AI Automation / No-Code Automation

Diagnose why a no-code automation is failing — API errors, data type mismatches, permission issues — stops hours of frustration.
Difficulty: Intermediate
Model: GPT-4 / Claude / Gemini
Use Case: Debugging, Error Diagnosis
Updated: May 2026
Why This Prompt Exists
No-code tools produce cryptic error messages — or worse, fail silently. Most users try random fixes for hours. This prompt brings systematic debugging.

You get:

- hours wasted trying random fixes (“maybe if I move this module…”)
- error messages you don’t understand (“invalid input syntax for type JSON”)
- automations that work sometimes but not always (intermittent failures)
- no way to reproduce the problem (works on your test data, fails on real data)
- asking for help but not knowing what information to provide

But debugging can be systematic:

- reproduce: what triggers the failure?
- isolate: which step fails? (binary search)
- examine: what’s the input data at that step?
- compare: what’s different between working and failing cases?
- fix: change one variable at a time

Without systematic debugging, you chase ghosts.

This prompt diagnoses no-code automation failures.

The Prompt

```
Assume the role of a no-code debugging expert who diagnoses automation failures.

Your task is to identify the root cause of an automation failure.

Generate:

1. PROBLEM STATEMENT
   - What should happen
   - What actually happens
   - When it happens (always / sometimes / only with specific data)

2. ERROR INFORMATION
   - Error message (if any)
   - Where the error occurs (module/step number)
   - Screenshot description (if available)

3. REPRODUCTION STEPS
   - Step 1: [what you do]
   - Step 2: [what happens]
   - Step 3: [where it fails]

4. ROOT CAUSE ANALYSIS

| Possible Cause | Check | Likelihood | Evidence |
```

```
|----------------|-------|------------|----------|
| Data type mismatch | [field type vs. expected] | High/Med/Low |
[evidence] |
| Missing required field | [field is empty] | High/Med/Low |
[evidence] |
| API rate limit | [429 response] | High/Med/Low | [evidence] |
| Permission denied | [auth error] | High/Med/Low | [evidence] |

5. DIAGNOSTIC STEPS TO CONFIRM
   - Step to test: [what to change to confirm root cause]

6. FIX RECOMMENDATION
   - What to change
   - Where to change it
   - Expected result

7. PREVENTION
   - How to avoid this failure in the future

INPUTS:

Tool/platform:
[MAKE / ZAPIER / BUBBLE / AIRTABLE / RETOOL / N8N / OTHER]

Error description:
[PASTE ERROR MESSAGE OR DESCRIBE WHAT HAPPENS]

What should happen:
[E.G., "New Airtable record should create Google Doc"]
```

```
What actually happens:
[E.G., "Nothing — scenario stops with error"]

Failing data sample (if available):
[PASTE DATA THAT CAUSES FAILURE]

Working data sample (for comparison):
[PASTE DATA THAT WORKS]

RULES:
- Isolate the failing step first (binary search: disable half the
steps)
- Compare working and failing data — the difference is often the cause
- Check data types — most no-code errors are type mismatches
- Check for empty fields where required data is expected
- Check API rate limits for high-volume automations
- Reproduce with minimal test case before asking for help
```

How To Use It

- Isolate the failing step first — disable half the steps, see if it still fails (binary search).
- Compare working and failing data — the difference is often the cause.
- Check data types — most no-code errors are type mismatches (text vs. number vs. date).
- Check for empty fields where required data is expected.
- Check API rate limits for high-volume automations (429 errors).

Example Input

Tool/platform:

“Make.com”

Error description:
“Scenario fails at the ‘Create Google Doc’ step. Error: ‘Invalid input: body parameter missing'”

What should happen:
“When new row added to Airtable, create Google Doc from template, then email to client”

What actually happens:
“Scenario stops at Google Doc step — no doc created, no email sent”

Failing data sample:
“Row with empty ‘Client Name’ field”

Working data sample:
“Row with ‘Client Name’ filled in”

Why It Works
Most no-code debugging is trial and error — change something randomly, test, repeat. Hours wasted.

This framework improves outcomes by forcing:

- reproduction steps (how to make it fail consistently)
- root cause hypothesis (what’s likely wrong)
- diagnostic confirmation (test before fixing)
- specific fix (not a guess)
- prevention (how to avoid next time)

Failure modes this diagnoses:

- Data type mismatches (most common — text vs. number vs. date)
- Missing required fields (empty values where data expected)
- API rate limits (429 errors, especially on free tiers)
- Authentication expiry (tokens expire, need refresh)
- Field name changes (API schema updated, mapping broken)

This improves on: Generic “why isn’t this working?” questions that get vague answers. Provides structured diagnostic checklist.

Related to: NCA-03 (Make), NCA-04 (Bubble), NCA-06 (n8n) — specific debugging for each platform.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, debugging frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also Make.com Scenario Architect