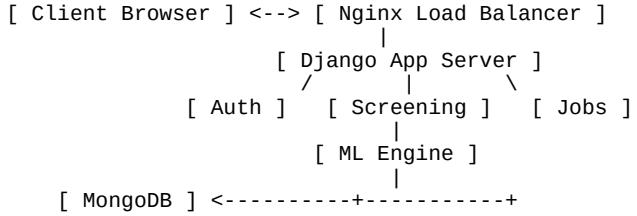
AI-Powered Resume Screening System

End-to-End Technical Report

This document aims to provide a comprehensive technical overview of the architecture, database design, ML pipeline, and deployment strategy for the AI Resume Screener project.

1. System Architecture

The system follows a scalable Model-View-Controller (MVC) architecture, utilizing Django for the backend and a decoupled ML engine.



Key Components:

• **Django REST Framework:** Handling API requests.

• **MongoDB (Djongo):** Storing unstructured resume data.

• **Scikit-learn:** Powering the TF-IDF and Cosine Similarity engine.

2. Database Design (MongoDB)

We leverage MongoDB for its flexibility with schema-less data.

Collection	Description	Key Fields
users	Auth & Roles	username, password, role
job_postings	Job Requirements	_id, title, skills, recruiter_id
resumes	Candidate Data	_id, file_path, parsed_text
screening_results	ML Outcomes	job_id, resume_id, score

3. NLP & ML Pipeline

The core intelligence of the system.

1. Text Extraction (pdfminer.six)
2. Cleaning (Lowercase, Remove Stopwords)
3. Tokenization & Lemmatization
4. TF-IDF Vectorization
5. Cosine Similarity Calculation

```
# Logic Snippet
tfidf = TfidfVectorizer()
vectors = tfidf.fit_transform([job_text] + resume_texts)
scores = cosine_similarity(vectors[0], vectors[1:])
```

4. API Documentation

RESTful endpoints protected by JWT.

Method	Endpoint	Purpose
POST	/api/auth/login/	Get JWT Tokens
POST	/api/jobs/	Create Job Posting
POST	/api/resumes/	Upload CV (Multipart)
POST	/api/screen/{id}/	Trigger ML Screening

5. Dashboard & Analytics

Recruiters view ranked candidates in real-time.

Features:

• Visual Progress Bars for matching scores.

• Color-coded prioritization (Green/Yellow/Red).

• Drill-down view into missing skills.

6. Deployment Strategy

Production Stack:

• Docker + Docker Compose for containerization.

• Gunicorn as the WSGI application server.

• Nginx as the reverse proxy.

• GitHub Actions for CI/CD.