

The five shapes of a Claude Code build

Anything you ask Claude Code to build will likely take one of these five main forms. Find yours before you start, and every decision after that gets easier.



Ten questions to ask yourself first

Answer them in plain words, in a note. That note becomes your brief for Claude Code.

1 What problem does it solve, and for whom?

"My assistant spends two hours a week copying invoices into a spreadsheet."

2 What goes in?

Files you already have, things you'll type, photos, data from another service.

3 Does it need to remember anything between uses?

Nothing, a few days, or years of records that must never be lost.

4 Who uses it besides you?

Nobody, one partner, your team, your clients, the public. Each step up adds work.

5 Where will you use it?

At your desk, on your phone, on both. Be honest about the phone part.

Five more, and your brief is done

6 Does it work while your computer is off?

If it must watch, remind, or receive things at 3am, it needs to live somewhere that never sleeps.

7 Does it connect to tools you already use?

Gmail, Calendar, Sheets, WhatsApp, your bank. Each connection is real work; name them all.

8 How bad is it if it breaks or leaks?

Annoying is fine to learn on. Money, legal, medical or client data deserves a second pair of eyes.

9 What can it cost per month?

Zero is possible for many personal tools. Sharing with others usually means a small hosting bill.

10 Does it have to look good, or just work?

A tool for you can be ugly. A thing clients see is part of your brand.



Local tool

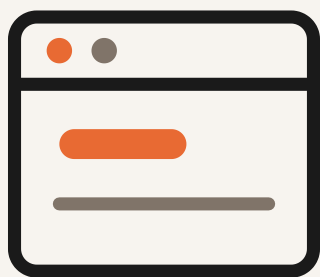
Lives on your own computer, works on your own files. No internet, no login, no monthly bill. The simplest shape by far.

How you recognize it: often it has no screens at all; you run it and the result appears as a file. It only works while you're there. It can be ugly, because nobody else ever sees it.

For example: renames 500 photos by date · turns a folder of scanned invoices into one clean spreadsheet · searches all your old proposals for the paragraph you need · finds duplicates in your contact list · turns voice memos into organized text notes.

START THE CONVERSATION LIKE THIS

"Let's build a local tool. I have a folder of 500 scanned invoices, and I want one spreadsheet with date, vendor and amount. Everything stays on my machine."



Web app

Opens from a link, on any device. Save it to a phone's home screen and it feels like an app. The right answer most of the time.

How you recognize it: it has an interface, it lives at an address, and more than one device can reach it. If others use it, it needs a login.

For example: a tracker you and your partner both update · a portal where each client sees their own documents · a booking page for your workshop.

Where the data lives: not necessarily a "real" database. A Google Sheet works for small things. Sturdier options exist; it depends on lifespan, people, and monthly cost.

START THE CONVERSATION LIKE THIS

"Let's build a web app that my two partners will also use. Before any code, propose where the data should live, what it costs monthly, and which accounts I need."



Phone app

The kind you install from the App Store. It's the shape people picture first, and the most work: developer accounts, Apple's review, waiting.

How you recognize you need it: the phone itself is the point. It must buzz you, run in the background, or talk to the phone's hardware. If it just shows information, a web app does it with a tenth of the effort.

For example: a reminder that buzzes when you arrive somewhere · reading a Bluetooth sensor · tracking a run with the screen off · scanning documents with the camera, offline, in the field.

Rule of thumb: prove the idea as a web app first. Move here only when the web version hits a wall.

START THE CONVERSATION LIKE THIS

"I think this needs to be a real phone app, because it has to buzz me at a location. Tell me honestly if a web app could do it. If not, plan the build and explain each step."



Background service

No screen and no buttons. It wakes up on a schedule or when something happens, does one job, and tells you. You forget it exists; that's the point.

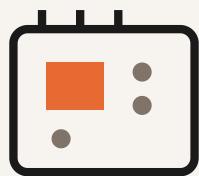
How you recognize it: nobody ever "opens" it. Its interface is your inbox or your phone's notifications. Its two big questions are when it should wake up, and where it lives so it never sleeps.

For example: watches a government page and emails you when a rule changes · files every invoice that lands in your inbox · a summary of yesterday's sales at 7am · alerts you when a competitor changes prices.

One catch: to run with your laptop closed it needs a home in the cloud. Usually a few dollars a month, not hundreds.

START THE CONVERSATION LIKE THIS

"Let's build a background service that checks this page every morning and emails me if anything changed. It must run with my laptop closed; tell me where it will live and what that costs."



Hardware

Software that touches the physical world, through a small computer you can hold. A Raspberry Pi (about \$60) is the usual one.

How you recognize it: the information you want does not exist online; something must sense it, count it, or show it in the room. The code lives on the device, and the device needs power, wifi, and a place to sit.

For example: a screen in the kitchen with today's schedule · a sensor that texts you when the greenhouse gets too hot · one button that logs "the machine jammed again" · a light that turns red when your calendar says do not disturb.

Less scary than it sounds: photograph your wiring and Claude checks it like a senior engineer looking over your shoulder.

START THE CONVERSATION LIKE THIS

"Let's build this on a Raspberry Pi. Here's what I want it to sense and show. Give me the shopping list first, then we wire it up together, checking photos as we go."

THE SHORTCUT

Which shape is yours?

Only you, at your computer, on your files

Local tool

Anyone you choose, from a link, on any device

Web app

It must buzz your pocket or use the phone's hardware

Phone app

Nobody watches it work; it runs and reports

Background service

It measures or shows something in the room

Hardware

When two fit, take the simpler one. Bigger things are combinations: shapes snapped together.

The words Claude will use

Eight words cover most of the confusion. None of them is complicated.

Interface

What you see and click.

Database

Where the facts are kept so they survive a restart. Can be as humble as a spreadsheet or a single file.

Authentication

Proving who you are. A login.

Authorization

What each person may see or change once they're in. If clients each see their own documents, this is the wall between them.

Four more, and you're fluent

Host

The computer that keeps your thing running when yours is closed. You rent one for a few dollars a month.

Integration

A connection to something you already use: your calendar, your email, a weather feed, WhatsApp.

Deploy

Move it from "works on my machine" to "runs where the right people can reach it."

Secrets

Keys and passwords. They live in the host's locked settings. Never in the code, never in a screenshot.

Write three small files before you build

Plain text, in your own words, in the project folder. Claude reads them every session, so the project survives closed conversations, changed tools, and three months of not looking.

project.md

Your answers to the ten questions. Who it's for, what goes in, who sees it, what it may cost.

decisions.md

Every choice that touched money, privacy or your data: the options you were given, what you picked, and why.

scenarios.md

Real situations it must handle, in your words: "I photograph a receipt and the right record appears." These become your tests.

PASTE THIS INTO YOUR CLAUDE.MD

Make Claude explain before it chooses

The difference between a build that goes well and one that goes sideways is whether choices get explained before they get made.

Before any choice that affects cost, privacy, my data, or my ability to move this later: stop and explain it in plain English. Give me two or three real options, recommend one, and tell me what becomes harder to change afterwards.

Ask before anything public, costly, or irreversible. Keep keys and passwords out of the code.

When you say it works, show it running against my scenarios. I test it myself before we call it done.

No model can tell you it works. Only you can.

Test it yourself, on the device where it will really be used, against the situations you wrote down in scenarios.md.

Want to scope what's in your drawer together?

cal.com/santiago-restrepo/60min

Santiago Restrepo · AI Sherpa · one useful thing, roughly weekly