

# AI app building: tools and techniques

What we covered in our first session, in short. Each item: the name, then what it is or why it helps.

## Working with Claude Code

**Plan first, then build.** Ask for a plan, read it, then say "go". You get far fewer wrong turns than when it starts coding straight away.

**Describe the outcome, not the tech.** Say who the app is for and what they should be able to do. Let the agent pick the tools unless you have a real reason. Forcing a language or framework too early can send it the wrong way.

**Talk to it.** Dictating ideas by voice works well, even if it rambles. Keep an ideas note and paste it in when you start a session.

**Let it do the work.** It can run commands, create folders and set up services through their command-line tools. If it says it can't, remind it that it has access and ask it to try.

**One chat per topic.** Start a new session for anything unrelated. Long mixed chats get slower, more expensive and less accurate.

**Watch the context meter.** When the conversation fills up, it gets summarised and details can be lost. Put files and screenshots in the project folder and point the agent at them instead of pasting them into the chat.

**Idle sessions cost more to resume.** After a few minutes without activity (up to an hour), the cache expires and your next message makes it re-read the whole conversation.

**Subagents.** The main agent can hand simple jobs (searching, small edits, running tests) to helper agents on cheaper, faster models and keep the strongest model for planning and review.

**Remote Control.** Continue a Claude Code session from the Claude app on your phone, for example to capture an idea away from the computer.

**CLAUDE.md and memory.** A CLAUDE.md file holds standing instructions the agent reads every time: one global file in `~/claude/` and one per project. Memory files keep facts between sessions.

## Extending the agent

**Skills.** A folder with instructions (a SKILL.md file, a bit like a big reusable prompt, plus optional scripts) that the agent loads when a task matches. Turn anything you do repeatedly into a skill.

**find-skills.** A skill that searches the open skills ecosystem for one that fits what you're trying to do.

**Plugins.** Bundles of skills, commands and checks you install with `/plugin`. Examples: Superpowers (planning, debugging and testing habits), Caveman (short answers

that save tokens), Impeccable and frontend-design (better-looking interfaces, fewer "made by AI" patterns).

**MCP (Model Context Protocol).** Connects the agent to other tools and data: files, a browser, email, calendar, your own app. Add them from the connectors/settings. Keep only the ones you use, because each one takes up room in the conversation. You can ask the agent to build an MCP server for your own app.

## Building and shipping

**Web first, then phone.** Build a website that works well on phones, test it as a PWA, and wrap it as a store app later if you need one.

**GitHub.** Stores your code and its full history. Make changes on a branch and merge them with a pull request.

**Vercel.** Hosts the web app and redeploys when you push code. Free Hobby plan for personal, non-commercial projects; the Pro plan (\$20/month) is needed once you charge money.

**Supabase.** Database, sign-in (email, Google) and file storage in one. The free tier is fine for small apps, but free projects pause after a week without use. Pro is \$25/month.

**Keep each user's data separate.** Use row-level security rules so every user can only see their own data, then test it by signing in as two different users.

**PWA (Progressive Web App).** "Add to Home Screen" in the phone browser's menu turns the site into an app icon. The quickest way to test on phones and share with friends.

**Store apps.** Tools like Capacitor wrap your website into iOS and Android apps. Apple Developer costs \$99/year; Google Play \$25 once, and new personal accounts must first run a closed test with testers for two weeks.

**AI APIs.** Use the OpenAI platform (pay-as-you-go credits) or OpenRouter (one key for many models, some of them free) to put AI inside your app. This is billed separately from ChatGPT or Claude subscriptions.

**Cost guardrails.** One API key per app so you can see what each one costs; a monthly spend limit with email alerts; require sign-in for AI features; a free quota per user and a paid plan above it.

**Stripe.** Payments and subscriptions. Start in test mode with fake cards; going live needs your identity and bank details verified.

**Resend.** Email for your app (notifications, sign-in emails). The free tier covers 3,000 emails a month (100 a day). Emailing other people requires your own domain.

**Domain.** Optional: about \$10-15 a year. Every app gets a free `yourapp.vercel.app` address to start with.

## Quality and automation

**Tests.** Ask the agent to write tests for every feature and run them. Tests catch the same mistakes models tend to repeat.

**Make the agent prove it.** Have it run the app, click through the main flows, take phone-size screenshots, and tell you what it could not check.

**Avoid "AI slop".** No fake numbers or reviews, no purple gradients, one accent colour, designed empty and loading states, readable on a phone. Write these rules into a skill or `CLAUDE.md` so every app follows them.

**Copy a style you like.** Give a screenshot or a website address (or use a design skill) and ask for the same colours, fonts and spacing. Take the style, never the logos or text.

**CI test gate.** GitHub Actions runs your tests on every pull request. Only merge when it's green.

**AI code review.** CodeRabbit (a GitHub app, free for public repos) reviews each pull request; an agent can then fix what it finds in a follow-up pull request.

**Tiered auto-merge.** Small, low-risk changes merge automatically once the tests pass. Bigger ones get a second review and your OK.

**Self-healing loop.** The app logs its errors; an agent looks into each one, writes a fix with a test, the checks run, and low-risk fixes merge.

**Regular project review.** Now and then, have the strongest model review the whole project: missing tests, structure, duplicated code, AI slop.

**Always-on agents and scheduled jobs.** Agents that run on a spare computer or server (for example OpenClaw or Hermes Agent), or scheduled tasks that check things every day. Give them only the access they need and a spending cap.

## Staying safe

**API keys.** Keep them in `.env` files or your host's settings, never in your code or on GitHub. Don't paste them into chats, screenshots or recordings.

**Security as you grow.** Automated checks plus your own spot checks are enough to start. Once you have many users or sensitive data, have a security professional review the app.