Blueprint Architectural pour un Écosystème Culinaire Intelligent : Reconstruction d'un Schéma Relationnel

Partie I : Le Lexique Alimentaire Universel : Modélisation des Données Fondamentales

La pierre angulaire de tout système culinaire intelligent réside dans la modélisation de ses entités fondamentales. Le succès de l'application — de la correspondance des recettes à la gestion des stocks — dépend de la robustesse et de la granularité de ce modèle de données de base. Cette section analyse l'architecture d'une hiérarchie alimentaire à quatre niveaux, une structure conçue pour passer de manière transparente des concepts alimentaires abstraits aux produits tangibles et commercialisables. Cette approche est la mise en œuvre technique directe du concept de « Garde-Manger Global » envisagé comme une source unique de vérité pour tous les produits alimentaires.[1]

1.1 La Hiérarchie Alimentaire à Quatre Niveaux : du Concept au Produit

La stratégie centrale du schéma est sa capacité à modéliser les aliments à quatre niveaux d'abstraction distincts, comme en témoigne la structure des tables fournies.[1] Cette hiérarchie est essentielle pour permettre au système de répondre à des requêtes complexes avec précision et flexibilité.

canonical_foods

La table canonical_foods représente la racine abstraite de la hiérarchie. Chaque

enregistrement est un concept alimentaire fondamental, dépouillé de toute variation ou transformation (par exemple, « Pomme », « Tomate », « Blanc de poulet »).

- **Structure et Rôle** : Les colonnes clés incluent id (clé primaire), canonical_name (le nom normalisé), category_id et subcategory_id (liens vers les taxonomies de classification), ainsi que des propriétés physiques de base comme unit_weight_grams et density_g_per_ml.[1] Cette table sert de point d'ancrage ultime pour les données nutritionnelles génériques (nutrition_id) et la catégorisation des ingrédients, garantissant qu'une recherche pour « oignon » résout en une seule entité canonique, évitant ainsi la redondance des données.[1]

cultivars

Le deuxième niveau, la table cultivars, introduit la notion de variété biologique. Elle permet de distinguer des variations spécifiques au sein d'un aliment canonique (par exemple, « Granny Smith » ou « Golden Delicious » pour la pomme ; « San Marzano » pour la tomate).

- **Structure et Rôle** : La table est directement liée à canonical_foods via une clé étrangère canonical_food_id. Elle contient des champs tels que cultivar_name et synonyms.[1] Ce niveau de granularité permet de stocker des informations nuancées telles que des profils de saveur spécifiques, des saisonnalités précises ou des origines géographiques, enrichissant ainsi potentiellement la logique de recommandation future pour suggérer, par exemple, le meilleur cultivar de pomme pour une tarte.

archetypes

La table archetypes est sans doute l'entité la plus critique de la hiérarchie. Un archétype représente une forme spécifique ou un état de transformation d'un aliment (par exemple, « Compote de pommes », « Tomates concassées en conserve », « Blanc de poulet fumé »). C'est le principal pont entre les besoins d'une recette et l'inventaire d'un utilisateur.

- **Structure et Rôle** : Sa structure est conçue pour être flexible, avec des clés étrangères optionnelles vers canonical_food_id et cultivar_id. Un archétype peut être dérivé soit d'un aliment générique (ex: "Farine de blé"), soit d'un cultivar spécifique (ex: "Farine de blé Durum"). Elle possède ses propres attributs déterminants tels que process (ex: "en conserve", "séché", "fumé"), shelf_life_days (durée de conservation avant ouverture), et open_shelf_life_days (durée de conservation après ouverture).[1] C'est à ce niveau que les recettes expriment leurs besoins, en demandant un « archétype » plutôt qu'un produit de

marque spécifique.

products

Le dernier niveau de la hiérarchie est la table products. Elle représente une unité de gestion de stock (SKU) spécifique, achetable, avec une marque et une taille d'emballage (par exemple, « Compote de Pommes Sans Sucre Ajouté Andros, Pot de 720g »). C'est à ce niveau que la plupart des articles de l'inventaire physique de l'utilisateur seront suivis.

- **Structure et Rôle** : La table products est liée à archetypes via la clé étrangère archetype_id. Ses colonnes incluent name, brand, ean (code-barres), package_size, et package_unit.[1] Cette table complète la hiérarchie en connectant le monde abstrait de la planification culinaire au monde concret des achats et de la gestion des stocks.

Le tableau suivant illustre le flux de données à travers cette hiérarchie avec deux exemples concrets, démontrant comment le système passe du général au spécifique.

Niveau de Hiérarchie	Table	Exemple 1 : Tomate	Exemple 2 : Blé
1. Concept	**canonical_foods**	id: 101, canonical_name: 'Tomate'	id: 205, canonical_name: 'Blé'
2. Variété	**cultivars**	id: 55, canonical_food_id: 101, cultivar_name: 'San Marzano'	id: 82, canonical_food_id: 205, cultivar_name: 'Blé Dur'
3. Forme/Processus	**archetypes**	id: 742, canonical_food_id: 101, name: 'Tomates pelées en conserve'	id: 910, cultivar_id: 82, name: 'Semoule de blé dur'
4. Produit SKU	**products**	id: uuid-1, archetype_id: 742, brand: 'Mutti', name: 'Polpa',	id: uuid-2, archetype_id: 910, brand: 'Barilla', name: 'Semola

		package_size: 400, package_unit: 'g'	Rimacinata', package_size: 1, package_unit: 'kg'

1.2 Implications Stratégiques de la Hiérarchie Alimentaire

Cette structure à quatre niveaux n'est pas une simple complexité technique ; elle est le fondement de plusieurs des fonctionnalités les plus puissantes du système.

L'une des décisions de conception les plus importantes est le découplage intentionnel des recettes et des marques. Le schéma de la table recipe_ingredients montre des clés étrangères vers archetype_id et canonical_food_id, mais pas vers products.[1] Ce choix architectural est une mise en œuvre directe de la vision d'un modèle de recette universel et non commercial.[1] En faisant en sorte qu'une recette demande des « tomates concassées en conserve » (un archétype) plutôt que des « Tomates Concassées Marque X » (un produit), le système reste agnostique aux marques. Cela a une conséquence directe et puissante sur la fonctionnalité anti-gaspillage : l'algorithme de « Filtrage par Disponibilité » peut faire correspondre une exigence de recette à *n'importe quelle* marque de cet archétype que l'utilisateur possède dans son inventaire, maximisant ainsi les chances d'utiliser les produits disponibles avant leur péremption.[1] La structure de la base de données impose donc un principe philosophique fondamental de l'application.

De plus, la hiérarchie permet de définir les données nutritionnelles et physiques au niveau le plus approprié, avec la possibilité d'hériter ou de surcharger ces données en descendant dans la hiérarchie. Le schéma montre que canonical_foods, cultivars, et archetypes peuvent tous avoir des colonnes pour density_g_per_ml et grams_per_unit.[1] Un aliment canonique (« Tomate ») a une densité de base. Un cultivar spécifique (« Tomate Roma ») peut avoir une densité légèrement différente. Un archétype résultant (« Concentré de tomate ») aura une densité radicalement différente. Lorsqu'une conversion d'unité est nécessaire (par exemple, de "1 tasse" à "grammes"), l'application peut rechercher la valeur la plus spécifique disponible, en commençant par le niveau archetype et en remontant la hiérarchie si nécessaire. Cela permet des conversions de quantité beaucoup plus précises qu'un modèle d'ingrédient à un seul niveau, ce qui est essentiel pour la précision de l'« Algorithme de Calcul Nutritionnel des Recettes ».[1]

1.3 Taxonomies et Métadonnées de Support

Pour rendre ce lexique alimentaire consultable et organisable, plusieurs tables de support fournissent un échafaudage de classification et de contexte.

- **reference_categories & reference_subcategories** : Ces tables forment le système de classification principal (par exemple, Produits laitiers -> Fromage -> Fromage à pâte dure).[1] Cette structure soutient directement les fonctionnalités de filtrage et d'organisation mentionnées tout au long de la conception du système.[1]
- **processes & canonical_food_processes** : Cet ensemble de tables modélise les transformations alimentaires (par exemple, Fermentation, Fumage, Marinade). Il s'agit d'une fonctionnalité sophistiquée qui permet des requêtes avancées comme « trouver tous les aliments fermentés » et fournit des données structurées pour la colonne process de la table archetypes.[1]
- **countries & canonical_food_origins** : Ces tables permettent de suivre la provenance des aliments, une fonctionnalité qui peut prendre en charge les préférences futures des utilisateurs pour l'approvisionnement local ou des produits régionaux spécifiques.[1]

Partie II : La Recette Déconstruite : Un Cadre Modulaire pour la Logique Culinaire

Cette section détaille comment les recettes sont transformées de blocs de texte monolithiques en entités dynamiques, interrogeables et "conscientes" sur le plan informatique. Cette déconstruction est la condition préalable à toutes les fonctionnalités intelligentes, du calcul nutritionnel aux recommandations basées sur l'inventaire, comme l'exige la vision du système.[1]

2.1 L'Entité recipes de Base

La table recipes est le cœur du modèle de recette. Elle contient les métadonnées primaires qui permettent le filtrage, l'affichage et l'interaction de base.[1]

- **Champs de Métadonnées** : La table contient des champs essentiels tels que name, description, prep_time_minutes, cook_time_minutes et servings.
- **Champs Taxonomiques** : Des champs comme cooking_method (ex: "Rôtissage", "Friture") et role (un type défini par l'utilisateur, par exemple "Plat principal",

"Accompagnement", "Dessert") mettent directement en œuvre le système de classification multi-facettes décrit comme une exigence fondamentale pour un filtrage sophistiqué.[1] Ces champs permettent de classer une recette selon plusieurs dimensions simultanément.

2.2 recipe_ingredients : Le Lien Crucial

La table de jonction recipe_ingredients est le composant le plus critique du modèle, transformant une simple liste d'ingrédients en une structure de données puissante.[1]

- **Structure de Clé Étrangère Flexible** : La table contient des clés étrangères archetype_id et canonical_food_id, toutes deux NULLABLE.[1] Cette décision de conception est un élément clé pour la flexibilité du système.
- **Attributs de Relation** : Chaque ligne contient des attributs spécifiques à la relation entre une recette et un ingrédient, notamment quantity et unit, qui sont les entrées directes pour l'algorithme de calcul nutritionnel et le système de conversion d'unités. La colonne notes (ex: "haché finement") fournit un contexte essentiel pour l'utilisateur, comme le préconise le modèle de données universel.[1]

2.3 Flexibilité, Précision et Compromis du Système

Le design de la table recipe_ingredients révèle des choix architecturaux importants avec des implications profondes pour le système.

La décision de permettre à un ingrédient de recette de se lier soit à un canonical_food, soit à un archetype est une fonctionnalité puissante qui équilibre la facilité de saisie et la précision. Un utilisateur entrant une simple recette familiale pourrait simplement spécifier « Oignon » (un canonical_food). Une recette plus détaillée provenant d'une source culinaire professionnelle spécifierait « Oignon jaune émincé » (un archetype). Le schéma prend en charge les deux scénarios [1], ce qui améliore l'expérience utilisateur et l'adaptabilité du système. Cependant, cela crée un défi pour l'intégrité des données : quel lien doit être priorisé? La logique applicative doit être conçue pour donner la priorité à archetype_id lorsqu'il est présent, car il contient des informations plus spécifiques sur la forme et le processus, qui sont essentielles pour une cuisson précise et une correspondance d'inventaire fiable. Le canonical_food_id devrait servir de solution de repli.

Cependant, une lacune importante existe dans le schéma actuel : l'absence de support

explicite pour les sous-recettes (par exemple, une recette de « Pâte brisée » utilisée comme ingrédient dans une recette de « Tarte aux pommes »). La vision d'un écosystème culinaire complet et modulaire implique que cette capacité est nécessaire.[1] Un système véritablement modulaire devrait permettre aux recettes d'être des ingrédients dans d'autres recettes. Actuellement, la table recipe_ingredients ne se lie qu'à des articles alimentaires.[1] Pour combler cette lacune, il est fortement recommandé d'ajouter une clé étrangère sub_recipe_id (nullable) à la table recipe_ingredients. Cela créerait une relation récursive, permettant la composition de plats complexes à partir de composants plus simples et réutilisables, augmentant considérablement la modularité et la puissance du système.

2.4 Instructions Structurées, Accords et Classification

Le reste du modèle de recette fournit la structure pour le contexte et l'organisation.

- **instructions** : Cette table modélise correctement la préparation comme une liste ordonnée d'étapes (step_number, description).[1] Cette approche granulaire est une mise en œuvre directe du modèle d'instructions déconstruites, permettant une présentation étape par étape à l'utilisateur et ouvrant la voie à de futures intégrations avec des appareils de cuisine intelligents.[1]
- **tags & recipe_tags** : Cette relation plusieurs-à-plusieurs permet un étiquetage subjectif et fonctionnel (ex: "Rapide", "Festif", "Adapté aux enfants"), répondant au besoin d'une « taxonomie multi-facettes » pour capturer le contexte non culinaire d'une recette.[1]
- **recipe_pairings** : Cette table de jonction modélise la composition de repas complets en liant un main_recipe_id à un side_recipe_id.[1] Elle fournit la base structurelle pour l'« Association Algorithmique des Saveurs », permettant au système de recommander des combinaisons de plats harmonieuses et de composer des menus cohérents.[1]

Partie III : Le Garde-Manger Dynamique : Inventaire, Logistique et Réduction du Gaspillage

Cette section se concentre sur les composants du schéma qui font le pont entre le modèle numérique et la cuisine physique de l'utilisateur. C'est la mise en œuvre de la philosophie de la « Cuisine Zéro Déchet », un pilier central de la proposition de valeur du système.[1]

3.1 Modélisation des inventory_lots

La table inventory_lots est au cœur de la gestion des stocks. Il est crucial de comprendre que cette table ne représente pas un *type* d'aliment, mais un *lot* physique spécifique d'un aliment appartenant à l'utilisateur.

- **Attributs Fondamentaux** : La table contient des colonnes essentielles pour le suivi logistique : initial_qty, qty_remaining, unit, acquired_on, et la colonne critique expiration_date.[1] Ces champs sont les entrées directes pour le calcul du « Score d'Urgence » dans l'algorithme anti-gaspillage, qui priorise les recettes utilisant des ingrédients proches de leur date de péremption.[1]
- **Organisation Physique** : Les champs storage_method (avec des valeurs comme 'pantry', 'fridge', 'freezer') et storage_place permettent à l'utilisateur de cartographier son espace de stockage physique, aidant à localiser les articles.[1]

3.2 Le Défi de l'Inventaire Polymorphe

L'analyse de la table inventory_lots révèle une décision de conception problématique qui doit être corrigée pour garantir l'intégrité et la stabilité du système.

Le schéma fourni pour inventory_lots contient les colonnes product_type (varchar) et product_id (bigint).[1] Il s'agit d'une implémentation classique d'une association polymorphe, destinée à permettre à un lot d'inventaire de se rapporter à différents types d'entités parentes (par exemple, products ou archetypes). Bien que flexible en théorie, cette approche présente de graves inconvénients en pratique. Premièrement, elle brise l'intégrité référentielle au niveau de la base de données, car une contrainte de clé étrangère standard ne peut pas être appliquée à une colonne qui peut faire référence à plusieurs tables. Cela augmente considérablement le risque de données orphelines et de corruption. Deuxièmement, une incohérence fatale existe dans les types de données : inventory_lots.product_id est défini comme bigint, tandis que la clé primaire de la table products est uuid.[1] Cette incompatibilité rend la relation non fonctionnelle.

Il est impératif de rejeter ce modèle polymorphe. La solution la plus robuste et la plus simple est d'appliquer une contrainte métier : tout ce qui entre dans l'inventaire doit être un product. Si un utilisateur ajoute des "carottes en vrac" sans marque, le système doit créer en arrière-plan une entrée de produit générique "Carottes en vrac" pour cet utilisateur. Cela simplifie énormément le modèle. Par conséquent, la structure de inventory_lots doit être modifiée : la colonne product_id doit être changée en type uuid et devenir une clé étrangère directe et non nullable vers products.id. Les colonnes product_type doivent être supprimées.

Cette modification garantit l'intégrité référentielle, simplifie les requêtes et élimine l'ambiguïté du modèle.

3.3 Le Système Universel de Conversion d'Unités

Pour que le système puisse raisonner sur les quantités (par exemple, comparer "1 tasse de farine" dans une recette avec "500g de farine" dans l'inventaire), un système de conversion d'unités robuste est essentiel. Le schéma aborde ce problème avec une approche à deux niveaux.[1]

- **unit_conversions_generic** : Cette table stocke les facteurs de conversion universels et mathématiquement constants (par exemple, 1 tasse = 236.588 ml, 1 kg = 1000 g).
- **unit_conversions_product** : Cette table gère les conversions non standard et dépendantes de l'ingrédient, en particulier les conversions volume-masse (par exemple, pour un produit de farine spécifique, 1 tasse = 120g ; pour du sucre, 1 tasse = 200g). Cette précision est indispensable pour l'algorithme de calcul nutritionnel, qui doit convertir toutes les quantités d'ingrédients en une unité standard (grammes) pour une analyse précise.[1]

3.4 Table locations

La table locations offre une couche supplémentaire de personnalisation pour l'organisation de l'utilisateur. Elle permet aux utilisateurs de définir des emplacements de stockage personnalisés (par exemple, "Étagère du haut du garde-manger", "Congélateur du sous-sol") qui peuvent être liés aux lots d'inventaire, allant au-delà des catégories par défaut de storage_method.[1]

Partie IV : Le Carburant du Moteur : Données Nutritionnelles et Personnalisation de l'Utilisateur

Cette section détaille les modèles de données qui alimentent l'« intelligence » du système : sa capacité à calculer la nutrition avec précision et à apprendre et s'adapter à l'identité culinaire

unique de l'utilisateur, conformément à la vision exposée dans le cadre conceptuel.[1]

4.1 Le Registre Nutritionnel Centralisé

La précision nutritionnelle est une exigence fondamentale.[1] Le schéma met en œuvre cela grâce à un modèle de données normalisé et centralisé.

- **nutritional_data** : Cette table stocke des informations détaillées sur les macro et micronutriments, généralement pour une portion de 100g.[1] En la concevant comme une table centrale liée depuis canonical_foods (via nutrition_id), le schéma évite la duplication des données et garantit une source unique de vérité pour les informations nutritionnelles, comme le préconise le concept de "Registre Nutritionnel".[1]
- **cooking_nutrition_factors** : Cette table représente une fonctionnalité très avancée. Elle est conçue pour stocker des facteurs de correction qui modélisent l'impact des processus de cuisson sur la teneur en nutriments.[1] C'est le mécanisme de mise en œuvre des « Facteurs de Rétention (FRT) », permettant au système d'ajuster les valeurs nutritionnelles pour tenir compte, par exemple, de la dégradation de la vitamine C lors de l'ébullition. Cela élève le calcul nutritionnel d'une simple somme d'ingrédients bruts à une estimation beaucoup plus scientifiquement précise.[1]

4.2 Modélisation de l'Identité Culinaire de l'Utilisateur

La personnalisation est la clé de l'engagement à long terme de l'utilisateur. Le système doit construire un modèle vivant de l'identité culinaire de chaque utilisateur.[1]

- **users** : Le schéma fourni pour la table users présente de graves problèmes d'intégrité, avec de multiples définitions de clés primaires conflictuelles, le rendant inutilisable.[1] Il doit être remplacé par un schéma propre et standard, probablement basé sur la structure auth.users de Supabase dont il semble être une version corrompue. Un schéma standard fournirait une base sécurisée et fiable pour l'authentification et l'identification des utilisateurs.
- **user_recipe_interactions** : Cette table est le flux de données brutes qui alimente l'apprentissage du système. Elle enregistre chaque action significative de l'utilisateur : rating, is_favorite, cook_count, last_cooked_date.[1] Ces données sont essentielles pour construire le « profil de goût dynamique » décrit dans le cadre conceptuel.[1] Elles alimentent directement les algorithmes de filtrage collaboratif (trouver des utilisateurs

similaires) et d'exploitation (recommander des plats favoris éprouvés).

4.3 Le Profil Utilisateur Manquant : Une Lacune Critique du Schéma

L'analyse comparative du schéma technique [1] et du document de vision [1] révèle une lacune critique : l'absence totale de tables pour stocker les préférences explicites et les contraintes de l'utilisateur. Le document de vision décrit en détail le « Profil Utilisateur Dynamique », qui inclut des préférences statiques telles que les allergies, les régimes alimentaires (« Végétarien », « Sans Gluten »), les objectifs de santé (perte de poids, prise de muscle) et les ingrédients détestés. La table user_recipe_interactions capture le comportement implicite, mais pas les contraintes explicites et strictes.

Sans ces données, le système ne peut pas effectuer les filtrages les plus élémentaires requis par l'utilisateur, comme "montre-moi des recettes sans gluten" ou "aide-moi à respecter mon objectif de 2000 calories par jour". Cette fonctionnalité est la pierre angulaire du concept de « Nutritionniste Personnel ».[1]

Pour combler cette lacune, la création d'un nouveau "Module de Profil Utilisateur" est indispensable. Il devrait comprendre les tables suivantes :

Table	Colonnes Clés	Objectif
user_profiles	user_id (PK, FK vers users), novelty_preference (numeric)	Stocke les préférences générales. Le novelty_preference implémente le curseur "Habitude vs. Nouveauté".
diets	id (PK), name (text, unique)	Table de référence pour les régimes (Végétarien, Vegan, Sans Gluten, etc.).
user_diets	user_id (PK, FK), diet_id (PK, FK)	Table de jonction plusieurs-à-plusieurs pour lier les utilisateurs à leurs régimes.
user_allergies	user_id (PK, FK),	Table de jonction pour lier

	canonical_food_id (PK, FK)	les utilisateurs aux aliments canoniques auxquels ils sont allergiques.
user_health_goals	user_id (PK, FK), target_calories (numeric), target_protein_g (numeric), etc.	Stocke les cibles nutritionnelles quantitatives quotidiennes pour un utilisateur.

4.4 Planification et Programmation des Repas

Enfin, le schéma fournit la structure pour organiser les recettes sélectionnées en un plan de repas cohérent.

- **meal_plans & planned_meals** : Ces tables permettent de concrétiser la planification.[1] La table meal_plans définit un conteneur pour une période donnée (par exemple, une semaine) pour un utilisateur. La table planned_meals insère ensuite une recette spécifique (recipe_id) dans un créneau défini par une date (meal_date) et un type de repas (meal_type, par exemple, Déjeuner, Dîner). C'est le résultat final du processus de synthèse qui génère un plan de repas hebdomadaire réalisable.[1]

Partie V : Synthèse du Système : ERD Complet et Chemins de Requête

Cette section finale unifie l'analyse précédente en une vue holistique, en décrivant le schéma complet et en démontrant comment il fonctionne en pratique pour fournir les fonctionnalités clés de l'application.

5.1 Le Diagramme Entité-Relation (ERD) Intégré

Un diagramme entité-relation complet servirait de plan directeur pour l'équipe de

développement. Il représenterait visuellement l'ensemble de la structure de la base de données, y compris toutes les tables analysées et corrigées de la source initiale [1], ainsi que les nouvelles tables proposées pour le profil utilisateur et la gestion des sous-recettes. Les relations (un-à-un, un-à-plusieurs, plusieurs-à-plusieurs) seraient clairement dessinées, avec les clés primaires et étrangères marquées. Ce diagramme est l'artefact final qui consolide toutes les décisions architecturales prises dans ce document.

5.2 Chemins de Requête Clés (Scénarios Narratifs)

Pour traduire le schéma statique en action dynamique, voici des exemples narratifs de la manière dont la base de données répondrait aux requêtes les plus complexes du système.

Scénario 1 : La Recommandation Anti-Gaspillage

Requête utilisateur : « Trouver une recette que je peux faire ce soir en utilisant le poulet dans mon frigo qui expire demain. »

1. **Filtrage de l'Inventaire** : La requête commence dans la table inventory_lots. Le système filtre les enregistrements pour le user_id actuel où expiration_date est demain et qty_remaining > 0. Il récupère le product_id (UUID) du lot de poulet.
2. **Identification de l'Archétype** : Le système joint inventory_lots à products en utilisant le product_id pour trouver l'enregistrement du produit. À partir de là, il récupère l'archetype_id (par exemple, 'Blanc de poulet frais').
3. **Recherche de Recettes Compatibles** : Le système interroge ensuite la table recipe_ingredients, recherchant toutes les entrées où archetype_id correspond à celui du 'Blanc de poulet frais'. Cela lui donne une liste de recipe_id potentiels.
4. **Classement et Présentation** : Le système récupère les détails de chaque recette potentielle depuis la table recipes. L'algorithme de classement applique alors le « Score d'Urgence » [1], donnant une priorité maximale à ces recettes car elles utilisent un ingrédient sur le point d'expirer. D'autres facteurs, comme le « Score de Complétude » (combien d'autres ingrédients l'utilisateur possède déjà), sont également appliqués avant de présenter la liste classée à l'utilisateur.

Scénario 2 : Le Plan de Repas Nutritionnellement Conscient

Requête utilisateur : « Générer un plan de repas pour la semaine prochaine pour un utilisateur allergique aux arachides avec un objectif de 2000 calories par jour. »

1. **Récupération des Contraintes** : Le système interroge les tables du module de profil utilisateur proposé. Il récupère l'ID de l'aliment canonique pour les 'Arachides' depuis la table user_allergies et l'objectif de 2000 kcal depuis user_health_goals.
2. **Filtrage Initial des Recettes** : Le système effectue un premier filtrage massif sur la base de données des recettes. Il exclut toutes les recettes qui contiennent l'ID de l'aliment canonique 'Arachides' dans leur table recipe_ingredients (via une jointure avec archetypes puis canonical_foods).
3. **Génération du Plan (Optimisation sous Contraintes)** : L'algorithme de planification hebdomadaire [1] commence à assembler un plan. Pour chaque jour, il maintient un budget calorique. Lorsqu'il sélectionne un dîner, il filtre l'ensemble des recettes autorisées pour n'inclure que celles dont le profil nutritionnel par portion ne ferait pas dépasser le budget quotidien restant.
4. **Optimisation Secondaire** : Au sein de cet ensemble de recettes valides, l'algorithme optimise pour des contraintes souples : il maximise l'utilisation des ingrédients de l'inventaire (Scénario 1), la satisfaction prédite de l'utilisateur (basée sur user_recipe_interactions), et la variété des cuisines, avant de remplir la table planned_meals.

Scénario 3 : Calcul de la Nutrition d'une Recette

Requête : « Calculer le profil nutritionnel pour une portion de la recette 'Quiche Lorraine'. »

1. **Itération des Ingrédients** : Le système récupère tous les enregistrements de recipe_ingredients pour le recipe_id de la Quiche Lorraine.
2. **Conversion et Standardisation** : Pour chaque ingrédient, il effectue une jointure vers archetypes ou canonical_foods pour accéder à son nutrition_id et à ses propriétés physiques (density_g_per_ml, etc.). Il utilise la quantity et l'unit de recipe_ingredients.
3. **Appel au Moteur de Conversion** : Le système utilise les tables unit_conversions_generic et unit_conversions_product pour convertir la quantité de chaque ingrédient en une unité standard de masse : les grammes. Par exemple, "200 ml de crème" est converti en grammes en utilisant la densité de la crème.
4. **Agrégation des Nutriments** : Pour chaque ingrédient, le système récupère son profil de la table nutritional_data (qui est par 100g). Il met à l'échelle les valeurs de chaque nutriment en fonction de la masse calculée en grammes. Ces valeurs sont additionnées pour tous les ingrédients pour obtenir un total brut pour la recette.
5. **Application des Facteurs de Cuisson** : Le système identifie la cooking_method de la

recette ("Cuisson au four") dans la table recipes. Il interroge la table cooking_nutrition_factors pour trouver les facteurs de rétention pertinents (par exemple, pour les vitamines sensibles à la chaleur) et ajuste les totaux de nutriments en conséquence.

6. **Calcul par Portion** : Enfin, le système récupère la valeur servings de la table recipes et divise les totaux de nutriments ajustés par ce nombre pour obtenir le profil nutritionnel final par portion. Ce processus démontre la synthèse d'au moins six tables différentes pour réaliser une seule fonctionnalité critique.[1]

Conclusion et Recommandations

Ce document a reconstruit et documenté un schéma de base de données complet pour un écosystème culinaire intelligent. L'architecture qui en résulte, basée sur une hiérarchie alimentaire à quatre niveaux et des modèles de recettes déconstruites, est robuste, évolutive et étroitement alignée sur les objectifs stratégiques d'une application axée sur la personnalisation, la santé et la réduction du gaspillage.

L'analyse a identifié plusieurs points forts, notamment la séparation stratégique des recettes et des marques, et un système sophistiqué de gestion des données nutritionnelles. Cependant, elle a également mis en évidence des faiblesses critiques dans le schéma initial fourni, qui doivent être corrigées pour assurer la viabilité du système.

Les recommandations suivantes sont donc formulées pour l'équipe de développement :

1. **Correction Impérative de inventory_lots** : Le modèle d'association polymorphe doit être abandonné. La colonne product_id doit être modifiée en type uuid et devenir une clé étrangère directe et non nullable vers products.id.
2. **Reconstruction de la Table users** : La table users actuelle est corrompue et doit être remplacée par un schéma standard et sécurisé, tel que celui fourni par le système d'authentification sous-jacent.
3. **Implémentation du Module de Profil Utilisateur** : La lacune la plus importante est l'absence de stockage des préférences explicites de l'utilisateur. La création des tables user_profiles, diets, user_diets, user_allergies, et user_health_goals est une priorité absolue pour activer les fonctionnalités de filtrage et de personnalisation de base.
4. **Extension pour les Sous-Recettes** : Pour une modularité et une réutilisabilité maximales, l'ajout d'une colonne sub_recipe_id à la table recipe_ingredients est fortement recommandé.

En mettant en œuvre ces corrections et ajouts, le schéma présenté dans ce document fournira une base de données solide et pérenne, capable de soutenir la gamme complète des

fonctionnalités intelligentes envisagées et de faire de l'application un véritable assistant culinaire personnalisé.

Sources des citations

1. Planificateur de repas intelligent _ conception et intégration.pdf