Resume Match Checker – Project Specification (Updated 30 October)

Project Name: ResumeSync
Developer: Shamim Shakil
Version: 1.0 Specification (Revised on 30 October)

Changes and updates marked in color coded comments
Implemented
Updated Plan
Aborted
Thoughts/Experience

General Description

ResumeSync is an intelligent resume–job matching tool that helps job seekers evaluate how well their resumes align with specific job postings. Unlike basic keyword matchers, ResumeSync leverages AI to highlight strengths, weaknesses, and missing skills, providing actionable improvement suggestions.

Based on advisor feedback, the **initial focus (Version 1)** will be a **command-line prototype** that demonstrates the core workflow:

1. Scrape real job postings from LinkedIn with JobSpy.(implemented)
2. Allow the user to select one job from the list. (implemented)
3. Extract text from an uploaded resume (PDF). (merged this with open AI comparison section)
4. Use OpenAI API to compare resume text with job description. .(implemented)
5. Print results in the terminal (strengths, weaknesses, skill gaps, and optional match score). (Added a simplified version of it)

Future versions (Version 2+) will expand to a Streamlit-based interface with dashboards, history, and export functionality.

External Mechanisms

- **JobSpy library** – Scrape LinkedIn job postings (results in a Pandas DataFrame or CSV).
- **PyPDF2** – Extract resume text from uploaded PDFs.
- **OpenAI API** – Perform AI-powered text comparison and generate structured analysis.
- **Streamlit (Version 2)** – Build a simple, accessible user interface.

Version 1 (Command-line Prototype)

Features

- Prompt user for a job search term (e.g., “UX Designer”).
- JobSpy scrapes 10–50 job postings. (5 jobs are visible at a time to avoid LinkedIn rate limitation)
- Display results in a numbered list for user selection.
- Accept a resume file (PDF) and extract text.
- Send both resume text and job description to OpenAI API.
- Display analysis in terminal: (implemented)
    - Strengths (skills/keywords present).
    - Weaknesses or missing skills.
    - Recommendations for resume improvement.
    - Optional numeric match score (1–10).

Version 2 (Enhanced Web Interface)

- Streamlit app with drag-and-drop resume upload.
- Job search through form input.
- Analysis results displayed with visual dashboards.
- Bulk resume analysis + history tracking.
- Export results to PDF report.

Task Vignettes

Vignette 1: Job Search

Sarah enters “UX Designer.” ResumeSync scrapes 50 LinkedIn postings and displays them as a numbered list. Sarah chooses job #3 (“UX Designer at Apple – Cupertino, CA”).
(User can view 5 Job Posting)

Vignette 2: Resume Upload

Sarah uploads her PDF resume. The system extracts text and previews the first 200 characters for confirmation. (avoided for complexity in CLI, in background it matches the keywords from resume/cv with Job descriptions keywords)

Vignette 3: AI Analysis

Sarah runs the analysis. After a short delay, results appear in her terminal:

- Match Score: 7.2/10
- Strengths: prototyping, user research
- Missing Keywords: accessibility, WCAG, design systems
- Recommendations: Add accessibility-related projects

(implemented)

Vignette 4: Export (Future Version)

In Version 2, Sarah will be able to save her analysis history and export PDF reports.

Technical Flow (V1)

User input (job search term)

↓

JobSpy scrape → jobs.csv User Select Location→ list of jobs (total 5)

↓

User selects job by number

↓

Resume upload → PyPDF2 → text extraction

↓

OpenAI API (resume text + job description)

↓

Output: strengths, weaknesses, gaps, score

Core Components

1. **Job Scraping Module**
   - Function: scrape_jobs(search_term, location, count=10) (current count =5)
   - Output: DataFrame/CSV (in CLI, rendered text) with job_url, title, company, location.
2. **Resume Processing Module**
   - Function: extract_resume_text(uploaded_file)
   - Output: Cleaned resume text string. (this operation done in background)
3. **AI Analysis Module** (implemented)
   - Function: analyze_resume_match(resume_text, job_description)
   - Output: dictionary with score, strengths, gaps, recommendations.
4. **CLI Interface** (implemented)
   - Input: terminal prompts for job search, selection, and resume path.
   - Output: printed analysis results.

Self-Assessment

- **Change from initial sketch:** Start with CLI instead of UI (Streamlit deferred to Version 2).
- **Confidence:** 7.5/10 — scraping + text extraction manageable; prompt engineering needs refining. (AI part don't give me hassle, job description extraction from private link/logged in link, indeed, zip recruiter, Glass door has been aborted due to restriction by the provider, no public job link available to scrap in current scope)
- **Biggest challenge:** OpenAI API costs and handling errors.
- **Areas needing help:** Prompt design, JobSpy setup, caching API responses. (to keep the codebase lean, to make it more responsive in GUI)
- **Future direction:** Expand to web app, add history and optimization dashboard.