Workflow Documentation

Project: coRecruit Platform

AI-Driven Recruitment Solution

Version: 3.0 (Final)

Date: August 1st, 2025

Prepared by: Team Siberbyte

Table of Contents

1. Introduction

This workflow document outlines the key processes and system interactions within the coRecruit Platform, an offline-capable, AI-powered recruitment solution. The workflows described enable efficient resume processing, semantic job matching, and continuous machine learning improvements driven by HR feedback.

2. Objectives

- Automate resume parsing from various file formats (PDF, DOCX, DOC).
- Enable semantic matching of candidates against job descriptions using advanced AI models.
- Incorporate HR feedback to continuously improve the accuracy and relevance of matches.
- Support offline processing capabilities to maintain productivity without constant internet connectivity.
- Provide role-based access control ensuring secure data handling and operational integrity.

3. Scope

This document covers workflows related to:

- Resume Processing
- Job Matching
- Feedback Integration
- User Access and Permissions

4. Roles and Responsibilities

Role	Access and Responsibilities
HR Manager	View candidates, post jobs, provide feedback on matches.
Recruiter	View candidates, access dashboard.
Basic Admin	Manage user accounts and job categories.
Advanced Admin	Manage users, jobs, and monitor system health.
Full Admin	Full system access including security configurations.

5. Workflow Details

5.1 Resume Processing Workflow

Objective: Efficiently extract and standardize candidate data from resumes uploaded in multiple formats.

Trigger: HR Manager or Recruiter uploads CV files (PDF/DOCX/DOC).

Description:

1. **Upload:** User uploads resumes through the frontend (React-based portal).
2. **Field Extraction:**
   - **Rule-Based Parsing:**
     PyMuPDF and spaCy are used to extract structured fields such as name, email, phone number, and basic contact information.
   - **LLM-Based Extraction (Mistral-7B 4-bit):**
     Mistral-7B is used to extract and refine complex fields such as experience, skills, and education.
3. **Vector Embedding:** Candidate information is converted to vector embeddings via the all-MiniLM-L6-v2 model and stored in ChromaDB for semantic search.
4. **Storage:** Parsed and processed data is stored in the SQLite3 database. Django’s default security features ensure safe data handling, including hashed passwords, secure session management, and protection for data in transit via HTTPS.

5.2 Semantic Job Matching Workflow

Objective: Provide HR and recruiters with ranked candidates based on semantic relevance to job postings.

Trigger: Job descriptions (JDs) are posted or available for matching.

Description:

1. **Job Posting:** HR Manager posts new job listing through the platform.
2. **Embedding:** Job description text is embedded using the same embedding model to ensure uniform vector space.
3. **Matching:** ChromaDB performs similarity search between job embeddings and candidate embeddings.
4. **Ranking:** Candidates are ranked by semantic similarity scores combining skills, role, experience, and other factors.
5. **Output:** Ranked list displayed on the dashboard for HR Manager and Recruiter.

5.3 HR Feedback Integration Workflow

Objective: Enhance the AI system results over time using human-in-the-loop feedback.

Trigger: HR Manager reviews candidate matches and provides feedback on suggested scores and recommendations.

Description:

1. **Feedback Submission:** HR reviews match results and provides feedback via View Candidates page. Feedback includes suggested score adjustments and qualitative comments.
2. **Feedback Storage:** The system stores the feedback in the local file system. When generating future results, it converts the stored feedback into vector embeddings and uses them alongside relevant job and candidate IDs.
3. **Model Guidance:** During future inferences for similar roles or candidates, the model retrieves relevant feedback embeddings from the stored data.

4. **Outcome:** The system gradually improves alignment with HR preferences across similar roles, leveraging feedback memory to enhance relevance and reduce misalignment.

5.4 User Access and Permission Workflow

Objective: Ensure secure and role-based access to the platform's functionalities.

Description:

1. **User Login:** Users authenticate using POST /api/login/.
2. **Role Verification:** Access rights are enforced based on roles stored in the useraccount table (full_admin, basic_admin, advanced_admin, hr_manager, recruiter).
3. **Permission Control:** Based on role, users can perform actions such as posting jobs, viewing candidates, managing categories, or monitoring system health.
4. **Account Management:** Admins manage user status, roles, and permissions through frontend interfaces calling respective APIs, e.g., GET /api/useraccounts/.

6. System Interactions

- **Frontend:** React 19 with Shadcn Radix UI components provide user interfaces for dashboards, candidate views, job postings, and admin features.
- **Backend:** Django APIs handle all business logic, authentication, and database interaction with SQLite3.
- **AI Module:** Combines rule-based parsing (PyMuPDF, spaCy), Mistral-7B model, and ChromaDB for semantic similarity search.
- **Monitoring:** Prometheus collects system metrics; Grafana visualizes system health and performance for admins.
- **Infrastructure:** The entire solution is containerized with Docker and deployed using GitHub Actions CI/CD pipelines.

7. Security and Compliance

Frontend Security:

- Centralized and scalable state management using **Zustand**.

- **Persistent login** is maintained through localStorage, ensuring user sessions persist across reloads.
- **Secure logout** clears both Zustand state and localStorage, ensuring clean session termination.
- **Multi-user login prevention** enforces one active session per user.
- **Cross-tab authentication sync** ensures session consistency across browser tabs.
- **Input validation** is handled using **Zod schemas**, providing strong client-side data integrity checks.
- **Role-Based Access Control (RBAC)** is implemented on the frontend to restrict access based on user roles.

Backend Security:

- Resume data and user information are stored securely using Django's default security features.
- Sensitive operations are protected with **CSRF**, **XSS**, and **SQL injection** safeguards provided by Django.

Authentication & Authorization:

- User credentials are hashed using **PBKDF2 with SHA-256** (Django default).
- Access control is enforced on both frontend and backend based on user roles.

8. References

- coRecruit Technical Documentation v3.0

Document Control

Version	Date	Author	Change Description
1.0	June 30, 2025	Development Team	Initial Draft
2.0	July 15, 2025	Development Team	Implemented Django Models & AI Module
3.0	July 31,2025	Development Team	Feedback Integration & Final Testing