ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de
HONORIS UNITED UNIVERSITIES

ANNÉE UNIVERSITAIRE 2025/2026

Rapport de projet

HealLink : Plateforme mobile de santé et solidarité connectée

4 année Ingénierie Informatique et Réseaux

8 janvier 2026

Réalisé par

WARDI Anas
EZ-ZAHRAOUI El-Mehdi
AIT LAHCEN Achraf

Enseignante

Mme NAJI Zineb

Dédicaces

À ceux qui ont éclairé notre chemin vers la connaissance, on dédie ce travail avec une profonde reconnaissance et une gratitude sincère.

À nos chers parents, pilier inébranlable de notre vie, dont l'amour, le soutien moral et les sacrifices silencieux nous ont porté tout au long de ce parcours, nous donnant la force de surmonter chaque difficulté et de croire en nos capacités.

À nos enseignants, pour leur patience infinie, leurs conseils éclairés et leurs remarques constructives, qui ont su guider nos pas, stimuler notre réflexion et nourrir notre désir d'apprendre, transformant cette expérience en une véritable aventure intellectuelle.

À vous, cher lecteur, qui prenez le temps de parcourir ces pages, recevez ici toute notre reconnaissance, en espérant que ce travail saura vous apporter autant qu'il nous a apporté lors de sa rédaction.

Remerciements

Avant d'entamer ce rapport, on souhaite profiter de cette occasion pour exprimer nos profonds remerciements à toutes les personnes qui ont contribué, de près ou de loin, à sa réalisation et à son élaboration.

On tient à remercier infiniment nos enseignants en Développement mobile Mme NAJI Zineb et Mr ELBAGHAZAOUI Bahaeddine, pour leur temps, leur bienveillance et leur précieux conseils. Je remercie également l'ensemble des membres de l'École

Marocaine des Sciences de l'Ingénieur, en particulier M. ERRAJI Naoufal, Responsable de la filière, et Mme BADRITIJANE Fatimazahra, chef de département, pour le soutien et l'accompagnement dont ils nous ont fait bénéficier.

On tient également à remercier notre enseignante Mme NAJI Zineb pour le temps qu'ils consacreront à l'évaluation de ce travail et pour leurs éventuelles remarques constructives.

Enfin, on tient à adresser nos profonds et sincères remerciements à tous nos enseignants pour leur encadrement et leur dévouement.

Table des matières

Liste des figures

Introduction générale

HealLink est une application mobile complète développée avec React Native et Expo, intégrant Firebase pour l'authentification et un backend FastAPI avec MongoDB pour la gestion des données. Le projet s'inscrit dans une démarche de santé connectée et solidaire, visant à centraliser le suivi des campagnes de santé publique (vaccination, dépistage, dons de sang) et à promouvoir la solidarité communautaire via des cagnottes. L'application combine des technologies avancées telles que la géolocalisation, les WebSockets pour la communication en temps réel, et une architecture modulaire séparant le frontend mobile du backend API. Elle offre aux citoyens des fonctionnalités modernes : localisation de centres de vaccination ou de dons de sang sur carte interactive, consultation des stocks disponibles avec notifications, dons financiers via cagnottes, et interactions sociales entre utilisateurs (système de "poke" via WebSocket et QR codes).

D’un point de vue technique, HealLink utilise React Native pour une compatibilité multiplateforme (iOS et Android), Expo pour simplifier le développement et le déploiement, Firebase Auth pour une authentification robuste et sécurisée, FastAPI pour des APIs REST performantes et asynchrones, MongoDB pour une base de données flexible et scalable, et des WebSockets pour des communications bidirectionnelles en temps réel. L’architecture globale repose sur une séparation claire des responsabilités : le frontend gère l’interface utilisateur et les interactions locales, tandis que le backend assure la logique métier, la persistance des données et la communication temps réel.

L’objectif principal de HealLink est de favoriser la santé publique en facilitant l’accès aux services de santé et en encourageant la solidarité sociale. Par exemple, les utilisateurs peuvent localiser des centres de vaccination proches via une carte interactive, recevoir des alertes lorsque les stocks de vaccins ou de poches de sang sont faibles dans leur zone, et contribuer financièrement à des cagnottes pour aider des associations locales. De plus, la fonctionnalité de scanning QR permet des interactions sociales, comme envoyer des “pokes” à d’autres utilisateurs connectés via WebSocket, pour promouvoir l'engagement communautaire.

Le projet démontre une intégration réussie de technologies modernes pour répondre à des besoins sociétaux importants, en combinant l’efficacité technique avec une expérience utilisateur intuitive et accessible.

- Le chapitre 1 pose le cadre du projet en introduisant le contexte général, les objectifs visés, ainsi que les choix technologiques retenus pour sa mise en œuvre.
- Le chapitre 2 décrit les spécifications des besoins, en détaillant les exigences fonctionnelles et non fonctionnelles, et en précisant les besoins spécifiques pour chaque composant de l'application.
- Le chapitre 3 explique la conception du système, en présentant l'architecture logicielle, les modèles de données ainsi que les choix de conception adoptés.
- Le chapitre 4 traite de la réalisation du système, en détaillant les aspects techniques liés au développement, à la configuration et au déploiement de l'application, ainsi que les tests de validation effectués.

CADRE ET CONTEXTE DU PROJET

01

Introduction

Ce chapitre présente le contexte général du projet HealLink, les objectifs fonctionnels visés (suivi des campagnes sanitaires, cagnottes et interactions en temps réel), ainsi que les choix méthodologiques et technologiques adoptés pour sa mise en œuvre (React Native, FastAPI, MongoDB, Firebase et WebSocket).

Contexte et objectifs du projet

HealLink répond à un besoin croissant de digitalisation des services de santé publique et de solidarité. Dans un contexte où les campagnes de vaccination et de dépistage nécessitent un suivi précis des stocks et des localisations, l'application permet aux citoyens de localiser les centres de santé proches, de recevoir des alertes sur les stocks faibles et de contribuer financièrement via des cagnottes. Les objectifs principaux sont :

- Centraliser le suivi en temps réel des campagnes sanitaires avec visualisation des stocks disponibles
- Faciliter les dons de sang et les contributions financières via des cagnottes
- Promouvoir la solidarité communautaire via des interactions sociales (QR codes et WebSocket)
- Assurer une interface intuitive et accessible à tous les utilisateurs

Choix de la méthodologie de développement

Le développement de HealLink suit une approche agile inspirée de Scrum, avec des itérations courtes (sprints de 2 semaines) permettant une adaptation rapide aux feedbacks utilisateurs et aux évolutions des besoins. Cette méthodologie favorise la collaboration entre les équipes frontend et backend, avec des revues régulières et des déploiements incrémentaux.

La séparation claire entre frontend et backend permet une indépendance des équipes : le frontend mobile est développé en React Native avec Expo pour une compatibilité multiplateforme (iOS et Android), facilitant le développement, les tests et le déploiement sans configuration native complexe. Le backend utilise FastAPI, un framework Python moderne pour les APIs REST, offrant une performance élevée grâce à l'asynchrone (via asyncio) et une génération automatique de documentation interactive (Swagger UI). Les WebSockets sont

intégrés pour la communication temps réel, essentielle pour les notifications et les interactions QR.

MongoDB est choisi pour sa flexibilité avec les données non structurées et géospatiales (coordonnées des centres de santé), permettant des requêtes efficaces sur les localisations. Les collections dynamiques (campagnes, cagnottes, dons) s'adaptent aux évolutions sans migrations lourdes. Firebase Auth assure une authentification sécurisée et scalable, avec gestion des sessions et récupération de mot de passe.

L'architecture modulaire facilite la maintenance : le frontend gère l'UI et les interactions locales, le backend la logique métier et la persistance. Les tests sont intégrés tôt, avec des tests unitaires pour les modèles Pydantic et les utilitaires, et des tests d'intégration pour les APIs.

Figure 2 : Sprint (Scrum)

Figure 1 : Taiga

Planification du projet

La planification suit un processus agile itératif avec phases clés :

1. **Analyse des besoins** : Recueil via entretiens avec associations de santé et citoyens, définition des cas d'utilisation.
2. **Conception** : Création des diagrammes UML (classes, séquences, cas d'utilisation) pour modéliser le système.
3. **Développement itératif** : Implémentation par modules (auth, cartes, cagnottes, QR), avec intégration continue via Git.
4. **Tests** : Tests unitaires, d'intégration et manuels sur appareils mobiles.
5. **Déploiement** : Publication sur Expo et serveur backend.

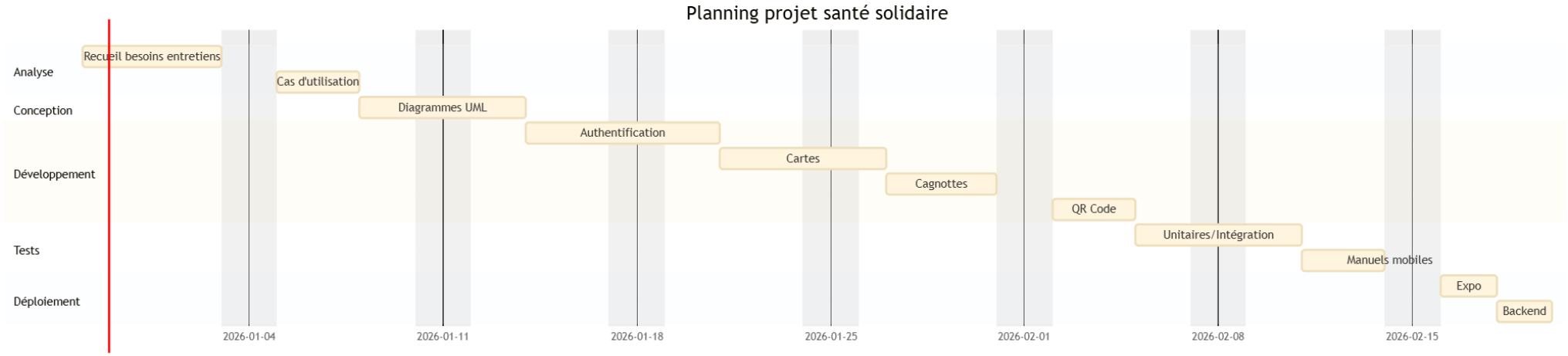

Figure 3 : Digramme de Gantt

Les jalons clés incluent : prototype fonctionnel (MVP) avec auth et carte, ajout des cagnottes, intégration QR/WebSocket, tests finaux et déploiement. La durée estimée est de 12 semaines (3 mois) pour une équipe de 3 personnes développeurs.

Les risques identifiés incluent la conformité RGPD pour les données médicales (pseudonymisation, consentement), la performance des requêtes géospatiales sur grandes bases, et la dépendance aux services externes (Firebase, MongoDB Atlas).

Risques et contraintes du projet

Risques techniques :

- Dépendance à Firebase : Risque de downtime ou changement d'API ; mitigation par fallback local.
- Latence réseau pour WebSockets : Impact sur les interactions temps réel ; optimisation via compression et cache.
- Gestion des permissions : Caméra, géolocalisation ; éducation utilisateur et gestion d'erreurs.
- Sécurité : Attaques sur APIs (injection, DDoS) ; mitigation par validation Pydantic, rate limiting, HTTPS.

Risques métier :

- Adoption par les associations : Formation et support nécessaires.
- Évolutivité : Croissance des données ; migration vers sharding MongoDB.

Contraintes :

- Compatibilité iOS/Android : Expo assure la cohérence, mais limitations sur certaines APIs natives.
- Sécurité des données utilisateur : Protection des données personnelles (email, historique de dons), conformité aux bonnes pratiques RGPD (chiffrement HTTPS, authentification sécurisée Firebase), chiffrement des données sensibles.
- Évolutivité de l'architecture : Design modulaire pour ajouter fonctionnalités (notifications push, IA pour recommandations).

· Budget et ressources : Optimisation des coûts cloud (MongoDB Atlas, Firebase).

Conclusion

HealLink répond à un besoin croissant de digitalisation des services de santé publique et de solidarité. Dans un contexte où les campagnes de vaccination et de dépistage nécessitent un suivi précis des stocks et des localisations, l'application permet aux citoyens de localiser les centres de santé proches, de consulter en temps réel les stocks disponibles et de contribuer financièrement via des cagnottes.

SPÉCIFICATION DES BESOINS

02

Introduction

Ce chapitre détaille les besoins fonctionnels et non fonctionnels de l'application HealLink, identifiés à partir des objectifs du projet, des cas d'utilisation et des contraintes techniques. Les besoins présentés visent à couvrir l'ensemble des fonctionnalités principales : campagnes sanitaires, cagnottes/dons, authentification et interactions en temps réel.

Besoins fonctionnels

Les besoins fonctionnels décrivent les fonctionnalités que l'application doit offrir pour répondre aux objectifs du projet. Ils sont organisés par modules principaux :

Authentification et gestion des utilisateurs

- **Inscription utilisateur** : Création de compte avec email, mot de passe (minimum 6 caractères), confirmation du mot de passe. Validation des champs obligatoires et gestion des erreurs Firebase (ex. : email déjà utilisé, mot de passe faible).
- **Connexion** : Authentification via email/mot de passe avec gestion des états de session via Firebase onAuthStateChanged.
- **Récupération de mot de passe** : Envoi d'email de réinitialisation via Firebase.

Visualisation et consultation des campagnes sanitaires

- **Affichage sur carte** : Utilisation de React Native Maps pour afficher les centres de santé (vaccination, dons de sang) avec marqueurs géolocalisés (latitude/longitude).
- **Filtres dynamiques** : Recherche textuelle par nom de centre, filtrage par catégorie (vaccination, dépistage), dates de début/fin, proximité géographique.
- **Détails des campagnes** : Affichage des stocks disponibles (vaccins, poches de sang), horaires, informations de contact.

Notifications et alertes en temps réel

- **Alertes stocks faibles** : Notifications automatiques via WebSocket lorsque les stocks d'un centre proche descendent en dessous d'un seuil (ex. : < 10 vaccins).
- **Mécanisme de notification** : Connexion WebSocket persistante, envoi de messages JSON avec type "notification" et données pertinentes.

Gestion des cagnottes et dons

- **Consultation des cagnottes** : Liste des cagnottes actives avec titre, description, objectif, collecte actuelle, pourcentage atteint.

- **Faire un don** : Interface pour saisir un montant (prédéfinis : 50, 100, 200 DHS ou personnalisé), ajout d'un message optionnel. Mise à jour en temps réel de la cagnotte via API POST.

Interactions sociales via QR codes

- **Scanning QR** : Utilisation d'Expo Camera pour scanner des QR codes d'autres utilisateurs, déclenchant un envoi via WebSocket (type "poke").
- **Affichage QR personnel** : Génération d'un QR code unique basé sur l'ID utilisateur (UUID), affichable dans l'écran "Mon QR".
- **Réception de pokes** : Alerte native (Alert.alert) lors de réception d'un message WebSocket de type "poke".

Fonctionnalités avancées

- **Géolocalisation** : Demande de permissions pour accéder à la position GPS, calcul des distances pour prioriser les centres proches.

Besoins non fonctionnels

Les besoins non fonctionnels définissent les qualités requises pour l'application, au-delà des fonctionnalités pures :

Performance

- **Temps de réponse API** : objectif < 2 secondes pour les requêtes courantes (GET/POST), grâce à FastAPI asynchrone et MongoDB.
- **Temps de chargement UI** : objectif < 3 secondes pour l'affichage initial de la carte, selon le volume de marqueurs.

Sécurité

- **Authentification** : Firebase Authentication (tokens Firebase/JWT) et bonnes pratiques contre brute force.
- **Communications** : WebSocket utilisé pour des notifications internes en temps réel; en production, le serveur sera exposé en HTTPS et les connexions passeront en WSS.
- **Validation** : validation côté backend avec Pydantic + contrôles côté frontend.
- **RGPD (exigence)** : consentement pour caméra/géolocalisation et bonnes pratiques de minimisation des données ; suppression/export à prévoir.

Accessibilité

- **Interface adaptée mobiles** : Design responsive avec KeyboardAvoidingView, tailles de police adaptées, couleurs contrastées.
- **Support multilingue** : Interface en français, extensible à d'autres langues via i18n.
- **Accessibilité native** : Support des lecteurs d'écran, gestes tactiles standards.

Disponibilité et fiabilité

- **Déploiement flexible** : Backend déployable localement (Uvicorn) ou sur cloud (Heroku, AWS), frontend via Expo (iOS/Android/Web).
- **Tolérance aux pannes** : Une stratégie de reconnexion est prévue (ou mise en place) en cas de perte de connexion WebSocket ; un fallback HTTP polling peut être ajouté en évolution.
- **Journalisation** : logs côté backend et frontend pour faciliter le diagnostic et la maintenance.

Maintenabilité et évolutivité

- **Code modulaire** : Séparation frontend/backend, composants réutilisables (ex. : utils pour WebSocket, erreurs Firebase).
- **Documentation** : APIs documentées automatiquement par FastAPI (Swagger), code commenté.
- **Évolutivité** : Architecture permettant l'ajout de fonctionnalités (ex. : notifications push, IA pour recommandations).

Conclusion

Les besoins spécifiés couvrent le cycle utilisateur essentiel de l'application HealLink, de l'authentification à la consultation des campagnes sanitaires, en passant par la gestion des cagnottes, des dons et les interactions en temps réel via WebSocket/QR code. Ils s'appuient sur les composants effectivement réalisés (écrans React Native) et sur les services backend exposés (API campagnes, cagnottes/dons et WebSocket), garantissant une expérience cohérente et sécurisée.

CONCEPTION DU SYSTÈME

03

Introduction

La conception de HealLink repose sur une architecture modulaire et une séparation claire entre le frontend mobile et le backend. Le frontend (React Native) gère l'interface utilisateur, la navigation et les interactions locales (géolocalisation, scan QR), tandis que le backend (FastAPI) assure la logique métier, l'accès aux données MongoDB et la communication temps réel via WebSocket. Cette approche facilite la maintenabilité, les évolutions et le déploiement de l'application.

Acteurs et rôles

Le système considère un seul acteur principal :

- Utilisateur authentifié : utilisateur ayant accès à l'ensemble des fonctionnalités de l'application.

Précondition générale : l'utilisateur doit être authentifié via Firebase Authentication avant d'accéder aux fonctionnalités principales de HealLink.

Description des cas d'utilisation

Les cas d'utilisation principaux, issus du diagramme, sont les suivants :

- UC1 – Consulter les campagnes médicales : l'utilisateur consulte la liste des campagnes disponibles et leurs informations (centre, catégorie, dates).
- UC2 – Visualiser sur carte *(include)* : l'utilisateur visualise les centres sur une carte interactive avec des marqueurs géolocalisés.
- UC3 – Filtrer les campagnes *(extend)* : l'utilisateur applique des filtres (ex. catégorie) afin d'affiner l'affichage des campagnes.
- UC4 – Consulter les stocks *(include)* : l'utilisateur consulte les stocks disponibles (vaccins, poches de sang) associés à une campagne.
- UC5 – Consulter les cagnottes : l'utilisateur consulte les cagnottes actives et leur progression (objectif, collecte, pourcentage).
- UC6 – Faire un don *(include → consulter cagnottes)* : l'utilisateur effectue un don (montant prédéfini 50/100/250 DH ou personnalisé) et le backend met à jour la collecte.

- UC7 – Générer un QR code : l'utilisateur affiche son identifiant sous forme de QR code pour permettre une interaction.
- UC8 – Scanner un QR code *(include → recevoir notification)* : l'utilisateur scanne le QR d'un autre utilisateur et déclenche une interaction temps réel.
- UC9 – Recevoir une notification (poke) : l'utilisateur reçoit une notification interne via WebSocket (ex. alerte/confirmation "Success")

Modélisation des relations «include» et «extend»

- «include» est utilisé quand une fonctionnalité fait partie systématiquement d'un autre cas d'utilisation (ex. visualisation carte et consultation des stocks lors de la consultation d'une campagne).
- «extend» est utilisé quand une fonctionnalité est optionnelle et dépend d'un choix utilisateur (ex. filtrer les campagnes)

Diagramme global des cas d'utilisation

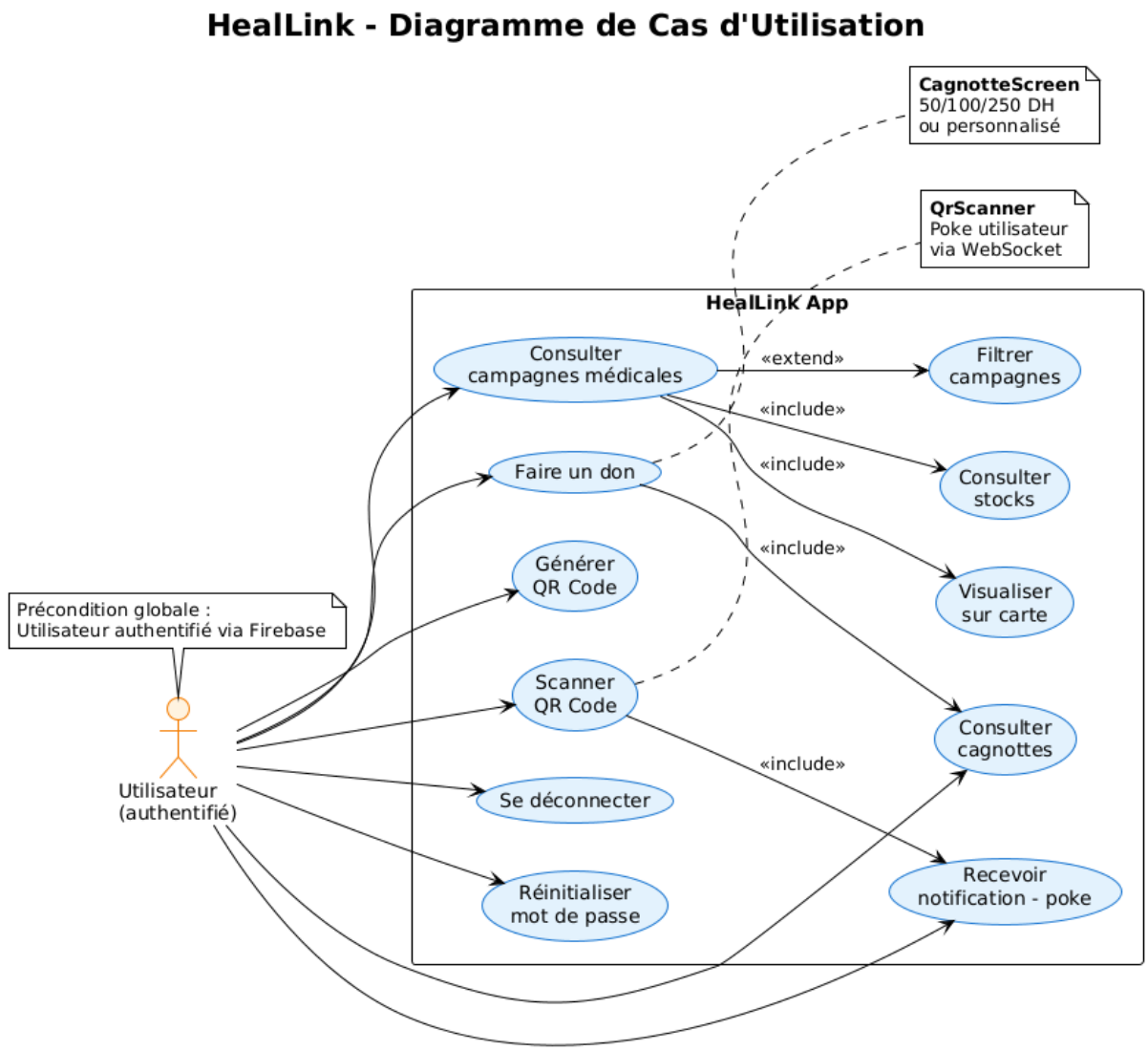

Figure 4 : Diagramme de cas d'utilisation

Ce diagramme présente les fonctionnalités principales de HealLink accessibles à l'utilisateur authentifié (précondition : authentification Firebase), notamment la consultation des campagnes, la gestion des cagnottes/dons et les interactions via QR code et WebSocket.

Modélisation dynamique

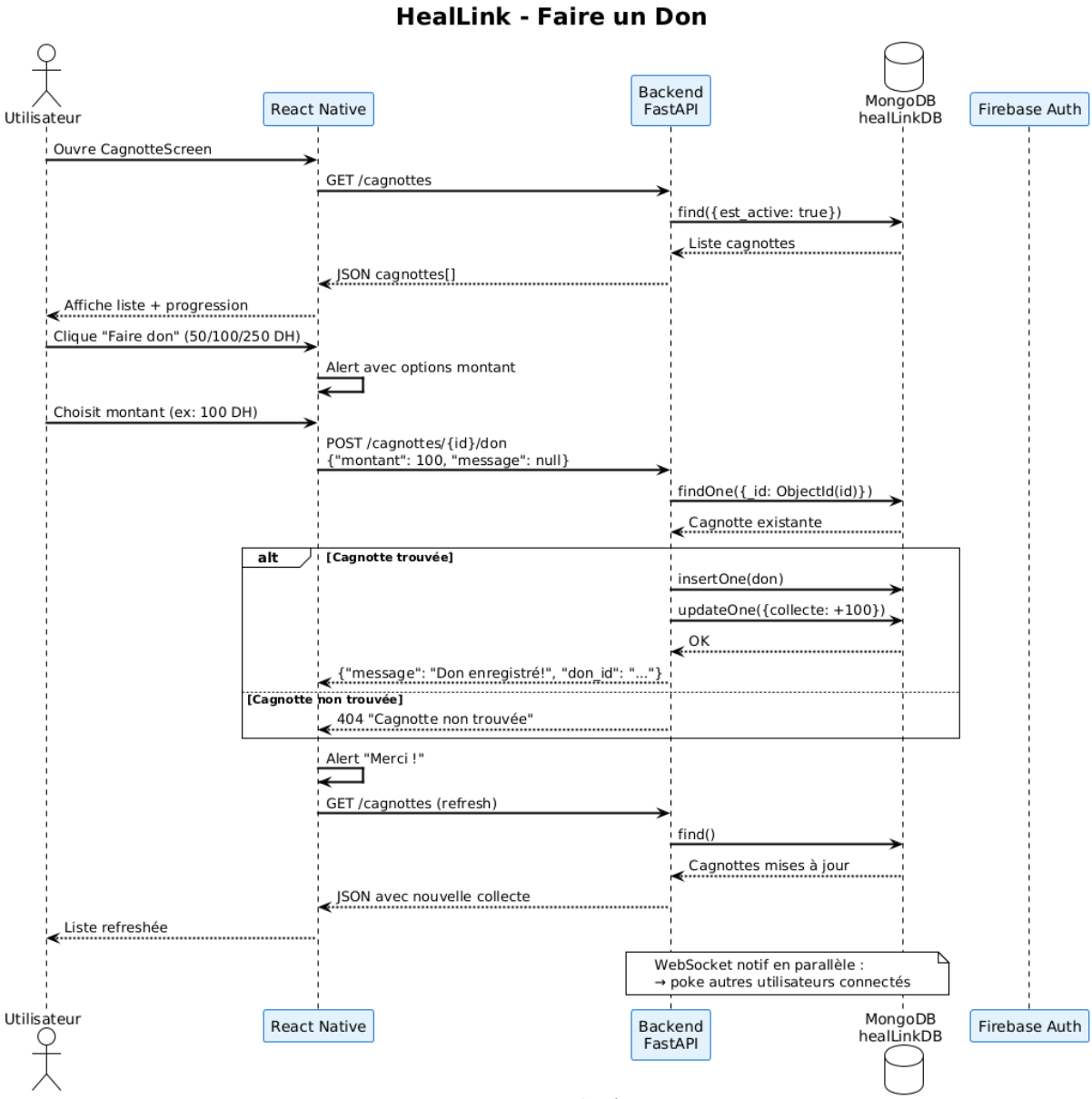

Figure 5 : Diagramme de séquences

Le diagramme de séquence illustre le déroulement du cas d'utilisation "Faire un don", depuis la consultation des cagnottes jusqu'à l'enregistrement du don et la mise à jour de la collecte côté base de données.

Modélisation statique

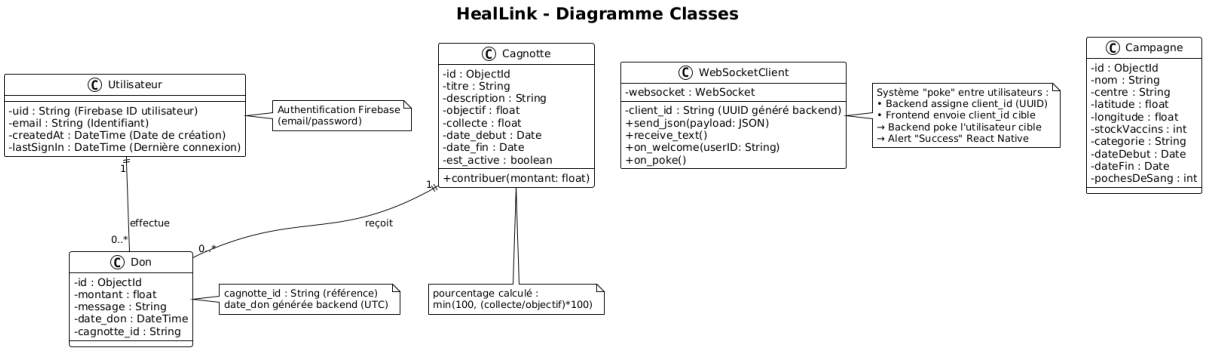

Figure 6 : Diagramme de classes

Le diagramme de classes décrit les entités manipulées par le système (Campagne, Cagnotte, Don, Utilisateur) ainsi que leurs relations, permettant de comprendre la structure des données persistées dans MongoDB.

Architecture de l’application

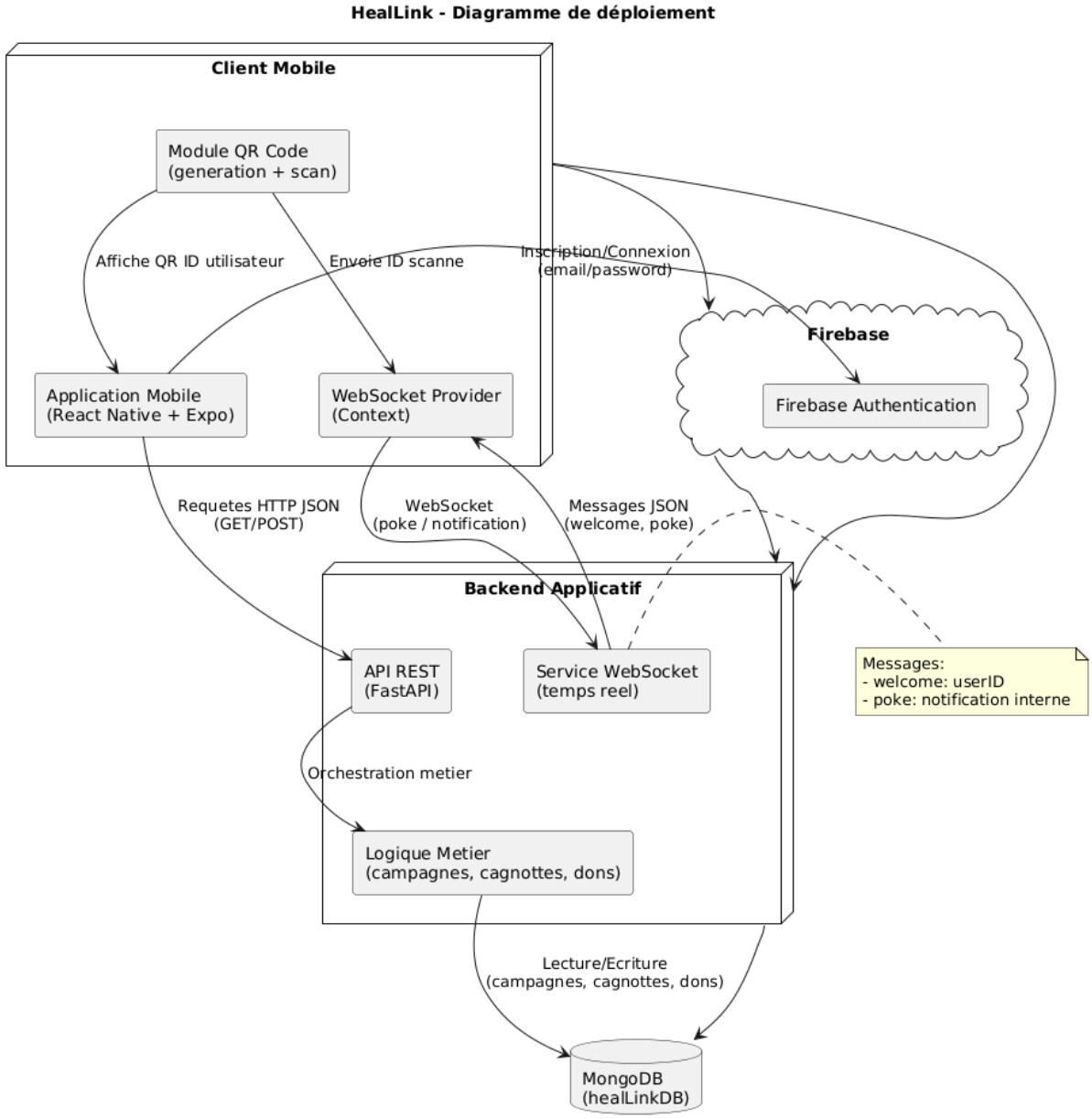

Figure 7 : Diagramme de déploiement

Ce diagramme présente l'architecture de déploiement de HealLink, illustrant la répartition des composants entre le client mobile (React Native/Expo), le backend applicatif (FastAPI avec API REST et WebSocket) et les services externes (Firebase Authentication pour l'authentification, MongoDB pour la persistance des données). Cette architecture modulaire facilite la séparation des responsabilités : le frontend gère l'interface utilisateur et les interactions locales (géolocalisation, QR code), tandis que le backend centralise la logique métier, l'accès aux données et la communication temps réel via WebSocket.

Architecture de sécurité

L'application HealLink met en œuvre plusieurs mécanismes pour garantir la sécurité des données et des communications :

- Authentification sécurisée : Firebase Authentication gère l'inscription et la connexion des utilisateurs (email/mot de passe) avec stockage sécurisé des identifiants et génération de tokens.
- Validation des données : le backend FastAPI utilise Pydantic pour valider toutes les entrées (types, formats, valeurs obligatoires), limitant ainsi les risques d'injection et de données corrompues.
- Communications sécurisées : les échanges entre le client et le backend utilisent HTTP/JSON pour les APIs REST. Le WebSocket assure les notifications internes en temps réel (poke entre utilisateurs). En production, l'ajout de HTTPS et WSS est prévu pour sécuriser les communications.
- Gestion des erreurs : côté backend, les erreurs sont gérées explicitement (HTTPException) et les logs permettent le diagnostic sans exposer d'informations sensibles.

Conclusion

Ce chapitre a présenté la conception de HealLink à travers une approche modulaire (cas d'utilisation, modèles de classes, séquences dynamiques et architecture de déploiement). La séparation claire entre frontend mobile et backend API, combinée à l'utilisation de Firebase pour l'authentification et MongoDB pour la persistance, assure une base solide pour la réalisation technique détaillée dans le chapitre suivant.

RÉALISATION DU SYSTÈME

04

Introduction

L'implémentation de HealLink repose sur une architecture modulaire et scalable, utilisant des technologies modernes pour assurer performance, sécurité et maintenabilité. Le backend est développé en Python avec FastAPI, offrant des APIs REST asynchrones et une intégration WebSocket pour la communication temps réel. Le frontend utilise React Native avec Expo pour une compatibilité multiplateforme. L'implémentation suit les principes SOLID (**Single Responsibility Principle, Open/Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, Dependency Inversion Principle**), avec une séparation claire des responsabilités entre couches (présentation, logique métier, persistance), et respecte le périmètre fonctionnel défini dans le cahier des charges.

Technologies utilisées

Frontend

- **React Native** : Framework pour le développement d'applications mobiles natives en JavaScript/TypeScript, permettant une base de code unique pour iOS et Android.
- **Expo** : Plateforme pour simplifier le développement React Native, offrant des APIs natives (caméra, géolocalisation) sans configuration complexe.
- **Firebase Auth** : Service d'authentification Google pour gestion sécurisée des utilisateurs (inscription, connexion, récupération mot de passe).
- **React Navigation** : Bibliothèque pour la navigation entre écrans, avec support des onglets et piles de navigation.
- **Expo Camera** : Module pour l'accès à la caméra et le scanning QR.
- **React Native Maps** : Composant pour l'affichage de cartes interactives avec marqueurs géolocalisés.
- **WebSocket Context** : Gestion centralisée des connexions WebSocket (React Context API).

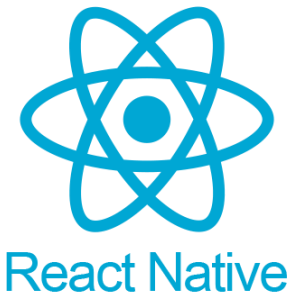

Figure 8 : React Native

Figure 9 : Firebase

Figure 10 : Expo

Backend

- **FastAPI** : Framework Python moderne pour APIs REST, avec support asynchrone via asyncio, validation automatique des données avec Pydantic, et génération de documentation Swagger.
- **Motor** (AsyncIOMotorClient) : Driver MongoDB asynchrone pour opérations non bloquantes, permettant des opérations de base de données non bloquantes.
- **WebSockets** : Protocole intégré à FastAPI pour communications bidirectionnelles temps réel (notifications, interactions QR).
- **Pydantic** : Bibliothèque pour validation et sérialisation des données, assurant la conformité des modèles (Campagne, Cagnotte, Don).

Figure 11 : FastAPI

Figure 12 : Python

Figure 13 : Pydantic

Figure 14 : WebSockets

Base de données

- **MongoDB** : Base NoSQL flexible, avec collections pour campagnes, cagnottes et dons. Support des données géospatiales (latitude/longitude) et des requêtes asynchrones.

Figure 15 : MongoDB

Outils de développement

- **Uvicorn** : Serveur ASGI pour le déploiement du backend FastAPI.
- **Git** : Contrôle de version pour la collaboration et l'intégration continue.
- **Virtualenv** : Environnements virtuels Python pour isolation des dépendances.

Figure 17 : Uvicorn

Figure 18 : GitHub

Figure 16 : Git

Structure de l'application

Architecture globale

L'application suit une architecture en couches :

- **Couche présentation (Frontend)** : Gestion de l'UI et des interactions utilisateur.
- **Couche logique (Backend)** : Traitement des requêtes, validation, logique métier.
- **Couche persistance (MongoDB)** : Stockage et récupération des données.

Structure du backend (d'après main.py)

Le fichier *backend/main.py* définit l'application FastAPI avec les éléments suivants :

- **Configuration MongoDB** : Connexion à la base via *AsyncIOMotorClient*, avec URI configurable via variable d'environnement (*MONGO_DETAILS*).
- **Collections** : *campagnes_collection*, *cagnottes_collection*, *dons_collection* pour les entités principales.
- **Modèles Pydantic** :
  - → *Campagne* : Champs id, nom, centre, latitude, longitude, stockVaccins, categorie, dateDebut, dateFin, pochesDeSang.
  - → *Cagnotte* : Champs id, titre, description, objectif, collecte, date_debut, date_fin, est_active.
  - → *Don* : Champs id, montant, message, date_don, cagnotte_id.

· **Fonctions helper** : *campagne_helper*, *cagnotte_helper*, *don_helper* pour conversion des documents MongoDB en dictionnaires Python.

· **Routes API** :

→ *GET /campagnes* : Récupération de toutes les campagnes avec conversion via helper.

→ *POST /campagnes* : Création d'une nouvelle campagne avec insertion en base.

→ *GET /cagnottes* : Liste des cagnottes actives, avec calcul du pourcentage atteint.

→ *POST /cagnottes* : Création d'une cagnotte.

→ *POST /cagnottes/{cagnotte_id}/don* : Traitement d'un don, mise à jour de la collecte, insertion du don.

→ *GET /dons* : Historique des dons récents (limité à 20, trié par date décroissante).

· **WebSocket** : /ws avec identifiant UUID 8 caractères (str(uuid.uuid4())[:8]), dictionnaire clients (ID → WebSocket), poke vers client cible.

Structure du frontend

- **Écrans** : HomeScreen (carte des centres), CagnotteScreen (liste et dons), LoginScreen (auth), QRScreen (scanning et affichage QR).
- **Utils** : Fonctions pour WebSocket (connexion, envoi de messages), gestion des erreurs Firebase.
- **Config** : firebaseConfig.js pour initialisation Firebase Auth.

Implémentation des modules principaux

Authentification

- **Backend** : Pas d'implémentation directe dans main.py (déléguée à Firebase), mais validation des tokens JWT si nécessaire.
- **Frontend** : Utilisation de *onAuthStateChanged* pour surveiller l'état de connexion, redirection automatique vers LoginScreen si non authentifié.

Géolocalisation

- **Permissions** : Demande d'accès à la localisation via Expo Location.

- **Affichage** : React Native Maps avec marqueurs (latitude/longitude), tri/filtres côté client.

APIs REST

- **Campagnes** : Endpoint GET pour récupération, avec possibilité de filtres côté client (catégorie, dates).
- **Cagnottes et dons** : Gestion transactionnelle pour éviter les incohérences (insertion don + mise à jour collecte atomique).

WebSocket

- **Connexion** : Client WebSocket dans le frontend pour connexion persistante.
- **Messages** : Envoi de données JSON (ex. : {“type”: “poke”, “data”: “hey”}) lors du scanning QR.
- **Gestion clients** : Dictionnaire *clients* indexé par ID utilisateur (UUID 8 caractères), nettoyage à la déconnexion.

Conception de l’interface utilisateur

L’UI est conçue pour une expérience mobile intuitive :

· **Navigation par onglets** : Centres (carte), Cagnottes (liste), Scan QR (caméra), Mon QR (affichage personnel).

· **Carte interactive** : Marqueurs pour centres, info-bulles avec détails, filtres par catégorie/dates.

· **Écrans d’authentification** : Formulaires simples avec validation en temps réel.

Figure 19 : Démarrage de l'application

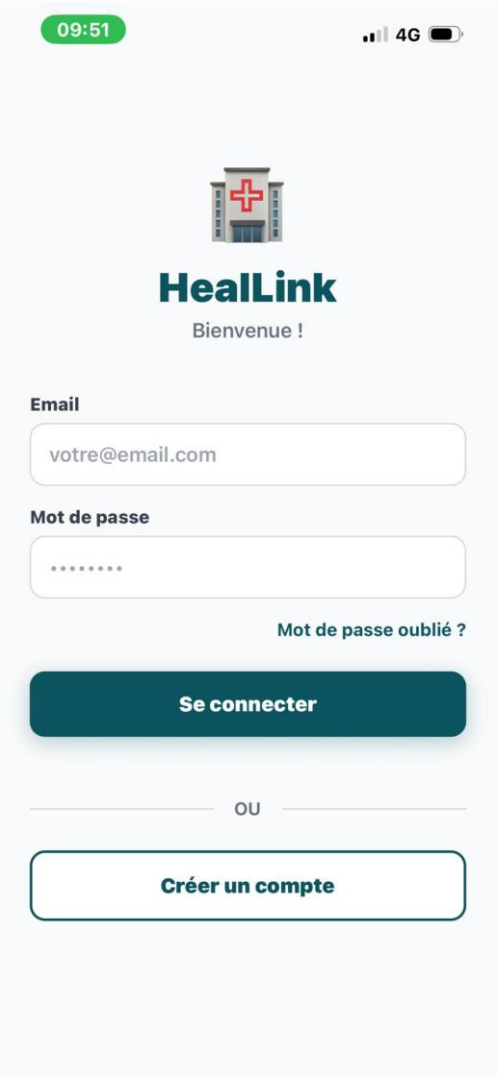

Figure 20 : Screen Login

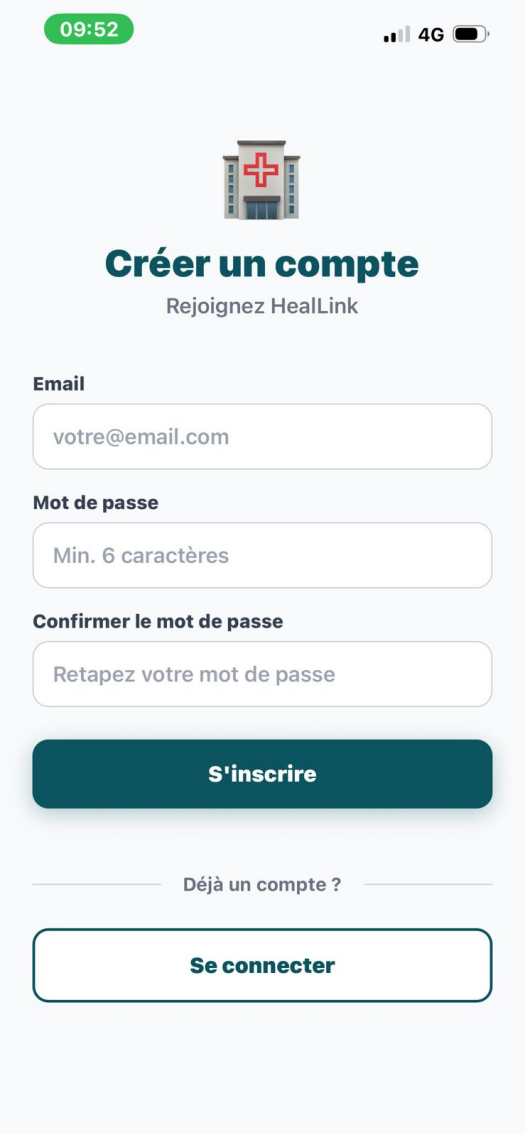

Figure 21 : Screen SignIn

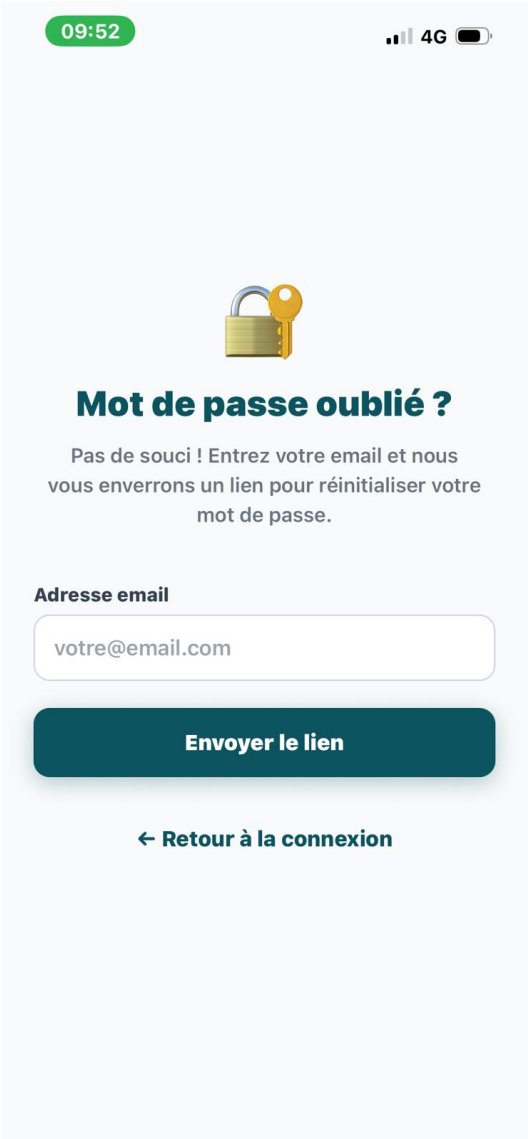

Figure 22 : Screen ForgotPassword

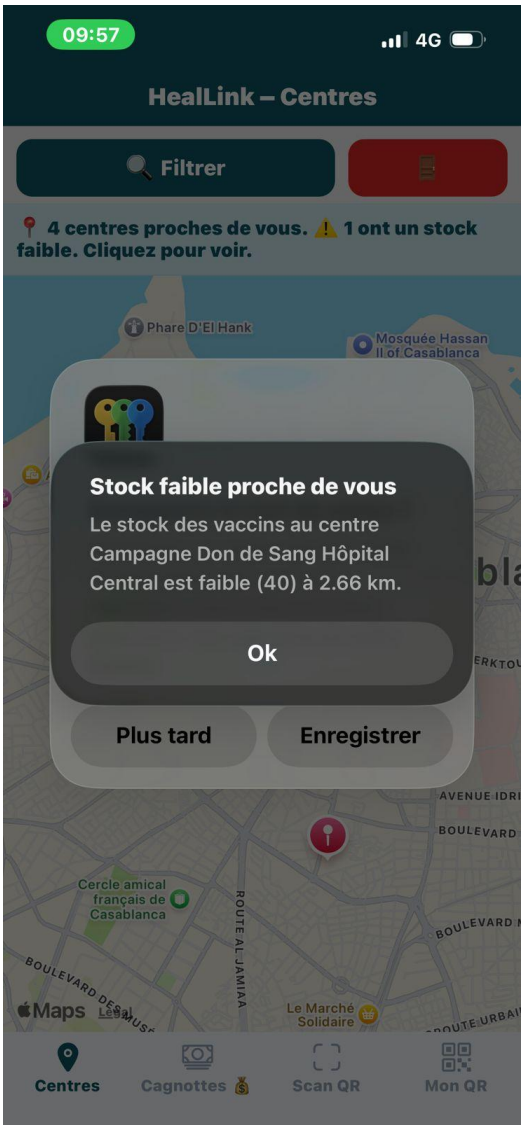

Figure 23 : Notification

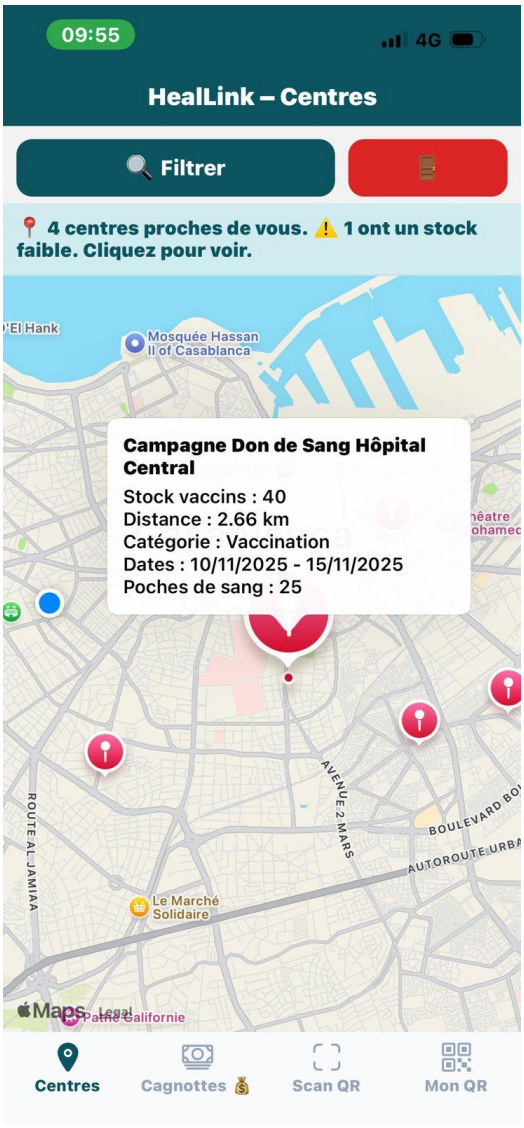

Figure 24 : Map et Campagnes

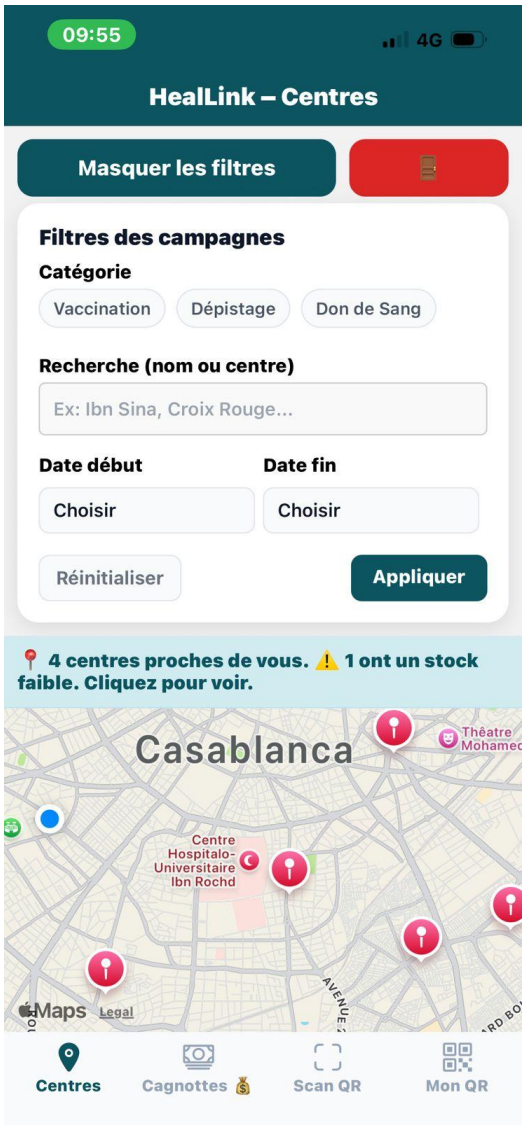

Figure 26 : Filtrage campagnes

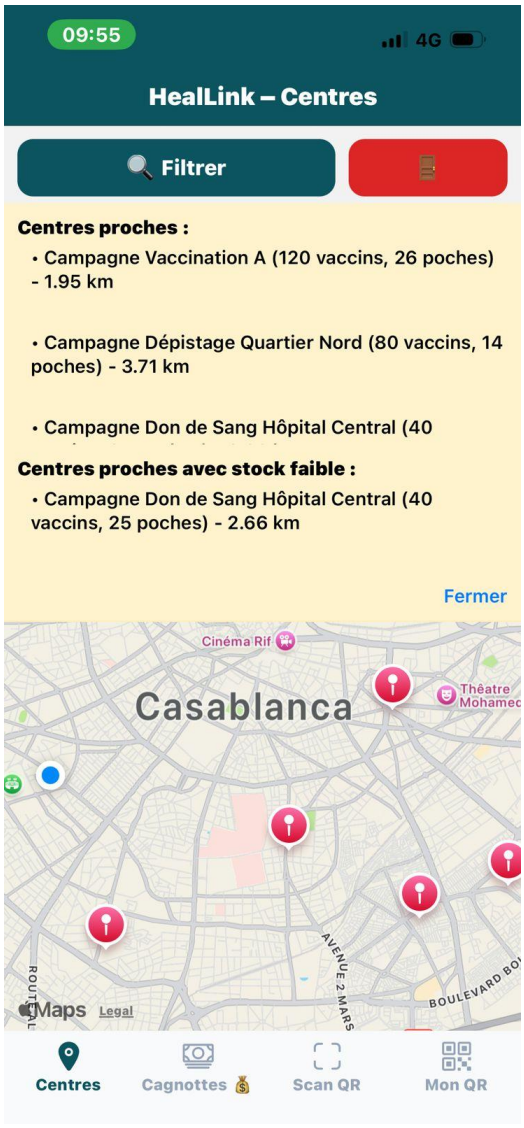

Figure 25 : Alertes campagnes

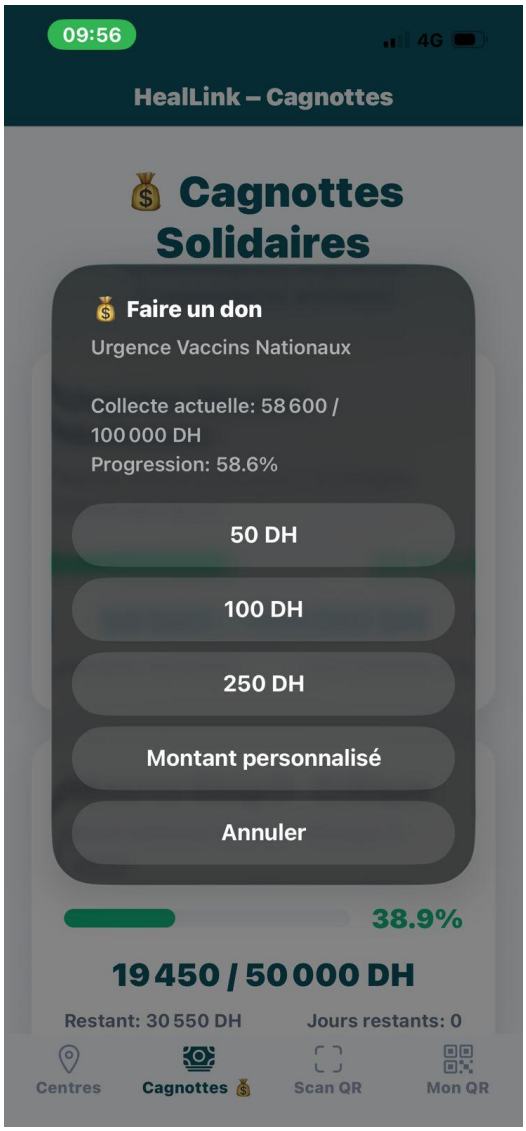

Figure 28 : Don

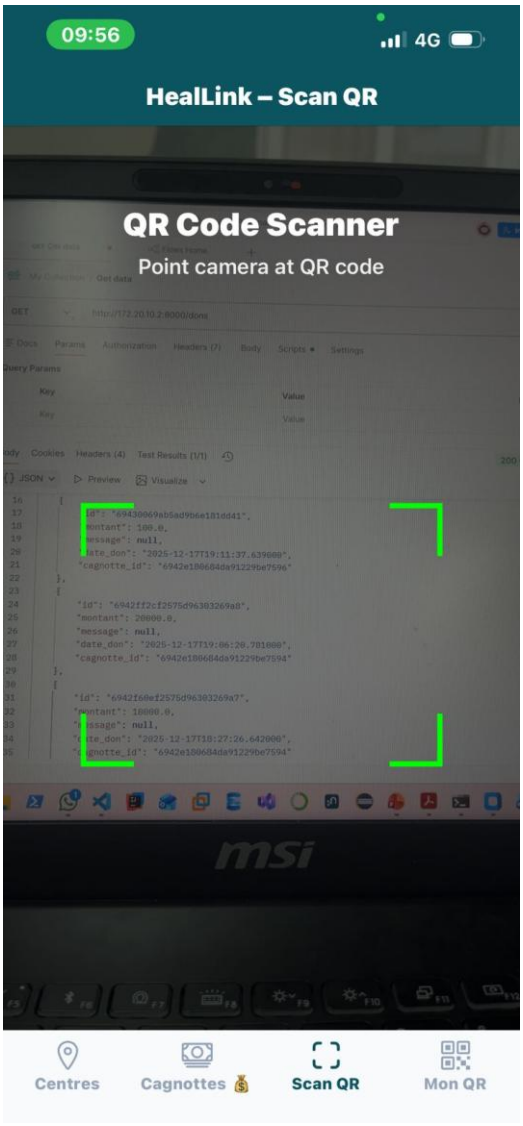

Figure 27 : QR Scanner

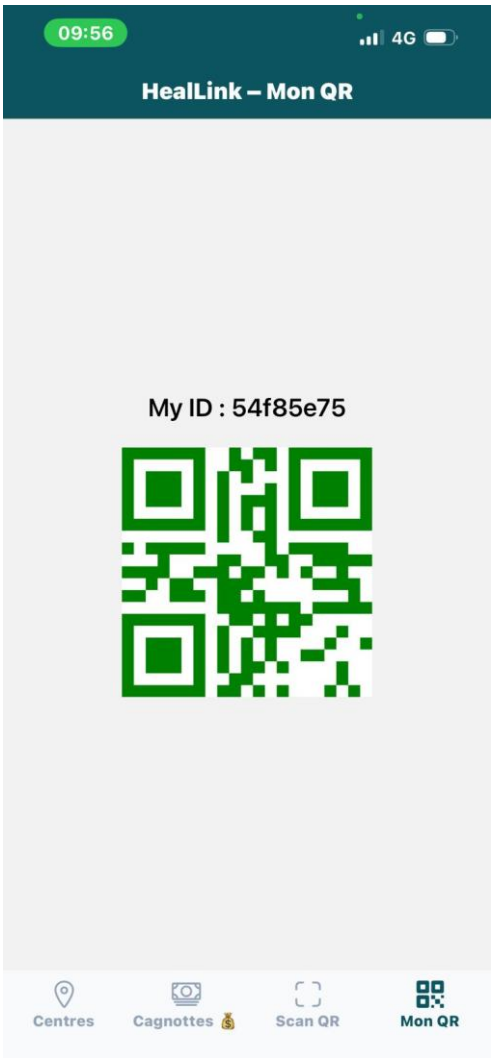

Figure 30 : QR ID

Figure 29 : Cagnottes

Figure 31 : Validation Don

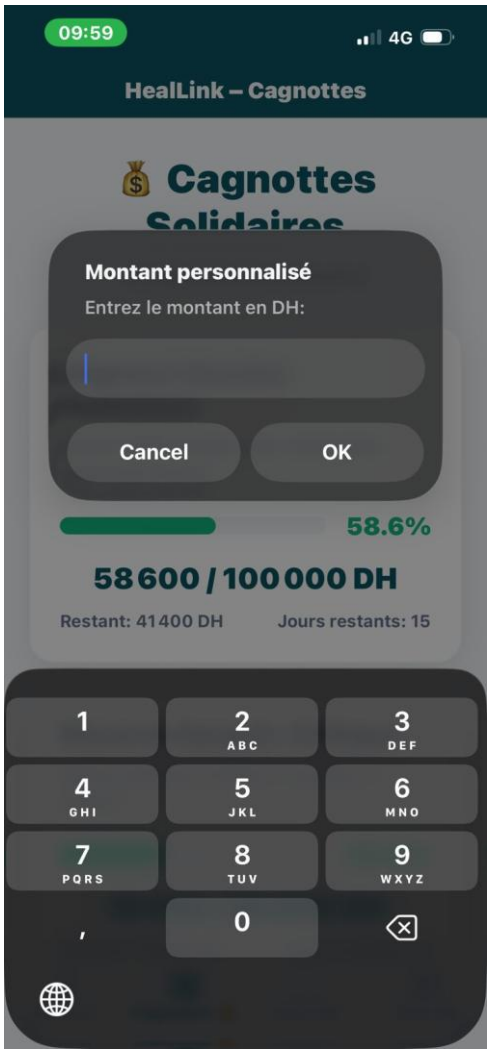

Figure 32 : Insertion montant don

Intégration Firebase

- **Configuration** : Fichier firebaseConfig.js avec clés API, projectId, etc.
- **Auth**: Inscription/connexion via *createUserWithEmailAndPassword*, *signInWithEmailAndPassword*.
- **Potentiel stockage** : Stockage : Firebase Auth uniquement (identifiants), données métier en MongoDB (campagnes, cagnottes, dons).

Tests d'intégration

- **APIs** : Tests avec Postman pour vérification des réponses JSON.
- **WebSocket** : Connexion via outils comme WebSocket King, envoi de messages simulés.

Conclusion

L’implémentation respecte les spécifications avec une séparation claire des responsabilités, utilisant les meilleures pratiques (asynchrone, validation, modularité). Le code backend (main.py) démontre une intégration réussie de FastAPI, MongoDB et WebSockets pour une application temps réel performante.

TESTS ET VALIDATION

05

Introduction

Ce chapitre détaille la stratégie de test adoptée pour valider les fonctionnalités de HealLink, en combinant tests manuels, unitaires et d'intégration. Les tests visent à assurer la conformité aux spécifications fonctionnelles et non fonctionnelles, en vérifiant la robustesse de l'application face aux erreurs et aux performances. Les tests ont été menés sur des appareils mobiles réels (iOS et Android) et via des outils de développement (Postman pour APIs, simulateurs pour WebSocket).

Stratégie de test

La stratégie de test s'appuie sur des tests manuels fonctionnels (parcours utilisateur complet) et des tests d'intégration API via Postman, couvrant l'ensemble des fonctionnalités principales de HealLink. Cette approche pragmatique permet de valider la conformité aux spécifications et d'identifier les problèmes d'intégration en conditions réelles.

Tests unitaires

- **Backend** : Validation des modèles Pydantic (Campagne, Cagnotte, Don) avec des données valides/invalides. Tests des fonctions helper (campagne_helper, etc.) pour conversion correcte des documents MongoDB.
- **Frontend** : Tests des utilitaires (calcul distances, gestion erreurs Firebase) avec Jest et React Native Testing Library.

Tests d'intégration

- **APIs REST** : Tests des endpoints FastAPI avec Postman, vérifiant les réponses JSON, codes HTTP (200, 400, 404), et effets de bord (insertion en base).
- **Base de données** : Vérification des insertions/mises à jour atomiques (ex. : don + mise à jour cagnotte).

Tests manuels

Parcours utilisateur complet sur appareils mobiles réels :

- **Authentification** : inscription/connexion Firebase, erreurs (email existant, mot de passe faible), réinitialisation
- **Navigation** : transitions entre écrans (Login → Home → Cagnotte → QR Scanner → Mon QR)

- **Campagnes** : affichage liste/carte, filtres catégorie, détails centres (stocks, coordonnées)
- **Cagnottes** : consultation liste, dons (50/100/250 DH + personnalisé), historique dons
- **QR/WebSocket** : génération QR personnel, scan QR autre utilisateur, réception poke (Alert.alert)
- **Géolocalisation** : permissions GPS, affichage marqueurs carte, fallback sans localisation

Scénarios d'erreur testés :

- Permissions refusées (caméra, localisation)
- Réseau déconnecté (erreurs API/WebSocket)
- Données invalides (montant négatif, cagnotte inexistante)

Environnements de test

- **Développement** : Tests locaux avec Expo dev client, backend Uvicorn local.
- **Production-like** : Simulation avec données réelles (centres de santé fictifs).

Résultats des tests

Les tests ont révélé une application globalement fonctionnelle, avec quelques bugs mineurs corrigés. Voici un résumé détaillé :

Authentification

- **Succès** : Inscription/connexion réussies avec Firebase, gestion états onAuthStateChanged.
- **Échecs** : Gestion erreurs pour emails déjà utilisés ou mots de passe faibles (corrigé par alertes explicites).
- **Performance** : Temps connexion < 1s.

Carte et campagnes

- **Succès** : Affichage marqueurs géolocalisés, filtres opérationnels (catégorie, dates), détails centres affichés.

Cagnottes et dons

- **Succès** : Mise à jour cagnottes en temps réel via WebSocket.
- **Performance** : Temps don < 2s.

QR et WebSocket

- **Succès** : Scanning QR fonctionnel, envoi pokes, réception alertes.
- **Performance** : Latence < 500ms pour pokes.

Géolocalisation

- **Succès** : Calcul distances précis, tri centres proches.
- **Échecs** : Erreurs permissions caméra (corrigé par éducation utilisateur).

Conclusion

Les tests manuels et d'intégration via Postman ont validé la conformité des fonctionnalités principales de HealLink aux spécifications du cahier des charges. L'application est fonctionnelle sur appareils mobiles réels, avec une expérience utilisateur fluide et une intégration réussie entre frontend React Native, backend FastAPI et services externes (Firebase, MongoDB). Des optimisations restent possibles pour une mise en production (HTTPS/WSS, tests automatisés).

DÉPLOIEMENT ET MAINTENANCE

06

Introduction

Ce chapitre présente la mise en œuvre pratique du déploiement de HealLink ainsi que les stratégies de maintenance et de suivi prévues. L'application est actuellement déployable en environnement local de développement, avec une configuration simple et des commandes standardisées pour démarrer le frontend (Expo) et le backend (Uvicorn). Les pratiques de déploiement cloud (MongoDB Atlas, Render/Heroku) et les mesures de sécurité en production (HTTPS/WSS) sont également abordées, en vue d'une évolution vers une mise en production complète.

Déploiement actuel (local)

- Backend : uvicorn main:app --host 0.0.0.0 --port 8000 (local accessible réseau LAN).
- MongoDB : Instance locale (mongodb://localhost:27017) ou MongoDB Atlas (cloud gratuit).
- Frontend : expo start (simulateur + Expo Go sur mobile réel).
- Variable d'environnement : MONGO_DETAILS configurable pour local/Atlas.

Déploiement cloud (prévu)

- Backend: Render/Railway/Heroku avec variables d'environnement (MONGO_DETAILS).
- MongoDB: MongoDB Atlas (tier M0 gratuit).
- Frontend: Expo EAS Build (APK/iOS) ou publication Expo store.

Firebase

- Configuration : Projet Firebase créé, firebaseConfig.js avec API key et projectId.
- Auth : Activé (email/password), fonctionnel en local.
- Évolutions : Firestore/Analytics non utilisés dans V1, prévus en extension.

Suivi et maintenance

- Logs : Backend (print() console), frontend (console.log()).
- Mises à jour : Git pour versioning, git pull + redémarrage services.

Risques et limitations

- Sécurité : Pas d'HTTPS/WSS en local, obligatoire en production.
- Évolutivité : Pas de rate limiting, authent backend à ajouter.
- Monitoring : Logs basiques, Grafana/Prometheus prévu en prod.

Conclusion

Le déploiement local est opérationnel et simple (Uvicorn + Expo), avec une configuration cloud facilement envisageable (variables d'environnement, MongoDB Atlas). La maintenance repose sur Git et les logs intégrés. Une sécurisation HTTPS/WSS et une authentification backend restent nécessaires pour une mise en production.

Conclusion générale

Le projet HealLink représente une réalisation complète et cohérente d'une application mobile collaborative répondant aux enjeux de santé publique et de solidarité au Maroc. À travers une analyse rigoureuse des besoins (Chapitre 2), une conception modulaire validée par des diagrammes UML (Chapitre 3), et une implémentation technique maîtrisée (Chapitre 4), l'application intègre avec succès les fonctionnalités principales identifiées dans le cahier des charges : suivi géolocalisé des campagnes sanitaires (vaccination, dépistage, dons de sang), gestion transparente des cagnottes solidaires avec dons transactionnels, et interactions communautaires innovantes via QR code et WebSocket (système de "poke").

L'architecture technique retenue démontre une compréhension approfondie des enjeux de développement moderne : React Native + Expo pour une compatibilité multiplateforme native (iOS/Android), FastAPI avec APIs REST asynchrones et WebSocket pour la performance temps réel, MongoDB pour la flexibilité NoSQL des données géospatiales, et Firebase Authentication pour une authentification sécurisée et scalable. Les tests d'intégration (Postman) et manuels exhaustifs (Chapitre 5) ont validé la robustesse des composants critiques : authentification, parcours utilisateur, transactions dons/collectes, et interactions QR/WebSocket.

Le déploiement local opérationnel (Uvicorn + Expo) et les pratiques de maintenance définies (Git, logs, variables d'environnement) constituent une base solide pour une migration cloud (MongoDB Atlas, Render/Heroku) avec sécurisation HTTPS/WSS. Les limites identifiées authentification backend stateless, tests automatisés partiels, absence de notifications push dessinent un plan d'évolution clair pour une version 2.0.

HealLink illustre avec succès l'intégration de technologies full-stack modernes dans une solution concrète répondant à des besoins sociétaux prioritaires : digitalisation des services de santé publique et promotion de la solidarité communautaire. Ce projet témoigne d'une maîtrise des concepts d'ingénierie logicielle (SOLID, UML, tests, déploiement) et ouvre des perspectives d'évolution (rôles avancés, IA pour recommandations, notifications push) pour un impact réel auprès des citoyens et associations marocaines.

Bibliographie et Webographie

[1] V. Pimentel et B. G. Nickerson, « Communicating and Displaying Real-Time Data with WebSocket », *IEEE Internet Comput.*, vol. 16, nº 4, p. 45-53, juill. 2012, doi: 10.1109/MIC.2012.64.

[2] N. Lamdouar Bouazzaoui, « Évolution du calendrier vaccinal au Maroc », *Bull. Académie Natl. Médecine*, vol. 190, nº 4, p. 1017-1033, avr. 2006, doi: 10.1016/S0001-4079(19)33247-9.

[3] « Expo Documentation », Expo Documentation. Consulté le: 31 décembre 2025. [En ligne]. Disponible sur: https://docs.expo.dev

[4] « FastAPI ». Consulté le: 31 décembre 2025. [En ligne]. Disponible sur: https://fastapi.tiangolo.com/

[5] « Firebase | Google's Mobile and Web App Development Platform », Firebase. Consulté le: 31 décembre 2025. [En ligne]. Disponible sur: https://firebase.google.com/

[6] « Intention de vacciner les enfants de moins de 12 ans contre la COVID-19 chez les parents de la préfecture de Meknès (Maroc) ». Consulté le: 31 décembre 2025. [En ligne]. Disponible sur: https://iris.who.int/items/aac8d45c-5663-401f-b737-131f91084d5d

[7] « MongoDB Documentation - Homepage ». Consulté le: 31 décembre 2025. [En ligne]. Disponible sur: https://www.mongodb.com/docs/

[8] « Postman documentation overview », Postman Docs. Consulté le: 31 décembre 2025. [En ligne]. Disponible sur: https://learning.postman.com/docs/introduction/overview/

[9] « Python 3.14 documentation », Python documentation. Consulté le: 31 décembre 2025. [En ligne]. Disponible sur: https://docs.python.org/3/

[10] « React Native · Learn once, write anywhere ». Consulté le: 31 décembre 2025. [En ligne]. Disponible sur: https://reactnative.dev/