How I'm Building the PMO RAID Register

Author: Apurv Duhan, PMP | **Last updated:** 14 June 2026

What this document is: The spec (the other file) covers *what* this tool does and *why*. This one covers *how* I plan to build it — the tools I picked, how the information is organized, what screens and buttons exist, and the order I'll build things in.

It has two readers in mind. The first is the AI coding agent that will actually write the code: the clearer my plan, the closer the result matches what I intended, instead of the agent guessing. The second is any person reviewing my work — a teammate or a hiring manager — who can read this and see that I made deliberate decisions up front rather than letting the AI improvise.

---

1. The short version of what I'm building

A small web app that one project manager runs on their own computer. It tracks the four things every project has to keep an eye on — Risks, Assumptions, Issues, and Dependencies — keeps a full history of every change, and prints a clean status summary to share with leadership.

---

2. The choices I made, and why

I kept every choice as plain and boring as I could, on purpose. One of the project's core rules is "use the simplest thing that does the job," so I deliberately avoided anything fancy.

- **It runs on the PM's own machine, with no login.** Only one person uses this tool, so building accounts, passwords, and a hosted server would be effort spent solving a problem I don't have. If a future version needs multiple users, that's when I'd add it.
- **All the data lives in a single file on that machine.** I'm using SQLite for this, which is a tiny, built-in database that needs zero setup. For one person tracking a few hundred items, it's more than enough and there's nothing to install or maintain.
- **The pages are plain web pages, not a heavy app.** I'm building it in Python with a lightweight web framework (FastAPI) and simple HTML pages. Fewer moving parts means it's easier to keep accessible for everyone, and nothing on the page tracks the user or calls out to the internet.
- **The change history is locked.** Every edit gets written to a separate history record that the app is never allowed to alter or erase. I'm backing that up at the database level too,

so even a future coding mistake can't quietly rewrite the audit trail. This is the part that makes the whole "auditable" promise real, so it's not optional.

---

3. Checking the plan against the project's own rules

Before any code gets written, I check this plan against the non-negotiable rules I set for the project (those live in the constitution file). Here's how the plan honors each, in plain terms:

- **Everything traces back to a requirement.** Every screen, button, and test points back to a numbered requirement from the spec, so I can always show the line from "what we agreed to build" to "here's the code and the test that proves it."
- **Nothing changes without a record.** Every create, edit, and delete writes a history entry, and that history can't be altered.
- **Tests come first.** For each thing the tool should do, I write the check that proves it works *before* writing the code that does it.
- **Privacy stays tight.** The tool only stores an item owner's name and email. No analytics, no trackers, no data leaving the machine.
- **It's usable by everyone.** The pages work with a keyboard, have clear labels, and never rely on color alone to show status (so it's readable for colorblind users).
- **Problems are never silent.** If something goes wrong, the user sees a plain-English message and the technical detail gets logged.

If any choice had broken one of these rules, I'd fix the plan before building — not patch it afterward.

---

4. How the pieces fit together

The information. The tool keeps two kinds of records:

1. *RAID items* — the actual things being tracked. Each one has a type (Risk, Assumption, Issue, or Dependency), a title and description, an owner (name and email), a severity (Low to Critical), a status (Open, In Progress, Mitigating, Resolved, Closed), and the dates it was created and last changed. Each gets a readable ID like `RISK-001`.
2. *History entries* — a permanent, unchangeable log. Each entry records what changed, who changed it, when, and the before-and-after values.

The screens and actions. Below is everything the tool can do. The numbers in brackets are the requirement each action satisfies — that's the traceability thread I mentioned, and it's worth keeping because it's exactly what proves the work was built to spec.

- See the full register, with filters for type and status and a live count *(req. 7)*
- Add a new item, which assigns its ID and logs the creation *(reqs. 1, 2, 5)*
- Open a single item and read its full change history *(req. 6)*
- Edit an item, logging each field that changed *(reqs. 4, 5)*
- Delete an item, but only after a confirmation step, and the deletion is logged *(reqs. 8, 5)*
- Export one printable status report, grouped by type and stamped with the date and time *(req. 9)*

Anywhere the user can type something in, the tool checks the input and, if it's wrong, shows a clear message without throwing away the rest of what they entered *(reqs. 10, 11, 12)*.

---

5. The order I'll build it in

I won't write the task-by-task list here — that's the next step. But the order follows a few simple rules: the data and the locked history come first, then the input checks, then the screens, then the export, and finally a pass to confirm it's accessible and that errors are handled cleanly. For each feature, the test that proves it works gets written before the feature itself.

---

6. What I'm deliberately keeping simple

I didn't have to bend any of the project's rules to make this plan work, and I didn't add anything the spec didn't ask for. If a future need forces something more complicated (multiple users, for example), I'll write down why here and update the project rules first — so even the decision to add complexity is on the record.

---

7. Where things stand

- ☑ Spec finished, no open questions
- ☑ Plan checked against the project rules — all clear
- ☑ Decisions and structure written down (this document)
- ☐ Break the plan into a task list
- ☐ Final consistency check across all the documents
- ☐ Build it
