Unified Pharma Intelligence Feed

Problem Overview

Pharmaceutical companies like AstraZeneca and Pfizer publish important updates through press releases covering drug approvals, trial results, safety information, and regulatory milestones. However, this information is distributed across multiple websites, inconsistently formatted, and not structured for easy analysis or monitoring.

The problem is the absence of a unified system that can reliably collect, structure, and present this data along with key metadata (e.g., company, date, drug, indication, regulatory status).

Core Challenge

1. **Scraping** press releases of the top 5 Pharma companies (Pfizer, Merck, Novo Nordisk, Abbive and AstraZeneca) and related metadata from pharma company websites.
2. **Standardizing** the extracted content into a consistent structured format.
3. **Displaying** the information in a clear, searchable, user-friendly interface.
4. **Refreshing data periodically** to keep information current.

This challenge involves data scraping, backend and frontend

Technical Details

Data Ingestion (Scraping)

Write a Python script to scrape press releases specifically for **Jan 1, 2026 – Present**.

1. Target the **Top 5 Pharmaceutical Companies** (see placeholders below).
2. Extract the following specific fields for every release:
    a. Published Date & Year
    b. Title
    c. Category
    d. URL

Storage & Normalization

1. Normalize data from all 5 distinct sources into a single, unified format.
2. Design and implement a **Relational Database Schema** to store the cleaned data efficiently.

Search Enablement

1. Read the stored data from the database.
2. Index the data into **Elasticsearch** to enable full-text search operations.

Constraints & Stack

Parameter	Requirement
Language	Python
Target Date Range	Start of Year 2026 – Present
Database	Relational DB (PostgreSQL, MySQL, SQLite)
Search Engine	Elasticsearch
Output	Normalized Data Schema & Indexed Search

Target Data Sources (Top 5)

1. **Pfizer**: https://www.pfizer.com/newsroom/press-releases
2. **AstraZeneca**: https://www.astrazeneca.com/media-centre/press-releases.html
3. **Merck**: https://www.merck.com/media/news/
4. **Johnson & Johnson**: https://www.jnj.com/media-center/press-releases
5. **Novo Nordisk**: https://www.novonordisk.com/news-and-media/news-and-ir-materials.html

Evaluation Criteria

How will this solution be judged?

1. **Normalization Quality:** Are all the fields consistent across all 5 companies? (e.g., Is "12 Jan 26" and "January 12, 2026" stored identically?)
2. **Schema Design:** Is the relational DB schema robust and scalable?
3. **Search Functionality:** Can we successfully query the Elasticsearch index and retrieve accurate results?
4. **Code Quality:** Is the Python script modular, error-handled, and clean?

5. **Accuracy:** Does the scraped data matches with the source.
6. **Scalability:** Can new pharmaceutical sources be added with minimal code changes?

Bonus Challenge (Nice to Have)

Automated Delta Updates: Design the script to run essentially as a cron job. It should be able to run daily, detect only new or updated press releases since the last run, and ingest the "delta" without creating duplicates.

Server (Backend)

The server must be built using Flask or FastAPI and implement the following endpoints:

A. Route: Retrieve All

- Method: GET
- Endpoint: /press-releases/all
- Behavior: Returns a paginated list of all press releases currently stored in the Elasticsearch index.

B. Route: Search & Filter

- Method: POST
- Endpoint: /press-releases/search
- Behavior:
    1. Primary Search: Performs a full-text search specifically on the Title field (e.g., searching "Oncology" should match titles containing that word).
    2. Secondary Filtering: Applies strict filters to the search results based on metadata fields (Company, Category, Date).
- Logic: The response must only return documents that match the Title query AND satisfy all provided filters.

C. Elasticsearch Integration

- The server must not query the relational DB directly for searches; it must query the Elasticsearch index to ensure high-speed retrieval.

Bonus Challenge (Nice to Have)

1. Implement a caching layer (e.g., Redis). If a user makes the exact same search request twice within 5 minutes, the second response should come instantly from the cache, bypassing Elasticsearch.

2. Implement auto search suggestions. As user starts typing into the search box, BE should show suggestions.

User Search Interface (Frontend)

A. The Search Bar (Primary Query)

1. A prominent search input field where users enter keywords (e.g., "Vaccine", "Phase 3").
2. This input maps to the Title search in the backend.

B. The Filter Panel (Metadata Controls)

- A sidebar or filter bar containing controls for:
  - Company: Dropdown or Checkboxes (e.g., Pfizer, AstraZeneca).
  - Category: Dropdown (e.g., Oncology, Regulatory).
  - Date Range: A date picker (Start/End) or preset toggles (Last 30 Days, Year to Date).
- These controls map to the Filters in the backend.

C. Results Display

- Display search results in a clean, readable format (Cards or Table).
- Each result item must show:
  - Title (clickable, leading to the original URL).
  - Company Name (ideally with a badge/tag).
  - Published Date (formatted nicely, e.g., "Jan 12, 2026").
  - Category.

Evaluation Criteria

How will this solution be judged?

1. Integration Accuracy: Does the UI correctly pass both the text query and the selected filters to the API?
2. Usability: Is it clear which filters are currently active? Can the user clear them easily?
3. Readability: Is the data presented in a way that is easy to scan? (e.g., visual hierarchy between Title and Date).
4. Resilience: Does the app break if the API returns an error, or does it show a user-friendly message?

Bonus Challenge (Nice to Have)

1. Implement an export feature, that will allow users to download all the information in an Excel format.
2. The "Insights" Widget
    a. Instead of just listing rows of data, build a visualization widget at the top of the dashboard that summarizes the search results.
    b. Trend Line: A line chart showing the volume of press releases over time (e.g., "Pfizer had a spike in releases in March").
    c. Category Breakdown: A pie or bar chart showing the distribution of categories (e.g., "60% Oncology, 40% Regulatory").
    d. Dynamic Update: These charts must update in real-time as the user applies filters (e.g., if I filter to "AstraZeneca," the charts redraw to show only AstraZeneca's data).