



Software Requirements Specification  
Version 1.0

# AI Resume Ranker

Theme: Smart Resume Ranker for Recruiters

Project Name: AI Resume Ranker

Category: AI and Machine Learning Mania



# Table of Contents

1.1	2
1.2	3
1.3	6
1.4	6
1.5	7
1.6	7
1.7	8
1.8	<b>Error! Bookmark not defined.</b>
1.8.1	10
1.8.2	10
1.9	11

## 1.1 Background and Necessity for the Application

In the current highly competitive employment landscape, hiring managers are often inundated with numerous resumes for a single position, each featuring distinct formatting and content. Traditional manual resume screening approaches are time-consuming, labor-intensive, and susceptible to human error, potentially impeding the efficient selection of suitable candidates. However, with progress in Natural Language Processing (NLP) and Machine Learning technologies, automated systems have emerged as practical solutions for resume parsing, information extraction, and matching applicants to job requirements.

**AI Resume Ranker** employs a blend of NLP techniques and machine learning algorithms to recognize crucial sections within resumes, including applicant's name, contact details, skills, educational history, and professional experience. By combining rule-based methods with machine learning, the tool ensures high precision in information extraction, even when dealing with diverse resume layouts. Furthermore, it filters resumes according to specific recruiter criteria, facilitating more efficient candidate selection.





## 1.2 Proposed Solution

The architecture of the proposed AI-powered resume ranking system is designed to automate the candidate shortlisting process by leveraging NLP and intelligent ranking algorithms. The goal of the system is to accurately match resumes with job descriptions provided by recruiters, thereby reducing manual effort and ensuring efficient candidate selection.

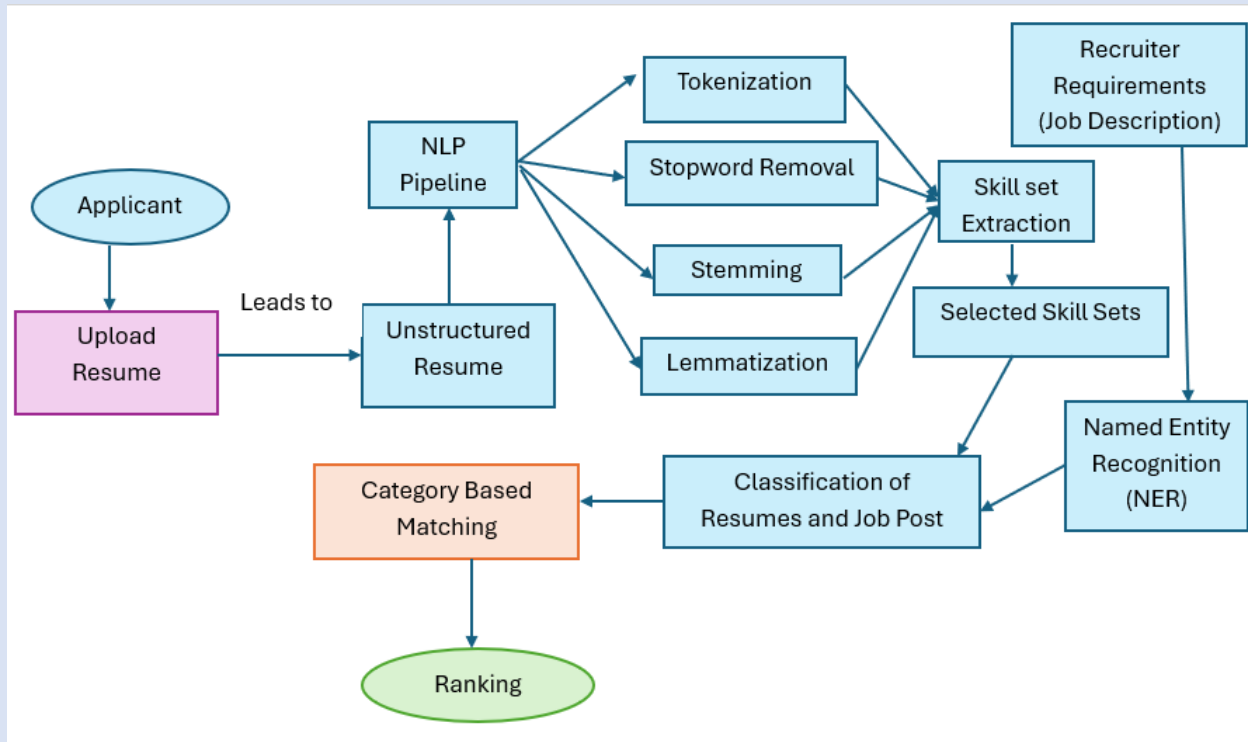
The process begins when an applicant uploads a resume, which is initially treated as unstructured data. This unstructured resume is passed into the NLP pipeline where several preprocessing techniques are applied, including tokenization, stopword removal, stemming, and lemmatization. These steps help to normalize and prepare the resume text for further analysis.

Following pre-processing, the system performs skill set extraction to identify and extract technical and domain-specific skills from the resume content. These extracted skills are then, organized into a set of selected skills. At the same time, recruiter requirements or job descriptions are processed using Named Entity Recognition (NER) to identify key expectations such as required skills, experience levels, and job roles.

The selected skill sets from the resumes and the extracted information from job descriptions are then, passed to a classification module. This component classifies resumes based on their relevance and alignment with job posts. In parallel, a category-based matching technique is used to compare resumes and job categories to ensure only the most relevant resumes are ranked higher.

Finally, the outputs of the classification and category-matching modules are integrated into a ranking system, which generates a list of the most suitable candidates. This ranked list assists recruiters in making informed decisions efficiently by presenting the best-fit applicants based on both skill relevance and category alignment.

The sample architecture for AI Resume Ranker application is as follows:



### *Sample Architecture of the Application*

#### **Dataset and Features**

The dataset for this project is provided to you. The dataset used in this project comprises 228 Word documents (.docx), each representing an individual resume. These resumes vary in size and content, ranging from 24 KB to 90 KB. It includes candidates' details with diverse professional backgrounds such as Full Stack Developers, Business Analysts, Project Managers, and Software Engineers.

Each document contains unstructured textual data, including personal details, technical and soft skills, educational background, certifications, and professional experience. This dataset serves as the foundation for building an AI-powered system that analyzes, ranks, and matches resumes against specific job roles or criteria using NLP and Machine Learning techniques.

Steps to build the model for AI Resume Ranker are as follows:

1. **Resume Dataset Ingestion:** The system begins with the ingestion of resumes in JSON format. These structured documents contain key applicant information, including education, work experience, skills, and certifications. The JSON format facilitates easy parsing and information extraction during subsequent stages.



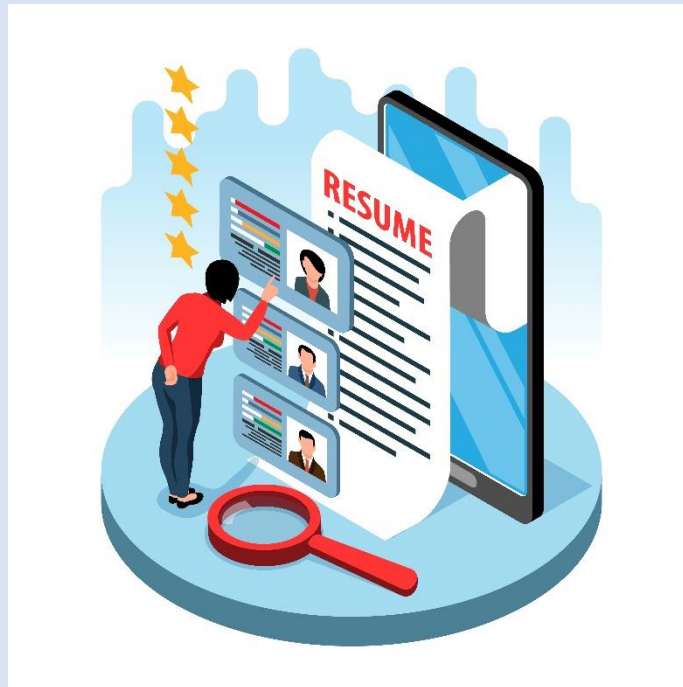
2. **Text Pre-processing:** The pre-processing stage is critical for cleaning and normalizing the input text data. It involves several sub-steps:
  - **Tokenization:** Each resume is broken down into tokens, which are the smallest units of text (example, words or symbols).
  - **Stopwords Removal:** Common words that do not contribute to the semantic meaning (such as 'is', 'the', or 'at') are removed to reduce noise and dimensionality.
  - **Stemming and Lemmatization:** Words are reduced to their base or root form. Stemming applies rule-based reduction (example, 'running' to 'run'), while lemmatization uses vocabulary and grammar (example, 'better' to 'good').

This step enhances the accuracy of matching and recognition tasks.

3. **Named Entity Recognition (NER):** The cleansed and normalized tokens are passed through a NER module. NER is an NLP technique that identifies key entities in text, such as skills (example, Python), roles (example, Data Analyst), education, and company names.
4. **Skillset-Based Resume Extraction:** After entities have been recognized, the system focuses on skillset extraction. This phase filters resumes by identifying and isolating relevant skills and technical competencies required for the job. This significantly narrows down the candidate pool to only those who meet the core technical criteria specified by the employer.
5. **Matching Engine:** Once both resumes and job descriptions have been transformed into structured representations (based on extracted skills and entities), a matching algorithm compares two. The engine calculates a matching score based on keyword overlap, semantic similarity, and contextual alignment between job responsibilities and experience. This score indicates the level of relevance of each candidate to the job profile.
6. **Scoring, Ranking, and Shortlisting:** The scores generated by the matching engine are used to score and rank. Select the top candidates who meet or exceed a predefined threshold or simply pick the highest-ranked applicants for final review by recruiters.
7. **Output to Recruiters** The system outputs a ranked and filtered list of candidates who most closely match the job requirements. This allows



recruiters to focus only on high-potential candidates, drastically reducing time-to-hire and ensuring merit-based selection.



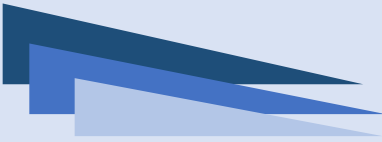
## 1.3 Purpose of the Document

This document outlines the development plan for the AI Resume Ranker using AI and ML techniques. It ensures stakeholders have a common understanding, promotes effective communication, and equips the team to deliver a high-quality application.

Key areas include recruiters focusing only on high-potential candidates, drastically reducing time-to-hire and ensuring merit-based selection. This document is intended for both stakeholders and developers.

## 1.4 Scope of Project

The scope of the project includes developing a complete Machine Learning pipeline for AI Resume Ranker. The project aims to automate and streamline the candidate shortlisting process by analyzing resumes and matching them to job descriptions using NLP techniques. It extracts key information such as skills, roles, and qualifications, calculates a relevance score, and presents a ranked list



of top candidates through a user-friendly interface. The system also supports recruiter feedback to improve accuracy over time.



## 1.5 Constraints

The project may face several constraints, such as limited support for non-English resumes and strict reliance on well-structured PDF or DOCX formats. Additionally, system accuracy depends heavily on the quality of job descriptions and available computational resources. Ensuring data privacy and avoiding bias are also critical limitations to consider.

## 1.6 Functional Requirements

This comprehensive project aims to provide an effective tool for predicting earth temperature, leveraging advanced ML techniques to ensure accessibility and scalability.

Functional requirements are explained as follows:

- i. **Resume Upload** - Users (recruiters or admins) should be able to upload single or multiple .docx resume files through the web interface or application.
- ii. **Job Description Input** – Users should be able to input or upload a Job Description (JD) against which the resumes will be evaluated.







**iii. Candidate Scoring, Ranking, and Shortlisting** – The system should use the matching engine scores to rank candidates and automatically shortlist those who meet or exceed a predefined threshold, enabling recruiters to review only the highest-ranked applicants.

**iv. Search and Filter** – Users should be able to search candidates by keywords, filter by ranking score, years of experience, specific skills, or qualifications.



**v. Resume Parsing** – The system should isolate resumes that contain job-relevant skills and competencies. It must extract key technical skillsets from resumes to align with the employer's requirements. Only candidates meeting core technical criteria should be retained for further analysis.

**vi. Ranked Resume Display** – The system should output a ranked and filtered list of top-matching candidates, helping recruiters focus on high-potential profiles, reduce time-to-hire, and support merit-based selection.



**vii. Resume Download or View** – Provide options to view or download the original resume documents of shortlisted candidates.

**viii. Export Results** – Enable export of ranked candidate lists into Excel or PDF format for offline review or reporting.

**ix. User Interface** – The system should offer an intuitive interface for uploading multiple resumes and a job description, then display a ranked list of top-matching candidates with highlighted skills, experience, and relevance scores.

In addition, the system should implement a feedback loop that captures recruiter's decisions (example, hired or rejected) to continuously retrain and enhance the model's accuracy over time.



## 1.7 Non-Functional Requirements

There are several non-functional requirements that should be fulfilled by the application.

1. **Performance:** The system should process and rank resumes in under five seconds per job description to ensure smooth recruiter workflow.
2. **Scalable:** The system should be capable of handling bulk uploads (example, 500+ resumes) without significant performance degradation.
3. **Usable:** The interface should be intuitive and user-friendly for non-technical HR users, requiring minimal training.
4. **Secure:** All uploaded resumes and job data should be securely stored and processed with access control and encryption to protect sensitive personal information.
5. **Compatible:** The tool should support commonly used file formats such as PDF and DOCX and be accessible across major browsers and devices.



These are the bare minimum expectations from the project. It is a must to implement the **FUNCTIONAL** and **NON-FUNCTIONAL** requirements given in this SRS.

Once they are complete, you can use your own creativity and imagination to add more features if required.



## 1.8 Interface Requirements

### 1.8.1 *Hardware*

---

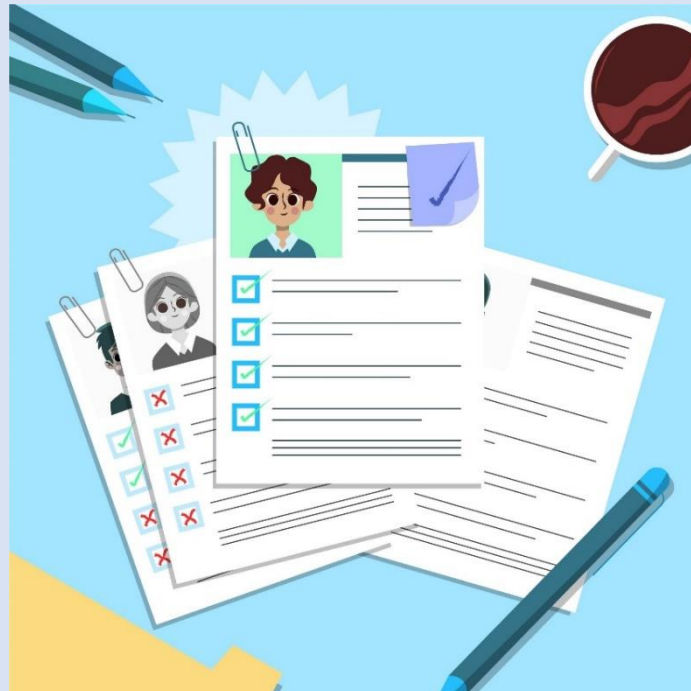
Intel Core i5/i7 Processor or higher  
8 GB RAM or higher  
Color SVGA monitor  
500 GB Hard Disk space  
Mouse and Keyboard

### 1.8.2 *Software*

---

Technologies to be used:

1. **Frontend:** HTML5/Streamlit or any other frontend programming languages
2. **Backend:** Flask/Django
3. **Data Store:** JSON/TXT/CSV/Word/PDF
4. **Programming/IDE:** R/Python, Jupyter Notebook, Anaconda, Google Colab
5. **Libraries:** python-docx, PyPDF2, NER, NLTK, regex, scikitlearn, NumPy, Pandas, Matplotlib





## 1.9 Project Deliverables

You will design and build the project and submit it along with a complete project report that includes:

- Problem Definition
- Design specifications
- Diagrams such as Dialog Flow
- Test Data Used in the Project
- Project Installation Instructions
- Proper Steps to execute the project
- Link of GitHub for accessing the uploaded project code (Link should have public access)
- Link of published blog

The source code, including .ipynb files for Jupyter Notebook and Google Colab, should be shared via GitHub. Appropriate access permissions should be granted to users to allow testing for Jupyter Notebook and Google Colab. The consolidated project must be submitted on GitHub with a ReadMe.doc file listing assumptions (if any) made at your end.

Please provide the GitHub URL where the project has been uploaded for sharing. The repository on GitHub should have public access. Documentation is a very important part of the project; hence, all crucial aspects of the project must be documented properly. Ensure that documentation is complete and comprehensive.

You should publish a blog of minimum 2000 words on any free blogging Website such as Blogger, Tumblr, Ghost, or any other blogging Website. The link of the published blog should be submitted along with the project documentation.

**Do NOT copy content or code from GPTs or other AI tools, although you are permitted to use images generated by AI tools for any visual representation purposes. It is mandatory to mention such tools used in case you add any AI generated images.**

**Submit a video (.mp4 file) demonstrating the working of the application, including all the functionalities of the project. This is MANDATORY.**

*~~~ End of Document ~~~*