VIBE Coding Exercise

VIBE Coding Exercise

Build a small **agentic web app** in a **2–3 hour timebox**. This is a *VIBE-Code-style* exercise: we expect you to lean on **natural-language tooling**, **agentic coding workflows**, and pragmatic engineering judgment to deliver a working thin-slice quickly.

Your submission should demonstrate:

- You can build and connect **client + server** web code.
- You can design an **agent** that plans and takes actions (not just chat).
- You can incorporate **knowledge** in a visible, useful way.
- You can prove you successfully executed **real web interactions**.

Goal

Build a small web app where an **agent** accepts a user goal in natural language, produces a lightweight plan, and completes the task by taking actions on both:

- **Server-side** (tools/workflows you implement).
- **Client-side** (actions performed in the browser UI, beyond rendering text).

Choose any domain. Keep it small but end-to-end.

Requirements (intentionally broad):

1) **Agent behaviour**
   - Accepts a freeform user goal.
   - Produces a lightweight plan and executes steps.
   - Displays a clear **activity log/trace** (what it did, what it tried, what succeeded/failed).

2) **Server-side actions**

Implement **at least 3** server-side tools/actions the agent can call. These should do real work (not only mocked text). Examples include fetching data, transforming data, creating/updating records, running a workflow, calling an external API, etc.

3) Client-side actions

Implement **at least 2** client-side actions the agent can trigger inside the browser. These should be observable UI behaviours (beyond printing text), such as updating UI state, filling a form, selecting filters, generating/downloading an artifact, manipulating a table/canvas, etc.

4) Knowledge

Add a simple knowledge mechanism the agent can use (e.g., stored notes/snippets, session history, retrieval, tagged items). The UI should make it obvious when and how knowledge influenced the result.

5) Front-end quality

Provide a clean UI that shows:

- Conversation
- Agent plan + trace/activity log.
- Results/artifacts (saved items, outputs, etc.).

6) Control & safety

Provide basic user control:

- At least one action requires **user approval** (or a "dry run" mode).
- No silent destructive actions.

Required: Proof of successful web coding

Include evidence that you implemented and validated real web behavior.
Your submission must include a visible **"web proof"** moment demonstrating:

- Client → server requests occurred (e.g., DevTools Network).
- A server tool/action executed.

- The UI changed as a result.

This can be a short segment in the demo video and/or a short **README** section.

Deliverables:

1. **Link** to a deployed app **or** a repo link with one-command run instructions.
2. **Short video demo (2–4 minutes)** showing:
   - Goal → plan → server actions → client actions → result.
   - Knowledge being used.
   - Activity log/trace.
   - The **web proof** moment (Network/Console/trace).
3. **README (brief)** covering:
   - What you built and why.
   - Server tools/actions + client actions.
   - Where knowledge lives and how it's used.
   - How to run.
   - Tradeoffs + next steps.

Evaluation:

We'll evaluate:

- End-to-end functionality within the timebox.
- Clear client/server split; actions are real.
- Agent trace is understandable.
- Knowledge influences behavior.
- UI clarity and product sense.
- Code organization and basic robustness.