AI-Powered Resume Screening System

System Architecture Document

1. High-Level Architecture

The system follows a Model-View-Controller (MVC) pattern with a Service-Repository layer for ML logic.

```
        Client (Browser) -> Nginx -> Django Backend
                                        |
                                +-------+-------+
                                |               |
                           MongoDB (Data)    ML Engine
```

2. Component Detail

2.1 Backend (Django + DRF)

Modular apps structure:

• **apps.accounts**: Custom User, Role management, JWT Auth.

• **apps.jobs**: Job Posting CRUD.

• **apps.resumes**: File uploads & Text extraction (PDF/DOCX).

• **apps.screening**: Bridge between Data and ML Engine.

• **apps.web**: SSR UI using Django Templates.

2.2 Database (MongoDB)

Using MongoDB via djongo for flexibility with unstructured resume data.

Collections: *auth_user, api_resume, api_jobposting, api_screeningresult*

2.3 ML / NLP Pipeline

Decoupled Python package **ml_engine**.

1. Input: Job Description + Resumes

2. Preprocessing: Tokenize, Lemmatize (NLTK)

3. Vectorization: TF-IDF

4. Ranking: Cosine Similarity

5. Output: Similarity Scores (0-1)

3. Resume Screening Lifecycle

```
    1. User uploads Resume (POST /api/resumes/)
    2. Server extracts text & saves to MongoDB
    3. User triggers Screening (POST /api/screen/{job_id})
    4. ML Engine fetches data, computes TF-IDF & Cosine Similarity
    5. Results saved to ScreeningResult collection
    6. Frontend displays ranked candidates
```

4. Authentication

JWT (JSON Web Token) based flow.

• content-type: application/json

• Authorization: Bearer