APPROACH SUMMARY – Recipe Aggregation and Menu Creation

1. STRATEGY & ARCHITECTURE

Problem: Given recipes and N guests, return cost-minimized shopping list + menu card. Diet Problem + Multiple-Choice Knapsack: discrete servings, discrete SKUs (1 for $2, 5 for $5), recipe synergy (shared ingredients reduce waste).

Architecture: FastAPI monolith, Postgres (Recipe, Ingredient, RecipeIngredient, SKU, MenuPlan, LLMCallLog), Redis (Celery). Frontend: React+Vite, Tailwind. DSPy for LLM. Sync upload (recipes.py); async (progress.py) w/ SSE for large batches. Celery workers async SKU fetch; queue decouples parse from price lookup.

Schema: Ingredient: canonical_name, base_unit (g/ml/count/tbsp/tsp), base_unit_qty. RecipeIngredient: quantity, unit. SKU: price, size, quantity_in_base_unit, expires_at. Demand and supply share base_unit for dimensional consistency.

ILP: x_r (recipe batches), y_sk (SKU count). Minimize Σ cost·y + ε·Σ x (batch penalty). Constraints: servings ≥ target; meal-type (appetizer:1, entree:2/person); demand ≤ supply/ingredient. Filters (meal_config, store_slugs, allergens, include_every_recipe, required_recipes) configure constraints. PuLP/CBC, time_limit 10s. 100 recipes (~1000 ing, ~5 SKUs/ing) → ~5600 vars, <1s solve.

2. AI PARSING PIPELINE

Upload → parse (structured: name, servings, instructions; unstructured: ingredients). Per ingredient: (1) Match (LLM): existing|new|similar + canonical_name; similar → keep_specific|generalize|substitute. (2) Unit normalize (LLM): base_unit, normalized_qty via ontology (tbsp=15ml, lb=453.59g, etc.); canonical drives base (oil→ml, butter→g, garlic→count). (3) Get-or-create Ingredient, RecipeIngredient. (4) Creation event → enqueue fetch_skus (Celery). Captions/descriptions only after plan (Generate Final Materials) to avoid excess LLM calls.

3. PROMPTING EXPERIMENTATION

Strategies: Short direct prompts; tried few-shot, multi-step, CoT for unit/SKU. CoT better accuracy, more latency/cost; reverted to single-shot. Tradeoff: accuracy vs speed. Converged gpt-4o-mini everywhere—good enough, fast, cheap. Tried gpt-4o for SKU math; rate limits w/ parallel workers; reverted. Versioning: prompt_version in run_with_logging, v1–v15; cache keys include version. No DSPy optimizers; manual iteration. Ontology: UNIT_CONVERSION_ONTOLOGY; refined canonical→base (oil→ml NEVER count, cheese→g) to fix bugs. Output parsing: extractors for LLM prompt-leakage ("Base Unit: g").

4. SCRAPER & SKU FETCHING

Started Parse.bot API; cost → built custom scraper. Instacart: GraphQL, Apollo persisted (SearchCrossRetailerGroupResults, Items batch, Autosuggestions). slugs/IDs: retailer_id, shop_id, zone_id, postal_code. Cross-retailer → signpostRetailerShopIds; broad shopIds, best shop, resolve itemIds via Items batch. Region: postal_code sets zone; prices vary. Cookie: Playwright /store?address=X, /store/s?k=bananas; cache 30min/postal. Redis lock: one worker fetches cookies (avoids parallel timeouts). SKU fetch = Celery on ingredient creation. TTL 24h; Beat every 30min refreshes no-SKU ingredients. 403 → retry. Future: session plane, token issuer, multi-session cycling.

5. HYBRID INGREDIENT RETRIEVAL

≤20 existing: full list to LLM. >20: embedding retrieval, top-k=10 to LLM. sentence-transformers (all-MiniLM-L6-v2) or TF-IDF. Lightweight; single-word ingredients; scales; no full-corpus embeddings.

6. FRONTEND & OBSERVABILITY

Frontend: Drag-drop, progress/file, filters (meal, stores, allergens), plan + consolidated list + menu card. Generate Final Materials → DSPy descriptions + dish-color (food-vibe bg). PDF via html2canvas+jsPDF. Logging: hierarchical (recipe.upload, sku.fetch, llm.call), [TIMING] latency_ms. DSPy trace, llmcallog. Tests: pytest (API, unit norm, match, retrieval). Expand for CI/CD.

7. FUTURE WORK

Session/token issuer plane; use-up-existing (SQL join pantry, ILP constraint); VM deploy (more workers); multi-user graph DB (Neo4j), ingredient index, recipe recs; DSPy Compile/MIPRO; CI/CD + regression tests.