Scaling

Resume Ranking System: Scalable Architecture and Best Practices

Overview

This document outlines the architectural design and best practices for a **Resume Ranking System**, consisting of two primary components:

1. **Resume Extraction Module (extract.py)** – Extracts relevant criteria from jd.
2. **Resume Ranking Module (rank.py)** – Ranks resumes based on specific criteria.

To ensure scalability and efficiency, the system should adopt the following optimizations:

---

1. Leveraging Amazon S3 for Scalable Resume Storage

Problem Statement

When dealing with a large volume of resumes, sending the entire document to the backend for processing can lead to scalability and performance issues. Instead of directly uploading resumes, a more efficient approach is required.

Proposed Solution

- The **frontend** should upload the resume files to **Amazon S3**.
- Instead of sending the actual file to the backend API, the **S3 URL** of the uploaded file should be sent.
- The **backend API** will then use the S3 URL to fetch and process the resumes as needed.

Advantages

✅ Reduces backend storage and processing overhead.
✅ Improves scalability for handling large-scale resume uploads.
✅ Enhances system reliability and efficiency.

---

2. Using MongoDB for Efficient Data Management

Problem Statement

Currently, the extracted resume data is stored as JSON and passed between modules (extract.py → rank.py). This results in unnecessary data transfers and redundancy.

Proposed Solution

- **Replace JSON-based data flow with MongoDB.**
- When extract.py processes a resume, it should store the extracted data in MongoDB.
- Each resume entry will be assigned a **unique ID** (resume_id).
- Instead of passing JSON data to rank.py, only the resume_id should be provided.
- rank.py can then retrieve the resume details from MongoDB using this ID for ranking.

Advantages

✅ **Optimized Data Flow** – Eliminates redundant data transfers between modules.
✅ **Structured Storage** – Enables organized data management.
✅ **Scalability** – Easily supports large datasets and queries efficiently.
✅ **Persistence** – Resume data remains stored for future ranking and analysis.

---

3. Implementing Background Processing in FastAPI for Ranking

Problem Statement

If the ranking module (rank.py) is executed synchronously, it may cause **timeouts** when processing a large number of resumes. The system needs to handle ranking efficiently without blocking the API.

Proposed Solution

- **Use FastAPI's background tasks** to process ranking asynchronously.
- When the API receives the ranking request, it:
    1. **Adds the ranking task to the background process.**
    2. **Immediately returns a process_id** to the client, indicating the ranking has started.
    3. The ranking is performed in the background, preventing request timeouts.
    4. The client can later query the status of the ranking using the process_id.

Implementation Steps

1. **When the API receives the ranking request:**
    - Store the task in **MongoDB** with a **process ID** and status (pending).
    - Immediately return the process_id to the client.
2. **The background process fetches the resumes and performs ranking.**
3. **Once ranking is complete, update the process status (completed) in MongoDB.**

4. **Clients can poll the status using the process_id.**

Advantages

✅ **Prevents API timeouts** when ranking a large volume of resumes.
✅ **Improves user experience** by allowing asynchronous processing.
✅ **Enables scalability** by handling multiple ranking requests efficiently.
✅ **Allows status tracking** for long-running tasks.

Implementation Workflow

1. Resume Extraction (extract.py)

- Uploads resume to **S3**.
- Extracts structured information.
- Stores extracted data in **MongoDB** with a **unique resume_id**.
- Returns resume_id to the frontend/backend for ranking.

2. Resume Ranking (rank.py) with Background Processing

- Receives resume_id from the frontend/backend.
- **Immediately returns a process_id** to the client.
- Adds ranking task to the **FastAPI background worker**.
- Retrieves resume details from **MongoDB**.
- Performs ranking based on the given criteria.
- Updates ranking results and **process status** (completed).
- Clients can query ranking **status/results** using process_id.