What the PMO RAID Register Is, and What It Needs to Do

What this document is: This is the plain description of the tool — what it's for and what it has to do — written so anyone can read and approve it without needing a technical background. A separate document covers *how* it gets built. This one stays focused on the *what* and the *why*.

Status: Ready to build (all open questions answered, see section 5) **Author:** Apurv Duhan, PMP **Last updated:** 14 June 2026

---

1. The problem I'm solving

Every project has four things a manager constantly has to watch: **Risks** (what could go wrong), **Assumptions** (what we're taking for granted), **Issues** (what's already gone wrong), and **Dependencies** (what we're waiting on from someone else). Together these are called a RAID log.

Most teams keep this in a spreadsheet that gets messy fast. There's no record of who changed what, ownership gets fuzzy, and pulling together a clean update for leadership means rebuilding the whole thing by hand every week.

This tool fixes that. It gives a project manager one place to log these items, keep them current, see the full history behind every one of them, and print a tidy status summary on demand.

---

2. Who uses it and how

One project manager, on their own computer. There's no sign-in and no shared server in this first version, because only one person uses it. Every change is recorded under a display name the PM sets once when they start using the tool.

---

3. What it should do, walked through

Here are the main things a PM should be able to do, described as they'd actually happen:

Logging a new item. The PM adds an item, picks its type (Risk, Assumption, Issue, or Dependency), gives it a title and description, names an owner, and sets how serious it is. The tool gives it a readable ID like `RISK-001`, marks it "Open," and records that it was created.

Updating an item without losing the past. The PM changes an item's status or edits its details. The item shows the new information, but its history still shows what it said before, who changed it, and when.

Finding things quickly. The PM filters the list by type and status — say, only open dependencies — and sees a count of what matches.

Producing a status report. The PM exports a single printable summary, grouped by type, showing each item's current status, owner, and severity, stamped with the date and time it was produced.

Removing something safely. The PM deletes an item, but only after confirming they mean to, and the deletion is recorded in the history.

A few edge cases worth stating plainly: an item with no title is refused with a clear message; exporting an empty list still produces a valid report that simply says nothing's been logged yet; and a bad entry (like an invalid severity) is refused without wiping out the rest of what the PM typed.

---

4. The specific requirements

These are the testable requirements. Each is numbered so that, later, every screen and every test can point back to the exact one it satisfies. That numbered thread is what lets me prove the finished tool actually does what was agreed.

1. Let the user create an item with a type, title, description, owner (name and email), severity, and status, choosing type, severity, and status from set lists (see requirement 13).
2. Give every item a unique, readable ID when it's created.
3. Support a sensible set of statuses for an item's life cycle (from requirement 13).
4. Let the user edit any changeable detail of an existing item.
5. Record every create, edit, and delete in a permanent history that can't be altered — capturing the time, who did it, and the before-and-after values. The "who" is the local display name (requirement 12).
6. Show an item's full history whenever the user asks for it.
7. Let the user filter the list by type and status, with a live count of what matches.
8. Only delete an item after the user explicitly confirms.

9. Export one printable status report grouped by type, showing current status, owner, and severity, stamped with the date and time.
10. Refuse invalid entries with a clear, plain message, without discarding the user's other valid input.
11. Never rely on color alone to show an item's status, so it's readable for colorblind users.
12. Use one display name the user sets locally as the "who" for all history; no passwords or accounts in this version.
13. Use these set lists, fixed for now:
    - **Type:** Risk, Assumption, Issue, Dependency
    - **Severity:** Low, Medium, High, Critical
    - **Status:** Open, In Progress, Mitigating, Resolved, Closed

---

5. Questions I answered before building

Early on, three things were genuinely open. Rather than let the tool guess, I settled them on the record:

- **One user or many?** One, local, no sign-in for this version. Multiple users can come later.
- **One export format or several?** One printable report. A data-file export can come later.
- **Should the type/severity/status lists be editable?** No, they're fixed for now. Making them editable can come later.

---

6. What this version deliberately leaves out

So expectations are clear, this first version does **not** include: tracking multiple projects at once, user accounts or permissions, connections to other project-management tools, email or notifications, or two people editing at the same time. Each of those is a deliberate "later," not an oversight.

---

7. Sign-off checklist

- ☑ All the open questions are answered (section 5)
- ☐ Every requirement can be tested
- ☐ Every requirement traces to something in section 3 or a project rule
- ☐ No technical "how" has crept into this document — it stays about the "what"

- [ ] The locked history covers every change, with no exceptions
- [ ] A non-technical stakeholder has read this and approved it

---