AI MADE EZ · LEVEL 4 SUMMARY

Build with Claude

8 lessons · about 52 minutes in all · free

01 Your first order by code **02** Pick the chef by job **03** Same plate every time **04** Wire the appliances
05 Prep once, serve many **06** The recipe library, built **07** Taste tests in code **08** Opening night

01 Your first order by code.

- Your own shop or app can call Claude directly.
- Four fields do most of the work.
- Every plate comes with a receipt.

Console = where you get keys and try requests
Playground = test a request, then copy its code
Key lives in ANTHROPIC_API_KEY, never in code
Request = model · max_tokens · messages · system
Model ID = the exact name, like claude-opus-5-5

Say it: “I made my first call: model, max_tokens, messages, and the key stays in an environment variable.”

02 Pick the chef by job.

- Start on Opus. Switch when tests say so.
- Prices are per million tokens.
- Effort sets how hard it thinks.

Haiku 5.5: fast, cheap, simple high-volume jobs
Sonnet 5.5: balanced · Opus 5.5: the usual default
Fable 5.2: the hardest, longest work
MTok = a million tokens; output costs 5x input
Effort = low · medium · high · xhigh · max

Say it: “We start on Opus 5.5, then test Haiku, Sonnet and lower effort on real tasks.”

03 Same plate every time.

- Code needs the same shape every time.
- One setting locks the shape.
- Still check the plate.

JSON Schema = the plate diagram: fields and types
output_config.format = lock the final answer’s shape
No extra fields allowed (your developer sets this)
Strict tool use = tool inputs match the schema
Validation = still check it in code

Say it: “When code reads the reply, we give Claude a JSON schema and still validate the result.”

04 Wire the appliances.

- The loop: ask, run, return, repeat.
- stop_reason “tool_use” means your code’s move.
- Some appliances Anthropic runs for you.

tools = name · description · input_schema
stop_reason tool_use = your code’s turn
tool_use block = Claude’s order, with an id
tool_result block = your answer, same id
Parallel calls = answer all, in one message

Say it: “Our code runs the loop: Claude orders a tool, we run it and send the result.”

06 The recipe library, built.

- Small library? Hand Claude the whole book.
- Big library? Fetch only the right pages.
- Make every answer show its page.

Files API = upload once, reuse by file_id
Document block = hand Claude a PDF or text
Small library: send it whole
Big library: chunk, embed, retrieve, cite
Voyage AI = embeddings (Anthropic has none)

Say it: “Small libraries go in whole with citations. Big ones get retrieval: embeddings plus BM25, reranked, cited.”

08 Opening night.

- Some errors mean wait. Some mean fix it.
- Wait longer each time you retry.
- Cap the bill before the rush.

429 = slow down; wait retry-after seconds
429 with no retry-after = maybe the tier spend cap
400 “specified usage limits” = your own spend limit
529 overloaded = backoff: longer waits, plus jitter
Other 400 · 401 · 404 = fix it, don’t retry

Say it: “Before launch: a spend limit, keys on the server, retries with backoff, and request IDs logged.”

05 Prep once, serve many.

- Caching reuses the same start of your prompt.
- One changed letter spoils the prep.
- Batches trade speed for half price.

cache_control = mark the end of the shared part
Order: tools, system, messages; stable first
Cache TTL = 5 min default, 1 hour costs more
Cached reads = a small fraction of normal input
Cache hit = cache_read_input_tokens above 0

Say it: “We cache the shared start of each prompt and batch overnight jobs at half price.”

07 Taste tests in code.

- An eval has three parts.
- Use the cheapest grader that measures the real thing.
- Score cost and speed too.

Eval = cases + a way to run + a way to grade
Code grader = exact match, schema check: free
Pairwise comparison = judge picks the better of two
Rubric = what a judge scores one answer on
Baseline = today’s results, saved · Pass rate = share that pass

Say it: “Every change reruns the same cases against the baseline, logging cost and time.”

Prompts to try

Change the parts in [brackets] to fit you. One a day is plenty.

01 · Your first order by code.

ORDER #001 copy · paste · send

I don’t code yet. Walk me through a first Claude API call to write a product description.
1. Plain words: the Console, API keys, environment variables, how keys leak.
2. The request: model, max_tokens, system, messages.
3. A labeled sample reply: content blocks, stop_reason, usage.
4. Then write a product description for me (ask me for details), as my app would get it back.
5. End with a 5-line brief I could send a developer.

02 · Pick the chef by job.

ORDER #002 copy · paste · send

Pick a Claude model for: sort support emails into 5 labels. What matters most: lowest cost. I don’t code yet.
1. Compare Haiku 5.5, Sonnet 5.5, Opus 5.5 and Fable 5.2 with current prices (say if they come from memory).
2. Recommend one model and effort level.
3. What test result means I switch.
End with a 10-row test sheet for my job (input, expected answer) as a table I can paste into a spreadsheet.

03 · Same plate every time.

ORDER #003 copy · paste · send

My app will pull data from receipts with the Claude API. I don’t code yet.
1. 4 to 6 fields, each with a type, in a table.
2. Structured outputs or a strict tool: which fits?
3. Three ways it can still go wrong, and how to check each.
End with the finished JSON Schema as one block, a note on each field, and 3 sample inputs to test it on.

04 · Wire the appliances.

ORDER #004 copy · paste · send

I’m planning a café ordering bot on the Claude API with one tool: “check what’s in stock”. I don’t code yet.
1. Write the tool definition and explain each part.
2. Walk one example through the loop, showing where stop_reason changes.
3. Parallel tool calls, and which tools Anthropic runs versus my code.
End with a tool spec to hand a developer: name, when to use it, inputs, output, 3 test questions.

05 · Prep once, serve many.

06 · The recipe library, built.

ORDER #005 copy · paste · send

```
My app sends a long product manual
on every Claude API call, to answer
questions live. I don't code yet.
1. Prompt caching, plainly, and 3
things that break it.
2. Do Message Batches fit? (Most
finish within an hour; unfinished
requests expire after 24 hours.)
End with a cost table for 1,000
requests, cached vs. not, that I can
paste into a spreadsheet (say if
prices come from memory).
```

ORDER #006 copy · paste · send

```
I want the Claude API to answer
questions from a staff handbook
(under 100 pages). I don't code yet.
1. Whole documents or retrieval, for
my size? Why?
2. If retrieval, each step in plain
words: chunk, embed, store, search,
rerank, answer with citations.
End with a document-prep checklist I
can tick (gather, clean up, leave
out), then 5 test questions, each
with the page its answer is on, plus
one the documents can't answer.
```

07 · Taste tests in code.

ORDER #007 copy · paste · send

```
Plan a test for an email sorter on
the Claude API. Its output: a label
from a list. I don't code yet.
1. Recommend a grader: code check,
pairwise comparison, rubric judge or
a person.
2. What to record per case; how to
rerun it before every change.
End with 10 test cases (3 tricky, 1
to flag) as a table I can paste into
a spreadsheet: input, expected, how
to grade.
```

08 · Opening night.

ORDER #008 copy · paste · send

```
I'm about to launch a customer chat
bot on the Claude API. My biggest
worry: surprise bills. I don't code
yet.
1. A small table: which errors to
retry, which to fix.
2. The 3 checks that matter most for
my worry.
End with a plain-words launch
checklist (keys, spend limits,
retries, streaming, logging, evals)
as boxes I can tick, one owner per
box.
```

> You can build a small Claude app and talk to developers as an equal.

Made by Alexandra Vargas · not affiliated with Anthropic