Resume Processing System - Technical

Challenge

# Deadline: Thursday, May 29th, 2025 at 11:59 PM

**Overview**

Build a complete resume processing system that automatically extracts information from resume files, stores data in a relational database, and provides a web interface for viewing and shortlisting candidates.

# System Architecture Requirements

### File Storage System

**Objective:** Store and manage resume files

* + **Local Option:** Create a dedicated folder structure on your local machine
  + **Cloud Option:** Use cloud storage services like:
    - AWS S3
    - Google Cloud Storage
    - Azure Blob Storage
    - Any other cloud file storage service

##### Requirements:

* + Support common resume formats (PDF, DOC, DOCX, TXT)
  + Organize files with clear naming conventions
  + Implement file upload/management capabilities

### Resume Processing & Data Extraction

**Objective:** Parse resume files and extract structured information

##### Suggested Fields to Extract:

* + Full Name
  + Email Address
  + Phone Number
  + Location (City, State/Country)
  + Education (Degree, Institution, Year)
  + Work Experience (Company, Position, Duration)
  + Skills (Technical and Soft Skills)
  + Years of Experience (calculated or extracted)

**Implementation Options:**

* + Use libraries like pd f plumbe r, PyPDF2, python —docx for text extraction
  + Implement regex patterns for information extraction
  + Use NLP libraries like spaCy or NLTK for advanced parsing
  + Consider using pre-trained models for entity extraction

#### Relational Database Design

**Objective:** Store extracted resume data in a structured format Suggested Database Options:

* + PostgreSQL (Recommended for production-like experience)
  + MySQL (Widely used, good documentation)
  + SQLite (Lightweight, easy setup for development)
  + Microsoft SQL Server (Enterprise-grade option) Suggested Database Schema (simplify as needed): Candidates Table:
* candidate id (Primary Key, Auto-increment)
* full name (VARCHAR, NOT NULL)
* email (VARCHAR, UNIQUE)
* phone (VARCHAR)
* location (VARCHAR)
* years\_experience (INTEGER)
* resume\_fiIe\_path (VARCHAR)
* status (ENUM: 'pending’, ’shortlisted', ’rejected')
* created\_at (TIMESTAMP)
* updated\_at (TIMESTAMP) Education Table:
* education id (Primary Key)
* candidate id (Foreign Key)
* degree (VARCHAR)
* institution (VARCHAR)
* graduation year (INTEGER)
* gpa (DECIMAL, optional) Skills Table:
* skiII\_id (Primary Key)
* candidate\_id (Foreign Key)
* skiII\_name (VARCHAR)
* skiII\_category (ENUM: 'technical', 'soft', 'language', 'other')
* proficiency\_IeveI (ENUM: 'beginner', 'intermediate', 'advanced', 'expert')

#### Backend API Development

**Objective:** Create RESTfuI APIs to manage data and business logic

##### Required Endpoints:

* + POST / api/ resumes/ upload - Upload and process resume files
  + GET / ap:I/candidates - Retrieve all candidates with pagination
  + GET /ap1/candidates/ (id} - Get specific candidate details
  + POST / api/candidates/ shortlist - Process shortlisting criteria
  + PUT /api/candidates/ (1d}/ status - Update candidate status
  + GET /ap1/dashboa rd/ stats - Get dashboard statistics

##### Technology Suggestions:

* + Python: Flask, FastAPl, Django REST Framework
  + **Node.js:** Express.js, Nest.js
  + Java: Spring Boot
  + C#: ASP.NET Core
  + Go: Gin, Echo
  + Ruby: Ruby on Rails API

#### Frontend Dashboard

**Objective:** Create an intuitive web interface for viewing and managing candidates

##### Suggested Features:

* + Candidate List View:

a Searchable and filterable table a Pagination support

Sort by different criteria a Status indicators

##### Shortlisting Interface:

* + - Input form for shortlisting criteria
    - Real-time progress indicator
    - Results display
  + Individual Candidate View:
    - Detailed candidate information
    - Resume file viewer/download
    - Status update controls

##### Technology Suggestions:

* + JavaScript: React.js, Vue.js, Angular
  + Python: Streamlit (for rapid prototyping)
  + **Server-side:** Django Templates, Flask-Jinja2
  + CSS Frameworks: Bootstrap, Tailwind CSS, Material-UI

### Shortlisting System

**Objective:** Implement automated candidate shortlisting based on criteria

##### Option A: Keyword-Based Matching (Simpler)

* + Create weighted scoring system
  + Match keywords in skills, experience, education
  + Implement fuzzy matching for better results
  + Allow user to set minimum score thresholds

##### Option B: Small Language Model (Advanced)

* + **Suggested Models:**
    - Sentence Transformers (all-MiniLM-L6-v2)
    - DistilBERT
    - TinyBERT
    - Local Ollama models (Llama, Phi-3, Gemma-2B)

##### Implementation Approach:

* + - Embed job requirements and resume content
    - Calculate similarity scores
    - Rank candidates based on semantic matching Consider using Hugging Face Transformers library

##### Shortlisting Criteria Examples:

* + Required skills matching
  + Minimum years of experience
  + Education level requirements
  + Location preferences

## Integration & Workflow

##### Complete System Flow:

1. User uploads resume files to file storage
2. System automatically processes files and extracts data
3. Extracted data is stored in relational database
4. Frontend displays all candidates in dashboard
5. User inputs shortlisting criteria through web interface
6. Backend API processes criteria and ranks candidates
7. Top 10% of candidates are marked as "shortlisted"
8. Database is updated with new status
9. Frontend refreshes to show shortlisting results
10. User can view shortlisted candidates and their details

# Technical Implementation Guidelines

## Error Handling & Logging

* + Implement comprehensive error handling
  + Add logging for debugging and monitoring
  + Handle edge cases (corrupted files, parsing errors)
  + Provide meaningful error messages to users

## Security Considerations

* + Input validation and sanitization
  + SQL injection prevention
  + File upload security (type validation, size limits)
  + Basic authentication (if time permits)

## Performance Optimization

* + Database indexing on frequently queried fields
  + Efficient file processing (consider async processing)
  + Frontend loading states and progress indicators
  + API response caching where appropriate

# Deliverables

### Code Repository

* + **GitHub Repository:** Must be public and well-organized
  + **README.md:** Comprehensive setup and usage instructions
  + Project Structure: Clear folder organization
  + **Dependencies:** Requirements file (requirements.txt, package.json, etc.)

### Documentation

* + **Architecture Diagram:** Visual representation of system components
  + Database Schema: ERD or schema documentation
  + **API Documentation:** Endpoint descriptions and examples

# Evaluation Criteria

### Technical Implementation (70%)

* + Code quality and organization
  + Proper use of chosen technologies
  + Database design and relationships
  + API design and functionality
  + Frontend usability and design

## Problem-Solving Approach (30%)

* + Algorithm efficiency for resume parsing
  + Shortlisting logic implementation
  + Error handling and edge cases
  + System integration completeness

# Submission Requirements

## GitHub Repository Must Include:

* + [ ] Complete source code
  + [ ] README.md
  + [ ] Database schema/migration files
  + [ ] Requirements/dependencies file

### Pre-Interview Preparation:

Be ready to explain and demonstrate:

* + Technology stack choices and reasoning
  + Database design decisions
  + Shortlisting algorithm implementation
  + Challenges faced and solutions implemented
  + Potential improvements and scalability considerations

Resources and Tips

#### Helpful Libraries by Language:

##### Python:

* + pdIp 1umbe r, PyPDF2 for PDF processing
  + python —docx for Word documents
  + pandas for data manipulation
  + sq1a1 chemy for database ORM
  + flask or fast api for API development

##### JavaScript/Node.js:

* + pdf— pa rse for PDF processing
  + mammoth for Word documents
  + seque1i ze or p risma for database ORM
  + exp ress for API development
  + mu1te r for file uploads

##### Documentation Resources:

* + Database design best practices
  + RESTfuI API design principles
  + Frontend UI/UX guidelines
  + Git best practices and README templates

Important Notes

* + **Flexibility:** Focus on your strengths. If you're stronger in backend, spend more time on robust APIs and database design. If you're frontend-focused, create an exceptional user interface.
  + **MVP Approach:** Start with a minimum viable product and then add enhancements. It's better to have a working basic system than an incomplete complex one.
  + **Communication:** Document your thought process, especially for features you couldn't complete due to time constraints.
  + **Testing:** While not mandatory, including basic tests will demonstrate good development practices.
  + **Version Control:** Make regular commits with meaningful messages to show your development process.

# Questions?

If you encounter blockers or need clarification on requirements, document your assumptions and proceed with your best judgment. Be prepared to discuss these decisions during the technical interview.

Final Submission: Share your GitHub repository link before the deadline. Ensure the repository is public and contains all necessary files for running your application.

Good luck! We're excited to see your creative solutions and technical implementations.