PROVINCE DU HAINAUT
UNIVERSITE DU TRAVAIL
INSTITUT D'ENSEIGNEMENT TECHNIQUE COMMERCIAL
DE PROMOTION SOCIALE

Eat-Free

Projet de développement SGBD

Rapport remis par Ilhami Kandemir

2025, Informatique

Année académique 2025-2026

Table des matières

3. Introduction

Contexte du projet

Dans un contexte où le gaspillage alimentaire et la gestion du budget domestique sont des préoccupations majeures, il devient difficile pour les particuliers de suivre précisément ce qu'ils possèdent dans leurs placards et de planifier leurs repas en conséquence.

Objectifs

L'objectif principal de ce projet est de développer l'application "Eat-Free", une solution desktop permettant de :

1. Gérer un inventaire d'ingrédients (stockage, quantités, dates de péremption).
2. Créer et gérer des recettes utilisant ces ingrédients.
3. Planifier des repas sur un calendrier.
4. Suivre la consommation quotidienne via un journal alimentaire.

Description de la problématique

La problématique centrale est la gestion de données relationnelles complexes entre des stocks (inventaire), des abstractions (recettes) et des événements temporels (repas planifiés). Le défi technique réside dans la mise en place d'une base de données relationnelle robuste (SGBD) capable de maintenir l'intégrité de ces liens et de supporter la charge via un serveur dédié.

Méthodologie

Le développement a suivi une approche agile avec des itérations successives :

1. Conception de la base de données (SQL).
2. Développement du back-end (Node.js/Electron).
3. Création de l'interface utilisateur (Vue.js).
4. Tests et intégration

4. Analyse des besoins

Acteurs du système

- **L'Utilisateur Principal :** Personne gérant le stock alimentaire du foyer et planifiant les repas.

Cas d'utilisation (Use Cases)

Les principales fonctionnalités identifiées sont :

- **Gestion des ingrédients :** Créer, modifier, supprimer un ingrédient (CRUD).
- **Gestion de l'inventaire :** Ajouter des quantités à un ingrédient existant, modifier et supprimer.
- **Gestion des recettes :** Créer une recette en associant plusieurs ingrédients avec des quantités spécifiques.
- **Planification :** Assigner une recette à une date et un type de repas (Petit-déjeuner, Déjeuner, Dîner).
- **Journalisation :** Enregistrer les repas effectivement consommés pour un suivi nutritionnel (calories, protéines, etc.).

Contraintes techniques

- **SGBD :** MySQL Server (choisi pour sa robustesse, sa gestion avancée de la concurrence et son architecture client-serveur standard dans l'industrie).
- **Langage :** TypeScript (pour la sécurité du typage), SQL (pour les requêtes complexes).
- **Framework :** Electron (pour l'application desktop) et Vue.js (pour l'interface).
- **ORM :** Prisma (utilisé pour la définition du schéma et les migrations), complété par des scripts SQL natifs pour des fonctionnalités avancées.

5. Modélisation

Modèle Conceptuel des Données (MCD)

Le système s'articule autour de l'utilisateur qui interagit avec son inventaire, ses recettes et son planning.

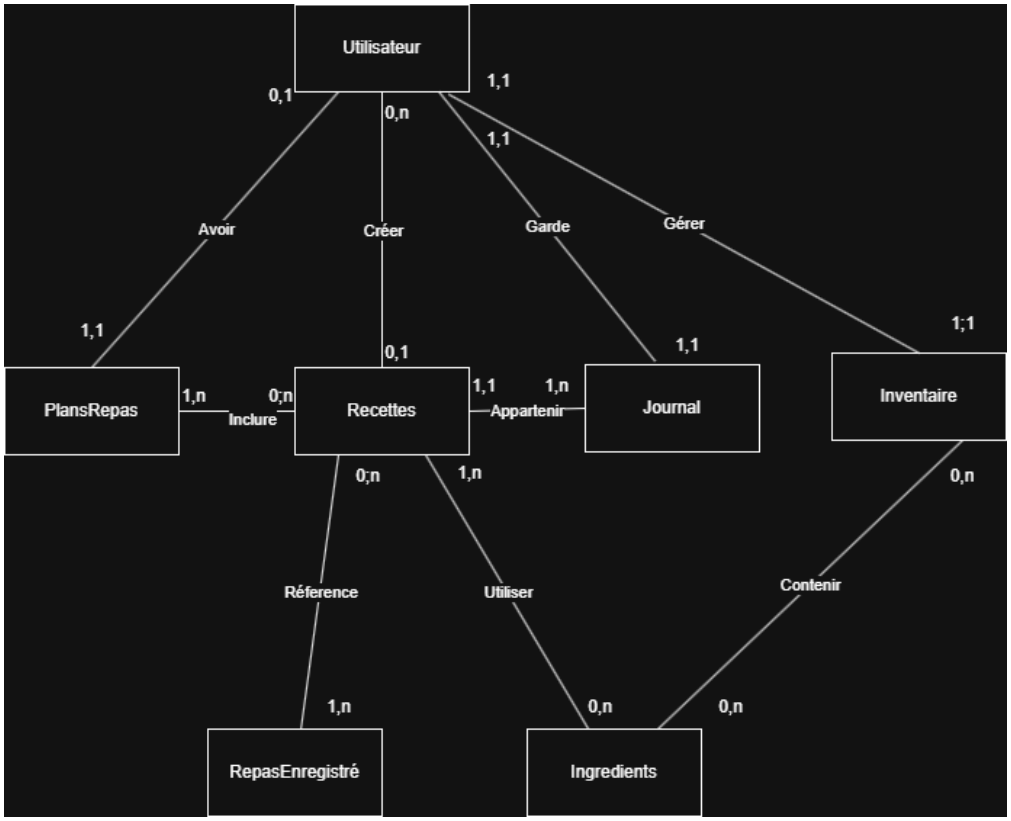
Entités principales :

- **User (Utilisateur) :** L'entité centrale possédant les données.
- **Ingredient (Ingrédient) :** Les produits de base (ex: Riz, Pomme).
- **Recipe (Recette) :** Un plat composé d'ingrédients.
- **MealPlan (Plan de Repas) :** Une planification à une date donnée.
- **Inventory (Inventaire) :** Association caractérisée entre un utilisateur et un ingrédient (stock).

Représentation graphique (Diagramme Entité-Relation) :

```
User ||--o{ Inventory : "Gère"
User ||--o{ Recipe : "Crée"
User ||--o{ MealPlan : "Planifie"
User ||--o{ Journal : "Enregistre"

Ingredient ||--o{ Inventory : "Est stocké dans"
Ingredient ||--o{ RecipeIngredient : "Compose"
```

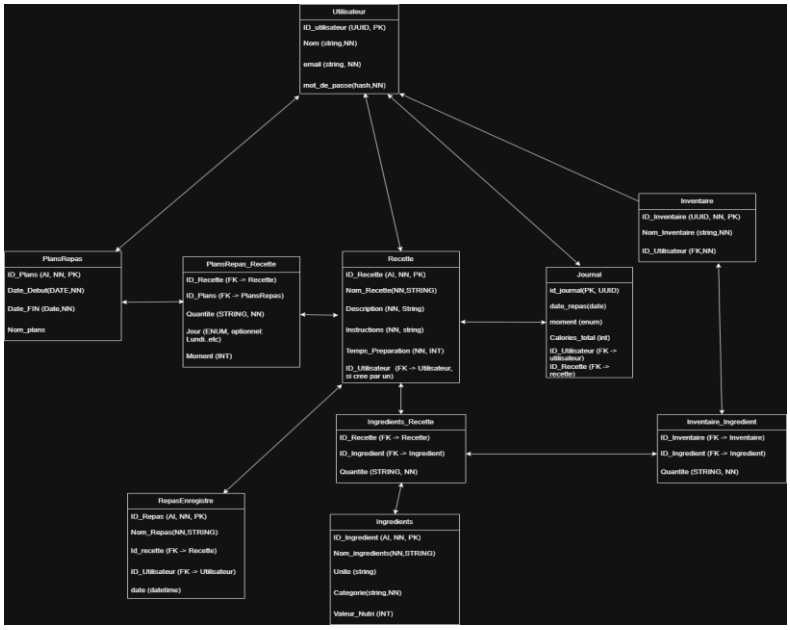
```
Recipe ||--o{ RecipeIngredient : "Contient"
Recipe ||--o{ MealPlanRecipe : "Est planifié dans"

MealPlan ||--o{ MealPlanRecipe : "Inclut"
```



Modèle Logique des Données (MLD)

Le schéma relationnel a été normalisé pour éviter la redondance des données nutritionnelles.
*(Légende : **Clé Primaire (Gras)**, #Clé Étrangère)*

- **users** (**id**, username, password_hash, created_at)
- **ingredients** (**id**, name, calories, proteins, carbs, fats, image_url)
- **recipes** (**id**, name, instructions, prep_time, cook_time, servings, #user_id)
- **inventory** (**id**, #user_id, #ingredient_id, quantity, unit, expiration_date)
- **recipe_ingredients** (**#recipe_id**, **#ingredient_id**, quantity, unit)
- **meal_plans** (**id**, #user_id, date, meal_type)
- **meal_plan_recipes** (**#meal_plan_id**, **#recipe_id**)
- **journal** (**id**, #user_id, date, meal_type, #recipe_id, #ingredient_id, calories, proteins, carbs, fats)
- **saved_recipes** (**#user_id**, **#recipe_id**)



6. Conception et architecture du projet

Organisation de la base de données

La base de données est structurée pour minimiser la redondance. Par exemple, les informations nutritionnelles sont stockées uniquement dans la table `ingredients` et non répétées dans l'inventaire ou les recettes. La table `recipe_ingredients` permet une relation *Many-to-Many* entre les recettes et les ingrédients.

Architecture Logicielle

L'application suit une architecture modulaire séparant le Frontend et le Backend :

- **Frontend (Renderer) :** Vue.js gère l'affichage et l'interaction utilisateur.
- **Backend (Main) :** Node.js gère la logique métier et la communication avec le serveur de base de données.
- **SGBD :** MySQL Server stocke les données de manière centralisée et persistante.
- **IPC (Inter-Process Communication) :** Un pont sécurisé (`preload.ts`) permet au Frontend de demander des données au Backend, qui relaie ensuite les requêtes SQL au serveur MySQL.

Sécurité et Intégrité

- **Intégrité référentielle :** Les clés étrangères (`FOREIGN KEY`) configurées dans MySQL assurent que l'on ne peut pas, par exemple, ajouter un stock pour un ingrédient qui n'existe pas

7. Implémentation

Scripts SQL et Base de Données

La création de la base de données s'appuie sur le fichier `schema.sql`. Exemple de création de table (syntaxe MySQL) :

```
CREATE TABLE IF NOT EXISTS ingredients (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    calories FLOAT DEFAULT 0,
    proteins FLOAT DEFAULT 0,
    -- autres colonnes...
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Des triggers ont été envisagés (voir `routines.sql`) pour automatiser certaines tâches, comme la mise à jour automatique des dates de modification (`updated_at`).

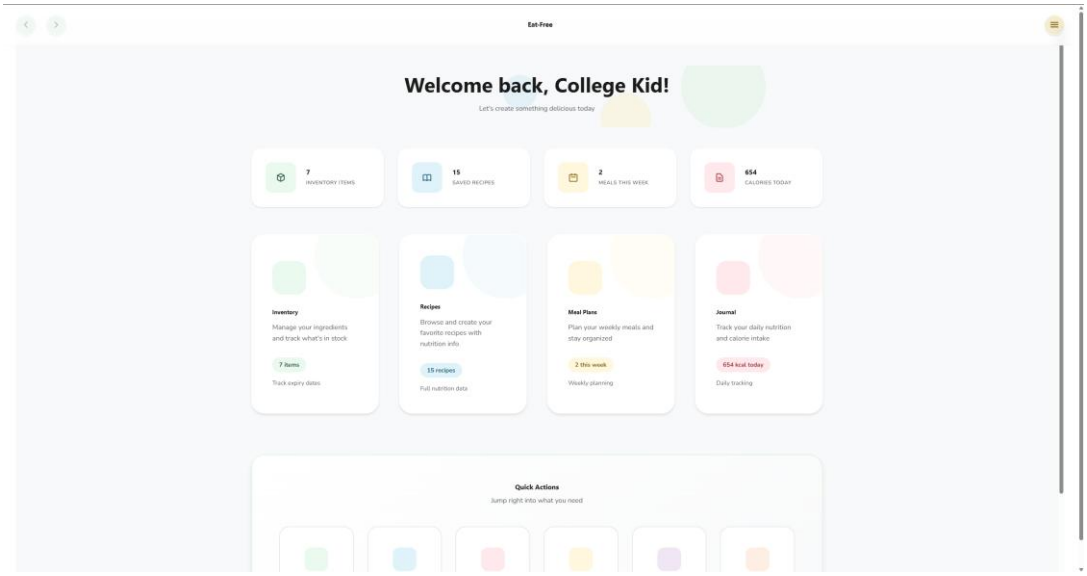
Développement des fonctionnalités (CRUD)

L'application permet la gestion complète des données. *Exemple technique :* Le fichier `ingredientsRepository.ts` contient les méthodes pour interagir avec la base :

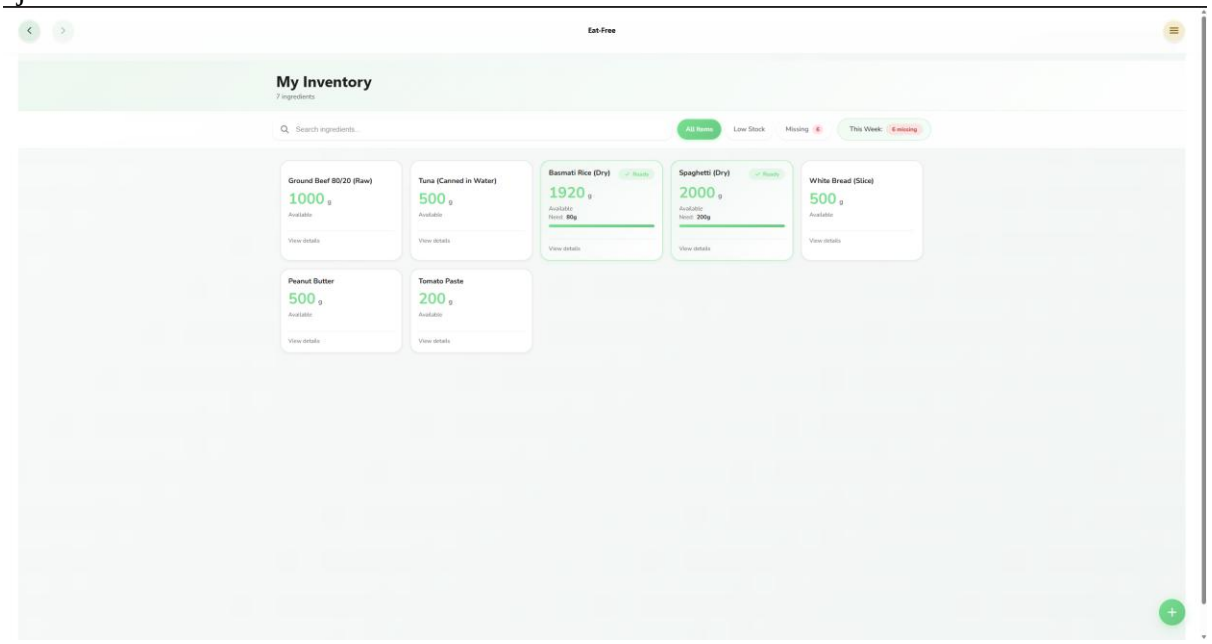
- `createIngredient` (INSERT)
- `getAllIngredients` (SELECT)
- `updateIngredient` (UPDATE)

Interface Utilisateur (Screenshots)

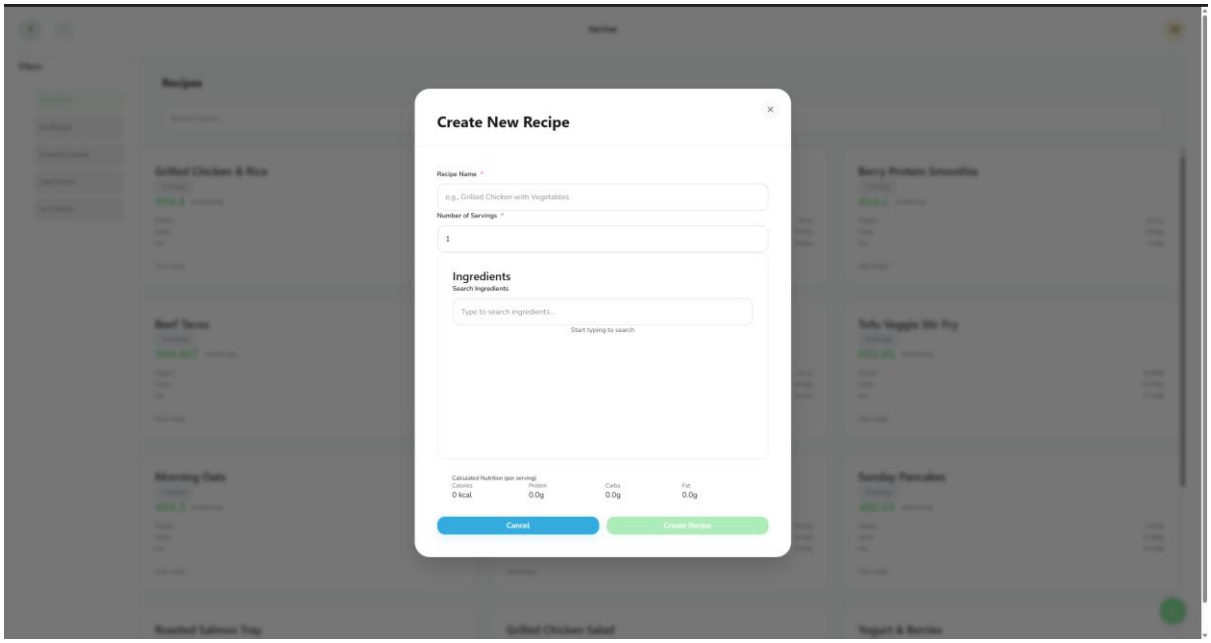
1. Tableau de bord (Dashboard) Cette page affiche un résumé des calories et des repas prévus



2. Gestion de l'Inventaire Interface permettant de voir les ingrédients en stock et d'en ajouter.



3. Création de Recette Formulaire permettant d'ajouter une recette et de sélectionner les ingrédients.



Problèmes rencontrés et solutions

- **Problème :** Connexion asynchrone au serveur MySQL dans le cycle de vie de l'application Electron.
    - *Solution :* Mise en place d'un gestionnaire de connexion robuste avec gestion des erreurs au démarrage.
- **Problème :** Cohérence des données nutritionnelles lors de la modification d'une recette.
    - *Solution :* Recalcul dynamique côté application lors de l'affichage, basé sur les ingrédients liés.

8. Tests et validation

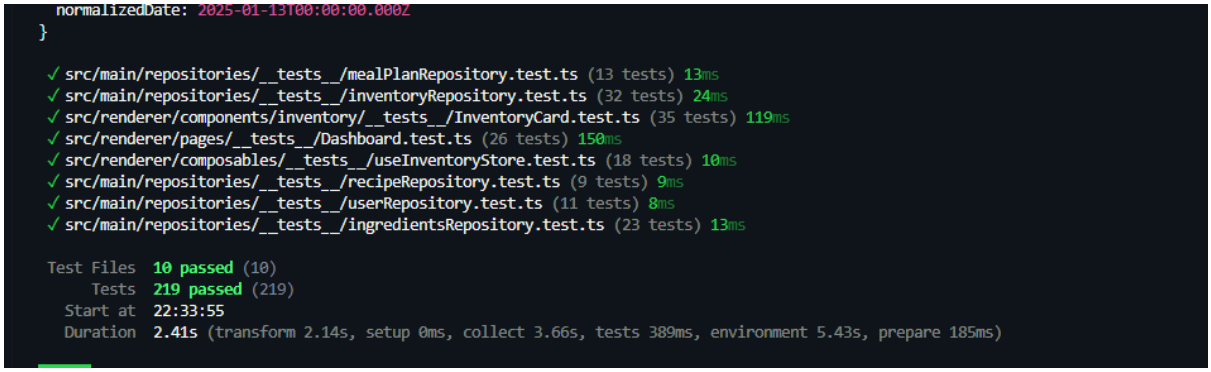
Stratégie de tests

Nous avons utilisé une approche mixte :

1. **Tests Unitaires :** Utilisation de **Vitest** (configuré dans `vitest.config.ts`) pour tester la logique des repositories (fichiers dans `__tests__`).
2. **Tests SQL :** Scripts dédiés (`testing-joins.sql`, `testing-triggers.sql`) exécutés directement sur le serveur MySQL pour vérifier que les requêtes complexes renvoient les bons résultats.

Jeux d'essai

Des données fictives (Seeding) ont été créées via le fichier `seeding.sql` pour peupler la base avec des ingrédients de base (Pomme, Riz, Poulet) et des utilisateurs de test.



Vérification des contraintes

Les tests ont confirmé que la suppression d'un utilisateur entraîne bien la suppression en cascade de ses données d'inventaire (grâce aux contraintes `ON DELETE CASCADE` définies dans le schéma MySQL).

9. Résultats et discussion

Bilan technique

L'application est fonctionnelle et permet de réaliser l'ensemble du cycle : Achat (Inventaire) -> Cuisine (Recette) -> Consommation (Journal). L'utilisation de MySQL Server assure une gestion performante des données et une évolutivité future (multi-utilisateurs possible).

Analyse critique

- **Points forts :** Interface moderne et réactive (Vue.js), architecture propre (Repository pattern), typage fort (TypeScript), robustesse du SGBD MySQL.
- **Limites :** L'application nécessite une connexion au serveur MySQL local pour fonctionner.

Améliorations futures

- Ajouter une fonctionnalité de génération automatique de liste de courses basée sur le plan de repas.
- Exporter les données en PDF/CSV.
- Ajouter des graphiques pour l'analyse nutritionnelle sur le mois.
- Ajout d’un IA pour générer des recettes a partir de l’inventaire

10. Références bibliographiques

1. Documentation officielle Electron : https://www.electronjs.org/docs
2. Documentation Vue.js : https://vuejs.org/guide/introduction.html
3. Documentation MySQL : https://dev.mysql.com/doc/
4. Documentation Prisma : https://www.prisma.io/docs
5. Gemini
6. ChatGpt
7. Claude

11. Annexes

Annexe 1 : Extrait du Schéma SQL (`schema.sql`)

```sql
CREATE DATABASE IF NOT EXISTS eat_free
  DEFAULT CHARACTER SET utf8mb4
  COLLATE utf8mb4_0900_ai_ci;
USE eat_free;

SET NAMES utf8mb4;
SET time_zone = '+00:00';
SET sql_mode =
'STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION,ERROR_FOR_DIVISION_BY_ZERO,NO_ZERO
_DATE,NO_ZERO_IN_DATE';

-- ---------- USERS ----------
CREATE TABLE IF NOT EXISTS User (
  id            INT PRIMARY KEY AUTO_INCREMENT,
  email         VARCHAR(255) NOT NULL UNIQUE,
  name          VARCHAR(120) NOT NULL,
  created_at    TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ---------- INGREDIENTS (global catalog) ----------
CREATE TABLE IF NOT EXISTS Ingredients (
  id                          INT PRIMARY KEY AUTO_INCREMENT,
  name                        VARCHAR(255) NOT NULL UNIQUE,
  kcal_per_100g               FLOAT NOT NULL DEFAULT 0,
  protein_g_per_100g          FLOAT NOT NULL DEFAULT 0,
  carbs_g_per_100g            FLOAT NOT NULL DEFAULT 0,
  fat_g_per_100g              FLOAT NOT NULL DEFAULT 0,
  created_at                  TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ---------- RECIPES ----------
-- user_id is NULL => global/admin recipe
CREATE TABLE IF NOT EXISTS Recipe (
```

```
  id                          INT PRIMARY KEY AUTO_INCREMENT,
  user_id                     INT NULL,
  name                        VARCHAR(255) NOT NULL,
  servings                    FLOAT NOT NULL DEFAULT 1,

  -- cached per-serving nutrition (denormalized for speed)
  kcal_per_serving            FLOAT NOT NULL DEFAULT 0,
  protein_g_per_serving       FLOAT NOT NULL DEFAULT 0,
  carbs_g_per_serving         FLOAT NOT NULL DEFAULT 0,
  fat_g_per_serving           FLOAT NOT NULL DEFAULT 0,

  created_at                  TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,

  CONSTRAINT fk_recipe_user
    FOREIGN KEY (user_id) REFERENCES User(id)
    ON DELETE SET NULL,
  CONSTRAINT chk_servings_positive CHECK (servings > 0)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Lines of a recipe: quantities stored in grams
CREATE TABLE IF NOT EXISTS Recipe_Ingredients (
  recipe_id        INT NOT NULL,
  ingredient_id    INT NOT NULL,
  qty_grams        FLOAT NOT NULL CHECK (qty_grams >= 0),
  notes            VARCHAR(255) NULL,

  PRIMARY KEY (recipe_id, ingredient_id),
  CONSTRAINT fk_recipe_ing_recipe  FOREIGN KEY (recipe_id)   REFERENCES
Recipe(id)     ON DELETE CASCADE,
  CONSTRAINT fk_recipe_ing_ingredient     FOREIGN KEY (ingredient_id)
REFERENCES Ingredients(id) ON DELETE RESTRICT
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ---------- INVENTORY ----------
CREATE TABLE IF NOT EXISTS Inventory (
  id            INT PRIMARY KEY AUTO_INCREMENT,
  user_id       INT NOT NULL UNIQUE,
  created_at    TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT fk_inventory_user FOREIGN KEY (user_id) REFERENCES User(id) ON
DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- current stock per ingredient (can be negative to show deficits)
CREATE TABLE IF NOT EXISTS Inventory_Ingredient (
  inventory_id   INT NOT NULL,
```

Annexe 2 : Exemple de code TypeScript (`ingredientsRepository.ts`)

```typescript
async getIngredients(): Promise<Ingredient[]> {
    const result = await this.dbclient.ingredients.findMany();

    return result.map((ing) => ({
      id: (ing.id),
      name: ing.name,
      kcal_per_100g: (ing.kcal_per_100g),
      protein_g_per_100g: (ing.protein_g_per_100g),
      carbs_g_per_100g: (ing.carbs_g_per_100g),
      fat_g_per_100g: (ing.fat_g_per_100g),
      created_at: ing.created_at.toISOString(),
    })) as Ingredient[];
  }
```