Take-Home Assignment

Multi-Agent Task Orchestration System

Stack	React / Next.js · Python

CONFIDENTIAL — For candidate use only. Do not redistribute.

OVERVIEW

You will design and partially implement a Multi-Agent Task Orchestration System — a lightweight platform where multiple AI agents collaborate to complete a complex task (such as researching a topic, writing a report, and reviewing it for quality).

> ✦ **What we're really evaluating**
>
> This is NOT about finishing everything. We care about your thought process: how you break down ambiguity, make architectural trade-offs, communicate decisions, and write clean, intentional code. A well-reasoned partial solution beats a rushed complete one.

THE SCENARIO

Imagine a user submits a request like: "Research the pros and cons of microservices vs. monoliths and produce a summary report." Your system should orchestrate multiple agents to handle this:

- **Planner Agent —** Breaks the user's request into discrete sub-tasks and determines execution order.
- **Researcher Agent —** Gathers information for each sub-task (simulated or stubbed is fine).
- **Writer Agent —** Synthesizes research into a draft report.
- **Reviewer Agent —** Evaluates the draft, provides feedback, and can send it back for revision.

You do not need to integrate a real LLM. Agents can return hardcoded or templated responses. The focus is on the orchestration layer and the user-facing experience.

REQUIREMENTS

Backend (Python)

Build a small API service that models and orchestrates the agent pipeline.

- **Agent abstraction:** Define a base Agent class or protocol. Each agent should have a consistent interface (e.g., receive input, return output + status).
- **Orchestrator:** A coordinator that manages agent execution order, passes outputs between agents, and handles the review feedback loop.
- **State model:** Track the overall task status (e.g., planning → researching → writing → reviewing → done) so the frontend can display progress.

- **API endpoints:** At minimum: POST to submit a new task, GET to check task status and results. Bonus: a WebSocket or SSE endpoint for real-time progress updates.

Technology: Use any Python framework you're comfortable with (FastAPI, Flask, Django, etc.). Keep dependencies minimal.

Frontend (React / Next.js)

Build a simple UI that lets a user interact with the orchestration system.

- **Task submission:** A form where the user types their request and submits it.
- **Progress visualization:** Show which agent is currently active, what the pipeline looks like, and ideally update in real time.
- **Results display:** Render the final output (the "report") and show the chain of agent actions that led to it.
- **Reviewer feedback:** If the reviewer agent sends the draft back for revision, reflect this in the UI (e.g., show the feedback, then show the revised version).

Technology: React with Next.js. Use any styling approach you prefer (Tailwind, CSS Modules, etc.).

WHAT TO SUBMIT

1. **Working code** in a Git repository (GitHub, GitLab, or a zip). Include a README with setup instructions.
2. **Design document** (markdown or PDF, 1–2 pages) covering:
   - Your architectural decisions and why you made them.
   - Trade-offs you considered (e.g., polling vs. WebSockets, sync vs. async).
   - What you would do differently or add with more time.
   - Any assumptions you made.
3. **Brief walkthrough** (optional but encouraged): a 3–5 minute Loom or screen recording walking through your solution.

TIME GUIDANCE

We expect you to spend approximately 2–4 hours on this. Here's how we suggest you allocate your time:

Phase	Suggested Time	Focus
Planning & Design	**30–45 min**	Read the brief, sketch architecture, identify trade-offs

Backend	**45–60 min**	Agent abstractions, orchestrator logic, API endpoints
Frontend	**45–60 min**	Task submission, progress UI, results display
Design Doc + Polish	**20–30 min**	Write-up, README, final clean-up

> ⏱ **It’s okay to not finish**
>
> If you run out of time, stop coding and write about what you would have done next in your design document. Thoughtful incomplete work tells us more than rushed complete work.

EVALUATION CRITERIA

Here’s exactly what our reviewers will be looking for, in order of importance:

Criterion	Weight	What we look for
System Design Thinking	**30%**	How you decompose the problem, define boundaries between agents, and reason about data flow and failure modes.
Code Quality	**25%**	Clean abstractions, readable code, sensible naming, separation of concerns. We’d rather see 200 clean lines than 800 messy ones.
Communication	**20%**	Quality of your design doc and commit messages. Can you articulate trade-offs clearly?
Frontend Craft	**15%**	Intuitive UX for the progress/status flow. Doesn’t need to be beautiful — needs to be clear and functional.
Bonus Points	**10%**	Error handling, testing, real-time updates, creative touches, thoughtful README.

STRETCH GOALS (OPTIONAL)

Only attempt these if you’ve completed the core requirements and have time remaining. They’re not expected.

- **Retry / error handling:** What happens if an agent “fails”? Show how the orchestrator recovers.
- **Parallel agents:** Can the Researcher run multiple sub-tasks concurrently? How does that change your orchestrator design?
- **Agent configuration:** Let the user customize the pipeline (e.g., skip the review step, add a “Fact Checker” agent).
- **Persistent state:** Store task history so users can revisit past results.
- **Testing:** Unit tests for the orchestrator logic and/or component tests for the frontend.

GROUND RULES

- You may use any open-source libraries, documentation, or AI coding assistants. If you do, be prepared to explain every line of your code in a follow-up interview.
- Do not share this assignment with anyone or post it publicly.
- If anything is unclear, make a reasonable assumption and document it. We will not answer clarifying questions — ambiguity handling is part of the evaluation.
- Submit within the agreed timeline. Late submissions without prior notice will not be reviewed.

Good luck — we're excited to see how you think!