Project Documentation: Automated Resume Relevance Check System

- **Project For:** Innomatics Research Labs
- **Document Version:** 2.0
- **Date:** September 20, 2025
- **Status:** Design & Planning
- **Change Log (v2.0):**
    - *Project re-scoped to align with the Innomatics Research Labs problem statement.*
    - *Architecture updated to a hybrid model: **PostgreSQL** for metadata and **FAISS** for in-memory, job-specific vector indexing.*
    - *Added a major new feature: **Personalized Feedback Generation for Students.***
    - *User personas clarified: **Placement Team** and **Students.***
    - *Defined detailed PostgreSQL schemas and specific FastAPI routes.*
    - *Incorporated **LangGraph** for stateful analysis and **LangSmith** for observability.*
    - *Outlined the design for the Placement Team's web dashboard.*

Part 1: Updated Product Vision & Architecture

1.1. High-Level Project Overview

The Automated Resume Relevance Check System is an AI-powered platform designed for Innomatics Research Labs to automate, standardize, and scale the resume evaluation process. It will process thousands of weekly student applications against specific job descriptions (JDs), providing a rapid, consistent, and actionable screening solution. The system's primary outputs include a **Relevance Score**, a **Suitability Verdict**, a list of **Skill Gaps**, and **Personalized Improvement Feedback** for students, all accessible via a central web dashboard for the placement team.

1.2. System Portals & User Experience

The system will feature two primary interfaces:

1. **Placement Team Dashboard (Production: Next.js/React)**
    - **Functionality:** Allows placement team members to upload new JDs, manage job postings, view a ranked list of applicants for each job, and drill down into the detailed evaluation of any student's resume.
2. **Student Upload Portal (Future Scope)**
    - **Functionality:** A simple interface for students to upload their resumes for specific job applications. In the MVP, this can be a simple upload form managed by the placement team.

1.3. Backend Architecture (FastAPI)

- **Framework:** Python with FastAPI for a high-performance, asynchronous API.
- **Data Storage (Hybrid Model):**
    - **Metadata Store (PostgreSQL):** The primary database for all structured, persistent data. This includes user info, job descriptions, extracted resume text, all final scores, verdicts, and generated feedback.
    - **Vector Store (FAISS):** Used for high-speed, in-memory semantic search. For each job analysis, a *temporary, job-specific FAISS index* is built, used for the semantic match, and then discarded or archived. This is highly efficient for the "batch processing" nature of the problem.
- **Asynchronous Task Queue (Celery & Redis):** Manages the entire multi-step resume analysis pipeline in the background, providing a non-blocking experience.
- **Secure File Storage (AWS S3):** Stores all uploaded PDF/DOCX files for resumes and JDs.

1.4. AI Engine & Orchestration

- **Orchestration: LangChain** and **LangGraph** are used to create a stateful, multi-step pipeline for the analysis. This allows the system to pass the state of a resume (e.g., raw text, extracted skills, hard match score) through a graph of processing nodes.
- **Observability: LangSmith** is integrated to trace, debug, and evaluate the performance of the entire LLM and logic chain, ensuring reliability and consistency.
- **Core AI Tasks:**
    1. **JD Parsing:** An LLM call to extract structured data (must-have skills, qualifications, etc.) from the unstructured text of a JD.
    2. **Resume Parsing:** Text extraction (PyMuPDF, python-docx) and normalization (spaCy/NLTK).
    3. **Hybrid Scoring:**
        - **Hard Match:** Keyword checks (TF-IDF, fuzzy matching) for mandatory skills.
        - **Semantic Match:** Generating embeddings (HuggingFace models) and using `FAISS` to find the cosine similarity between the resume and JD.
    4. **Personalized Feedback Generation:** A final, targeted LLM call that uses the scoring results (e.g., missing skills, low semantic score) to generate constructive, actionable feedback for the student.

Part 2: Detailed Backend Flow, Schemas, and API Routes

2.1. PostgreSQL Database Schemas

- **`jobs` table:**
    - `job_id` (PK), `title`, `company_name`, `location`, `full_jd_text`, `status` ('OPEN', 'CLOSED'), `structured_jd_json` (from the first LLM call), `created_at`.
- **`students` table:**

    - `student_id` (PK), `name`, `email`, `location` ('Hyderabad', 'Bangalore', etc.).
- **`resumes` table:**
    - `resume_id` (PK), `student_id` (FK), `s3_key`, `upload_date`, `extracted_text_clean`.
- **`evaluations` table:**
    - `evaluation_id` (PK), `job_id` (FK), `resume_id` (FK), `relevance_score` (0-100), `suitability_verdict` ('High', 'Medium', 'Low'), `hard_match_score`, `semantic_match_score`, `missing_elements_json`, `generated_feedback_text`, `status` ('PENDING', 'COMPLETED', 'FAILED'), `evaluated_at`.

2.2. Core API Routes (FastAPI)

- **Job Management:**
    - `POST /jobs`: Upload a new JD. **Body:** `{ "title": "...", "company_name": "...", "location": "...", "full_jd_text": "..." }`. **Response:** `{ "job_id": "...", "message": "Job created and analysis initiated." }`. This endpoint also triggers the asynchronous analysis task.
    - `GET /jobs`: List all jobs. **Response:** A list of job objects.
- **Resume Management:**
    - `POST /resumes/upload`: **Body:** `{ "student_id": "...", "file_name": "..." }`. **Response:** `{ "upload_url": "...", "s3_key": "..." }`. Provides a secure, pre-signed S3 URL for the client to upload the resume file.
- **Dashboard & Results:**
    - `GET /jobs/{job_id}/results`: Get the ranked list of candidates for a specific job. **Response:** A list of evaluation objects, pre-sorted by `relevance_score`.
    - `GET /evaluations/{evaluation_id}`: Get the detailed drill-down for a single evaluation, including the personalized feedback.

2.3. Detailed Asynchronous Analysis Workflow

Triggered by `POST /jobs`, which calls `start_job_analysis.delay(job_id)`.

1. **Phase 1: Setup**
    - Fetch job details and the list of all student resumes to be evaluated from PostgreSQL.
    - Make a single LLM call to parse the JD into a structured JSON object (skills, etc.) and save it back to the `jobs` table.
2. **Phase 2: Parallel Resume Processing**
    - For each resume, execute the following steps in parallel:
        - **Text Extraction:** Download from S3 and extract clean text using PyMuPDF/OCR.

        - **Hard Match Scoring:** Run keyword and fuzzy-matching checks against the structured JD data.
        - **Embedding Generation:** Create a vector embedding for the resume text using a HuggingFace model.
        - **Initial DB Write:** Store the extracted text and hard match score in the `evaluations` table.
3. **Phase 3: Batch Semantic Analysis (FAISS)**
    - *After all resumes are processed in Phase 2*, the system gathers all the generated resume embeddings.
    - **Build FAISS Index:** A `FAISS` index is built *in-memory* containing the vectors of all resumes for this specific job.
    - **Generate JD Embedding:** Create the vector embedding for the job description.
    - **Search & Score:** The JD embedding is used to search the FAISS index, yielding a semantic similarity score for every resume in the batch.
    - **DB Update:** The `semantic_match_score` for each evaluation is updated in PostgreSQL.
4. **Phase 4: Finalization & Feedback Generation**
    - **Calculate Composite Score:** A final weighted score is calculated in the database for all candidates based on their hard and semantic match scores. The `relevance_score` and `suitability_verdict` are set.
    - **Generate Feedback (Targeted LLM Calls):** For each student, a final, targeted LLM call is made.
        - **Prompt:** `"You are a career coach for Innomatics. A student's resume was evaluated for the '{job_title}' role. Their score was {score}/100. They are missing these key skills: {missing_skills_list}. Their experience was a {semantic_verdict} fit. Write a short, encouraging paragraph of personalized feedback with actionable advice on how to improve their resume for similar roles."`
        - The `generated_feedback_text` is saved to the `evaluations` table.
    - **Job Completion:** The job status is marked as 'COMPLETED'.

Part 3: The Placement Team Dashboard Display

The data from `GET /jobs/{job_id}/results` will be displayed in a clean, interactive dashboard.

1. **Main View: Candidate Leaderboard**
    - A searchable and filterable table of all applicants for a selected job.
    - **Columns:**
        - `Student Name`

        - `Relevance Score`: Displayed as a number (e.g., 88) with a color-coded progress bar (e.g., green for high scores).
        - `Suitability Verdict`: A colored tag/badge (e.g., High - Green, Medium - Yellow, Low - Red).
        - `Missing Skills`: A list of tags showing the key skills the candidate is missing.
        - `View Details` (Button).
    - **Filtering Controls:** Allow the placement team to filter by Verdict, Score Range, and Student Location.
2. **Detail View (On "View Details" click):**
    - A pop-up or separate page showing:
        - **Student & Score Summary:** Name, Score, Verdict.
        - **The Resume:** An embedded viewer (iframe) to display the original resume PDF from S3.
        - **Detailed Breakdown:** A list showing the Hard Match vs. Semantic Match scores.
        - **Personalized Feedback:** The full text of the generated feedback, ready for the placement team to review and potentially share with the student.