EXPENSE WORKFLOW

AI Expense Tracker

AI-assisted expense capture and reimbursement workflow

A lightweight automation that turns a natural-language expense submission into a structured reimbursement record using AI interpretation, deterministic formatting and spreadsheet automation.

ROLE

Workflow architecture, AI integration, data transformation, testing

STACK

Google Forms, Zapier, OpenAI API, Google Sheets

STATUS

Working Prototype

Expense Submission → AI Interpretation → Field Separation → Structured Record

The Problem

Expense reporting is simple in theory, but much of the work falls on the employee. Receipts must be saved, expenses remembered and information manually transferred into structured reimbursement fields.

The project began with a simple question: **Could an employee describe an expense naturally and let automation handle the structured data entry?**

That question became more practical when I reviewed the expense report used by a frequent business traveler. I decided the prototype should capture payment method, business purpose, reimbursement status and receipt documentation.

Rather than recreate the spreadsheet as another manual form, I wanted to reduce structured entry for the employee while still producing the record the business needed.

DESIGN GOAL

Make submitting an expense easier without making the resulting expense record less useful.

CASE STUDY

The Approach

I separated the workflow into two jobs: make expense submission easy for the employee, then transform that submission into the structured information needed downstream.

Google Forms provided a lightweight intake layer. Instead of asking the user to manually populate every spreadsheet field, the form could capture a natural-language expense description along with supporting information such as payment method, business purpose and an optional receipt.

The submission then passed through Zapier to the OpenAI API, where the unstructured expense description could be interpreted and converted into consistent fields such as vendor, category and amount.

From there, deterministic automation handled the rest. Zapier formatted and routed the extracted values into the appropriate Google Sheets columns while preserving the employee's original submission for reference.

Key Design Decision: Use AI where interpretation is useful. Use deterministic automation where consistency matters.

CASE STUDY

The Solution

The finished prototype turns a simple expense submission into a structured record without requiring the employee to enter the same information manually into a spreadsheet.

The workflow follows five stages:

Expense Submission → AI Interpretation → Field Separation → Data Formatting → Structured Record

The resulting expense record organizes the information into three groups:

Groups

EXPENSE

What did the employee purchase, where and for how much?

CONTEXT

How was the expense paid and what was its business purpose?

RECORD

What information does the reimbursement record need to capture?

The result is a cleaner separation between what the employee needs to provide and what the system can organize automatically.

Structuring the AI Output

The first end-to-end version worked, but the spreadsheet exposed an important problem.

The OpenAI model could interpret the expense correctly, but Zapier initially received the response as one block of text. When I mapped that output to Vendor, Category, Amount and the other spreadsheet columns, the same complete AI response appeared repeatedly instead of each value landing in its own field.

The problem was not the expense interpretation. It was the handoff between AI and the deterministic workflow.

I changed the AI response to follow a predictable structure, with each value returned in a fixed order and separated by a consistent delimiter:

Vendor | Category | Amount | Payment Method | Business Purpose | Reimbursable | Notes

Formatter by Zapier could then split that response into individual segments. Each segment was mapped independently to its corresponding spreadsheet column.

Unstructured Submission → Structured AI Response → Split Values → Individual Fields

This turned the AI response from something a person could read into data the rest of the workflow could reliably use.

Key Design Decision: Treat the AI output as an interface between systems, not simply as a written response.

SYSTEM
ARCHITECTURE

How the System Works

A visual overview of how a simple expense submission becomes interpreted, separated, formatted and stored as a structured expense record.



Debugging the Data Flow

Once the basic workflow was connected, I tested it end to end with real form submissions rather than relying only on the setup screens.

That exposed several issues that were easy to miss while configuring individual steps:

1. **Repeated AI output:** The full OpenAI response initially appeared in multiple spreadsheet columns instead of separate values.
2. **Output structure:** The response needed a predictable field order and delimiter before Zapier could separate it reliably.
3. **Stale test data:** After changing the OpenAI instructions, Zapier continued using an earlier test response until I explicitly retested the API step.
4. **Hidden configuration:** Zapier's current Formatter interface placed some controls, including custom segment values, behind secondary options that were not immediately visible.
5. **Field alignment:** The first separated output was structurally correct, but several values landed in the wrong columns. I traced each segment back to its source and corrected the mappings.
6. **End-to-end validation:** After the mappings were corrected, I submitted another expense and verified the resulting spreadsheet row rather than assuming the individual steps would work together.

FAIL-SAFE PRINCIPLE

Validate each handoff, then test the complete workflow through the same path a real user will take.

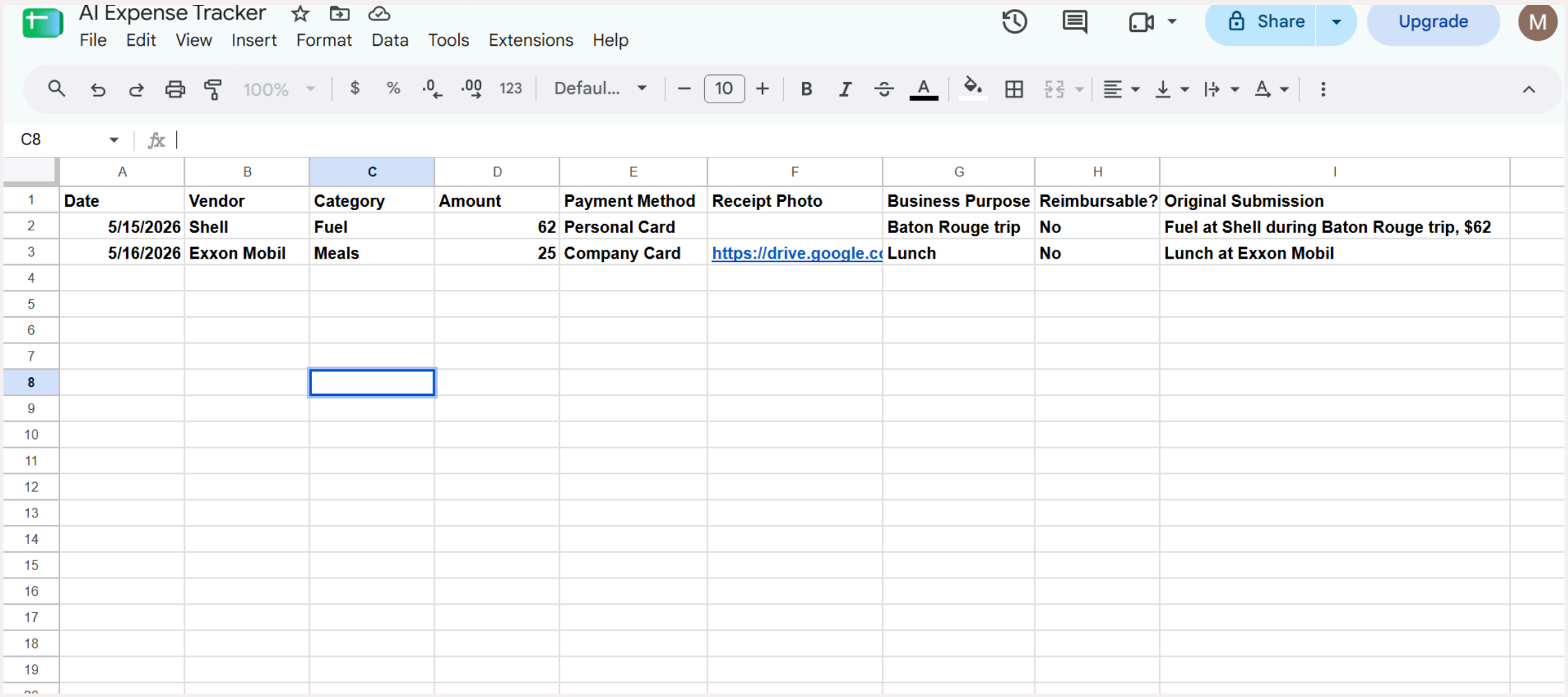
SYSTEM
EVIDENCE

Structured Output in Practice

The completed workflow turned an informal expense submission into a structured record, with each extracted value mapped to its intended spreadsheet field.

STRUCTURED OUTPUT
Date · Vendor · Category · Amount · Payment Method · Business Purpose · Reimbursable

SOURCE EVIDENCE
Optional Receipt · Original Submission



	A	B	C	D	E	F	G	H	I
1	Date	Vendor	Category	Amount	Payment Method	Receipt Photo	Business Purpose	Reimbursable?	Original Submission
2	5/15/2026	Shell	Fuel	62	Personal Card		Baton Rouge trip	No	Fuel at Shell during Baton Rouge trip, $62
3	5/16/2026	Exxon Mobil	Meals	25	Company Card	https://drive.google.cc	Lunch	No	Lunch at Exxon Mobil
4									
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									
16									
17									
18									
19									

Handling the Details

Getting the expense fields into the correct columns solved the core data-flow problem, but end-to-end testing exposed several smaller details that still affected the quality of the finished record.

Optional receipts needed to remain optional
The form allowed an expense to be submitted without a receipt, but an incorrect field mapping initially populated the Receipt Photo column anyway. I traced the value back to its source and remapped the column directly to the Google Forms upload field so it remains blank when no receipt is provided.

Dates needed their own transformation
The Google Forms submission time arrived as a timestamp rather than a clean expense-report date. Changing the display format in Google Sheets alone did not solve the problem, so I added a Date/Time Formatter step before the spreadsheet action.

Original input needed to remain visible
The employee's expense description is stored alongside the structured fields rather than discarded after processing. This preserves a simple reference point if an extracted value ever needs to be checked.

Together, these changes reinforced a broader implementation principle:

A workflow is not finished when the automation runs. It is finished when the resulting data behaves the way the user expects.

Where AI Fits

The OpenAI API has a specific role in this workflow: interpreting an expense written in ordinary language and turning it into predictable information the automation can use.

That distinction matters because the rest of the workflow does not require AI.

Google Forms handles intake. Zapier moves data between systems. Formatter separates and normalizes values. Google Sheets stores the resulting record. Once the expense has been interpreted, these steps benefit from predictable rules rather than additional AI decisions.

The architecture therefore separates interpretation from execution:

Natural-Language Expense → OpenAI API Interpretation → Deterministic Transformation → Structured Record

This also makes the workflow easier to evolve. The input method can change without redesigning every downstream step.

A future receipt-first version could use a vision-capable model to interpret an uploaded receipt and extract available information such as vendor, date and total. Those values could then enter the same deterministic transformation and storage workflow already established by the prototype.

AI DESIGN PRINCIPLE

Use AI to interpret ambiguity. Use automation to handle what should remain predictable.

The Result

The finished prototype is a working AI-assisted expense workflow that turns a simple employee submission into an organized spreadsheet record.

The completed system includes:

- Lightweight expense intake through Google Forms
- Optional receipt upload
- Expense interpretation through the OpenAI API
- Predictable AI output structured for downstream automation
- Field separation and mapping through Formatter by Zapier
- Date normalization before spreadsheet entry
- Structured Google Sheets output
- Preservation of the original employee submission
- Blank receipt handling when no image is provided

The workflow was tested end to end using new form submissions rather than only testing individual automation steps. The final tests confirmed that extracted values reached their intended columns and that optional and formatted fields behaved correctly.

I then demonstrated the working prototype to the business traveler whose existing expense process helped shape the project. He responded positively to the concept and planned to show it to his boss.

His work situation subsequently changed when he was assigned to help establish a sister company, so the expense workflow did not move into organizational adoption.

Final architecture:

Expense Submission → OpenAI API → Field Separation → Data Formatting → Structured Expense Record

The result remains a working prototype rather than a deployed expense system, but the real-user demonstration provided useful validation that the workflow addressed a recognizable administrative problem.

What I Learned

The biggest lesson from this project was that connecting applications is only part of workflow automation. The harder problem is making sure the data moving between those applications remains predictable and usable.

The first workflow technically ran, but the result was not useful because the OpenAI response arrived as one block of text. Solving that required me to think about the boundaries between systems: what one step produces, what the next step expects and how to create a reliable structure between them.

The project reinforced several principles I want to carry into future automation work:

- **Test the complete workflow:** Individual steps can succeed while the end-to-end result is still wrong.
- **Inspect the data at each handoff:** Understanding what each system is actually receiving makes troubleshooting much more deliberate.
- **Preserve the source:** Keeping the original submission alongside AI-generated fields creates a simple way to review the interpretation later.
- **Handle edge cases intentionally:** Optional uploads, date formats and stale test data can affect the user experience even when the core automation works.
- **Separate interpretation from execution:** AI is useful for understanding messy input, while deterministic automation is better suited to predictable transformations and routing.

Next Version

The next version would address the pain point identified by the intended user: manual receipt entry. Instead of requiring an expense description, a vision-capable model could extract available information directly from a receipt photo and pass it into the existing downstream workflow.

The goal would not be to rebuild the system, but to replace more of the manual intake while preserving the structured output already proven in the prototype.

At larger scale, that version would also require evaluating model choice, API cost and processing volume rather than assuming the prototype architecture should simply be multiplied across every employee.