FREE CHECKLIST · FOR APPS BUILT WITH CLAUDE CODE AND OTHER AI TOOLS

Safe to launch?

Five checks to run before real customers trust your app with their money or their data. Each one comes with the exact prompt to give Claude Code and a test you can do yourself in two minutes.

IT WORKS ✓ ≠ SAFE TO LAUNCH?

p.2	The one-page launch checklist: print this one	ALL 5 CHECKS
p.3	Accounts · Other people's data	CHECKS 1–2
p.4	Secrets · Payments	CHECKS 3–4
p.5	Tests that can fail · 3 more questions worth asking	CHECK 5
p.6	Can you judge the answers? Self-check + hiring questions	SELF-CHECK

How to use this: print page 2. Work through one check at a time: paste the prompt into Claude Code, read what it reports, then do the two-minute test yourself. Don't tick a box because Claude said so. Tick it because you saw it. Anything in `[brackets]` is yours to fill in.

The one-page launch checklist

Your app passed "does it work when everyone behaves?" These boxes are for "what happens when someone doesn't?"

1 Accounts

WHO CAN GET IN?

- [ ] Asked for two password resets: the **first link no longer works**
- [ ] Reset links **expire** (I know after how long: ____ minutes)
- [ ] Reset page shows the **same message** for a real email and a fake one
- [ ] A new account **can't be used** until its email is verified

2 Other people's data

CAN A SEE B?

- [ ] Two test accounts: A **can't open** B's pages, orders or files, even by editing the URL
- [ ] Claude listed every page, API route and table with customer data, and the rule protecting each
- [ ] Supabase users: **Row Level Security is on** for every table, with policies
- [ ] There's a test where A tries to read B's records, and it **fails**

3 Secrets

KEYS IN THE OPEN?

- [ ] No secret key is sent to the browser (searched the code **and** the Git history)
- [ ] Secrets live in server-side environment variables; `.env` is in `.gitignore`
- [ ] Every key that was **ever** exposed has been rotated, not just moved

4 Payments

WHEN IT GETS MESSY

- [ ] Every webhook's **Stripe signature is verified**
- [ ] Each event ID is processed **once**; events can arrive in any order
- [ ] Tested with test cards: double-click on Pay · declined card · refund. I saw what happened to the order each time

5 Tests that can fail

EVERY BUG FIX

- [ ] I wrote the checks in plain English **before** the feature
- [ ] For each bug: I saw the test **fail first**, then pass after the fix
- [ ] Broke checkout on purpose: a test **caught it**

CHECKS 1–2

Accounts. Other people's data.

You tested your login as you. Attackers don't log in as you. These two checks look at your app the way a stranger and a curious customer would.

1 Who can get into an account?

PROMPT · ACCOUNTS

```
Review our sign-up, login and password reset against the OWASP Forgot Password Cheat Sheet. Check that reset links are random, single use and expire. Check that the reset page shows the same message whether or not the email exists. Check that new accounts must verify their email. List what fails, fix it, and show me a test for each.
```

Your 2-minute test:

- [ ] Request a reset twice. Click the **first** link. — SHOULD: NOT WORK
- [ ] Type an email that isn't a customer on the reset page. — SHOULD: SAME MESSAGE
- [ ] Sign up with an address you never open, then try to log in and use the app. — SHOULD: BE BLOCKED

Why the message matters: "No account found" tells an attacker which emails are real customers. A good reset page says something like "If that email has an account, we've sent a link."

2 Can customer A see customer B?

The URL test. Log in as a customer and open an order. If the address bar says something like `/orders/1042`, change it to `1043`. If someone else's order loads, every customer can read every other customer's data. Security people call this **broken access control**, number one on the OWASP Top 10.

- [ ] Account A opens B's order, page and file URLs — SHOULD: DENIED
- [ ] Logged out, open any customer URL — SHOULD: LOGIN

PROMPT · CUSTOMER DATA

```
List every page, API route and database table that holds customer data. For each one, show me the rule that stops a logged-in customer from reading or changing another customer's data. If we use Supabase, confirm Row Level Security is on for every table, with policies. Then write a test that logs in as customer A and tries to read customer B's records. It must fail.
```

CHECKS 3–4

Secrets. Payments.

Keys that end up in the browser, and payment messages that arrive twice or not from Stripe at all. Both look fine in a demo and both cost real money after launch.

3 Are your keys in the open?

PROMPT · SECRETS

```
Search the whole project, including the Git history, for API keys, secret keys and passwords. Tell me which ones are sent to the browser. Move every secret to server-side environment variables, make sure the .env file is in .gitignore, and list every key I should rotate.
```

Browser is OK	Keys meant to be public, e.g. Stripe "publishable" (pk_) or Supabase "anon/publishable" keys, *with* access rules in place
Server only	Stripe secret (sk_), Supabase service role / secret, email and AI API keys, database passwords

Rotate, don't just move. If a key was ever in browser code or a Git commit, assume someone copied it. Create a new key in that service's dashboard, update your environment variables, then delete the old key.

4 What happens when payment gets messy?

When a payment goes through, Stripe sends your app a message called a **webhook**. Stripe's docs warn the same message can arrive more than once and not always in order. And if your app doesn't check the signature, anyone can send a fake "paid".

PROMPT · PAYMENTS

```
Check our payment flow against Stripe's webhook best practices. Verify the Stripe signature on every webhook. Record each event ID so the same event is never processed twice. Don't rely on events arriving in order. Then test, using Stripe's test cards: a double-click on Pay, a declined card, and a refund. Show me what happened to the order each time.
```

Your test (Stripe test mode):

- [ ] Double-click Pay with 4242 4242 4242 4242 — **1 ORDER**
- [ ] Pay with 4000 0000 0000 0002 — **NO ORDER**
- [ ] Refund a test order in Stripe — **ORDER UPDATES**

Use any future expiry date and any 3-digit CVC. Run these in Stripe's **test mode** only, never with a live key.

Ask Claude to show you the order record after each test, not just the payment screen. The payment can look fine while the order is wrong.

CHECK 5

Tests that can fail

"All 24 tests pass" sounds great. But the same AI wrote the code and the tests. A test can check what the code *does* instead of what it *should* do, or fake the very part it's meant to check. Green ticks only mean something if a test can turn red.

5a Write the checks yourself, before the feature

```
Feature: [name]
Check 1: When [situation], then [result]
Check 2: When ______, then ______
Check 3: When ______, then ______

Example: discount codes
Code applied twice → discount counted once
Discount bigger than the cart → total is $0, never negative
Expired code → clear message, full price
```

PROMPT · NEW FEATURE

Here are my checks for this feature, in plain English: [paste checks]. Turn each one into a test. Show me each test failing before you build the feature, then passing after.

This is habit 4 from episode 1: if you can't verify it, it isn't done.

5b Make every test prove it can fail

PROMPT · EVERY BUG

Before you fix this bug, write a test that fails because of it, and show me it failing. Then fix the bug and show me the same test passing.

PROMPT · ONCE IN A WHILE

Break the checkout logic on purpose and tell me which test catches it. Then undo the change.

If no test catches it, you've found a gap before your customers did. Ask Claude to add the missing test, then run the same break again.

3 more questions worth asking before launch

Backups	"If the database were deleted tonight, what would we lose and how would we restore it? Show me."
Login limits	"What stops someone trying thousands of passwords on one account?"
Error alerts	"If checkout starts failing at 2 a.m., how would I find out?"

SELF-CHECK

Can you judge the answers?

Most of these checks are an afternoon's work, and for a lot of apps that's all you need. But the prompts help you ask the right questions. They don't make you able to judge the answers. Be honest with these four.

1	**Are real customers about to trust your app with their money or personal data?**	☐ YES	☐ NO
2	**Did any of the five checks fail, and you're not sure the fix is right?**	☐ YES	☐ NO
3	**If Claude said "access control looks good", would you have to take its word for it?**	☐ YES	☐ NO
4	**Would a leak or a payment bug seriously hurt your business or your customers?**	☐ YES	☐ NO

0–1 yes: you're in good shape
Run the checklist, keep the tests, and repeat checks 2–5 after every big change.

2+ yes: get a second pair of eyes
That's not a failure. It's the point where an experienced delivery team earns its fee: a review before launch costs far less than a leak after it.

If you bring in help, ask these first

- **Have you launched apps that handle real payments and personal data?** Ask for one example.
- **Will you review what I have before quoting a rebuild?** Most AI-built apps need fixes, not a restart.
- **How will you show me it's fixed?** You want tests and evidence, not "trust us".
- **Who owns the code, the keys and the accounts?** The answer should be you, always.

WANT A PRE-LAUNCH REVIEW?

I'm an AI project manager who has shipped software for US clients for years. Book a free session and we'll tell you plainly which of these five checks your app passes.

Book a free session →

calendar.app.google/UJVDaVoiGHBNNRGB8