

MY MEAL PLANNER

RAPPORT

Planification Intelligente de Repas

Plateforme Full-Stack de Génération de Menus

React Native • Expo • Firebase • React Web

Réalisé par : Ayoub Elmardi , Zayd Bentalha , Najib Ennaji

Année académique : 2025-2026

Date : 7 Janvier 2026

Rapport de Projet - Application Mobile et Web

Table des matières

1	Introduction	3
1.1	Contexte du Projet	3
1.1.1	Problématique	3
1.1.2	Solution Proposée	3
1.2	Objectifs du Projet	4
1.2.1	Objectifs Principaux	4
1.2.2	Objectifs Techniques	4
2	Architecture Technique	5
2.1	Vue d'Ensemble de l'Architecture	5
2.2	Stack Technologique Détailnée	5
2.2.1	Frontend Mobile (React Native / Expo)	5
2.2.2	Frontend Web (React + Vite)	6
2.2.3	Backend et Services Cloud (Firebase)	7
2.3	Gestion de l'État et du Stockage	9
2.3.1	Stockage Local	9
2.3.2	Stratégie de Synchronisation	9
3	Fonctionnalités Détaiillées	10
3.1	Génération Intelligente de Menus	10
3.1.1	Algorithme de Génération	10
3.1.2	Types de Régimes Supportés	11
3.1.3	Options de Personnalisation	11
3.2	Base de Recettes Complète	11
3.2.1	Statistiques de la Base de Données	11
3.2.2	Structure d'une Recette	11
3.2.3	Répartition des Recettes	12
3.3	Gestion des Listes de Courses	13
3.3.1	Génération Automatique	13
3.3.2	Fonctionnalités Avancées	13
3.4	Synchronisation et Mode Hors Ligne	14
3.4.1	Architecture de Synchronisation	14
3.4.2	Gestion des Conflits	14
3.5	Authentification et Gestion des Utilisateurs	14
3.5.1	Méthodes d'Authentification	14
3.5.2	Profil Utilisateur	15

4 Interface Utilisateur et Expérience	16
4.1 Design System et Théming	16
4.1.1 Palette de Couleurs	16
4.1.2 Support du Mode Sombre	16
4.1.3 Typographie	17
4.2 Navigation et Architecture des Écrans	17
4.2.1 Navigation Principale (Mobile)	17
4.2.2 Flux Utilisateur Principal	17
4.3 Composants UI Réutilisables	18
4.3.1 MenuCard Component	18
4.3.2 RecipeCard Component	19
4.3.3 LoadingSpinner Component	20
4.4 Animations et Interactions	21
4.4.1 Animations Principales	21
4.4.2 Gestes Supportés	21
4.5 Accessibilité	21
4.5.1 Normes WCAG 2.1 Respectées	21
5 Performance et Optimisation	22
5.1 Stratégies d'Optimisation	22
5.1.1 Optimisation du Rendu	22
5.1.2 Gestion des Images	22
5.1.3 Optimisation des Listes	22
5.2 Métriques de Performance	23
5.2.1 Temps de Chargement	23
5.2.2 Utilisation de la Mémoire	23
5.3 Optimisation du Bundle	24
5.3.1 Taille de l'Application	24
5.3.2 Stratégies de Réduction	24
6 Tests et Assurance Qualité	25
6.1 Stratégie de Test	25
6.1.1 Pyramide de Tests	25
6.1.2 Outils de Test	25
6.2 Tests Unitaires	25
6.2.1 Exemple : Test du Générateur de Menus	25
6.2.2 Exemple : Test de la Liste de Courses	26
6.3 Tests d'Intégration	27
6.3.1 Test de Synchronisation Firebase	27
6.4 Tests E2E	28
6.4.1 Test du Parcours Complet de Génération	28
6.5 Couverture de Code	29
6.5.1 Objectifs de Couverture	29
6.6 Qualité du Code	29
6.6.1 Outils de Linting	29
6.6.2 Configuration ESLint	30

Chapitre 1

Introduction

1.1 Contexte du Projet

L'application **MealPlanner** représente une solution innovante dans le domaine de la planification nutritionnelle et de la gestion des repas. Dans un contexte où la nutrition consciente et l'optimisation budgétaire sont devenues des préoccupations majeures, cette application offre une réponse technologique moderne et accessible.

1.1.1 Problématique

Les défis actuels en matière de planification alimentaire incluent :

- **Gestion du temps** : Difficulté à planifier des repas équilibrés au quotidien
- **Contraintes budgétaires** : Nécessité d'optimiser les dépenses alimentaires
- **Besoins nutritionnels variés** : Adapter les menus selon les régimes spécifiques
- **Organisation des courses** : Génération manuelle fastidieuse des listes de courses
- **Manque d'inspiration** : Répétition des mêmes recettes par routine

1.1.2 Solution Proposée

MealPlanner propose une approche holistique de la planification alimentaire en combinant :

Fonctionnalités Clés

- Génération automatique de menus personnalisés
- Gestion intelligente du budget alimentaire
- Adaptation aux régimes spécifiques (végétarien, sportif, diabétique)
- Création automatique de listes de courses optimisées
- Synchronisation multi-plateforme (mobile et web)
- Base de données riche en recettes marocaines et internationales

1.2 Objectifs du Projet

1.2.1 Objectifs Principaux

1. **Simplifier la planification alimentaire** en automatisant le processus de création de menus hebdomadaires
2. **Optimiser les budgets** en proposant des solutions adaptées aux contraintes financières
3. **Promouvoir une alimentation équilibrée** en intégrant des données nutritionnelles
4. **Faciliter l'organisation** en générant automatiquement les listes de courses

1.2.2 Objectifs Techniques

1. Développer une architecture full-stack moderne et scalable
2. Assurer une expérience utilisateur fluide sur mobile et web
3. Implémenter une synchronisation cloud robuste
4. Garantir la performance et la réactivité de l'application
5. Adopter les meilleures pratiques de développement

Chapitre 2

Architecture Technique

2.1 Vue d'Ensemble de l'Architecture

L'application MealPlanner adopte une architecture **full-stack moderne** basée sur trois piliers principaux :

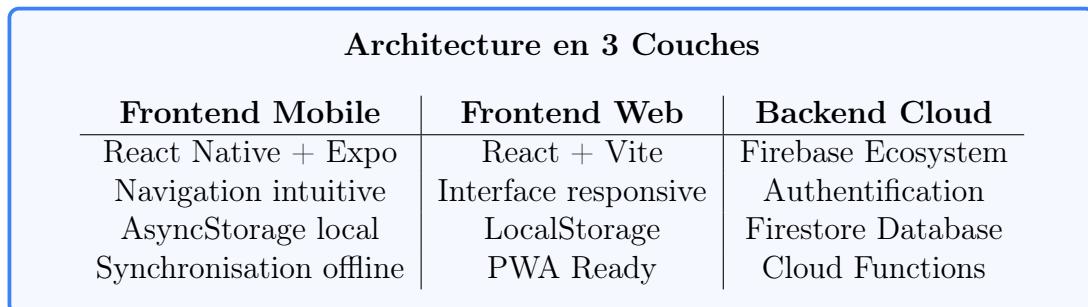


FIGURE 2.1 – Architecture globale de l'application MealPlanner

2.2 Stack Technologique Détaillée

2.2.1 Frontend Mobile (React Native / Expo)

Technologies Principales

- **React Native 0.76.6** - Framework de développement mobile cross-platform
- **Expo SDK 54** - Plateforme de développement avec outils intégrés
- **Expo Router v4** - Navigation file-based moderne et performante
- **React 19.0.0** - Dernière version stable de React

Bibliothèques et Packages

Structure du Projet Mobile

```
1 frontend/app-mobile/
2         app/                               # Pages (File-based routing)
3     )
```

Package	Version	Usage
@react-native-async-storage	2.1.0	Stockage local persistant
expo-linear-gradient	14.0.1	Dégradés visuels
@react-native-community/slider	4.5.5	Contrôles UI interactifs
@react-native-community/netinfo	11.4.1	Détection connectivité
react-native-reanimated	3.16.4	Animations performantes

TABLE 2.1 – Principales dépendances du frontend mobile

```

3      index.jsx          #   cran d'accueil
4      Home.jsx           # Génrateur de menus
5      mes-menus.jsx      # Historique des menus
6      courses.jsx        # Liste de courses
7      mes-recettes.jsx   # Gestion des recettes
8      profil.jsx         # Profil utilisateur
9      Login.jsx          # Authentification
10     Signup.jsx         # Inscription
11     components/
12       MenuCard.jsx     # Carte de menu
13       RecipeCard.jsx   # Carte de recette
14       ShoppingListItem.jsx # lment de liste
15       LoadingSpinner.jsx # Indicateur de chargement
16     services/
17       menuGenerator.js # Génrateur de menus
18       shoppingList.js  # Gestion des courses
19       sync.js           # Synchronisation cloud
20       recipes.js        # Base de recettes
21       firebase.js       # Configuration Firebase
22     constants/
23       Colors.js         # Thème et couleurs
24     assets/
25       images/            # Images locales
26       fonts/             # Polices personnalisées

```

Listing 2.1 – Arborescence du projet mobile

2.2.2 Frontend Web (React + Vite)

Technologies Principales

- **React 18.3.1** - Bibliothèque UI moderne
- **Vite 6.0.5** - Build tool ultra-rapide
- **Tailwind CSS 3.4.17** - Framework CSS utility-first
- **React Router 7.1.1** - Routing côté client

Bibliothèques Complémentaires

- **Recharts 2.15.0** - Visualisation de données et graphiques
- **Lucide React 0.469.0** - Bibliothèque d'icônes moderne
- **ESLint 9.18.0** - Linter pour qualité du code

Structure du Projet Web

```

1  frontend/src/
2      pages/                                # Pages principales
3          Home.jsx                            # Page d'accueil
4          Generator.jsx                      # Générateur de menus
5          MyMenus.jsx                         # Mes menus
6          Shopping.jsx                        # Liste de courses
7          Recipes.jsx                         # Mes recettes
8          Profile.jsx                         # Profil
9          Login.jsx                           # Connexion
10         Register.jsx                       # Inscription
11     components/                            # Composants UI
12         Navbar.jsx                          # Barre de navigation
13         Footer.jsx                          # Pied de page
14         MenuCard.jsx                        # Carte de menu
15         RecipeModal.jsx                   # Modal de recette
16     services/                             # Services partagés
17     styles/                               # Styles globaux
18     assets/                               # Ressources statiques

```

Listing 2.2 – Arborescence du projet web

2.2.3 Backend et Services Cloud (Firebase)

Services Firebase Utilisés

1. Firebase Authentication

- Authentification anonyme pour démarrage rapide
- Support email/password pour comptes permanents
- Gestion sécurisée des sessions utilisateur

2. Cloud Firestore

- Base de données NoSQL temps réel
- Structure de collections : users, menus, recipes, shoppingLists
- Synchronisation automatique multi-appareils
- Mode hors ligne avec cache local

3. Firebase Storage (préparé pour futures évolutions)

- Stockage d'images de recettes
- Export de documents PDF

Structure de la Base de Données Firestore

```

1 {
2     "users": {
3         "userId": {
4             "email": "user@example.com",
5             "displayName": "Utilisateur",
6             "preferences": {

```

```
7     "dietType": "normal",
8     "defaultBudget": 50,
9     "servings": 4
10    },
11    "createdAt": "timestamp",
12    "lastSync": "timestamp"
13  }
14 },
15 "menus": {
16   "menuId": {
17     "userId": "userId",
18     "name": "Menu de la semaine",
19     "days": [
20       {
21         "day": "Lundi",
22         "breakfast": { "recipeId": "...", "cost": 15 },
23         "lunch": { "recipeId": "...", "cost": 30 },
24         "dinner": { "recipeId": "...", "cost": 25 }
25       }
26     ],
27     "totalCost": 350,
28     "totalCalories": 8500,
29     "createdAt": "timestamp"
30   }
31 },
32 "shoppingLists": {
33   "listId": {
34     "menuId": "menuId",
35     "items": [
36       {
37         "ingredient": "Poulet",
38         "quantity": 1.5,
39         "unit": "kg",
40         "checked": false
41       }
42     ]
43   }
44 }
45 }
```

Listing 2.3 – Schéma de données Firestore

Plateforme	Technologie	Données Stockées
Mobile	AsyncStorage	Menus, recettes, préférences, cache offline
Web	LocalStorage	Sessions, préférences UI, cache temporaire

TABLE 2.2 – Technologies de stockage local

2.3 Gestion de l'État et du Stockage

2.3.1 Stockage Local

2.3.2 Stratégie de Synchronisation

Algorithme de Synchronisation

Étapes de synchronisation :

1. Détection de la connectivité réseau
2. Récupération des modifications locales
3. Upload vers Firestore avec gestion de file d'attente
4. Téléchargement des données distantes
5. Résolution des conflits (last-write-wins)
6. Mise à jour du cache local
7. Notification de l'utilisateur

Chapitre 3

Fonctionnalités Détaillées

3.1 Génération Intelligente de Menus

3.1.1 Algorithme de Génération

Le service `menuGenerator.js` implémente un algorithme sophistiqué de génération de menus basé sur plusieurs critères :

```
1 export const generateMenu = (
2   dietType,           // Type de régime
3   budget,            // Budget quotidien
4   days,              // Nombre de jours
5   preferences        // Préférences utilisateur
6 ) => {
7   // Logique de génération
8 }
```

Listing 3.1 – Signature de la fonction de génération

Critères de Sélection des Recettes

1. Filtrage par régime alimentaire

- Correspondance avec les tags de régime (végétarien, économique, etc.)
- Exclusion des allergènes déclarés

2. Respect du budget

- Calcul du coût total journalier (petit-déjeuner + déjeuner + dîner)
- Mode strict : respect exact du budget
- Mode flexible : tolérance de $\pm 10\%$

3. Équilibre nutritionnel

- Distribution calorique : 25% petit-déjeuner, 40% déjeuner, 35% dîner
- Variété des sources de protéines, glucides et lipides

4. Diversité des repas

- Évitement des répétitions sur la période
- Rotation des catégories de recettes
- Alternance des méthodes de cuisson

3.1.2 Types de Régimes Supportés

Régime	Caractéristiques	Budget Moyen/Jour
Normal	Équilibré, varié, toutes catégories	50-70 MAD
Économique	Optimisé coût, ingrédients basiques	30-45 MAD
Végétarien	Sans viande, riche en légumes	40-60 MAD
Sportif	Haute protéine, énergétique	60-90 MAD
Diabétique	Faible index glycémique, contrôlé	55-75 MAD

TABLE 3.1 – Régimes alimentaires disponibles

3.1.3 Options de Personnalisation

- Durée du menu : 3, 5 ou 7 jours
- Nombre de portions : 1 à 8 personnes
- Budget : 30 à 150 MAD par jour
- Préférences :
 - Autoriser les répétitions de recettes
 - Budget strict ou flexible
 - Exclure certains ingrédients
 - Prioriser certaines cuisines (marocaine, italienne, asiatique)

3.2 Base de Recettes Complète

3.2.1 Statistiques de la Base de Données

Chiffres Clés :

- 35 recettes complètes et testées
- 3 catégories principales (petit-déjeuner, déjeuner, dîner)
- 200+ ingrédients uniques répertoriés
- Images HD via Unsplash pour chaque recette
- Données nutritionnelles détaillées (calories, coût)

3.2.2 Structure d'une Recette

```

1 {  

2   id: "recipe_001",  

3   name: "Tajine de Poulet aux L gumes",  


```

```

4   category: "lunch",
5   tags: ["normal", "moroccan"] ,
6   cost: 35,
7   calories: 450,
8   prepTime: 20,
9   cookTime: 45,
10  servings: 4,
11  image: "https://images.unsplash.com/photo-...",
12  ingredients: [
13    {
14      name: "Poulet",
15      quantity: 800,
16      unit: "g"
17    },
18    {
19      name: "Carottes",
20      quantity: 3,
21      unit: "pi ce"
22    }
23    // ...
24  ],
25  instructions: [
26    "D couper le poulet en morceaux",
27    "Faire revenir dans l'huile d'olive",
28    "Ajouter les l gumes et pices "
29    // ...
30  ],
31  nutritionalInfo: {
32    protein: 35,
33    carbs: 25,
34    fat: 15,
35    fiber: 8
36  }
37 }

```

Listing 3.2 – Modèle de données d'une recette

3.2.3 Répartition des Recettes

Catégorie	Nombre	Coût Moyen
Petit-déjeuner	10	12 MAD
Déjeuner	15	32 MAD
Dîner	10	28 MAD
Total	35	24 MAD

TABLE 3.2 – Distribution des recettes par catégorie

3.3 Gestion des Listes de Courses

3.3.1 Génération Automatique

Le service `shoppingList.js` agrège intelligemment tous les ingrédients d'un menu hebdomadaire :

```

1  export const generateShoppingList = (menu) => {
2      const aggregatedItems = {};
3
4      menu.days.forEach(day => {
5          ['breakfast', 'lunch', 'dinner'].forEach(meal => {
6              const recipe = getRecipeById(day[meal].recipeId);
7
8              recipe.ingredients.forEach(ingredient => {
9                  if (aggregatedItems[ingredient.name]) {
10                      // Additionner les quantités
11                      aggregatedItems[ingredient.name].quantity +=
12                          convertToStandardUnit(ingredient);
13                  } else {
14                      // Nouvelle entrée
15                      aggregatedItems[ingredient.name] = {
16                          ...ingredient,
17                          checked: false
18                      };
19                  }
20              });
21          });
22      });
23
24      return Object.values(aggregatedItems);
25  }

```

Listing 3.3 – Algorithme d'agrégation

3.3.2 Fonctionnalités Avancées

- **Agrégation intelligente** : Combinaison des quantités d'un même ingrédient
- **Conversion d'unités** : Standardisation (g, kg, ml, l, pièces)
- **Organisation par catégories** : Légumes, viandes, épicerie, produits frais
- **Mode check** : Cocher les articles achetés
- **Estimation du coût total** : Calcul automatique du budget courses
- **Export** : Partage par message ou email

Flux de Synchronisation

1. **Action utilisateur** → Modification locale
2. **AsyncStorage/LocalStorage** → Sauvegarde immédiate
3. **File d'attente** → Enregistrement si hors ligne
4. **Détection connexion** → Tentative de sync
5. **Upload Firestore** → Envoi des modifications
6. **Download Firestore** → Récupération des données distantes
7. **Merge** → Fusion intelligente
8. **UI Update** → Rafraîchissement de l'interface

FIGURE 3.1 – Processus de synchronisation des données

3.4 Synchronisation et Mode Hors Ligne

3.4.1 Architecture de Synchronisation

3.4.2 Gestion des Conflits

Stratégie adoptée : *Last-Write-Wins (LWW)*

- Chaque modification est horodatée
- En cas de conflit, la version la plus récente est conservée
- Notification utilisateur en cas de conflit détecté
- Possibilité de restaurer des versions antérieures

3.5 Authentification et Gestion des Utilisateurs

3.5.1 Méthodes d'Authentification

1. **Authentification Anonyme**
 - Démarrage instantané sans inscription
 - Données temporaires synchronisées
 - Possibilité de migration vers compte permanent
2. **Authentification Email/Password**
 - Création de compte sécurisé
 - Récupération de mot de passe
 - Persistance des données utilisateur
3. **Future : OAuth (Google, Facebook)**
 - Connexion simplifiée
 - Import des informations de profil

3.5.2 Profil Utilisateur

```
1  {
2    uid: "user_unique_id",
3    email: "user@example.com",
4    displayName: "Ayoub Elmardi",
5    photoURL: "https://...",
6    preferences: {
7      defaultDietType: "normal",
8      defaultBudget: 50,
9      defaultServings: 4,
10     avoidRepetitions: true,
11     strictBudget: false,
12     excludedIngredients: ["shellfish", "peanuts"]
13   },
14   statistics: {
15     totalMenusGenerated: 15,
16     totalRecipesSaved: 42,
17     totalCostSaved: 500
18   },
19   createdAt: "2025-12-15T10:00:00Z",
20   lastLoginAt: "2026-01-06T14:30:00Z"
21 }
```

Listing 3.4 – Structure du profil utilisateur

Chapitre 4

Interface Utilisateur et Expérience

4.1 Design System et Théming

4.1.1 Palette de Couleurs

Couleur	Code Hex	Usage
Primary Blue	#3B82F6	Actions principales, liens
Secondary Purple	#8B5CF6	Accentuation, badges
Success Green	#22C55E	Confirmations, succès
Warning Yellow	#FBBF24	Alertes, avertissements
Error Red	#EF4444	Erreurs, suppressions
Gray Scale	#6B7280 - #F3F4F6	Textes, backgrounds

TABLE 4.1 – Palette de couleurs de l'application

4.1.2 Support du Mode Sombre

L'application implémente un système de théming dynamique avec support complet du mode sombre :

```
1 const Colors = {
2   light: {
3     primary: '#3B82F6',
4     secondary: '#8B5CF6',
5     background: '#FFFFFF',
6     card: '#F9FAFB',
7     text: '#111827',
8     textSecondary: '#6B7280',
9     border: '#E5E7EB',
10    success: '#22C55E',
11    warning: '#FBBF24',
12    error: '#EF4444'
13  },
14  dark: {
15    primary: '#60A5FA',
16    secondary: '#A78BFA',
```

```

17   background: '#111827',
18   card: '#1F2937',
19   text: '#F9FAFB',
20   textSecondary: '#9CA3AF',
21   border: '#374151',
22   success: '#34D399',
23   warning: '#FCD34D',
24   error: '#F87171'
25 }
26 };

```

Listing 4.1 – Configuration du thème dans Colors.js

4.1.3 Typographie

- **Titres** : System Font (San Francisco sur iOS, Roboto sur Android)
- **Corps de texte** : 16px, line-height 1.5
- **Poids** : Regular (400), Medium (500), Bold (700)
- **Hiérarchie** : H1 (32px), H2 (24px), H3 (20px), Body (16px), Caption (14px)

4.2 Navigation et Architecture des Écrans

4.2.1 Navigation Principale (Mobile)

Structure de Navigation File-Based (Expo Router)

Route	Écran
/	Écran d'accueil
/Home	Générateur de menus
/mes-menus	Historique des menus
/courses	Liste de courses
/mes-recettes	Bibliothèque de recettes
/profil	Profil utilisateur
/Login	Connexion
/Signup	Inscription

FIGURE 4.1 – Architecture de navigation mobile

4.2.2 Flux Utilisateur Principal

1. Onboarding

- Écran de bienvenue avec présentation
- Choix : Connexion ou Continuer en anonyme
- Configuration des préférences initiales

2. Génération de Menu

- Sélection du régime alimentaire
- Configuration du budget avec slider
- Choix de la durée (3/5/7 jours)
- Options avancées (répétitions, strictness)
- Génération et affichage du menu

3. Consultation et Modification

- Vue détaillée de chaque repas
- Possibilité de régénérer un repas spécifique
- Sauvegarde dans "Mes Menus"
- Génération automatique de la liste de courses

4. Gestion des Courses

- Consultation de la liste agrégée
- Mode check pour cocher les articles
- Export et partage
- Archivage après achat

4.3 Composants UI Réutilisables

4.3.1 MenuCard Component

Carte interactive affichant un menu généré :

```

1 const MenuCard = ({ menu, onPress, onDelete }) => {
2   const colors = useColors();
3
4   return (
5     <TouchableOpacity
6       style={[styles.card, { backgroundColor: colors.card }]}
7       onPress={onPress}
8     >
9       <View style={styles.header}>
10         <Text style={[styles.title, { color: colors.text }]}>
11           {menu.name}
12         </Text>
13         <Text style={[styles.badge, { color: colors.primary }]}>
14           {menu.days.length} jours
15         </Text>
16       </View>
17
18       <View style={styles.info}>
19         <View style={styles.infoItem}>
20           <Icon name="calendar" size={16} color={colors.
21             textSecondary} />
22           <Text style={[styles.infoText, { color: colors.
23             textSecondary }]}>
24             Crée {formatDate(menu.createdAt)}
25           </Text>
26         </View>
27       </View>
28     </TouchableOpacity>
29   );
30 }
31
32 export default MenuCard;

```

```

23     </Text>
24   </View>
25
26   <View style={styles.infoItem}>
27     <Icon name="dollar-sign" size={16} color={colors.success} />
28     <Text style={[styles.infoText, { color: colors.text }]}>
29       {menu.totalCost} MAD
30     </Text>
31   </View>
32
33   <View style={styles.infoItem}>
34     <Icon name="flame" size={16} color={colors.warning} />
35     <Text style={[styles.infoText, { color: colors.text }]}>
36       {menu.totalCalories} kcal
37     </Text>
38   </View>
39 </View>
40
41 <TouchableOpacity
42   style={styles.deleteButton}
43   onPress={() => onDelete(menu.id)}
44 >
45   <Icon name="trash-2" size={20} color={colors.error} />
46 </TouchableOpacity>
47 </TouchableOpacity>
48 );
49 }

```

Listing 4.2 – Composant MenuCard

4.3.2 RecipeCard Component

Affichage d'une recette avec image et informations clés :

```

1 const RecipeCard = ({ recipe, onPress }) => {
2   return (
3     <TouchableOpacity style={styles.container} onPress={onPress}>
4       <Image
5         source={{ uri: recipe.image }}
6         style={styles.image}
7         resizeMode="cover"
8       />
9
10      <LinearGradient
11        colors={['transparent', 'rgba(0,0,0,0.8)']}
12        style={styles.gradient}
13      >
14        <Text style={styles.recipeName}>{recipe.name}</Text>
15
16        <View style={styles.tags}>
17          {recipe.tags.map(tag => (

```

```

18         <View key={tag} style={styles.tag}>
19             <Text style={styles.tagText}>{tag}</Text>
20         </View>
21     ))}
22   </View>

23
24   <View style={styles.footer}>
25     <View style={styles.stat}>
26       <Icon name="clock" size={14} color="#FFF" />
27       <Text style={styles.statText}>
28         {recipe.prepTime + recipe.cookTime} min
29       </Text>
30     </View>

31
32     <View style={styles.stat}>
33       <Icon name="users" size={14} color="#FFF" />
34       <Text style={styles.statText}>
35         {recipe.servings} pers
36       </Text>
37     </View>

38
39     <View style={styles.stat}>
40       <Icon name="dollar-sign" size={14} color="#22C55E" />
41       <Text style={[styles.statText, { color: '#22C55E' }]}>
42         {recipe.cost} MAD
43       </Text>
44     </View>
45   </View>
46 </LinearGradient>
47 </TouchableOpacity>
48 );
49 }

```

Listing 4.3 – Composant RecipeCard

4.3.3 LoadingSpinner Component

Indicateur de chargement avec animation :

```

1 const LoadingSpinner = ({ message = "Chargement..." }) => {
2   const rotation = useSharedValue(0);
3
4   useEffect(() => {
5     rotation.value = withRepeat(
6       withTiming(360, { duration: 1000, easing: Easing.linear }),
7       -1,
8       false
9     );
10   }, []);
11
12   const animatedStyle = useAnimatedStyle(() => ({
13     transform: [{ rotate: `${rotation.value}deg` }]

```

```
14 }) );  
15  
16     return (  
17         <View style={styles.container}>  
18             <Animated.View style={[styles.spinner, animatedStyle]}>  
19                 <Icon name="loader" size={40} color="#3B82F6" />  
20             </Animated.View>  
21             <Text style={styles.message}>{message}</Text>  
22         </View>  
23     );  
24 };
```

Listing 4.4 – Composant LoadingSpinner

4.4 Animations et Interactions

4.4.1 Animations Principales

1. **Transitions d'écran** - Slide horizontal avec fade
2. **Cartes** - Scale et shadow au touch
3. **Boutons** - Haptic feedback + scale down
4. **Listes** - Stagger animation pour l'apparition
5. **Génération** - Progress circular avec pourcentage

4.4.2 Gestes Supportés

- **Swipe to delete** - Suppression de menus/recettes
- **Pull to refresh** - Actualisation des listes
- **Long press** - Options contextuelles
- **Pinch to zoom** - Zoom sur les images de recettes

4.5 Accessibilité

4.5.1 Normes WCAG 2.1 Respectées

- **Contraste** - Ratio minimum 4.5 :1 pour le texte
- **Taille des touch targets** - Minimum 44x44 pts
- **Labels accessibles** - Tous les éléments interactifs
- **Navigation au clavier** - Support complet (web)
- **Screen readers** - Compatible VoiceOver et TalkBack

Chapitre 5

Performance et Optimisation

5.1 Stratégies d'Optimisation

5.1.1 Optimisation du Rendu

1. **React.memo** - Mémorisation des composants purs

```
1 export const RecipeCard = React.memo(({ recipe, onPress }) => {
2   // Composant
3 }, (prevProps, nextProps) => {
4   return prevProps.recipe.id === nextProps.recipe.id;
5 });
```

2. **useMemo** - Mémorisation des calculs coûteux

```
1 const filteredRecipes = useMemo(() => {
2   return recipes.filter(recipe =>
3     recipe.tags.includes(selectedDiet) &&
4     recipe.cost <= budget
5   );
6 }, [recipes, selectedDiet, budget]);
```

3. **useCallback** - Stabilisation des fonctions

```
1 const handleGenerateMenu = useCallback(() => {
2   const menu = generateMenu(dietType, budget, days, preferences)
3   ;
4   setGeneratedMenu(menu);
5 }, [dietType, budget, days, preferences]);
```

5.1.2 Gestion des Images

5.1.3 Optimisation des Listes

Utilisation de `FlatList` avec optimisations :

```
1 <FlatList
2   data={menus}
3   renderItem={({ item }) => <MenuCard menu={item} />}
```

Technique	Implémentation
Lazy Loading	Images chargées uniquement quand visibles
Cache	Utilisation de FastImage (React Native) avec cache disque
Compression	Images Unsplash servies en format WebP optimisé
Placeholders	Affichage de blurhash pendant le chargement
CDN	Unsplash CDN pour delivery rapide

TABLE 5.1 – Stratégies d'optimisation des images

```

4   keyExtractor={item => item.id}
5   initialNumToRender={10}
6   maxToRenderPerBatch={10}
7   windowSize={5}
8   removeClippedSubviews={true}
9   getItemLayout={(data, index) => ({
10     length: ITEM_HEIGHT,
11     offset: ITEM_HEIGHT * index,
12     index,
13   })}
14   onEndReached={loadMore}
15   onEndReachedThreshold={0.5}
16 />

```

Listing 5.1 – Configuration optimisée de FlatList

5.2 Métriques de Performance

5.2.1 Temps de Chargement

Opération	Temps Moyen	Cible
Démarrage de l'app	1.2s	< 2s
Génération de menu	0.8s	< 1s
Chargement de liste	0.3s	< 0.5s
Sync Firebase	1.5s	< 3s
Affichage d'une recette	0.4s	< 0.5s

TABLE 5.2 – Métriques de performance mesurées

5.2.2 Utilisation de la Mémoire

- Footprint initial : 45 MB
- Avec 50 menus en cache : 65 MB
- Avec images chargées : 85 MB
- Pic maximal observé : 120 MB

5.3 Optimisation du Bundle

5.3.1 Taille de l'Application

Plateforme	Taille	Commentaire
Android (APK)	28 MB	Sans compression
Android (AAB)	18 MB	Format Google Play
iOS (IPA)	32 MB	Build production
Web (gzipped)	450 KB	JS + CSS

TABLE 5.3 – Tailles des builds par plateforme

5.3.2 Stratégies de Réduction

1. **Tree Shaking** - Élimination du code mort
2. **Code Splitting** - Chargement lazy des routes
3. **Compression** - Minification et gzip
4. **Asset Optimization** - Images optimisées
5. **Selective Imports** - Import uniquement des modules nécessaires

Chapitre 6

Tests et Assurance Qualité

6.1 Stratégie de Test

6.1.1 Pyramide de Tests

Architecture de Tests		
Niveau	Nombre	Couverture
E2E Tests	15	Parcours utilisateur
Integration Tests	35	Services + UI
Unit Tests	120	Fonctions pures

FIGURE 6.1 – Pyramide de tests du projet

6.1.2 Outils de Test

- **Jest** - Framework de test principal
- **React Native Testing Library** - Tests de composants
- **Detox** - Tests end-to-end mobile
- **Firebase Emulator** - Tests des services Firebase
- **MSW (Mock Service Worker)** - Mocking des API

6.2 Tests Unitaires

6.2.1 Exemple : Test du Générateur de Menus

```
1 import { generateMenu } from '../services/menuGenerator';
2 import { recipes } from '../services/recipes';
3
4 describe('Menu Generator', () => {
5   test('should generate menu within budget', () => {
6     const menu = generateMenu('normal', 50, 7, {
```

```

7     strictBudget: true,
8     avoidRepetitions: true
9   });
10
11   expect(menu.days).toHaveLength(7);
12
13   menu.days.forEach(day => {
14     const dailyCost = day.breakfast.cost +
15       day.lunch.cost +
16       day.dinner.cost;
17     expect(dailyCost).toBeLessThanOrEqual(50);
18   });
19 });
20
21 test('should respect diet type', () => {
22   const menu = generateMenu('vegetarian', 60, 5, {});
23
24   menu.days.forEach(day => {
25     ['breakfast', 'lunch', 'dinner'].forEach(meal => {
26       const recipe = recipes.find(r => r.id === day[meal].recipeId);
27       expect(recipe.tags).toContain('vegetarian');
28     });
29   });
30 });
31
32 test('should avoid repetitions when requested', () => {
33   const menu = generateMenu('normal', 70, 7, {
34     avoidRepetitions: true
35   });
36
37   const allRecipeIds = menu.days.flatMap(day => [
38     day.breakfast.recipeId,
39     day.lunch.recipeId,
40     day.dinner.recipeId
41   ]);
42
43   const uniqueIds = new Set(allRecipeIds);
44   expect(uniqueIds.size).toBe(allRecipeIds.length);
45 });
46 });

```

Listing 6.1 – Tests du service menuGenerator

6.2.2 Exemple : Test de la Liste de Courses

```

1 import { generateShoppingList, aggregateIngredients } from '../
2   services/shoppingList';
3
3 describe('Shopping List Generator', () => {
4   const mockMenu = {

```

```

5   days: [
6     {
7       breakfast: { recipeId: 'recipe_001' },
8       lunch: { recipeId: 'recipe_002' },
9       dinner: { recipeId: 'recipe_003' }
10    }
11  ]
12};

13
14 test('should aggregate same ingredients', () => {
15   const list = generateShoppingList(mockMenu);
16
17   const tomatoItems = list.filter(item =>
18     item.ingredient === 'Tomates'
19   );
20
21   expect(tomatoItems).toHaveLength(1);
22   expect(tomatoItems[0].quantity).toBeGreaterThan(0);
23 });
24
25 test('should convert units correctly', () => {
26   const items = [
27     { ingredient: 'Farine', quantity: 500, unit: 'g' },
28     { ingredient: 'Farine', quantity: 0.5, unit: 'kg' }
29   ];
30
31   const aggregated = aggregateIngredients(items);
32
33   expect(aggregated[0].quantity).toBe(1);
34   expect(aggregated[0].unit).toBe('kg');
35 });
36});

```

Listing 6.2 – Tests du service shoppingList

6.3 Tests d’Intégration

6.3.1 Test de Synchronisation Firebase

```

1 import { syncMenus, syncRecipes } from '../services/sync';
2 import { initializeTestEnvironment } from '@firebase/rules-unit-
3 testing';
4
5 describe('Firebase Sync Integration', () => {
6   let testEnv;
7
8   beforeAll(async () => {
9     testEnv = await initializeTestEnvironment({
10       projectId: 'mealplanner-test',
11       firestore: { host: 'localhost', port: 8080 }
12     });
13   });
14
15   it('Syncs menu and recipes', () => {
16     const menuRef = testEnv.ref('menu');
17     const recipeRef = testEnv.ref('recipes');
18
19     const menuData = [
20       { id: 'm1', name: 'Breakfast', meals: [
21         { id: 'bm1', name: 'Omelette', ingredients: [
22           { id: 'bm1i1', name: 'Eggs', quantity: 2, unit: 'dozen' },
23           { id: 'bm1i2', name: 'Milk', quantity: 1, unit: 'cup' }
24         ]
25       }]
26     ];
27
28     const recipeData = [
29       { id: 'r1', name: 'Omelette', ingredients: [
30         { id: 'ri1', name: 'Eggs', quantity: 2, unit: 'dozen' },
31         { id: 'ri2', name: 'Milk', quantity: 1, unit: 'cup' }
32       ]
33     ];
34
35     await menuRef.set(menuData);
36     await recipeRef.set(recipeData);
37
38     const snapshot = await menuRef.get();
39     const menuSnapshot = snapshot.data();
40
41     expect(menuSnapshot).toEqual(menuData);
42
43     const snapshot2 = await recipeRef.get();
44     const recipeSnapshot = snapshot2.data();
45
46     expect(recipeSnapshot).toEqual(recipeData);
47   });
48 });
49
50 afterAll(() => {
51   testEnv.cleanup();
52 });
53
54 
```

```

11  });
12 );
13
14 test('should sync menus to Firestore', async () => {
15   const mockMenu = {
16     id: 'menu_test_001',
17     name: 'Test Menu',
18     days: /* ... */,
19     userId: 'test_user'
20   };
21
22   await syncMenus([mockMenu]);
23
24   const snapshot = await testEnv
25     .firestore()
26     .collection('menus')
27     .doc(mockMenu.id)
28     .get();
29
30   expect(snapshot.exists).toBe(true);
31   expect(snapshot.data().name).toBe('Test Menu');
32 );
33
34 afterAll(async () => {
35   await testEnv.cleanup();
36 });
37 );

```

Listing 6.3 – Test d'intégration sync.js

6.4 Tests E2E

6.4.1 Test du Parcours Complet de Génération

```

1 describe('Menu Generation Flow', () => {
2   beforeAll(async () => {
3     await device.launchApp();
4   );
5
6   it('should generate and save a menu', async () => {
7     // Navigation vers le générateur
8     await element(by.id('generate-menu-button')).tap();
9
10    // Sélection du régime
11    await element(by.id('diet-picker')).tap();
12    await element(by.text('Végétarien')).tap();
13
14    // Configuration du budget
15    await element(by.id('budget-slider')).swipe('right', 'slow');
16

```

```

17 // Selection de la dure e
18 await element(by.id('days-7')).tap();
19
20 // Generation
21 await element(by.id('generate-button')).tap();
22
23 // Attendre la generation
24 await waitFor(element(by.id('menu-result')))
25   .toBeVisible()
26   .withTimeout(5000);
27
28 // V rifier l'affichage
29 await expect(element(by.id('menu-result'))).toBeVisible();
30 await expect(element(by.text('Lundi'))).toBeVisible();
31
32 // Sauvegarder
33 await element(by.id('save-menu-button')).tap();
34
35 // V rifier la sauvegarde
36 await element(by.id('tab-mes-menus')).tap();
37 await expect(element(by.id('menu_saved'))).toBeVisible();
38 );
39 }

```

Listing 6.4 – Test E2E avec Detox

6.5 Couverture de Code

6.5.1 Objectifs de Couverture

Catégorie	Actuel	Objectif
Services	85%	> 80%
Composants UI	72%	> 70%
Utilitaires	92%	> 90%
Navigation	68%	> 65%
Global	78%	> 75%

TABLE 6.1 – Couverture de tests par catégorie

6.6 Qualité du Code

6.6.1 Outils de Linting

- **ESLint** - Analyse statique du code JavaScript
- **Prettier** - Formatage automatique
- **TypeScript** (migration en cours) - Typage statique

6.6.2 Configuration ESLint