Documentation for the News RSS Feed Application

Objective

The primary objective of this application is to:

1. **Collect news articles** from various RSS feeds.
2. **Classify articles** into predefined categories such as "Terrorism", "Natural Disasters", etc., using basic Natural Language Processing (NLP).
3. **Store the articles** in a PostgreSQL database while avoiding duplicate entries.
4. Use **Celery** to process tasks asynchronously and **Redis** as a message broker for distributed task management.

System Architecture

This application consists of several components working together:

- **RSS Feed Parsing**: Fetches and extracts news articles from the provided RSS feed URLs using the `feedparser` library.
- **Task Queue (Celery)**: Manages background processing of articles, distributing tasks among workers using Redis as the message broker.
- **Natural Language Processing (spaCy)**: Classifies each article into one of several categories based on its content.
- **Database (PostgreSQL)**: Stores the extracted and classified articles while ensuring no duplicates are added.

Key Components and Design Choices

1. RSS Feed Parsing

- The **`feedparser`** library was chosen for its simplicity and reliability in handling RSS feeds.
- The function `parse_rss_feed(feed_url)` extracts articles from the provided list of RSS feeds.
    - Each RSS feed entry includes the **title**, **summary**, **publication date**, and **source URL**.
    - If the feed cannot be parsed due to network or format issues, the function handles the error gracefully by logging it and skipping to the next feed.

2. Database

- The database used is **PostgreSQL**, a robust relational database system.
- The **SQLAlchemy ORM** was used to interact with the database, which abstracts away the need to write raw SQL queries.
    - The `NewsArticle` table stores the following fields: `id`, `title`, `summary`, `pub_date`, `source_url`, and `category`.
    - The `source_url` is a **unique constraint** to prevent duplicate articles from being inserted into the database.

Database Schema

```
sql
Copy code
CREATE TABLE news_articles (
    id SERIAL PRIMARY KEY,
    title TEXT NOT NULL,
    summary TEXT NOT NULL,
    pub_date TIMESTAMP NOT NULL,
    source_url TEXT UNIQUE NOT NULL,
    category TEXT NOT NULL
);
```

3. Celery (Task Queue and Asynchronous Processing)

- **Celery** was chosen to handle the background task of processing articles asynchronously.
    - Articles are fetched and sent to the Celery task queue with `process_article.delay(article_data)`.
    - Workers then process each article in the background, classifying and storing them.
- **Redis** acts as the message broker for Celery. It manages the queue of tasks sent by the main application, and Celery workers pick tasks from this queue.

4. NLP-Based Classification (spaCy)

- The **spaCy** NLP library was used to perform basic content classification. The `classify_article(content)` function scans the article's content for specific keywords (like "terror", "earthquake", etc.) to assign categories.
- Although this is a keyword-based approach, the use of spaCy allows for easy expansion into more sophisticated models such as topic modeling or sentiment analysis in the future.

5. Logging and Error Handling

- A centralized logging system is implemented using Python's **`logging`** module.

    - Logging captures major events, including successful parsing, article processing, and errors (e.g., duplicate articles, parsing failures).
    - Errors are handled gracefully at every stage: RSS parsing, task processing, and database insertion.
- The **retry mechanism** in Celery is employed to reattempt task execution in case of transient errors (e.g., network issues or temporary database unavailability).

Application Workflow

1. **Main Application**:
    - The `process_feeds()` function iterates over a list of RSS feeds, parses each one, and extracts news articles.
    - Each article is then sent to the Celery task queue for background processing using the `.delay()` method.
2. **Celery Workers**:
    - Celery workers execute the `process_article` task. Each worker processes articles asynchronously by:
        - Classifying the article based on its content.
        - Saving the classified article in the PostgreSQL database.
    - If the article's `source_url` is already in the database, it is skipped as a duplicate, and a warning is logged.

Testing and Monitoring

- **Testing**:
    - The application was tested using a variety of RSS feeds. Special attention was given to handling malformed data, duplicate articles, and network interruptions.
- **Monitoring**:
    - **Flower** can be added to the system for monitoring Celery workers in real time. It provides a web interface that allows you to monitor running, completed, and failed tasks.

Potential Improvements

- **Advanced NLP Models**: Currently, classification is done through simple keyword matching. This can be extended to more sophisticated techniques like Named Entity Recognition (NER), topic modeling, or pre-trained models.

- **Feed Error Handling**: The application could retry failed feed fetches after a specified period instead of skipping them.
- **Frontend**: A web interface could be developed to display the stored articles and allow searching or filtering based on category or date.

Step-by-Step Full Code Implementation:

1. Install the Required Dependencies

Make sure you have the following libraries installed:

```
bash
Copy code
pip install feedparser SQLAlchemy celery spacy psycopg2 requests redis
python -m spacy download en_core_web_sm  # For spaCy language model
```

2. Set Up Your Database (PostgreSQL)

Create a PostgreSQL database:

```
bash
Copy code
CREATE DATABASE news_db;
```

2. Running Celery Worker

To start Celery workers (to process tasks in the background), open a terminal and run:

```
bash
Copy code
celery -A app worker --loglevel=info
```

This will start a Celery worker that listens for tasks and executes them.

3. Start Redis

Ensure that Redis is running on your machine by starting the Redis server:

```
bash
Copy code
sudo service redis-server start
```

4. Running the Main Application

Run the main application to fetch and process RSS feeds:

```
bash
Copy code
python app.py
```

Conclusion

This application demonstrates the implementation of an ETL (Extract, Transform, Load) pipeline for RSS feed processing, using Celery for distributed task management, spaCy for NLP-based classification, and SQLAlchemy for database interaction. The system is designed to be scalable and robust, with built-in error handling, logging, and retry mechanisms.