LAB A — AI-assisted Client Onboarding Agent (FinTech)

1. Introduction

This document reports on the design and implementation of a Client Onboarding Automation built in n8n.The goal of this automation is to support a FinTech-style onboarding process where client data is evaluated using a combination of business rules, KYC verification, and credit-score checks, and then a provisional decision is produced for senior review.

The report describes the overall workflow, the logic implemented in each node, how I designed the prompts and JSON schema for the AI agent, and which n8n features I used. It also includes my thought process, trade-offs, and a short plan for further testing.

2. Business Context & Objectives

In a typical FinTech onboarding process, analysts must manually check:

- Whether the client's nationality is allowed by policy,
- Whether the KYC checks pass, and
- Whether the credit score is above a minimum threshold or needs manual review.

My objective with this project was to simulate this decision process end-to-end in n8n, using AI to:

- Interpret incoming client data,
- Decide which tools to call (KYC and/or credit score), and
- Produce a structured provisional decision (approve, manual_review_senior, manual_review_board, or reject) with a clear explanation.

This should give my senior a concrete example of how n8n can orchestrate AI agents, business rules, and external checks in a single automated workflow.

3. Workflow overview

At a high level, the workflow works as follows:

1. A Webhook receives client onboarding data (ID, name, email, nationality, status, free text, etc.).
2. A System Variables / Policies node provides configuration like ALLOWED_NATIONALITIES, CREDIT_MIN, and BOARD_MARGIN_POINTS.
3. An AI Agent Planner (first agent) reads the client data + system variables and:
    - Apply the business rules,
    - Decides which tools need to be called next (kyc, credit_score, or both),
    - Proposal provisional action (approve / manual review / reject),
    - Returns a strictly structured JSON object.
4. Depending on the tools requested, the workflow is:
    - A KYC simulation branch,
    - A credit score simulation branch.
5. A Merge node combines the simulation results.
6. A final AI step formats the overall decision in a way that is easy for management to read or for another system to consume.

Throughout this process, I made use of n8n's visual editor (drag-and-drop mapping, schema view, logs) to iterate quickly and keep the logic transparent.

AI Agent 1 – Planner (Pre-Check Decision Maker)

Purpose:

AI Agent 1 is responsible for analyzing the initial client data and applying business rules before any expensive or conditional checks (like KYC or credit score) are executed.

Its job is to:

- Analyze the client's identity fields
- Apply all nationality, student, risk, and missing-data rules
- Decide *which tools need to be executed*
- Output a preliminary "provisional action"
  (approve / manual review senior / manual review board / reject)

Why must this be AI Agent #1:

- It behaves like an "orchestrator" or pre-screening AI supervisor.
- It avoids calling unnecessary tools (for example, rejecting a client without paying for KYC or credit scoring).
- It keeps rule-based reasoning separate from the final summary, making the workflow more transparent.

AI Agent 2 – Final Decision Formatter (Post-Check Synthesizer)

Purpose:

AI Agent 2 is used *after* the KYC and credit score simulations are completed.

Its job is to:

1. Take the real results (KYC pass/fail, identity issues, credit score value)
2. Combine them with the planner's initial provisional action
3. Generate a clean final decision for internal reporting
4. Produce a human-readable explanation + a machine-readable structure

This agent acts as the “final aggregator”, creating the final report that management or downstream systems could consume.

Why Two AI Agents Instead of One?

1. Separation of concerns = more predictable automation

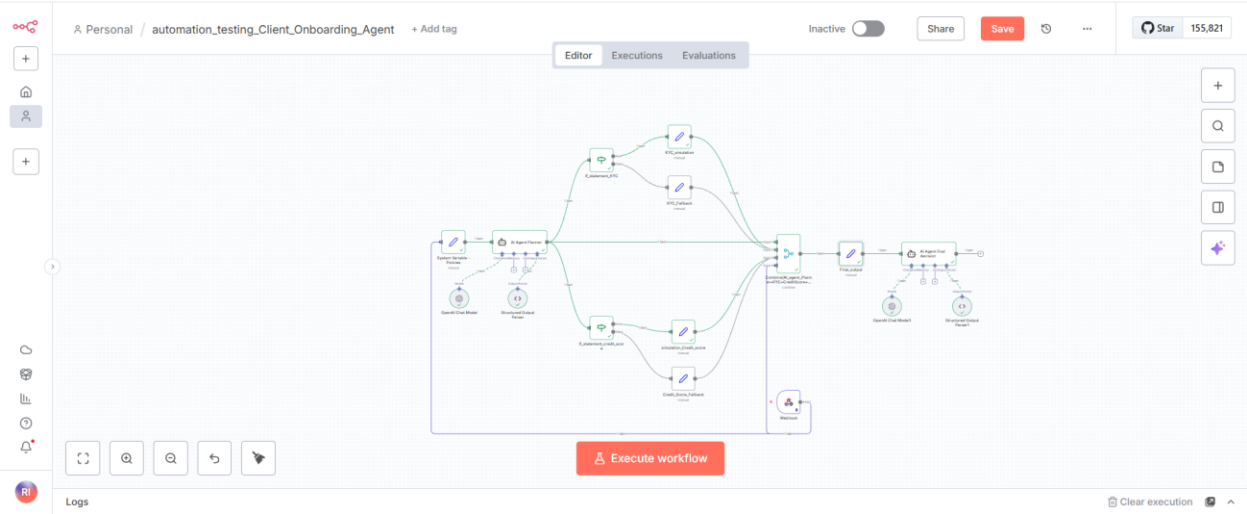
Trying to make one AI agent do planning + rule reasoning + tool decisions + final formatting would make the prompt extremely long and more error prone.

Splitting into two:

- Agent 1 = decide what needs to happen next
- Agent 2 = summarize results after everything has happened

This mirrors how real onboarding team’s work:

- A junior agent screens the client first (pre-check)
- A senior analyst generates the final report



STEP 1 — Log In to n8n Cloud

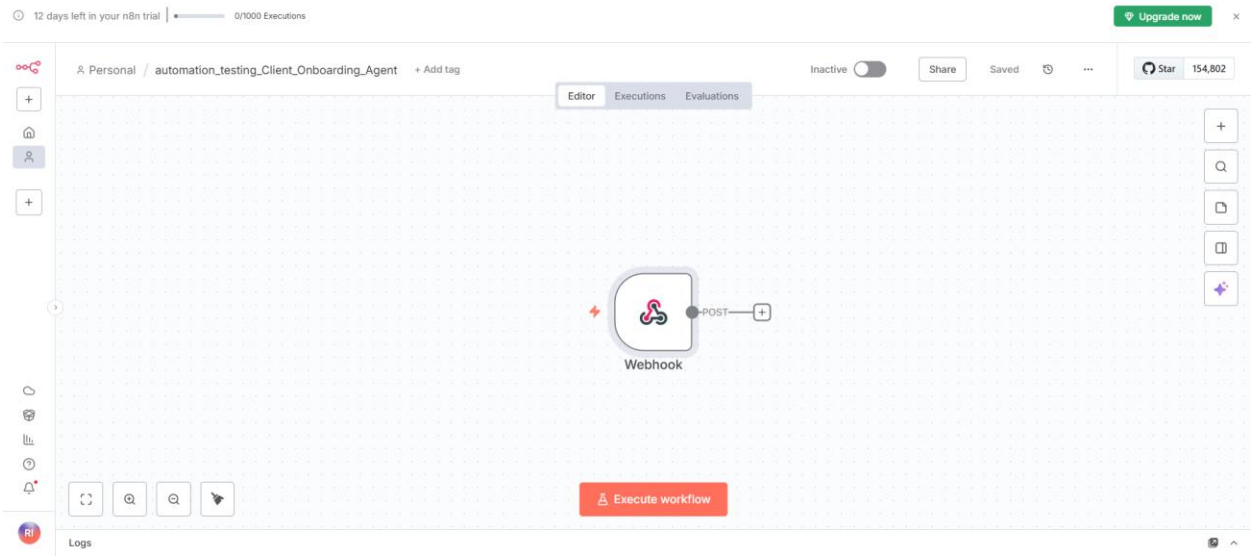
Create a n8n account.

Link:

https://n8n.io/?ps_partner_key=OGVmN2FiZjIyZmJl&ps_xid=zSMHTs3c8g2z9D&gsxid=zSMHTs3c8g2z9D&gspk=OGVmN2FiZjIyZmJl&gad_source=1&gad_campaignid=23226852217&gbraid=0AAAABBzbtJDAfgfO7xRx-QxMqOPB5lvQK&gclid=CjwKCAiA8bvIBhBJEiwAu5ayrJSAPXJ86u9A8-Brdg4EWcLeqTnjxHunENQe9wdvQSxi0cgmLR7rYhoCqmwQAvD_BwE

STEP 2 — Create Your First Workspace

Click **New Workflow** and create a new Workflow then name it.



Step 3 – Create a WEBHOOK

The webhook is:

-A URL
-That listens for incoming data
-And triggers an action when data arrives

You can think of it like: “A mailbox”

– The webhook URL is the address
– Someone “sends” data to the address
– n8n “receives” the letter
– n8n then starts the workflow

Webhook Example — FinTech Client Onboarding

Imagine a client filling in a form on a website:

```
[
{
"ALLOWED_NATIONALITIES":
[
"India",
"Singapore",
"Malaysia"
],
"CREDIT_MIN":
500,
"BOARD_MARGIN_POINTS":
3
}
]
```

Instead of the website saving data in a database, then wait for a script:

The website **sends this data directly to your webhook**
- n8n receives it immediately
- n8n triggers your onboarding workflow
- AI agent reads it
- KYC tool runs
- Credit check runs
- Compliance team notified....All in seconds.

It’s just an endpoint waiting for data. You don’t store anything inside the webhook. You send data *to* the webhook from the outside.

OPTION 1 — Use cURL

1. What is curl?

curl is a small tool you run in your computer's terminal to send a request to a URL.

It lets you:

- send JSON
- test APIs
- test webhooks
- send GET, POST, PUT, DELETE requests
- simulate what other systems will send

Every automation tool (Zapier, n8n, Make, Airflow) uses curl in examples. Curl is the fastest way to "simulate" a real event hitting your webhook. Instead of waiting for a real client onboarding form or a real insurance claim submission...You just:

"Send a JSON test payload directly to your webhook using curl."

This allows you to test the workflow repeatedly.

Curl template:

test.readme

```
curl -X METHOD "URL" \
-H "header-key: header-value" \
-d 'JSON_DATA'
```

Let's break it down:

-curl:

The command itself.

- -X METHOD:

Tell curl which HTTP method to use:

- POST (send data)

- GET (fetch data)
- PUT (update)
- DELETE (delete)

Your n8n webhook uses: “POST”

- "URL":

This is the webhook URL generated by n8n.

-H "header: value":

Header → tells curl you’re sending JSON.

-d ' {...json...} ':

This is the data you’re sending.

Example:

test.readme

```
curl -X POST "YOUR_WEBHOOK_URL" \
-H "Content-Type: application/json" \
-d '{
  "client_id": "00-000",
  "full_name": "Isaac Jacob",
  "email": "fakemail@gmail.com",
  "account_number": "A00000",
  "nationality": "Belgium",
  "free_text": "international contractor",
  "status": "student"
}'
```

OPTION 2 — Use the built-in “Mock Data” in n8n

METHOD 1 — “Mock Data” inside the Webhook Node

This lets you feed JSON directly into the Webhook node during testing.

Steps:

1. Click your Webhook Trigger node.

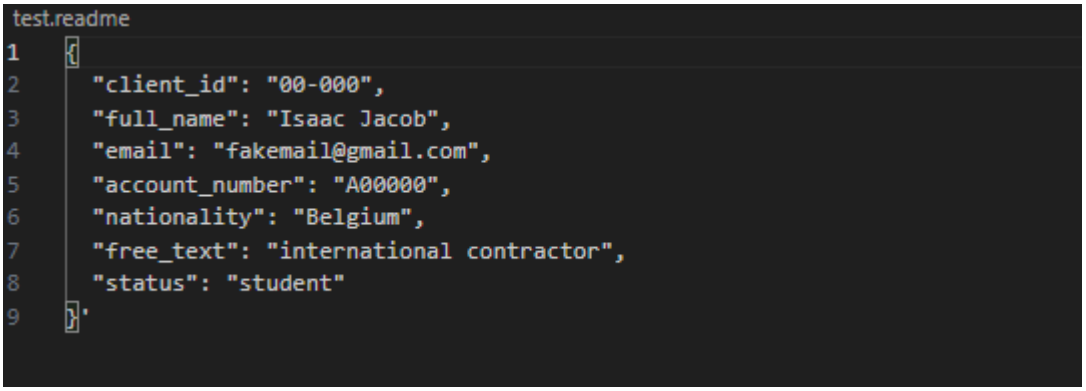
On the right side you will see:

- *Node details*
- *Parameters*
- *Mock Data* panel

2. Find the button:

"Set Mock Data"
(It is inside the Webhook node settings, on the right panel.)

3. Paste your JSON:



4. Click Save Mock Data

5. Click Execute Node

n8n will simulate:

- the webhook receiving data
- starting the workflow
- passing the JSON to the next node (AI Agent)

METHOD 2 — “Listen for Test Event”

(For real-time simulation without curl)

This method simulates a real incoming webhook request.

- Steps:

1. Click your Webhook Trigger

2. Click the button:

- Listen for Test Event

Your node now says:

Waiting for webhook call...

3. In the same right panel, scroll down and click:

“Send Example Data”

n8n will automatically send mock JSON into your own webhook to help you test.

You can also customize it.

METHOD 3 — Add a “Set Node” instead of a Webhook

(When you want to test without external requests)

This replaces a webhook completely.

- Good when developing
- Not good when connecting to real systems

Steps:

1. Delete the Webhook node

2. Add Set node
3. Add the fields manually

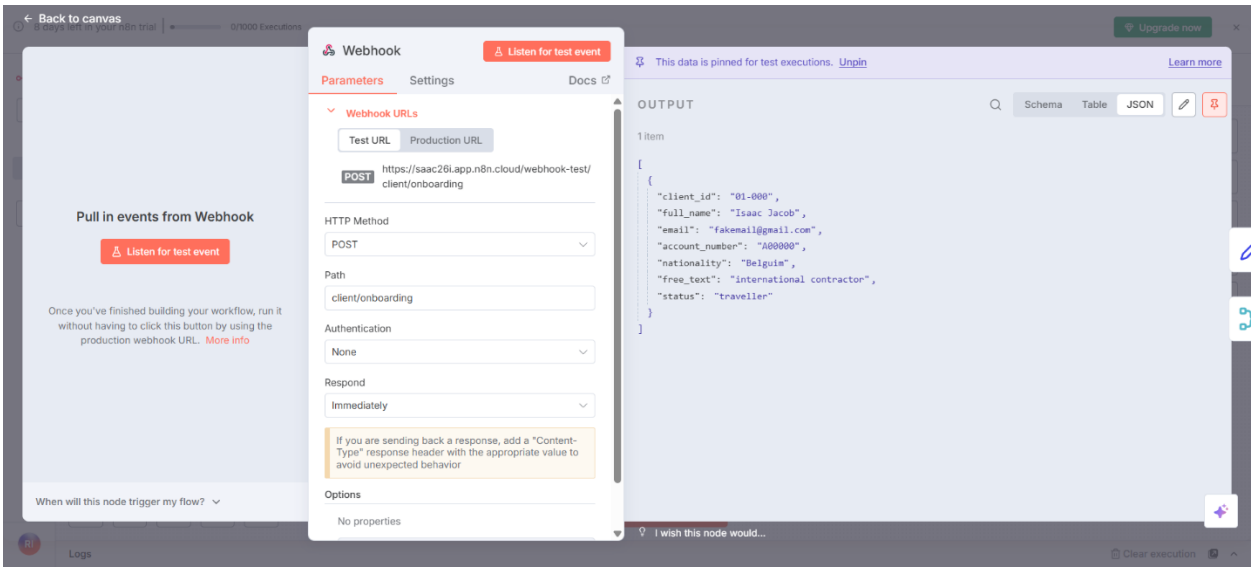
For our experiment we are going to use the **“Mock Data” inside the Webhook Node**.

4. Detailed Workflow Design

4.1 Webhook Entry Point

The Webhook node is the starting point of the workflow.

- Method: POST
- Path: /client/onboarding
- Example payload I used for testing:



This allows me to trigger the flow from tools like Postman or from a future front-end, and it also lets me pin test data in n8n so that I can re-run only downstream nodes while building.

4.2 System Variables & Policy Configuration

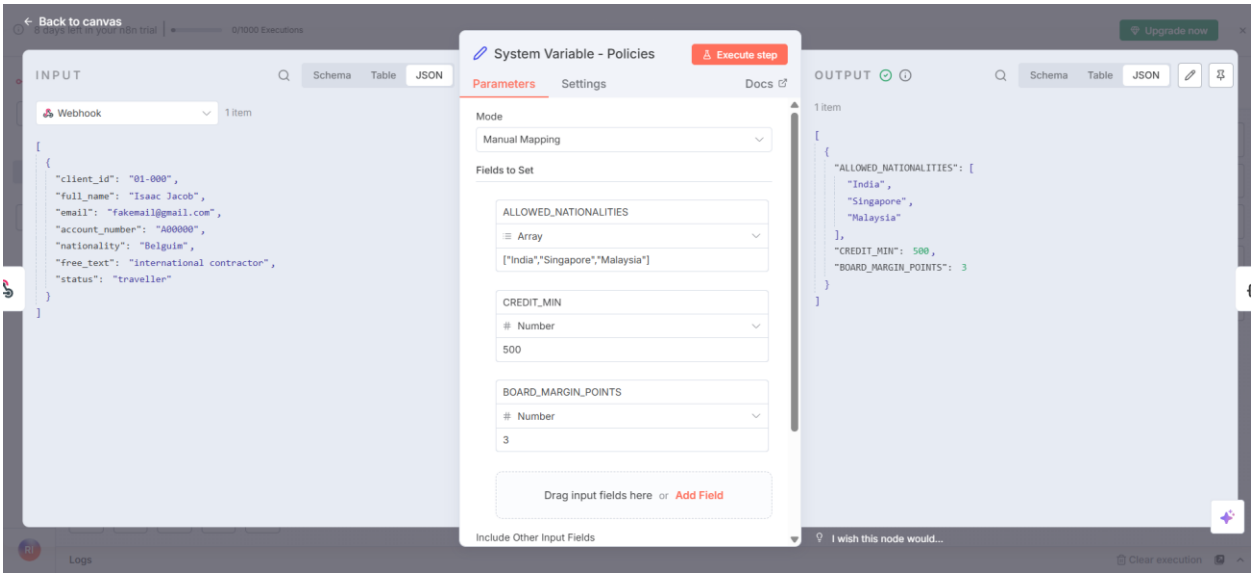
Right after the webhook, I created a node to store policy parameters that the AI planner needs:

```
ALLOWED_NATIONALITIES: e.g. ["India", "Singapore", "Malaysia"]

CREDIT_MIN: e.g. 500

BOARD_MARGIN_POINTS: e.g. 3
```

I used n8n's drag-and-drop mapping and JSON editor to set these values. This design means I can later adjust policies (for example change CREDIT_MIN from 500 to 550) without touching the AI prompt itself.



4.3 AI Agent Planner Prompt Design & Structure

Designing the prompt for the Planner Agent is one of the most important parts of the entire automation.
This agent is responsible for interpreting raw client data, applying business rules, and deciding which verification tools must be executed next. Because this step directly controls the workflow branching, the prompt must be precise, predictable, and strongly structured.

Below I explain how to design a good prompt for this type of agent, the structure I used, and why each component is necessary when building AI-driven branches in n8n.

4.3.1 How to Create a Good Prompt for n8n AI Agents

A good prompt for an AI agent in n8n follows six core principles:

1. Define the AI's Role Clearly

The first line of the prompt should define *who the AI is supposed to be*.

Example from my implementation:

"You are an automated Client Onboarding Planner for a FinTech company."

This ensures the model behaves as a rule-based decision engine, not as a creative assistant.

2. Specify the Exact Inputs the AI Will Receive

AI Agents cannot guess what fields exist in the incoming data. Listing each field avoids hallucinations and ensures deterministic behavior.

In my case, the planner receives:

- client_id
- full_name
- email
- account_number
- nationality
- status
- free_text

- System variables (ALLOWED_NATIONALITIES, CREDIT_MIN, BOARD_MARGIN_POINTS)

Using n8n expressions (drag-and-drop or {{$json.FIELD}}) guarantees the model always sees the latest values from the workflow.

3. Describe the Available Tools Explicitly

The planner needs to know which tools exist and what tool returns.

Example:

- kyc → returns { "passed": boolean, "issues": [] }
- credit_score → returns { "creditScore": number }

Without this, AI may invent fake tools or try to call operations that do not exist in the workflow.

4. Embed Business Rules in a Clear, Mechanical Way

This is the heart of the prompt.

Each rule should be written in a simple, unambiguous, machine-like format, as you would in documentation for engineers or compliance auditors.

Example rules I used:

- Nationality gate
- Student exception
- Credit score minimum
- Missing essential identity fields
- Conditions for senior vs board manual review
- Approve only if all criteria are satisfied

Writing them this way makes the planner reproducible and consistent.

5. Tell the Agent Exactly What You Expect It to Do

For the Planner Agent, the instructions were:

“Analyze ONLY the input now. Decide which tools to call next and propose a provisional action according to the rules.”

This prevents the model from trying to format a final report or make assumptions about KYC results before the tools run.

4.3.2 AI Agent Planner (First Agent)

The AI Agent Planner is the core “brain” of the first stage. Its job is to decide:

- Which tools to call next: ["kyc"], ["credit_score"], ["kyc", "credit_score"], or [].
- What provisional action to take, according to the business rules.

Input

The prompt explicitly tells the model which fields it will receive:

- Client fields: client_id, full_name, email, account_number, nationality, status, free_text.
- System variables injected using n8n expressions:
  - ✓ ALLOWED_NATIONALITIES = {{$json.ALLOWED_NATIONALITIES}}
  - ✓ CREDIT_MIN = {{$json.CREDIT_MIN}}
  - ✓ BOARD_MARGIN_POINTS = {{$json.BOARD_MARGIN_POINTS}}

This shows a key n8n feature: I can *directly drag or reference JSON fields* into the prompt, so the AI always works with the latest configuration.

Business Rules inside the Prompt

Inside the prompt, I encoded the decision logic in natural language but in a very structured way. For example:

- **Nationality gate**
    - If nationality is not in ALLOWED_NATIONALITIES and status == "student" → provisional_action = "manual_review_board".
    - If nationality is not allowed and status is not student → provisional_action = "reject".
- **Student special case**
    - If status == "student" and credit score are within BOARD_MARGIN_POINTS below CREDIT_MIN → provisional_action = "manual_review_board".
- **Standard manual review**
    - If nationality is allowed but KYC fails OR credit score too low → manual_review_senior.
- **Approved**
    - Only when nationality is allowed, KYC passed, credit score ≥ CREDIT_MIN, and no major issues.
- **Missing critical identity fields**
    - If key fields like full_name, email, account_number, nationality are missing → prefer manual_review_senior or reject.

This approach keeps all the compliance logic visible in the prompt, which is easy to read and change for non-technical stakeholders.

Tools Available

The planner is only allowed to use:

- kyc → returns { "passed": boolean, "issues": [] }
- credit_score → returns { "creditScore": number }

This is clearly described in the prompt, so the model does not invent tools that do not exist in the workflow.

Required Output Format

To make the output reliable, I instructed:

```
{
  "provisional_action": "approve | manual_review_senior | manual_review_board | reject",
  "reason": "",
  "callTools": []
}
```

Full prompt:

You are an automated Client Onboarding Planner for a FinTech company.

Inputs you will receive:

- client fields: client_id, full_name, email, account_number, nationality, status, free_text.

- system variables:

- ALLOWED_NATIONALITIES (array) = {{$json.ALLOWED_NATIONALITIES}}

- CREDIT_MIN = {{$json.CREDIT_MIN}}

- BOARD_MARGIN_POINTS = {{$json.BOARD_MARGIN_POINTS}}

Tools available: ["kyc", "credit_score"].

- "kyc" returns { "passed": boolean, "issues": [] }.

- "credit_score" returns { "creditScore": number }.

Business rules (apply strictly; case-insensitive checks):

1) Nationality gate:

- If nationality NOT in ALLOWED_NATIONALITIES AND status == "student" → provisional_action = "manual_review_board".

- If nationality NOT in ALLOWED_NATIONALITIES and status != "student" → provisional_action = "reject".

2) If status == "student" AND final credit score is within BOARD_MARGIN_POINTS below CREDIT_MIN → provisional_action = "manual_review_board".

3) If nationality is allowed BUT (KYC fails OR credit score < CREDIT_MIN and not within board margin) → provisional_action = "manual_review_senior".

4) Approve only if: nationality allowed, KYC passed, credit score ≥ CREDIT_MIN, and no major issues.

5) If critical identity fields are missing (full_name, email, account_number, nationality), prefer "manual_review_senior" (or "reject" if severely incomplete/suspicious).

Business rules (apply strictly; case-insensitive checks):

1) Nationality gate:

- If nationality NOT in ALLOWED_NATIONALITIES AND status == "student" → provisional_action = "manual_review_board".

- If nationality NOT in ALLOWED_NATIONALITIES and status != "student" → provisional_action = "reject".

2) If status == "student" AND final credit score is within BOARD_MARGIN_POINTS below CREDIT_MIN → provisional_action = "manual_review_board".

3) If nationality is allowed BUT (KYC fails OR credit score < CREDIT_MIN and not within board margin) → provisional_action = "manual_review_senior".

4) Approve only if: nationality allowed, KYC passed, credit score ≥ CREDIT_MIN, and no major issues.

5) If critical identity fields are missing (full_name, email, account_number, nationality), prefer "manual_review_senior" (or "reject" if severely incomplete/suspicious).

Your job in this first step:

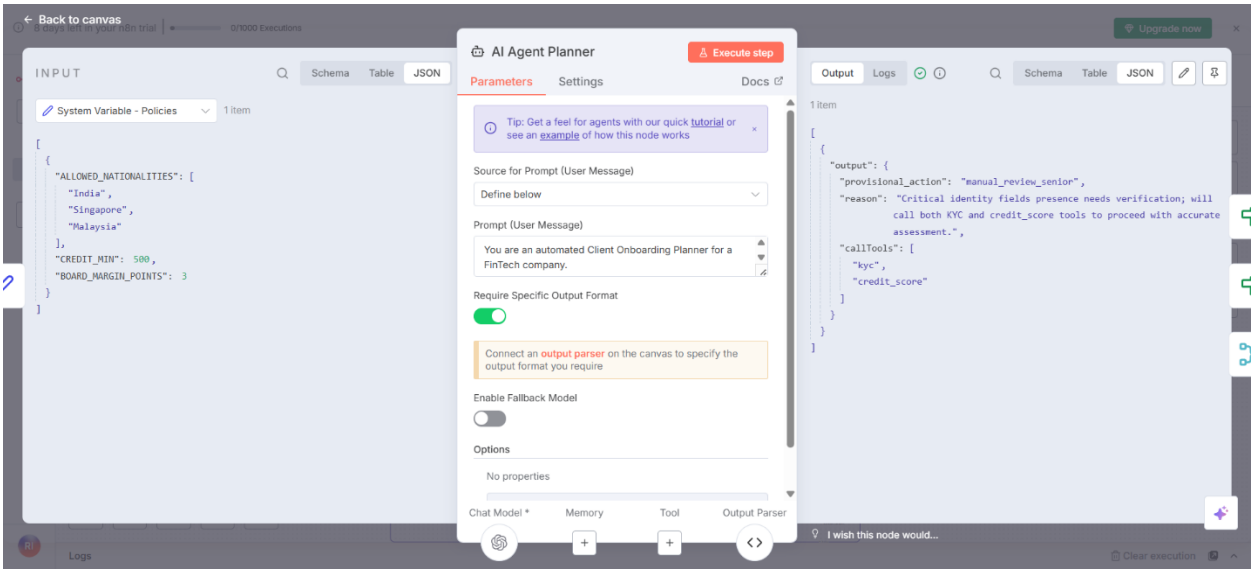
- Analyze ONLY the input now.

- Decide which tools to call next: ["kyc"], ["credit_score"], ["kyc","credit_score"], or [].

- Propose a provisional action per rules (final decision may change after tool results).

Return ONLY this JSON (no extra text):

```
{
  "provisional_action": "approve | manual_review_senior | manual_review_board | reject",
  "reason": "",
  "callTools": []
}
```



5. Prompt Engineering & Structured Output

5.1 Planner Prompt Design (Summary)

Here is a shortened description of the final planner prompt:

- Role: “You are an automated Client Onboarding Planner for a FinTech company.”
- Input: Explicit list of client fields + policy variables from n8n.
- Business rules: The five rules you wrote (nationality gate, student rules, approval conditions, handling missing fields).
- Tool section: Describes exactly what kyc and credit_score return.
- Job description: Analyze the input, choose tools, and propose provisional action.
- Output instruction: Return *only* the JSON object with provisional_action, reason, and callTools.

You can paste the full prompt as an appendix if you like; in the main body it’s better to summarize like this.

5.2 JSON Schema & Structured Output Parser

To enforce the format, I attached a Structured Output Parser node with the following JSON schema:

- Type: object
- Required keys: ["provisional_action", "reason", "callTools"]
- provisional_action: string, enum ["approve", "manual_review_senior", "manual_review_board", "reject"]
- reason: string
- callTools: array of strings, each "kyc" or "credit_score"
- additionalProperties: false

This means that even if the model tries to add extra fields or make small mistakes, n8n validates and parses it into a clean structure that the rest of the workflow can trust.

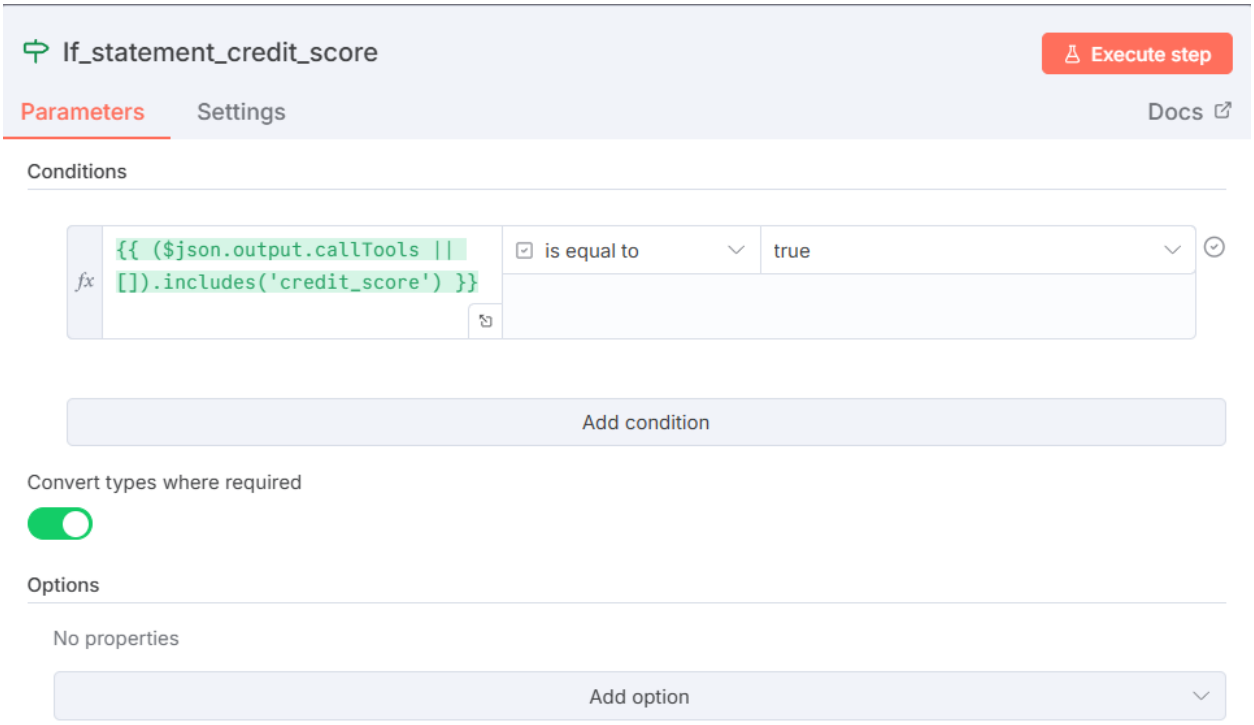
```
{
  "type": "object",
  "required": ["provisional_action", "reason", "callTools"],
  "properties": {
    "provisional_action": {
      "type": "string",
      "enum": ["approve", "manual_review_senior", "manual_review_board", "reject"]
    },
    "reason": { "type": "string" },
    "callTools": {
      "type": "array",
      "items": { "type": "string", "enum": ["kyc", "credit_score"] }
    }
  },
  "additionalProperties": false
}
```

4.4 Conditional Execution of KYC and Credit Score Checks

Once the Planner Agent returns its structured output, the next step is deciding which verification checks to execute. This is done through a set of IF nodes that inspect the array "callTools" returned by the agent.

4.4.1 IF Node – Should We Run Credit Score?

One IF node checks whether "credit_score" is included:



- If TRUE: the workflow follows the credit-score branch
- If FALSE: it triggers the Credit Score Fallback node

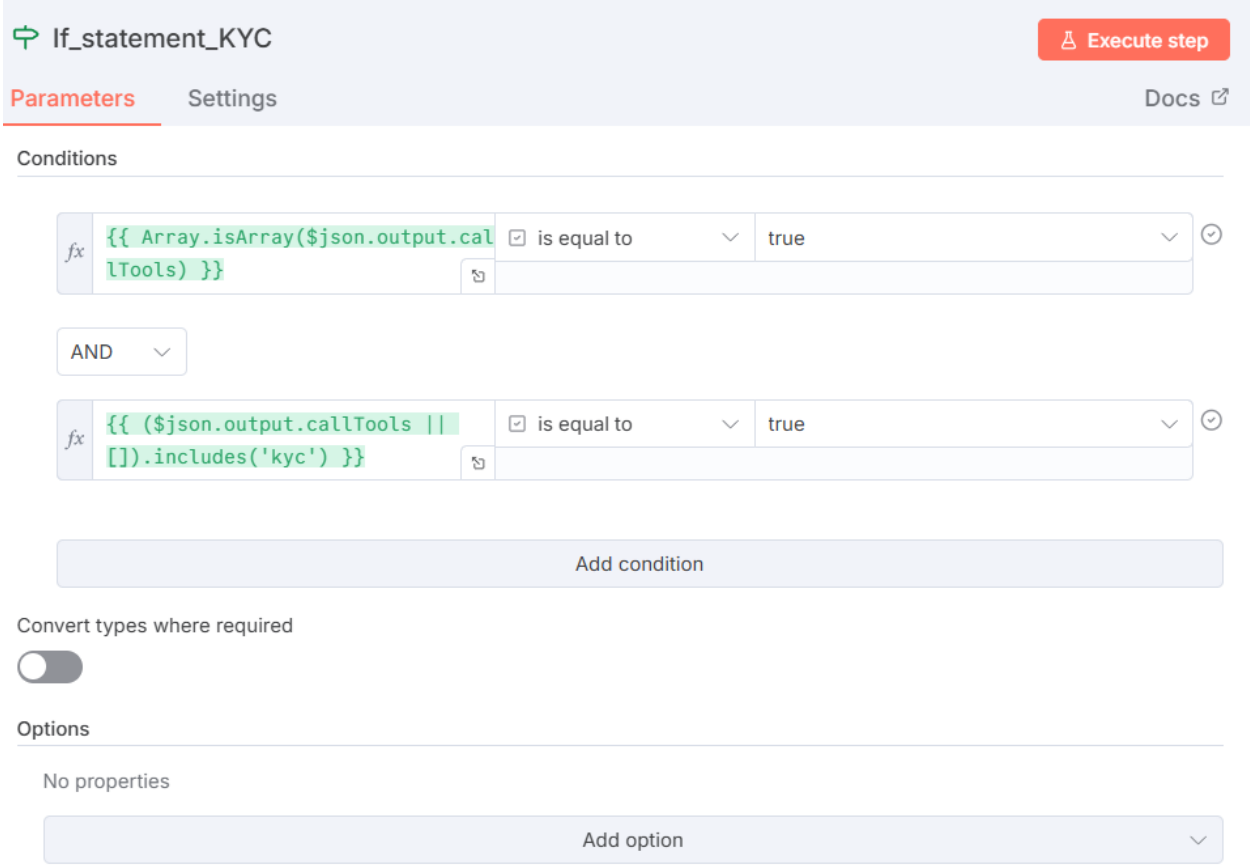
Screenshot: IF block for credit_score
Caption: “Figure X – Conditional branching based on Planner output (credit score).”

This design ensures that:

- The system only pays for necessary checks
- The Planner Agent fully controls workflow execution
- Future rules can add/remove checks without restructuring the workflow

4.4.2 IF Node – Should We Run KYC?

The KYC branch uses the same logic but checks for "kyc":



- If TRUE: run KYC simulation
- If FALSE: trigger KYC Fallback

Screenshot: IF block for KYC
Caption: "Figure X – Conditional branching for KYC execution."

4.5 Simulation Nodes (Mock Implementations)

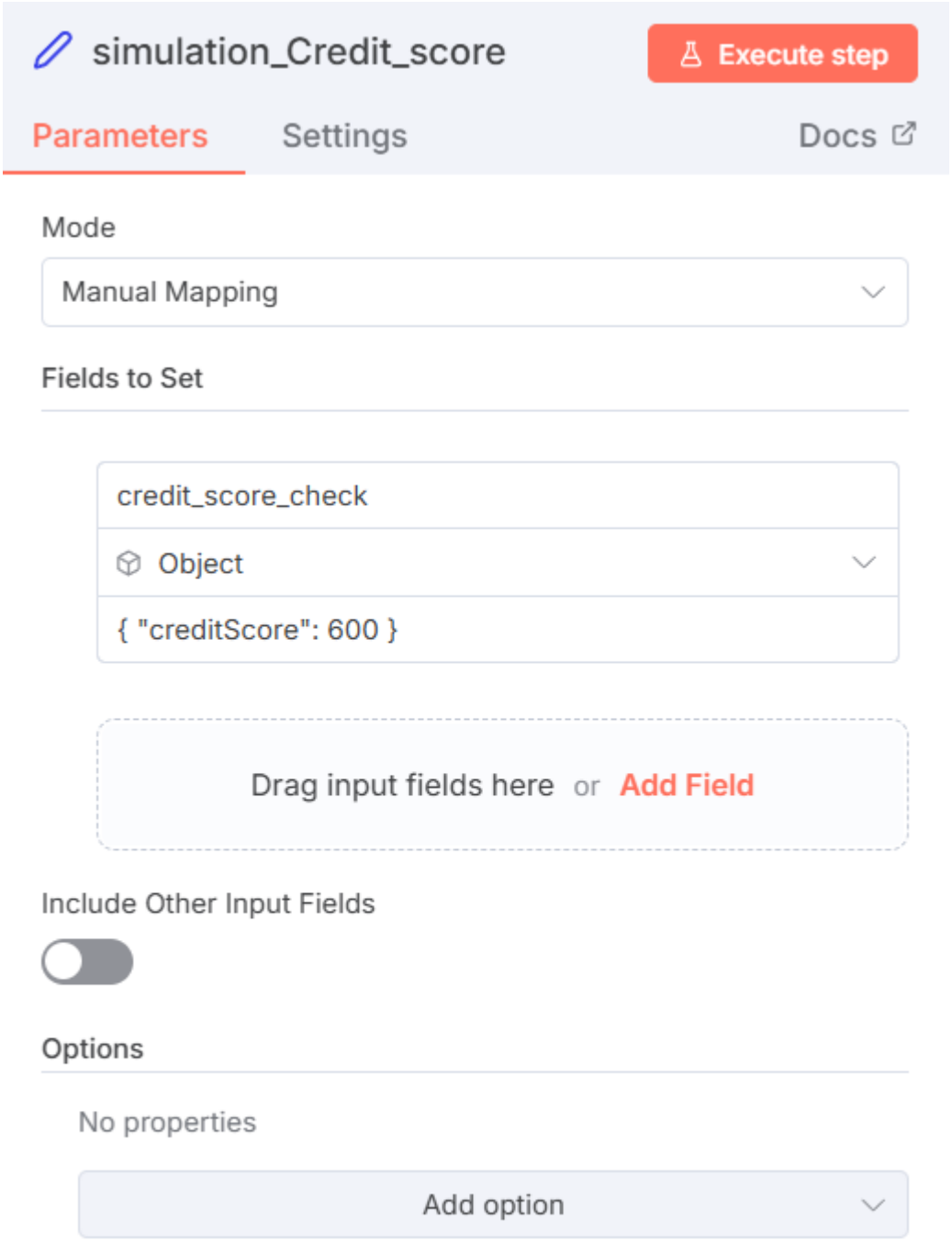
Since you are building a testing and demonstration workflow, both KYC and credit score checks are simulated using Set nodes.

These nodes allow you to define expected output formats without calling external APIs.

4.5.1 KYC Simulation Node

Configuration:

- Mode: Manual Mapping
- Output:



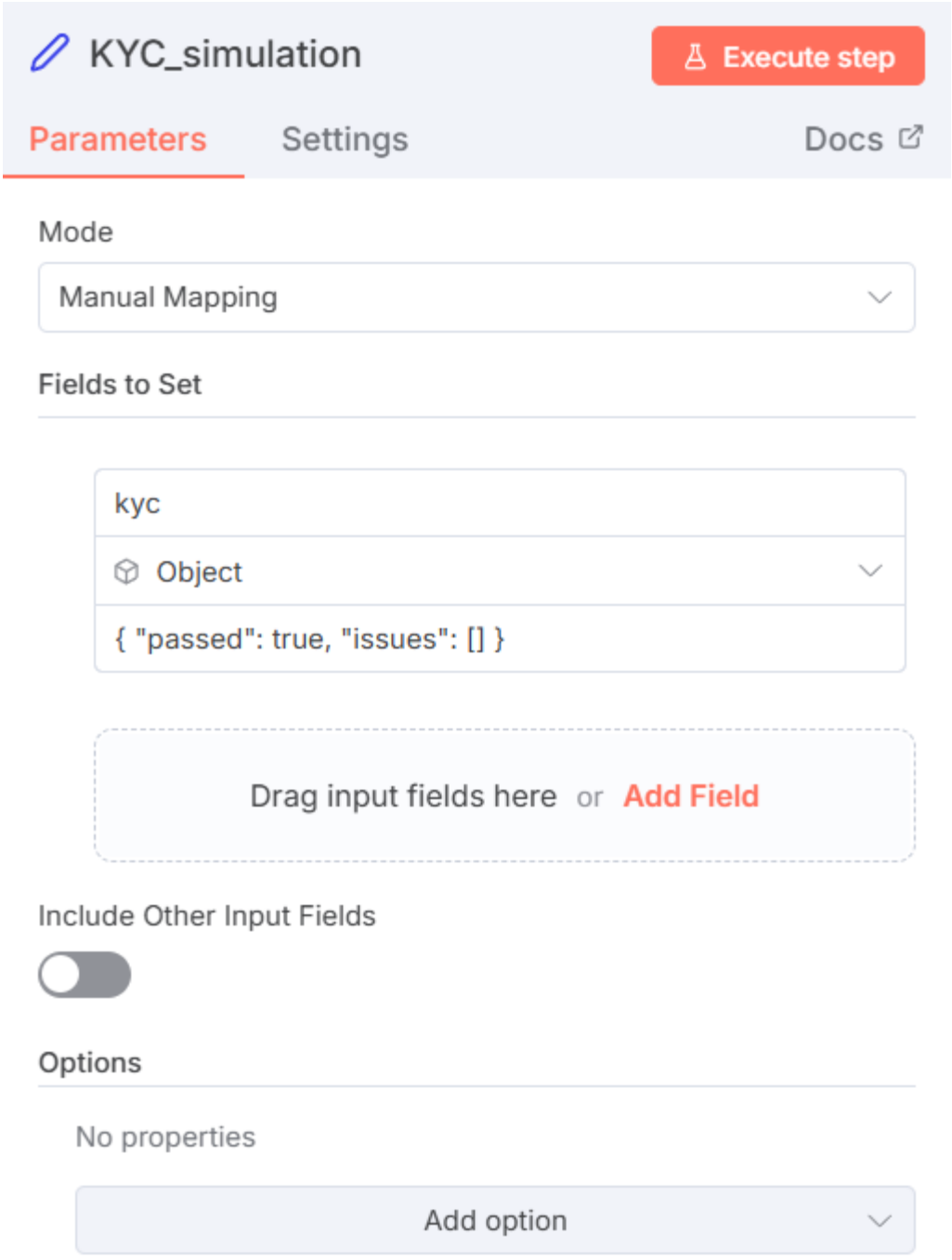
- Guarantees AI Agent 2 receives predictable data
- Maintains full compatibility with rule-based reasoning

- Allows you to test multiple onboarding scenarios safely

4.5.2 Credit Score Simulation Node

Configuration:

- Mode: Manual Mapping
- Output:



4.6 Fallback Nodes

Fallback nodes ensure that the workflow remains consistent even if the Planner decides not to call a specific tool.

This is essential because downstream steps expect both KYC and credit score fields to exist otherwise the merge would break.

4.7 Combining All Results (Merge Node)

Once the branches complete (or fallback fires), the workflow uses a Combine node to merge:

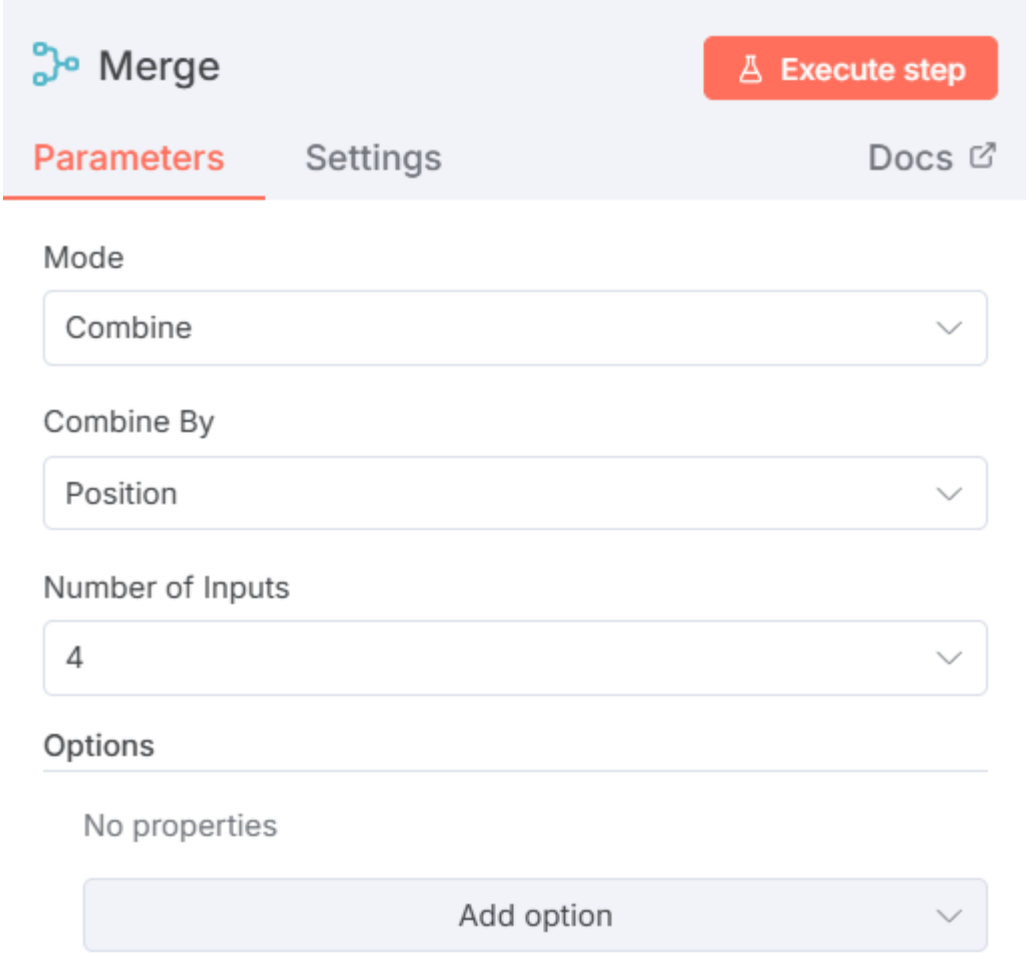
1. The Planner output
2. The KYC result
3. The Credit Score result
4. The original client input

Configuration:

- Mode: Combine
- Combine By: Position
- Number of Inputs: 4

Why “Combine by Position”?

Because each branch returns one item, using position ensures all results align correctly in a single JSON object.



4.8 Final AI Agent – Decision Synthesis & Reporting

After the KYC and credit score simulations (or fallbacks) are merged with the Planner's initial decision, the workflow proceeds to the Final AI Agent.

This agent's role is completely different from the Planner Agent introduced in section 4.3. Instead of deciding which tools to call, the Final Agent focuses on producing a comprehensive, reliable final onboarding summary.

The output of this agent is the final artifact intended for:

- Internal analysts
- Senior reviewers
- Potentially another automated system
- Audit or compliance logs

4.8.1 Purpose of the Final AI Agent

The Final AI Agent operates as the decision synthesizer of the workflow. Its responsibilities include:

1. Reading all the merged inputs:
    - Planner provisional_action
    - KYC results
    - Credit score results
    - Client identity fields
2. Applying the same business rules a second time, but now with full information
3. Producing a final, polished, structured output containing:
    - The final onboarding decision (approve / manual review / reject)
    - Reasons for the decision
    - Evidence and risk indicators
    - Any recommended next steps
4. Formatting the final result in JSON for downstream use

This agent transforms raw operational data into a decision record.

4.8.2 Inputs Provided to the Final Agent

The Final Agent receives the fully merged JSON object, including:

- output.provisional_action (from AI Agent 1)
- kyc.passed, kyc.issues[]
- credit_score_check.creditScore
- All client fields:
    - ✓ client_id
    - ✓ full_name
    - ✓ email
    - ✓ nationality
    - ✓ account_number
    - ✓ status
    - ✓ free_text

This gives it the complete context required to make a final determination.

4.8.3 Designing the Final Agent Prompt

The structure of your final prompt mirrors best practices for AI-based evaluators:

A) Role Definition

"You are an automated senior onboarding analyst."

This signals that the agent must behave like a compliance expert.

B) Clear Task Definition

The agent is instructed to:

- Re-evaluate the client based on *all* data
- Confirm or override the Planner's initial action
- Produce the final structured onboarding decision
- Explain the reasoning clearly and concisely

- Follow the business rules exactly

This prevents the agent from improvising or contradicting core logic.

C) Business Rules Reinforced

Just like the Planner Agent, the Final Agent re-applies:

- Nationality rules
- Student rules
- Credit score thresholds
- Missing identity field checks
- KYC pass/fail conditions

Repeating the rules ensures that even if one of the simulation nodes changes, the final decision remains consistent.

Final full prompt:

You are the FINAL Client Onboarding Compliance Agent for a FinTech company.

You receive a context object containing:

- applicant: client_id, full_name, email, account_number, nationality, status, free_text

- checks: kyc {passed, issues}, credit_score_check {creditScore}

- planner_output: { provisional_action, reason, callTools }

Context (JSON):

{{ JSON.stringify($json.context) }}

Business rules (apply strictly, case-insensitive where relevant):

1) Nationality gate:

- If nationality is NOT in the allowed list AND status == "student" → final_action = "manual_review_board".

- If nationality is NOT in the allowed list AND status != "student" → final_action = "reject".

2) If status == "student" AND final credit score is within BOARD_MARGIN_POINTS below CREDIT_MIN → "manual_review_board".

3) If nationality is allowed BUT (KYC fails OR credit score < CREDIT_MIN and not within board margin) → "manual_review_senior".

4) Approve only if: nationality allowed, KYC passed, credit score ≥ CREDIT_MIN, and no major issues.

5) If critical identity fields are missing (full_name, email, account_number, nationality), prefer "manual_review_senior" (or "reject" if severely incomplete/suspicious).

Policies (constants for this decision):

- ALLOWED_NATIONALITIES: {{ JSON.stringify($item(0).$node["System Variable - Policies"].json.ALLOWED_NATIONALITIES) }}

- CREDIT_MIN: {{ $item(0).$node["System Variable - Policies"].json.CREDIT_MIN }}

- BOARD_MARGIN_POINTS: {{ $item(0).$node["System Variable - Policies"].json.BOARD_MARGIN_POINTS }}

Your task:

- Use ONLY the JSON in Context to decide.

- Do NOT ask for more input.

- Return ONLY the JSON below (no prose):

```
{
  "output": {
    "client_id": "{{ $json.context.applicant.client_id }}",
    "final_action": "approve | manual_review_senior | manual_review_board | reject",
    "reason": "clear, short justification referencing the rules used",
    "risk_level": "low | medium | high",
    "tools_used": ["kyc", "credit_score"]
  }
}
```

5. Use of n8n Features

Throughout this project, I explored and actively used a wide range of n8n features. These features enabled me to design a fully functional, robust, modular AI-driven onboarding workflow. Below is a detailed breakdown of the n8n capabilities used and how they contributed to the automation.

5.1 Webhook Trigger – Automated Entry Point

The workflow begins with a Webhook node configured to receive onboarding requests via:

- Method: POST
- Path: /client/onboarding

I used the pinned test data feature to simulate different user submissions without triggering the entire pipeline. This allowed rapid iteration and testing of downstream nodes.

What I learned:

- Webhooks are ideal for integrating n8n with external apps, forms, or CRMs.
- “Pinned data” makes it possible to test complex flows without resending HTTP requests every time.
- Webhooks immediately return outputs, enabling fast debugging of the early-day workflow stages.

5.2 Set Node & System Variables – Policy as Configuration

I created a System Variables / Policies node to centralize business rules and parameters:

- ALLOWED_NATIONALITIES
- CREDIT_MIN
- BOARD_MARGIN_POINTS

By storing these values in a Set node, I learned that:

- Configuration can be easily updated without touching the AI prompts

- Drag-and-drop mapping and expressions ({{$json.FIELD}}) make it easy to reference these values anywhere
- Keeping policy logic separate makes the workflow more maintainable

This follows best engineering practices ("policy should not be hard-coded inside logic").

5.3 AI Agent Node – Planning & Business Logic Enforcement

The AI Agent Planner is one of the cores n8n AI features I used.

What makes it powerful:

- It accepts a large, structured prompt
- It supports tool calling, which the workflow uses to decide whether to run KYC or credit score checks
- It integrates with a Structured Output Parser, ensuring safe and deterministic outputs

Key learnings:

- The AI Agent must be given a very clear role
- Inputs and tools need to be described precisely to avoid hallucination
- Business rules should be written in a “mechanical style” to be interpreted consistently
- Structured output is essential for downstream branching

This module taught me how to treat AI as a “logic executor,” not a text generator.

5.4 Structured Output Parser – Enforcing JSON Validity

Both AI agents use a Structured Output Parser with a JSON Schema.

Why I used it:

- To enforce strict typing (string, array, enum, object)
- To prevent extra fields or malformed JSON
- To guarantee predictable workflow routing
- To make the output compatible with other systems and dashboards

What I learned:

- The parser catches formatting issues early
- AI models sometimes need to be forced into compliance
- Adding enums improves clarity and prevents creative outputs
- JSON schemas make workflows safer and easier to debug

This feature is critical for enterprise-grade automation.

5.5 Combine Node – Multi-Input Merging

The Combine node (formerly the Merge node) is configured as:

- Mode: Combine
- Inputs: Planner, KYC result, credit score result, client data
- Combine by: Position
- Number of Inputs: 4

This produces a unified JSON structure containing all relevant data for the final decision.

What I learned:

- Combine nodes ensure deterministic merging
- Input order matters
- They are extremely useful in complex workflows with parallel branches
- Combination mode allows clean, readable transformations instead of custom code

5.6 Drag-and-Drop Mapping & Expressions

This was one of the most useful UX features I discovered.

I learned how to:

- Drag input fields directly into Set nodes
- Injecting JSON fields into prompts

5.7 Execution Testing, Mock Data & Logs

During development, I heavily used:

- Pinned input data
- Execute Step (node-by-node execution)
- The Data Pane (Schema, Table, JSON views)
- Execution logs

These tools allowed me to:

- Debug individual nodes in isolation
- Verify JSON structures before merging
- Validate whether IF conditions correctly evaluated
- Observe exactly what each node contributed to the workflow

What I learned:

- n8n provides extremely transparent debugging tools
- You can build complex flows incrementally
- Test mode prevents incorrect changes from affecting production

5.8 Low-Code / No-Code Capabilities of n8n (and Their Limits)

One of the strongest advantages of n8n is its low-code / no-code design philosophy. During this project, I was able to build a fully functional AI-driven onboarding workflow using mostly visual tools:

- Drag-and-drop mapping
- Set nodes
- The built-in AI Agent nodes
- IF conditions
- Simulation nodes
- Structured output parsers
- Visual branching and merging

This allowed me to focus on business logic, data flow, and compliance rules rather than traditional programming.

What "low-code/no-code" means in practice

n8n enables users to:

- Create workflows without writing traditional scripts
- Connect services using prebuilt integrations
- Manipulate data using visual tools
- Generate prompts and test AI agents directly in the UI
- Use expressions without having to write full JavaScript functions

For many simples to medium-complexity automations, this is more than enough. Examples include:

- Notifications
- Email processing
- Webhooks
- CRM updates
- AI summarization workflows
- Basic API interactions

The platform is genuinely accessible even to users without a developer background.

When Low-Code Is Not Enough: The Role of JavaScript and SQL

While n8n is powerful as a no-code tool, building precise, multi-step, enterprise-grade workflows often require deeper technical skills especially when integrating multiple third-party systems through APIs.

Based on this onboarding automation project, I noticed several situations where basic coding knowledge becomes necessary:

1. Advanced API Handling

Some APIs require:

- Custom authentication logic,
- Pagination,
- Token refresh flows,
- Cryptographic signatures,
- Non-standard content types.

These often require small JavaScript snippets in:

- Function nodes
- FunctionItem nodes
- Custom HTTP headers or parameters

2. Data Transformation at Scale

For certain datasets, the Set node is not enough. More advanced transformations require JavaScript, for example:

- Nested JSON restructuring
- Conditional logic inside arrays
- Dynamic field generation

- Dlattening complex API responses

3. SQL Knowledge for Database Integrations

When storing or querying structured onboarding data, SQL becomes essential:

- writing SELECT queries
- filtering by score thresholds
- aggregating onboarding results
- joining data across tables

n8n supports databases like Postgres, MySQL, SQLite but meaningful operations require SQL literacy.

4. Error Handling and Failover Logic

Complex workflows are often needed:

- try/catch via function nodes
- conditional retries
- dynamic error messages
- sanitizing malformed data

These require some JavaScript and logical thinking.

AI-Assisted Workflow Generation in n8n

One of the standouts features I used during this project is n8n's capability to generate workflows using AI. Instead of starting from a blank canvas, n8n allows you to create an entire workflow by simply describing your goal in natural language. For example, I could instruct:

"Build a client onboarding flow that evaluates nationality rules, triggers KYC and credit score checks, merges results, and outputs a final decision."

Based on this description, n8n automatically generates:

- A prototype workflow
- The key nodes
- Conditional logic structure
- Initial AI agent prompts
- Placeholders for API calls
- A visual layout of the automation

This dramatically speeds up the early stages of design.It gives you something to work with immediately rather than manually creating every step.

However, the real strength lies in the combination:

AI builds the structure → I fine-tune it manually

After n8n generated the initial flow, I improved it by:

- Rewriting the AI prompts
- Adding strict Structured Output Parsers
- Designing fallback nodes for skipped tools
- Refining IF expressions
- Improving testability
- Adjusting node parameters
- And reorganizing the workflow to match real business rules

This hybrid approach AI-assisted creation + manual refinement allowed me to work much faster while still ensuring accuracy, compliance, and reliability.
It is especially useful for complex flows where the basic structure is repetitive or lengthy to build manually.

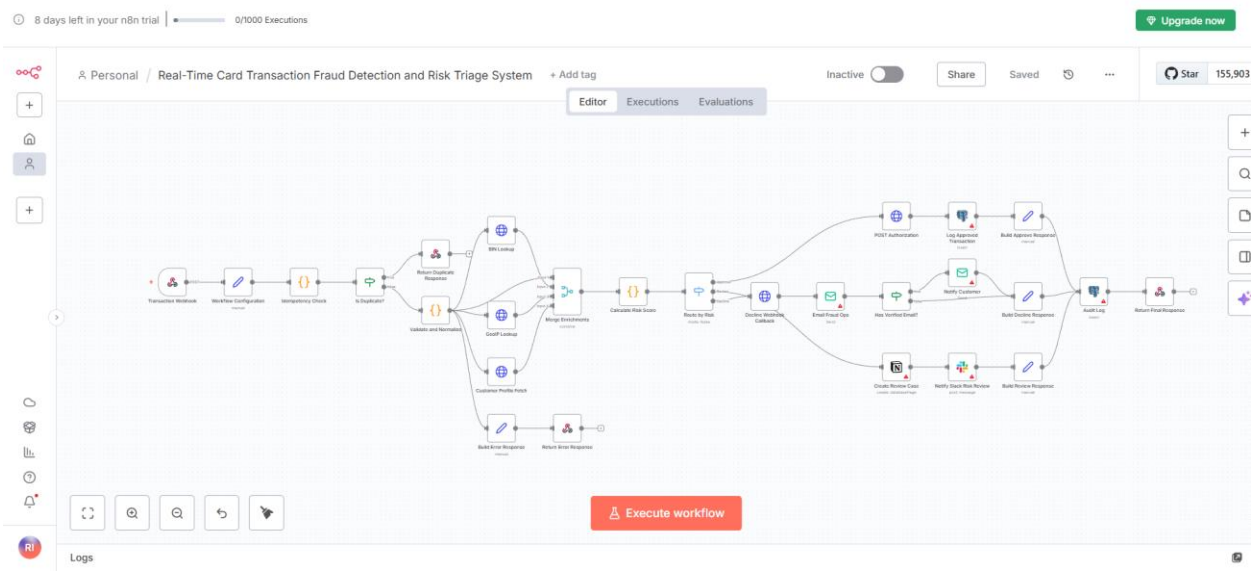
In summary, n8n's AI workflow builder served as:

- a brainstorming tool,
- a rapid prototyping assistant,
- and a structural scaffold
  for a workflow that I later converted into a robust and carefully engineered onboarding pipeline.

Example of prompt:

“Build an n8n fintech automation that triages incoming card transactions for fraud risk and routes outcomes. The workflow is triggered by a Webhook receiving JSON {txId, timestamp, amount, currency, merchantName, mcc, cardLast4, bin, channel, ip, customerId}; it must enforce idempotency (ignore duplicate txId), validate required fields, and normalize currency/amount. Enrich the event via BIN/IIN lookup (issuer, card type, country), GeoIP on ip, and a Customer Profile fetch (KYC tier, velocity in last 24h, chargeback history) from our REST API. Compute a risk score (0–100) using (a) deterministic rules (high-risk MCCs, cross-border + card-present mismatch, abnormal amount vs customer median, device/IP anomalies) and (b) optional AI/LLM sentiment/notes parser on recent support tickets to add ±5 risk. Classify outcome by thresholds: approve (<30), review (30–69), decline (≥70). For approve, POST an authorization recommendation to our processor and log to a Database/Sheet with latency metrics; for review, open a Case in our helpdesk (or Notion/Sheets row) with a compact evidence bundle and notify Slack #risk-review; for decline, send a webhook callback to the processor, email the fraud ops alias, and (if customer has verified email) trigger a templated notification advising they can contact support. All branches must append an audit record (original payload, enrichments, rules fired, final decision, timestamps, workflow version) and return a JSON response to the webhook caller like: {"txId":"...","decision":"review","risk":58,"reasonCodes":["CROSS_BORDER","AMOUNT_SPIKE"],"caseId":"...","latencyMs":1234}. Include retry with backoff for external calls, timeouts, circuit-breaker skip after repeated failures, and a Fail-safe path that returns {"decision":"review"} on unexpected errors so transactions never auto-approve when uncertain.”

Result:


8 days left in your n8n trial
0/1000 Executions
Upgrade now
Personal / Real-Time Card Transaction Fraud Detection and Risk Triage System
+ Add tag
Inactive
Share
Saved
Star 155,903
Editor
Executions
Evaluations
Execute workflow
Logs