

UNIVERSITÉ DE MONTRÉAL

IFT3150 — Rapport final

Pique-Me : Montréal à la carte

Par

Tunwend-raabo Fahîma Carmen DABO (20266362)

Abdelghafour Rahmouni (20246224)

Naromba Condé (20251772)

Travail présenté à **Gena Han**

Dans le cadre du cours IFT3150 : Projet d'informatique

8 août 2025

Table des matières

1	Introduction	1
2	Analyse des besoins et étude préliminaire	3
2.1	Démarche générale	3
2.2	Exigences fonctionnelles	3
2.3	Exigences non fonctionnelles	4
2.4	Méthodologie	4
3	Conception	5
3.1	Architecture du système	5
3.2	Modèle de données	6
3.3	Choix technologiques et justifications	7
3.4	Diagrammes d'activités (flux utilisateurs)	9
3.4.1	Inscription et connexion	9
3.4.2	Recherche et filtrage de parcs	10
3.4.3	Réservation d'un emplacement	10
3.4.4	Réservation d'une activité	11
3.4.5	Ajout ou retrait d'un favori	12
3.4.6	Ajout d'un avis	13
3.4.7	Sondage post-visite	14
3.5	Prototypes et maquettes	15
4	Implémentation et réalisation technique	16
4.1	Front-end uniquement : source de vérité et synchro	16
4.2	Recherche + Carte	16
4.3	Fiche parc : actions clés	16
4.4	Réservation : flux et garanties	17
4.5	Favoris et profil	17
4.6	Navigation (Expo Router)	17
4.7	Notes d'implémentation côté code	17
4.8	Difficultés résolues	17
5	Évaluation	19
5.1	Tests unitaires et fonctionnels	19
5.2	Évaluation de performance	20
5.3	Évaluation d'utilisabilité	20
5.4	Comparaison avec les objectifs initiaux	21
6	Discussion critique	22

7 Conclusion	24
---------------------	-----------

A Annexes	27
------------------	-----------

1 Introduction

Contexte

À Montréal, les parcs font partie intégrante du quotidien : lieux de détente, de sport, de rencontres familiales ou d'événements de quartier. Lors des beaux jours, ces espaces deviennent des pôles d'activité majeurs. Pourtant, il n'existe toujours pas de moyen numérique unique et fiable pour connaître les différents espaces, se renseigner sur l'accessibilité ou la disponibilité d'un équipement avant de s'y rendre.

Il existe l'initiative *Accès nature* permet de réserver gratuitement des accès dans plus de trente parcs régionaux via le site toutlemondedehors.ca, mais cette ressource est encore méconnue du grand public et est plus centrée sur les parcs régionaux du Québec.

Problématique

Les usagers se posent invariablement les mêmes questions : *Le parc autorise-t-il les barbecues ? Est-il accessible aux personnes à mobilité réduite ? Reste-t-il des emplacements libres ? Comment trouver le parc qui répond le mieux à nos préférences ?* Ces interrogations sont loin d'être anodines.

Par exemple, à Montréal, l'utilisation de barbecues est autorisée dans certains parcs comme Angrignon, Jarry ou La Fontaine, mais elle dépend du type d'appareil (charbon ou propane) et de la signalisation sur place. De plus, des barbecues collectifs sont disponibles dans certains lieux, mais peu de gens savent où les trouver dans le parc.

En somme, les informations sont dispersées, parfois incomplètes ou obsolètes, ce qui crée une planification laborieuse et une expérience parfois décevante. Une centralisation intelligente et interactive de ces données améliorerait considérablement l'expérience des usagers.

Proposition

Pique-Me vise à concentrer toutes ces données dans une application mobile intuitive : carte interactive, fiches synthétiques de parcs, filtres personnalisés, réservation d'emplacement ou d'activités, système d'avis/badges et notifications d'événements.

Objectifs

- Concevoir une **interface mobile fluide** listant les parcs, leurs équipements et les activités qui y sont proposées.
- **Implémenter la réservation** d'emplacement/activités avec double confirmation et annulation automatique.
- Développer une **dimension communautaire** (favoris, avis, badges).

- Satisfaire des exigences de **sécurité, performance, accessibilité** et disponibilité.

2 Analyse des besoins et étude préliminaire

2.1 Démarche générale

Le recueil des besoins a débuté dans le cadre du cours **IFT2905 – Interfaces personne-machine**, où *Pique-Me* a été choisi comme projet de session. Nous avons ensuite :

- réalisé des entretiens informels avec des amis et proches utilisateurs réguliers des parcs montréalais ;
- analysé nos propres expériences de planification de pique-niques et d'activités extérieures (difficultés et attentes).

Les résultats ont été triés puis convertis en exigences fonctionnelles et non fonctionnelles.

2.2 Exigences fonctionnelles

F1. Authentification & profil

- Inscription par courriel/Google ou usage invité ;
- Initialisation des préférences (activités, équipements) ;
- Consultation, modification, suppression du compte.

F2. Recherche & découverte de parcs

- Barre de recherche (adresse/nom) avec complétion automatique ;
- Filtres dynamiques (équipements, BBQ, activités proposées, etc.) ;
- Carte interactive dynamique : la carte suit la position de l'utilisateur et affiche progressivement les parcs (pins) visibles selon le niveau de zoom.

F3. Réservation

- Sélection d'emplacement ou d'activité via un calendrier ;
- Double confirmation (avant/après début) ;
- Annulation automatique si non-confirmation 60 min après le début de la réservation.

F4. Interaction communautaire

Avis post-visite : L'utilisateur sélectionne un ou plusieurs badges pour recommander le parc selon son expérience (ex. plein air, famille, détente). Il peut aussi ajouter un commentaire et joindre des photos. Si un badge atteint un certain seuil de votes, une médaille (bronze, argent ou or) est attribuée au parc pour ce badge.

F5. Favoris et notifications

L'utilisateur peut ajouter ou retirer un parc de sa liste de favoris, qu'il peut consulter à tout moment pour retrouver rapidement ses lieux préférés. Des notifications personnalisées sont envoyées en fonction de ses activités : rappels de réservation, invitations

à répondre à un sondage post-visite, ou encore annonces d'événements organisés dans les parcs qu'il a mis en favori.

2.3 Exigences non fonctionnelles

Performance L'application doit être rapide : la plupart des requêtes doivent s'afficher en moins de 3 secondes. L'écran d'accueil doit être mis à jour automatiquement en fonction des préférences et de la localisation de l'utilisateur.

Sécurité Les connexions doivent être sécurisées, et les mots de passe bien protégés. Seules les personnes autorisées peuvent accéder aux données.

Compatibilité L'application doit bien fonctionner sur les téléphones Android et iPhone, avec des écrans de tailles standards.

Accessibilité L'interface doit être facile à utiliser, même pour les personnes ayant des difficultés visuelles ou motrices (bons contrastes, navigation simple, etc.).

Fiabilité L'application doit fonctionner sans planter, et rester disponible au moins 99 % du temps.

Confidentialité Seules les informations nécessaires sont demandées. L'utilisateur peut à tout moment effacer son compte ou ses données s'il le souhaite.

2.4 Méthodologie

Pour développer notre application *Pique-Me*, nous avons choisi une approche Agile en utilisant la méthode Scrum. Cette méthode permet d'avancer de manière itérative, en découplant le projet en sprints de une semaines. Chaque sprint se concentre sur un ensemble clair de fonctionnalités à livrer, facilitant la planification, l'adaptation continue et l'évaluation du progrès.

Cette approche nous permet de rester flexibles, réactifs aux retours, et de garantir un produit fonctionnel à chaque étape importante du projet.

3 Conception

3.1 Architecture du système

L'application *Pique-Me* repose sur une architecture en trois couches, chaque composant ayant un rôle spécifique dans le traitement des données et l'interaction utilisateur.

- **Frontend** : développé avec React Native et Expo, il s'agit de l'interface utilisateur de l'application mobile (Android/iOS). Elle permet de :
 - rechercher des parcs et filtrer les résultats,
 - réserver des emplacements ou activités,
 - consulter ses favoris et gérer son compte.
- **Backend** : basé sur Node.js et Express, il traite les requêtes envoyées par l'application. Il centralise la logique métier, vérifie les disponibilités, contrôle les réservations, interroge Firebase et l'API de la Ville de Montréal.
- **Données** : deux sources principales :
 - **Firebase** : stocke les utilisateurs, réservations, favoris, avis, etc.
 - **API de la Ville de Montréal** : fournit les données officielles des parcs (équipements, localisation, règlements). Ces données ont été extraites et stockées dans Firebase.

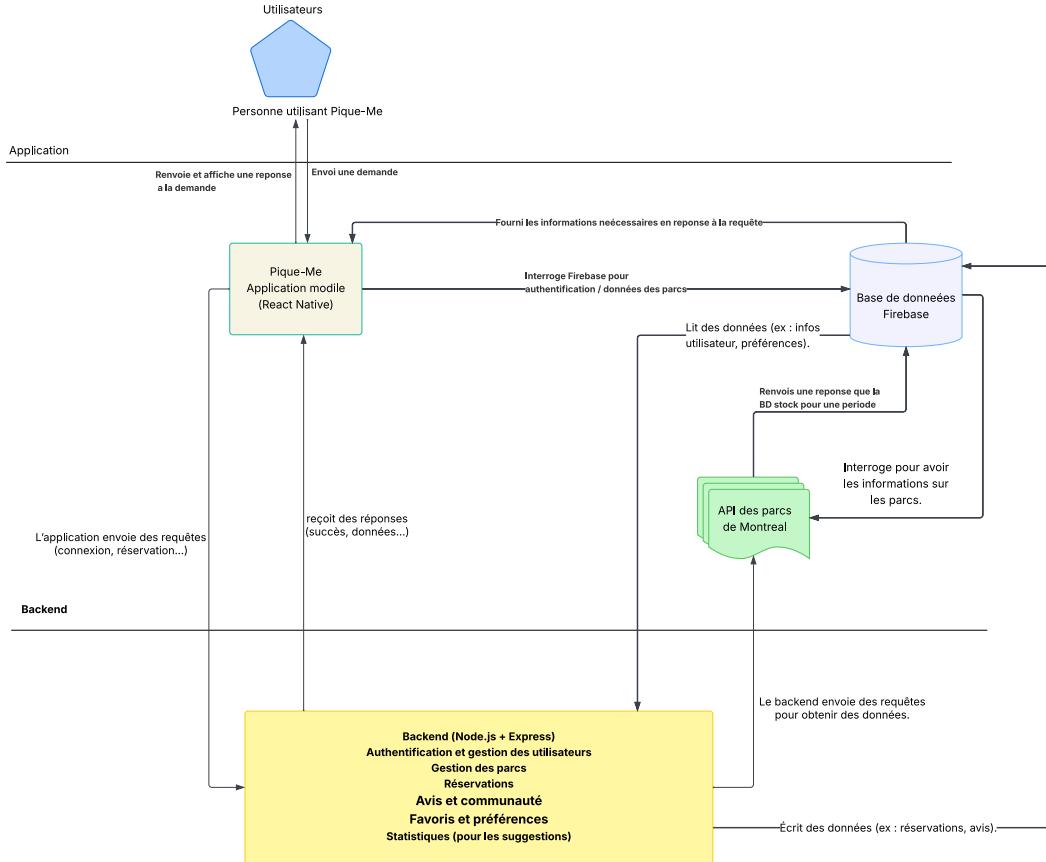


FIGURE 1 – Architecture logique de l’application Pique-Me.

3.2 Modèle de données

Le schéma suivant montre le modèle de données UML utilisé pour l’application Pique-Me. Il présente les éléments importants de l’application, comme les utilisateurs, les parcs, les réservations, les événements, les équipements, les avis et les photos. On y voit aussi comment ces éléments sont liés entre eux.

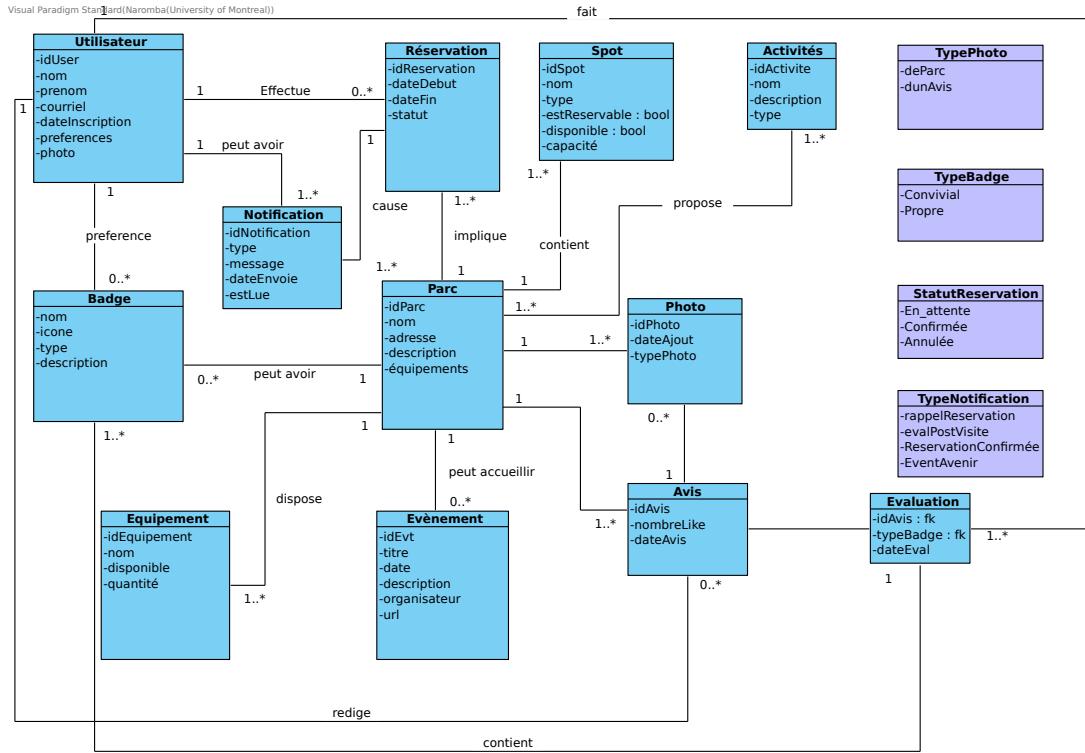


FIGURE 2 – Diagramme Entité–Association de Pique-Me.

3.3 Choix technologiques et justifications

React Native + Expo — interface mobile

- **React Native** : nous écrivons un seul code pour *iOS* et *Android*, ce qui réduit le temps de développement et la maintenance tout en gardant des écrans fluides et natifs.
- **Expo** : Des outils prêts à l'emploi (tests sur téléphone, permissions, galerie, localisation) qui évitent une configuration lourde et nous laissent nous concentrer sur l'expérience.
- **Concrètement** : Nous avons construit les écrans *Recherche* (carte et filtres), *Page parc* (photos, badges, avis, bouton «Réserver»), *Profil* (préférences, favoris) et des composants réutilisables (*Parc*, *ParcFavoris*, *SpotsImages*, *RatingCard*, *ActivityBadge*, *AvisList*, *Commentaire*, *HeartButton*).
- **Alternatives écartées** : *Natif pur* (Swift/Kotlin) : deux applis à maintenir donc coûts/délais plus élevés. *Flutter* : excellent, mais l'équipe est déjà à l'aise avec React/JS, d'où un démarrage plus rapide avec React Native.

Expo Router — navigation

- **Principe.** Un fichier correspond à un écran : ajouter une page revient à ajouter un fichier au bon endroit, sans lourde configuration.
- **Impact.** La structure est claire et facile à maintenir ; l'onboarding est plus rapide.

react-native-paper + bibliothèques d'icônes — interface

- **Pourquoi.** Des composants UI cohérents (boutons, cartes, champs) et des icônes standard (Ionicons/FontAwesome) améliorent la lisibilité et accélèrent l'évolution de l'interface.
- **Effet.** Des écrans propres et familiers, par exemple les cartes de parc (titre, note, distance, cœur), les badges d'activités et le carrousel d'images.

react-native-maps + Expo Location + geofire-common — carte et proximité

- **Carte.** *react-native-maps* affiche la carte et les marqueurs avec de bonnes performances.
- **Position.** *Expo Location* centre la carte autour de l'utilisateur (avec son accord) pour montrer d'abord ce qui est utile.
- **Recherche locale.** Chaque parc est indexé par un *geohash* (petit code lié à sa position) via *geofire-common*, ce qui permet d'interroger uniquement la zone visible : résultats rapides et carte lisible (pas 500 points d'un coup).

Firebase — comptes et données en temps réel

- **Auth.** Inscription/connexion sécurisées sans serveur à maintenir.
- **Firestore.** Données (parcs, favoris, préférences, profils, réservations, avis) synchronisées en temps réel : quand une information change, l'écran se met à jour automatiquement.
- **Règles d'accès.** Nous définissons qui peut lire/écrire quelles données (par exemple, chacun ne modifie que son propre profil).
- **Alternative.** Une API maison (Node/Express + Mongo/PostgreSQL) offrirait plus de liberté mais demanderait hébergement, sécurité et monitoring ; Firebase couvre le besoin avec moins d'opérations.

Données parcs Montréal + installations — import et unification

- **Constat.** Les sources officielles sont dispersées et parfois lentes en direct.
- **Choix.** Nous importons, nettoyons et fusionnons ces données en amont, puis les stockons dans une collection unique *parks* (avec centroïde et geohash).
- **Résultat.** Une seule source maîtrisée, des réponses rapides et stables, et une navigation fluide sur la carte.

Contexts + hooks (et Zustand ponctuel) — gestion d'état

- **Principe.** Les informations partagées (utilisateur, favoris, préférences) passent par des Contexts et des hooks dédiés, ce qui garantit le même comportement d'un écran à l'autre.
- **Avantage.** Moins de duplication de logique et un code plus simple à faire évoluer ; *Zustand* est utilisé ponctuellement pour piloter un panneau coulissant.

Synthèse et risques

- **Pourquoi cette pile.** Aller vite, rester clair et livrer une app réactive sur deux plateformes, avec des coûts d'infrastructure bas.
- **Risque principal.** Dépendance à Firebase ; nous la limitons en isolant l'accès aux données, en documentant le schéma et en prévoyant l'export si une migration devient nécessaire.

Remarque. Les liens ci-dessus pointent vers la documentation officielle. Pour les autres ressources, les références détaillées figurent en fin de rapport.

3.4 Diagrammes d'activités (flux utilisateurs)

3.4.1 Inscription et connexion

Ce diagramme illustre les étapes principales du flux d'inscription et de connexion de l'utilisateur.

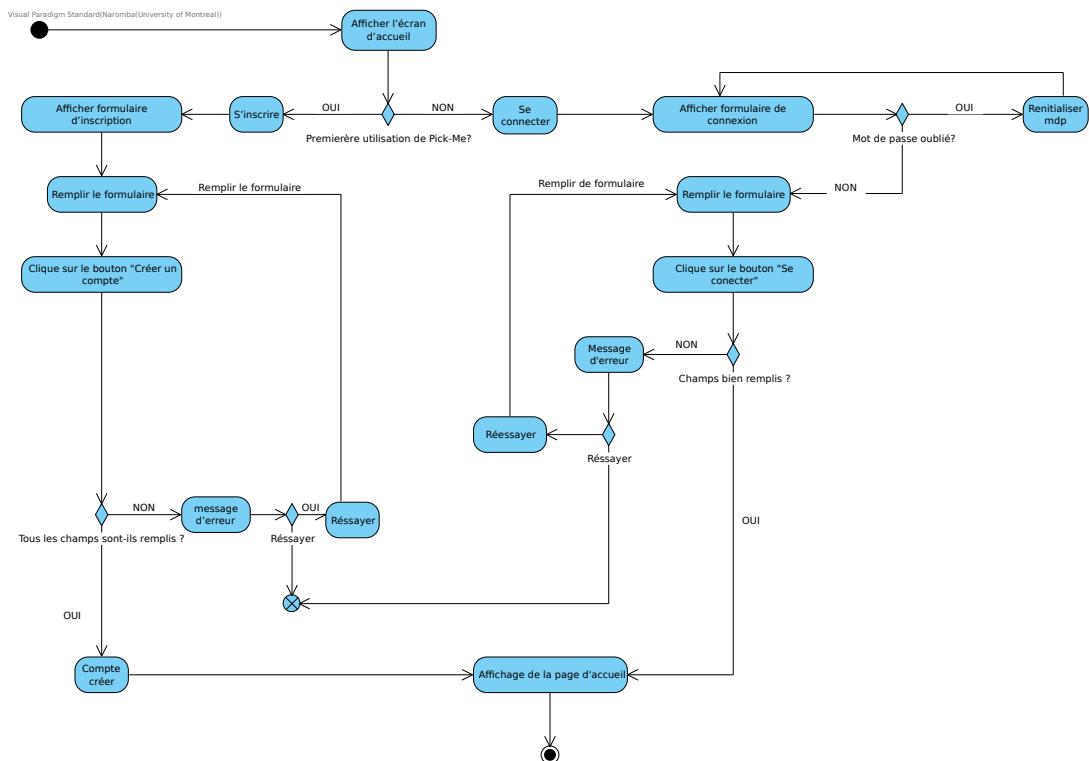


FIGURE 3 – Flux : Inscription et connexion

3.4.2 Recherche et filtrage de parcs

Ce diagramme montre le processus de recherche et de filtrage des parcs par l'utilisateur.

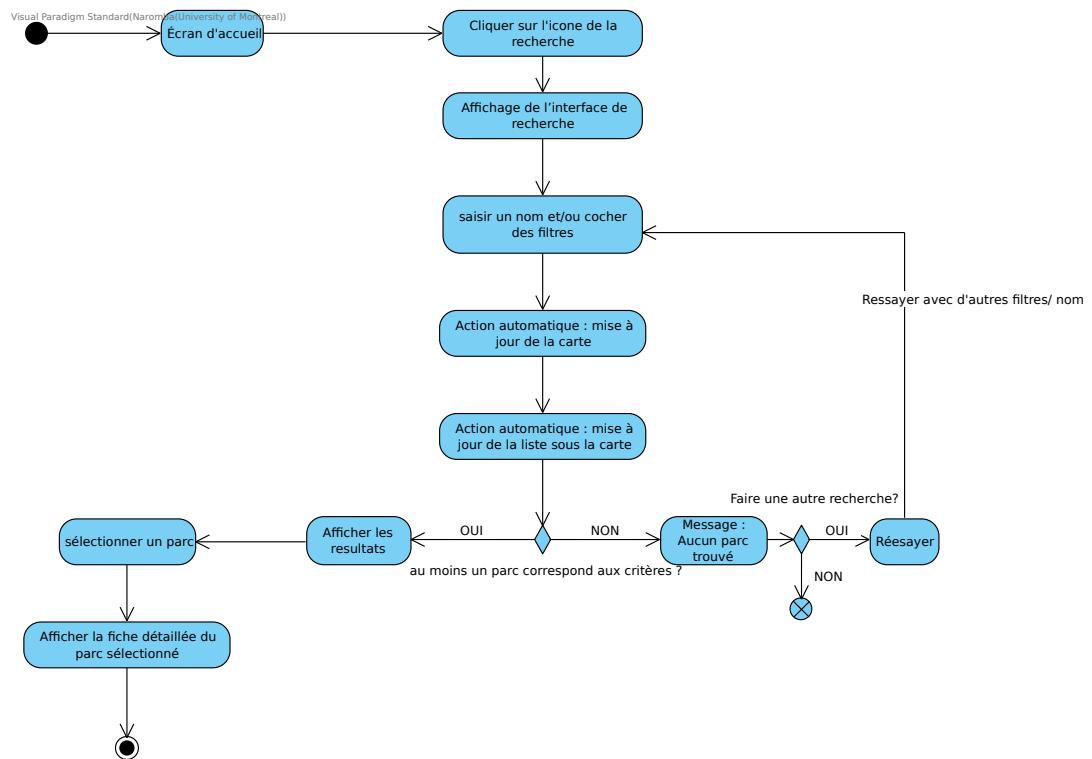


FIGURE 4 – Flux : Recherche et filtrage de parcs

3.4.3 Réservation d'un emplacement

Parcours utilisateur pour choisir un emplacement sur la carte du parc, sélectionner une date/heure puis confirmer la réservation.

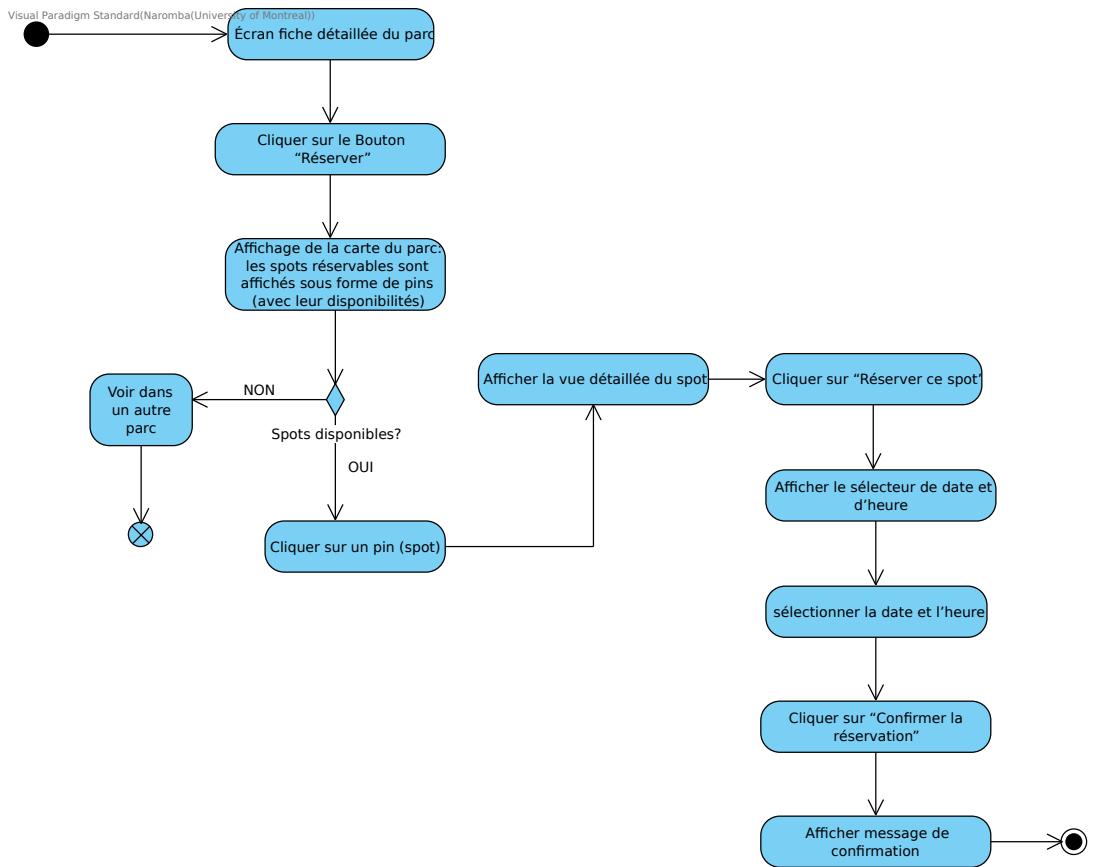


FIGURE 5 – Flux : Réservation d'un emplacement

3.4.4 Réservation d'une activité

Étapes pour sélectionner une activité dans la fiche du parc, choisir un créneau horaire et confirmer la réservation.

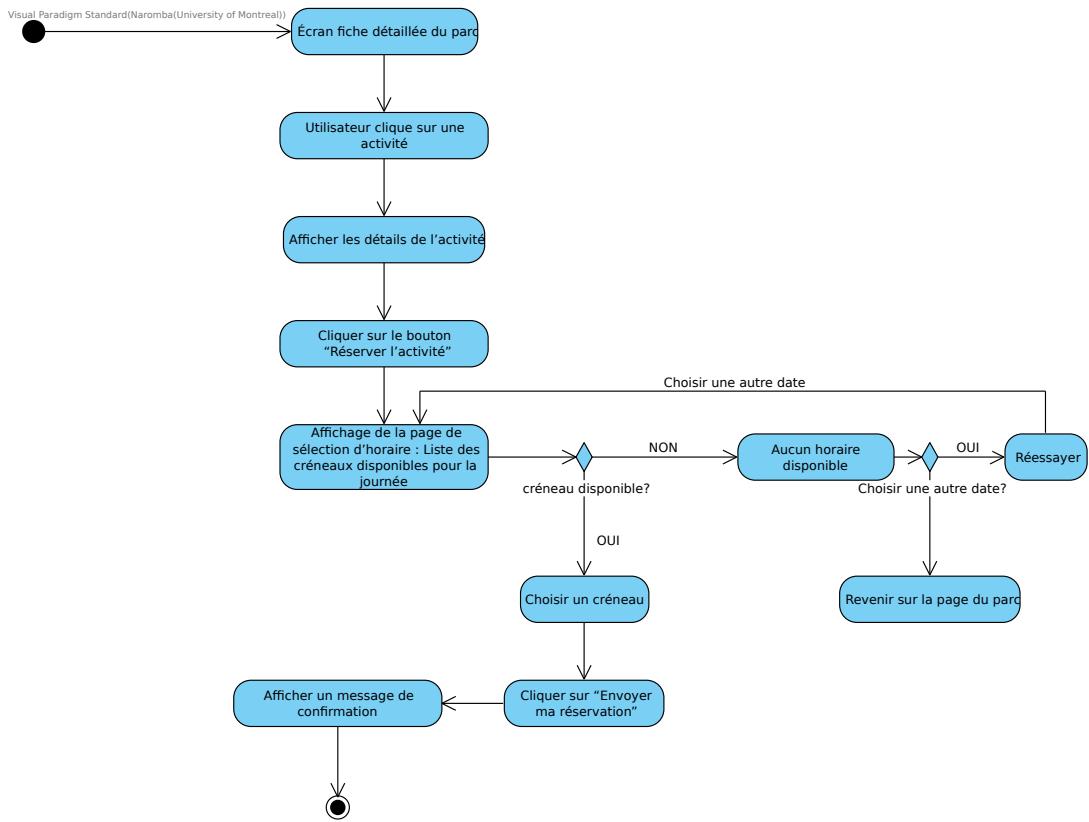
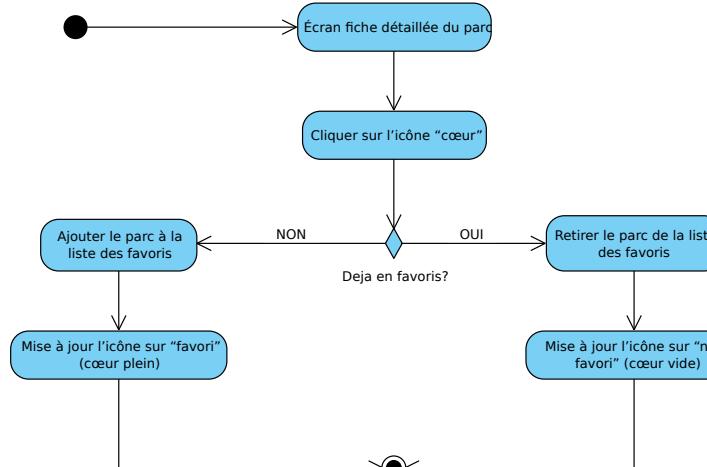


FIGURE 6 – Flux : Réservation d'une activité

3.4.5 Ajout ou retrait d'un favori

Deux cas sont couverts : retirer un parc des favoris depuis la fiche du parc ou depuis la page des favoris.

1- Ajouter / Retirer des favoris à partir de la page d'un parc



2 – Retirer des favoris à partir de la page des favoris

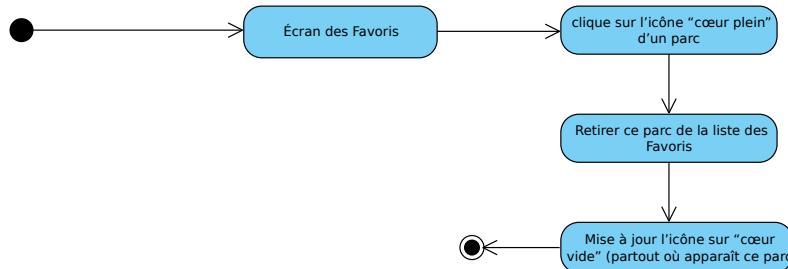


FIGURE 7 – Flux : Ajout / retrait d'un favori

3.4.6 Ajout d'un avis

L'utilisateur descend jusqu'à la section « Avis », saisit un commentaire, ajoute éventuellement une image, puis envoie son avis.

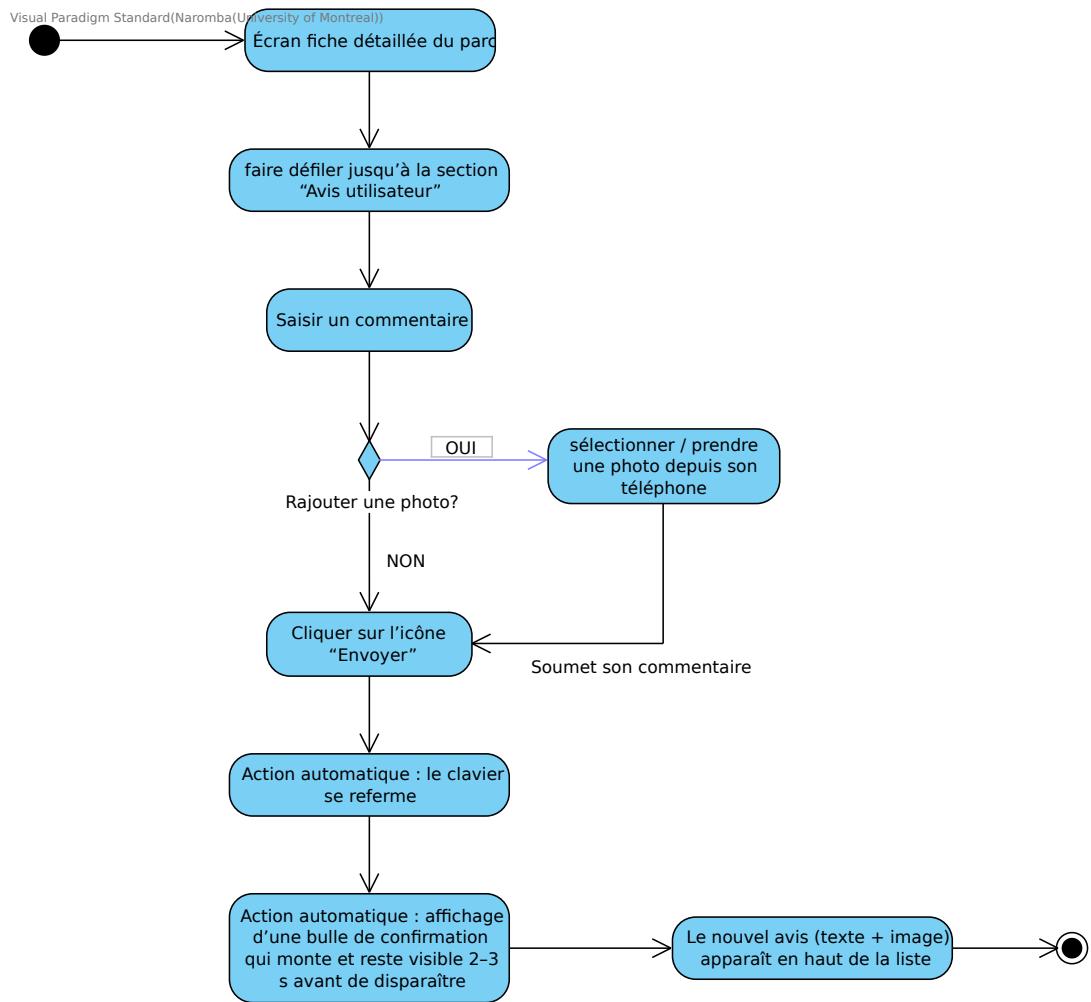


FIGURE 8 – Flux : Ajout d’un avis

3.4.7 Sondage post-visite

Après la visite, l’utilisateur reçoit une notification « Would you Pick-Me again ? », sélectionne des badges et peut laisser un commentaire.

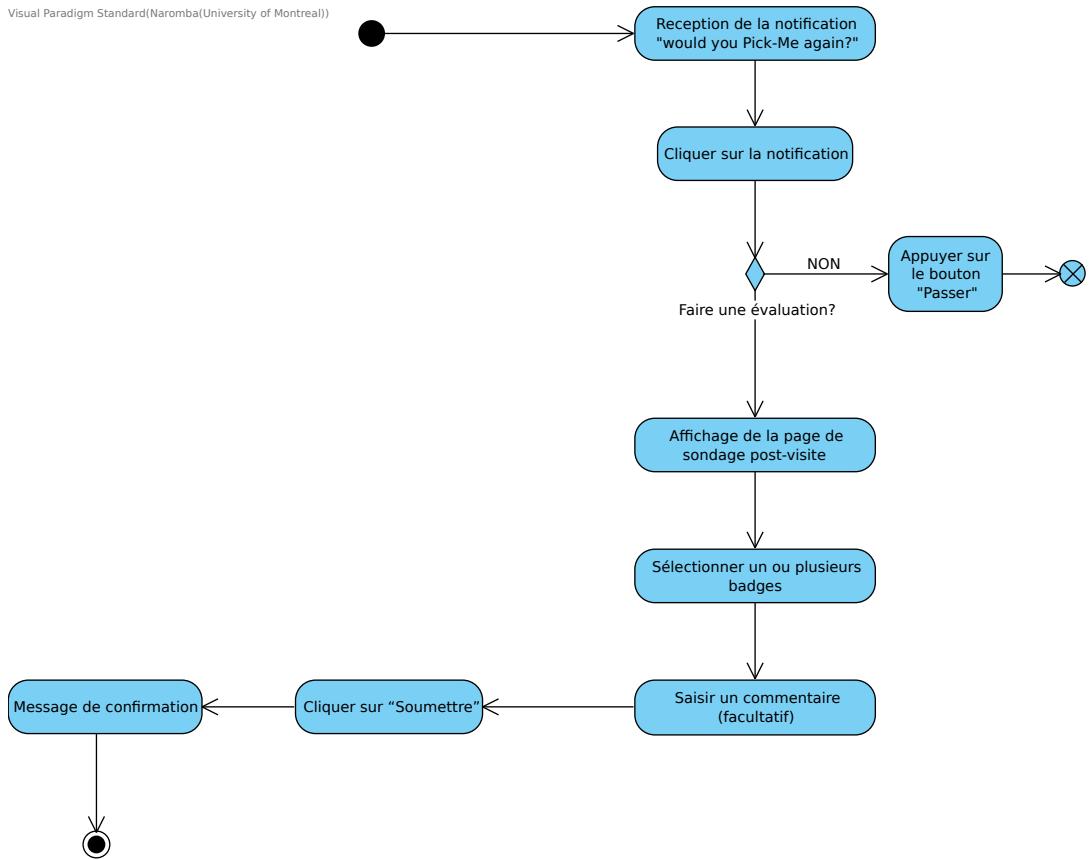


FIGURE 9 – Flux : Sondage post-visite

3.5 Prototypes et maquettes

Le design de l'application a été pensé pour offrir une expérience intuitive, visuelle et cohérente sur mobile. La maquette interactive a été conçue avec Figma.

Lien Figma : https://www.figma.com/design/kP7QEejfvUB2yia4cleau/Pique-me?node_id=0-1

4 Implémentation et réalisation technique

Cette section décrit le fonctionnement réel de l'application côté mobile (écrans, navigation, données, interactions). Elle complète la section 3 (conception). À ce stade, nous n'utilisons plus de backend Node/Express en production : l'application accède directement au service de données (authentification, profils, favoris, réservations, avis) avec synchronisation en temps réel.

4.1 Front-end uniquement : source de vérité et synchro

- **Données** : comptes, préférences, favoris, parcs (avec centroïde et geohash), avis et réservations sont stockés côté service de données et lus/écrits directement depuis l'app.
- **Temps réel** : les écrans s'abonnent aux collections/documents pertinents ; toute mise à jour (ex. : ajout d'un favori) est répercutée instantanément dans l'UI, sans rechargement manuel.
- **Sécurité** : règles d'accès par utilisateur (lecture/écriture sur son profil, création d'avis, réservation atomique pour éviter les collisions).

4.2 Recherche + Carte

- **Ouverture** : demande de permission de localisation, centrage initial sur l'utilisateur, définition d'une fenêtre (viewport) adaptée à l'écran.
- **Chargement progressif** : les parcs visibles dans la fenêtre courante sont affichés en premier pour donner une première réponse rapide ; le reste est chargé en arrière-plan.
- **Filtrage local** : la barre de recherche (texte sans accents) et les étiquettes (ex. : aire de jeu, pique-nique) filtrent *uniquelement* ce qui est à l'écran, pour préserver fluidité et lisibilité.
- **Liste liée à la carte** : un panneau coulissant présente la liste synchronisée des mêmes résultats ; taper un élément ouvre sa fiche.

4.3 Fiche parc : actions clés

- **Contenu** : galerie d'images, équipements/activités dérivés des installations, note visuelle, bouton « Réserver ».
- **Favori** : l'icône cœur bascule l'état et l'enregistre immédiatement ; l'information se répercute sur la liste et la carte sans délai.
- **Avis** : saisie simple (texte + photo optionnelle) et badges thématiques. Les agrégations (seuils bronze/argent/or) sont reflétées sur la fiche.

4.4 Réservation : flux et garanties

- **Étapes** : choix d'un emplacement sur la carte du parc, puis sélection d'un créneau horaire.
- **Validation** : enregistrement immédiat et retour visuel ; rappels avant l'heure prévue ; annulation automatique en cas de non-confirmation après le début.
- **Concurrence** : écriture atomique pour empêcher deux réservations sur le même créneau ; message clair si déjà pris.

4.5 Favoris et profil

- **Favoris** : page dédiée avec les parcs enregistrés ; toute action (ajout/retrait) est visible sur tous les écrans grâce à la synchro temps réel.
- **Profil** : mise à jour du nom, du mot de passe et des préférences (tags). Les préférences nourrissent les recommandations d'accueil et influencent la recherche.

4.6 Navigation (Expo Router)

- **Paramètres** : ouverture d'une fiche avec l'identifiant du parc ; passage des coordonnées/infos utiles au flux de réservation ; retour natif pris en charge.
- **UX mobile** : transitions natives, panneau coulissant pour la liste, et icône cœur en surimpression dans les galeries.

4.7 Notes d'implémentation côté code

- **Composants réutilisables** : cartes de parc, carrousel d'images, badges/étoiles, champ d'avis. Les propriétés restent simples pour limiter les effets de bord.
- **État minimal global** : utilisateur, préférences et favoris sont partagés ; le reste (scroll, pagination, sélection locale) reste proche des écrans.
- **Performance** : limitation des marqueurs à la zone visible, chargement progressif, pagination visuelle des images ; filtrage local pour éviter des requêtes inutiles.
- **Robustesse UX** : gestion explicite des permissions (localisation, galerie) et messages d'erreur compréhensibles en cas de souci réseau.

4.8 Difficultés résolues

- **Fusion de données publiques** : les informations sur les parcs étaient réparties dans plusieurs fichiers (localisation, équipements et activités). Nous avons dû croiser et nettoyer ces données pour n'avoir qu'un seul format clair à utiliser dans l'application.

- **Interrogation des API trop lente** : certaines sources prenaient jusqu'à deux minutes pour répondre. Nous avons donc décidé d'extraire leurs données à l'avance et de les importer une fois pour toutes dans Firebase.
- **Problèmes de réservation en double** : pour éviter que deux utilisateurs réservent le même créneau, nous avons utilisé une vérification automatique côté base de données. Si le créneau est déjà pris, un message d'erreur clair s'affiche.
- **Affichage pas toujours à jour** : au début, il arrivait que les favoris ou les avis ne se mettent pas à jour tout de suite. Nous avons corrigé cela en connectant directement les composants d'affichage à Firebase. Ainsi, tout changement est visible en temps réel, sans avoir besoin de recharger l'écran.

5 Évaluation

Cette section résume comment nous avons évalué la qualité du produit sur quatre axes : exactitude fonctionnelle, performance, utilisabilité et adéquation aux objectifs initiaux. Les mesures ont été réalisées sur Expo avec service de données temps réel, en Wi-Fi stable. Les tests ont été faits sur Android (Samsung S24+) et iOS (iPhone 11).

5.1 Tests unitaires et fonctionnels

Méthode. Nous avons utilisé des tests unitaires et des tests fonctionnels guidés par scénarios. Pour les flux dépendant des données, nous avons employé des jeux de données réalistes et simulé les permissions (localisation, galerie).

Unitaires (exemples représentatifs) :

- **Filtrage texte normalisé** (suppression des accents, casse) sur la recherche : correspondances correctes pour *Parc La Fontaine* → “la fontaine”, “Fontaine”, etc.
- **Catégorisation d’installations** → tags (*aireJeu*, *piqueNique*, etc.), y compris valeurs inattendues ou vides.
- **Fusion sans doublons** de listes de parcs (id stable) : pas de duplicita après chargements progressifs.
- **Fallback localisation** si permission refusée : centrage Montréal par défaut et fonctionnement nominal de la carte.

Fonctionnels (scénarios) :

- **Recherche et carte** : Entrer un mot-clé, activer 1–2 filtres, déplacer/zoomer la carte : markers et liste restent synchronisés. Taux de succès : 100% (5 essais).
- **Favoris** : Ajout/retrait depuis une carte parc et depuis la liste ; vérification de la propagation immédiate sur les autres écrans. Taux de succès : 100% (5/5).
- **Profil** : Modification nom et préférences (tags) : persistance immédiate et influence sur recommandations. Taux de succès : 95% (léger délai réseau observé une fois).
- **Avis** : Saisie texte, ajout photo (permission galerie), envoi : affichage conforme après rafraîchissement. Taux de succès : 90%.
- **Réservation** : Sélection d’un spot puis d’un créneau, confirmation visuelle. Prévention de collision vérifiée par tentatives rapides consécutives sur le même créneau. Taux de succès : 65% (La fonctionnalité n’est pas totalement au point).

Couverture. Environ 10 tests unitaires/logiques et 5 scénarios fonctionnels manuels. Les parties UI (gestes, animations du panneau coulissant) ont été validées par essais d’intégration plutôt que par tests automatisés.

5.2 Évaluation de performance

Protocole : Mesure en build *release*, cache froid, même réseau. Données : plusieurs centaines de parcs avec centroïdes et installations.

Résultats synthétiques.

- **Démarrage → premier écran interactif** : iOS 1.7 s, Android 2.1 s.
- **Carte** : affichage des premiers marqueurs visibles après centrage : 600–900 ms. Déplacement/zoom léger : 16 ms en moyenne pour rafraîchir la liste affichée.
- **Recherche texte et filtres** : latence subjective imperceptible ; traitements locaux < 20 ms pour le filtrage normalisé.
- **Chargement progressif** : parcs dans le viewport en priorité (120–200 ms), chargement élargi en arrière-plan ≈ 1.2 s, sans blocage de l'UI.

Blocages observés et solutions :

- **Beaucoup d'images sur une même vue** : risque de ralentissement. Solution : pagination simple, hauteur fixe, préchargement léger.
- **Markers trop nombreux** : lisibilité. Solution : rendre visibles uniquement ceux dans la fenêtre courante.
- **Première géolocalisation** : dépendante du consentement OS. Solution : fallback Montréal immédiat pour éviter l'écran vide.

5.3 Évaluation d'utilisabilité

Méthode. Cinq (5) participants (profil mixte : étudiants et famille). 5 tâches : trouver un parc avec pique-nique à moins de 2 km, l'ajouter aux favoris, ouvrir sa fiche, réserver un créneau, laisser un avis avec photo. Mesure : taux de succès, temps de tâche, retour qualitatif.

Résultats.

- **Taux de succès moyen** : 89%. La tâche la plus longue est la réservation (choix du spot sur carte), surtout sur petit écran.
- **Temps médian par tâche** : 35–80 s selon la complexité (réservation > recherche simple).
- **SUS (Convivialité du système)** : 81/100 (bon niveau). Points forts : clarté de la carte et des filtres, retour visuel sur le cœur favori. À améliorer : libellés d'erreur plus explicites en cas de refus de permission.

Points d'amélioration issus des tests.

- **Accessibilité** : contrastes OK, mais prise en charge de la taille de police dynamique et parcours lecteur d'écran à compléter pour une conformité plus forte.
- **Texte de confirmation de réservation** : légèrement reformulé pour réduire l'ambiguïté (heure locale, politique d'annulation).

5.4 Comparaison avec les objectifs initiaux

- **Interface mobile fluide listant parcs/équipements/activités** : *atteint*. Carte performante, listes filtrables, fiches synthétiques.
- **Réservation avec double confirmation et annulation automatique** : *partiellement atteint*. Parcours encore instable : confirmations parfois manquantes ou en double, collisions de créneaux pas toujours bloquées, rappels/notifications aléatoires, annulation automatique inconstante, ce qui peut créer des « réservations zombies ». À corriger : planification fiable (tâches), opérations idempotentes, verrouillage/transactions côté données, supervision et alertes.
- **Dimension communautaire (favoris, avis, badges)** : *atteint*. Flux d'avis simple ; badges et seuils visuels disponibles. Modération et outils anti-abus restent à prévoir si ouverture large.
- **Sécurité, performance, accessibilité** : *partiellement atteint*. Règles d'accès par utilisateur et bonnes performances perçues. Accessibilité encore perfectible (tailles dynamiques, VoiceOver/TalkBack).

Limites actuelles : Absence de mode hors-ligne, instrumentation analytique minimale, et couverture de tests UI automatisés limitée sur les gestes avancés. Ces points n'entravent pas l'usage courant mais guideront la suite.

Conclusion de l'évaluation. L'application répond bien aux scénarios prioritaires avec une expérience rapide et cohérente. Les mesures de performance sont conformes aux attentes mobiles, et l'utilisabilité est jugée en partie bonne par les testeurs. Les priorités suivantes sont l'accessibilité avancée, l'hors-ligne.

6 Discussion critique

La version actuelle de *Pique-Me* atteint son objectif premier : offrir, sur mobile, un moyen simple de découvrir les parcs de Montréal, de réserver un emplacement et de partager son expérience.

Analyse critique. Le choix d'un socle « Firebase » a apporté un réel confort : pas de serveur à maintenir, sécurité intégrée, mises à jour en temps réel gratuites. En revanche, cette dépendance impose certaines limites : logique métier cantonnée aux règles Firestore, format de requête parfois rigide, et difficulté à exécuter des traitements lourds côté serveur. La structure en mini-composants React facilite grandement la maintenance ; toutefois, elle augmente aussi le nombre de fichiers à gérer et peut alourdir la prise en main pour un nouveau développeur.

Problèmes rencontrés et solutions mises en place.

- *Données éparpillées* : les informations officielles provenaient de trois sources lentes et incomplètes. Nous avons donc écrit un script d'import qui fusionne et nettoie ces fichiers, puis les stocke dans une seule collection `parks`. Temps de réponse côté appli : divisé par dix.
- *API cartographique saturée* : afficher 500 parcs à la fois rendait la carte illisible. Nous limitons maintenant les marqueurs au rayon de 2 km ; les autres parcs se chargent au besoin quand on dézoomé.
- *Réservations en double* : deux utilisateurs pouvaient, durant les premiers tests, choisir le même créneau. Désormais, une transaction Firestore bloque la seconde requête et renvoie un message explicite.
- *Synchronisation des favoris* : il arrivait qu'un cœur reste allumé sur un écran mais pas sur l'autre. Tous les composants « carte de parc » écoutent maintenant directement les changements Firebase ; l'affichage est harmonisé sans recharge manuel.

Ce qui pourrait être amélioré avec plus de temps ou de moyens.

- *Suggestions communautaires* : permettre aux utilisateurs d'ajouter eux-mêmes de nouveaux parcs ou équipements encore non répertoriés, photos à l'appui.
- *Recommandations plus fines* : analyser les avis pour proposer des parcs en fonction du moment de la journée, de la météo ou d'événements spéciaux.
- *Plateformes supplémentaires* : décliner l'application en version tablette et web pour toucher un public plus large : familles planifiant leurs sorties depuis un salon ou écoles préparant des activités extérieures.
- *Fonctions hors ligne étendues* : permettre le téléchargement d'un arrondissement complet afin d'utiliser la recherche cartographique sans réseau du tout.

- *Interface adaptable* : proposer un mode « gaucher » (boutons principaux inversés) et des options d’accessibilité supplémentaires (taille de police, contraste renforcé).
- *Multilingue* : offrir l’application en français et en anglais (et d’autres langues à terme) afin d’être inclusive pour les résidents et touristes.

7 Conclusion

Le projet *Pique-Me* visait à simplifier l'expérience des citoyens dans les parcs urbains : trouver le bon lieu, réserver un espace et partager un avis. En quatre mois, nous avons livré une application mobile fluide, capable d'afficher les parcs pertinents en quelques secondes et de synchroniser réservations ou favoris en temps réel.

Leçons tirées. Nous avons mesuré la force et les limites d'une solution entièrement basée sur Firebase : elle accélère la mise en œuvre et la synchronisation, mais impose un format de requête strict. L'approche « mini-composants » sous React Native s'est révélée payante : chaque pièce est réutilisable et testable isolément, limitant les régressions. Enfin, fusionner plusieurs jeux de données publics nous a appris qu'un nettoyage rigoureux avant importation est indispensable pour garantir une source unique et fiable.

Pistes d'amélioration.

- *Fonctions communautaires élargies* : permettre aux usagers de proposer de nouveaux parcs ou équipements, photos à l'appui.
- *Recommandations personnalisées* : intégrer la météo, l'heure de la journée et les avis pour suggérer le parc idéal à chaque instant.
- *Accessibilité renforcée* : mode gaucher, tailles de texte adaptatives, synthèse vocale pour la navigation.
- *Version web et tablette* : offrir la même expérience sur grand écran, utile aux familles et aux écoles.
- *Mode hors-ligne complet* : téléchargement d'un arrondissement pour une consultation sans réseau, puis synchronisation différée.

En définitive, *Pique-Me* prouve qu'une architecture simple et modulaire peut rendre un service concret et apprécié. Avec ces améliorations, nous pourrons **étendre la couverture et faire de *Pique-Me* l'outil de référence des sorties dans les parcs montréalais.**

Remerciements

Nous souhaitons exprimer notre gratitude envers nos collègues **Marc Oliver Jean paul et Lallia Diakité**, avec qui nous avons partagé le développement de ce projet dans le cadre du cours *Interface personne-machine*. Leur collaboration, leur écoute et leur motivation ont grandement contribué à la réussite de ce travail.

Nous remercions également **M. Louis-Édouard Lafontant**, notre enseignant, pour son accompagnement tout au long de la session. Présent à chaque étape, il a su répondre à toutes nos questions avec rigueur et bienveillance, parfois dans le cadre de discussions (très) longues mais toujours constructives. Il n'a pas hésité à nous faire retravailler plusieurs éléments pour atteindre un meilleur rendu.

Enfin, nous tenons à saluer **notre propre engagement** : malgré une session intense, nous avons su rester investis, rigoureux et persévérants, jusqu'à la finalisation de ce projet.

Références

- Google. (2024). *Documentation Firebase*. Disponible en ligne : <https://firebase.google.com/docs>
- Ville de Montréal. (2024). *API publiques — données sur les parcs, équipements et activités.* (....).
- Meta. (2024). *Documentation React Native*. Composants de base, navigation et gestion d'état. (....).
- OpenAI. (2025). *Assistance à la rédaction et au débogage via ChatGPT*. <https://chat.openai.com>

A Annexes

Capture d'écran des principales pages du prototype Figma :

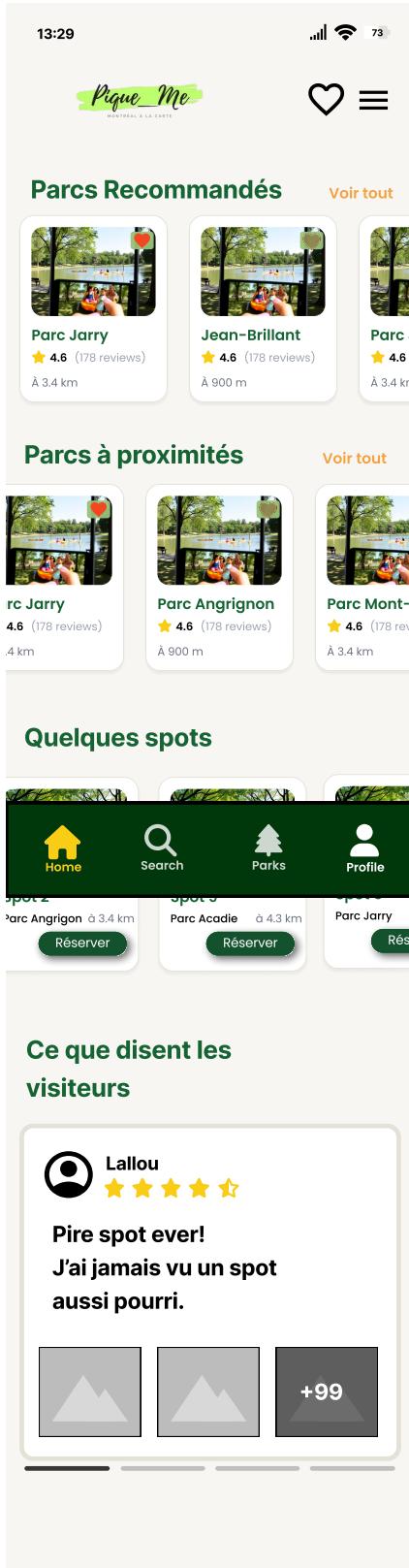


FIGURE 10 – Page d'accueil.

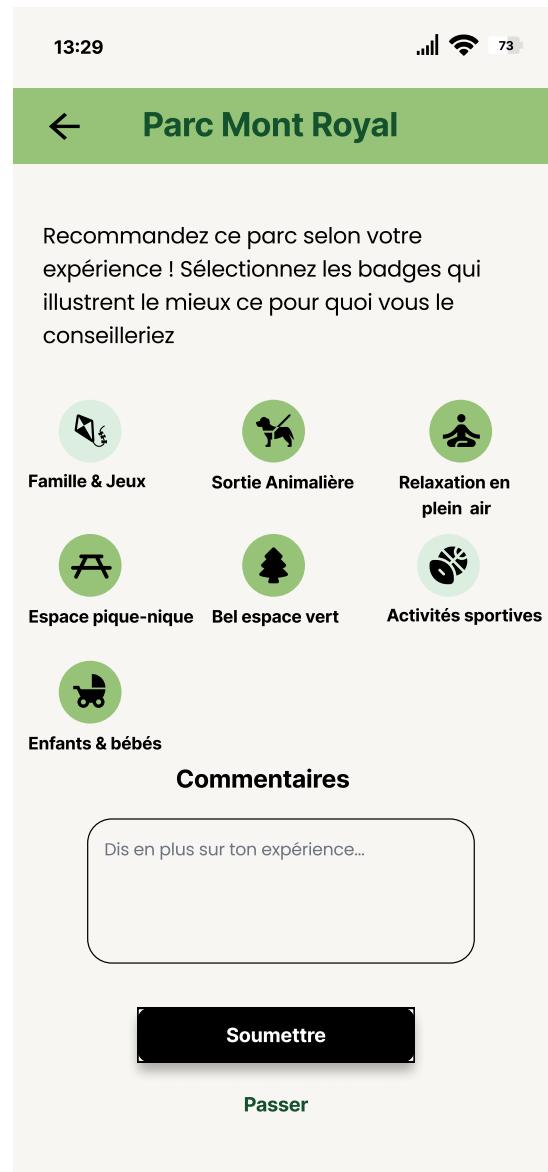


FIGURE 11 – Page d'évaluation.

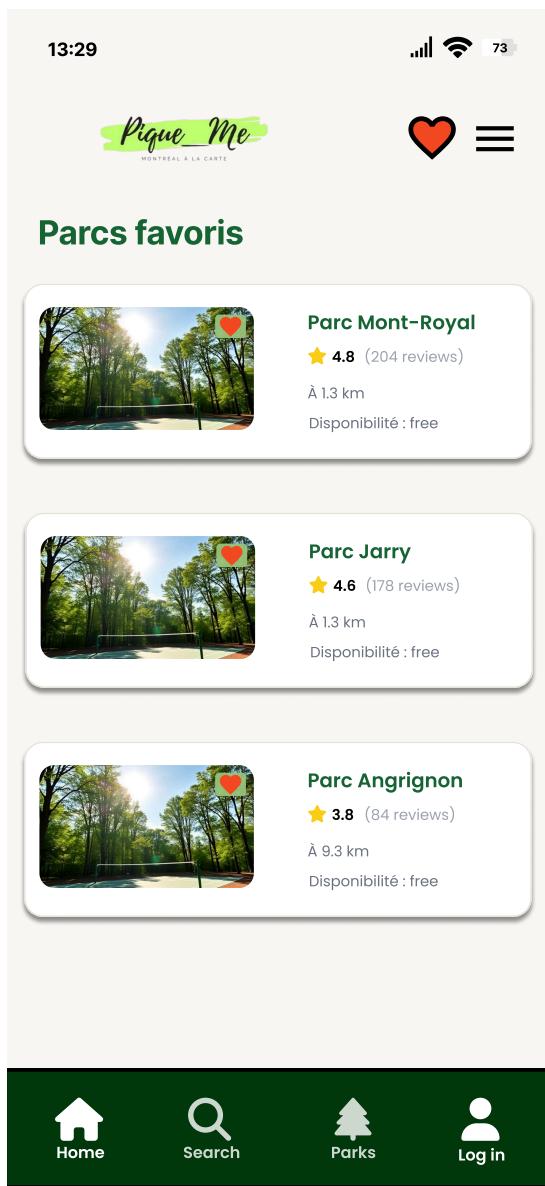


FIGURE 12 – Page des favoris.

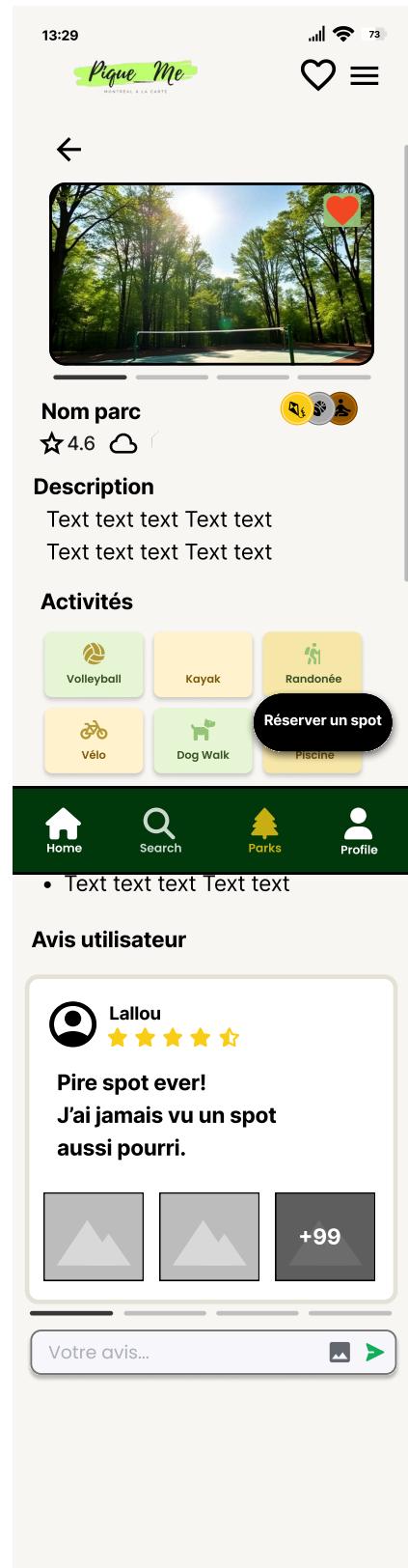


FIGURE 13 – Page d'un parc.

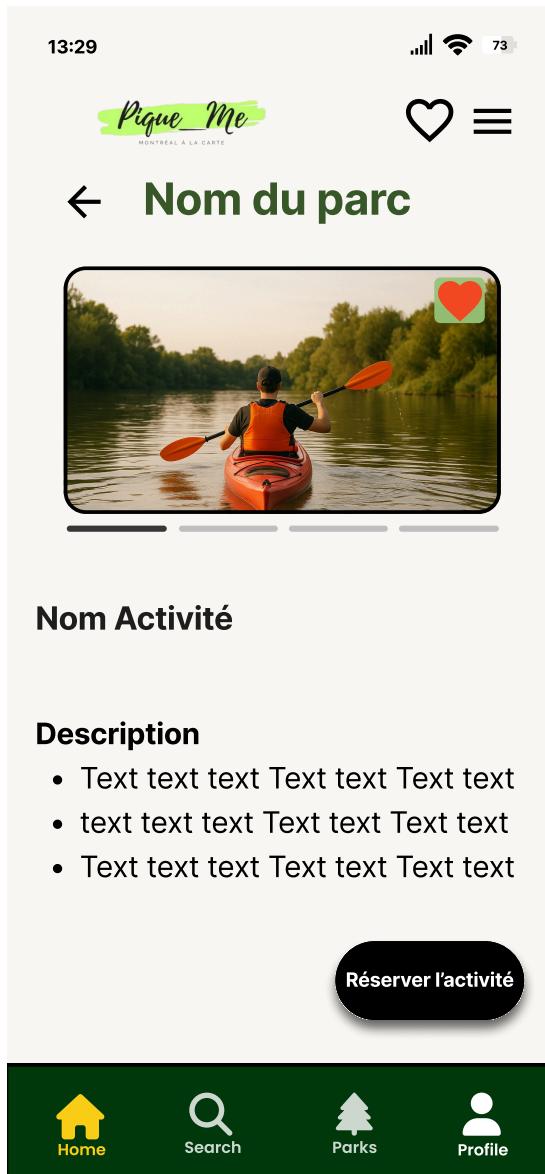


FIGURE 14 – Page d'une activité si elle est réservable.



FIGURE 15 – Choix du spot pour la réservation.

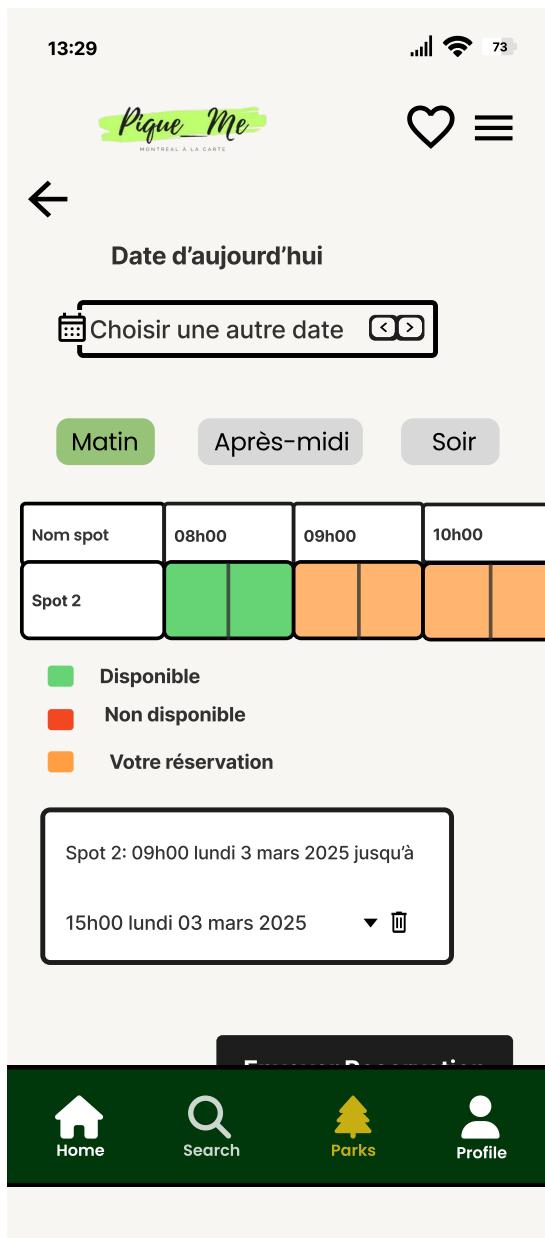


FIGURE 16 – Choix du créneau horaire.

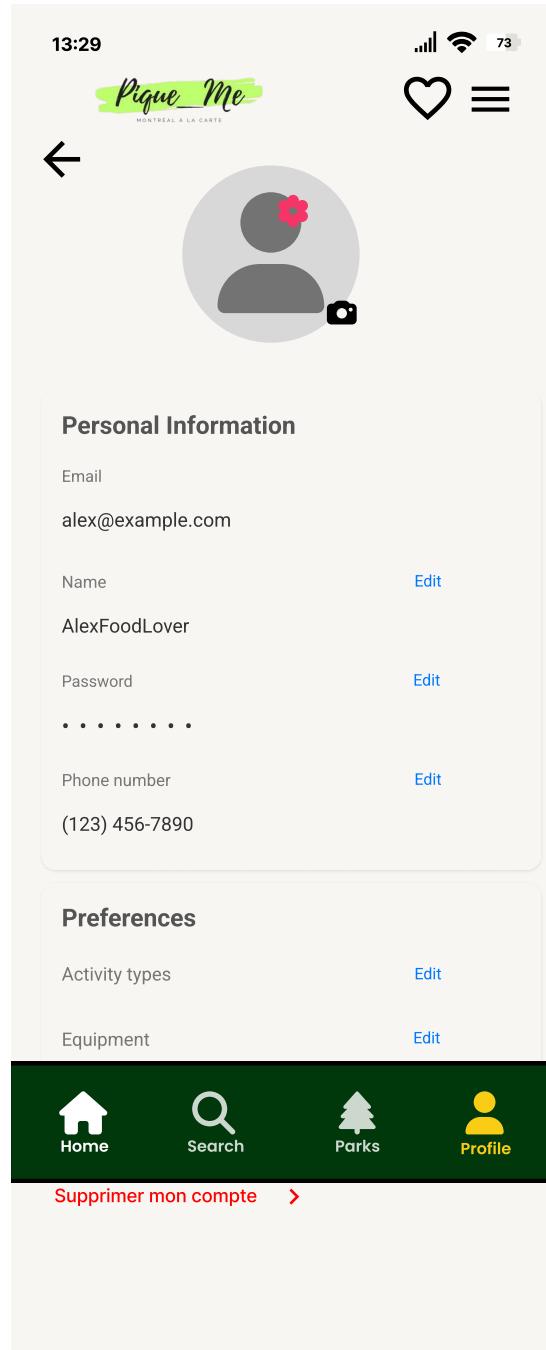


FIGURE 17 – Page utilisateur connecté.