RAPPORT DE PROJET- TRAVELSHARE

Jivan Rochemont Master ICO

Application Mobile Android de Partage de Photos de Voyage

1. INTRODUCTION

1.1 Présentation du Projet

TravelShare est une application mobile Android native développée en Java qui permet aux voyageurs de partager leurs photos avec géolocalisation, de découvrir de nouveaux lieux à travers les publications d'autres utilisateurs, et de créer des groupes de voyage collaboratifs. L'application s'intègre également avec TravelPaths, une application complémentaire de génération d'itinéraires de visite.

1.2 Objectifs

L'objectif principal était de créer une plateforme sociale centrée sur le voyage permettant :

- Le partage de photos géolocalisées avec contexte détaillé (type de lieu, description, instructions)
- La découverte collaborative de destinations via un flux social et une carte interactive
- La création de groupes privés pour partager des souvenirs de voyage entre amis
- L'intégration avec une application tierce pour transformer les découvertes en parcours de visite concrets

1.3 Technologies Utilisées

Backend & Services Cloud :

- Firebase Authentication pour la gestion des comptes utilisateurs
- Cloud Firestore (base de données NoSQL temps réel) pour stocker les données
- Firebase Storage pour l'hébergement des images
- Firebase Cloud Functions (Node.js 20, 2nd Gen) pour la logique serveur automatisée
- Firebase Cloud Messaging (FCM) pour les notifications push

Frontend Android :

- Langage : Java
- SDK minimum : Android 7.0 (API 24)
- SDK cible : Android 14 (API 36)
- Architecture : MVVM partiel avec couche de services métier
- Build System : Gradle avec Kotlin DSL

APIs et Bibliothèques :

- Google Maps SDK for Android pour la cartographie
- Places API, Geocoding API, Directions API pour les fonctionnalités de localisation

- Navigation Component pour la navigation entre fragments
- Glide pour le chargement optimisé des images
- Material Design Components pour l'interface utilisateur

---

2. ARCHITECTURE DE L'APPLICATION

2.1 Architecture Générale

L'application suit une architecture en couches :

Couche Présentation (Fragments & Activities) : Gère l'interface utilisateur et les interactions. Chaque fonctionnalité majeure est implémentée dans un Fragment distinct (HomeFragment, SearchFragment, MapFragment, NotificationsFragment, ProfileFragment) qui communique avec la couche métier via des services.

Couche Métier (Services) : Contient toute la logique applicative encapsulée dans des classes de service spécialisées :

- `AuthService` : Gestion de l'authentification et du profil utilisateur
- `PhotoService` : CRUD des photos, likes, signalements
- `CommentService` : Gestion des commentaires
- `GroupService` : Gestion des groupes de voyage
- `NotificationService` : Gestion des notifications locales

Cette approche permet de réutiliser la logique métier dans plusieurs fragments .

Couche Données (Firebase) : Communication directe avec les services Firebase via les SDK officiels. Les données sont structurées en 5 collections Firestore principales : users, photos, groups, comments, notifications.

Backend Serverless (Cloud Functions) : Quatre fonctions Cloud automatisent les processus critiques :

- Envoi automatique de notifications push lors de likes/commentaires
- Nettoyage périodique des anciennes notifications (>30 jours)
- Gestion centralisée de la logique sensible côté serveur

2.2 Navigation et Expérience Utilisateur

L'application utilise le Navigation Component d'Android avec une Bottom Navigation Bar comportant 5 destinations principales. Un Floating Action Button permet d'accéder rapidement à la publication de photos depuis n'importe quel écran. Cette approche offre une navigation intuitive conforme aux standards Material Design.

---

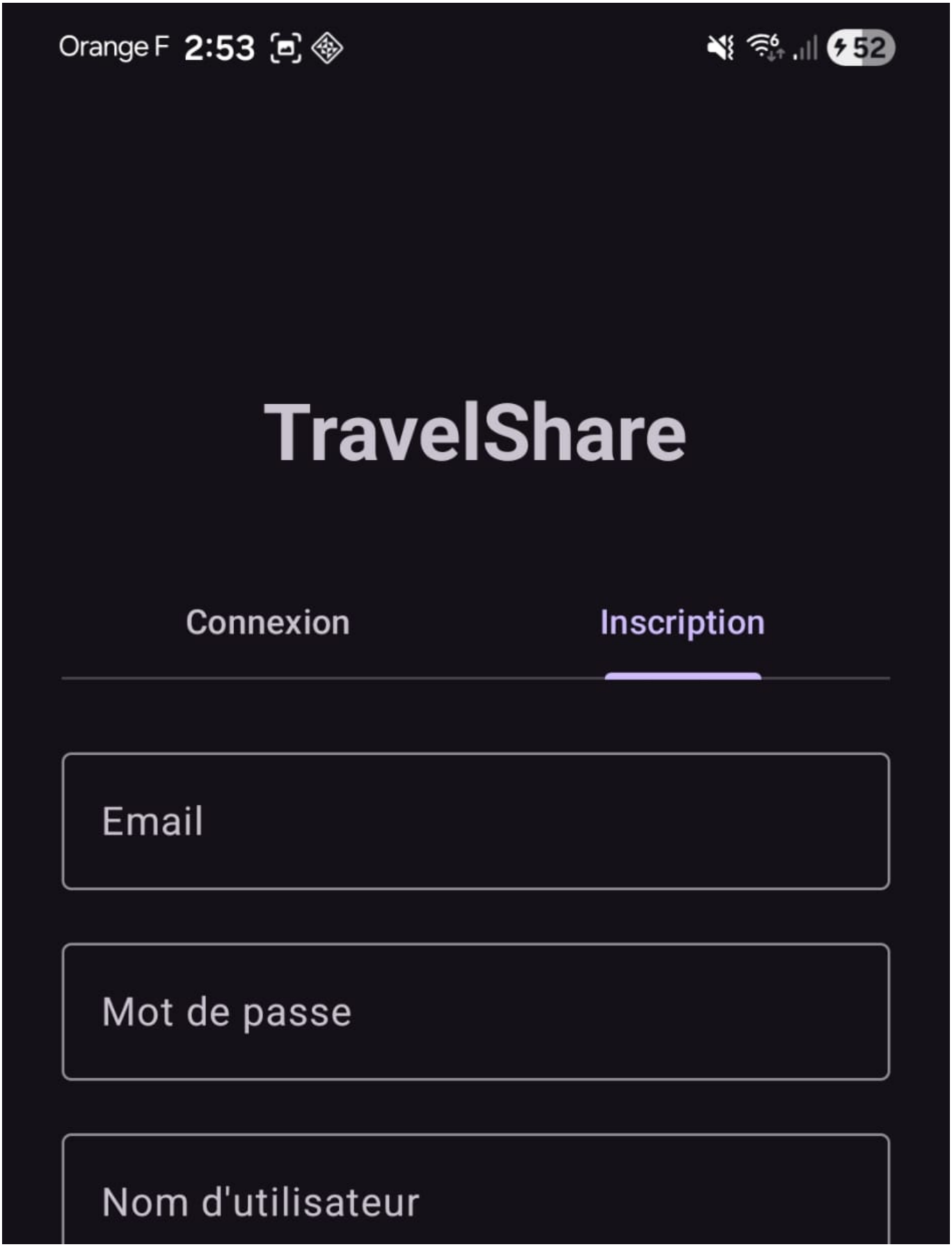
3. FONCTIONNALITÉS PRINCIPALES

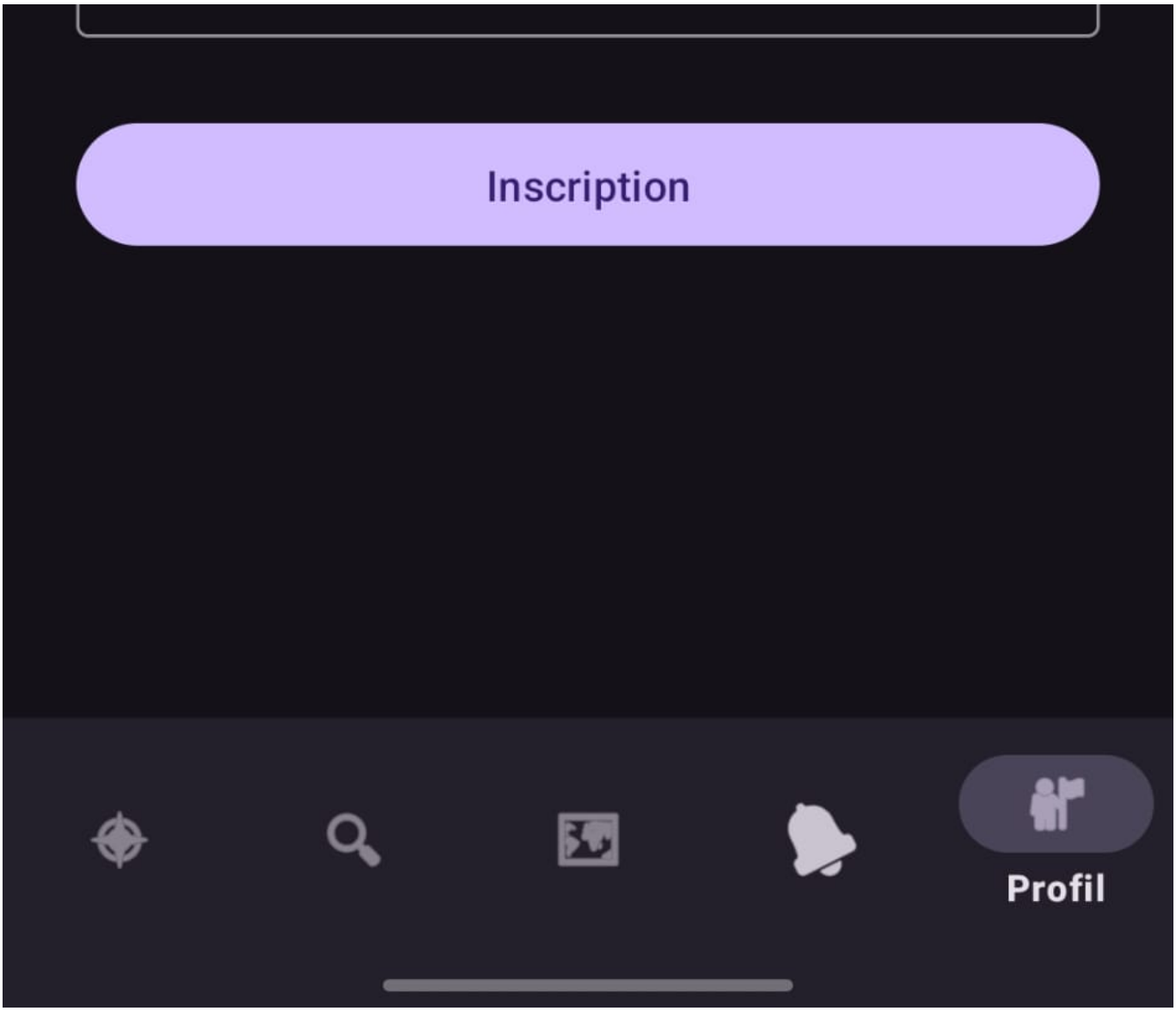
3.1 Système d'Authentification et Gestion de Profil

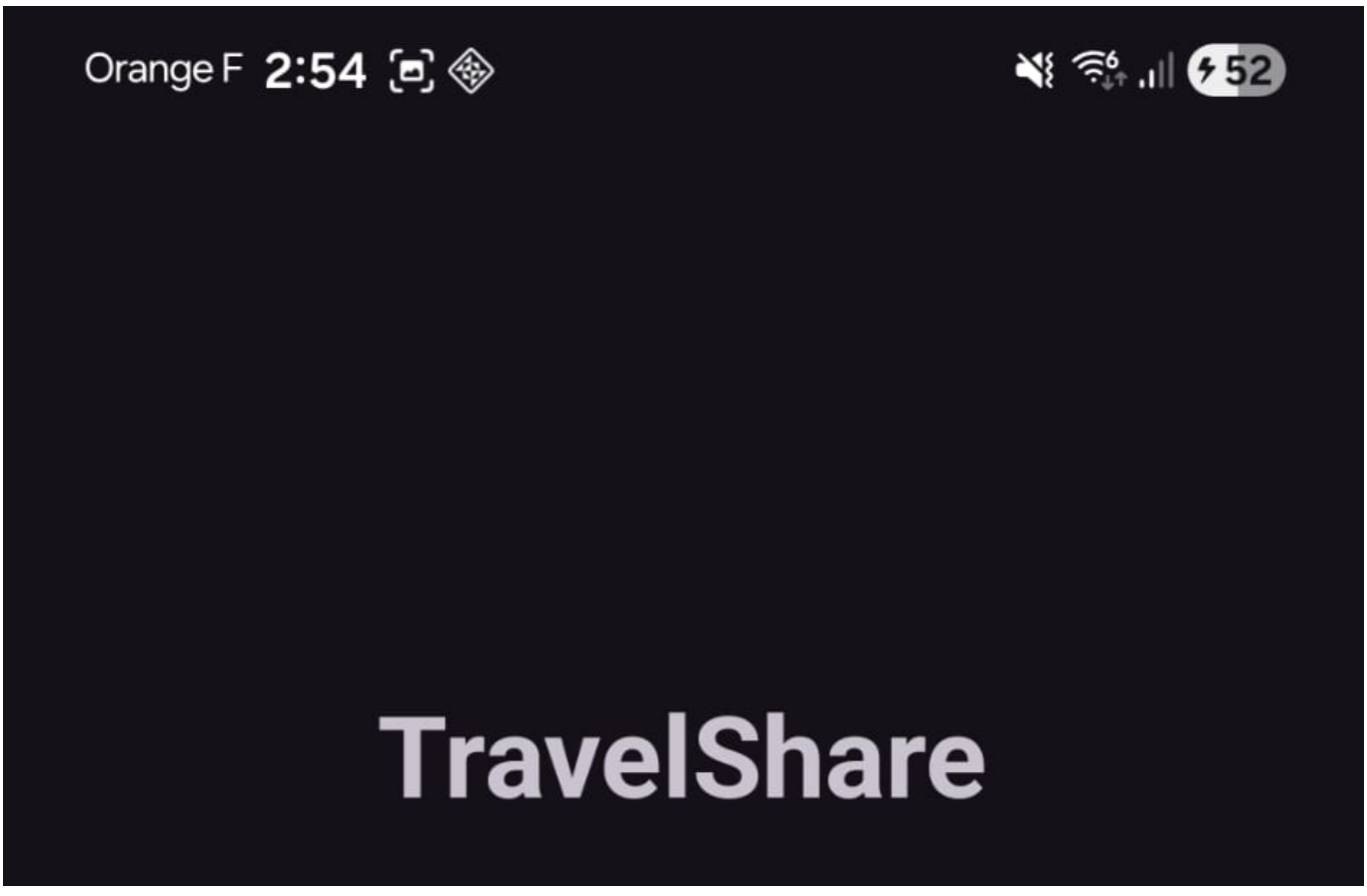
Implémentation : Le système d'authentification repose sur Firebase Authentication qui gère de manière sécurisée les comptes utilisateurs. J'ai implémenté trois modes d'accès pour offrir une flexibilité maximale aux

utilisateurs.

Inscription et Connexion : Lors de l'inscription, l'utilisateur fournit un email et un mot de passe. Firebase Authentication valide automatiquement le format de l'email et applique des règles de sécurité pour les mots de passe. Une fois le compte créé, un document utilisateur est automatiquement créé dans Firestore contenant les informations de base (email, username, dates de création).




Inscription
Profil


Orange F 2:54
52
TravelShare

Connexion

Inscription

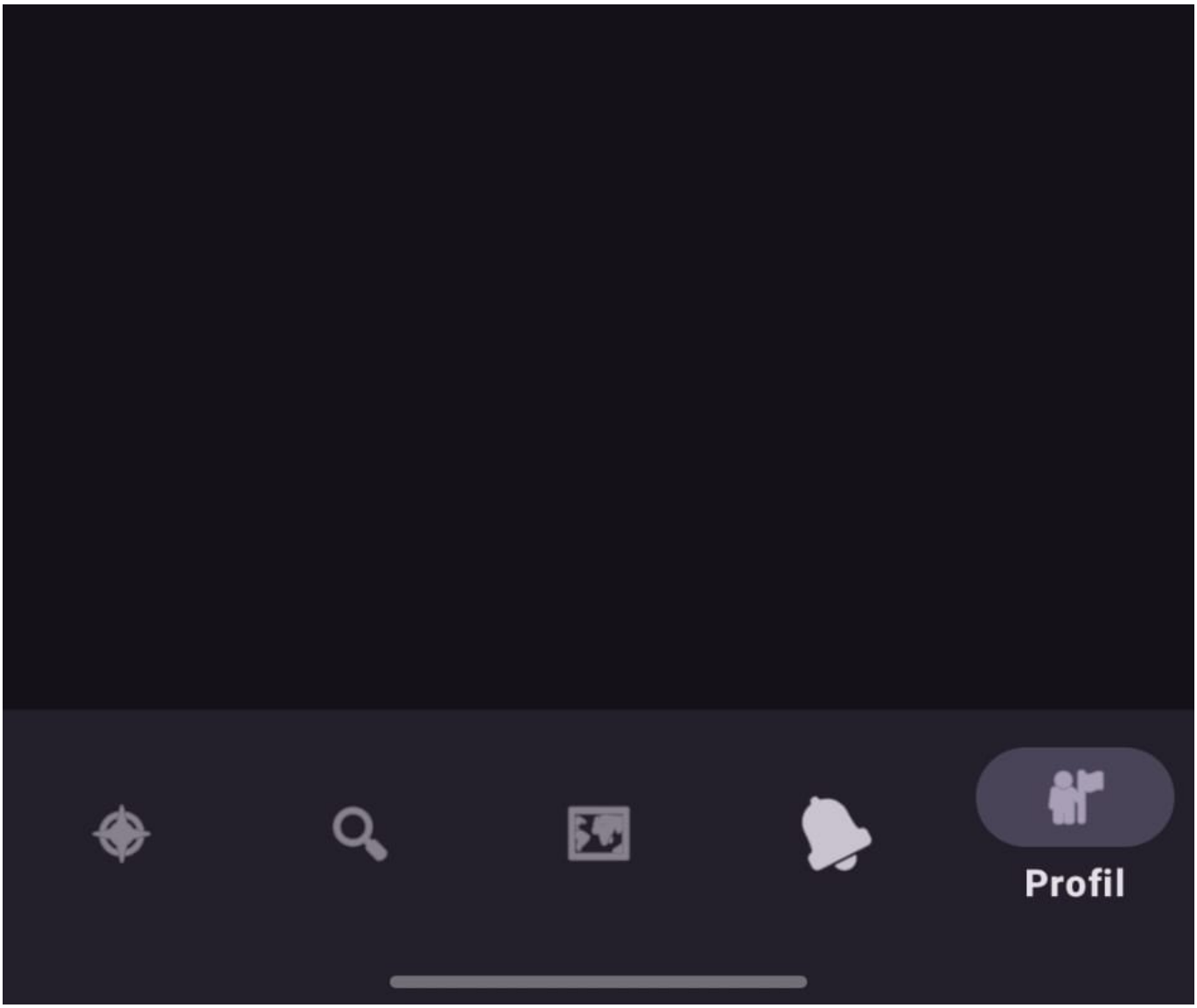
Email

Mot de passe

Connexion

Profil

Mode Anonyme : J'ai également implémenté un mode anonyme permettant aux utilisateurs de découvrir l'application sans créer de compte. Firebase Authentication crée un compte temporaire avec un UID unique, ce qui permet de suivre les actions de l'utilisateur tout en préservant sa vie privée. L'utilisateur peut ensuite convertir son compte anonyme en compte permanent s'il le souhaite.



Gestion du Profil : Chaque utilisateur peut personnaliser son profil avec un nom d'utilisateur unique, une biographie descriptive et une photo de profil. Les modifications sont sauvegardées en temps réel dans Firestore. Pour la photo de profil, l'image est d'abord compressée côté client pour réduire la bande passante, puis uploadée sur Firebase Storage, et l'URL résultante est stockée dans le document utilisateur.

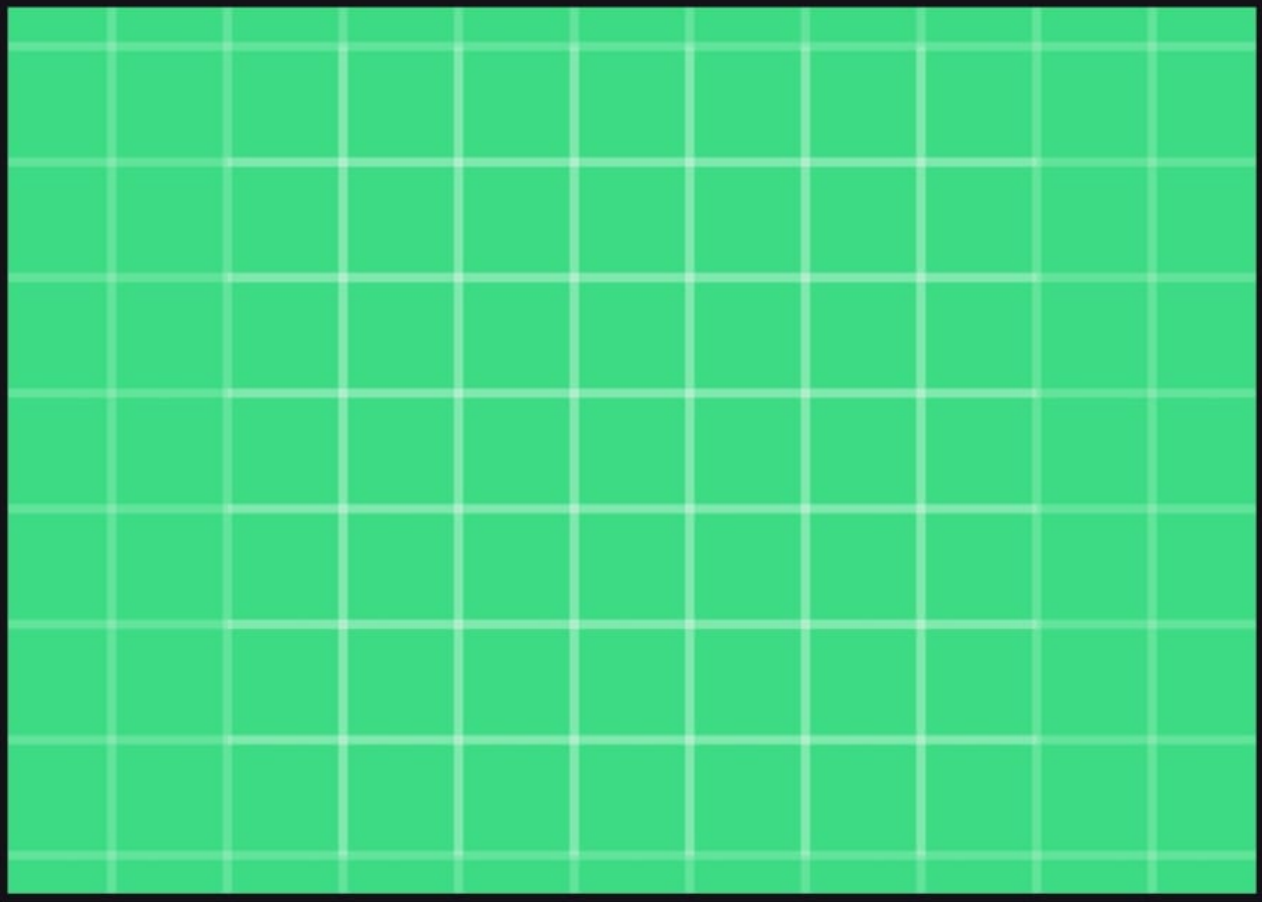
Sécurité : Les tokens FCM (Firebase Cloud Messaging) sont automatiquement mis à jour et stockés dans le profil utilisateur lors de chaque connexion, ce qui permet l'envoi de notifications push même si l'utilisateur change d'appareil.

3.2 Publication et Géolocalisation de Photos

Implémentation : La publication de photos est le cœur de l'application. J'ai développé un processus en plusieurs étapes pour enrichir chaque photo d'informations contextuelles riches.

Sélection de la Photo : L'utilisateur peut sélectionner une photo depuis sa galerie via un Intent Android standard. Une fois sélectionnée, l'image est chargée et affichée en aperçu. Pour optimiser les performances et réduire les coûts de stockage, j'ai implémenté un système de compression automatique qui réduit la taille de l'image .

Publier une photo

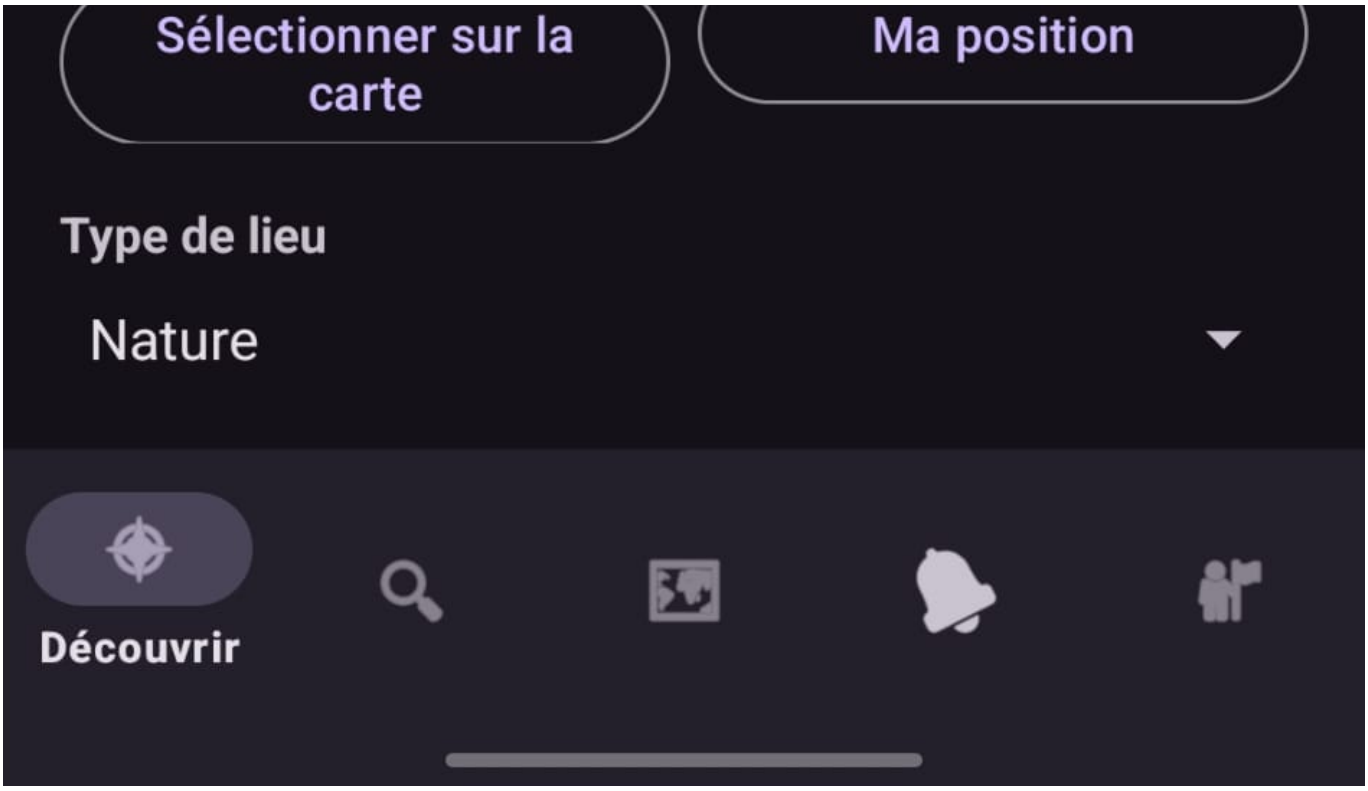


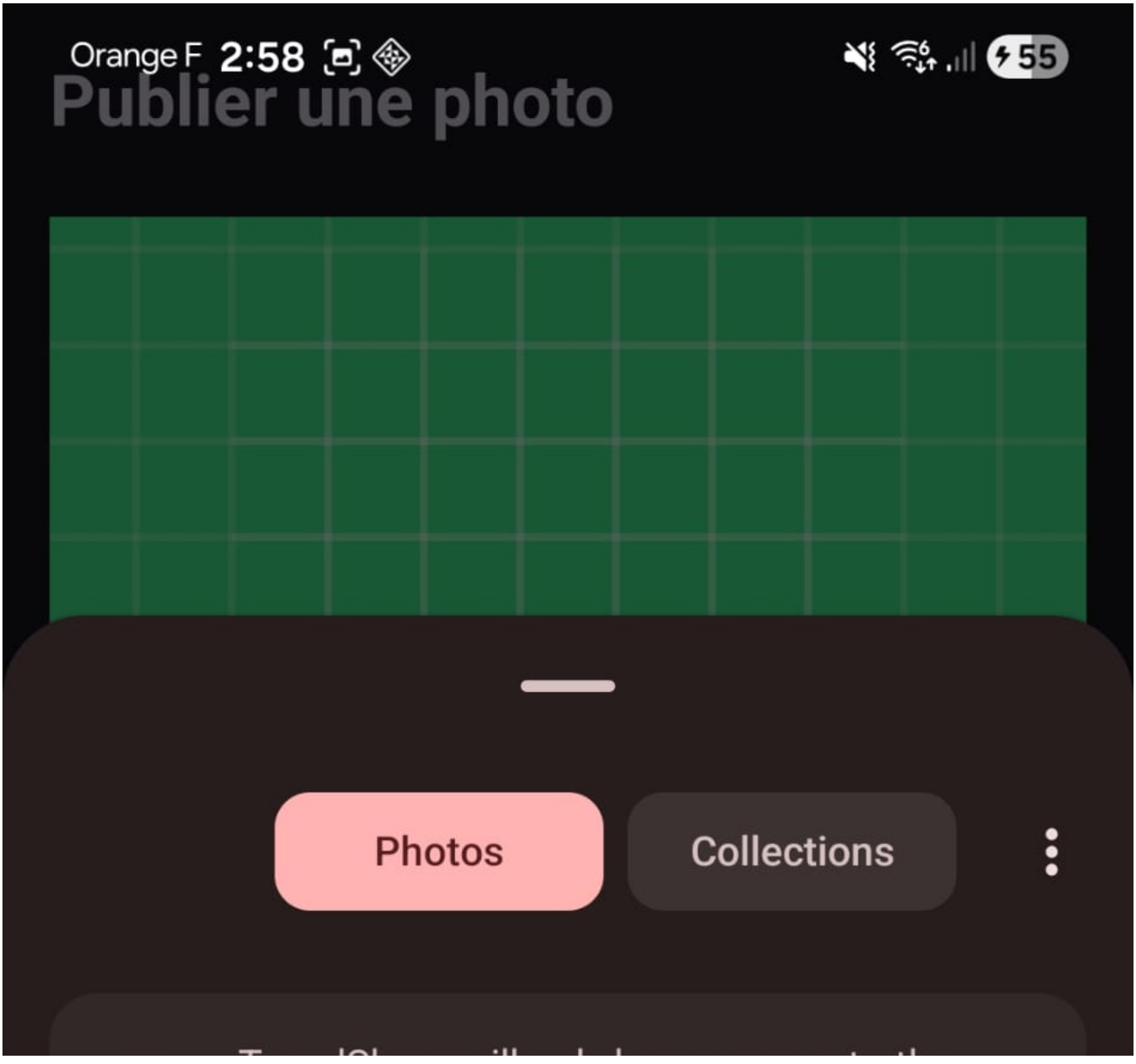
Sélectionner une photo

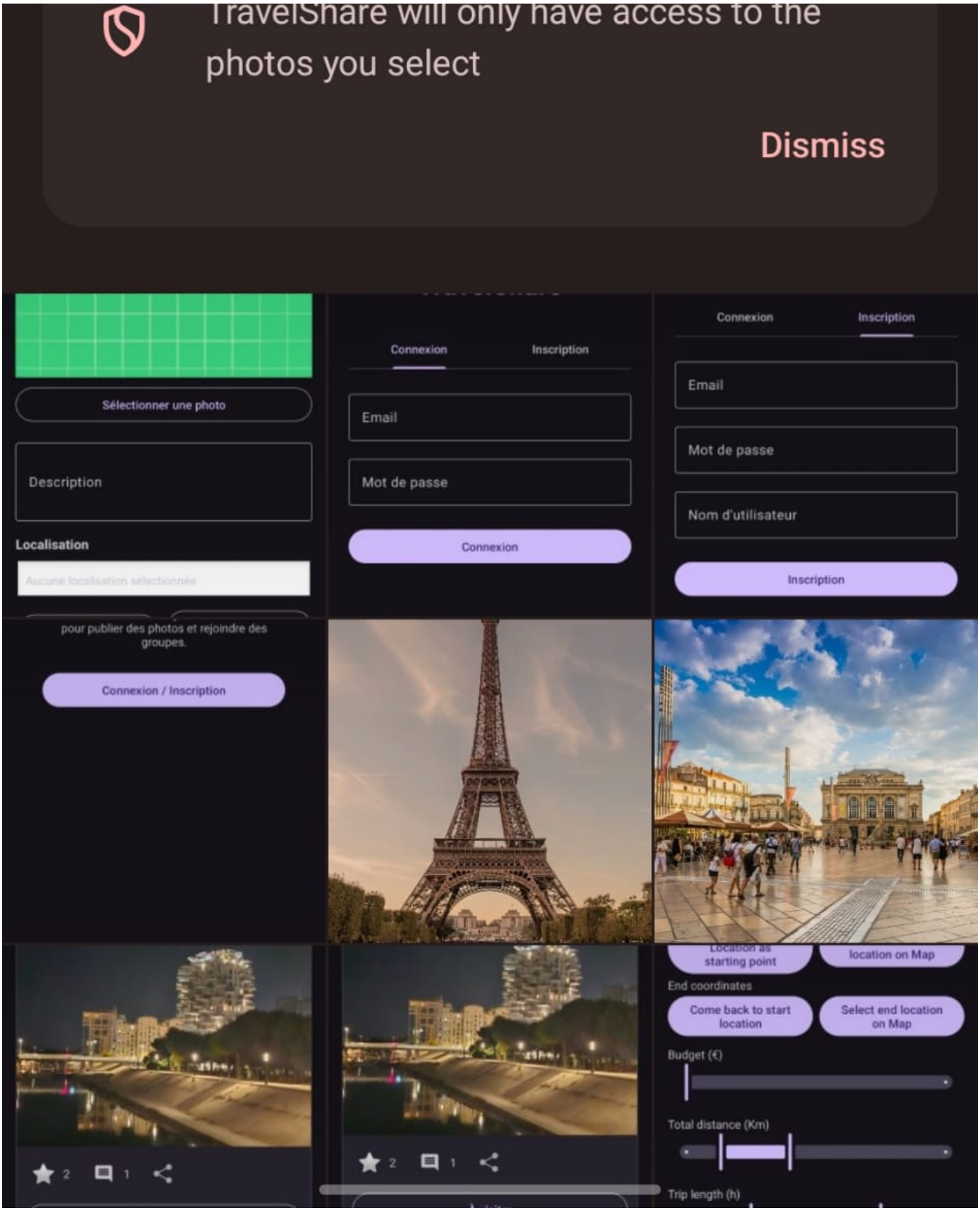
Description

Localisation

Aucune localisation sélectionnée


Sélectionner sur la carte
Ma position
Type de lieu
Nature
Découvrir

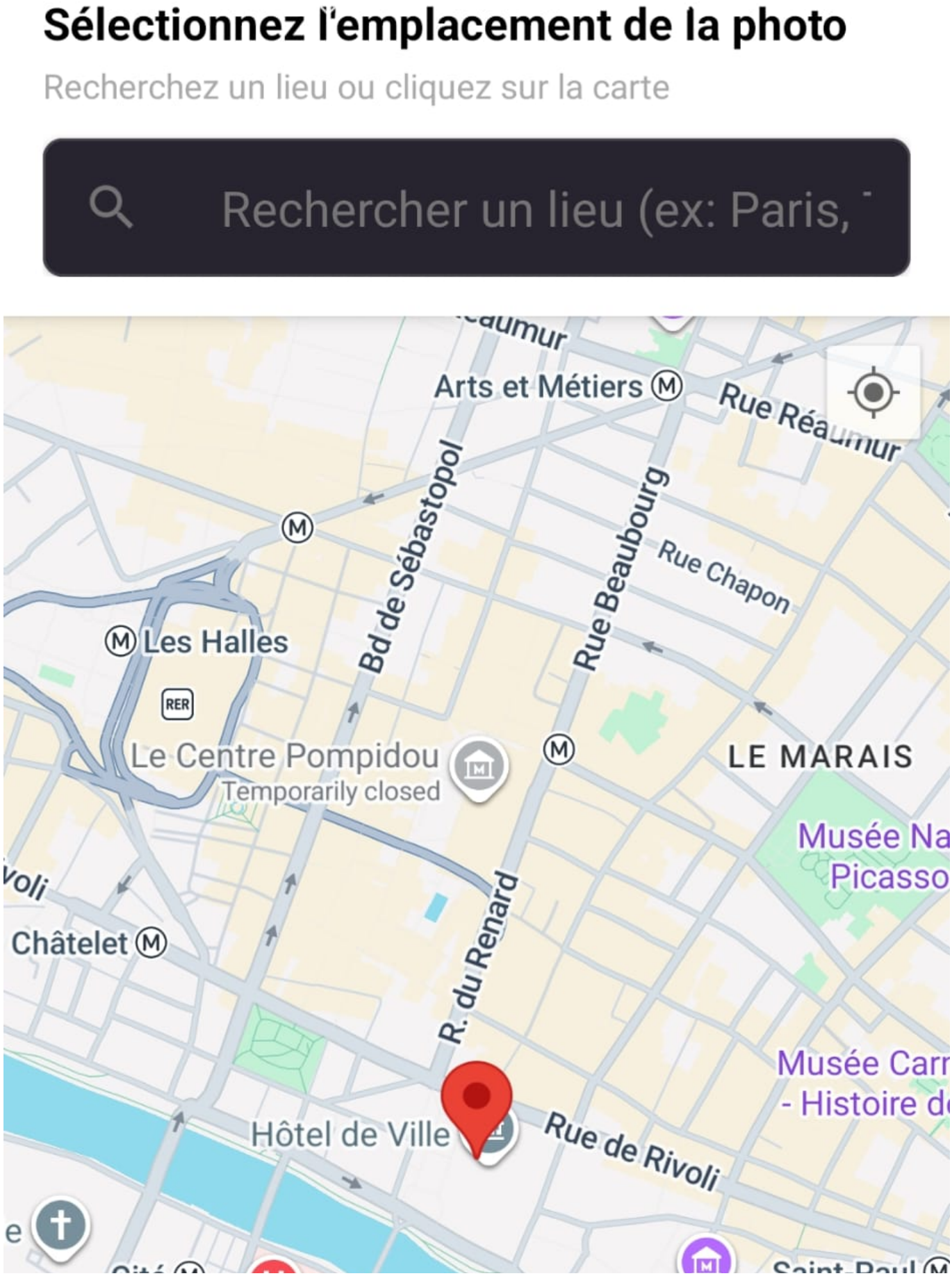

Orange F 2:58
55
Publier une photo
Photos
Collections

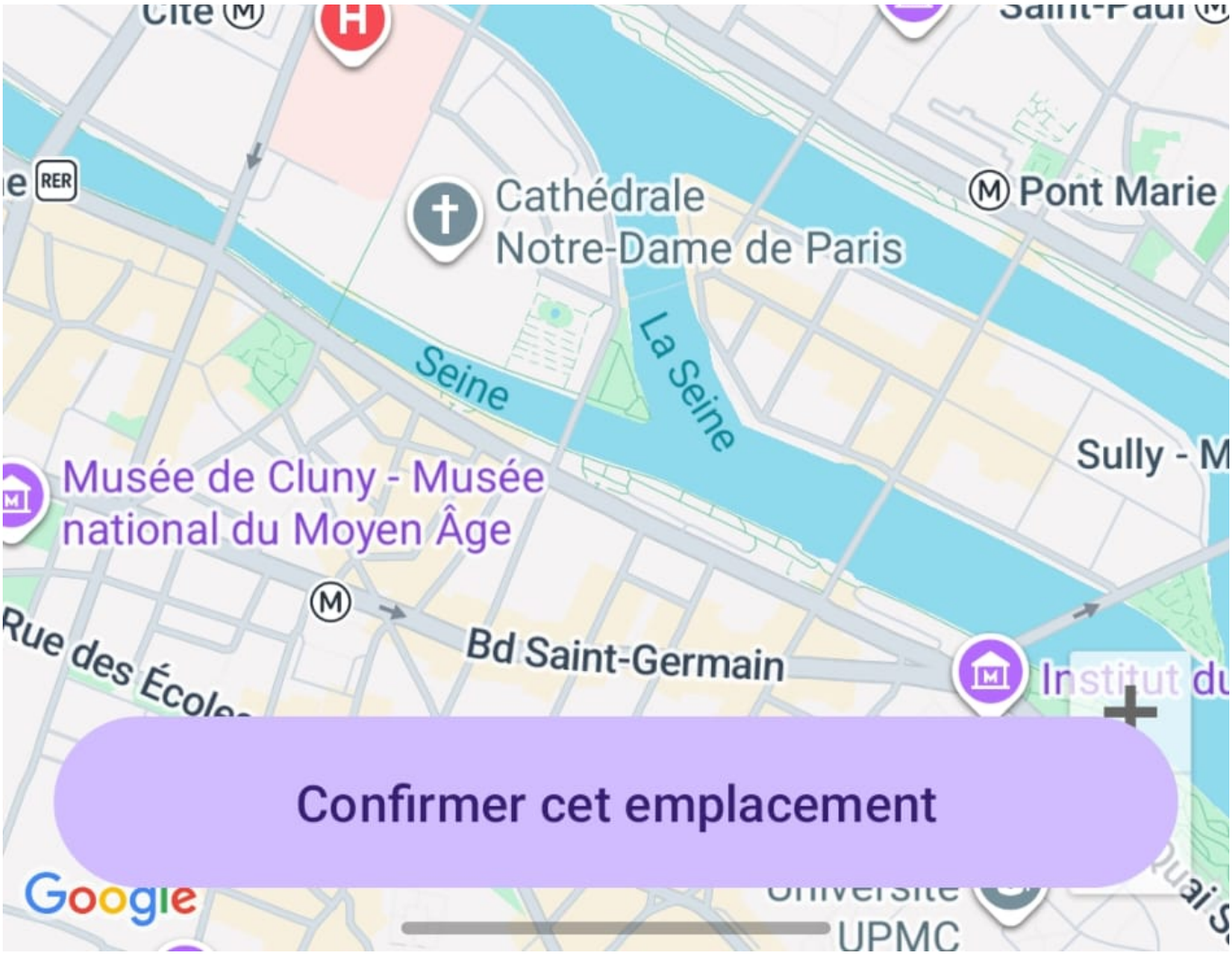


Géolocalisation : Pour la localisation, j'ai implémenté deux approches complémentaires :

1. **Sélection manuelle sur carte** : L'utilisateur peut placer un marqueur sur une carte Google Maps en cliquant à l'endroit exact où la photo a été prise. Lorsqu'il clique, je capture les coordonnées GPS (latitude/longitude) et j'utilise l'API Geocoding pour récupérer automatiquement l'adresse complète, la ville et le pays correspondants.

2. **Position GPS actuelle** : Un bouton permet d'utiliser la localisation actuelle de l'appareil via le service de localisation Android. Cette option est utile si l'utilisateur publie une photo immédiatement après l'avoir prise.





Enrichissement Contextuel : J'ai ajouté plusieurs champs pour enrichir chaque photo :

- **Description** : Texte libre permettant de raconter l'histoire derrière la photo
- **Type de lieu** : Sélection parmi 10 catégories prédéfinies (Monument, Musée, Restaurant, Nature, Plage, Montagne, Ville, Parc, Lac, Autre)
- **Instructions d'accès** : Conseils pour se rendre au lieu (transports, difficultés, horaires)

Visibilité et Partage : L'utilisateur peut choisir de rendre sa photo publique (visible par tous dans le flux principal et sur la carte) ou de la partager uniquement avec des groupes spécifiques via un système de sélection multiple.

Processus d'Upload : Une fois toutes les informations saisies, le processus d'upload se déroule en deux étapes :

1. Upload de l'image compressée sur Firebase Storage dans un dossier dédié (`photos/{userId}/{timestamp}.jpg`)
2. Création d'un document Firestore contenant toutes les métadonnées et l'URL de l'image

Un indicateur de progression informe l'utilisateur de l'avancement de l'upload.

3.3 Découverte et Flux Social

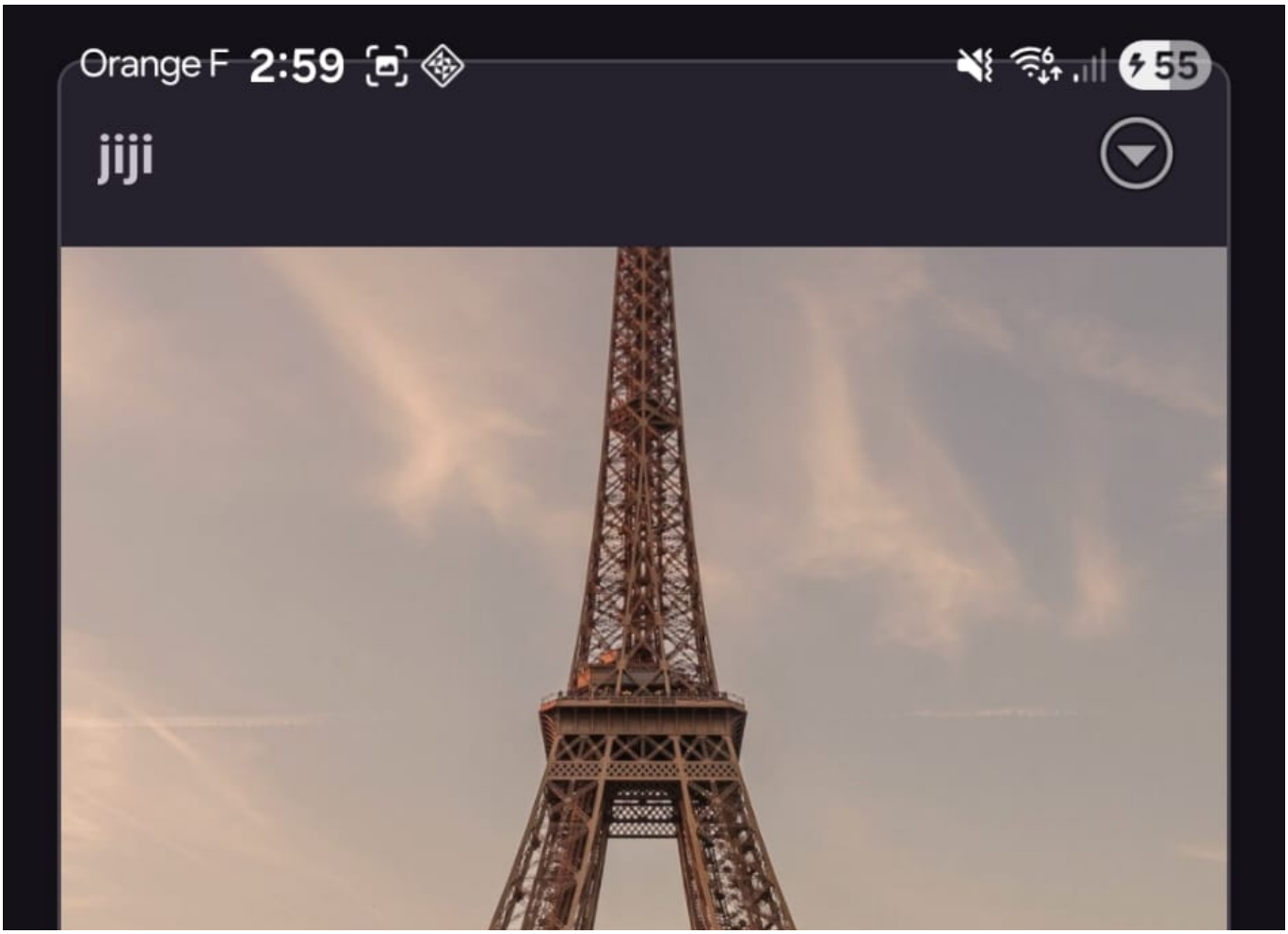
Implémentation : Le HomeFragment affiche un flux de photos publiques permettant aux utilisateurs de découvrir de nouveaux lieux.

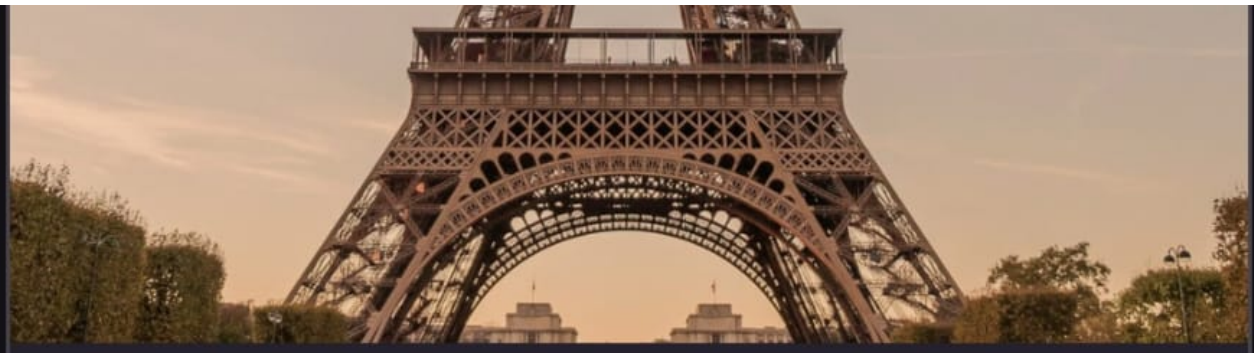
Chargement des Photos : Les photos sont récupérées depuis Firestore avec une requête filtrée sur le champ `isPublic = true`. Pour optimiser les performances, j'ai limité le nombre de photos chargées initialement à 20. Les images sont chargées de manière asynchrone via la bibliothèque Glide qui gère automatiquement le cache et les placeholders.

Rafraîchissement : Un SwipeRefreshLayout permet de rafraîchir le flux en glissant vers le bas. Cette action déclenche une nouvelle requête Firestore qui peut récupérer de nouvelles photos publiées depuis la dernière consultation.

Interactions Sociales : Sur chaque photo du flux, l'utilisateur peut :

- **Liker** : Un bouton cœur change d'état (rempli/vide) lors du clic. Le compteur de likes s'incrémente/décrémente en temps réel. Côté backend, une transaction Firestore garantit la cohérence des compteurs même en cas d'accès concurrent. Le like est enregistré dans une sous-collection `likes` de la photo pour permettre de savoir qui a liké.
- **Commenter** : Un bouton ouvre le PhotoDetailFragment où l'utilisateur peut voir tous les commentaires et en ajouter. Les commentaires sont stockés dans une collection Firestore séparée avec référence à la photo et à l'auteur.
- **Partager avec un groupe** : Un bouton permet d'ajouter la photo à un ou plusieurs groupes dont l'utilisateur est membre. Techniquement, cela ajoute l'ID du groupe dans le tableau `sharedWithGroupIds` de la photo.
- **Visiter (Intégration TravelPaths)** : Ce bouton sera détaillé dans la section intégration inter-applications.





0 0

À visiter

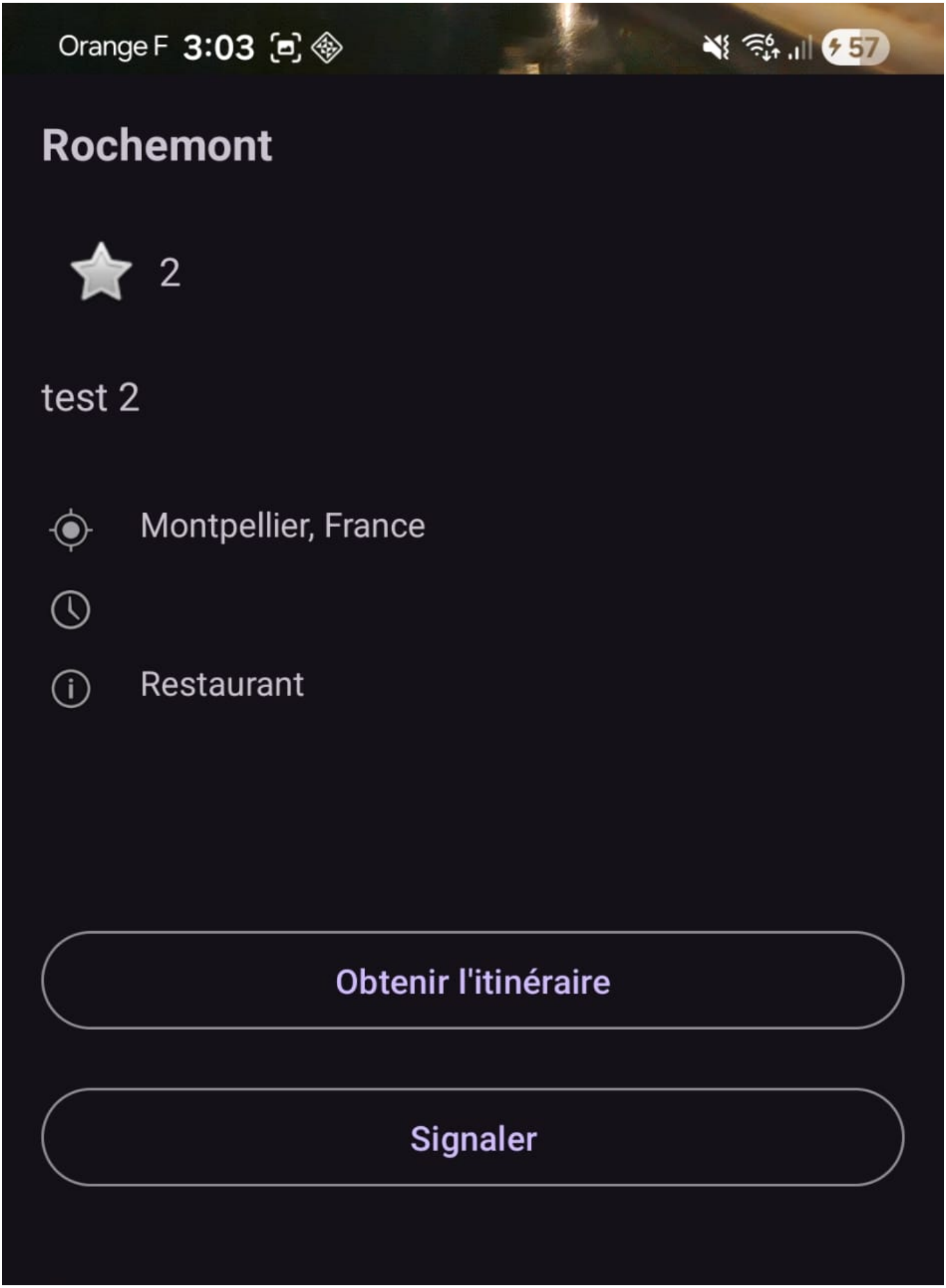
tour

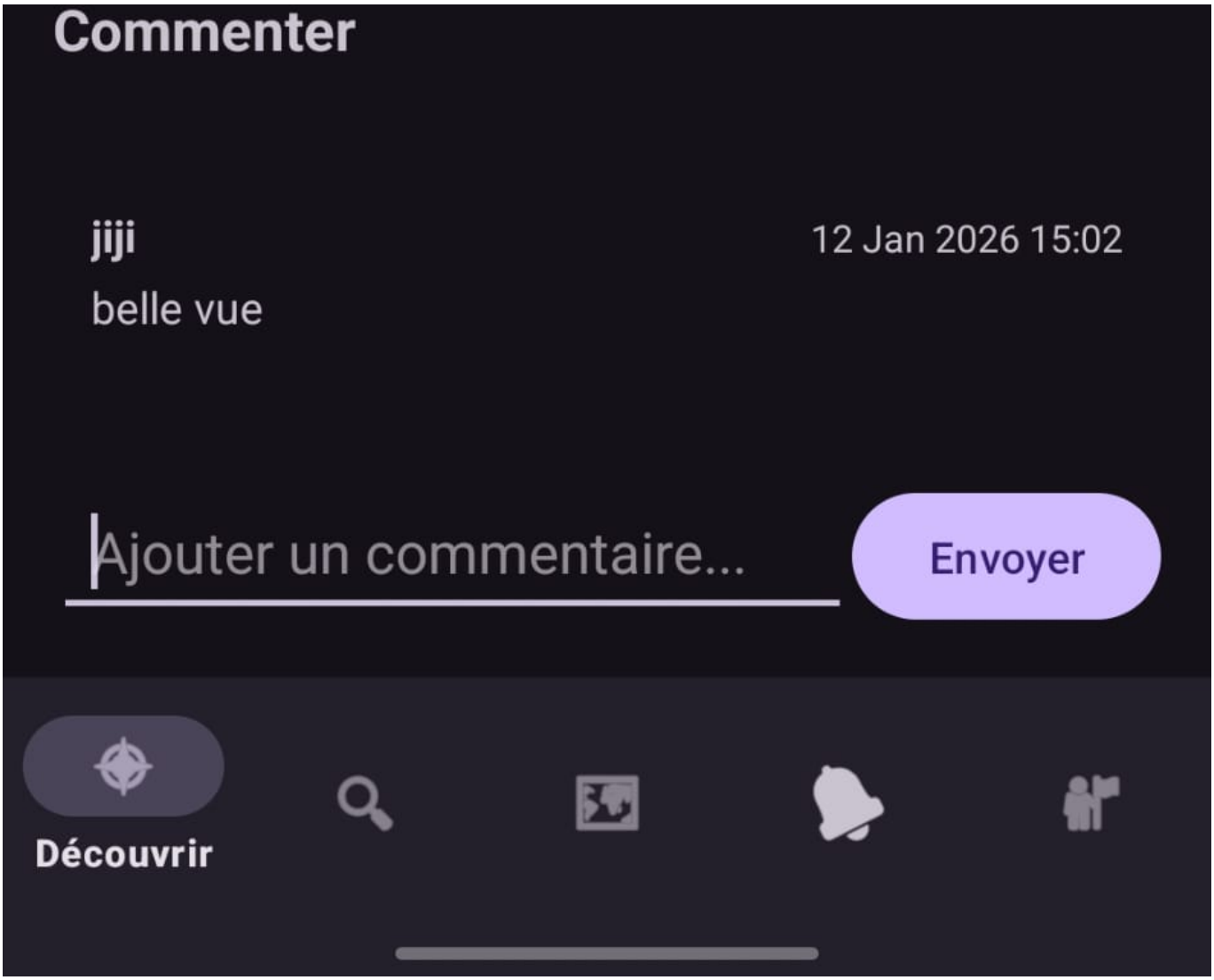
Paris, France 10 Jan 2026

jiji

Découvrir

Affichage des Détails : En cliquant sur une photo, l'utilisateur accède à PhotoDetailFragment qui affiche l'image en plein écran avec toutes les métadonnées : auteur, date, description, localisation, type de lieu, instructions. Un RecyclerView affiche tous les commentaires avec leur auteur et date. Un bouton "Obtenir l'itinéraire" lance Google Maps avec les coordonnées de la photo.





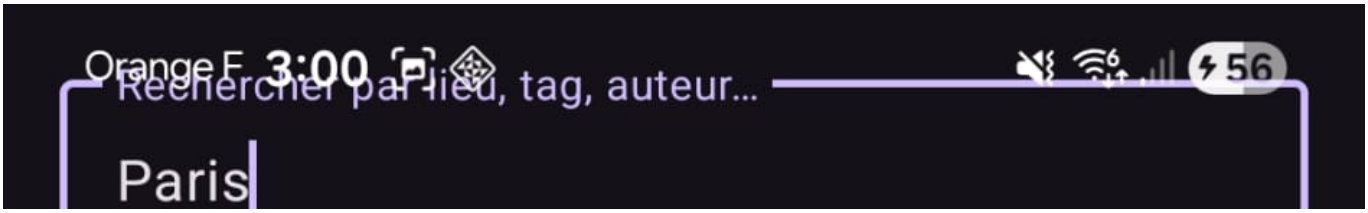
3.4 Recherche et Filtrage de Photos

Implémentation : Le SearchFragment permet aux utilisateurs de trouver des photos spécifiques selon différents critères. J'ai implémenté un système de recherche combinant texte libre et filtres par catégorie.

Recherche Textuelle : Un champ de recherche avec un TextWatcher déclenche une recherche en temps réel à chaque modification du texte. Les requêtes Firestore recherchent les correspondances dans les champs `city` et `country` des localisations de photos. Pour optimiser les performances, j'ai ajouté un délai de 300ms avant de lancer la recherche (debouncing) pour éviter trop de requêtes pendant la frappe.

Filtrage par Type : Un Spinner permet de filtrer les résultats par type de lieu. Lorsqu'un type est sélectionné (Monument, Musée, Restaurant, etc.), la requête Firestore ajoute une clause WHERE sur le champ `photoType`. Les filtres peuvent être combinés avec la recherche textuelle pour affiner les résultats.

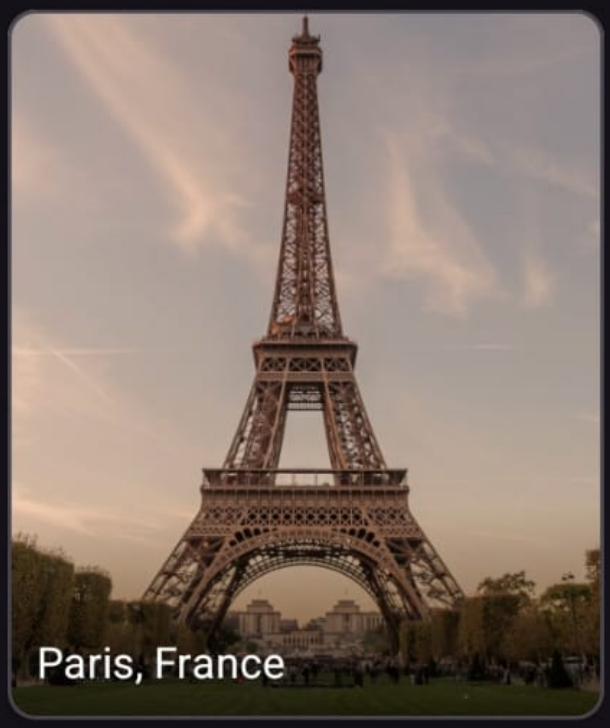
Affichage des Résultats : Les résultats s'affichent dans une grille (GridLayoutManager avec 2 colonnes) pour maximiser le nombre de photos visibles. Chaque photo affiche une miniature, le nom de la ville et le type de lieu. Un clic ouvre les détails complets de la photo.

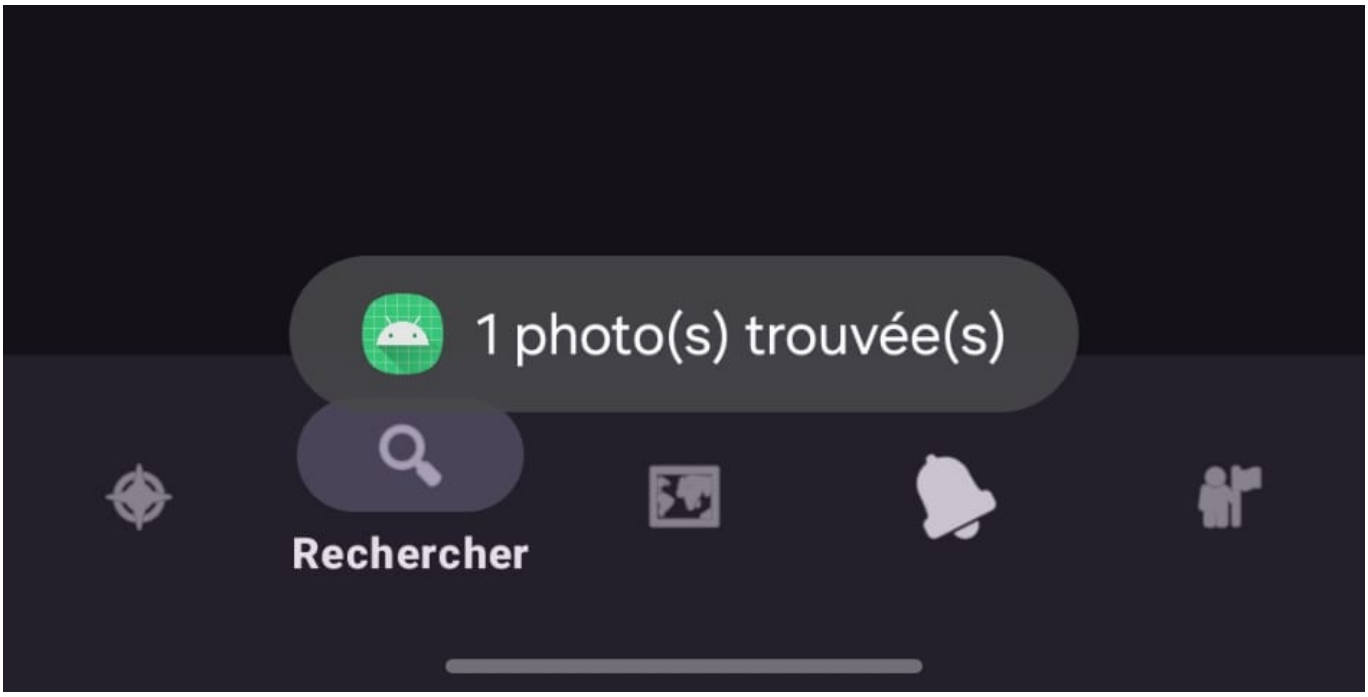


Filtrer par type

Tous les types

Rechercher



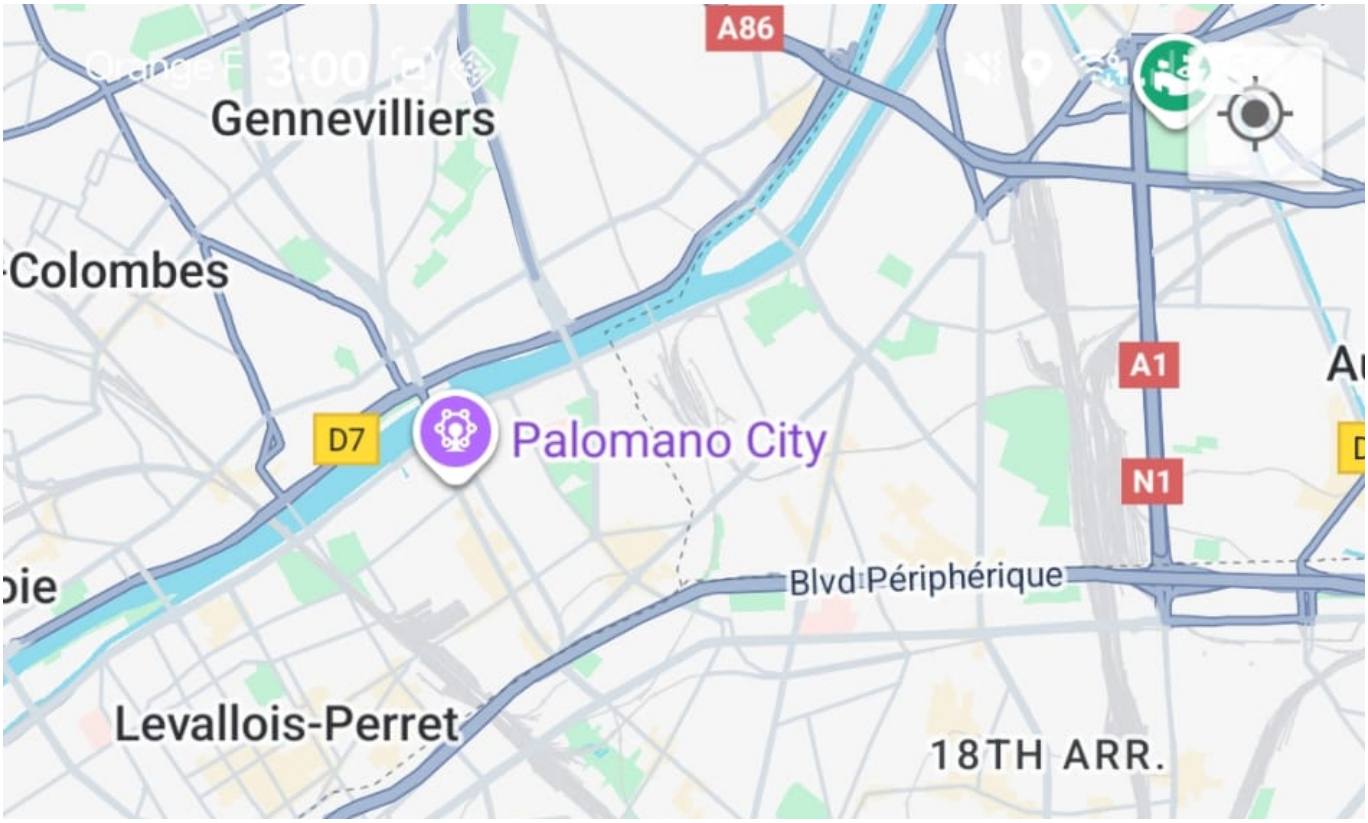


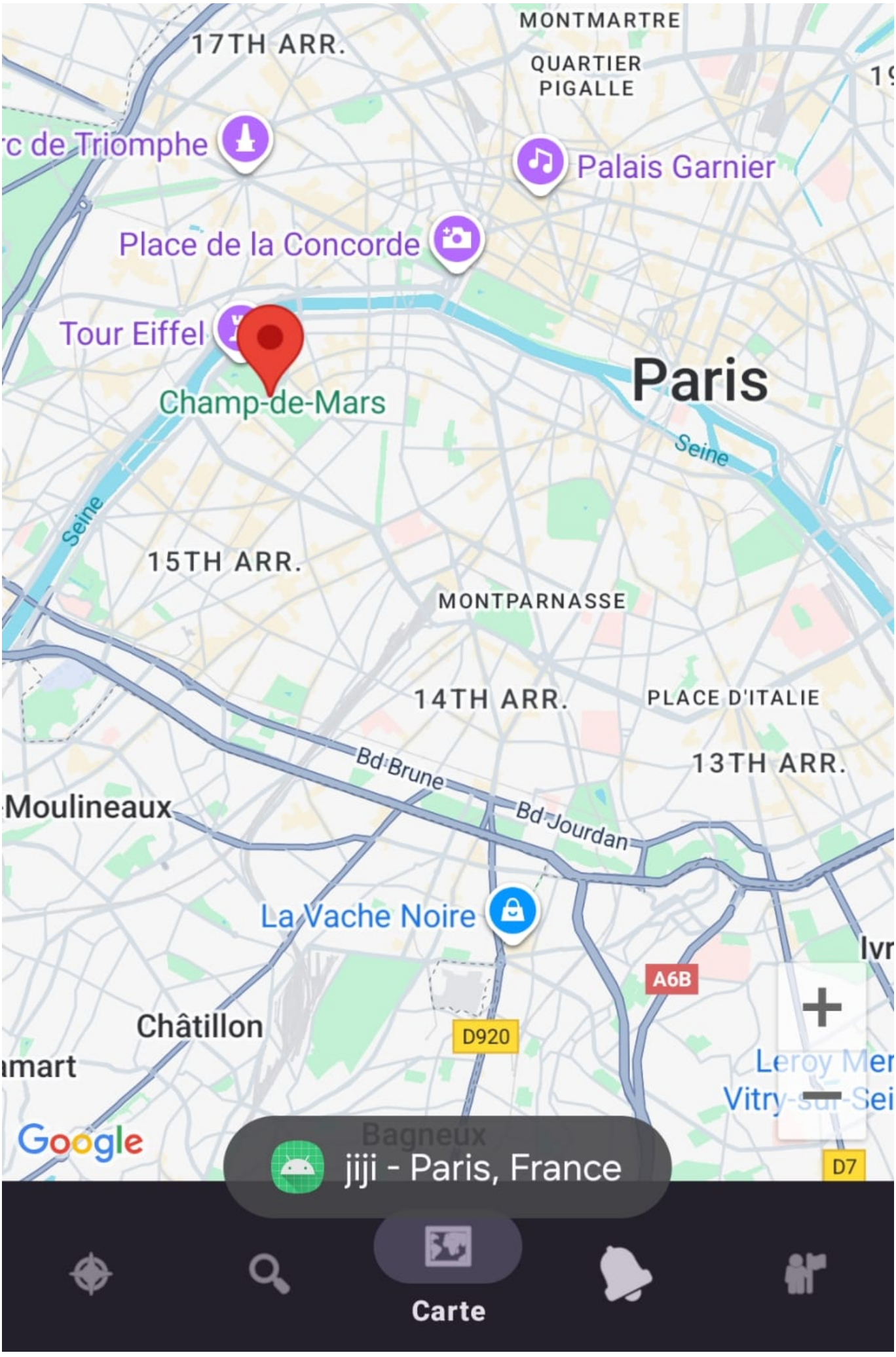
3.5 Visualisation Cartographique

Implémentation : Le MapFragment affiche toutes les photos publiques sur une carte Google Maps interactive, offrant une perspective géographique unique sur les découvertes des voyageurs.

Chargement des Marqueurs : Au chargement du fragment, une requête Firestore récupère toutes les photos publiques ayant des coordonnées GPS valides. Pour chaque photo, je crée un marqueur personnalisé (BitmapDescriptorFactory.defaultMarker rouge) à sa position exacte. Les données de la photo sont stockées dans le tag du marqueur pour un accès rapide lors du clic.

Interaction avec les Marqueurs : Lorsque l'utilisateur clique sur un marqueur, une InfoWindow personnalisée s'affiche montrant le nom de l'auteur et l'adresse du lieu.




MONTMARTRE
17TH ARR.
QUARTIER
PIGALLE
Palais Garnier
Place de la Concorde
Tour Eiffel
Champ-de-Mars
Paris
Seine
Seine
15TH ARR.
MONTPARNASSE
14TH ARR.
PLACE D'ITALIE
13TH ARR.
Bd Brune
Bd Jourdan
Moulineaux
La Vache Noire
A6B
Châtillon
D920
Google
jiji - Paris, France
D7
Carte

3.6 Groupes de Voyage Collaboratifs

Implémentation : Le système de groupes permet de partager des photos de manière privée avec un cercle restreint de personnes (amis, famille, collègues de voyage).

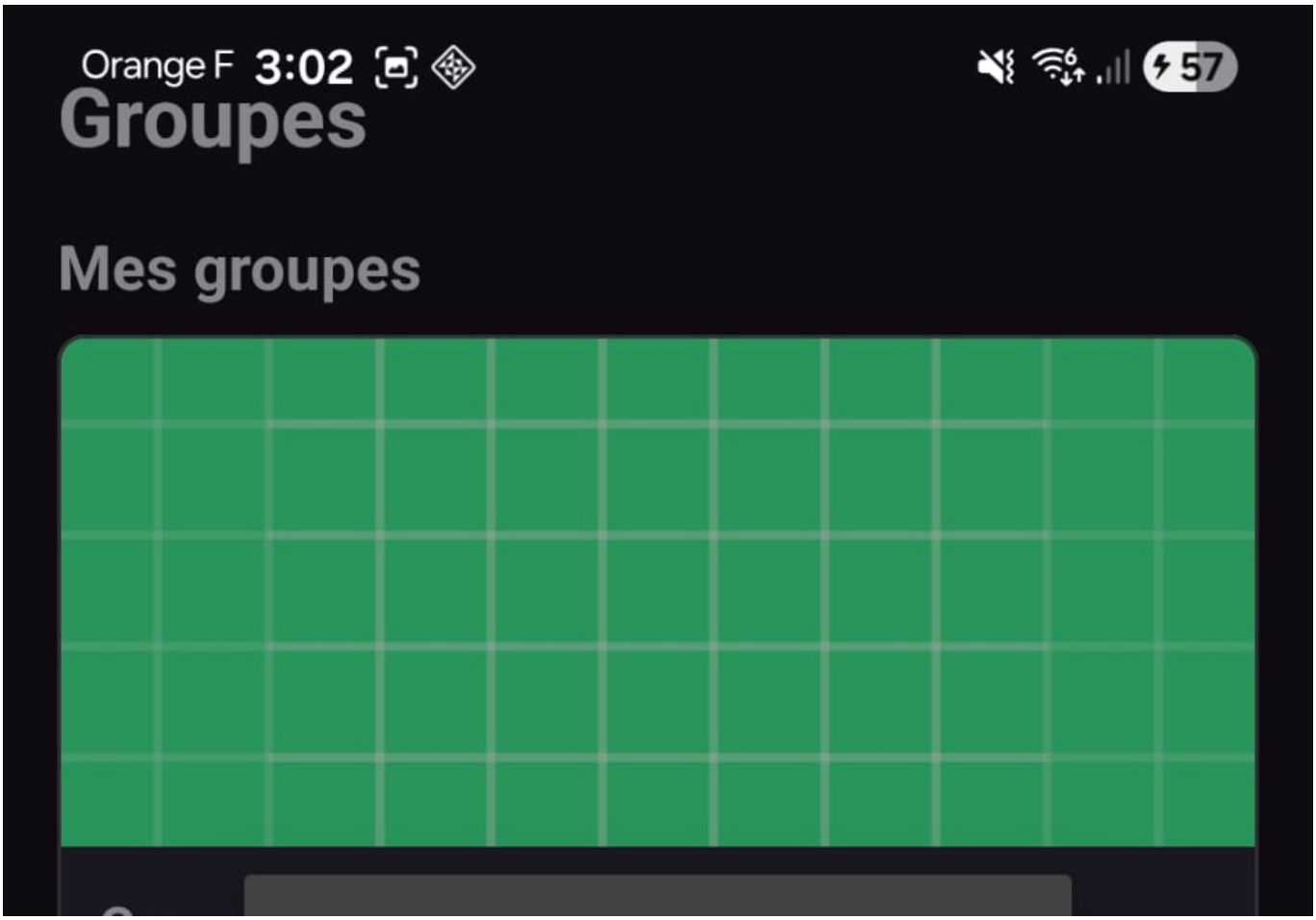
Création de Groupes : L'utilisateur peut créer un groupe en fournissant un nom et une description. Un document est créé dans la collection Firestore `groups` avec le créateur comme premier membre. L'ID du groupe est automatiquement ajouté au tableau `groupIds` de l'utilisateur.

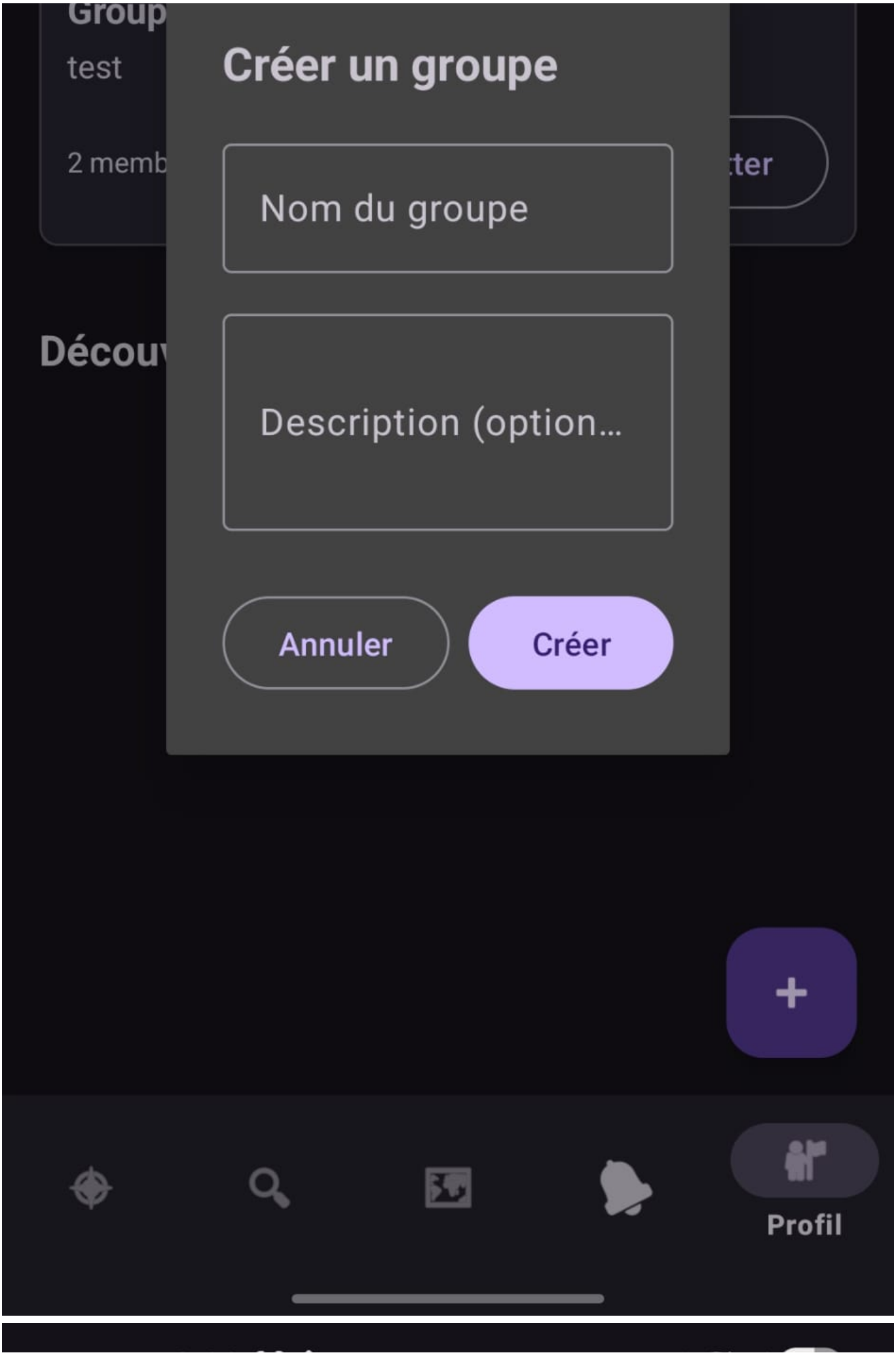
Découverte et Adhésion : Tous les groupes existants sont affichés dans une liste consultable. L'utilisateur peut parcourir les groupes, lire leurs descriptions et rejoindre ceux qui l'intéressent. L'adhésion ajoute automatiquement l'ID de l'utilisateur au tableau `memberIds` du groupe et vice-versa.

Partage de Photos : Lors de la publication d'une photo, l'utilisateur peut sélectionner un ou plusieurs groupes avec lesquels partager. Les IDs des groupes sélectionnés sont stockés dans le tableau `sharedWithGroupIds` de la photo. Une photo peut être à la fois publique ET partagée avec des groupes.

Consultation des Photos de Groupe : En cliquant sur un groupe, l'utilisateur voit toutes les photos partagées avec ce groupe dans une grille. La requête Firestore utilise `arrayContains` sur le champ `sharedWithGroupIds` pour récupérer uniquement les photos pertinentes.

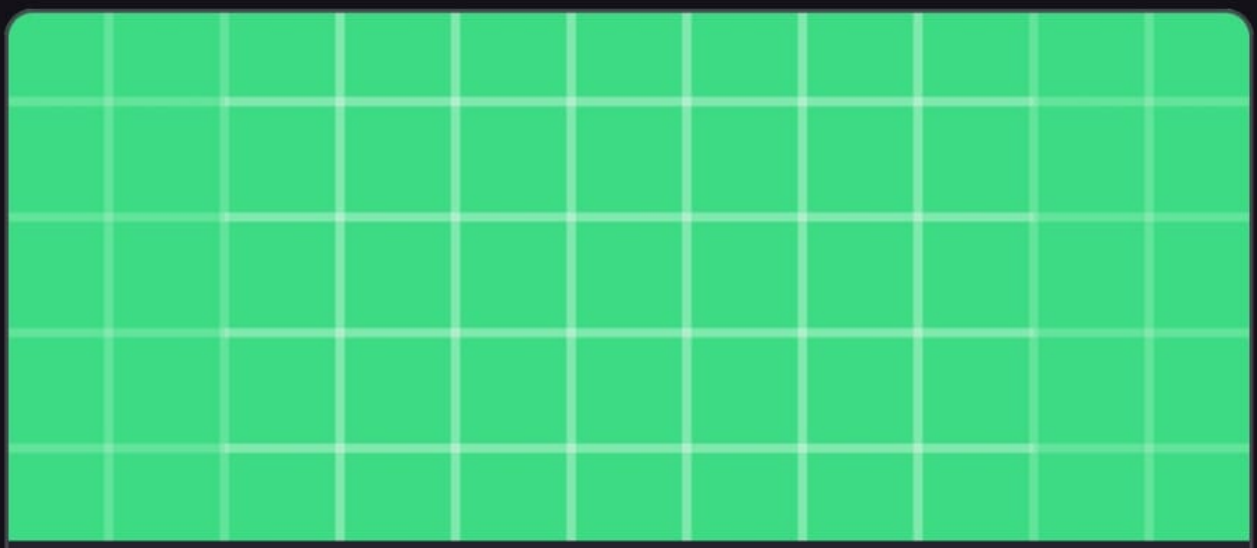
Gestion des Appartenances : L'utilisateur peut quitter un groupe à tout moment. Cette action retire son ID du groupe et l'ID du groupe de son profil. Les photos qu'il a partagées avec ce groupe restent visibles aux autres membres.




Group
test
2 memb
tter
Découv
Créer un groupe
Nom du groupe
Description (option...
Annuler
Créer
Profil

Groupes

Mes groupes



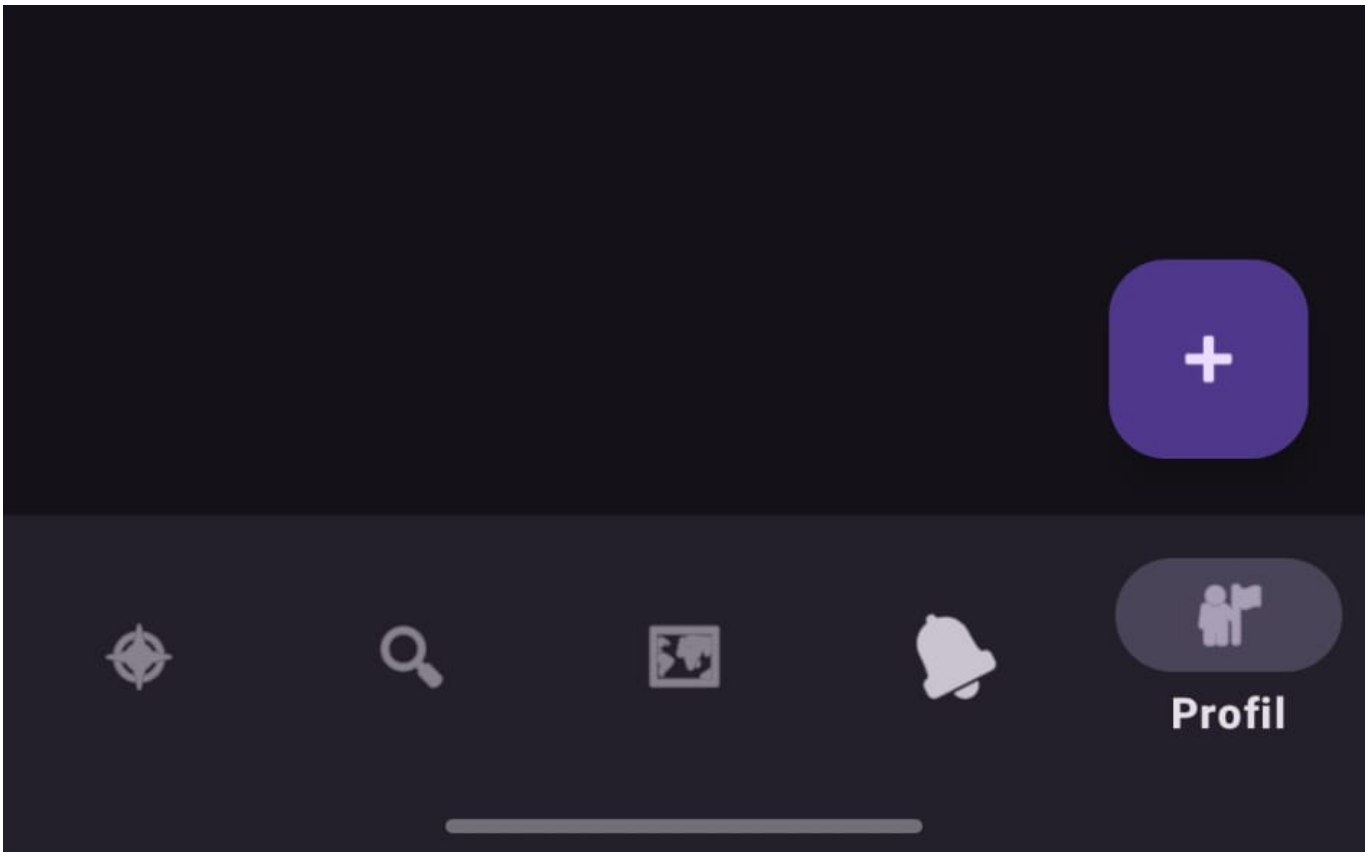
Groupe jivan

test

2 membres

Quitter

Découvrir des groupes



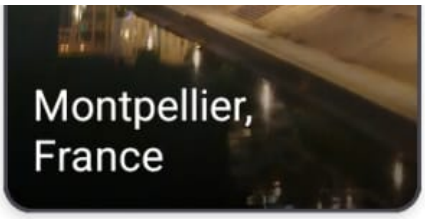
Orange F 3:02

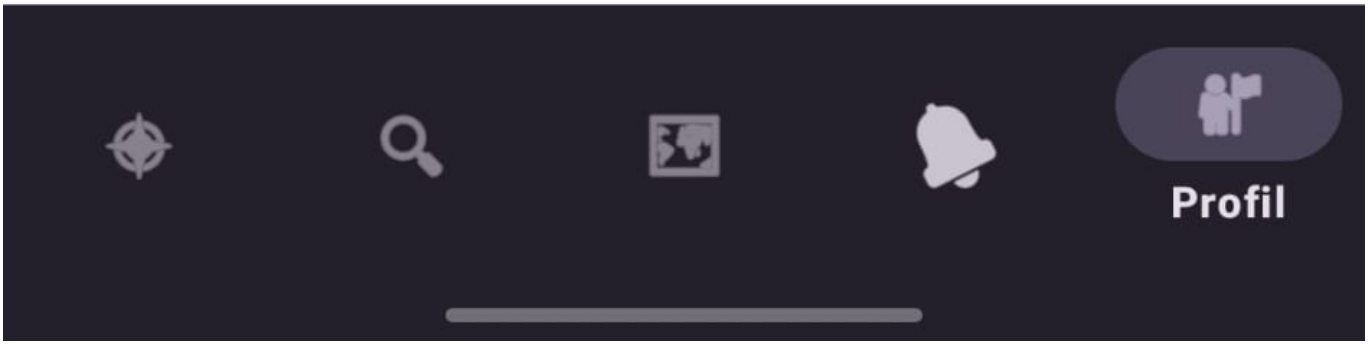
Groupe jivan

test

2 membre(s)

Photos partagées





3.7 Profil Personnel et Gestion de Contenu

Implémentation : Le ProfileFragment est le centre de contrôle personnel de l'utilisateur où il peut gérer son identité et son contenu.

Affichage du Profil : Le profil affiche une image par défaut, le nom d'utilisateur, la biographie, les photos publiées et les groupes dont l'utilisateur est membre.

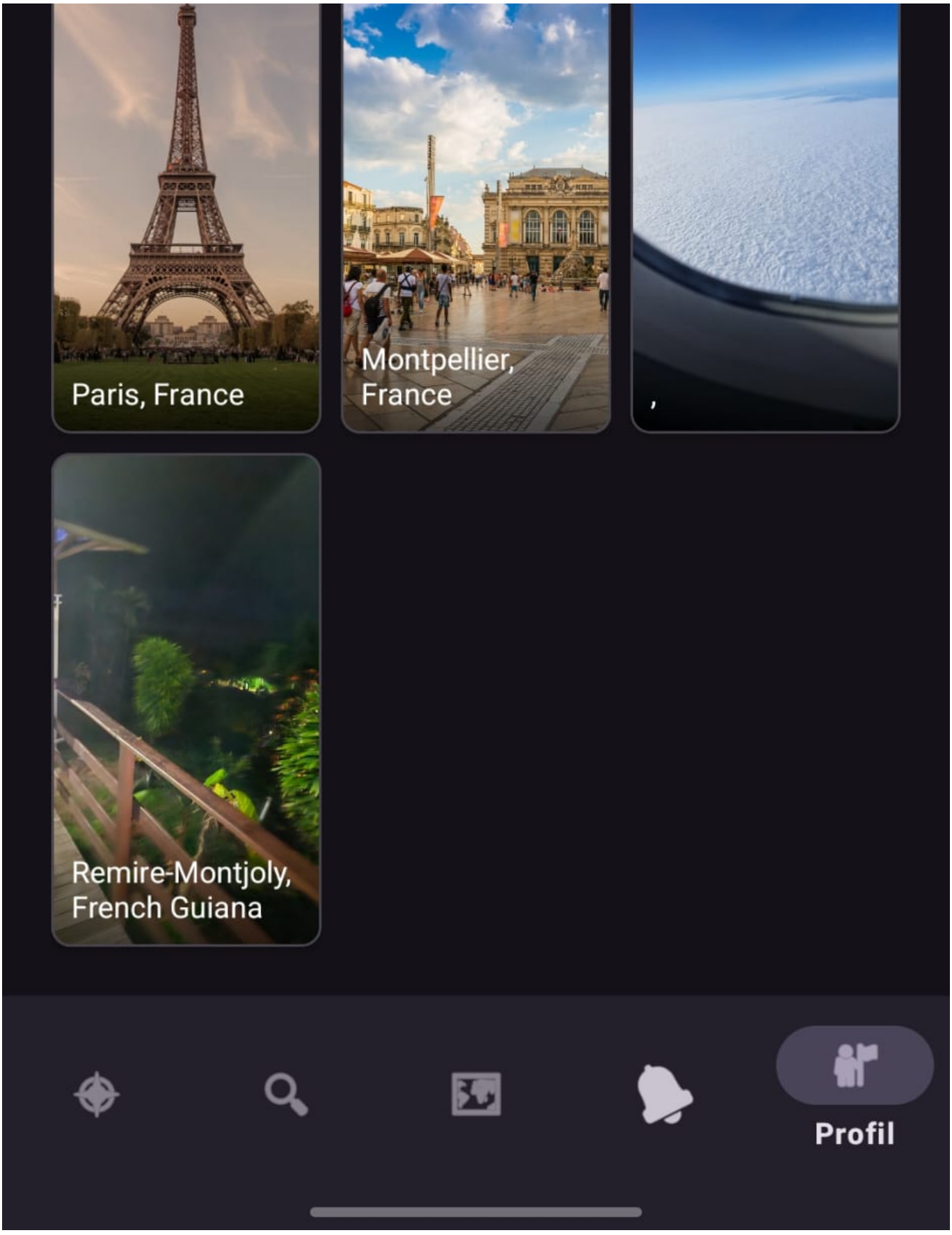
Galerie Personnelle : Toutes les photos publiées par l'utilisateur s'affichent dans une grille sous le profil. Une requête Firestore filtre sur `authorId` égal à l'ID de l'utilisateur connecté. Cette galerie permet de visualiser rapidement tout son contenu partagé.

Édition du Profil : Des boutons permettent de modifier le nom d'utilisateur, la biographie . Pour la photo de profil, le processus est similaire à l'upload de photos : sélection depuis la galerie, compression, upload sur Storage, mise à jour de l'URL dans Firestore.

Suppression de Photos : L'utilisateur peut supprimer ses propres photos. Un appui long sur une photo dans sa galerie affiche une option de suppression qui:

1. Supprime l'image de Firebase Storage
2. Supprime le document Firestore de la photo
3. Supprime toutes les sous-collections associées (commentaires, likes)
4. Met à jour les compteurs de l'utilisateur

Déconnexion : Un bouton de déconnexion appelle Firebase Authentication pour se déconnecter et redirige vers l'écran de connexion. Le token FCM est invalidé côté serveur pour arrêter les notifications sur cet appareil.



3.8 Système de Notifications Push

Implémentation : J'ai mis en place un système automatisé utilisant Firebase Cloud Messaging et Cloud Functions.

Architecture Backend : J'ai développé quatre Cloud Functions en Node.js (2nd Generation) déployées sur Firebase :

1. **sendLikeNotification** : Se déclenche automatiquement lorsqu'un document est créé dans la sous-collection `likes` d'une photo. La fonction récupère l'auteur de la photo, vérifie qu'il n'est pas celui qui a liké (pas de notification à soi-même), récupère son token FCM, et envoie une notification push avec le message "X a aimé votre photo".

2. **sendCommentNotification** : Fonctionne de manière similaire mais se déclenche lors de la création d'un commentaire. Le message est "X a commenté votre photo".

3. **sendNotificationOnCreate** : Fonction générique qui enregistre également la notification dans Firestore pour l'historique.

4. **cleanupOldNotifications** : Cloud Scheduler qui s'exécute quotidiennement pour supprimer les notifications de plus de 30 jours, évitant ainsi l'accumulation de données.

Gestion des Tokens FCM : Lors de la connexion, l'application récupère le token FCM unique de l'appareil et le stocke dans le document utilisateur Firestore. Ce token est automatiquement mis à jour si Firebase le renouvelle. Cela permet d'envoyer des notifications même si l'utilisateur change d'appareil.

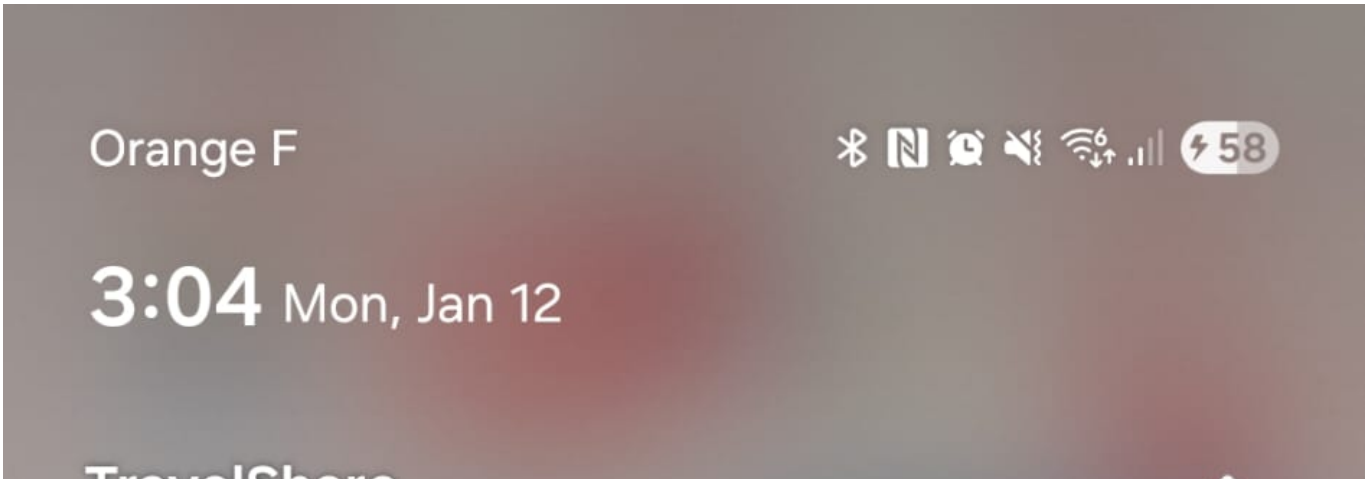
Interface de Notifications : Le NotificationsFragment affiche toutes les notifications de l'utilisateur dans une liste chronologique. Chaque notification affiche une icône selon son type (cœur pour like, bulle pour commentaire), le nom de l'utilisateur qui a effectué l'action, le temps écoulé depuis l'événement, et un aperçu de la photo concernée.

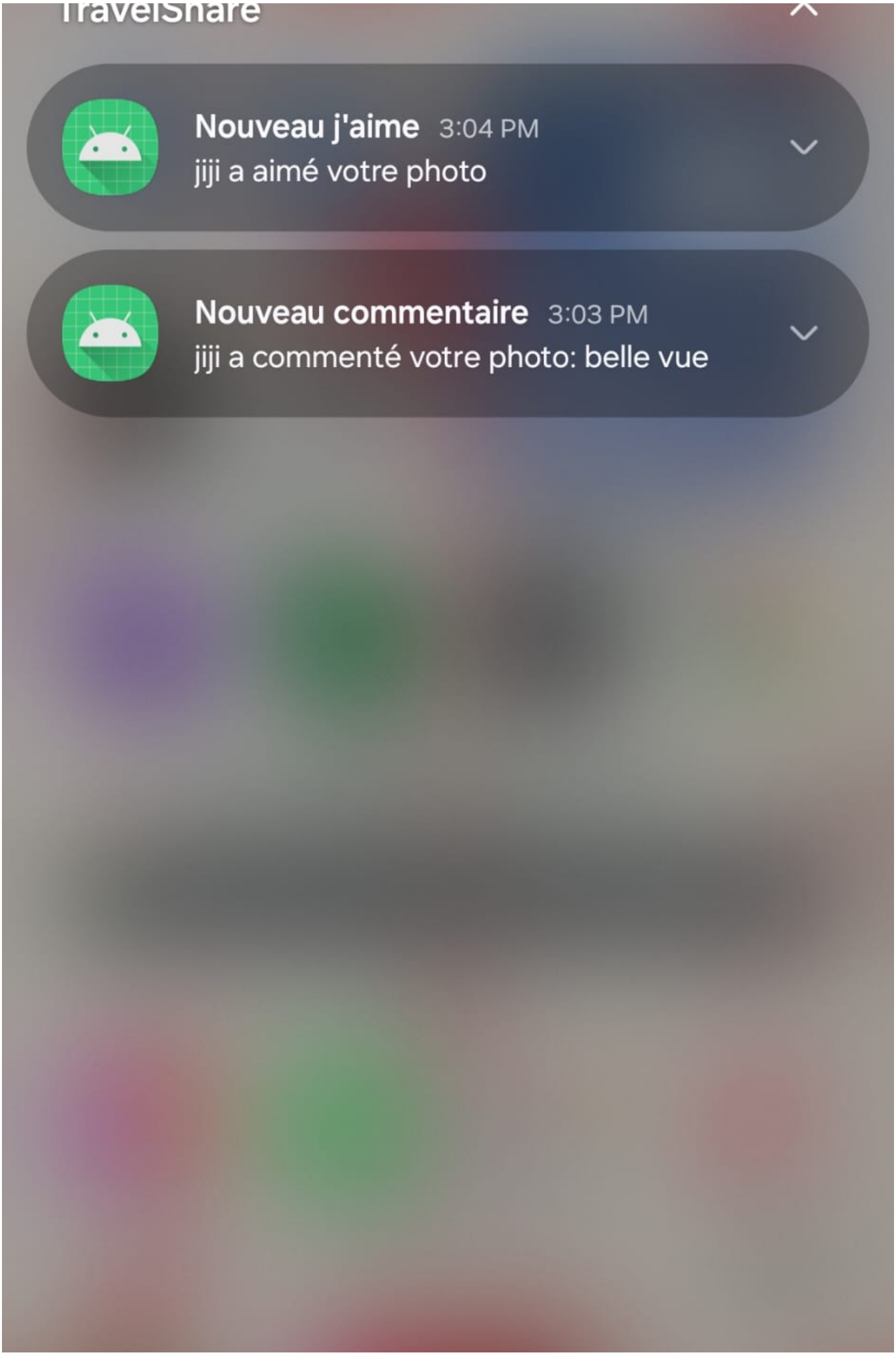
Badge de Notifications Non Lues : Un badge rouge sur l'icône de notifications dans la barre de navigation indique le nombre de notifications non lues. Ce compteur se met à jour en temps réel grâce à un listener Firestore sur la collection notifications.

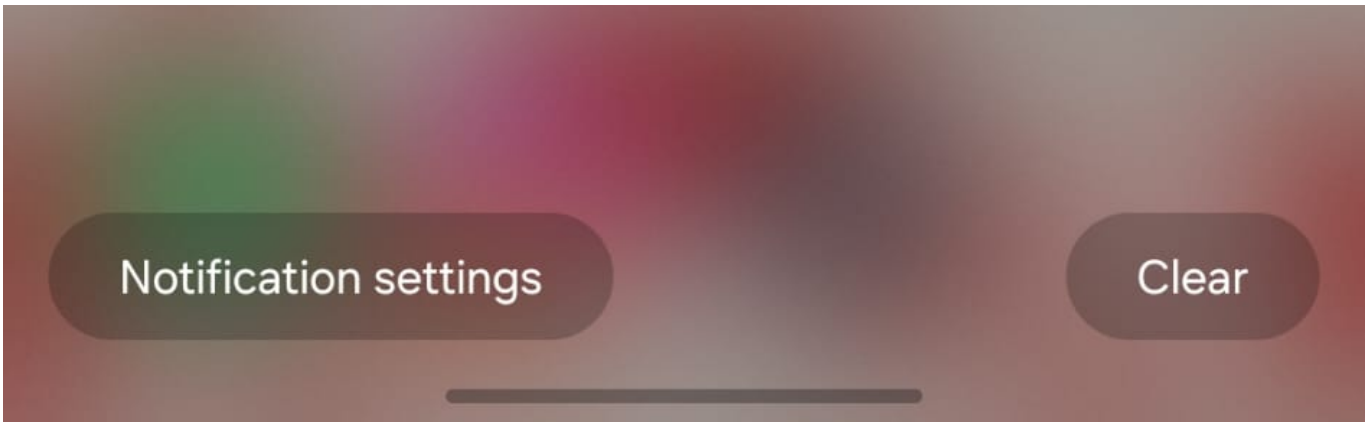
Interactions : L'utilisateur peut :

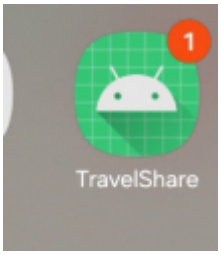
- Cliquer sur une notification pour être redirigé vers la photo concernée
- Marquer une notification comme lue (son fond change de couleur)
- Les notifications sont automatiquement marquées comme lues après consultation

Réception Hors-Ligne : Grâce à FCM, les notifications arrivent même si l'application est fermée ou en arrière-plan. Un service Firebase Messaging reçoit les notifications et les affiche dans la barre de notification Android avec une icône personnalisée.




TravelShare
Nouveau j'aime 3:04 PM
jiji a aimé votre photo
Nouveau commentaire 3:03 PM
jiji a commenté votre photo: belle vue





3.9 Intégration Inter-Applications avec TravelPaths

Contexte : TravelPaths est une application développée par un collègue qui génère des parcours de visite optimisés en fonction des préférences de l'utilisateur. L'intégration permet aux utilisateurs de TravelShare d'exporter facilement des lieux intéressants découverts via des photos vers TravelPaths pour créer des itinéraires de voyage.

Implémentation Technique : J'ai utilisé le système d'Intents Android pour permettre la communication entre les deux applications indépendantes.

Mécanisme d'Intent : Sur chaque photo du flux d'accueil, un bouton "À visiter" déclenche la création d'un Intent explicite ciblant l'activité `PathDesignerActivity` de TravelPaths. L'Intent utilise l'action personnalisée `com.travelpath.ACTION_OPEN_PATH_DESIGNER_ACTIVITY` définie dans le manifest de TravelPaths.

Données Transférées : L'Intent contient plusieurs extras permettant à TravelPaths de créer une destination complète :

- **Coordonnées GPS** : Latitude et longitude exactes extraites de l'objet Location de la photo
- **Nom du lieu** : Nom de la ville ou adresse récupéré via Geocoding
- **Description** : La description complète fournie par l'auteur de la photo
- **Type de lieu** : La catégorie (Monument, Restaurant, etc.) transformée en string
- **URL de l'image** : Lien vers la photo sur Firebase Storage pour affichage dans TravelPaths

Exemple de code simplifié :

```
Intent intent = new Intent("com.travelpath.ACTION_OPEN_PATH_DESIGNER_ACTIVITY");
intent.setPackage("com.travelpath");
intent.putExtra("EXTRA_LAT", latitude);
intent.putExtra("EXTRA_LON", longitude);
intent.putExtra("EXTRA_NAME", cityName);
intent.putExtra("EXTRA_DESC", description);
startActivity(intent);
```

Gestion des Erreurs : Avant de lancer l'Intent, je vérifie :

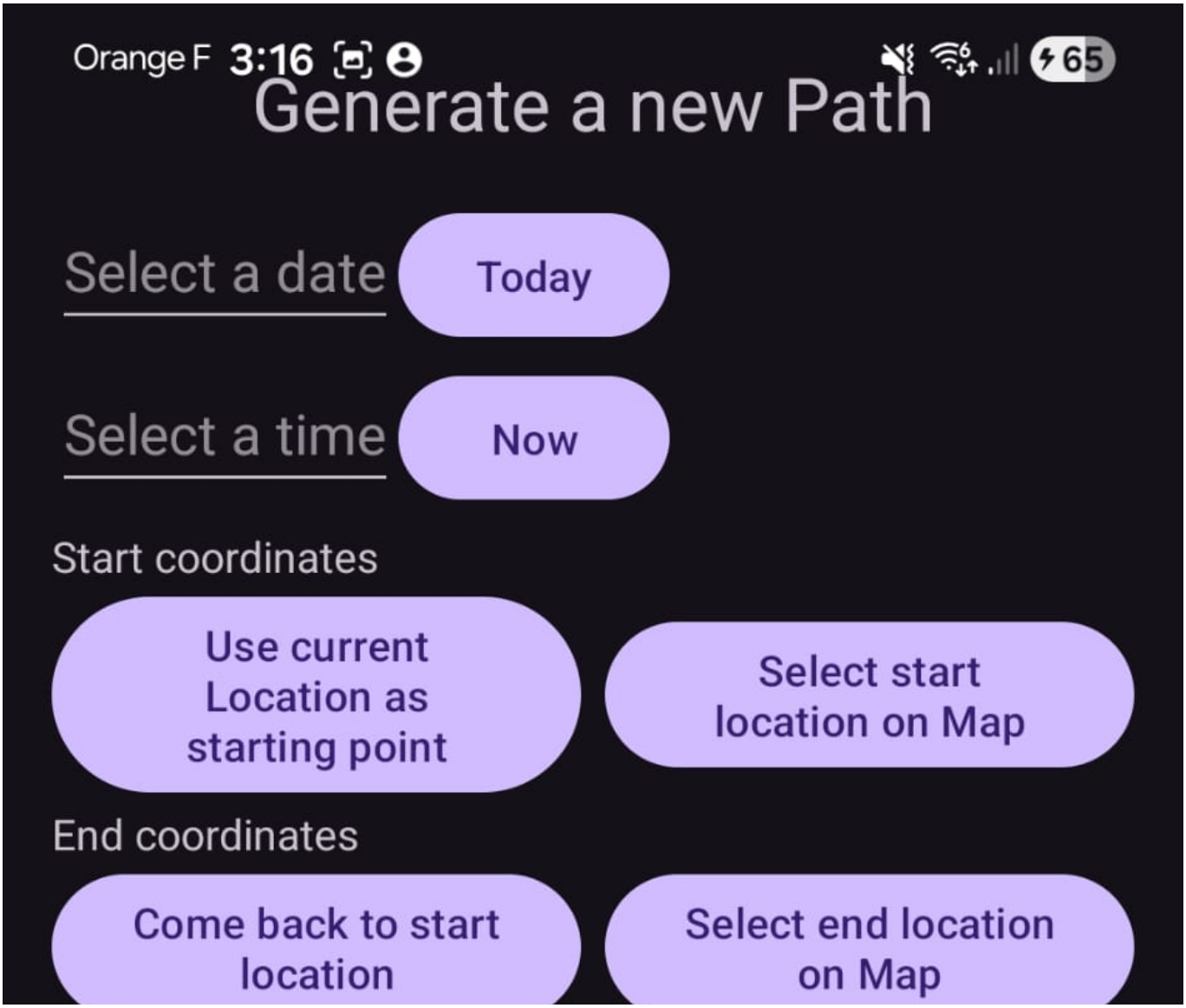
1. Que la photo possède une localisation valide (latitude/longitude non nulles)
2. Que TravelPaths est installé sur l'appareil

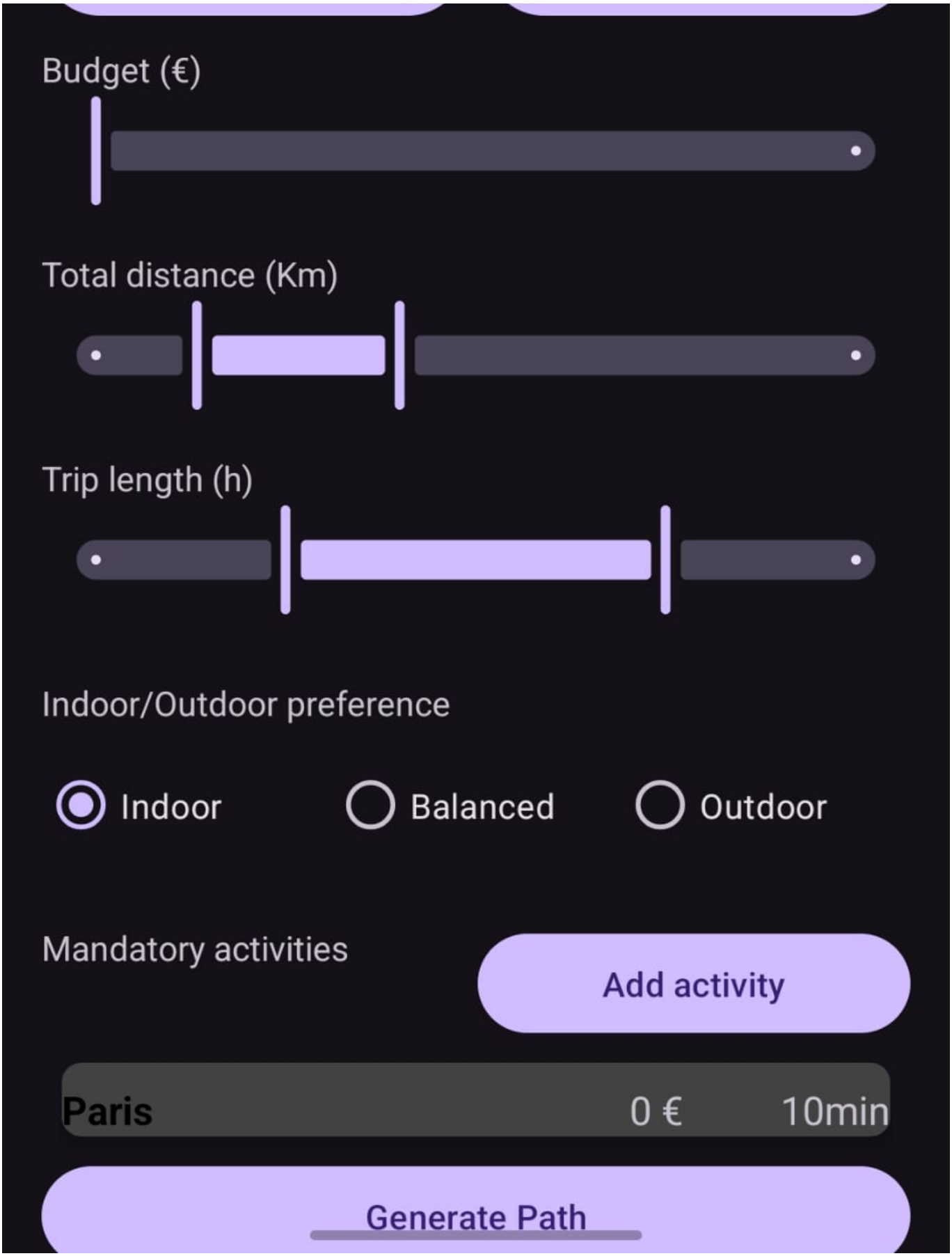
Si la localisation manque, un Toast informe l'utilisateur. Si TravelPaths n'est pas installé, l'exception ActivityNotFoundException est capturée et un message approprié s'affiche.

Déclaration dans le Manifest : Pour permettre la communication, j'ai ajouté une requête de package dans le manifest de TravelShare :

```
<queries>
 <package android:name="com.travelpath" />
</queries>
```

Expérience Utilisateur : l'utilisateur découvre une photo de la tour Eiffel, clique sur "À visiter", et TravelPaths s'ouvre instantanément avec ce lieu pré-rempli comme destination possible dans son générateur de parcours. Il peut ensuite ajouter d'autres lieux, définir ses préférences (budget, durée, effort) et générer un itinéraire complet.





---

4. FONCTIONNALITÉS SECONDAIRES

4.1 Détails Complets de Photo

Lorsque l'utilisateur clique sur une photo depuis n'importe où dans l'application, le PhotoDetailFragment s'ouvre en plein écran. Ce fragment centralise toutes les informations et interactions possibles avec une photo :

Affichage des Métadonnées : L'image s'affiche en grand format via Glide avec zoom activé. En dessous, toutes les informations contextuelles sont présentées : nom et photo de profil de l'auteur, date de publication formatée (ex: "Il y a 2 jours"), description complète, localisation précise (adresse, ville, pays), type de lieu avec une icône appropriée, et instructions d'accès si fournies.

Statistiques Sociales : Les compteurs de likes et de commentaires sont affichés en temps réel. L'état du bouton like reflète si l'utilisateur actuel a déjà liké la photo.

Section Commentaires : Un RecyclerView affiche tous les commentaires avec le nom de l'auteur, le texte et la date. Un champ de saisie en bas permet d'ajouter un nouveau commentaire instantanément. Les commentaires se rechargent automatiquement grâce à un listener Firestore.

Actions Disponibles :

- Bouton "Obtenir l'itinéraire" : Lance Google Maps en mode navigation avec les coordonnées de la photo comme destination
- Bouton "Partager avec groupe" : Permet d'ajouter la photo à un ou plusieurs groupes
- Bouton "Signaler" : Pour signaler du contenu inapproprié (incrémente le compteur de signalements)
- Bouton "Supprimer" : Visible uniquement si l'utilisateur est l'auteur de la photo

4.2 Gestion des Commentaires

Les commentaires sont stockés dans une collection Firestore séparée avec une référence à la photo parente. Chaque commentaire contient l'ID et le nom de l'auteur, le texte, et un timestamp. Un index composite Firestore permet de récupérer efficacement tous les commentaires d'une photo triés par date. La suppression d'une photo supprime automatiquement tous ses commentaires grâce à une Cloud Function de nettoyage.

4.3 Système de Signalements

Les utilisateurs peuvent signaler des photos inappropriées. Chaque signalement incrémente un compteur `reportsCount` sur la photo. Lorsqu'une photo atteint un certain seuil de signalements (ex: 5), une Cloud Function pourrait automatiquement la masquer et notifier un modérateur. Cette fonctionnalité n'est pas encore entièrement implémentée côté modération mais la structure est en place.

4.5 Gestion des Permissions Android

L'application nécessite plusieurs permissions qui doivent être demandées. J'ai implémenté un système de gestion des permissions qui :

- Demande la permission de localisation lors de la première utilisation de la carte ou lors de l'utilisation de la position actuelle
- Demande la permission de stockage/galerie lors de la sélection d'une photo
- Demande la permission de notifications sur Android 13+ pour recevoir les notifications push

---

5. STRUCTURE DE LA BASE DE DONNÉES FIRESTORE

La base de données Firestore est organisée en 5 collections principales optimisées pour les requêtes fréquentes :

5.1 Collection `users`

Stocke les profils utilisateurs avec :

- `id` : UID Firebase Authentication
- `email` : Email de connexion
- `username` : Nom d'utilisateur unique
- `profileImageUrl` : URL de la photo de profil sur Storage
- `bio` : Biographie descriptive
- `fcmToken` : Token pour notifications push
- `groupIds` : Tableau des IDs de groupes dont l'utilisateur est membre
- `createdAt` : Date de création du compte

5.2 Collection `photos`

Contient toutes les photos publiées :

- `id` : Identifiant unique auto-généré
- `authorId` et `authorName` : Référence à l'auteur
- `imageUrl` : URL sur Firebase Storage
- `description` : Description fournie par l'auteur
- `location` : Objet contenant latitude, longitude, address, city, country
- `photoType` : Enum (MONUMENT, MUSEUM, RESTAURANT, etc.)
- `travelInstructions` : Conseils d'accès
- `isPublic` : Visibilité publique (boolean)
- `sharedWithGroupIds` : Tableau des IDs de groupes avec qui la photo est partagée
- `likesCount`, `commentsCount`, `reportsCount` : Compteurs
- `createdAt`, `uploadDate` : Timestamps

Index composites :

- `isPublic` + `createdAt` pour le flux d'accueil
- `authorId` + `createdAt` pour les profils
- `sharedWithGroupIds` + `createdAt` pour les groupes

5.3 Collection `groups`

Gère les groupes de voyage :

- `id` : Identifiant unique
- `name` : Nom du groupe
- `description` : Description du voyage
- `memberIds` : Tableau des IDs des membres
- `createdBy` : ID du créateur
- `createdAt` : Date de création

5.4 Collection `comments`

Stocke tous les commentaires :

- `id` : Identifiant unique
- `photoId` : Référence à la photo commentée
- `authorId` et `authorName` : Auteur du commentaire
- `content` : Texte du commentaire
- `createdAt` : Date de publication

Index composite : `photoId` + `createdAt` pour récupérer les commentaires d'une photo

5.5 Collection `notifications`

Historique des notifications :

- `id` : Identifiant unique
- `userId` : Destinataire de la notification
- `type` : "like" ou "comment"
- `title` et `message` : Contenu de la notification
- `photoId` : Photo concernée
- `fromUserId` et `fromUserName` : Utilisateur ayant déclenché la notification
- `read` : Statut de lecture (boolean)
- `createdAt` : Date de création

Règles de sécurité : Chaque utilisateur ne peut lire que ses propres notifications.

---

6. CLOUD FUNCTIONS ET LOGIQUE SERVEUR

J'ai déployé quatre Cloud Functions (2nd Generation, Node.js 20) sur Firebase pour automatiser des processus :

6.1 sendLikeNotification

Trigger : `onCreate` sur la sous-collection `photos/{photoId}/likes/{likeId}` **Logique :**

1. Récupère le document de la photo
2. Vérifie que l'auteur du like n'est pas l'auteur de la photo
3. Récupère le document de l'auteur de la photo pour obtenir son token FCM
4. Envoie une notification push via FCM avec le message personnalisé
5. Crée un document dans la collection `notifications` pour l'historique

6.2 sendCommentNotification

Trigger : `onCreate` sur `comments/{commentId}` **Logique :** Similaire à sendLikeNotification mais pour les commentaires

6.3 sendNotificationOnCreate

Trigger : `onCreate` sur `notifications/{notifId}` **Logique :** Fonction générique qui envoie le push lorsqu'une notification est créée manuellement dans Firestore

6.4 cleanupOldNotifications

Trigger : Cloud Scheduler (cron quotidien à minuit) **Logique :**

1. Calcule la date d'il y a 30 jours
2. Requête Firestore pour trouver toutes les notifications plus anciennes
3. Suppression par batch (max 500 par exécution)
4. Log des résultats

Déploiement : Les fonctions sont déployées avec `firebase deploy --only functions` et tournent dans la région us-central1.

---

8. PERSPECTIVES D'AMÉLIORATION

Bien que l'application soit fonctionnelle et réponde au cahier des charges, plusieurs améliorations pourraient être apportées :

8.1 Mode Hors-Ligne Complet

Actuellement, l'application nécessite une connexion Internet pour fonctionner. L'implémentation du cache Firestore permettrait de consulter les photos déjà chargées hors ligne et de mettre en file d'attente les actions (likes, commentaires) pour synchronisation ultérieure.

8.2 Clustering des Marqueurs sur la Carte

Avec de nombreuses photos dans une même zone géographique, la carte devient surchargée. Un système de clustering regrouperait les marqueurs proches et afficherait un nombre, s'ouvrant en marqueurs individuels lors du zoom.

8.3 Messagerie Directe

Permettre aux utilisateurs de s'envoyer des messages privés.

8.4 Système de Modération Avancé

Actuellement, les signalements incrémentent simplement un compteur. Un tableau de bord administrateur permettrait de gérer les contenus signalés et de bannir les utilisateurs problématiques.

8.5 Recommandations Personnalisées

Un algorithme pourrait analyser les types de lieux préférés de l'utilisateur (basé sur ses likes et publications) et recommander des photos similaires dans le flux d'accueil.

---

9. CONCLUSION

Le développement de TravelShare a été un projet enrichissant qui m'a permis de maîtriser l'ensemble de l'écosystème Firebase (Authentication, Firestore, Storage, Cloud Functions, Cloud Messaging) ainsi que les APIs Google Maps.

L'intégration avec TravelPaths via Intents Android a également été instructive, montrant comment des applications indépendantes peuvent collaborer .