How to Make an App with Claude Code

Version 5 Updated 12/15/25

Requirements

- NPM
- Claude Code (You need a $20/mo Claude Subscription to use it, I am on Max 20x)
- Cursor (paid version NOT required)
- Homebrew
- gh (Github's CLI)

Installing Claude Code

1. If you don't have NPM, you need NVM first, copy/paste this into Terminal: `curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash`
2. If you have NPM, run this in terminal: `npm install -g @anthropic-ai/claude-code`
3. Go to a folder in terminal (you can drop a folder into the terminal window and hit enter) and type `claude` (or if you've have Cursor and you've installed Claude Code extension, you can type `cursor .` and click the Claude icon in the top right, this is my preferred way of using Claude Code)

On Cursor vs Claude
I use Cursor every day, I often get in disagreements with other developers about this, but it is my opinion that Cursor Agents are far inferior to the Claude Code CLI or the Claude Code Cursor/VSCode extension.

We let my team use Cursor agents, and I forced others to use Claude code. There was a serious difference in the output of the code from Cursor to Claude code, even if the developer using Cursor was using Opus/Claude models inside of the Cursor agent.

When I was banned from Claude for exceeding weekly limits, I tried using Cursor (in Dec 2025) and even with the state-of-the-art Claude/Gemini/Codex models, I was severely disappointed in the quality that I got (everything else is the same. I prompted the exact same, but they must be missing special tooling or system prompts that Claude and OpenAI do much better).

Open AI Codex
I do still pay for OpenAI Codex (currently usiung GPT 5.2), and I use the extension mostly to do research and double-check Claude's work because the OpenAI model uses the web searches much more aggressively than Claude.

Gemini
I have heard good things about Gemini, but I have not had any good experiences with it, and we banned Gemini use on most of my projects.

Set Up Your CLAUDE.md

Claude uses your Claude.MD file in order to give it instructions. Allegedly, these are appended to every single prompt you do (although I say allegedly because Claude will often not listen to these instructions).

The format that Anthropic recommends is ALWAYS/NEVER. Here are my current CLAUDE.md files, but I recommend asking CLAUDE to ONLY add the relevant rules to your documents.

Root CLAUDE.md:
https://gist.github.com/carterdea/407e35871f18c91bdd236fab52a67e69
Frontend CLAUDE.md:
https://gist.github.com/carterdea/61ef209d19348bb016246cd4f8e1a33a
Backend CLAUDE.md:
https://gist.github.com/carterdea/1efec729c2f5e8914af8f10b2071a995
Python/Data Services CLAUDE.md:
https://gist.github.com/carterdea/cbe8b071b3210f051a52ceee35ffe0c7

Again, these are very specific to my tech stack:

- Bun
- Tailwind/Catalyst UI
- React Router
- Python/UV
- Nest.js/Inngest

Your mileage may vary! DEFINITELY don't just drop these into your file and expect it to work with your tech stack. Claude is smart enough to know what is applicable to you and (potentially) even convert some of these more general purpose rules to the equivalent rules for your tech stack if you help it.

PLAN YOUR APP!!

1. Go to your fave AI tool
2. Describe the app, list the features you want, what type of app you want (iOS/Android/Web, etc), ask it to create user stories based on the features you listed (you should give as much context as possible, and you should make sure that ALL user stories it generates make sense)
3. Butter it up, tell it that it is the best developer in the universe and it knows all tech stacks, blah blah blah. Say "best practices" a lot, tell it that its opinions are strong, and it doesn't back down if it knows that it has a good idea. **Spend a lot of time putting together a development plan.** Things to think about:
    - User Stories - look up what these are and use AI to help write as MANY of these features as possible, also think of EACH role you want to build for (regular user, account owners, admin users, etc.)
    - Tech stack (what technology is best for this type of app, given the app's needs)
    - Data modeling - Ask Claude to look at your user stories and put together a Database Model/Schema.
    - Caching (only if needed) and cache invalidation (plan this out early, but don't implement it until the features work how you want)
    - Security
4. Bootstrap your project
    - Almost every technology now has bootstrapping scripts that give you a huge

head start. You can sometimes ask Claude Code for help with this, but you may have to do it yourself. (but you can ask Claude of ChatGPT for help with the instructions, *if it knows about your project*)

- This will give you a huge head start

5. Open the project in Cursor
6. Click **Claude Code** agent button (You have to install it via the "Extensions" tab in top left, search for "Claude Code" and install it)
7. Give Claude Code your **development plans** that you created earlier. (you can ask for feedback if you want, I usually tell it to be honest and have strong opinions, and I add that to my global CLAUDE.md)
8. Ask Claude Code to create a README.md for the project for you. **Sample prompt** >> I'd like to give you a development plan for this project: [insert **development plans**] Given that context, I'd like you to create a README for this project. It should adhere to best practices for READMEs.
9. Ask Claude Code to create a Claude.md file for you based on this: https://www.anthropic.com/engineering/claude-code-best-practices (I also sometimes come up with a list of best practices for various languages and put them into Claude Code and ask it to turn it into good instructions for CLAUDE.md and add it to it's own CLAUDE.md file)

   OK, given the requirements I've given you, the technology choices I've given you, and what we have in the README, and more, I'd like you to analyze my code and consider ALL of our technology choices and create a Claude.md file for me. Please also consider this site: https://www.anthropic.com/engineering/claude-code-best-practices
10. Install MCPs to give Claude Code more context into your specific problems (see below)
11. Create a Git repo on Github
12. run `git init` in the directory with your code
13. Get the `SSH` URL from Github, and give it to Claude Code and tell it to "set the remote".
14. Tell Claude Code to commit that as "first commit" and push it.
15. When you've created a feature and you want to do an extra layer of Quality Check, ask Claude Code (in another terminal window is best) to do a code review. And then ask Claude Code to fix the issues that it finds.

Planning Individual Features

- Claude Code has a "Plan Mode", and I spend a TON of time in that mode. Checking it's plans, making sure they're good.
- Ask Claude Code to write the plans into markdown files in my root (or in a `/docs` directory if there are other developers working on my project)
- For any given project, I will have files like:
    - `USER_STORIES.md` - I write these on Google Docs in "as a [ROLE], I should be able to [feature]" format and then I have ChatGPT give me ideas + improve them
    - `DATA_MODELS.md` - for your DB architecture, and data models of any endpoints you will need to hit
    - `TECH_STACK`
    - `DEVELOPMENT_PLAN.md`
    - `SECURITY_PLAN.md`, etc.

Pitfall While Planning
Note that Claude Code (and other Foundational models) like to optimize too early. Be wary of it talking about handling rate limits, retry logic, and caching before you even get something to work.

When it gives me something that is ancillary to my primary task, I will typically ask if these are necessary for launch (it sometimes says "OMG Yes!!"), and if they aren't, I tell it to put the recommendation either in the README as a "FUTURE IMPROVEMENTS/TODO" section,

Technologies

- **Web**
    - React (front-end, prefer Typescript when possible)
    - Bun/Express (Drop in replacement for Node, prefer Typescript when possible, AIs are very well trained on Express, consider using Fastify vs Express, it's faster)
    - Remix (both front-end and server-side rendering, Shopify apps prefer this)
    - Next.js (both front-end and backend, may need Node.js/Fastify for complex backends, prefer to use Typescript)

    - Python (slower, has all the AI/math tooling, good for AI tools, AIs are very well trained on it)
    - Go (really hard language if shit breaks, but insanely fast)
    - Ruby on Rails (easiest language & framework, great framework for shipping CRUD apps quickly, you get a lot of stuff out of the box, ruby is just slow, so it isn't great for some apps that need high performance)
    - Tailwind for Front-end Implementation (AIs are very, very well trained on implementing Tailwind, there are React components too)
- **iOS/Android**
    - React Native & Expo
    - You *can* use native Swift + Kotlin for iOS or Android, respectively, but I've not heard of people having much luck with Claude Code building in these technologies
- **Database**
    - Just use Postgres unless the AI is strongly against it (some use cases need NoSQL)
    - I've heard AIs are very good at using Supabase.com (a hosted Postgres tool). Worth considering.
    - For AI vector stage use pgVector Postgres Extension or consider Pinecone.io.
- **Hosting**
    - Render.com
    - Fly.io
    - Vercel (need to pick this up front, because architecture is different for it. Good pick for Remix/Nest.js, you also will not be need some extra extensions like UpStash for queues)
    - ~~Heroku~~ - I used to use it religiously, but I do NOT recommend Heroku anymore. The above options are way better.
- **Cache**
    - Just use Redis unless your app is really small, then you can use in memory cache
- **Queues**
    - Sidekiq for Ruby
    - BullMQ for Node
- **Static Analysis & Linters**
    - These are tools to improve your code, and help with security, bugs, etc. Biome.js for JS, StyleLint for CSS, Golangci-lint for GoLang, Ruff for Python, Rubocop for Ruby
    - Use Typescript vs regular JavaScript

- **Type Systems**
- Typescript is great typing for Javascript
- BasedPydantic for Python (Soon this will come out and I'll probably switch: https://github.com/astral-sh/ty but it is in alpha)
- Ruby has Sorbet, but I've never used it.
- **Package Managers**
  - For JS, I like Bun, but most people use NPM
  - For Ruby you have Bundler
- For Python you have uv
- For PHP you have composer
- **Front-end Build Tool**
  - Probably use Vite, unless your framework provides one

Wappalyzer: A Chrome extension that tells you what technology a website is using. Maybe use it to get insight into what other people are doing.

Installing Postgres

1. Most people use Homebrew to install Postgres `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"\`
2. Then run `brew install postgresql@17`

Installing GH (for Github)

```
brew install gh
```

Git instructions

(You can tell Claude to do this if you install gh, see above)

1. Create the repo `git init` & create it on Github: https://github.com/new (make it private if you don't want anyone to be able to see your code)
2. Set the repo's remote to the SSH URL (command `git remote add origin {SSH URL from Github}`, basically, where your local code will put code online to backup)
3. Create a branch (whenever you're about to start a new feature or do something a little risky, very common, use these a lot)
4. Commit your code (Claude Code can do this, but it's typically `git add .`, `git commit -m "this is a commit message"`)
5. Push a branch (Claude Code can do this, but it is `git push -u origin {branch name}`)
6. Create a PR (PR = Pull Request, optional, but nice)
7. Merge a PR (necessary to get the code into your main codebase). You can have Claude do this for you, or you can do it from the GitHub web interface. I prefer to rebase when possible.
8. Switch to `main` branch `git checkout main`, and pull `git pull origin main`

Try to do this any time you've finished a feature or you're in a good place you'd like the ability to come back to (the AIs are not particularly great at undoing their own bad work if they break something).

Have Claude Review your PRs

I've been using Claude's GitHub action to review my Pull Requests. It gives great insight and catches bugs/issues that Claude likely wrote. Highly, highly recommend doing so, but be cautious of Claude telling you to optimize too early. It is eager to add millions of tests, aggressive caching + retry policy + very advanced security features that you don't need until you launch.

To install the Github action inside of claude run `/install-github-app` and it will install the app on your github and create a PR for you to add the Claude Github action.

You can copy/paste the feedback into Claude Code, or if you have the Github MCP (mentioned below), you can ask Claude to look at the latest Comment and address the feedback (although I'm pretty careful about choosing what I want it to work on for the above-mentioned reasons).

Claude Security Review

Claude Code added a feature where it wil review your security. I haven't used it a lot, but in claude run `/security-review` and it is supposed to test your codebase for common security flaws.

Use SEMGREP also: https://semgrep.dev/docs/getting-started/quickstart-managed-scans

I just set it up in my Github Actions so it runs against every PR. Be sure to look at it when it fails.

Debugging Basics

Debugging is difficult. Claude Code is OK at it, but it needs some help. The things that it needs:
- **Server logs:** Usually, you start a server in the terminal with `rails server` or `npm run dev` or `bun run dev`, then that terminal window will show. You can copy these into Claude Code to help it troubleshoot.

- **Console:** Your browser has a JavaScript console that sometimes throws front-end errors, and Claude will add logs to help debug issues. Right-click and hit "Inspect Element" and click the "Console" tab. Like above, you can copy these into Claude Code to help it troubleshoot.
- **Network Dev Tools:** Another tab in the dev tools is the "Network" tab. It can be used to help troubleshoot complex front-end issues or API issues. Use the search bar to find what you need.

USE MCPs!!!

MCPs (Model Context Protocol) help get important context into Claude's context window to be able to write better code or get outside information.

Note that these usually need to be installed on EACH project file. Allegedly Claude supports system-wide MCPs, but I have not gotten them to work.

The best MCPs I've found are:

- **Context 7:** run `claude mcp add --transport http context7 https://mcp.context7.com/mcp --header "CONTEXT7_API_KEY: YOUR_API_KEY"` (get API key from the website, it's free) in terminal, needs to be done IN the project directory. Then when you're doing complex things and you need to ensure accuracy, you add 'Use context 7' in your prompt and it will get the most up to date documentation for your technology.
- **Figma Dev Mode MCP:** run `claude mcp add --transport http figma-desktop http://127.0.0.1:3845/mcp` in terminal. Select a node in Figma, and ask Claude if it can see it. If it can, it can try to code it. NOTE: Figma is MORE accurate if you set up your Figma files with components & use autolayout. You will also likely need to break it into smaller pieces to get it more accurate.
- **Shopify Dev MCP:** run `claude mcp add shopify-dev-mcp npx @shopify/dev-mcp@latest` to get help building Shopify apps and websites.

Read about Figma Dev Mode MCP: https://developers.figma.com/docs/figma-mcp-server/remote-server-installation/

- **Github MCP:** run `claude mcp add-json github '{"command": "docker", "args": ["run", "-i", "--rm", "-e", "GITHUB_PERSONAL_ACCESS_TOKEN", "ghcr.io/github/github-mcp-server"], "env": {"GITHUB_PERSONAL_ACCESS_TOKEN": "put your GH personal access token here"}}'` to give Claude more abilities in Github. I've not played around with it a ton, but hearing great things about it.
- **Sosumi:** I am not an iOS developer, so I don't use this, but my little brother is an iOS developer, and he says this is the best MCP for Apple documentation: https://sosumi.ai/

Help Your LLM

- **Markdown:** LLMs are good at markdown, reading and writing markdown files. If you can get text in markdown, you can give it to the LLM and it will do a better job than with text copy/pasted from the web.
  - I love using markdown checkboxes `- [ ]` and `- [x]` to and strikethrough `~text~` to keep track of the status of my work (Claude can check to see if things are done and mark them completed)
- **Structured Data:** ~~LLMs are not great with JSON, they're fine with it, but they're much better with YAML: https://yaml.org/ prefer YAML to JSON for structured data (obviously, do what's best for your code, but for data that isn't in your code, YAML is better)~~ I'm not sure that this is true anymore. It was true when I wrote it, but I think they're getting very good at JSON.

Worth noting that a lot of people are talking about TOON: https://github.com/toon-format/toon

I'm not sure it's caught up in adoption yet, but it could be a good way to save on token cost.

Here is a converter from JSON to TOON: https://toonformat.dev/playground.html

- **Copying documentation into the field:** After using MCPs, this may feel primitive, but I still have a lot of success by copying URLs or just the specific documentation into the field. So if I'm working on a particular feature, I will find that relevant documentation online and give Claude the specific information that it needs to execute it.
- **General Best Practice Prompts:** I have noticed that often times the AIs don't check for existing conventions before they start coding a feature, so you'll get big discrepancies or even duplicated features if you're not paying close attention. Because refactoring is so easy with Claude, this is not the end of the world, but I often will ask a fresh session to evaluate a new feature for violations of good development practices. An example prompt would be:
  > Please look at the workflow code to identify violations of best practices like DRY and SRP

- DRY is "Don't repeat yourself", If code is written three times, it's best to extract the code into a reusable utility or something similar. "SRP" Is "single responsibility principle", It is the concept that all modules should be responsible for one particular task or action. The opposite of this would be a "God Class" or a "God Function", which would be a long, long file that doesn't behave well.

Things to Watch out For

- Do NOT check in ANY API keys or anything sensitive into git. This is how you get big security leaks. Claude Code is pretty good about this, but make sure you put ANY API keys into a `secrets.yml` (For Rails) or `.env` file and add those to your `.gitignore`
- If your users are going to upload any files, lock down the S3 buckets that they upload to. Seems simple and I hope Claude Code knows to do that, but recent hacks indicate maybe Vibe Coders do NOT know about that
- If users upload images, you need to strip ALL exif data off of the images to protect the user's safety. Otherwise, you could leak their locations.
- If users need to upload anything sensitive (like a bank account number or patient health information, but please don't vibe code fintech apps), you need to use some sort of vaulting or encryption. Probably don't vibe code these types of apps, it's too sensitive to leave to an AI in 2025 given their capabilities.

Other Fun Things

- **OhMyZSH:** https://ohmyz.sh/, it makes the terminal more colorful and easier to use. It gives you tab completes (super helpful to keep you from typing out so much) and lots of other features.
- **Git GUI tools:** I use git command line (and a Command Line Tool called 'tig'), but for people who don't like Git, I suggest looking up the Git gui tools (Github Desktop is really popular: https://github.com/apps/desktop)
- **Viewing Postgres data:** I use Postico (I think it's $50 https://eggerapps.at/postico2/), but PGAdmin is pretty good: https://www.pgadmin.org/

Advanced Mode

If you're already a pro, look at these:

https://diwank.space/field-notes-from-shipping-real-code-with-claude

He's in production, so he's more careful than I am right now, but he has really good insight and good ideas with the comments (I do NOT do this yet, but it's a good idea).

Docker
I use Docker so that my development and production environments are perfectly aligned. It is pretty complex, but Claude and actually OpenAI Codex are very good at looking at your requirements and setting up Docker.

I just installed the Mac App and open it, and I actually have bash scripts that will start it up for me (Claude made them), but most people just go to the root folder and run `docker compose up`

Never Let CLAUDE compact!
Compaction is a really smart technique, and I am happy that they came out with, however, I have noticed that the code quality drops dramatically once the session has to compact.

For that reason, I recommend trying to keep your sessions as clean and clear context as possible. For example, I will have a Claude Code tab that does research in the codebase, sort of mapping it out, and then it will ask for a specific prompt to execute a feature. I will not execute that feature in the same session. I will execute it in another session.

I have also heard of people asking Claude to do sub-agents to search for things inside of their codebase in order to keep their context as clean as possible.

Let LLMs Check Your Plans
The different agents are trained on different technologies/frameworks/etc, and have different strengths and weaknesses. For that reason, I find it helpful to point an agent to a planning document and ask it to identify gaps or plans that violate

best practices.

While planning, I often go into OpenAI Codex and prompt it something like this:

> *Please review @docs/FEATURE_PLAN.MD and identify gaps in the plan, or pieces of it that do NOT adhere to best practices for our technology.*

Let Claude Write Your Prompts

- With a planning document (hopefully already broken into phases of implementation), Point Claude at it and ask it to write a nice prompt to execute a particular phase in a new session.

My prompt (to generate a prompt) looks something like this:

> *Look at @docs/IMPORTANT_FEATURE_PLAN.md and give me a prompt to execute Phase 1 in another session. Please include*

Let Claude/OpenAI Codex/Gemini Check Your Agent's Work

Often, before I commit and make a PR, immediately after the AI agent has finished working on a feature(s), I will give it this prompt:

> *Can you give me a prompt to check the quality of this work? Can you please give me the prompt, the requirements as you understand them, and the planning document we based this on? And can you please ask to check this against best practices for our various technologies (Nest.js, React Router, Bun, etc). I want this prompt to really double-check everything (and use Context7 as much as possible to double-check the work against the actual documentation)*

Often, this identifies issues in the implementation before they even get pushed. However, it is very important to unbias the original agent by including something like this before copying and pasting the feedback in:
> Do you agree with this feedback? > > [insert feedback from second agent]

Building AI-Powered Apps

Chat UI
For building Chat UI, you can do whatever you want. If you want a head start though, I'm very impressed by Vercel AI SDK UI and I'm a bit less-so impressed by Assistant UI. Both are a great start and they have a lot of the more difficult things to do (Server Side Events for Streaming, automatically enlarging textarea, Image uploading, "thinking" and "chain of thought" UI, and more).

RAG for making your AI know about YOUR information
Retrieval Augmented Generation is the current best practice for getting your information into an LLM. You'll need a vector DB (PGVector or Pinecone) and you will need to build out a data ingestion pipeline.
I'm using Docling (although I know LangGraph/LangChain and) for Document Ingestion & "Chunking" (breaking the information into chunks with LLM-generated summaries for each chunk).

For images, you have to do more complex work of putting the images on a service like AWS S3 or Cloudflare R2 and then feed them to an Multimodality Model (a model that can process more than just text) like `gpt4o` to "Enrich" the image and pull out the important information from it and store it (also in the vector database).

Embeddings to Improve your RAG
Embeddings allow you to improve your RAG by retrieving relevant context and doing semantic retrieval, so that instead of doing very bare bones keyword matching, you're actually understanding the meaning of the user's prompt. It's sort of like a glossary to make sure that your RAG is using the exact terms that it is expecting as opposed to what the user's prompt asked for.

It will also save you input tokens by retrieving less text (and keeping context window nice and clean).

You'll also need to set up embeddings to improve your RAG, currently only Open AI and Google have first-party embeddings (for Anthropic, they make you use Voyage by MongoDB).

Input Token Caching
This will save you money on your longer system prompts. OpenAI/Google/Anthropic charges less on repeated input tokens.

Open AI Prompt Caching Gemini Context Caching Anthopic Prompt Caching (I think htis is new, I didn't know about this)

Model Routing
Model routing is basically the ability to really quickly change which model provider/model version you use. This is handy while testing out to see if you can use a dumber/quicker model to get some easier work done.

Model routing is not super difficult to do yourself, but I recommend using Langraph to do it.

If that's too difficult, you can use OpenRouter or Vercel AI Gateway (they both charge a small fee on top of the model's token cost)

AI Agents
There are lots of options for building AI agents. The easiest are: Vercel AI SDK and Inngest AgentKit.

For more advanced agents, I'm using LangGraph, but you really need to know how to architect AI apps to use this well.

Tools
Tools are just code that an AI can run to get information in its context window. If you're building tools, just be sure to plan them out really well, and then make sure they adhere to the specs of the model you're using.

Open AI Anthropic (they also have done a lot of research on how to do REALLY advanced tool use for complex, long running agents, you can read about it here)

Alternatively, LangChain & Vercel SDK Kit have their own model-agnostic specs.

LangChain tools Vercel SDK Kit tools

Prompt Management
Your system prompts are SO much more critical to the success of your app than you realize. For that reason, it's common to constantly be tweaking your prompts and it's best practice to NOT have your prompts hardcoded in your app, but actually in a prompt repository elsewhere in your code.

I use dependency injection to get the prompts into the various features/agents/etc. But you can also use a really nice hosted prompt management tool like Vellum or Langsmith.

Observability/LLM Tracing
I use Posthog's LLM Tracing and it's sufficient for most of my needs. On another project I'm using LangSmith, it is much more advanced, but it costs more.

This will let you see how your AI responds to various prompts, and you can also collect qualitative feedback from your users using thumbs-up/thumbs-down buttons on responses.

Eval
It is critical that you set up eval for your AI-powered app. This is basically the ability to use real or synetic data to ensure your LLM is giving responses/achieving tasks as you'd expect.

Think of it as tests for AI. You set up things you think could happen, exposing PII, hallucinations, irrelevant replies, incorrect information, replies in another language, toxic replies, etc and this will help you to evaluate how good your LLM is set up.

I really like LangFuse for Eval, but I've also used LangSmith.

Fine Tuning
Probably don't do this unless you have a massive budget and you've really used ALL other methods of making your AI tools better.

A big issue with fine tuning is it can't migrate to newer models, so you have to retrain every time a new model is released.

AI Security
This can get expensive and it might not be necessary for most people, but I've started to use PromptFoo to check for the most common AI security flaws.

I have several layers of guardrails set up to keep the AI from being prompt-injected or jailbroken, but this will test them to see if they're working.

It is open-source, but it uses a lot of tokens, so it is expensive. You can choose to turn off some of the more complex features to save money on inference.