

The Vibe Coder's *Playbook*

Everything you need to know to build real things with AI: websites, mobile apps, AI models, backends. Includes how to plan, design, build, test, secure, ship, and pay for it, plus copy-paste prompts for every step.

FOR: YOU · BUILT AROUND YOUR CLAUDE CODE SETUP

Web

Next.js, Vite, PWAs, SEO, deploy

Mobile

Expo, Flutter, Play Store & App Store

AI & Models

APIs, RAG, agents, fine-tuning

Ship safely

Security, costs, legal, launch

Contents & the 12 golden rules

01	The 12 golden rules	2	10	Design that doesn't look AI-made	12
02	Your installed toolkit	3–4	11	Security: the non-negotiables	13
03	The vibe-coding loop	5	12	Testing & debugging	14
04	Prompting secrets	6	13	Git, deploy & rollback	15
05	Choosing a stack	7	14	Speed, SEO & analytics	16
06	Web app blueprint	8	15	Money, costs & legal	17
07	Mobile app blueprint	9	16	Launch checklist	18
08	AI apps & building models	10	17	Glossary & cheat sheet	19
09	Backend, data & auth	11			

01 · The 12 golden rules of vibe coding

1. **Plan before you prompt.** Write 5–10 lines describing who the app is for, what it does, and what "done" looks like. Vague asks get vague apps.
2. **One feature per conversation.** Small, focused asks beat "build me Instagram". Run `/clear` between unrelated tasks.
3. **Commit before every big change.** Git is your undo button. No git means no safety net.
4. **Show, don't describe.** Paste screenshots of bugs and reference designs. A picture replaces 200 words.
5. **Paste the exact error.** Copy the full red text from the terminal or browser console. Never just say "it's broken".
6. **Make the AI verify its own work.** Ask it to run the app, open the page in a browser (Playwright), and check that the change works.
7. **Secrets never go in frontend code.** API keys belong in `.env` files and server code only. Anything shipped to the browser is public.
8. **Use boring, popular tech.** AI knows React, Next.js, Supabase, Firebase, Expo, and Python deeply. Exotic stacks lead to hallucinations.
9. **Read what it changed.** You don't need to understand every line, but skim the diff. Ask "explain this change in simple words".
10. **Test on a real phone.** Most users are on mobile. Check every screen at 360px wide.
11. **Keep a CLAUDE.md.** It holds project rules, stack, and do/don't lists, and the AI reads it every session (`/init` creates one).
12. **Ship small, ship often.** A live, ugly v1 teaches more than a perfect app that never launches.

THE REAL SECRET

Vibe coding isn't about knowing syntax. It's about **knowing what to ask for**, **knowing what can go wrong**, and **checking the result**. This playbook gives you the vocabulary and checklists for all three.

Your installed toolkit

These are installed globally, so they work in every project. Skills switch on automatically when your request matches. You can also call one by name, for example *"use the taste skill"*.

From the video: design skills NEW

Skill	What it does	Say this
design-taste-frontend Taste Skill v2	Stops generic "AI slop" UI. Reads your brief, picks a design direction, and tunes variance, motion, and density.	"Build the landing page using the taste skill"
high-end-visual-design	Agency-level fonts, spacing, shadows, and cards. Blocks the cheap-looking defaults.	"Make this feel premium"
redesign-existing-projects	Audits an existing site and upgrades it without breaking features.	"Redesign my homepage, keep all features"
minimalist-ui · industrial-brutalist-ui · gpt-taste	Style presets: calm editorial, raw blueprint, or bold GSAP motion.	"Use minimalist-ui for the dashboard"
image-to-code	Turns a screenshot or design image into matching code.	"Recreate this screenshot as a React page"
imagegen-frontend-web / -mobile · brandkit	Generates design reference images, app screens, and brand boards (needs an image model).	"Make mockups for a food app"
stitch-design-taste	Writes a DESIGN.md design system file for your project.	"Create a DESIGN.md for this app"
web-design-guidelines Vercel	Audits UI against 100+ rules: accessibility, focus, tap targets, forms, loading states.	"Review my UI against the web guidelines"
vercel-react-best-practices · composition-patterns	React/Next.js performance and clean component structure.	"Check this React code for performance"
playwright-cli	Lets Claude open a real browser, click around, read console errors, and take screenshots.	"Open localhost:5173 and test sign-up"
full-output-enforcement	Stops truncated code and <code>// ...rest here</code> placeholders.	"Give me the complete file"

MANAGING THEM

Type `/plugin` to browse, enable, or disable plugins. Skills live in `~/.claude/skills`. Update skills with `npx skills update -g`. Some MCP plugins (Supabase, Firebase, GitHub, Context7) ask you to sign in the first time you use them.

Plugins & the design library

Awesome DESIGN.md: 70+ ready-made design systems NEW

Video tool #3. These are real design languages (Stripe, Linear, Notion, Vercel, Claude...) written as Markdown that AI agents follow. It's downloaded to `C:\Users\91992\awesome-design-md\design-md\`, one folder per brand, each with a `DESIGN.md` plus `preview.html` so you can see it first.

PROMPT — copy & paste

Copy `C:\Users\91992\awesome-design-md\design-md\linear\DESIGN.md` into this project root. Rebuild the dashboard following `DESIGN.md` exactly (type, spacing, colors), but keep our own logo and brand color.

Official Claude Code plugins

Plugin	Use it for	Plugin	Use it for
frontend-design	Distinctive production UI	security-guidance	Warns on risky code as it's written
context7	Up-to-date library docs (no outdated APIs)	code-review	<code>/code-review</code> finds bugs before shipping
feature-dev	Guided explore → plan → build workflow	code-simplifier	Cleans messy AI code
commit-commands	<code>/commit</code> , push, and PR in one step	typescript-lsp	Real type errors, smarter edits
supabase · firebase	Database, auth, and storage from chat	expo	React Native mobile apps
github	Issues, PRs, repos	vercel · claude-md-management	Deploys · keeping CLAUDE.md sharp

Which tool when?

Situation	Reach for
Starting a new site from scratch	Plan mode → design-taste-frontend + a <code>DESIGN.md</code>
"My site looks cheap / AI-made"	redesign-existing-projects → web-design-guidelines audit
Using a library the AI gets wrong	context7 : "use context7 for the latest Expo docs"
"Does it actually work?"	playwright-cli : open, click, screenshot, read console
Big feature with many files	<code>/feature-dev</code> guided workflow
Before pushing to production	<code>/code-review</code> + <code>/security-review</code> → <code>/commit</code>
Database, auth, storage tasks	supabase or firebase plugin
Building a phone app	expo plugin + imagegen-frontend-mobile for mockups
I have a screenshot of a design	image-to-code : "recreate this as a page"
Code became spaghetti	code-simplifier : "simplify this file, same behaviour"

The vibe-coding loop

Every project, big or small, runs through five stages. Skipping **Plan** and **Verify** is the #1 reason vibe-coded apps fall apart.

1 Plan: write a mini-PRD (Product Requirements Doc)

PROMPT — copy & paste

```
I want to build [APP]. Don't write code yet.
Users: [who]. Problem: [what pain]. Core features for v1: [3-5 bullets].
Platform: [web / mobile / both]. Must work on cheap Android phones.
Ask me up to 8 questions to fill gaps, then write PLAN.md with:
pages/screens, data model (tables + fields), user roles, tech stack
with reasons, and a step-by-step build order split into small milestones.
```

Use **Plan mode** (Shift+Tab twice) so Claude researches and proposes without editing files.

2 Design: decide the look before building

PROMPT — copy & paste

```
Using the taste skill, propose 2 design directions for this app (fonts,
colors, spacing, component style) as a DESIGN.md. Mobile-first.
Avoid: purple gradients, glassmorphism everywhere, emoji icons, generic hero.
```

3 Build: one milestone at a time

PROMPT — copy & paste

```
Implement milestone 2 from PLAN.md only. Follow DESIGN.md and CLAUDE.md.
When done: run the build, fix all errors, and list every file you changed.
```

4 Verify: make the AI prove it works

PROMPT — copy & paste

```
Start the dev server, then use playwright-cli to open the app at 390px
and 1280px widths. Test: sign up → create item → edit → delete → log out.
Screenshot each step, read the browser console, and fix anything broken.
```

5 Ship: commit, deploy, check live

PROMPT — copy & paste

```
Run /code-review on my changes, fix real issues, then commit with a clear
message. Deploy to production and open the live URL to confirm it works.
```

GOOD HABITS

- Keep a `PLAN.md` checklist and tick milestones off.
- Start fresh chats for new features. Long chats get confused.
- Save decisions to `CLAUDE.md` ("we use Supabase, not Firebase").

TRAPS

- "Build the whole app" in one prompt.
- Accepting changes you never ran.
- Asking for 10 fixes at once. Fix one, test, repeat.

Prompting secrets

The anatomy of a great prompt

Part	Example
Goal	"Add a password reset flow"
Context	"Auth uses Supabase; login page is <code>src/pages/Login.jsx</code> "
Constraints	"Don't change the login design. No new libraries. Must work offline-first."
Done means	"User gets email → clicks link → sets password → lands on dashboard"
Verify	"Test it in the browser and show me a screenshot"

Templates you'll use every week

Bug report

PROMPT — copy & paste
Bug: [what happens] instead of [expected].
Steps: 1) ... 2) ... 3) ...
Happens on: [phone/desktop, browser].
Error: [paste full console/terminal text]
Screenshot attached. Find the root cause first, explain it, then fix it. Don't patch symptoms.

Understand code

PROMPT — copy & paste
Explain how [feature] works in this project like I'm new to coding. Which files, what order, where data comes from.

Stuck in a loop

PROMPT — copy & paste
Stop. We've tried 3 fixes that didn't work. List 5 possible root causes ranked by likelihood, add logging to confirm which one it is, then fix only that.

Before a risky change

PROMPT — copy & paste
Before editing, tell me which files you'll change, what could break, and how we roll back. Wait for my OK.

Power moves in Claude Code

Move	Why
<code>/init</code>	Creates CLAUDE.md, the project memory read at every session start
<code>/clear</code> · <code>/compact</code>	Fresh context for a new task · shrinks a long chat while keeping key facts
Shift+Tab	Cycle modes: normal → auto-accept edits → Plan mode
<code>@file</code>	Point at an exact file so the AI doesn't guess
Esc · Esc Esc	Interrupt a wrong direction · rewind to an earlier message

WORDS THAT CHANGE RESULTS

"root cause" (no band-aids) · "minimal change" (no rewrites) · "don't add libraries" · "match the existing code style" · "mobile-first" · "production-ready" (error handling and loading states) · "ask me before deleting anything"

Choosing a stack

Pick the most popular option the AI knows best. Here's a default choice for almost anything you might want to build.

You want to build	Default stack	Why / alternatives
Landing page / portfolio	Astro or plain HTML + Tailwind → Vercel / Netlify	Fast, free hosting, great SEO
Web app (SaaS, portal)	Next.js (or Vite + React) · Tailwind · shadcn/ui · Supabase	Biggest ecosystem; Firebase if you want Google's stack
Installable web app	Any web stack + PWA (manifest + service worker)	No app store needed, one codebase
Mobile app (iOS + Android)	Expo (React Native) + Supabase/Firebase	Flutter is also great; native Swift/Kotlin only for heavy device features
Desktop app	Tauri (small, fast) or Electron	Reuse your web UI
AI chatbot / AI feature	Next.js + Vercel AI SDK or Anthropic SDK	Streaming, tool calling built in
Train / fine-tune a model	Python · PyTorch · Hugging Face · Google Colab / Kaggle GPU	Free GPUs to start; see page 10
Data / ML experiments	Python · Jupyter · pandas · scikit-learn	Classic ML beats deep learning on small tabular data
API / backend only	Node (Hono/Express) or Python (FastAPI)	FastAPI if you're also doing ML
Chrome extension	Manifest V3 + Vite (WXT or Plasmo framework)	Publish to the Chrome Web Store for a one-time fee
Bots (Telegram/Discord/WhatsApp)	Node or Python + platform API, hosted on Railway/Render	WhatsApp needs the Meta Business API
2D game	Phaser (web) or Godot	Unity for 3D or console games
Automation / scripts	Python or Node scripts · n8n for no-code flows	Run on a schedule with cron or GitHub Actions

Backend-as-a-service: Supabase vs Firebase

	Supabase	Firebase
Database	PostgreSQL (SQL, relations, joins)	Firestore (NoSQL documents)
Security	Row Level Security (RLS) policies in SQL	Security Rules files
Best for	Structured data, reports, admin dashboards	Realtime chat, mobile-first apps, push (FCM)
Gotcha	Free projects pause after inactivity	Costs per read, so watch loops that re-read data

RULE OF THUMB

If you can't explain why you need something exotic, don't use it. **React + Tailwind + Supabase + Vercel** covers about 80% of ideas. Ask: *"What's the simplest stack that does this, and why?"*

Web app blueprint

The layers of every web app

Layer	Popular choice	What you must know
Frontend	React / Next.js, Tailwind CSS, shadcn/ui	Components, state, routing. Next.js adds server rendering for SEO.
Auth	Supabase Auth, Firebase Auth, Clerk	Never build your own password storage. Verify email. Protect routes on the <i>server</i> .
Database	Postgres (Supabase / Neon), Firestore	Design tables first. Every table needs access rules.
File storage	Supabase Storage, Firebase Storage, Cloudflare R2	Limit file size and type. Private buckets for personal docs.
Payments	Razorpay (India: UPI, cards), Stripe (global)	Confirm payment via webhook on the server, never trust the browser.
Email	Resend, Brevo, SendGrid	Set up SPF/DKIM on your domain or mail lands in spam.
Push notifications	Web Push (VAPID), Firebase Cloud Messaging	Ask permission after a user action, not on page load.
Hosting	Vercel, Netlify, Cloudflare Pages	Every git push → preview URL; main branch → production.
Domain	Cloudflare, Namecheap, GoDaddy	Point DNS (A / CNAME records) to your host. HTTPS is automatic.

Folder structure the AI expects

TERMINAL

```
src/  
  components/ reusable UI pieces  
  pages/ or app/ screens / routes  
  lib/ supabase client, helpers  
  hooks/ custom React hooks  
  styles/  
public/ images, icons, manifest  
.env.local SECRETS (never commit)  
CLAUDE.md PLAN.md DESIGN.md
```

Starting a new web project

PROMPT — copy & paste

```
Create a new Next.js app with TypeScript,  
Tailwind, and shadcn/ui in this folder.  
Add Supabase auth (email + Google),  
a protected /dashboard, and a .env.example.  
Set up CLAUDE.md with the stack and rules.
```

Every page needs these states

AI often only builds the "happy path". Always ask for: **loading** (skeleton or spinner) · **empty** ("No items yet, create one") · **error** (friendly message and retry) · **success** (toast or confirmation) · **offline** (for PWAs) · **not logged in** (redirect to login) · **no permission** (403 page) · **404**.

MAKE IT INSTALLABLE (PWA)

Add `manifest.json` (name, icons 192/512, theme color), a service worker for offline caching, and HTTPS. On Android, users get "Install app". On iPhone, they use Share → Add to Home Screen. Push notifications on iPhone only work after the app is installed.

Mobile app blueprint

Four ways to get on a phone

Option	Pros	Cons	Pick when
PWA	One web codebase, no store fees, instant updates	Limited iOS features, not in stores by default	Portals, tools, college/community apps
Expo / React Native	JS/React skills, true native app, OTA updates, AI knows it well	Some native modules need a "dev build"	Default choice for store apps
Flutter	Beautiful, fast UI, one codebase for many platforms	Dart language, bigger app size	Animation-heavy or custom UI
Native (Swift / Kotlin)	Full device power, best performance	Two codebases to maintain	AR, heavy camera or Bluetooth work

Expo quick path

PROMPT — copy & paste

Create an Expo app (TypeScript, Expo Router) with tabs: Home, Search, Profile. Use Supabase for auth and data. Add a splash screen and app icon placeholders. Explain how to run it on my Android phone with Expo Go.

1. **Develop:** `npx expo start` → scan the QR with the Expo Go app on your phone.
2. **Build:** `eas build -p android` (cloud build, no Android Studio needed) → `.aab` file.
3. **Update:** `eas update` pushes JS fixes instantly without store review (OTA).

Publishing to the stores

Google Play

- Developer account: **one-time \$25**
- New personal accounts must run a **closed test with testers for 14 days** before going public. Check the current rules.
- Needs: privacy policy URL, data safety form, screenshots, content rating
- **Back up your upload/signing key.** Lose it and you can't easily update the app.

Apple App Store

- Apple Developer Program: **\$99/year**
- Builds need a Mac, or use **EAS Build** in the cloud
- Stricter review; common rejections: crashes, broken login, missing "delete account", vague privacy labels
- Test with **TestFlight** first

MOBILE MUST-HAVES

- Ask for permissions (camera, location) **only when needed**, with a reason shown first.
- Tap targets ≥ 44 –48px, readable text ≥ 16 px, respect the safe area (notch, home bar).
- Handle no internet gracefully. Test on a cheap, slow Android phone, not just your own.
- If users can create an account, they must be able to **delete it in-app** (both stores require this).

AI apps & building models

Two different jobs: **using** an existing model inside your app (99% of cases) vs. **training** your own (rarely needed, and only for narrow tasks).

A · Using AI models in your app

Pattern	What it is	Example
Plain API call	Send text, get text. Call it from your server (the API key stays secret).	Summarise notes, rewrite text
Structured output	Ask for JSON matching a schema, then validate it.	Extract fields from a form or receipt
Streaming	Show words as they're generated, so the UI feels fast.	Chat interfaces
RAG	Split your docs into chunks → embeddings → vector DB (pgvector) → retrieve relevant chunks → send to the model with the question.	"Chat with our college notes"
Tool calling	The model asks your code to run functions (search DB, send email).	Booking assistant
Agents	A loop of think → use tool → observe → repeat until done.	Research or coding helpers

PROMPT — copy & paste

Add an AI "summarise" button to the notes page. Call the Claude API from a server route (never from the browser), key in .env. Stream the response, cap output length, handle errors, and rate-limit to 20 requests per user per day.

AI COST TRAPS

Paid APIs bill per token. Always: put the key server-side, rate-limit per user, cap max tokens, cache repeated answers, and pick the smallest model that works (e.g. Haiku-class for simple tasks, bigger models for reasoning). Set a **spending limit** in the provider dashboard on day one.

B · Training or fine-tuning your own model

- 1. Define the task:** classify, detect, predict, or generate? What's the input and output?
- 2. Data first:** collect and label examples. Split into train / validation / test (e.g. 80/10/10). Bad data means a bad model.
- 3. Baseline:** try a simple model or a prompt to a big LLM first. Often good enough.
- 4. Train:** fine-tune a pretrained model (transfer learning) instead of starting from scratch.
- 5. Evaluate:** accuracy, precision/recall, F1, measured on the *test* set you never trained on.
- 6. Deploy:** Hugging Face Spaces, Replicate, a FastAPI server, or on-device (TFLite / ONNX).

Task	Start with
Tabular prediction	scikit-learn, XGBoost
Image classify / detect	Fine-tune ViT / YOLO
Text classify	Fine-tune DistilBERT
Custom chat style	LoRA on a small open LLM (Unsloth)
Run LLMs locally	Ollama (installed on your PC)
Free GPUs	Google Colab, Kaggle

Key words: epoch (one pass over the data) · overfitting (memorises training data, fails on new data) · LoRA (cheap fine-tuning that trains small adapter layers) · quantisation (shrink a model to run on weaker hardware) · inference (using the trained model).

Backend, data & auth

Design your data before your screens

PROMPT — copy & paste

Before coding, design the database for this app: tables, columns with types, primary/foreign keys, indexes, and who can read/write each table (RLS). Show it as a table list plus a SQL migration file. Explain each relation simply.

Core concepts

- **Table / row / column:** a spreadsheet with strict types.
- **Primary key:** unique id (use `uuid`).
- **Foreign key:** links rows (`post.user_id` → `users.id`).
- **Index:** makes searches fast on big tables.
- **Migration:** a versioned SQL file that changes the schema. Keep them in git, in order.
- **RLS:** database-level rules such as "users can only read their own rows".

Where code runs (it matters!)

Client (browser/app)	Server
UI, validation for UX	Real validation & permissions
Public keys only (<code>anon</code>)	Secret keys, <code>service_role</code>
Can be edited by any user	Trusted

Server options: Next.js API routes, Supabase Edge Functions, Firebase Cloud Functions, Vercel Functions.

Auth & roles done right

- Use the provider's auth (Supabase / Firebase / Clerk). Add **Google sign-in**, since most users prefer it.
- Store roles (`student` , `teacher` , `admin`) in a `profiles` table. **Users must not be able to edit their own role.**
- Check roles on the **server / RLS**. Hiding an admin button is not security.
- Approval flows: new sensitive roles start as `pending` until an admin approves.
- Sessions expire. Handle "token expired" by re-logging the user in, not by crashing.

Other backend building blocks

Need	Use
Scheduled jobs	Vercel Cron, Supabase <code>pg_cron</code> , GitHub Actions schedule
Background / slow work	Queues (Vercel Queues, Inngest, Trigger.dev); don't make the user wait
Realtime updates	Supabase Realtime, Firestore listeners, WebSockets
Search	Postgres full-text search → Algolia / Meilisearch when big
Caching	CDN cache for pages, Redis (Upstash) for hot data
Webhooks	Endpoints other services call (payments, GitHub); verify the signature

Design that doesn't look AI-made

Users judge trust in about 50 milliseconds. Most AI-generated sites look the same, so here's how to escape the template look.

SIGNS OF "AI SLOP"

- Purple-blue gradients and glowing blobs
- Glassmorphism cards on everything
- Emoji as icons, 🚀 in every heading
- "Unlock the power of..." / "Revolutionize your..." copy
- Three identical feature cards with icons
- Everything centered, everything the same size
- Animated gradient text (often renders as a solid box)

WHAT PREMIUM LOOKS LIKE

- One strong typeface pairing (e.g. a serif display + clean sans)
- Generous whitespace and a consistent spacing scale (4/8px)
- 1 brand color + neutrals; accents used rarely
- Real photos, real content, real numbers
- Clear hierarchy: one main action per screen
- Subtle motion with a purpose ($\leq 300\text{ms}$)
- Style fits the audience (a government portal $\neq$ a startup)

Your design workflow

1. **Collect references:** screenshots of 3–5 sites you love (Dribbble, Mobbin for apps, Land-book, Godly).
2. **Lock a design system:** pick a `DESIGN.md` from `awesome-design-md` or generate one with *stitch-design-taste*.
3. **Build with taste:** "use the taste skill and follow DESIGN.md".
4. **Audit:** "review my UI against web-design-guidelines and fix the issues".
5. **Check on a phone** and in dark mode.

PROMPT — copy & paste

```
Here are 3 screenshots of sites I like [attach]. Extract their shared design language (type, spacing, colors, radius, shadows) into DESIGN.md, then redesign my homepage to match, using the redesign-existing-projects skill. Keep all features and links working.
```

Accessibility basics (also improves SEO)

Rule	Why
Text contrast $\geq 4.5 : 1$	Readable in sunlight and for low vision
Every image has <code>alt</code> text	Screen readers, Google Images
Every input has a visible label	Placeholders disappear when typing
Works with keyboard only (Tab, Enter)	Visible focus ring; no keyboard traps
Don't block zoom; base font 16px	iPhones auto-zoom smaller inputs
Don't rely on color alone	Add icons or text to errors and statuses

Security: the non-negotiables

Vibe-coded apps get hacked for boring reasons: leaked keys, missing access rules, and trusting the browser. Run this checklist before every launch.

The checklist

- ☐ **No secrets in frontend code or git.** Keys go in `.env`, and `.env` goes in `.gitignore`. Leaked a key? **Rotate it immediately.** Deleting the commit isn't enough.
- ☐ **Know your keys:** `NEXT_PUBLIC_` / `VITE_` variables are *shipped to the browser*. Supabase `anon` key = public (fine). `service_role`, OpenAI/Anthropic, Razorpay secret, and DB passwords = server only.
- ☐ **RLS on every Supabase table** / Security Rules in Firebase. Test: log in as user A and try to read user B's data.
- ☐ **Validate on the server.** Re-check prices, roles, and ownership server-side. Use a schema library (Zod / Pydantic).
- ☐ **Authorization on every endpoint:** "Is this user logged in?" *and* "Is this user allowed to touch *this* record?"
- ☐ **Rate limiting** on login, sign-up, OTP, AI, and email-sending endpoints, so bots can't drain your wallet.
- ☐ **File uploads:** limit type and size, store in private buckets, never execute uploaded files.
- ☐ **Escape output:** never inject raw user HTML (`dangerouslySetInnerHTML` , `innerHTML`) without sanitising (XSS).
- ☐ **Parameterised queries only.** Never build SQL by gluing strings together (SQL injection).
- ☐ **Payments:** confirm via a signed webhook. Never mark "paid" because the browser said so.
- ☐ **Dependencies:** `npm audit`. Don't install random packages the AI invents. Check they exist and are popular (typosquatting is real).
- ☐ **Admin accounts:** strong unique password + 2FA on GitHub, Vercel, Supabase, Google, domain registrar.
- ☐ **Error messages** shown to users shouldn't reveal stack traces, SQL, or keys.
- ☐ **Backups:** know how to restore your database. Export important data regularly.

PROMPT — copy & paste

```
Do a full security review of this project as an attacker would. Check:
exposed secrets, tables without RLS, endpoints without auth/ownership checks,
client-side-only validation, missing rate limits, XSS, unsafe uploads, and
webhook verification. Rank findings by severity and fix critical ones first.
```

Tools: `/security-review` (built in) · **security-guidance** plugin (installed) · Supabase dashboard → Advisors → Security · GitHub secret scanning.

Testing & debugging

Where errors show up

Place	How to see it
Browser console	F12 → Console (red text). On phone: chrome://inspect over USB.
Network tab	F12 → Network: failed requests (4xx/5xx), slow calls, response body.
Terminal / dev server	Build errors, server route crashes.
Hosting logs	Vercel → Project → Logs; Supabase → Logs; Firebase → Functions logs.
Production errors	Sentry (free tier): get alerted when real users crash.

HTTP status codes you'll meet

200 OK · 201 created · 301/302 redirect · 400 bad input · 401 not logged in · 403 logged in but not allowed (often RLS) · 404 not found · 409 conflict/duplicate · 429 rate limited · 500 server crashed · 502/504 server down or timed out · **CORS error**: the server doesn't allow your domain.

The debugging method

1. **Reproduce** it reliably. Note exact steps, device, and account.
2. **Read the error**. File name and line number tell you where to look.
3. **Isolate**: did it work before? Use git history to find the change that broke it.
4. **Hypothesise** → **log** → **confirm**. Don't guess-fix three things at once.
5. **Fix the root cause**, then re-test the original steps *and* nearby features.

Tests worth having (ask the AI to write them)

E2E (Playwright)

Clicks through real flows in a browser: sign up, pay, post. **Highest value for vibe coders.**

Unit (Vitest / Jest / pytest)

Tests small functions: price calculation, date formatting, permissions.

PROMPT — copy & paste

Write Playwright end-to-end tests for our 5 most important user flows. Run them, fix failures in the app (not by weakening tests), and add an npm script "test:e2e". Then show me how to run them before each deploy.

PHONE TESTING SECRET

Run `npm run dev -- --host`, connect your phone to the same Wi-Fi, and open `http://YOUR-PC-IP:PORT`. Find your IP with `ipconfig`. Or deploy a preview URL and open it on the phone.

Git, deploy & rollback

Git in 10 commands

Command	Meaning
<code>git init</code>	Start tracking this folder
<code>git status</code>	What changed?
<code>git add -A + git commit -m "msg"</code>	Save a checkpoint
<code>git log --oneline</code>	History of checkpoints
<code>git diff</code>	Exactly what changed since the last checkpoint
<code>git switch -c feature-x</code>	New branch to experiment safely
<code>git push</code>	Upload to GitHub (triggers auto-deploy)
<code>git restore file</code>	Throw away uncommitted changes to a file
<code>git revert <id></code>	Safely undo a pushed commit
<code>git stash</code>	Park changes temporarily

Or just say: *"commit this with a good message"* / *"undo everything since the last commit"* (/commit from commit-commands).

Environments

Env	Where	Rule
Local	localhost	Break anything; use test data
Preview	Auto URL per branch/PR	Share with testers, check on phone
Production	Your domain	Real users. Separate keys and DB from local if possible.

Env variables must be added in the **hosting dashboard** too (Vercel → Settings → Environment Variables). Missing env vars in production is the #1 "works locally, broken live" cause.

Rollback plan

- **Vercel / Netlify:** Deployments → pick the last good one → *Promote / Publish*. Takes seconds.
- **Database changes** are the dangerous part. Back up before migrations. Prefer additive changes (add a column, don't rename or drop one).
- **Mobile:** store builds can't be rolled back instantly. Use OTA updates (EAS Update) for JS fixes and staged rollouts on Play.

DEPLOY RULE

Never deploy on a Friday night or right before an exam/event when users depend on the app. Deploy, open the live site, click the main flows, and check the logs for 10 minutes.

Speed, SEO & analytics

Performance: Core Web Vitals

Metric	Means	Good
LCP	How fast the main content appears	< 2.5 s
INP	How fast the page reacts to taps	< 200 ms
CLS	How much things jump around while loading	< 0.1

Quick wins: compress images (WebP/AVIF, correct size) · lazy-load below-the-fold images · set image width/height · fewer fonts (2 max, `font-display: swap`) · code-split big pages · avoid huge libraries for small jobs · cache API responses. Measure with **PageSpeed Insights** or Chrome Lighthouse.

SEO: getting found on Google

- ☐ Unique `<title>` (≤ 60 chars) and meta description per page
- ☐ One `<h1>` per page, logical h2/h3
- ☐ Clean URLs: `/courses/physics` not `?id=82`
- ☐ `sitemap.xml` + `robots.txt`
- ☐ Open Graph tags (nice WhatsApp/LinkedIn previews)
- ☐ Submit the site in **Google Search Console**
- ☐ Structured data (JSON-LD) for organisation, articles, FAQs
- ☐ Server-rendered content (Next.js/Astro); pure SPAs index worse
- ☐ Fast + mobile-friendly = ranking factors
- ☐ Real, useful content beats keyword stuffing

PROMPT — copy & paste

Audit this site's SEO: titles, meta descriptions, headings, OG tags, sitemap, robots.txt, structured data, alt text, and Core Web Vitals. Fix what you can and give me the list of things I must do in Google Search Console.

Analytics: know what users actually do

Tool	Good for
Vercel Analytics / Plausible	Simple, privacy-friendly page views
Google Analytics 4	Free, detailed, traffic sources
PostHog	Funnels, session replays, feature flags
Microsoft Clarity	Free heatmaps and recordings

Track a few **key events** (signed_up, created_post, paid). Not everything. Ask: "where do users drop off?"

Money, costs & legal

Run on free tiers, but know the limits

Service	Free tier covers	Watch out for
Vercel / Netlify	Hobby sites, previews, serverless	Hobby plan is for non-commercial use; bandwidth and function limits
Supabase	Small DB, auth, storage	Projects pause when inactive; DB size cap
Firebase (Spark)	Auth, Firestore quotas	Cloud Functions need the paid Blaze plan; reads add up
Cloudflare	DNS, CDN, Pages, R2 storage	Very generous; great for static sites
AI APIs	Usually paid per token	Set hard spending limits ; rate-limit users
Domain	Not free	A .com/.in/.online costs a few hundred to ~₹1,000/yr; check the renewal price

Free tiers and prices change often. Ask: "check current pricing for X and estimate my monthly cost at 1,000 users".

BILL-SHOCK PREVENTION

Turn on **budget alerts** on every paid service. Never put a credit card on a project with a possible infinite loop (a function that triggers itself, a `useEffect` refetching forever). Check usage dashboards weekly in the first month.

Taking payments (India)

- **Razorpay / Cashfree / PhonePe PG:** UPI, cards, netbanking. KYC needed (PAN, bank account; business proof for higher limits).
- Gateway fees are usually ~2% per transaction. Check the current rate.
- Need: Terms, Privacy, **Refund & Cancellation** and Contact pages on your site (gateways check these).
- GST registration becomes mandatory above the turnover threshold. Talk to a CA once you earn.

Legal basics

Must have	Why
Privacy Policy	Required by the app stores, Google sign-in, and payment gateways; required under India's DPDP Act 2023 if you collect personal data
Terms of Service	Your rules, liability limits, account termination
Consent	Clear consent for data use; let users access and delete their data
Licenses	Only use images, fonts, and code you're allowed to (Unsplash, Google Fonts, MIT code)
Minors	Extra care (parental consent) if users are under 18

AI can draft these documents, but have someone qualified review them for anything commercial or sensitive (health, finance, children's data).

The launch checklist

Product

- ☐ Main flows work end-to-end (sign up → core action → log out)
- ☐ Loading, empty, and error states on every page
- ☐ Works at 360px phone width and on desktop
- ☐ Tested on a real Android phone + iPhone (or Safari)
- ☐ Forgot password and delete account work
- ☐ No placeholder text, lorem ipsum, or test data
- ☐ Favicon, app icon, splash screen, 404 page
- ☐ Emails arrive (and not in spam)

Security

- ☐ Security checklist on page 13 completed
- ☐ RLS / rules tested with two different accounts
- ☐ Production env variables set; no secrets in git
- ☐ 2FA on all your dashboards
- ☐ Backup and restore plan

Growth & legal

- ☐ Custom domain + HTTPS
- ☐ Titles, descriptions, OG image, sitemap
- ☐ Google Search Console verified
- ☐ Analytics + error monitoring (Sentry)
- ☐ Privacy policy, terms, contact page
- ☐ Feedback channel (form, email, WhatsApp)

After launch

- ☐ Watch logs and errors daily for the first week
- ☐ Fix the top 3 user complaints first
- ☐ Weekly: check costs, update dependencies
- ☐ Keep CLAUDE.md and README up to date
- ☐ Write a changelog users can read

PROMPT — copy & paste

Act as a strict QA lead. Go through this launch checklist against my project: [paste checklist]. For each item tell me PASS / FAIL / CAN'T CHECK with evidence, then fix every FAIL you can and list what I must do manually.

LAUNCH WHERE YOUR USERS ARE

College groups, WhatsApp communities, LinkedIn, Instagram reels showing the app in 15 seconds, Product Hunt, Reddit (read the sub rules), and QR posters where people physically are. A short demo video beats a long description.

Glossary & cheat sheet

Term	Plain meaning	Term	Plain meaning
API	A way for programs to talk to each other	Endpoint	One URL of an API (/api/posts)
Frontend	What users see (runs in the browser/app)	Backend	Server, database, logic users can't see
Framework	Pre-built structure (Next.js, Flutter)	Library	Reusable code you plug in (date-fns)
npm / pip	Package installers for JS / Python	package.json	List of a JS project's dependencies + scripts
Env variable	Config/secret kept outside code (.env)	Build	Turning source code into deployable files
SPA / SSR / SSG	Rendered in browser / on server per request / pre-built at deploy	Hydration	Browser making server HTML interactive
CRUD	Create, Read, Update, Delete	Migration	Versioned database change
RLS	Database rules per row/user	JWT / token	Proof you're logged in
OAuth	"Sign in with Google" style login	Webhook	A service calling your URL when something happens
CORS	Browser rule about which sites may call an API	CDN	Servers worldwide caching your files
DNS	Maps your domain to your server	SSL / HTTPS	Encrypted connection (the padlock)
PWA	Website installable like an app	Service worker	Background script for offline + push
OTA update	Updating app code without store review	MCP	Protocol letting AI use external tools
Skill	Instructions that teach Claude a task	Plugin	Bundle of skills, commands, MCP servers
Token (AI)	Chunk of text (~¾ word) AI is billed by	Context window	How much the AI can "see" at once
Embedding	Text turned into numbers for similarity search	Hallucination	AI confidently inventing things
CI/CD	Auto test + deploy on every push	Refactor	Restructure code without changing behaviour

Bookmark these

Docs: [react.dev](#) · [nextjs.org/docs](#) · [supabase.com/docs](#) · [firebase.google.com/docs](#) · [docs.expo.dev](#) · [docs.flutter.dev](#) · [developer.mozilla.org](#) (MDN) · [huggingface.co/learn](#) · [docs.claude.com](#)

Design: [github.com/VoltAgent/awesome-design-md](#) · [tasteskill.dev](#) · [mobbin.com](#) · [ui.shadcn.com](#) · [fonts.google.com](#) · [colors.co](#)

Check & test: [pagespeed.web.dev](#) · [search.google.com/search-console](#) · [securityheaders.com](#) · [caniuse.com](#)

Skills: [skills.sh](#) · `npx skills find` · `/plugin` in Claude Code

FINAL SECRET

When you don't know what to ask, ask the AI: *"I'm a vibe coder building X. What am I not thinking about that could hurt me later?"* Then put the answers on your checklist.