DAY SEVEN

07 /10

The Claude Developer Platform — *Claude inside your own software*

The apps and Claude Code are finished products. The API is the raw material: a way to put Claude inside software you build yourself. Today is the conceptual map of that surface — what it is, what a request looks like, and how to build on it responsibly.

BY THE END OF DAY 7, YOU WILL BE ABLE TO TEACH OTHERS TO —

- Say when the API is the right surface
- Read a first request in curl or Python
- Name the major capabilities built on it
- Describe the shape of a Messages request
- Explain that the API is stateless
- List the responsibilities of building with it

01 Why today matters

ORIENTATION

Everything so far has been a product someone else built — the chat apps, Claude Code. The API is the layer underneath: a direct line to the model that developers use to build their own products. A support tool that drafts replies, a feature that summarises documents in your app, an internal agent — all of it is the API, wrapped in software.

You do not need to write production code to teach this day well. You need the map: what the API is, the shape of a request, and — most importantly — what it means to build responsibly on top of it. Teach it conceptually; the curl example is there to make it concrete, not to turn the room into engineers.

A NOTE ON THE LAB

Today's lab is partly a paper exercise — many learners will not have API keys, and that is fine. The goal is fluency with the concepts, so that when an engineer joins the conversation, the non-engineer can still follow it.

02 When the API is the right surface

CORE CONCEPT

Day 10 covers the full "which surface" map, but the principle for the API is clear enough to teach now: you reach for the API when Claude needs to be part of something else.

> THE MENTAL MODEL
>
> **Ingredient, not dish.**
>
> The chat apps and Claude Code are finished dishes — you use them as they are. The API is an ingredient: it does nothing on its own, but it lets you cook something that did not exist before. If the need is "I want to use Claude," that is the apps. If it is "I want my software to use Claude," that is the API.

SIGNS THE API IS THE ANSWER

- ✓ The capability needs to live inside an existing product or workflow
- ✓ It must run many times, automatically, without a person prompting each one
- ✓ It needs to be wired to your own systems, data, or interface
- ✓ You are building something for other people to use, not using it yourself

03 The shape of a request

CORE CONCEPT

The core of the API is the Messages endpoint. Strip away the syntax and a request is just a few things: which model, how long the answer may be, an optional set of standing instructions, and the conversation so far.

3.1 A first request, in Python

```
import anthropic

client = anthropic.Anthropic()

message = client.messages.create(
    model="claude-opus-4-7",
    max_tokens=1024,
    messages=[
        {"role": "user", "content": "Hello, Claude"}
    ],
)
print(message.content[0].text)
```

That is a complete, working call. The same request in curl is a POST to the Messages endpoint with three headers — your API key, the content type, and an API version — and the same fields as JSON in the body.

3.2 The pieces that matter

- `model` — which Claude you want, by its model string (Day 1's family of models).
- `max_tokens` — a ceiling on the length of the response.
- `messages` — the conversation, as a list of user and assistant turns. The first turn is always the user.
- `system` — optional; the persistent instruction layer from Day 4, as its own field.

3.3 The API is stateless

This is the one fact most likely to surprise newcomers. The API remembers nothing between calls. To continue a conversation, you send the whole history back each time. The chat apps feel like they remember because the app is doing that work for you — storing the turns and resending them. On the raw API, that bookkeeping is yours.

> **KEY IDEA TO INSTALL**
>
> A Messages request is just model + length limit + standing instructions + the conversation. And the API forgets everything between calls — continuity is something the builder provides, not something the API keeps.

04 What you can build on it

CORE CONCEPT

The basic request is a foundation. A set of capabilities sits on top of it — each one a door to a different kind of product.

- **Tool use.** Claude can be given tools — functions it may call — so it can fetch data or take actions, not just talk. This is the engine under agents.
- **Streaming.** The response can arrive token by token, so an interface can show it forming rather than waiting for the whole thing.
- **Vision.** Requests can include images, not just text — Claude can read a chart, a screenshot, a scanned page.
- **Batch processing.** Large volumes of requests can be sent to be processed asynchronously, at a substantially reduced cost — ideal for bulk jobs that are not time-sensitive.
- **Prompt caching.** Repeated context can be cached so it is not reprocessed every call — cheaper and faster for workloads that reuse the same large preamble.
- **Extended thinking.** The Day 4 idea — giving Claude room to reason before answering — available to build on directly.

> *"The basic request is the foundation. Tools, streaming, vision and batch are the doors it opens."*

05 Building on it responsibly

CORE CONCEPT

Building with Claude carries responsibilities that simply using a finished app does not. Teach these as the price of admission, not as fine print.

- **Guard your keys.** An API key is a credential that spends real money. It belongs in a secure configuration — never in code you share, never in a public repository, never in the browser.
- **Mind the cost.** Usage is billed per token, input and output. At scale that adds up. Batch processing and prompt caching exist partly to manage it; so does simply estimating before you ship.
- **Evaluate before you trust.** A feature that works in three test cases is not a feature that works. Build a set of real examples and check the output against them — and keep checking as you change things.
- **Respect the limits.** The API enforces rate and spend limits, organised into tiers. Design for them rather than being surprised by them.
- **Carry the earlier lessons in.** Clear prompts, structure, grounding, human review — none of that stops mattering because the prompt is now coming from code. If anything it matters more, because no one is reading each response by hand.

> **THE BUILDER'S SHIFT**
>
> In the apps, a human reads every answer. In an API product, the software acts on the answer automatically — so the quality bar, the testing, and the safety thinking all have to move into the build. That shift is the real subject of today.

LAB 07 ~35 MIN

Design a feature, trace a request

A largely conceptual lab. Each learner designs one small API-powered feature and traces a single request and response through it — no keys required.

1. **Pick a real need** in your own work that "my software using Claude" would solve — a drafting step, a summary, a classification.
2. **Decide it is genuinely the API.** Check it against Section 02's signs. Could the apps do it instead? If so, say why the API still fits — or change the example.
3. **Sketch the request.** Write out, in plain terms: which model, roughly what the system instructions say, what one user message looks like.
4. **Trace the response.** Describe what comes back and what your software does with it next — and where a person, if anywhere, reviews it.
5. **Name two responsibilities** from Section 05 that this feature would have to take seriously, and how.

a learner who can describe an API-powered feature end to end — and explain why the responsibility for quality sits in the build, not in a human reading each reply.

TEACHING NOTES

How to teach Day 7 well

OPEN WITH THIS

Ask: “When you use a banking app, do you think about the bank’s systems underneath?” You do not — but someone built that layer. The API is that layer for Claude. Today is about being able to think one level down, even if you never write the code.

PACE & EMPHASIS

Two ideas must land: the request shape (03) and “stateless” (03.3), and the responsibility shift (05). The capability list (04) is a tour, not a deep dive — name the doors, do not walk through every room. Do not let the curl example intimidate non-coders; read it aloud as plain English.

DISCUSSION PROMPTS

· What in your work is “I want to use Claude” versus “I want my software to use Claude”? · Why does statelessness change how you would design a multi-turn feature? · Which Section 05 responsibility would your team be weakest at, honestly?

COMMON MISCONCEPTIONS TO PRE-EMPT

“The API is just the chat app for programmers.”
No — it is an ingredient with no interface. You build the product; it builds nothing on its own.

“It remembers the conversation like the app does.”
It is stateless. The app’s memory is the app resending history — on the raw API that is your job.

“If it works in my tests, it works.”
A handful of cases is not evaluation. Real examples, checked repeatedly, are the bar.

IF YOU ONLY HAVE 30 MINUTES Teach “ingredient not dish” (02), the request shape and statelessness (03), and the responsibility shift (05). Skip the capability tour or name it in one breath. Do Lab steps 1–3 verbally as a group.

Day 7 Cheat Sheet

The API	A direct line to the model that developers use to build their own Claude-powered products.
Ingredient, not dish	It has no interface of its own. “I want my software to use Claude” → the API.
Messages endpoint	The core of the API — a POST with model, length limit, optional system, and the conversation.
The request fields	`model` · `max_tokens` · `messages` (user/assistant turns, user first) · `system` (optional standing instructions).
Stateless	The API remembers nothing between calls — you resend the whole conversation each time.
Capabilities	Tool use, streaming, vision, batch processing, prompt caching, extended thinking.
Guard your keys	An API key spends real money — secure config only, never shared or public.
Cost	Billed per token, in and out. Batch and caching help manage it; so does estimating first.
Evaluate	Working in three cases is not working. Real examples, checked and re-checked.

Check for understanding

Five questions. Learners should be able to answer all five before Day 8.

1. “Ingredient, not dish” — what does that say about when to choose the API over the apps?
2. Name the four things that make up a basic Messages request.
3. What does it mean that the API is stateless, and how do the chat apps feel like they remember?
4. Name three capabilities built on top of the basic request.
5. Why does the responsibility for quality move “into the build” when you use the API?

> **Answer notes —** 1) The API has no interface of its own — you choose it when your software needs to use Claude, not when a person does. 2) model, max_tokens, messages (the conversation), and optionally system. 3) It remembers nothing between calls, so you resend the full history each time; the apps store and resend the turns for you, which feels like memory. 4) Any three: tool use, streaming, vision, batch processing, prompt caching, extended thinking. 5) The software acts on each response automatically with no human reading it, so testing, the quality bar, and safety thinking all have to be built in.

Day 7 in five lines

- The API is the layer beneath the apps — a way to put Claude inside software you build yourself.
- Choose it when your software needs Claude, not when a person does: ingredient, not finished dish.
- A Messages request is model, length limit, optional standing instructions, and the conversation.
- The API is stateless — continuity is something the builder provides, not something it keeps.
- Building responsibly means guarding keys, watching cost, evaluating output, and carrying every earlier lesson in.

TOMORROW — DAY 8 → Connecting Claude to Everything — MCP, connectors, and agents