Survive the Vibe

The Launch-Risk Checklist for Half-Built AI Apps

A founder-facing diagnostic for deciding whether an app is actually ready to launch, needs targeted stabilization, or is still being held together by optimism.

How to use this

Work through each section honestly. If you answer **no**, **not really**, or **only when I am the one testing it** to several items, treat that as real launch risk. The goal is not perfection. The goal is to stop confusing a demoable product with a trustworthy one.

Scoring shortcut: Mostly yes = probably launchable with normal caution. Several shaky answers = targeted stabilization likely needed. Multiple red zones = do not treat this like a ready product yet.

1. Users

Onboarding

- Can a new user sign up without staff intervention?
- Does the account land in the right workspace, role, or plan state?
- Can invited users complete the flow without weirdness?
- Are basic error states understandable instead of cryptic?

Core flow

- Can users complete the main job the product promises?
- Does that still work outside the ideal happy path?
- Can a user recover from a partial failure without support?
- Do permissions behave the way the UI implies?

Trust signals

- Can someone other than the founder test the flow and succeed?
- Would you be comfortable watching a stranger use it live?
- Are the riskiest edge cases at least known and documented?
- Does the product fail in understandable ways?

2. Money

Billing

- Would you trust a real customer to pay through the current flow?
- Do plan changes and cancellations behave predictably?
- Are webhooks and retries observable enough to debug?
- Can you explain exactly how entitlement logic works?

Access control

- Does paid access reliably unlock the right features?
- Can users lose access incorrectly because of stale state?
- Is there one clear source of truth for subscription status?
- Would a billing failure create a support fire?

Revenue risk

- Would you notice if the app silently undercharged or over-granted?
- Can someone manually verify and correct a bad account state?
- Is there a clean path for refunds or account fixes?
- Do you trust this enough to put it in front of paying users?

3. Operations

Deploys

- Can someone besides the original builder deploy it safely?
- Are environment variables and secrets documented?
- Is there a rollback path if a release goes sideways?
- Do staging and production behave close enough to trust?

Observability

- Do you know where to look when the app breaks?
- Are critical flows logged or otherwise visible?
- Would you notice failures before users complain?
- Can you isolate whether a problem is frontend, backend, auth, or infra?

Ownership

- Could another engineer explain the architecture after a short handoff?
- Is important product logic written down anywhere?
- Do you know which areas are fragile versus merely ugly?
- Is the app becoming more legible over time instead of less?

4. Decision quality

Patch: The structure is mostly sound. The problem is local: edge cases, shallow bugs, missing recovery, small deploy cleanup, or product messaging around a basically valid core.

Refactor or partial rebuild: A concentrated risk zone — auth, billing, onboarding, entitlement logic, or deploy process — is making the whole product harder to trust than it should be.

5. Three final questions

- If this app got 10x more usage next month, what would scare you first?
- If the original builder disappeared for two weeks, what would become too risky to touch?
- If a customer paid tomorrow, what part of the product would make you nervous to promise confidently?

Next step

If this checklist exposed concentrated risk, do not just squint harder. That usually means the product needs diagnosis, tighter priorities, and a calmer plan for stabilization before launch pressure gets more expensive.