CPE Data Collection and API Development

Objective:

This assignment is designed to evaluate your ability to work with XML data, interact with databases, and build a RESTful API to expose the data. You will retrieve data from a public XML feed, extract specific information, store it in a database, and create an API to interact with the data.

Requirements:

1. **Retrieve and Parse XML Data**:

    - Download the CPE dictionary XML feed from this link.
    - Parse the XML to extract the following details for each CPE entry:
        1. **CPE Title**: The title of the CPE.
        2. **CPE 22 URI**: The URI for CPE version 2.2.
        3. **CPE 23 URI**: The URI for CPE version 2.3.
        4. **Reference Links**: All associated reference URLs for the CPE.
        5. **CPE 22/23 Deprecation Date**: The date when the CPE was deprecated (if available). Both CPE 22 and CPE 23 might have different deprecation dates, so store them separately.
2. **Database Design and Storage**:

    - Create a database to store the extracted data. You can use SQL or NoSQL (based on your preference), but ensure the database schema supports the following columns:
        1. `cpe_title` (VARCHAR)
        2. `cpe_22_uri` (TEXT)
        3. `cpe_23_uri` (TEXT)
        4. `reference_links` (TEXT[], JSON, or another appropriate format)
        5. `cpe_22_deprecation_date` (DATE)
        6. `cpe_23_deprecation_date` (DATE)
    - Ensure that the database schema is designed to store and query this data efficiently. Provide SQL scripts or instructions for setting up the database.
3. **API Development**:

    - Build a RESTful API to interact with the stored data. Use any web framework you are comfortable with (e.g., Flask, Express.js, Django).
    - **Endpoints**:
        1. **Get All CPEs (Paginated)**:

            - **URL**: `/api/cpes`
            - **Method**: `GET`
            - **Query Parameters**:

        - `page`: The page number (default: 1).
        - `limit`: The number of items per page (default: 10).
    - **Response**:
        - List all CPE entries, ordered by the `id` (primary key).
        - Paginate results to handle large datasets.
2. **Example Request**:

```
Unset
GET /api/cpes?page=1&limit=10
```

3. **Example Response**:

```
Unset
{
  "page": 1,
  "limit": 10,
  "total": 100,
  "data": [
    {
      "id": 1,
      "cpe_title": "Example CPE 1",
      "cpe_22_uri": "uri1",
      "cpe_23_uri": "uri1_23",
      "reference_links": ["link1", "link2"],
      "cpe_22_deprecation_date": "2023-05-01",
      "cpe_23_deprecation_date": "2024-06-01"
    }
  ]
}
```

4. **Search CPEs**:

    - **URL**: `/api/cpes/search`
    - **Method**: `GET`
    - **Query Parameters**:
        - `cpe_title`: (Optional) Search by title.
        - `cpe_22_uri`: (Optional) Search by CPE 22 URI.

        - `cpe_23_uri`: (Optional) Search by CPE 23 URI.
        - `deprecation_date`: (Optional) Provide a date, and return all CPEs deprecated prior to that date (either `cpe_22_deprecation_date` or `cpe_23_deprecation_date`).
5. **Example Request**:

```
Unset
GET
/api/cpes/search?cpe_title=example&deprecation_date=2024-01-01
```

6. **Example Response**:

```
Unset
{
  "data": [
    {
      "id": 3,
      "cpe_title": "Example CPE 3",
      "cpe_22_uri": "uri3",
      "cpe_23_uri": "uri3_23",
      "reference_links": ["link1", "link2"],
      "cpe_22_deprecation_date": "2023-12-15",
      "cpe_23_deprecation_date": "2024-02-10"
    }
  ]
}
```

Submission Instructions:

1. **Code**:

    - Submit the source code for parsing the XML data, database setup, and the API code preferably as a git repo.
    - Include any necessary documentation for setting up and running the application, including any dependencies or libraries used.

2. **Database**:

    - Provide the database schema (SQL scripts or setup instructions).
    - If using a hosted database, provide connection details or instructions for setting it up.
3. **API Testing**:

    - Include example requests and responses for the implemented API endpoints.
    - Optionally, write unit tests to validate the functionality of your endpoints (e.g., using a testing framework like PyTest for Python or Mocha for Node.js).

Evaluation Criteria:

- **Correctness**: Does the implementation retrieve and store the correct data from the XML file? Do the API endpoints work as expected?
- **Code Quality**: Is the code clean, readable, and well-structured? Are best practices followed?
- **Database Design**: Is the database schema well-designed and efficient? Does it properly support the necessary queries?
- **API Design**: Are the API endpoints well-designed and functional? Are they well-documented and easy to understand?

Frontend (UI):

Objective:

This assignment is designed to evaluate your ability to interact with RESTful API, present the data from response with a paginated grid view where users can filter, paginate, sort, search the data whatever they are looking for.

Requirements:

1. Call RESTful API to fetch all CPEs information and render it in a table with below mentioned columns.
    a. Title - **Truncated if width of the column is less than data**
    b. URL_22
    c. URL_23
    d. Deprecated Date 22 - Should be formatted with **MMM DD, YYYY**
    e. Deprecated Date 23 - Should be formatted with **MMM DD, YYYY**
    f. References - list of URLs with X more - List 2 links one by one with truncated and tooltip, clicking on X more should open a small popover to render all links with truncated. **All links are targeted to the new tab.**

2. Add Field (Cell) level filter and use /search API to retrieve data based on user applied search in respective fields.
3. Also, handle pagination and results per page can be customizable by users starting from 15 to 50.
4. If no results are found, show a fallback screen with a message (Nice to Have).
5. If no data is found, show a fallback screen with a message (Nice to Have).