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

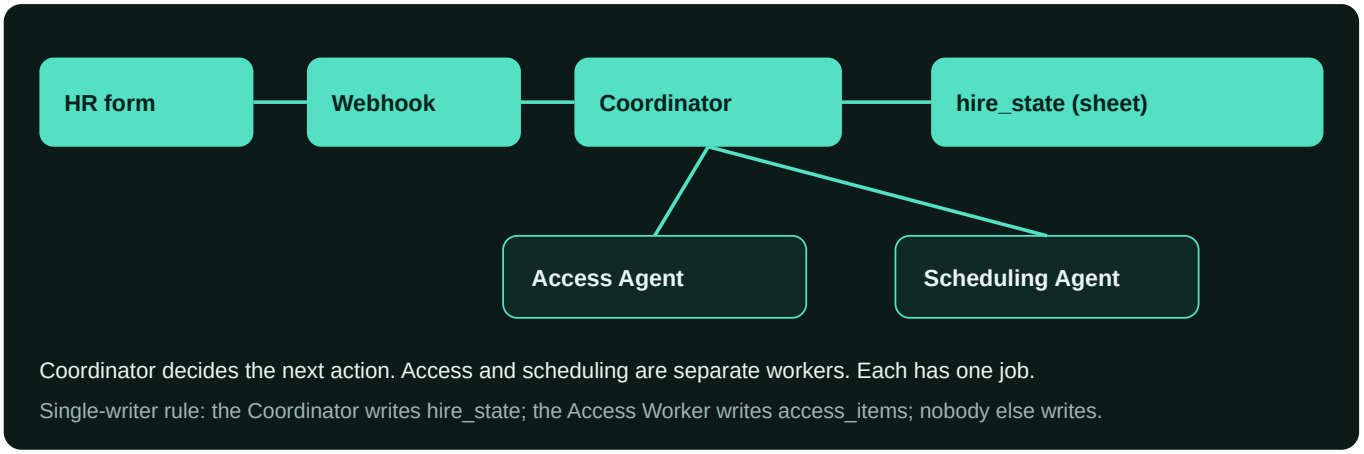


Figure 1. The onboarding system. Arrows show who calls whom.

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.