MINI PACK

It Broke in Production

Find your logs, catch errors before your users tell you, and fix what breaks on the live site.

Written for use with Claude Code, Cursor, ChatGPT or Claude · Not yet tested against them

10 prompts · 4 sections

Welcome! How to Use This Pack

Your app worked when you tested it. Then real people started using it, and now someone says "it's broken" with no other details. You don't see an error. You're not sure where the logs are. And the AI keeps asking what the error message says.

This pack walks you through it, one step at a time. First, find where your app's logs live and what they're telling you. Then add error tracking, so a new error reaches your inbox before it reaches a user's complaint. Then make your errors useful: a clear message for users and a searchable log line for you. Last, a short weekly check and a simple way to write down what went wrong, so it doesn't happen twice.

One safety rule before you start: logs often contain email addresses, tokens, and other private details. Before you paste logs into any AI, remove passwords, API keys, tokens, and anything from your `.env` file. Prompt 3.2 helps you stop logging those things in the first place.

Act fast on Vercel's logs. Vercel's docs say runtime logs are kept for 1 hour on the Hobby plan and 1 day on Pro (up to 30 days with Observability Plus). If a user reports a problem, copy the matching log lines right away, before they disappear. Error tracking (Section 2) keeps its own record of each error, so you depend less on that window.

Using our other packs too? The Vibe Code Starter Pack has prompts for fixing a failed deploy and explaining an error you can see. Fix My Bugs is for bugs you hit while building. Survive Your First Traffic Spike (Prompt 1.2) is for when the whole site is down under heavy traffic. This pack is for the everyday errors real users run into after launch.

How each prompt is laid out

- **Title**: what the prompt is.
- **Use this when...**: one or two sentences on what the prompt is for and when to reach for it.
- **Use in**: which AI to paste it into.
- **Fill in**: the blanks to replace before you paste.
- **The prompt**: the box you copy.
- **Tips to customize**: optional ways to change it.

Anything in **[BRACKETS IN CAPS]** is a blank to fill in.

Written for use with: Claude Code or Cursor's Agent, opened in your project, for prompts that change code, and ChatGPT or Claude in a normal chat for prompts that only need what you paste in. The examples use Vercel, Supabase, Next.js, and Sentry, but the prompts ask the AI to adapt to your stack. Not yet tested against them.

The words, in one line each. Production is your live site that real users visit. **Logs** are the lines your app writes as it runs, kept by your host and your database. **Error tracking** is a service (like Sentry) that collects your app's errors in one place and groups repeats together. An **alert** is a message (usually email) you get when something new breaks. An **uptime check** is a service visiting your site every few minutes to see if it's still up. A **request ID** is a code that ties one visitor's action to the log lines it created.

Quick Start: Find Your Problem

- **No idea where your logs are?** Prompt 1.1.
- **A user says "it's broken" and that's all you know?** Prompt 1.2.
- **Want to see every error, not just the ones people report?** Prompt 2.1.
- **Want an email when something new breaks?** Prompt 2.2.
- **Want to know the site is down before a user tells you?** Prompt 2.3.
- **Users see a blank page or "something went wrong" with no next step?** Prompt 3.1.
- **Your logs are a mess, or full of private data?** Prompt 3.2.
- **Ready to fix a live bug without breaking something else?** Prompt 3.3.
- **Want a quick routine to catch problems early?** Prompt 4.1.
- **Just fixed something bad and want to make sure it never happens again?** Prompt 4.2.

Golden rule: Find the exact log line first, then fix. A fix based on a guess is just a new bug.

What's Inside

1. Find Out What Happened (2 prompts)
2. Hear About Errors Before Your Users Do (3 prompts)
3. Make Errors Useful, Then Fix Them (3 prompts)
4. Keep It From Happening Again (2 prompts)

SECTION 1

Find Out What Happened

Before you can fix anything, you need to see what the app saw. That means knowing where each kind of log lives.

1.1 Where Are My Logs?

Use this when something broke on your live site and you don't know where to look. It's for getting a short map of every place your app keeps logs, what each one shows, and how to search it.

Use in: Claude Code or Cursor's Agent, opened in your project.

Fill in: [WHERE MY APP IS HOSTED] (e.g. "Vercel"), [MY DATABASE AND LOGIN SERVICE] (e.g. "Supabase for the database and login"), [OTHER SERVICES I USE] (e.g. "Stripe for payments, Resend for email", or "none").

```
Help me find every place my live app keeps logs. I'm a beginner, so keep it plain.
Don't change any code.

Where my app is hosted: [WHERE MY APP IS HOSTED]
My database and login service: [MY DATABASE AND LOGIN SERVICE]
Other services I use: [OTHER SERVICES I USE]

Please:
1. Look through the project so you know my real setup: framework, server code, API
routes, and which outside services the code calls. Tell me if anything you find
doesn't match what I wrote above.
2. Make a short table with one row per place logs live (for example: my host's
runtime logs, build logs, database logs, login logs, and any outside service's
dashboard). For each row, include: what kind of problem shows up there, where to
find it, and one example of what an error there might look like.
3. Tell me where my own console.log and console.error lines end up once the app is
live.
4. Tell me how long each place keeps logs, if you know. If you're not sure, tell
me where to check in that service's docs instead of guessing.
5. Give me the 3 places to check first when a user says "it's broken," in order.
Keep the whole answer under one page.
```

💡 **Tips to customize:** On Vercel, open your project and choose **Logs** in the sidebar to see runtime logs. You can filter by level (warning, error, fatal), status code, and route, and Vercel's docs note that the times shown are in UTC. From the terminal, `vercel logs --environment production --query 'level:error' --since 1h` shows recent production errors. Build logs are separate and live on the deployment itself. In Supabase, the **Logs** page in your dashboard lets you pick a log type,

such as API Gateway, Postgres, Auth, Storage, or Edge Functions, and filter by level or text. How long Supabase keeps logs depends on your plan.

1.2 Turn "It's Broken" Into a Fix Plan

Use this when a user reports a problem with almost no details, and you need to figure out what actually happened. It's for matching their report to your logs, reproducing the bug yourself, and making a plan before anyone touches the code.

Use in: Claude Code or Cursor's Agent, opened in your project. A normal chat works too if you paste the relevant code.

Fill in: [WHAT THE USER SAID] (paste their message word for word), [WHEN IT HAPPENED] (e.g. "around 2:15 PM my time on Tuesday"), [WHAT THEY WERE DOING] (e.g. "saving their profile", or "not sure"), [MATCHING LOG LINES] (paste the log lines from around that time, with emails, tokens, and keys removed, or write "none found").

```
A user reported a problem on my live app. Help me figure out what happened and
make a plan. Don't change any code yet.

What the user said: [WHAT THE USER SAID]
When it happened: [WHEN IT HAPPENED]
What they were doing: [WHAT THEY WERE DOING]
Matching log lines: [MATCHING LOG LINES]

Please:
1. Tell me in plain words what you think happened, and quote the exact log line
that supports it. If the logs don't show anything useful, say so.
2. List up to 3 possible causes, most likely first. For each one, say what in the
code or logs points to it.
3. Write steps I can follow to make the bug happen myself, on my own computer or a
preview deploy, not on the live site with real data.
4. Tell me what to check or log to rule each cause in or out.
5. List up to 3 short questions I could ask the user that would help most, written
so a non-technical person can answer them.
Wait until I've reproduced the bug before suggesting a fix.
```

💡 **Tips to customize:** Convert the time the user gives you to UTC before you search Vercel's logs, since that's the time zone Vercel shows. If you use Next.js and the user saw a "something went wrong" page, the Next.js docs explain that in production, errors from server code reach the browser as a generic message with an ID called a digest. Search your server logs for that digest to find the real error. If you can't reproduce it after a couple of tries, Prompt 1.3 in our Fix My Bugs pack helps you build a small reproduction.

SECTION 2

Hear About Errors Before Your Users Do

Most users who hit an error never report it. They just leave. Error tracking and an uptime check tell you anyway.

2.1 Add Error Tracking to My App

Use this when you only find out about errors because someone complains. It's for adding an error tracking service (Sentry, in this example) so every error on your live site gets recorded with the details you need to fix it.

Use in: Claude Code or Cursor's Agent, opened in your project.

Fill in: [MY FRAMEWORK AND VERSION] (e.g. "Next.js 15 with the App Router"; check package.json if you're not sure), [ERROR TRACKING SERVICE] (e.g. "Sentry"), [ANYTHING I DON'T WANT RECORDED] (e.g. "session replays", "users' email addresses", or "nothing extra").

```
Help me add error tracking to my app so I can see errors that happen on the live
site. Explain each step in plain words, and show me every file you plan to change
before you change it.

My framework and version: [MY FRAMEWORK AND VERSION]
Error tracking service: [ERROR TRACKING SERVICE]
Anything I don't want recorded: [ANYTHING I DON'T WANT RECORDED]

Please:
1. Check the service's current official setup guide for my framework, and tell me
which version of the guide you're following. Don't rely on memory, because setup
steps change.
2. Tell me exactly which steps I need to do myself (like creating an account and a
project, or running a setup command that asks me questions) and which ones you'll
do.
3. Make sure errors from both the browser and the server get reported.
4. Keep secret values, like auth tokens, out of git and out of any code that runs
in the browser. Tell me which values I need to add to my host's environment
variables.
5. Turn off or skip anything in the list of things I don't want recorded, and tell
me how you did it.
6. Give me a safe way to send one test error, and tell me where to look in the
service's dashboard to confirm it arrived.
7. Remind me to remove any test error page or button once it works.
```

💡 **Tips to customize:** For Next.js, Sentry's docs say to run `npx @sentry/wizard@latest -i nextjs`. The wizard asks whether you want tracing, Session Replay, and a route that gets around ad blockers, and it can create an example page for a test error. It also stores your auth token in a file called `.env.sentry-build-plugin` and adds that file to `.gitignore`. Session Replay records a video-like view of what happened in a user's browser, so turn it off if you don't need it. Check what your Sentry plan includes before you turn on extra features.

2.2 Get an Email When Something New Breaks

Use this when error tracking is set up but you never open the dashboard. It's for choosing a few alerts that tell you about new or growing problems, without flooding your inbox.

Use in: ChatGPT or Claude, in a normal chat. Or your coding tool if you want it to check your setup.

Fill in: [MY ERROR TRACKING SERVICE] (e.g. "Sentry"), [HOW I WANT TO BE TOLD] (e.g. "email only", or "email plus a Slack channel"), [HOW MANY ALERTS A WEEK I CAN HANDLE] (e.g. "a few a week at most").

```
I've set up error tracking for my live app. Help me set up alerts so I find out
about real problems without getting buried in noise. Keep it beginner-friendly.

My error tracking service: [MY ERROR TRACKING SERVICE]
How I want to be told: [HOW I WANT TO BE TOLD]
How many alerts a week I can handle: [HOW MANY ALERTS A WEEK I CAN HANDLE]

Please:
1. Recommend no more than 3 alerts to start with. For each one, explain in one
sentence what it catches and why it matters.
2. Give me click-by-click steps for setting up each one, based on the service's
current docs. If you're not sure where a setting lives, tell me which docs page to
check instead of guessing menu names.
3. Make sure the alerts only cover my live site (production), not my own testing.
4. Tell me what to do the first time each alert fires: what to look at first, and
how to decide if it's urgent.
5. Tell me how to tune an alert if it fires too often.
```

💡 **Tips to customize:** In Sentry, alerts live under **Monitors > Alerts**, where you click **Create Alert**. Sentry's docs list triggers such as "A new issue is created," "An issue escalates," and "A resolved issue regresses" (it came back after you marked it fixed). You can filter an alert by environment so only production errors reach you. A good first setup is one alert for new issues and one for issues that come back.

2.3 Know When the Site Is Down

Use this when you want to hear that your site is down from a robot, not from a user. It's for adding a simple uptime check, plus a health page that shows whether your app can still reach its database.

Use in: Claude Code or Cursor's Agent, opened in your project.

Fill in: [MY LIVE URL] (e.g. "https://myapp.com"), [MY DATABASE] (e.g. "Supabase"), [UPTIME SERVICE] (e.g. "Sentry's Uptime Monitors", or "suggest one").

```
Help me set up an uptime check so I know right away if my live site goes down.
Explain each step in plain words.

My live URL: [MY LIVE URL]
My database: [MY DATABASE]
Uptime service: [UPTIME SERVICE]

Please:
1. Add a small health check route to my app (for example /api/health) that returns
a success response when the app can reach the database, and an error response when
it can't. Use the cheapest possible database check.
2. Make sure the health route never shows secrets, error details, version numbers,
or user data. It should only say "ok" or "not ok."
3. Make sure it can't be used to overload my database, and that it doesn't need a
login.
4. Show me the code before you save it, and tell me how to test it on my own
computer.
5. Tell me how to point the uptime service at my home page and at the health
route, and which settings to pick: how often to check, how long to wait, and how
many failures in a row before I get told.
6. Tell me how to make sure the alert reaches me, and how to test it without
taking my real site down.
```

💡 **Tips to customize:** Sentry's docs say it automatically creates an uptime monitor for the hostname that shows up most in your error data, and you can add your own from the Monitors page. You can check every 1, 5, 10, 20, or 30 minutes, or every hour. By default, Sentry opens an issue after 3 failed checks in a row, so with a 5-minute check, that's about 15 minutes of downtime. A check passes when the page returns a 2xx status code. Connect your uptime monitor to an alert (Prompt 2.2) so the issue actually reaches you.

SECTION 3

Make Errors Useful, Then Fix Them

A good error does two jobs: it tells the user what to do next, and it tells you exactly what went wrong.

3.1 Replace Silent Failures With Clear Messages

Use this when users click a button and nothing happens, see a blank page, or get a vague "something went wrong" with no way forward. It's for finding the places where errors get swallowed, and giving users a clear next step.

Use in: Claude Code or Cursor's Agent, opened in your project.

Fill in: [MY FRAMEWORK] (e.g. "Next.js App Router"), [THE MOST IMPORTANT ACTIONS IN MY APP] (e.g. "sign up, log in, save a habit, pay"), [HOW I WANT ERRORS TO SOUND] (e.g. "friendly and short, no technical words").

```
Find places in my app where errors fail silently, and fix them so users always
know what happened and what to do next. Start with a list, not changes.

My framework: [MY FRAMEWORK]
The most important actions in my app: [THE MOST IMPORTANT ACTIONS IN MY APP]
How I want errors to sound: [HOW I WANT ERRORS TO SOUND]

Please:
1. Search the code for errors that get swallowed: empty catch blocks, failed
requests with no message, form submits with no loading or error state, and pages
with no error screen. List each one with its file and what the user sees today.
2. Rank the list by how much it hurts users, starting with the important actions I
listed.
3. For each item, suggest the fix in one line:
   - Expected problems (wrong password, missing field, card declined) should show
a specific, friendly message and keep what the user typed.
   - Unexpected crashes should show an error screen with a "try again" button and
a way to get help, not a blank page.
4. The user should never see stack traces, database errors, or other technical
details. Those go in the server logs instead.
5. After I approve, fix the top 3 items only, then show me how to test each one.
```

💡 **Tips to customize:** If you use the Next.js App Router, its docs suggest handling expected errors (like a failed form check) as return values instead of throwing them, and catching unexpected ones with an `error.tsx` file in the route folder. An `error.tsx` file catches errors in that part of the app, and errors move up to the nearest one. A `global-error.tsx` file in the root `app` folder catches

errors in the root layout. For the error copy, read each message out loud. If it wouldn't make sense to a friend, rewrite it.

3.2 Log Lines Future-You Can Actually Use

Use this when your logs are either empty or full of noise, and you can't tell which line belongs to which problem. It's for adding a few clear log lines in the places that matter, without logging anything private.

Use in: Claude Code or Cursor's Agent, opened in your project.

Fill in: [THE PARTS OF MY APP THAT BREAK MOST] (e.g. "checkout and the signup email"), [WHERE MY LOGS GO] (e.g. "Vercel runtime logs").

```
Help me improve the logging on my server code so I can quickly tell what went
wrong in production. Keep it simple. I don't need a logging framework unless you
think it's really worth it.

The parts of my app that break most: [THE PARTS OF MY APP THAT BREAK MOST]
Where my logs go: [WHERE MY LOGS GO]

Please:
1. Find my server code (API routes, server actions, webhook handlers, background
jobs) and list what each one logs today.
2. Suggest one consistent format for log lines, such as a short event name plus a
few key details, so I can search for them. Show me 3 examples.
3. Add a log line where each important action fails, including: what the app was
trying to do, a user ID or request ID (not an email), and the error message. Use
console.error for real errors so my host marks them as errors.
4. Remove or reduce noisy logs that don't help.
5. Make sure nothing logs passwords, tokens, API keys, full card details, whole
request bodies, or other private data. Point out any place that does today.
6. Show me the changes before you save them, and keep them small.
```

💡 **Tips to customize:** Vercel's docs say each `console.log` line becomes its own log entry, and `console.error` lines show up at the error level, so you can filter for them. Vercel also limits logs to 256 lines per request, so a log line inside a big loop can push out the lines you actually need. Once you have error tracking (Prompt 2.1), ask the AI whether a failure should also be sent there, not only to the logs.

3.3 Fix the Live Bug Without Breaking Something Else

Use this when you've found the cause of a production bug and you're ready to fix it. It's for making the smallest safe fix, testing it on a preview before it goes live, and knowing how to undo it.

Use in: Claude Code or Cursor's Agent, opened in your project.

Fill in: [THE BUG] (e.g. "saving a profile fails when the bio is empty"), [THE CAUSE WE FOUND] (e.g. "the database column doesn't allow empty values"), [THE LOG LINE THAT PROVES IT] (paste it, with private data removed), [HOW I DEPLOY] (e.g. "push to main and Vercel deploys automatically").

```
We found the cause of a bug on my live site. Help me fix it safely. Real users are
on this site, so careful beats fast.

The bug: [THE BUG]
The cause we found: [THE CAUSE WE FOUND]
The log line that proves it: [THE LOG LINE THAT PROVES IT]
How I deploy: [HOW I DEPLOY]

Please:
1. Tell me in one or two sentences why this cause explains both the bug and the
log line. If it doesn't fully explain them, say so before we go on.
2. Propose the smallest change that fixes it. List every file you'd touch, and
don't change anything unrelated.
3. Check whether the bug could have damaged any data (half-saved records,
duplicate charges, missing rows). If it could have, tell me how to check, but
don't change any data without my OK.
4. Add a test, or a written check I can repeat, that fails before the fix and
passes after it.
5. Tell me how to try the fix on a preview deploy or locally before it goes live.
6. Tell me how to undo the fix quickly if it causes a new problem.
Show me the plan first. Make the change only after I approve it.
```

💡 **Tips to customize:** If the fix touches the database structure, ask the AI to write the change as a migration and explain whether it can be undone. If the bug is losing users money or data right now, ask for the fastest safe stopgap first (like temporarily hiding the broken button), then the real fix. Never let the AI test against your live database with real users' data.

SECTION 4

Keep It From Happening Again

Ten minutes a week and a short note after each problem will catch most issues while they're still small.

4.1 A 10-Minute Weekly Health Check

Use this when you want a simple habit that catches problems early, before they pile up. It's for building a short weekly checklist that fits your exact setup.

Use in: Claude Code or Cursor's Agent, opened in your project. A normal chat works too if you list your services.

Fill in: [MY SERVICES] (e.g. "Vercel, Supabase, Sentry, Stripe"), [WHAT MATTERS MOST IN MY APP] (e.g. "signups and payments"), [DAY AND TIME I'LL DO THIS] (e.g. "Monday morning").

```
Make me a weekly health check for my live app that takes about 10 minutes. I'm a
beginner, so make each step specific.

My services: [MY SERVICES]
What matters most in my app: [WHAT MATTERS MOST IN MY APP]
Day and time I'll do this: [DAY AND TIME I'LL DO THIS]

Please:
1. Look at my project so the checklist matches my real setup.
2. Write a checklist of 6 to 8 items. For each one, say where to look, what "fine"
looks like, and what should make me dig deeper. Cover:
   - new or growing errors in my error tracking
   - failed or slow requests in my host's logs
   - errors in my database and login logs
   - whether my most important actions still work (try them myself on the live
site)
   - failed payments or emails, if I use those services
   - whether my uptime check is still on and still alerting me
3. Point out any check that won't work because a service doesn't keep logs long
enough, and suggest what to do instead.
4. End with a one-line template I can fill in each week to keep a simple record,
like "Date / errors / anything to fix."
Keep it to one page so I'll actually use it.
```

💡 **Tips to customize:** Remember Vercel's log retention: on the Hobby plan, runtime logs only last an hour, so a weekly look at Vercel's logs won't show last week. Your error tracking is the better

weekly source. Put the check on your calendar. If you want, paste the weekly notes back into the AI once a month and ask, "What keeps coming up?"

4.2 Write Down What Went Wrong (So It Doesn't Happen Twice)

Use this when you've just fixed a real problem on your live site. It's for a short, blame-free note on what happened, why, and the one or two changes that would stop it from coming back.

Use in: ChatGPT or Claude, in a normal chat. Or your coding tool, so it can check the code and git history.

Fill in: [WHAT USERS SAW] (e.g. "signups failed for about 3 hours"), [WHAT CAUSED IT] (e.g. "an environment variable was missing after I renamed it"), [HOW WE FOUND OUT] (e.g. "a user emailed me"), [WHAT WE CHANGED TO FIX IT].

```
I just fixed a problem on my live app. Help me write a short note about it so I
learn from it and it doesn't happen again. Keep it blame-free and plain. This is
just for me.

What users saw: [WHAT USERS SAW]
What caused it: [WHAT CAUSED IT]
How we found out: [HOW WE FOUND OUT]
What we changed to fix it: [WHAT WE CHANGED TO FIX IT]

Please:
1. Write the note in 5 short parts: what happened, the impact (who was affected
and for how long, as far as I know), the cause, the fix, and what we'll change.
2. Ask me questions if any part is unclear instead of filling it in yourself. Mark
anything you're not sure about with "(not sure)".
3. For "what we'll change," suggest at most 3 actions, ranked by how much each
would help. Include at least one that would have told me sooner, like an alert, a
log line, or an uptime check.
4. Tell me if a user-facing message would help (for example, telling affected
users it's fixed), and draft one short version I can edit and send myself.
Keep the note under 25 lines.
```

💡 **Tips to customize:** Keep these notes together in one file in your project, like `INCIDENTS.md`, so you and your AI can look back at them. If the same kind of problem shows up twice, add a line about it to your rules file so the AI watches for it. Our One Rules File pack shows you how.

Bonus: Magic Phrases for Production Problems

- **"Quote the exact log line that proves it."**
- **"Don't change any code until I approve the plan."**
- **"Reproduce it first, then fix it."**
- **"Smallest fix only. Don't touch anything unrelated."**
- **"Tell the user what to do next. Put the technical details in the logs."**
- **"Never log passwords, tokens, or private data."**
- **"Check the current docs instead of guessing menu names."**

It Broke in Production

That's the whole pack: **10 prompts across 4 sections**. Find your logs, add error tracking and an uptime check, make every error useful, fix carefully, and write down what you learned.

Happy building!