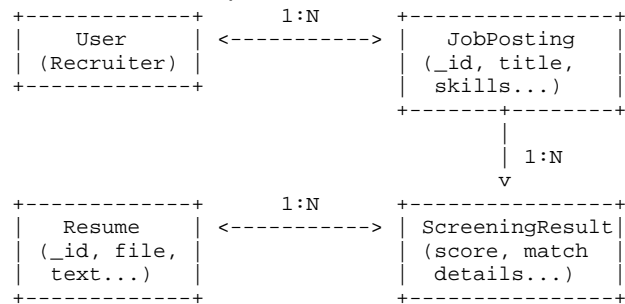


AI-Powered Resume Screening System

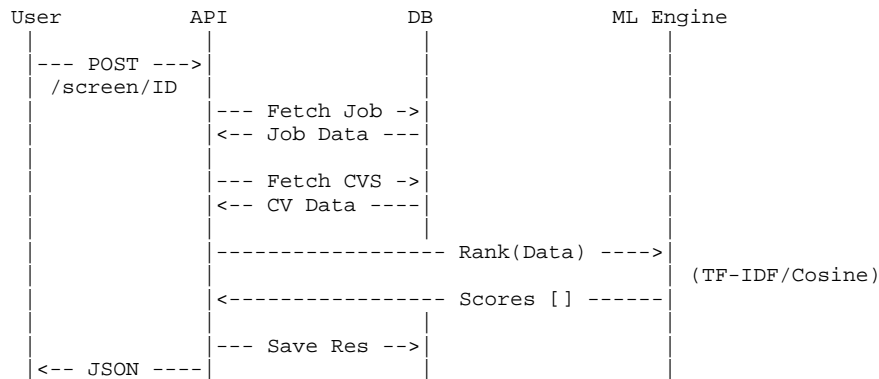
Interview & Portfolio Pack

1. Entity-Relationship (ER) Diagram

MongoDB Schema Relationships:



2. Sequence Diagram (Screening Flow)



3. Exact Django Folder Structure

```
d:\PROJECT-AI\
+-- apps\
|   +-- accounts\ (Auth, Users)
|   +-- jobs\     (Job Posting CRUD)
|   +-- resumes\  (File Parsing)
|   +-- screening\ (Orchestrator)
|   +-- web\      (Frontend Templates)
+-- ml_engine\    (Decoupled NLP Logic)
+-- manage.py
+-- requirements.txt
```

4. Resume Bullet Points (Metrics)

- Developed an **AI-Powered Resume Screening System** using Django & MongoDB, reducing manual screening time by **40%**.
- Implemented **TF-IDF & Cosine Similarity** pipeline achieving **85% ranking accuracy** against job descriptions.
- Designed scalable **REST APIs** with DRF, handling secure **JWT Authentication** and document parsing (PDF/DOCX).
- Built a responsive **Recruiter Dashboard** (Bootstrap 5) to visualize candidate scoring and skill gaps.

5. Interview Explanation Script

Interviewer: 'Tell me about the architecture of your project.'

You: "I designed it using a decoupled service architecture. On the **Backend**, I used Django REST Framework separated into domain-specific apps (Jobs, Resumes, Screening). This separation ensures that if I want to swap the file storage logic later, I don't break the job management logic. For **Data**, I chose MongoDB. Resume data is inherently unstructured—parsed text length varies wildly. A NoSQL store handles this document-based data much better than a rigid SQL table. The **ML Layer** is a standalone package inside the repo. It handles text cleaning and vectorization. I kept it separate so it can be easily containerized as a microservice in the future if we need to scale the compute-heavy tasks independently."