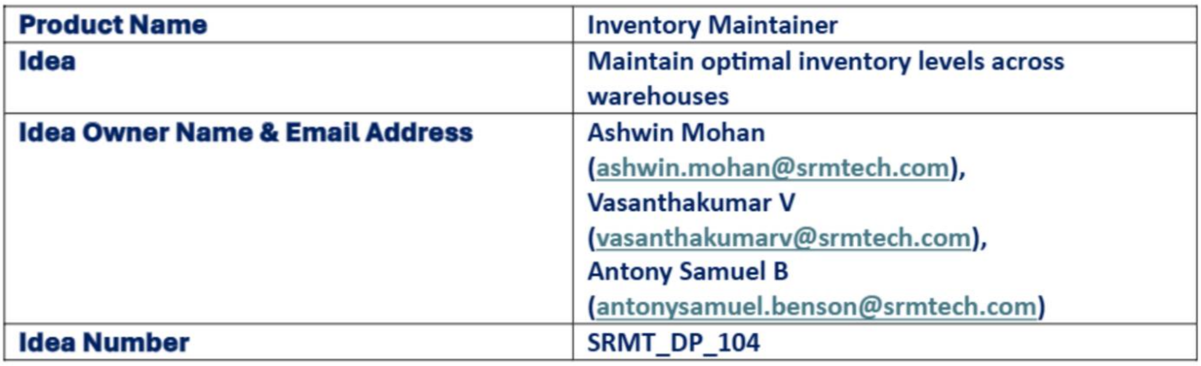
LogiTrack : The Inventory Maintainer

Detailed Execution Plan

CODE TRACK SELECTED: Resourcify - Efficiency Redefined, Resource Optimized

IDEA SELECTED:



Product Name	Inventory Maintainer
Idea	Maintain optimal inventory levels across warehouses
Idea Owner Name & Email Address	Ashwin Mohan (ashwin.mohan@srmtech.com), Vasanthakumar V (vasanthakumarv@srmtech.com), Antony Samuel B (antonysamuel.benson@srmtech.com)
Idea Number	SRMT_DP_104

TEAM NAME: NEURAL NEXUS

TEAM MEMBERS:

1) TANISH PODDAR
2) NIDHI NAYANA
3) NISHANT RANJAN

Phase 1: Core Optimization Engine

OBJECTIVE: Build the brain of the tool to calculate the best way to distribute inventory.

1. **DATA PREPARATION**
   - Mock Data Creation
     - Create sample sales data (e.g., product IDs, regions, historical sales volumes, dates).
     - Define warehouse details (locations, storage capacity, operational costs).
     - Save data in CSV files or a lightweight SQLite database for easy access.
   - Example
     - Warehouse A: Located in Texas, max capacity = 10,000 units, cost per unit = $2.
     - Product X: Sold 500 units/month in California, and 300 units/month in New York.

2. **OPTIMIZATION MODEL DEVELOPMENT**
   - Linear Programming Setup
     - Use Python's PuLP library to define the problem:
       - Goal: Minimize total costs (transportation + storage).
       - Variables: How much inventory to send from each warehouse to each region.
       - Constraints: Warehouse capacity, regional demand fulfilment.
   - Math Behind It
     - Example equation: Total Cost = (Cost to ship from Warehouse A to Region X × Units shipped) + (Warehouse A storage cost × Units stored)
     - Solve using PuLP's algorithms to find the cheapest, most efficient distribution plan.

3. **VALIDATION**
   - Test the model with mock data to ensure it:
     - Avoids split shipments (e.g., sending Product X to California from both Texas and Ohio).
     - Prioritizes warehouses closer to high-demand regions to reduce delivery miles.
   - Generate metrics like "15% fewer shipments" or "10% lower transportation costs."

Phase 2: User Dashboard Development

Objective: Create a simple interface for users to interact with the tool.

1. **Streamlit Dashboard Setup**
    - **Features**:
        - Data Upload: Users can upload sales/warehouse data via CSV.
        - Optimization Button: Click to run the model and generate recommendations.
        - Results Display: Show optimized inventory allocation, cost savings, and warehouse utilization.
2. **Visualization**
    - **Geographic Maps**: Use Python's folium library to plot warehouse locations and demand hotspots.
    - **Graphs**:
        - Bar charts comparing costs before/after optimization.
        - Pie charts showing warehouse utilization (e.g., 80% of Warehouse A's capacity used).

Phase 3: Integration & Testing

Objective: Ensure all components work together smoothly.

1. **Backend-Frontend Connection**
    - Link the optimization engine to the Streamlit dashboard so users get instant results.
    - Example: When a user clicks "Optimize," the model processes data and returns results in <10 seconds.
2. **Edge Case Testing**
    - Test scenarios like:
        - Sudden Demand Spike: What if a region's demand doubles overnight?
        - Warehouse Closure: How does the tool reroute inventory if a warehouse is unavailable?
    - Adjust the model to handle these cases (e.g., prioritize backup warehouses).

Phase 4: Additional Features

Objective: Add advanced capabilities if time allows.

1. **Demand Forecasting**
   - Integrate a **time-series model** (e.g., Facebook's Prophet library) to predict future sales trends.
   - Example: Use past 3 years of sales data to forecast next quarter's demand for Product X.

2. **Simulation Mode**
   - Let users tweak variables (e.g., warehouse capacity, transport costs) via sliders in the dashboard.
   - Show real-time impacts: "Increasing Warehouse A's capacity by 20% reduces costs by $5,000."

Phase 5: Final Touch

Objective: Give a final touch to the project.

1. **Key Metrics**
   - Highlight outcomes like:
     - 15% reduction in split shipments.
     - 12% lower transportation costs.
     - 20% faster order picking due to optimized warehouse layouts.
2. **Demo Flow**
   - **Step 1**: Show the problem (e.g., inefficient inventory allocation causing high costs).
   - **Step 2**: Upload sample data to the dashboard and click "Optimize."
   - **Step 3**: Display results: maps, graphs, and cost savings.
   - **Step 4**: Simulate a "what-if" scenario (e.g., warehouse closure) to show adaptability.

Tools & Technologies

- **Python**: Programming language.
- **PuLP**: Library for solving optimization problems (like a math wizard for inventory).
- **Streamlit**: Turns Python scripts into interactive web apps (no web development needed).
- **SQLite/CSV**: Lightweight data storage.
- **Plotly/Folium**: Creates graphs and maps for visual storytelling.