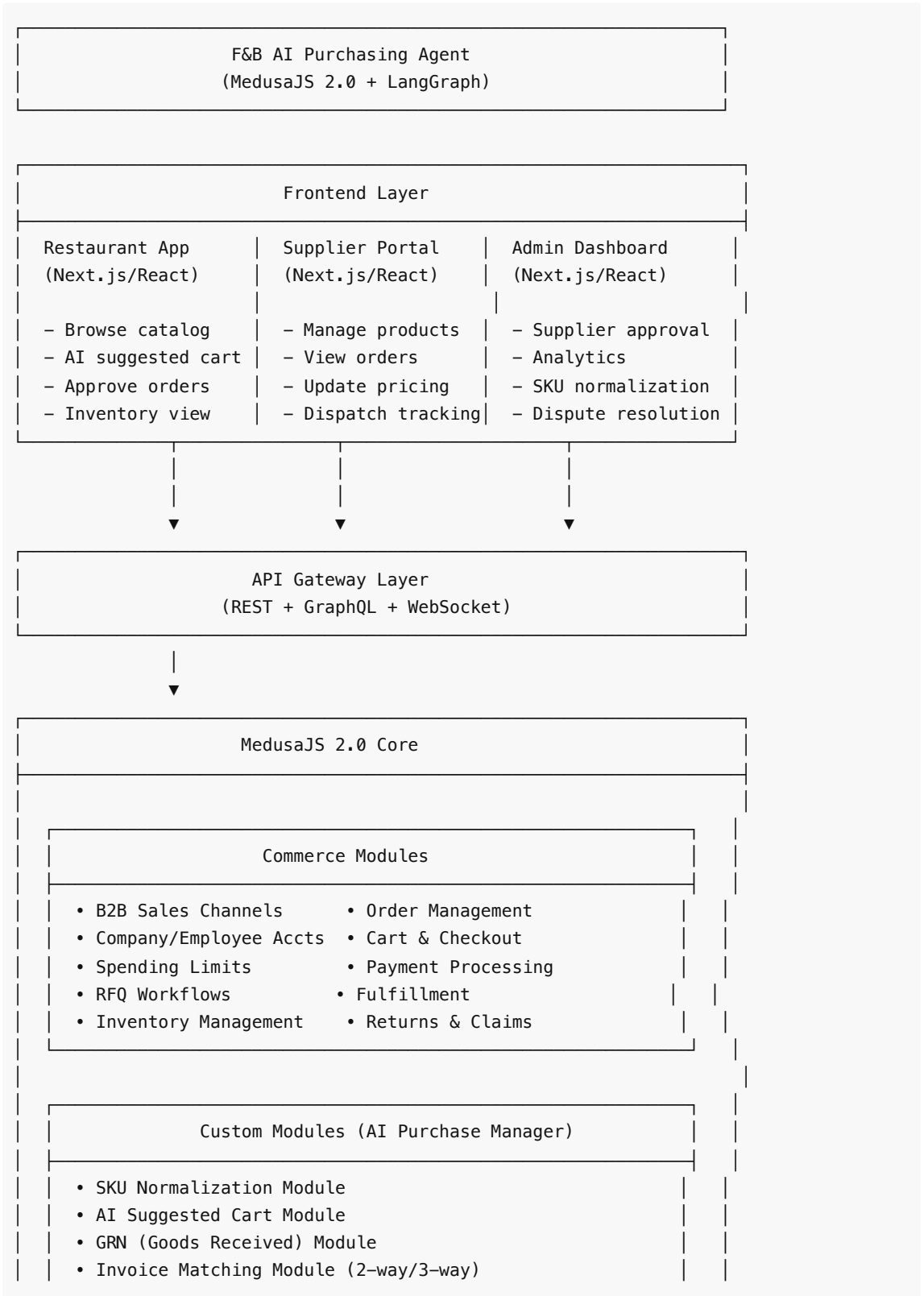
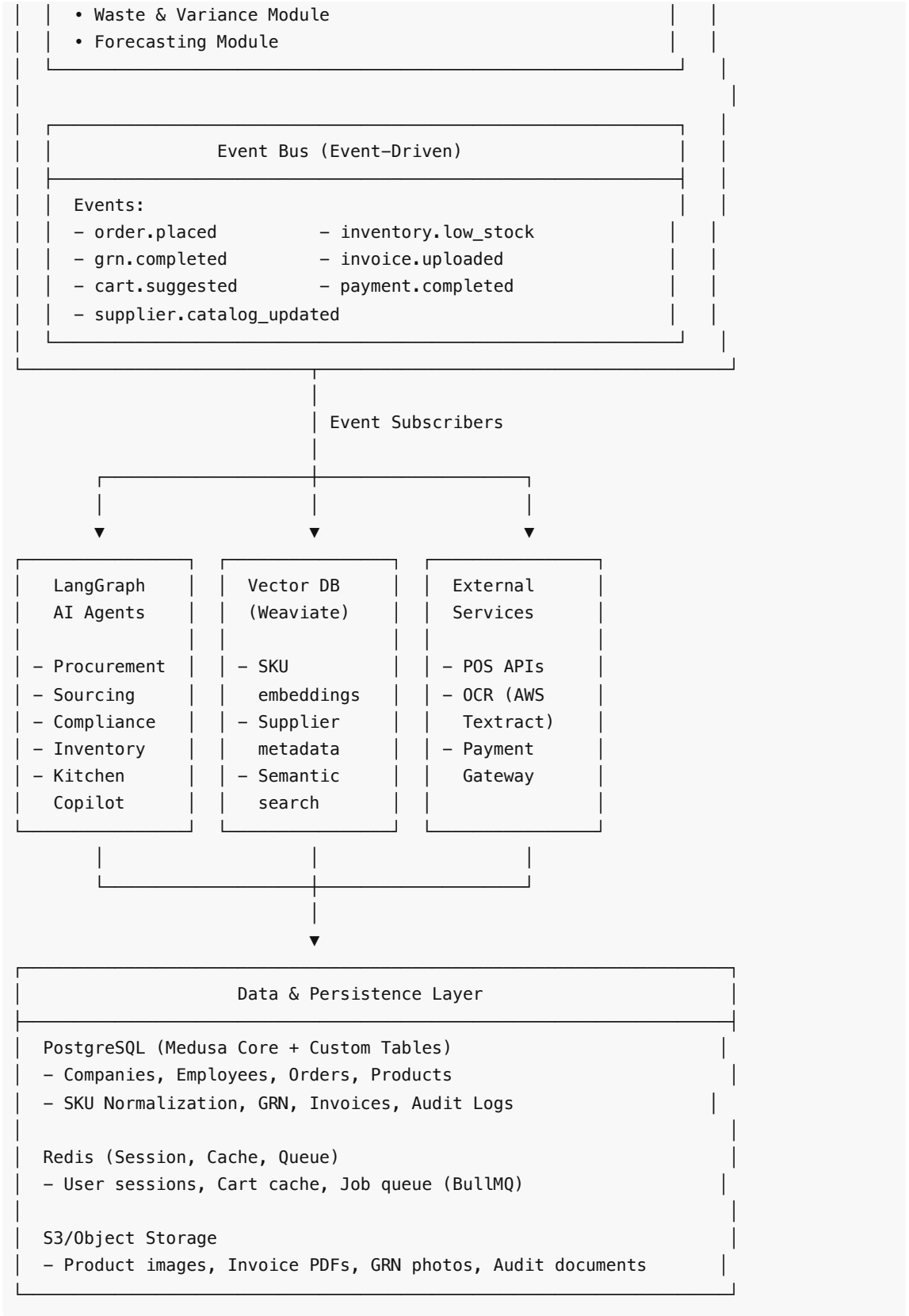
MedusaJS 2.0 + LangGraph Architecture

System Overview





MedusaJS 2.0 B2B Architecture

Core B2B Features (Out-of-the-Box)

```
┌─────────────────────────────────────────────────────────────┐
│                    MedusaJS B2B Starter                     │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  Company Accounts                                           │
│  ├─ Company profile (name, tax ID, billing address)         │
│  ├─ Employee sub-accounts with roles                        │
│  ├─ Spending limits per employee/department                 │
│  └─ Approval workflows (cart → manager → PO)                │
│                                                             │
│  Request for Quote (RFQ)                                    │
│  ├─ Create RFQ with multiple items                          │
│  ├─ Send to multiple suppliers                              │
│  ├─ Receive & compare quotes                                │
│  └─ Convert quote → order                                   │
│                                                             │
│  Custom Pricing                                             │
│  ├─ Price lists per company                                 │
│  ├─ Volume discounts                                        │
│  ├─ Contract pricing                                        │
│  └─ Currency support (AED, USD, etc.)                       │
│                                                             │
│  Payment Terms                                              │
│  ├─ Net 30, Net 60 terms                                    │
│  ├─ Credit limits                                           │
│  ├─ Invoice generation                                      │
│  └─ Payment tracking                                        │
└─────────────────────────────────────────────────────────────┘
```

Custom Module Integration Points

MedusaJS allows you to inject custom logic at key lifecycle points:

```
┌─────────────────────────────────────────────────────────────┐
│          MedusaJS Custom Module Injection Points            │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  1. Subscriber Pattern (Event-Driven)                       │
│     ┌─────────────────────────────────────────────┐         │
│     │  order.placed event                         │         │
│     │         ↓                                   │         │
│     │  Custom Subscriber listens                  │         │
│     │         ↓                                   │         │
│     │  Trigger LangGraph Agent                    │         │
│     │         ↓                                   │         │
│     │  AI validates order → update status         │         │
│     └─────────────────────────────────────────────┘         │
│                                                             │
│  2. API Routes (Custom Endpoints)                           │
```

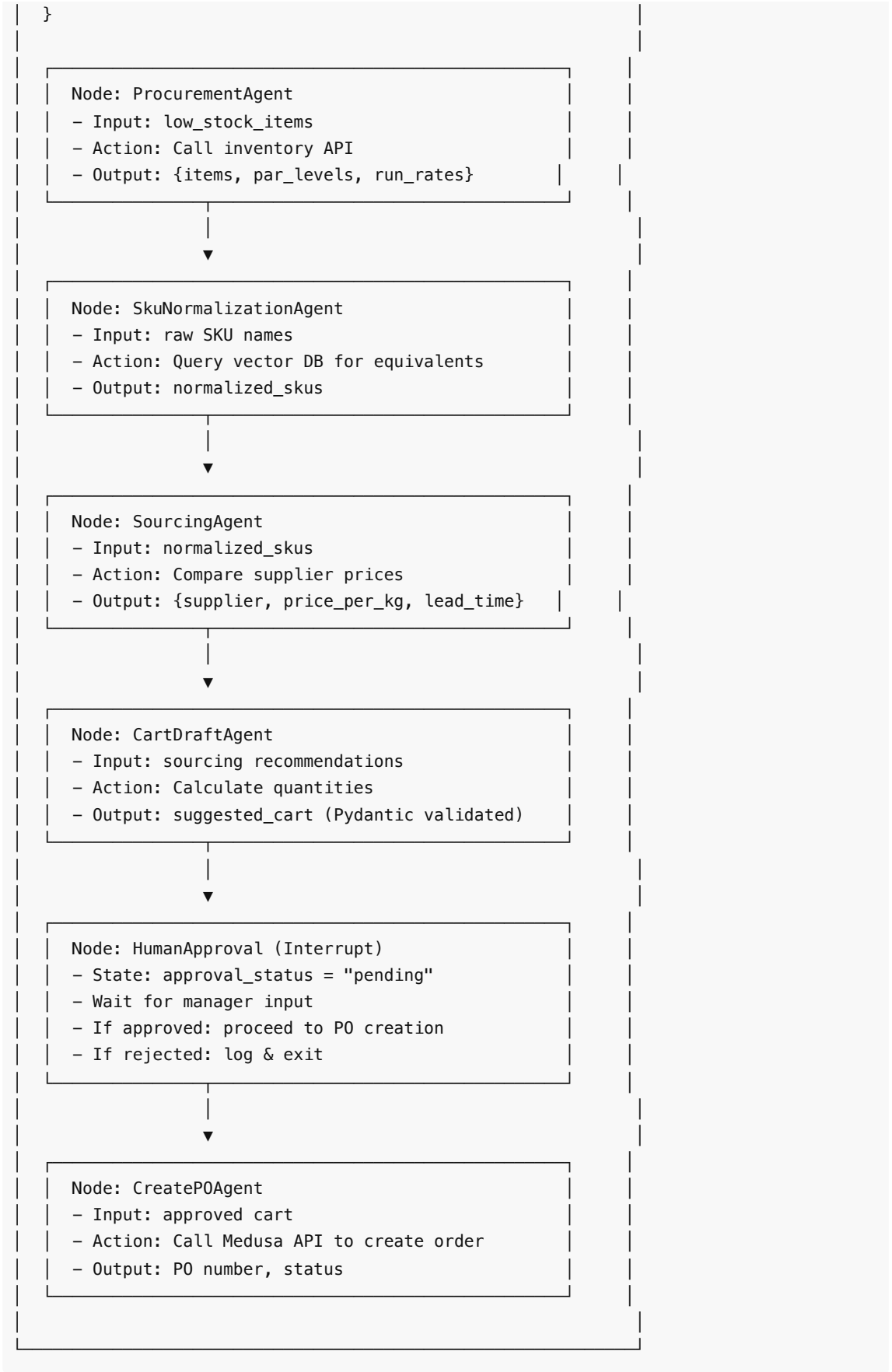
```
│     ┌──────────────────────────────────────────────────┐      │
│     │  POST /admin/sku-normalize                       │      │
│     │  POST /store/ai-cart-suggest                     │      │
│     │  POST /admin/invoice-match                       │      │
│     │  GET  /store/waste-insights                      │      │
│     └──────────────────────────────────────────────────┘      │
│                                                                │
│  3. Middleware (Request/Response Interception)                 │
│     ┌──────────────────────────────────────────────────┐      │
│     │  Inject AI pricing recommendations               │      │
│     │  Add explainability metadata to cart             │      │
│     │  Log all financial actions to audit trail        │      │
│     └──────────────────────────────────────────────────┘      │
│                                                                │
│  4. Services (Business Logic Layer)                            │
│     ┌──────────────────────────────────────────────────┐      │
│     │  class SkuNormalizationService {                 │      │
│     │    async normalize(catalog) { ... }              │      │
│     │  }                                               │      │
│     │  class AiSuggestedCartService {                  │      │
│     │    async generate(branch, parLevels) { ... }     │      │
│     │  }                                               │      │
│     └──────────────────────────────────────────────────┘      │
│                                                                │
│  5. Workflows (Multi-Step Transactions)                        │
│     ┌──────────────────────────────────────────────────┐      │
│     │  createOrderWithGrnWorkflow:                     │      │
│     │    step1: validate cart                          │      │
│     │    step2: create order                           │      │
│     │    step3: reserve inventory                      │      │
│     │    step4: notify supplier                        │      │
│     │    step5: schedule GRN reminder                  │      │
│     └──────────────────────────────────────────────────┘      │
└────────────────────────────────────────────────────────────────┘
```

LangGraph Agentic Workflows

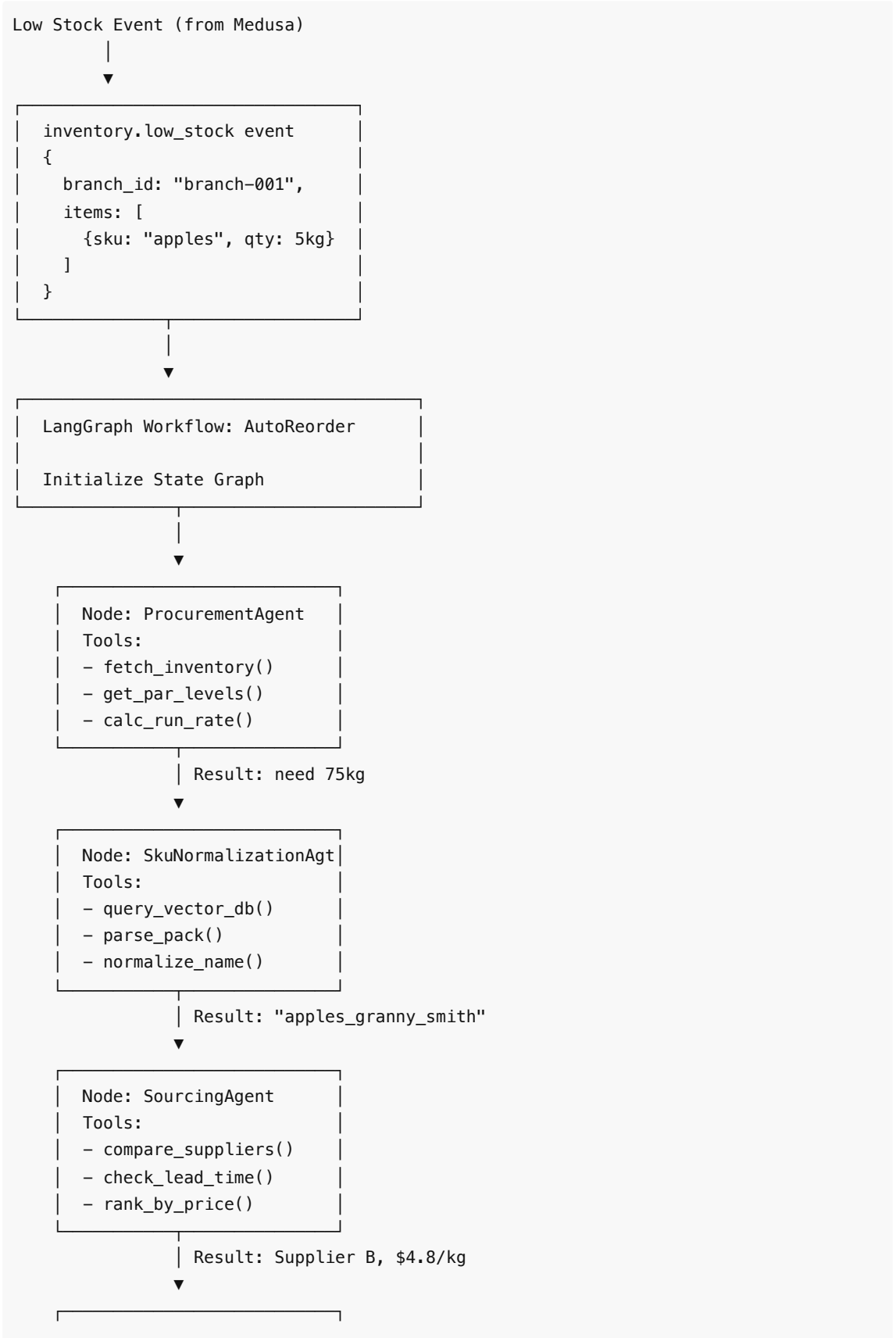
LangGraph enables **stateful, multi-step AI reasoning** with tool-calling and human-in-the-loop approval.

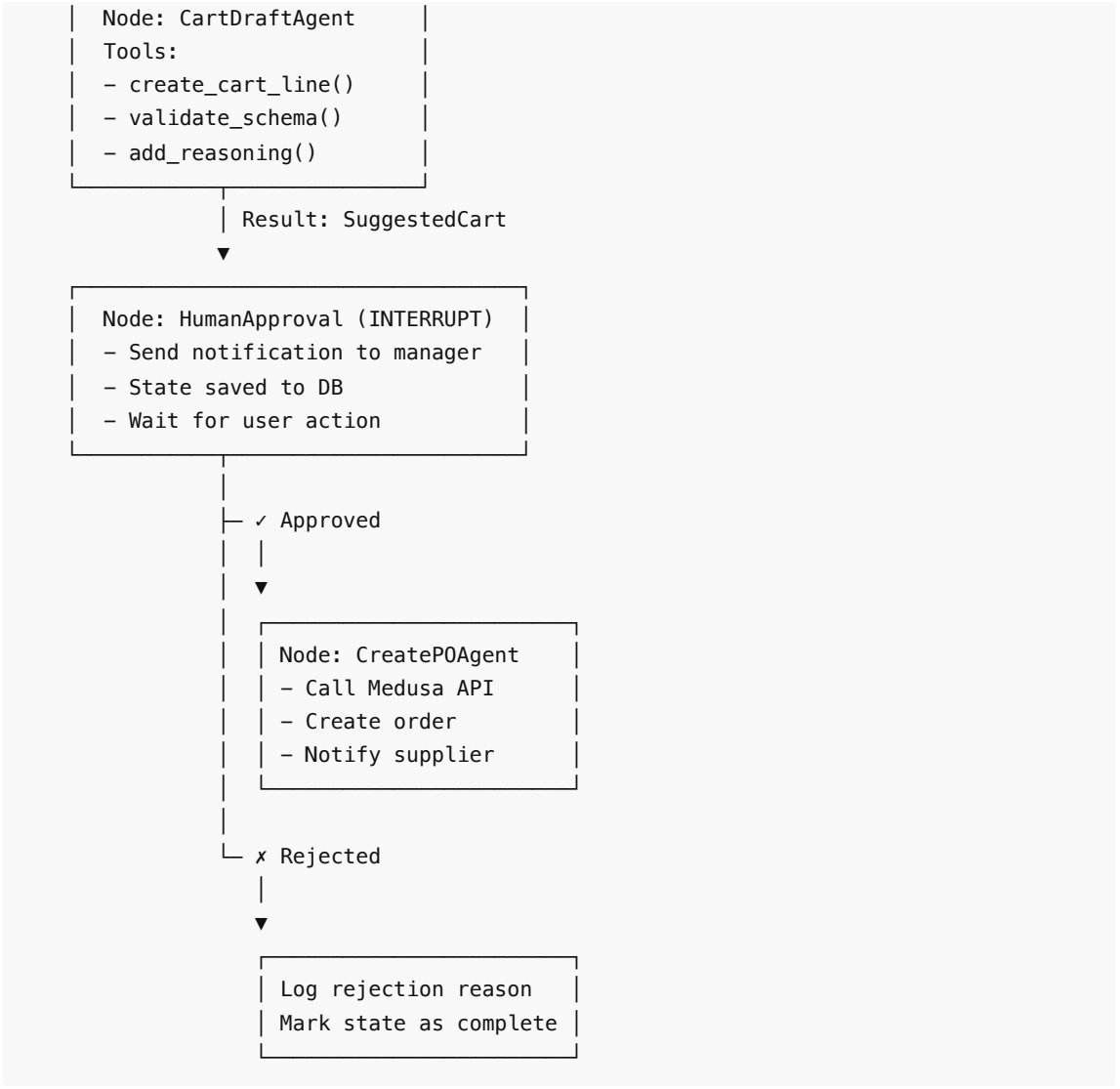
LangGraph Architecture

```
┌──────────────────────────────────────────────────────────┐
│                   LangGraph State Graph                  │
├──────────────────────────────────────────────────────────┤
│                                                          │
│  State: {                                                │
│    branch_id: "branch-001",                              │
│    low_stock_items: [...],                               │
│    suggested_cart: null,                                 │
│    approval_status: "pending",                           │
│    tool_logs: []                                         │
```


}
Node: ProcurementAgent
- Input: low_stock_items
- Action: Call inventory API
- Output: {items, par_levels, run_rates}
Node: SkuNormalizationAgent
- Input: raw SKU names
- Action: Query vector DB for equivalents
- Output: normalized_skus
Node: SourcingAgent
- Input: normalized_skus
- Action: Compare supplier prices
- Output: {supplier, price_per_kg, lead_time}
Node: CartDraftAgent
- Input: sourcing recommendations
- Action: Calculate quantities
- Output: suggested_cart (Pydantic validated)
Node: HumanApproval (Interrupt)
- State: approval_status = "pending"
- Wait for manager input
- If approved: proceed to PO creation
- If rejected: log & exit
Node: CreatePOAgent
- Input: approved cart
- Action: Call Medusa API to create order
- Output: PO number, status

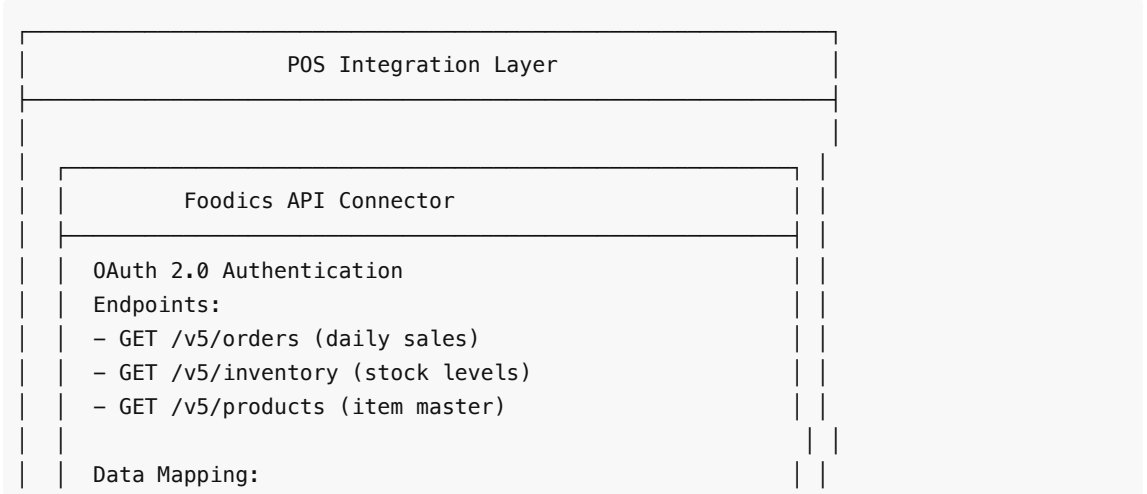
LangGraph Event Flow





Data Acquisition Layer

POS Integration Architecture



```
│   │  Foodics Product ID → Internal Normalized SKU          │ │
│   │                                                        │ │
│   │  Webhook Support:                                      │ │
│   │  - order.created → depletes inventory                  │ │
│   │  - inventory.adjusted → sync stock                     │ │
│   └────────────────────────────────────────────────────────┘ │
│                                                              │
│   ┌────────────────────────────────────────────────────────┐ │
│   │       Oracle Simphony Connector                        │ │
│   ├────────────────────────────────────────────────────────┤ │
│   │  STSG2 REST API                                        │ │
│   │  Endpoints:                                            │ │
│   │  - GET /transactions (POS sales)                       │ │
│   │  - GET /menuItems (menu with recipes)                  │ │
│   │  - GET /inventory (stock depletion)                    │ │
│   │                                                        │ │
│   │  Recipe-Based Depletion:                               │ │
│   │  Order "Chicken Burger" → deplete:                     │ │
│   │    - 150g chicken breast                               │ │
│   │    - 1 bun                                             │ │
│   │    - 20g lettuce, etc.                                 │ │
│   └────────────────────────────────────────────────────────┘ │
│                                                              │
│   ┌────────────────────────────────────────────────────────┐ │
│   │         Generic POS Adapter (Fallback)                 │ │
│   ├────────────────────────────────────────────────────────┤ │
│   │  CSV/Excel Upload:                                     │ │
│   │  - Daily sales report upload                           │ │
│   │  - Manual SKU mapping                                  │ │
│   │  - Scheduled ETL jobs                                  │ │
│   └────────────────────────────────────────────────────────┘ │
└──────────────────────────────────────────────────────────────┘
```

Invoice OCR Pipeline

```
┌──────────────────────────────────────────────────────────────┐
│                    Invoice OCR Pipeline                      │
├──────────────────────────────────────────────────────────────┤
│                                                              │
│  Step 1: Upload                                              │
│   ┌────────────────────────────────────────────────────────┐ │
│   │  Supplier uploads PDF/image via app                    │ │
│   │         ↓                                              │ │
│   │  Store in S3: invoices/{supplier_id}/{invoice_no}.pdf  │ │
│   └────────────────────────────────────────────────────────┘ │
│                                                              │
│  Step 2: OCR Extraction                                      │
│   ┌────────────────────────────────────────────────────────┐ │
│   │  Trigger: invoice.uploaded event                       │ │
│   │         ↓                                              │ │
│   │  AWS Textract (AnalyzeExpense API)                     │ │
```

```
│  │  OR                                                    │ │
│  │  Google Document AI (Invoice Parser)                   │ │
│  │         ↓                                              │ │
│  │  Extracted:                                            │ │
│  │  {                                                     │ │
│  │    invoice_no: "INV-12345",                            │ │
│  │    date: "2026-02-04",                                 │ │
│  │    supplier: "Global Foods",                           │ │
│  │    line_items: [                                       │ │
│  │      {desc: "Apples Granny Smith",                     │ │
│  │       qty: "50kg", price: "$240"},                     │ │
│  │      ...                                               │ │
│  │    ],                                                  │ │
│  │    total: "$1,250"                                     │ │
│  │  }                                                     │ │
│  └────────────────────────────────────────────────────────┘ │
│                                                             │
│  Step 3: AI Validation & Normalization                      │
│  ┌────────────────────────────────────────────────────────┐ │
│  │  LangGraph Agent: InvoiceValidationAgent               │ │
│  │  - Parse line items with LLM                           │ │
│  │  - Map to Normalized SKUs                              │ │
│  │  - Extract pack info (50kg → 50 × 1kg)                 │ │
│  │  - Calculate price per kg                              │ │
│  └────────────────────────────────────────────────────────┘ │
│                                                             │
│  Step 4: 2-Way / 3-Way Match                                │
│  ┌────────────────────────────────────────────────────────┐ │
│  │  Compare:                                              │ │
│  │  - PO line items                                       │ │
│  │  - GRN received quantities                             │ │
│  │  - Invoice billed quantities                           │ │
│  │         ↓                                              │ │
│  │  Flag exceptions:                                      │ │
│  │  - Short delivery (GRN < Invoice)                      │ │
│  │  - Overcharge (Invoice price > PO price)               │ │
│  │  - Missing items                                       │ │
│  └────────────────────────────────────────────────────────┘ │
└─────────────────────────────────────────────────────────────┘
```

SKU Normalization Engine (The Moat)

Vector DB Architecture (Weaviate)

```
┌─────────────────────────────────────────────────────────────┐
│                Weaviate Vector Database Schema              │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  Collection: NormalizedSKU                                  │
│  ┌────────────────────────────────────────────────────────┐ │
│  │  {                                                     │ │
```

```
│  │    id: "uuid",                                          │ │
│  │    name: "Apples Granny Smith",                         │ │
│  │    category: "Fresh Produce",                           │ │
│  │    attributes: {                                        │ │
│  │      grade: "A",                                        │ │
│  │      origin: "UAE",                                     │ │
│  │      organic: false                                     │ │
│  │    },                                                    │ │
│  │    unit: "kilogram",                                    │ │
│  │    aliases: ["Green Apples", "Pommes Granny Smith"],    │ │
│  │    embedding: [0.123, 0.456, ...],  ← OpenAI Ada-002    │ │
│  │    equiv_group: "apples_granny_smith_001"               │ │
│  │  }                                                       │ │
│  └──────────────────────────────────────────────────────────┘ │
│                                                               │
│  Collection: SupplierCatalog                                  │
│  ┌──────────────────────────────────────────────────────────┐ │
│  │  {                                                       │ │
│  │    id: "uuid",                                          │ │
│  │    supplier_id: "supplier-001",                         │ │
│  │    sku: "GFD-001",                                      │ │
│  │    name: "Fresh Apples Granny Smith",                   │ │
│  │    pack: "10 x 1kg",                                    │ │
│  │    unit_price: 38.00,                                   │ │
│  │    currency: "USD",                                     │ │
│  │    normalized_sku_id: "uuid",  ← Link to NormalizedSKU  │ │
│  │    embedding: [0.120, 0.452, ...],                      │ │
│  │    last_updated: "2026-02-04"                           │ │
│  │  }                                                       │ │
│  └──────────────────────────────────────────────────────────┘ │
└───────────────────────────────────────────────────────────────┘
```

Normalization Workflow

```
New Supplier Catalog Upload
         │
         ▼
┌──────────────────────────────┐
│  Admin uploads CSV/Excel     │
│  Columns:                    │
│  - SKU, Name, Pack, Price    │
└──────────────┬───────────────┘
               │
               ▼
┌──────────────────────────────────────────────────────┐
│  LangGraph Agent: CatalogNormalizationAgent          │
│                                                       │
│  For each row:                                        │
│  Step 1: Generate embedding                           │
│    - Input: "Fresh Apples Granny Smith 10x1kg"       │
│    - Call: OpenAI embeddings API                      │
```

```
│    - Output: [0.123, 0.456, ...]                    │
│                                                      │
│  Step 2: Similarity search in Weaviate               │
│    - Query vector DB with embedding                  │
│    - Return top 3 matches with cosine similarity     │
│    - Threshold: > 0.85 = "same product"              │
│                                                      │
│  Step 3: LLM attribute extraction                    │
│    - Prompt:                                         │
│      "Extract: weight, grade, origin, organic        │
│       from 'Fresh Apples Granny Smith 10x1kg'"       │
│    - Output: {weight: 10kg, grade: null, ...}        │
│                                                      │
│  Step 4: Pack parsing                                │
│    - Input: "10 x 1kg"                               │
│    - Parse: {count: 10, size: 1, unit: "kg"}         │
│    - Total weight: 10kg                              │
│                                                      │
│  Step 5: Price normalization                         │
│    - unit_price / total_weight = price_per_kg        │
│    - $38 / 10kg = $3.80/kg                           │
│                                                      │
│  Step 6: Link or create Normalized SKU               │
│    - If match found: link supplier SKU               │
│    - If no match: create new Normalized SKU          │
│    - Update Weaviate                                 │
└──────────────────────────────────────────────────────┘
        │
        ▼
┌──────────────────────────────────┐
│  Admin Review Screen             │
│  - Show suggested matches        │
│  - Allow manual override         │
│  - Approve normalization         │
└──────────────────────────────────┘
```

Deployment Roadmap (Step-by-Step)

Phase 0: Foundation (Weeks 1-2)

```
┌──────────────────────────────────────────────────────┐
│  Infrastructure Setup                                │
├──────────────────────────────────────────────────────┤
│  ✓ Provision AWS/GCP infrastructure                  │
│  ✓ Set up PostgreSQL (RDS or managed)                │
│  ✓ Set up Redis (ElastiCache)                        │
│  ✓ Set up S3 buckets (invoices, images, backups)     │
│  ✓ Install Weaviate (managed or self-hosted)         │
│  ✓ Set up CI/CD pipeline (GitHub Actions)            │
└──────────────────────────────────────────────────────┘
```

```
┌─────────────────────────────────────────────────┐
│  MedusaJS 2.0 Installation                      │
├─────────────────────────────────────────────────┤
│  ✓ Initialize Medusa project                    │
│  ✓ Install B2B plugin                           │
│  ✓ Configure PostgreSQL connection              │
│  ✓ Run migrations                               │
│  ✓ Seed initial data (regions, currencies)      │
│  ✓ Deploy admin dashboard                       │
└─────────────────────────────────────────────────┘
```

Phase 1: SKU Normalization Module (Weeks 3-4)

```
┌─────────────────────────────────────────────────┐
│  Build Normalization Engine                     │
├─────────────────────────────────────────────────┤
│  ✓ Create custom Medusa module: SkuNormalizationService
│  ✓ Integrate Weaviate client                    │
│  ✓ Build CSV upload endpoint                    │
│  ✓ Implement embedding generation (OpenAI)      │
│  ✓ Build similarity search logic                │
│  ✓ Implement pack parser & unit converter       │
│  ✓ Create admin UI for review & approval        │
│  ✓ Test with 1,000 real supplier SKUs           │
└─────────────────────────────────────────────────┘

Acceptance:
- Upload 3 supplier catalogs (300+ SKUs each)
- Achieve 90%+ auto-match accuracy
- Manual review reduces duplicates by 80%
```

Phase 2: POS Integration (Weeks 5-6)

```
┌─────────────────────────────────────────────────┐
│  Foodics Integration (Primary)                  │
├─────────────────────────────────────────────────┤
│  ✓ Register OAuth app with Foodics              │
│  ✓ Build auth flow (3-legged OAuth)             │
│  ✓ Create FoodicsConnector service              │
│  ✓ Implement daily sales sync (cron job)        │
│  ✓ Map Foodics product IDs → Normalized SKUs    │
│  ✓ Calculate theoretical inventory depletion    │
│  ✓ Set up webhook listeners (order.created)     │
│  ✓ Pilot with 1 restaurant (1 week)             │
└─────────────────────────────────────────────────┘

Acceptance:
- Real-time depletion within 5 minutes of sale
- 95%+ SKU mapping accuracy
- Low stock alerts trigger within 1 hour
```

Phase 3: LangGraph AI Agents (Weeks 7-9)

```
┌─────────────────────────────────────────────────────────┐
│  Kitchen Copilot Agent (MVP)                            │
├─────────────────────────────────────────────────────────┤
│  ✓ Install LangGraph + LangChain                        │
│  ✓ Build state graph: PrepPlanAgent                     │
│  ✓ Tools:                                               │
│    - fetch_sales_forecast()                             │
│    - get_current_inventory()                            │
│    - get_recipes()                                      │
│  ✓ Generate daily prep list:                            │
│    "Based on forecast 120 orders, prep:                 │
│     - 18kg chicken breast                               │
│     - 50 burger buns                                    │
│     - 3kg lettuce"                                      │
│  ✓ Deploy as Medusa subscriber                          │
│  ✓ Pilot with 1 chef (2 weeks)                          │
└─────────────────────────────────────────────────────────┘

Acceptance:
- Prep list generated by 6am daily
- Chef feedback: 80%+ accuracy
- Reduces prep waste by 15%
```

Phase 4: AI Suggested Cart (Weeks 10-12)

```
┌─────────────────────────────────────────────────────────┐
│  Autopilot Purchasing Agent                             │
├─────────────────────────────────────────────────────────┤
│  ✓ Build LangGraph workflow: AutoReorderWorkflow        │
│  ✓ Nodes:                                               │
│    - ProcurementAgent                                   │
│    - SkuNormalizationAgent                              │
│    - SourcingAgent                                      │
│    - CartDraftAgent                                     │
│    - HumanApproval (interrupt)                          │
│    - CreatePOAgent                                      │
│  ✓ Trigger: inventory.low_stock event                   │
│  ✓ Build approval UI in restaurant app                  │
│  ✓ Implement explainability (why this qty/supplier)     │
│  ✓ Capture manager edits as feedback                    │
│  ✓ Pilot with 3 branches (4 weeks)                      │
└─────────────────────────────────────────────────────────┘

Acceptance:
- 70%+ of AI carts approved without edits
- 30% reduction in ordering time
- Measurable cost savings (5–10% lower $/kg)
```

Phase 5: Invoice OCR & Matching (Weeks 13-15)

```
┌─────────────────────────────────────────────────────┐
│  OCR Pipeline + Invoice Matching                    │
├─────────────────────────────────────────────────────┤
│  ✓ Set up AWS Textract (or Google Document AI)      │
│  ✓ Build invoice upload endpoint                    │
│  ✓ Implement OCR extraction logic                   │
│  ✓ Build InvoiceValidationAgent (LangGraph)         │
│  ✓ Implement 2-way match (PO vs Invoice)            │
│  ✓ Implement 3-way match (PO vs GRN vs Invoice)     │
│  ✓ Build exception handling UI                      │
│  ✓ Pilot with 100 invoices                          │
└─────────────────────────────────────────────────────┘

Acceptance:
- OCR accuracy > 90% for structured invoices
- 2-way match reduces approval time by 60%
- 3-way match catches 95%+ of discrepancies
```

Phase 6: Scale & Optimize (Weeks 16-20)

```
┌─────────────────────────────────────────────────────┐
│  Production Optimization                            │
├─────────────────────────────────────────────────────┤
│  ✓ Add Oracle Simphony connector                    │
│  ✓ Implement waste & variance tracking              │
│  ✓ Build forecasting models (time series)           │
│  ✓ Add payment/credit integrations                  │
│  ✓ Launch overseas supplier portal                  │
│  ✓ Scale to 50 restaurants                          │
│  ✓ Onboard 200 suppliers                            │
│  ✓ Process 10,000 orders/month                      │
└─────────────────────────────────────────────────────┘
```

Tech Stack Summary

Layer	Technology
Commerce Core	MedusaJS 2.0 (Node.js, TypeScript)
Database	PostgreSQL 15+
Cache/Queue	Redis, BullMQ
Vector DB	Weaviate (or Pinecone)
AI Orchestration	LangGraph + LangChain
LLM	OpenAI GPT-4 (or Claude)

Embeddings	OpenAI Ada-002
OCR	AWS Textract or Google Document AI
POS Connectors	Foodics API, Oracle Simphony STSG2
Frontend	Next.js 14+ (React, TypeScript)
Storage	AWS S3 (or GCP Cloud Storage)
Hosting	AWS ECS/EKS or GCP Cloud Run
CI/CD	GitHub Actions
Monitoring	Datadog or New Relic

Key Benefits of This Architecture

✅ **Modular**: Medusa's plugin system keeps AI logic decoupled from core commerce
✅ **Event-Driven**: Low stock triggers AI without polling
✅ **Stateful AI**: LangGraph maintains context across multi-step workflows
✅ **Human-in-the-Loop**: Manager approval is a first-class citizen
✅ **Explainable**: Every AI decision includes reasoning
✅ **Scalable**: PostgreSQL + Redis + Weaviate handle 1M+ SKUs
✅ **UAE-Optimized**: Multi-currency, Arabic support, local POS integrations
✅ **B2B Native**: Company accounts, RFQs, credit terms out-of-the-box