Stock Monitoring Feature (Estimated Restock Date System)

1. Overview

This document describes the complete workflow, logic, and database structure for the **Stock Monitoring** feature implemented in the MVP. This feature helps users anticipate when essential household or personal items will run out, without requiring manual quantity measurement.

The system uses **estimated depletion duration** and **early warning periods** to automatically compute when a restocking alert should be generated.

---

2. User Workflow Summary

The entire workflow from adding an item to receiving a warning can be summarized as follows:

Step 1 — User Adds an Item

The user provides: - **Item Name** (e.g., "Ice", "Rice", "Deodorant") - **Days Until Empty** → Approximate number of days the item typically lasts - **Warning Days Before** → How many days before the expected depletion to notify

Example: - Ice lasts ~10 days → `days_until_empty = 10` - User wants a reminder 3 days early → `warning_days_before = 3`

Step 2 — System Computes Restock Date

Upon creation (or update), the backend calculates:

```
estimated_restock_date = created_at + days_until_empty
```

If created on Jan 1 and lasts 10 days: - Restock date = Jan 11 - Warning period begins = Jan 8

This value is stored in the database.

Step 3 — Daily System Check

A backend cron job or scheduled process runs once per day and checks:

```
If current_date >= estimated_restock_date - warning_days_before:
      Trigger low stock warning
```

Warnings can be delivered via: - Dashboard notifications - Email (future enhancement) - In-app alerts

---

Step 4 — User Restocks the Item

When the user buys the item again, they click **“Restocked”**.

The backend then: 1. Updates `created_at` to the current date 2. Recalculates `estimated_restock_date`

This resets the cycle.

---

3. Database Schema (Final MVP Version)

```
CREATE TABLE stock_items (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,

    item_name VARCHAR(255) NOT NULL,

    days_until_empty INT NOT NULL,                    -- Estimated duration
    warning_days_before INT NOT NULL DEFAULT 2,       -- Days before to warn
    estimated_restock_date DATE NOT NULL,             -- Auto-calculated

    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    INDEX (user_id)
);
```

---

4. Restocking Process Logic

When the user taps the "Restock" button:

Backend performs:

```
created_at = CURRENT_TIMESTAMP
estimated_restock_date = CURRENT_DATE + INTERVAL days_until_empty DAY
```

User sees:

- Last restocked date → updated
- Next restock date → recalculated

- Warning schedule → updated

---

5. Low Stock Warning Logic

A daily scheduled script runs:

Pseudocode

```
for each stock item:
    if today >= (estimated_restock_date - warning_days_before):
        status = "LOW STOCK"
    else:
        status = "OK"
```

Example

Item: Ice - lasts 10 days - warn 3 days early - restock: Jan 1

Results: - estimated_restock_date = Jan 11 - warnings start from Jan 8

---

6. Advantages of This System

- No need to track physical quantity
- Very user-friendly
- Works for all items with predictable consumption
- Minimal data entry
- Perfect for a college lifestyle
- Reduces cognitive load and improves household planning

---

7. Future Enhancements (Post-MVP)

- Visual consumption timeline
- Automated learning (system adjusts days_until_empty based on restock history)
- Smart consumption estimation using weekly usage frequency
- Push notifications
- Multi-user shared household stock tracking

---

8. Conclusion

This feature is now fully defined, documented, and ready for implementation in both backend and frontend.

Next steps include building: - Backend REST endpoints for stock management - Cron job functionality - React UI for item creation, display, warnings, and restocking actions.