End-to-End Enhanced Report for the N8N Workflow

Outstanding Balances Report

Prepared: 12 September 2025

Author: Automation Team (generated from workflow artifacts)

Contents

1 Executive summary

This document is the E2E technical and operational report for the *Outstanding Balances Report* n8n workflow (two modes: on-demand voice request and monthly automated run). The high-level steps and business rules were taken from the provided workflow spec. :contentReference[oaicite:3]index=3 The node list and code snippets come from the JSON export of the workflow.

Key deliverables:

- A readable walk-through of every node and its role in the pipeline.
- Sanitized, copy-ready code fragments (no secrets).
- Placeholders for screenshots and diagrams.
- Testing, monitoring, security and operational guidance.

2 Goals and scope

- Produce a tenant outstanding balances report for Active leases (Buildium data).
- Identify late fees since the last 6th of the month, calculate days delinquent and assign urgency (Normal / High / Urgent).
- Save detailed rows to Google Sheets and create a Monday.com task to follow up.
- Support: (A) On-demand via Telegram voice/text (user receives summary reply) and (B) Monthly scheduled run (silent, creates artifacts).

3 High-level workflow overview

The workflow has two triggers:

1. **On-demand voice/text** — Telegram Trigger node receives user voice or text and the flow transcribes (if needed), interprets the query and produces the report.
2. **Scheduled monthly** - Cron mode scheduled for the 6th of every month at 09:00 (runs silently, creates files and tasks).

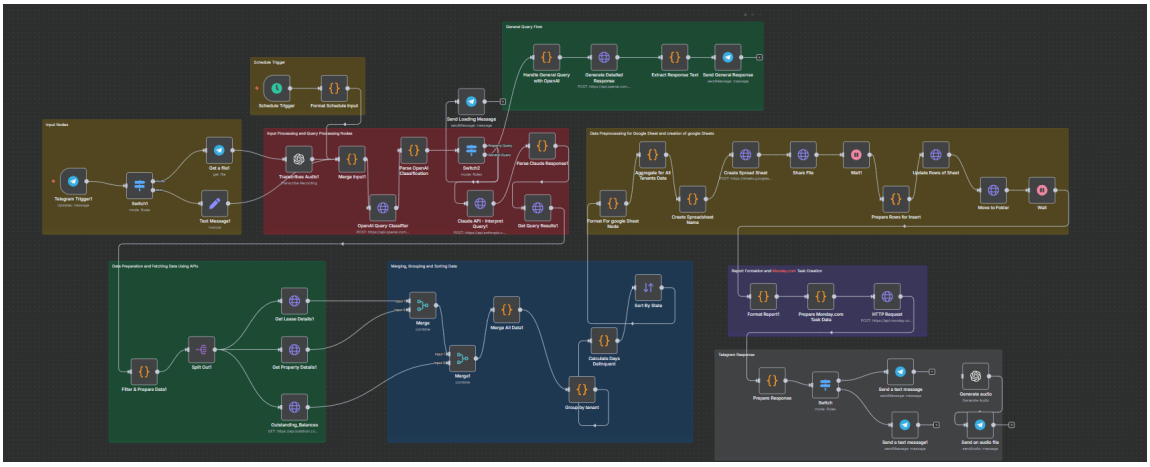

Figure 1: Workflow overview

4 Step-by-step node breakdown

Below each step lists: **node name**, **node type**, **purpose**, **inputs**, **outputs**, **sanitized parameters or pseudocode**, **error-handling suggestions**.

4.1 Step 1 — Telegram Trigger (Input)

Node: `Telegram Trigger1` (telegramTrigger) **Purpose:** receives incoming Telegram messages (voice or text).

Inputs:

- Telegram update payload (message, voice file id, chat id).

Outputs:

- Raw Telegram message for downstream switch/processing.

Notes and suggestions:

- Validate message source (allowlist chat IDs or usernames if required).
- Rate limit: if you expect many triggers, queue or debounce similar voice requests.

4.2 Step 2 — Switch (voice vs text) + Get File + Transcription

Nodes: `Switch1`, `Get a file1`, `Transcribes Audio1` (AssemblyAI / OpenAI) **Purpose:** determine if incoming message is voice or text; if voice, download voice file and transcribe to text.

Sanitized pseudocode:

```
// switch: if message.voice.file_id exists => route to get-file -> transcribe
// else use message.text
```

Error handling:

- If file download fails: retry with exponential backoff (configurable up to 3 attempts).
- If transcription fails: fall back to sending an error reply to the user or create a task marked 'Transcription Failed'.

4.3 Step 3 — Natural Language Interpretation (Claude/LLM)

Node: `Claude API - Interpret Query1` (httpRequest) **Purpose:** send transcribed text to an LLM with a system prompt instructing which Buildium endpoints to use; the LLM returns a structured JSON response containing `intent`, `primary_endpoint`, `data_needed`, etc. (Note: the workflow includes this behavior in a HTTP node).

Sanitized example prompt summary (NOT actual key):

```
System: You are Iris, assistant for Buildium...
User: "Iris, generate outstanding balances for active leases"
Response expected: JSON { "intent": "outstanding_balances_report", "primary_endpoint": "/v1/leases/outstandingbalances", "primary_params": { "leasestatuses":"Active" }, ...}
```

Error handling: If LLM returns invalid JSON, attempt a clean parse step (strip triple backticks) then validate; if still invalid, log and mark for manual review.

4.4 Step 4 — Buildium API: Query outstanding balances

Node: `Get Query Results1` (httpRequest) **Purpose:** call Buildium endpoint (primary endpoint) to fetch outstanding balances for Active leases; then later call lease/property/transactions endpoints per lease. The workflow expects the Buildium API endpoints to include: `/v1/leases/outstandingbalances`, `/v1/leases/{LeaseId}`, `/v1/rentals/{PropertyId}`, and `/v1/leases/{LeaseId}/transactions`. **Important security note:** the actual HTTP headers in the JSON contain sensitive client id/secret values. In production, store these in n8n credentials or in environment secrets — do **not** paste raw keys into documents.

Sanitized example request:

```
GET https://api.buildium.com/v1/leases/outstandingbalances?
    leasestatuses=Active
Headers:
  x-buildium-client-id: <REDACTED>
  x-buildium-client-secret: <REDACTED>
  Accept: application/json
```

Batching / Throttling: The workflow uses batching (batchSize:1, batchInterval:500ms) for per-lease calls to avoid hitting API rate limits; keep these values configurable.

4.5 Step 5 — Filter Prepare Data / Split Out / Per-Lease API calls

Nodes: `Filter & Prepare Data1`, `Split Out1`, `Get Lease Details1`, `Get Property Details1`, `Get Late Fee Transactions1` **Purpose:** for each lease with a balance, fetch lease details, property details and transactions since last sixth to compute late fees. The last-6th calculation is implemented in a Parse node.

Last-6th logic (summary): If today's day-of-month ≥ 6 then lastSixth = current month 6th, else lastSixth = previous month 6th. The date is passed to the transactions query as `transactiondatefrom`. (Exact logic is implemented in the workflow).

4.6 Step 6 — Merge / Merge All Data / Grouping

Nodes: `Merge`, `Merge1`, `Merge All Data1`, `Group by tenant` **Purpose:** combine results from different API calls into one canonical tenant record, sum up late fees since last 6th and collect transaction memos/dates. The Merge All Data node's JS collects tenant names, address, balances and late fees. (sanitized implementation in appendix).

Data model (per tenant):

- tenantName, propertyAddress, propertyState
- leaseId, propertyId, unitNumber
- currentBalance, balance0To30Days, balance31To60Days, balance61To90Days, balanceOver90Days
- lateFeesSinceLastSixth (sum of transaction amounts with memo containing 'late'/'fee'/'penalty')
- transactionDates, transactionMemos
- daysDelinquent (computed later)

4.7 Step 7 — Calculate Days Delinquent & Priority

Node: `Calculate Days Delinquent` **Purpose:** compute an estimated number of days delinquent from the earliest late fee transaction date (if exists) or infer from aging buckets; assign priority as:

- 6–15 days: **Normal**
- 16–30 days: **High**
- 30+ days: **Urgent**

This priority mapping is implemented in the node code.

4.8 Step 8 — Format for Google Sheets / Create Spreadsheet / Update rows

Nodes: `Format For google Sheet Node`, `Aggregate for All Tenants Data`, `Create Spread Sheet`, `Prepare Rows for Insert`, `Update Rows of Sheet` **Purpose:** prepare rows with headers, create a Google Spreadsheet with title `"Outstanding Balances - [MM-DD-YYYY] [HH-MM]"`, insert rows and move file to the target Drive folder. The workflow sets folderId to a specific Drive folder (redacted).

Spreadsheet header: `Tenant Name | Property Address | State | Outstanding Balance | Days Delinquent | Priority`

Note: Use Google's `values:batchUpdate` with `USER_ENTERED` to allow currency formatting and formulas if needed.

4.9 Step 9 — Format Report & Telegram/Monday.com outputs

Nodes: `Format Report1`, `Prepare Monday.com Task Data`, `HTTP Request (Monday)`, `Send Report to Telegram1`, `Prepare Response` **Purpose:** create a user-friendly text/voice report for Telegram (or silently complete for scheduled runs), create a Monday.com task with description and link to the created spreadsheet. The Report and Monday creation code are present in the workflow.

Report summary example (sanitized):

```
       OUTSTANDING BALANCE REPORT
Generated: 2025-10-05 09:00:00
Last 6th: 2025-10-06

SUMMARY:
    Total Tenants: 120
    Tenants with Outstanding Balance: 24
    Total Outstanding: $12,345.67
    Priority Breakdown: Urgent: 2, High: 6, Normal: 16
```

4.10 Send notifications

Telegram: For on-demand voice users, the workflow sends a voice/text reply: `"Found [X] tenants across [Y] states owing` [*Amount*].*Createdreportinyourfolderandassignedtasktocarlosforfoll*
up."

`Monday.com: A GraphQL mutation creates an item (board id and person id are present in the JSON but should be stored in credentials). The mutation is passed to Monday's API node. (See sanitized mutation in Appendix.)`

5 Business logic rules

- `Last-6th calculation: determine the last 6th of month relative to today | transaction queries use this date to find new late fees.`
- `Late fee detection: transaction memos examined for keywords: late, fee, penalty (case insensitive). Any matching charge is treated as a late fee and summed.`

- Priority classification: as described in Step 7.

6 Operational concerns, error handling and recommendations

6.1 Credentials & secrets

- Store Buildium, Google, Telegram, Monday and LLM credentials in n8n's built-in credential store or in a secure secrets manager | never hardcode tokens in nodes or documents.
- Rotate credentials periodically and use least privilege (scoped API keys).

6.2 Retries, batching, and rate limiting

- Per-lease calls are batched with a small delay to avoid hitting API limits. Keep the batching parameters configurable.
- Use exponential backoff for transient HTTP errors (429, 5xx).

6.3 Validation and fallbacks

- Validate LLM JSON outputs before acting. If malformed, log the raw LLM response and route to a human queue.
- If Buildium returns partial data, insert placeholder values in the sheet (e.g., N/A) and flag the row for manual review.

6.4 Monitoring and alerts

- Log key metrics: API call counts, errors, created spreadsheet IDs, number of tenants processed, total outstanding. Persist logs to a monitoring platform (Datadog/CloudWatch/Elastic).
- If >0 critical/urgent tenants appear, send an immediate alert (email/Slack) to operations.

7 Testing checklist

1. Unit test the JS code blocks (mock inputs) | verify last-sixth calculation, late fee parsing, and priority assignment.
2. End-to-end test in sandbox Buildium account (test leases transactions).
3. Test the Telegram voice flow with a known voice file; check transcription accuracy and parsing by the LLM.
4. Test Google Sheets creation, insertion, and Drive file move.
5. Test Monday.com GraphQL mutation in a test board.

8 Improvements and optional features

- Add a *dry-run* mode to preview report generation without creating Sheets or tasks.
- Add an interactive Slack/Teams notification instead of Telegram for internal users.
- Add an audit trail sheet with raw API responses (sanitized) for compliance.
- Add confidence scoring for LLM interpretation and fallback rules for ambiguous queries.

9 Screenshots of Each Portion

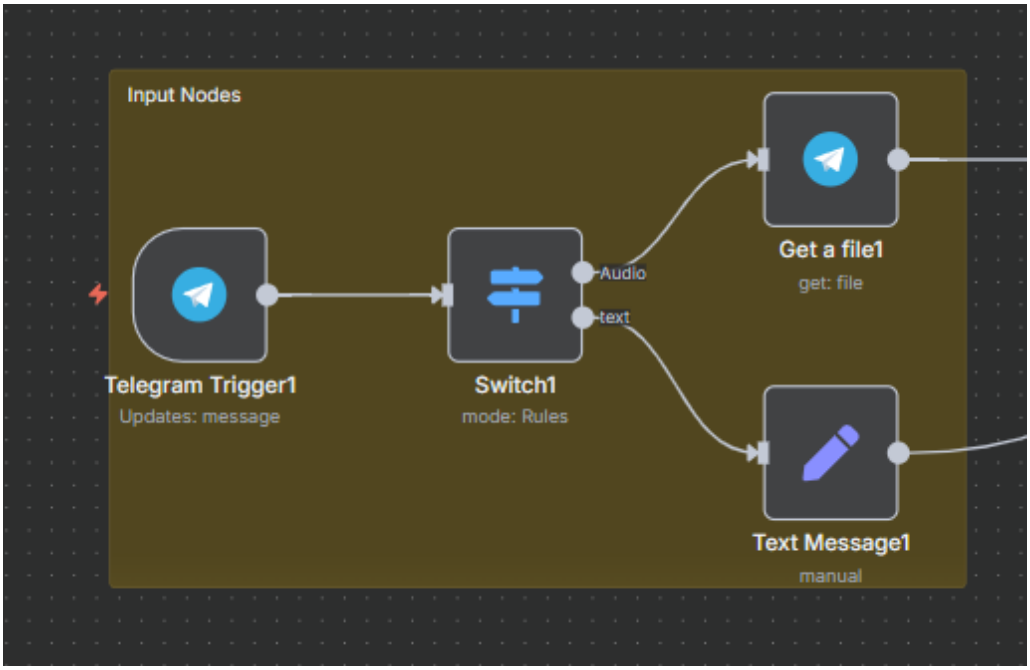

Figure 2: Inputs and transcription nodes

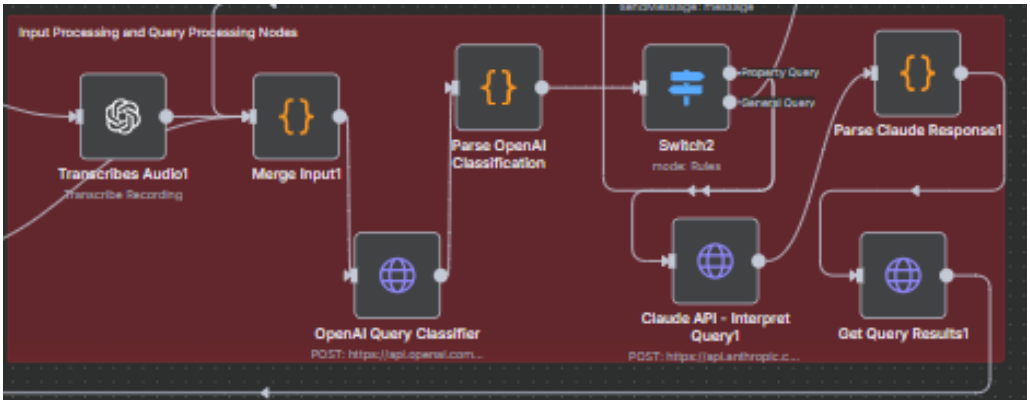

Figure 3: Input Processing Nodes

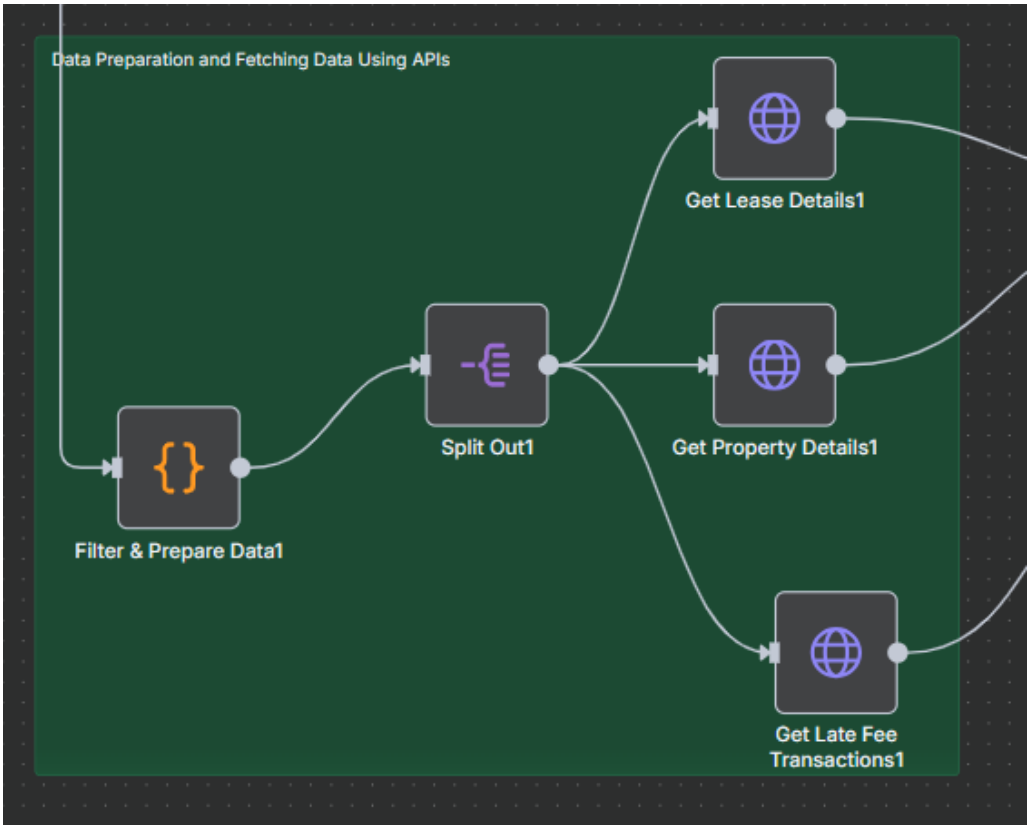

Figure 4: Prompt Preparation and fetching Data Using APIS

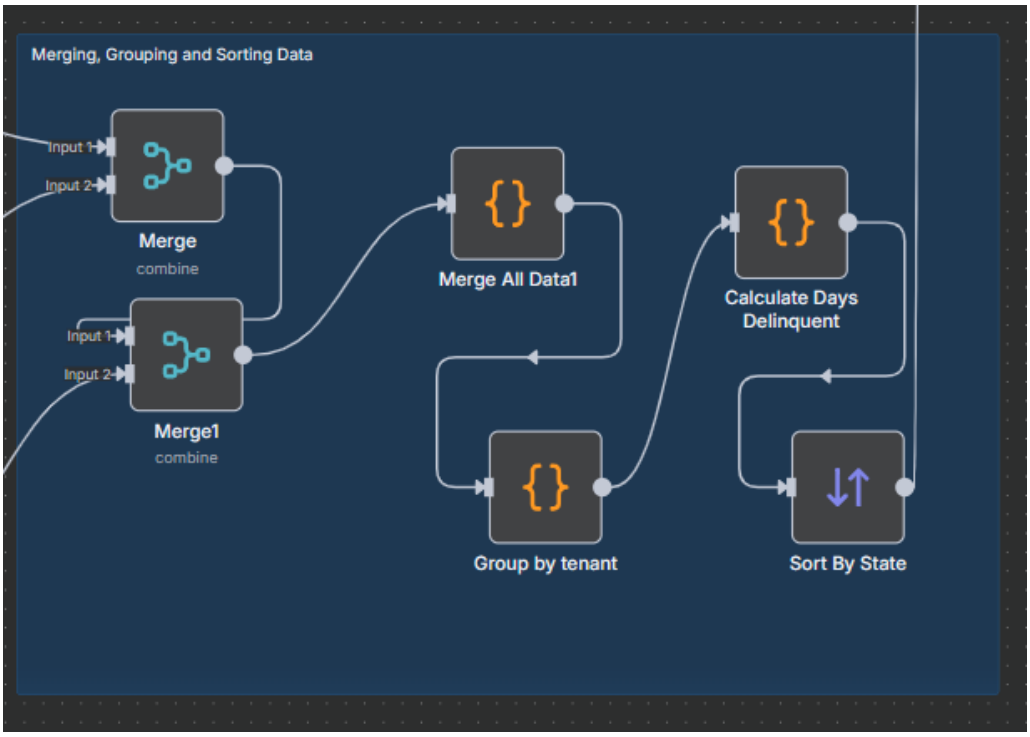

Figure 5: Merging, grouping and calculation nodes

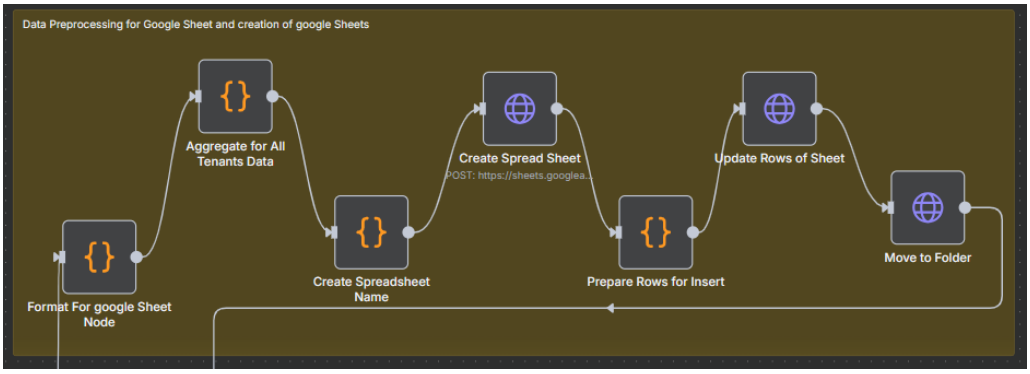

Figure 6: Preparation and Upload Data on sheets and drive Nodes

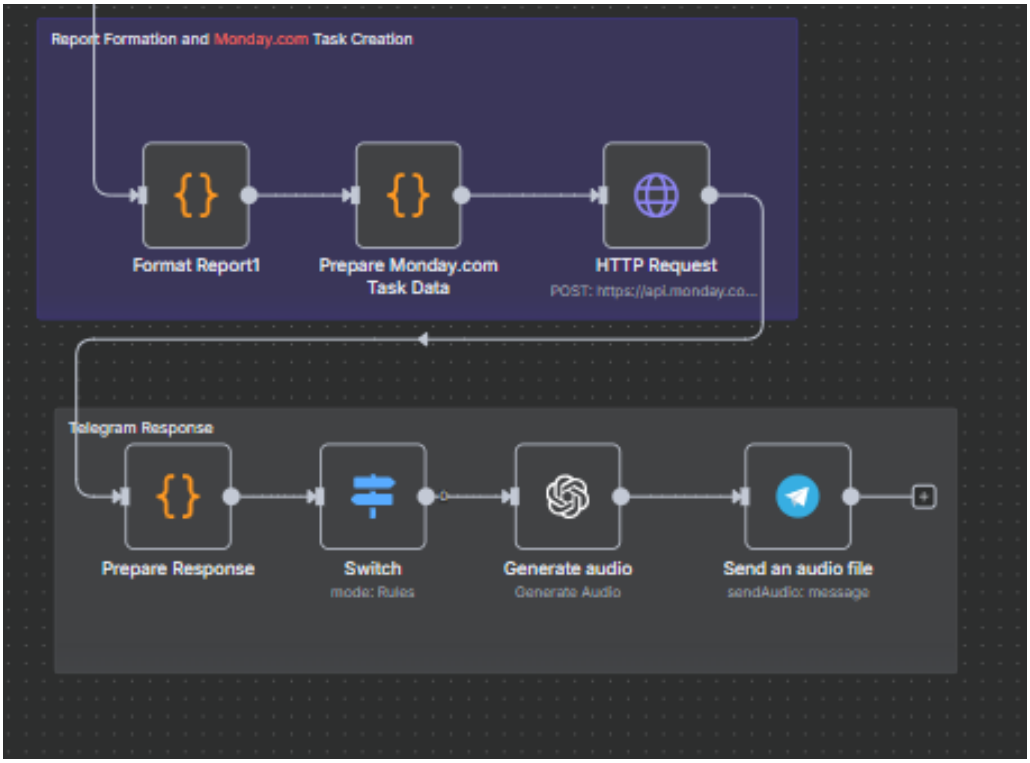

Figure 7: Formatting Report and Output Report nodes

10 Appendix A — Sanitized node summary (extracted from JSON)

Below is a short, sanitized table of the most important nodes and purpose (full JSON available separately). Sensitive fields redacted. (This table is derived from the JSON export.)

Node name	Type	Purpose / notes
Telegram Trigger1	telegramTrigger	Accept voice/text input; webhook receives message (on-demand).
Switch1	switch	Routes voice vs text to appropriate flow.

Get a file1	telegram	Downloads voice file using file id (if voice).
Transcribes Audio1	OpenAI/AssemblyAI	Transcribes voice to text.
Claude API - Interpret Query1	httpRequest	Sends text to LLM to identify intent and required Buildium endpoints.
Get Query Results1	httpRequest	Calls Buildium outstanding balances endpoint (Active leases).
Get Lease Details1	httpRequest	GET /v1/leases/LeaseId (per lease).
Get Property Details1	httpRequest	GET /v1/rentals/PropertyId (per property).
Get Late Fee Transactions1 glaccountids=15	httpRequest	GET /v1/leases/LeaseId/transactions?transactiondat
Merge All Data1	code	Consolidates lease/property/transaction data per lease and computes late fee sums.
Calculate Days Delinquent	code	Derives days delinquent and priority.
Format For google Sheet Node	code	Formats rows for spreadsheet insertion.
Create Spread Sheet	httpRequest	Creates Google spreadsheet; spreadsheetName set by JS.
Format Report1	code	Builds summary message(s) for Telegram and full report payload (pagination for Telegram).
Prepare Monday.com Task Data	code	Builds GraphQL mutation to create task (sanitized).

11 Appendix B — Example report table (LaTeX-ready)

Below is a sample longtable layout you can paste into other documents when presenting results.

Tenant Name	Property (Address)	State	Outstanding Balance	Days Delinquent
John Doe	123 Elm St, Unit 2	NY	$1,250.00	45
Jane Smith	200 Oak Ave, Apt 4	CA	$800.00	12
Acme Corp	10 Industrial Rd	TX	$3,500.00	90+

12 Appendix C — Sanitized code snippets

Sanitized Buildium request (JS/HTTP node)

```
// Example (sanitized)
GET https://api.buildium.com/v1/leases/outstandingbalances
```

```
Query: leasestatuses=Active
Headers:
  x-buildium-client-id: <REDACTED>
  x-buildium-client-secret: <REDACTED>
  Accept: application/json
```

Sanitized Monday.com mutation (constructed in the flow)

```
mutation {
  create_item (
    board_id: <REDACTED_BOARD>,
    group_id: "topics",
    item_name: "Outstanding Balances Review - Oct 05, 2025",
    column_values: "{\"person\":{...}, \"long_text\":{\"text\":\"
       Outstanding balance report generated by Iris...\"}}"
  ) {
    id
    name
  }
}
```