Rapport d'analyse technique

my-meal-planner

Auteur : **Analyse automatique**

Date : 2026-01-05

Repo : aayop/my-meal-planner (ID: 1127735899)

Résumé (une phrase) : Analyse technique et recommandations pour améliorer l'architecture, la qualité et la sécurité du projet my-meal-planner (front-end web & mobile).

Contents

Résumé exécutif

Objectif : Fournir une évaluation technique complète et des actions priorisées pour rendre my-meal-planner plus maintenable, sécurisée et prête pour distribution (web + mobile).

Constats rapides :

- Le dépôt contient une application front-end basée sur Vite/React et un sous-projet mobile Expo (chemins : `frontend/` et `frontend/app-mobile/`).
- Stack : JavaScript/TypeScript (configs présentes), React, Expo, Firebase (mobile), Tailwind/Vite.
- Pas de tests unitaires visibles ni de workflows CI publics détectés.
- Dépendances modernes mais certaines versions doivent être vérifiées pour compatibilité et vulnérabilités.

Trois recommandations principales (priorité) :

1. Ajouter pipeline CI (lint, build, tests) et scans SCA automatisés — priorité haute.
2. Introduire une suite minimale de tests (unitaires + e2e simple pour le front) — priorité haute.
3. Mettre à jour et verrouiller dépendances critiques (Firebase, Expo), puis activer Dependabot/renovate — priorité moyenne.

1 Introduction et objectifs

Contexte et périmètre Le dépôt analysé est accessible publiquement à l'URL : https://github.com/aayop/my-meal-planner. L'analyse porte principalement sur les fichiers accessibles publiquement, en particulier le répertoire `frontend/` (web + mobile) et les fichiers de configuration à la racine.

Méthodologie La méthode d'analyse utilisée consiste en :

- Lecture des fichiers README et des fichiers de configuration (`package.json`, `vite.config.js`, `tsconfig.json`, `tailwind.config.js`, `eslint.config.js`).
- Inspection de l'arborescence publique (`frontend/`, `frontend/app-mobile/`).
- Vérification de la présence de scripts npm et d'indices de CI/CD.

Limites L'analyse est limitée aux éléments publics du dépôt. Les PRs privées, historiques internes ou workflows non committés au dépôt public ne sont pas accessibles et ne sont donc pas traités ici.

2 Présentation du projet et fonctionnalités

Description synthétique

Nom : My Meal Planner — projet multi-plateforme (web + mobile) visant à planifier des repas, avec probable synchronisation via Firebase pour la version mobile.

Structure observée

Le dépôt présente au moins les éléments suivants :

- `frontend/` — application web (Vite + React).
- `frontend/app-mobile/` — application mobile (Expo / React Native).
- Configs : `vite.config.js`, `tsconfig.json`, `tailwind.config.js`, `eslint.config.js`.

Fonctionnalités attendues

D'après les dépendances et la structure :

1. Dashboard web avec visualisations (`recharts`).
2. Création / édition / planification de repas (formulaires).
3. Synchronisation mobile via Firebase (authentification / stockage des données).
4. Navigation multi-écrans (react-router / expo-router).

3 Stack technologique et dépendances

Langages et frameworks

- Langage principal : JavaScript (présence d'un `tsconfig.json` et d'`@types/*` indiquant usage possible de TypeScript).
- Front-end web : React + Vite, Tailwind CSS, Recharts.
- Mobile : Expo (React Native), Expo Router, Firebase client.

Extraits de dépendances (sélection)

```
"dependencies": {
  "react": "19.1.0",
  "react-dom": "19.1.0",
  "recharts": "^3.6.0",
  "expo": "^54.0.30"
}
```

(Voir `frontend/package.json` et `frontend/app-mobile/package.json` pour la liste complète.)

Remarques sur compatibilité

Les versions d'Expo, React Native et Firebase doivent être testées conjointement : certaines mises à jour majeures peuvent casser la compatibilité (ex. changements d'API Firebase ou incompatibilités Expo/RN).

4 Architecture et conception logicielle

Organisation générale

Le dépôt adopte une organisation mono-repo légère :

- `frontend/` : application web Vite + React
- `frontend/app-mobile/` : application Expo
- `assets/`, `public/`, `src/` sont présents sous `frontend/`

Diagramme d'architecture (texte)

```
App Web (frontend/)
   index.html
   src/
      components/
      pages/
      services/
   package.json

App Mobile (frontend/app-mobile/)
   app/
   package.json
   scripts/
```

Composants et responsabilités (recommandation)

- Isoler la logique d'accès aux données (Firebase) dans un module 'services/firebase' pour faciliter le test et la maintenance.
- Séparer UI & logique métier : composants purs (dumb) vs conteneurs (smart).
- Si des fonctions sont partagées entre web et mobile, envisager un paquet 'packages/shared/'.

5 Analyse du code et qualité

Organisation du dépôt

Fichiers importants :

- `frontend/README.md` – template Vite + React (lien : frontend/README.md)
- `frontend/package.json` – scripts et dépendances (lien : frontend/package.json)
- `frontend/app-mobile/package.json` – scripts Expo (lien : app-mobile/package.json)

Qualité observée

- Présence d'un linter (`eslint.config.js`) et script `lint` : bonne pratique.
- Absence de script `test` visible dans `package.json` : manque de tests automatisés.
- Présence d'un `tsconfig.json` et de `@types/*` : intention d'utiliser TypeScript ou support pour typings.

Extrait commenté (scripts)

```
{
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  }
}
```

Remarque : ajouter un script `"test": "jest"` et configurer Jest/RTL permettrait d'automatiser la vérification des composants.

Recommandations qualité

1. Ajouter des tests unitaires pour les composants critiques (Jest + React Testing Library).
2. Exécuter `eslint -fix` automatiquement via pre-commit et CI.
3. Mettre en place des règles de revue de code et templates PR.

6 Tests, CI/CD et processus de développement

État actuel

Aucun workflow GitHub Actions détecté dans la racine publique. Les scripts npm exposés sont : `dev`, `build`, `lint` (web) et `start`, `lint` (mobile).

Recommandations

1. Ajouter un workflow CI minimal déclenché sur `push` et `pull_request` : `npm ci`, `npm run lint`, `npm run build`.
2. Lorsque des tests sont ajoutés, exécuter `npm test` dans le pipeline.
3. Ajouter des checks automatiques (Dependabot, SCA) dans CI.

7 Sécurité, dépendances et vulnérabilités

Observations

- Dépendances clés : `firebase` (mobile) en [1]2.7.0, `expo 54.0.30`.
- `Pas d'indication visible de secrets committés dans les fichiers consultés (bon signe), mais il est nécessaire d'exécuter un scan pour en être sûr.`
- `Pas d'outil SCA (Dependabot/renovate) détecté publiquement.`

Actions recommandées

1. Activer Dependabot ou Renovate pour PRs automatiques de MAJ de dépendances.
2. Intégrer un scanner SCA (Snyk, GitHub Advanced Security ou OSSIndex) dans le pipeline CI.
3. Scanner l'historique Git pour secrets (git-secrets, truffleHog) et bloquer commits contenant secrets via pre-commit.

8 Qualité UX / Accessibilité

Observations

- Utilisation de Tailwind CSS et Recharts est adaptée pour une interface moderne et responsive.
- Sans captures d'écran ou démonstration, l'évaluation UX complète n'est pas possible.

Recommandations pratiques

- Intégrer des audits automatisés (Lighthouse, axe-core) dans le pipeline CI.
- Ajouter des tests d'accessibilité pour les composants (axe-core + jest-axe).
- Vérifier focus states, navigation clavier et contraste des couleurs.

9 Recommandations techniques et roadmap

Court terme (0–4 semaines)

1. Mettre en place un workflow GitHub Actions minimal : install, lint, build (web). Est. 1–2 jours.
2. Ajouter tests unitaires basiques (Jest + React Testing Library) pour composants critiques. Est. 3–7 jours.
3. Activer Dependabot / Renovate et corriger vulnérabilités bloquantes. Est. <1 jour pour activation.

Moyen terme (1–3 mois)

- Introduire e2e tests (Cypress / Playwright) pour flux principaux.
- Structurer code partagé (extraire utilitaires dans `packages/shared/` si nécessaire).
- Documenter architecture et processus de release (CHANGELOG, release checklist).

Long terme (3–6 mois)

- Automatiser releases (semantic-release).
- Mettre en place monitoring et crash reporting (Sentry / Firebase Crashlytics).

Risques principaux

- Incompatibilités lors de mises à jour entre Expo / Firebase / React Native.
- Absence actuelle de CI/tests pouvant permettre l'introduction de régressions.

Conclusion

my-meal-planner présente une base technique saine avec des choix modernes (Vite, React, Expo, Tailwind). L'effort prioritaire doit porter sur l'automatisation (CI & SCA) et la mise en place de tests pour améliorer la confiance dans les releases et réduire les risques. Une roadmap progressive (CI → tests → e2e → monitoring) fournira bénéfices rapides et mesurables.

A Annexes

A.1 A. Fichiers importants (liens)

https://github.com/aayop/my-meal-planner
frontend/README.md
frontend/package.json
frontend/app-mobile/package.json
frontend/vite.config.js
frontend/eslint.config.js
frontend/index.html

A.2 B. Extraits de code clés

1) Extrait de `frontend/package.json` (scripts)

```
{
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  }
}
```

2) Extrait de `frontend/app-mobile/package.json` (scripts & dépendances)

```
{
  "scripts": {
    "start": "expo start",
    "android": "expo start --android",
    "ios": "expo start --ios",
    "lint": "expo lint"
  },
  "dependencies": {
    "firebase": "^12.7.0",
    "expo": "~54.0.30",
    "react-native": "0.81.5"
  }
}
```

A.3 C. Commandes pour exécuter localement

1. Cloner le dépôt :

```
git clone https://github.com/aayop/my-meal-planner.git
```

2. Web (frontend) :

```
cd my-meal-planner/frontend
npm install
npm run dev
# ou pour build
npm run build
npm run preview
```

3. Mobile (Expo) :

```
cd my-meal-planner/frontend/app-mobile
npm install
npx expo start
```

Note : il peut être nécessaire d'installer `expo-cli` globalement (`npm i -g expo-cli`) ou d'utiliser `npx`.

A.4 D. Issues / PRs

Au moment de l'analyse, le dépôt public indique 0 issues ouvertes. Si des issues privées existent, elles n'étaient pas accessibles.

A.5 E. Méthode d'analyse

Analyse basée sur la lecture des fichiers publics accessibles via l'API GitHub : README, `package.json` (web et mobile), fichiers de configuration (vite, eslint, tailwind, tsconfig), et inspection de l'arborescence.

B Checklist finale — Actions à court terme

1. Mettre en place CI GitHub Actions minimal : `npm ci` → `npm run lint` → `npm run build` (exécuter sur chaque PR). (Est. 1–2 jours)
2. Ajouter tests unitaires basiques (Jest + React Testing Library) et script `test` dans `frontend/package.json`. (Est. 3–7 jours)
3. Activer Dependabot (ou Renovate) pour mises à jour automatiques des dépendances et corriger vulnérabilités bloquantes. (Est. <1 jour pour activation)

Fin du rapport.