CODE WITH ANTHONY | FREE CHECKLIST

Production Readiness Checklist for AI-Built Apps

Your AI built the app. This checklist tells you, in plain English, whether it is actually ready for real users, real data, and real money - and what to fix first if it is not.

Use my tokens to get it set up right, not yours.

By **Anthony James**, senior software engineer, 15+ years of experience **codewithanthony.com**

Works on your laptop is not the same as ready

Most AI-built apps do not fail because the idea was bad. They fail at the same predictable points: a secret key left in the code, a database anyone can read, no backup, no way to undo a bad change, and a deploy that only works on one machine. Every item below is one of those points, written so you can check it without being an engineer.

How to use this checklist (about 20 minutes)

- Go section by section. For each item, mark **Yes**, **Not yet**, or **Not sure**.
- Ask your AI coding tool to help you verify items. Paste the item in and ask: "Show me the evidence in this codebase."
- Treat **Not sure** as **Not yet**. If you cannot prove it, it is not done.
- Fix anything marked STOP-SHIP before you invite real users.

What you will need open

- Your app's code folder and your AI coding tool
- Your hosting, database, and domain accounts
- Your payment account, if you take payments
- A notebook for the fix list you will build

There are **35 checks in 7 sections**, plus a copy-and-paste agent-instructions template on page 5 that will make every future AI session safer.

How to read your score

Your result	What it means
30-35 Yes	**Ready to launch carefully.** Ship to a small group first, watch errors for a week, then open up.
20-29 Yes	**Close, with real gaps.** Fix the stop-ship items and any section where you scored under half, then re-check.
Under 20 Yes	**Not ready for real users yet.** This is normal for an AI-built first version. Work the priority order on page 6, or get an engineer to do this pass with you.

A note on honesty: This checklist catches the failures that take most AI-built apps down. It is not a full security audit, a legal review, or a guarantee. If your app handles health, financial, or children's data, or takes significant payments, get a professional review before launch.

1 Secrets and security basics

The fastest way an AI-built app gets compromised is a key left where anyone can find it.

- [ ] **No secret keys live in your code.** STOP-SHIP
  API keys, database passwords, and payment keys are stored as environment variables, not typed into code, prompts, or screenshots. **Red flag:** search your code for "sk-", "api_key", and "password". If a real key appears, it is exposed.
- [ ] **Any key that was ever shared has been replaced.**
  If a key was pasted into a chat, committed to GitHub, or shown in a video, treat it as public. Create a new one and delete the old one in the provider's dashboard.
- [ ] **Your code repository is private, or contains no secrets.**
  Check the visibility setting on GitHub, GitLab, or wherever your code lives. A public repo plus one leaked key is a complete breach.
- [ ] **Every account has a strong, unique password and two-factor authentication.**
  That means hosting, database, domain, email, payment, and AI accounts. Your domain account is the crown jewel: whoever controls it controls your whole business.
- [ ] **Your app uses HTTPS everywhere.**
  The padlock shows on every page, and http:// automatically redirects to https://. Most modern hosts do this free. Never take logins or payments without it.

2 Accounts, sign-in, and data

If permissions are loose, any user can see or change another user's data. AI tools get this wrong often.

- [ ] **Users can only see and edit their own data.** STOP-SHIP
  Test it: create two test accounts, add data in the first, then try to view, edit, or delete it from the second, including by changing the ID number in the web address.
- [ ] **Your database is not open to the public internet.**
  Database security rules or access controls are set to deny by default, with specific allowances, not "allow all". Ask your AI to show you the rules and explain each line.
- [ ] **Passwords are stored hashed, never in plain text.**
  If you can read a user's actual password anywhere in your database, stop and fix this before launch. Use your platform's built-in sign-in instead of hand-rolled password code where you can.
- [ ] **Admin access is separate and limited.**
  There is a clear way to tell an admin from a normal user, it is checked on the server (not just by hiding a button), and only your account has it.
- [ ] **Forms check input on the server, not just in the browser.**
  Required fields, file types and sizes, and text lengths are enforced where users cannot bypass them. Browser-only checks are decoration, not protection.

Stuck on an item? That is useful information, not a failure. Write down exactly which item and why. That list is what an engineer, or a well-prompted AI session with the template on page 5, should work from.

3 Code health and AI maintainability

This is why fixes break other features: the codebase has outgrown what your AI can hold in mind.

- [ ] **Your project has an agent-instructions file, and your AI follows it.**
  A file such as AGENTS.md, CLAUDE.md, or .cursorrules at the top of the project. No file yet? Start with the template on page 5 today.
- [ ] **There is one way to do each common thing.**
  One way to call your database, one way to show errors, one way to style a button. **Red flag:** the same feature built two or three different ways in different files.
- [ ] **No giant, do-everything files.**
  Single files thousands of lines long that handle sign-in, payments, and pages all at once are where AI edits go wrong. Large areas should be split by job.
- [ ] **Dead code and duplicate copies are removed.**
  Old pages, ".copy" and "final-v2" files, and commented-out blocks confuse your AI into editing the wrong version. If you are not sure it is used, back it up outside the project, then delete it.
- [ ] **A short README explains how to run the app from scratch.**
  What to install, which keys are needed (by name, not value), the exact start command, and where the production version lives. If a new developer, or a fresh AI session, cannot start from it, it is not finished.

4 Testing and change safety

Without a safety net, every improvement is a gamble with the features that already work.

- [ ] **Your code is in version control, committed regularly.** STOP-SHIP
  Git (or your host's equivalent) with a commit before and after every working change. This is your undo button. No version control means one bad AI edit can end the project.
- [ ] **You can name your last known-good version, and get back to it.**
  Actually practice restoring it once. A backup you have never restored is a hope, not a backup.
- [ ] **At least a few automated tests cover the money paths.**
  Sign up, sign in, the main job your app does, and checkout if you take payments. They run with one command, and you run them after every AI change.
- [ ] **You test changes somewhere that is not live users.**
  A preview or staging deploy, or at minimum a local run-through of the core flows, before anything reaches production.
- [ ] **Your AI is required to report what it changed and what it tested.**
  Put this in your instructions file (page 5). "Done" with no summary, no test run, and a rewrite of unrelated files is how regressions slip in.

5 Deployment and hosting

Production is its own place, with its own settings. It is not a copy of your laptop.

- [ ] **You can deploy with a repeatable, written process.** STOP-SHIP
  One command, one button, or a short checklist that works every time, ideally automatic from version control. If deploying depends on memory or luck, write it down and test it.
- [ ] **Production uses its own database and its own keys.**
  Real users never share a database with your experiments, and test payment keys are never the ones charging real cards.
- [ ] **All production settings are set on the host, and listed by name.**
  Every environment variable production needs is configured in your hosting dashboard, and your README lists each name so a rebuild is not guesswork.
- [ ] **Errors in production do not show technical details to users.**
  Visitors see a friendly error page. Stack traces, database messages, and file paths stay in your private logs, because they are a map for attackers.
- [ ] **Your custom domain works, with www and without, over HTTPS.**
  Test both versions, test on your phone with Wi-Fi off, and confirm the domain renewal is on, with a payment method that will not expire silently.

6 Backups, monitoring, and recovery

Without this section, the first sign of trouble is an angry user, and the first data loss is permanent.

- [ ] **Your database backs up automatically.** STOP-SHIP
  Turned on in your provider's dashboard, running at least daily, and you know how long backups are kept. Confirm it with your eyes, do not assume the default is on.
- [ ] **You have restored a backup once, on purpose.**
  Restore into a test location and confirm real data comes back. Do this before you need it for real.
- [ ] **You get alerted when the app goes down or errors spike.**
  Uptime monitoring and error tracking (even free tiers) that email or message you. Silence should mean healthy, not unmonitored.
- [ ] **You can see basic usage and costs, and billing alerts are on.**
  How many users, what is using storage or AI credits, and an alert before a surprise bill. AI features in particular can run up costs while you sleep.
- [ ] **You have a one-page "if it breaks" note.**
  Where it is hosted, where backups live, how to roll back the last deploy, and who to contact. Future you, at midnight, will be grateful.

7 Launch and ownership essentials

Owning the code means little if someone else owns the accounts it runs on.

- [ ] **Every account is in your name, on your email, with your payment method.**
  Hosting, domain, database, payments, and analytics. If your app runs on a tool's, freelancer's, or friend's account, move it before launch.
- [ ] **Payments run in live mode, tested with a real small purchase.**
  Test mode off, payouts connected to your bank, receipts and refund handling working, and failed payments showing a clear message.
- [ ] **Privacy policy and terms exist, and match what your app actually does.**
  Linked in the footer, covering what data you collect, what you share it with (including AI providers), and how users can ask for deletion.
- [ ] **Users can contact you, and can delete their account and data.**
  A working contact email on the site, and a real process, even a manual one, for account and data deletion requests.
- [ ] **You have launched to a small test group before the public.**
  Five to ten real people using real flows, while you watch. They will find in an afternoon what months of solo testing missed.

Bonus: starter agent-instructions template

Copy this into a file named **`AGENTS.md`** at the top of your project (Claude Code users: `CLAUDE.md`; Cursor users: `.cursorrules`). Replace every bracket. Keep it under a page: short instructions get followed, long ones get ignored.

```
# Agent Instructions - [Your App Name]

What this is: [One sentence: what the app does and who uses it.]
Stack: [Framework, language, database, host - ask your AI to list them.]
Run it: [Install command] / [Start command] / [Test command]

Rules - always
- Make the smallest change that solves the problem. Do not rewrite
  working code, rename things, or reformat files you were not asked
  to touch.
- Before editing, say which files you will change and why. After
  editing, summarize what changed and run the tests. Report the
  test results, including failures.
- Never put secret keys in code. Use environment variables.
- Never change the database structure or delete data without
  asking me first.
- If you are unsure what I mean, ask one clarifying question
  instead of guessing.

How this project is organized
- [Folder] = [what lives there, in a few words, one line per folder]
- Shared helpers live in [folder]. Use them instead of writing new ones.

Definition of done
- The requested change works, existing tests still pass, and you
  have told me what you changed, what you tested, and what you
  did not test.
```

You have your score. Fix in this order.

Do not try to fix all 35 at once. This order protects users and data first, then makes the app safe to change, then makes it launchable.

1

Stop-ship items first.
Exposed secrets (S1), users seeing each other's data (S2), no version control (S4), no repeatable deploy (S5), no database backup (S6). Nothing else matters until these pass.

2

Make change safe.
Add the agent-instructions file (page 5), restore-test your backup, and get tests on sign-in and your main flow (S3-S4).

3

Then launch small.
Ownership, payments, and legal basics (S7), alerts on (S6), and five to ten test users before any public announcement.

A worked example of the loop this prevents

Without instructions, you ask your AI to fix a button. It rewrites the page, breaks sign-in, and you spend your next ten prompts, and your tokens, getting back to where you started. With an instructions file, a committed last-good version, and tests on sign-in, the same fix stays small, gets verified in minutes, and can be undone in one step. The AI did not get smarter. The foundation got better.

WANT AN ENGINEER TO DO THIS PASS FOR YOU?

The AI Codebase Checkup - $249 flat

I go through your actual codebase the way this checklist goes through the idea of it, and you get the evidence, not just the boxes.

- A short written report: architecture sanity check, security red flags, and a clear production deploy-readiness verdict
- A prioritized fix list you can hand to your AI tool, another developer, or me
- A 30-minute walkthrough call to go through it together

Follow-on fix and build work, if you want it, is quoted up front at $80/hr. No lock-in, no platform, your code stays yours.

Book a free 15-minute fit call or start with the Checkup:
codewithanthony.com

About Anthony

Anthony James is a senior software engineer with 15+ years of professional experience, working AI-native every day. Code With Anthony helps nontechnical and semi-technical founders finish, secure, and deploy the software they built with AI, and actually own it.

Keep this checklist working

Re-run it after any big change, before taking payments, and before any public launch push. Print a fresh copy each time and date it. Watching the Yes count climb is the progress report.

My score: _____ / 35 **Date:** _____

Copyright Code With Anthony. Share this checklist freely with credit. It is general information, not legal, security, or financial advice for your specific situation.