Rapport d'analyse de sécurité du *Client iOS*

Analyse fonctionnelle et technique

Projet : Client IOS

Auteur :

Wissame ISMAEL

Date : 9 novembre 2025

Ce document présente une analyse de la sécurité du projet de « Client iOS » couvrant la sécurité fonctionnelle (usages, permissions) et la sécurité technique (architecture, données, communication). Il se base sur la fiche technique `fiche_technique_&_sécurité.md`.

Table des matières

1 Introduction

Ce rapport présente une analyse de la sécurité du projet de Client iOS tel que décrit dans le document `fiche_technique_&_sécurité.md`.

Le projet vise à créer le frontend mobile (application iOS développée avec React Native / Expo) pour la plateforme "Démocratie Permanente". Le périmètre de ce document inclut l'application iOS mais exclut l'API backend et l'infrastructure serveur.

Les fonctionnalités techniques reposent sur la communication via des requêtes HTTP (Axios) vers une API backend, la gestion de la navigation interne (React Navigation) et le stockage local sécurisé de tokens d'authentification (expo-secure-store).

L'architecture technique repose sur un framework React Native (0.76.x) et la toolchain Expo (SDK 52), utilisant TypeScript (5.6.x) pour le typage statique du code.

L'objectif de ce rapport est d'évaluer le projet sous deux angles : sécurité fonctionnelle (gestion des permissions et interactions utilisateur) et sécurité technique (robustesse de l'architecture frontend et des mesures de protection des données).

2 Analyse de la sécurité fonctionnelle

La sécurité fonctionnelle concerne la façon dont l'application gère les interactions avec l'utilisateur, ses données et les ressources de l'appareil.

2.1 Permissions et accès aux ressources

L'application gère les permissions de manière explicite. Les demandes d'accès aux ressources matérielles sensibles (ex : micro, caméra, géolocalisation) sont validées via les API natives d'iOS, conformément aux directives de la plateforme.

2.2 Validation des entrées côté client

Une validation côté client (input sanitization) est appliquée sur les formulaires et les payloads envoyés au backend. Ceci constitue une première ligne de défense pour fiabiliser les données envoyées, en complément des validations qui doivent être opérées côté serveur (hors périmètre).

2.3 Mises à jour (OTA)

Les mises à jour sont gérées via OTA (Over-The-Air) par le service Expo Updates. Ces mises à jour sont signées et contrôlées pour garantir l'intégrité du code déployé sur les appareils des utilisateurs et permettre des correctifs de sécurité rapides.

3 Analyse de la sécurité technique

La sécurité technique évalue l'implémentation des contrôles de sécurité dans l'architecture de l'application cliente.

3.1 Architecture et communications

L'architecture de communication repose exclusivement sur **HTTPS (TLS 1.3)**. Aucune donnée n'est transmise en clair entre le client iOS et l'API backend. La librairie Axios (1.x) est utilisée pour les requêtes HTTP, avec une configuration des timeouts et une gestion des erreurs réseau.

3.2 Stockage des données sensibles (Tokens)

Les données sensibles, et plus particulièrement le **JWT (jeton d'accès) et le refresh token**, sont stockées localement via le module `expo-secure-store`. Ce module utilise le **Keychain** natif d'iOS (ou Keystore sur Android), assurant que les tokens sont chiffrés au repos sur l'appareil et protégés par les mécanismes de sécurité du système d'exploitation.

3.3 Authentification

L'authentification est gérée côté backend (hors périmètre) via JWT. L'application iOS est responsable de la réception de ces tokens lors de la connexion et de leur stockage sécurisé (voir 3.2). Elle assure ensuite l'inclusion du token d'accès dans les en-têtes des requêtes authentifiées.

3.4 Protection du code

En production, le code applicatif fait l'objet de **minification et d'obfuscation**. Cette mesure vise à complexifier la rétro-ingénierie (reverse engineering) de l'application par un attaquant.

3.5 Environnement de build et déploiement (CI/CD)

Le processus de build est automatisé :

— **CI/CD** : GitHub Actions est utilisé pour les builds et tests automatisés. Les secrets nécessaires au build (clés d'API, etc.) sont protégés via les « GitHub Secrets ».

— **Build Natif** : EAS Build (Expo Cloud) est utilisé pour la génération des builds iOS signés. Les clés d'application sont stockées et chiffrées côté Expo Cloud.

4 Synthèse de la sécurité

Points clés de la sécurité frontend

- **Sécurité des communications** : Usage strict de HTTPS (TLS 1.3) pour toutes les communications avec l'API.
- **Stockage sécurisé des tokens** : Utilisation du Keychain iOS (via `expo-secure-store`) pour les tokens d'authentification (JWT).
- **Intégrité du code** : Mises à jour OTA signées (Expo Updates) et protection du code en production (minification, obfuscation).
- **Gestion des accès** : Demandes explicites des permissions iOS (micro, caméra...) et validation des entrées côté client.
- **Processus de build sécurisé** : Gestion des secrets centralisée via GitHub Actions et EAS Build pour la signature et le stockage des clés d'application.
- **Périmètre clair** : La sécurité globale de la plateforme dépend fortement de la robustesse de l'API Backend (authentification, contrôles d'accès, validation), qui est explicitement exclue du périmètre de ce client.