Meal Planning Application – High■Level Architecture & Phased Build Plan

1. Application Overview

This document describes the architecture, data model, and phased implementation plan for a family meal planning application. The system tracks pantry inventory (ingredient and packaged items), manages a recipe catalog, determines cookability with unit conversion, generates grocery lists, builds meal plans (breakfast, lunch, dinner), and ultimately supports LLM■driven planning via an MCP server.

2. High■Level Architecture

The system is composed of four major layers: • A Spring Boot backend that acts as the system of record and enforces all business rules. • MongoDB for canonical data storage (pantry, recipes, plans, lists, preferences). • A Python AI service for embeddings, vector search, and agent logic. • A React + TypeScript + Tailwind UI for all user interaction. Qdrant is used as the vector database for semantic recipe retrieval.

3. Locked■In Product & Engineering Decisions

- User■entered units are preserved; quantities are compared using basic unit conversion at runtime.
- Pantry inventory supports both ingredient■level and packaged items via a unified 'lot' model.
- Ingredient matching starts as exact string matching with a user■managed mapping dictionary that learns over time.
- Pantry items are never deleted automatically; quantities may reach zero but never go negative.
- Partial cooking is not allowed; a recipe must be fully feasible to cook.
- Recipe servings can be scaled explicitly via user■provided multipliers.
- Meal plans include breakfast, lunch, and dinner.
- LLM interaction is implemented via an MCP server with tool calling.

4. Phase 1 – Core CRUD & Data Foundations

Phase 1 focuses on establishing the system's foundation: CRUD operations, data persistence, recipe seeding, vector indexing, and basic UI screens.

Backend Deliverables: • Pantry lot CRUD (ingredient and packaged items). • Recipe CRUD APIs. • MongoDB repositories and indexes. • Idempotent recipe seeding from recipes_seed.json.

Vector Deliverables: • Python batch job to embed recipes using a local model. • Qdrant collection creation and upsert logic. • Optional back■reference storage of vector IDs in Mongo.

Frontend Deliverables: • Pantry list and editor. • Recipe list, detail, and editor. • Basic routing, API client, and error handling.

5. Phase 2 – Cooking Logic, Grocery Lists, Meal Planning

Phase 2 introduces the core value of the application: determining what can be cooked, updating inventory, and planning meals across multiple days.

Key Backend Components: • UnitConversionService supporting mass, volume, and count units. • Recipe feasibility engine with lot■level allocation. • Strict pantry decrement rules (no negatives, no partials). • Grocery list aggregation across recipes and meal plans. • Meal plan CRUD supporting breakfast, lunch, and dinner.

User Preferences: • Ingredient mapping dictionary (e.g., 'bell pepper' → 'pepper'). • Mappings are applied during feasibility checks and can be managed by the user.

6. Phase 3 – LLM Chatbot, Meal Planner Agent, MCP

Phase 3 introduces intelligent assistance via an LLM integrated using the Model Context Protocol (MCP).

MCP Server Responsibilities: • Expose pantry, recipe, planning, and grocery list tools. • Enforce strict input/output schemas. • Call Spring Boot and Python services on behalf of the LLM.

Agent Capabilities: • Conversational recipe discovery using semantic search. • Multi■step meal planning with constraints and variety. • Pantry■aware feasibility checking. • Proposal■based planning requiring explicit user confirmation before persistence.

7. Conclusion

This phased approach ensures a strong deterministic core before introducing probabilistic AI behavior. By enforcing strict business rules in the backend and exposing capabilities through MCP tools, the system remains reliable, extensible, and safe while enabling advanced agent■driven workflows.