AI Automation / No-Code Automation

Design visual automation scenarios with modules, routers, aggregators, and error handling — builds reliable workflows.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Automation Design, Workflow Engineering
Updated: May 2026
Why This Prompt Exists
Make.com (formerly Integromat) is powerful but complex. Most users build linear scenarios that break on the first error — no routers, no error handling, no aggregators.

You get:

- scenarios that fail silently (no error routing)
- data processing that stops on first error (missing downstream data)
- webhook responses that timeout (no immediate response)
- bundles of 100+ operations with no aggregator (slow, expensive)
- scenarios that are impossible to debug (no error branches)

But reliable scenarios have structure:

- trigger: webhook, schedule, watch records
- routers: parallel branches for different conditions
- error handling: separate path for failures
- aggregators: collect multiple records before processing
- filters: skip operations when conditions aren’t met

Without architecture, scenarios collapse under real-world conditions.

This prompt designs reliable Make.com scenarios.

The Prompt

```
Assume the role of a Make.com automation architect who designs
reliable scenarios.

Your task is to design a Make.com scenario blueprint for a given
automation.

Generate:

1. TRIGGER SPECIFICATION
   - Module: [Webhook / Schedule / Watch Records]
   - Configuration: [URL for webhook, cron for schedule, filter for
watch]
   - Expected payload (for webhooks)

2. SCENARIO FLOW (text-based diagram)

   TRIGGER → ROUTER → Branch A → Action 1 → Action 2
                    → Branch B → Action 3
                    → Error Branch → Notification

3. MODULE SEQUENCE (detailed)

| Step | Module | App/Event | Configuration | Bundling |
|------|--------|-----------|---------------|----------|
| 1 | Trigger | [app] | [settings] | N/A |
```

| 2 | Router | Condition | [if condition] | N/A |
| 3 | Action | [app] | [field mappings] | 1 bundle |

4. ROUTER CONDITIONS
   - Branch A: [condition, e.g., "status = new"]
   - Branch B: [condition, e.g., "status = priority"]
   - Default branch: [if no condition met]

5. ERROR HANDLING
   - Error source: [which modules might fail]
   - Error branch: [what to do on failure]
   - Retry configuration: [number of retries, delay]

6. AGGREGATOR CONFIGURATION (if needed)
   - Source module: [where data comes from]
   - Aggregator type: [Array / Text / Numeric]
   - Batch size: [X items]

7. FILTERS
   - Where to add filters (and conditions)

8. WEBHOOK RESPONSE (if webhook trigger)
   - Immediate response: [200 OK / 202 Accepted / error]
   - What to return

9. OPERATIONS ESTIMATE
   - Estimated operations per run: [X]
   - Estimated monthly ops: [Y]

```
INPUTS:

Automation description:
[E.G., "When a new row is added to Airtable, create a Google Doc,
convert to PDF, email to client"]

Source app details:
[E.G., "Airtable base with table 'Orders'"]

Volume estimate:
[E.G., "200 runs per day, 1-10 bundles per run"]

Error sensitivity:
[LOW (can skip errors) / MEDIUM (retry) / HIGH (must not fail)]

RULES:
- Add error handling to every module that can fail (API calls,
webhooks, external services)
- Use routers to process different conditions in parallel
- Use aggregators when processing batches of 50+ records (reduces
operations)
- Always respond to webhooks immediately (200 OK) before long
processing
- Add filters before expensive operations (don't process unnecessary
records)
- Test error branches by simulating failures
```

How To Use It

- Add error handling to EVERY module that calls an external API — they will fail

eventually.

- Use routers for different conditions, not separate scenarios (easier to manage).
- Always respond to webhooks immediately (200 OK) before long processing — or they'll timeout.
- Add filters BEFORE expensive operations to save operations.
- Test your error branches by intentionally causing failures (wrong API key, missing data).

Example Input

Automation description:
"When a new form is submitted in Typeform, create a contact in HubSpot, send a welcome email via Gmail, and add a task in Asana."

Source app details:
"Typeform, HubSpot, Gmail, Asana"

Volume estimate:
"500 form submissions per day"

Error sensitivity:
"MEDIUM — retry failed actions, but don't stop the whole scenario"

Why It Works
Most Make.com scenarios are linear chains that break on first error — fine for hobby projects, disastrous for business automation.

This framework improves outcomes by forcing:

- trigger specification (what starts the scenario?)
- router conditions (different paths for different data)
- error handling (what happens when a module fails?)

- aggregator use (batch processing for efficiency)
- webhook response strategy (prevent timeouts)

Failure modes this prevents:

- Webhook timeout on long-running scenarios (process asynchronously)
- Operation limits exceeded (use aggregators for batches)
- Silent failures (error branches with notifications)
- Processing everything twice (filters before expensive modules)

This improves on: Basic “if this then that” tutorials that skip error handling. Real automation needs retries, fallbacks, and monitoring.

Related to: NCA-02 (Airtable Schema) for data structure; NCA-06 (n8n) for alternative 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 Airtable Schema Designer