Task: Building a Contextual News Data Retrieval System

Objective:

Design and implement a backend system that can fetch and organize news articles from a data source, simulating different API functionalities, and enrich these articles with LLM-generated insights. This system should demonstrate the ability to:

- Understand the nuances of a user's query, including their location.
- Utilize an LLM to identify key entities, concepts, and the user's intent to refine data retrieval.
- Retrieve data from a pre-defined data source, simulating different API endpoints.
- Process, rank, and enrich the results for relevance and comprehensiveness, considering the user's location.
- Return a JSON response with the most relevant news articles.

Technical Requirements:

1. **LLM Interaction:**
    - Use a publicly available LLM API (e.g., OpenAI, Google Cloud Language API) to process the user's news query.
    - Extract relevant entities (e.g., people, organizations, locations, events) and key concepts from the query.
    - Determine the user's intent to select the most appropriate data retrieval strategy (simulated API endpoint).
    - Example:
        - Input Query: "Latest developments in the Elon Musk Twitter acquisition near Palo Alto"
        - Expected LLM Output:
            - Entities: "Elon Musk", "Twitter", "Palo Alto"
            - Intent: "nearby"
        - Input Query: "Top technology news from the New York Times"
        - Expected LLM Output:
            - Entities: "New York Times"
            - Intent: "category", "source"
    - **While the application can accept the entities, concepts, and intent directly through the API, we prefer that you showcase your skills by using an LLM for this purpose.**

2. **Data Retrieval:**

    - The news article data will be given as a file with mail. You will fetch the data and store it in a database of your choice (SQL or NoSQL)

      .

3. **Data Format**

    - The data in the news_data file will be organized in JSON files. Each file will contain an array of news articles. Here's the format for each article:


```
{
  "id": "b1793e11-85f1-47f4-b836-ddc21dd8991e",
  "title": "Paris police chase over running of red light ends in pileup | DW News",
  "description": "Paris police chase over running of red light ends in pileup | DW News...",
  "url": "https://www.youtube.com/@dwnews",
  "publication_date": "2025-03-24T11:08:11",
  "source_name": "DW",
  "category": [
    "General"
  ],
  "relevance_score": 0.51,
  "latitude": 21.754075,
  "longitude": 80.560129
}
```


4. **API EndPoints**

    - **"category"**: Retrieve articles from a specific category (e.g., "Technology", "Business", "Sports"). The category will be provided as part of the user query.
    - **"score"**: Retrieve articles based on a relevance score. The data will contain a "relevance_score" field. Retrieve articles with a score above a threshold (e.g., 0.7).
    - **"search"**: Retrieve articles based on a search query. Perform a text search within the article title and description. The search query will be derived from the user's input.
    - **"source"**: Retrieve articles from specific sources (e.g., "New York Times", "Reuters"). The source will be provided as part of the user query.
    - **"nearby"**: Retrieve articles published within a specified radius (e.g., 10km) of a given location (latitude and longitude). The user's location will be included in the query.

Bonus API – Trending News by Location:

Trending News Feed (Location-Based)

Implement an additional API endpoint that returns a “Trending News Feed” tailored to a user's location. This should simulate a feed similar to “what’s trending near me” based on recent user engagement with articles.

Key Requirements:

- **“Simulate a Stream of User Events”**
  Introduce a simulated stream of user activity events (e.g., views, clicks) that tie users to specific articles and locations.
  The candidate is expected to design the data model and event structure.
- **“Trending Logic”**
  Process user interaction data to compute a trending score for articles.
  Consider factors such as:
  - Volume and type of interactions.
  - Recency of interactions.
  - Geographical relevance (proximity to user's location).
- **Trending Feed Endpoint:**
  - Endpoint: GET /trending
  - Query Parameters: lat, lon, limit

  The response should return top trending articles within the predefined/dynamic location radius, ranked by computed trending score.

Caching Requirement:
Implement caching for trending feeds by location to improve response times. You may use a geospatial segmentation approach (e.g., segmenting locations into clusters).

5. **Data Processing, Ranking, and Enrichment:**

   - Process the articles retrieved from the database based on the simulated API endpoint selected by the LLM or through user input.
   - Rank the articles for relevance to the user's query.
     - For "category", and "source" rank by publication date (most recent first).
     - For "score", rank by relevance_score (highest first).
     - For "search", rank by a combination of relevance_score and text matching score (how well the title/description matches the search query).
     - For "nearby", rank by distance from the user's location (closest first). You may use the Haversine formula or another appropriate method to calculate the distance between two points given their latitude and longitude.
   - Enrich the news articles with data from the LLM:
     - Include a summary of the article's content (generated by the LLM).

6. **Output:**

    - Return the top 5 most relevant news articles in a JSON format. The structure should be consistent across all simulated API endpoints. Each article should include:
        - Title
        - Description
        - URL
        - Publication Date
        - Source Name
        - Category
        - Relevance Score
        - LLM-Generated Summary

Example Output:


```
{
  "articles": [
    {
      "title": "Article Title 1",
      "description": "Article Description 1",
      "url": "https://www.example.com/article1",
      "publication_date": "2024-04-28T10:00:00Z",
      "source_name": "Example News",
      "category": "Technology",
      "relevance_score": 0.92,
      "llm_summary": "This article discusses the latest developments in...",
      "latitude": 37.4220,
      "longitude": -122.0840
    },
```

```
{
   "title": "Article Title 2",
   "description": "Article Description 2",
   "url": "https://www.example.com/article2",
   "publication_date": "2024-04-27T11:00:00Z",
   "source_name": "Another News Source",,
... (3 more articles)]}
```

Additional Guidance:

- The above task is designed to be flexible, allowing you to demonstrate your skills at different levels of depth. While completing the entire task will showcase a strong understanding of your backend skills, even completing the initial stages can provide valuable insights into your abilities. So feel free to complete the task at whatever stage you can do best with minimum apis to load data from file to one your preferred DB and create a minimum of 2-3 api endpoints to fetch the data.
- **Database Setup:** You'll need to have a database server set up and running (either a local instance or a cloud-managed database).
- **Dependencies:** Make sure you have the necessary libraries installed.

API Design Considerations:

- **RESTful Principles:** Design the API following RESTful principles. Use appropriate HTTP methods (e.g., GET, POST) and status codes.
- **Endpoint Structure:** Consider a base URL (e.g., /api/news) and then specific endpoints for each simulated API function (e.g., /api/news/category, /api/news/search, /api/news/nearby).
- **Versioning:** Include API versioning in the URL (e.g., /api/v1/news) to allow for future updates.

Input Handling:

- **Query Parameters:** Use query parameters to pass user input to the API (e.g., GET /api/news/search?query=Elon+Musk&location=Palo+Alto).
- **Location Input:** For the "nearby" endpoint, expect latitude and longitude parameters (e.g., GET /api/news/nearby?lat=37.4220&lon=-122.0840&radius=10).

- **Error Handling:** Implement robust error handling. Return appropriate HTTP error codes and informative error messages in the response body.

Output Formatting:

- **JSON Format:** Return the news articles in JSON format, as specified in the task description.
- **Consistent Structure:** Maintain a consistent output structure across all API endpoints.
- **Metadata:** Consider including metadata in the response, such as the total number of results, the page number, and the query that was used.