THE PLAYBOOK · EDITION 1

Vibe101

From a blank page to a secure, legal, findable product — one checkbox at a time. No coding background needed.

FREE SAMPLE

ENGLISH

How to use this guide

Go in order

Each chapter assumes the one before it is finished. Skipping ahead is how projects get messy.

Tick the box

Every step has a checkbox. Click it in a PDF reader (Preview, Acrobat, Chrome) and it stays ticked when you save.

Copy the prompts

Dark boxes are things to paste into Claude. Replace the words in [square brackets] with your own.

Trust "Done when"

Each step ends with a check you can see for yourself. If you can't see it, the step isn't finished.

Stuck?

Paste the exact error message into Claude and ask: "explain this simply, then fix it". Never paste passwords or keys.

Money first

Every chapter marks what costs money, and how to cap it. Set the caps before you build, not after.

Vibe101 is educational material, not legal, tax or security advice for your specific situation. Laws and platform rules change: check the current rules in your country, or ask a professional, before you launch.

Contents

00 Idea & scope — Know what you're building, for whom, and the smallest version worth launching.

01 Accounts & tools — The few accounts you need, in the right order, and what each one is for.

02 Project setup — One folder, one repository, one memory file — so nothing is ever lost.

03 Building with Claude — How to ask, review and ship small changes without breaking things.

04 **Security first** — Keep secrets secret, lock the database, cap the bills, attack your own app.

05 Legal pre-flight — The traps that get small sites fined: consent, fonts, email, renewals.

06 Money & messages — Take payments, send email and use AI without surprise bills.

07 Going live — Put it on a real domain, with one safe way to deploy.

08 Be found on Google & Bing — Sitemaps, canonicals, redirects and webmaster tools, done right.

09 Launch & first customers — Compliant outreach, content and what to measure.

10 Run & maintain — Backups, updates, key rotation and a calm incident checklist.

CHAPTER 04

Security first

Nobody can read, change or run up a bill on anything they shouldn't. Do this chapter before you show the product to anyone.

TIME	YOU NEED
2–3 hours	Your project folder, your database dashboard, Claude Code

01 Keep secrets out of your code

A secret is any password, API key or token. If it lands in your code, anyone who sees the code can use it. Anything in the browser is public, so a key in front-end code is a leaked key.

- Put secrets only in a file called `.env.local` (or in your host's Environment Variables screen).
- Add `.env*` to `.gitignore` and lock the file: `chmod 600 .env.local`.
- Variables starting with `VITE_` or `NEXT_PUBLIC_` are sent to every visitor. Never put a secret there.
- Before every commit run `git status`. Never use `git add -A` without looking first.

PASTE INTO CLAUDE

```
Check this project for secrets in the code or git history, for .env files that are not git-ignored, and for secret-looking values in VITE_ or NEXT_PUBLIC_ variables. List every finding and fix it. Do not print the secret values.
```

DONE WHEN `git status` shows no `.env` file, and the check above reports nothing.

02 Lock the database

Your database has two keys. The public one lives in the browser; the secret one lives only on your server. Row Level Security (RLS) is the rule that stops the public key from reading other people's rows.

- Turn RLS on for every table. A table with RLS on and no policy is closed — that is the safe default.
- Write one policy per need ("a user sees only their own rows"). Keep them boring and short.
- Use the secret key only in server code, never in the browser.

PASTE INTO CLAUDE

```
List every database table. For each one tell me whether RLS is on and show its policies. Turn RLS on wherever it is off, and write the smallest policy that lets a signed-in user read and change only their own rows.
```

DONE WHEN Using only the public key, you cannot read another user's row. Try it and see the error.

03 Rate-limit every public or costly route

Anything a stranger can call — login, sign-up, contact form, AI chat — can be called 10,000 times by a script. A rate limit says "too many requests, try later".

- Limit by IP address and, once signed in, by user.
- Hosts that run many small copies of your server (serverless) do not share memory, so a counter kept in memory is useless. Use a shared store such as Redis or a database table.
- Return the standard answer `429 Too Many Requests`.

PASTE INTO CLAUDE

```
List every public API route in this project. Add a rate limit to each one, using a shared store that works on serverless hosting. Tighter limits for login, sign-up, email-sending and AI routes. Add a test that fires 30 quick requests and expects a 429.
```

DONE WHEN Thirty rapid requests to your login route end in `429`.

04 Harden the login

Hiding a button is not security. The server must check who you are and what you may do on every request.

- Check permissions on the server for every action, not only in the screen.
- Require a decent password (12+ characters) or use magic-link sign-in; offer 2-factor sign-in for admins.
- Slow down repeated failures. Never ship a default username or password.
- Admin areas: allow only an explicit list of staff emails.

PASTE INTO CLAUDE

```
Review every route that changes data. Show me which ones check the user's identity and permission on the server and which rely on the screen hiding the option. Fix the gaps, and add a test that calls each protected route without a login and expects it to be refused.
```

DONE WHEN Calling an admin route without logging in returns an error — never data.

05 Cap what AI and paid services can cost

A bug, a loop or an attacker can turn an AI or SMS feature into a four-figure bill overnight. Caps turn a disaster into an annoyance.

- Set a hard monthly limit in the provider's dashboard.
- Give each user a monthly allowance (credits) and stop when it runs out.
- Put a maximum number of steps on any loop that calls a paid service.
- Switch on a spend alert to your email.

PASTE INTO CLAUDE

```
Find every call to a paid API in this project (AI, SMS, email, maps). For each, add a per-user monthly allowance, a maximum number of loop iterations and a clear error when the limit is reached. List what you changed.
```

DONE WHEN You hit the limit on purpose in testing and see the clear error, not a crash.

06 Fail closed, never open

When a safety check cannot run — a missing key, a service that is down — the answer must be "no". A check that quietly switches itself off when its key is missing is an open door.

- Search for code like "only check if the key exists". Make it refuse instead.
- Remove any "temporary" bypass before launch.

PASTE INTO CLAUDE

```
Search the code for security checks that are skipped when a secret or setting is missing (captcha, webhook signature, auth). Change each so it refuses the request instead, and explain every change in one line.
```

DONE WHEN With the secret removed, the protected action is refused, not allowed.

07 Attack your own app

A green scanner proves little. Try to break it the way a stranger would, then keep the tests so they run forever.

- Ask for another user's data by changing the number in the URL.
- Call every route with no login, a wrong login and a huge input.
- Type `<script>` into every text box and see what shows up.
- Click the buy button twice, quickly.

PASTE INTO CLAUDE

```
Act as an attacker against this app on my local machine. Write automated tests that try: reading another user's data, calling protected routes without login, oversized inputs, script injection in text fields, and double-submitting payments. Run them and report what got through.
```

DONE WHEN Every attack test fails to get in, and the tests are saved in the project.

08 Clean up, then ship

Leftovers are where leaks hide. Do this before your first public deploy and after each big change.

- Delete junk: `.DS_Store`, logs, build folders, test files, unused packages.
- If a key was ever pasted into a chat, an email or a commit, treat it as leaked: create a new one and delete the old.
- Check public data files. A downloadable file with owner phone numbers or emails is a breach.

PASTE INTO CLAUDE

```
List files in this repo that should not be committed (junk, logs, build output, env files,
test leftovers) and anything in the public bundle that contains personal data. Remove them
and add the right entries to .gitignore.
```

DONE WHEN `git status` is clean and the public files contain no personal data.

Common traps

- A real case: a stray `.env.local` was committed once. One `git status` glance would have caught it.
- A real case: a public data file shipped owners' phone numbers to every visitor's browser. Keep private columns out of anything the browser downloads.
- A counter kept in memory looks like a rate limit but does nothing on serverless hosting.
- Keys pasted into a chat count as leaked. Replace them.

← BACK TO CONTENTS

VIBE101

You just finished one chapter of eleven

This free sample is Chapter 04, Security first. The full guide adds idea and scope, accounts, project setup, building with Claude, legal pre-flight, money and messages, going live, search visibility, launch and maintenance, plus a final check and glossary.

Get the full Vibe101 at vibe101.eu