Multi-Agentic Solution (Reasoning + Function Calling + ReAct)

Goals

- Enable a multi-agent workflow that reasons over procurement tasks, calls tools/APIs, and records actions.
- Support retrieval (Vector DB) for catalog, vendor, and policy knowledge.
- Integrate POS data for consumption, depletion, and forecasting.
- Enforce safety and data validation with guardrails and schemas.

Recommended Stack (modular)

- Orchestration: LangChain **or** CrewAI
- Agent framework (optional): AutoGPT (for long-running goal loops)
- Vector DB: Pinecone **or** Weaviate
- Guardrails: NVIDIA NeMo Guardrails **or** Pydantic-based validation
- API layer: FastAPI
- Storage: Postgres (transactions, audit logs) + Vector DB (embeddings)

Agent Roles

1. **Reasoning Agent (Planner)**
    - Breaks down tasks into steps and decides which tools to call.
    - Produces a plan and success criteria.
2. **Catalog/Normalization Agent**
    - Normalizes SKU names, units, and pack sizes.
    - Calls parsers and unit conversion tools.
3. **Sourcing/Comparison Agent**
    - Compares suppliers on normalized price, lead time, MOQ, reliability.
    - Produces ranked options and justifications.
4. **Autopilot Purchasing Agent**
    - Generates suggested carts based on par levels, lead time, and run-rate.
    - Requires approval workflow.
5. **Compliance & Finance Agent**
    - Runs 2-way/3-way matches (PO vs GRN vs Invoice).
    - Flags exceptions, ensures audit logs.
6. **POS/Inventory Agent**
    - Ingests POS depletion and inventory adjustments.
    - Generates low-stock alerts and reorder triggers.

ReAct Pattern (Reason + Act)

Each agent follows a structured loop:

1. **Reason**: analyze current state + objective.
2. **Act**: call tools (DB, POS API, catalog parser, vector search).
3. **Observe**: inspect results.
4. **Update**: refine plan or finalize action.

Function Calling (tooling)

Key tools to expose:

- `parse_pack(pack_text)` -> PackInfo
- `normalize_name(name)` -> normalized string
- `convert_to_kg(pack_info)` -> kg
- `compare_quotes(items)` -> sorted options
- `generate_po(draft)` -> PO
- `record_grn(receipt)` -> GRN
- `match_invoice(po, grn, invoice)` -> match result
- `fetch_pos_sales(range)` -> sales summary
- `fetch_inventory()` -> stock snapshot
- `save_audit_log(entry)`

Retrieval (Vector DB)

Index data:

- Supplier catalogs
- Price lists
- Policies (approvals, substitution rules)
- Product specs and substitutions

Common queries:

- “Find equivalent SKUs for X”
- “Approved substitutes for Y”
- “Recent pricing trends for Z”

POS Integrations (API)

Minimum data required:

- Item sales by day (SKU, qty)
- Inventory adjustments
- Locations/branches

Approach:

- Start with one POS (e.g., Foodics or Oracle Micros) and implement a connector.
- Normalize POS item IDs to internal Normalized SKU mapping.

Guardrails

- **NeMo Guardrails** for policy constraints and safe responses.
- **Pydantic** for strict schema validation of tool inputs/outputs.

Phased Delivery (aligned with PRD)

- Phase 1: Catalog normalization + quote comparison
- Phase 2: GRN + invoice matching + audit logs
- Phase 3: Inventory + low stock
- Phase 4: AI suggested cart + approvals

Acceptance Criteria (technical)

- All AI actions are logged with inputs, tool calls, and outputs.
- Human approval required for PO creation.
- SKU normalization achieves comparable $/kg across suppliers.