LexML API Integration Instructions for Search Tool

This document outlines the steps for integrating the LexML API into a search tool, with a defined fallback mechanism. The primary method for searching will be via the LexML API. In case of API failure, a web scraper will be used as a secondary method. Only after these two attempts should a fallback method be considered.

1. LexML API as Primary Search Method

The LexML project provides an Open Data API that can be used to retrieve metadata for various content types. The main endpoint for content metadata is `https://projeto.lexml.gov.br/apidata/content`.

1.1. API Endpoint and Request Structure

To search for content, you will interact with the `/apidata/content` endpoint. While the provided documentation lists various content types (News, Image, File, Folder, Document, Event) and their respective Dublin Core metadata, it does not explicitly detail query parameters for searching within these content types. Based on the structure, it is likely that specific content types or search terms would be appended to the base URL or passed as query parameters. Further investigation or experimentation with the API would be required to determine the exact search query syntax.

Example (Hypothetical) Request Structure:

```
GET https://projeto.lexml.gov.br/apidata/content?
type=Document&query=your_search_term
```

Or, if searching within a specific content type:

```
GET https://projeto.lexml.gov.br/apidata/content/Document?
query=your_search_term
```

Note: The actual query parameters and their values (e.g., `type`, `query`) are inferred and need to be confirmed through testing or by finding more detailed API documentation (which was not readily available during the initial research).

1.2. Expected API Response

The API is expected to return data in JSON format. Each content item will likely contain Dublin Core metadata fields such as:

- `identifier`: The content's address on the website.
- `description`: Dublin Core element description - resource details.
- `title`: Dublin Core element title - resource name.
- `creator`: Dublin Core element creator - resource author.
- `uri`: Permanent URI of the content.
- `uid`: Unique identifier of the content.

Example (Hypothetical) JSON Response Structure:

```
[
  {
    "identifier": "https://projeto.lexml.gov.br/documentos/doc1",
    "description": "Description of document 1",
    "title": "Document Title 1",
    "creator": "Author 1",
    "uri": "urn:lex:br:federal:document:2023-01-01;doc1",
    "uid": "unique_id_1"
  },
  {
    "identifier": "https://projeto.lexml.gov.br/noticias/news1",
    "description": "Description of news item 1",
    "title": "News Title 1",
    "creator": "Author 2",
    "uri": "urn:lex:br:federal:news:2023-02-01;news1",
    "uid": "unique_id_2"
  }
]
```

1.3. Implementation Steps for API Integration

1. **Construct API Request:** Formulate the HTTP GET request to the LexML API endpoint (`https://projeto.lexml.gov.br/apidata/content`) with appropriate query parameters for the search term and desired content type (if applicable).
2. **Execute Request:** Use an HTTP client library in your chosen programming language (e.g., `requests` in Python, `fetch` in JavaScript) to send the request.
3. **Handle Response:** Parse the JSON response. Iterate through the returned array of objects and extract the `title`, `description`, `uri`, and `identifier` for each relevant content item.

4. **Display Results:** Present the extracted information in your search tool's interface.

2. Fallback to Web Scraper (if API Fails)

If the LexML API request fails (e.g., due to network issues, API downtime, or invalid request), the search tool should automatically attempt to perform the search using a web scraper on the LexML website (`https://projeto.lexml.gov.br`).

2.1. Web Scraper Implementation Considerations

1. **Target URL:** The search functionality on the LexML website can be accessed via the search bar. After entering a search term in the search bar (index 12 on the homepage) and pressing Enter, the URL typically changes to `https://projeto.lexml.gov.br/@@search?SearchableText=your_search_term`.
2. **HTML Parsing:** Use a web scraping library (e.g., Beautiful Soup in Python, Cheerio in Node.js) to parse the HTML content of the search results page.
3. **Extract Information:** Identify the HTML elements that contain the search results (e.g., titles, descriptions, links) and extract the relevant text and URLs.
4. **Rate Limiting and Politeness:** Implement appropriate delays between requests to avoid overwhelming the server and to comply with `robots.txt` rules (if any).
5. **Error Handling:** Implement robust error handling for HTTP errors, parsing errors, and changes in website structure.

2.2. Implementation Steps for Web Scraper Fallback

1. **Check API Status:** After an API request, check the HTTP status code. If it indicates an error (e.g., 4xx, 5xx), proceed to the web scraping step.
2. **Construct Scraper URL:** Build the search URL for the LexML website using the user's search query.
3. **Execute Scraper Request:** Send an HTTP GET request to the scraper URL.
4. **Parse HTML:** Parse the HTML response to extract search results.
5. **Display Results:** Present the extracted information in your search tool's interface.

3. Final Fallback Method

Only after both the LexML API and the web scraper attempts have failed should the application resort to a final fallback method. This final fallback method should be defined based on the specific requirements of your application and could include:

- **Displaying an error message:** Informing the user that the search functionality is currently unavailable.
- **Using a cached version of data:** If your application maintains a local cache of previously retrieved LexML data, this could be used as a last resort.
- **Suggesting manual search:** Directing the user to manually search on the LexML website.

Crucially, the application should clearly communicate to the user which method is being used (API, web scraper, or final fallback) and if any method has failed. For instance, if the API fails, a message like "LexML API is currently unavailable. Attempting search via web scraper..." could be displayed. If the web scraper also fails, a message indicating complete search failure should be shown.