



The AI Coding **Stack**

A practical guide to building iOS apps with AI — from design to database to deployment. Walks through the tools and workflow in the order you'd actually move through them: design the UI, decide on your database, pick your tech stack, and choose the AI tools that write and automate your code.

WHO THIS GUIDE IS FOR

Beginners — if you've never built an app before, each section explains what a tool is and why it's in the stack, not just its name.

Technical readers — if you already know your way around a codebase, you'll find the specific tool choices, trade-offs, and reasoning for each decision so you can adapt the stack to your own project.

01–02 · Design & Data

UI design tools that hand off to AI coding, and the database options that fit an app just getting started.

03–04 · Build & Automate

The framework to build in, and the AI tools — coding assistants and agents — that write and run the code.



01 · UI DESIGN

From idea to **screens**

The goal in this stage is to go from an idea to a set of screens you can hand off to an AI coding tool, like Claude Code, to turn into real, working app code.

Figma Design

The industry-standard design tool for building app interfaces — buttons, layouts, colors, fonts, built just like any professional app screen.

Figma Make

Generates wireframes from a simple prompt or rough idea — no design skills required. The fastest way from "I have an idea" to a visual layout.

Mobbin + Mobbin MCP

A searchable library of real, live app screens — proven UI patterns for onboarding, checkout, settings. Its MCP connects that library directly to Claude Code (page 4).

Pinterest

General visual inspiration — color palettes, layout ideas, branding direction — especially useful in the earliest, most exploratory phase.

Claude Design

Anthropic's design tool for generating and refining UI concepts directly with Claude — useful for exploring layout and style options alongside your other reference sources.

WHY IT MATTERS FOR AI CODING

Whether it's a **Claude Design** link, a Figma file, or a Mobbin pattern connected via an **MCP** (Model Context Protocol — a connector that lets an agent fetch from an outside service itself, explained on page 4), giving Claude Code a direct reference lets it "see" the real layout, spacing and colors, and generate code that matches instead of guessing.

Putting it together

STEP 1**Collect references**

Gather UI inspiration and proven patterns from Mobbin and Pinterest.

**STEP 2****Design**

Feed references into Claude Design or Figma Make, use Mobbin MCP directly with Claude Code, or build your own wireframes in Figma.

**STEP 3****Build**

Hand the wireframes to Claude Code — a Claude Design link, a Figma file, or a connected Mobbin MCP — to generate the real app code.



01 · UI DESIGN

Give your agent **eyes**

A coding agent that only sees your written description of a screen is guessing at everything it can't see. Handing it an actual UI reference — a Claude Design link, Figma file, a screenshot, a Mobbin pattern — replaces guesswork with ground truth.

- 1 Removes ambiguity.** "Make it clean and modern" means nothing precise to an agent. A reference file fixes exact spacing, color values, type sizes, and component structure so there's nothing left to interpret.
- 2 Keeps the app visually consistent.** Without a shared reference, an agent tends to reinvent styling choices screen by screen. Pointing it at the same design file or pattern library keeps buttons, spacing, and colors consistent across every screen it generates.
- 3 Cuts down revision loops.** A vague prompt produces a vague first draft, which then takes several rounds of "no, more like this" to correct. Starting from a concrete reference gets the first pass much closer to what you actually want.
- 4 Gives the agent real UI vocabulary.** Reference patterns from Mobbin or a Figma component library show the agent proven, real-world solutions to common UI problems (onboarding, empty states, settings) instead of it improvising a layout from scratch.

WAYS TO GIVE A REFERENCE

Connect a live design file with the **Figma MCP**, paste in a screenshot or exported frame directly, link a **Mobbin** pattern via its MCP, or point the agent at an existing screen in the codebase to match. Any of these beats a text-only description.



What an MCP actually **is**

MCP stands for **Model Context Protocol** — an open standard for connecting an AI agent to an outside tool or data source. Without one, you paste things into the chat by hand. With one, the agent can go and fetch what it needs itself.

- 1 **It's a connector, not a model.** An MCP server is a small piece of software that exposes one service — Figma, Mobbin, a database — through a common interface. Claude Code speaks that interface, so any MCP server works with it without custom integration code.
- 2 **You install it once, per machine.** The commands below register the server with Claude Code and persist across projects and sessions. You authenticate in the browser the first time and don't think about it again.
- 3 **It replaces copy-paste, not judgement.** The agent can now pull the actual Figma frame or Mobbin pattern on demand — but you still decide which reference is the right one to point it at.

COPY-PASTE · TERMINAL · CONNECT THE DESIGN MCPS

```
# Mobbin — a library of real, shipped app screens
claude mcp add mobbin --scope user --transport http https://api.mobbin.com/mcp

# Figma Dev Mode — the Figma desktop app must be running
claude mcp add --transport sse figma http://127.0.0.1:3845/sse

# Check they registered
claude mcp list
```

Mobbin opens a browser window to authenticate with your Mobbin account on first use — there's no API key to paste. If `claude mcp list` shows a server as failed, the usual cause is the Figma desktop app not running, or Dev Mode's MCP server not enabled in Figma's preferences.



02 · DATABASE

Where your data **lives**

Your database is where your app's data lives — user accounts, posts, messages, whatever your app needs to store and retrieve.

Firestore — recommended default

Google's flexible, document-based database, part of Firebase. Billed pay-per-use, so you only pay for the reads, writes, and storage you actually use — a good fit for apps just starting out without predictable, high traffic.

Firebase Realtime Database

Firebase's original database, optimized for data that needs to sync instantly across users — live chat, live location, collaborative features. For most general app data, Firestore is the better default; reach for this specifically when a feature needs real-time sync.

A NOTE ON SUPABASE

Avoid Supabase unless absolutely necessary. It's a capable platform, but its pricing can escalate quickly and unpredictably as usage grows, which makes it easy to rack up a large, unexpected bill — especially for solo developers or small teams without dedicated cost monitoring. Stick with Firestore/Firebase unless you have a specific, well-understood reason Supabase is required (e.g. you need native Postgres/SQL).

NEED	USE	WHY
General app data	Firestore	Pay-per-use, flexible document model, good default
Real-time sync	Realtime Database	Optimized for instant multi-user sync
Native SQL/Postgres	Supabase	Only if you have a specific reason — watch the bill



03 · TECH STACK

Build once, ship **fast**

React Native is the recommended framework for building the app itself, rather than native Swift — and EAS handles getting it built and deployed.

Why React Native over Swift/Xcode

- 1 More flexible and builds faster.** You can iterate on features more quickly than in native Swift development.
- 2 Xcode is slow to live in — in my experience.** This one is opinion, not measurement: a clean build plus re-indexing on a mid-size project is minutes rather than seconds, and the SwiftUI preview canvas is the part most likely to stall or need a restart mid-session. React Native's Fast Refresh puts a code change on screen in about a second. You'll still need Xcode for StoreKit testing and device builds — you just won't be writing code in it all day.
- 3 Agents generate React Native more reliably than SwiftUI.** This is the one claim here with evidence behind it: *SwiftEval* (arXiv 2505.24324), a Swift-specific LLM code benchmark, was built precisely because SwiftUI is poorly represented in public training data and existing benchmarks, and cross-language evaluations consistently put Swift below the JavaScript family. React Native's component model mirrors web patterns the models are saturated in, so you get working code on the first attempt more often — and first-attempt-correct is the whole economics of building this way.

This is the canonical statement of that claim for this whole guide series — the other guides point here rather than re-asserting it.

BEGINNER NOTE

React Native lets you write your app once (mostly in JavaScript/TypeScript) and run it on both iOS and Android, instead of writing entirely separate native code for each platform.



03 · TECH STACK

Getting it shipped

Expo Application Service compiles your app on Apple-provisioned machines and handles code signing and provisioning profiles — historically the part of iOS development that ate a weekend before you'd written any code.

COPY-PASTE · TERMINAL · BUILD AND SUBMIT

```
# One-time, in your project
eas login
eas build:configure

# Build a production iOS binary (EAS handles signing)
eas build --platform ios --profile production

# Upload it to App Store Connect — a separate step
eas submit --platform ios --latest

# Useful along the way
eas build:list
eas credentials
```

- 1 **Build and submit are two commands, not one.** A green build is not a submission — `eas build` produces a binary, `eas submit` uploads it. Forgetting the second one is the most common "why isn't it in TestFlight yet?"
- 2 **You don't need a Mac for the build itself.** EAS builds remotely, which is what makes signing and provisioning someone else's problem. You'll still want a Mac for Xcode-specific work like StoreKit testing.
- 3 **Set `"autoIncrement": true` in `eas.json`.** Every upload needs a build number higher than the last for that version. Automate it once and never think about it again.

DURING DEVELOPMENT, THOUGH

EAS is for producing distributable builds, not for your day-to-day loop. While building, you run locally — `npx expo start` in Expo Go if your app has no native modules, or `npx expo run:ios --device` once it does (Firebase, RevenueCat, notifications and StoreKit all require the latter). Reach for EAS when something needs to leave your machine.



04 · AI CODING TOOLS

The tools that **write** it

These are the AI assistants that actually write your app's code, working from your designs and requirements.

TOOL	MAKER	ROLE
Claude Code	Anthropic	Primary tool referenced throughout this guide — turns designs (via Figma MCP) into working React Native code
Codex	OpenAI	Alternative / complementary coding agent
Antigravity	Google	Another option in the same category

Beginner tip: you don't need all three. Most builders pick one as their primary driver and occasionally cross-check a tricky problem with another.

Agent automation (Advanced)

For running agents on larger, more autonomous tasks — or several at once.

Warp

A terminal rebuilt around agents rather than around a shell prompt — blocks of command output the agent can read back, and multiple agent sessions side by side in one window.

Reach for it when: you're running several long agent tasks at once and losing track of which pane is doing what. Skip it if one Claude Code session in your existing terminal is keeping up.

Superset

Orchestration on top of coding agents: define a repeatable job once — its setup steps, the prompt, the repo it runs against — and have it run on a schedule or on demand in a clean workspace.

Reach for it when: you catch yourself typing the same maintenance prompt every week. It earns its keep on recurring work, not on one-off features.

Opencode

An open-source terminal coding agent that is model-agnostic — point it at whichever provider you have credits with, or at a local model.

Reach for it when: you need to self-host, or want to compare the same task across models.

Do you need any of these?

Not on your first app. All three solve the problem of running *many* agent tasks — which you only have once one agent in one terminal stops being enough.



05 · RECAP

The full stack at a **glance**

Every stage of the stack, in the order you'd move through it while building.

STAGE	TOOL(S)
UI designing and wireframing	Figma Make, Figma Design, Claude Design
UI references	Mobbin + Mobbin MCP, Pinterest
Design → code bridge	Figma MCP/Mobbin MCP/Claude Design → Claude Code
Database (default)	Firestore (pay-per-use)
Database (real-time)	Firebase Realtime Database
Database (larger scale)	Supabase — unless truly necessary
App framework	React Native
Deployment	Expo Application Service (EAS)
AI coding	Claude Code, Codex, Antigravity
Agent automation	Warp, Superset, Opencode

You now have

- 1 A path from **rough idea to real screens** using Figma Make, Figma Design, Claude Design and UI references with Mobbin and Pinterest.
- 2 A **design-to-code bridge** via the MCPs into Claude Code.
- 3 A **database choice** that won't surprise you with the bill.
- 4 A **framework and deployment pipeline** built for fast iteration and AI generation.
- 5 A set of **AI coding and automation tools** to actually write and run the code.



06 · WHERE TO GO NEXT

One guide per **stage**

This guide is the map. Each stage on it has a guide of its own that goes properly deep — read them when you reach that stage, not before.

GUIDE	PICKS UP WITH
Designing UI with AI	The design stage in depth — prompting, negative prompting, recognizing slop, accessibility with real tools
Database Setup	Schema modeling with AI, a real <code>firestore.rules</code> file, composite indexes, and wiring Firestore into React Native
Subscriptions with RevenueCat	Paywalls, entitlements, the purchase code, and testing it all before your products are approved
Launching on the App Store	Everything after the code works — accounts, EAS build/submit, the listing, and the review gotchas
Idea to MVP in 1 Day	This whole stack compressed into eight hours, for when you only need to know whether the idea is worth building
Idea to App Store	The combined guide — all seven stages end to end, including the validation and analytics stages the others don't cover

IF YOU READ ONE MORE THING

Read **Designing UI with AI** next. The design stage is where the largest quality difference between two otherwise identical apps gets decided, and it's the stage most people skip straight past on the way to writing code.