Project Documentation: Talent Scout - AI Hiring Assistant

This document provides a comprehensive breakdown of the Talent Scout Streamlit application, a sophisticated AI-powered chatbot designed to automate the initial technical screening of job candidates.

1. Overview

Talent Scout is a stateful, multi-lingual Streamlit web application that functions as an interactive hiring assistant. It guides candidates through a multi-step application process, collects their personal and professional details, parses their resume, and then dynamically generates and administers a 5-question technical screening using an LLM.

The application is designed to be resilient, with fallback mechanisms for API failures, and it persists candidate data locally for review and to allow candidates to resume a previous session.

2. Core Features

- **Multi-Step Candidate Registration:** A guided wizard (steps 0-4) collects candidate information, including PII, preferred language, tech stack, job role, and experience.
- **Resume Parsing:** Supports .pdf, .docx, and .txt file uploads. It extracts raw text to be used as context for question generation.
- **Dynamic Question Generation:** Uses the OpenRouter API (specifically the qwen/qwen3-coder:free model) to generate 5 unique, tailored technical questions based on the candidate's job role, tech stack, and resume.
- **Robust Fallback:** If the LLM call fails or returns a malformed response, the app serves a set of hard-coded get_fallback_questions to ensure the screening can continue.
- **Interactive Screening:** Presents the 5 generated questions one by one. Candidate answers are stored in st.session_state.
- **Paste Prevention:** Includes custom CSS and JavaScript to disable pasting into answer text areas, encouraging original responses.
- **AI-Powered Q&A:** After the screening, candidates enter a general chat (Step 6) where they can ask questions. The AI provides answers (get_llm_answer) and can even generate performance feedback (generate_feedback) on the screening.
- **Multi-Lingual Support:** The entire UI and all AI interactions can be translated into one of the supported languages (LANGUAGES) using an LLM-based translate_text function.
- **Session Persistence:**
  - Checks for returning candidates (check_past_candidate) by email/phone.
  - Saves all interview data (questions, answers, chat) to a unique JSON file per candidate (save_candidate_responses).
- **Data Export:** Provides download buttons for both the candidate and recruiter to get a JSON or TXT copy of the screening.
- **Theming:** A simple toggle in the sidebar switches between dark and light themes.

3. Tech Stack & Dependencies

- **Core:** Python 3
- **Web Framework:** streamlit
- **LLM API:** requests (to connect to OpenRouter)
- **Configuration:** python-dotenv (to load OPENROUTER_API_KEY)

- **Document Parsing:**
    - PyPDF2: For .pdf files.
    - python-docx: For .docx files.
- **Utilities:** json, csv, uuid, hashlib, re

4. Setup & Running the Application

1. **Install Dependencies:** pip install streamlit requests pypdf2 python-docx python-dotenv
2. **Set Up Environment Variables:** Create a file named .env in the same directory as the script and add your OpenRouter API key:
3. OPENROUTER_API_KEY="your_api_key_here" The application will also fall back to checking Streamlit secrets, which is useful for deployment.
4. **Run the Application:** streamlit run your_script_name.p

5. Application Flow (State Machine)

The entire application operates as a state machine controlled by st.session_state.step.

- **Step 0: Initial Info & Welcome**
    - Greets the user and displays a form (initial_candidate_form) to collect full_name, email, phone, and language.
    - Validates input using validate_email and validate_phone.
    - **Logic:** On submit, it checks for a past candidate (check_past_candidate).
        - **If found:** Loads all their data from JSON, sets step = 6 (Q&A chat), and welcomes them back.
        - **If new:** Saves info, sets step = 1.
- **Step 1: Tech Stack**
    - Displays a multi-select box for DEFAULT_TECHS and a text input for custom techs.
    - **Logic:** On submit, saves to st.session_state.selected_techs, sets step = 2.
- **Step 2: Job Role**
    - Displays a st.selectbox for JOB_ROLES.
    - **Logic:** On submit, saves to st.session_state.job_role, sets step = 3.
- **Step 3: Additional Details**
    - Displays a form (candidate_form) for years_experience, location, and consent_store checkbox.
    - **Logic:** On submit, updates st.session_state.candidate_info, sets step = 4.
- **Step 4: Resume Upload**
    - Displays a st.file_uploader.
    - **Logic:** On submit, parse_resume is called to extract text. The text is saved to st.session_state.resume_text. Sets step = 5.
- **Step 5: Technical Screening**
    - This is the core screening logic.
    - **1. Generation:** If st.session_state.generated_questions is None, it calls build_question_generation_prompt and call_openrouter.
        - It parses the JSON response (using extract_json_from_text if needed).
        - If the API call fails or returns invalid JSON, it calls get_fallback_questions.
    - **2. Answering:** It iterates from st.session_state.current_question_index = 0 to 4.
        - For each index, it displays the corresponding question and a st.text_area inside a form.
        - **On "Submit Answer":** The answer is saved to st.session_state.candidate_answers, save_question_answer_pair is called, and the current_question_index is incremented.
        - **On "Skip Question":** "Skipped" is saved as the answer.

    - **3. Completion:** After the 5th question, it calls save_candidate_responses to save the full session to JSON and sets step = 6.
- **Step 6: Open Q&A**
    - Displays a chat input form.
    - **Logic:**
        - If the user says "no", "end", etc., it moves to step = 7.
        - If the user asks for "feedback", it calls generate_feedback and displays the result.
        - For any other query, it calls get_llm_answer and displays the result.
- **Step 7: Session End**
    - The chat input is removed.
    - The "Thank You" message (from ENDINGS) is displayed.
    - Download buttons for the session data are shown in the sidebar.

---

6. Function Breakdown (Core Components)

Resume Parsing

- **parse_resume(uploaded_file)**: The main entry point. Checks file extension and calls the appropriate helper.
- **extract_text_from_pdf_bytes(file_bytes)**: Uses PyPDF2.PdfReader to read a PDF from io.BytesIO and extracts text page by page.
- **extract_text_from_docx_bytes(file_bytes)**: Uses docx.Document to read a DOCX file and extracts text from paragraphs.
- **extract_text_from_txt_bytes(file_bytes)**: Decodes bytes as utf-8 (or latin-1 as a fallback).

LLM & API Helpers

- **call_openrouter(messages, timeout)**: The single point of contact for the OpenRouter API. It constructs the headers (with Authorization: Bearer ...) and body, sends the POST request, and handles all HTTP/network errors.
- **build_question_generation_prompt(...)**: A crucial prompt engineering function. It constructs the large system prompt and user payload, injecting candidate info, resume text, and the application_id as a seed for unique questions.
- **generate_feedback(...)**: Another prompt engineering function that asks the LLM to evaluate the candidate's answers against the provided guidelines.
- **get_llm_answer(...)**: Handles the general Q&A by providing candidate info and chat history as context.
- **translate_text(...)**: A simple utility that asks the LLM to act as a translator.
- **get_sentiment(...)**: Asks the LLM to classify text as "positive", "negative", or "neutral".
- **extract_json_from_text(text)**: A robust parser to find and extract a valid JSON object from a string that might be wrapped in markdown or other text.

Data Persistence & Validation

- **validate_email(email)** / **validate_phone(phone, ...)**: Simple validation functions to ensure basic data quality.
- **save_candidate_responses(...)**: Saves the *complete* session state (all info, questions, answers, history) into a single, comprehensive JSON file named {candidate_name}_with_{application_id}.json.
- **save_candidate_info(...)**: Appends a *new row* to registered_candidates.csv. This file acts as a lightweight database index to find the main JSON file for a returning user.

- **check_past_candidate(email, phone)**: Reads registered_candidates.csv to find a matching user. If found, it reads their full session from the corresponding JSON file.
- **save_question_answer_pair(...)**: Appends a single JSON line to {candidate_name}_with_{application_id}_answers.json *every time* a candidate submits an answer. This is good for logging/recovery if the app crashes mid-screening.
- **save_submission_anonymized(...)**: If the candidate consents, this saves non-PII metadata (job role, tech stack, experience) to submissions_anonymized.jsonl