Full Stack Developer Assignment:

Build a Mini Task Tracker (Next.js + Node.js)

Purpose

We want to see how you approach a small, end-to-end feature in a modern full-stack setup. Please implement a minimal Task Tracker that demonstrates practical judgment, clean structure, and attention to UX.

Timebox

- Total time: **45 minutes** (we'll reserve ~5–10 minutes afterward for a quick walkthrough).

Scope of Work

Build a small web app using **Next.js** for the front end and a **Node.js** API (Next.js API routes are fine) for the backend. No database setup is required; use simple in-memory storage for the session.

What to Build

Core User Journey

1. As a user, we can **create** a task by entering a title (required) and an optional description.
2. We can **view** a list of tasks.
3. We can **filter** tasks by status (All, Active, Completed) and **search** by text across title/description.
4. We can **mark tasks as done/undone**.
5. We can **delete** a task.
6. If something goes wrong (e.g., invalid input), we see a clear, actionable error message.

Functional Requirements

- **Task fields**: id, title, description (optional), done (boolean), created date/time.
- **Validation**: title is required and reasonably short; show a friendly error message on invalid input.
- **Filtering**: All / Active / Completed views.
- **Search**: case-insensitive text search across title and description.
- **State preservation**: filters and search state should persist on page refresh (e.g., via URL query parameters or an equivalent approach).
- **Feedback**: show visible confirmation of actions (e.g., task added, toggled, deleted) and error states.

API Expectations (behavioral, not code)

- Provide a small, clearly organized API for listing, creating, updating (e.g., toggle done or edit fields), and deleting tasks.
- Use appropriate HTTP semantics and status codes (success vs. validation error vs. not found).
- Keep data in memory for the session; no database is required.

Front-End Expectations

- One main page that includes:
    - A **create task** form (title required, description optional).
    - **Filters and search** controls that drive the list view.
    - A **task list** showing: title, optional description, created date, status (done/undone), and a way to delete.
- UX should be straightforward and responsive enough for typical desktop usage.
- Error messages should be clear and displayed near the relevant UI.

Non-Functional Expectations

- **Structure & clarity**: sensible folders/files and separation of concerns.
- **Readability**: clear naming, comments only where truly helpful.
- **Accessibility basics**: form labels, focus states, and keyboard usability for core actions.
- **Resilience**: graceful handling of empty states, loading states, and error states.
- **Performance**: avoid unnecessary re-renders or blocking operations for the size of this app.

Stretch (only if time remains)

- User-friendly client-side validation that mirrors server rules.
- Optimistic updates for quick toggles and creates.
- Persist the in-memory data to a simple local file for the duration of the run.
- Basic automated test of one API behavior.

What to Submit

- A link to the code repository (or a zipped project).
- A short **README** that explains:
    - How to run the app locally.
    - Any trade-offs or shortcuts taken due to the time limit.
    - Anything you would improve next and why.

How We Will Evaluate

- **Correctness (Does it work?)**
  Core flows—create, list, filter, search, toggle, delete—function as described with appropriate errors and statuses.

- **Code Quality (Is it maintainable?)**
  File structure, clarity, and sensible boundaries between UI and API.

- **UX & Accessibility (Is it usable?)**
  Clean, predictable interactions; basic a11y covered.

- **Validation & Errors (Is it safe to use?)**
  Server-side checks with clear user feedback.

- **Polish (Did you make smart choices?)**
  Small touches that improve the experience, within the timebox.