Cahier des Charges Fonctionnel et Technique

Projet : Cuisine Assistante (v2.0)

Version : 2.0 (Refactorisation "Audit") **Date :** 1 novembre 2025 **Auteur :** Matteo (Concept) & Gemini (Développement IA)

1. Introduction et Objectifs

1.1. Nom du Projet

Cuisine Assistante

1.2. Objectif Principal

Développer une application web (Progressive Web App) d'assistance culinaire personnelle, destinée à simplifier la gestion quotidienne de la cuisine. L'application doit permettre à un utilisateur (cible principale : la mère du concepteur) de gérer son inventaire (stock et équipements), de générer des idées de recettes basées sur cet inventaire via une IA (Google Gemini), de planifier ses repas et de générer des listes de courses.

1.3. Philosophie du Projet

L'application doit être :

- **Intuitive :** Interface épurée, minimaliste et moderne (inspirée d'Apple/Notion).
- **Intelligente :** L'IA est au cœur de l'expérience, automatisant les tâches fastidieuses (génération, catégorisation, planification).
- **Aérienne :** L'expérience utilisateur (UX) doit être fluide, rapide et reposante, grâce à des espacements généreux et un design "soft".

2. Architecture Technique

2.1. Stack Technologique

- **Frontend :** React v18 (Hooks, Contexte)
- **Bundler :** Vite
- **Styling :** TailwindCSS (JIT)
- **Backend (BaaS) :** Google Firebase
  - **Base de données :** Firestore (NoSQL, temps réel)
  - **Authentification :** Firebase Authentication (Anonyme ou Token)
- **Services IA :** Google Gemini API (Modèle Flash)

2.2. Structure du Code (Logique)

Le code (actuellement monofichier) suit une architecture modulaire V2.0 conçue pour la maintenabilité et l'évolution :

- `AppContext` : Un Contexte React global centralise l'état (DB, Auth, Données, Vue active, Toasts).
- **Services (Isolation) :**
  - `firestoreService` : Objet regroupant toute la logique CRUD (add, update, delete, set, listenToCollection).
  - `geminiService` : Objet regroupant tous les appels à l'IA (génération, catégorisation, formatage) et la logique de simulation (mode démo).
- **Hooks Personnalisés :**
  - `useAuth` : Gère le cycle de vie de l'authentification Firebase.
  - `useFirestoreQuery` : Hook générique pour écouter en temps réel une collection Firestore de l'utilisateur.
- **Composants "Features" :** Les composants principaux (Stock, Recettes, etc.) sont découplés de la logique métier et consomment le Contexte.

3. Identité Visuelle (UI/UX)

3.1. Palette de Couleurs

- **Fond Principal (Cartes) :** Blanc Pur (`#FFFFFF`)
- **Fond Secondaire (Beige) :** `#FCEED0` (Barre de navigation, Footer)
- **Fond Global (Crème) :** `#FFF9F2` (Arrière-plan de l'application)
- **Accent (Vert Menthe) :** `#3BA16D` (Header, boutons primaires, onglets actifs)
- **Danger (Corail) :** `#F56C6C` (Boutons de suppression, alertes)
- **Texte :** `#2F2F2F` (Principal), `#555` (Secondaire)

3.2. Typographie

- **Famille :** "Inter" (Sans-serif, importée)
- **Hiérarchie :** Titres de section (ex: "Mon Garde-Manger") en `text-3xl` ou `text-4xl` , texte courant en `text-base` .

3.3. Design & Ergonomie

- **Layout :** Structure "Bloc" classique (Header / Nav / Main / Footer).
- **Contenu :** Le contenu principal s'affiche dans une carte blanche centrée (`max-w-4xl`) avec une ombre douce (`shadow-lg`) et des bordures légères (`border border-[#EAEAEA]`).
- **Navigation :** Barre d'onglets (`<nav>`) sous le header, utilisant des boutons arrondis (`rounded-full`) avec un fond vert pour l'onglet actif.
- **Feedback :** Remplacement total des `alert()` par un système de **Toaster** (notifications "pop-up" en haut à droite, vertes pour succès, rouges pour erreur).
- **Composants Standardisés :** `Button` (variantes), `Input` , `Select` , `GeneratingLoader` (animation), `EmptyState` .

4. Fonctionnalités Détaillées (Modules)

4.1. Module 1 : Stock (Garde-Manger)

- **Fonctionnalité :** CRUD (Ajout/Suppression) d'ingrédients dans Firestore.
- **Données (V2.1) :** Ajout de la **Quantité** (nombre) et de l'**Unité** (g, L, pièce(s), etc.) via un formulaire enrichi.
- **IA (V2.2 - Simulée) :** Appel au `geminiService.categorizeIngredient()` lors de l'ajout.
- **Affichage (V2.2) :** Les ingrédients sont groupés par catégorie (Fruits, Légumes, Épicerie...) et triés alphabétiquement.

4.2. Module 2 : Équipement

- **Fonctionnalité :** CRUD (Ajout/Suppression) simple d'équipements (Four, Wok...).
- **Affichage :** Liste simple.

4.3. Module 3 : Recettes (Génération IA)

- **Fonctionnalité :** Formulaire de génération permettant de spécifier :
  - Régime (Végétarien, Vegan...)
  - Nombre de personnes
  - Temps de préparation max
  - Difficulté
  - Requête textuelle libre.
- **IA (Simulée) :** Appel au `geminiService.generateRecipe()` qui utilise les filtres et (à terme) le stock pour générer une recette au format JSON (selon `RECIPE_SCHEMA`).
- **Affichage :** Affiche la recette générée via le composant `RecipeDisplay`.
- **Interaction :**
  - Bouton "Ajouter les ingrédients manquants" (ajoute à la collection "shopping_list").
  - Bouton "Sauvegarder dans mes favoris".
  - Bouton "Modifier la recette" (passe en mode édition).

4.4. Module 4 : Favoris (Carnet de recettes)

- **Fonctionnalité (V2.1) :** CRUD (Suppression) des recettes sauvegardées.
- **Affichage :** Liste des recettes. Un clic ouvre une modale (`RecipeDisplay`) pour voir le détail.
- **Édition (V2.1) :** Permet l'édition manuelle complète d'une recette sauvegardée (titre, ingrédients, instructions...).
- **Notation (V2.1) :** Permet d'attribuer une note (étoiles) et une note textuelle personnelle, sauvegardée sur Firestore.
- **IA (V2.1 - Simulée) :** Module d'importation. L'utilisateur colle un texte brut, `geminiService.formatImportedRecipe()` le transforme en JSON structuré et le sauvegarde.

4.5. Module 5 : Courses

- **Fonctionnalité :** CRUD (Ajout manuel, Suppression).

- **Interaction :**
  - Permet de cocher/décocher un article (état `purchased: true/false`).
  - L'état est sauvegardé en temps réel sur Firestore.
- **Affichage :** Tri automatique (non achetés en haut, achetés barrés en bas).

4.6. Module 6 : Planification

- **Affichage :** Vue calendrier des 7 prochains jours.
- **Fonctionnalité (V2.3) :**
  - Permet d'assigner manuellement une recette (depuis les Favoris) à un créneau (Petit-déjeuner, Déjeuner, Dîner).
  - Sauvegarde le planning dans un document unique Firestore (`planning/weekly_plan`).
- **IA (V2.3 - Simulée) :** Bouton "Générer (IA)" qui appelle `geminiService.generateWeeklyPlan()` pour remplir automatiquement la semaine (en utilisant les favoris).

5. Prochaines Étapes (Audit v2.0 non implémenté)

5.1. Priorité Moyenne

- **Profil Utilisateur :** Créer un nouveau module "Profil" pour stocker les préférences globales (régime par défaut, allergies) qui pré-rempliront le formulaire de génération de recettes.
- **IA - Analyse Nutritionnelle :** Utiliser l'IA pour ajouter les valeurs nutritionnelles réelles (Calories, Protéines...) au schéma de recette.
- **Stock - Dates de Péremption :** Ajouter un champ "Date" au module Stock et créer des alertes.

5.2. Priorité Basse

- **IA - Visuels :** Utiliser un modèle d'IA multimodal (Gemini) ou de génération d'image (Imagen) pour ajouter une image de plat à la recette générée.
- **Export/Partage :** Permettre l'export de la Liste de Courses ou d'une Recette en PDF.
- **Déploiement PWA :** Création d'un `manifest.json` et d'un `service-worker.js` pour l'installation mobile et le mode hors-ligne partiel.