RecklessBear Stock Tracking System Ultimate Guide, Architecture Overview, and Improvements

Generated: 22 Jan 2026 | Focus: materials-first manufacturing stock (custom apparel)

What this document is: A complete, implementation-ready explanation of how your current Airtable-based stock system works, why it is correct for custom apparel, and how to rebuild it cleanly inside a web app + Supabase + automations (n8n/Trello).

Executive Summary

- Your model is **raw-materials-first** (fabric, trims, accessories), not product SKU inventory. This is the correct model for made-to-order sublimation apparel.
- The heart of the system is a **BOM** (Bill of Materials): how much of each material is used per 1 unit of a product variant (product + size).
- Stock should only change through **atomic database transactions** (Supabase) triggered by business events (e.g., order moves to Printing). Airtable formulas are fine for visibility but not safe for writes.
- Today you have overlapping logs (restock log, movement log, transactions, line items). For Phase 1 you can simplify to: **Materials**, **Product Materials (BOM)**, **Stock Movements**, optional **Transactions**.

Core Philosophy: Materials-First Manufacturing

You are not stocking finished goods. Each order is custom. So you track the inputs (materials) and deduct them dynamically using BOM rules. This unlocks accurate stock, realistic lead times, and low-stock warnings that actually matter.

Key concept: You never track “Netball Jersey stock”. You track “Polyester Fabric (200gsm) - meters”, “Elastic - meters”, “Zips - units”, etc.

Key Entities and Terms

Use these terms consistently across the web app, database, and automations so humans and LLMs interpret the system the same way.

Term	Meaning
Material	Raw input tracked in stock (fabric, trims, accessories, ink).
Product Variant	A product + size (and optionally other dimensions) that has a specific BOM.
BOM (Bill of Materials)	Rules that define per-unit material consumption for a variant.
Order / Job	A confirmed customer request with quantities per product variant.
Stock Movement	An immutable record of a material quantity change (+ restock, - consumption).
Transaction	Optional grouping layer: one business event that creates many movements.
Trigger Event	The business moment you allow stock to change (e.g., Trello -> Printing).

Current Airtable Base: Tables and What They Do

This section explains how your current Airtable system functions today, based on the actual schema in the base.

1. Materials Inventory (Source of Truth)

Tracks on-hand quantities for each raw material and flags low stock. This is the core table you keep in any rebuild.

Field	Type	Purpose / Notes
Material Name	Text	Primary key/display name for raw material.
Unit	Single select	Meters, units, rolls, etc.
Quantity in Stock(m)	Number	Current on-hand quantity (unit implied by field name).
Minimum Stock Level	Number	Below this you are 'low'.
Restock Threshold	Number	Target trigger level to restock (can be >= minimum).
Low Stock Indicator	Formula	Boolean/label derived from quantity vs minimum/threshold.
Supplier	Text	Preferred supplier or vendor.
Notes	Text	Freeform notes.
Restock Movement Log	Link	Links to restock events (currently).

Notes: The field name “Quantity in Stock(m)” bakes a unit into the column. In a database-backed app, keep quantity numeric and use Unit as the truth.

2. Product Material Usage (BOM / Rules)

Defines material consumption per 1 unit of a product variant. This is not inventory; it is the rule engine that drives deductions.

Field	Type	Purpose / Notes
Product Name	Text	Product or variant name (e.g., Rugby Jersey).
Material 1 Required	Single select	Material name (should be linked to Materials Inventory).
Size	Single select	Variant dimension.
Quantity Used	Number	Quantity per 1 unit of product, in the unit specified.
Unit	Single select	Unit for Quantity Used (ideally inherited from material).
Material 2	Single select	Second material (non-scalable approach).
Quantity Used 2	Text	Quantity for material 2 (should be number).

Material 2 Unit	Single select	Unit for material 2.
Last Modified By	Last modified by	Audit.
Last Modified	Last modified time	Audit.

Risk: “Material 1 / Material 2” is not scalable. A real BOM should support any number of materials. Quantity Used 2 is currently stored as text.

3. Restock Movement Log (Manual Restock Entry)

Captures restocks and attempts to calculate new stock quantities via lookups/formulas. This exists because Airtable cannot safely enforce transactional updates.

Field	Type	Purpose / Notes
Order Name	Text	Reference for restock purchase/order.
Date	Created time	Timestamp of log record creation.
Material Added	Link (multiple)	Materials being restocked.
Quantity Added	Number	Amount added.
Type	Single select	Restocked/Adjustment/etc.
Supllier	Text	Supplier name (typo in field).
Material Name (from Material Used)	Lookup	Lookup from linked records.
Updated By	Last modified by	Who updated.
Material Name (Text)	Formula	Flattened material names to text.
Quantity in Stock (from Material Added)	Lookup	Current stock before/after (ambiguous).
New Stock Quantity	Formula	Calculated new stock (risky in Airtable).
Quantity Updated	Single select	Status flag.

Risk: Calculating “New Stock Quantity” in Airtable is not atomic. Two updates can overwrite each other. In the web app, restock should create movements and the database updates stock in one transaction.

4. Stock Movement Log (Low-Level Audit)

Tracks individual material movements (consumed/restocked). This maps perfectly to a database table like stock_movements.

Field	Type	Purpose / Notes
Date	Date time	Movement timestamp.

Order Name	Text	Reference (order/job/restock).
Material Used	Text	Material consumed (should be link).
Material Added	Text	Material restocked (should be link).
Material Name (from Inventory Item)	Lookup	Lookup if you link materials.
Quantity Used/Added	Number	Signed amount or absolute with Type.
Type	Single select	Consumed / Restocked / Adjustment.
Updated By	Single select	Who did it (should be user/ref).
Total Material Usage Per Order	Formula	Derived per order (reporting).

Risk: Materials are stored as text fields. This should be a link/foreign key to Materials Inventory for integrity and reporting.

How the System Works End-to-End

You effectively have two flows: (A) Production consumption and (B) Restocking. The trigger point you described is when a Trello card moves to Printing, which is a good real-world moment to commit consumption.

A. Production (Consumption) Flow

1. Lead comes in and is handled by reps (sales status: Contacted -> Quote Sent -> Quote Approved).
2. When a quote is approved, a job is created in Trello. Production status is tracked by card list (Design -> Printing -> Packing -> Delivered).
3. **Trigger:** When the Trello card moves to **Printing**, you treat quantities as final and commit stock consumption.
4. System queries BOM for each product variant and multiplies: **order_qty x quantity_per_unit** for each material.
5. System creates stock movement rows (negative quantities) for each material consumed.
6. System updates materials inventory quantities atomically and checks thresholds.
7. Customer alerts are sent as the card moves through stages (email/WhatsApp).

B. Restock Flow

1. Admin records a restock (what material, how much, supplier, reference).
2. System creates stock movement rows (positive quantities) for each material added.
3. System updates materials inventory quantities atomically.
4. Low-stock flags resolve automatically once quantity rises above minimum.

C. Dual-Status Model (Sales vs Production)

You correctly want two parallel statuses:

- **Sales Status**: lead handling and conversion (Contacted, Quote Sent, Quote Approved, etc.).
- **Production Status**: manufacturing pipeline (Design, Printing, Packing, Delivered, etc.).
- Rule of thumb: sales status lives on the lead/order record; production status mirrors the Trello list and is synced back to the database for reporting.

Recommended Web App + Supabase Architecture

The goal is to move all stock math and writes into Supabase so updates are atomic, auditable, and safe under concurrency. Airtable becomes optional (or is replaced fully).

Minimum Clean Schema (Phase 1)

- **materials**: id, name, unit, qty_in_stock, min_stock, restock_threshold, supplier, created_at, updated_at
- **product_materials** (BOM): id, product_id (or product_name), size, material_id, qty_per_unit, unit_override nullable
- **stock_movements** (immutable): id, material_id, qty_delta (+/-), type (consumed/restocked/adjustment), source (order/manual/audit), reference_type, reference_id, created_at, created_by
- **orders**: id, order_code, customer_id, sales_status, production_status, trello_card_id, created_at, updated_at
- **order_items**: id, order_id, product_id/name, size, quantity

Optional (Phase 2): stock_transactions + stock_transaction_items if you want grouped events, richer reporting, and reconciliations.

Atomic Stock Update Rule (Non-Negotiable)

Every time you change stock, you do it inside one database transaction:

- Insert movement rows (append-only).
- Update materials.qty_in_stock using a single SQL statement per material (or a stored procedure).
- If any material would go negative beyond allowed rules, abort the entire transaction and return an error.
- Return a deterministic response so the automation can be retried safely (idempotency).

Idempotency (Prevents Double Deduction)

When Trello triggers can fire more than once (or you re-run a workflow), you must guarantee you do not deduct twice.

- Use a unique key like (source='order', reference_id=order_id, type='consumed', stage='printing')
- Before applying a deduction, check if a movement batch with that key already exists.
- Alternatively: store a boolean flag on the order: stock_deducted_at printing.

System Responsibilities: Web App vs n8n vs Trello

To keep the system stable, each component must have clear boundaries.

Web App (Source of Truth + Admin UX)

- CRUD materials, BOM rules, and suppliers.
- Create and manage orders and order items.
- Perform stock deductions and restocks via server-side endpoints (or DB functions).
- Provide dashboards: low stock, usage history, cost estimates (later), audit trails.
- Provide a WhatsApp inbox UI if you are using Cloud API (WABA) so staff can reply to inbound messages.

Trello (Production Status UI)

- Visual pipeline for production stages (lists).
- Single card per job/order. Card movement represents stage changes.
- Card contains order reference_id and minimal production notes.

n8n (Orchestration + Notifications)

- Listen to events (Trello card move, order approved, etc.).
- Call the web app API to perform actions (do not compute stock with spreadsheets).
- Send notifications (email/WhatsApp) to reps and customers.
- Send weekly CEO summaries, low-stock alerts, and failure alerts.
- Retry safely on transient failures using idempotent API calls.

High-Impact Improvements (What to Change Next)

These are prioritized from 'must do' to 'nice to have'.

Must Do

- **Refactor BOM structure**: one row per (product, size, material). Remove Material 1/2 columns. Make all materials foreign keys.
- **Make stock writes atomic**: all deductions/restocks happen in Supabase via a single transaction or stored procedure.
- **Use a single movement log**: one immutable table that records every stock delta; avoid duplicate log tables.
- **Enforce idempotency**: prevent double deduction when Trello triggers fire twice or workflows are re-run.
- **Normalize units**: quantity is numeric; unit is a field. Avoid embedding units into column names.

Should Do

- **Split statuses**: keep Sales Status and Production Status separate and synced. Production status should mirror Trello list.
- **Add permissions**: only admins can edit BOM and stock. Everyone can view. Movements are append-only.
- **Add validation**: block deductions if BOM missing, or if quantities are zero/invalid; surface clear errors to admin.
- **Introduce batch operations**: deduct multiple materials in one call for performance and transactional safety.

Nice to Have

- **Transactions layer**: group movements under a transaction id for reporting.
- **Costing**: store material cost per unit and compute estimated order cost from BOM x qty.
- **Forecasting**: projected stock based on upcoming jobs in design/printing.
- **Supplier re-order automation**: generate draft purchase orders when below threshold.

Implementation Blueprint (Web App Build)

This section gives an LLM-friendly, developer-ready blueprint: endpoints, events, and rules.

Primary Events and Triggers

- **order.quote_approved**: create Trello job card; set sales_status='quote_approved'; optionally set production_status='design'.
- **trello.stage_changed**: sync production_status to DB; if stage becomes 'printing' -> execute stock_deduct(order_id).
- **stock.restocked**: create movement(s) + update material quantities.
- **stock.low**: notify ops/admin; optionally create restock task.

Suggested API Endpoints (Server)

- POST /api/orders/{orderId}/deduct-stock -> Deducts stock for the order based on BOM, idempotent by stage.
- POST /api/materials/{materialId}/restock -> Adds stock (creates movements + updates qty).
- POST /api/trello/sync-stage -> Updates production_status from Trello webhook payload.
- GET /api/materials/low-stock -> Returns materials below min/restock threshold for dashboards/alerts.
- GET /api/stock/movements?ref=ORDER123 -> Audit trail for a job/order.

Stored Procedure Outline (Recommended)

A database function makes deductions safe and fast. High-level steps:

1. Input: order_id, stage_key (e.g., 'printing'), actor_id
2. Check idempotency: if already deducted for this order+stage -> return 'already_done'
3. Join order_items to product_materials to compute required material deltas
4. Validate: all BOM rows exist; quantities positive; optional allow negative stock rule
5. Insert stock_movements rows (one per material)
6. Update materials.qty_in_stock = qty_in_stock - required_qty for each material
7. Commit and return summary

Testing Checklist (Do Not Skip)

- Deduct stock for an order with 2 products and 3 materials each; confirm correct totals.
- Re-run the same deduction call; confirm no double deduction (idempotency).
- Trigger two deductions concurrently; confirm stock is consistent (atomicity).
- Restock a material; confirm movement logged and stock updated.
- Low-stock alert: set qty below minimum; confirm indicator + notifications.

If you want, I can turn this into: (1) a Supabase SQL schema + RLS policies, (2) a set of n8n workflows for Trello stage sync + WhatsApp/email alerts, and (3) an admin UI page list for your web app.