FIRINGup Auto-Reorder System

Intelligent Inventory Replenishment Planning Document

Core Algorithm: When to Reorder

```
Reorder Point = (Daily Consumption Rate x Lead Time) + Safety Stock
```

1. Consumption Rate Calculation

Two primary data sources to work with:

Invoices (receiving): Know when X quantity arrived

Counts (depletion): Know when quantity dropped

Calculation approaches (simple to complex):

Simple: (qty_received) / (days_until_next_receive)

Better: Track count-to-count deltas with dates, weighted toward recent behavior

Best: If recipe data + sales exist, predict consumption from sales (more accurate than counting)

Consider: Weekly patterns (weekends busier?), seasonality, trends

2. Supplier Lead Time (User Input)

For each supplier, capture:

• Lead time in business days (order to delivery)

• Order cutoff time (e.g., 'order by 2pm for next-day')

• Delivery days (some only deliver Tue/Thu)

• Minimum order value/quantity

3. Safety Stock

Buffer for consumption variability. Two approaches:

Fixed: Always keep X days extra (e.g., 2-3 days)

Dynamic: Based on consumption variance (if flour usage swings wildly, keep more buffer)

4. Additional Factors to Consider

Factor	Why It Matters
Perishability	Don't over-order items with short shelf life
Case breaks	Round up to orderable units (can't order 0.3 cases)
Volume discounts	Maybe order more if hitting a price break
Supplier grouping	Batch items from same supplier into one order
Minimum order values	Add lower-priority items to hit supplier minimums
Price trends	Optional: flag if current price is unusually high/low vs history

5. The Notification Flow

1. System detects: Ingredient will hit reorder point in X days
2. Groups with other items from same supplier needing reorder
3. Calculates suggested quantities (consumption rate x target days of stock)
4. Sends notification to admin:

   "Sysco Order Suggested"
   "3 items projected to run low"
   "Est. total: $847"
   [Review & Approve] [Dismiss] [Snooze 1 Day]

5. On approve: Sends pre-formatted order email to supplier (or logs for manual ordering)

6. Data Requirements to Add

Per Supplier:

• Lead time (business days)

• Order cutoff time

• Delivery days

• Contact email/phone for orders

• Minimum order value

Per Ingredient (optional overrides):

• Par level (manual override of algorithm)

• Reorder enabled (yes/no toggle)

• Preferred order quantity

System Settings:

• Default safety stock days

• How far ahead to look (notify X days before reorder point)

• Notification preferences (email, SMS, both)

7. Implementation Phases

Phase 1 - Practical MVP:

• Simple moving average consumption

• Fixed lead times per supplier

• Fixed safety stock (e.g., 3 days)

• Email notification with order summary

• Manual approval triggers email to supplier

Phase 2 - Smarter:

• Weighted consumption (recent weeks matter more)

• Dynamic safety stock based on variance

• Supplier order batching with minimum value logic

• Recipe-based consumption prediction

Phase 3 - Advanced:

• Day-of-week consumption patterns

• Seasonal adjustments

• Price optimization suggestions

• Machine learning predictions

Modeled vs Actual Inventory Tracking

Modeled (Theoretical) Inventory:

• Sales x Recipe ingredients = what *should* have been used

• Updates in real-time with every sale

• Great for continuous consumption rate tracking

Actual Inventory:

• Physical counts = what's *really* there

• Ground truth, but only at count moments

The Delta Tells a Story:

```
Modeled says: 50 lbs flour remaining
Actual count: 42 lbs flour remaining
Variance: -8 lbs (16% loss)
```

This variance captures: waste/spillage, portion control drift, recipe inaccuracies, theft, recording errors

For Auto-Reorder:

1. **Consumption Rate:** Use modeled (sales-driven) for predictions - it's real-time and granular

2. **Calibration Factor:** Apply correction based on historical modeled-vs-actual variance

3. **Dynamic Safety Stock:** High variance items need bigger buffers

4. **Reorder Trigger:** Projected Depletion = Current Actual + (Daily Modeled Consumption x Variance Factor)

System Architecture: The Seven Layers

Layer 1: Data Foundation

What exists (already in DB):

• Invoices - receiving dates, quantities, costs, supplier

• Inventory counts - actual quantities at points in time

• Sales transactions - what was sold, when

• Recipes - ingredient requirements per product

• Suppliers - contact info, terms

Database additions needed:

suppliers table additions:

```
lead_time_days (integer)
order_cutoff_time (e.g., '14:00')
delivery_days (e.g., 'tue,thu,sat')
order_email
minimum_order_value
```

ingredients table additions:

```
reorder_enabled (boolean, default true)
par_level_override (nullable - manual override)
safety_stock_days (nullable - per-item override)
```

new table: reorder_settings

```
default_safety_stock_days
lookahead_days (how far ahead to warn)
notification_email
notification_phone
```

Layer 2: Consumption Engine

Runs periodically (or on-demand) to calculate consumption metrics per ingredient:

1. MODELED CONSUMPTION (from sales)

• Query recent sales (e.g., last 30 days)

• Join to recipes -> sum ingredient usage

• Calculate: daily_modeled_consumption = total_used / days

2. ACTUAL CONSUMPTION (from counts)

• Query count history + invoice receipts

• Calculate: actual_depleted = (starting + received - ending)

• Calculate: daily_actual_consumption = actual_depleted / days

3. VARIANCE FACTOR

• variance = actual / modeled (e.g., 1.15 means 15% more actual)

• Store rolling average variance

4. ADJUSTED CONSUMPTION RATE

• adjusted_daily = daily_modeled x variance_factor

• This is your best predictor

Output: ingredient_consumption_metrics table

• ingredient_id

• daily_modeled_rate

• daily_actual_rate

• variance_factor

• confidence_score (based on data quality/quantity)

• last_calculated

Layer 3: Reorder Point Calculator

For each ingredient, determine when to reorder:

```
current_stock = latest count (or modeled current)
adjusted_daily_consumption = from Layer 2
lead_time = supplier.lead_time_days
safety_stock = ingredient.safety_stock_days OR settings.default

days_of_stock = current_stock / adjusted_daily_consumption
reorder_point_days = lead_time + safety_stock
```

TRIGGER when: days_of_stock <= reorder_point_days + lookahead_days

Output: reorder_alerts table

• ingredient_id

• current_stock

• days_remaining

• suggested_order_qty (e.g., 2 weeks of stock)

• urgency (critical/soon/upcoming)

• supplier_id

• created_at

• status (pending/notified/approved/dismissed)

Layer 4: Order Aggregator

Groups alerts into actionable supplier orders:

For each supplier with pending alerts:

1. Collect all ingredients needing reorder

2. Calculate quantities (round to case sizes)

3. Check against minimum order value

   - If under minimum: flag lower-priority items to add

4. Calculate estimated total cost

5. Determine optimal order date (based on delivery days)

Output: draft_order

• supplier_id

• order_date

• delivery_date (estimated)

• line_items: [{ingredient, qty, unit, est_cost}]

• total_estimated

• status: draft

Layer 5: Notification Service

Sends alerts to admins when draft_order created or updated:

Example Notification:

Sysco Order Recommended
4 items need reordering
Est. total: $1,247
Order by: Tomorrow 2pm for Thu delivery

- Flour (50lb) x 2 cases - $89
- Tomatoes (case) x 3 - $156
- Cheese (block) x 5 - $423
- Olive Oil (gal) x 2 - $67

[Approve] [Modify] [Snooze 1 Day] [Dismiss]

Delivery channels:

• Email (with action links)

• SMS (short version + link)

• In-app notification

Layer 6: Action Handler

Processes admin responses:

On APPROVE:

1. Generate formal order (PDF or email body)

2. Send to supplier.order_email

3. Log order in purchase_orders table

4. Update reorder_alerts status

5. Set expected_delivery_date

On MODIFY:

• Open order editor UI

• Allow qty changes, item removal

• Re-approve when ready

On SNOOZE:

• Set alert.snooze_until = now + 1 day

• Will re-notify tomorrow

On DISMISS:

• Mark alert dismissed

• Optional: 'Don't suggest for X days'

Layer 7: Learning Loop (Phase 2+)

Improves predictions over time:

After each count:

1. Compare predicted vs actual

2. Adjust variance_factor

3. Flag recipes that may be wrong

After each delivery:

1. Record actual lead time

2. Update supplier.lead_time_days (rolling avg)

Weekly:

1. Recalculate consumption patterns

2. Detect trend changes

3. Flag anomalies for review

Visual Architecture Overview

ADMIN DASHBOARD
Notifications | Pending Orders | Settings

NOTIFICATION SERVICE
Email / SMS / In-App Alerts

ORDER AGGREGATOR
Group by supplier, check minimums, schedule

REORDER POINT CALCULATOR
When to order? How much? How urgent?

CONSUMPTION ENGINE
Modeled (sales x recipes) + Actual (counts) + Variance

DATA FOUNDATION
Invoices | Counts | Sales | Recipes | Suppliers

Key Questions to Resolve

1. What communication method preferred for supplier orders (email, portal, phone)?
2. Should the system auto-send orders or always require human approval?
3. What level of safety stock is acceptable (cost of overstock vs risk of stockout)?
4. Are there seasonal patterns that need to be accounted for?
5. Which suppliers have API/portal ordering vs email-only?