Vibe safely. *The non-engineer's build sheet.*

How to build with AI when you are not a software engineer: what to decide before the first prompt, how to structure the build so it can be reviewed, and how to know when to stop.

v1.0 · September 2026
ai-sustained.com
Companion to Case Study 006

THE RULE THAT CARRIES THIS PAGE

A demo proves it works. It does not prove it is safe. Before anyone but you depends on the thing, someone who can read the code signs it off. Everything below is about making that sign-off short.

THE LIFECYCLE **Decide what → Design how → Build → Test → Release → Run and fix**

Vibe coding made one of these cheap. The other five cost what they always did. Skip one and it comes back as an incident.

A The gate test

Four questions sort any idea into a lane. Answer them before you open the chat. One amber answer puts the whole app in amber; one red puts it in red.

Ask yourself	GREEN · VIBE AWAY **No engineer needed**	AMBER · VIBE, THEN REVIEW **Before anyone else depends on it**	RED · NOT WITHOUT AN ENGINEER **Design it with one from the start**
Who else uses it?	Only you.	A few colleagues you can name.	Anyone outside the business, or anyone you cannot name.
Whose data is in it?	Yours, or made up.	Staff or customer names, internal figures.	Payment, health, children's, or anything a regulator has a view on.
What does it write to?	Nothing, or a copy you can throw away.	A shared store the team relies on.	A system of record: finance, HR, the customer database.
What happens when it breaks, or you leave?	You run it again.	Someone has to fix it this week.	It must be recoverable today, by someone who is not you.
Typical examples	Personal dashboards, calculators, trackers, one-off analysis scripts, throwaway prototypes and mock-ups, first-draft generators a human checks, static pages with no login or form.	Shared internal tools, small team apps, automations that run unattended, anything holding an API key, anything that replaced a shared spreadsheet.	Customer-facing apps with accounts, anything taking payment, anything writing to a system of record, anything the business needs back within the day.

Green is the lane where being wrong costs you a rerun instead of an incident.

B Before the first prompt

One line each, written down, kept with the code. If you cannot answer one, that is the first prompt.

01 **Users.** Who, how many, which roles, who is admin. Write the number of users the design assumes.

02 **Access.** How people get in, and where the rules are enforced. Never only in the browser: anything decided on the user's machine can be undone on the user's machine.

03 **Data.** What it holds, whose it is, whether any of it is personal, and where it lives. If it is personal, the app is amber at best.

04 **Ownership.** Which account and environment the business owns. Never a personal one.

05 **Done.** The evidence needed before anyone else uses it, agreed in advance. A walkthrough is one input, and it is the weakest one.

C While you build

06 **Version control from day one.** Ask for a repository before the first feature. Every change recorded with a note on why, and a way back when the AI breaks something.

07 **Plan first.** Ask for a plan and a file structure before any code. Screens, rules and data access in separate files.

08 **One feature at a time.** Each with a sentence saying what a user can do and what they cannot. That sentence is your test.

09 **Rules out of reach.** Every rule that matters lives server-side or in database rules. Ask the AI to show you where, and how a user would get round it.

10 **No secrets in the front end.** If a feature needs a key in the browser, cut the feature. Cosmetic extras leak keys.

11 **Tests as you go, run by a machine.** Ask the AI to wire a pipeline (CI/CD): tests run on every change, and the app deploys only when they pass. An hour, once. Nobody hand-carries files to production again.

12 **A README the AI updates.** What it does, how it is deployed, what is known to be weak. Your reviewer starts here.

13 **Delete what you are not using.** Dead dependencies, sample features, anything installed to see if it worked.

D Before anyone else depends on it

14 **Move it home.** A business-owned account and environment, with backups switched on and someone other than you holding the keys.

15 **Company sign-in.** Through the organisation's identity provider. Retire any password login you run yourself.

16 **Roles, enforced.** Where users cannot reach them, least privilege by default. If you simplified roles on purpose, write down why and for how many users. The note turns a shortcut into a decision.

17 **A reviewer who can read the code.** Internal engineer or external agency, either works. Hand them the README, your own risk list and permission to be blunt.

18 **Run it like a service.** Monitoring, a named owner and a handover note. If you left tomorrow, who runs it?

E Ask the reviewer for

- → What must change before wider use, in priority order.
- → What they could not test, and what would.
- → Whether it is remediation or a rebuild, and why.
- → Who supports it afterwards, and what that costs.

F Bin these

- × Production running in a personal cloud account.
- × A role dropped for convenience with nothing written down about what replaces it, or for how many users that was fine.
- × A key shipped to the browser to power a cosmetic feature.
- × One file, one author, no tests, and a README that admits it.
- × Releasing from a laptop: whatever was last on your machine is the live app.
- × Treating the demo as the release decision.