# 🤖 MaTanne v2 - Document de Référence

## Assistant Familial Intelligent avec IA

### 3.6 Mode Mains Libres Complet 🚀

**Objectif** : Utilisation 100% vocale pendant cuisine

**Fonctionnalités**

- Commandes vocales (Web Speech API)
- Retour audio synthèse vocale
- Contrôle timers vocal
- Navigation recette mains libres
- Activation wake word "Hey MaTanne"

**Commandes**

```
"Hey MaTanne, étape suivante"
"Hey MaTanne, timer 10 minutes"
"Hey MaTanne, combien de sel ?"
"Hey MaTanne, pause timer"
```

---

### 3.7 App Mobile Native 🚀

**Objectif** : Expérience mobile optimisée

**Framework**

- React Native ou Flutter
- Synchronisation cloud temps réel
- Mode hors-ligne (cache local)

**Fonctionnalités Mobile**

- Scan code-barres natif
- Photo factures
- Notifications push
- Widget courses (Today screen)
- Mode sombre adaptatif
- Partage familial

---

### 3.8 Intégrations Tierces 🚀

**Objectif** : Connexions services externes

**Intégrations Prioritaires**

- **Yuka** : Scores qualité produits
- **Open Food Facts** : Base données produits
- **Google Calendar** : Sync planning repas
- **Todoist/Notion** : Export tâches courses
- **Deliveroo/UberEats** : Import commandes
- **MyFitnessPal** : Sync nutrition

---

### 3.9 Communauté & Partage 🚀

**Objectif** : Fonctionnalités sociales (opt-in)

**Fonctionnalités**

- Partage recettes publiques
- Bibliothèque communautaire
- Notation recettes ( ⭐ ⭐ ⭐ ⭐ ⭐ )
- Commentaires et photos
- Fork recettes (adaptation personnelle)
- Top recettes semaine/mois
- Profils publics anonymisés

**Gamification Sociale**

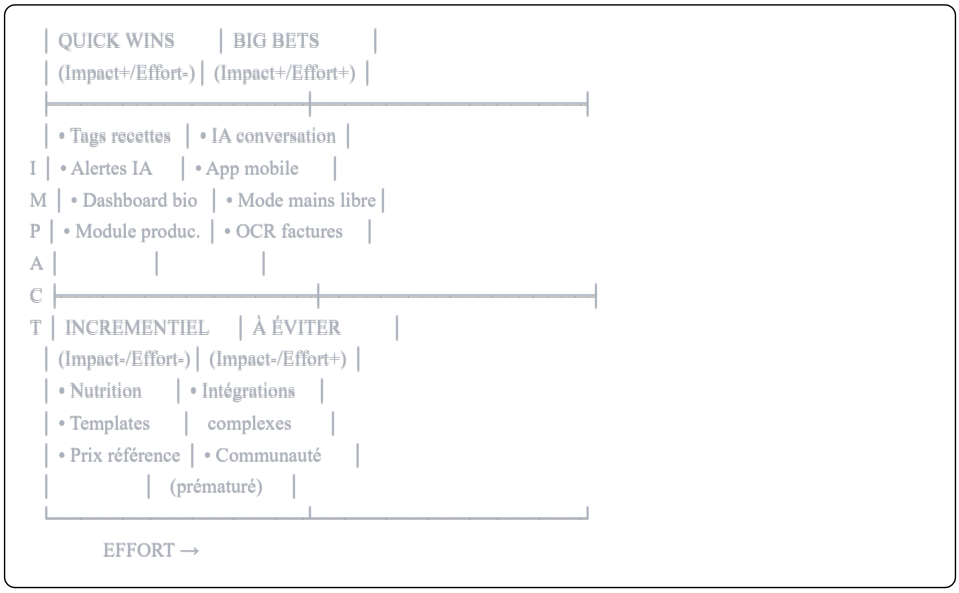
- Leaderboards (opt-in anonyme)
- Défis communautaires
- Achievements partagés
- Groupes (ex: "Parents Bio 2025")

---

## 📊 Roadmap & Priorisation

### 🎯 Critères de Priorisation

**Matrice Impact / Effort**




---

### 🗓️ Planning Détaillé

**PHASE 1 : FONDATIONS ✅ (Semaines 1-3)**

**Semaine 1 : Tags & Configuration**

```
Lundi-Mardi : Tags Recettes
├─ Modèle : +colonnes bio, local, robots, scores
├─ UI : Filtres, badges, icônes
└─ Service : Calcul auto scores

Mercredi-Jeudi : Config Batch
├─ Modèle : +batch_preferences ConfigPlanning
├─ UI : Widget config paramètres
└─ Service : Validation + génération adaptée

Vendredi : Module Producteurs
├─ Modèle : Table Producteur
├─ UI : Liste + fiche + formulaire
└─ Service : CRUD producteurs

🎯 Résultat S1 : Tags opérationnels + Batch configurable + Producteurs intégrés
```

**Semaine 2 : IA & Alertes**

```
Lundi-Mardi : Alertes Rachats IA
├─ Service : predict_consommation()
├─ UI : Dashboard alertes triées urgence
└─ Intégration : Lien courses + producteurs

Mercredi-Vendredi : Dashboard Bio/Local
├─ Service : Calcul métriques bio/local
├─ UI : Dashboard visuel avec graphiques
├─ Rapport : Template email mensuel
└─ Tests : Validation calculs

🎯 Résultat S2 : Prédictions stock + Dashboard impact complet
```

**Semaine 3 : Polissage & Tests**

```
Lundi-Mercredi : Intégrations
├─ Lien recettes ↔ producteurs
├─ Filtres avancés bio/local
├─ Stats enrichies
└─ Tests bout-en-bout

Jeudi-Vendredi : Documentation & Release
├─ README mis à jour
├─ Guide utilisateur nouvelles features
├─ Tests utilisateurs (famille)
└─ Déploiement production

🎯 Résultat S3 : Phase 1 complète, stable, documentée
```

✅ **Livrables Phase 1**

- [x] Tags bio/local/robots recettes
- [x] Configuration batch personnalisée
- [x] Module producteurs opérationnel
- [x] Alertes rachats prédictives
- [x] Dashboard bio/local complet
- [x] Documentation complète

---

**PHASE 2 : OPTIMISATION ⭐ (Semaines 4-8)**

**Semaine 4 : Planning IA Optimisé**

```
Lundi-Mercredi : Amélioration Génération
├─ Prompt : Intégration robots + bio/local
├─ Service : Contraintes enrichies
├─ Parsing : Validation Pydantic étendue
└─ Tests : Scénarios complexes

Jeudi-Vendredi : Templates Planning
├─ Modèle : Table TemplatePlanning
├─ UI : Sauvegarde/Galerie/Application
└─ Service : CRUD templates

🎯 Résultat S4 : Planning IA ultra-optimisé + Templates réutilisables
```

**Semaine 5 : Nutrition**

```
Lundi-Mercredi : Modèle Nutrition
├─ Modèle : +nutrition Recette, Table NutritionJour
├─ Service : Calcul nutrition automatique
└─ API : Open Food Facts (données nutrition)

Jeudi-Vendredi : UI Nutrition
├─ Dashboard nutrition journalier
├─ Graphiques évolution
├─ Alertes déséquilibres
└─ Suggestions IA

🎯 Résultat S5 : Tracking nutrition opérationnel
```

**Semaine 6-7 : Mode Session Active**

```
Semaine 6 : Backend Session
├─ Service : Gestion état session
├─ Timers : Multiples synchronisés
├─ Workflow : Étapes guidées
└─ Persistance : Sauvegarde progression

Semaine 7 : UI Session
├─ Interface session active
├─ Checklist temps réel
├─ Vue robots (statut)
├─ Alertes sonores
└─ Mode vocal (phase 1)

🎯 Résultat S6-7 : Mode Session Active complet et testé
```

**Semaine 8 : Courses Multi-Magasins**

```
Lundi-Mercredi : Service Multi-Magasins
├─ Répartition intelligente articles
├─ Calcul trajets optimaux
├─ Intégration producteurs
└─ Prix par magasin

Jeudi-Vendredi : UI Multi-Magasins
├─ Liste organisée magasin/rayon/producteur
├─ Cartes + distance + horaires
├─ Mode "Course en cours"
└─ Export PDF/Print

🎯 Résultat S8 : Courses multi-magasins optimisées
```

✅ **Livrables Phase 2**

- [ ] Planning IA avec tous critères (robots, bio, bébé)

- [ ] Templates planning sauvegardables
- [ ] Nutrition complète avec dashboard
- [ ] Mode Session Active batch cooking
- [ ] Courses multi-magasins avec trajets
- [ ] Prix références avec historique

---

**PHASE 3 : EXCELLENCE 🚀 (Semaines 9-12+)**

**Semaine 9 : Scan Code-Barres**

```
Lundi-Mercredi : Intégration Scanner
├─ Tech : Streamlit + JavaScript caméra
├─ API : Open Food Facts
├─ Service : Mapping produit → inventaire
└─ Tests : Différents codes-barres

Jeudi-Vendredi : UI Scanner
├─ Modal scanner
├─ Prévisualisation données
├─ Validation/édition
└─ Fallback saisie manuelle

🎯 Résultat S9 : Scan code-barres opérationnel
```

**Semaine 10 : OCR Factures**

```
Lundi-Mercredi : Backend OCR
├─ Tech : Tesseract ou Google Vision
├─ Parsing : Détection articles + prix
├─ Mapping : Articles → inventaire
└─ Tests : Différents formats factures

Jeudi-Vendredi : UI OCR
├─ Upload/photo facture
├─ Prévisualisation extraction
├─ Validation ligne par ligne
└─ Import batch

🎯 Résultat S10 : OCR factures automatique
```

**Semaine 11 : Alertes Géolocalisées**

```
Lundi-Mercredi : Backend Géolocalisation
├─ Geofencing zones (producteurs, magasins)
├─ Détection proximité
├─ Génération alertes contextuelles
└─ Respect vie privée (opt-in)

Jeudi-Vendredi : UI Alertes Géo
├─ Configuration zones
├─ Push notifications
├─ Tests terrain
└─ Paramètres confidentialité

🎯 Résultat S11 : Alertes géolocalisées actives
```

**Semaine 12 : IA Conversationnelle**

```
Lundi-Mercredi : Backend IA Chat
├─ Mistral AI conversationnel
├─ Context window
├─ Commandes cuisine
└─ Intégration Session Active

Jeudi-Vendredi : UI Chat
├─ Interface chat
├─ Mode vocal (Web Speech API)
├─ Commandes wake word
└─ Tests utilisateurs

🎯 Résultat S12 : Assistant conversationnel opérationnel
```

**Semaine 13+ : Expansions**

```
- Import YouTube recettes
- Mode mains libres complet
- App mobile native (3 mois projet dédié)
- Intégrations tierces
- Communauté & partage
```

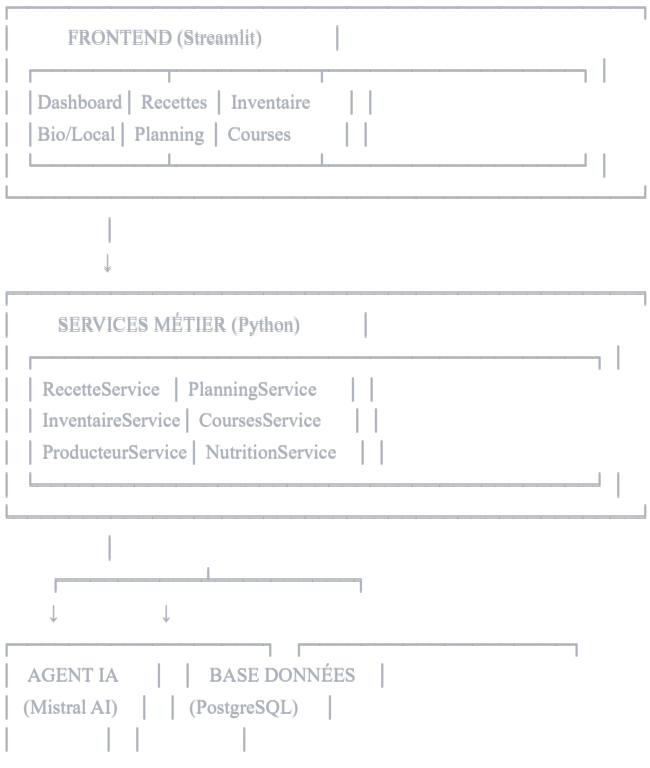
### ✅ Livrables Phase 3

- [ ] Scan code-barres inventaire
- [ ] OCR factures automatique
- [ ] Alertes géolocalisées
- [ ] IA conversationnelle cuisine
- [ ] Import YouTube recettes
- [ ] Mode mains libres
- [ ] App mobile native
- [ ] Intégrations tierces
- [ ] Communauté & partage

---

## 🎨 Spécifications Techniques

**Architecture Cible**



```
│ • Génération  │  │ • Recettes    │
│ • Prédiction  │  │ • Inventaire  │
│ • Optimisation│  │ • Planning    │
│ • Chat        │  │ • Producteurs │
└──────────────────┘ │ • Nutrition   │
                     └──────────────┘
```

---

### Stack Technique Complète

#### Backend

```yaml
Langage: Python 3.11+
Framework Web: Streamlit 1.30+
ORM: SQLAlchemy 2.0+
Validation: Pydantic 2.0+
Database: PostgreSQL 15+ (Supabase)
Migration: Alembic
Cache: Redis (optionnel)
```

#### IA & ML

```yaml
LLM: Mistral AI (API)
  - Modèle: mistral-small-latest
  - Temperature: 0.7
  - Max tokens: 500-2500

OCR: Tesseract / Google Vision API
Speech: Web Speech API
Nutrition: Open Food Facts API
Products: Open Food Facts API
```

#### Frontend

```yaml
UI: Streamlit components
Charts: Plotly / Recharts
Icons: Lucide / Emoji
Styling: Custom CSS + Streamlit theming
Responsive: Mobile-first design
```

#### Infrastructure

```yaml
Hosting: Streamlit Cloud
VCS: GitHub
CI/CD: GitHub Actions
Secrets: Streamlit Secrets / .env
Monitoring: Streamlit logs + custom
Backup: Supabase automated
```

---

### Standards de Code

#### Structure Fichier Python

```python
```

```python
"""
Nom Module - Description courte
Description longue optionnelle
"""
import standard_libs
import third_party_libs

from src.core import core_modules
from src.services import services

# Constants
CONSTANTE = "valeur"

# Pydantic models (si applicable)
class ModeleName(BaseModel):
    """Documentation"""
    field: type

# Service class
class ServiceName:
    """Documentation"""

    def __init__(self):
        """Initialize"""
        pass

    def public_method(self) -> ReturnType:
        """
        Documentation claire

        Args:
            param: Description

        Returns:
            Description retour
        """
        pass

    def _private_method(self):
        """Méthode interne"""
        pass

# Factory (si applicable)
def create_service() -> ServiceName:
    """Factory pattern"""
    return ServiceName()

# Instance globale (si applicable)
service_name = ServiceName()
```

### Conventions Nommage

```python
```

```python
# Variables & fonctions : snake_case
ma_variable = 42
def ma_fonction():
    pass

# Classes : PascalCase
class MaClasse:
    pass

# Constantes : UPPER_SNAKE_CASE
MA_CONSTANTE = "valeur"

# Privé : _prefixe
def _fonction_privee():
    pass

# Modèles DB : Français, PascalCase
class Recette(Base):
    pass
```

**Type Hints Obligatoires**

```python
# ✅ BON
def calculer(a: int, b: int) -> int:
    return a + b

# ❌ MAUVAIS
def calculer(a, b):
    return a + b
```

**Docstrings Google Style**

```python
def fonction_exemple(param1: str, param2: int) -> bool:
    """
    Description courte sur une ligne.

    Description longue sur plusieurs lignes si nécessaire.
    Explique le comportement en détail.

    Args:
        param1: Description du paramètre 1
        param2: Description du paramètre 2

    Returns:
        Description du retour

    Raises:
        ValueError: Si param2 < 0

    Examples:
        >>> fonction_exemple("test", 42)
        True
    """
    if param2 < 0:
        raise ValueError("param2 doit être positif")
    return True
```

---

**Gestion Erreurs**

**Try-Catch Patterns**

```python
# ✅ BON : Spécifique et loggé
try:
    result = operation_risquee()
except ValueError as e:
    logger.error(f"Erreur validation: {e}")
    return None
except DatabaseError as e:
    logger.exception("Erreur DB critique")
    raise

# ❌ MAUVAIS : Catch all générique
try:
    result = operation()
except:
    pass  # Silence errors
```

### Logging Standards

```python
import logging

logger = logging.getLogger(__name__)

# Niveaux appropriés
logger.debug("Info détaillée debug")
logger.info("Action importante réussie")
logger.warning("Situation anormale mais gérée")
logger.error("Erreur nécessitant attention")
logger.exception("Erreur avec traceback complet")
```

---

## Tests

### Structure Tests

```
tests/
├── unit/
│   ├── test_services.py
│   ├── test_models.py
│   └── test_utils.py
├── integration/
│   ├── test_api.py
│   └── test_database.py
└── e2e/
    └── test_workflows.py
```

### Tests Unitaires Pattern

```python
```

```python
import pytest
from src.services.recette_service import recette_service

class TestRecetteService:
    """Tests RecetteService"""

    def setup_method(self):
        """Setup avant chaque test"""
        self.service = recette_service

    def test_create_recette_success(self):
        """Test création recette valide"""
        data = {
            "nom": "Test",
            "temps_preparation": 10,
            # ...
        }
        result = self.service.create(data)
        assert result.nom == "Test"

    def test_create_recette_invalid_fails(self):
        """Test création recette invalide échoue"""
        with pytest.raises(ValueError):
            self.service.create({})
```

**Couverture Minimale**

- Services métier : 80%+
- Modèles : 90%+
- Utils : 95%+
- UI : Tests E2E prioritaires

---

**Performance**

**Guidelines Performances**

**Base de Données**

```python
# ✅ BON : Eager loading
recettes = db.query(Recette).options(
    joinedload(Recette.ingredients),
    joinedload(Recette.etapes)
).all()

# ❌ MAUVAIS : N+1 queries
recettes = db.query(Recette).all()
for r in recettes:
    print(r.ingredients)  # Query par recette !
```

**Cache**

```python
# ✅ BON : Cache avec TTL
result = StateManager.cache_get("key", ttl=3600)
if not result:
    result = calcul_couteux()
    StateManager.cache_set("key", result)

# ❌ MAUVAIS : Recalcul systématique
result = calcul_couteux()  # À chaque fois !
```

### IA

```python
# ✅ BON : Cache + rate limiting
can_call, error = RateLimiter.can_call()
if not can_call:
    raise ValueError(error)

cached = AICache.get(prompt, params)
if cached:
    return cached

result = await ai_call(prompt)
AICache.set(prompt, params, result)

# ❌ MAUVAIS : Appels IA non optimisés
result = await ai_call(prompt)  # Toujours appeler !
```

---

### Sécurité

#### Secrets Management

```python
# ✅ BON : Secrets Streamlit
api_key = st.secrets["mistral"]["api_key"]

# ❌ MAUVAIS : Hardcodé
api_key = "abc123"  # JAMAIS !
```

#### SQL Injection Protection

```python
# ✅ BON : ORM paramétrisé
recette = db.query(Recette).filter(
    Recette.nom == user_input
).first()

# ❌ MAUVAIS : String formatting
query = f"SELECT * FROM recettes WHERE nom = '{user_input}'"
```

#### Input Validation

```python
# ✅ BON : Pydantic validation
from pydantic import BaseModel, validator

class RecetteInput(BaseModel):
    nom: str
    temps_preparation: int

    @validator('nom')
    def clean_nom(cls, v):
        return v.strip()[:200]  # Sanitize

# Usage
validated = RecetteInput(**user_data)
```

---

### Documentation

#### README Structure

```markdown
# Nom Projet

Description courte

## 🚀 Quick Start

## 📋 Prérequis

## 🛠️ Installation

## ⚙️ Configuration

## 📖 Usage

## 🧪 Tests

## 🤝 Contributing

## 📄 License
```

#### Inline Comments

```python
# ✅ BON : Explique le POURQUOI
# Utilise eager loading pour éviter N+1 queries
recettes = db.query(Recette).options(joinedload(...))

# ❌ MAUVAIS : Explique le QUOI (évident)
# Récupère les recettes
recettes = db.query(Recette).all()
```

---

## 🎯 Métriques de Succès

#### KPIs Techniques

```yaml
Performance:
  - Temps chargement page < 2s
  - Temps génération IA < 10s
  - Uptime > 99.5%

Qualité:
  - Code coverage > 80%
  - 0 erreurs critiques production
  - < 5 bugs mineurs / semaine

Adoption:
  - Utilisateurs actifs hebdo
  - Taux rétention 30j > 70%
  - NPS (Net Promoter Score) > 50
```

#### KPIs Fonctionnels

```yaml
```

```
Batch Cooking:
  - Sessions/semaine : 2-3
  - Gain temps : 7h/semaine
  - Taux succès : 95%+

Bio & Local:
  - Score bio moyen : 80%+
  - Score local moyen : 60%+
  - Économies : 125€/mois

Jules (Bébé):
  - % bio : 98%+
  - % maison : 100%
  - Diversification suivie

Gaspillage:
  - 0kg gaspillage > 21j
  - Alertes suivies : 90%+
```

---

## 📞 Support & Maintenance

### Canaux Support

- **GitHub Issues** : Bugs, features requests
- **Discussions** : Questions, aide
- **Email** : Support direct (optionnel)

### Release Process

```yaml
1. Développement:
  - Feature branches
  - Pull requests
  - Code review

2. Testing:
  - Tests automatisés (CI)
  - Tests manuels
  - QA validation

3. Staging:
  - Déploiement pre-prod
  - Tests utilisateurs
  - Validation finale

4. Production:
  - Déploiement Streamlit Cloud
  - Monitoring 24h
  - Rollback si problème

5. Documentation:
  - CHANGELOG mis à jour
  - Release notes
  - User notifications
```

### Versioning

```
vMAJOR.MINOR.PATCH

v2.1.0 → Phase 1 complete
v2.2.0 → Phase 2 complete
```

```
v3.0.0 → Phase 3 complete (breaking changes)

Format release notes:
## v2.1.0 (2025-02-15)

### 🎉 Nouveautés
- Tags bio/local recettes
- Dashboard impact

### 🐛 Corrections
- Fix cache IA
- Fix navigation

### ⚠️ Breaking Changes
- Aucun
```

---

## 🎓 Ressources

### Documentation Technique

- Streamlit Docs
- SQLAlchemy Docs
- Pydantic Docs
- Mistral AI Docs

### Références Externes

- Open Food Facts API
- Google Vision OCR
- Web Speech API
- React Native / Flutter

### Guides Internes

```
docs/
├── setup.md          # Installation
├── architecture.md   # Architecture détaillée
├── api.md            # API reference
├── services.md       # Services guide
└── contributing.md   # Guide contribution
```

---

## ✅ Checklist Implémentation

### Phase 1 (Semaines 1-3)

- [ ] Tags recettes (bio, local, robots, scores)
- [ ] Configuration batch personnalisée
- [ ] Module producteurs (CRUD complet)
- [ ] Alertes rachats IA prédictives
- [ ] Dashboard bio/local avec graphiques
- [ ] Tests + Documentation

### Phase 2 (Semaines 4-8)

- [ ] Planning IA optimisé (tous critères)
- [ ] Templates planning sauvegardables
- [ ] Nutrition complète recettes
- [ ] Mode Session Active batch

- [ ] Courses multi-magasins
- [ ] Prix références + historique
- [ ] Tests + Documentation

**Phase 3 (Semaines 9-12+)**

- [ ] Scan code-barres inventaire
- [ ] OCR factures automatique
- [ ] Alertes géolocalisées
- [ ] IA conversationnelle
- [ ] Import YouTube recettes
- [ ] Mode mains libres complet
- [ ] App mobile native
- [ ] Intégrations tierces
- [ ] Communauté & partage
- [ ] Tests + Documentation

---

## 🏁 Conclusion

### Vision Finale

MaTanne v2 sera **l'assistant familial de référence** pour les familles soucieuses de :

- ✅ **Qualité** : Bio, local, saison
- ✅ **Efficacité** : Batch cooking, robots
- ✅ **Santé** : Nutrition, bébé
- ✅ **Économie** : Budget optimisé
- ✅ **Impact** : Écologique et social

### Prochaines Étapes Immédiates

1. **Validation roadmap** avec utilisateurs
2. **Setup environnement** Phase 1
3. **Sprint 1** : Tags recettes (Semaine 1)
4. **Itération continue** selon feedback

---

*Document vivant - Dernière mise à jour : 2025-01-20 Version : 2.0.0*

## 📑 Table des Matières



---

## 🎯 Vue d'Ensemble

### Mission

MaTanne est un assistant familial intelligent qui aide à gérer le quotidien : cuisine, courses, planning, famille et maison, avec une forte orientation **bio, local et batch cooking**.

**Stack Technique**

- **Frontend** : Streamlit (Python)
- **Backend** : Python 3.11+
- **Base de données** : PostgreSQL (Supabase)
- **IA** : Mistral AI (API)
- **Déploiement** : Streamlit Cloud + GitHub
- **Architecture** : Modulaire avec services dédiés

**Caractéristiques Uniques**

- ✅ Génération IA de recettes, plannings et listes de courses
- ✅ Gestion multi-formats (bébé, batch cooking, robots cuisine)
- ✅ Tracking bio/local avec dashboard impact
- ✅ Import web automatique (Marmiton, 750g, etc.)
- ✅ Système de cache intelligent pour optimiser les appels IA

---

## 🧬 Architecture Existante

**Structure des Répertoires**

```
src/
├── app.py                    # Application principale Streamlit
├── core/                     # Cœur de l'application
│   ├── config.py             # Configuration (secrets, env)
│   ├── database.py           # Connexion PostgreSQL
│   ├── models.py             # Modèles SQLAlchemy (français)
│   ├── ai_agent.py           # Agent IA Mistral
│   ├── ai_cache.py           # Cache + Rate limiting IA
│   ├── state_manager.py      # Gestion état centralisée
│   ├── validators.py         # Validation Pydantic
│   └── base_service.py       # Service CRUD générique
│
├── modules/                  # Modules fonctionnels
│   ├── accueil.py            # Dashboard principal
│   └── cuisine/              # Module Cuisine
│       ├── recettes.py       # Gestion recettes
│       ├── inventaire.py     # Gestion stock
│       ├── courses.py        # Listes de courses
│       └── planning_semaine.py  # Planning hebdomadaire
│
├── services/                 # Services métier
│   ├── recette_service.py        # CRUD recettes
│   ├── recette_edition_service.py  # Édition recettes
│   ├── recette_version_service.py  # Versions (bébé, batch)
│   ├── ai_recette_service.py     # Génération IA recettes
│   ├── web_scraper.py            # Import web (Marmiton...)
│   ├── import_export.py          # Import/Export formats
│   ├── inventaire/               # Services inventaire
│   │   ├── inventaire_service.py
│   │   ├── inventaire_ai_service.py
│   │   └── inventaire_io_service.py
│   ├── courses/                  # Services courses
│   │   ├── courses_service.py
│   │   └── courses_ai_service.py
│   └── planning/                 # Services planning
│       ├── planning_service.py
│       ├── planning_generation_service.py
│       └── repas_service.py
│
```

```
├── ui/                     # Composants UI réutilisables
│   ├── components.py           # Cartes, badges, filtres...
│   └── recette_components.py   # Composants spécifiques recettes
│
└── utils/                  # Utilitaires
    ├── formatters.py           # Formatage quantités, prix...
    └── __init__.py
```

**Base de Données (Modèles Principaux)**

🍳 **Cuisine**

- **Ingredient** : Ingrédients de base
- **Recette** : Recettes complètes avec métadonnées
- **RecetteIngredient** : Relation recette ↔ ingrédient
- **EtapeRecette** : Étapes de préparation ordonnées
- **VersionRecette** : Versions alternatives (bébé, batch)

📦 **Inventaire & Courses**

- **ArticleInventaire** : Stock avec alertes
- **ArticleCourses** : Liste de courses avec priorités

🗓️ **Planning**

- **PlanningHebdomadaire** : Planning semaine
- **RepasPlanning** : Repas individuels
- **ConfigPlanningUtilisateur** : Configuration personnalisée

👶 **Famille**

- **ProfilEnfant** : Profils enfants (Jules, etc.)
- **EntreeBienEtre** : Suivi quotidien
- **Routine** : Routines et tâches

**Services Actuels**

⚙️ **Core Services**

- **AgentIA** : Interface unifiée Mistral AI
- **AICache** : Cache réponses IA + TTL
- **RateLimiter** : Limite appels (30/h, 100/j)
- **StateManager** : État centralisé (navigation, cache, notifications)
- **BaseService<T>** : CRUD générique réutilisable

🍳 **Services Cuisine**

- **RecetteService** : CRUD + recherche avancée
- **AIRecetteService** : Génération IA avec parsing Pydantic
- **RecetteEditionService** : Modification + duplication
- **RecetteVersionService** : Génération versions (bébé, batch)
- **RecipeWebScraper** : Import Marmiton, 750g, Cuisine AZ

📦 **Services Inventaire**

- **InventaireService** : CRUD + alertes automatiques
- **InventaireAIService** : Détection gaspillage, suggestions recettes

### 🛒 Services Courses

- **CoursesService** : CRUD + génération depuis stock/repas
- **CoursesAIService** : Optimisation IA (budget, rayons, alternatives)

### 🗓️ Services Planning

- **PlanningService** : CRUD planning hebdomadaire
- **PlanningGenerationService** : Génération IA complète
- **RepasService** : Manipulation repas (ajout, déplacement, échange)

## Fonctionnalités Clés Actuelles

### ✨ Génération IA

- ✅ **Recettes** : Génération selon critères (saison, type repas, ingrédients)
- ✅ **Planning** : Planning semaine complet avec équilibrage
- ✅ **Courses** : Liste optimisée par rayons et magasins
- ✅ **Versions** : Adaptation bébé et batch cooking automatique
- ✅ **Cache intelligent** : Évite appels IA redondants

### 📚 Recettes

- ✅ CRUD complet avec relations (ingrédients, étapes, versions)
- ✅ Recherche avancée (multi-critères, filtres)
- ✅ Import web (Marmiton, 750g, Cuisine AZ)
- ✅ Import/Export (JSON, Markdown, CSV)
- ✅ Tags : rapide, équilibré, bébé, batch, congélation
- ✅ Images automatiques (Unsplash)
- ✅ Duplication recettes
- ✅ Versions alternatives (bébé 6-18 mois, batch cooking)

### 📦 Inventaire

- ✅ Gestion stock avec alertes (stock bas, péremption)
- ✅ Catégories et emplacements
- ✅ Calcul statut automatique (ok, sous_seuil, critique, peremption_proche)
- ✅ Ajout courses depuis stock bas
- ✅ Vérification faisabilité recettes
- ✅ Import/Export CSV
- ✅ Détection gaspillage IA
- ✅ Suggestions recettes selon stock

### 🛒 Courses

- ✅ Liste active avec priorités (haute, moyenne, basse)
- ✅ Génération automatique (stock bas + repas planifiés)
- ✅ Organisation par rayons et magasins
- ✅ Optimisation IA (budget, alternatives, conseils)
- ✅ Marquage achat avec option ajout inventaire
- ✅ Historique achats
- ✅ Statistiques et top articles

### 🗓️ Planning Semaine

- ✅ Planning hebdomadaire avec types repas configurables
- ✅ Génération IA complète (équilibrage, variété)
- ✅ Configuration foyer (adultes, enfants, bébé)
- ✅ Mode batch cooking (jours personnalisables)
- ✅ Mode bébé (adaptations automatiques)
- ✅ Ajout/modification/suppression repas manuel
- ✅ Déplacement et échange repas
- ✅ Modal batch cooking (préparation groupée)

### 🎨 UI/UX

- ✅ Navigation centralisée avec breadcrumb
- ✅ Composants réutilisables (cartes, badges, filtres)
- ✅ Pagination et recherche
- ✅ Mode debug intégré
- ✅ Notifications/toasts
- ✅ Stats et métriques visuelles
- ✅ Empty states avec actions

---

## 🆕 Nouvelles Fonctionnalités

### 🎯 Phase 1 : FONDATIONS (Semaines 1-3)

#### 1.1 Tags & Métadonnées Recettes 🔥

**Objectif** : Enrichir les recettes avec tags bio, local, robots

**Modifications Base de Données**

```python
# Ajouter colonnes à Recette
est_bio: bool = False
est_local: bool = False
compatible_cookeo: bool = False
compatible_monsieur_cuisine: bool = False
compatible_airfryer: bool = False
score_bio: int = 0  # 0-100
score_local: int = 0  # 0-100
producteurs: List[str] = []  # Liste producteurs utilisés
```

**UI**

- Filtres bio/local/robots dans recherche recettes
- Badges visuels sur cartes recettes
- Score bio/local affiché (0-100%)
- Icônes robots (Cookeo, MC, Airfryer)

**Services**

- Calcul automatique scores bio/local selon ingrédients
- Validation compatibilité robots selon type cuisson

---

### 1.2 Configuration Batch Personnalisée 🔥

**Objectif** : Configurer finement le batch cooking

**Modifications Base de Données**

```python
# Ajouter à ConfigPlanningUtilisateur
batch_preferences: Dict = {
    "portions_default": 8,  # Portions par défaut
    "jours_semaine": [0, 6],  # Lundi, Dimanche
    "types_privilegies": ["plats", "accompagnements"],
    "robots_utilises": ["cookeo", "airfryer"],
    "congelation_active": True,
    "temps_max_session": 180  # 3h max par session
}
```

**UI**

- Widget configuration batch dans paramètres
- Sélection jours de la semaine (calendrier visuel)
- Choix robots à privilégier
- Slider portions par défaut
- Toggle congélation

**Services**

- Génération planning IA adapté selon config batch
- Validation temps total session
- Suggestions optimisation selon robots

---

### 1.3 Module Producteurs 🔥

**Objectif** : Carnet d'adresses producteurs locaux

**Nouvelle Table**

```python
```

```
class Producteur(Base):
    __tablename__ = "producteurs"

    id: int
    nom: str  # "Ferme Martin"
    type: str  # "fermier", "maraîcher", "fromager", "boucher"
    adresse: str
    telephone: str
    email: Optional[str]
    site_web: Optional[str]

    # Horaires
    jours_ouverture: Dict  # {"lundi": "8h-12h", ...}

    # Produits
    categories_produits: List[str]  # ["œufs", "volaille", "légumes"]

    # Logistique
    distance_km: float
    temps_trajet_min: int
    panier_moyen_euro: float

    # Tracking
    dernier_achat: date
    nb_achats_total: int
    montant_total: float

    # Préférences
    favoris: bool
    notes: str
    tags: List[str]  # ["bio", "amap", "marché"]

    cree_le: datetime
```

**UI Module Producteurs**

- Liste producteurs avec filtres (type, distance, favoris)
- Fiche détaillée producteur
  - Infos contact
  - Horaires visualisés
  - Historique achats
  - Carte localisation (optionnel)
- Ajout/édition producteur
- Import/export carnet adresses

**Intégrations**

- Lien courses → producteurs (suggestion selon liste)
- Lien recettes → producteurs (ingrédients locaux)
- Rappels achat récurrent (AMAP, colis viande)

---

**1.4 Alertes Rachats IA Prédictives** 🔥

**Objectif** : Prédire quand racheter selon consommation

**Service IA**

python

```
async def predire_consommation(
    article: Dict,
    historique: List[Dict]  # Historique achats/consommation
) -> Dict:
    """
    Prédit :
    - Jours avant épuisement
    - Date rachat suggérée
    - Quantité optimale
    - Producteur/magasin recommandé
    """
```

**UI Alertes**

- Section "À racheter cette semaine" (dashboard)
- Tri par urgence (URGENT <3j, BIENTÔT <7j, PRÉVENTIF >7j)
- Icônes visuelles ( 🔴 urgent, 🟠 bientôt, 🟡 préventif)
- Suggestion producteur + lien ajout courses
- Notification push (optionnel)

**Algorithme**

1. Analyser historique consommation (30 derniers jours)
2. Détecter patterns (ex: lait 200mL/jour pour Jules)
3. Calculer stock restant en jours
4. Suggérer rachat optimal (quantité + timing)

---

**1.5 Dashboard Bio & Local** 🔥

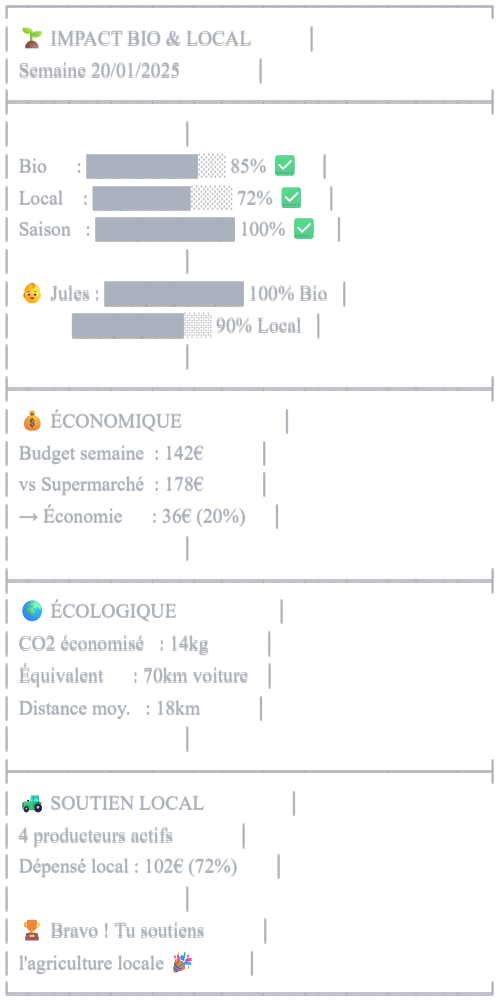
**Objectif** : Visualiser impact bio/local en temps réel

**Métriques Calculées**

python

```
def calculer_dashboard_bio_local(
    recettes_semaine: List[Recette],
    courses: List[Article]
) -> Dict:
    return {
        "bio_pct": 85,  # % articles bio
        "local_pct": 72,  # % articles locaux
        "saison_pct": 100,  # % saison
        "jules_bio_pct": 100,  # % bio pour Jules
        "jules_local_pct": 90,
        "economie_vs_supermarche": 36,  # Euros
        "co2_economise_kg": 14,
        "distance_moy_km": 18,
        "nb_producteurs": 4,
        "depense_locale": 102
    }
```

**UI Dashboard**




**Rapport Mensuel Automatique**

- Email/PDF récapitulatif fin de mois
- Comparaison mois précédent
- Objectifs atteints/manqués
- Suggestions améliorations

---

## 🚀 Phase 2 : OPTIMISATION (Semaines 4-8)

### 2.1 Génération Planning IA Optimisée ⭐

**Objectif** : Intégrer tous les critères (Jules + Robots + Bio/Local)

**Amélioration Prompt IA**

```python
```

```
# Ajouter au prompt existant
ROBOTS DISPONIBLES: Cookeo, Monsieur Cuisine, Airfryer
→ Privilégier recettes compatibles
→ Paralléliser préparations

BIO & LOCAL:
⇒ Minimum 80% bio
⇒ Minimum 60% local
→ Utiliser producteurs préférés

JULES (18 mois):
→ Adapter textures et portions
→ Privilégier versions bébé
⇒ Éviter allergènes
```

**Service**

```python
async def generer_planning_complet(
    semaine_debut: date,
    config: ConfigPlanningUtilisateur,
    contraintes: Dict = {
        "bio_min": 80,
        "local_min": 60,
        "robots": ["cookeo", "airfryer"],
        "bebe": True,
        "batch_actif": True
    }
) -> PlanningGenere
```

---

### 2.2 Templates Planning Sauvegardables ⭐

**Objectif** : Réutiliser plannings efficaces

**Nouvelle Table**

```python
class TemplatePlanning(Base):
    __tablename__ = "templates_planning"

    id: int
    nom: str  # "Planning Hiver Bio"
    description: str

    # Structure
    repas_structure: Dict  # Structure 7 jours
    recettes_ids: List[int]

    # Métadonnées
    saison: str
    score_bio: int
    score_local: int
    compatible_batch: bool
    compatible_bebe: bool

    # Stats utilisation
    nb_utilisations: int
    note_moyenne: float

    utilisateur_id: int
    cree_le: datetime
```

**UI**

- Bouton "Sauvegarder comme template" sur planning validé
- Galerie templates personnels
- Filtres (saison, bio, batch, bébé)
- Bouton "Appliquer template" → génère planning depuis template
- Partage templates (optionnel)

---

**2.3 Nutrition Complète Recettes** ⭐

**Objectif** : Tracking nutrition détaillé

**Modifications Base de Données**

```python
# Ajouter à Recette
nutrition: Dict = {
    "calories": 450,
    "proteines_g": 28,
    "glucides_g": 45,
    "lipides_g": 12,
    "fibres_g": 8,
    "sel_g": 1.2,
    "vitamines": {"A": 25, "C": 80},  # % AJR
    "mineraux": {"fer": 15, "calcium": 30}
}

# Nouveau modèle
class NutritionJour(Base):
    __tablename__ = "nutrition_jour"

    date: date
    utilisateur_id: int

    # Objectifs
    calories_objectif: int = 2000
    proteines_objectif: int = 50

    # Réalisé
    calories_total: int
    proteines_total: int
    glucides_total: int
    lipides_total: int

    # Par repas
    repas_details: Dict
```

**UI**

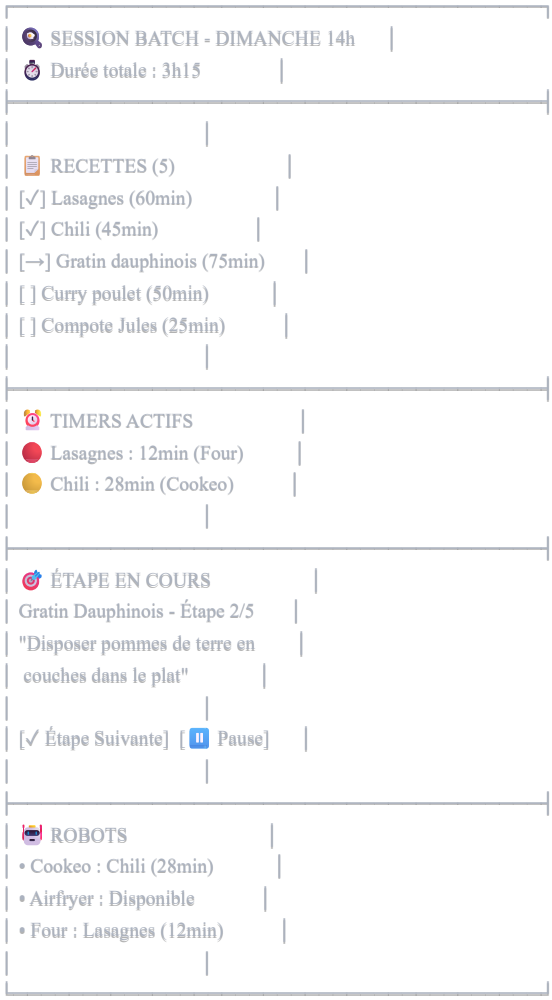
- Affichage nutrition sur fiche recette
- Dashboard nutrition journalier (graphiques)
- Alertes déséquilibres
- Suggestions IA équilibrage

---

**2.4 Mode "Session Active" Batch** ⭐

**Objectif** : Guider pendant la session batch

**UI Session Active**




**Fonctionnalités**

- Checklist recettes temps réel
- Timers multiples synchronisés
- Étapes guidées une par une
- Vue robots (disponible/occupé)
- Mode vocal (optionnel : "OK MaTanne, étape suivante")
- Alertes sonores fin timers
- Historique session (temps réel vs estimé)

---

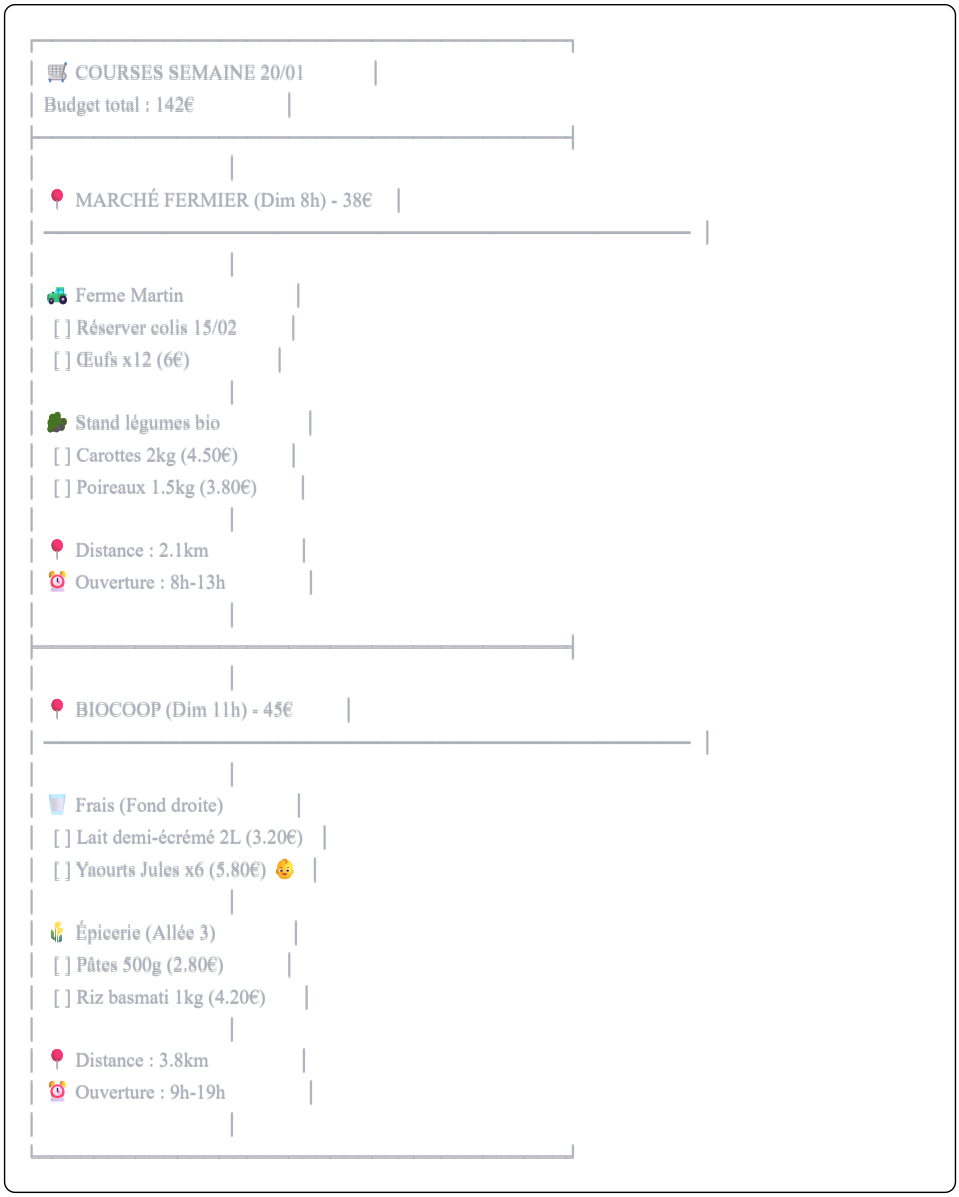
**2.5 Liste Courses Multi-Magasins** ⭐

**Objectif** : Organiser courses par magasin

**Service**

```python
def generer_liste_multi_magasins(
    articles: List[Article],
    magasins_preferences: List[str]
) -> Dict[str, List[Article]]:
    return {
        "Marché Fermier": [...],
        "Biocoop": [...],
        "Grand Frais": [...]
    }
```

**UI**




**Fonctionnalités**

- Organisation automatique par magasin
- Sous-organisation par rayon/producteur
- Affichage distance et horaires
- Calcul trajet optimal
- Budget par magasin
- Mode "Course en cours" (checklist mobile)

---

**2.6 Prix Références & Budget** ⭐

**Objectif** : Tracking prix et optimisation budget

**Nouvelle Table**

python

```python
class PrixReference(Base):
    __tablename__ = "prix_references"

    ingredient_id: int
    magasin: str
    prix_unitaire: float
    unite: str
    date_releve: date

    # Historique
    prix_moyen_30j: float
    prix_min_30j: float
    prix_max_30j: float

    # Alertes
    alerte_hausse: bool
```

**UI**

- Import automatique factures (OCR optionnel)
- Saisie manuelle prix
- Graphique évolution prix ingrédients clés
- Alertes hausse significative
- Suggestions alternatives si prix élevé

---

### 🏆 Phase 3 : EXCELLENCE (Semaines 9-12+)

#### 3.1 Scan Code-Barres Inventaire 🚀

**Objectif** : Ajout rapide articles par scan

**Technologie**

- API Open Food Facts (base produits mondiale)
- Streamlit + JavaScript pour accès caméra
- Fallback saisie manuelle

**Workflow**

1. Scanner code-barres produit
2. Récupération infos (nom, marque, nutrition, bio)
3. Validation/ajustement quantité
4. Ajout inventaire

---

#### 3.2 OCR Factures Automatique 🚀

**Objectif** : Import automatique achats depuis photo facture

**Technologie**

- Tesseract OCR ou API (Google Vision, AWS Textract)
- Parsing intelligent lignes factures
- Mapping articles inventaire

**Workflow**

1. Photo/upload facture
2. OCR extraction texte

3. Détection articles + prix
4. Validation utilisateur
5. Import inventaire + historique prix

---

**3.3 Alertes Géolocalisées** 🚀

**Objectif** : Notifications selon position

**Exemples**

- "Tu passes devant Biocoop, besoin de lait ?"
- "Marché fermier ouvert, ton panier AMAP est prêt"
- "Ferme Martin à 500m, colis viande disponible"

**Implémentation**

- Geofencing (web ou app mobile)
- Push notifications
- Respect vie privée (opt-in, données locales)

---

**3.4 IA Conversationnelle Cuisine** 🚀

**Objectif** : Assistant vocal/chat temps réel

**Exemples Interactions**

```
User: "Comment je fais les lasagnes ?"
IA: "Voici la recette. Tu veux que je te guide étape par étape ?"

User: "Oui"
IA: "Étape 1: Faire revenir la viande hachée. Dis-moi quand c'est prêt."

User: "C'est prêt"
IA: "Parfait ! Étape 2: Ajouter les tomates concassées..."
```

**Technologie**

- Mistral AI conversationnel
- Context window (historique conversation)
- Intégration mode "Session Active"

---

**3.5 Import YouTube Recettes** 🚀

**Objectif** : Convertir vidéos YouTube en recettes

**Workflow**

1. URL vidéo YouTube
2. Extraction transcription (YouTube API)
3. Parsing IA ingrédients + étapes
4. Génération recette structurée
5. Validation/édition utilisateur

---