Google Sheets Design –

The solution is implemented as a single Google Spreadsheet containing multiple, purpose-specific tabs. This structure centralizes data, simplifies permissions, and makes it easy for n8n to orchestrate workflows across a single document.

---

1. Master Dashboard

Role:
Primary system of record for all projects and the main control interface for the client.

Key columns:

- ProjectID – Unique identifier for each project; used as the primary key for all synchronisation logic.
- Referred By / Pricing By – Dropdown fields sourced from a Config sheet to standardize who referred and who is pricing the job.
- Platform Name / WebLink – Identify the originating platform (e.g., BuildingConnected) and store a direct link to the opportunity.
- GC, GC Rep, GC Rep Contact – Captures the general contractor and key contact details, often auto-filled by automation.
- Project Name, Scope, Project Files – Core descriptive information and file links (e.g., shared drive folders).
- Project Due Date – External due date coming from the platform or client.
- Takeoff Assigned By / Takeoff Assigned To – Assignment controls indicating who allocated the work and which developer is responsible.
- Date Assigned – Assignment timestamp, automatically set when a developer is first selected.
- Exp Anticipated Date – Internally calculated expectation date (e.g., due date minus 10 days).
- Takeoff Received Date – Actual date when the completed takeoff is received back from the developer.
- Dollar Value – Estimated or final value associated with the opportunity.
- Status / Completed? – Normalized workflow state (Pending, In Progress, Completed, Cancelled / Lost, etc.) plus a simple Yes/No completion flag.
- Notes / Latest Update, Last Updated – Narrative progress updates and a timestamp of the last change, providing operational visibility.

This sheet drives all automations; all other sheets derive from or synchronise back to the Master Dashboard.

2. Developer Sheets (Dev_MK, Dev_MD, Dev_LA, Dev_Indiv1, …)

Role:
Individual work queues for each developer/estimator, tailored for day-to-day execution while remaining tightly linked to Master.

Structure:
Each developer tab (e.g., `Dev_MK`) contains:

- A subset of read-only fields mirrored from Master: `ProjectID`, `ProjectName`, `WebLink`, `ProjectFiles`, `GC`, `DateAssigned`, `Exp Anticipated Date`.
- A controlled set of editable fields for the developer:
    - Status – Developer's current working state (Pending, In Progress, With Competitor, On Hold, Completed, Cancelled / Lost).
    - Completed? – Simple Yes/No indicator when the takeoff is fully delivered.
    - Dollar Value – Optional estimate or final value, entered by the developer.
    - Notes / Latest Update – Free-text commentary on progress, issues, or handover notes.

n8n synchronises these changes back to the Master Dashboard so that management views remain accurate without requiring developers to touch Master directly.

3. Active_Projects View

Role:
Operational view focusing exclusively on work that is still in flight.

Implementation:

- Implemented as a dedicated tab (`Active_Projects`) using a FILTER formula that selects rows from the Master sheet where:
    - Completed? = "No", and
    - Status is one of `Pending`, `In Progress`, `With Competitor`, or `On Hold`.

This view provides the client with a real-time, clutter-free list of all active engagements, without manually hiding or moving rows.

4. Completed_Projects View

Role:
Historical and reporting view for closed work, allowing easy analysis of completed or lost opportunities.

Implementation:

- Implemented as a separate tab (`Completed_Projects`) with a FILTER formula over the Master sheet that includes rows where:
  - **Completed? = "Yes"`, or
  - Status is `Completed` or `Cancelled / Lost`.

Projects automatically flow out of the active view into this completed archive once developers or the client mark them as completed, ensuring that the Master Dashboard remains a single source of truth while the views provide curated slices for day-to-day use and reporting.

https://docs.google.com/spreadsheets/d/1EFDIjX7rpgw7A5xraILU-5kDIrxpT4wnufxWhKF_F58/edit?usp=sharing

Flow 1 – WebLink Auto-Fill (n8n + Google Sheets)

Purpose

Automate the population of project metadata in the Master sheet whenever a user pastes a project opportunity link (WebLink) and flags it for processing. This reduces manual data entry and standardizes how project information enters the system.

Description

1. Google Sheets Trigger – Master (rowUpdate)

- Monitors the Master sheet for changes.
- Configured to fire when the Trigger column is updated (for example, when a user types `FETCH`).

- When triggered, it outputs the full row as JSON, including `ProjectID`, `WebLink`, and `Trigger`. This event represents "a new WebLink is ready to be processed."

---

2. Code – Platform detection and input normalization

- Receives the row from the trigger and applies basic validation:
  - Ensures `Trigger` is set to the expected value (e.g., `FETCH`).
  - Ensures a `WebLink` is present.
- Derives the platform name by inspecting the URL (e.g., detecting `buildingconnected.com`, `procore.com`, etc.).
- Produces a clean payload containing the `projectId`, `webLink`, `platform` name, and any other identifiers required downstream.

---

3. HTTP Request – Fetch remote content

- Performs an HTTP GET request to the URL contained in `webLink`.
- Attempts to retrieve either JSON or HTML for the opportunity page on the external platform.
- The raw response (body, status, headers) is passed forward for parsing.
- If the platform requires authentication, this node can be extended later with API keys or cookies.

---

4. Code1 – Extract project fields

- Interprets the HTTP response returned by the previous node.
- Attempts to extract key attributes such as:
  - General Contractor (GC) name
  - GC representative and contact details
  - Project name
  - Project due date
- Where the platform response cannot be parsed (for example, due to login requirements), it sets sensible placeholder values like "Not found" so the flow can continue gracefully.
- The node outputs a unified object with all extracted or placeholder fields along with the original `projectId` and `platform`.

---

5. Code2 – Calculate Expected Anticipated Date

- Uses the retrieved Project Due Date to compute the internal planning date.
- Business rule: Exp Anticipated Date = Project Due Date – 10 days.
- Handles multiple date formats (e.g., `DD/MM/YYYY` and `YYYY-MM-DD`), converts them to a standard representation, subtracts 10 days, and stores the result as `expAnticipatedDate`.
- This node enriches the record with scheduling intelligence before it is written back to the sheet.

---

6. Update row in sheet – Persist into Master

- Writes the enriched data back into the originating row in the Master sheet.
- Uses ProjectID as the key column so the correct row is updated.
- Typical columns updated:
  - `PlatformName`
  - `GC`
  - `GCRep`
  - `GCRepContact`
  - `ProjectName`
  - `ProjectDueDate`
  - `ExpAnticipatedDate`
- The Master sheet now contains a complete, standardized record for that project, without the user manually copying data from the external platform.

---

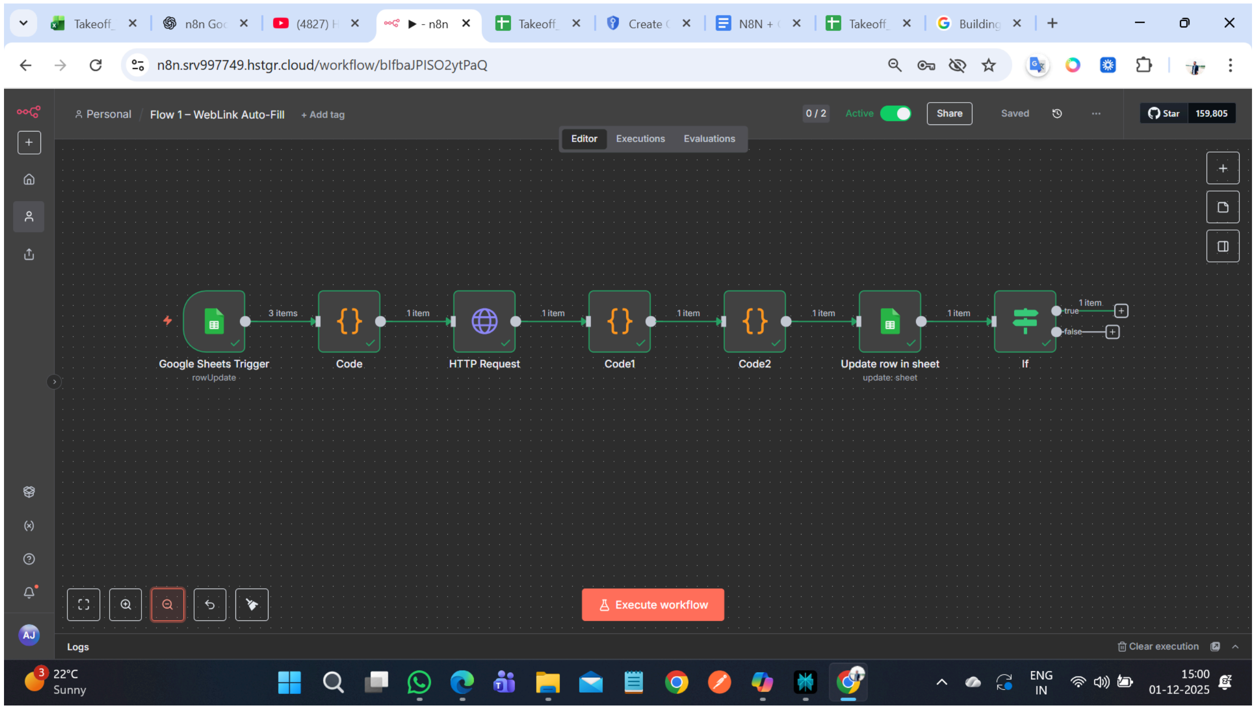
7. If – Optional quality / error check

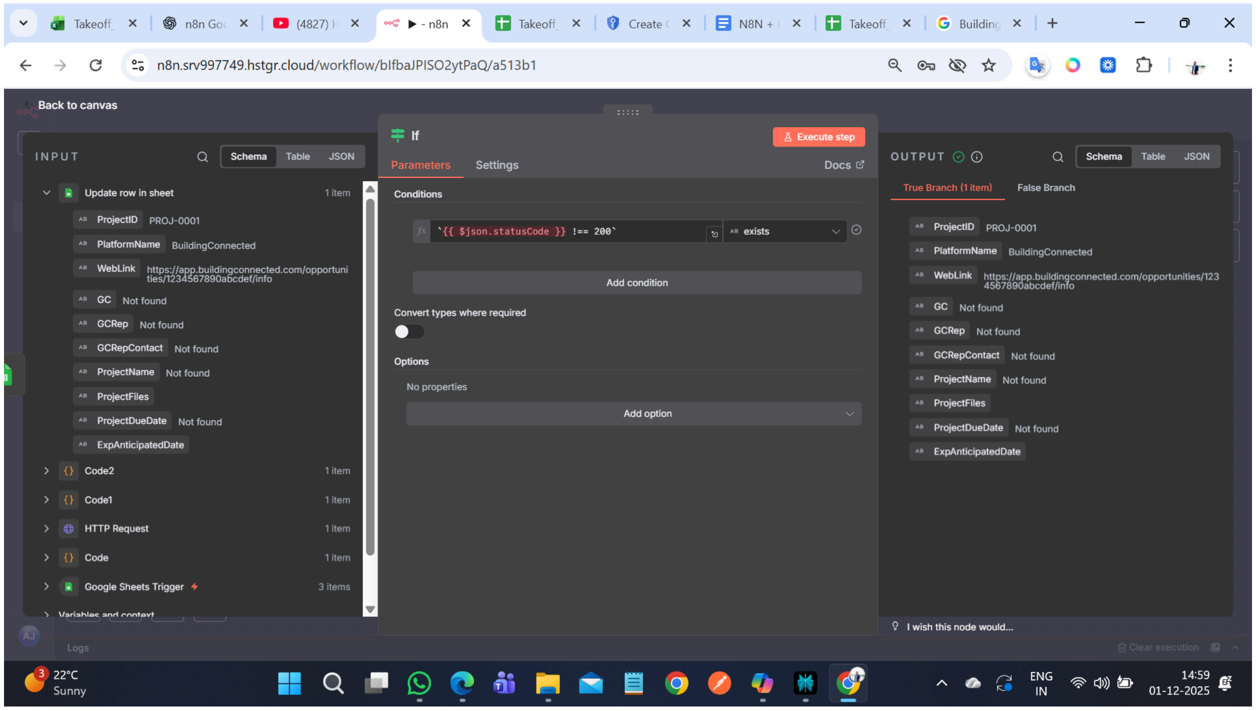
- Evaluates the result of the earlier steps to determine whether the fetch was successful.
- Example conditions:
  - Project name or GC is still “Not found”
  - HTTP status was not 200, or response body was empty
- True path (error): can update a “Notes / API Status” column in Master (e.g., “WebLink fetch failed – please fill manually”).
- False path (success): no action needed; the row is already updated with valid data.

End-to-End Behaviour

1. A user adds a new row in the Master sheet and pastes the opportunity WebLink.
2. The user types `FETCH` in the Trigger column.
3. Flow 1 runs automatically, detects the platform, fetches the remote page, extracts key fields, computes the expected date, and updates the same row.
4. If something goes wrong (no access, invalid link), the flow flags the issue in a notes/status column instead of silently failing.

This makes the Master sheet the consistent, structured entry point for all new projects while offloading repetitive data entry to n8n.





INPUT

Update row in sheet — 1 item

ProjectID	PROJ-0001
PlatformName	BuildingConnected
WebLink	https://app.buildingconnected.com/opportunities/1234567890abcdef/info
GC	Not found
GCRep	Not found
GCRepContact	Not found
ProjectName	Not found
ProjectFiles	
ProjectDueDate	Not found
ExpAnticipatedDate	

Code2 — 1 item
Code1 — 1 item
HTTP Request — 1 item
Code — 1 item
Google Sheets Trigger — 3 items
Variables and context

Conditions

```
`{{ $json.statusCode }} !== 200`    exists
```

Add condition

Convert types where required

Options

No properties

Add option

OUTPUT

True Branch (1 item) | False Branch

ProjectID	PROJ-0001
PlatformName	BuildingConnected
WebLink	https://app.buildingconnected.com/opportunities/1234567890abcdef/info
GC	Not found
GCRep	Not found
GCRepContact	Not found
ProjectName	Not found
ProjectFiles	
ProjectDueDate	Not found
ExpAnticipatedDate	

I wish this node would...

Flow 2 – Assign Project to Developer

Flow 2 orchestrates the hand-off from the central Master Dashboard to the individual developer worklists. It ensures that every assignment in the Master is reflected, in real time, in the appropriate Developer sheet, while maintaining Master as the single source of truth.

---

1. Google Sheets Trigger – Master (assignment event)

The workflow is initiated by a Google Sheets Trigger configured on the Master sheet.
It listens specifically for updates to the TakeoffAssignedTo column. When this field transitions from blank to a valid developer name (e.g., MK, MD, LA), the trigger captures the full row payload (including ProjectID and all metadata) and passes it into the workflow as a structured JSON item.

---

2. Get row(s) in sheet – Normalize Master data

To guarantee data integrity, a dedicated “Get row(s) in sheet” node re-reads the corresponding record from Master using ProjectID as the key.
This step ensures that any concurrent edits are accounted for and that subsequent logic operates on the authoritative row, not just the delta that fired the trigger.

---

3. Code – Derive routing and assignment metadata

A Code node then transforms the raw Master row into a clean, automation-ready payload. Typical operations include:

- Extracting core fields: ProjectID, ProjectName, ProjectFiles, WebLink, GC, ProjectDueDate, ExpAnticipatedDate, TakeoffAssignedTo.
- Initializing DateAssigned: if this field is empty, it is set to the current date; otherwise, the existing value is preserved to maintain assignment history.
- Resolving the target Developer sheet via a routing rule, for example:
    - MK → `Dev_MK`
    - MD → `Dev_MD`
    - LA → `Dev_LA`, etc.

The node outputs a concise JSON object containing `projectId`, all key attributes, and the computed `devSheet` name to drive downstream operations.

---

4. Update row in sheet – Stamp assignment in Master

The first Google Sheets write operation updates the originating Master row.
Using ProjectID as the key column, the node sets DateAssigned to the value computed in the Code node. This step ensures that the Master Dashboard maintains a reliable timestamp for when each project was formally allocated to a developer, supporting downstream reporting and SLA analysis.

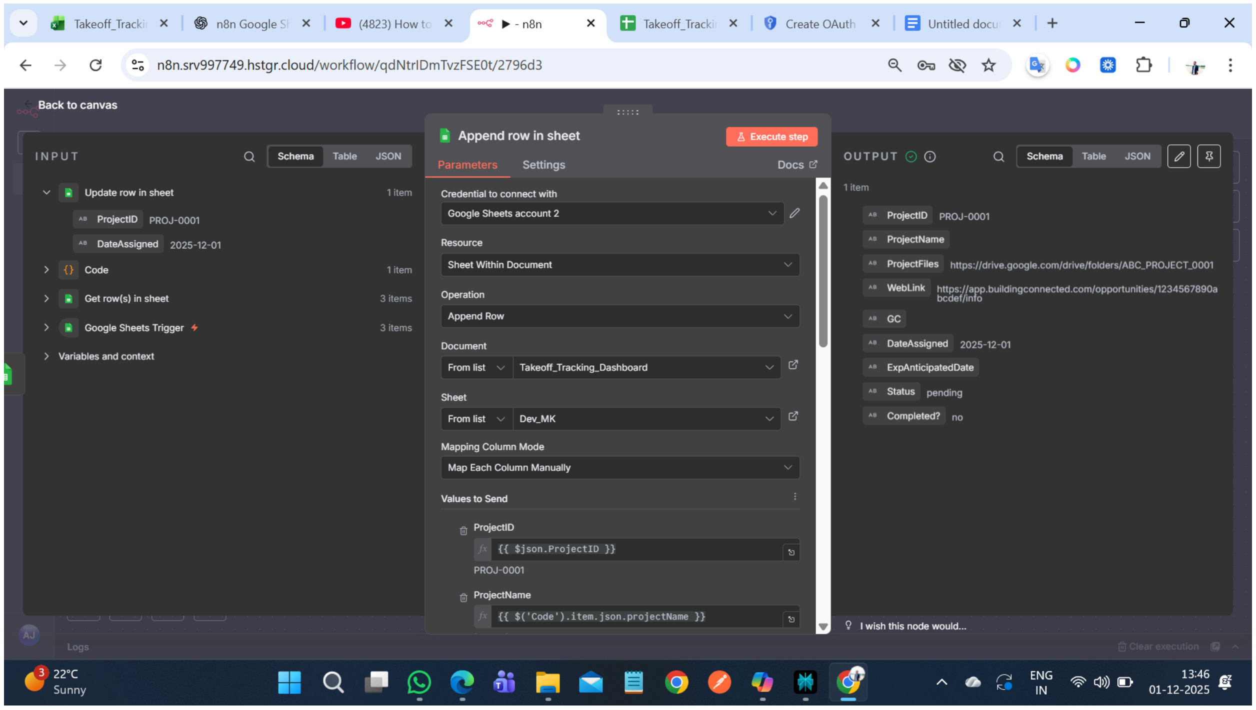
---

5. Append row in sheet – Materialize the task in the Developer sheet

Finally, an “Append row in sheet” node writes the assignment into the correct Developer sheet determined earlier (`devSheet`).
It creates a new record containing:

- All shared fields from Master (ProjectID, ProjectName, WebLink, ProjectFiles, GC, DateAssigned, ExpAnticipatedDate).
- Standardized initial workflow state:
  - Status = “Pending”
  - Completed? = “No”
  - Other delivery fields (TakeoffReceivedDate, DollarValue, NotesLatestUpdate) intentionally left blank for the developer to populate.

This gives each developer a curated, always-up-to-date backlog of assigned work, without any manual copying from the Master sheet. The client continues to manage assignments centrally, while developers operate from their own focused views, with n8n guaranteeing consistency between the two.


Append row in sheet

Credential to connect with: Google Sheets account 2

Resource: Sheet Within Document

Operation: Append Row

Document: From list — Takeoff_Tracking_Dashboard

Sheet: From list — Dev_MK

Mapping Column Mode: Map Each Column Manually

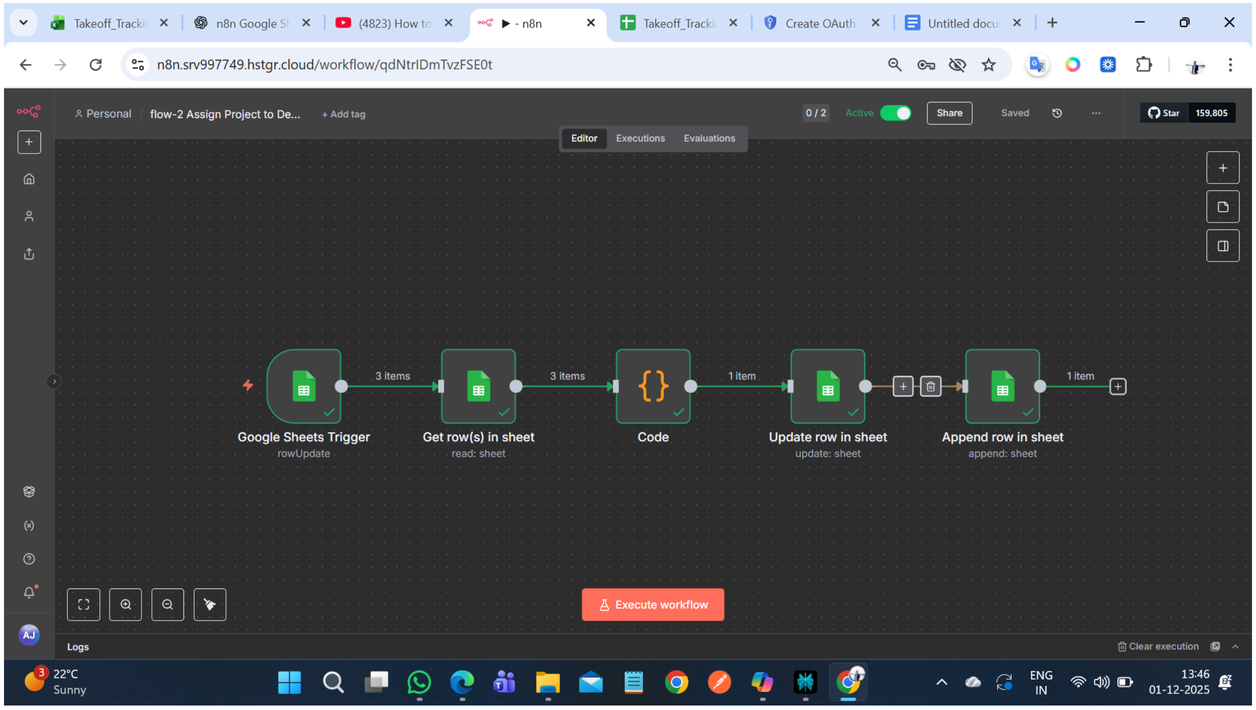
Values to Send

ProjectID
```
{{ $json.ProjectID }}
```
PROJ-0001

ProjectName
```
{{ $('Code').item.json.projectName }}
```

OUTPUT — 1 item

ProjectID	PROJ-0001
ProjectName	
ProjectFiles	https://drive.google.com/drive/folders/ABC_PROJECT_0001
WebLink	https://app.buildingconnected.com/opportunities/1234567890abcdef/info
GC	
DateAssigned	2025-12-01
ExpAnticipatedDate	
Status	pending
Completed?	no



Flow 3 – Sync Developer Sheet Back to Master

Purpose

Flow 3 keeps the Master Dashboard fully aligned with the latest updates made by developers in their individual sheets. Any change to key delivery fields in a developer sheet (status, completion flag, value, notes) is reflected back in the Master, and completion automatically drives the Takeoff Received Date.

Description

1. Google Sheets Trigger – Developer sheet (rowUpdate)

- Monitors a specific developer sheet (e.g., `Dev_MK`) for row updates.
- Configured to watch only the fields that developers are allowed to change: Status, Completed?, DollarValue, and NotesLatestUpdate.
- When any of these values changes, the trigger emits the updated developer row as JSON, including `ProjectID`.

---

2. Dev_MK – Read current developer row

- A standard Google Sheets "read: sheet" node that re-reads the full row from the `Dev_MK` sheet using ProjectID as the key.
- This ensures the workflow works with a clean, authoritative snapshot of the developer's row, including all editable and reference fields.

---

3. master sheet – Locate corresponding Master row

- Another Google Sheets "read: sheet" node, this time targeting the Master sheet.
- Uses the `ProjectID` from the developer row to retrieve the single matching record in Master.
- This establishes the Master–Developer pairing and gives the Code node visibility into both versions of the record if needed.

---

4. Code – Build synchronized payload

- Consolidates the data from the developer sheet (and optionally the current Master row) and decides what needs to be updated.
- Typical responsibilities:
    - Read the latest values from the developer sheet: `Status`, `Completed?`, `DollarValue`, `NotesLatestUpdate`.

    - If Completed? has transitioned to `Yes` and TakeoffReceivedDate is empty in Master, compute `TakeoffReceivedDate = today`.
    - Compute a `LastUpdated` timestamp to record when the sync occurred.
- Outputs a normalized JSON object containing `projectId` plus all fields that should now be pushed into Master (and optionally mirrored back into the developer sheet).

---

5. master sheet1 – Update Master from developer changes

- A Google Sheets "update: sheet" node that writes back to the Master sheet.
- Uses ProjectID as the key column to update the correct row.
- Updates the following fields in Master based on the Code node output:
    - `Status`
    - `Completed?`
    - `DollarValue`
    - `NotesLatestUpdate`
    - `TakeoffReceivedDate` (when completion is confirmed)
    - `LastUpdated` (timestamp).
- This ensures the Master Dashboard always reflects the latest view of progress, without requiring developers to touch Master directly.

---

6. Master sheet – Optional mirror of TakeoffReceivedDate into Dev sheet

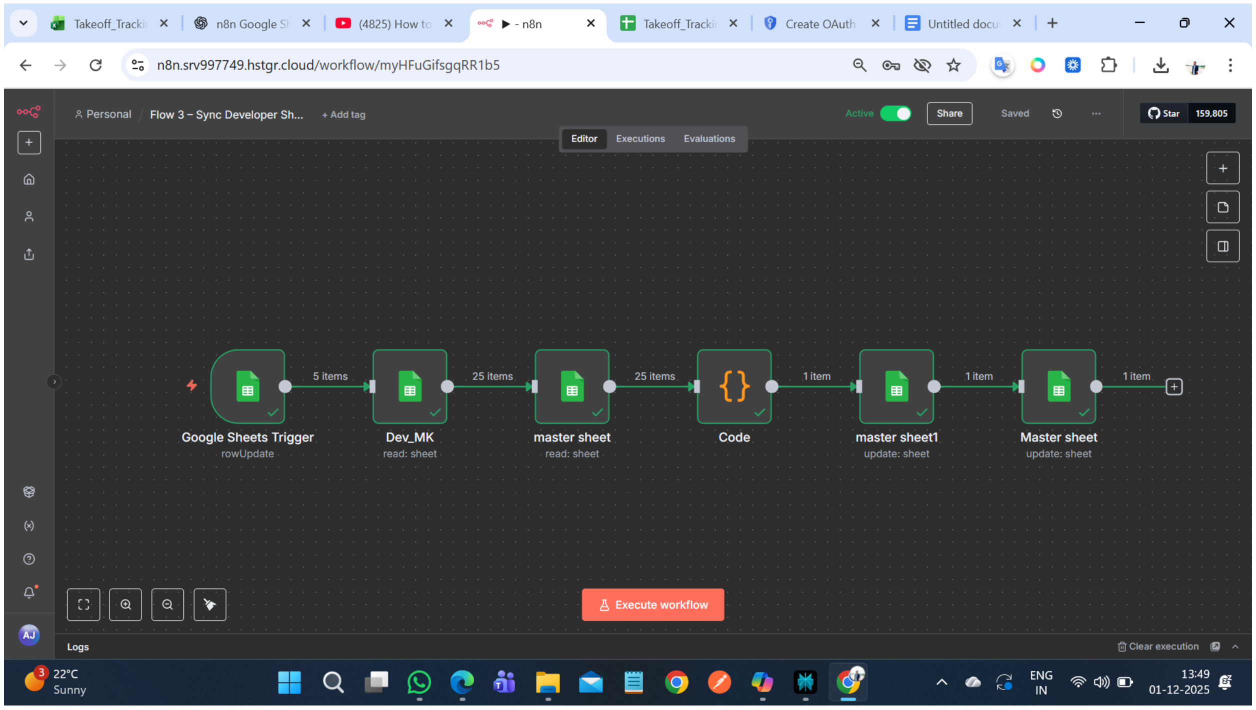
- A final Google Sheets "update: sheet" node (despite the label, it targets the developer sheet again, depending on your configuration).
- When `TakeoffReceivedDate` is newly set in Master, this node can mirror the same date back into the developer row so both sheets show the same received date.
- This keeps the developer sheet and Master perfectly synchronized around the moment of completion.

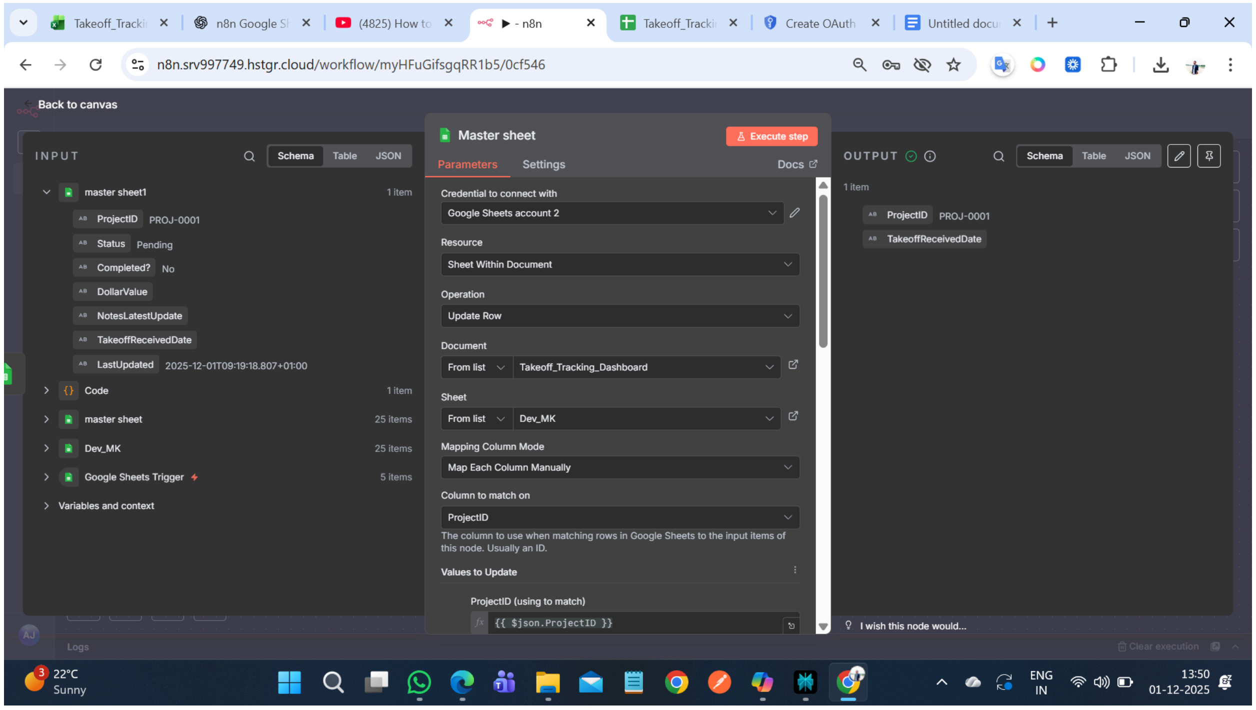
End-to-End Behaviour

1. Developer updates their row in `Dev_MK` (e.g., sets Status = In Progress, later Completed? = Yes, populates DollarValue and NotesLatestUpdate).
2. Flow 3 is triggered automatically and reads both the Dev row and corresponding Master row.

3. The Code node decides what needs updating and calculates any missing dates.
4. The Master sheet is updated so management immediately sees the new status, value, and notes.
5. When a job is marked completed, TakeoffReceivedDate is automatically set, and optionally mirrored back into the developer sheet.

This design allows developers to work exclusively in their own sheets while guaranteeing that the Master Dashboard remains a single, accurate source of truth for all reporting and oversight.





ProjectID	PROJ-0001
Status	Pending
Completed?	No
DollarValue	
NotesLatestUpdate	
TakeoffReceivedDate	
LastUpdated	2025-12-01T09:19:18.807+01:00

Credential to connect with: Google Sheets account 2

Resource: Sheet Within Document

Operation: Update Row

Document: From list — Takeoff_Tracking_Dashboard

Sheet: From list — Dev_MK

Mapping Column Mode: Map Each Column Manually

Column to match on: ProjectID

The column to use when matching rows in Google Sheets to the input items of this node. Usually an ID.

Values to Update

ProjectID (using to match)

```
{{ $json.ProjectID }}
```

ProjectID	PROJ-0001
TakeoffReceivedDate	

Flow 4 – Daily Pending Reminder (n8n + Google Sheets + Email)

Purpose

Flow 4 delivers automated, personalized email reminders to developers, listing all their open (pending) takeoff tasks. By leveraging live data from Google Sheets and a scheduled trigger in n8n, it guarantees consistent communication and helps ensure that no project is overlooked.

Description

1. Schedule Trigger

- The workflow begins with an n8n Schedule Trigger, typically set to run every weekday morning at a specified hour (e.g., 09:00 IST).
- This guarantees that reminders are dispatched regardless of manual actions, even if the team forgets to check the sheet.

---

2. Get row(s) in sheet – Read Master project list

- This node reads all current project rows from the Master sheet.
- It provides the raw data source for subsequent filtering, ensuring that the status for every project is checked daily.

---

3. Code – Filter for pending, assigned, and not-received projects

- Processes all Master rows and retains only those with:
  - Status = "Pending", "In Progress", "With Competitor", or "On Hold"
  - `TakeoffAssignedTo` set (not blank)
  - `TakeoffReceivedDate` still blank (i.e., not delivered)
- Organizes the filtered list by developer, so each subsequent message addresses the right person with only their assigned tasks.

---

4. Get row(s) in sheet1 – Lookup developer emails

- For each developer group, this node pulls their email address from a central Config or Email sheet.
- Ensures reminders are always sent to the correct, current recipient, based on Master's assignments.

---

5. Code1 – Generate personalized email content

- Compiles the relevant data for each developer (project ID, name, due date, anticipated date, status, GC, etc.) into a table or human-friendly list.
- Dynamically sets the email subject (e.g., "Pending Takeoffs – MK – 01/12/2025") and prepares a body summarizing open tasks.

---

6. Code2 – Final email formatting and checks

- Performs any final cleanup on the email data (e.g., ensuring lists are non-empty, formatting dates, handling "no pending tasks" gracefully).
- This step allows for customized messages, and can include priority flags, CCs, or escalation logic for overdue tasks.

---

7. Send a message – Email dispatch (Gmail/SMTP node)

- Uses the compiled address, subject, and body to send an actual email to each developer with their current backlog.
- By scheduling and automating this step, the workflow reinforces personal responsibility and keeps projects moving forward without requiring manual reminders from management.

End-to-End Behaviour

1. At 9:00 AM (or chosen time), n8n triggers the workflow.
2. The system reads the Master project list, filters for all ongoing, not-yet-completed projects for each developer.
3. For each developer, it looks up their email, compiles a relevant list of all open/pending projects, builds a clear summary email, and sends it directly to their inbox.

4. Each developer receives only their specific assignments (with details), every morning, automatically.

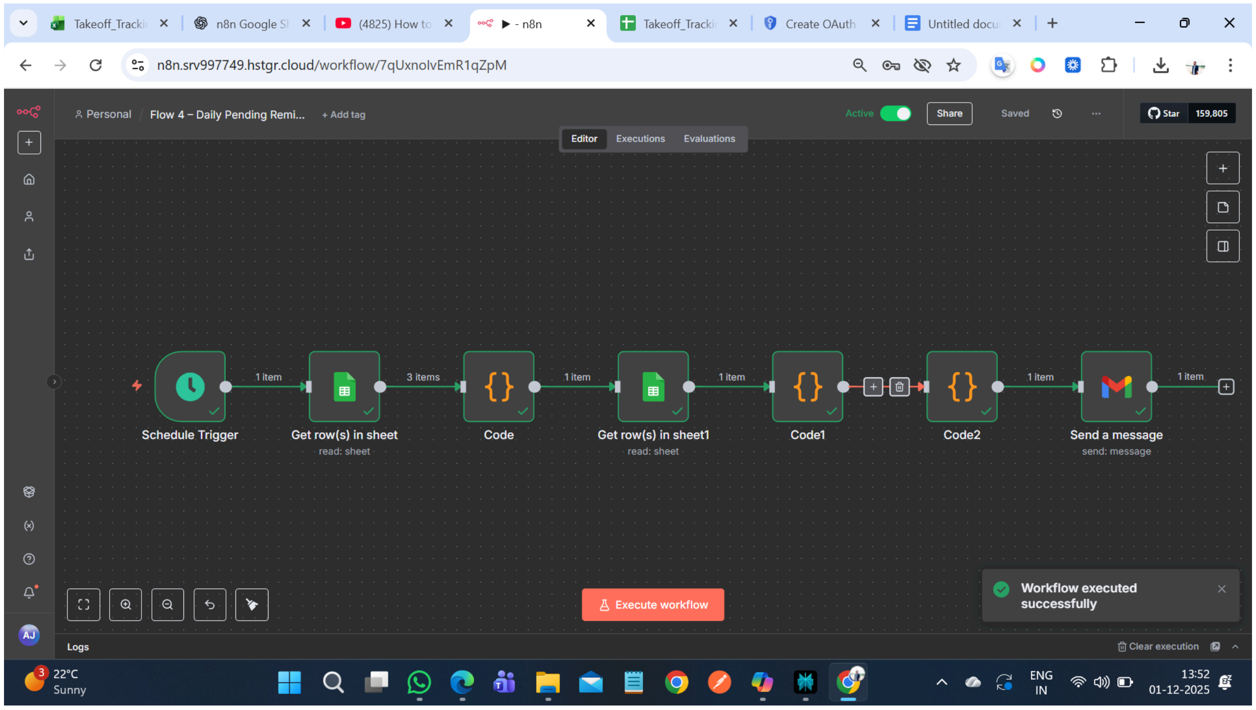
Value

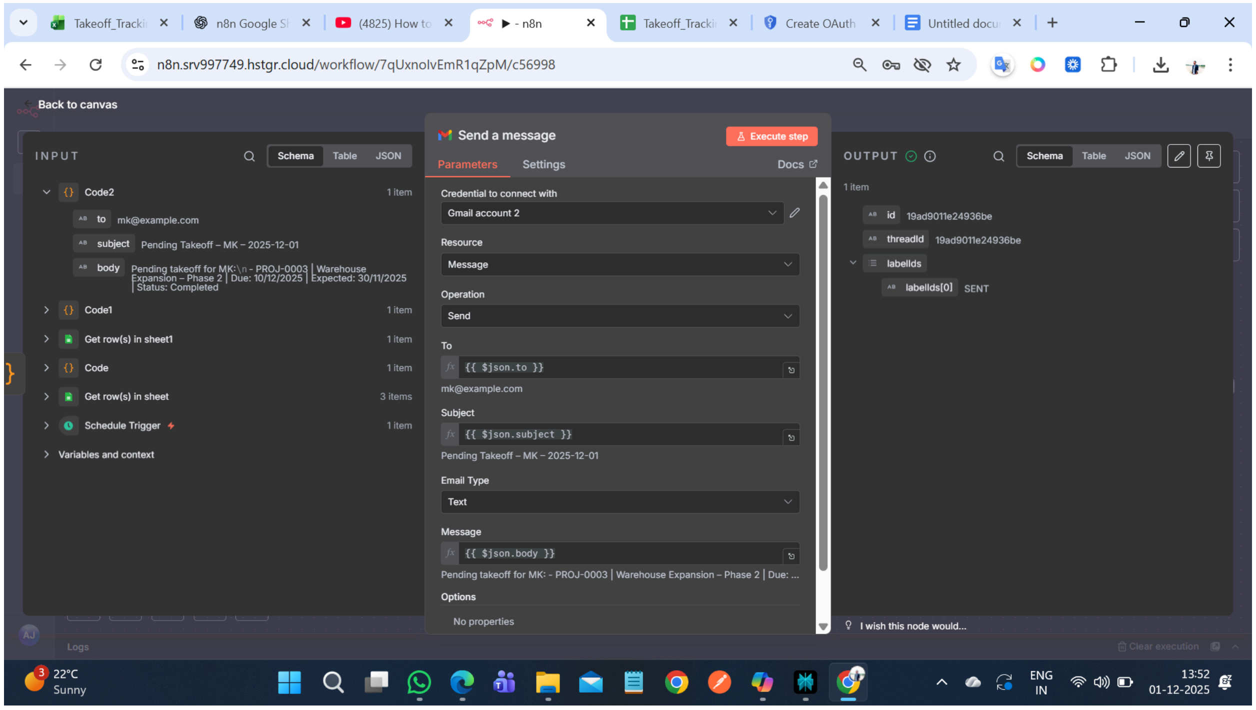
This automation ensures developers are aware of their open tasks each day, without manual follow-up by leads or coordinators. It promotes accountability, keeps work moving, and reduces the risk of forgotten or overlooked projects. Management can be confident that reminders occur 100% of the time.

1 source

Ask a follow-up

.




n8n.srv997749.hstgr.cloud/workflow/7qUxnolvEmR1qZpM/c56998

Back to canvas

INPUT (Schema | Table | JSON)

Code2 — 1 item
- to: mk@example.com
- subject: Pending Takeoff – MK – 2025-12-01
- body: Pending takeoff for MK:\n - PROJ-0003 | Warehouse Expansion – Phase 2 | Due: 10/12/2025 | Expected: 30/11/2025 | Status: Completed

Code1 — 1 item
Get row(s) in sheet1 — 1 item
Code — 1 item
Get row(s) in sheet — 3 items
Schedule Trigger — 1 item
Variables and context

Send a message (Execute step)

Parameters | Settings | Docs

Credential to connect with: Gmail account 2

Resource: Message

Operation: Send

To: `{{ $json.to }}`
mk@example.com

Subject: `{{ $json.subject }}`
Pending Takeoff – MK – 2025-12-01

Email Type: Text

Message: `{{ $json.body }}`
Pending takeoff for MK: - PROJ-0003 | Warehouse Expansion – Phase 2 | Due: ...

Options
No properties

OUTPUT (Schema | Table | JSON)

1 item
- id: 19ad9011e24936be
- threadId: 19ad9011e24936be
- labelIds
  - labelIds[0]: SENT

I wish this node would...