Ai Pathway

The AI-Native Build Kit

From one paragraph of intent to a running prototype — the prompt pipeline.

eWorks Accelerator 2026

Build with us: learn@aipathway.org

1. AI Customer Discovery Prompt

You are a skeptical product manager + customer-discovery coach.

Your job is to decide whether this idea is **worth building**, not to design it. Challenge my assumptions; if it's weak, say so.

Step 1 — interview me first. Ask me up to 10 sharp questions about the problem, customers, current workflow, alternatives, buyer/budget, and any **regulatory / ethical / safety / privacy / trust constraints** this domain carries. **Wait for my answers. Do not analyze yet.**
Step 2 — only after I answer, produce:

1. **Pains** — prioritized, in a table by customer segment (which is sharpest? which is riskiest?)
2. **Who feels it most acutely** (and who *buys* — name the budget owner)
3. **Alternatives** and why they're inadequate (including "inertia / nothing")
4. **Assumptions** — as a labeled list **A1, A2, …**, each tagged **[RISKY]** or **[normal]** (these will feed the hypotheses step)
5. **Personas** (named, with their goal + their hesitation)
6. **JTBD** per persona (job / trigger / desired outcome)
7. **Smallest MVP** — *and what to deliberately NOT build yet*
8. **Verdict:** BUILD / VALIDATE-FIRST / REFRAME — with a one-line **kill condition** (what result means stop or shrink the product)

Keep it decision-useful, not exhaustive. Domain constraints must appear explicitly, not be assumed away.

2. AI Product Specification

Input: the discovery output above — specifically its prioritized pains, personas/JTBD, and labeled assumptions A1–A5. Reference them; do not re-derive them. You own the solution, discovery owns the problem.
Produce a spec with:

- Vision (1 paragraph)
- Users & goals — reference the personas by name from discovery
- Functional requirements — and for each, tag which pain/JTBD it serves (e.g. FR3 → P1, Teacher-JTBD). No requirement that doesn't trace to a discovered pain.
- Cross-functional (non-functional) — security/privacy/compliance/trust that the domain constraints from discovery demand
- Success metrics — measurable, tied to the top pains
- MVP scope and explicit Out-of-scope (carry forward discovery's "deliberately not yet")
- Technical assumptions

Keep it a solution definition, not a re-listing of the problem. Flag any requirement that depends on a [RISKY] assumption — that dependency feeds the hypotheses step.

3. Product Hypotheses

Input: the [RISKY] assumptions A1–A5 from discovery and the MVP scope from the spec. Turn each risky assumption into one testable hypothesis — do not invent new ones.
For each hypothesis output:

- ID + source (H1 ← A1)
- the bet (falsifiable statement)
- risk level and why
- the cheapest experiment that could disprove it (who, how many, what metric)
- success criterion AND kill criterion (what result means stop or shrink the product)
- Prioritization: rank by (risk × cheapness) — riskiest+cheapest first

End with the single "validate this before writing the tech spec" hypothesis. This ranked list is exactly what Step 6 (Validate) executes.

4. UX Specification (Figma.md)

Input: the spec (functional requirements, user flows, MVP scope, MVP rules) and the personas + domain tone from discovery.

Realize the spec — do not add features. Every screen must trace to a functional requirement or user flow.

If realizing the interface reveals a needed field/state/rule that the spec doesn't cover, do NOT invent it silently — list it under SPEC-GAPS for me to fold back into the spec. (The chain loops; this is where.)

Own the interface layer only.

Output:

- Design principles — derived from the domain tone in discovery (e.g. trust, low cognitive load), with your palette + type choices to match
- Information architecture / navigation
- Screens — for each, a consistent block:
- Purpose · serves → FR/flow · components · data shown → (which data-model entities) · states (default / empty / loading / error) · primary action
- Global components (reused across screens)
- State taxonomy — empty, loading, error, permission-denied — as first-class, not afterthoughts
- Responsive behavior
- Figma page structure (how to organize it) + → next step: prototype
- SPEC-GAPS (anything the UX needed that the spec lacked)

Honor the spec's MVP rules/constraints exactly. Keep it buildable — concrete enough that a coding agent can turn it into a working prototype in step 5. Exportable file.

5. Create a Prototype (Node.js / Vercel)

6. Validate