The Trial Issue Log Project:

Goal

Build a system to track issues discovered during a clinical trial site visit. The app should let a user:

- Create/edit/resolve issues
- See a list with basic filters
- View a simple dashboard (counts by status & severity)

The system should come with a clear Readme file to explain how can I run it on my local machine. The best-case scenario (and please leave it to the end), is a system that has a repository that I can pull from and a working version on AWS that I can try from my browser.

Please use your time wisely, it is not meant for you to dedicate more than 3-5 hours on the system (not including learning time if needed) - whatever you manage to do, please send it over, I will review it. After that we will be in touch and you can explain what you managed and didn't manage to do in the time you had.

Keep it simple, but working end-to-end at each point: database + backend API + frontend UI, deployed on AWS free tier is a +.

Tech Constraints (must)

- **Language/Runtime:** Node.js (LTS)
- **Framework:** Any Node framework that works for you. Please explain your choice.
- **Database:** Whatever you see fit - please explain what led you to the decision (for the home assignment, and if we would take this to production and grow the product)
- **Frontend:** Your choice - please explain what led you to the decision (for the home assignment, and if we would take this to production and grow the product)
- **Deploy:** AWS free tier

Core Features (what to build)

1. **CRUD for Issues**
   - Fields:
     - `id` (auto)
     - `title` (string, required)
     - `description` (text, required)
     - `site` (string, e.g., “Site-101”)
     - `severity` (enum: `minor | major | critical`)
     - `status` (enum: `open | in_progress | resolved`)
     - `createdAt` (datetime)
     - `updatedAt` (datetime)
   - Create, read (list + detail), update, delete.
2. **List & Filters**
   - Table/list of issues with:
     - Text search on `title`
     - Filter by `status` and `severity`
     - Sort by `createdAt` (desc)
3. **Dashboard**
   - Simple counts:
     - By `status` (open/in_progress/resolved)
     - By `severity` (minor/major/critical)

4. **Import Data**
    - Provide an **"Import CSV"** button or a **one-time script/endpoint** to ingest a provided CSV (see sample below).
    - On success, show issues in the list.
5. **Basic UX**
    - Clean page with:
        - Create Issue form
        - Issues table with filters
        - Dashboard summary chips/cards

API - As you see fit.

Use JSON. Validate inputs; return 400s on bad data.

DB Schema - as you see fit based on DB selection

Please explain your choice

Sample CSV (should be provided in the repo as `issues.csv`)

```
title,description,site,severity,status,createdAt
Missing consent form,Consent form not in file for patient
003,Site-101,major,open,2025-05-01T09:00:00Z
Late visit,Visit week 4 occurred on week
6,Site-202,minor,in_progress,2025-05-03T12:30:00Z
Drug temp excursion,IP stored above max temp for 6
hours,Site-101,critical,open,2025-05-10T08:15:00Z
```

```
Unblinded email,Coordinator emailed treatment arm to
CRA,Site-303,major,resolved,2025-05-14T16:00:00Z
```

Frontend (suggested)

- Issues page:
    - Top area: **Create Issue** form (title, description, site, severity)
    - Filters: search box, dropdowns for status & severity
    - Table of issues: title, site, severity, status, createdAt; actions: **Edit**, **Resolve** (sets status), **Delete**
    - **Upload batch (from csv file)**
- Dashboard page: two rows of chips graphs:
    - `Open: X · In Progress: Y · Resolved: Z`
    - `Minor: A · Major: B · Critical: C`
- Navigation?
- Other pages - as you see fit

Deployment & Architecture:

You should be able to deploy your app to AWS using the free tier. Please explain (and, if possible, demonstrate) how you would push your app to the cloud. Describe the process and why you believe this is the best approach, as well as the best practices you follow.

Deliverables

- **GitHub repo** with:
    - `README.md` (setup, run, deploy, endpoint examples, trade-offs)
    - `server/` (or root) Node app code
    - `frontend/` (if applicable) or templates

    - `db/` (Schema/Tables/Description)
    - `issues.csv`
- **Working AWS URL,** we can open - if you managed to push it to AWS
- Input validation + minimal error handling

What we'll evaluate

1. **Architecture and the ability to explain what you have**
2. **Code quality & structure** (readability, modularity, naming)
3. **Simplicity under a time box** (good choices, not over-engineering)
4. **Working demo** (end-to-end correctness)
5. **API & data modeling** (clean schema, validations)
6. **Deployment hygiene**

Nice-to-Have (only if time remains)

- Inline "Resolve" button that flips `status` → `resolved`
- Pagination (server-side or client-side)
- Minimal tests (1–2 unit tests on validation or a route)
- Lightweight auth stub (single hardcoded user)