FREYA

Take-Home: “Agent Console”

Hurrah! Here’s your chance to show off just how good of a fit you are solving real world problems that we wrestle with day-to-day here at Freya. Completing this exercise means that…

Goal

Build an e2e **production-grade web interface** and a sample **livekit agent** where a user can:

1. **Create & manage prompts** for an AI agent/assistant (use in memory store),
2. **Interact** with the agent through **voice** and **chat** in real-time,
3. **Inspect logs/metrics** about recent sessions and message latency

The solution must be **portable via Docker**, with a **TypeScript/Next.js** web-app and a **Livekit** agent.

Requirements

Functional

- **Auth (lightweight)**: Email-less session (magic “dev” login) or simple token gate is fine.
- **Prompt Library (lightweight in-memory)**: CRUD for prompts (title, body, tags). *Bonus:* Version each save.
- **Live Session**:
  - Start a session with a selected prompt.
  - **Send messages** to the agent and **stream** assistant tokens back in real-time.
  - **Send audio input** to the agent and **stream** assistant voice back in real-time.
  - Show **message timestamps**, token rate (tokens/sec), and running latency.
- **History**: Persist sessions & messages; list last 10 sessions; open a past session read-only.

- **Observability**: A small **Metrics** view:
  - Avg first-token latency
  - Avg tokens/sec
  - Error rate (last 24h)
- **Resilience**: When the connection drops, auto-retry and show a non-blocking toast.

Non-Functional

- **Production-minded**: reasonable error boundaries, logging, rate-limits, input validation, and retries.
- **Security hygiene**: don't leak envs to client; basic CORS; avoid obvious injection vectors.
- **Tests**: a few focused unit tests (backend) + component/interaction test(s) (frontend).

Tech Constraints (use these)

- **Full Stack**: **Next.js** for the application e2e
- **LLM stub**: Provide an **agent** (livekit) that streams tokens
- **Frontend**: **Next.js (latest)** + **TypeScript** for data fetching.
  - **WebRTC & LiveKit Room.io** client for streaming.
  - State kept minimal; server cache via TanStack Query.
- **UI**: your choice; keep it clean. A split-pane "Console" is ideal: left (history/prompts), center (chat), right (metrics/logs).
- **Docker**: one command should bring up the full stack and agent (docker compose up).
- **DevOps**: provide a minimal **compose** file, **.env.example**, and health checks,

---

Frontend Features (Next.js + TS)

- **Routes**
  - /login (lightweight auth)
  - /console (main):

        - **Left**: Prompt Library (search, tag filter) + Recent Sessions
        - **Center**: Live Chat (compose, send, stream tokens in the UI; show latency & token/s)
        - **Right**: Metrics widget + last 20 log lines
- **UX**:
    - Command-K (or /) to quick-switch prompts.
    - Auto-scroll on new tokens; pause on hover.
    - Toasts for reconnect/retry & errors.
- **Testing**:
    - One React Testing Library spec (send message, assert tokens render).

Evaluation Rubric (what we score)

1. **Correctness & UX** (35%)
    - Streaming works; sessions persist; prompts CRUD; reconnects behave.
    - UI clarity, error handling, keyboard affordances.
2. **Code Quality** (25%)
    - Types, readability, boundaries, small modules, test quality.
3. **Operational Readiness** (20%)
    - Docker, health checks, env separation, logs/metrics, simple rate-limit.
4. **Security & Hygiene** (10%)
    - Input validation, secrets handling, CORS, basic auth gate.
5. **Stretch / Depth** (10%)
    - LiveKit/WebRTC PTT mode, TTS/ASR path, tracing (OpenTelemetry), or a small admin panel.

Bonus (optional, choose any)

- **Help Center (Qdrant RAG)**: Hook the livekit agent with access to artificial help-center Q&A style collection and perform recall
- **OpenTelemetry** traces for “message roundtrip” TTS and other related component metrics
- **Nginx** reverse proxy and **MongoDB** to setup prompt library

---

Submission

Repo structure (example):

/app (Next.js, TS)
/agent (LiveKit Agent + WS)

(docker-compose.yml, Dockerfile, migrations)
.env.example
README.md

README should specify:

- Setup (local & docker)
- Design notes (why these choices)
- Tradeoffs & what you’d do next for production
- API overview
- **Tests**:
  - Backend: 3–5 unit tests (streaming generator, token rate calc, error pathway).
  - Frontend: at least 1 component/integration test.

Please send us a link to a publicly visible repository or a .zip file containing the full source code to your work.