FoodGenie 2025: Technical architecture blueprint for a modern recipe platform

Building an intelligent food platform in 2025 demands a sophisticated polyglot persistence architecture combining PostgreSQL, Neo4j, vector databases, and Azure AI services—all orchestrated through modern Laravel patterns. This research reveals **proven design patterns** from companies like Gousto (Neo4j for food ontologies) and Samsung Food (Whisk AI), alongside concrete cost models showing that a production-ready AI-powered recipe platform can run for **$1,600-3,800/month at 10K daily active users**.

The most critical insight: hybrid database architectures work best when each database handles what it does naturally—PostgreSQL for transactional data, Neo4j for ingredient relationship graphs, vector databases for semantic search, and Meilisearch for user-facing typo-tolerant search. Azure's ecosystem, particularly PostgreSQL Flexible Server with pgvector, offers the most cost-effective path for a Laravel application requiring vector search without managing separate infrastructure.

Research Area 1: Polyglot persistence patterns that actually work

The four-database architecture for food platforms

Research identifies four core polyglot persistence patterns, with **service-oriented polyglot persistence** emerging as the optimal approach for FoodGenie. This architecture assigns each data type to its ideal database:

Data Type	Database	Rationale
Core recipes, users, orders	PostgreSQL	ACID compliance, structured data, Laravel Eloquent native
Ingredient relationships, food ontology	Neo4j	Graph relationships, substitution chains, hierarchies
Semantic recipe search	Qdrant or Weaviate	Vector similarity, recipe embeddings
User-facing search	Meilisearch	Sub-50ms typo-tolerant search
Session data, API cache	Redis	Sub-millisecond latency, TTL support

The synchronization challenge between these databases is solved through **Change Data Capture (CDC)** using Debezium connected to Kafka. Neo4j 5+ includes native CDC support via procedures like `db.cdc.current()`, enabling real-time synchronization when recipes are created or modified in PostgreSQL.

Neo4j for food knowledge graphs: The Gousto case study

Gousto, the UK meal kit company, provides the most relevant case study for FoodGenie's Neo4j implementation. Their graph database calculates similarity scores between dishes based on ingredients, cuisine type, dish type, presentation style, and customer use case (convenience vs adventure). The key insight:

recipe and ingredient attributes are **highly interconnected**, making graph traversal 7x faster than relational JOINs for recommendation queries.

The recommended Neo4j data model uses these node types:

- `:Recipe` - Core recipe entity with properties like title, cookTime, difficulty
- `:Ingredient` - Individual ingredients with states (diced, chopped, raw)
- `:Category` - Ingredient hierarchies (dairy → cheese → cheddar)
- `:Cuisine` - Geographic/cultural classifications enabling fusion recipe support
- `:DietType` - Dietary classifications for filtering (vegan, gluten-free, keto)

Relationship types include `[:CONTAINS_INGREDIENT]`, `[:SUBSTITUTES_FOR]`, and `[:PAIRS_WITH]` for flavor compatibility. A critical modeling decision: **model ingredients as nodes, not properties**, which reduces database hits by approximately 7x when querying through ingredient relationships.

Vector database selection: Qdrant edges out competitors

Benchmarks show **Qdrant** delivers the highest requests-per-second with lowest latencies among open-source options, while **Weaviate** offers superior GraphQL APIs and built-in vectorization modules. For FoodGenie's semantic recipe search, Qdrant's payload filtering capability enables combining vector similarity with metadata filters (cuisine, cooking time, dietary restrictions) in single queries.

The cost comparison at **50,000 vectors** (approximately 50,000 recipes):

- Qdrant Cloud: ~$9/month
- Weaviate Cloud: Storage-based pricing
- Pinecone: $20-30/month
- Chroma: Free tier available

For embedding recipes, research recommends **Sentence Transformers** (`all-MiniLM-L6-v2`) for local deployment or OpenAI's `text-embedding-3-small` for managed simplicity. Recipe embeddings should combine title, description, and ingredient list into a single embedding vector, enabling queries like "healthy dessert without sugar" to return semantically relevant results.

Redis caching patterns and the cache invalidation challenge

The **cache-aside pattern** with event-driven invalidation solves FoodGenie's multi-database consistency needs. When a recipe updates in PostgreSQL, Debezium captures the change, publishes to Kafka, and a cache invalidation service:

1. Removes Redis cache keys matching the recipe
2. Triggers Neo4j relationship updates
3. Queues Meilisearch re-indexing

For cache TTLs, research suggests **5-minute TTLs for dynamic lists** (trending recipes, search results) and **1-hour TTLs for individual recipe data** that changes infrequently.

Research Area 2: Azure services optimized for AI-heavy food applications

Azure OpenAI pricing makes GPT-4o-mini the workhorse model

The pricing landscape has shifted dramatically—**GPT-4 costs dropped 83% in 16 months**. Current Azure OpenAI pricing (late 2024):

Model	Input (per 1M tokens)	Output (per 1M tokens)	Best For
GPT-4o	$2.50	$10.00	Complex recipe generation, multi-step reasoning
GPT-4o-mini	$0.15	$0.60	Simple queries, ingredient suggestions
text-embedding-3-small	$0.02	N/A	Recipe embeddings

The **Batch API** offers 50% discounts for jobs completing within 24 hours—ideal for bulk recipe embedding generation during off-peak hours. For FoodGenie's recipe generation at scale, a single generation (~200 input + 400 output tokens) costs approximately **$0.0004 with GPT-4o-mini** versus $0.007 with GPT-4o.

PostgreSQL Flexible Server with pgvector beats Cosmos DB for most scenarios

Azure Database for PostgreSQL Flexible Server with the **pgvector extension** emerges as the recommended primary database. It combines relational ACID compliance with native vector search, eliminating the need for a separate vector database in simpler deployments. The **DiskANN index support** enables high-performance vector search at scale.

Pricing comparison (East US region):

- PostgreSQL Burstable B2s (2 vCores, 4GB): ~$25/month
- PostgreSQL General Purpose D4s_v3 (4 vCores, 16GB): ~$250/month
- Cosmos DB (1,000 RU/s baseline): ~$58/month minimum

Cosmos DB's RU-based pricing creates unpredictable costs for workloads with variable query complexity. PostgreSQL's vCore model provides predictable billing, and Laravel's Eloquent ORM works natively without adaptation.

For graph database needs, **Neo4j Aura on Azure Marketplace** ($65/month professional tier) provides managed graph capabilities rather than fighting Cosmos DB Gremlin API's limited graph algorithm support.

Azure AI Search handles hybrid search natively

Azure AI Search's hybrid search capability combines keyword and vector search in single queries with adjustable semantic ratios. Services created after April 2024 receive **5-6x higher storage capacity** at the same price point.

Recommended tier progression:

- **Basic ($73/month)**: Up to 15GB storage, suitable for <100K recipes
- **S1 Standard ($250/month)**: Up to 160GB, suits 100K-500K recipes
- **L1 Storage Optimized ($1,000/month)**: 1TB for massive recipe databases

For FoodGenie's recipe search requiring typo tolerance and instant results, **Meilisearch remains the better choice** for user-facing search ($30/month cloud or free self-hosted), while Azure AI Search handles

semantic/vector search for recommendation engines.

Laravel hosting: App Service for stability, Container Apps for workers

Azure App Service provides the most stable Laravel hosting with native PHP runtime, built-in deployment slots, and predictable pricing. Container Apps excels for background workers with scale-to-zero capability during idle periods.

Recommended configuration:

- **Primary API**: App Service P1V2 ($85/month) for always-on reliability
- **Background Workers**: Container Apps with scale-to-zero for queue processing
- **Cost savings**: Container Apps' free grant (180,000 vCPU-seconds monthly) covers light workloads

Azure Document Intelligence for receipt scanning

The **Prebuilt Receipt model** ($10/1,000 pages) extracts merchant name, date, line items, quantities, and totals with **90-95% accuracy** on standard receipt formats. For grocery receipts with store-specific abbreviations ("BNLS CKNBRST" = "Boneless Chicken Breast"), a post-processing LLM step using GPT-4o-mini can resolve abbreviations.

Research Area 3: Laravel patterns for multi-database AI applications

Laravel Prism unifies LLM integration across providers

Prism PHP provides a Laravel-first interface supporting OpenAI, Anthropic, Gemini, Ollama, Mistral, and Groq through a unified API. Key capabilities for FoodGenie:

```
// Structured recipe parsing with JSON Schema
$response = Prism::text()
    ->using(Provider::OpenAI, 'gpt-4o-mini')
    ->withPrompt('Parse this recipe: ' . $rawRecipeText)
    ->asStructured([
        'type' => 'object',
        'properties' => [
            'title' => ['type' => 'string'],
            'ingredients' => ['type' => 'array'],
            'steps' => ['type' => 'array'],
            'cooking_time' => ['type' => 'integer'],
        ]
    ]);
```

For tool/function calling, Prism enables the LLM to call nutrition lookup services, ingredient substitution databases, or external APIs during generation—critical for accurate nutritional information in generated recipes.

Neo4j integration via laudis/neo4j-php-client

The **official Neo4j PHP driver** (`laudis/neo4j-php-client`) provides Laravel 11/12 compatibility with transaction support. The **NeoEloquent** package (`ulobby/neoeloquent`) offers Eloquent-like syntax but has limited Laravel 12 compatibility.

Register the Neo4j client as a singleton in `AppServiceProvider`:

```
$this->app->singleton(Neo4jClientInterface::class, function ($app) {
    return ClientBuilder::create()
        ->withDriver('bolt', 'bolt://'.config('database.neo4j.host'))
        ->build();
});
```

For FoodGenie's ingredient relationship queries, write Cypher directly through the client for complex graph traversals while using PostgreSQL/Eloquent for standard CRUD operations.

Laravel Scout with Meilisearch enables instant recipe search

Configuration for recipe-specific faceted search:

```
// config/scout.php
'index-settings' => [
    Recipe::class => [
        'filterableAttributes' => ['cuisine', 'difficulty', 'cooking_time',
'dietary_tags'],
        'sortableAttributes' => ['created_at', 'rating', 'cooking_time'],
        'searchableAttributes' => ['title', 'description', 'ingredients'],
    ],
],
```

Meilisearch's built-in typo tolerance handles common misspellings ("chiken" → "chicken", "spageti" → "spaghetti") without configuration—essential for recipe search user experience.

Laravel Horizon configuration for AI workloads

AI tasks require extended timeouts and dedicated worker pools:

```
// config/horizon.php
'supervisor-ai' => [
    'queue' => ['ai-generation', 'ai-analysis'],
    'balance' => 'auto', // Dynamic scaling based on queue length
    'maxProcesses' => 10,
    'memory' => 256, // Higher for AI tasks
    'timeout' => 300, // 5 minutes for LLM calls
    'tries' => 3,
    'backoff' => [30, 60, 120], // Exponential backoff for rate limits
],
```

For long-running AI tasks, implement **job batching** to process recipe description generation, nutrition analysis, and tag generation in parallel with combined success/failure callbacks.

Inertia.js with Vue 3 for seamless SPA experience

Laravel Breeze with Inertia.js provides server-side routing with SPA-like interactivity. For recipe pages:

```
// Controller
return Inertia::render('Recipes/Show', [
    'recipe' => RecipeResource::make($recipe->load(['ingredients', 'steps'])),
    'relatedRecipes' => RecipeResource::collection($recipe->related()),
]);
```

Use **Pinia** for client-side state management (favorites, recent searches) and **partial reloads** (`router.reload({ only: ['recipes'] })`) for performance optimization when only specific data changes.

Research Area 4: Building a comprehensive food taxonomy

FoodOn ontology provides semantic foundation

FoodOn (Open Food Ontology) offers over 9,600 generic food product categories based on the FDA's LanguaL 14-facet indexing system. It covers plant/animal sources, preservation methods, cooking processes, and packaging—providing the semantic backbone for ingredient classification.

Integration approach for FoodGenie:

1. Import FoodOn identifiers (FOODON_XXXXXXXX format) as canonical ingredient references
2. Build synonym tables mapping user input to FoodOn IDs
3. Use FoodOn's hierarchy for automatic allergen inference (if ingredient is dairy → may contain milk allergen)

The **FoodKG** knowledge graph built on FoodOn demonstrates practical implementation, linking ingredients, recipes, and nutrition data with SPARQL query support.

Open Food Facts API for barcode scanning

The free Open Food Facts database contains **1.7M+ products** from 150 countries with barcode lookup, Nutri-Score calculation, and AI-powered data extraction from product photos. Key endpoint:

```
GET /api/v2/product/{barcode}
```

Returns structured data including ingredients, allergens, and per-100g nutritional values. Coverage is strongest for European products; supplement with USDA FoodData Central for US market depth.

Ingredient parsing with NER achieves 92%+ accuracy

Named Entity Recognition extracts structured data from freeform ingredient strings. The **spaCy-transformer** approach achieves 92%+ F1 scores on ingredient parsing, extracting:

Entity Type	Example
QUANTITY	"2", "1/2", "3-4"
UNIT	"cups", "tablespoons", "oz"
NAME	"flour", "chicken breast"
STATE	"diced", "chopped", "melted"

The Python `ingredient-parser` library provides a ready-made solution. For Laravel integration, either run as a Python microservice or use LLM-based parsing as fallback.

Allergen systems must support both US Big 9 and EU 14

The US "Big 9" allergens (milk, eggs, fish, shellfish, tree nuts, peanuts, wheat, soy, sesame) differ from EU's 14 allergens which add gluten-containing cereals, celery, mustard, sulphites, lupin, and molluscs. FoodGenie must support both regulatory frameworks for global market viability.

SuperCook's inventory-to-recipe matching algorithm

The leading "what can I cook" algorithm:

1. User maintains ingredient inventory (voice input supported)
2. System assumes salt, pepper, water as defaults
3. Scans recipe database for exact matches
4. Shows "one ingredient away" recipes with missing items highlighted
5. Filters by cuisine, dietary restrictions, equipment

Spoonacular's `findByIngredients` endpoint provides API access to similar functionality for prototyping before building custom matching.

Research Area 5: AI architecture decisions with concrete cost models

RAG architecture: RecipeRAG sets the benchmark

The **RecipeRAG** paper (2024) establishes state-of-the-art for recipe retrieval-augmented generation using:

- **Stochastic Diversified Retrieval Augmentation (SDRA)**: Retrieves semantically related recipes from a vector store
- **Self-consistency Ensemble Voting**: Different retrieved recipes during inference ensure agreement
- Achieved best results on Recipe1M dataset for both generation and ingredient recognition

For FoodGenie, recipes average **256-512 tokens**—chunk by logical sections (ingredients, instructions, nutritional info) while maintaining recipe context within chunks.

Recipe parsing: Rule-based with AI fallback

The Python **recipe-scrapers** library supports **592 cooking websites** with Schema.org/Recipe parsing. The recommended hybrid approach:

1. Try rule-based parsing first (JSON-LD, Microdata, RDFa)
2. Fall back to AI extraction (GPT-4o-mini with structured output) for unsupported sites
3. Human review queue for low-confidence extractions

This approach handles ~75% of recipes automatically via JSON-LD, with AI processing the remainder.

Receipt OCR: Veryfi or Taggun for production

Service	Key Features	Accuracy
Veryfi	Multi-modal AI, fraud detection, 91 currencies	Enterprise-grade
Taggun	Line item extraction, 50+ countries, real-time	~98% core accuracy
Azure Document Intelligence	Azure-native, prebuilt receipt model	90-95% standard formats

For grocery receipts with store-specific abbreviations, add a post-processing step using GPT-4o-mini to resolve abbreviations and normalize product names to FoodOn identifiers.

SLMs enable cost-effective on-device deployment

Phi-3-mini (3.8B parameters) achieves strong reasoning performance while running at **12+ tokens/second on iPhone 14** using 1.8GB memory when quantized. For FoodGenie, SLMs handle simple ingredient lookups and basic queries, reserving GPT-4o for complex recipe generation.

Tiered model selection:

- **Simple queries (80%)**: GPT-4o-mini at $0.15/1M input tokens
- **Complex generation (20%)**: GPT-4o at $2.50/1M input tokens
- **Edge deployment**: Phi-3-mini for offline ingredient recognition

Production cost model for 10K daily active users

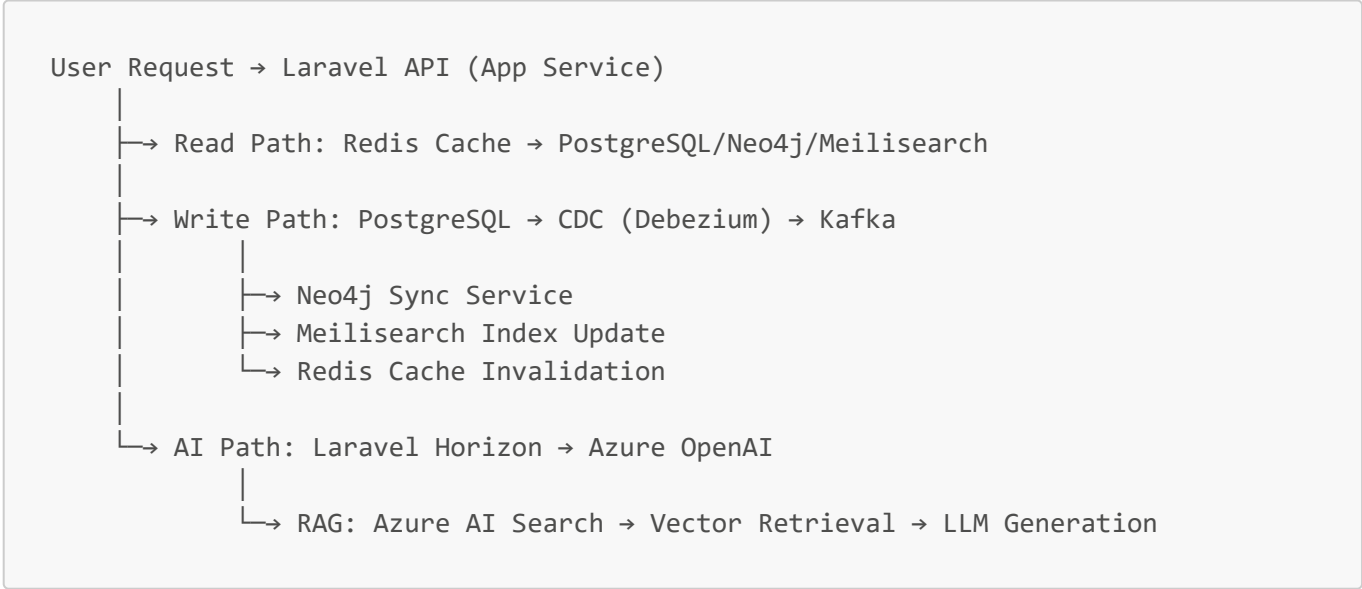
Component	Monthly Cost
LLM API (tiered 80/20 mini/4o)	$500-1,500
Receipt OCR (50K scans)	$500-1,000
Vector database hosting	$100-300
Azure PostgreSQL D4s_v3	$250
Azure App Service P1V2	$85
Redis Standard C2	$100
Meilisearch Cloud	$30
Azure AI Search Basic	$73
Total	**$1,638-3,338**

Recommended architecture synthesis

Primary technology stack

Layer	Technology	Rationale
Framework	Laravel 11/12	Mature ecosystem, Eloquent ORM, Horizon for queues
Primary Database	Azure PostgreSQL Flexible Server + pgvector	ACID compliance, native vector search, cost-effective
Graph Database	Neo4j Aura on Azure	Ingredient relationships, substitution graphs
Search	Meilisearch (user-facing) + Azure AI Search (semantic)	Typo tolerance + vector hybrid search
Cache	Azure Cache for Redis Standard	Sessions, API caching, rate limiting
LLM Integration	Laravel Prism + Azure OpenAI	Unified interface, provider flexibility
Frontend	Vue 3 + Inertia.js	SPA experience without separate API
Background Jobs	Laravel Horizon + Container Apps	AI task monitoring, scale-to-zero workers

Data flow architecture



Critical success factors

1. **Start with PostgreSQL + pgvector** before adding separate vector databases—it handles most semantic search needs
2. **Implement CDC from day one**—retrofitting synchronization across databases is extremely difficult
3. **Cache aggressively**—AI responses, search results, and nutrition lookups should all hit Redis first

4. **Use tiered LLM selection**—route 80% of queries to GPT-4o-mini, reserving GPT-4o for complex generation
5. **Monitor costs weekly**—AI API spending can spike unexpectedly; implement hard spending caps

Key risks and mitigations

Risk	Mitigation
Recipe hallucinations	RAG with verified sources, confidence scoring, human review queue
Cross-database consistency	CDC with Kafka, SAGA pattern for critical transactions
AI cost overruns	Tiered models, response caching, batch API for non-real-time tasks
Cold starts on Container Apps	Minimum replica = 1 for API, scale-to-zero only for workers
Neo4j query performance	Proper indexing on :Ingredient(name), :Recipe(id); model ingredients as nodes

This architecture provides a **production-ready foundation** scaling from startup (10K users, ~$2,000/month) to enterprise (1M+ users) while maintaining flexibility to adopt emerging AI capabilities as they mature.