MongoDB Schema & Migration Strategy – Phase 1 (Verbatim)

This document captures the MongoDB schema and migration strategy for Phase 1, designed to minimize schema churn while enabling Phase 2/3 expansion (cooking logic, grocery lists, MCP tools).

2.1 Collections (Phase 1)

Phase 1 collections:
- pantryLots: unified lot model (ingredient + packaged)
- recipes: deterministic recipe storage

Phase 1 intentionally excludes meal plans, grocery lists, and mapping dictionaries (those are Phase 2/3 additions).

2.2 Document Shape Standards

Timestamps
- createdAt (immutable)
- updatedAt (set on every write)

Soft deletion
- Phase 1: not used by default
- Future-compatible field: deletedAt?: ISODate

Schema versioning
- Add schemaVersion: int field to every document
- Phase 1 sets schemaVersion: 1 on new writes

2.3 Schema Definitions (Phase 1)

A) pantryLots

Purpose: represent ingredient-level items and packaged items with a single “lot” abstraction.

Recommended document:
- _id: ObjectId
- schemaVersion: 1
- name: string
- type: INGREDIENT | PACKAGED
- quantity: number >= 0
- unit: string
- metadata: open object
- createdAt / updatedAt

Indexes:
- { nameNormalized: 1 } (recommended)
- { type: 1, nameNormalized: 1 } (recommended)

Uniqueness:
- Phase 1: do not enforce uniqueness on name; allow multiple lots with same name (brand/package differences).

B) recipes

Purpose: deterministic recipe storage, seeded and user-editable.

Recommended document:
- _id: ObjectId
- schemaVersion: 1
- name, ingredients[], instructions[], servings, tags[]
- embedding (optional): qdrantCollection, vectorId, embeddedAt
- createdAt / updatedAt

Indexes:
- { nameNormalized: 1 }
- { tags: 1 }
- optional { externalKey: 1 } unique/sparse for seeded recipes

Seeding idempotency key (recommended):
- externalId / externalKey (best)
- otherwise normalized name as surrogate (acceptable, higher collision risk)

2.4 Validation Approach

Option A (recommended early): application-enforced validation
- Java/Jakarta validation on request DTOs
- Keep Mongo flexible initially

Option B: Mongo collection validators (JSON Schema)
- Enforce invariants even if bypassing API
- Good guardrails once schema stabilizes

A hybrid approach is often best: strict in Spring Boot, Mongo validators for essential fields.

2.5 Migration Strategy

Use forward-only, idempotent migrations with a schemaMigrations collection to record applied changes.

Execution models:
- Startup migrator in Spring Boot (simple)

- Separate migration job (recommended for production discipline)
- Migration frameworks (e.g., Mongock) if desired later

Migration record example:
{ _id: "2026_01_18_add_schemaVersion", appliedAt: ISODate, appVersion: "1.0.0-phase1" }

2.6 Concrete Phase 1 Migrations

Migration 001: create indexes
- pantryLots: { type: 1, nameNormalized: 1 }
- recipes: { nameNormalized: 1 }, { tags: 1 }

Migration 002: backfill schemaVersion + timestamps (if needed)
- Set schemaVersion: 1 where missing
- Set createdAt/updatedAt where missing

Migration 003 (optional): add externalKey for seeded recipes
- Add externalKey to each seeded recipe
- Create unique sparse index on externalKey (seed-only)

Embedding back-reference:
- embedding subdocument is optional and written by the Python indexing job

2.7 Forward Compatibility Hooks

Ingredient identity evolution (Phase 2+)
- Keep ingredients[].name as-is for Phase 1 (exact string matching)
- Reserve optional fields for later: canonicalName?, ingredientId?

Vector lifecycle
- embedding is optional
- allows re-embedding into recipes_v2 without losing provenance

Future collections (Phase 2/3)
- mealPlans, groceryLists, ingredientMappings, userPreferences
- keep separate from pantryLots/recipes to preserve boundaries