Hector Garcia

The vibe coding pre launch checklist

GUIDES · 9 MIN READ · SEP 8, 2026

You built something with AI and it works on your screen. This is the gap between working and safe to hand to real users. Run the eight prompts below before you launch: each one points an AI at your code and tells you exactly what to fix. Start with the keys and cost prompts, those are the ones that can cost you thousands overnight.

WORKING IS NOT THE SAME AS REAL

Vibe coding gets you to working in a night. The app runs, the demo looks great, you send it to a group chat. Working means it runs on your screen right now.

Real means other people can use it, their data is safe, your costs are predictable, and it does not fall over the first time something unexpected happens. Getting to real is the part nobody films.

Note: You do not need everything a full engineering team would build. You need the short list that stops the expensive mistakes. Paste each prompt into Cursor with your repo open, or into Claude with your code attached.

COST

AI features cost money for every user you get, and a leaked key or a runaway loop can turn a weekend project into a four figure bill overnight.

FIND EVERY PLACE THIS APP SPENDS MONEY

```
You are a senior engineer reviewing my project before launch.

Codebase context:
[point Cursor at the repo, or paste your stack and main files]

Task:
1. List every external service this app calls that charges money: model APIs, hosting, database, storage, email, anything metered.
2. For each one, tell me where in the code the calls happen and what triggers them.
3. Flag any call that runs in a loop, on every request, or without a cache.
4. Tell me exactly where to set a hard spend cap and a billing alert for each provider.

Output format:
- One section per provider
- Under each: trigger, file and line, risk level (low, medium, high), the setting to change
```

WORK OUT WHAT ONE USER COSTS ME

```
You are helping me price and protect a vibe coded app before launch.

Details:
- What the app does: [one line]
- AI calls per normal user session: [guess if unsure]
- Model and rough tokens per call: [model, or say unknown]
- Current price to the user: [free, or amount]

Task:
1. Estimate the AI and infra cost of one active user per day and per month. Show your assumptions.
2. Estimate the cost if the app suddenly gets 10,000 users in a week.
3. Tell me the point where this loses money and what to change first: rate limits, caching, a cheaper model, or a paywall.

Output format:
- Cost per user (day and month)
- Viral scenario cost
- The single most important change, and why
```

DATA

If you store real user information, that is a legal responsibility, not a detail. A leak is not something you can fix after the fact.

SHOW ME EVERY PIECE OF USER DATA I TOUCH

```
You are a privacy focused engineer auditing my app before launch.

Codebase context:
[point Cursor at the repo, or paste your schema and API routes]

Task:
1. List every field of user data the app collects, stores, or sends anywhere.
2. For each field, say where it is stored, how long it is kept, and everything that
can read it, including third party APIs and AI calls.
3. Flag any field that is collected but never actually used.
4. Flag any personal data sent to a model provider and whether that provider trains
on it.

Output format:
- A table: field, stored where, retention, who can read it, used for
- A short list of fields to stop collecting
- A short list of real risks, most serious first
```

TRIM WHAT I STORE AND ADD A DELETE PATH

```
You are helping me tighten data handling in a vibe coded app.

Current data model:
[paste schema or describe it]

Task:
1. For each stored field, tell me if I actually need it for the core feature. Recommend
keep or drop.
2. Show me the code changes to stop collecting the dropped fields.
3. Add a way for a user to request deletion of their data, and describe what that
function needs to remove.
4. Give me one plain English sentence I can put in a privacy note for this app.

Output format:
- Keep or drop list with one reason each
- Code changes, file by file
- The deletion function outline
- The privacy note sentence
```

SECURITY

Most damage comes from a handful of basics being skipped. These prompts check them for you.

FIND LEAKED KEYS AND CLIENT SIDE SECRETS

```
You are a security engineer reviewing my project before launch.

Codebase context:
[point Cursor at the repo]

Task:
1. Find every API key, token, password, or secret in the codebase, including in client side code, config files, and committed history.
2. For each, say whether it is currently exposed to the browser or the public repo.
3. Tell me which ones must move to server side environment variables and how to do it for my stack.
4. List the keys I should rotate now because they may already be leaked.

Output format:
- Table: secret, location, exposed (yes or no), fix
- A rotate now list
- The exact steps to move secrets server side for [your framework and host]
```

CHECK THAT USERS CAN ONLY TOUCH THEIR OWN DATA

```
You are a senior engineer checking authorization in my app before launch.

Codebase context:
[point Cursor at the repo, or paste your API routes and auth setup]

Task:
1. List every endpoint or query that reads or writes data.
2. For each, tell me what stops user A from reading or changing user B's data. If nothing does, say so clearly.
3. Flag any route that trusts an id from the request without checking the logged in user owns it.
4. Give me the smallest set of fixes to close the gaps.

Output format:
- Table: route, action, ownership check present (yes, no, partial), risk
- Ordered fix list, highest risk first
```

STABILITY

Vibe coded projects break quietly, because there are no tests and every AI edit can undo something that already worked.

WRITE TESTS FOR WHAT THE APP DEPENDS ON

```
You are a senior engineer helping me make a vibe coded app safe to change.

Codebase context:
[point Cursor at the repo]

Task:
1. Identify the 5 to 10 functions or flows the whole app depends on: auth, payments, the core action, data writes.
2. For each, write tests that cover the normal case and the two most likely failure cases.
3. Add a single command I can run to execute all of them.
4. Tell me which currently fail, if any.

Output format:
- The test files, ready to paste in
- The run command
- A list of anything already broken
```

CHECK THE LAST CHANGES DID NOT BREAK ANYTHING

```
You are reviewing my app right before I deploy.

Recent changes:
[paste the diff, the last few commits, or describe what you just built]

Task:
1. List everything these changes could have broken, including features that seem unrelated.
2. For each, give me a 30 second manual check to confirm it still works.
3. Flag anything that needs a test before this ships.
4. Give me a go or no go recommendation.

Output format:
- Risk list with a manual check for each
- Tests to add
- Go or no go, with the one thing that would change your answer
```

IF A PROMPT TURNS SOMETHING UP

1. Close the cost and security gaps before you launch. Those are the ones that get expensive fast.
2. Data and stability gaps can ship with a written plan to fix them in week one, as long as the risk is low.
3. Want a second set of eyes on something you are about to launch? Reply to the DM that sent you this and we will run through it together.

Questions? Reply to the newsletter or reach out via email.

Hector Garcia
@hex.gar · contact@hexgarcia.com