Rapport du Projet

Système de Suivi des Dépenses

Réalisé par :
EL KINANI CHIHAB
ABDELLAOUI IBTYSSAM

Cours : DEV MOBILE
Année Académique : 2025–2026

Table des matières

Introduction Générale

La gestion des dépenses personnelles nécessite un outil *simple*, *fiable* et *mobile*. Le projet **Expense Tracker** vise à fournir une application multiplateforme (iOS/Android) construite avec React Native (Expo), adoptant une approche *offline-first* pour garantir un fonctionnement sans connexion, tout en offrant une synchronisation optionnelle via le cloud (Firebase).

Les contributions clés incluent : une expérience utilisateur fluide (navigation par onglets, respect des safe areas), une persistance locale robuste (SQLite), une authentification et une sauvegarde *par utilisateur*, ainsi que des statistiques utiles (moyenne mensuelle, prédiction, alertes). Nous détaillons dans ce rapport le contexte, les exigences, la conception, l'architecture technique et la réalisation de l'application, avec des extraits de code et des captures d'écran.

Chapitre 1: Contexte Général du Projet

1.1. Introduction

Ce chapitre présente le **contexte** et les **motivations** du projet : offrir un suivi de dépenses *accessible* sur mobile, avec des **fonctionnalités essentielles** (ajout rapide, historique, statistiques) et une **fiabilité hors-ligne**. L'approche *offline-first* assure une continuité d'usage ; la synchronisation cloud par utilisateur permet de sauvegarder/restaurer les données de manière sécurisée.

1.2. Description du Projet

Vision

L'objectif principal de ce projet est de concevoir une application mobile de suivi des dépenses simple, fiable et orientée *offline-first*. L'application permet aux utilisateurs de gérer leurs dépenses quotidiennes sans dépendre d'une connexion Internet, tout en offrant des fonctionnalités de sauvegarde et de restauration des données par utilisateur via Firebase.

Objectifs

Les objectifs majeurs du projet sont les suivants :

- Permettre l'ajout et la consultation rapide des dépenses personnelles ;
- Fournir des statistiques utiles telles que les totaux mensuels, la moyenne des dépenses et une tendance de consommation ;
- Assurer une isolation stricte des données pour chaque utilisateur ;
- Garantir un fonctionnement hors-ligne grâce à un stockage local ;
- Offrir une fonctionnalité de sauvegarde et de restauration sécurisée via le cloud.

Périmètre fonctionnel

Le périmètre fonctionnel de l'application couvre :

- L'authentification des utilisateurs (inscription, connexion et déconnexion) ;
- La gestion complète des dépenses (création, consultation et suppression) ;
- L'historique mensuel des dépenses avec regroupement par périodes ;
- L'affichage de statistiques et d'indicateurs financiers ;
- Les paramètres de l'application, incluant la sauvegarde, la restauration et la réinitialisation locale des données.

Utilisateurs cibles

Cette application s'adresse principalement aux particuliers souhaitant suivre leurs dépenses quotidiennes de manière simple et efficace, sans complexité inutile. Le système est conçu pour supporter plusieurs utilisateurs sur un même appareil, avec une séparation complète des données.

Principales fonctionnalités

Les fonctionnalités clés offertes par l'application sont :

- Authentification sécurisée par email et mot de passe avec persistance de session ;
- Saisie, affichage et suppression des dépenses avec catégorisation et tri par date ;
- Consultation de l'historique mensuel sous forme de sections ;
- Calcul de statistiques incluant les totaux mensuels, la moyenne, une prédiction simple et une alerte en cas de dépassement de la moyenne ;
- Sauvegarde et restauration des données par utilisateur via Firebase Realtime Database.

Contraintes et choix techniques

Plusieurs contraintes et choix techniques ont guidé la réalisation du projet :

- Utilisation de React Native avec Expo pour un développement multiplateforme ;
- Stockage local des données via SQLite afin d'assurer performance et accessibilité hors-ligne ;
- Filtrage systématique des données par `userId` pour garantir l'isolation des comptes ;
- Conception d'une interface utilisateur cohérente et adaptée aux *safe areas* mobiles ;
- Mise en place de règles de sécurité côté Firebase afin de limiter l'accès aux données de sauvegarde au chemin `backup/<uid>`.

1.3. État de l'existant

Plusieurs applications de gestion des dépenses existent (ex. Mint, YNAB, Monefy). Elles proposent des fonctionnalités avancées (catégorisation, synchronisation multi-plateformes, budgétisation). Toutefois, leurs limites dans notre contexte sont :

— Complexité d'usage pour des besoins simples et quotidiens.
— Dépendance forte au réseau : sans connexion, l'expérience peut être dégradée.
— Modèle de données parfois opaque et non exportable facilement.

1.4. Problématique

Concevoir une application **simple**, **offline-first**, **multi-utilisateur**, permettant :

— L'ajout et la consultation rapide des dépenses.
— Des **statistiques utiles** (moyenne mensuelle, prédiction simple, alertes).
— Une **sauvegarde/restauration** cloud fiable, par **utilisateur**.

1.5. Solution proposée

— **Front mobile** : React Native (Expo) pour accélérer le développement, unifier iOS/Android et bénéficier d'outils de dev.
— **Données locales** : SQLite (`expo-sqlite`) pour une persistance fiable, requêtes rapides et un modèle structuré.
— **Cloud** : Firebase Authentication (Email/Password) + Realtime Database pour sauvegarder/restaurer les données *par utilisateur*.
— **Navigation** : React Navigation avec un *AuthNavigator* (Login/Register) et un *AppNavigator* (Dashboard, Add, History, Settings).

1.6. Objectifs

Fonctionnels : gestion des dépenses (CRUD), statistiques, backup/restore par utilisateur, connexion/déconnexion.

Techniques : offline-first, performances locales, séparation stricte des données par utilisateur, UX cohérente (safe areas, tab bar).

1.7. Conclusion du chapitre

La solution proposée répond aux besoins identifiés et prépare l'étude détaillée des exigences, de l'architecture et des choix techniques.

Chapitre 2: Étude, Analyse et Conception

2.1. Cahier des charges

2.1.1. Spécifications fonctionnelles

— Authentification : inscription (avec `displayName`), connexion, persistance de session, déconnexion.
— Dépenses : ajout, suppression, historique mensuel, total, statistiques (moyenne, prédiction, alerte au-dessus de la moyenne).
— Sauvegarde/Restaurer : opérations cloud **par utilisateur**.
— UI/UX : navigation par onglets, design épuré, prise en charge des safe areas.

2.1.2. Spécifications techniques

— Expo SDK + React Native, `react-navigation` (tabs + stack).
— SQLite via `expo-sqlite` (tables locales), schéma avec colonne `userId`.
— Firebase Auth + Realtime Database ; persistance Auth via `@react-native-async-storage/async-stora`

2.1.3. Choix de la base de données

SQLite est retenue pour :

(i) sa robustesse et ses requêtes performantes,

(ii) son modèle structuré (contraintes, index),

(iii) une meilleure fiabilité que les stockages clé/valeur simples.

Alternatives évaluées : *AsyncStorage seul* (trop limité pour requêtes/agrégations), *Realm* (performant mais ajoute une dépendance native et une complexité de déploiement). SQLite via Expo offre un bon compromis simplicité/performance.

2.2. Analyse

2.2.1. Flux d'authentification

Initialisation : `initializeAuth(app, { persistence: getReactNativePersistence(AsyncStorage) })`.

Écoute de session : `onAuthStateChanged` met à jour l'état global et choisit la pile de navigation :

Listing 2.1 – Surveillance de la session et routing

```
useEffect(() => {
  initDatabase();
  const unsub = onAuthStateChange(async (user) => {
    setUser(user);
    if (user) {
      await ensureUserScopedSchemaAndBackfill(user.uid);
    }
    setLoading(false);
  });
  return () => unsub();
}, []);
```

2.2.2. Flux des dépenses

- **Ajout** : insertion dans SQLite avec `userId` ; recalcul du total et rafraîchissement de la liste.
- **Lecture** : toutes les requêtes filtrent par `userId` (isolement multi-utilisateurs). Rafraîchissements via `useEffect` (montage) et `useFocusEffect` (retour sur l'écran).
- **Historique** : groupement par mois et calculs par section.

2.2.3. Flux de sauvegarde/restauration

- **Sauvegarde** : collecte des dépenses locales de l'utilisateur et envoi dans RTDB sous `backup/<uid>`.
- **Restauration** : récupération RTDB puis remplacement *des lignes locales de cet utilisateur uniquement.*

2.3. Conception

2.3.1. Diagramme de cas d'utilisation

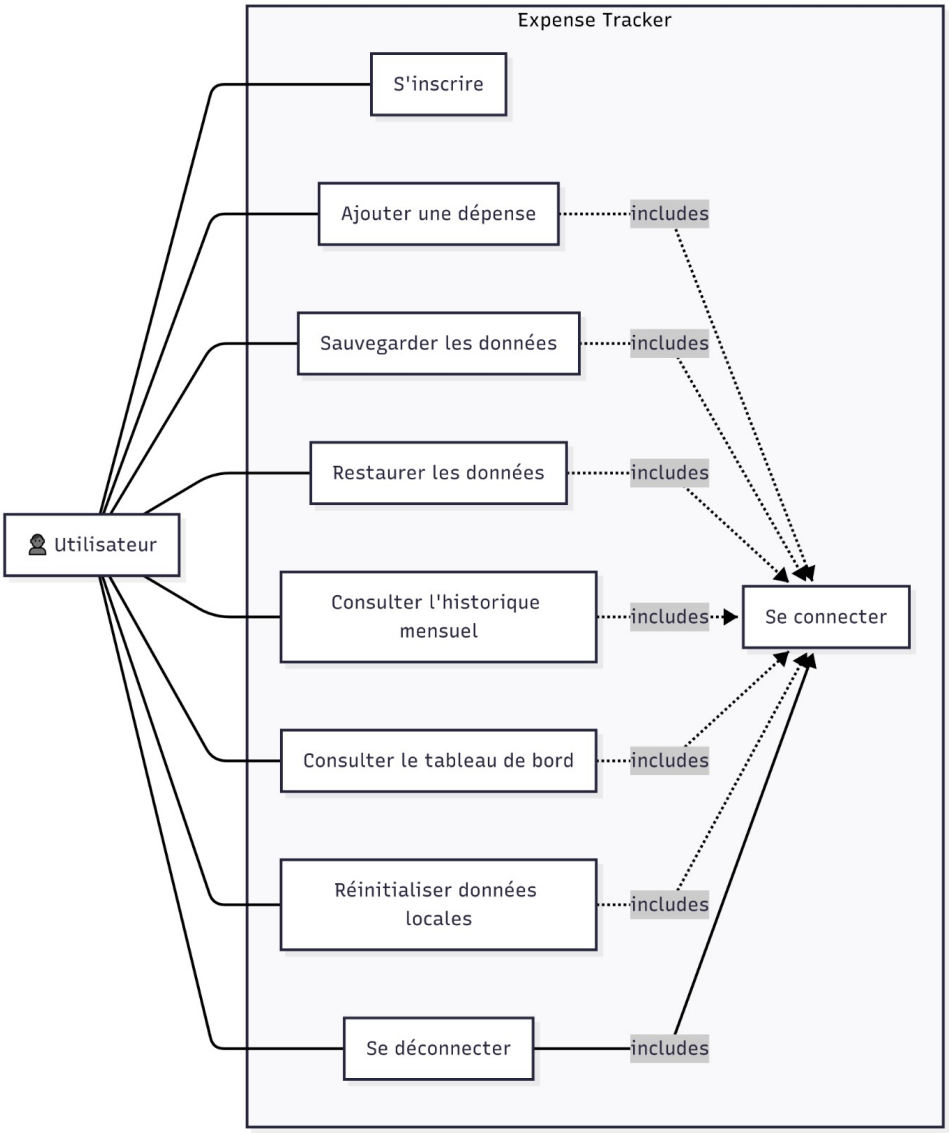

FIGURE 2.1 – Diagramme de cas d'utilisation de l'application Expense Tracker

2.3.2. Diagramme de cas classe

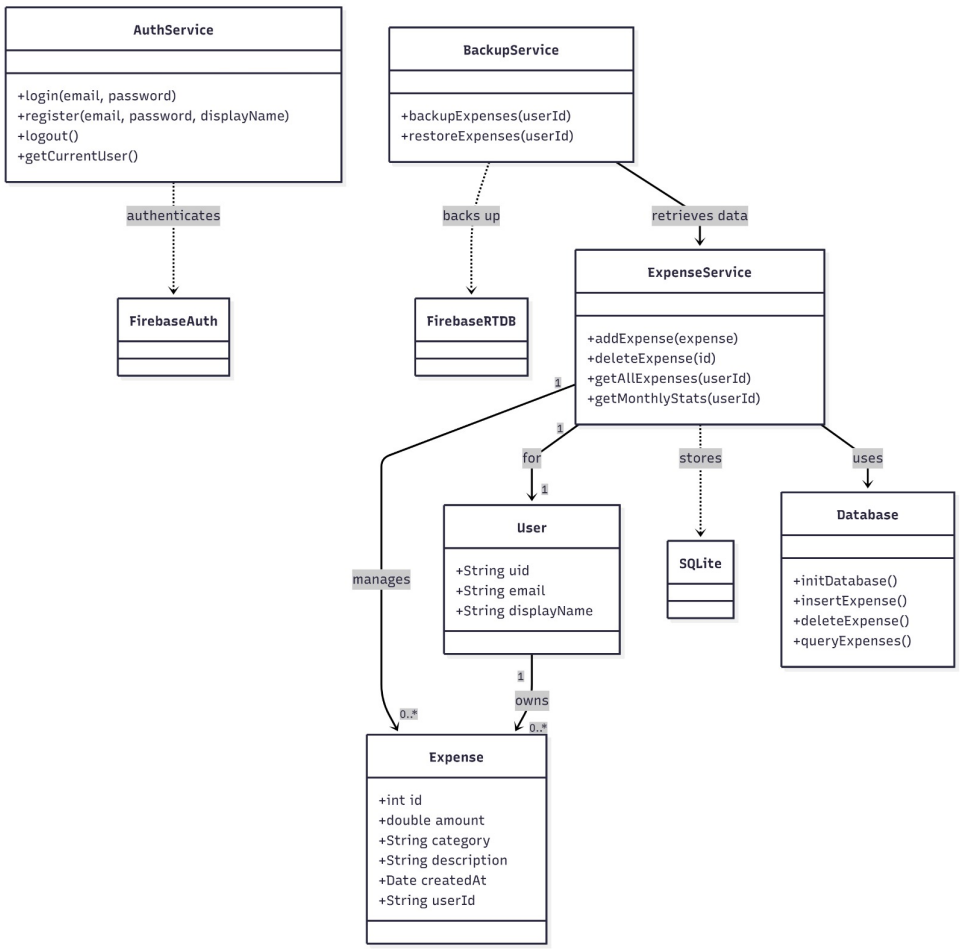

FIGURE 2.2 – Diagramme de classe de l'application Expense Tracker

2.3.3. Modèle et schéma

Entité Expense : `{ id, amount, category, description, createdAt, userId }`.

Services : *authService* (auth/écoute), *expenseService* (CRUD SQLite + migration `userId`), *backupService* (RTDB par user).

2.3.4. Extraits clés

Migration et filtrage par utilisateur :

Listing 2.2 – Migration et requêtes filtrées par userId

```
export const ensureUserScopedSchemaAndBackfill = async (userId) => {
  const cols = await db.getAllAsync("PRAGMA table_info('expenses')");
  const hasUserId = cols?.some?.(c => c.name === 'userId');
  if (!hasUserId) await db.execAsync('ALTER TABLE expenses ADD COLUMN userId
      TEXT');
```

```
  if (userId) await db.runAsync('UPDATE expenses SET userId = ? WHERE userId IS NULL', [userId]);
};

export const getAllExpenses = async () => {
  const userId = getCurrentUser()?.uid || 'anonymous';
  await ensureUserScopedSchemaAndBackfill(userId);
  return db.getAllAsync('SELECT * FROM expenses WHERE userId = ? ORDER BY createdAt DESC', [userId]);
};
```

Sauvegarde/restauration par utilisateur :

Listing 2.3 – Backup/restore par uid dans Firebase RTDB

```
const uid = getCurrentUser().uid;
const backupRef = ref(firebaseDB, `backup/${uid}`);
await set(backupRef, { expenses, timestamp: Date.now(), count: expenses.length });
// Restore: lire backup/${uid}, purger les rows locales de cet uid,
// r ins rer (avec userId = uid).
```

Chapitre 3: Étude Technique

3.1. Architecture

Navigation (AuthNavigator, AppNavigator), services (auth, expense, backup), base locale (SQLite), intégration Firebase, et gestion des safe areas/tab bar.

3.1.1. Arborescence du projet

```
ExpenseTracker/
  App.js
  src/
    components/
    navigation/
    screens/
    services/
    database/
    config/
    utils/
```

3.2. Choix des outils

Expo/React Native : itérations rapides, outils intégrés (Metro, DevTools), cross-platform. Alternatives (*bare RN*) nécessitent plus de configuration native.

SQLite (expo-sqlite) : stockage structuré, requêtes performantes, contrôle local complet (mieux qu'AsyncStorage seul). Comparé à *Realm*, SQLite reste plus simple à distribuer avec Expo.

Firebase Auth + RTDB : backend managé pour l'authentification et la sauvegarde simple par utilisateur. Firestore est une alternative ; RTDB suffit pour un modèle simple (backup par utilisateur).

react-native-safe-area-context : respect des zones sécurisées (encoches, barre système), confort visuel.

3.3. Modèle de données et migrations

Table `expenses` : `id` (PK), `amount`, `category`, `description`, `createdAt`, `userId`.

Migration automatique au login via `ensureUserScopedSchemaAndBackfill` (ajout colonne si besoin + rétro-écriture du `userId` manquant). Toutes les requêtes CRUD filtrent par `userId`. Indices recommandés : `userId`, `createdAt` pour accélérer l'historique.

3.4. Sécurité

Clés API limitées côté client ; règles RTDB recommandées pour restreindre `backup/uid` à l'utilisateur ; pas de secrets critiques côté mobile.

Chapitre 4: Réalisation

4.1. Écrans et fonctionnalités

4.1.1. Authentification

Login/Register (Email/Password), persistance de session (AsyncStorage), message d'erreur clair (ex. : `auth/configuration-not-found`).

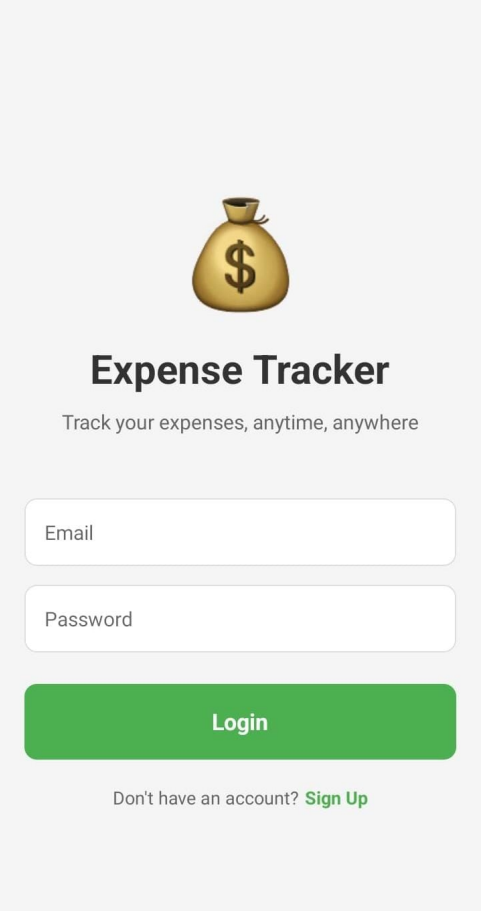

FIGURE 4.1 – Interface Login de l'application Expense Tracker

4.1.2. Dashboard

Statistiques : dépenses du mois, moyenne, prédiction, alerte si au-dessus de la moyenne. Affiche `displayName` ou email.

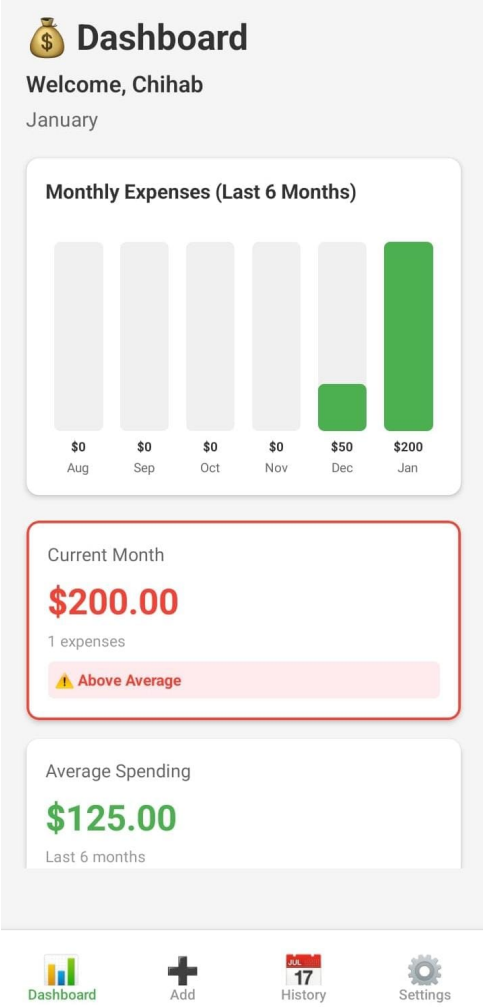

FIGURE 4.2 – Interface Dashboard l'application Expense Tracker

4.1.3. Ajout de dépenses

Formulaire d'ajout, liste et total dynamique ; requêtes filtrées par `userId`.

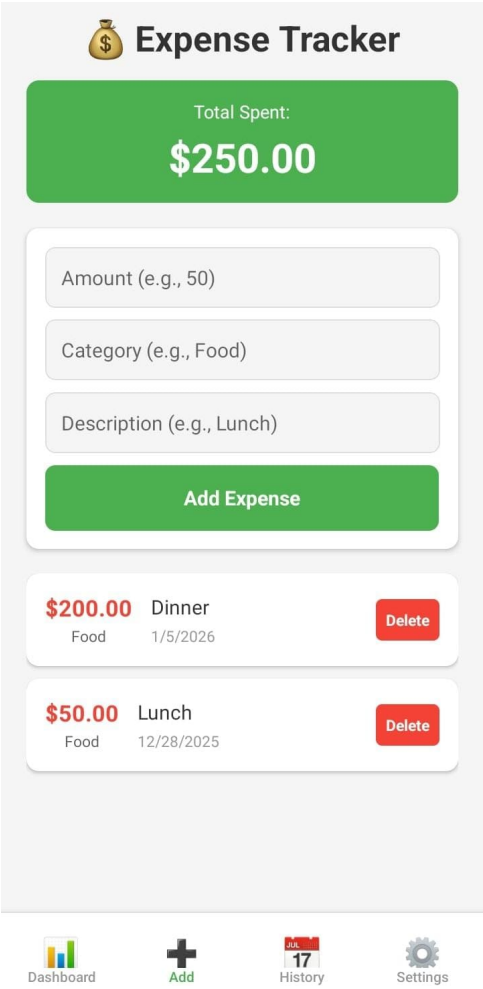

FIGURE 4.3 – Ajout de dépenses

4.1.4. Historique

Groupement par mois avec totaux par section, suppression d'éléments, rafraîchissement à la focalisation.

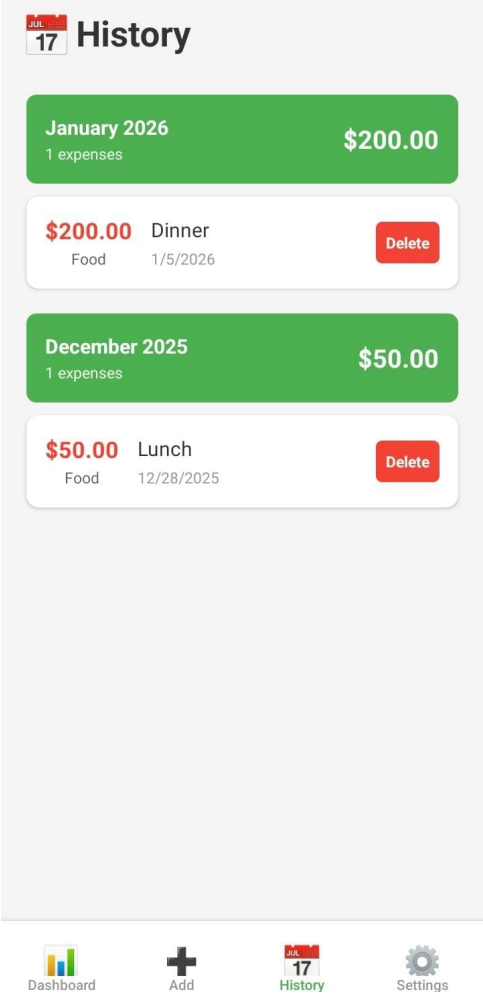

FIGURE 4.4 – Interface ajout de dépenses de l'application Expense Tracker

4.1.5. Paramètres

Sauvegarde/Restaurer par utilisateur, Déconnexion, Réinitialisation des données locales pour l'utilisateur courant.

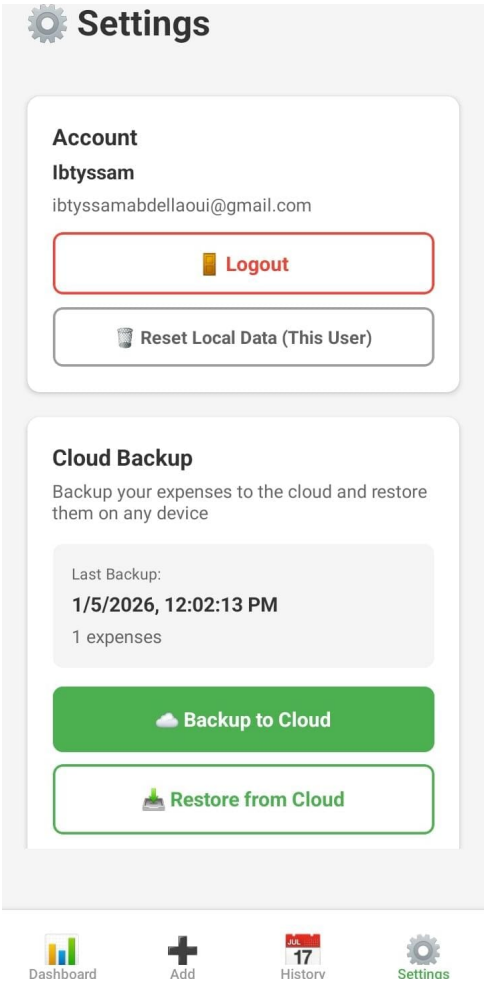

FIGURE 4.5 – Interface Paramètres de l'application Expense Tracker

— **Multi-utilisateur** : isolation confirmée (lectures filtrées par `userId`).
— **Backup/Restore** : pour chaque utilisateur, chemin RTDB séparé (`backup/uid`).
— **UX** : safe areas, bar de navigation relevée, contenu non masqué.
— **Offline** : opérations locales fonctionnelles sans réseau ; synchronisation via backup.

Chapitre 5: Gestion de Projet (Taiga)

5.1. Approche et organisation

Gestion agile avec Taiga

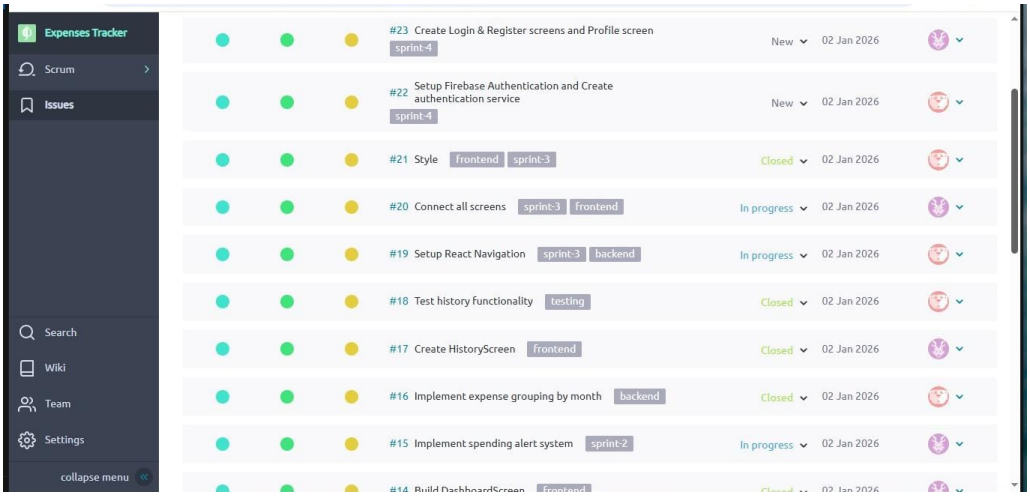

FIGURE 5.1 – Backlog Taiga

Chapitre 6: Difficultés rencontrées et solutions

— **Compatibilité React Navigation** : versions et dépendances transverses. *Solution* : alignement des packages et nettoyage du cache Expo.

— **Persistance Auth** : configuration `initializeAuth` et AsyncStorage. *Solution* : persistence explicite et `onAuthStateChanged` centralisé.

— **Migrations SQLite** : ajout `userId` sans perte. *Solution* : `ALTER TABLE` + backfill ciblé.

— **Performance liste** : rafraîchissements fréquents. *Solution* : requêtes filtrées, memoization et rechargement à la focalisation.

— **Sauvegarde RTDB** : schéma et règles de sécurité. *Solution* : segmentation par `backup/uid` et règles limitatives.

Conclusion Générale

Le projet *Expense Tracker* fournit une application mobile robuste, simple et *offline-first*, avec séparation stricte des données par utilisateur et une sauvegarde cloud fiable. Les évolutions envisagées : règles de sécurité avancées, synchronisation temps réel, budget prévisionnel, tests automatisés et déploiement store.

Webographie

Cette webographie regroupe l'ensemble des ressources documentaires utilisées pour la conception, le développement et la validation du projet **Expense Tracker**. Les sources officielles ont été privilégiées afin de garantir la fiabilité technique et méthodologique.

Documentation Officielle

React Native & Expo

— Expo Documentation. `https://docs.expo.dev/` Documentation officielle pour le développement d'applications React Native avec Expo. Consulté en 2025.

— React Native Documentation. `https://reactnative.dev/docs/getting-started` Guide officiel de React Native. Consulté en 2025.

— React Hooks Reference. `https://react.dev/reference/react` Documentation officielle des Hooks React (`useState`, `useEffect`, `useFocusEffect`). Consulté en 2025.

Navigation

— React Navigation Documentation. `https://reactnavigation.org/docs/getting-started` Bibliothèque de navigation pour React Native. Consulté en 2025.

— Bottom Tab Navigator. `https://reactnavigation.org/docs/bottom-tab-navigator` Guide de navigation par onglets. Consulté en 2025.

— Stack Navigator. `https://reactnavigation.org/docs/native-stack-navigator` Navigation en pile pour les écrans d'authentification. Consulté en 2025.

Base de Données Locale

— Expo SQLite. `https://docs.expo.dev/versions/latest/sdk/sqlite/` Documentation officielle pour l'utilisation de SQLite avec Expo. Consulté en 2025.

— SQLite Tutorial. `https://www.sqlitetutorial.net/` Guide sur les requêtes SQL et la gestion des bases SQLite. Consulté en 2025.

— DB Browser for SQLite. https://sqlitebrowser.org/ Outil graphique pour la visualisation et la gestion des bases SQLite. Consulté en 2025.

Firebase (Cloud et Authentification)

— Firebase Documentation. https://firebase.google.com/docs Documentation principale de Firebase. Consulté en 2025.
— Firebase Realtime Database. https://firebase.google.com/docs/database Guide de la base de données temps réel Firebase. Consulté en 2025.
— Firebase Authentication. https://firebase.google.com/docs/auth Documentation sur l'authentification Firebase (Email/Mot de passe). Consulté en 2025.
— Firebase Web SDK. https://firebase.google.com/docs/web/setup Configuration du SDK Firebase pour React Native. Consulté en 2025.

Gestion de Projet et Versioning

— Taiga.io. https://tree.taiga.io/ Outil de gestion de projet agile utilisé pour organiser les sprints. Consulté en 2025.
— GitHub. https://github.com/ Plateforme de gestion de versions Git. Consulté en 2025.
— Git Documentation. https://git-scm.com/doc Documentation officielle de Git. Consulté en 2025.

Tutoriels et Guides

— React Native Express. https://www.reactnative.express/ Guide complet pour l'apprentissage de React Native. Consulté en 2025.
— Expo SQLite Example. https://github.com/expo/examples/tree/master/with-sqlite Exemple officiel d'intégration de SQLite avec Expo. Consulté en 2025.
— Firebase Authentication in React Native. https://rnfirebase.io/auth/usage Guide d'intégration de l'authentification Firebase. Consulté en 2025.

Sécurité et Bonnes Pratiques

— Firebase Security Rules. https://firebase.google.com/docs/rules Documentation sur la sécurisation des données Firebase. Consulté en 2025.
— React Native Security Guide. https://reactnative.dev/docs/security Guide des bonnes pratiques de sécurité mobile. Consulté en 2025.

Architecture et Méthodologie

— Offline-First Architecture. `https://offlinefirst.org/` Principes des architectures offline-first. Consulté en 2025.
— Scrum Guide. `https://www.scrum.org/resources/scrum-guide` Guide officiel de la méthodologie Scrum. Consulté en 2025.