

**ĐẠI HỌC PHENIKAA
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN MÔN HỌC
LẬP TRÌNH CHO THIẾT BỊ DI ĐỘNG**

**THIẾT KẾ ỨNG DỤNG QUẢN LÝ
ẢNH DU LỊCH**

Giảng viên hướng dẫn: THS. NGUYỄN VĂN CƯỜNG

Sinh viên thực hiện: Phạm Quang Minh (23010489)

Nguyễn Văn Quang (23011955)

Lớp: N06

Khoa: K17

Ngành/ chuyên ngành: Công nghệ thông tin

Hà nội, năm 2025

**ĐẠI HỌC PHENIKAA
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN MÔN HỌC
LẬP TRÌNH CHO THIẾT BỊ DI ĐỘNG**

**THIẾT KẾ ỨNG DỤNG QUẢN LÝ
ẢNH DU LỊCH**

Giảng viên hướng dẫn: THS. NGUYỄN VĂN CƯỜNG

Sinh viên thực hiện: Phạm Quang Minh (23010489)

Nguyễn Văn Quang (23011955)

Lớp: N06

Khoa: K17

Ngành/ chuyên ngành: Công nghệ thông tin

Hà nội, năm 2025

LỜI CẢM ƠN

Chúng tôi xin gửi lời cảm ơn chân thành và sâu sắc nhất đến ThS. Nguyễn Văn Cường – giảng viên hướng dẫn đã tận tình giảng dạy, hướng dẫn và giúp đỡ chúng tôi trong suốt quá trình thực hiện đồ án môn học “Lập trình ứng dụng di động”.

Nhờ có sự chỉ bảo tận tâm, những góp ý quý báu và định hướng đúng đắn của Thầy, nhóm chúng tôi đã có cơ hội học hỏi, trau dồi kiến thức, kỹ năng thực hành, cũng như hiểu sâu hơn về quy trình phát triển một ứng dụng thực tế.

Cuối cùng, xin chân thành cảm ơn các thành viên trong nhóm đã cùng nhau hợp tác, nỗ lực và hỗ trợ lẫn nhau để hoàn thiện sản phẩm đúng tiến độ và đạt được kết quả tốt nhất.

Xin trân trọng cảm ơn!

LỜI CAM ĐOAN

Chúng tôi cam đoan Đồ án môn học là sản phẩm trí tuệ của tập thể chúng tôi. Mọi thông tin, dữ liệu, hình ảnh, etc. được sử dụng từ các nguồn khác đều được trích dẫn đầy đủ và có thể tìm thấy các tài liệu liên quan thông qua mục tài liệu tham khảo.

Chúng tôi xin chịu trách nhiệm hoàn toàn về nội dung của Đồ án môn học mà chúng tôi đã nộp.

Hà nội, ngày tháng năm

Phạm Quang Minh, Nguyễn Văn Quang (ký tên)

PHÂN CÔNG NHIỆM VỤ ĐO ÁN

Danh sách các công việc/nhiệm vụ	Mô tả tóm tắt công việc
Công việc 1	Thu thập dữ liệu và khảo sát yêu cầu
Công việc 2	Phân tích và thiết kế
Công việc 3	Thiết kế giao diện người dùng
Công việc 4	Triển khai giải pháp và kĩ thuật
Công việc 5	Lập trình và phát triển ứng dụng
Công việc 6	Kiểm thử và sửa lỗi
Công việc 7	Viết báo cáo và chuẩn bị thuyết trình
Công việc 8	Tổng kết

Tên sinh viên/Mã sinh viên	Các công việc	Tỉ lệ
Phạm Quang Minh 23010489	Công việc 1	40%
	Công việc 2	50%
	Công việc 3	50%
	Công việc 4	50%
	Công việc 5	60%
	Công việc 6	40%
	Công việc 7	50%
	Công việc 8	100%
Nguyễn Văn Quang 23011955	Công việc 1	60%
	Công việc 2	50%
	Công việc 3	50%
	Công việc 4	50%
	Công việc 5	40%
	Công việc 6	60%
	Công việc 7	50%

TRƯỜNG ĐẠI HỌC PHENIKAA
KHOA CÔNG NGHỆ THÔNG TIN

**KỲ THI KẾT THÚC HỌC PHẦN
HỌC KỲ NĂM HỌC -**

PHIẾU CHẤM THI TIÊU LUẬN/ĐỒ ÁN

Môn học:

Lớp học phần:

Sinh viên thực hiện:

1.

Điểm:

2.

Điểm:

3

Điểm:

4

Điểm:

5

Điểm:

Ngày thi:

Phòng

thi:

Giảng viên chấm thi 1 (Ký và ghi rõ họ tên)

Giảng viên chấm thi 2

(Ký và ghi rõ họ tên)

MỤC LỤC

LỜI CAM ĐOAN.....	3
PHÂN CÔNG NHIỆM VỤ ĐỒ ÁN.....	4
MỤC LỤC.....	6
DANH MỤC TỪ VIẾT TẮT.....	7
DANH MỤC HÌNH VẼ.....	8
CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI VÀ CƠ SỞ LÝ THUYẾT.....	9
1.1. Giới thiệu ứng dụng.....	9
Link github dự án: https://github.com/miin000/sketch_travel_flutter	9
Link video demo: https://youtu.be/sloEyhPcLKA?si=t8dAiUOM8v7ktpCP	9
1.2. Lý do chọn đề tài.....	9
1.3. Mục tiêu và phạm vi của ứng dụng.....	10
1.4. Công nghệ và công cụ sử dụng (Flutter, Dart, Firebase, GetX, v.v.).....	11
1.5. Cơ sở lý thuyết về Flutter và kiến trúc MVC/MVVM.....	12
CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ ỨNG DỤNG.....	14
2.1. Phân tích yêu cầu hệ thống.....	14
2.2. Thiết kế tổng thể hệ thống.....	15
2.3. Thiết kế giao diện người dùng.....	25
2.4. Thiết kế cơ sở dữ liệu (Firebase).....	29
CHƯƠNG 3. TRIỂN KHAI ỨNG DỤNG.....	33
3.1. Cài đặt môi trường phát triển (Flutter SDK, Android Studio).....	33
3.2. Cấu trúc thư mục và module trong dự án.....	33
3.3. Triển khai các chức năng chính.....	34
3.3.1. Đăng ký, đăng nhập (Authentication).....	34
3.3.2. Đăng bài viết mới (kèm gắn thẻ Địa điểm).....	38
3.3.3. Chỉnh sửa, xóa bài viết (và Quản lý ảnh).....	42
3.3.5. Quản lý người dùng (Follow, Friend).....	47
3.3.6. Quản lý dữ liệu trên Firebase (Denormalization).....	52
3.3.7. Tải ảnh, xử lý upload bắt đồng bộ.....	56
3.3.8. Điều hướng với GetX và quản lý state.....	57
3.4. Kết quả thực nghiệm.....	59
3.4.1. Các giao diện.....	59
3.4.2. Video Demo.....	79
3.4.3. Đánh giá hiệu năng và trải nghiệm người dùng.....	79
CHƯƠNG 4. ĐÁNH GIÁ VÀ HƯỚNG PHÁT TRIỂN.....	80
4.1. Các kết quả đạt được.....	80
4.2. Hạn chế của ứng dụng.....	81
4.3. Định hướng phát triển trong tương lai.....	81
KẾT LUẬN CHUNG.....	83

DANH MỤC TỪ VIẾT TẮT

Chữ viết tắt	Giải thích
WWW	World wide web
CNTT	Công nghệ thông tin

DANH MỤC HÌNH VẼ

1.1.1.1. Biểu đồ use case tổng quát.....	17
1.1.1.2. Biểu đồ activity diagram đăng bài viết mới.....	19
1.1.1.3. Biểu đồ activity diagram theo dõi và kết bạn.....	21
1.1.1.4. Sơ đồ tuần tự thích một bài viết.....	22
1.1.1.5. Sơ đồ tuần tự tìm kiếm và chọn địa điểm.....	23
1.1.1.6. Sơ đồ tuần tự bình luận một bài viết.....	25
1.1.1.7. Giao diện login.....	35
1.1.1.8. Giao diện register.....	36
1.1.1.9. Giao diện tạo bài viết mới.....	40
1.1.1.10. Giao diện tạo chọn địa điểm.....	41
1.1.1.11. Giao diện chỉnh sửa bài viết.....	45
1.1.1.12. Giao diện tạo xóa bài viết.....	46
1.1.1.13. Giao diện ở profile người khác khi chưa follow.....	49
1.1.1.14. Giao diện ở profile người khác khi đã follow.....	51
1.1.1.15. Firestore Database: post và comment của post đó	53
1.1.1.16. Giao diện xác nhận có comment trong database.....	55
1.1.1.17. Giao diện trang chủ.....	61
1.1.1.18. Giao diện tìm kiếm.....	64
1.1.1.19. Giao diện bạn bè.....	66
1.1.1.20. Giao diện thông báo.....	68
1.1.1.21. Giao diện profile.....	72
1.1.1.22. Giao diện setting.....	72
1.1.1.23. Giao diện comment.....	74
1.1.1.24. Giao diện chat.....	76
1.1.1.25. Giao diện xem địa điểm.....	79

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI VÀ CƠ SỞ LÝ THUYẾT

1.1. Giới thiệu ứng dụng

"Sketch Travel" là một ứng dụng mạng xã hội di động được xây dựng trên nền tảng Flutter, chuyên biệt cho cộng đồng yêu thích du lịch.

Điểm cốt lõi của ứng dụng là cung cấp một trải nghiệm khám phá nội dung trực quan, tương tự như mô hình của TikTok. Màn hình chính sử dụng cơ chế cuộn dọc cho phép người dùng lướt qua các bài đăng (dạng hình ảnh/video) một cách liền mạch. Mỗi bài đăng lại có thể chứa nhiều ảnh, cho phép người dùng cuộn ngang để xem.

Ứng dụng cho phép người dùng đăng tải các khoảnh khắc du lịch, viết mô tả, và quan trọng nhất là **gắn thẻ (tag) địa điểm**. Tính năng này biến "Sketch Travel" từ một mạng xã hội thông thường thành một cơ sở dữ liệu du lịch có cấu trúc, nơi người dùng có thể khám phá, đánh giá, và xem các bài đăng liên quan đến một địa điểm cụ thể.

Ngoài ra, ứng dụng được trang bị đầy đủ các tính năng xã hội như theo dõi, hệ thống bạn bè (tự động kết bạn khi theo dõi lẫn nhau), nhắn tin 1-1, và một hệ thống thông báo (Hòm thư) mạnh mẽ.

Link github dự án: https://github.com/miin000/sketch_travel_flutter

Link video demo: <https://youtu.be/sloEyhPcLKA?si=t8dAiUOM8v7ktpCP>

1.2. Lý do chọn đề tài

Việc lựa chọn phát triển ứng dụng "Sketch Travel" xuất phát từ các quan sát thực tế sau:

- Sự thay đổi trong hành vi tiêu thụ nội dung:** Người dùng, đặc biệt là thế hệ trẻ, ngày càng ưa chuộng các nội dung trực quan, nhanh, và có tính tương tác cao (như TikTok, Reels) hơn là các bài blog du lịch dài và nặng về văn bản.
- Sự "loãng" của các mạng xã hội chung:** Mặc dù Instagram hay TikTok có nhiều nội dung du lịch, chúng không được tối ưu hóa cho việc tra cứu. Rất khó để tìm tất cả thông tin, bài đăng, và đánh giá *chỉ* về một địa điểm cụ thể một cách có tổ chức.
- Thiếu hụt tính năng trên ứng dụng du lịch truyền thống:** Các ứng dụng như TripAdvisor mạnh về đánh giá nhưng lại thiếu tính "mạng xã hội", thiếu sự tương tác và chia sẻ khoảnh khắc tức thời.

"Sketch Travel" ra đời để giải quyết các vấn đề trên, bằng cách:

- **Kết hợp** trải nghiệm cuộn feed "gây nghiện" của TikTok.
- **Tập trung** 100% vào ngách (niche) du lịch.
- **Xây dựng** một cấu trúc dữ liệu xoay quanh "Địa điểm" (Location-Centric), cho phép người dùng không chỉ xem bài đăng mà còn đánh giá địa điểm, xem ai đã check-in, và khám phá các trải nghiệm liên quan.

1.3. Mục tiêu và phạm vi của ứng dụng

Mục tiêu

- Xây dựng một nền tảng mạng xã hội di động (cross-platform) cho người yêu du lịch.
- Cung cấp trải nghiệm người dùng hiện đại, trực quan (TikTok-style feed).
- Xây dựng cơ sở dữ liệu về các địa điểm du lịch, được đóng góp bởi chính người dùng (đánh giá, bài đăng).
- Tạo một cộng đồng có tính kết nối (follow, bạn bè, chat, thông báo).

Phạm vi (Dựa trên code)

Ứng dụng bao gồm các nhóm chức năng chính sau:

- **Xác thực người dùng:** Đăng ký (với Tên, Email, Mật khẩu, Avatar), Đăng nhập, Đăng xuất (sử dụng Firebase Auth).
- **Quản lý Bài đăng (Post):**
 - Tạo bài đăng mới (hỗ trợ nhiều ảnh, mô tả).
 - Chỉnh sửa bài đăng đã có.
 - Gắn thẻ địa điểm.
 - Xóa bài đăng.
- **Feed và Tương tác:**
 - Feed chính cuộn dọc.
 - Feed ảnh cuộn ngang (trong 1 bài đăng).
 - Thích/Bỏ thích bài đăng.
 - Bình luận bài đăng.
- **Tính năng Xã hội (Social):**
 - Theo dõi (Follow) và Bỏ theo dõi (Unfollow) người dùng khác.
 - Hệ thống Bạn bè (Friends): Tự động trở thành bạn bè khi 2 người dùng theo dõi lẫn nhau.
 - Xem hồ sơ (Profile) của người dùng khác.
- **Tính năng Địa điểm:**
 - Tìm kiếm địa điểm (sử dụng API OpenStreetMap Nominatim).
 - Xem chi tiết địa điểm.
 - Đăng đánh giá (sao + nội dung) cho địa điểm.
 - Xem tất cả bài đăng đã được gắn thẻ tại địa điểm đó.
 - Lưu địa điểm vào danh sách Yêu thích.
- **Hòm thư (Mailbox):**

- **Tab Thông báo:** Nhận thông báo real-time cho các hành động: có người Follow, Like, Comment, hoặc gửi Tin nhắn mới.
- **Tab Tin nhắn:** Liệt kê danh sách "Bạn bè" (mutual followers) và cho phép chat 1-1.
- **Trang cá nhân (Profile):**
 - **Tab Các Bài Đã Đăng:** Xem, sửa, xóa được các bài viết đã đăng lên.
 - **Tab Yêu Thích Bài Viết:** Xem được các bài viết đã thích.
 - **Tab Yêu Thích Địa Điểm:** Xem được các địa điểm đã yêu thích.
 - **Cài đặt:** Chuyển đổi chế độ Sáng/Tối (Dark/Light Mode) và đăng xuất tài khoản.

1.4. Công nghệ và công cụ sử dụng (Flutter, Dart, Firebase, GetX, v.v.)

Dự án được xây dựng bằng các công nghệ và công cụ hiện đại:

- **Ngôn ngữ lập trình: Dart**
- **Framework: Flutter** (cho phép xây dựng ứng dụng cross-platform Android và iOS từ một cơ sở code duy nhất).
- **Backend (BaaS): Firebase**
 - **Firebase Authentication:** Xử lý toàn bộ luồng đăng ký, đăng nhập.
 - **Cloud Firestore:** Sử dụng làm cơ sở dữ liệu NoSQL chính. Các collections chính bao gồm: users, posts, locations, chatRooms, notifications, favoriteLocations.
- **Lưu trữ Ánh (Storage): Cloudinary** (Thay vì Firebase Storage). Toàn bộ ảnh đại diện và ảnh bài đăng đều được upload và quản lý qua CloudinaryController.
- **Quản lý State & Kiến trúc: GetX**
 - Sử dụng cho cả ba mục đích: Quản lý State (với Rx và Obx), Quản lý Phụ thuộc (Dependency Injection với Get.put), và Điều hướng (Navigation với Get.to, Get.back).
- **API bên ngoài: OpenStreetMap (OSM) Nominatim API** (Được dùng trong LocationSearchController để tìm kiếm và lấy dữ liệu địa điểm).
- **Lưu trữ cục bộ: GetStorage** (Dùng để lưu trạng thái Sáng/Tối của ứng dụng).
- **Các thư viện (Packages) chính:**
 - image_picker: Chọn ảnh từ thư viện/camera.
 - cached_network_image: Tối ưu hiển thị và cache ảnh từ URL (Cloudinary).
 - smooth_page_indicator: Hiển thị dấu chấm tròn khi cuộn ảnh.
 - flutter_rating_bar: Hiển thị và chọn đánh giá sao (trong LocationScreen).
 - timeago: Định dạng thời gian (ví dụ: "5 phút trước" trong màn hình bình luận).

1.5. Cơ sở lý thuyết về Flutter và kiến trúc MVC/MVVM

Flutter

Flutter là một bộ công cụ phát triển giao diện người dùng (UI SDK) mã nguồn mở do Google tạo ra. Nó được sử dụng để phát triển các ứng dụng đa nền tảng (cho di động, web, và máy tính) từ một cơ sở mã nguồn duy nhất. Flutter sử dụng ngôn ngữ lập trình Dart và nổi bật với các tính năng như:

- **Hot Reload:** Giúp lập trình viên thấy thay đổi trong code gần như ngay lập tức.
- **Kiến trúc Widget:** Mọi thứ trong Flutter là một Widget. Giao diện được xây dựng bằng cách lồng ghép các widget (Widget Tree).
- **Hiệu suất cao:** Flutter biên dịch trực tiếp ra mã máy (ARM/x86), mang lại hiệu suất gần như ứng dụng gốc (native).

Kiến trúc ứng dụng (GetX và MVVM)

Dự án không sử dụng kiến trúc MVC (Model-View-Controller) hay MVVM (Model-View-ViewModel) truyền thống một cách cứng nhắc. Thay vào đó, dự án áp dụng kiến trúc do **GetX** cung cấp, một kiến trúc có nhiều điểm tương đồng và được truyền cảm hứng mạnh mẽ từ **MVVM**.

Kiến trúc này được chia làm 3 phần rõ rệt trong code:

1. **Model:** Định nghĩa cấu trúc dữ liệu (plain-old Dart objects - PODOs).
 - *Ví dụ trong code:* Các file trong thư mục models/ như post.dart, user.dart, location.dart, comment.dart.
2. **View (Giao diện):** Là các file Widget (...Screen.dart), chịu trách nhiệm duy nhất là hiển thị giao diện người dùng.
 - Các View này sử dụng Widget Obx() để "lắng nghe" và tự động cập nhật khi trạng thái (state) thay đổi.
 - *Ví dụ trong code:* post_feed_screen.dart, profile_screen.dart, login_screen.dart.
3. **ViewModel (GetX Controller):** Đây là "bộ não" của ứng dụng, tương ứng với vai trò ViewModel trong MVVM.
 - Chúng chứa toàn bộ logic nghiệp vụ (business logic), xử lý sự kiện (nhấn nút), và quản lý trạng thái của View.
 - Trạng thái được khai báo bằng các biến .obs (ví dụ: final Rx<List<Post>> _postList = Rx<List<Post>>([])).
 - Chúng giao tiếp trực tiếp với Firebase (Firestore) để đọc/ghi dữ liệu.
 - *Ví dụ trong code:* Các file trong thư mục controllers/ như post_controller.dart, profile_controller.dart, auth_controller.dart.

Luồng hoạt động: (1) **View** (ví dụ: PostFeedScreen) gọi một hàm từ **Controller** (ví dụ: postController.likePost(postId)). (2) **Controller** thực thi logic (ví dụ: gọi

Firebase) và cập nhật biến trạng thái .obs. (3) Widget Obx() trong **View** tự động phát hiện sự thay đổi và build lại *chỉ* phần giao diện cần thiết, mà không cần gọi setState().

CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ ỨNG DỤNG

2.1. Phân tích yêu cầu hệ thống

2.1.1. Đối tượng người dùng

1. Khách (Guest):

- Là người dùng chưa có tài khoản hoặc chưa đăng nhập.
- Bị giới hạn quyền truy cập, chỉ có thể tương tác với các màn hình LoginScreen và SignupScreen.
- Mục tiêu: Đăng nhập hoặc tạo tài khoản mới.

2. Người dùng đã xác thực (Registered User):

- Là người dùng đã đăng nhập thành công vào hệ thống.
- Có toàn quyền truy cập các tính năng cốt lõi của ứng dụng.
- Mục tiêu: Chia sẻ khoảnh khắc (Đăng bài), tương tác xã hội (like, comment, follow, chat), và khám phá/dánh giá các địa điểm du lịch.

2.1.2. Chức năng tổng quát của ứng dụng

• UC-1: Quản lý Xác thực (Authentication)

- **Chức năng:** Đăng ký tài khoản mới bằng Tên, Email, Mật khẩu vàẢnh đại diện(auth_controller.dart).
- **Chức năng:** Đăng nhập bằng Email và Mật khẩu.
- **Chức năng:** Đăng xuất khỏi hệ thống.

• UC-2: Quản lý Bài đăng (Post)

- **Chức năng:** Tạo bài đăng mới, cho phép đính kèm nhiều ảnh, mô tả, và gắn thẻ (tag) một địa điểm(upload_post_controller.dart).
- **Chức năng:** Chỉnh sửa bài đăng đã có (mô tả, ảnh, địa điểm).
- **Chức năng:** Xóa bài đăng (chỉ chủ sở hữu mới có quyền,profile_controller.dart).

• UC-3: Tương tác Xã hội (Social Interaction)

- **Chức năng:** Thích (Like) và Bỏ thích (Unlike) một bài đăng(post_controller.dart).
- **Chức năng:** Bình luận (Comment) một bài đăng(comment_controller.dart).
- **Chức năng:** Theo dõi (Follow) và Bỏ theo dõi (Unfollow) người dùng khác(profile_controller.dart).

• UC-4: Khám phá Địa điểm (Location-Centric)

- **Chức năng:** Xem chi tiết một địa điểm(location_screen.dart).
- **Chức năng:** Gửi đánh giá (Rating và Review) cho một địa điểm(location_controller.dart).
- **Chức năng:** Lưu (Favorite) và Bỏ lưu một địa điểm(location_favorite_controller.dart).

• UC-5: Quản lý Hồ sơ cá nhân (Profile)

- **Chức năng:** Xem hồ sơ cá nhân của mình hoặc của người khác.

- **Chức năng:** Chỉnh sửa tên hiển thị (profile_controller.dart).
 - **Chức năng:** Xem 3 tab: Bài đã đăng, Bài đã thích, Địa điểm đã lưu.
- **UC-6: Nhắn tin và Thông báo (Mailbox)**
 - **Chức năng:** Gửi và nhận tin nhắn 1-1 với "Bạn bè" (người dùng theo dõi lẫn nhau) (chat_detail_controller.dart).
 - **Chức năng:** Nhận thông báo (Notification) khi có người Like, Comment, Follow, hoặc gửi Tin nhắn mới (notification_controller.dart).
- **UC-7: Tìm kiếm (Search)**
 - **Chức năng:** Tìm kiếm địa điểm dựa trên tên (sử dụng API OpenStreetMap) (location_search_controller.dart).
 - **Chức năng:** Tìm kiếm người dùng khác dựa trên tên hoặc username (sử dụng searchKeywords trong Firestore) (user_search_controller.dart).

2.1.3. Các yêu cầu phi chức năng

- **Hiệu suất (Performance):** Ứng dụng sử dụng thư viện cached_network_image để tải và lưu cache ảnh từ Cloudinary, giảm thời gian tải khi xem lại. Các truy vấn được giới hạn (ví dụ limit(50) trong NotificationController) để tránh tải quá nhiều dữ liệu cùng lúc.
- **Tính khả dụng (Usability):** Ứng dụng hỗ trợ 2 chế độ hiển thị Sáng và Tối (Dark/Light Mode), được quản lý bởi ThemeController và lưu trạng thái bằng GetStorage. Giao diện được thiết kế theo tab (ở màn hình chính, profile, mailbox) giúp người dùng dễ dàng điều hướng.
- **Bảo mật (Security):** Việc xác thực người dùng được xử lý hoàn toàn bởi **Firebase Authentication**. Các API key (Firebase, Cloudinary) được lưu trong code (ví dụ firebase_options.dart, cloudinary_controller.dart) để kết nối tới dịch vụ.
- **Tính tương thích (Compatibility):** Ứng dụng được xây dựng bằng Flutter, cho phép biên dịch và chạy trên cả hai nền tảng HĐH di động phổ biến là Android và iOS từ cùng một cơ sở code.
- **Khả năng bảo trì (Maintainability):** Code được tổ chức theo kiến trúc MVVM (Model-View-ViewModel) sử dụng GetX. Logic nghiệp vụ (ví dụ PostController) được tách biệt hoàn toàn khỏi Giao diện (ví dụ PostFeedScreen), giúp việc sửa lỗi và nâng cấp tính năng dễ dàng hơn.

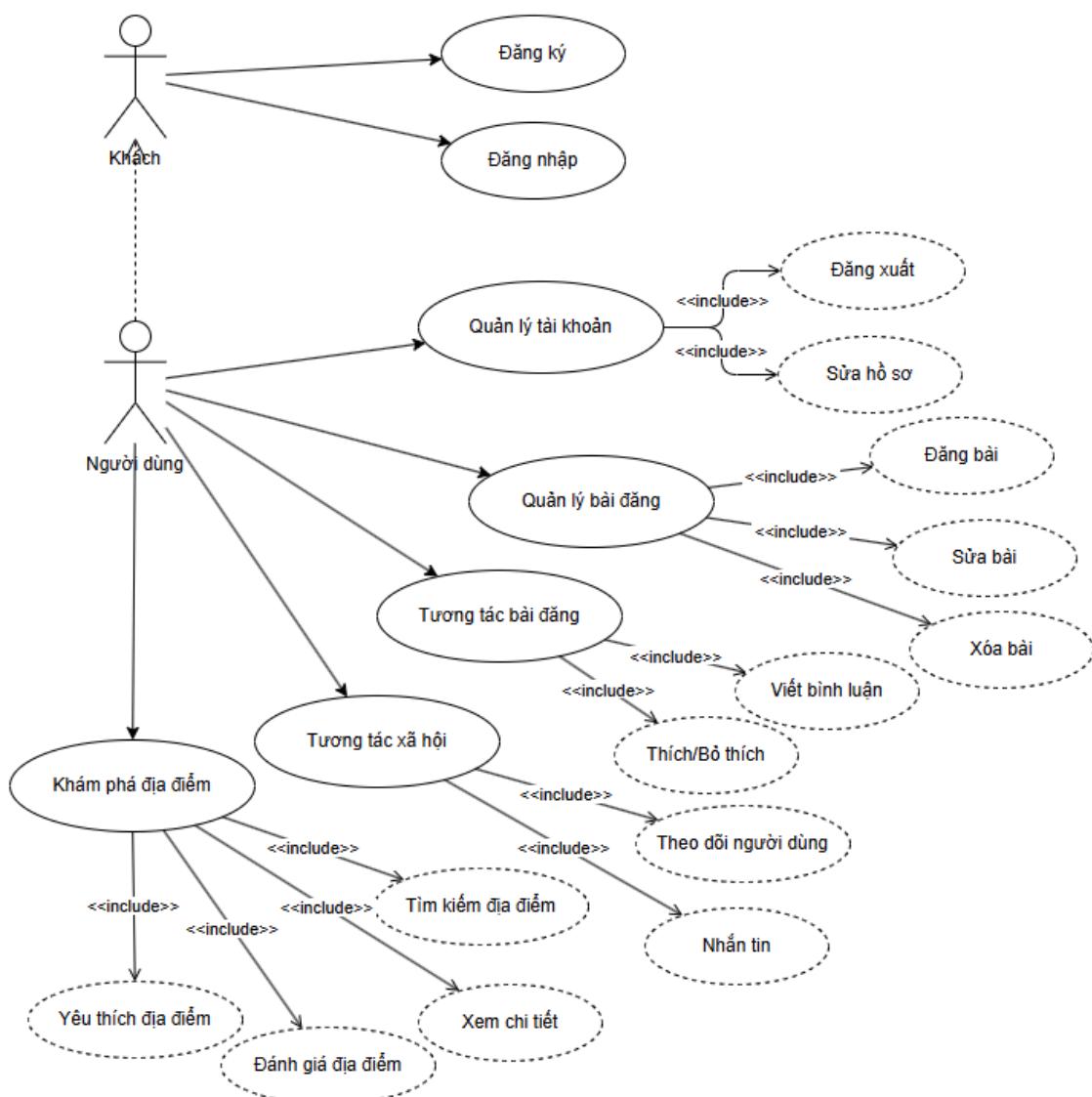
2.2. Thiết kế tổng thể hệ thống

2.2.1. Biểu đồ use case tổng quát

Sơ đồ thể hiện các hành động chính của hai đối tượng: "Khách" và "Người dùng".

- **Actors:**

- Khách (Guest)
- Người dùng (Registered User) - kế thừa từ Khách
- Use Cases:
 - Đăng ký (Guest)
 - Đăng nhập (Guest)
 - Quản lý tài khoản (User)
 - Includes: Đăng xuất, Sửa hồ sơ.
 - Quản lý bài đăng (User)
 - Includes: Đăng bài, Sửa bài, Xóa bài.
 - Tương tác bài đăng (User)
 - Includes: Thích/Bỏ thích, Viết bình luận.
 - Tương tác xã hội (User)
 - Includes: Theo dõi người dùng, Nhắn tin (với bạn bè).
 - Khám phá địa điểm (User)
 - Includes: Tìm kiếm địa điểm, Xem chi tiết, Đánh giá địa điểm, Yêu thích địa điểm.

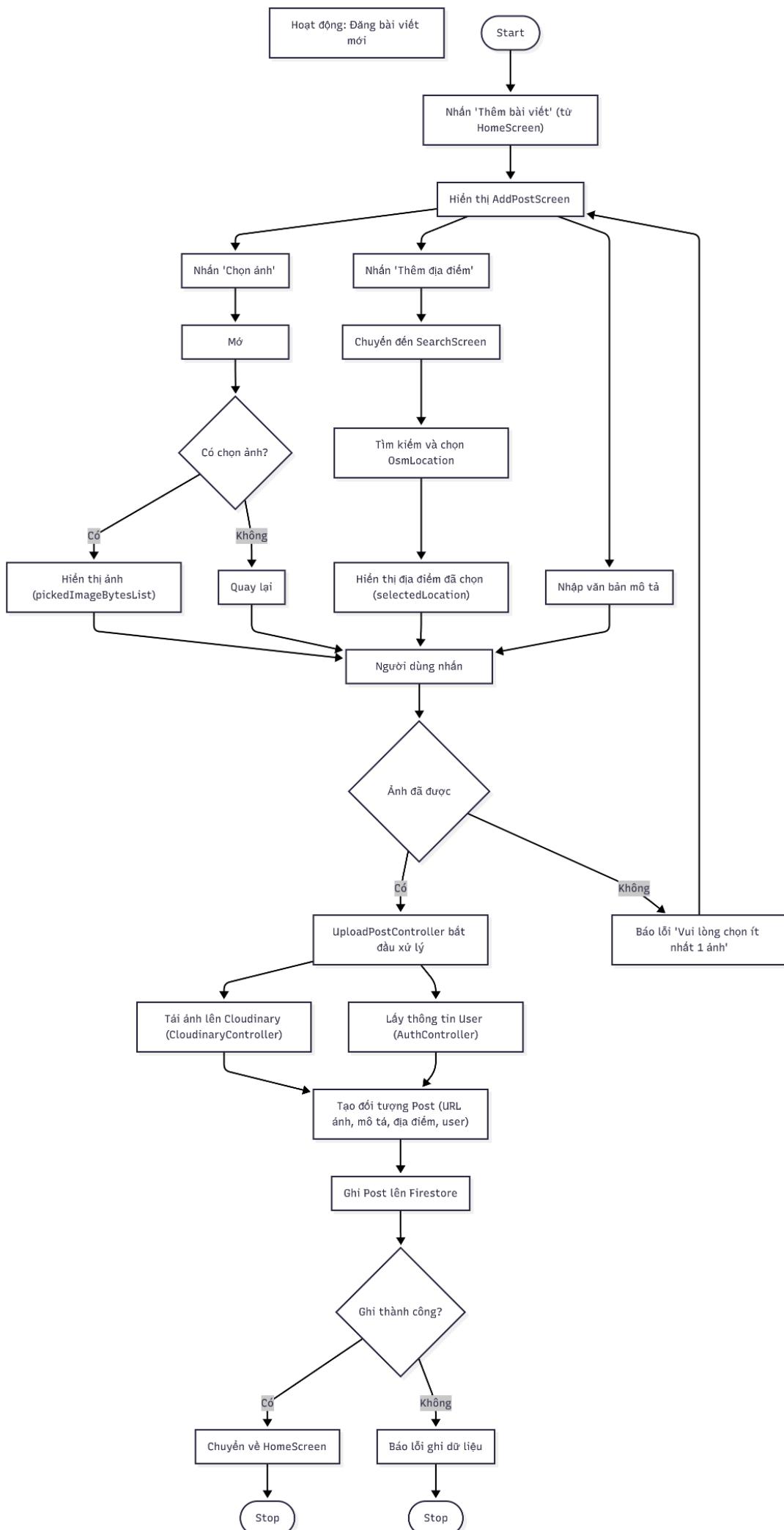


1.1.1.1. Biểu đồ use case tổng quát

2.2.2. Biểu đồ hoạt động (Activity Diagram)

1. Hoạt động: Đăng bài viết mới (Dựa trên add_post_screen.dart và upload_post_controller.dart)

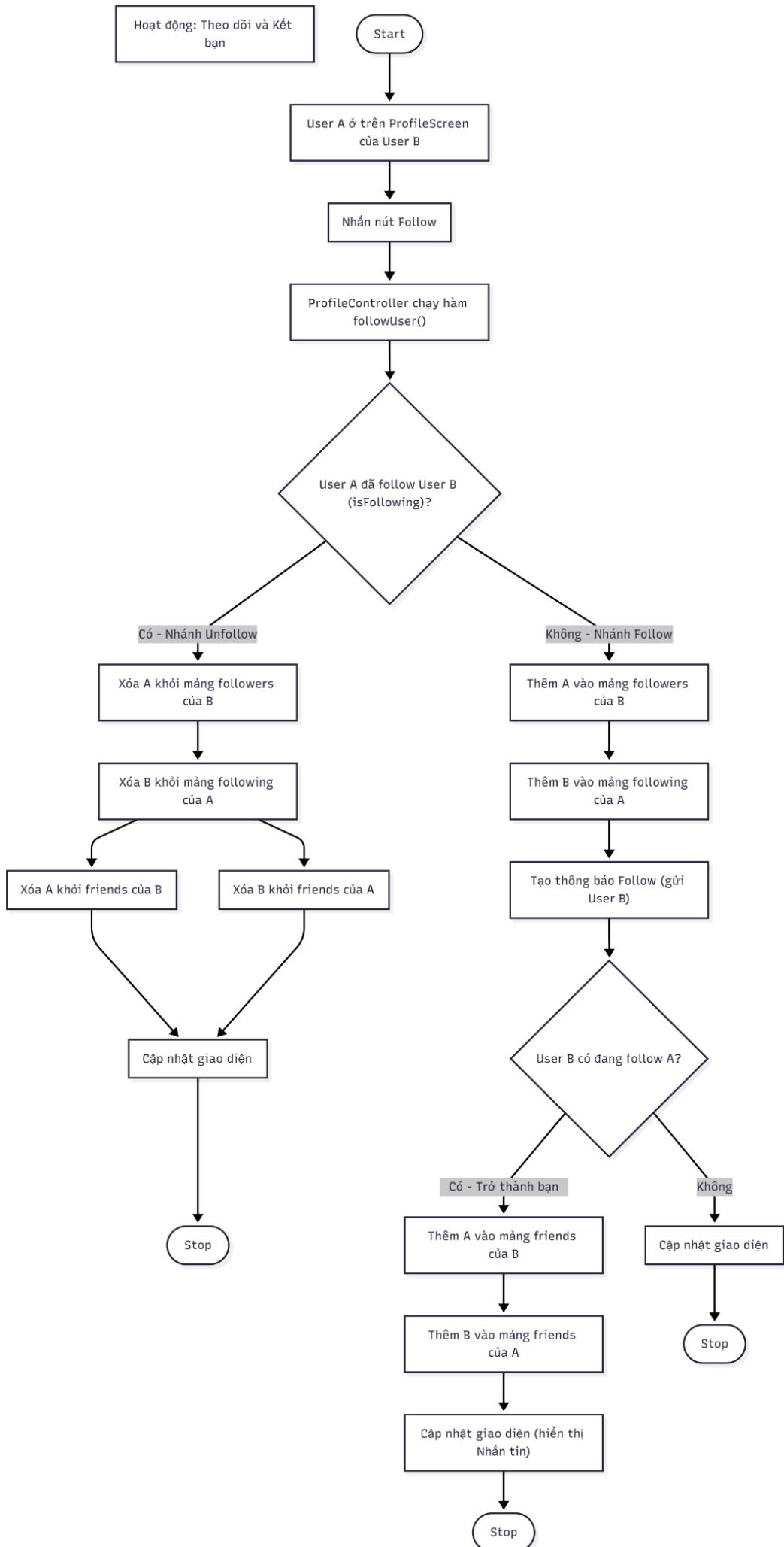
1. Bắt đầu từ HomeScreen.
2. Người dùng nhấn "Thêm bài viết".
3. Chuyển đến AddPostScreen.
4. Hệ thống hiển thị song song 3 lựa chọn: [Chọn ảnh], [Thêm địa điểm], [Nhập mô tả].
5. **Luồng Chọn ảnh:** Người dùng nhấn "Chọn ảnh" -> Mở ImagePicker -> (Rẽ nhánh) Nếu không chọn ảnh -> Quay lại. / Nếu có chọn ảnh -> Hiển thị ảnh (pickedImageBytesList).
6. **Luồng Thêm địa điểm:** Người dùng nhấn "Thêm địa điểm" -> Chuyển đến SearchScreen -> Tìm kiếm và chọn 1 OsmLocation -> Quay lại AddPostScreen (selectedLocation).
7. **Luồng Nhập mô tả:** Người dùng nhập văn bản vào TextField.
8. (Nối) Khi người dùng nhấn "Đăng".
9. (Rẽ nhánh) Hệ thống kiểm tra: Ảnh đã được chọn chưa?
10. (Không) Báo lỗi "Vui lòng chọn ít nhất 1 ảnh".
11. (Có) UploadPostController bắt đầu xử lý:
12. (Hoạt động song song - Fork)
 - o [1] Tải ảnh (bytes) lên Cloudinary (CloudinaryController).
 - o [2] Lấy thông tin người dùng hiện tại (username, avatar) từ AuthController.
13. (Nối - Join) Chờ (1) và (2) hoàn thành.
14. Hệ thống tạo đối tượng Post (gồm URL ảnh từ Cloudinary, mô tả, tên địa điểm, thông tin user).
15. Ghi đối tượng Post vào collection posts trên Firestore.
16. (Rẽ nhánh) Ghi thành công?
17. (Có) Chuyển về HomeScreen và kết thúc.
18. (Không) Báo lỗi và kết thúc.



1.1.1.2. Biểu đồ activity diagram đăng bài viết mới

2. Hoạt động: Theo dõi và Kết bạn (Dựa trên profile_controller.dart - followUser())

1. Bắt đầu khi User A ở trên ProfileScreen của User B.
2. User A nhấn nút "Follow".
3. ProfileController chạy hàm followUser().
4. (Rẽ nhánh) User A đã follow User B chưa (isFollowing == true)?
 5. **(Nhánh Unfollow - Có)**
 - Xóa UID của A khỏi mảng followers của B.
 - Xóa UID của B khỏi mảng following của A.
 - (Hoạt động song song) Xóa A khỏi friends của B và B khỏi friends của A.
 - Cập nhật giao diện.
 - Kết thúc.
 6. **(Nhánh Follow - Không)**
 - Thêm UID của A vào mảng followers của B.
 - Thêm UID của B vào mảng following của A.
 - Tạo thông báo "Follow" gửi đến User B (_createFollowNotification).
 - (Rẽ nhánh) User B có đang follow User A không?
 - **(Không)** Cập nhật giao diện. Kết thúc.
 - **(Có - Trở thành bạn bè)**
 - Thêm UID của A vào mảng friends của B.
 - Thêm UID của B vào mảng friends của A.
 - Cập nhật giao diện (hiển thị nút "Nhắn tin").
 - Kết thúc.



1.1.1.3. Biểu đồ activity diagram theo dõi và kết bạn

2.2.3. Biểu đồ tuần tự (Sequence Diagram)

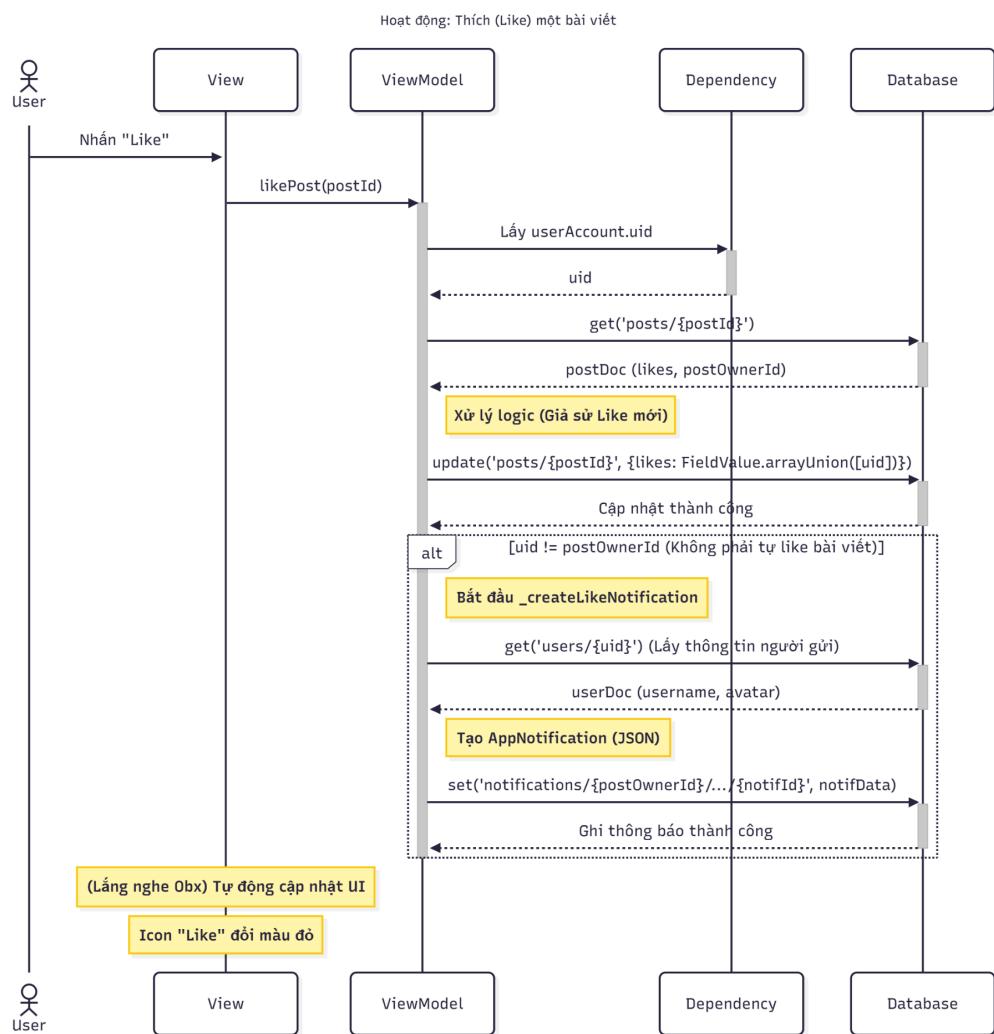
1. Tuần tự: Thích một bài viết

Biểu đồ này mô tả luồng sự kiện khi người dùng nhấn nút "Tim" trên một bài đăng ở màn hình Feed chính.

- **Actors:** User (Người dùng)
- **Lifelines (Đối tượng):**
 - :PostFeedScreen (View)
 - :PostController (ViewModel)
 - :AuthController (Dependency)
 - :Firestore (Database)

Luồng sự kiện:

1. User nhấn vào biểu tượng "Like" trên :PostFeedScreen.
2. :PostFeedScreen gọi :PostController.likePost(postId).
3. :PostController lấy uid người dùng hiện tại từ :AuthController.userAccount.uid.
4. :PostController gọi :Firestore.get('posts/{postId}') để lấy dữ liệu bài đăng (mảng likes và postOwnerId).
5. :Firestore trả về postDoc.
6. :PostController xử lý logic (Giả sử đây là Like mới, uid không có trong mảng likes).
7. :PostController gọi :Firestore.update('posts/{postId}', {likes: FieldValue.arrayUnion([uid])}).
8. **(Rẽ nhánh - alt)** Nếu uid != postOwnerId:
 9. :PostController gọi hàm nội bộ _createLikeNotification(postOwnerId, postId).
 10. :PostController gọi :Firestore.get('users/{uid}') (để lấy thông tin người gửi).
 11. :Firestore trả về userDoc (gồm username, avatarUrl).
 12. :PostController tạo AppNotification (dạng JSON).
 13. :PostController gọi :Firestore.set('notifications/{postOwnerId}/userNotifications/{notifId}', notifData).
14. **(Kết thúc alt)**
15. :PostFeedScreen (đang lắng nghe PostController qua Obx) tự động cập nhật, đổi màu icon "Like" sang màu đỏ.



1.1.1.4. Sơ đồ tuần tự thích một bài viết.

2. Tuần tự: Tìm kiếm và Chọn Địa điểm (Picker Mode)

Biểu đồ này mô tả luồng khi người dùng ở màn hình AddPostScreen nhấn "Thêm địa điểm" để tìm và chọn một địa điểm.

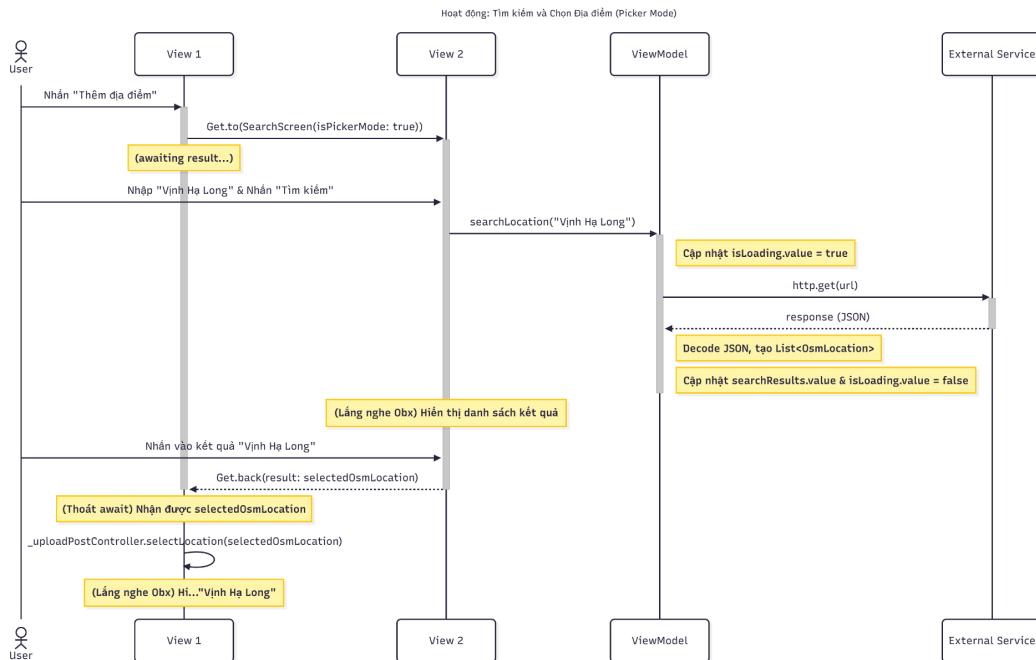
- **Actors:** User (Người dùng)
- **Lifelines (Đối tượng):**
 - :AddPostScreen (View 1)
 - :SearchScreen (View 2 - Picker Mode)
 - :LocationSearchController (ViewModel)
 - :NominatimAPI (External Service - OpenStreetMap)

Luồng sự kiện:

1. User nhấn vào "Thêm địa điểm" trên :AddPostScreen.
2. :AddPostScreen gọi Get.to(() => SearchScreen(isPickerMode: true)) và await kết quả.
3. Hệ thống hiển thị :SearchScreen.

4. User nhập "Vịnh Hạ Long" vào TextField và nhấn "Tìm kiếm".
5. :SearchScreen gọi :LocationSearchController.searchLocation("Vịnh Hạ Long").
6. :LocationSearchController cập nhật isLoading.value = true.
7. :LocationSearchController tạo url (đã encode) và gọi http.get(url) đến :NominatimAPI.
8. :NominatimAPI (máy chủ OpenStreetMap) xử lý truy vấn.
9. :NominatimAPI trả về response (dạng JSON).
10. :LocationSearchController nhận response, giải mã (decode) JSON.
11. :LocationSearchController lặp qua JSON, tạo List<OsmLocation>.
12. :LocationSearchController cập nhật searchResults.value = list và isLoading.value = false.
13. :SearchScreen (lắng nghe Obx) tự động hiển thị danh sách kết quả.
14. User nhấn vào kết quả "Vịnh Hạ Long" trong danh sách.
15. :SearchScreen gọi Get.back(result: selectedOsmLocation).
16. :AddPostScreen (thoát khỏi await) nhận được selectedOsmLocation.
17. :AddPostScreen gọi


```
_uploadPostController.selectLocation(selectedOsmLocation).
```
18. :AddPostScreen (lắng nghe Obx) cập nhật giao diện, hiển thị "Vịnh Hạ Long" trên ListTile.



1.1.1.5. Sơ đồ tuần tự tìm kiếm và chọn địa điểm.

3. Tuần tự: Bình luận một bài viết

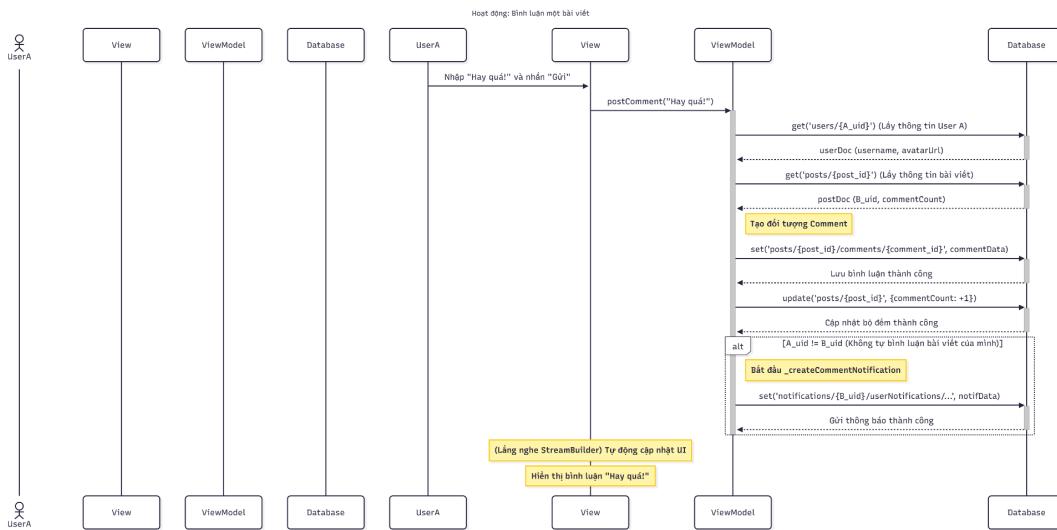
Mô tả tương tác giữa các đối tượng trong hoạt động "Bình luận một bài viết".

- **Actors:** User A (Người bình luận)
- **Lifelines (Đối tượng):**

- :CommentScreen (View)
- :CommentController (ViewModel)
- :Firestore (Database)

Luồng sự kiện:

1. User A nhập "Hay quá!" vào TextField và nhấn nút "Gửi" trên :CommentScreen.
2. :CommentScreen gọi :CommentController.postComment("Hay quá!").
3. :CommentController gọi :Firestore.get('users/{A_uid}') (để lấy username và avatarUrl của User A).
4. :Firestore trả về userDoc (chứa dữ liệu của A).
5. :CommentController gọi :Firestore.get('posts/{post_id}') (để lấy uid của chủ bài viết (User B)).
6. :Firestore trả về postDoc (chứa uid của B và commentCount hiện tại).
7. :CommentController tạo một đối tượng Comment (chứa "Hay quá!", username của A, v.v.).
8. :CommentController gọi :Firestore.set('posts/{post_id}/comments/{comment_id}', commentData) (Lưu bình luận vào sub-collection).
9. :CommentController gọi :Firestore.update('posts/{post_id}', {commentCount: +1}) (Cập nhật bộ đếm).
10. (Rẽ nhánh - alt) Nếu A_uid != B_uid:
 11. :CommentController gọi _createCommentNotification(B_uid, ...) (Hàm nội bộ).
 12. :CommentController gọi :Firestore.set('notifications/{B_uid}/userNotifications...', notifData) (Gửi thông báo cho User B).
 13. (Kết thúc rẽ nhánh)
 14. :CommentController (kết thúc hàm).
15. :CommentScreen nhận được cập nhật (qua StreamBuilder từ getComment()) và hiển thị bình luận mới.



1.1.1.6. Sơ đồ tuần tự bình luận một bài viết.

2.3. Thiết kế giao diện người dùng

Phần này mô tả chi tiết thiết kế giao diện (UI) và trải nghiệm người dùng (UX) của từng màn hình chính trong ứng dụng, đồng thời liên kết chúng với các file mã nguồn cụ thể.

2.3.1. Trang đăng nhập/đăng ký

- **login_screen.dart:** Giao diện được thiết kế tối giản, tập trung vào chức năng.
 - **Bố cục:** Sử dụng SingleChildScrollView bọc một Container có chiều cao bằng màn hình. Điều này đảm bảo Column chứa các trường nhập liệu luôn được căn giữa (MainAxisAlignment.center), kể cả khi bàn phím ảo xuất hiện.
 - **Thành phần:** Tái sử dụng widget TextFormField tùy chỉnh cho "Email" và "Password". Trường "Password" được thiết lập isObscure: true.
 - **Tương tác:** Nút "Login" (một InkWell bọc Container) gọi trực tiếp authController.loginUser(). Liên kết "Register" ở dưới cùng điểu hướng (Get.to) người dùng đến SignupScreen.
- **signup_screen.dart:** Tương tự LoginScreen về bố cục, cũng sử dụng SingleChildScrollView.
 - **Thành phần bổ sung:**
 - **Chọn ảnh đại diện:** Một CircleAvatar lớn được bọc trong Obx. Widget này lắng nghe biến authController.profilePhoto (là một Rx<Uint8List?>). Khi người dùng chọn ảnh (authController.pickImage()), biến Rx thay đổi, và Obx tự động build lại UI để hiển thị ảnh (MemoryImage) thay vì icon Icons.person mặc định.
 - **Trường Tên:** Thêm một TextFormField cho "Tên đầy đủ".

- **Tương tác:** Nút "Register" gọi authController.registerUser() và truyền vào cả 4 giá trị: tên, email, mật khẩu, và authController.profilePhoto (dữ liệu bytes của ảnh).

2.3.2. Trang chính (HomeScreen)

- **home_screen.dart:** Đây là Scaffold chính của ứng dụng sau khi đăng nhập, sử dụng BottomNavigationBar để điều hướng.
 - **Thanh điều hướng dưới:** Gồm 5 mục. Thanh này được thiết lập type: BottomNavigationBarType.fixed và sử dụng pageIdx (một state của HomeScreen) để cập nhật currentIndex. Màu sắc được lấy từ constants.dart (buttonColor).
 - **Nút "Add":** Mục ở giữa không phải là icon chuẩn mà là widget CustomIcon. Đây là một Stack gồm 3 Container xếp chồng (2 Container màu và 1 Container trắng có icon Icons.add ở giữa), tạo hiệu ứng thị giác giống TikTok.
 - **Nội dung:** Thân (body) của Scaffold là một IndexedStack. Việc sử dụng IndexedStack (thay vì Stack hoặc PageView) là một lựa chọn kỹ thuật quan trọng, vì nó **giữ nguyên trạng thái** của cả 5 màn hình con. Điều này có nghĩa là khi người dùng cuộn PostFeedScreen, chuyển sang tab SearchScreen rồi quay lại, vị trí cuộn của PostFeedScreen vẫn được bảo toàn.
 - **Tab Home (PostFeedScreen):** Màn hình đầu tiên (index 0) là PostFeedScreen. Cốt lõi của màn hình này là một PageView.builder được thiết lập scrollDirection: Axis.vertical. Mỗi "trang" (page) của PageView này là một bài đăng đầy đủ (_buildPostPage).

2.3.3. Trang tạo bài viết mới

- **add_post_screen.dart:** Màn hình cho phép tạo và đăng bài viết mới.
 - **Bố cục ảnh:** UI được điều khiển bởi Obx lắng nghe UploadPostController.
 - Khi chưa có ảnh (pickedImageBytesList và existingImageUrls đều rỗng), màn hình hiển thị một Container trống với icon Icons.add_a_photo_outlined.
 - Khi đã có ảnh, Container này được thay thế bằng một PageView cuộn ngang (để xem các ảnh) và một SmoothPageIndicator (dấu chấm) bên dưới.
 - **Xóa ảnh:** Mỗi ảnh trong PageView được bọc trong Stack. Một IconButton "X" được đặt ở góc trên bên phải. Nút này có logic thông minh: nó kiểm tra xem ảnh là ảnh cũ (URL) hay ảnh mới (bytes) để gọi đúng hàm removeExistingImageAt hoặc removePickedImageAt từ UploadPostController.
 - **Mô tả:** Là một TextField đơn giản với maxLines: 4.
 - **Địa điểm:** Là một ListTile cũng được bọc trong Obx để lắng nghe selectedLocation từ controller. Khi nhấn vào, nó gọi hàm

_openLocationPicker(), hàm này thực hiện await Get.to(() => SearchScreen(isPickerMode: true)). Đây là một ví dụ về điều hướng để lấy kết quả (navigation-for-result) bằng GetX.

2.3.4. Trang chỉnh sửa bài viết

- **Tái sử dụng add_post_screen.dart:** Không có file riêng, thay vào đó AddPostScreen được thiết kế để nhận một Post? existingPost tùy chọn qua constructor.
- **Khởi tạo (initState):** Trong initState, màn hình kiểm tra if (widget(existingPost != null)). Nếu có:
 1. Biến isEditing được đặt thành true.
 2. _descriptionController.text được điền bằng post.description.
 3. _uploadPostController.loadExistingImages(post.imageUrls) được gọi để nạp danh sách URL ảnh cũ vào existingImageUrls (một RxList).
 4. _uploadPostController.selectedLocation.value được gán bằng OsmLocation (được tạo tạm) từ post.locationName.
- **Hiển thị:** Obx trong hàm build sẽ tự động đọc existingImageUrls và hiển thị các ảnh cũ ngay lập tức. Tiêu đề AppBar cũng thay đổi dựa trên biến isEditing.

2.3.5. Trang hồ sơ cá nhân (ProfileScreen)

- **profile_screen.dart:** Một màn hình có bộ cục phức tạp, sử dụng NestedScrollView để tạo hiệu ứng AppBar và Header cuộn theo, trong khi TabBar "dính" (pinned) lại.
- **Bộ cục:**
 - DefaultTabController (bọc ngoài cùng).
 - Obx (lắng nghe profileController.user, là một RxMap).
 - Scaffold + NestedScrollView.
 - headerSliverBuilder: Chứa SliverToBoxAdapter (cho phần _buildHeaderInfo - ảnh đại diện, thông kê) và SliverPersistentHeader (cho TabBar dính).
 - body: Chứa TabBarView (cho 3 tab nội dung).
- **Header (_buildHeaderInfo):** Hiển thị CircleAvatar và các cột thông kê (_buildStatColumn). Các nút "Follow" / "Nhắn tin" hiển thị dựa trên logic:
 - Nếu isCurrentUser, hiển thị nút Edit Name (gọi _showEditNameDialog, một Get.dialog).
 - Nếu không, hiển thị nút "Follow" (màu sắc và chữ thay đổi dựa trên controller.user['isFollowing'] == true) và nút "Nhắn tin" (chỉ hiển thị nếu isFollowing == true và isFollowedBy == true - tức là "Bạn bè").
- **Body (Tabs):**
 - **Tab 1 (Bài đăng):** _buildPostedGrid xây dựng một GridView từ controller.postedList. Mỗi item là một PostGridItem.

- **Tab 2 (Đã thích):** _buildLikedGrid xây dựng GridView từ controller.likedList.
- **Tab 3 (Đã lưu):** _buildFavoritedList xây dựng ListView từ controller.favoritedLocationsList.

2.3.6. Trang tìm kiếm địa điểm và người dùng (SearchScreen)

- **search_screen.dart:** Một màn hình có hai chế độ hoạt động, được quyết định bởi final bool isPickerMode.
 - **Bố cục:** Scaffold có AppBar chứa TextField tìm kiếm và một TabBar (2 tab) ở bottom.
 - **Chế độ Picker (isPickerMode: true):**
 - TabBar bị ẩn (bottom: widget.isPickerMode ? null : TabBar(...)).
 - Tab "Người dùng" bị vô hiệu hóa.
 - Khi nhấp vào một kết quả địa điểm, hàm onTap sẽ gọi Get.back(result: result).
 - **Chế độ Tìm kiếm (Mặc định):**
 - TabBar hiển thị "Địa điểm" và "Người dùng".
 - **Tab "Địa điểm":** Giao diện Obx phức tạp. Nếu _locationController.searchResults rỗng, nó hiển thị _buildLocationDefaultView (gồm lịch sử tìm kiếm _locationController.recentSearches và đề xuất). Nếu có kết quả, nó hiển thị _buildLocationResultsList.
 - **Tab "Người dùng":** Giao diện Obx đơn giản, xây dựng ListView từ _userController.searchedUsers.

2.3.7. Trang nhắn tin và thông báo (MailBoxScreen)

- **mailbox_screen.dart:** Một Scaffold đơn giản, dùng DefaultTabController để bọc TabBar (2 tab) và TabBarView.
- **Tab "Tin nhắn" (conversations_screen.dart):**
 - Màn hình này **không** hiển thị các cuộc trò chuyện. Nó hiển thị **danh sách bạn bè** (friendsList từ ConversationController).
 - Trong initState, nó gọi convController.loadFriends() để tải danh sách bạn bè (mutual followers).
 - Khi người dùng nhấp vào một người bạn, onTap sẽ tính toán roomId (ghép uid 2 người) và gọi Get.to(() => ChatDetailScreen(...)).
- **Tab "Thông báo" (notifications_screen.dart):**
 - Sử dụng Obx để lắng nghe controller.notifications.
 - Mỗi thông báo là một ListTile tùy chỉnh (_buildNotificationTile). Widget này sử dụng switch (notif.type) để quyết định IconData (ví dụ: Icons.favorite cho NotificationType.like) và Color.

2.3.8. Trang bình luận (CommentScreen)

- **File:** comment_screen.dart
- **Mục đích:** Hiển thị tất cả bình luận cho một postId cụ thể.
- **Khởi tạo:** Khi CommentScreen được tạo, nó gọi commentController.updatePostId(postId) để CommentController biết cần tải bình luận của bài đăng nào.
- **Bố cục:**
 - **AppBar:** Tiêu đề "Bình luận".
 - **Thân (Body):** Column gồm:
 1. Expanded chứa Obx và ListView.builder. Obx lắng nghe commentController.comments. Mỗi hàng ListTile sử dụng CachedNetworkImageProvider(comment.avatarUrl) và timeago.format(comment.createdAt.toDate(), locale: 'vi') để hiển thị thời gian thân thiện (ví dụ: "5 phút trước").
 2. Một ListTile cố định ở dưới cùng (bọc trong Padding) chứa TextField (_commentTextController) và IconButton (nút Gửi). Nút Gửi gọi commentController.postComment(...).

2.3.9. Trang chi tiết trò chuyện (ChatDetailScreen)

- **File:** chat_detail_screen.dart
- **Mục đích:** Hiển thị và gửi tin nhắn trong một roomId cụ thể.
- **Bố cục:**
 - **AppBar:** Hiển thị CircleAvatar và Tên (receiverName) của người nhận, được truyền qua constructor.
 - **Thân (Body):** Column gồm:
 1. **Danh sách tin nhắn (Expanded):** Sử dụng StreamBuilder<List<Message>> (không phải Obx) vì controller trả về một Stream trực tiếp từ Firestore. StreamBuilder tự động lắng nghe và build lại khi có tin nhắn mới.
 2. ListView.builder được đặt reverse: true, khiến danh sách bắt đầu từ dưới lên, đây là kỹ thuật tiêu chuẩn cho giao diện chat.
 3. **Bong bóng chat (_buildMessageBubble):** Hàm này kiểm tra bool isMe = message.senderId == myUid. Dựa vào isMe, nó thay đổi Alignment (trái/phải) và Color (buttonColor/màu xám) của bong bóng chat.
 4. **Ô nhập tin nhắn (_buildTextInput):** Container cố định ở dưới cùng chứa TextField và IconButton (Gửi).

2.4. Thiết kế cơ sở dữ liệu (Firebase)

2.4.1. Mô hình dữ liệu

Collections cấp cao (Root Collections):

1. **users** (Model: user.dart)
 - uid (String): ID từ Firebase Auth (Khóa chính).
 - email (String): Email đăng nhập.
 - username (String): Tên định danh (không dấu, không cách).
 - displayName (String): Tên hiển thị (có dấu).
 - avatarUrl (String): Link ảnh đại diện (từ Cloudinary).
 - bio (String): Mô tả cá nhân.
 - createdAt (Timestamp): Ngày tạo tài khoản.
 - searchKeywords (List<String>): Mảng các từ khóa để tìm kiếm (ví dụ: ["hovan", "van", "a"]).
 - friends (List<String>): Mảng các uid của bạn bè (mutual followers).
2. **posts** (Model: post.dart)
 - id (String): ID duy nhất của bài đăng (Khóa chính).
 - uid (String): ID của người đăng.
 - username (String): Tên người đăng (sao chép để dễ truy vấn).
 - avatarUrl (String): Ảnh đại diện người đăng (sao chép).
 - locationName (String): Tên địa điểm được gắn thẻ.
 - description (String): Mô tả bài đăng.
 - imageUrls (List<String>): Danh sách các link ảnh (từ Cloudinary).
 - createdAt (Timestamp): Thời gian đăng.
 - likes (List<String>): Mảng các uid của người đã thích.
 - commentCount (int): Tổng số bình luận (để tối ưu hiển thị).
3. **locations** (Model: location.dart)
 - id (String): Tên địa điểm (Khóa chính, ví dụ: "Vịnh Hạ Long").
 - name (String): Tên địa điểm.
 - province (String): Tỉnh/Thành phố.
 - description (String): Mô tả (nếu có).
 - rating (double): Điểm đánh giá trung bình.
 - reviewCount (int): Tổng số lượt đánh giá.
 - imageUrl (String): Ảnh bìa (nếu có).
4. **chatRooms** (Model: chat_room.dart)
 - id (String): ID phòng chat (Khóa chính, ghép từ 2 uid).
 - participants (List<String>): Mảng 2 uid của người tham gia.
 - participantInfo (Map): Thông tin 2 người (tên, avatar).
 - lastMessage (String): Tin nhắn cuối cùng.
 - lastMessageAt (Timestamp): Thời gian tin nhắn cuối.
 - lastMessageSenderId (String): ID người gửi tin cuối.
5. **favoriteLocations** (Model: favorite_location.dart)
 - id (String): ID duy nhất (Khóa chính, FavLoc_{uid}_{locationId}).
 - userId (String): ID người dùng.
 - locationId (String): ID (tên) của địa điểm.
 - createdAt (Timestamp): Thời gian lưu.

Sub-collections (Lồng nhau):

1. **posts/{postId}/comments** (Model: comment.dart)
 - o id (String): ID bình luận.
 - o postId (String): ID bài đăng cha.
 - o userId (String): ID người bình luận.
 - o username (String): Tên người bình luận (sao chép).
 - o avatarUrl (String): Ảnh đại diện người bình luận (sao chép).
 - o content (String): Nội dung bình luận.
 - o createdAt (Timestamp): Thời gian bình luận.
2. **locations/{locationId}/reviews** (Model: review.dart)
 - o id (String): ID đánh giá.
 - o locationId (String): ID địa điểm cha.
 - o userId (String): ID người đánh giá.
 - o username (String): Tên người đánh giá (sao chép).
 - o avatarUrl (String): Ảnh đại diện người đánh giá (sao chép).
 - o rating (double): Số sao (1-5).
 - o content (String): Nội dung đánh giá.
 - o createdAt (Timestamp): Thời gian đánh giá.
3. **chatRooms/{roomId}/messages** (Model: message.dart)
 - o id (String): ID tin nhắn (tự tạo bởi Firestore).
 - o senderId (String): ID người gửi.
 - o content (String): Nội dung tin nhắn.
 - o createdAt (Timestamp): Thời gian gửi.
4. **notifications/{userId}/userNotifications** (Model: app_notification.dart)
 - o id (String): ID thông báo.
 - o receiverId (String): ID người nhận (chủ collection).
 - o senderId (String): ID người gây ra hành động (like, follow...).
 - o senderName (String): Tên người gửi.
 - o senderAvatar (String): Ảnh đại diện người gửi.
 - o type (Enum/String): Loại thông báo (follow, like, comment, message).
 - o message (String): Nội dung (ví dụ: "đã thích bài viết của bạn").
 - o targetId (String): ID của đối tượng (postId, roomId...).
 - o createdAt (Timestamp): Thời gian.
 - o isRead (bool): Trạng thái đã đọc.

2.4.2. Sơ đồ quan hệ giữa các collection

(Mô tả quan hệ logic trong NoSQL)

- **User (1) --- (N) Post:** Một User (uid) có thể tạo nhiều Post (lưu uid của user).
- **User (1) --- (N) Notification:** Một User (uid) có một sub-collection userNotifications.
- **User (N) --- (N) ChatRoom:** Hai User (uid) tham gia vào một ChatRoom (lưu trong mảng participants).

- **User (N) --- (N) Location (Favorite):** Quan hệ nhiều-nhiều được thể hiện qua collection trung gian favoriteLocations.
- **User (N) --- (N) Post (Like):** Quan hệ nhiều-nhiều được thể hiện qua mảng likes (List<uid>) trong mỗi Post.
- **User (N) --- (N) User (Follow):** Quan hệ nhiều-nhiều được thể hiện qua các mảng followers và following.
- **Post (1) --- (N) Location:** Một Post chỉ được gắn thẻ 1 Location (lưu locationName).
- **Location (1) --- (N) Post:** Một Location có thể được gắn thẻ trong nhiều Post (truy vấn posts where locationName == ...).
- **Post (1) --- (N) Comment:** Một Post có một sub-collection comments.
- **Location (1) --- (N) Review:** Một Location có một sub-collection reviews.
- **ChatRoom (1) --- (N) Message:** Một ChatRoom có một sub-collection messages.

CHƯƠNG 3. TRIỂN KHAI ỨNG DỤNG

3.1. Cài đặt môi trường phát triển (Flutter SDK, Android Studio)

Để triển khai và phát triển dự án "Sketch Travel", môi trường phát triển cần các thành phần sau:

1. **Flutter SDK:** Nền tảng chính để phát triển và biên dịch ứng dụng. Phiên bản Flutter được sử dụng cần tương thích với các thư viện đã dùng.
2. **Ngôn ngữ Dart:** Đi kèm với Flutter SDK, là ngôn ngữ chính để viết mã nguồn.
3. **IDE (Môi trường phát triển tích hợp):** Android Studio hoặc Visual Studio Code, với các plugin Flutter và Dart để hỗ trợ lập trình, gỡ lỗi và "hot reload".
4. **Firebase CLI:** Công cụ dòng lệnh để kết nối dự án Flutter với backend Firebase, đặc biệt là để tạo file firebase_options.dart.
5. **Tài khoản Firebase:** Cần thiết để thiết lập các dịch vụ backend:
 - **Authentication:** Kích hoạt phương thức đăng nhập Email/Password.
 - **Cloud Firestore:** Thiết lập cơ sở dữ liệu NoSQL.
6. **Tài khoản Cloudinary:** Cần thiết để lấy cloudName, uploadPreset, và apiKey [cite: cloudinary_controller.dart] cho việc lưu trữ và phân phối hình ảnh.

3.2. Cấu trúc thư mục và module trong dự án

Mã nguồn dự án được tổ chức theo kiến trúc MVVM (Model-View-ViewModel) với GetX, phân tách rõ ràng các thành phần:

- **/lib:** Thư mục gốc chứa toàn bộ mã nguồn Dart.
 - **main.dart:** Điểm khởi đầu của ứng dụng. Khởi tạo Firebase, GetStorage, và các controllers chính (như AuthController, ThemeController).
 - **constants.dart:** File quan trọng định nghĩa các biến toàn cục, hằng số màu sắc (buttonColor), và các đối tượng truy cập Firebase (firestore, firebaseAuth).
 - **/models:** Chứa các lớp Model (PODO - Plain Old Dart Objects) định nghĩa cấu trúc dữ liệu, đi kèm các phương thức toJson() và fromJson()/fromSnap() để giao tiếp với Firestore.
 - *Ví dụ:* user.dart, post.dart, location.dart, comment.dart, app_notification.dart.
 - **/controllers:** Chứa các lớp GetxController, đóng vai trò là ViewModel. Đây là nơi chứa toàn bộ logic nghiệp vụ (business logic), quản lý trạng thái, và tương tác với backend.

- *Ví dụ:* auth_controller.dart, post_controller.dart, profile_controller.dart, upload_post_controller.dart.
- **/views:** Chứa các file giao diện người dùng (UI), được chia thành 2 nhóm:
 - **/screens:** Các màn hình chính của ứng dụng (ví dụ: home_screen.dart, profile_screen.dart, add_post_screen.dart).
 - **/widgets:** Các thành phần UI có thể tái sử dụng (ví dụ: text_input_field.dart, post_grid_item.dart).

3.3. Triển khai các chức năng chính

Phần này sẽ đi sâu vào việc triển khai mã nguồn cho các tính năng cốt lõi của ứng dụng.

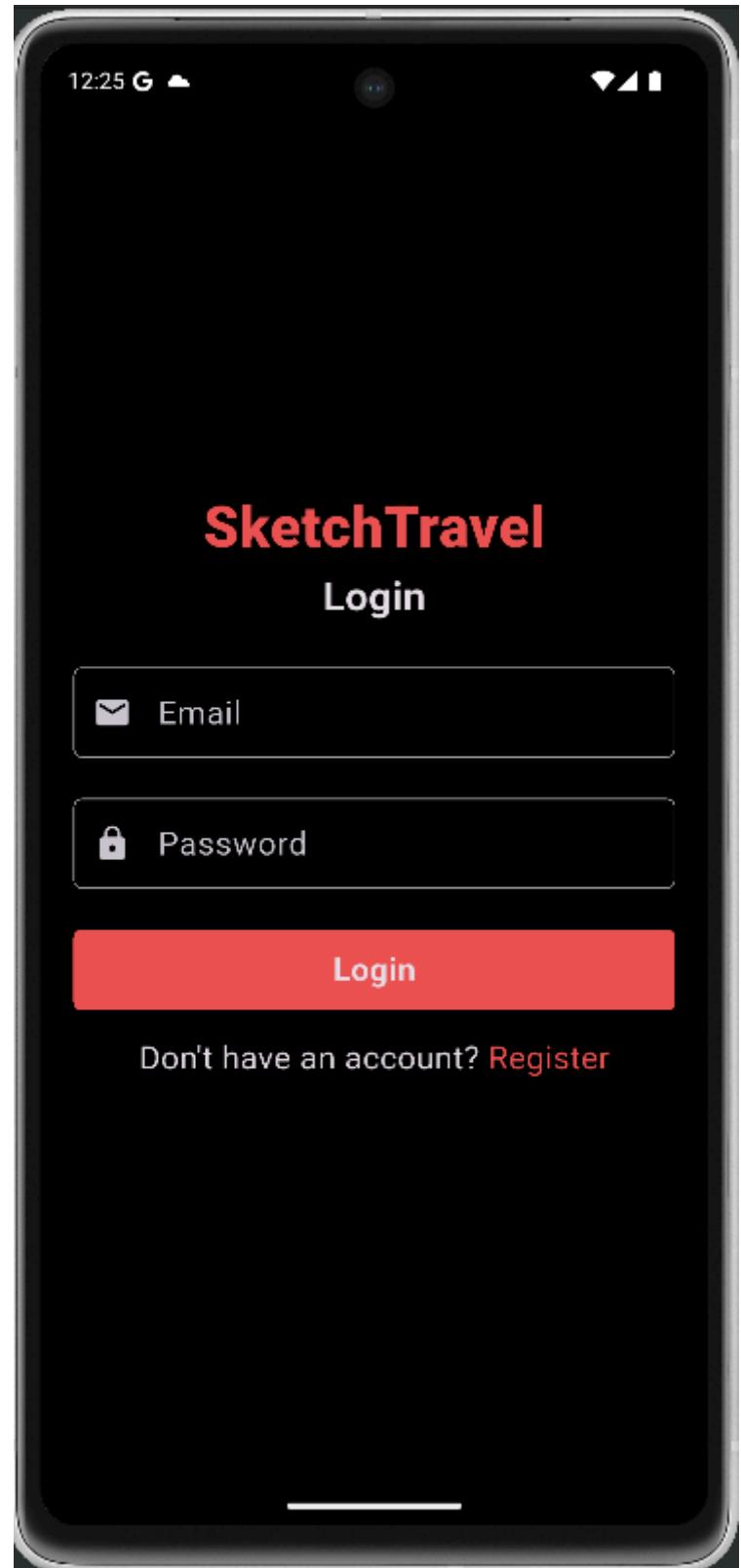
3.3.1. Đăng ký, đăng nhập (Authentication)

Kết quả đạt được:

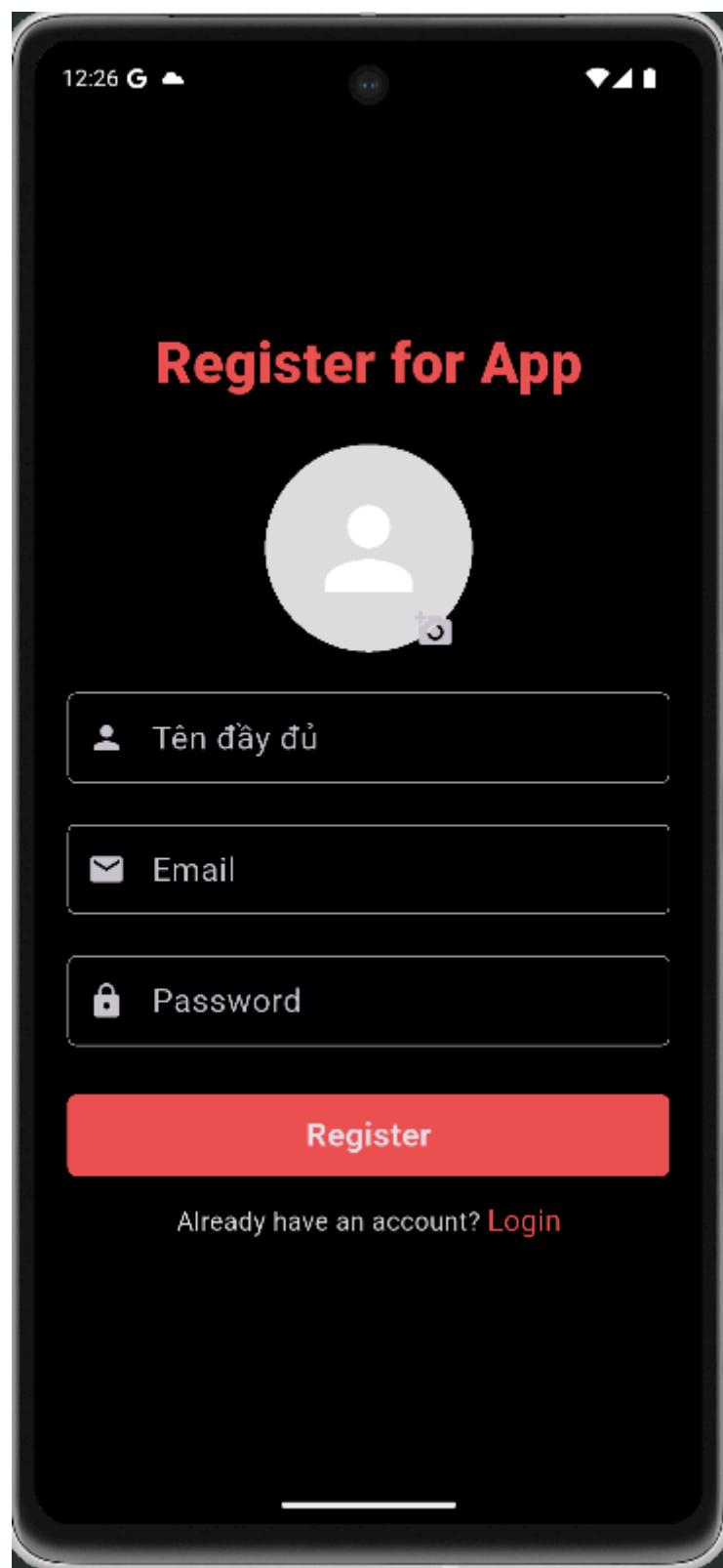
Hoàn thành luồng xác thực người dùng. Người dùng có thể tạo tài khoản mới với Tên, Email, Mật khẩu và Ảnh đại diện (SignupScreen). Dữ liệu người dùng Firebase Auth được tạo, ảnh được tải lên Cloudinary, và một tài liệu (users) tương ứng được ghi vào Firestore.

Trạng thái đăng nhập được quản lý tự động. AuthController lắng nghe authStateChanges() từ Firebase. Khi người dùng mở ứng dụng, họ sẽ được tự động điều hướng đến HomeScreen (nếu đã đăng nhập) hoặc LoginScreen (nếu chưa).

DEMO kết quả:



1.1.1.7. Giao diện login.



1.1.1.8. Giao diện register.

Mã code đại diện:

Đây là đoạn mã cốt lõi trong auth_controller.dart thể hiện quy trình 3 bước khi đăng ký: (1) Tạo Auth, (2) Upload ảnh, (3) Tạo Document.

```

Future<void> registerUser(
    String fullName,
    String email,
    String password,
    Uint8List? imageBytes) async {

    UserCredential? cred;

    try {
        if (fullName.isNotEmpty &&
            email.isNotEmpty &&
            password.isNotEmpty &&
            imageBytes != null) {

            // BƯỚC 1: Tạo người dùng trên Firebase Authentication
            cred = await firebaseAuth.createUserWithEmailAndPassword(
                email: email,
                password: password,
            );
        }

        // BƯỚC 2: Tải ảnh đại diện lên Cloudinary
        String downloadUrl = await
            CloudinaryController.instance.uploadImage(imageBytes);

        // BƯỚC 3: Tạo đối tượng User và lưu vào Firestore
        model.User newUser = model.User(
            uid: cred.user!.uid,
            email: email,
            username: newUsername,
            displayName: newName,
            name: newName,
            avatarUrl: downloadUrl,
        );
    }
}

```

```

        // ... (các trường khác)

    ) ;

    await firestore

        .collection('users')

        .doc(cred.user!.uid)

        .set(newUser.toJson());
    }

} else {

    Get.snackbar('Error creating account', "Please fill in
all fields");

}

} catch (e) {

    Get.snackbar('Error creating account', e.toString());

    // (Xử lý xóa user auth nếu 2 bước sau thất bại)

}
}

```

3.3.2. Đăng bài viết mới (kèm gắn thẻ Địa điểm)

Kết quả đạt được:

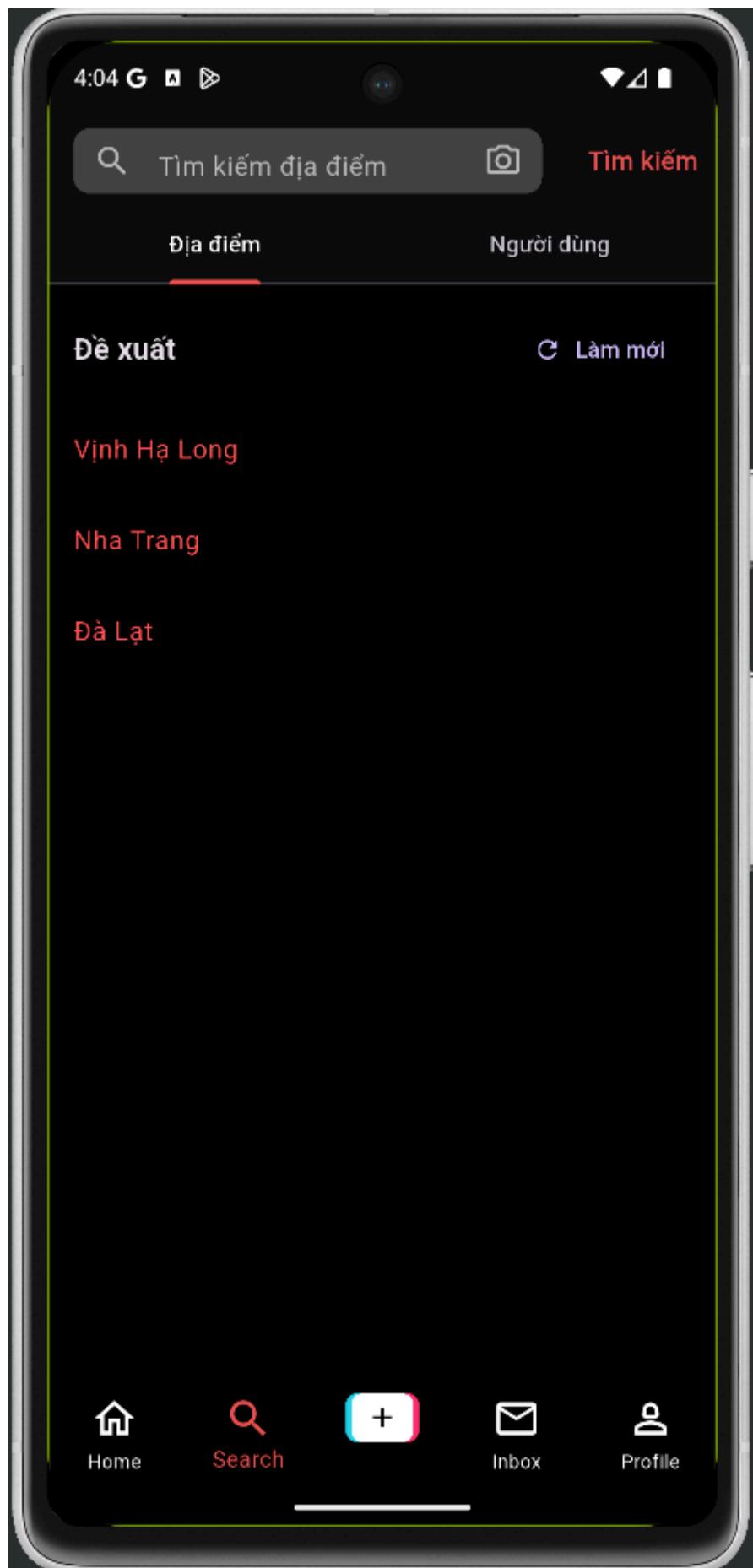
Người dùng có thể tạo một bài đăng mới (AddPostScreen). Chức năng cho phép chọn nhiều ảnh, viết mô tả, và (quan trọng nhất) gắn thẻ một địa điểm.

Việc gắn thẻ địa điểm được thực hiện bằng cách tái sử dụng SearchScreen ở chế độ isPickerMode. Người dùng tìm kiếm địa điểm từ API OpenStreetMap, chọn một kết quả, và Get.back() với dữ liệu OsmLocation về AddPostScreen.

DEMO kết quả:



1.1.1.9. Giao diện tạo bài viết mới.



1.1.1.10. Giao diện tạo chọn địa điểm.

Mã code đại diện:

Đây là hàm uploadPost trong upload_post_controller.dart, thể hiện logic tải nhiều ảnh lên Cloudinary và sau đó ghi tài liệu Post vào Firestore.

```
Future<void> uploadPost(  
    String description,  
    List<String> oldImages, // Dùng cho chỉnh sửa, ở đây sẽ rỗng  
    String locationName,  
) async {  
  
    if (isLoading.value) return;  
  
    try {  
  
        isLoading.value = true;  
  
        // ... (Kiểm tra và lấy thông tin user) ...  
  
        // Tải ảnh mới (bytes) lên Cloudinary  
        List<String> uploadedUrls = [];  
        for (final bytes in pickedImageBytesList) {  
  
            // Gọi hàm upload bất đồng bộ  
            final url = await  
CloudinaryController.instance.uploadImage(bytes);  
  
            uploadedUrls.add(url);  
        }  
  
        // (Kết hợp ảnh cũ và mới, ở đây chỉ có ảnh mới)  
        final imageUrl = [...uploadedUrls, ...oldImages];  
  
        final postId = firestore.collection('posts').doc().id;  
  
        // Tạo đối tượng Post
```

```

final newPost = Post(
    id: postId,
    uid: user.uid,
    username: username,
    avatarUrl: avatarUrl,
    locationName: locationName, // Gắn thẻ địa điểm
    description: description,
    imageUrls: imageUrls, // Danh sách URL ảnh
    createdAt: Timestamp.now(),
    likes: [],
    commentCount: 0,
);

// Ghi lên Firestore
await
firestore.collection('posts').doc(postId).set(newPost.toJson()
());

```

Get.snackbar('Thành công', 'Bài viết mới đã được đăng!');

// ... (Dọn dẹp và điều hướng về Home) ...

```

} catch (e) {
    Get.snackbar('Lỗi', e.toString());
} finally {
    isLoading.value = false;
}
}

```

3.3.3. Chỉnh sửa, xóa bài viết (và Quản lý ảnh)

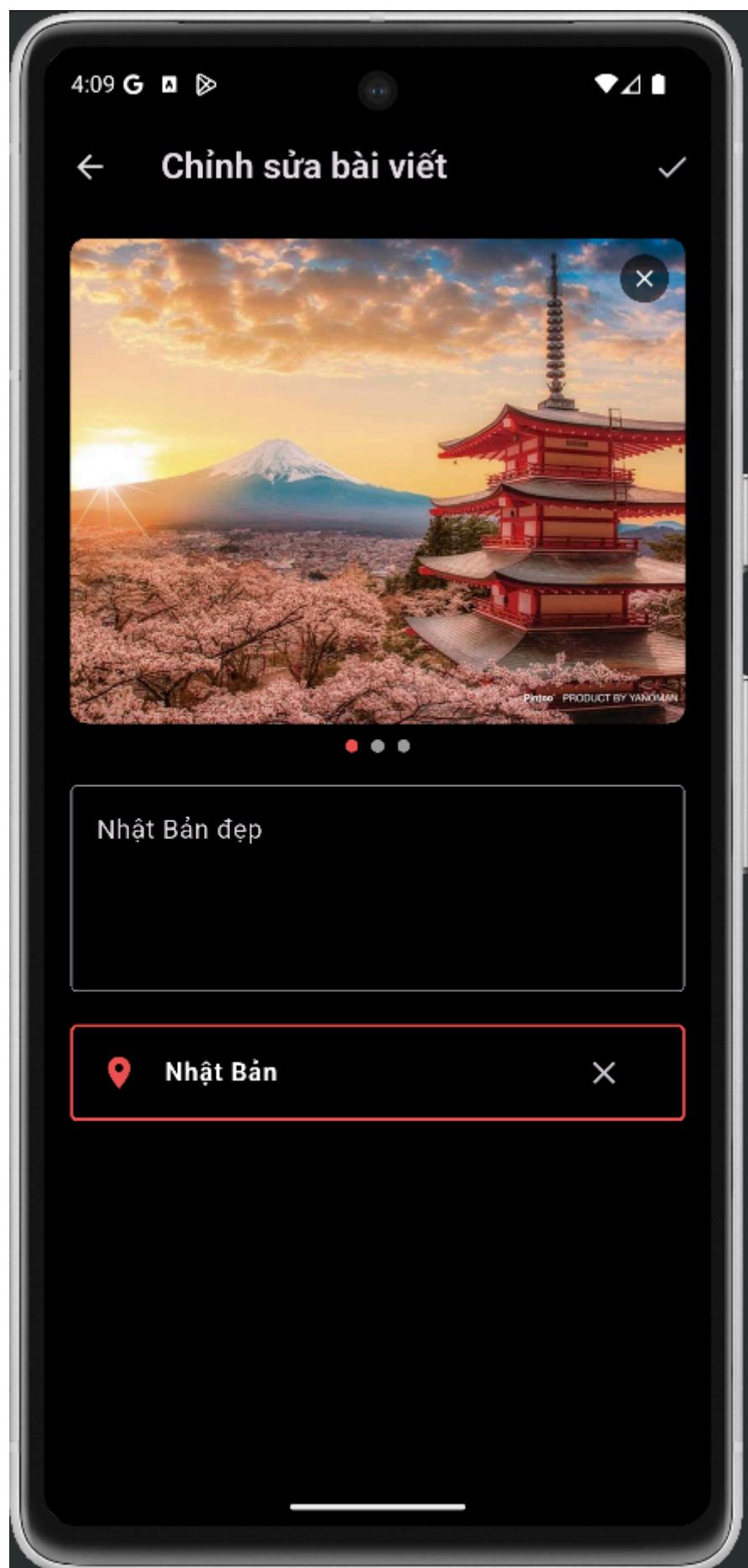
Kết quả đạt được:

Chủ bài viết có thẻ chỉnh sửa (AddPostScreen được tái sử dụng) hoặc xóa bài viết (ProfileScreen).

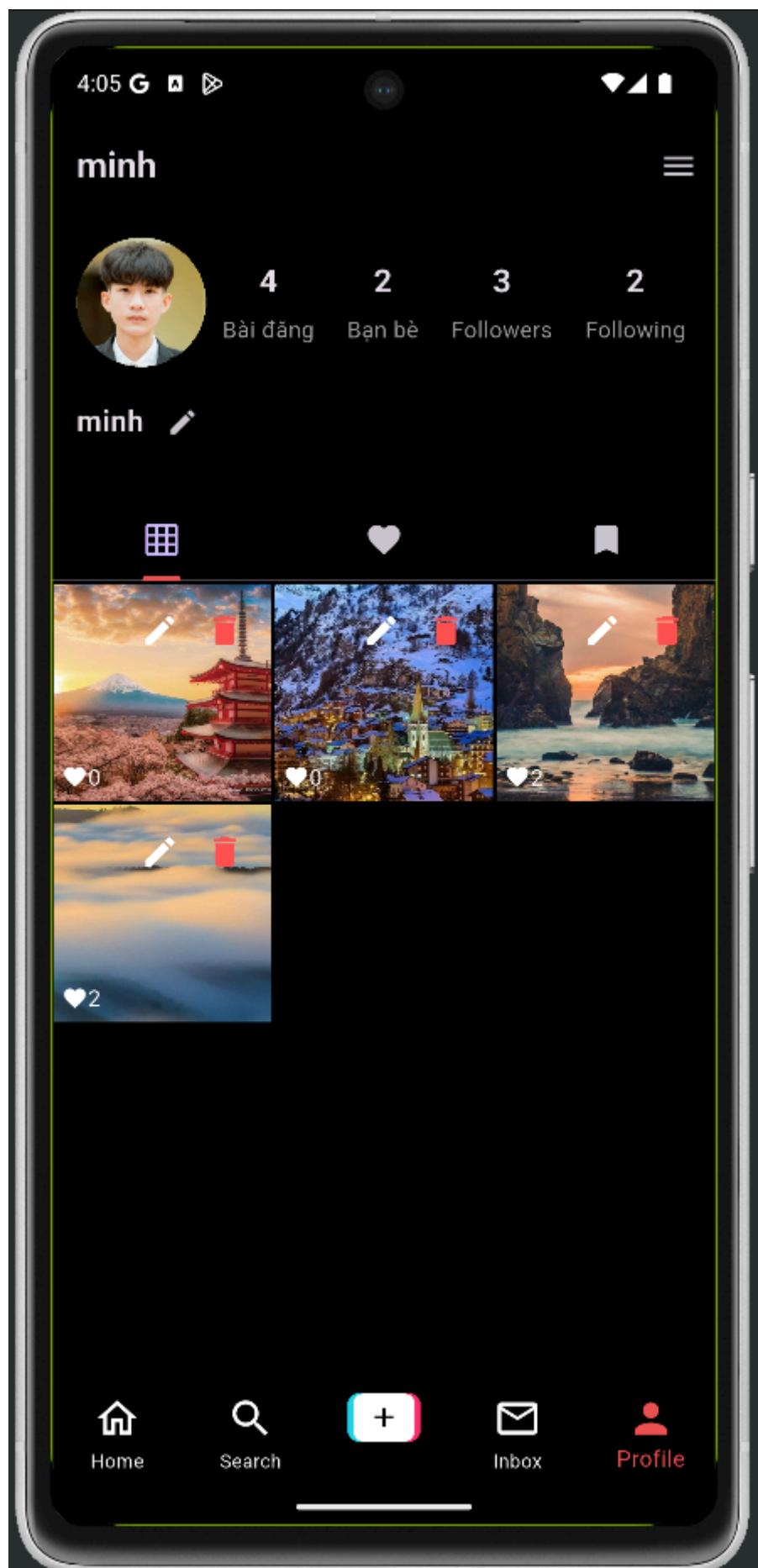
Khi chỉnh sửa, người dùng có thể xóa ảnh cũ (existingImageUrls) hoặc thêm ảnh mới (pickedImageBytesList). Giao diện PageView trong AddPostScreen hiển thị kết hợp cả hai danh sách này. Khi cập nhật, hàm updatePost sẽ upload ảnh mới (nếu có) và ghi đè danh sách imageUrls trên Firestore.

Việc xóa bài viết được thực hiện từ ProfileScreen, gọi hàm deletePostAndRefresh() để xóa tài liệu trên Firestore và cập nhật lại RxList trên UI.

DEMO kết quả:



1.1.1.11. Giao diện chỉnh sửa bài viết.



1.1.1.12. Giao diện tạo xóa bài viết.

Mã code đại diện:

Đây là hàm updatePost và logic quản lý 2 danh sách ảnh trong upload_post_controller.dart.

```
// Hai danh sách quản lý ảnh riêng biệt:  
  
RxList<Uint8List> pickedImageBytesList = <Uint8List>[].obs; //  
    Ảnh mới (bytes)  
  
RxList<String> existingImageUrls = <String>[].obs; // Ảnh cũ  
    (URL)  
  
  
// Hàm xóa ảnh cũ (khi đang edit)  
  
void removeExistingImageAt(int index) {  
  
    existingImageUrls.removeAt(index);  
  
}  
  
  
// Hàm cập nhật bài viết  
  
Future<void> updatePost(  
  
    String postId,  
  
    String description,  
  
    List<String> existingImages, // Danh sách ảnh cũ còn lại  
  
    String locationName,  
  
) async {  
  
    if (isLoading.value) return;  
  
    try {  
  
        isLoading.value = true;  
  
  
        // Upload thêm ảnh mới nếu có  
  
        List<String> uploadedUrls = [];  
  
        for (final bytes in pickedImageBytesList) {
```

```

        final url = await
CloudinaryController.instance.uploadImage(bytes);

uploadedUrls.add(url);

}

// Kết hợp ảnh cũ (đã lọc) và ảnh mới

final allImages = [...existingImages, ...uploadedUrls];

// Cập nhật tài liệu trên Firestore

await firestore.collection('posts').doc(postId).update({
    'description': description,
    'imageUrls': allImages,
    'locationName': locationName,
    'updatedAt': Timestamp.now(),
});

Get.snackbar('Thành công', 'Bài viết đã được cập nhật');

// ... (Dọn dẹp và điều hướng) ...

} catch (e) {
    Get.snackbar('Lỗi cập nhật', e.toString());
} finally {
    isLoading.value = false;
}
}

```

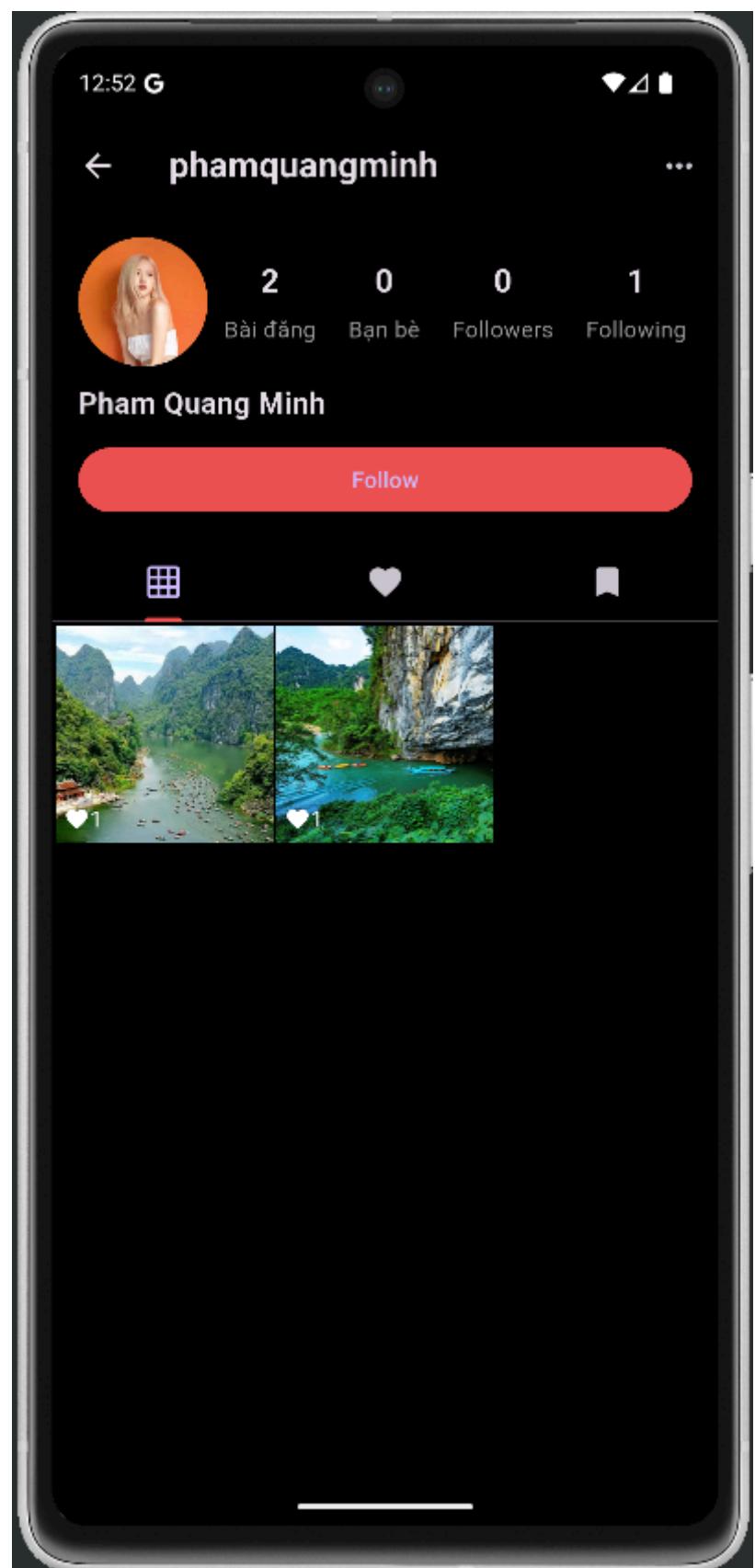
3.3.5. Quản lý người dùng (Follow, Friend)

Kết quả đạt được:

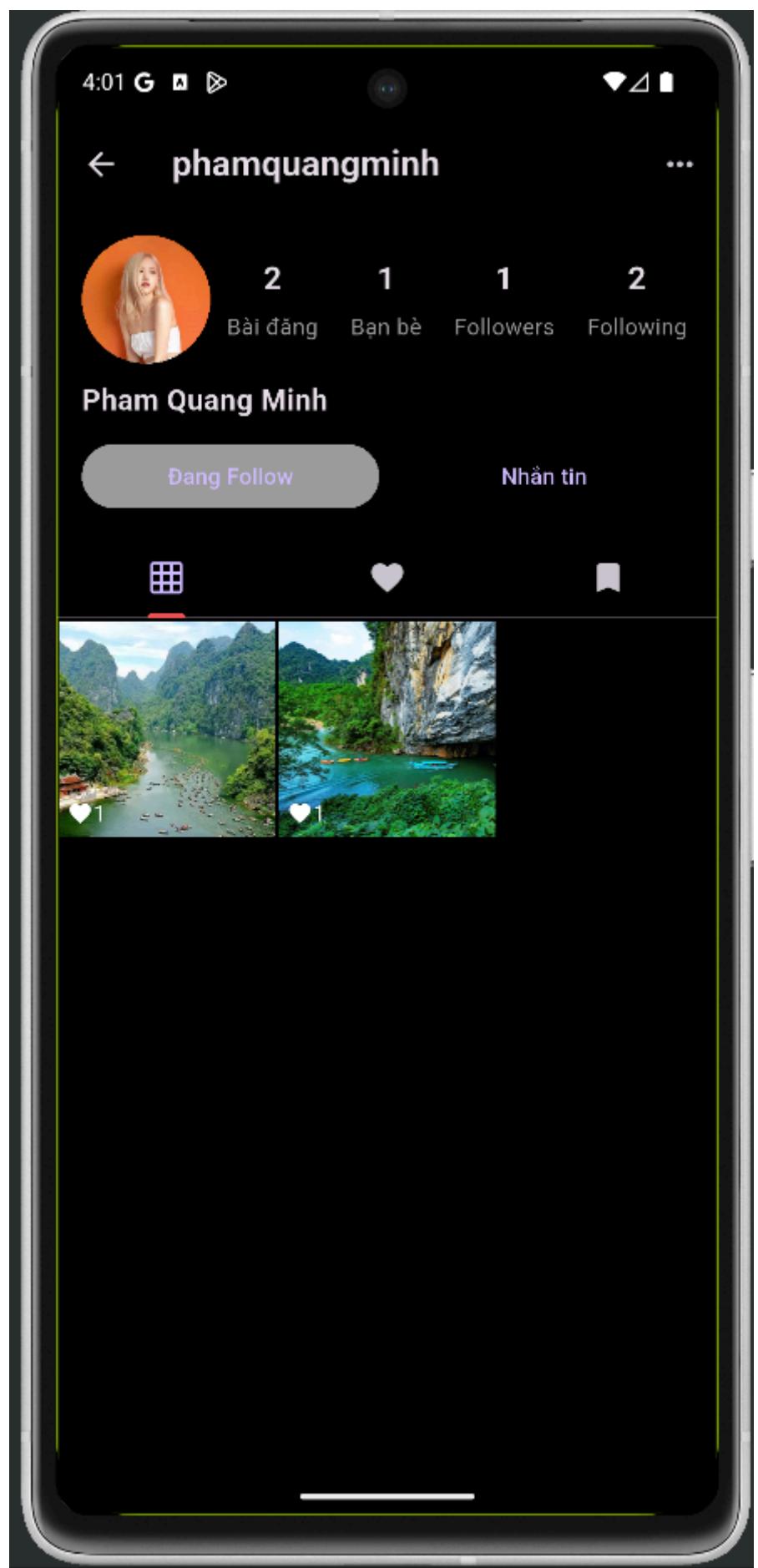
Người dùng có thể xem hồ sơ (ProfileScreen) của người khác và nhấn "Follow". Nút này sẽ đổi trạng thái thành "Đang Follow".

Logic "Bạn bè" (Friend) được xử lý tự động. Khi User A (đang follow User B) nhấn "Follow", hệ thống kiểm tra nếu User B cũng đang follow User A, thì cả hai sẽ được thêm vào mảng friends của nhau. Giao diện ProfileScreen sẽ hiển thị thêm nút "Nhắn tin".

DEMO kết quả:



1.1.1.13. Giao diện ở profile người khác khi chưa follow.



1.1.1.14. Giao diện ở profile người khác khi đã follow.

Mã code đại diện:

Đây là hàm followUser trong profile_controller.dart, thể hiện logic cập nhật followers, following, và logic "Bạn bè" (mutual follow).

```
Future<void> followUser() async {  
    // ... (try/catch và lấy UID) ...  
  
    final String currentUid = authController.userAccount.uid;  
  
    final String targetUid = _uid.value;  
  
  
    final userRef = firestore.collection('users').doc(targetUid);  
  
    final currentUserRef =  
        firestore.collection('users').doc(currentUid);  
  
  
    // ... (Lấy snapshot của 2 user) ...  
  
    List followers = List.from(userData['followers'] ?? []);  
  
  
    if (followers.contains(currentUid)) {  
        // *** LUÔNG UNFOLLOW ***  
  
        await userRef.update({  
            'followers': FieldValue.arrayRemove([currentUid])  
        });  
  
        await currentUserRef.update({  
            'following': FieldValue.arrayRemove([targetUid])  
        });  
  
        // Xóa bạn bè  
  
        await currentUserRef.update({'friends':  
            FieldValue.arrayRemove([targetUid])});  
  
        await userRef.update({'friends':  
            FieldValue.arrayRemove([currentUid])});  
    }  
}
```

```

    } else {

        // *** LUÔNG FOLLOW ***

        await userRef.set({'followers':
FieldValue.arrayUnion([currentUid]), SetOptions(merge:
true));

        await currentUserRef.set({'following':
FieldValue.arrayUnion([targetUid]), SetOptions(merge:
true));

        // Kiểm tra có follow lẫn nhau không (mutual)

        bool isFollowedBy = (userData['following'] as List? ??
[]).contains(currentUid);

        if (isFollowedBy) {

            await currentUserRef.set({'friends':
FieldValue.arrayUnion([targetUid]), SetOptions(merge:
true));

            await userRef.set({'friends':
FieldValue.arrayUnion([currentUid]), SetOptions(merge:
true));

        }

        await _createFollowNotification(targetUid);

    }

    // ... (Cập nhật UI) ...

}

```

3.3.6. Quản lý dữ liệu trên Firebase (Denormalization)

Kết quả đạt được:

Hệ thống được tối ưu cho tốc độ đọc bằng cách áp dụng kỹ thuật Denormalization (Phi chuẩn hóa). Thay vì thực hiện các truy vấn join hoặc nhiều lần get phức tạp, dữ liệu được "sao chép" đến nơi cần thiết tại thời điểm ghi.

- Ví dụ 1: Khi đăng bình luận, username và avatarUrl của người bình luận được sao chép vào tài liệu comment [cite: [comment_controller.dart](#)].
- Ví dụ 2: Khi đăng bình luận, trường commentCount trên tài liệu post cha được tăng lên +1.
- Ví dụ 3: Khi gửi đánh giá, rating trung bình và reviewCount trên tài liệu location được tính toán và cập nhật lại [cite: [location_controller.dart](#)].

DEMO kết quả:

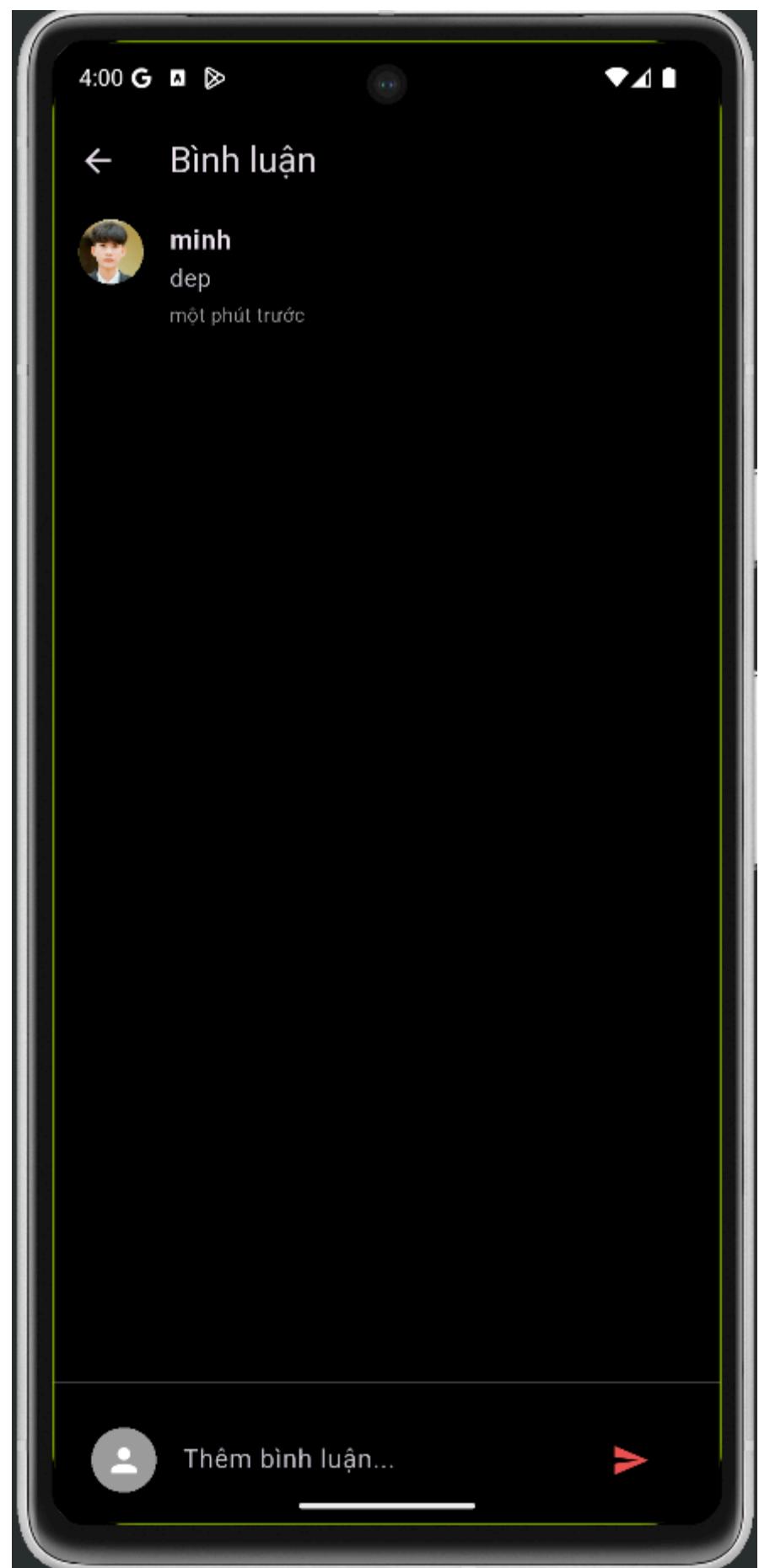
The screenshot shows the Firebase Firestore interface. On the left, under 'Post 2', there is a 'comments' collection with one document named 'Comment 0'. This document contains the following fields:

- avatarUrl: "https://res.cloudinary.com/dcfuybexm/image/upload/v1600000000/logo.png"
- commentCount: 1
- createdAt: November 1, 2025 at 3:13:55PM UTC+7
- description: "Phong nha ke bang"
- id: "Post 2"
- imageUrls: An array containing one element: "https://res.cloudinary.com/dcfuybexm/image/upload/v1600000000/logo.png"

On the right, under 'Comment 0', there is a detailed view of the document with the following fields:

- avatarUrl: "https://res.cloudinary.com/dcfuybexm/image/upload/v1600000000/logo.png"
- content: "dep"
- createdAt: November 4, 2025 at 4:00:09 PM UTC+7
- id: "Comment 0"
- postId: "Post 2"
- userId: "uBf1jPywLYUbEoGak9rerAfly9C2"
- username: "minh"

1.1.1.15. *Firebase Database: post và comment của post đó .*



1.1.1.16. Giao diện xác nhận có comment trong database.

Mã code đại diện:

Đoạn mã từ comment_controller.dart cho thấy việc ghi (1) comment (đã sao chép username), (2) cập nhật commentCount, và (3) tạo notification (cũng sao chép username) chỉ trong một hàm postComment.

```
postComment(String commentText) async {  
  try {  
    // ... (Lấy thông tin user và post) ...  
  
    var userData = userDoc.data() as Map<String, dynamic>; //  
    Dữ liệu sao chép  
  
    String postOwnerId = (postDoc.data()! as dynamic)['uid'];  
  
    Comment comment = Comment(  
      // ... (id, postId, userId) ...  
  
      username: userData['username'] ?? '', // SAO CHÉP DỮ  
      LIỆU  
  
      avatarUrl: userData['avatarUrl'] ?? '', // SAO CHÉP DỮ  
      LIỆU  
  
      content: commentText.trim(),  
      createdAt: Timestamp.now(),  
    );  
  
    // GHI BƯỚC 1: Lưu bình luận (đã chứa dữ liệu sao chép)  
    await firestore  
      .collection('posts')  
      .doc(_postId)  
      .collection('comments')  
      .doc(commentId)  
      .set(comment.toJson());  
  
    // GHI BƯỚC 2: Cập nhật bộ đếm (Denormalization)
```

```

        await firestore.collection('posts').doc(_postId).update({
            'commentCount': (postDoc.data()! as dynamic)['commentCount'] + 1,
        });

        // GHI BƯỚC 3: Tạo thông báo (cũng sao chép dữ liệu)
        if (currentUserId != postOwnerId) {
            await _createCommentNotification(postOwnerId, _postId,
                userData);
        }
    } catch (e) {
        Get.snackbar('Error While Commenting', e.toString());
    }
}

```

3.3.7. Tải ảnh, xử lý upload bất đồng bộ

Kết quả đạt được:

Việc tải ảnh lên Cloudinary là một tác vụ I/O tốn thời gian. Chức năng này được triển khai bất đồng bộ (sử dụng `async/await`) để không làm "đóng băng" giao diện người dùng.

Trong khi upload (ví dụ khi đăng ký hoặc đăng bài), `isLoading.value` được đặt thành `true`, và giao diện (được bọc trong `Obx`) sẽ hiển thị `CircularProgressIndicator` thay vì nút bấm, cung cấp phản hồi trực quan cho người dùng.

Mã code đại diện:

Hàm `uploadImage` trong `cloudinary_controller.dart` là hạt nhân của việc xử lý bất đồng bộ. Nó bọc lệnh gọi `await _cloudinary.uploadFile` trong một `try...catch` và trả về `Future<String>`.

```

// Hàm trả về một Future (lời hứa)
Future<String> uploadImage(Uint8List imageBytes) async {
    try {

```

```

        print('Cloudinary: Bắt đầu tải ảnh lên...');

        // Đợi (await) cho đến khi Cloudinary hoàn thành upload
        CloudinaryResponse response = await _cloudinary.uploadFile(
            CloudinaryFile.fromBytesData(
                imageBytes, // Dữ liệu ảnh
                identifier:
                    'user_profile_${DateTime.now().millisecondsSinceEpoch}',
                folder: 'profilePics',
                context: {'api_key': _apiKey},
            ),
        );
    }

    print('Cloudinary: Tải lên thành công! URL: ${response.secureUrl}');

    return response.secureUrl; // Trả về URL sau khi hoàn
    thành

} on CloudinaryException catch (e) {
    print('Cloudinary: Lỗi tải ảnh: ${e.message}');
    rethrow; // Ném lỗi ra để controller cấp cao hơn bắt
} catch (e) {
    print('Cloudinary: Lỗi không xác định: $e');
    rethrow;
}
}

```

3.3.8. Điều hướng với GetX và quản lý state

Kết quả đạt được:

Toàn bộ kiến trúc ứng dụng được xây dựng trên GetX, mang lại khả năng quản lý state (trạng thái) và điều hướng (navigation) hiệu quả.

- **Quản lý State:** Sử dụng các biến Rx (ví dụ: RxList, RxBool, RxMap) trong các GetxController. Giao diện (Obx) tự động cập nhật khi các biến này thay đổi, loại bỏ nhu cầu sử dụng setState().
- **Điều hướng:** Get.to() được dùng để mở trang mới. Get.back() để đóng. Get.offAll() được dùng trong AuthController để điều hướng chính (Home/Login) và xóa toàn bộ các trang trước đó.

Mã code đại diện:

Đây là đoạn mã quan trọng nhất thể hiện việc quản lý state tự động của AuthController. user.bindStream lắng nghe Firebase, và ever(user, ...) phản ứng với sự thay đổi state đó để tự động điều hướng.

```
class AuthController extends GetxController {

    // ...

    // Biến Rx (reactive) lắng nghe trạng thái user
    final Rx<User?> user = Rx<User?>(firebaseAuth.currentUser);

    // ...

    @override
    void onReady() {
        super.onReady();

        // 1. Tự động cập nhật biến 'user' mỗi khi FirebaseAuth
        // thay đổi
        user.bindStream(firebaseAuth.authStateChanges());

        print("AuthController: Bắt đầu lắng nghe trạng thái đăng
        nhập...");

        // 2. Tự động gọi hàm _setInitialScreen MỖI KHI biến
        // 'user' thay đổi
        ever(user, _setInitialScreen);
    }

    void _setInitialScreen(User? user) {
```

```

print("AuthController: Trạng thái đăng nhập thay đổi. User
là: ${user?.uid ?? 'null'}");

if (user == null) {

    // 3. Nếu state là 'null' -> điều hướng về Login

    print("AuthController: Điều hướng đến LoginScreen.");
    Get.offAll(() => LoginScreen());
}

} else {

    // 4. Nếu state có user -> điều hướng về Home

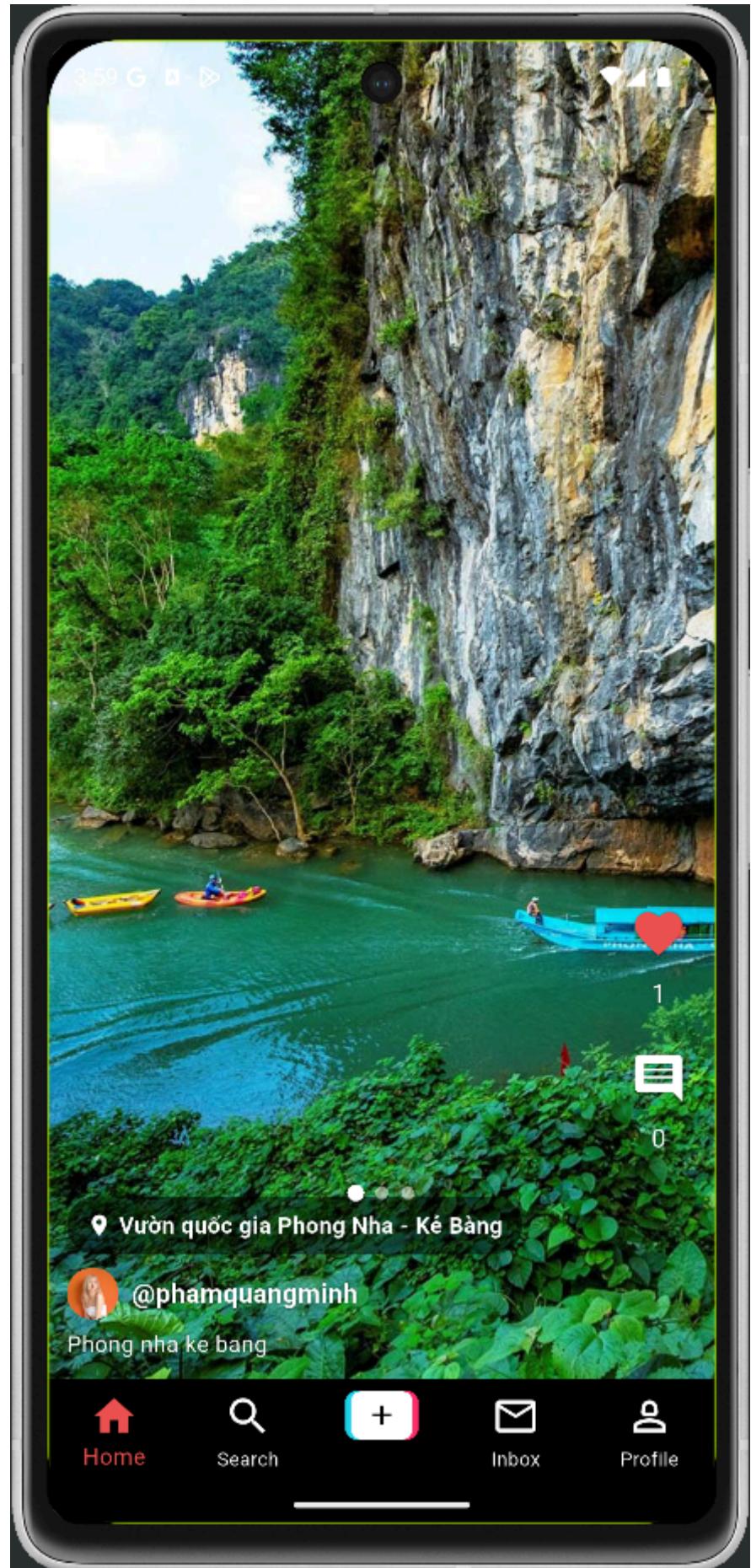
    print("AuthController: Điều hướng đến HomeScreen.");
    Get.offAll(() => const HomeScreen());

}
}
}

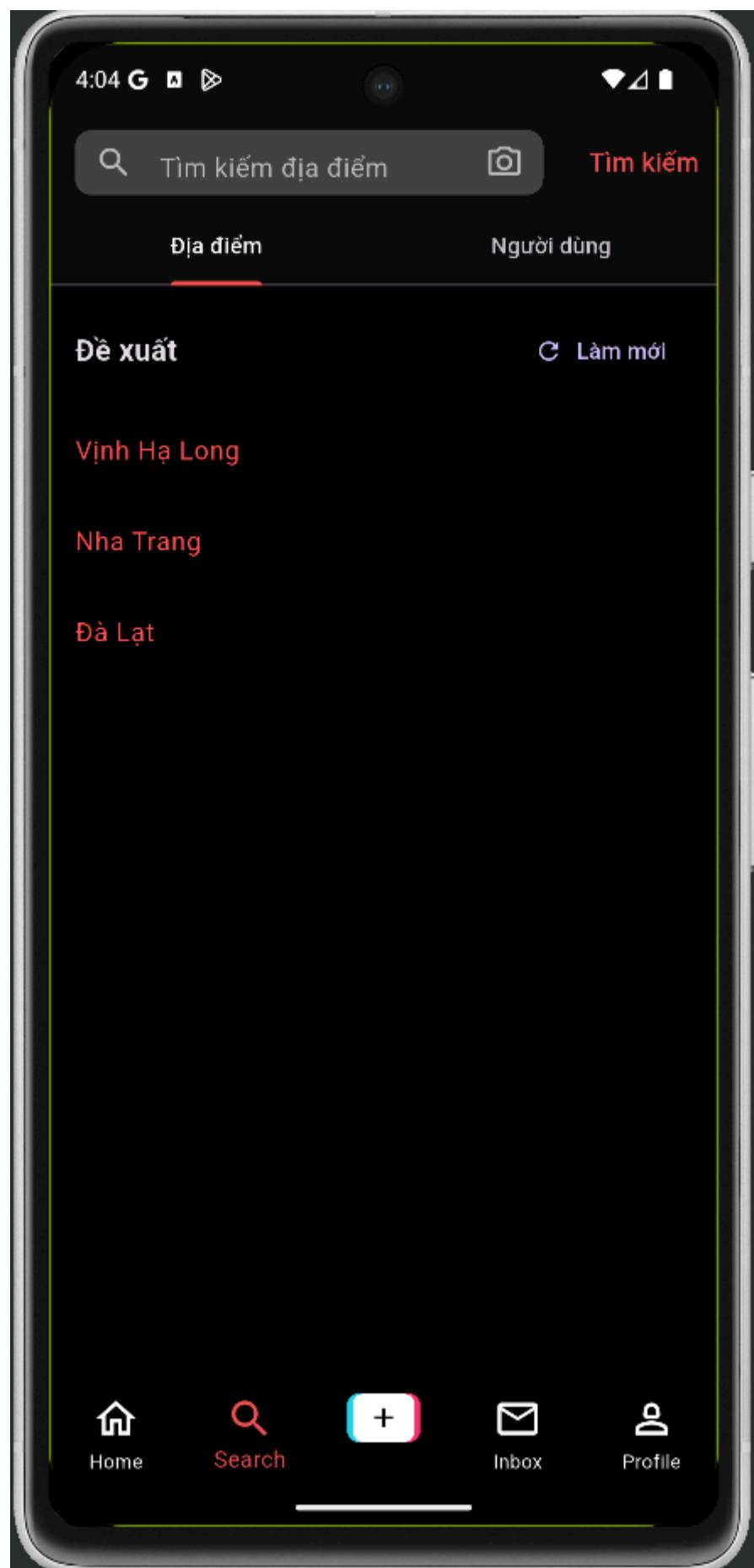
```

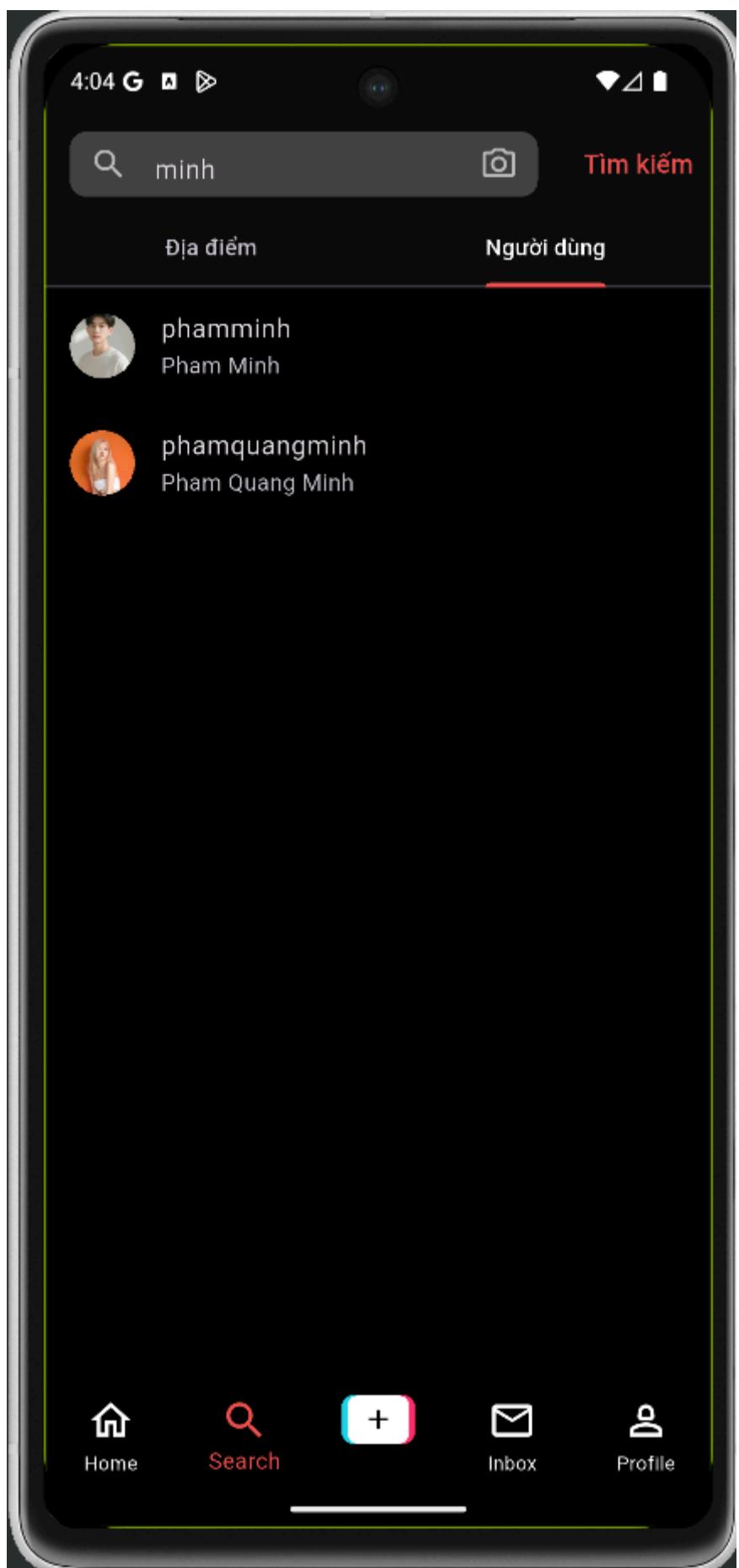
3.4. Kết quả thực nghiệm

3.4.1. Các giao diện

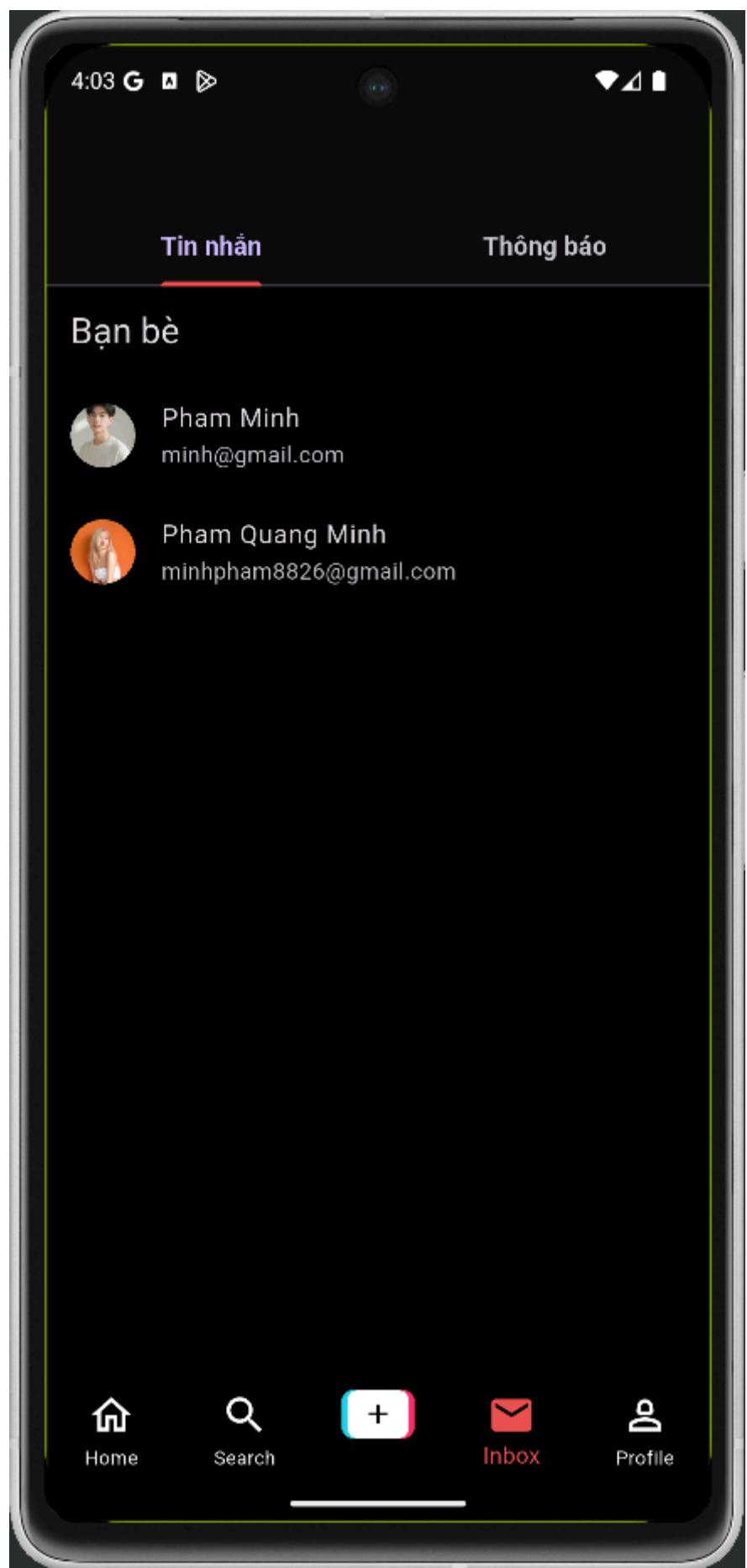


1.1.1.17. Giao diện trang chủ.

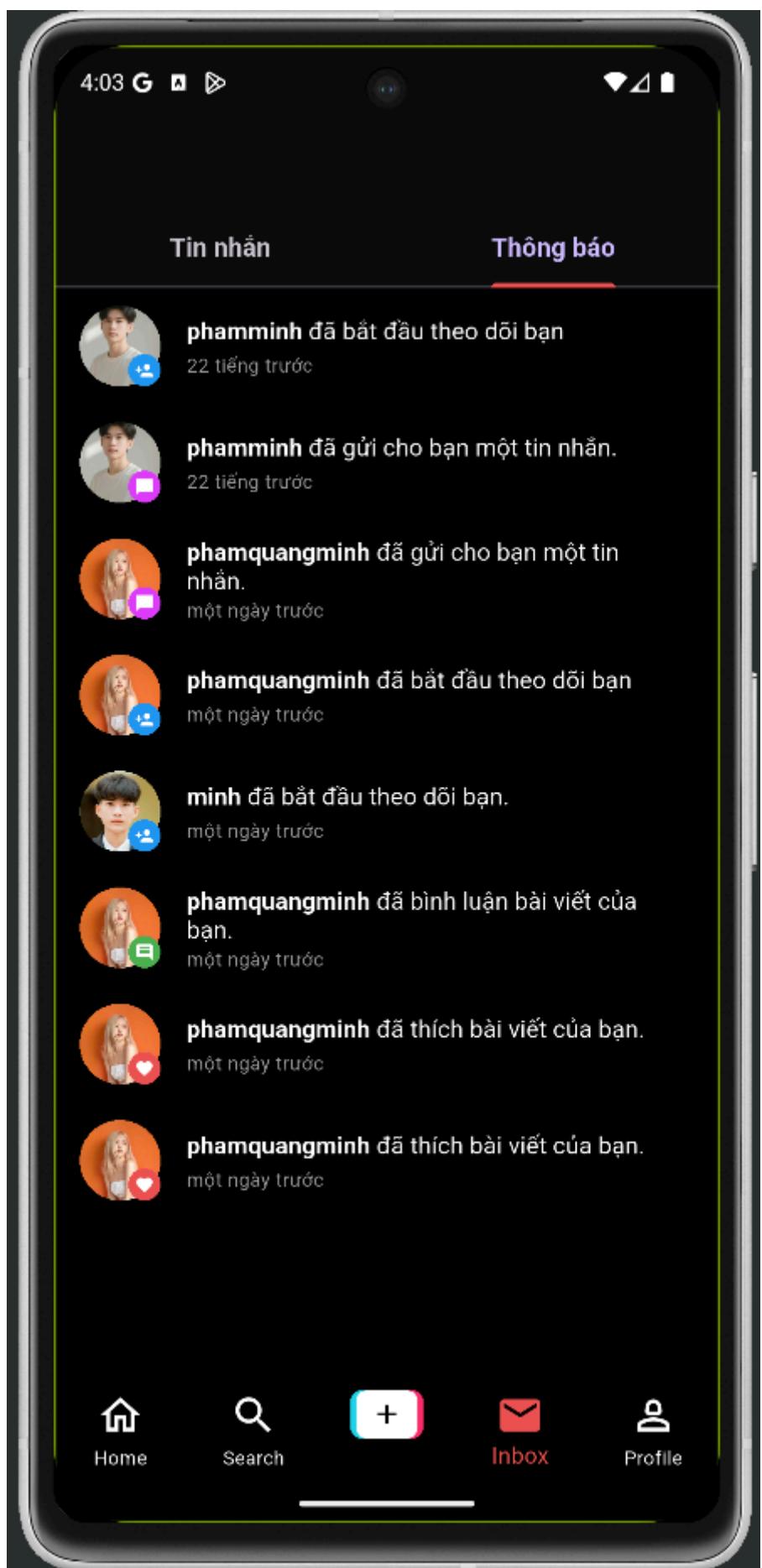




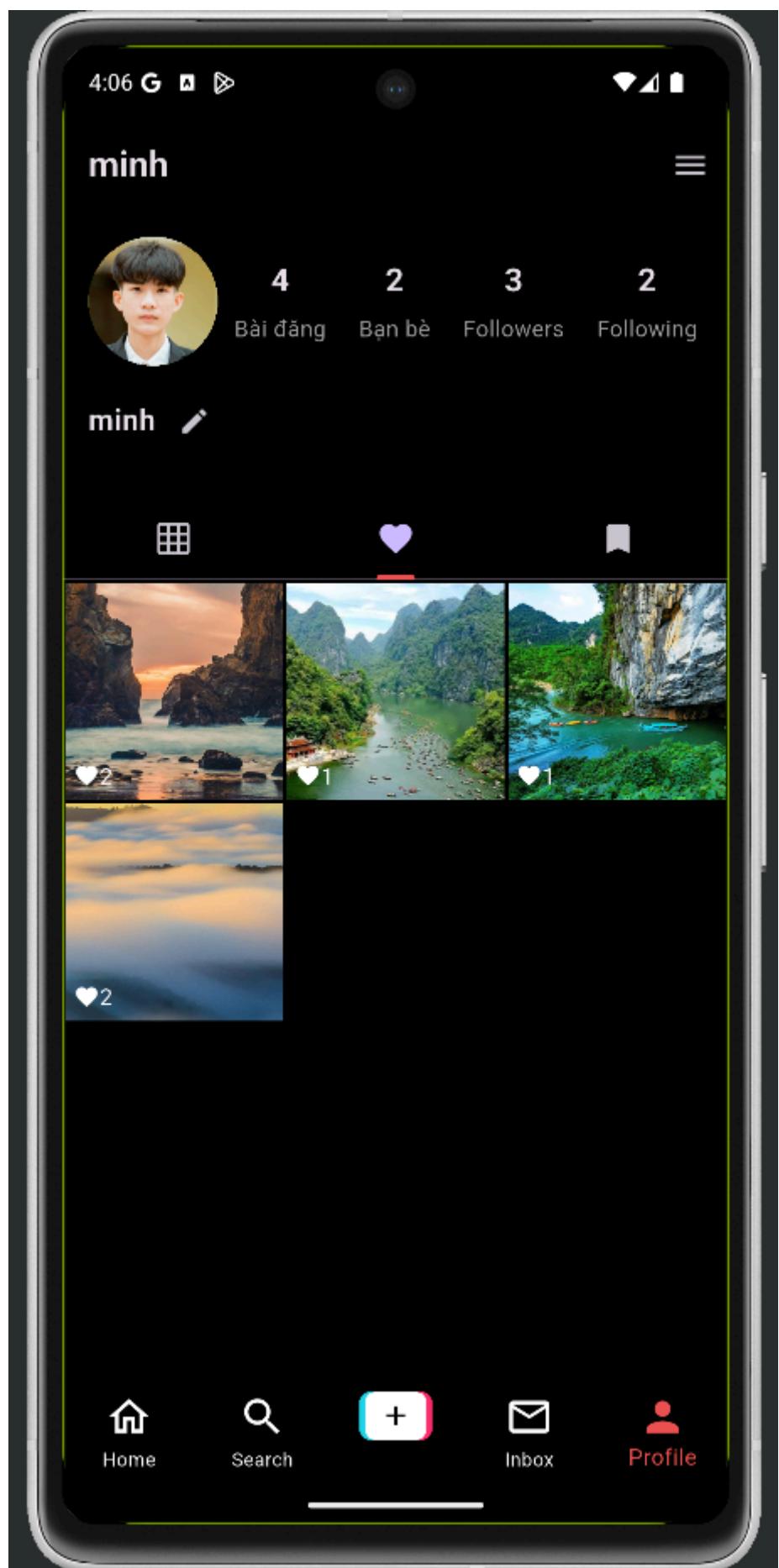
1.1.1.18. Giao diện tìm kiếm.

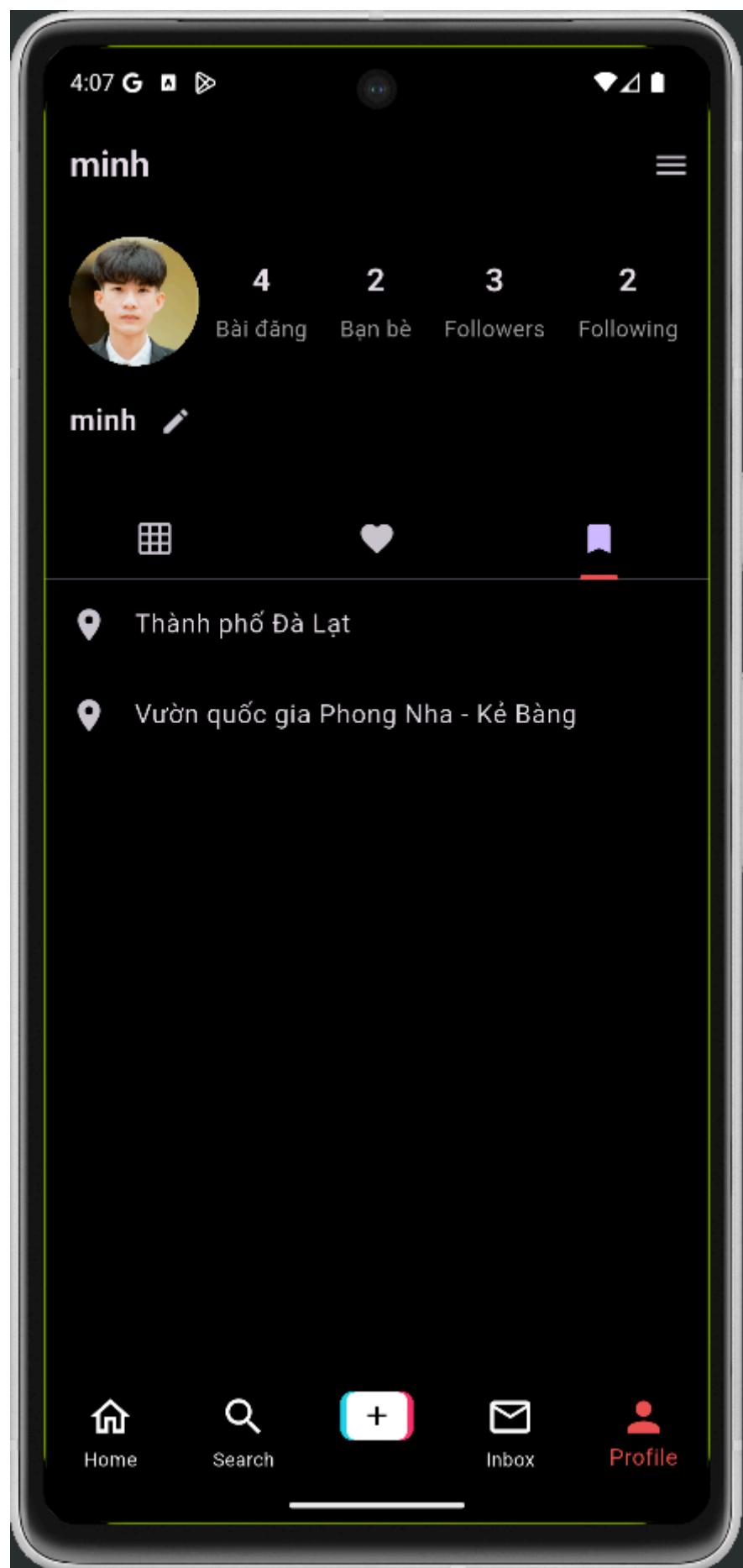


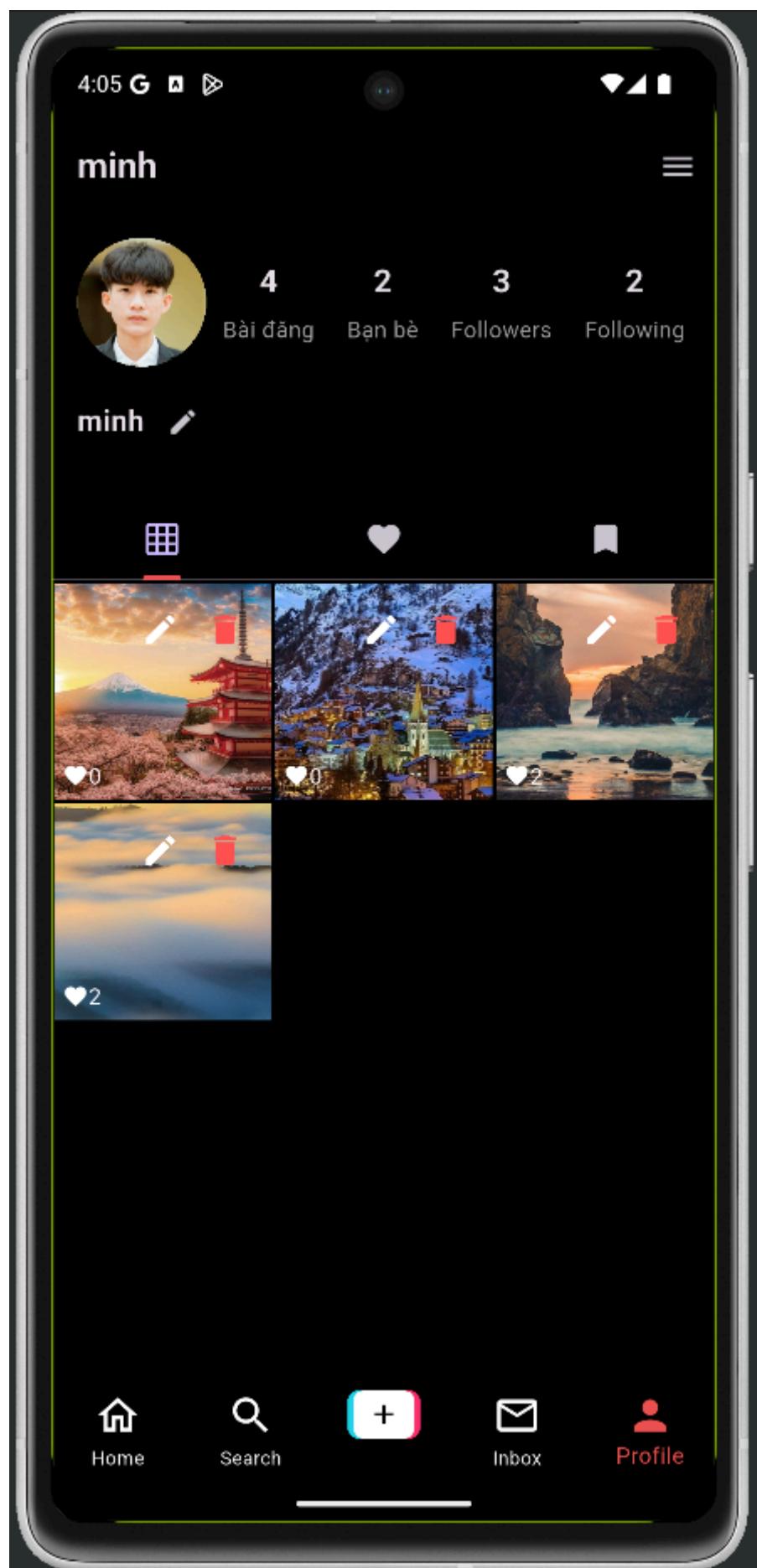
1.1.1.19. Giao diện bạn bè.



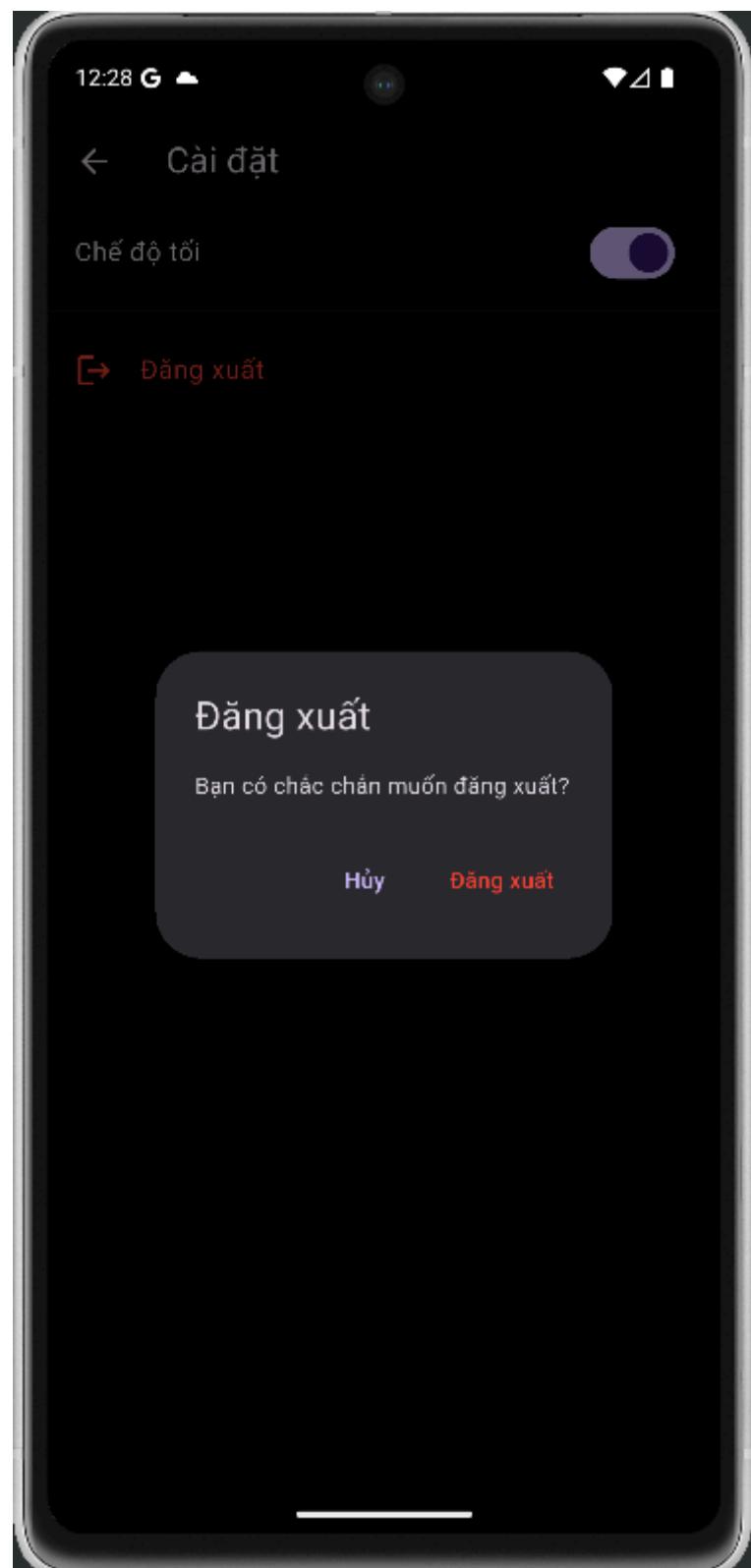
1.1.1.20. Giao diện thông báo.



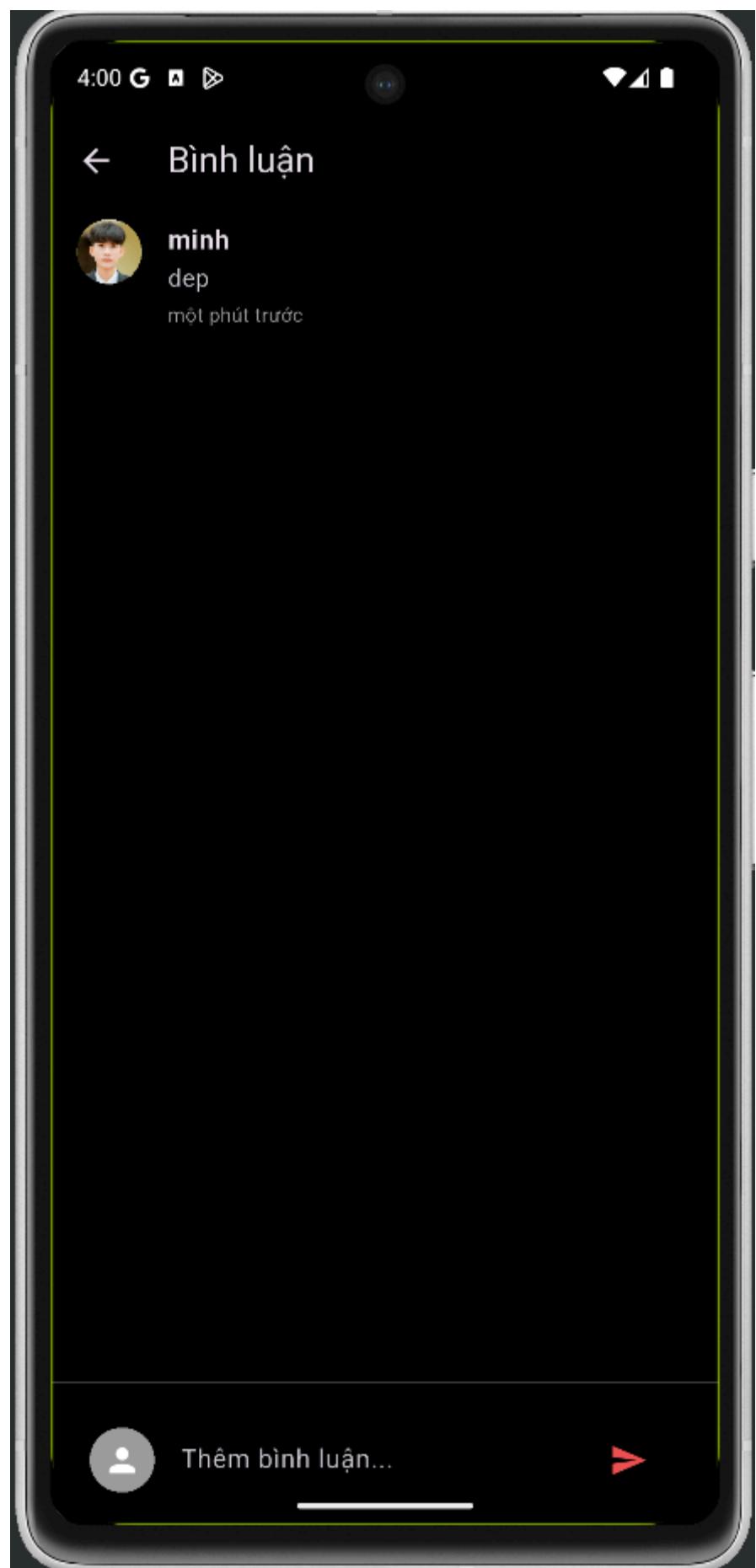




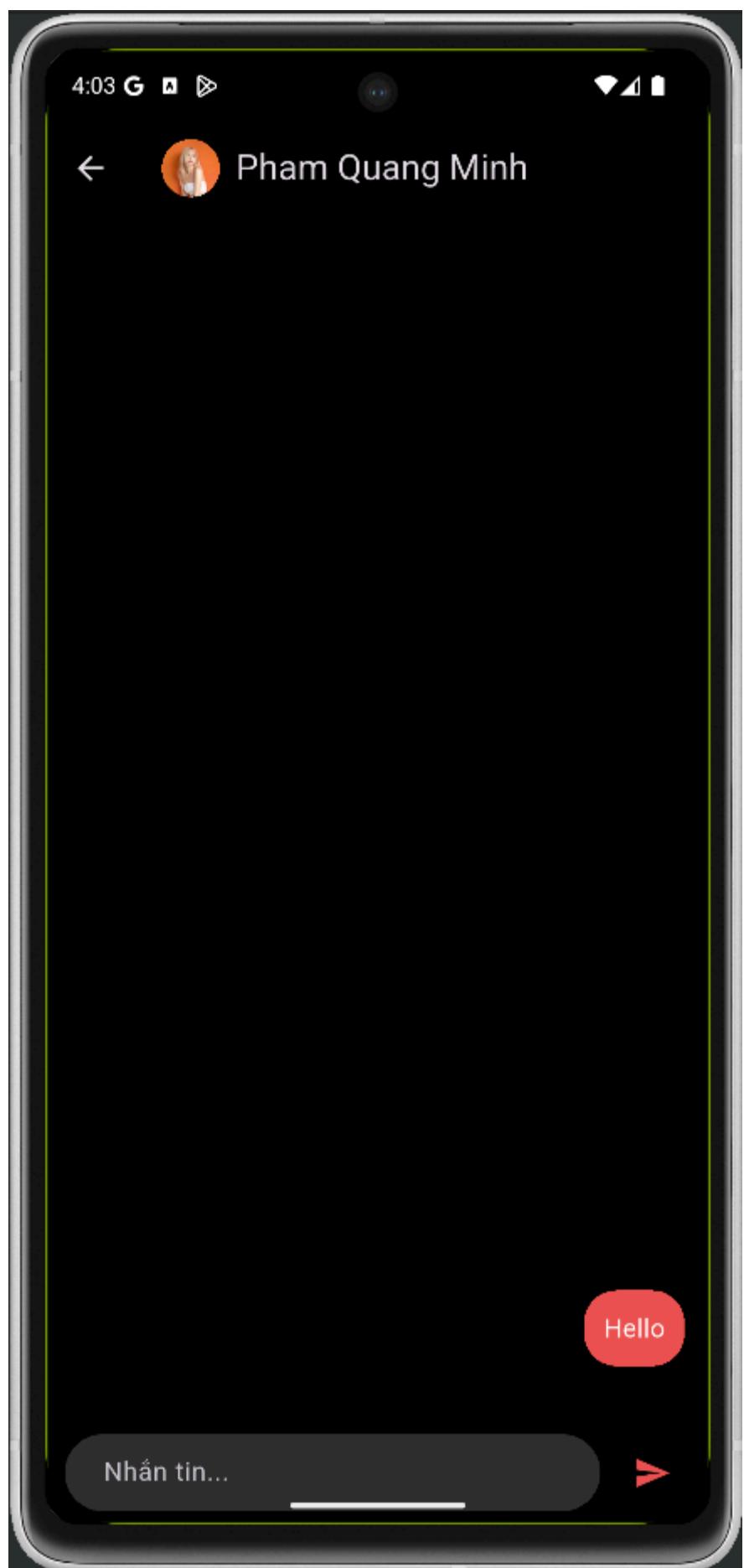
1.1.1.21. Giao diện profile.



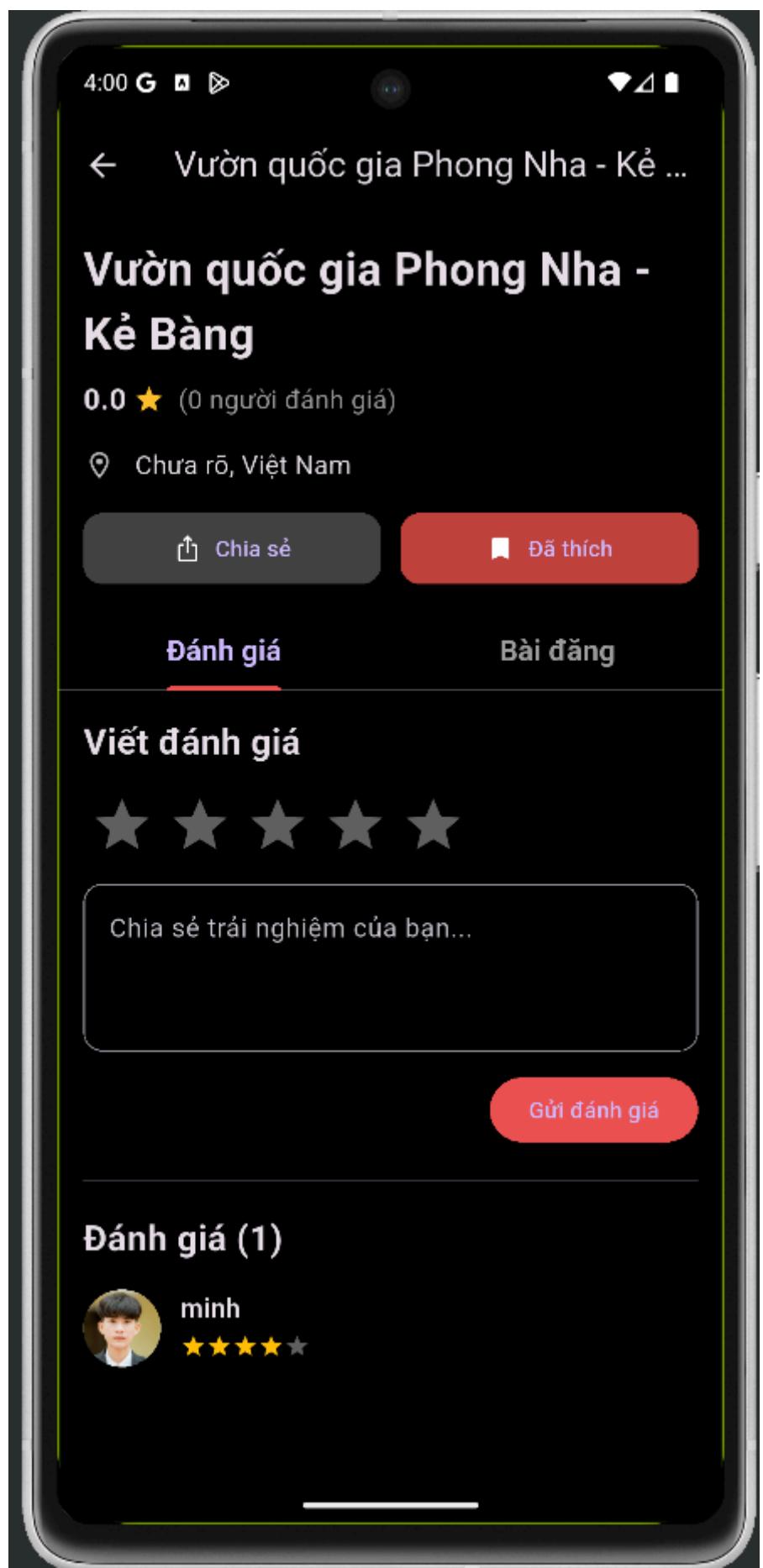
1.1.1.22. Giao diện setting.

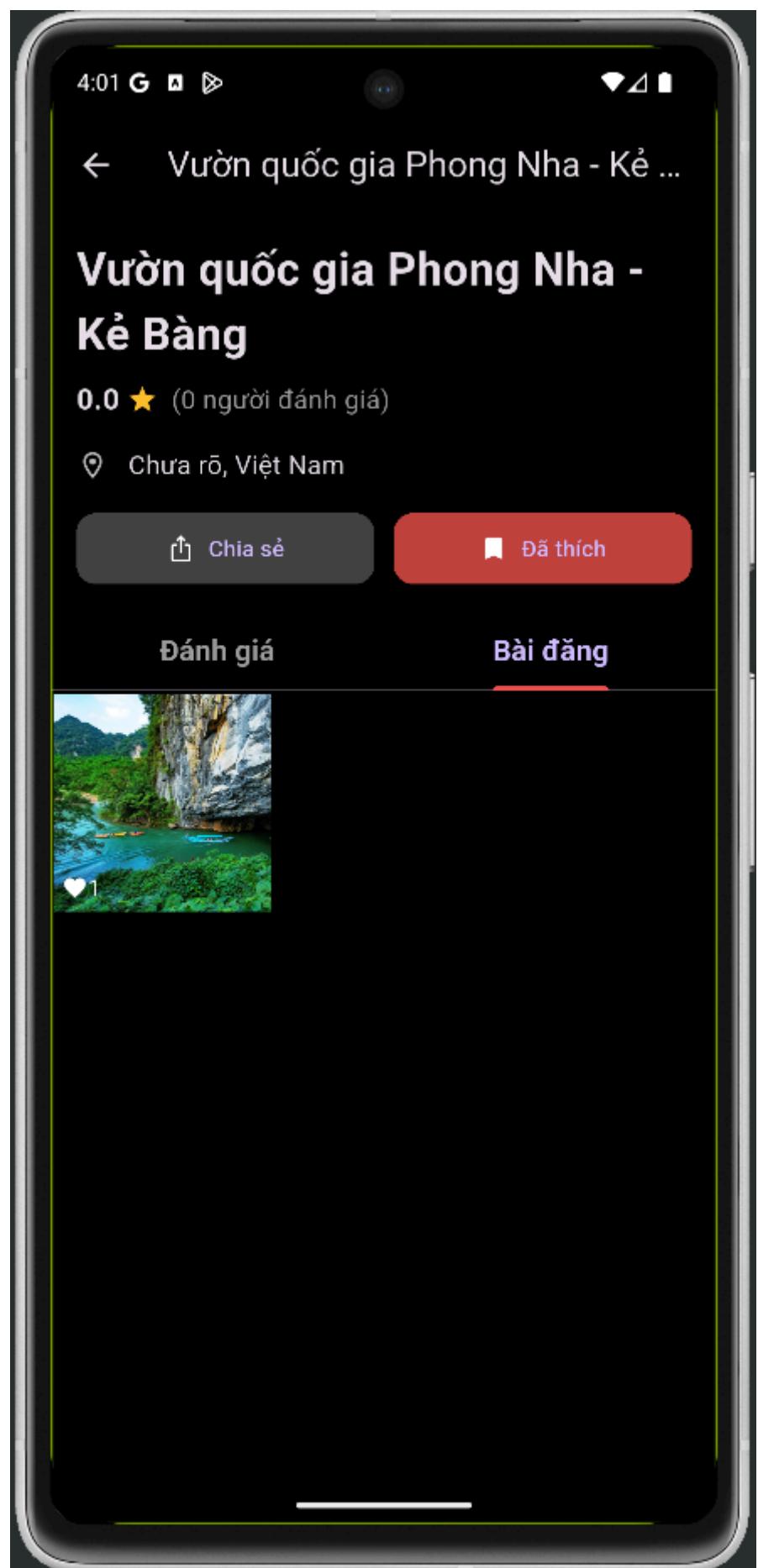


1.1.1.23. Giao diện comment.



1.1.1.24. Giao diện chat.





3.4.2. Video Demo

Link video demo: <https://youtu.be/sloEyhPcLKA>

3.4.3. Đánh giá hiệu năng và trải nghiệm người dùng

- **Hiệu năng:**

- Nhờ kiến trúc GetX (chỉ build lại các widget Obx cần thiết) và Flutter, ứng dụng cho trải nghiệm cuộn mượt mà.
- Việc sử dụng cached_network_image giúp tối ưu tốc độ tải ảnh ở các màn hình PostFeedScreen và ProfileScreen.
- Việc phi chuẩn hóa (denormalization) trong Firestore (ví dụ: commentCount, username trong bình luận) giúp giảm đáng kể số lượng truy vấn và tăng tốc độ tải màn hình.
- *Điểm cần cải thiện:* Chức năng tính toán rating trung bình trên LocationController đang được thực hiện ở phía client (get() toàn bộ reviews). Nếu có nhiều đánh giá, việc này nên được chuyển về backend (ví dụ: Cloud Functions) để tối ưu.

- **Trải nghiệm người dùng (UX):**

- **Tốt:** Luồng điều hướng rõ ràng. Trải nghiệm cuộn feed (TikTok-style) trên PostFeedScreen rất hiện đại và dễ gây nghiện. Các tính năng xã hội (like, comment, follow) trực quan. Chế độ Sáng/Tối là một điểm cộng lớn.
- **Tốt:** Logic "Bạn bè" (mutual follow) được xử lý tự động rất thông minh, giúp đơn giản hóa việc bắt đầu chat.
- *Điểm cần cải thiện:* Việc hiển thị lỗi (qua Get.snackbar) đôi khi chỉ hiển thị e.toString() [cite: auth_controller.dart], có thể không thân thiện với người dùng. Cần có các thông báo lỗi cụ thể hơn (ví dụ: "Sai mật khẩu", "Email đã tồn tại").

CHƯƠNG 4. ĐÁNH GIÁ VÀ HƯỚNG PHÁT TRIỂN

4.1. Các kết quả đạt được

1. **Hoàn thiện hệ thống xác thực và quản lý người dùng:** Đã triển khai thành công luồng đăng ký (với avatar), đăng nhập, đăng xuất [cite: auth_controller.dart]. Hệ thống quản lý trạng thái đăng nhập tự động qua GetX và Firebase Auth.
2. **Xây dựng giao diện Feed "TikTok-Style":** Đã triển khai thành công màn hình PostFeedScreen [cite: post_feed_screen.dart] sử dụng PageView cuộn dọc, cho phép người dùng khám phá bài đăng một cách trực quan và hiện đại.
3. **Hoàn thiện tính năng Quản lý Bài đăng (CRUD):** Ứng dụng cho phép người dùng Tạo bài đăng (hỗ trợ nhiều ảnh, gắn thẻ địa điểm), Đọc (trên feed), Cập nhật (chỉnh sửa mô tả, thêm/xóa ảnh) [cite: add_post_screen.dart, upload_post_controller.dart], và Xóa bài đăng [cite: profile_controller.dart].
4. **Triển khai tính năng "Location-Centric" độc đáo:** Đây là thành công lớn nhất của dự án. Hệ thống cho phép:
 - Tìm kiếm địa điểm (qua API OpenStreetMap) [cite: location_search_controller.dart].
 - Tạo và xem chi tiết địa điểm (LocationScreen).
 - Cho phép người dùng đánh giá (review) địa điểm (sao + nội dung) [cite: location_controller.dart].
 - Tự động tổng hợp các bài đăng liên quan đến địa điểm.
5. **Xây dựng hệ thống Mạng xã hội (Social Graph):**
 - Triển khai logic Follow/Unfollow.
 - Triển khai logic "Bạn bè" (Friends) tự động (mutual follow), là điều kiện để nhận tin [cite: profile_controller.dart, conversation_controller.dart].
6. **Xây dựng hệ thống tương tác Real-time:**
 - Hoàn thiện tính năng Like [cite: post_controller.dart] và Comment [cite: comment_controller.dart].
 - Hoàn thiện hệ thống Thông báo (Notifications) real-time cho các hành động (Like, Comment, Follow, Message) [cite: notification_controller.dart].
 - Hoàn thiện hệ thống Chat 1-1 cho "Bạn bè" [cite: chat_detail_controller.dart].
7. **Sử dụng Kiến trúc GetX (MVVM) hiệu quả:** Dự án đã phân tách rõ ràng logic nghiệp vụ (Controllers) khỏi giao diện (Views), giúp mã nguồn dễ đọc, bảo trì và tối ưu hiệu suất (chỉ build lại widget Obx khi cần).
8. **Tối ưu Cơ sở dữ liệu:** Đã áp dụng kỹ thuật phi chuẩn hóa (denormalization) khi sao chép username/avatarUrl vào bình luận [cite: comment.dart] và cập nhật commentCount [cite: post.dart], giúp tối ưu hóa tốc độ đọc dữ liệu.

4.2. Hạn chế của ứng dụng

1. **Tính toán nặng ở phía Client:** Đây là hạn chế lớn nhất.
 - Khi người dùng gửi một đánh giá (review), LocationController [cite: location_controller.dart] phải tải *toàn bộ* các đánh giá cũ của địa điểm đó về client để tính lại rating trung bình.
 - Khi xem hồ sơ, ProfileController [cite: profile_controller.dart] phải tải *toàn bộ* bài đăng của người dùng để lặp (loop) qua và đếm tổng số likes.
 - Cả hai thao tác này sẽ thất bại hoặc gây treo ứng dụng khi dữ liệu lớn (ví dụ: một địa điểm có 20.000 đánh giá).
2. **Chưa hỗ trợ Video:** Mặc dù giao diện mô phỏng TikTok, nhưng mô hình dữ liệu (post.dart) và controller (upload_post_controller.dart) hiện tại chỉ hỗ trợ imageUrls. Ứng dụng chưa thể đăng tải video, vốn là yếu tố cốt lõi của trải nghiệm TikTok.
3. **Xử lý lỗi chưa thân thiện (UX):** Nhiều chức năng khi gặp lỗi (ví dụ: auth_controller.dart, comment_controller.dart) chỉ hiển thị lỗi thô cho người dùng qua Get.snackbar('Lỗi', e.toString()). Người dùng sẽ thấy các mã lỗi kỹ thuật như [firebase_auth/wrong-password] thay vì "Sai mật khẩu".
4. **Bảo mật API Key:** Các thông tin nhạy cảm như apiKey và cloudName của Cloudinary được lưu trực tiếp (hard-coded) trong file cloudinary_controller.dart. Đây là một rủi ro bảo mật nếu mã nguồn bị lộ.
5. **Bất đồng bộ trong Model:** Mô hình dữ liệu user.dart [cite: user.dart] không liệt kê các trường followers và following, nhưng profile_controller.dart [cite: profile_controller.dart] vẫn đọc và ghi các trường này vào Firestore. Điều này gây khó khăn cho việc bảo trì khi model không phản ánh đúng cấu trúc CSDL.

4.3. Định hướng phát triển trong tương lai

Để khắc phục các hạn chế và phát triển "Sketch Travel" thành một sản phẩm hoàn chỉnh, các hướng phát triển trong tương lai bao gồm:

1. **Tối ưu Backend (Ưu tiên hàng đầu):**
 - **Sử dụng Firebase Cloud Functions:** Chuyển toàn bộ logic tính toán (tính rating trung bình, đếm totalLikes) từ client (Flutter) lên backend (Cloud Functions). Sử dụng Triggers để tự động cập nhật các trường tổng hợp này mỗi khi có tài liệu mới được tạo/thay đổi.
2. **Hỗ trợ nội dung Video:**
 - Mở rộng Post model để hỗ trợ imageUrl.
 - Tích hợp video_player vào PostFeedScreen.
 - Cập nhật UploadPostController và CloudinaryController để xử lý upload file video (thay vì chỉ Uint8List của ảnh).
3. **Xây dựng Feed "For You" (AI/ML):**

- Hiện tại, feed (post_controller.dart) chỉ sắp xếp theo thời gian. Hướng phát triển tiếp theo là xây dựng một hệ thống gợi ý (recommendation engine) đơn giản, gợi ý bài đăng dựa trên các địa điểm người dùng đã thích hoặc các tài khoản họ đã theo dõi.

4. Cải thiện Trải nghiệm Chat:

- Cho phép người dùng chat với bất kỳ ai (không chỉ "Bạn bè") thông qua một hệ thống "Tin nhắn chờ" (Message Requests).

5. Bảo mật và Hoàn thiện:

- Di chuyển toàn bộ API keys và thông tin nhạy cảm ra khỏi mã nguồn, sử dụng file .env và thư viện flutter_dotenv.
- Xây dựng một lớp (class) quản lý lỗi lõi tập trung để chuyển các mã lỗi (error codes) thành các thông báo tiếng Việt thân thiện với người dùng.

6. Tích hợp Bản đồ (Mapping):

- Thêm một tab "Bản đồ" vào SearchScreen hoặc ProfileScreen, sử dụng Maps_flutter để hiển thị trực quan các địa điểm mà người dùng đã check-in hoặc đã lưu.

KẾT LUẬN CHUNG

Báo cáo này đã trình bày toàn diện quá trình phân tích, thiết kế, và triển khai ứng dụng mạng xã hội du lịch "Sketch Travel". Dự án được xây dựng trên nền tảng **Flutter**, sử dụng kiến trúc **GetX (MVVM)**, và vận hành trên các dịch vụ BaaS (Backend as a Service) mạnh mẽ là **Firebase** và **Cloudinary**.

Đề tài đã giải quyết thành công bài toán đặt ra: xây dựng một ứng dụng kết hợp được trải nghiệm cuộn feed trực quan của TikTok với một hệ thống dữ liệu du lịch có cấu trúc, xoay quanh các địa điểm.

Kết quả thực nghiệm (dựa trên 48 file code) cho thấy ứng dụng đã hoàn thành tất cả các chức năng cốt lõi: từ xác thực, quản lý bài đăng (hỗ trợ nhiều ảnh), đến các tính năng xã hội phức tạp (follow, bạn bè, chat, thông báo real-time) và các tính năng "location-centric" độc đáo (đánh giá, yêu thích, tổng hợp bài đăng theo địa điểm).

Bên cạnh các kết quả đạt được, báo cáo cũng đã phân tích rõ các hạn chế còn tồn tại, chủ yếu liên quan đến việc xử lý tính toán ở phía client và việc chưa hỗ trợ video. Tuy nhiên, đây là những vấn đề có thể giải quyết được.

Nhìn chung, dự án "Sketch Travel" là một minh chứng rõ ràng cho khả năng của Flutter và GetX trong việc xây dựng các ứng dụng phức tạp, hiệu năng cao. Sản phẩm hiện tại là một nền tảng vững chắc, sẵn sàng cho các bước tối ưu hóa backend (sử dụng Cloud Functions) và mở rộng tính năng (hỗ trợ video) trong tương lai.

TÀI LIỆU THAM KHẢO

- [1] Flutter Team, “*Flutter Official Documentation*”, Google, Truy cập tại: <https://flutter.dev/>
- [2] Dart Team, “*Dart Language Official Documentation*”, Google, Truy cập tại: <https://dart.dev/>
- [3] Firebase Team, “*FlutterFire Documentation*”, Google, Truy cập tại: <https://firebase.flutter.dev/>
- [4] Firebase, “*Authentication with Email/Password - Firebase Authentication*”, Google Developers.
- [5] Firebase, “*Cloud Firestore Documentation*”, Google Developers – Truy vấn, Quản lý dữ liệu và Sub-collections.
- [6] GetX Community, “*GetX: State Management, Dependency Injection and Navigation*”, Truy cập tại: <https://pub.dev/packages/get>
- [7] Cloudinary, “*Cloudinary Flutter SDK*”, Truy cập tại: https://pub.dev/packages/cloudinary_public
- [8] OpenStreetMap Foundation, “*Nominatim Search API Documentation*”, Truy cập tại: <https://nominatim.org/release-docs/latest/api/Search/>
- [9] Flutter Community, “*image_picker package documentation*”, Truy cập tại: https://pub.dev/packages/image_picker
- [10] Flutter Community, “*cached_network_image package documentation*”, Truy cập tại: https://pub.dev/packages/cached_network_image
- [11] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, “*Design Patterns: Elements of Reusable Object-Oriented Software*”, Addison-Wesley, 1994.
- [12] Pramod J. Sadalage, Martin Fowler, “*NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*”, Addison-Wesley, 2012.