WINDOWS, MACOS OR LINUX · VERSION 1.0 · 7 NOVEMBER 2026

BUILD AN AI THAT READS YOUR RECEIPTS

A goal-driven build prompt for your coding agent, plus the mistakes and tests that turn a ”read this photo” demo into spending records you can trust.



WHAT IT TEACHES — Multimodal AI · Structured output · Validation in code

01 One build prompt you paste once

02 The mistakes that break this build

03 Acceptance tests to prove it works

04 A repair prompt for when it doesn’t

BEFORE YOU START

START HERE

01

Create a new folder called My Receipt Tracker. Inside it, create a folder called sample_receipts and add 10 receipt photos: some clear, some crumpled, one long supermarket receipt, one in a different layout, one blurry.

02

For three of them, type the correct merchant, date and total into a note. These are your answer key.

03

Open a coding agent that can create files and run commands on your computer (for example Codex or Claude Code). Copy both parts of the build prompt into it as one message and send it. The agent works through every stage and stops only when it needs you.

04

Run the acceptance tests before you rely on the totals. Keep the mistakes page and repair prompt for later.

WHAT YOU NEED

A computer with at least 8 GB of RAM (16 GB is better for local vision models), a few GB of free disk space, an internet connection for setup, a coding agent with access to that computer, and Ollama to run a vision model locally. A cloud vision API also works; the prompt lets the agent choose with you.

WHAT YOU ARE BUILDING

Upload or drop a receipt photo. The AI reads the merchant, date, items, prices and total. You check the result next to the photo, correct anything wrong, and save. Each receipt becomes rows in a spreadsheet with a category. A dashboard shows spending by month and category.

KEEP THE SCOPE SMALL

One user, one currency to start, a fixed list of categories. The AI reads; ordinary code checks, stores and adds up. Get one receipt right, then ten, then the dashboard.

THE BUILD PROMPT · PART 1

PASTE BOTH PARTS AS ONE MESSAGE

Copy part 1 and part 2 into your coding agent as one message.

BUILD PROMPT · PART 1 COPY INTO YOUR AGENT

GOAL

Build a personal receipt tracker in this project folder. I add a photo of a receipt; a vision-capable AI model extracts the merchant, date, line items, quantities, prices, tax and total into a fixed structure; I review and correct it beside the photo; the app saves it to a spreadsheet and shows my spending by month and category.

DONE MEANS

On my sample receipts, the merchant, date and total are correct for every clear receipt. Unreadable fields are left empty and flagged, never invented. Every total is checked by code against the item sum. Nothing reaches the spreadsheet until I approve it. The dashboard totals match the spreadsheet exactly.

HOW TO WORK

Work through every stage below without waiting for me. Verify each stage with a real test before you start the next. Stop only when you need something only I can do: choose local or cloud, check an extraction against a photo, or approve a change. Explain each handoff in plain English. Keep a TEST_LOG.md with every check, its result and the exact error if it failed.

STAGE 1: INSPECT, CHOOSE AND PREPARE

Inspect my operating system, CPU, RAM, GPU and free disk space. Record them. Create an isolated Python environment for this project only.

HANDOFF: explain the trade-off between a local vision model in Ollama (private, free to run, slower, less accurate on hard photos) and a cloud vision API (faster, often more accurate, costs per receipt, sends images off my computer). Recommend one for my hardware. Wait for my choice.

If I choose local, check Ollama is installed and pull a vision model from the Ollama library that fits my RAM and supports structured output; record which and why. If I choose cloud, ask me for the API key, store it in a local .env file that is excluded from any sharing, and never print it. Record versions.

STAGE 2: DEFINE THE STRUCTURE FIRST

Before any AI call, define the receipt schema in code: merchant, date (ISO format), currency, line items (description, quantity, unit price, line total), subtotal, tax, total, payment method, and a confidence or "unreadable" flag per field. Amounts are decimals stored in the smallest currency unit or as Decimal, never floats. Unknown values are null.

Ask me for my default currency, my date format (day-first or month-first), and confirm this category list or let me edit it: groceries, eating out, transport, fuel, utilities, health, shopping, entertainment, other.

STAGE 3: EXTRACT ONE RECEIPT

Prepare each image before the model sees it: fix rotation from the photo's EXIF data, resize very large photos, keep the original file. Send the image with a prompt that asks for the schema and pass the schema as the required output format. Instruct the model to copy text exactly as printed and to return null for anything it cannot read.

Validate the response against the schema in code. If it fails validation, retry once with the error message, then mark the receipt for manual entry. Run my three answer-key receipts and report field-by-field accuracy.

THE BUILD PROMPT · PART 2

CONTINUE THE SAME MESSAGE

Continue the same message.

BUILD PROMPT · PART 2 COPY INTO YOUR AGENT

STAGE 4: LET CODE DO THE CHECKING

After extraction, ordinary code checks the result. It adds up the line totals and compares them with the subtotal and total, allowing for tax and rounding. It checks the date is valid and not in the future. It checks for duplicates by merchant, date and total. Any failed check becomes a visible warning on the review screen. The model never does the maths.

Assign a category with the model from my fixed list only, plus a rule file I can edit (for example, a merchant name always maps to a category). Rules win over the model.

STAGE 5: REVIEW BEFORE SAVE

Build a simple local web page. Left: the receipt photo, zoomable. Right: the extracted fields, editable, with warnings highlighted. Buttons: Save, Skip, Re-read. Nothing is written to the spreadsheet until I press Save. Record whether I corrected each field, so we can measure accuracy over time.

Save approved receipts to a CSV (UTF-8, opens correctly in Excel and Google Sheets) or an SQLite database with a CSV export: one row per line item, plus a receipt ID, the image file name and the category. Store images and data in a visible folder outside the build files.

STAGE 6: DASHBOARD AND PROOF

Add a dashboard: spending per month, per category, top merchants, and a list of receipts with warnings. Calculate every number in code from the saved data.

Process all ten sample receipts. Report: fields extracted correctly without correction, fields corrected, receipts that needed manual entry, and average time per receipt. Run every acceptance test in this pack. Mark each PASS, FAIL or NOT RUN in TEST_LOG.md. Repair failures before you add any feature.

DELIVER

The command or launcher I use to start the app, the project folder, the data folder, the schema, the category rules file, the model or API used and its version, a requirements lock file, a short README and the test log. List which checks still need my eyes. No telemetry.

WHEN SOMETHING BREAKS

MISTAKES TO AVOID

Give this page to the agent when something goes wrong.

ASKING FOR "THE DETAILS" INSTEAD OF A FIXED STRUCTURE

Free-text answers change shape every time and break your spreadsheet. Define the schema first and require the model to return it.

LETTING THE MODEL INVENT WHAT IT CANNOT READ

A faded total becomes a confident wrong number. Allow null, flag it, and ask you.

LETTING THE MODEL DO ARITHMETIC

Models misadd. Code adds up the items and compares them with the printed total. This split (AI interprets, code calculates and stores) is the most important design rule in this project.

SAVING WITHOUT REVIEW

One wrong total poisons a month of reports. Show the photo beside the fields and save only after approval.

STORING MONEY AS FLOATS

0.1 + 0.2 is not exactly 0.3 in floating point. Use decimals or whole minor units, and store the currency separately.

AMBIGUOUS DATES

03/04 is March or April depending on where the receipt came from. Set your date format and convert to ISO dates.

SIDEWAYS AND HUGE PHOTOS

Phones store rotation as metadata. Fix rotation and resize before extraction, or accuracy and speed drop.

CATEGORIES THAT DRIFT

Let the model invent categories and you get "Food", "food & drink" and "Groceries" for the same shop. Use a fixed list and merchant rules.

TESTING ONLY CLEAN RECEIPTS

Your real receipts are crumpled, faded and long. Test with those.

LEAKING YOUR API KEY

Keep keys in a local .env file, never in code, screenshots or shared folders.

PROVE IT WORKS

ACCEPTANCE TESTS

Use your own receipts.

T01	CLEAN START	Start the app from the launcher. Expect the upload page and the model ready.	PASS · FAIL · NOT RUN
T02	ANSWER KEY	Process the three answer-key receipts. Expect correct merchant, date and total on all clear ones.	PASS · FAIL · NOT RUN
T03	SCHEMA	Inspect the saved output. Expect every field in the schema, with null for unknowns and no extra fields.	PASS · FAIL · NOT RUN
T04	UNREADABLE FIELD	Process the blurry receipt. Expect empty, flagged fields, not invented values.	PASS · FAIL · NOT RUN
T05	MATHS CHECK	Edit one item price on the review screen so the sum no longer matches. Expect a visible warning.	PASS · FAIL · NOT RUN
T06	LONG RECEIPT	Process the long supermarket receipt. Expect every line item and a matching total.	PASS · FAIL · NOT RUN
T07	ROTATION	Upload a sideways photo. Expect the same result as the upright one.	PASS · FAIL · NOT RUN
T08	DUPLICATE	Upload the same receipt twice. Expect a duplicate warning.	PASS · FAIL · NOT RUN
T09	SPREADSHEET	Open the CSV in Excel or Google Sheets. Expect correct currency symbols, dates and one row per item.	PASS · FAIL · NOT RUN
T10	DASHBOARD	Add up one category by hand from the spreadsheet. Expect the dashboard to show the same number.	PASS · FAIL · NOT RUN

For every test record: date, app version, steps, expected result, actual result, PASS / FAIL / NOT RUN.

FIX IT · LEARN FROM IT

REPAIR PROMPT AND EXPERIMENTS

USE THIS REPAIR PROMPT

REPAIR PROMPT — COPY INTO YOUR AGENT

The failed test is [test ID].

I did: [exact steps and the receipt file name].

Expected: [field values or result].

Actual: [what came back, or the exact error].

Find whether the failure is in image preparation, extraction, validation, categorisation or storage. Make the smallest repair and rerun this test plus the answer-key receipts. Keep my saved data and images. Update the test log.

EXPERIMENTS THAT TEACH YOU HOW MULTIMODAL AI WORKS

Change one thing at a time and rerun your answer-key receipts.

01 Remove the schema and ask for "the receipt details". Compare the shape of three answers.

02 Send the same photo at full size and at a quarter size. Watch accuracy and speed.

03 Ask the model to calculate the total itself, then compare with code. This shows why code does the maths.

04 Try a second vision model on the hardest receipt. Note which fields each one gets wrong.

KEEP THIS WITH THE PROJECT

SOURCES, LIMITS AND BUILD RECORD

SOURCES

Ollama
https://ollama.com

Ollama structured outputs (works with vision models)
https://ollama.com/blog/structured-outputs

Ollama vision models
https://ollama.com/search?c=vision

LIMITS

This pack is an agent-assisted build brief; run the tests on your own computer. AI extraction makes mistakes, especially on faded or handwritten receipts; the review step exists for that reason. This is a personal tracker, not accounting or tax software. A cloud API sends receipt images off your computer and charges per request; check its privacy terms. Coding-agent access has its own cost.

YOUR BUILD RECORD

Operating system / RAM / GPU _____

Local model or cloud API / version _____

Currency / date format _____

Answer-key accuracy _____

Data folder _____

Receipts processed in first week _____

BUILD SOMETHING. THEN UNDERSTAND HOW IT WORKS.

Change one thing at a time. Switch a model. Change an instruction. Measure how long something takes. Read the error. That is where the learning happens.

ALL FIVE STARTER PACKS · FREE · NO EMAIL NEEDED

01	AUDIOBOOK GENERATOR	Text-to-speech · Text preparation · Long-job engineering
02	OFFLINE DICTATION APP	Speech recognition · Local inference · Desktop packaging
03	PERSONAL KNOWLEDGE SEARCH	Embeddings · Semantic search · Retrieval (RAG)
04	AI RECEIPT TRACKER	Multimodal AI · Structured output · Validation in code
05	AI FILE ORGANISER	AI agents · Tool calling · Safe permissions

Get all five at drtolu.com/resources/guides-and-prompts