personal Background

Name: Prathap V
GitHub: https://github.com/vprathap21

LinkedIn : https://www.linkedin.com/in/prathap-v-751566249/

Email: 211501064@rajalakshmi.edu.in

API Document Link: https://cve-information-project-1.onrender.com/api-docs

GitHub Repository Link: https://github.com/vprathap21/cve-information-project

City: Chennai

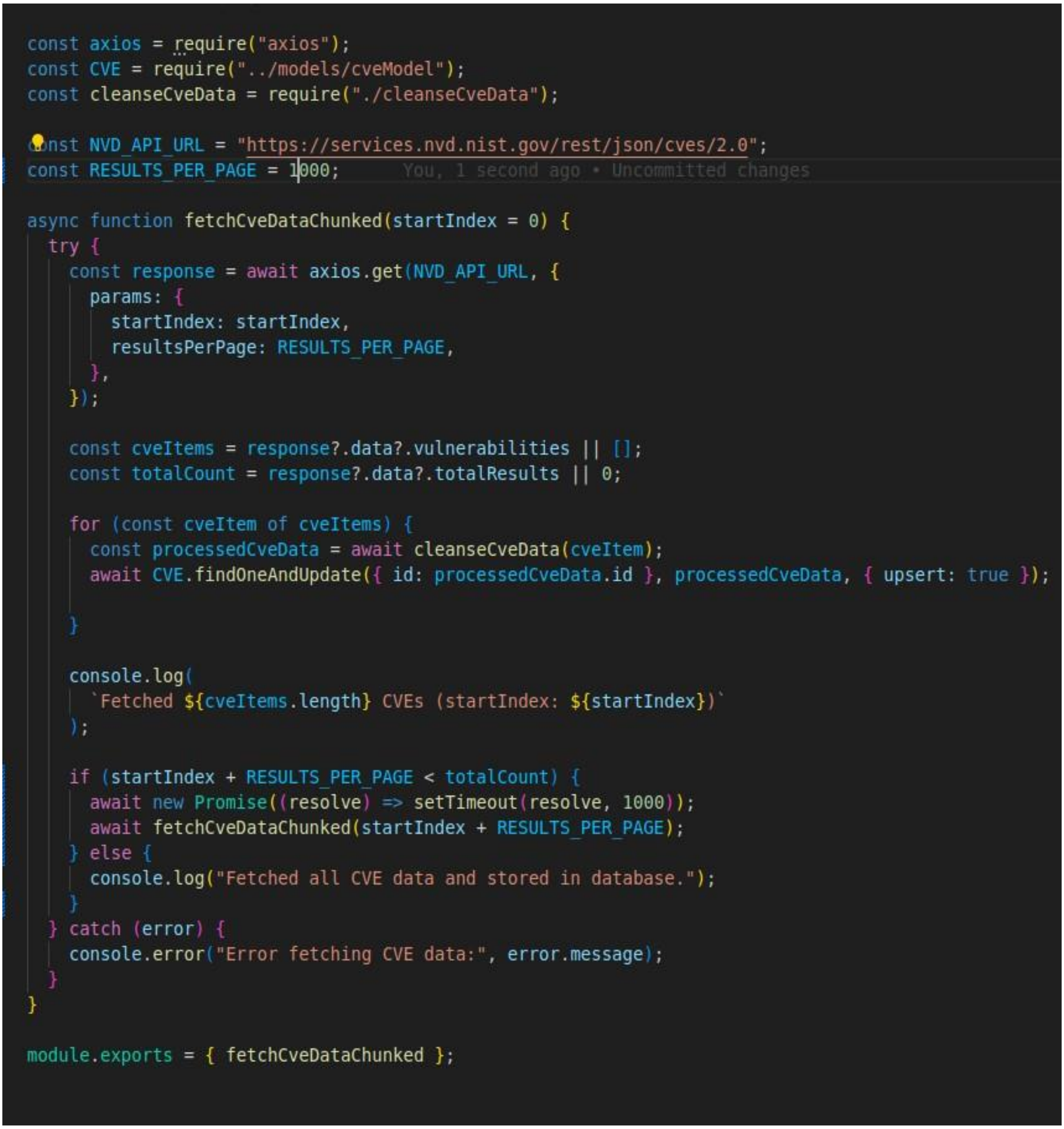
III-year Artificial Intelligence and Machine Learning

Rajalakshmi Engineering College

Project Details:

1. Consume CVE Information from the API:

I designed a recursive asynchronous function**, fetchCveDataChunked**, to fetch CVE data from the NIST API. It retrieves data in manageable chunks, preventing overload. Pagination ensures we only request 1,000 entries at a time. Cleaned data is then efficiently stored in MongoDB using Mongoose's upsert feature, ensuring seamless integration. Lastly, error handling ensures any issues during data retrieval are promptly addressed.

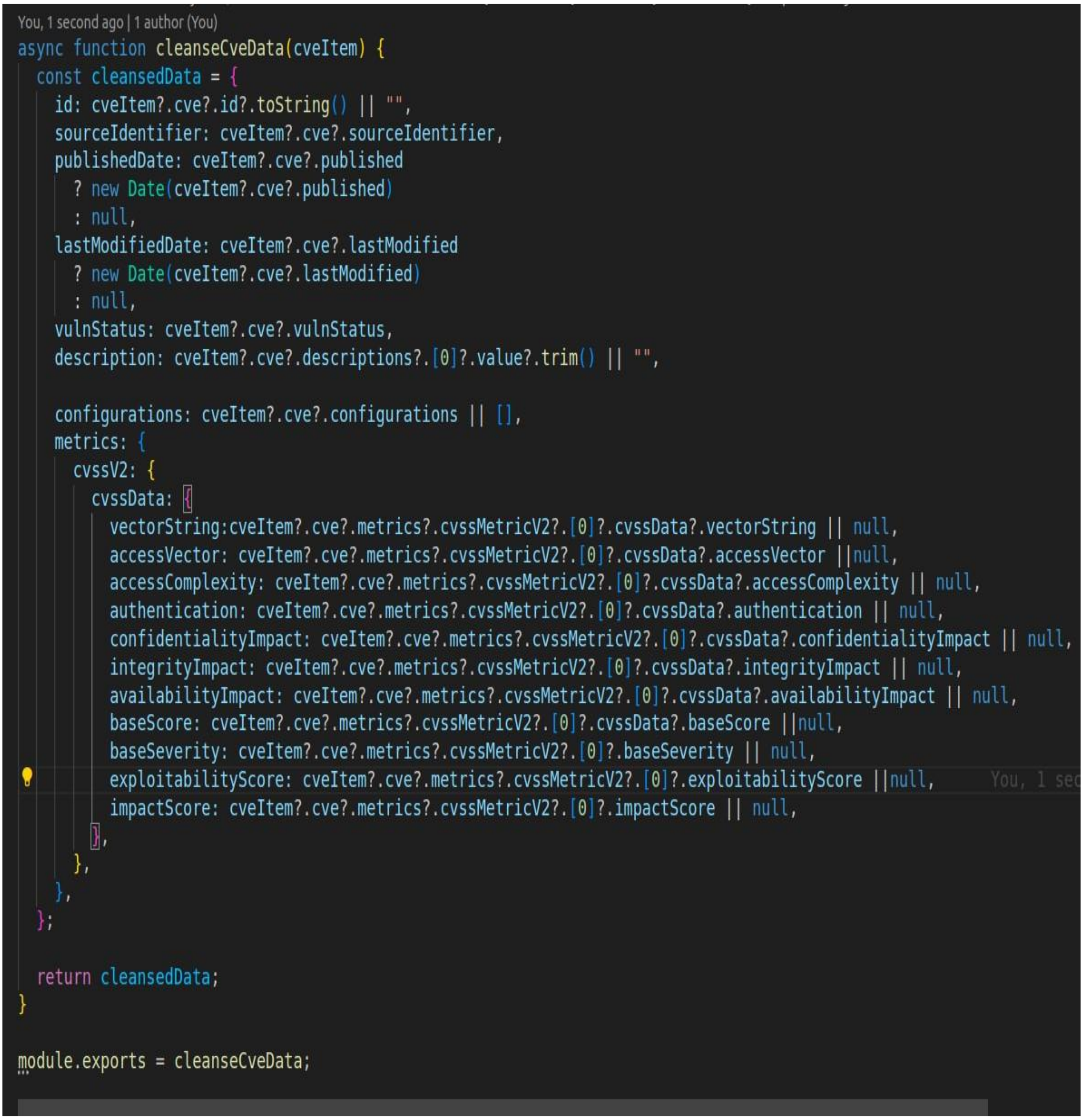


```javascript
const axios = require("axios");
const CVE = require("../models/cveModel");
const cleanseCveData = require("./cleanseCveData");

const NVD_API_URL = "https://services.nvd.nist.gov/rest/json/cves/2.0";
const RESULTS_PER_PAGE = 1000;      You, 1 second ago • Uncommitted changes

async function fetchCveDataChunked(startIndex = 0) {
  try {
    const response = await axios.get(NVD_API_URL, {
      params: {
        startIndex: startIndex,
        resultsPerPage: RESULTS_PER_PAGE,
      },
    });

    const cveItems = response?.data?.vulnerabilities || [];
    const totalCount = response?.data?.totalResults || 0;

    for (const cveItem of cveItems) {
      const processedCveData = await cleanseCveData(cveItem);
      await CVE.findOneAndUpdate({ id: processedCveData.id }, processedCveData, { upsert: true });

    }

    console.log(
      `Fetched ${cveItems.length} CVEs (startIndex: ${startIndex})`
    );

    if (startIndex + RESULTS_PER_PAGE < totalCount) {
      await new Promise((resolve) => setTimeout(resolve, 1000));
      await fetchCveDataChunked(startIndex + RESULTS_PER_PAGE);
    } else {
      console.log("Fetched all CVE data and stored in database.");
    }
  } catch (error) {
    console.error("Error fetching CVE data:", error.message);
  }
}

module.exports = { fetchCveDataChunked };
```

2. Data Cleansing and Deduplication:

In ensuring the quality of our data, I implemented several data cleansing techniques within the **cleanseCveData function**. This involved utilizing optional chaining (?.) extensively to handle potentially undefined properties without causing errors. For instance, I used optional chaining to access nested properties such as **cveItem?.cve?.id?.toString()**. This technique allows us to safely access properties without encountering errors if they are undefined.

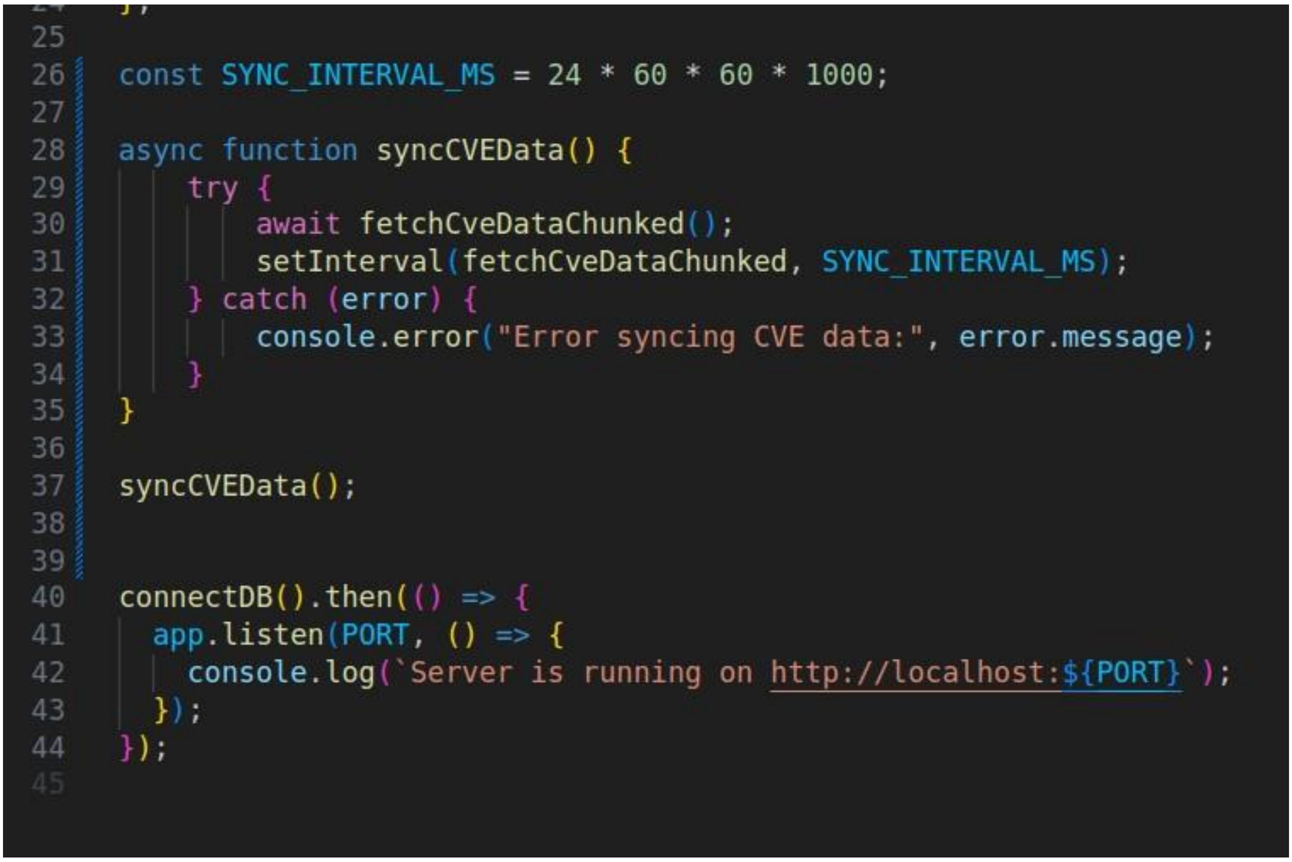
Additionally, I employed the || operator to provide default values for properties that may be undefined. For example, I used the expression **cveItem?.cve?.descriptions?.[0]?.value?.trim() || ""** to ensure that the description property is always a string, even if the value is undefined.



```
You, 1 second ago | 1 author (You)
async function cleanseCveData(cveItem) {
  const cleansedData = {
    id: cveItem?.cve?.id?.toString() || "",
    sourceIdentifier: cveItem?.cve?.sourceIdentifier,
    publishedDate: cveItem?.cve?.published
      ? new Date(cveItem?.cve?.published)
      : null,
    lastModifiedDate: cveItem?.cve?.lastModified
      ? new Date(cveItem?.cve?.lastModified)
      : null,
    vulnStatus: cveItem?.cve?.vulnStatus,
    description: cveItem?.cve?.descriptions?.[0]?.value?.trim() || "",

    configurations: cveItem?.cve?.configurations || [],
    metrics: {
      cvssV2: {
        cvssData: {
          vectorString:cveItem?.cve?.metrics?.cvssMetricV2?.[0]?.cvssData?.vectorString || null,
          accessVector: cveItem?.cve?.metrics?.cvssMetricV2?.[0]?.cvssData?.accessVector ||null,
          accessComplexity: cveItem?.cve?.metrics?.cvssMetricV2?.[0]?.cvssData?.accessComplexity || null,
          authentication: cveItem?.cve?.metrics?.cvssMetricV2?.[0]?.cvssData?.authentication || null,
          confidentialityImpact: cveItem?.cve?.metrics?.cvssMetricV2?.[0]?.cvssData?.confidentialityImpact || null,
          integrityImpact: cveItem?.cve?.metrics?.cvssMetricV2?.[0]?.cvssData?.integrityImpact || null,
          availabilityImpact: cveItem?.cve?.metrics?.cvssMetricV2?.[0]?.cvssData?.availabilityImpact || null,
          baseScore: cveItem?.cve?.metrics?.cvssMetricV2?.[0]?.cvssData?.baseScore ||null,
          baseSeverity: cveItem?.cve?.metrics?.cvssMetricV2?.[0]?.baseSeverity || null,
          exploitabilityScore: cveItem?.cve?.metrics?.cvssMetricV2?.[0]?.exploitabilityScore ||null,      You, 1 sec
          impactScore: cveItem?.cve?.metrics?.cvssMetricV2?.[0]?.impactScore || null,
        },
      },
    },
  };

  return cleansedData;
}

module.exports = cleanseCveData;
```

3. Periodic Synchronization into Database

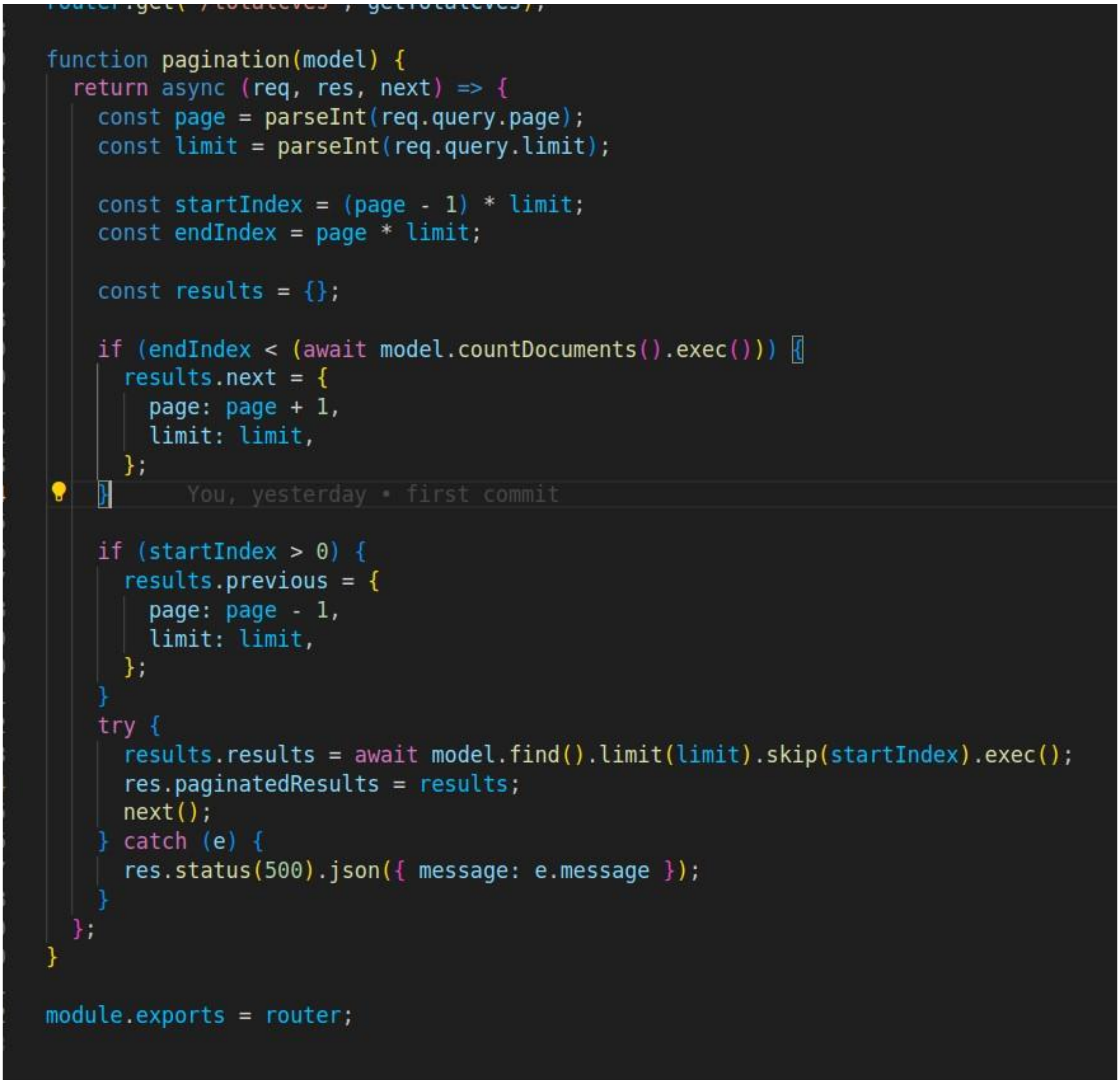
To keep our CVE data fresh, I set up this code to run like an automatic updater.This code initiates a periodic synchronization process, ensuring our CVE data stays up-to-date. Initially, it calls fetchAndSaveCVEData to fetch and store data. Then, it sets up an interval to repeatedly call this function at defined intervals (24 hours). If anything goes wrong during this update process, no worries, the code catches those errors and logs them for us to review later.



```
});

const SYNC_INTERVAL_MS = 24 * 60 * 60 * 1000;

async function syncCVEData() {
    try {
        await fetchCveDataChunked();
        setInterval(fetchCveDataChunked, SYNC_INTERVAL_MS);
    } catch (error) {
        console.error("Error syncing CVE data:", error.message);
    }
}

syncCVEData();


connectDB().then(() => {
  app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
  });
});
```

4. Server-side Pagination Functionality

I did server side pagination for fetching the data from the database. The function extracts page and limit parameters from the request query to determine the document range. It initializes an empty results object for storing paginated data and prepares pagination metadata. It checks if more documents exist beyond the current page and sets up next and previous properties accordingly. Then, it executes a database query using find() with limit and skip options to fetch paginated documents. Paginated results are stored in the results object and attached to the response (res.paginatedResults) for easy access by subsequent middleware or route handlers.



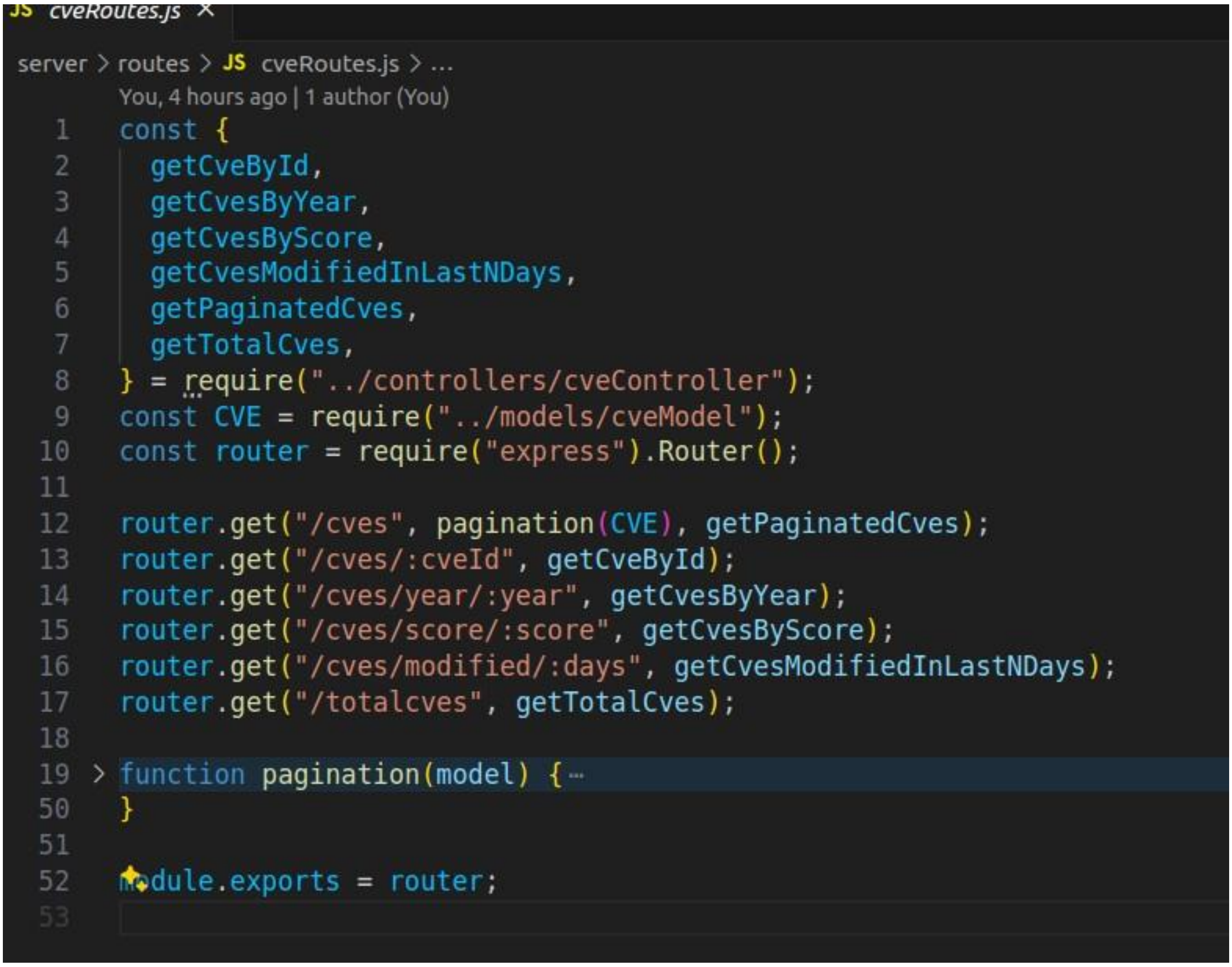
```
router.get("/totalevs", getTotalevs);

function pagination(model) {
  return async (req, res, next) => {
    const page = parseInt(req.query.page);
    const limit = parseInt(req.query.limit);

    const startIndex = (page - 1) * limit;
    const endIndex = page * limit;

    const results = {};

    if (endIndex < (await model.countDocuments().exec())) {
      results.next = {
        page: page + 1,
        limit: limit,
      };
    }       You, yesterday • first commit

    if (startIndex > 0) {
      results.previous = {
        page: page - 1,
        limit: limit,
      };
    }
    try {
      results.results = await model.find().limit(limit).skip(startIndex).exec();
      res.paginatedResults = results;
      next();
    } catch (e) {
      res.status(500).json({ message: e.message });
    }
  };
}

module.exports = router;
```

API Development:

My Thought Process:

In designing the API routes and middleware, I prioritized modularity and scalability to ensure the codebase remains organized and adaptable to future requirements. By separating route definitions and controllers into distinct files, I aimed code maintainability and modularity

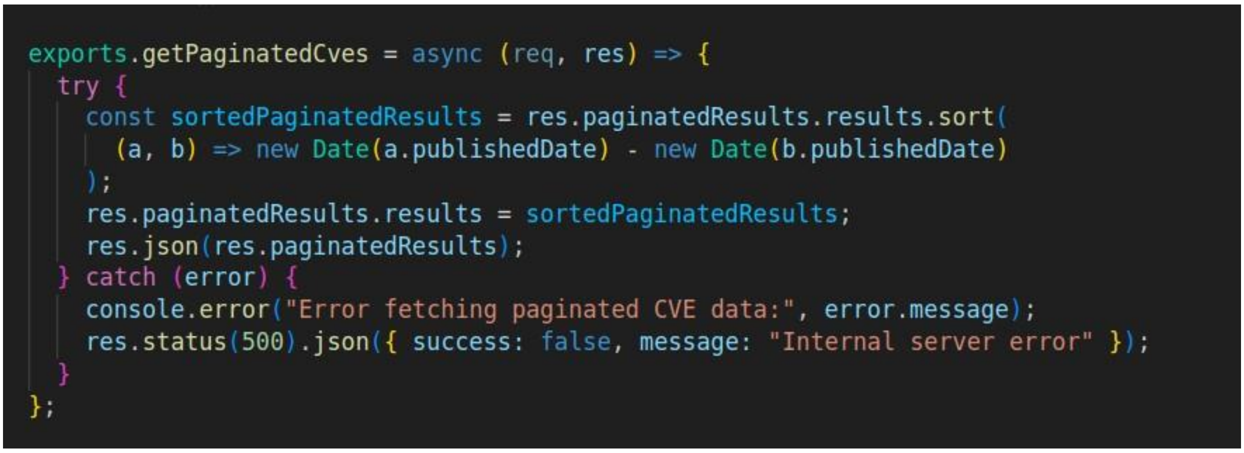
Furthermore, the implementation of pagination middleware reflects my attention to optimizing API performance, especially when handling large datasets. This middleware abstracts pagination logic away from individual route handlers, promoting code reuse and simplifying future enhancements.


JS cveRoutes.js ×
server > routes > JS cveRoutes.js > ...
You, 4 hours ago | 1 author (You)

```javascript
const {
  getCveById,
  getCvesByYear,
  getCvesByScore,
  getCvesModifiedInLastNDays,
  getPaginatedCves,
  getTotalCves,
} = require("../controllers/cveController");
const CVE = require("../models/cveModel");
const router = require("express").Router();

router.get("/cves", pagination(CVE), getPaginatedCves);
router.get("/cves/:cveId", getCveById);
router.get("/cves/year/:year", getCvesByYear);
router.get("/cves/score/:score", getCvesByScore);
router.get("/cves/modified/:days", getCvesModifiedInLastNDays);
router.get("/totalcves", getTotalCves);

function pagination(model) {⋯
}

module.exports = router;
```

1. getPaginatedCves Function:

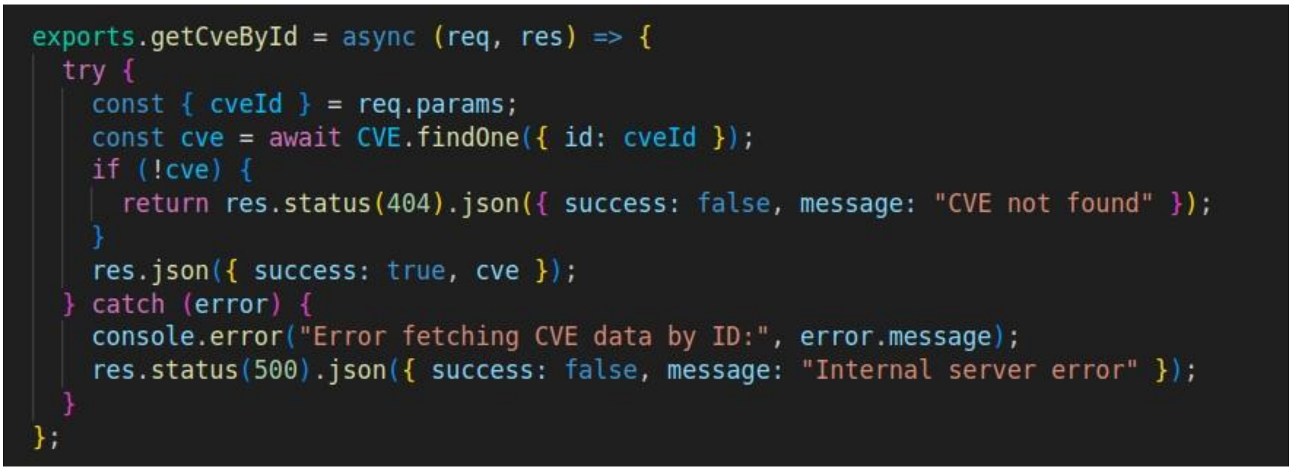
- This function is responsible for sorting and returning paginated CVE data.
- First, I sort the paginated results based on the publishedDate in ascending order.
- Then, I update the paginated results object with the sorted data and send it back as a JSON response.
- Error handling is in place to catch any exceptions during the process and return an appropriate error message if encountered.



```
exports.getPaginatedCves = async (req, res) => {
  try {
    const sortedPaginatedResults = res.paginatedResults.results.sort(
      (a, b) => new Date(a.publishedDate) - new Date(b.publishedDate)
    );
    res.paginatedResults.results = sortedPaginatedResults;
    res.json(res.paginatedResults);
  } catch (error) {
    console.error("Error fetching paginated CVE data:", error.message);
    res.status(500).json({ success: false, message: "Internal server error" });
  }
};
```

2. getCveById Function:

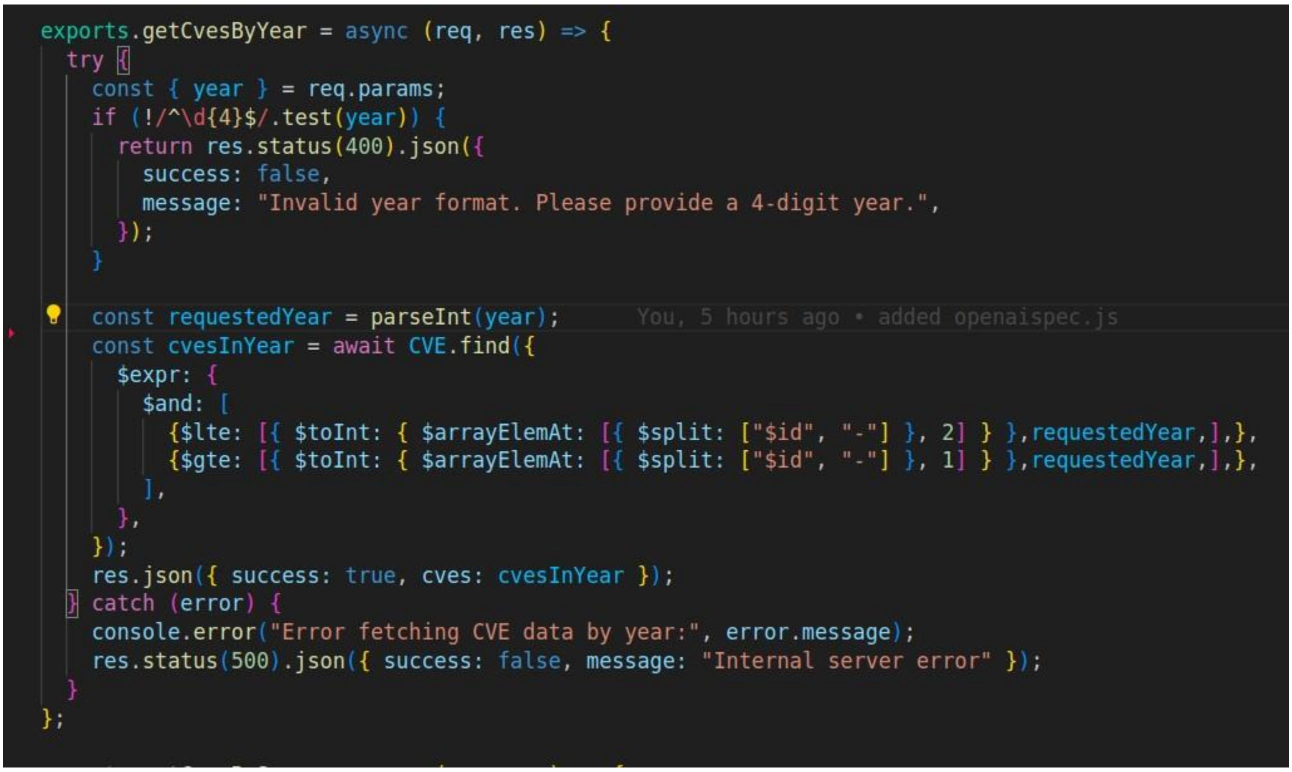
getCveById function retrieves CVE details based on a specific CVE ID extracted from the request URL. It queries the CVE model to find the corresponding document and returns it as part of a JSON response with a success indicator if found. Error handling ensures that relevant error messages are returned if the CVE document is not found or if an error occurs during the retrieval process.



```
exports.getCveById = async (req, res) => {
  try {
    const { cveId } = req.params;
    const cve = await CVE.findOne({ id: cveId });
    if (!cve) {
      return res.status(404).json({ success: false, message: "CVE not found" });
    }
    res.json({ success: true, cve });
  } catch (error) {
    console.error("Error fetching CVE data by ID:", error.message);
    res.status(500).json({ success: false, message: "Internal server error" });
  }
};
```

3. getCvesByYear Function:

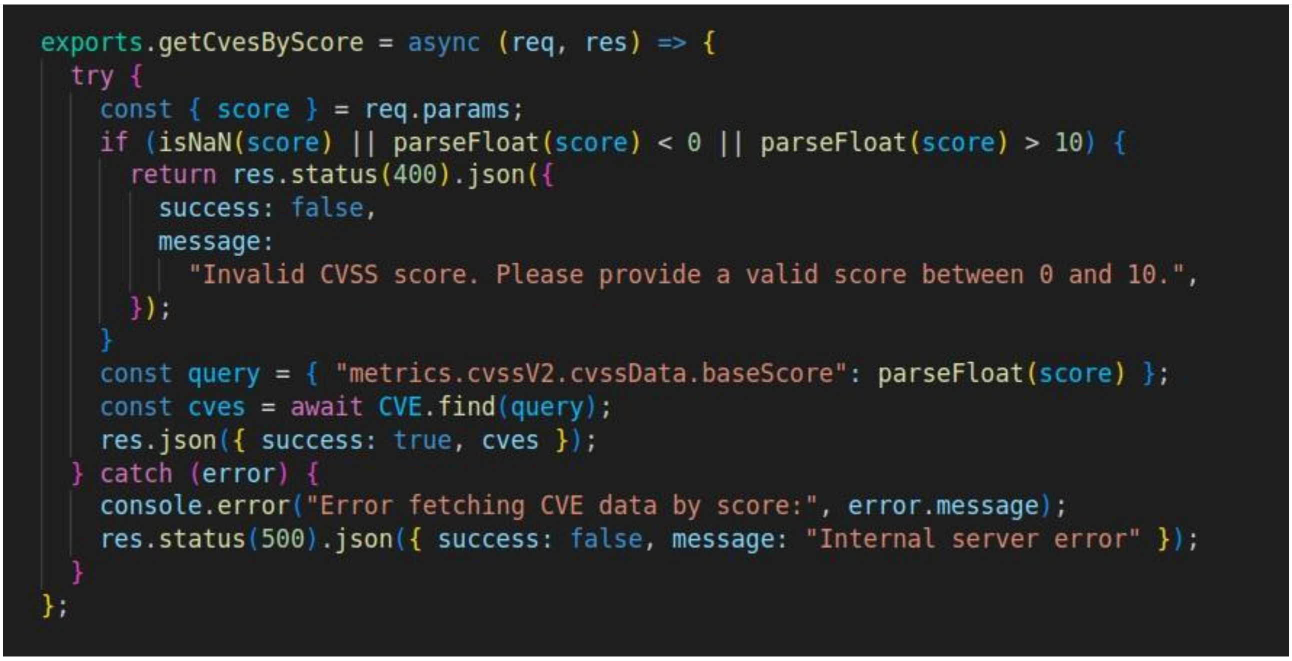
This function retrieves CVE details for a specific year by extracting the year parameter from the request URL and validating its format. Then, it constructs a MongoDB query to find CVE documents where the requested year falls within the start and end years of the CVE ID. The matching CVE documents are sent back as part of a JSON response along with a success indicator. Proper error handling is in place to address invalid year formats or any potential errors that might occur during the process.



```
exports.getCvesByYear = async (req, res) => {
  try {
    const { year } = req.params;
    if (!/^\d{4}$/.test(year)) {
      return res.status(400).json({
        success: false,
        message: "Invalid year format. Please provide a 4-digit year.",
      });
    }

    const requestedYear = parseInt(year);      You, 5 hours ago • added openaispec.js
    const cvesInYear = await CVE.find({
      $expr: {
        $and: [
          {$lte: [{ $toInt: { $arrayElemAt: [{ $split: ["$id", "-"] }, 2] } },requestedYear,],},
          {$gte: [{ $toInt: { $arrayElemAt: [{ $split: ["$id", "-"] }, 1] } },requestedYear,],},
        ],
      },
    });
    res.json({ success: true, cves: cvesInYear });
  } catch (error) {
    console.error("Error fetching CVE data by year:", error.message);
    res.status(500).json({ success: false, message: "Internal server error" });
  }
};
```

4. getCvesByScore Function:

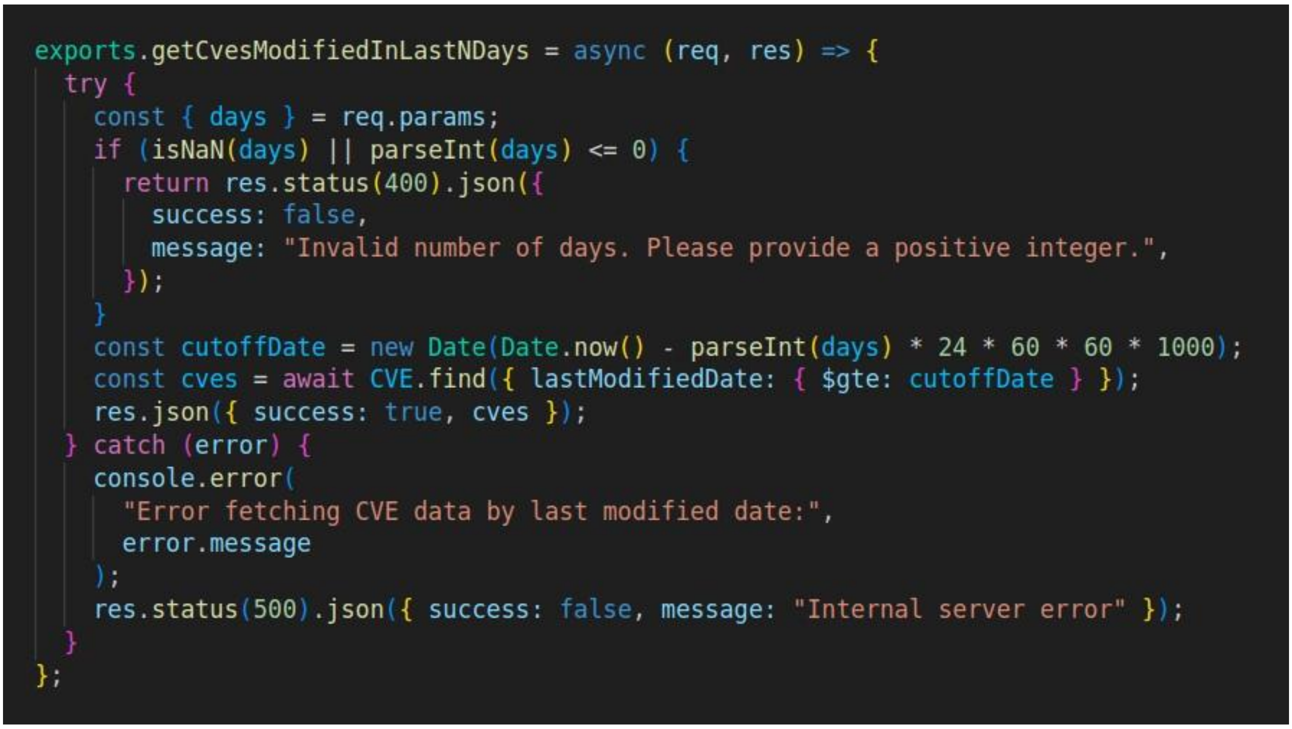
getCvesByScore function retrieves CVE details based on a specified CVSS score by extracting the score parameter from the request URL and validating its range. A MongoDB query is then constructed to find CVE documents where the baseScore matches the specified score. The matching CVE documents are sent back as part of a JSON response along with a success indicator. Error handling is implemented to ensure that appropriate error messages are returned for invalid score inputs or any errors encountered during the process.



```
exports.getCvesByScore = async (req, res) => {
  try {
    const { score } = req.params;
    if (isNaN(score) || parseFloat(score) < 0 || parseFloat(score) > 10) {
      return res.status(400).json({
        success: false,
        message:
          "Invalid CVSS score. Please provide a valid score between 0 and 10.",
      });
    }
    const query = { "metrics.cvssV2.cvssData.baseScore": parseFloat(score) };
    const cves = await CVE.find(query);
    res.json({ success: true, cves });
  } catch (error) {
    console.error("Error fetching CVE data by score:", error.message);
    res.status(500).json({ success: false, message: "Internal server error" });
  }
};
```

5. getCvesModifiedInLastNDays Function

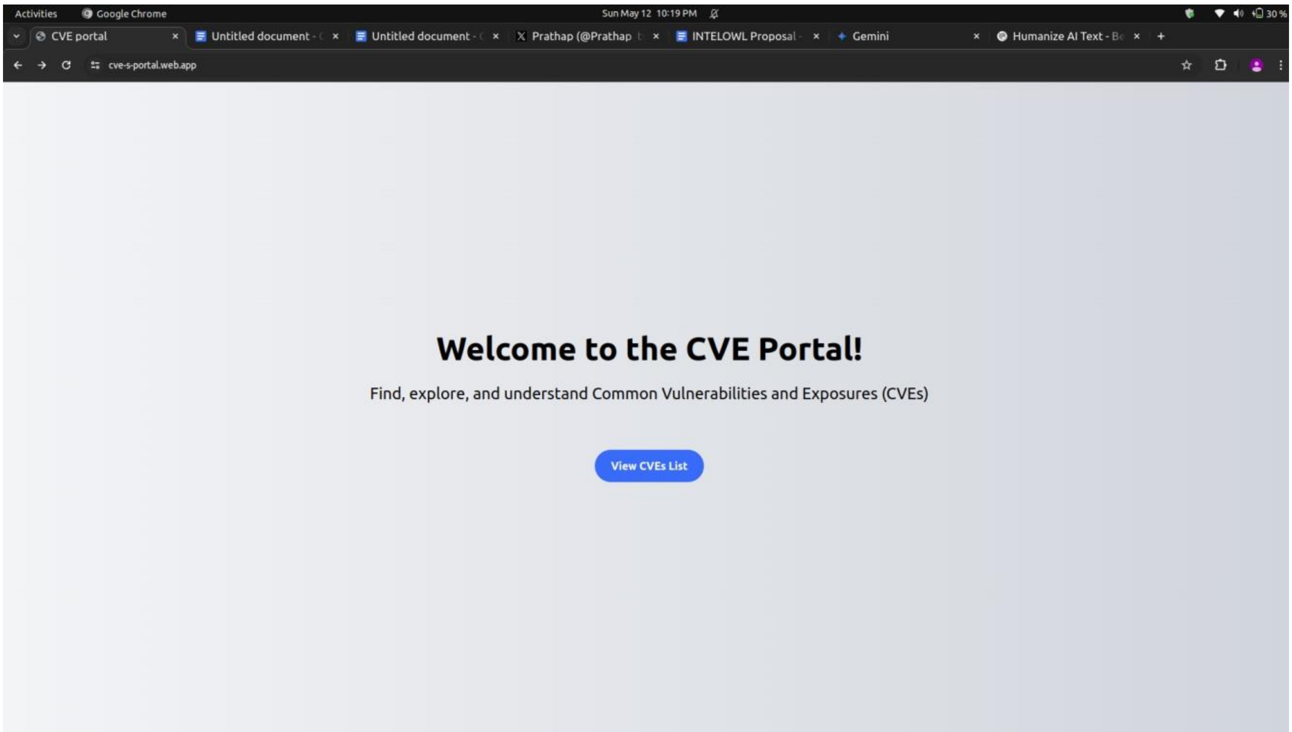
This function retrieves CVE details modified within the last N days by extracting the days parameter from the request URL and validating it as a positive integer. A cutoff date is then calculated based on the number of days provided, and a MongoDB query is constructed to find CVE documents modified on or after that date. The matching CVE documents are sent back as part of a JSON response along with a success indicator. Proper error handling is implemented to address invalid input for the number of days or any potential errors that might occur during the process.



```
exports.getCvesModifiedInLastNDays = async (req, res) => {
  try {
    const { days } = req.params;
    if (isNaN(days) || parseInt(days) <= 0) {
      return res.status(400).json({
        success: false,
        message: "Invalid number of days. Please provide a positive integer.",
      });
    }
    const cutoffDate = new Date(Date.now() - parseInt(days) * 24 * 60 * 60 * 1000);
    const cves = await CVE.find({ lastModifiedDate: { $gte: cutoffDate } });
    res.json({ success: true, cves });
  } catch (error) {
    console.error(
      "Error fetching CVE data by last modified date:",
      error.message
    );
    res.status(500).json({ success: false, message: "Internal server error" });
  }
};
```

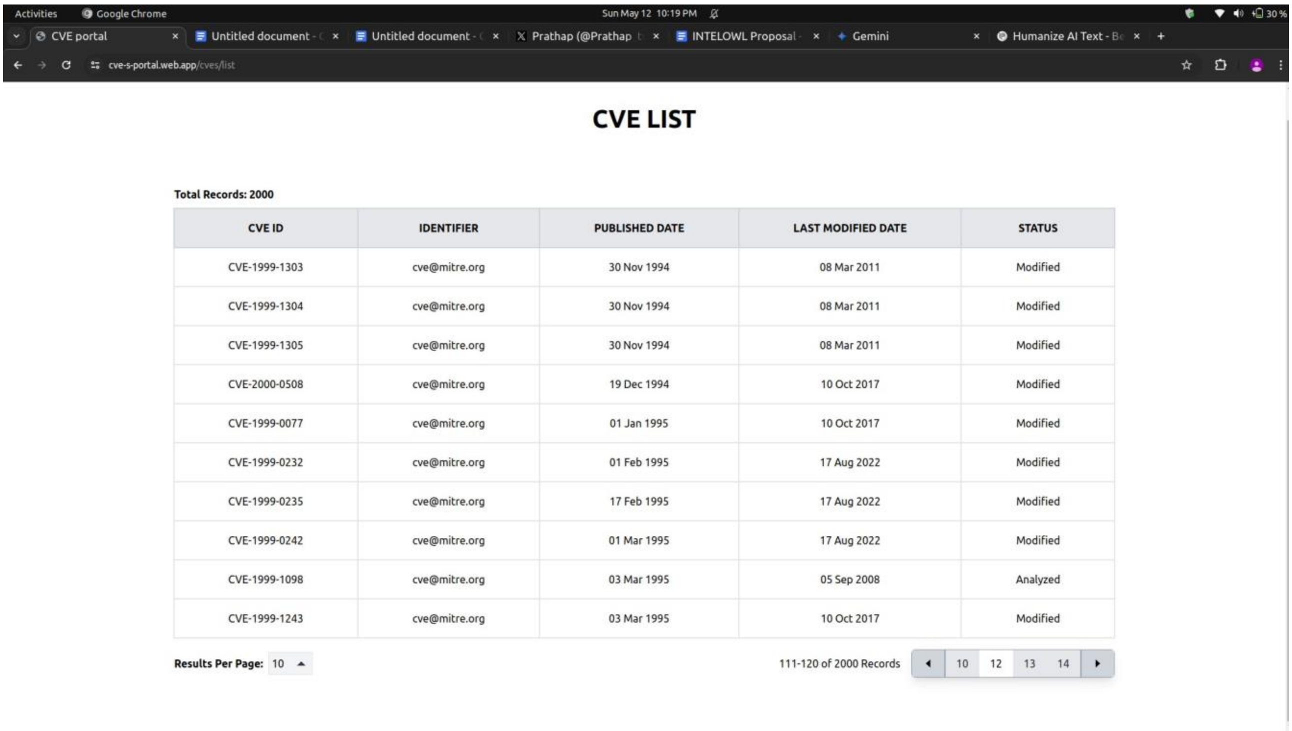
UI Visualization:

Retrieve CVE data from the developed APIs and present it in an interactive UI format for users to explore.



Welcome to the CVE Portal!

Find, explore, and understand Common Vulnerabilities and Exposures (CVEs)



CVE LIST

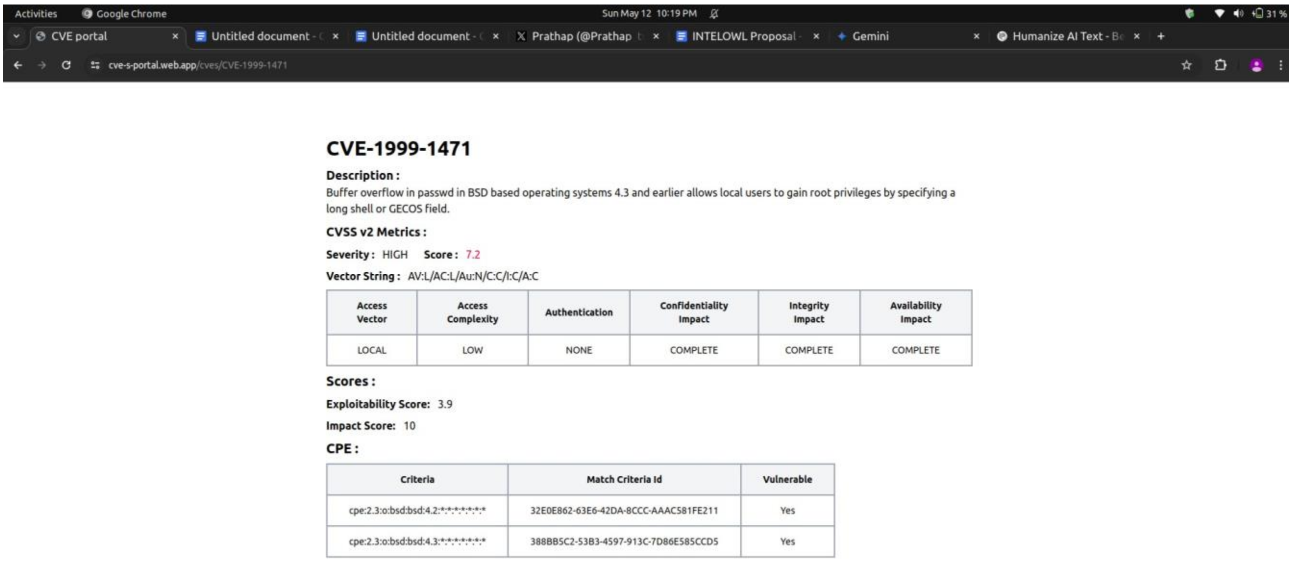
Total Records: 2000

CVE ID	IDENTIFIER	PUBLISHED DATE	LAST MODIFIED DATE	STATUS
CVE-1999-1303	cve@mitre.org	30 Nov 1994	08 Mar 2011	Modified
CVE-1999-1304	cve@mitre.org	30 Nov 1994	08 Mar 2011	Modified
CVE-1999-1305	cve@mitre.org	30 Nov 1994	08 Mar 2011	Modified
CVE-2000-0508	cve@mitre.org	19 Dec 1994	10 Oct 2017	Modified
CVE-1999-0077	cve@mitre.org	01 Jan 1995	10 Oct 2017	Modified
CVE-1999-0232	cve@mitre.org	01 Feb 1995	17 Aug 2022	Modified
CVE-1999-0235	cve@mitre.org	17 Feb 1995	17 Aug 2022	Modified
CVE-1999-0242	cve@mitre.org	01 Mar 1995	17 Aug 2022	Modified
CVE-1999-1098	cve@mitre.org	03 Mar 1995	05 Sep 2008	Analyzed
CVE-1999-1243	cve@mitre.org	03 Mar 1995	10 Oct 2017	Modified

Results Per Page: 10 ▲

111-120 of 2000 Records ◀ 10 12 13 14 ▶

When a row is clicked, I use the useNavigate hook in React to navigate to the second page. I pass the URL with the cveId in the next page. In the subsequent page, I extract the cveId from the URL and make an API call to fetch the details of the clicked CVE item.



CVE-1999-1471

Description :
Buffer overflow in passwd in BSD based operating systems 4.3 and earlier allows local users to gain root privileges by specifying a long shell or GECOS field.

CVSS v2 Metrics :

Severity : HIGH **Score :** 7.2

Vector String : AV:L/AC:L/Au:N/C:C/I:C/A:C

Access Vector	Access Complexity	Authentication	Confidentiality Impact	Integrity Impact	Availability Impact
LOCAL	LOW	NONE	COMPLETE	COMPLETE	COMPLETE

Scores :

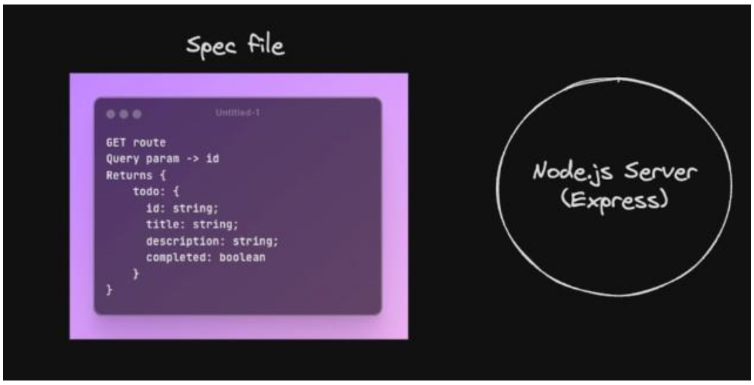
Exploitability Score: 3.9

Impact Score: 10

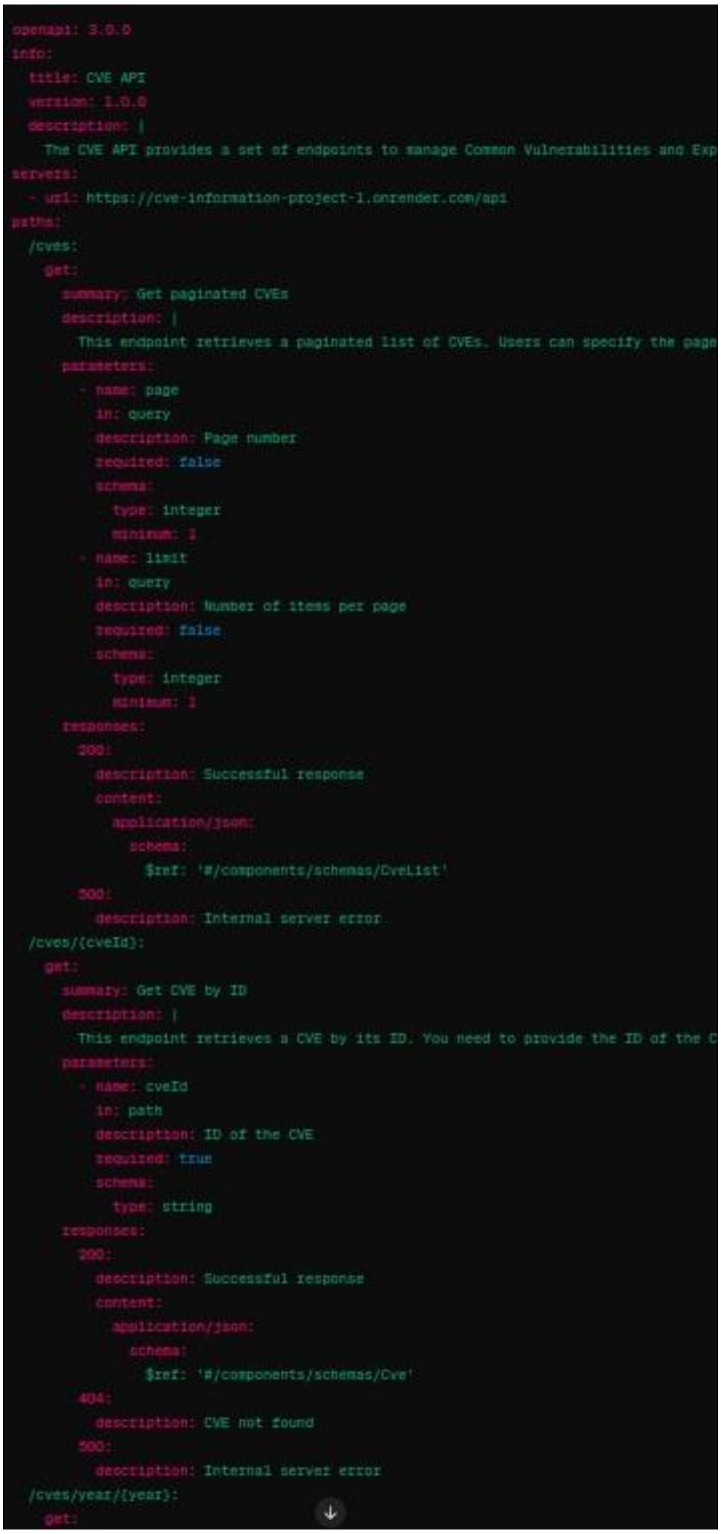
CPE :

Criteria	Match Criteria Id	Vulnerable
cpe:2.3:o:bsd:bsd:4.2:*:*:*:*:*:*:*	32E0E862-63E6-42DA-8CCC-AAAC581FE211	Yes
cpe:2.3:o:bsd:bsd:4.3:*:*:*:*:*:*:*	388BB5C2-53B3-4597-913C-7D86E585CCD5	Yes

I created an OpenAPI specification for each of my APIs. With the help of this OpenAPI specification file, I generated the Swagger UI document for my API. This allows the API consumer to easily communicate with the API without exposing my backend code. The document is wellwritten and explains every API properly.

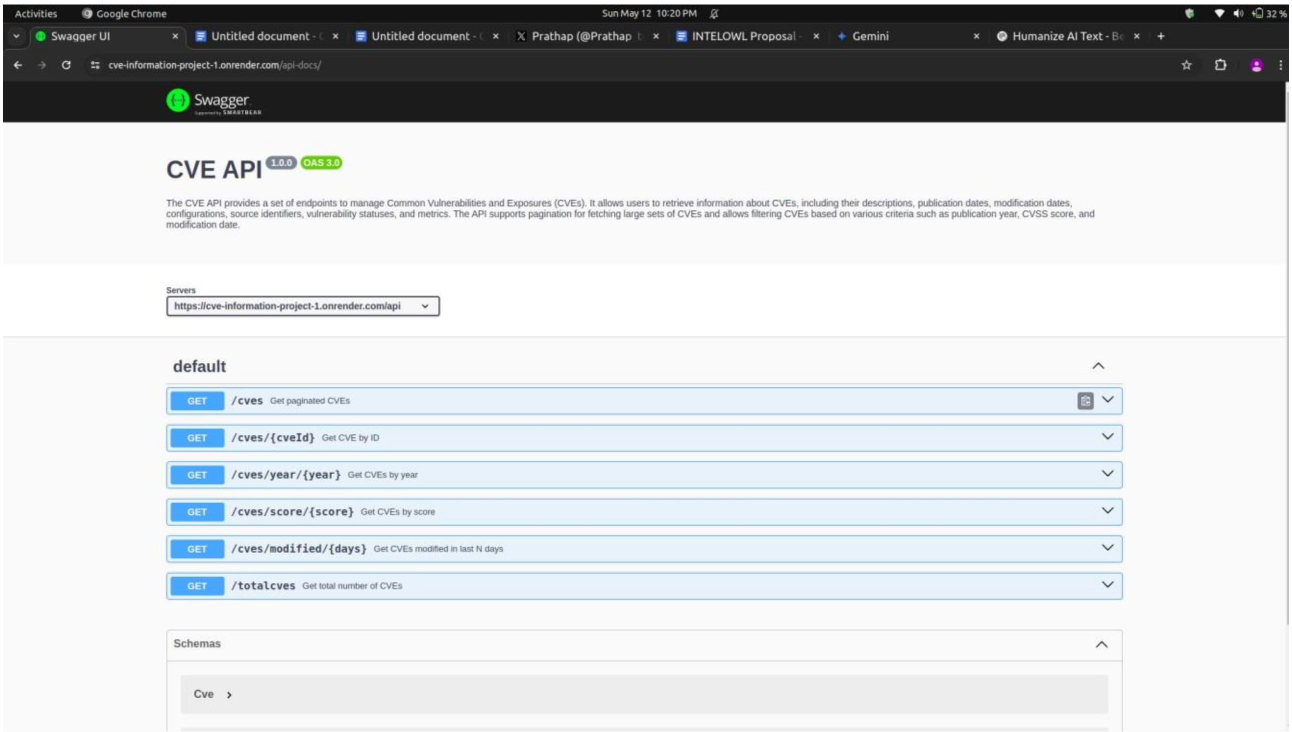


```
GET route
Query param -> id
Returns {
    todo: {
      id: string;
      title: string;
      description: string;
      completed: boolean
    }
}
```



```yaml
openapi: 3.0.0
info:
  title: CVE API
  version: 1.0.0
  description: |
    The CVE API provides a set of endpoints to manage Common Vulnerabilities and Exp
servers:
  - url: https://cve-information-project-1.onrender.com/api
paths:
  /cves:
    get:
      summary: Get paginated CVEs
      description: |
        This endpoint retrieves a paginated list of CVEs. Users can specify the page
      parameters:
        - name: page
          in: query
          description: Page number
          required: false
          schema:
            type: integer
            minimum: 1
        - name: limit
          in: query
          description: Number of items per page
          required: false
          schema:
            type: integer
            minimum: 1
      responses:
        200:
          description: Successful response
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/CveList'
        500:
          description: Internal server error
  /cves/{cveId}:
    get:
      summary: Get CVE by ID
      description: |
        This endpoint retrieves a CVE by its ID. You need to provide the ID of the C
      parameters:
        - name: cveId
          in: path
          description: ID of the CVE
          required: true
          schema:
            type: string
      responses:
        200:
          description: Successful response
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Cve'
        404:
          description: CVE not found
        500:
          description: Internal server error
  /cves/year/{year}:
    get:
```

I have a very long single YAML file where I've listed all my routes. This allows me to generate the API documentation using Swagger-UI-Express.



CVE API 1.0.0 OAS 3.0

The CVE API provides a set of endpoints to manage Common Vulnerabilities and Exposures (CVEs). It allows users to retrieve information about CVEs, including their descriptions, publication dates, modification dates, configurations, source identifiers, vulnerability statuses, and metrics. The API supports pagination for fetching large sets of CVEs and allows filtering CVEs based on various criteria such as publication year, CVSS score, and modification date.

Servers
https://cve-information-project-1.onrender.com/api

default

GET `/cves` Get paginated CVEs

GET `/cves/{cveId}` Get CVE by ID

GET `/cves/year/{year}` Get CVEs by year

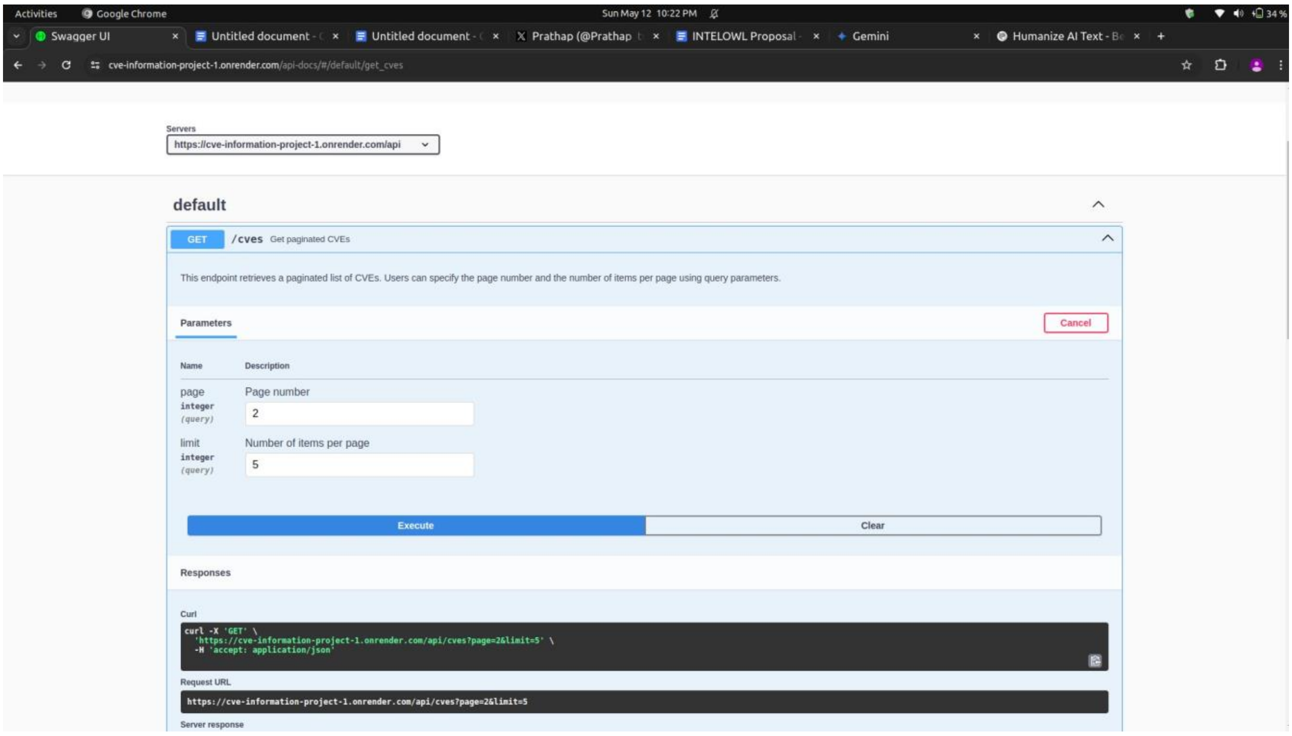
GET `/cves/score/{score}` Get CVEs by score

GET `/cves/modified/{days}` Get CVEs modified in last N days

GET `/totalcves` Get total number of CVEs

Schemas

Cve ›



Servers
https://cve-information-project-1.onrender.com/api

default

GET `/cves` Get paginated CVEs

This endpoint retrieves a paginated list of CVEs. Users can specify the page number and the number of items per page using query parameters.

Parameters Cancel

Name	Description
page integer *(query)*	Page number 2
limit integer *(query)*	Number of items per page 5

Execute Clear

Responses

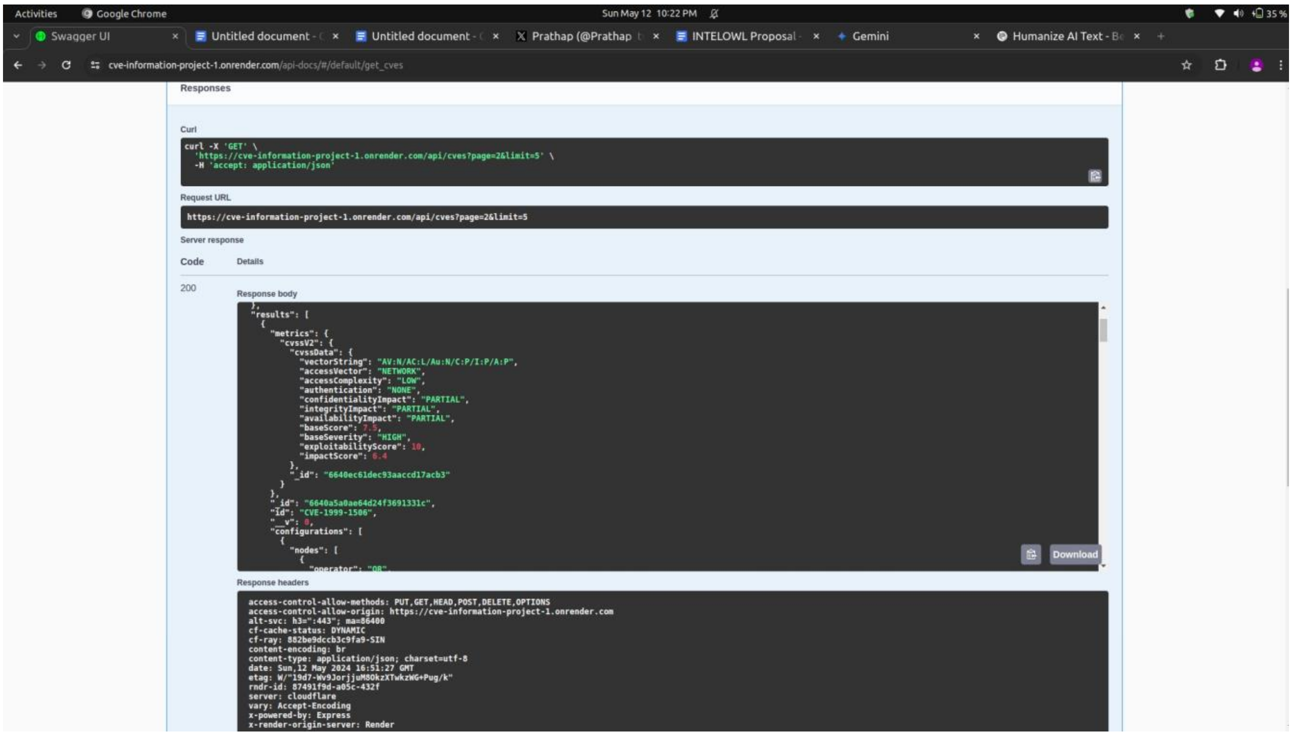
Curl

```
curl -X 'GET' \
  'https://cve-information-project-1.onrender.com/api/cves?page=2&limit=5' \
  -H 'accept: application/json'
```

Request URL

```
https://cve-information-project-1.onrender.com/api/cves?page=2&limit=5
```

Server response



```
curl -X 'GET' \
  'https://cve-information-project-1.onrender.com/api/cves?page=2&limit=5' \
  -H 'accept: application/json'
```

Request URL

```
https://cve-information-project-1.onrender.com/api/cves?page=2&limit=5
```

Server response

200

Response body

```
  },
  "results": [
    {
      "metrics": {
        "cvssV2": {
          "cvssData": {
            "vectorString": "AV:N/AC:L/Au:N/C:P/I:P/A:P",
            "accessVector": "NETWORK",
            "accessComplexity": "LOW",
            "authentication": "NONE",
            "confidentialityImpact": "PARTIAL",
            "integrityImpact": "PARTIAL",
            "availabilityImpact": "PARTIAL",
            "baseScore": 7.5,
            "baseSeverity": "HIGH",
            "exploitabilityScore": 10,
            "impactScore": 6.4
          },
          "_id": "6640ec61dec93aaccd17acb3"
        }
      },
      "_id": "6640a5a0ae64d24f3691331c",
      "id": "CVE-1999-1506",
      "__v": 0,
      "configurations": [
        {
          "nodes": [
            {
              "operator": "OR",
```

Response headers

```
access-control-allow-methods: PUT,GET,HEAD,POST,DELETE,OPTIONS
access-control-allow-origin: https://cve-information-project-1.onrender.com
alt-svc: h3=":443"; ma=86400
cf-cache-status: DYNAMIC
cf-ray: 882be9dccb3c9fa9-SIN
content-encoding: br
content-type: application/json; charset=utf-8
date: Sun,12 May 2024 16:51:27 GMT
etag: W/"19d7-Wv9JorjjuM8OkzXTwkzWG+Pug/k"
rndr-id: 87491f9d-a05c-432f
server: cloudflare
vary: Accept-Encoding
x-powered-by: Express
x-render-origin-server: Render
```

Finally, I deployed both the backend server and the API documentation website. This ensures that the backend server is accessible to users and can handle incoming requests effectively. Additionally, the API documentation website allows users to explore and understand the available APIs without needing to access the backend code directly. It provides clear explanations of each API endpoint and their respective functionalities.

Furthermore, I deployed the frontend to Firebase, which provides a reliable hosting solution for web applications.