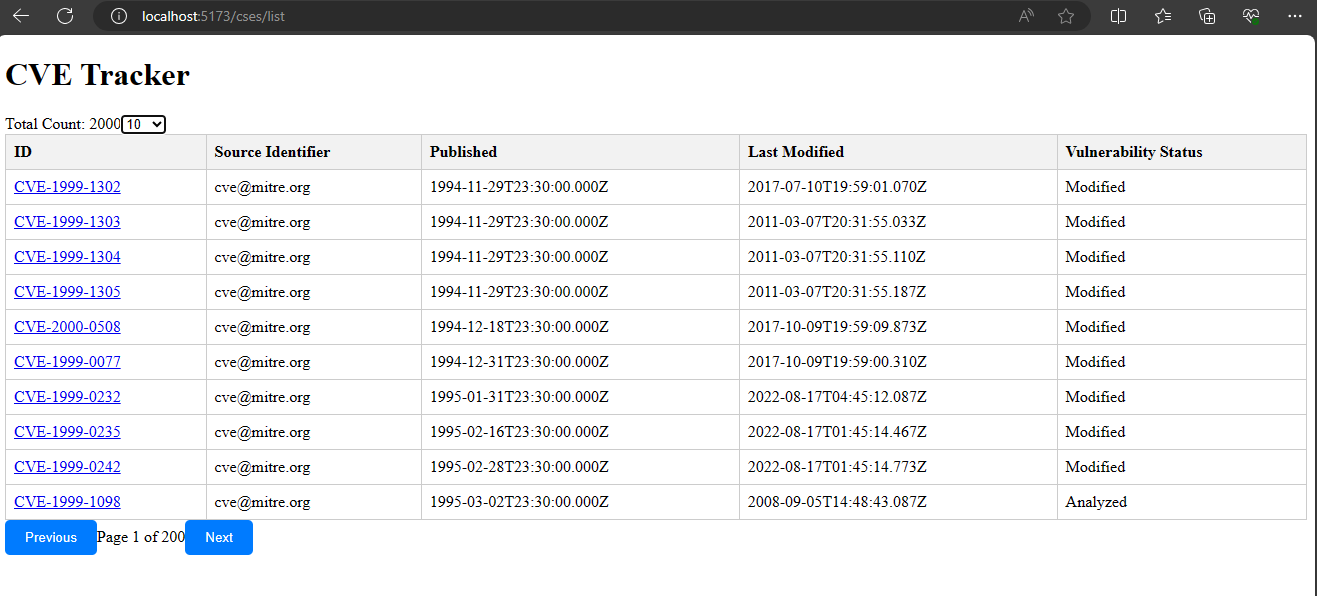
  );

}

export default CVETractor;

**OUTPUT (FIRST-PAGE ) :**

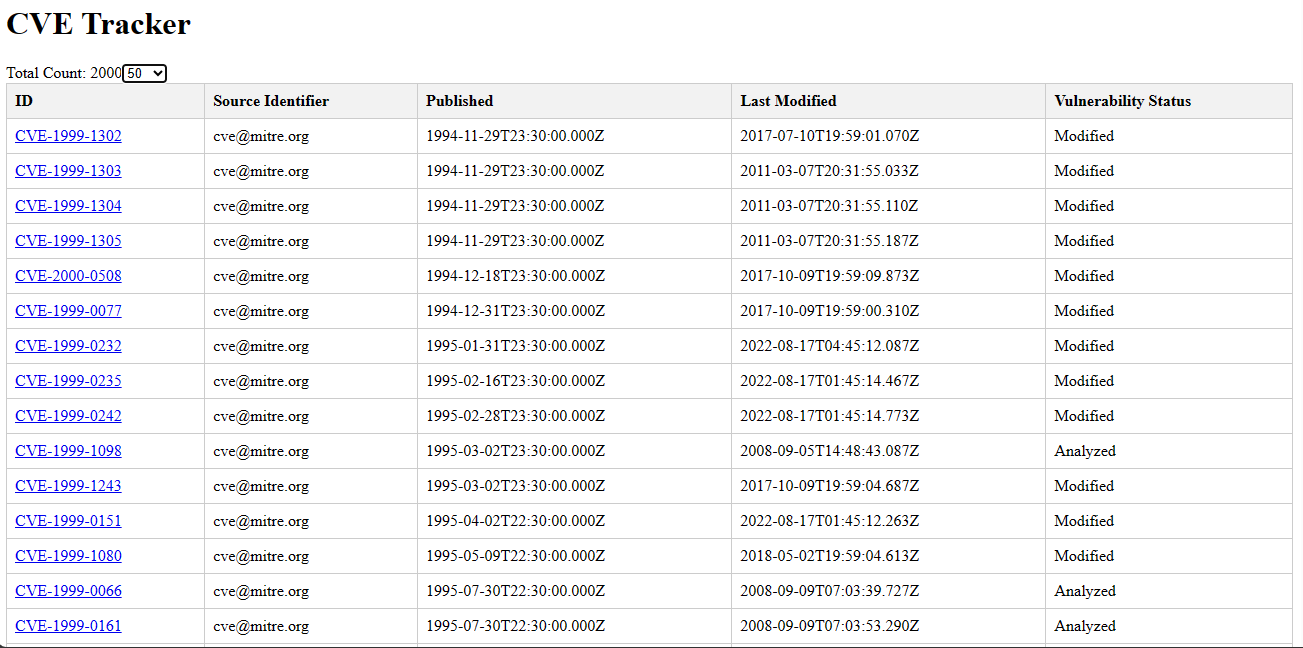
route path /cves/list, Read the API and display its results in a table with a "Total Records" count.



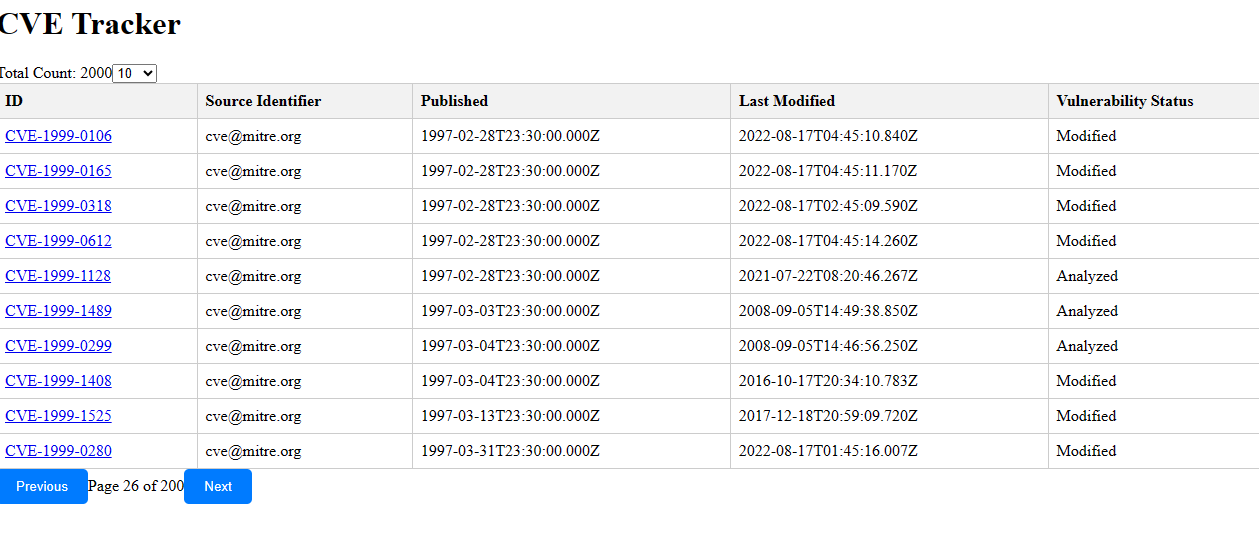
Include "Results Per Page" below the table, offering options of "10", "50", and "100",

with a default selection of "10". Whenever an option is chosen, execute the

respective API call to retrieve the records.(Good to have)



Add server side “Pagination” functionality (Optional - Added advantage)



**OUTPUT (SECOND-PAGE)**

When a row is clicked, navigate to the second page /cves/cve-1999-0334. The

second page should include the following:

* Perform an API call to retrieve the data of the selected CVE and display it in the user interface.

