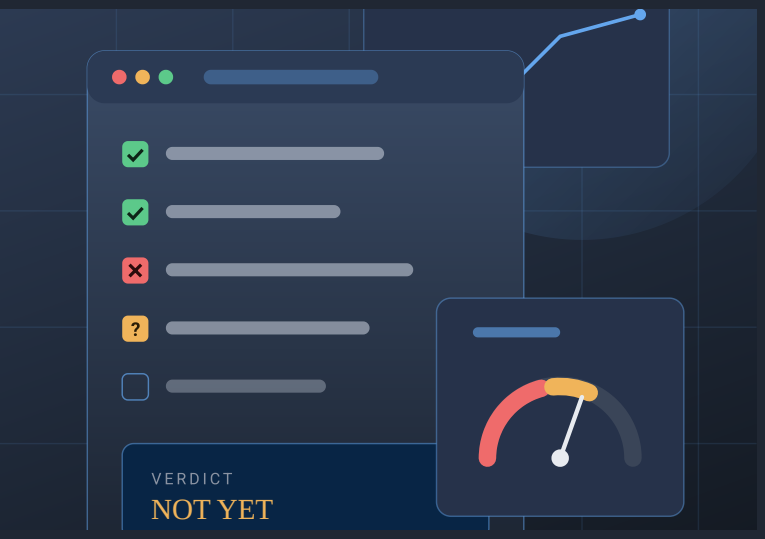

A VIBE CODER'S GUIDE TO PRODUCTION · FREE CHECKLIST

The 60-Minute Deploy Checklist for AI-Built Apps

Your app works in the demo, but that's not the same as **safe to let real users in**.

Most platforms now check your code to a degree when you publish, but even Lovable's own docs say its scans don't replace a thorough security review. A scanner can't log in as two people and try to read each other's data. It can't tell you what happens when Stripe goes down.

This checklist helps you go many steps further. **15 checks in about 60 minutes with one decision made for you at the end: can you deploy now or not.**

15	~60	7
checks	minutes	AI builders covered

Your Platform may Check the Code, but this Checks the App.

Most AI-built apps fail in the same few places. Published scans of live vibe-coded apps keep finding the same three things:

Open data

- Tables with no access rules
- User B can read User A's records
- Private files in public buckets

Leaked keys and money

- Secret keys shipped to the browser
- Payments that grant nothing, or grant everything
- No cap on AI or API spend

Silent failure

- Fixes that break something nearby
- No alert when it goes down
- Backups nobody has restored

A prompt result is a lead, not proof. The AI that wrote your code will often tell you it's fine. Mark a check as Pass only when the hands-on test passes.

How to Use This Checklist

Work top to bottom. Six checks are marked Blocks launch: if you only have 20 minutes, do those.

1

Pick your AI builder

Every prompt below rewrites itself for the tool you use.

2

Test it by hand

Each check has a short hands-on test. That's your evidence.

3

Run the prompt

Copy it into your AI tool. It reviews the code without changing anything.

4

Mark it honestly

Pass, Fail or Not sure. "Not sure" counts against deployment, for safety.

Set Up Your Prompts

Your choices save in this browser only. Nothing is sent anywhere.

WHAT DID YOU BUILD IT WITH?

Using the prompts from this PDF. Each prompt below is the core check. Before pasting it, add one line telling your tool not to edit anything (Lovable: use Chat mode; Cursor: Ask mode; Claude Code: plan mode). After it, paste these rules:

```
Rules for your answer:
- Do not change any code, settings or data. This is a review only.
- Report as a table: Finding | Where (file, table or route) | Evidence | Severity (Blocks launch / Fix soon / Can wait) | Smallest safe fix.
- If you can't confirm something from what you can see, write "Not verified" and tell me how to check it by hand. Don't guess.
- If you find nothing, say so plainly and list exactly what you checked.
```

The interactive version adds these for you, tailored to your builder, and works out your verdict as you go.

Section A

Data and Access

~20 min

Who can see what. This is where most published vibe-coding breaches start.

01

Database access rules on every table

BLOCKS LAUNCH

Does every table holding user data have row-level security (RLS) turned on, with policies that check who owns the row?

WHY IT MATTERS

Without access rules, anyone holding your public API key can read the table straight from the browser. A scan of 30,998 live vibe-coded apps found 57% of reachable Supabase-backed apps allowed unauthenticated table reads. CVE-2025-48757 (May 2025) found 170 of 1,645 Lovable apps with inadequate RLS.

Source: Reeve scan, Aug 2026 (vendor data) · LockShip

CHECK IT YOURSELF

1. Supabase: open the Table Editor. Any table flagged "RLS disabled" or "Unrestricted" is a fail.
2. Open the Policies page (under Authentication or Database, depending on your dashboard version). A policy whose condition is just `true` lets everyone in.
3. Not on Supabase? Find where your API checks the logged-in user before each query. No check means fail.

Pass when: Every table with user data has RLS on, and every policy ties rows to `auth.uid()` or a role checked on the server.

Prompt for any AI tool

```
Check 1: database access rules.

List every table in my database. For each table, tell me:
1. Is row-level security (RLS) enabled?
2. What policies exist for SELECT, INSERT, UPDATE and DELETE, and what does each one allow?
3. Does each policy limit rows to their owner (for example user_id = auth.uid()) or to a role
checked on the server?

Flag as Blocks launch:
- Any table holding user or business data with RLS disabled.
- Any policy whose condition is just true, or that lets the anon role read private data.
- Any UPDATE policy that lets a user change the owner column (user_id) or their own role.
- Any view or database function that bypasses RLS (security definer) and is callable from the
browser.

For each issue, write the SQL that would fix it, but do not run it.
```

Result: ○ Pass ○ Fail ○ Not sure

Can't answer this confidently? That's what the Launch Check covers.

02

Two-account test: users can't see each other's data

BLOCKS LAUNCH

Logged in as User B, can you read, edit or delete anything that belongs to User A?

WHY IT MATTERS

This is called IDOR (insecure direct object reference): change an ID in a URL or request, get someone else's data. Scanners read code; they don't log in as two people. A Lovable-hosted EdTech app exposed 18,697 user records in Feb 2026.

Source: The Register, via Hacker News

CHECK IT YOURSELF

1. Create two accounts. Use a normal window for User A and a private window for User B.
2. As User A, create something (a record, a document, an order). Copy its URL or ID.
3. As User B, open that URL. Then in DevTools → Network, right-click a request that loads data → Copy as fetch. Paste it in the Console, swap in User A's ID, press Enter.
4. Repeat for edit and delete.

Pass when: User B gets a 403, a 404 or an empty result every time.

Prompt for any AI tool

```
Check 2: two-account access test (IDOR).

Find every page, API route, server function and database query that loads, updates or deletes data
using an ID that comes from the browser (URL path, query string, request body, form field).

For each one:
1. Show where the server confirms the logged-in user owns that record, or is allowed to see it.
2. Flag any place where the server trusts a user ID, team ID or account ID sent from the browser
instead of reading it from the session.
3. Write one manual test step I can run as User B to try to read, edit or delete User A's record
through it.

Any missing ownership check is Blocks launch.
```

Result: ○ Pass ○ Fail ○ Not sure

Can't answer this confidently? That's what the Launch Check covers.

03

Admin and paid features are checked on the server

BLOCKS LAUNCH

Is every admin page, admin action and paid feature checked on the server, not just hidden in the interface?

WHY IT MATTERS

Hiding a button doesn't protect the action behind it. One developer who reviewed 80 vibe-coded repos summed up the pattern: "no auth on admin routes, API keys hardcoded in client JS".

Source: Hacker News

CHECK IT YOURSELF

1. Logged in as a normal user, type the admin URL directly (for example /admin).
2. DevTools → Application → Local Storage. Look for values like role or plan. Change `user` to `admin` or `free` to `pro`, then reload.
3. Copy an admin API request as fetch from an admin session. Run it in the Console while logged in as a normal user.

Pass when: Every attempt is refused by the server, not just redirected by the page.

Prompt for any AI tool

```
Check 3: roles and permissions.

Find every admin, staff or privileged page, API route, server function and edge function, plus
every feature limited to paying users.

For each one, show the exact line where the role or plan is checked on the server.

Flag as Blocks launch:
- Checks that only happen in the frontend (hidden buttons, route guards, conditional rendering).
- Roles or plans read from anything the user can change: localStorage, cookies they can edit, a
profile column they're allowed to update, or request parameters.
- Admin actions callable with only the public anon key.
```

Result: ○ Pass ○ Fail ○ Not sure

Can't answer this confidently? That's what the Launch Check covers.

04

Storage buckets and CORS are locked down

BLOCKS LAUNCH

Are uploaded files private unless they're meant to be public, and do your APIs only accept credentialed requests from your own domains?

WHY IT MATTERS

A public bucket means anyone with a link (or a guessable path) can download every file in it: IDs, invoices, photos. CORS is the browser rule for which websites may call your API. Allowing any origin with cookies switched on lets other sites make requests as your logged-in user.

CHECK IT YOURSELF

1. Supabase: Storage → open each bucket. "Public" means anyone with the URL can download every file in it.
2. As User A, upload a private file. Copy its URL. Open it in a private window while logged out.
3. DevTools → Network → click an API request → Response Headers. Note `Access-Control-Allow-Origin` and `Access-Control-Allow-Credentials`.

Pass when: Private files need a login or a short-lived signed URL. No endpoint echoes back any origin while allowing credentials.

Prompt for any AI tool

```
Check 4: file storage and CORS.

Part 1, storage. List every storage bucket, whether it's public or private, and the policies on it.
Flag:
- Private user files (ID documents, invoices, uploads, exports) in public buckets.
- File paths that are guessable (sequential numbers, user emails, plain user IDs).
- Upload rules that let a user overwrite or delete another user's files.
- No limit on file type or size for uploads.

Part 2, CORS. For every API route, server function and edge function, show the CORS settings. Flag:
- Any endpoint that reflects the request's Origin header back, or allows any origin, while also
allowing credentials (cookies).
- Wildcard origins on endpoints that return user data.
Tell me which origins should be allowed instead.
```

Result: ○ Pass ○ Fail ○ Not sure

Can't answer this confidently? That's what the Launch Check covers.

Section B

Secrets and Money

~15 min

Keys, payments and bills. The failures that cost you and your clients directly.

05

No secret keys in the browser or git history

BLOCKS LAUNCH

Are server-only keys (database service role, Stripe secret key, AI provider keys) kept out of the frontend and out of your git history?

WHY IT MATTERS

Anything in the frontend bundle is public, minified or not. Escape.tech found 400+ leaked secrets across about 1,400 production vibe-coded apps (Oct 2025). A key leaked from METR's vibe-coded dashboard was used for about $600,000 of model credits.

Source: Cloud Security Alliance, citing Escape.tech · Runtime Wire

CHECK IT YOURSELF

1. Open your live site. DevTools → Sources → Ctrl+Shift+F. Search for `sk_live`, `sk_test`, `service_role`, `sk-`, `secret`.
2. Search git history from the project folder: `git log -p -S "sk_live"` (repeat for each term). Lovable and Bolt users: connect GitHub first.
3. Check variable names. Anything starting `VITE_`, `NEXT_PUBLIC_` or `REACT_APP_` is shipped to the browser.
4. Note: the Supabase anon key and Stripe publishable key are meant to be public. They're only safe if Check 1 passes.

Pass when: No secret key in the bundle or history. Any key that was ever exposed has been rotated in the provider's dashboard, not just deleted from code.

Prompt for any AI tool

```
Check 5: exposed secrets.

Search the frontend code, environment files, build config, server functions and git history for
secrets. Look for:
- Supabase service_role key
- Stripe secret or restricted keys (sk_live_, sk_test_, rk_)
- OpenAI, Anthropic or other AI provider keys
- Database connection strings with passwords
- Webhook signing secrets
- Any environment variable prefixed VITE_, NEXT_PUBLIC_ or REACT_APP_ that holds a secret

For each one: where is it, does it ship to the browser, and was it ever committed to git?

Do not flag the Supabase anon key or Stripe publishable key: they're designed to be public. Do tell
me if RLS is the only thing protecting data behind the anon key.
```

```
For anything exposed, list the steps to rotate it and move it server-side. Do not print full secret
values in your answer: show the first 6 characters only.
```

Result: ○ Pass ○ Fail ○ Not sure

Can't answer this confidently? That's what the Launch Check covers.

06

Spending caps on every paid API

FIX SOON

If someone scripts 10,000 requests to your AI, email or SMS features tonight, is there a cap that stops the bill?

WHY IT MATTERS

Leaked keys and unlimited endpoints turn into invoices. One builder lost $300 to a key leak, and the provider's abuse alerts went to an inbox they didn't check. Another found £2,000 of charges on a key someone had already used.

Source: Hacker News · Utku Şen

CHECK IT YOURSELF

1. List every paid service your app calls: AI model, email, SMS, maps, storage, search.
2. In each provider's billing settings, set a hard monthly limit or, where that isn't offered, a budget alert.
3. Make sure billing and abuse emails go to an inbox you actually read.

Pass when: Every paid provider has a cap or an alert you'll see, and anonymous users can't trigger paid calls without a limit.

Prompt for any AI tool

```
Check 6: cost exposure.

List every paid third-party service my app calls (AI models, email, SMS, maps, storage, search,
anything billed per use).

For each one, tell me:
1. Which endpoint or function calls it.
2. Can a logged-out user trigger it? Can one logged-in user trigger it in a loop?
3. Is there a per-user or per-IP limit, a max token or size limit, or a daily cap in the code?
4. Worst-case cost if someone scripts 10,000 requests, using the provider's current public pricing.
If you don't know the price, say "Not verified" and tell me where to look.

Then tell me where to set a hard spending limit or budget alert in each provider's dashboard. If
you're not sure the menu exists or where it is, say so.
```

Result: ○ Pass ○ Fail ○ Not sure

Can't answer this confidently? That's what the Launch Check covers.

07

Payments grant access through a verified webhook

BLOCKS LAUNCH

Is paid access granted by a signature-verified Stripe webhook, and not by the success page someone lands on after checkout?

WHY IT MATTERS

If the success page grants access, anyone who finds the URL gets the paid tier free. If the webhook isn't verified, anyone can send a fake "paid" event. If it fails quietly, customers pay and get nothing, then ask for a chargeback.

Source: FetchSandbox

CHECK IT YOURSELF

1. Log in as a free user. Visit your checkout success URL directly, without paying. Did your plan change?
2. Stripe Dashboard → Webhooks (under Developers or Workbench). Open your endpoint. Any failed deliveries?
3. In test mode, with the Stripe CLI: `stripe trigger checkout.session.completed`. Confirm access is granted exactly once. Then cancel the test subscription and confirm access is removed.

Pass when: Access changes only after a signature-verified webhook. Duplicate events don't double-grant. Failed payments, refunds and cancellations remove access.

Prompt for any AI tool

```
Check 7: payments and webhooks.

Review my payment flow end to end and answer each question with the file and line that proves it:
1. Is paid access granted by the Stripe webhook, or by the checkout success page or redirect (which
anyone can visit)?
2. Is the webhook signature verified with the signing secret, using the raw request body?
3. Is processing idempotent? What happens if Stripe sends the same event twice?
4. What handles invoice.payment_failed, charge.refunded, customer.subscription.updated and
customer.subscription.deleted?
5. Is the price or plan taken from Stripe's data, or from something the browser sent?
6. If my webhook endpoint is down for an hour, what happens to payments made in that hour?

Then write a test-mode plan using the Stripe CLI (stripe listen, stripe trigger, stripe events
resend) that proves each answer.
```

Result: ○ Pass ○ Fail ○ Not sure

Can't answer this confidently? That's what the Launch Check covers.

08

Rate limits on login, sign-up and public forms

FIX SOON

Is there a server-side limit on login, sign-up, password reset, AI calls and public forms?

WHY IT MATTERS

Without a limit, password guessing, sign-up spam and cost attacks are free to run, all day. A limit set only in the browser doesn't count: scripts skip the browser.

CHECK IT YOURSELF

1. Try logging in with the wrong password 20 times in a minute. Does anything slow you down or block you?
2. Submit your contact or waitlist form 10 times quickly. Do 10 emails arrive?
3. Supabase: check the Rate Limits settings under Authentication.

Pass when: Repeated attempts get slowed or blocked by the server, and public forms have a CAPTCHA or similar check.

Prompt for any AI tool

```
Check 8: rate limiting.

Find every endpoint that is sensitive or costs money to run: login, sign-up, password reset, magic links, OTP codes, contact and waitlist forms, AI calls, file uploads, anything that sends email or SMS.

For each one, tell me:
1. Is there a rate limit? Is it enforced on the server, per user and per IP?
2. Is there a CAPTCHA or similar on public forms?
3. What error does the user see when they hit the limit?

Suggest the smallest limit for each that would stop a script without blocking a real user, and where in my stack to set it.
```

Result: ○ Pass ○ Fail ○ Not sure

Can't answer this confidently? That's what the Launch Check covers.

Section C

Behaviour and Failure

~15 min

What the app does when users, inputs or third parties don't behave. Scanners can't run these.

09

The server checks every input

FIX SOON

Does the server validate every input, and ignore fields the user shouldn't be able to set?

WHY IT MATTERS

Browser checks are for convenience. Anyone can skip them with DevTools. If your API accepts a role, price or user_id from the request, a user can set their own.

CHECK IT YOURSELF

1. DevTools → Network. Save something (for example your profile). Right-click that request → Copy as fetch.
2. Paste it into the Console. Add `"role":"admin"`, or change a price or user_id. Press Enter, then reload. Did it stick?
3. Type `<b>test</b>` into a text field and save. If it shows in bold anywhere, user text is being rendered as HTML.

Pass when: Extra or tampered fields are ignored or rejected by the server. User text always displays as plain text.

Prompt for any AI tool

```
Check 9: server-side input validation.

For every form and API route that creates or updates data, show where input is validated on the
server.

Flag:
- Anything validated only in the browser.
- Mass assignment: the server saves the whole request body, so a user could set fields like role,
is_admin, plan, price, credits, user_id or owner_id.
- SQL or database queries built by joining strings with user input.
- User content rendered as HTML without escaping (dangerouslySetInnerHTML, v-html, innerHTML,
markdown rendered without sanitising).
- No length or type limits on text fields and uploads.

For each issue, give the smallest fix: an allow-list of fields, a schema check (for example Zod),
or escaping.
```

Result: ○ Pass ○ Fail ○ Not sure

Can't answer this confidently? That's what the Launch Check covers.

10

Failures show a clear message, not a crash

FIX SOON

When Stripe, your AI model or your database is slow or down, does the user see a clear message, with nothing half-saved?

WHY IT MATTERS

AI-built apps are usually tested on the happy path only. The first outage shows a blank screen or a stack trace, or takes a payment with no order saved.

CHECK IT YOURSELF

1. DevTools → Network → change the throttling dropdown to Offline. Try your core action. What does the user see?
2. In a development copy, set your AI provider key to something wrong. Try the AI feature.
3. Look for raw error text, JSON or stack traces on screen.

Pass when: Every failure shows a plain message and a way to retry. No stack traces. Nothing left half-written.

Prompt for any AI tool

```
Check 10: failure handling.

For every call to something outside my app's code (database, Stripe, AI model, email, storage, any
third-party API), describe exactly what the user sees when it fails or times out.

Flag:
- Empty catch blocks, or errors that are logged and then ignored.
- Unhandled promise rejections and missing await.
- No timeout on calls to third parties.
- Raw error messages, stack traces or database errors shown to users.
- Operations that can half-complete: for example, the payment succeeds but the order isn't saved,
or the record is saved but the email never sends, with no retry.
- Buttons that can be double-clicked to submit twice.

For each one, suggest the simplest handling: a user-facing message, a retry, or a transaction.
```

Result: ○ Pass ○ Fail ○ Not sure

Can't answer this confidently? That's what the Launch Check covers.

11

Critical journeys retested after the last AI change

FIX SOON

Since the last AI edit, have you re-run sign-up, login, password reset, payment and your core action from start to finish?

WHY IT MATTERS

AI fixes often break something nearby. In Stack Overflow's 2025 developer survey, 66% named "AI solutions that are almost right, but not quite" as their top frustration. Almost right is how a working checkout breaks on launch day.

CHECK IT YOURSELF

1. Open a private window. Sign up with a new email address.
2. Log out. Log in. Reset the password using the email link.
3. Pay in test mode. Do the one thing users pay for. Cancel the plan.

Pass when: All five journeys work in a clean browser, after your most recent change.

Prompt for any AI tool

```
Check 11: critical journeys.

1. List the five journeys that matter most in my app: sign-up, login, password reset, payment, and
the core action users pay for. Use my actual routes and screens.
2. For each one, write a step-by-step manual test I can run in a private browser window in under 3
minutes, with the expected result after each step.
3. Look at the most recent changes to the code (git log, or the files changed most recently) and
tell me which journeys they touch, so I know what to retest first.
4. Optional: outline Playwright tests for these five journeys, but don't create the files.
```

Result: ○ Pass ○ Fail ○ Not sure

Can't answer this confidently? That's what the Launch Check covers.

12

Every dependency is real and patched

CAN WAIT

Does every package your app installs actually exist, come from the publisher you expect, and have no critical known vulnerabilities?

WHY IT MATTERS

AI tools sometimes invent package names. Attackers register those names and fill them with malicious code. Old packages carry known holes that bots scan for.

CHECK IT YOURSELF

1. In the project folder, run `npm audit --omit=dev`. Lovable and Bolt users: export to GitHub and run it locally, or check GitHub's Dependabot alerts.
2. For any package name you don't recognise, look it up on npmjs.com: publisher, weekly downloads, last publish date.

Pass when: No critical or high findings in production dependencies, and every package checks out on npmjs.com.

Prompt for any AI tool

```
Check 12: dependencies.

List every dependency in package.json (or requirements.txt or equivalent).

Flag any package that:
- You don't recognise as a well-known library, or whose name is one letter off a popular package.
- Isn't imported anywhere in the code (unused).
- Duplicates what another installed package already does.

You can't check download counts or publish dates from here, so mark those "Not verified" and list
the packages I should look up on npmjs.com myself.

Then tell me the command to run a vulnerability audit for my package manager, and how to read its
output.
```

Result: ○ Pass ○ Fail ○ Not sure

Can't answer this confidently? That's what the Launch Check covers.

Section D

Operate and Recover

~10 min

Finding out first, and getting back to a working state fast.

13

Errors and downtime reach you first

CAN WAIT

If your app breaks at 2am, do you find out before your users tell you?

WHY IT MATTERS

Monitoring doesn't prevent a failure. It decides whether the problem lasts 10 minutes or 3 days. Logs need care too: they mustn't store passwords, tokens or full personal data.

CHECK IT YOURSELF

1. Set up a free uptime checker on your home page and one API route.
2. Add error tracking to the frontend and the backend.
3. Trigger a test error and take the site down briefly. Confirm both alerts reach a phone or inbox you check.

Pass when: A test error and a test outage both alert you within minutes.

Prompt for any AI tool

```
Check 13: monitoring and logging.

1. What happens today if my app throws an error in production or goes down? Find any error
tracking, logging or uptime monitoring already in the code.
2. Check that logs never record passwords, tokens, API keys, full card numbers or more personal
data than needed.
3. Give me the minimum setup for my stack: error tracking for frontend and backend, an uptime check
on the home page and one API route, and billing alerts. Name the files you'd change, but don't
change them.
```

Result: ○ Pass ○ Fail ○ Not sure

Can't answer this confidently? That's what the Launch Check covers.

14

Backups exist and a restore works

FIX SOON

Is your database backed up, and have you restored a backup at least once?

WHY IT MATTERS

A backup you've never restored is a guess. In July 2025 Replit's agent deleted SaaStr's production database. If personal data leaks, UK GDPR says you must report it to the ICO within 72 hours when it risks people's rights. You can't do that without knowing what you hold and where.

Source: The Register · ICO

CHECK IT YOURSELF

1. Open your database provider's backup settings. Note frequency, retention and whether point-in-time recovery is on (on Supabase it depends on your plan).
2. Restore the latest backup into a separate project, never production. Time it.
3. Write down where personal data lives: tables, storage buckets, email tool, analytics.

Pass when: You've restored a backup once, you know how long it takes, and you have a list of where personal data lives.

Prompt for any AI tool

```
Check 14: backups and personal data.

1. How is my database backed up: provider, frequency, retention, and is point-in-time recovery on?
If you can't see this from the code, tell me exactly what to look for in the provider's dashboard.
2. Write a restore drill I can run into a separate project (never production), with a time target
and how to confirm the data is complete.
3. List every place my app stores personal data: tables and columns, storage buckets, logs, and
third-party services (email, analytics, payments). I need this to answer a deletion request or a
breach report.
```

Result: ○ Pass ○ Fail ○ Not sure

Can't answer this confidently? That's what the Launch Check covers.

15

Separate dev and prod, with a rollback plan

FIX SOON

Are development and production on separate databases and keys, and can you roll back a bad release in under 15 minutes?

WHY IT MATTERS

If your AI agent works against production data, one bad command can wipe it. If you can't roll back, every bad deploy becomes an outage while you debug live.

CHECK IT YOURSELF

1. Compare the database URL or project ID in your development and production settings. They should differ.
2. Find your hosting provider's deploy history. Practise redeploying the previous version.
3. Look for migrations or scripts that drop, truncate or delete data.

Pass when: Development never touches production data, and you've done one practice rollback.

Prompt for any AI tool

```
Check 15: environments and rollback.

1. Does my app use separate development and production databases, storage and API keys? Show me
where each is configured.
2. Could a test, a seed script or an AI agent session touch production data? Flag any script or
migration that drops, truncates or bulk-deletes data.
3. Are database migrations reversible? Which ones aren't?
4. Write the steps to roll back to the last working version of the app, and of the database schema,
in under 15 minutes on my hosting setup. If you don't know my host's exact steps, say so.
```

Result: ○ Pass ○ Fail ○ Not sure

Can't answer this confidently? That's what the Launch Check covers.

"Not Sure" Is an Answer. It's the Expensive One.

If any check above is still "Not sure", you're launching on a guess. A Launch Check replaces the guess with evidence: I test your app the way an attacker or an unlucky Monday would, and tell you what to fix first.

STEP 1 · DIAGNOSIS

Launch Check

$397 · about £310

Report within 24 hours of confirmed access (target)

- Automated scan plus engineer triage across the five failure classes that come up most in published scans of vibe-coded apps: database access rules (Supabase RLS), secrets in the client bundle, broken auth and IDOR, public storage and open CORS, payment webhooks.
- A Fix-First report: every finding ranked blocks launch / fix soon / can wait, with its location, steps to reproduce and a fix sketch.
- 30-minute readout. One free re-check within 30 days of your fixes.
- Optional fix quote per finding within 24 hours. The report is yours.

See the Launch Check →

STEP 2 · 7 DAYS

Launch-Ready Sign-Off

From $1,597 · $1,197 founding rate, first 3 clients

$397 off for Launch Check customers

- Audit against the Sentido Launch-Ready Standard v1.
- P1 fixes delivered as pull requests: rotate exposed secrets, tighten RLS, add rate limits, lock down CORS, fix auth boundaries.
- A signed, dated engineering verdict, valid for 90 days for the audited commit.
- 30-day warranty: a P1 issue present at sign-off that I missed is re-audited and fixed free.
- 45-minute readout with you or your AI builder.

Your $397 Launch Check fee comes off the Sign-Off if you book it within 30 days of your report.

See the Launch-Ready Sign-Off →

How access works

Read-only repo access plus a staging copy. Mutual NDA. Your code is deleted after delivery. I never touch production.

What it isn't

Not a penetration test. Not a compliance certification (SOC 2, ISO 27001). Not a guarantee of future security. Not a rewrite. If your app needs rebuilding, I'll tell you and refer you on.

4 Launch Checks a month. Bring this checklist, marked up. Your "Not sure" items are where I start.

Book a Launch Check

Carl Mills, senior .NET/Azure engineer and founder of Sentido Ltd. Prices in USD. Invoiced by Sentido Ltd.

Educational checklist, not a security audit. Passing every check doesn't mean your app is secure; it means you've covered the common failures. Prompts ask your AI tool to review, not change, your code. Check its output before acting on it. UI labels in third-party dashboards change often; if a menu named here has moved, search the provider's docs for the feature name. Checklist version 1.0, November 2026. © Sentido Ltd.

Sources (several are vendors who sell scanning or audits; treat their figures as directional):

1. Reeve scan of 30,998 apps, Aug 2026 (vibe-eval.com)
2. CVE-2025-48757 explained (LockShip)
3. Lovable-hosted app exposed 18K users (Hacker News)
4. The "Vibe Coding" Wall of Shame (Hacker News)
5. AI-generated CVE surge, citing Escape.tech (Cloud Security Alliance)
6. METR API key and $600,000 in credits (Runtime Wire)
7. Lost $300 due to an API key leak (Hacker News)
8. Your vibe-coded AI app can bankrupt you (Utku Şen)
9. Stripe charged them. The Lovable app still said free. (FetchSandbox)
10. Replit deleted a user's production database (The Register)
11. How Lovable protects your apps automatically (Lovable)
12. Report a personal data breach (ICO)