Lane – Full-Stack Developer Intern (MERN) Assignment

Context

Lane is an outcome-focused product discovery and planning platform. For this exercise, build a small, production-like **one-page web app** that lets users submit and explore product feedback with a **clear, modern UI**. You can take visual and interaction cues from the Lane website.

Important

- **Do not use AI tools** (ChatGPT/Copilot/CodeWhisperer/etc.) to write code, copy UI, or draft the README. We want to assess your **raw knowledge and decision-making**.
- Be honest about what you built and why. Partial solutions are fine if documented clearly.

What to build (One-page app)

A single-page React app (no multi-page routing) with the following sections/components on one screen:

1. **Header**
    - App name and a short tagline.
    - Optional: a compact "Add Feedback" button that opens a modal/drawer.
2. **Add Feedback**
    - Form fields: **Title (required)**, **Description (required)**, **Category (required: Bug / Feature / Improvement)**.
    - Submit creates a record via your API.
    - Show clear validation, loading state, and success/error feedback (e.g., toast).
3. **Feedback Explorer**
    - **List** of feedback items showing Title, Category, Created date, and a short Description preview.
    - **Sorting**: Newest first / Oldest first.
    - **Grouping**: Group by Category (three groups) and allow collapsing/expanding groups.
    - **Search or Filter**: Simple search by title or filter by category.

    - **Empty states**: Design an empty state for “no data” and “no results”.

4. **Nice to have (optional)**
    - Upvote/like count per item.
    - Simple JWT auth to restrict submission to logged-in users.

Tech requirements

- **Frontend**: React (TypeScript preferred), state via local state or a small store (Context/ Redux/Zustand). Styling with Tailwind or clean CSS.
- **Backend**: Node.js + Express with MongoDB (Mongoose).
- **API**:
    - `POST /feedback` – create feedback (server-side validation)
    - `GET /feedback` – list feedback with optional query params (`sort`, `category`, `q`)
- **Data model**: Feedback { id, title, description, category, createdAt, [votes] } and a separate Category collection or enum.

UI expectations (must-have)

- Clean, consistent typography and spacing.
- Clear visual hierarchy (titles, subtitles, cards, dividers).
- Responsive layout (desktop first; tablet/mobile acceptable).
- Thoughtful interaction design: disabled states, focus states, error messages, loading indicators, success toasts.
- Accessible basics: semantic HTML, labels for inputs, keyboard reachable controls.

Deployment

- On Vercel/Netlify or of your choice
- Share **live URLs**. Deployed app must talk to the deployed API.

Documentation (README.md)

Keep it short, specific, and professional:

1. **Overview**: What you built and why; brief architecture.
2. **Tech choices**: Frameworks/libraries and the reason for choosing them.
3. **How to run**: Local setup (env vars), seed data if any.
4. **API**: Endpoints, sample requests/responses.
5. **What's missing / trade-offs** and how you would improve it next.
6. **No-AI statement**: Confirm you did not use AI to write code/docs.

What we will evaluate

- **UI quality (30%)**: Visual clarity, responsiveness, states, and usability.
- **Code quality (25%)**: Structure, readability, separation of concerns, typings (if TS), linting.
- **Backend & API (20%)**: Schema design, validation, errors, predictable responses.
- **Sorting/grouping/search (15%)**: Correctness and performance at small scale.
- **Deployment & README (10%)**: Works end-to-end; clear instructions.

Submission

- GitHub repo(s) (frontend and backend; mono-repo also fine).
- Live links for frontend and backend.
- Include test notes (e.g., any seeded data) in the README.

Tips

- Keep scope tight; polish what you include.
- Start with schema + API, then UI states, then sorting/grouping.
- Prefer simple, reliable patterns over complex abstractions.