

Shopify Fulfillment and Shipment Notification Automation

How I replaced manual tracking emails with near real time shipment updates for an operation handling 500 to 600 orders each month

Workflow scale	Monthly volume	Estimated time saved
11 active workflows and 185 nodes	500 to 600 orders	33 to 60 staff hours per month

Designed and implemented end to end

Shopify AfterShip Supabase n8n Gmail Telegram Google Sheets

September 2026

Contents

1. Executive Summary
2. Business Problem
3. Solution Architecture
4. Workflow Implementation
5. Engineering Decisions
6. Reliability Security and Privacy
7. Results and Estimated Impact
8. Technology Stack and Contribution
9. Constraints and Next Steps

All customer, order, address, and tracking details visible in the included screenshots are test data approved for this portfolio document.

Executive Summary

I designed and implemented an automation system that manages the path from a new Shopify order to fulfillment, carrier tracking, and customer shipment updates. The client handles approximately 500 to 600 orders per month and previously sent tracking numbers manually, while Gmail messages could not stay synchronized with carrier status changes in real time.

The solution uses 11 active n8n workflows and 185 nodes across Shopify, AfterShip, Supabase and PostgreSQL, Google Sheets, Gmail, and Telegram. Claims, locks, a durable notification outbox, and six watchdogs protect the process. At an assumed four to six minutes of avoided manual work per order, estimated savings are 33 to 60 staff hours per month, with a 45.8 hour midpoint. These are estimates, not measured production results.

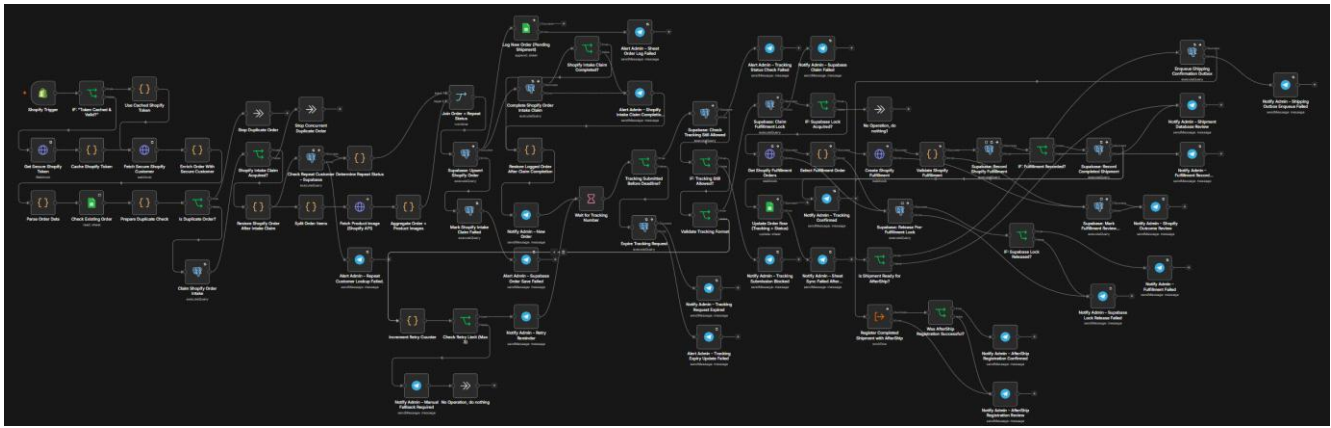


Figure 1 Main Shopify fulfillment workflow from order intake through fulfillment persistence and AfterShip registration

Business Problem

The client depended on staff to retrieve tracking details, prepare a message, and email each customer. That process did not scale well across 500 to 600 monthly orders. It also made shipment communication depend on someone noticing a carrier change and composing the correct Gmail message.

The operational risk extended beyond email. A reliable process also had to prevent duplicate order intake, avoid concurrent fulfillment attempts, preserve the result of a successful Shopify action, register the tracking number with AfterShip, and detect jobs that stopped between services.

Before	After
Staff manually prepared and sent tracking number emails.	A controlled outbox generates messages from trusted order and carrier data.
Gmail updates depended on a person noticing each shipment change.	Signed AfterShip events create notification work that is checked every 10 seconds.
Partial failures required manual investigation without a consistent recovery path.	Six watchdogs detect stale claims or missing persistence and route recovery or alerts.
Duplicate processing could reach external APIs.	Database claims and fulfillment locks guard nonrepeatable actions.

Enter J&T Tracking Number

ORDER TO PREPARE

Order: [REDACTED]

Customer: [REDACTED]

Total quantity: 3

ITEMS TO PACK

1. **The Inventory Not Tracked Snowboard** — Quantity: 1
2. **The Complete Snowboard** — Quantity: 1
3. **The Multi-location Snowboard** — Quantity: 1

IMPORTANT:

Confirm every product and quantity before booking the shipment with J&T.

Enter the J&T tracking number below.

Production format: JT followed by 13 digits.

Example: JT#####

Tracking Number *

Real: JTxxxxxxxxxxxxx | Test: ITOD-1-CNX...

Submit

Figure 2 Human tracking number form with customer and order identifiers redacted

Solution Architecture

One large orchestration workflow manages Shopify order intake, the human tracking handoff, and fulfillment. Four AfterShip workflows handle trusted registration, carrier event ingestion, notification preparation, and email delivery. Six scheduled watchdogs monitor pending work, claims, registration, fulfillment persistence, and notification state.

Supabase and PostgreSQL hold the durable operational state. Google Sheets provides a human readable view of orders and tracking. Gmail sends transactional messages, while Telegram carries administrator prompts, confirmations, review requests, and failure alerts.

Implementation component	Verified count
Active n8n workflows	11
Total nodes	185
PostgreSQL nodes	38
Telegram nodes	32
Conditional nodes	31
Code nodes	17
HTTP request nodes	9
Scheduled triggers	7
Watchdog workflows	6

Workflow Implementation

Shopify Order Intake and Duplicate Control

A Shopify order trigger starts the main workflow. The workflow obtains or reuses a cached Shopify access token, then performs a GraphQL lookup to enrich the event with customer details. Custom code normalizes order identifiers, customer details, currency, totals, line items, quantities, and product identifiers.

The workflow first checks Google Sheets for an existing order. It then acquires a database backed intake claim linked to the n8n execution ID. A concurrent execution stops when it cannot acquire the claim. If later persistence fails, the workflow marks the claim failed and alerts an administrator.

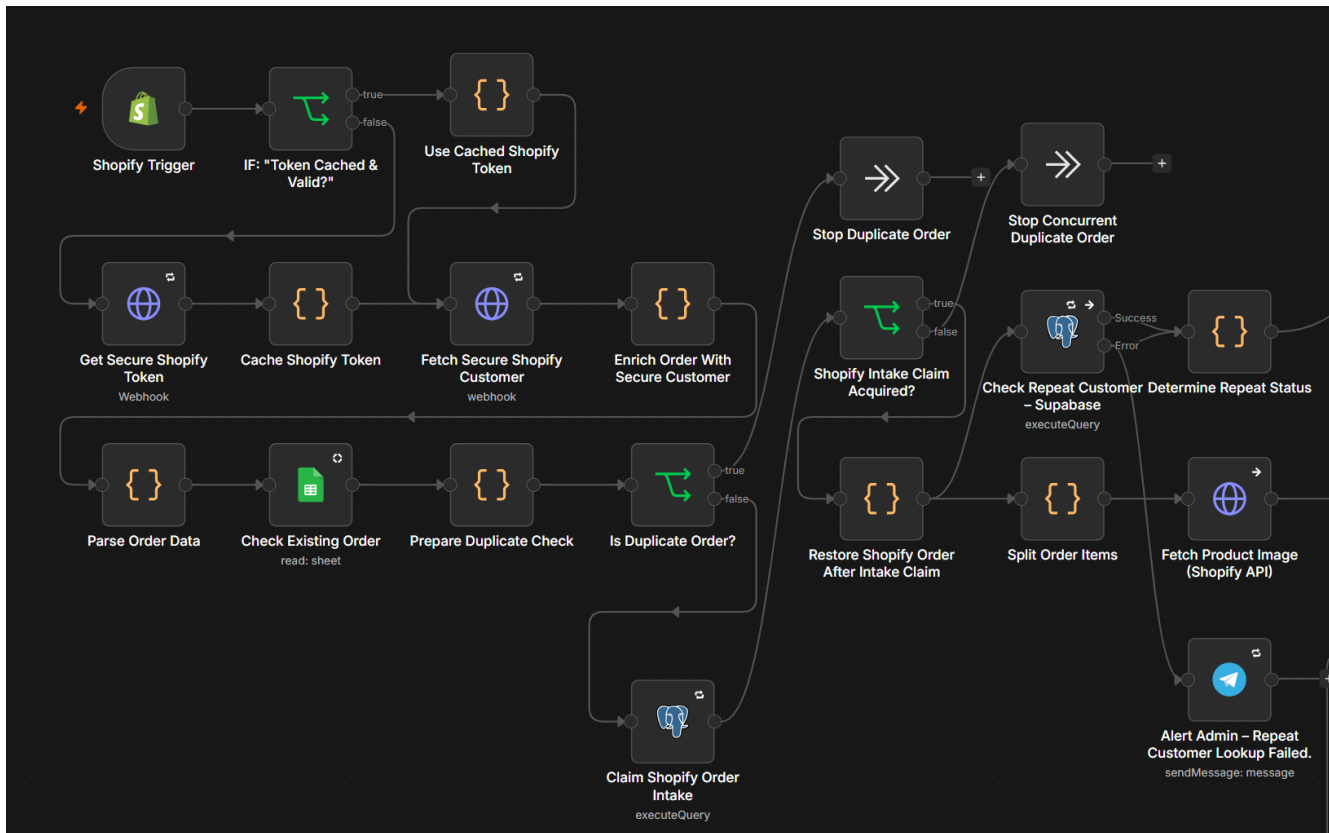


Figure 3 Order intake enrichment duplicate checking and database claim control

Human Tracking Collection

After the order is stored, Telegram prompts the administrator and the workflow pauses on an n8n form. The form shows the order, customer, item list, and total quantity. The administrator confirms the packed items and enters the J and T tracking number.

The form uses a configured time limit value of 72. The export does not explicitly store the unit, so the case study does not state one. Tracking validation accepts the production J and T pattern or a defined AfterShip test pattern. Invalid submissions enter a reminder loop capped at three attempts. The workflow requests manual fallback when the retry limit is reached.

Before fulfillment begins, a database check confirms that the order still accepts tracking. Late or obsolete submissions are blocked. A timed out request is marked expired and produces an administrator alert.

Locked Shopify Fulfillment

For an eligible tracking number, the workflow acquires a Supabase fulfillment lock. It retrieves Shopify fulfillment orders through GraphQL and continues only when exactly one open fulfillment order supports the CREATE FULFILLMENT action. Multiple eligible fulfillment orders stop for review to avoid an accidental partial shipment.

After the Shopify fulfillment request, custom validation checks GraphQL errors, user errors, and the returned fulfillment ID. The workflow records the external result in Supabase. If Shopify may have completed the action but local persistence is uncertain, the order is marked for review instead of sending the same mutation again.

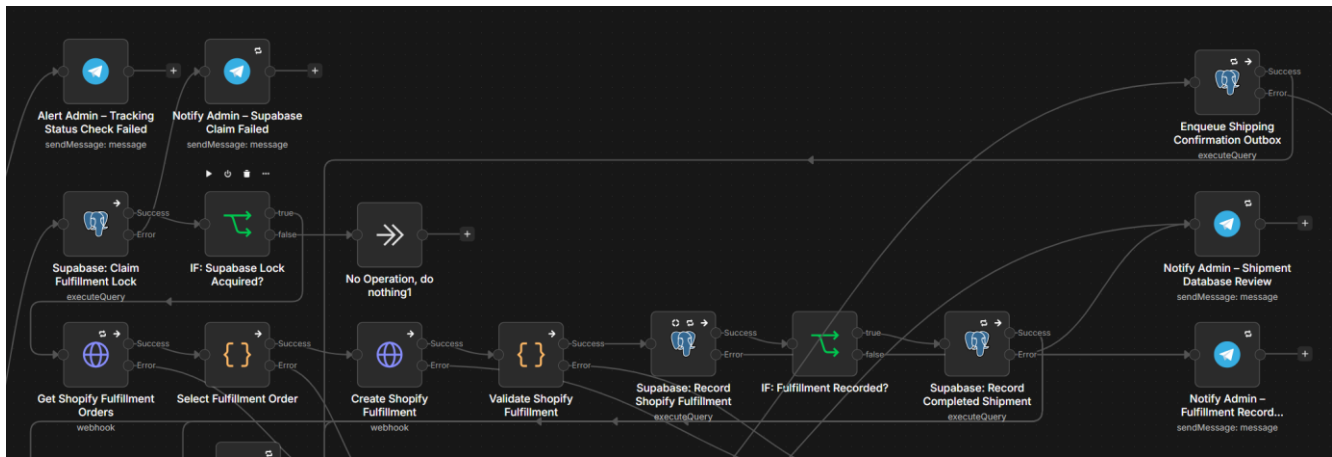


Figure 4 Fulfillment lock Shopify mutation validation and database persistence

Trusted AfterShip Registration

The tracking intake workflow validates the order ID, carrier, tracking format, and title. It reloads the recipient from the trusted Supabase order record rather than accepting an arbitrary email from the caller.

The registration workflow checks whether the tracking number should be registered, is already registered, belongs to another order, references an unknown order, is disabled, or already requires review. New registrations use a database claim to control concurrency. AfterShip must return status code 201 and a provider tracking ID before the registration is finalized. Rejected or ambiguous results enter a review state.

Signed Carrier Events

AfterShip status changes arrive through a webhook. The workflow extracts the raw body, calculates a cryptographic signature, and compares it with the AfterShip HMAC header. Missing or invalid signatures receive HTTP 401.

Accepted events are normalized into one tracking record. The workflow selects the latest checkpoint, converts timestamps into explicit time zone aware values, maps carrier tags to readable statuses, and captures checkpoint details and location. Durable ingestion returns HTTP 200 for accepted or intentionally ignored events, HTTP 401 for invalid signatures, and HTTP 503 when notification ingestion is unavailable and the sender should retry.

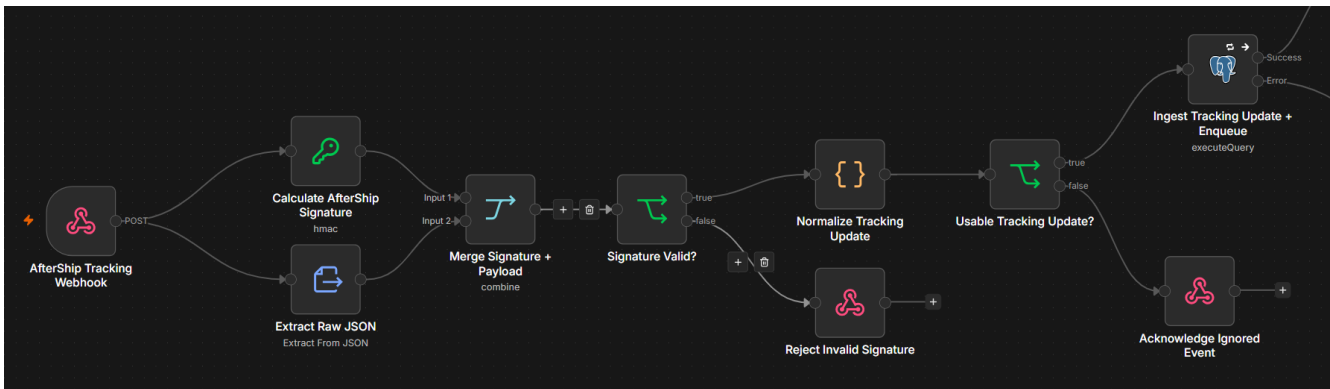


Figure 5 AfterShip webhook signature validation normalization and durable event ingestion

Customer Notification Outbox

A dedicated worker checks the notification outbox every 10 seconds. It claims the next job, confirms that the tracking number resolves to exactly one registered tracking record and trusted order, then loads the Shopify order and notification configuration.

Test mode routes messages to a designated test recipient. Live mode requires a production environment, LIVE notification mode, and an enabled live send flag. Invalid recipients, unsupported statuses, and already notified states are skipped. A status claim built from the tracking number and notification status reduces duplicate sends.

The worker creates escaped HTML email with order details, product images, tracking data, and a four stage progress indicator. Seven statuses have templates: Shipped, In transit, Out for delivery, Available for pickup, Delivered, Failed attempt, and Exception. Gmail sends the message, then the worker records the final sent or failed result. Errors during context loading, Shopify enrichment, configuration, claiming, delivery, or finalization are stored and can trigger Telegram alerts.



Woohoo! Your order is on the way.

Hi [REDACTED] Your package is moving through the carrier network.



In transit

The shipment is on the way

Location: 1000 E Test Ave, Reedley, California, 11111

[TRACK MY SHIPMENT](#)

Order summary



The Inventory Not Tracked Snowboard

Quantity 1
Price USD 949.95
Subtotal USD 949.95



The Multi-managed Snowboard

Quantity 1
Price USD 629.95
Subtotal USD 629.95



The Videographer Snowboard

Quantity 1
Price USD 885.95
Subtotal USD 885.95

Order	#1080
Tracking number	ITOD-2-CNX0911267482
Last updated	09-11-2026 08:09 PM PHT
Total	USD 2465.85



This is a transactional shipment update from CNexa Solutions.

Figure 7 Test shipment email with progress tracking and order summary

Watchdog and Recovery Workflows

Six scheduled workflows provide monitoring and recovery around the main path.

Watchdog	Detection rule	Response
Pending tracking request	Checks every 15 minutes for requests older than 24 hours	Alerts the administrator and suppresses repeat alerts for six hours
Shopify intake claim	Claims older than 20 minutes with a 50 record run limit	Uses row locking and alerts without duplicating concurrent watchdog work
AfterShip registration	Checks every 15 minutes for claims older than 15 minutes	Queries AfterShip and reconciles a confirmed provider record or alerts for review
Fulfillment persistence	Shopify fulfillment at least five minutes old without a shipment record	Reloads Shopify, validates identity and tracking, repairs persistence, and reenters tracking intake
Notification outbox	Pending or processing jobs older than 20 minutes and failed jobs without a next attempt	Claims up to 50 alerts and suppresses repeats for six hours
Notification claim	Delivery claims older than 20 minutes	Claims up to 50 alerts and suppresses repeats for six hours

The recovery workflows use two distinct responses. When an external system confirms the missing state, the automation reconciles the database and safely resumes processing. When the outcome cannot be proven, it records the uncertainty and alerts an administrator instead of guessing.

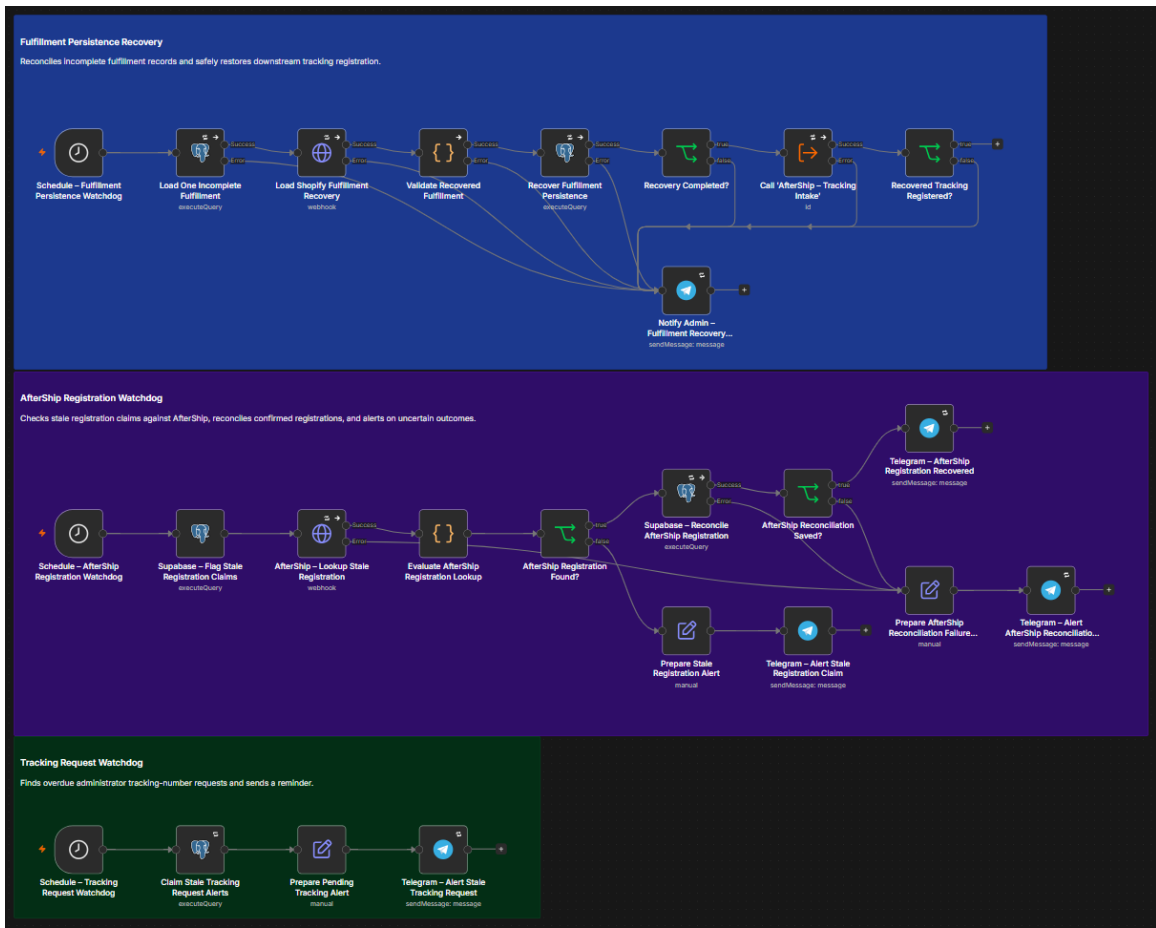


Figure 8 Representative recovery workflows for fulfillment registration and tracking requests

Engineering Decisions

Database Claims Control Duplicate Work

Order intake, fulfillment, AfterShip registration, outbox jobs, and notification status each use durable claims or locks. This reduces the chance that overlapping n8n executions perform the same external action twice.

Event Ingestion Is Separate from Email Delivery

The AfterShip webhook does not depend on Gmail completing in the same execution. It writes notification work to a persistent outbox, while a scheduled worker performs enrichment and delivery. The boundary supports retries and makes message state visible in the database.

Recovery Verifies External State

The fulfillment and registration watchdogs query Shopify or AfterShip before repairing local state. This avoids repeating an external mutation when the original action may have succeeded but the following database write failed.

Live Email Requires Explicit Configuration

Customer delivery requires a production environment, LIVE notification mode, and an enabled live send flag. Test mode uses a configured test recipient. These separate checks reduce the chance that development executions reach customers.

Customer Data Crosses a Defined Trust Boundary

Registration and notification workflows reload customer context from Supabase and Shopify. They do not rely only on values supplied by the calling workflow. Dynamic email content is escaped before it is inserted into HTML.

Reliability Security and Privacy

Verified safeguards include HMAC validation for AfterShip webhooks, n8n credential references for connected services, database claims associated with execution IDs, fulfillment locks, notification deduplication, explicit review states, retryable webhook responses, and scheduled stale state monitoring.

The published screenshots use test data. A production portfolio export should continue to hide credential references, Telegram chat IDs, Google Sheets identifiers, webhook and form URLs, Shopify and AfterShip object IDs, tracking numbers, customer details, database connection information, and internal execution IDs.

Results and Estimated Impact

Verified System Outcomes

- The implementation covers Shopify order intake, tracking collection, fulfillment, AfterShip registration, carrier event ingestion, and status based email.
- Seven customer shipment states have dedicated email templates.
- The notification worker checks durable outbox work every 10 seconds.
- Six watchdog workflows monitor stale claims, pending work, and incomplete persistence.
- Concurrency controls exist at order intake, fulfillment, registration, outbox job, and notification status levels.
- Ambiguous fulfillment and registration outcomes enter reconciliation or manual review instead of being recorded as successful.

Estimated Business Impact

Measure	Basis	Estimate
Monthly order volume	Client reported	500 to 600 orders
Manual effort avoided	Conservative assumption	4 to 6 minutes per order
Monthly time saved	Volume multiplied by minutes per order	33 to 60 staff hours
Midpoint	550 orders multiplied by 5 minutes divided by 60	45.8 staff hours per month
Annual equivalent	Monthly range multiplied by 12	About 400 to 720 staff hours

These values estimate the effort required to retrieve order details, prepare a tracking email, and send it. They do not assume that staff previously emailed every carrier checkpoint. Production time tracking or execution data would be required to report measured savings.

The architecture supports near real time processing rather than a guaranteed delivery time. AfterShip events enter through a webhook and the worker checks email jobs every 10 seconds, but Shopify, AfterShip, Gmail, database, and network latency still affect the final message.

Technology Stack and Contribution

Technology	Role in the system
n8n	Workflow orchestration, forms, schedules, branching, and recovery
Shopify	Order events, customer and order enrichment, and fulfillment creation
AfterShip	Tracking registration and shipment status webhooks
Supabase and PostgreSQL	Orders, shipments, claims, locks, registration state, and notification outbox
Google Sheets	Human readable operational order and tracking view
Gmail	Transactional shipment email delivery
Telegram	Administrator prompts, confirmations, escalation, and failure alerts
GraphQL and REST	External service integration

I designed and implemented the complete system independently. My work covered the solution architecture, all 11 n8n workflows, Shopify and AfterShip integrations, Supabase database functions, claims and locks, the notification outbox, seven email templates, Telegram alerts, all six watchdogs, testing, deployment, and operational safeguards.

The project required explicit state transitions and trust boundaries across several services. I defined when the workflow could retry automatically, when it had to verify an external system, and when an uncertain result required human review.

Constraints and Next Steps

The workflow exports verify architecture and configuration. The client reported monthly volume and the previous manual process. Time savings remain estimates until production measurements are available. The exports also do not expose the implementation of the database functions, so their internal transactional behavior is outside this case study.

- Capture execution history for successful paths and handled failures.
- Measure the time from carrier event receipt to Gmail delivery.
- Compare support contacts and missed shipment updates before and after deployment.
- Track notification retries, permanent failures, and watchdog recoveries in an operations dashboard.

Conclusion

This project demonstrates how I build automation around external side effects and uncertain outcomes. Claims and locks guard actions that should not run twice. Durable records separate event ingestion from delivery. Watchdogs recover what can be proven and alert an operator when the system needs judgment.

I build dependable automation for businesses that need customer communication, external APIs, and operational recovery to work as one system.