Opmaint — Web Development Intern Assignment

Build a Permit to Work module for a CMMS.

Time expected: 12–16 hours across the week. This is a build assignment, not a quiz.

Deadline: 7 days from the day you receive this.

Submit three things:

1.A GitHub repo - public, or private with `tanzeel@opmaint.com` invited. Commit as you go; we look at commit history.

2. A live deployed link - Vercel, Netlify, Render, Railway, Fly, anything free. If it isn't running on the internet, it doesn't count.

3. A Loom video, 6–8 minutes - you demoing the module and walking through one part of the code you're proud of and one part you'd rewrite. Camera on.

Email to tanzeel@opmaint.com with subject `Web Dev Intern Assignment — <Your Name>`.

Before you write any code

You found this listing through Cloutgency, so you may not have heard of us.

Go through opmaint.com properly first — the whole site, not just the homepage. Product pages, industry pages, pricing, blog. Spend 30 minutes there. Understand what a CMMS is, who uses it, and what kind of software you'd be building here.

You are building a module that would slot into a product like ours. Submissions that clearly didn't look at what a CMMS is end up building a generic to-do app with the word "permit" on it. We see it every time.

What a Permit to Work actually is

A Permit to Work (PTW) is a paper form, in most Indian plants still literally paper, that authorizes a dangerous job before anyone is allowed to start it.

Say a contractor needs to weld a bracket onto a pipe rack. Welding means open flame. Open flame near a plant that may have flammable vapour, oil residue, or a gas line kills people. So before he lifts the torch:

- Someone requests permission, describing the exact job, exact location, exact time window.
- A safety officer inspects the site and lists the precautions: isolate this line, test the atmosphere, keep a fire extinguisher and a fire watch present, remove combustible material within 10 metres.
- The relevant approvers sign off — typically the area/production owner who owns that equipment, and the safety officer.
- Only then does the work start, and only inside the approved time window.
- When the job is done the area is inspected and the permit is closed and handed back.

Permits expire. A permit issued for one shift is not valid the next shift. If conditions change — a gas alarm, a shift handover, an emergency — the permit is suspended immediately and work stops.

This is the module you're building. It is one of the highest-stakes screens in any CMMS, because when it's wrong, the failure mode is a person getting hurt and a plant having no record of who authorized what.

Go read two or three real PTW forms before you start. Search for "hot work permit format PDF" or look at how MaintainX, SafetyCulture, or Limble present permits. Twenty minutes of this will make your build ten times better.

Product Specification

Build the following. Where the spec is silent, make a decision and note it in your README.

1. Permit types

Support these four, at minimum:

| Type | Distinct fields it needs |

|---|---|

| **Hot Work** | Type of hot work (welding / grinding / cutting / soldering), fire watch assigned, fire extinguisher type present, combustibles cleared radius, gas test readings (LEL %, O_2 %) with test time |

| **Confined Space Entry** | Space ID, entry point, atmospheric test (O_2 %, LEL %, H_2S ppm, CO ppm) with test time, standby attendant name, rescue plan, ventilation method, entry/exit log |

| **Working at Height** | Height in metres, access method (scaffold / ladder / MEWP / rope), fall arrest equipment, anchor point checked, barricading below |

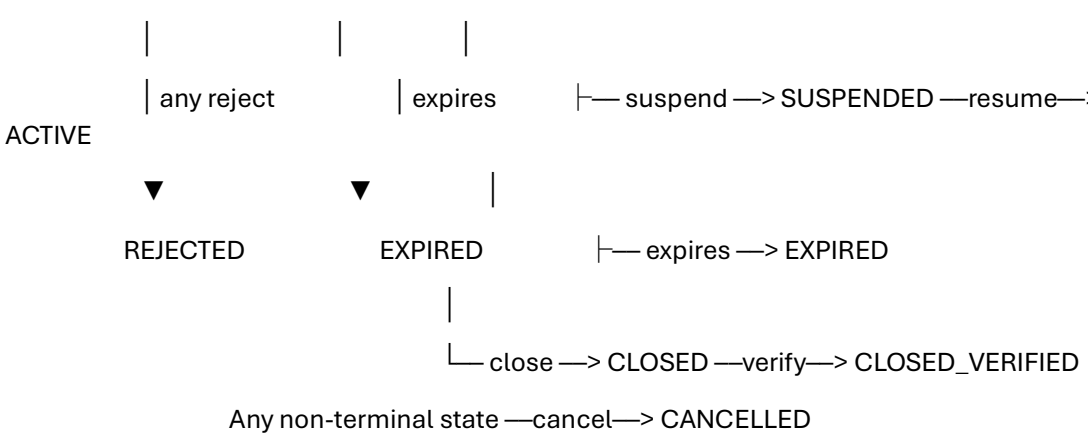
| **Electrical / Isolation (LOTO)** | Equipment tag, voltage level, isolation points list, lock numbers, tag numbers, earthing applied, tested dead by whom |

All four share a common core: requester, contractor/team performing the work, work description, exact location (plant → area → equipment), planned start and end datetime, hazards identified, PPE required, precautions checklist, approvals, status.

Design this properly. A junior developer builds four separate forms with copy-pasted code. Show us you can model a shared permit entity with type-specific fields, so a fifth permit type — say Excavation — can be added without rewriting anything. This is the single biggest thing we're evaluating.

2. Permit lifecycle

Implement this state machine and enforce it in the backend, not just the UI:



Rules that must hold:

- A permit cannot go ACTIVE unless **every** required approver has approved.

- A permit cannot go ACTIVE before its planned start time.

- A permit auto-expires when its validity window passes. An expired permit can never be reactivated — a new permit must be raised. (Extension is a separate, explicitly requested action; see below.)

- Work cannot be logged against a permit that isn't ACTIVE.

- Any illegal transition must be rejected server-side with a clear error, even if someone hits the API directly. **We will try.**

3. Roles and permissions

Four roles. Enforce server-side.

- **Requester** (technician / contractor supervisor) — creates and submits permits, closes their own permits, cannot approve anything.

- **Area Owner** — approves permits for equipment in their area only. Cannot approve for other areas.

- **Safety Officer** — approves any permit, can suspend any ACTIVE permit instantly, performs closure verification.

- **Admin** — full access, manages users, areas, equipment.

Key rule: **a person can never approve their own permit**, even if their role would otherwise allow it. Make sure this is tested.

Auth can be simple — email + password with sessions or JWT. Don't burn hours on OAuth.

4. Required screens

1. **Permit list / dashboard** — filterable by status, type, area, date range, and "my approvals pending". Must make the important thing obvious at a glance: what's active right now, and what's expiring in the next 2 hours.

2. **Create permit** — multi-step form that adapts to the permit type selected. Draft must be saveable.

3. **Permit detail** — the full permit, its approval trail, its audit log, and the actions available to the current user given their role and the permit's state. A user should never see a button they aren't allowed to press.

4. **Approval view** — an approver can see what they need to check, approve with a comment, or reject with a mandatory reason.

5. **Closure** — requester marks work complete with completion notes; safety officer verifies the area is clean and closes it out.

5. Audit trail

Every state change, every approval, every rejection, every suspension, every field edit after submission gets an immutable log entry: who, what, when, from-value, to-value, and any comment.

This is non-negotiable — in the real world this log is what gets produced during a safety audit or an incident investigation. It must be visible on the permit detail screen as a readable timeline, not a JSON dump.

6. The bits that separate good from average

Do these if you have time. In order of how much they'll impress us:

- **Expiry handling that actually works.** Permits expiring on a timer, a visible countdown on active permits, an "expiring soon" state. Think about how expiry works when nobody has the browser open.

- **Extension request flow.** A requester asks for +N hours before expiry; it needs re-approval by the safety officer; extensions are capped and logged.

- **Conflict detection.** Warn when a new hot work permit overlaps in time and location with an existing confined space permit. This is a real cause of accidents and almost nobody's software does it.

- **Mobile-first permit view.** The technician is holding a phone with gloves on, outdoors, in sunlight. Big targets, high contrast, works one-handed.

- **QR code per permit** so a safety walk-around can scan and see status.

- **Digital signature capture** on approval (canvas-drawn is fine).

7. Explicitly out of scope

Don't build these. Building them will not earn points and will cost you time:

- Offline sync

- Real-time websockets / multi-user live updates

- Notifications by email/SMS/push (a stub function that logs "would notify X" is fine and shows you thought about it)

- File/photo uploads (a URL field is enough)

- Multi-tenancy, billing, org onboarding

- A pretty marketing landing page for the module

Technical requirements

Stack: your choice, but we work in **React + TypeScript on the frontend and Node/TypeScript with Postgres on the backend**, so building in that gets you closest to the actual job. Next.js full-stack is a perfectly good answer. Python/Django is acceptable if it's genuinely your strongest.

Non-negotiables:

- A real database with a real schema. Not JSON files, not in-memory arrays.

- **Include your schema/migrations in the repo**, plus a seed script that creates 4 users (one per role), 2 plants, ~6 equipment items, and ~10 permits spread across different statuses. We must be able to log in and see a populated system in under two minutes.

- A README with: setup steps that actually work, demo login credentials for all four roles, the decisions you made where the spec was silent, what you'd build next, and what you knowingly left broken.

- At least a few tests around the state machine and the permission rules. We're not asking for full coverage — we're asking whether you know what's worth testing.

- Reasonable commit history. One giant "initial commit" is a bad signal.

On AI: use it. We use it every day and you'd be foolish not to. But you must be able to explain every line in the Loom, and we will ask about specific decisions. Code you can't defend is worse than no code. Add a short section in the README saying what you used it for.

How we'll evaluate

| Weight | What we're looking at |

|---|---|

| 30% | **Data model and state machine** — is the permit type abstraction clean? Are the rules enforced server-side? Can we break it by hitting the API directly? |

| 20% | **Does it actually work** — deployed, seeded, no dead ends, no crashes on the obvious paths. |

| 20% | **Domain understanding** — does the UI show that you understood what a permit *is*, or does it read as a generic CRUD app? |

| 15% | **Code quality** — structure, naming, sensible boundaries, error handling. Not cleverness. |

| 15% | **The Loom and README** — can you explain a decision, admit a weakness, and communicate like someone we'd want on a call with a customer? |

What will hurt you

- Frontend-only validation. We will bypass it.

- Four near-identical permit forms with duplicated code.

- Finished-looking but not deployed, or deployed but not seeded.

- Building the out-of-scope list instead of the core.

- A README that says "npm install && npm run dev" and doesn't work on a clean machine.

- Silence. If you're stuck for a day, email us — that's a good signal, not a bad one.

What we're actually testing

Whether you can take an unfamiliar, safety-critical business problem, model it properly, and ship something that runs. That's the entire job here.

A note on the role

Once you're in, you'll be visiting factories with us — plants around Chennai, helmet on, watching maintenance technicians and safety officers use software like this in conditions you cannot imagine from a desk. Developers who've stood on a shop floor build noticeably different software.

You don't need to visit anywhere for this assignment. But if that part of the job sounds like a chore rather than the interesting part, this isn't the right role for you.

Questions are welcome — email them. Asking a good question is a positive signal here.

put this for the web developer intern