Product Requirements Document (PRD)

Project Title: AI-powered Resume Analyzer & Job Match Recommender

1. Project Overview

The AI-powered Resume Analyzer & Job Match Recommender is a web-based application that leverages advanced Natural Language Processing (NLP) to automatically analyze user resumes, extract skills and qualifications, and recommend best-fit job roles. The platform utilizes transformer-based models for semantic matching and provides users with interactive skill-gap analyses, actionable feedback, and real-time job recommendations-all within an intuitive web interface.

2. Objectives

- Automate resume parsing and skill extraction with high accuracy
- Match candidates to relevant job roles using semantic similarity and standardized job databases
- Visualize skill gaps and provide actionable recommendations
- Enhance user experience with an interactive, real-time web application
- Demonstrate proficiency in data science, NLP, full-stack development, and MLOps best practices

3. Target Users

- Job seekers (students, recent graduates, professionals)
- Career counselors and educational institutions
- Recruiters and HR professionals (for demo/pilot purposes)
- Academic evaluators (as a capstone project)

4. Key Features & Functional Requirements

4.1. Resume Upload & Parsing

- F1.1: Support for PDF and DOCX resume upload through a secure web UI (Streamlit).
- F1.2: Extract raw text from uploaded resumes using pdfplumber (PDF) and python-docx (DOCX).
- F1.3: Parse key resume entities using spaCy NER and/or transformer-based models: Name, contact details, education, work experience, skills, certifications

4.2. Preprocessing & Data Cleaning

- F2.1: Clean and normalize extracted text (remove noise, handle encoding, correct spelling).
- F2.2: Tokenize, lemmatize, and remove stopwords.
- F2.3: Map extracted skills and job titles to standardized vocabularies (e.g., O*NET).

4.3. Embedding & Semantic Representation

- F3.1: Generate sentence/document embeddings for resumes and job descriptions using BERT/Sentence-BERT.
- F3.2: Store embeddings for efficient similarity search and retrieval.

4.4. Job Database Management

- F4.1: Integrate a structured database (CSV or SQL) of job titles and required skills (O*NET or custom).
- F4.2: Store job descriptions, industry, required skills, and average salary info.
- F4.3: Allow for periodic updates or enrichment from public APIs.

4.5. Matching & Recommendation Engine

- F5.1: Compute semantic similarity between candidate resume and job roles using cosine similarity over embeddings.
- F5.2: Rank top N job roles based on skill and experience fit.
- F5.3: Identify skill gaps for each recommended job.
- F5.4: Provide improvement suggestions (e.g., 'Consider acquiring XYZ skill for better fit').

4.6. Real-Time Job Postings Integration (Bonus)

- F6.1: Fetch live job postings from APIs (Indeed, LinkedIn, or scraped demo data).
- F6.2: Filter and recommend active job listings matching the candidate’s top roles.

4.7. Visualization & Reporting

- F7.1: Interactive dashboards to show: Top recommended jobs (with confidence scores), Radar/spider charts for skill gap analysis, Highlighted matched and missing skills, Job role details and career insights.
- F7.2: Option to download a personalized report (PDF/HTML).

4.8. User Experience (UX) & Security

- F8.1: Responsive, intuitive UI with clear instructions and real-time feedback.
- F8.2: Secure handling of uploaded documents (temporary, no persistent storage unless user agrees).
- F8.3: Accessibility compliance for all major user groups.

4.9. Deployment & DevOps

- F9.1: Containerize the app using Docker for reproducibility.
- F9.2: Deploy to Streamlit Cloud, AWS EC2, or similar.
- F9.3: Version control with GitHub; implement CI/CD pipelines (GitHub Actions).
- F9.4: Automated testing for major modules.

5. Non-Functional Requirements

- Performance: Resume analysis and job recommendations returned within 5 seconds for typical use cases.
- Scalability: Supports multiple users simultaneously (up to 20 for academic demo).
- Reliability: 99% uptime during demonstration windows.
- Maintainability: Modular codebase with thorough documentation and tests.
- Privacy: No resumes stored after session ends unless explicitly saved by user.

6. System Architecture Overview

Components:

- Frontend: Streamlit UI (file upload, results dashboard, user interaction)
- Backend: Resume Parser (pdfplumber, spaCy, python-docx); Data Preprocessing module; Embedding Model (HuggingFace Transformers); Matching & Recommendation Engine; Job Database (CSV/SQL, O*NET)
- Visualization: Plotly/Matplotlib charts embedded in UI
- DevOps: Docker for containerization, GitHub Actions for CI/CD, Cloud for deployment

Workflow:

1. User uploads resume via Streamlit UI.
2. Backend extracts and processes resume content.
3. Embeddings generated and compared to job database.
4. Top job matches and skill gaps computed.
5. Results visualized and shown to user.
6. Optional: Fetch and display live job postings.

7. Milestones & Timeline (Suggested for Capstone/Portfolio)

1. Week 1: Research, data sourcing, and requirements finalization
2. Week 2: Resume parser and basic UI prototype
3. Week 3: NLP pipeline, embedding integration, initial job matching
4. Week 4: Dashboard/visualization features and reporting
5. Week 5: MLOps setup, Docker, deployment, and user testing
6. Week 6: Documentation, polish, demo, and final submission

8. KPIs & Success Criteria

- Resume parsing accuracy: >90% for standard formats
- Skill extraction F1-score: >85%
- Job recommendation relevance: >85% match in pilot tests
- User satisfaction (demo feedback): 8/10 or higher
- App response time: <5 seconds for analysis

9. Risks & Mitigation

- Parsing errors with non-standard resumes: Use multiple extraction libraries; fallback to manual correction for demo.
- API access limits for job postings: Use cached/demo job data if live access fails.
- Large model performance on cloud: Optimize by using lightweight models (DistilBERT) or batch inference.

10. Deliverables

- Complete codebase (Python, Streamlit, Docker)
- Architecture diagram (mermaid.js/DiagramGPT)
- Sample resumes and test cases
- Live demo or deployed link
- User guide and technical documentation (README)
- PRD (this document)

11. References

- HuggingFace Transformers: https://huggingface.co/transformers/
- spaCy Documentation: https://spacy.io/
- pdfplumber: https://github.com/jsvine/pdfplumber
- O*NET Job Database: https://www.onetonline.org/
- Streamlit: https://streamlit.io/