Project Design & Scaffolding Report

Project: AI-Powered Human Resources & Career Management Platform

---

Section 1: Business Problem Breakdown & Analysis

1. **Problem Statement:**
   The business requires an AI-powered platform to serve two distinct user groups: individual job seekers and enterprise HR professionals. For job seekers, the system must assist with resume refinement, tailoring resumes to specific job descriptions, and preparing for interviews. For HR professionals, the system must automate critical tasks like candidate screening and ranking against job posters, summarizing candidate profiles, and generating tailored interview questions. The solution needs to handle various document formats (PDF, DOCX, Markdown, URLs) and perform complex, multi-step reasoning and generation tasks.
2. **LLM Suitability Analysis:**
   This problem is an excellent candidate for an LLM-based solution. The core tasks revolve around processing, understanding, comparing, and generating unstructured text (resumes, job descriptions, interview questions). LLMs excel at these natural language understanding (NLU) and natural language generation (NLG) tasks. The complexity, involving cross-referencing multiple documents, extracting specific entities (like skills and experience), and generating highly contextual content (like tailored resume sections or interview questions), goes beyond the capabilities of traditional rule-based software and is perfectly suited for the reasoning power of modern agentic LLM architectures.
3. **Core Tasks:**
   A. Job Seeker Functions:
   i. **Resume Ingestion & Analysis:** Parse and understand the content of an uploaded resume (PDF, DOCX).
   ii. **Resume Refinement:** Improve the clarity, grammar, and impact of the resume's content.
   iii. **Content Generation:** Create behavioral interview questions and corresponding answers based on the user's resume.
   iv. **Document Transformation:** Convert a resume into different formats

or styles (e.g., formal CV, personal website bio).

v. **Resume Tailoring:** Analyze one or more job descriptions and rewrite sections of the user's resume to align with the required skills and experience.

B. HR Professional Functions:

i. **Job Poster Ingestion:** Ingest and parse job posters from PDF or Markdown files.

ii. **Bulk Resume Ingestion:** Process a batch of candidate resumes (PDF, DOCX).

iii. **Candidate-to-Job Matching & Ranking:** Score and rank candidates based on their suitability for a given job poster.

iv. **Deep Profile Summarization:** Summarize resumes, including following hyperlinks within them (using the Deep Research Plugin) to gather additional context and generate a concise headline.

v. **Interview Kit Generation:** Create a comprehensive set of behavioral and technical interview questions based on both the candidate's resume and the job description.

Section 2: Recommended Solution Architecture

1. **Recommended Tool:**
   MCP (Multi-agent Collaboration Protocol) with the optional **Deep Research Plugin**.
2. **Justification:**
   An **MCP** architecture is the most appropriate choice for this platform due to the high complexity, the need for dynamic task execution, and the clear separation of concerns between different functions.
   - **Against Standard RAG:** A simple RAG system would be insufficient. While RAG is good for answering questions about a knowledge base (e.g., "Find resumes with Python experience"), it cannot perform the core generative and transformative tasks required here, such as rewriting a resume, generating interview questions, or ranking candidates based on a complex set of criteria. The problem is not just retrieval; it's about reasoning, creation, and collaboration.

- **Why MCP over a simple Agentic Workflow:** While an Agentic Workflow is a viable option, an MCP provides a more robust, scalable, and formalized framework. Given the two distinct user personas (Job Seeker, HR) and the variety of complex workflows, a structured protocol is superior. MCP allows for the creation of well-defined, reusable "specialist" agents (e.g., `ProfileAnalysisAgent`, `CandidateScoringAgent`) that can be orchestrated by a `Coordinator` agent in various combinations to fulfill diverse requests. This is more manageable and extensible than a hard-coded chain of agents.

3. **Detailed Workflow (Example: Candidate Matching & Ranking):**
   This workflow demonstrates how the agents collaborate under the MCP framework.
   i. **Initiation:** An HR user submits a Job Poster (PDF) and a list of candidate resumes (e.g., 10 PDFs) to the system's endpoint.
   ii. **Coordination:** The **HR Coordinator Agent** receives the request. It creates a new job task and breaks it down.
   iii. **Parallel Ingestion:** The `Coordinator` dispatches tasks to multiple instances of the **DocumentIngestionAgent**.
      - One instance processes the Job Poster PDF.
      - Ten instances process the ten candidate resume PDFs in parallel.

   iv. **Parallel Analysis:**
      - The parsed text of the Job Poster is sent to the **JobDescriptionAnalysisAgent**, which extracts key requirements, skills (required vs. nice-to-have), and experience levels into a structured JSON object.
      - The parsed text of each resume is sent to the **ProfileAnalysisAgent**. This agent extracts structured data (work history, skills, education). If a resume contains a link to a portfolio or blog, it can invoke the **Deep Research Plugin** to browse the URL and add a summary to the candidate's profile.

   v. **Scoring & Ranking:** The `Coordinator` gathers all structured profiles. For each candidate, it sends their structured profile and the

structured job description to the **CandidateScoringAgent**. This agent compares the two, calculates a suitability score, and provides a detailed text rationale for its score.

vi. **Synthesis & Output:** The `Coordinator` receives scores for all candidates. It sorts them from highest to lowest and synthesizes the results into a final JSON report.

vii. **Delivery:** The final report, containing the ranked list of candidates with their scores and rationales, is delivered to the HR user.

Section 3: Input and Output Specification

1. **Input Documents:**
   - **Types:** PDF, DOCX, Markdown (.md), and web URLs (for job postings or candidate portfolios).
   - **Sample Input (Resume Snippet in Markdown):**

```
# John Doe
Principal Software Engineer | San Francisco, CA | john.doe@email.com

## Summary
Innovative and results-driven engineer with 12+ years of experience

## Experience
**Big Tech Inc.** - *Principal Engineer* (2018 - Present)
- Led the 'Project Titan' team to re-architect our monolithic billing
- Designed and implemented a multi-tenant RAG system using LangChain,

## Skills
- **Programming:** Python (Expert), Go (Expert), TypeScript (Proficie
- **Cloud:** AWS, GCP, Kubernetes, Docker, Terraform
- **Databases:** PostgreSQL, Redis, Pinecone
```

2. **Output Deliverable:**
   - **Format:** The primary output for the candidate ranking task will be a JSON object. For resume refinement, it will be a DOCX file. For interview questions, a Markdown file.

- **Structure & Content (Example: Candidate Ranking JSON Output):**

```
{
  "job_poster_id": "JP-2024-0515",
  "job_title": "Senior AI Engineer",
  "summary": "Ranking of 5 candidates for the Senior AI Engineer role
  "ranked_candidates": [
    {
      "rank": 1,
      "candidate_name": "Jane Smith",
      "source_file": "jane_smith_resume.pdf",
      "match_score": 95,
      "rationale": "Excellent match. Jane has 8 years of direct exper
      "key_strengths": ["RAG Systems", "Python", "AWS", "Team Leaders
      "potential_gaps": ["No direct experience with GCP, which is a "
    },
    {
      "rank": 2,
      "candidate_name": "John Doe",
      "source_file": "john_doe_resume.pdf",
      "match_score": 88,
      "rationale": "Strong candidate. Exceeds experience requirements
      "key_strengths": ["Distributed Systems", "Python", "Go", "Cloud
      "potential_gaps": ["Focus has been on billing systems, not expl
    }
  ]
}
```

Section 4: Python Scaffolding (Tenant & ElementTemplates)

1. **Tenant Configuration:**

```python
from your_models_file import TenantConfiguration, TenantType, TaskType

# Assuming 'HUMAN_RESOURCES_AUTOMATION' is added to the TenantType Enum
# class TenantType(str, Enum):
#     HUMAN_RESOURCES_AUTOMATION = "HUMAN_RESOURCES_AUTOMATION"
#     ...

hr_tenant_config = TenantConfiguration(
    tenant_type=TenantType.HUMAN_RESOURCES_AUTOMATION,
    display_name="Human Resources & Career Management",
    description="A suite of tools for job seekers and HR professionals to manag
    default_task_type=TaskType.MCP,
    auto_provision_templates=True,
    is_active=True,
    created_by="AI_SOLUTIONS_ARCHITECT_LLM"
)

# Code to save this to the database
# await hr_tenant_config.insert()
```

2. **Element Templates:**

```
from your_models_file import ElementTemplate, TenantType, TaskType, ElementType

# --- Template 1: For the ProfileAnalysisAgent ---
profile_extractor_template = ElementTemplate(
    name="Resume_to_Structured_JSON_Extractor",
    description="Parses the raw text of a resume and extracts key information i
    tenant_type=TenantType.HUMAN_RESOURCES_AUTOMATION,
    task_type=TaskType.MCP,
    element_type=ElementType.PROMPT_TEMPLATE,
    generation_prompt="""
    You are an expert HR analyst AI. Your task is to meticulously parse the pro

    **Constraints:**
    - Adhere strictly to the JSON schema provided in the 'OUTPUT FORMAT' sectio
    - Do not invent information. If a section is missing from the resume, use a
    - Extract skills and categorize them if possible (e.g., 'programming_langua
    - Identify any URLs, especially for portfolios, GitHub, or personal blogs,

    **Context:**
    - Resume Text:
    ```{resume_text}```

    **Output Format (JSON):**
    ```json
    {{
      "contact_info": {{
        "name": "...",
        "email": "...",
        "phone": "...",
        "location": "..."
      }},
      "summary": "...",
      "external_links": ["...", "..."],
      "work_experience": [
        {{
          "company": "...",
          "job_title": "...",
          "start_date": "...",
          "end_date": "...",
```

```
          "responsibilities": ["...", "..."]
        }}
      ],
      "education": [
        {{
          "institution": "...",
          "degree": "...",
          "graduation_date": "..."
        }}
      ],
      "skills": {{
        "programming_languages": [],
        "cloud_technologies": [],
        "databases": [],
        "other_tools": []
      }}
    }}
    ```
    """,
    retrieval_prompt="Extract structured JSON from resume text",
    variables=["resume_text"],
    execution_config={"model": "gpt-4-turbo", "temperature": 0.1, "response_for
    tags=["hr", "parsing", "resume", "json"],
    status=ElementStatus.ACTIVE,
    created_by="AI_SOLUTIONS_ARCHITECT_LLM"
)

# --- Template 2: For the CandidateScoringAgent ---
candidate_scorer_template = ElementTemplate(
    name="Candidate_to_Job_Scorer",
    description="Scores and ranks a candidate's profile against a job descripti
    tenant_type=TenantType.HUMAN_RESOURCES_AUTOMATION,
    task_type=TaskType.MCP,
    element_type=ElementType.PROMPT_TEMPLATE,
    generation_prompt="""
    You are a highly experienced Senior Technical Recruiter. Your task is to ev

    Context:
    1. **Job Description (Structured):**
```
```

```
        ```{job_description_json}```
    2.  **Candidate Profile (Structured):**
        ```{candidate_profile_json}```

    **Instructions:**
    1.  **Analyze Alignment:** Compare the candidate's skills and work experien
    2.  **Calculate Score:** Generate a 'match_score' from 0 to 100, where 100
    3.  **Write Rationale:** In 2-3 sentences, provide a concise rationale for
    4.  **Identify Strengths & Gaps:** List the key strengths that align with t

    **Output Format (JSON only):**
    ```json
    {{
      "match_score": <integer>,
      "rationale": "...",
      "key_strengths": ["...", "..."],
      "potential_gaps": ["...", "..."]
    }}
    ```
    """,
    retrieval_prompt="Score candidate profile against job description",
    variables=["job_description_json", "candidate_profile_json"],
    execution_config={"model": "gpt-4-turbo", "temperature": 0.3, "response_for
    tags=["hr", "scoring", "ranking", "recruiting"],
    status=ElementStatus.ACTIVE,
    created_by="AI_SOLUTIONS_ARCHITECT_LLM"
)
```

Section 5: Evaluation Strategy

1. **Human-in-the-Loop Evaluation:**
    - **Methodology:** A panel of senior HR professionals and hiring managers will be presented with a set of 20 job descriptions and 100 anonymized resumes. They will first manually rank the top 5 candidates for each role. Then, they will be shown the AI's generated output (the ranked list and rationales). They will score the AI's output

using the rubric below and provide qualitative feedback on discrepancies. For resume tailoring tasks, they will review the "before" and "after" versions and score the quality of the improvements.

- **Evaluation Rubric (Candidate Ranking):**

Criteria	1 (Poor)	2 (Fair)	3 (Good)	4 (Excellent)
Ranking Accuracy	The AI's ranking significantly differs from the expert's judgment.	The AI's ranking has some correct placements but misses key candidates.	The AI's top 3 are mostly correct, with minor order differences.	The AI's ranking almost perfectly matches the expert's ranking.
Rationale Quality	Rationale is generic, irrelevant, or factually incorrect.	Rationale is plausible but lacks specific details from the documents.	Rationale is specific and correctly cites evidence from the resume/ JD.	Rationale provides deep, insightful analysis a human expert would.
Completeness	The AI fails to identify obvious strengths or critical gaps.	The AI identifies some, but not all, key strengths and gaps.	The AI correctly identifies most key strengths and gaps.	The AI provides a comprehensive list of all relevant strengths and gaps.
Helpfulness	The output is not useful and would be ignored in a real workflow.	The output provides some value but requires heavy editing.	The output is helpful and significantly speeds up the screening process.	The output is highly reliable and could be used with minimal oversight.

2. **LLM-as-a-Judge Evaluation:**
    - **Methodology:** An automated pipeline will be created. For each generated output (e.g., a ranked list or a tailored resume), a separate, powerful LLM (e.g., Claude 3 Opus or GPT-4o) will act as a "judge." The judge will be provided with the original inputs (resume, job description) and the AI system's output. It will then be asked to score the output based on a set of criteria and provide its reasoning, formatted as a JSON object for easy aggregation and analysis.
    - **Judge LLM Prompt:**

```
You are an impartial and meticulous AI evaluation expert, acting as a

**Source Documents:**
1.  **Job Description:**
    ```{job_description}```
2.  **Candidate Resume:**
    ```{resume_text}```

**AI System's Generated Output:**
```{ai_generated_json_assessment}```

**Evaluation Task:**
Please evaluate the AI's generated output based on the following crit

1.  **Factual Accuracy:** Does the rationale and scoring accurately r
2.  **Relevance:** Is the analysis focused on the most important crit
3.  **Depth of Insight:** Does the analysis go beyond simple keyword
4.  **Clarity:** Is the rationale clear, concise, and easy to underst

**Provide your final evaluation in the following JSON format:**
```json
{{
  "evaluation_scores": {{
    "factual_accuracy": <score_1_to_5>,
    "relevance": <score_1_to_5>,
    "depth_of_insight": <score_1_to_5>,
    "clarity": <score_1_to_5>
  }},
  "evaluation_summary": "Provide a brief, one-sentence summary of you
  "strengths_of_assessment": "List one or two things the AI's assessm
  "weaknesses_of_assessment": "List one or two areas for improvement
}}
```

- **Sample Code Snippet:**

```
import openai

client = openai.OpenAI(api_key="YOUR_API_KEY")

def evaluate_output(job_desc, resume, ai_output):
    judge_prompt = f"""
    You are an impartial and meticulous AI evaluation expert...
    [...rest of the prompt from above...]
    **Job Description:**
    ```{job_desc}```
    **Candidate Resume:**
    ```{resume}```
    **AI System's Generated Output:**
    ```{ai_output}```
    ...
    """
    response = client.chat.completions.create(
        model="gpt-4o", # Use a powerful judge model
        messages=[{"role": "user", "content": judge_prompt}],
        temperature=0.0,
        response_format={"type": "json_object"}
    )
    return response.choices[0].message.content

# Example usage:
# evaluation_result = evaluate_output(job_desc_text, resume_text, ai_
# print(evaluation_result)
```

Section 6: User Input

The user has requested the design of an agentic HR platform using the Multi-agent Collaboration Protocol (MCP). The platform should cater to two user types: job seekers and HR professionals. Key features include resume refinement and tailoring for job seekers, and candidate-to-job matching, resume summarization with hyperlink analysis, and interview question generation for HR professionals. The system must handle inputs like PDFs, DOCX, Markdown, and links.
```