System Design Document: CareerOS

Version: 1.0.0
Status: Draft / Implementation Ready
Project Type: Agentic AI / Career Management System

---

1. Executive Summary

CareerOS is an autonomous, agentic AI system designed to function as a long-term career operating system. Unlike static chatbots or simple job aggregators, CareerOS utilizes an **Agentic Cognitive Architecture** to Think, Plan, Act, and Reflect. It persistently tracks user progress, adapts to market feedback (e.g., rejections, interviews), and optimizes a user's career trajectory through continuous reinforcement learning loops.

2. Problem Statement & Scope

2.1 Core Challenge

Early-career professionals and students face a multivariate optimization problem:

- **Information Asymmetry:** Users do not know which skills yield the highest ROI in the current market.
- **Execution Paralysis:** Users struggle to convert long-term goals into daily executable tasks.
- **Feedback Loops:** Traditional platforms do not "learn" from user rejection; they simply present the next job listing.

2.2 Solution Scope

The system will:

1. Ingest and parse user artifacts (Resumes, GitHub, Portfolios).
2. Analyze real-time market data to identify skill gaps.
3. Orchestrate a multi-agent system to plan learning roadmaps and track applications.
4. Implement a memory architecture to store long-term context and adapt strategies based on outcomes.

---

3. High-Level System Architecture

The system follows a **Layered Agentic Architecture**, decoupling the user interface from the reasoning engines and memory stores.

3.1 Architectural Diagram

```
graph TD
    User[User / Client UI] -->|API Request| API[API Gateway / Orchestrator]

    subgraph "Agent Orchestration Layer (LangGraph/CrewAI)"
        API --> Planner[Orchestrator Agent]
        Planner --> Profile[Profile Intelligence Agent]
        Planner --> Market[Job Market Agent]
        Planner --> Roadmap[Roadmap Planning Agent]
        Planner --> Action[Action & App Agent]
        Planner --> Critic[Feedback & Learning Agent]
    end

    subgraph "Memory & Knowledge Layer"
        Profile <--> VectorDB[(Vector Store: Semantic)]
        Market <--> VectorDB
        Planner <--> RelationalDB[(PostgreSQL: Episodic)]
    end

    subgraph "External World"
        Market -->|Scrape/API| JobBoards[Job Boards/GitHub]
        Action -->|Fetch| Courses[Course APIs]
    end
```

3.2 The Agentic Loop (Cognitive Cycle)

The system operates on a continuous control loop, formally defined as:

$$State_{t+1} = f(State_t, Action_t, Observation_t)$$

1. **Think:** Analyze current skills (S_t) vs. Market Demand.
2. **Plan:** Generate a Directed Acyclic Graph (DAG) of tasks (Learning Roadmap).
3. **Act:** Execute tools (Recommend Job, Draft Resume).
4. **Observe:** Monitor Application Status (Interview/Rejection).
5. **Reflect:** Update weights in the strategy (S_{t+1}) based on success/failure.

---

4. Component Design Specifications

4.1 Career Profile Intelligence Engine

- **Role:** The entry point for user data ingestion. Converts unstructured documents into structured JSON profiles.
- **Input:** PDF/DOCX Resume, GitHub URL, LinkedIn export.
- **Process:**
    - **Text Extraction:** `PyMuPDF` for layout-preserved extraction.
    - **NER (Named Entity Recognition):** `spaCy` model fine-tuned on skill taxonomies to extract specific technologies.

    - **Embedding:** Sentence-Transformers to vectorize the user profile for semantic similarity matching.
- **Output JSON:**

```
{
  "profile_id": "uuid",
  "hard_skills": ["Python", "Docker", "FastAPI"],
  "soft_skills": ["Technical Writing", "Agile"],
  "experience_level": "Intermediate",
  "embedding_vector": [0.12, -0.45, 0.88, ...]
}
```

4.2 Job Market Reasoning Agent

- **Role:** Assesses the "value" of a skill or role in the current market.
- **Logic:**
    - Scrapes/ingests job descriptions (JDs) via Scrapy.
    - Calculates **Skill Gap Score**: The vector distance between User_Embedding and JD_Embedding.
    - **Demand Forecasting:** Analyzes frequency of keywords in JDs over the last 30 days.
- **Decision Gate:** If Match_Score > 0.75 $\rightarrow$ Recommend Application. If < 0.75 $\rightarrow$ Recommend Upskilling.

4.3 Personalized Skill Roadmap Planner

- **Role:** Deterministic planner that creates dependency graphs for learning.
- **Tools:** LangGraph for state management, Neo4j (optional) or NetworkX for dependency mapping.
- **Logic:**
    - Identifies missing skills (Set difference: $Target_{skills} - Current_{skills}$).
    - Queries "Learning Knowledge Base" for resources.
    - Output is a time-series schedule (e.g., "Week 1: Learn Pydantic").

4.4 Action Agent (Tracker & Application)

- **Role:** The "Hands" of the system.
- **Capabilities:**
    - **Resume Tailoring:** Uses LLM to rewrite bullet points to match JD keywords using Jinja2 templates + LLaMA-3.
    - **Opportunity Finder:** Runs background cron jobs (Celery) to fetch new hackathons or internships matching the user's readiness score.

4.5 Continuous Learning & Feedback Agent (The Differentiator)

- **Role:** The "Meta-Learner." It does not execute tasks; it evaluates the performance of other agents.
- **Feedback Mechanism:**
    - **Trigger:** User logs an update (e.g., "Rejection from Google").
    - **Analysis:** The agent retrieves the resume used and the job description. It performs a "Post-Mortem" analysis to identify why the match failed (e.g., "Missing System Design experience").
    - **Action:** Updates the **Long-Term Memory** to de-prioritize similar roles until the skill gap is filled.

---

5. Data Architecture & Memory

To support "Agentic" behavior, we utilize a hybrid memory architecture.

5.1 Semantic Memory (Vector Database)

- **Technology:** ChromaDB or FAISS.
- **Content:**
    - Skill Embeddings (ESCo / O*NET taxonomy).
    - Job Description Embeddings.
    - Course/Project Content Embeddings.

5.2 Episodic Memory (Relational Database)

- **Technology:** PostgreSQL.
- **Content:**
    - **Application Logs:** History of every job applied to.
    - **Interactions:** User feedback, agent decisions.
    - **Outcomes:** Status (Applied, Interview, Rejected, Offer).

5.3 Procedural Memory (Rule Store)

- **Technology:** JSON/YAML Rules Engine.
- **Content:** "If/Then" heuristics learned over time (e.g., "If user fails 3 Python interviews, trigger 'Advanced Python' module").

---

6. Technology Stack (Open Source / Free Tier)

Component	Technology Choice	Justification
LLM Inference	**Ollama (LLaMA-3-8B / Mistral)**	Local, free, high privacy.
Orchestration	**LangGraph / CrewAI**	State-of-the-art for cyclic agent flows.
Backend API	**FastAPI (Python 3.11)**	High performance, native async support.
Vector Store	**ChromaDB**	Open source, runs locally or in Docker.
NLP & parsing	**spaCy + PyMuPDF**	Industrial strength text processing.
Frontend	**Next.js + Tailwind**	React server components for fast UI.
Async Tasks	**Celery + Redis**	Background processing for scraping.
Deployment	**Docker Compose**	Containerization for easy reproducibility.

---

7. Security & Compliance

- **Data Minimization:** No scraping of login-walled platforms (LinkedIn) to avoid bans. Only public aggregation.
- **Encryption:** Resumes and personal data encrypted at rest (AES-256).

- **Local-First AI:** By using local LLMs (via Ollama), user career data does not leave the infrastructure, ensuring privacy.

---

8. Implementation Roadmap (Hackathon Strategy)

1. **Phase 1: Foundation (Hours 0-12)**
    - Set up FastAPI + ChromaDB.
    - Implement Resume Parsing & Profile Generation.
2. **Phase 2: Intelligence (Hours 12-24)**
    - Implement Job Scraper & Matching Logic.
    - Build the Roadmap Generator.
3. **Phase 3: The "Agent" Loop (Hours 24-36)**
    - Connect the Feedback Loop (Rejection $\rightarrow$ New Plan).
    - Build the Frontend Dashboard.
4. **Phase 4: Polish (Hours 36-48)**
    - Resume Tailoring output.
    - Final testing and "Why it wins" demo script.
