🕵️ People DeepSearch AI – Work Trial Spec (with basic frontend)

🎯 Objective

Build an MVP backend system for **People DeepSearch AI** — a background check and identity discovery tool that uncovers as much information as possible about a person based on minimal input (name, email, phone, or alias).

You may choose to include a **very basic frontend form** (freestyled by AI) to demonstrate how the backend can be used interactively. This is not a UI/UX challenge — a simple React/Vue/HTML form is fine.

A frontend form is not essential. If you can demonstrate the backend via APIs, its fine too.

This trial will evaluate:

- 🔍 API / scraping decisions
- 🧠 Use of AI agents or LLMs to build the most detailed possible profile for a person
- 📐 Backend architecture and system design
- 🖼️ Your front-end will **NOT** be evaluated

🕑 Timeline

- **Duration**: ~1 week (full-time)
- **Deliverables**: GitHub repo with code, demo video, sample profiles, and README

✅ What You'll Build

1. A functional backend system that:

- Accepts one or more of the following identifiers:
    - Full Name
    - Email
    - Phone Number
    - Username / Alias
- Optionally:
    - Country or city
    - **User may provide Free-text context that is LLM friendly** like:

> "Searching for info on a guy called John - he used to work in tech sales, maybe lives in Oceania. Think hes from New Zealand"

(If stuck and deciding between multiple identities, you may ask the user questions for additional data to narrow down the target identity to do a deep search on. You are not guaranteed to receive that data since the user might not know and should ideally be asking + interpreting in natural language. You can optionally ask the user to select between multiple identities before beginning the deepsearch on a specific identity)

- Aggregates and outputs structured person data from various sources (its up to you which sources you use):
    - People-data APIs
    - Scraped sources
    - Inferred insights via LLMs
    - Agentic AI for web scraping

2. A lightweight frontend (optional stack):

- Input fields for:
    - Name, Email, Phone, Username (any subset)
    - Free-text context (textarea)
- Submit button to trigger search
- Display final JSON result in readable format (table, preformatted block, or simple cards)
- Styling optional, focus is on visibility & demo, not design polish

---

🧠 Functional Requirements

🔍 Input Handling

- Normalize phone/email formats
- Allow multi-field input (e.g. name + location + text prompt)
- User context (textarea) is interpreted by the backend via an LLM prompt or embedding

🧠 LLM Agent Usage

- LLM agent should:
    - Interpret vague/natural language context provided by the user
    - Disambiguate multiple matching identities (if stuck and deciding between multiple identities, you may ask the user questions to narrow down the target identity to do a deep search on)

🌐 Data Collection

Use any combination of:

People APIs:

- Pipl, FullContact, PeopleDataLabs, Spokeo, Clearbit, Numverify, Whitepages

Web scraping:

- Social networks (LinkedIn, Twitter/X, GitHub, Instagram)
- Reverse lookup platforms
- Public government databases or directories

LLMs:

- To reason, fill in gaps, or normalize inconsistent data across sources
- Optional: Embedding + similarity search to match profiles from vague input

🤖 LLM Best Practices (for Agentic AI Backend Path)

If you implement the backend using an **agentic LLM approach**, follow best practices for modern agentic AI (rough example, you'll need to look into open source modern agentic AIs for best practices) :

1. **Extract to JSON:** First, use the LLM to parse all natural-language or mixed inputs into a strict JSON schema (`name`, `email`, `phone`, `alias`, `location`, etc.), with validation & retries on schema failure.
2. **Planning Phase:** Have the LLM generate a step-by-step plan (which tools to call, in what order, with budget/time constraints).
3. **Tool Execution:** Execute tools strictly according to the plan, logging inputs, outputs, errors, and confidence per step.
4. **Judge Pass:** At the end, run the LLM again as a validator to check schema compliance, resolve conflicts, assign confidence scores, and verify provenance of all fields before returning the final profile.

🔧 System Design (Include in README)

Include a diagram and notes explaining:

- Input → Processing → Output flow
- Orchestration of search agents (scrapers, APIs, LLMs)
- Task management (queues, retries, timeouts)
- Bot evasion strategy (proxy pools, headless browsers)
- Rate limiting and cost control (if paid APIs used)

💻 Example Tech Stack

Layer	Tool Options
Backend	Python (FastAPI), Node.js (Express)
Frontend (basic)	React / Vue / plain HTML + JS
Scraping	Playwright / Puppeteer / Requests
LLM API	OpenAI / Claude / Local (Ollama)
Storage (optional)	MongoDB / Redis / JSON in-memory
Task queue (opt)	Celery / BullMQ
Proxy/Bot bypass	BrightData, ScraperAPI, rotating pool

📽️ Demo Expectations

- Deploy locally or on localhost
- Accept inputs through the form
- Show sample output in a clear format (cards / JSON / tables)
- Demo flow: input → process → profile display

Submit a screen-recorded walkthrough (2–5 min) of:

1. Entering sample input
2. Submitting the query
3. Showing results on the frontend
4. Explaining logs or backend flow briefly
