AI Research Agent — Applicant Problem Statement

Overview

You will build an AI Research Agent that accepts a topic from a user, runs a small research workflow, and returns a structured result. The system should simulate how AI Agents work in production: taking input, deciding and planning steps, executing actions, persisting outputs, and showing explainable traces.

Detailed Agent Workflow

When a user submits a research topic:

1. Step 1 — Input Parsing: Validate the input and store the request in the DB.
2. Step 2 — Data Gathering (External API): Fetch relevant articles from a public API (Wikipedia, NewsAPI, HackerNews API, or any free/open API).
3. Step 3 — Processing: Extract top 5 articles, summarize each, and extract top keywords.
4. Step 4 — Result Persistence: Save the processed results and logs in DB.
5. Step 5 — Return to Frontend: Return topic, logs of each step, and final structured results.

Frontend Requirements

- Built with React/Next.js in TypeScript.
- Input form to submit a research topic.
- Task list to see all previous research requests.
- Task detail view showing logs and structured results.

Backend Requirements

- Implemented in FastAPI (Python) or Node.js with TypeScript.
- Endpoints:
  - POST /research → submit new topic, trigger workflow.
  - GET /research → list all topics.
  - GET /research/{id} → get logs and results.
- Must use a background job system:
  - FastAPI: Celery, RQ, or async tasks.
  - Node.js: Bull.js or worker queues.
- Use PostgreSQL or SQLite for persistence.
- Store topic, workflow logs, and structured results.

Deployment Requirements

- Backend and database must run in Docker.
- Frontend should be deployed on Vercel (or any cloud).
- Backend should be deployed on Render, Railway, Fly.io, AWS, GCP, or Azure.
- Provide public URLs and deployment instructions in the README.

Deliverables

1. Public GitHub repository with:
   - /backend (FastAPI or Node.js/TS)
   - /frontend (Next.js/TS)
   - docker-compose.yml for local run
   - README with setup + deployment instructions
   - .env.example
2. Publicly deployed app (frontend + backend).
3. Documentation including architecture diagram, workflow explanation, and notes on trade-offs.

Evaluation Rubric

- Backend (40%): API design, async tasks, DB schema, logs, background execution.
- Frontend (20%): Clear, working UI.
- Deployment (20%): Cloud deploy, Docker reproducibility.
- Code Quality (10%): Clean, modular, tested, TS usage.
- Documentation (10%): README, architecture, clarity.
- Bonus: Real LLM API integration, per-user history, observability, CI/CD pipeline.

Time Expectation

The challenge is designed to be solved within ~3 days (~15–20 hours).

Email me your deliverables at - kaifmohd5000@gmail.com