SkillNavora GenAI Launchpad · Written by SkillNavora · Part 4 of the automation series · Uses fake data only

n8n Multi-Agent Onboarding: From Zero to Hosted

You will build a small new-hire onboarding system. An HR form sends a webhook to n8n. A Coordinator agent records the hire in a Google Sheet, asks an Access Agent to plan system access, and asks a Scheduling Agent to plan meetings. Everything uses **fake data** and a **dedicated test account**. Follow the steps in order, and check the "You should see" line after each one.

> **Safety first.** Never connect your personal Gmail, your main Google Drive or real employee data to a practice workflow. Create a test Google account and a test sheet. Email sending stays off until a person approves it.

1. The big picture

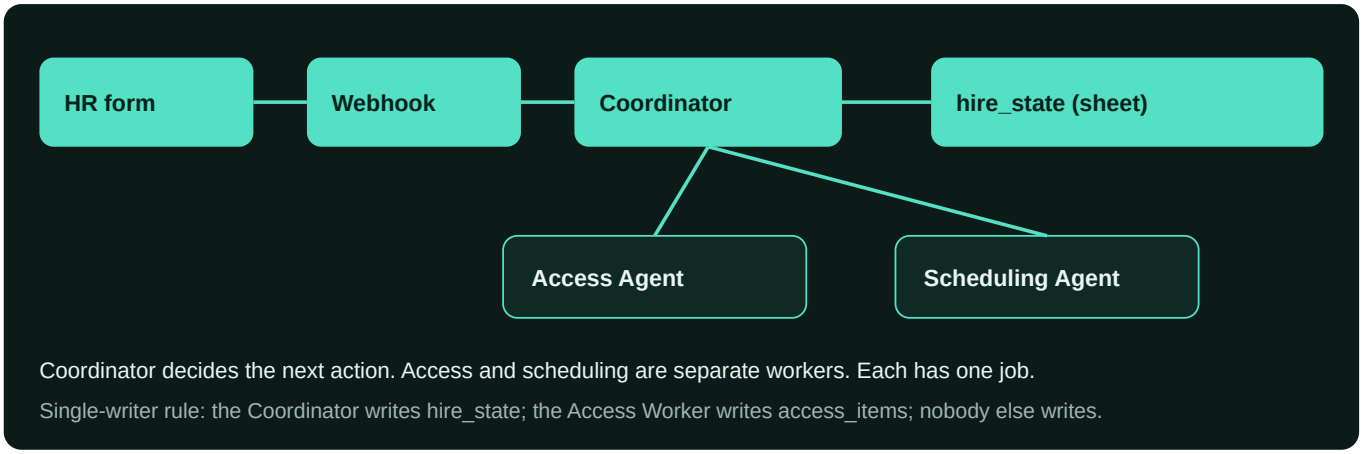


Figure 1. The onboarding system. Arrows show who calls whom.

2. Hosting n8n (choose one path)

n8n can run in three places. Pick one and keep the rest of the guide the same.

Option	Best for	What you manage
n8n Cloud (managed)	Learning, quick start	Nothing on the server; pay per plan
Self-hosted on a VPS (virtual server)	Full control, one fixed monthly server cost	Server updates, backups, HTTPS certificate
Local with Docker	Testing on your own computer only	Your computer must stay on; webhooks need a public address to receive real events

Self-hosting on a VPS: step by step

1. **Choose a VPS plan.** Hosting companies such as Hostinger sell VPS plans. Compare the monthly price, the RAM, and whether a one-click n8n template is included. Prices change; read the live cart before you pay. Hostinger's cart in the class showed a KVM 1 plan with a free domain for the first year, which is a promotion, not a permanent price.
2. **Point a domain or subdomain** (for example `n8n.example.com`) at the server's IP address.
3. **Install n8n** with the one-click template or with Docker. Set an encryption key and a strong admin password.
4. **Turn on HTTPS** so webhook URLs are secure.
5. **Back up** the n8n data folder and the encryption key separately.

You should see: the n8n login page at your domain, over HTTPS, and you can sign in to the editor.

3. Your first workflow: a Webhook test

A **webhook** is a doorbell. The workflow waits until something rings it. Instead of checking repeatedly for new hires, n8n waits, and the HR form sends a POST request when a hire is confirmed.

1. Click **Create workflow**, then add a **Webhook** node.
2. Set **HTTP Method** to POST, **Authentication** to None (for this practice), and **Respond** to Immediately.
3. Copy the **Test URL**. It looks like `https://your-n8n.example.com/webhook-test/<long-id>`.
4. Click **Listen for test event**.
5. In another tab, send a test submission (step 4 below shows the body to use), or paste this into a tool such as an HTTP client:

```
POST https://your-n8n.example.com/webhook-test/<long-id>
Content-Type: application/json

{
  "hire_id": "H-2026-0001",
  "candidate_name": "Test Candidate",
  "department": "Product",
  "role": "PM",
  "manager": { "name": "Test Manager", "email": "manager.test@example.com" },
  "start_date": "2026-09-01",
  "candidate_email": "candidate.test@example.com"
}
```

You should see: the Webhook node shows one item. Open the **JSON** view. Your fields sit under `body`. The `hire_id` is the field that identifies the person in every later step.

> **Test versus production.** The *test* URL works only while "Listen for test event" is open. The *production* URL works all the time, but only after you activate the workflow. Use the test URL while building, then switch the form to the production URL and activate.

Reading the webhook output

Part	What it holds	Use it for
headers	Who sent it, content type	Checking the request came from the form
params, query	URL parameters	Usually empty for a form POST
body	Your data	Everything your workflow needs
webhookUrl, executionMode	Which URL and mode ran	Confirming test versus production

Use the **Schema**, **Table** and **JSON** tabs in the output panel. The JSON view is the most reliable when you are learning.

4. The data contract

A **data contract** is the agreed list of fields and their meaning. Every part of the system uses the same names. Changing a name in one place breaks the others.

Field	Example	Rule
hire_id	H-2026-0001	Unique per hire. The primary key. Never reused.
candidate_name	Test Candidate	Required
candidate_email	candidate.test@example.com	Personal email; required until the work email exists
department, role	Product, PM	Must match a value in the access registry
manager.name, manager.email	Test Manager	Required
start_date	2026-09-01	Must be in the future
run_id	RUN-H-2026-0001-001	Generated by the Coordinator, one per attempt

5. The workbook: four tabs, one writer each

Create a Google Sheet named **onboarding_agents_workbook** using a test Google account. Each tab has **one writer**. This stops two parts of the system from overwriting each other.

Tab	Sole writer	Purpose
hire_state	Coordinator	One row per hire: overall, access and scheduling status
access_items	Access Worker	One row per hire per access item
meetings	Scheduling Worker (not built here)	One row per proposed meeting
access_registry	Hand-maintained (nobody writes it at runtime)	Which apps each role gets, and whether approval is needed

Status words

Tab and column	Allowed values	Terminal?
Overall, access and scheduling status	in_progress, complete, needs_attention	complete and needs_attention end the flow
access_items status	not_requested, requested, awaiting_approval, granted, not_granted	granted and not_granted are final
meetings status	pending, proposed, conflict, confirmed, not_scheduled	confirmed and not_scheduled are final

CSV templates (copy these into the tabs)

Use these headers exactly. Lookups are exact matches, so spelling and capital letters must match.

```
hire_state
hire_id,candidate_name,candidate_email,department,role,manager_name,manager_email,start_date,run_id,overall_status,access_status,sch

access_items
hire_id,run_id,app,level,status,approval_needed,updated_at

meetings
hire_id,run_id,role,meeting_name,duration_minutes,within_days,priority,status,proposed_time,updated_at

access_registry
row_key,app,level,role,drive_role,grant_mode,depends_on
Identity|create|PM,Identity,create,PM,,approval_required,
Identity|create|Designer,Identity,create,Designer,,approval_required,
Identity|create|Developer,Identity,create,Developer,,auto,
Slack|edit|PM,Slack,edit,PM,,auto,Identity|create
Slack|edit|Designer,Slack,edit,Designer,,approval_required,Identity|create
Slack|edit|Developer,Slack,edit,Developer,,auto,Identity|create
Jira|view|PM,Jira,view,PM,,auto,Identity|create
Jira|admin|Developer,Jira,admin,Developer,,auto,Identity|create
Figma|edit|Designer,Figma,edit,Designer,,approval_required,Identity|create
Onboarding Doc|view|PM,Onboarding Doc,view,PM,reader,approval_required,
Onboarding Doc|edit|Designer,Onboarding Doc,edit,Designer,writer,approval_required,
Onboarding Doc|view|Developer,Onboarding Doc,view,Developer,reader,auto,

meetings_template (the reference list, keep it in access_registry or a separate tab)
role,meeting_name,duration_minutes,within_days,priority,purpose
PM,Manager 1:1,60,2,1,Align on 30-60-90 day plan and immediate priorities
PM,Product Vision Walkthrough,45,5,2,Walk through product strategy, roadmap and OKRs
PM,Stakeholder Introduction,30,7,3,Meet key cross-functional stakeholders
Designer,Manager 1:1,60,2,1,Align on design principles and current projects
Designer,Design System Walkthrough,45,5,2,Walkthrough of the design system and tools
Developer,Manager 1:1,60,2,1,Align on engineering standards and sprint cadence
Developer,Tech Stack Walkthrough,45,5,2,Deep dive into the tech stack and dev environment
```

> **Open decisions.** The class workbook flagged one cell to decide before going live: the Developer grant mode for the Onboarding Doc. The sample above uses `auto` for read-only access; confirm your own policy. Delete any example rows before using the sheet for real.

Create the sheet

1. Create the spreadsheet and rename the first tab to `hire_state`.
2. Add the other tabs and paste each CSV header row.
3. Paste the registry rows into `access_registry`.

4. Share the sheet only with your test account.

6. The Google Sheets connection

1. In n8n, open **Credentials** and add a **Google Sheets** credential. Sign in with the test account and approve the access. Give only Sheets access.
2. Add a Google Sheets node, choose **Append Row**, and select the document and tab.
3. Map each field from the webhook body to a column. Use expressions such as `{{ $json.body.hire_id }}`.

You should see: after one test submission, a new row appears in `hire_state`. If it does not, check the document name, the tab name and the credential.

7. The Coordinator agent

The Coordinator is an AI Agent node with a chat model (Anthropic in the class), an **Update row** Google Sheets tool, and two tools that call the Access and Scheduling agents. Its job is to decide the next action, not to do the work itself.

Tool descriptions matter

The agent reads each tool's description to decide when to use it. Write it as a rule: "Update the hire_state row for a given hire_id. Use this to set overall_status, access_status and scheduling_status." Vague descriptions cause wrong calls.

Use the narrowest operation

Google Sheets offers Append, Update, Append or Update, Clear, Create and Delete. Give the agent **Update Row** only. Never give it Clear or Delete. Least permission is the rule.

System message (the instructions)

```
Role: You are the Coordinator for new-hire onboarding.
Steps:
1. Read the hire from the input. Confirm hire_id is present; if not, stop and set needs_attention.
2. Call the Access Worker with these named fields, each separately:
   hire_id, candidate_name, role, manager_name, manager_email, start_date, candidate_email,
   run_id (format RUN-{hire_id}-001).
   Do not join the fields into one string.
3. Call the Scheduling Worker with hire_id, candidate_name, role and start_date.
4. Update the hire_state row with the statuses you received.
Output format (JSON only):
{ "next_action": "dispatch | wait | close", "reasoning": "one sentence" }
Never send email. Never change access directly.
```

Why this helps: named fields prevent one long string that the workers cannot parse. Asking for JSON output makes the next node reliable.

Turn on **Return Intermediate Steps** while building, so you can see each tool call. Turn it off in production if the output gets long.

8. The Access Agent (a separate workflow)

The Access Agent is its own workflow, started by the Coordinator through an **Execute Workflow** or "Call workflow" tool. It receives the named fields from step 7.

1. **Start:** "When Executed by Another Workflow", with the input fields listed (run_id, hire_id, role, manager_name, manager_email, start_date, candidate_email, dispatched_at).
2. **Idempotency check:** a Get Row(s) step reads `access_items` filtered by `hire_id = {{ $json.hire_id }}`. If rows already exist for this hire, stop. This stops the same hire being processed twice.
3. **If node:** true branch means "already processed, stop"; false branch continues.
4. **AI Agent:** reads the role from the input, and uses the **Registry Lookup tool** (read `access_registry`) to find each app for that role.
5. **For each app:** if `grant_mode` is `auto`, write a row with status `requested`. If `approval_required`, write `awaiting_approval` and do not grant.
6. **Write Row tool:** appends to `access_items` only. Never writes to `hire_state`.

7. **Gmail (optional):** send a message only after a person approves. For practice, leave the Gmail tool disconnected.

You should see: running the same hire twice writes rows only once. The second run stops at the If node.

9. The Scheduling Agent (concept only in this build)

The Scheduling Agent reads `meetings`, proposes times for each role's meetings within the `within_days` window, and writes `proposed` rows. It must never confirm a meeting without a person. In this guide, build the Coordinator and Access Agent first; add scheduling once those work.

10. Duplicate prevention and error paths

Risk	Prevention
The same form submitted twice	Check `hire_id` exists before appending a new hire row.
Missing email or hire_id	Validate first; set `needs_attention` and stop.
Wrong role	Look up the role in the registry; if not found, stop with "needs_attention".
Webhook delivery fails	Check the executions list; re-send the test event; use a retry only for safe steps.
Access granted without approval	Rows with `approval_required` stay `awaiting_approval`.

Add an **Error Trigger** workflow that writes "needs_attention" and the error message to `hire_state`, so failures are visible in the sheet, not hidden in logs.

11. Monitor the workflow

- Open **Executions** for each run. A green run may still have made a wrong decision, so read the output of each node.
- Track the **failure rate**. One failure in one production run is a 100 percent failure rate, so judge trends over many runs.
- Track **tokens per run** from the agent's log, and compare with the cost guide in this series.
- Track **run time**. Multi-agent runs take seconds to minutes; set a timeout.

12. The HR Portal: intake form and dashboard

The HR Portal is a small web app (built in Lovable in the class). It has two screens.

Screen 1: New Hire Intake

Section	Fields	Rules
Candidate	Candidate name, candidate personal email	Name required; email must look like an email
Assignment	Department, role, manager	Department and role come from a list; manager from a list
Timing	Start date	Must be in the future
Action	Confirm hire	Generates the hire_id, then sends the webhook POST

Screen 2: Dashboard

A table of hires with columns: hire_id, name, role, start date, overall status, access status and scheduling status. Colour the status (in_progress, complete, needs_attention). Add a "Show webhook status" or "Export as CSV" button if you want operational help.

Build it in Lovable (outline)

1. Create a project and describe the two screens in plain English: "An HR portal with an intake form and a dashboard, using fake data only."
2. Ask for the form fields exactly as in the table above.
3. When the form is ready, paste the **test** webhook URL and ask: "Send the form data to this URL as JSON after Confirm hire."
4. Submit one fake hire and check that it appears in the n8n test panel.

5. Switch to the **production** URL, and activate the workflow in n8n.

> **Do not use real personal data** in a demo app. Fake names and test addresses only, and do not publish a live URL that accepts real submissions.

13. Build order (do not skip steps)

1. Host n8n and sign in.
2. Webhook test: send one fake body and read the JSON.
3. Write one row to `hire_state`.
4. Add the Coordinator with the system message, using a test model key.
5. Build the Access Agent, test duplicates, then connect it.
6. Add the form and the dashboard.
7. Switch to production URLs and activate.

14. Test checklist

Test	Pass when
Valid hire	One row in hire_state, access rows written
Same hire submitted twice	No duplicate access rows
Missing email	needs_attention, no access granted
Unknown role	needs_attention with a reason
Approval-required app	awaiting_approval, nothing granted
Webhook down	Failure visible in executions and in the sheet

15. Quick check

1. Why is `hire_id` the most important field?
2. Which tab does the Access Worker write to, and which does it never write?
3. Why should the Coordinator never be given the Delete operation?
4. What does the If node in the Access Agent prevent?

Answers: 1. It identifies the hire in every step and prevents duplicates. 2. Writes `access_items`; never `hire_state`. 3. Least permission; a mistake could erase the workbook. 4. Processing the same hire twice.

16. Glossary

Term	Plain meaning
Webhook	A URL that starts a workflow when something posts to it.
Test versus production URL	Test works only while you are listening; production works once the workflow is active.
Data contract	The agreed field names and meanings.
Single writer	Only one part of the system may write to each tab.
Idempotency	Running the same request twice gives the same result, with no duplicate rows.
Registry	The reference table of apps and approval rules per role.

- n8n documentation: docs.n8n.io (webhook and Google Sheets node pages).
- Your hosting provider's VPS documentation for installing Docker and HTTPS.

Part 2. Every n8n option you need, explained

This part is a reference. You can read it once, then come back when you meet a node you do not recognise. Node names follow n8n's usual labels; if a label differs slightly in your version, look for the closest match.

2.1 Words you will see on the canvas

Word	Meaning	Example
Workflow	The whole automation: a chain of nodes.	Onboarding workflow
Node	One step. Each node does one job.	Google Sheets: Append row
Trigger	The node that starts the workflow.	Webhook, Schedule, Manual
Connection (wire)	The line that passes data from one node to the next.	Webhook → Sheets
Item	One piece of data flowing through a wire. A node runs once per item unless told otherwise.	One hire = one item
Execution	One complete run of the workflow, with its inputs and outputs saved.	Run for hire H-2026-0001
Credential	A saved, encrypted login for an app.	Google Sheets account (test)
Expression	A formula inside a field, written between double curly braces.	`{{ $json.body.hire_id }}`
Sub-node (tool, model, memory)	A node that plugs into an AI Agent from below.	Anthropic Chat Model, Google Sheets tool

2.2 Trigger nodes: how a workflow starts

Trigger	Starts when	Use it for	Watch out
Manual trigger	You press Execute workflow	Building and testing	Not automatic; never leave it as the only trigger in production
Webhook	An app sends an HTTP request to the node's URL	Forms, apps, other systems pushing events	Use the test URL while listening; the production URL needs the workflow active
Schedule	At a time or interval you set	Daily reports, periodic checks	Time zone; each run is an execution that costs resources
Google Sheets trigger (polling)	New or changed rows appear	Reacting to new spreadsheet rows	Polls on an interval, so delays are normal
Execute Workflow Trigger ("When Executed by Another Workflow")	Another workflow calls it	Sub-workflows such as the Access Agent	Pass the fields you need explicitly
Error Trigger	Another workflow fails	Central error handling and alerts	Set it up in a separate workflow and link it in settings

2.3 Action and logic nodes you will use most

Node	What it does	Key setting
Google Sheets	Read rows (Get row(s)), add rows (Append row), change rows (Update row), upsert (Append or Update)	Document and sheet selection; the Operation dropdown
HTTP Request	Calls any web API (GET, POST, PUT, DELETE)	URL, method, headers, body; use a credential for keys
If	Splits items into true and false branches	Condition: field, operator, value
Switch	Routes items to many branches by value	Rules per branch; add a fallback output
Merge	Joins two branches back together	Mode: append, combine by key, or choose input
Set (Edit Fields)	Adds, renames or removes fields	Keep only the fields the next node needs
Code	Runs JavaScript or Python on the items	Use for exact rules (dates, validation); keep it small
Filter	Keeps only items that match a condition	Condition builder
Wait	Pauses for a time or until a webhook resumes it	Use it for approval waits
Loop Over Items (Split in Batches)	Processes items in groups	Batch size
Gmail	Sends or reads email	Use a test account; keep sending behind approval
Slack, Notion, and other app nodes	Post messages or change app records	Test channel or test workspace only

Stop and Error	Ends a branch on purpose with a reason	Message text for the log
Sticky note	A comment on the canvas	Document each section for future you

2.4 AI nodes

Node	Role	Notes
AI Agent	The decision-maker: reads its input, uses tools, loops until done	Needs a chat model; tools are optional but each one widens risk
Chat Model (Anthropic, OpenAI, Google, Groq and others)	The model the agent talks to	Choose by speed, quality and cost (tokens guide)
Memory	Remembers earlier turns in the same conversation	Off for stateless processing; on for chat
Tool (sub-node)	An action the agent may choose	Give each tool a clear description the agent can read
Output Parser	Forces a structured output such as JSON	Prefer structured output for anything another node reads
Basic LLM Chain	One model call with no tool loop	Use it when no decision-making is needed (cheaper and more predictable)

> **Choose the simplest node that works.** A fixed template is a Set node. A single model call is a Basic LLM Chain. Only a task that must choose steps and tools needs an AI Agent.

Part 3. Build your own n8n workflow: a 12-step recipe

Use this recipe for any workflow, not only onboarding. Each step has a check. Do not move on until the check passes.

1. **Write the goal in one sentence.** Example: "When HR confirms a hire, record it in the sheet and ask for access planning."
   Check: you can say what "done" looks like.
2. **List the inputs and outputs** with names and types. Use the data contract table from Part 1.
   Check: every input has a name and an example value.
3. **Choose the trigger**: Manual for building, Webhook for real events, Schedule for time-based work.
   Check: you know whether it runs once per event or on a timer.
4. **Add the first action node** (for example Google Sheets: Append row) and connect it to the trigger.
   Check: Execute the node. One row appears in the test sheet.
5. **Map fields with expressions**, for example `{{ $json.body.candidate_name }}`.
   Check: the mapped preview shows the right values, not "undefined".
6. **Add validation** with an If node: required fields present, email shape, start date in the future.
   Check: a bad input takes the false branch and writes "needs attention".
7. **Add the decision or AI step** only if a rule cannot do it. Start with a Basic LLM Chain before an AI Agent.
   Check: the output is structured and parseable.
8. **Add the external action last**, and put it behind an approval (Wait node or a manual step).
   Check: nothing external happens without approval in the test run.
9. **Add an error path**: an If on error, a Stop and Error node, or the Error Trigger workflow.
   Check: force a failure (a wrong sheet name). The failure appears in the log and the sheet shows "needs attention".
10. **Make it idempotent**: check whether the item already exists before writing it.
    Check: running the same input twice leaves one row.
11. **Test with five inputs:** normal, missing field, duplicate, bad date, and a tool failure.
    Check: each result matches the expected column in your test plan.
12. **Switch to production**: copy the production URL, update the form, activate the workflow, and run one fake end-to-end test.
    Check: the production execution shows success and the sheet shows one row.

Part 4. The multi-agent onboarding build, node by node

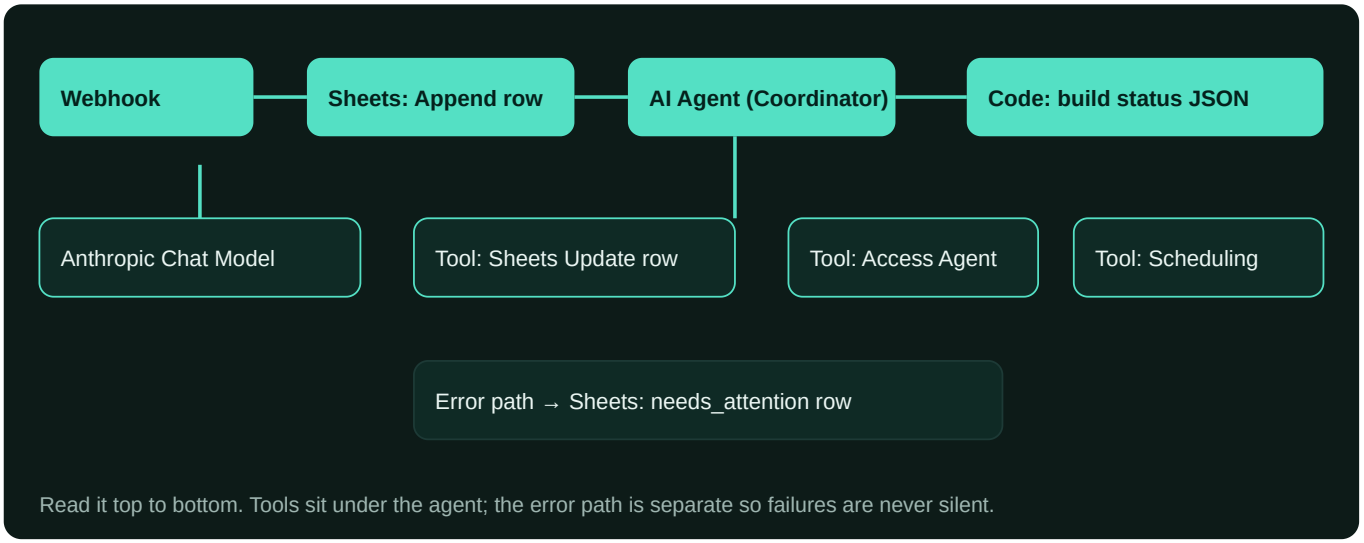


Figure 2. The Coordinator as built in n8n. Sub-nodes plug into the agent from below.

4.1 Coordinator (AI Agent)

1. Add an **AI Agent** after Sheets: Append row.
2. Attach **Anthropic Chat Model** to the Chat Model connector. Choose a model; start with the smallest that passes your test set.
3. Attach the tools: **Google Sheets (Update Row only)**, **Call Access Agent** (Execute Workflow tool), and **Call Scheduling Agent**.
4. Write each tool's description as a rule. Example for Sheets: "Update the hire_state row for a given hire_id. Use it to set overall_status, access_status and scheduling_status."
5. Paste the system message from Part 1 section 7, and leave Memory off.
6. Turn on **Return Intermediate Steps** while building.
7. Run once with the fake hire. Open each tool call in the output and confirm the named fields arrived separately.

4.2 Access Agent (separate workflow)

Node	Setting	Why
When Executed by Another Workflow	Input fields: run_id, hire_id, role, manager_name, manager_email, start_date, candidate_email, dispatched_at	Explicit input; nothing implicit
Sheets: Get row(s)	Document onboarding_agents_workbook, sheet access_items, filter hire_id = {{ $json.hire_id }}	Idempotency: look before you write
If	Condition: number of rows is greater than 0	True means already processed: stop
AI Agent with Registry Lookup tool	Tool reads access_registry for the role	Rules come from the sheet, not from memory
Code (or Set)	Turn each registry row into one access item with status requested or awaiting_approval	Deterministic: approval rules are code, not model choice
Sheets: Append row	Sheet access_items only	Single-writer rule
Gmail (optional, disconnected in practice)	Only after a person approves	No external message without approval

> **Single-writer rule.** The Access Worker writes access_items. The Coordinator writes hire_state. Neither writes the other's tab. When two parts write one table, rows overwrite each other and the status becomes unreliable.

4.3 Scheduling Agent (build after the first two work)

It reads the meetings template, proposes times inside each meeting's within_days window, and writes meetings rows with status proposed. A person confirms. It never sends invitations on its own in this guide.

Part 5. Hosting and operations, done safely

Topic	What to do	Why
Environment variables	Keep the encryption key, database settings and base URL in environment variables, not in workflow nodes	Changing one value changes every workflow
Credentials	Create them in n8n's credential store; use test accounts; least permission	Secrets stay encrypted and out of exported files
HTTPS	Put n8n behind HTTPS with a domain	Webhooks carry data; plain HTTP exposes it
Updates	Update on a schedule; read release notes; test in a copy first	Updates can change node behaviour
Backups	Back up the database and the encryption key separately	Without the key, saved credentials cannot be read
Logs	Review executions daily during testing; keep logs without personal data	Failures must be visible
Access control	One owner account; add test users only when needed	Fewer people can change production
Exports	Export workflows as JSON to version control, without credentials	History and rollback

Part 6. Test plan with pass criteria

Test	Input	Pass when
Happy path	Complete fake hire	One hire row; access rows; status complete or awaiting_approval
Missing field	No hire_id	needs_attention; no access rows
Duplicate hire	Same hire_id twice	One set of access rows
Malformed JSON	Body is not JSON	Request rejected or error row written; no crash
Unknown role	role = Analyst	needs_attention with a reason
Failed tool call	Wrong sheet name	Error path runs; sheet shows needs_attention
Timeout	Slow tool (simulate with Wait)	Run stops at the limit and is logged
Retry	Transient failure	Retry twice at most, then needs_attention
Human approval	Approval-required app	awaiting_approval; nothing granted until a person changes it

Part 7. Measure the workflow

Metric	How to calculate	Target to aim for while learning
Success rate	Successful executions ÷ all executions	Rising toward 100 percent on the test set
Failure rate	Failed executions ÷ all executions	Falling; investigate every failure
Duplicate rate	Duplicate hire rows ÷ hire rows	Zero
Time to complete	Median time from webhook to hire_state complete	Seconds for a test run
Escalation rate	needs_attention rows ÷ hire rows	Known and explained, not zero by force
Cost per run	Model tokens per run × price (tokens guide)	Recorded for every test batch

Part 8. Extended quick check and answer key

1. Which trigger starts a workflow when HR submits a form?
2. Where should a fixed approval rule live: in a prompt or in a registry and code?
3. Why put an If node before writing the access items?
4. A test run shows a green node but a wrong status. What do you check first?

Answers: 1. Webhook. 2. In the registry and code, so it does not depend on the model's memory. 3. It stops duplicates. 4. The node's output: the actual values it passed on, not just its colour.

Part 9. Final checklist before you go live

Item	Done
Test accounts only; no personal Gmail or Drive	[ ]
Fake data only in every test	[ ]
Test URL worked; production URL is set in the form	[ ]
Duplicate check present and tested	[ ]
Error path visible in the sheet	[ ]
Human approval before any external action	[ ]
Credentials stored in n8n, not in nodes or exports	[ ]
Backups of database and encryption key taken	[ ]

Part 10. Labs: five practice exercises with fake data

Lab 1: webhook in, row out (30 minutes)

1. Create a workflow with a Webhook node (POST, test URL) and a Google Sheets Append row node to a test sheet.
2. Send the fake hire body from Part 1. Map `hire_id`, `candidate_name` and `department`.
3. **Pass when:** one row appears, with the right values in the right columns, and the execution is green.

Lab 2: validation branch (30 minutes)

1. Add an If node after the webhook. Condition: `hire_id` is not empty AND `candidate_email` contains "@".
2. True branch: Append row to `hire_state`. False branch: Append row to a sheet named `review_queue` with the reason.
3. **Pass when:** a valid body lands in hire_state and an invalid body lands in review_queue, and nothing is lost.

Lab 3: duplicate check (40 minutes)

1. Before the append, add a Google Sheets Get row(s) node filtered by `hire_id`.
2. Add an If node: row count equals 0 goes to the append branch; otherwise go to a "duplicate" branch that writes a note to the review queue.
3. **Pass when:** sending the same body twice leaves one hire row and one review note.

Lab 4: approval wait (40 minutes)

1. Add a Wait node after the access plan is written, set to wait for a webhook resume.
2. Write the approval request to the sheet with status `awaiting_approval`.
3. Send a second test webhook to the resume URL, with the field `approved: true`.
4. **Pass when:** the status changes to `granted (simulated)` only after the second request, and never before.

Lab 5: forced failure and error route (30 minutes)

1. Rename the target sheet to a wrong name, then run the workflow.
2. Connect an error route that appends a row to `review_queue` with the error message.
3. **Pass when:** the failure appears in the executions list, the review queue shows the message, and the workflow does not report success.

Part 11. Troubleshooting table

Symptom	Most likely cause	What to check first
Webhook shows no data	Listening stopped, or wrong URL	Click "Listen for test event" again; confirm test versus production
Field shows "undefined"	Expression points to the wrong path	Look at the JSON view; the data sits under `body`
Sheet row has the wrong column	Mapping to the wrong column header	Compare header names in the sheet with the mapping
Agent calls the wrong tool	Tool description is vague	Rewrite it as a rule with the trigger condition

Agent loops	No stop rule	Set a maximum step count and a clear finish condition
Production run does nothing	Workflow not active	Activate the workflow; check the production URL in the form
Credential error	Expired or wrong account	Reconnect the test account; check the scopes granted
Duplicate rows	No idempotency check	Add Lab 3's check before every append

Part 12. Cost, speed and model choice for workflows

- **Use the cheapest step that works.** Set and If nodes cost nothing extra in model terms; AI nodes cost tokens on every run.
- **Measure per run:** record the model tokens reported by the agent for each test batch, then apply the tokens-guide formula.
- **Reduce input:** pass only the fields the agent needs; never the whole execution history.
- **Cap output:** set a maximum output length for every agent call.
- **Watch run time:** an agent that waits on several tools can take minutes. Set a timeout and log it.

Part 13. Full glossary

Term	Plain meaning
Node	One step in a workflow.
Trigger	The node that starts a workflow.
Execution	One saved run of a workflow.
Credential	A saved, encrypted login for an app.
Expression	A formula inside a field, such as `{{ $json.body.hire_id }}`.
Sub-node	A model, tool or memory node attached to an agent.
Webhook	A URL that starts a workflow when something posts to it.
Idempotency	Running the same input twice gives one result.
Single writer	Only one part of the system writes each table.
Error route	A path that runs when a step fails, so failures are visible.

Part 14. Quiz (answers follow)

1. Which node holds the hire data sent by the form?
2. Where is a fixed approval rule stored, and why?
3. What should happen to a duplicate hire?
4. A production run shows nothing. What do you check first?
5. Why should an AI node have a maximum output length?

Answers: 1. The Webhook node, under `body`. 2. In the registry and code, so it does not depend on the model's memory and can be audited. 3. It is recorded in review and no second set of rows is written. 4. Whether the workflow is active and the production URL is in the form. 5. It caps cost and keeps the answer a predictable length.

Part 15. Build-your-own planning sheet

Question	Your answer
What event starts my workflow?	[ ]
What fields must it contain?	[ ]
What is the final output, and where does it go?	[ ]
Which step needs judgement (AI), and which needs a rule (If, Set, Code)?	[ ]
Which step changes something outside the system, and what approval does it need?	[ ]

What happens on failure, and who is told?	[ ]
How do I prevent duplicates?	[ ]
How will I test it with fake data?	[ ]

Part 16. Industry workflows you can build with the same pattern

The onboarding build uses one pattern: an event starts the workflow, a rule or agent decides, a sheet records state, and a person approves anything external. Use the same pattern for other industries. Each example below uses fake data and a test account.

Industry	Trigger	Rule or agent step	State record	Human approval before
Retail	New product listing submitted by a seller form	Check price and category against the policy sheet; draft a description with a 50-word cap	Listing sheet with status draft, approved, rejected	Publishing the listing
Clinics	Appointment request webhook (fake patient data)	Check slot availability in a calendar sheet; propose two times	Booking sheet with status proposed, confirmed, declined	Any confirmation message to a patient
Logistics	Shipment exception from a tracking feed	Classify the exception (delay, damage, missing) using a fixed list	Exceptions sheet with status open, resolved, escalated	Rerouting or refunds
Education	Quiz submission from a form	Score against a rubric; draft feedback	Results sheet with status scored, reviewed	Sending feedback to a learner
Public service	Service request form	Route to a department from a fixed list	Requests sheet with status routed, acknowledged, closed	Any decision on eligibility (never automated here)

> **Eligibility, diagnosis and pricing decisions stay with a qualified person.** Automation may prepare the information; it must not make the decision.

16.1 Retail example, node by node

1. **Webhook** (POST): receives the listing form, test data only.
2. **If**: price is greater than zero and category is in the list. False goes to a review sheet.
3. **Sheets: Get row(s)** on the policy sheet for the category rules.
4. **AI Agent** (or a Basic LLM Chain) drafts a description under the word cap. Output is structured: title, description, word count.
5. **Code**: checks the word count and removes any price or claim from the draft.
6. **Sheets: Append row** to the listing sheet with status draft.
7. **Wait** for approval (a second webhook from the reviewer's test form).
8. **Sheets: Update row** to status approved. Publishing to a real store is out of scope for this lab.

Pass when: the draft never includes a price, the word count is under the cap, and nothing reaches status approved without the second webhook.

Part 17. Production readiness, in depth

Area	Question to answer	Evidence you need
Reliability	What happens when a tool is down?	Test with the tool disconnected; error route runs
Data protection	Which personal fields are stored, and for how long?	Field list with a retention rule
Access	Who can edit the workflow and the sheets?	Named owners; least permission on each credential
Change control	How is a change tested before it goes live?	Test run on fake data; export of the workflow JSON, without credentials
Monitoring	Who sees a failure, and how fast?	Error route writes to a visible sheet; daily review during the first week
Cost	What does one run cost in model tokens?	Recorded tokens per run; the tokens-guide formula
Rollback	How do you return to the last good version?	Saved export of the previous version

17.1 Monitoring routine

- **Daily (first week):** open the executions list; read the output of any red or unusual run; log the cause in the review sheet.

- **Weekly:** count success rate, duplicate rate, escalation rate and median time; compare with last week.
- **After any change:** run the five-input test set again before you trust the workflow.

Part 18. Security checklist for automations

Risk	Control
Credential leaks through exports	Export workflows without credentials; store exports in version control with no secrets
A public webhook accepts anything	Add an authentication check (header or signed token) before production; reject unsigned requests
Prompt injection through form text	Treat form text as data, never as instructions; keep the system message separate; limit agent tools
Over-broad tool access	One permission per tool; read-only where possible
Personal data in logs	Log IDs and statuses, not names or emails
Accidental external messages	No live email or chat node until a human approval step is in place and tested

Part 19. Build a second workflow alone (exit exercise)

1. Choose one industry from Part 16 and write the goal in one sentence.
2. Write the data contract: at least five fields, with one unique ID.
3. Build the trigger, one rule, one state record and one approval step, without an AI agent.
4. Add an AI step only where a rule cannot decide, and cap its output.
5. Run five fake inputs and fill in the test table from Part 6.
6. **Pass when:** the workflow passes the five tests, the duplicate check holds, and the approval step is required before any external action.

Part 21. Credentials and the approval gate, in practice

Credential	Test version	Production rule
Google Sheets	Test Google account, test sheet only	Service account with access to one named sheet
Model provider key	Test key with spending limit	Key stored in n8n credentials; rotated on a schedule
Email (Gmail)	Not connected during build	Connected only after the approval node is tested
Chat (Slack)	Private test channel	Named channel; messages contain IDs, not personal data

21.1 The approval gate, built and tested

1. After the access plan is written, add a **Wait** node set to resume on a webhook.
2. Write the request to the sheet with status `awaiting_approval` and the approver's name.
3. Send a test webhook to the resume URL with `approved: true` from the approver's test form.
4. **Pass when:** the status changes only after the resume call, and a second resume call with `approved: false` writes `not_granted`.

Part 22. Common design mistakes and fixes

Mistake	Why it hurts	Fix
One giant workflow for everything	Hard to test; one change breaks all	Split into sub-workflows with explicit inputs
Agent does arithmetic or date checks	Models make small errors	Use Code or Set nodes for exact rules
Status written by two parts	Rows overwrite each other	Single-writer rule per table
No stop rule on an agent	Loops and spends tokens	Maximum steps and a finish condition
Test data mixed with production	Fake records reach real sheets	Separate sheets and credentials for test and production

Part 23. Frequently asked questions

Question	Answer

Why use n8n instead of only an AI chat?	n8n runs the same steps every time, records state, and waits for approval; a chat does not.
Is an AI Agent always needed?	No. Most steps are rules. Use an agent only where a rule cannot decide.
Why keep a sheet if n8n has executions?	Executions show runs; the sheet shows the business state a team can read and act on.
What if the webhook URL changes?	Update the form to the production URL and activate the workflow again; test first.
How do I stop a runaway agent?	Deactivate the workflow, then fix the step limit and the stop rule before re-enabling.