|  |  |  |  |
| --- | --- | --- | --- |
| **ATS Resume Checker**  **Shortlisting & Screening Workflow** | | | |
| **Date:** 04/05/2025 |  | **Submitted By:** Jai Vilatkar |

**Problem Statement:**

Companies receive hundreds or thousands of resumes for job openings. Manual screening can be time-consuming and error-prone.

**Solution Overview:**

AI-Powered Resume ATS (Applicant Tracking System) Checker is designed to analyze and rank multiple resumes against a given job description. It leverages Google Gemini (Generative AI) to simulate how an ATS or recruiter would evaluate candidate resumes for role relevance, skill match, and keyword presence.

The solution automates the following:

● User Query Input: Provide a front-end dashboard where a hiring manage or orrecruiter can enter specific job requirements or queries.

● Develop end-to-end workflow.

○ Resume Retrieval: Connect to external storage services (Google Drive) to fetch resumes and related documents.

○ Analysis & Ranking: Use agentic search querying tactics—leveraging Natural Language Processing (NLP) and possibly self-corrective iterative methods—to analyze each resume and rank them based on their relevance to the input requirement.

● Results Display: Present the resumes in order of relevance, with the best match.

1. **Key Features**

✅ Multi-Resume Support: Upload multiple resumes (PDF/DOCX) simultaneously.

☁️ Cloud Integration: Fetch resumes directly from Google Drive using file IDs.

🤖 Generative AI Analysis: Uses Gemini to analyze resumes and generate:

- Match Score (0–100%)

- Key missing skills

- Suggestions for improvement

📊 Automated Ranking: Ranks resumes based on relevance to the job description.

📁 Support for Multiple Formats: Reads both PDF and DOCX formats with appropriate parsing logic.

2. **Tech Stack**

**Layer Tools Used**

Front-End Streamlit

Resume Parsing PyPDF2, python-docx

AI Model Google Gemini (via LangChain + langchain\_google\_genai)

Orchestration LangChain agent + StructuredTool

Cloud Storage Google Drive API (googleapiclient, service\_account)

Backend Auth Service Account JSON Credential

3. **Workflow Summary**

**User Inputs:**

- Uploads multiple resumes (locally or via Google Drive)

- Provides job description in a text field

**Text Extraction:**

- Extracts text from each resume using format-specific parsers

**AI Evaluation:**

- Sends the resume and job description to Gemini

- AI evaluates match score, missing skills, and recommendations

**Sorting and Display:**

- Results are sorted based on match score

- Users see ranked resumes along with AI feedback

4. **Use Case Scenarios**

- Recruiters: Prioritize candidates by match score before screening.

- Job Seekers: Optimize resume content based on job description.

**Detailed Solution:**

**1. Workflow & Architecture**

**a. Diagram & Description:**

The system comprises the following components:​

**- Front-End Dashboard:** Built using Streamlit, this interface allows users to upload resume files (PDFs or DOCXs) or provide Google Drive file IDs, and input a job description.​

**- Connectors to Storage Services:** Utilizes Google Drive API to fetch resumes stored in Google Drive.​

**- Processing Engine:** Handles text extraction from PDFs using PyPDF2 and from DOCX files using python-docx.​

**- Ranking Module:** Employs a Generative AI model (Gemini) to analyze and score resumes against the provided job description.​

**1. Component Diagram or Logical Architecture Diagram:**



**2. High-Level Flow-Based System Architecture Diagram:**



**3. Workflow Architecture Diagram:**



**b. Data Flow***:*​  
1. User uploads resumes or provides Google Drive file IDs via the Streamlit interface.​

2. The system retrieves and extracts text from these resumes.​ Resumes are processed using PyPDF2 or python-docx.

3. Job description is inputted by the user.

4. Each resume is passed to the Gemini model through a LangChain agent.

5. The model analyzes and scores each resume.

6. Results are displayed back on the Streamlit interface.​

**2. Front-End Development**

**Frameworks / Libraries**

The front-end is developed using Streamlit, a Python library that simplifies the creation of interactive web applications.

**The interface includes**:​  
1. File uploader for multiple resumes.​  
2. Text area for job description input.​  
3. Input field for Google Drive file IDs.​  
4. A button to trigger the resume analysis process.

**3. Back-End & API Integration**

**Storage Integration:**

The system integrates with Google Drive using the Google Drive API. Authentication is managed via service account credentials, ensuring secure access. It's crucial to handle these credentials securely and follow best practices for authentication.​  
  
**Document Processing:**

**- PDFs:** Text extraction is performed using PyPDF2. However, for more complex PDFs, consider using pdfminer.six for more accurate text extraction.  
**- DOCX:** Text extraction is handled using python-docx. For more comprehensive extraction, including headers and footers, python-docx2txt can be utilized.  
  
**4. NLP & Agentic Search Querying**

**NLP Libraries:**  The system leverages Google's Generative AI model (Gemini) to:​  
- Analyze the content of resumes and job descriptions.​  
- Identify key skills and experiences.​  
- Provide a compatibility score and suggestions for improvement.​

**Natural Language Processing:**  
- Uses Gemini model to understand and compare resume and JD.  
- Provides suggestions and missing skills.

**LangChain Agent:**  
- StructuredTool with input schema  
- Returns AI-generated analysis

**5. Ranking Algorithm**

The AI model generates a match score based on the alignment between the resume and job description. The results are then sorted based on this score to present the most compatible resumes at the top. Regex extracts score from AI response. Results sorted by score.  
  
**6. Development Best Practices**

**Documentation & Comments:**

Ensure that the codebase is well-documented with clear comments explaining the purpose and functionality of each component. A comprehensive README should be provided, detailing setup instructions, dependencies, and usage guidelines.​  
  
**Testing & Error Handling:**

Implement unit tests for critical functions such as text extraction and API integrations. Incorporate robust error handling, especially for external API calls, to manage exceptions gracefully and provide informative error messages to users.​

**7. File Structure**  
resume\_matcher/  
├── app.py  
├── utils/  
│   ├── parser.py  
  
│   ├── gdrive.py  
  
│   └── scoring.py  
  
├── prompts/  
  
│   └── compare\_prompt.txt  
  
├── credentials/  
  
│   └── service\_account.json  
  
├── requirements.txt  
  
└── README.md